On Combining Pipelining and s-Step Concepts in Preconditioned Conjugate Gradient Methods

Viktoria Mayer* Wilfried N. Gansterer†

Abstract. On large-scale parallel computers, global communication becomes a major bottleneck for the Preconditioned Conjugate Gradient (PCG) method, an iterative solver for large sparse linear systems. Scalable PCG variants reduce the number of global synchronization points by a factor of $\mathcal{O}(s)$ (s-step methods), or overlap communication with local computations (pipelined methods). The pipelined s-step PCG method P-sPCG$_{\text{mon}}$ combines these two approaches, but introduces significant local computational overhead increasing with step size s. Moreover, it suffers from poor numerical stability.

Choosing a suitable basis type for the s-step basis matrices usually strongly improves numerical stability. Thus, we generalize P-sPCG$_{\text{mon}}$, which was designed to only use the monomial basis, to support arbitrary basis types, denoting our new method as P-sPCG. Moreover, we also reformulate the more stable s-step method CA-PCG such that its global communication is overlapped with computations, denoting another new pipelined s-step method as P-CA-PCG.

Numerical experiments on 40 real-world test problems show that P-sPCG improves numerical stability compared to P-sPCG$_{\text{mon}}$, while P-CA-PCG is even more stable than P-sPCG. Strong scaling experiments with a synthetic test problem confirm that for larger values of s, P-CA-PCG outperforms P-sPCG and all existing pipelined and s-step methods except for one, which, however, showed significantly worse numerical stability in our experiments.

1 Introduction The Preconditioned Conjugate Gradient (PCG) algorithm is an important Krylov subspace method (KSM) for solving large sparse linear systems $\boldsymbol{Ax} = \boldsymbol{b}$ with a sparse symmetric and positive-definite (SPD) system matrix $\boldsymbol{A}$. PCG frequently computes scalar products of dense vectors, which are distributed among nodes and processes on parallel computers. These computations require global collectives that become major bottlenecks on large-scale parallel computers due to their limited parallel scalability and due to the synchronization points they introduce.

Scalable PCG variants aim to reduce these performance limitations: *Communication-hiding pipelined* PCG methods overlap global communication with local communication and computation [12, 9]. *Communication-avoiding s-step* methods aim to reduce global communication latency by a factor of $\mathcal{O}(s)$ [8, 18, 13]. These methods enhance the Krylov subspace by s vectors at once by computing a basis of dimension $\mathcal{O}(s)$ for the next s steps. In [17], a *combination* of pipelining and s-step ideas for PCG is presented, which we denote as P-sPCG$_{\text{mon}}$.

Although these scalable variants are mathematically equivalent to standard PCG in exact arithmetic, their different numerical error propagation in floating-point arithmetic may slow down or even prevent convergence in practice [1, 6]. Choosing a suitable basis type for computing the s-step basis matrices is known to be an important aspect when it comes to numerical stability in s-step methods [13]. However, in P-sPCG$_{\text{mon}}$ only the monomial basis can be used. This algorithm suffers from poor numerical stability and also introduces significant overhead in terms of local computation.

To improve the numerical stability of P-sPCG$_{\text{mon}}$, we propose a generalization P-sPCG of this method to allow the usage of arbitrary basis types. Moreover, we develop the new pipelined s-step algorithm P-CA-PCG based on a different s-step method CA-PCG [18], which significantly improves the numerical stability compared to P-sPCG$_{\text{mon}}$ and even P-sPCG. While P-sPCG$_{\text{mon}}$, P-sPCG and P-CA-PCG have the same communication-hiding and -avoiding properties, P-CA-PCG performs significantly better than P-sPCG$_{\text{mon}}$ and P-sPCG with higher values of s due to its much lower overhead in terms of local computation.

1.1 Related work The first s-step PCG algorithm sPCG$_{\text{mon}}$ [8] suffers from numerical instabilities as it is formulated such that only the monomial basis can be used. The newer s-step methods CA-PCG [18], CA-PCG3 [13] and sPCG [14] (an extended version of sPCG$_{\text{mon}}$) can use other basis types, such as the

*University of Vienna, UniVie Doctoral School Computer Science DoCS, Faculty of Computer Science, Vienna, Austria (viktoria.mayer@univie.ac.at).

†University of Vienna, Faculty of Computer Science, Vienna, Austria (wilfried.gansterer@univie.ac.at).

Newton or the Chebyshev basis, to increase numerical stability [13].

The Pipelined PCG (PPCG) method [12] hides global communication by overlapping it with the matrix-vector (MV) product and the preconditioner application of the current iteration. The algorithm presented in [9] overlaps communication over several iterations while the Predict-and-Recompute PPCG (P&R-PPCG) method [7] improves the numerical stability of PPCG. The pipelined algorithm in [16] requires only one synchronization step every two iterations.

1.2 Contributions of this work We propose two novel pipelined s-step PCG algorithms with better performance and numerical stability than the so far only pipelined s-step PCG algorithm P-sPCG$_{\text{mon}}$ [17].

1. We generalize P-sPCG$_{\text{mon}}$ [17] for arbitrary basis types to improve its numerical stability, denoting this new algorithm as P-sPCG.
2. We develop a new communication-hiding s-step PCG variant P-CA-PCG based on CA-PCG [18], which is less expensive for larger values of s and numerically more stable than P-sPCG$_{\text{mon}}$ and even P-sPCG.

We compare these new methods with existing pipelined and s-step PCG methods.

1.3 Synopsis In section 2, we summarize the existing methods sPCG$_{\text{mon}}$ [8], sPCG [14], P-sPCG$_{\text{mon}}$ [17] and CA-PCG [18]. In section 3, we generalize P-sPCG$_{\text{mon}}$ for arbitrary basis types. We show in section 4 how CA-PCG can be pipelined, proposing our new pipelined s-step variant P-CA-PCG. In section 5, we provide a theoretical cost analysis for all the algorithms discussed. Section 6 summarizes experimental evaluations of our novel solvers, both in terms of numerical stability and runtime performance. We conclude our work in section 7.

1.4 Notation 1-based indexing is used for matrices and vectors. For $1 \leq i, j \leq m$ and $1 \leq k, l \leq n$, $\boldsymbol{A}_{i:j,k:l}$ is a submatrix of $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ consisting of the elements in rows i to j and in columns k to l of $\boldsymbol{A}$. Columns k to l are selected using the simplified expression $\boldsymbol{A}_{k:l}$. $0_{i,j}$ denotes a zero matrix of size $i \times j$. $\boldsymbol{I}_{i,j}$ denotes a diagonal matrix of size $i \times j$ which has only ones on the diagonal.

2 Existing methods The PCG method (Algorithm 2.1) iteratively solves a system of linear equations $\boldsymbol{Ax} = \boldsymbol{b}$ 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 $\boldsymbol{b} \in \mathbb{R}^n$. To accelerate convergence, a preconditioner $\boldsymbol{M}^{-1} \in \mathbb{R}^{n \times n}$ is used. In this section, we review the concept of the Matrix Powers Kernel for computing the s-step basis matrices, the s-step algorithms sPCG/sPCG$_{\text{mon}}$ and CA-PCG as well as the pipelined version P-sPCG$_{\text{mon}}$ based on sPCG$_{\text{mon}}$.

Algorithm 2.1 Standard 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**
  $\boldsymbol{s}^{(i)} = \boldsymbol{A}\boldsymbol{p}^{(i)}$
  $\alpha^{(i)} = \boldsymbol{r}^{(i)^T}\boldsymbol{u}^{(i)} / \boldsymbol{p}^{(i)^T}\boldsymbol{s}^{(i)}$
  $\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} + \alpha^{(i)}\boldsymbol{p}^{(i)}$
  $\boldsymbol{r}^{(i+1)} = \boldsymbol{r}^{(i)} - \alpha^{(i)}\boldsymbol{s}^{(i)}$
  $\boldsymbol{u}^{(i+1)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(i+1)}$
  $\beta^{(i+1)} = \boldsymbol{r}^{(i+1)^T}\boldsymbol{u}^{(i+1)} / \boldsymbol{r}^{(i)^T}\boldsymbol{u}^{(i)}$
  $\boldsymbol{p}^{(i+1)} = \boldsymbol{u}^{(i+1)} + \beta^{(i+1)}\boldsymbol{p}^{(i)}$
end for

2.1 Matrix Powers Kernel Define a Krylov subspace $\mathcal{K}_{s+1}(\boldsymbol{A}, \boldsymbol{v})$ for a matrix $\boldsymbol{A}$ of size $n \times n$ and a vector $\boldsymbol{v}$ of length n as

$$\mathcal{K}_{s+1}(\boldsymbol{A}, \boldsymbol{v}) = \text{span}\{\boldsymbol{v}, \boldsymbol{Av}, \boldsymbol{A}^2\boldsymbol{v}, \ldots, \boldsymbol{A}^s\boldsymbol{v}\}.$$

The Matrix Powers Kernel (MPK) [11] computes the basis matrices

$$\boldsymbol{V} = \left[P_0(\boldsymbol{AM}^{-1})\boldsymbol{w}, \ldots, P_s(\boldsymbol{AM}^{-1})\boldsymbol{w}\right], \tag{2.1}$$

$$\boldsymbol{M}^{-1}\boldsymbol{V} = \left[P_0(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{v}, \ldots, P_s(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{v}\right], \tag{2.2}$$

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

$$\begin{aligned} &P_0(z) = 1, \quad P_1(z) = (z - \theta_0)P_0(z)/\gamma_0, \\ &P_l(z) = \frac{(z - \theta_{l-1})P_{l-1}(z) + \mu_{l-2}P_{l-2}(z))}{\gamma_{l-1}}, \ l \geq 2. \end{aligned} \tag{2.3}$$

The matrices in (2.1) and (2.2) span the Krylov subspaces $\mathcal{K}_{s+1}\left(\boldsymbol{AM}^{-1}, \boldsymbol{w}\right)$ and $\mathcal{K}_{s+1}\left(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{v}\right)$, respectively.

The choice of the basis is the main factor that influences stability of s-step KSMs [13]. Using the monomial basis, $\boldsymbol{V}$ is computed by repeatedly applying $\boldsymbol{A}$ and $\boldsymbol{M}^{-1}$ to a start vector. Consequently, the columns of $\boldsymbol{V}$ consist of the first iterates of the Power Iteration. In finite precision, these columns might not be linearly independent, which can lead to severe accuracy loss in s-step KSMs if $s > 5$ [18]. The Newton basis uses shifts obtained from estimated eigenvalues [15, 13], while the Chebyshev basis uses scaled and shifted Chebyshev polynomials [13].

To enable arbitrary basis types in s-step methods,

Algorithm 2.2 sPCG [14]

$\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}, \boldsymbol{P}^{(0)} = 0, \boldsymbol{AP}^{(0)} = 0$
for $k = 0, 1, ...$ until convergence **do**
  $\boldsymbol{S}^{(k)} = \mathcal{K}_{s+1}\left(\boldsymbol{AM}^{-1}, \boldsymbol{r}^{(k)}\right)$
  $\boldsymbol{U}^{(k)} = \mathcal{K}_{s}\left(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{u}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(k)}\right)$
  Compute $\boldsymbol{a}^{(k)}$ and $\boldsymbol{B}^{(k)}$
  $\boldsymbol{P}^{(k)} = \boldsymbol{U}^{(k)} + \boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}$
  $\boldsymbol{AP}^{(k)} = \boldsymbol{S}^{(k)}\boldsymbol{B} + \boldsymbol{AP}^{(k-1)}\boldsymbol{B}^{(k)}$
  $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{P}^{(k)}\boldsymbol{a}^{(k)}$
  $\boldsymbol{r}^{(k+1)} = \boldsymbol{r}^{(k)} - \boldsymbol{AP}^{(k)}\boldsymbol{a}^{(k)}$
end for

define the "change-of-basis" matrix $\boldsymbol{B}_i \in \mathbb{R}^{i\times(i-1)}$ as

$$\boldsymbol{B}_i = \begin{bmatrix} \theta_0 & \mu_0 & & \\ \gamma_0 & \theta_1 & \ddots & \\ & \gamma_1 & \ddots & \mu_{i-3} \\ & & \ddots & \theta_{i-2} \\ & & & \gamma_{i-2} \end{bmatrix}. \tag{2.4}$$

2.2 sPCG/sPCG$_{\text{mon}}$ In the s-step PCG method sPCG$_{\text{mon}}$ [8] and the extended version sPCG [14] (outlined in Algorithm 2.2) the iterations of standard PCG are computed in blocks of s. Only one global reduction operation is performed every s steps. The approximate solution is updated with

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

where the search direction matrix $\boldsymbol{P}^{(k)} = [\boldsymbol{p}^{(i)}, \ldots, \boldsymbol{p}^{(i+s-1)}]$ contains s search directions and the vector $\boldsymbol{a}^{(k)}$ of length s replaces the coefficients $\alpha^{(i)}, \ldots, \alpha^{(i+s-1)}$. The index i refers to the respective iteration in standard PCG, i.e. $\lfloor i/s \rfloor = k$. To perform s steps without communication, we compute the basis matrices span $\left(\boldsymbol{S}^{(k)}\right) = \mathcal{K}_{s+1}\left(\boldsymbol{AM}^{-1}, \boldsymbol{r}^{(sk)}\right)$ and $\boldsymbol{U}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{R}^{(k)}$ with the MPK as shown in (2.1) and (2.2), where $\boldsymbol{R}^{(k)}$ are the first s columns of $\boldsymbol{S}^{(k)}$ (the last column of (2.2) is ommitted as $\boldsymbol{U}^{(k)}$ only has s columns). Analogously to standard PCG, the search directions are updated with

$$\boldsymbol{P}^{(k)} = \boldsymbol{U}^{(k)} + \boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}, \tag{2.5}$$

where the matrix $\boldsymbol{B}^{(k)}$ of size $s \times s$ replaces the coefficients $\beta^{(i)}$ of standard PCG. Only one global reduction operation is required for the computation of $\boldsymbol{a}^{(k)}$ and $\boldsymbol{B}^{(k)}$.

The residual $\boldsymbol{r}^{(k)}$ is computed recursively [17, 14].

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k+1)} = \boldsymbol{r}^{(k)} - \boldsymbol{AP}^{(k)}\boldsymbol{a}^{(k)} \tag{2.6}$$

Multiplying (2.5) by $\boldsymbol{A}$, the matrix $\boldsymbol{AP}^{(k)}$ can be computed recursively.

$$\boldsymbol{AP}^{(k)} = \boldsymbol{AU}^{(k)} + \boldsymbol{AP}^{(k-1)}\boldsymbol{B}^{(k)} \tag{2.7}$$

To obtain $\boldsymbol{AU}^{(k)}$, define a "change-of-basis" matrix $\boldsymbol{B} = \boldsymbol{B}_{s+1}$ (see (2.4)) of size $(s+1) \times s$ such that

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

2.3 P-sPCG$_{\text{mon}}$ The pipelined s-step algorithm P-sPCG$_{\text{mon}}$ [17] overlaps the global communication for computing $\boldsymbol{a}^{(k)}$ and $\boldsymbol{B}^{(k)}$ with the MV products and preconditioner applications of the current iteration. In sPCG$_{\text{mon}}$, the s-step basis matrices $\boldsymbol{S}^{(k)}$ and $\boldsymbol{U}^{(k)}$ are computed with explicit MV products and preconditioner applications. As these matrices are required for computing $\boldsymbol{a}^{(k)}$ and $\boldsymbol{B}^{(k)}$, their computation cannot be overlapped with the global communication.

To enable recursive computation of the columns of $\boldsymbol{S}^{(k+1)}$ without explicit MV products and preconditioner applications, the basis matrix $\boldsymbol{S}^{(k)}$ is extended. The required explicitly computed MV products and preconditioner applications can be overlapped with the communication for computing $\boldsymbol{a}^{(k)}$ and $\boldsymbol{B}^{(k)}$ as their results are not needed for these computations.

By repeatedly multiplying (2.6) with $\boldsymbol{M}^{-1}$ and $\boldsymbol{A}$, we obtain recursive updates for all columns of $\boldsymbol{S}^{(k)}$ as well as the preconditioned basis matrix $\boldsymbol{T}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{S}^{(k)}$. For $j = 0, \ldots, s$, we have

$$\begin{aligned}(\boldsymbol{AM}^{-1})^j\boldsymbol{r}^{(k+1)} &= (\boldsymbol{AM}^{-1})^j\boldsymbol{r}^{(k)} \\ &\quad - (\boldsymbol{AM}^{-1})^j\boldsymbol{AP}^{(k)}\boldsymbol{a}^{(k)},\end{aligned} \tag{2.8}$$

$$\begin{aligned}(\boldsymbol{M}^{-1}\boldsymbol{A})^j\boldsymbol{u}^{(k+1)} &= (\boldsymbol{M}^{-1}\boldsymbol{A})^j\boldsymbol{u}^{(k)} \\ &\quad - (\boldsymbol{M}^{-1}\boldsymbol{A})^j\boldsymbol{M}^{-1}\boldsymbol{AP}^{(k)}\boldsymbol{a}^{(k)}.\end{aligned} \tag{2.9}$$

To update these vectors recursively, the following recursive computations are necessary. By repeatedly multiplying (2.7) with $\boldsymbol{M}^{-1}$ and $\boldsymbol{A}$, we obtain for $j = 0, \ldots, s$,

$$\begin{aligned}(\boldsymbol{AM}^{-1})^j\boldsymbol{AP}^{(k)} &= (\boldsymbol{AM}^{-1})^j\boldsymbol{AU}^{(k)} \\ &\quad + (\boldsymbol{AM}^{-1})^j\boldsymbol{AP}^{(k-1)}\boldsymbol{B}^{(k)},\end{aligned} \tag{2.10}$$

$$\begin{aligned}(\boldsymbol{M}^{-1}\boldsymbol{A})^{j+1}\boldsymbol{P}^{(k)} &= (\boldsymbol{M}^{-1}\boldsymbol{A})^{j+1}\boldsymbol{U}^{(k)} \\ &\quad + (\boldsymbol{M}^{-1}\boldsymbol{A})^{j+1}\boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}.\end{aligned} \tag{2.11}$$

The matrices $(\boldsymbol{AM}^{-1})^j\boldsymbol{AU}^{(k)}$ and $(\boldsymbol{M}^{-1}\boldsymbol{A})^{j+1}\boldsymbol{U}^{(k)}$ are submatrices of the extended basis matrices

$$\boldsymbol{S}_{ext}^{(k)} = \left[\boldsymbol{r}^{(k)}, (\boldsymbol{AM}^{-1})\boldsymbol{r}^{(k)}, \ldots, (\boldsymbol{AM}^{-1})^{2s}\boldsymbol{r}^{(k)}\right],$$
$$\boldsymbol{T}_{ext}^{(k)} = \left[\boldsymbol{u}^{(k)}, (\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{u}^{(k)}, \ldots, (\boldsymbol{M}^{-1}\boldsymbol{A})^{2s}\boldsymbol{u}^{(k)}\right].$$

Algorithm 2.3 CA-PCG [18]

$\boldsymbol{q}^{(0)} = \boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}, \boldsymbol{p}^{(0)} = \boldsymbol{u}^{(0)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(0)}$
for $k = 0, 1, ...$ until convergence **do**
  $\text{span}\left(\boldsymbol{Y}^{(k)}\right) = \mathcal{K}_{s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{q}^{(k)}\right) + \mathcal{K}_s\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{r}^{(k)}\right)$
  $\boldsymbol{Z}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{Y}^{(k)}$
  $\boldsymbol{G}^{(k)} = {\boldsymbol{Z}^{(k)}}^T\boldsymbol{Y}^{(k)}$
  "Inner iterations", see Algorithm 2.4
  $\left[\boldsymbol{q}^{(sk+s)}, \boldsymbol{r}^{(sk+s)}\right] = \boldsymbol{Y}^{(k)}\left[\boldsymbol{q}^{(sk+s)'}, \boldsymbol{r}^{(sk+s)'}\right]$
  $\left[\boldsymbol{p}^{(sk+s)}, \boldsymbol{u}^{(sk+s)}\right] = \boldsymbol{Z}^{(k)}\left[\boldsymbol{q}^{(sk+s)'}, \boldsymbol{r}^{(sk+s)'}\right]$
  $\boldsymbol{x}^{(sk+s)} = \boldsymbol{x}^{(sk)} + \boldsymbol{Z}^{(k)}\boldsymbol{x}^{(sk+s)'}$
end for

Algorithm 2.4 Inner Iterations of CA-PCG

$\boldsymbol{q}^{(sk)'}, = [1, 0_{1,2s}]^T, \boldsymbol{r}^{(sk)'} = [0_{1,s+1}, 1, 0_{1,s-1}]^T,$
$\boldsymbol{x}^{(sk)'} = [0_{1,2s+1}]^T$
for $j = 0, 1, ..., s-1$ **do**
  $\alpha^{(sk+j)} = \frac{\boldsymbol{r}^{(sk+j)'T}\boldsymbol{G}^{(k)}\boldsymbol{r}^{(sk+j)'}}{\boldsymbol{q}^{(sk+j)'T}\boldsymbol{G}^{(k)}\boldsymbol{B}\boldsymbol{q}^{(sk+j)'}}$
  $\boldsymbol{x}^{(sk+j+1)'} = \boldsymbol{x}^{(sk+j)'} + \alpha^{(sk+j)}\boldsymbol{q}^{(sk+j)'}$
  $\boldsymbol{r}^{(sk+j+1)'} = \boldsymbol{r}^{(sk+j)'} - \alpha^{(sk+j)}\boldsymbol{B}\boldsymbol{q}^{(sk+j)'}$
  $\beta^{(sk+j)} = \frac{\boldsymbol{r}^{(sk+j+1)'T}\boldsymbol{G}^{(k)}\boldsymbol{r}^{(sk+j+1)'}}{\boldsymbol{r}^{(sk+j)'T}\boldsymbol{G}^{(k)}\boldsymbol{r}^{(sk+j)'}}$
  $\boldsymbol{q}^{(sk+j+1)'} = \boldsymbol{r}^{(sk+j+1)'} + \beta^{(sk+j)}\boldsymbol{q}^{(sk+j)'}$
end for

The first $s+1$ columns of these two matrices correspond to $\boldsymbol{S}^{(k)}$ and $\boldsymbol{T}^{(k)}$ and are computed recursively with (2.8) and (2.9). The last s vectors are computed with explicit MV products and preconditioner applications. As these columns are not required for computing $\boldsymbol{a}^{(k)}$ and $\boldsymbol{B}^{(k)}$, the global synchronization points can be overlapped with these MV products and preconditioner applications.

2.4 CA-PCG The communication-avoiding PCG (CA-PCG) method [18] outlined in Algorithm 2.3 divides the loop of standard 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. Only one global global collective at the beginning of each outer iteration is required. CA-PCG linearly transforms the vectors of standard PCG such that the inner loop iterations $j = 0, \ldots, s-1$ can compute the next s steps in a changed basis without communication.

The unpreconditioned search direction is denoted as $\boldsymbol{q}^{(i)}$, i.e. $\boldsymbol{p}^{(i)} = \boldsymbol{P}\boldsymbol{q}^{(i)}$. By induction, for $0 \leq j \leq s \geq 1$,

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

The s-step basis matrices $\boldsymbol{Y}^{(k)} = \left[\boldsymbol{Q}^{(k)}, \boldsymbol{R}^{(k)}\right]$ and $\boldsymbol{Z}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{Y}^{(k)} = \left[\boldsymbol{P}^{(k)}, \boldsymbol{U}^{(k)}\right]$ with $\boldsymbol{P}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{Q}^{(k)}$ and $\boldsymbol{U}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{R}^{(k)}$ are computed at the beginning of outer iteration k such that

$$\text{span}\left(\boldsymbol{Q}^{(k)}\right) = \mathcal{K}_{s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{q}^{(sk)}\right),$$
$$\text{span}\left(\boldsymbol{R}^{(k)}\right) = \mathcal{K}_s\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{r}^{(sk)}\right).$$

Subsequently, using a single global reduction operation, the Gram matrix $\boldsymbol{G}^{(k)} = {\boldsymbol{Y}^{(k)}}^T\boldsymbol{Z}^{(k)}$ of size $(2s+1) \times (2s+1)$ is computed.

To compute the inner iterations in the changed basis, we define small vectors $\boldsymbol{q}^{(sk+j)'}$, $\boldsymbol{r}^{(sk+j)'}$ and $\boldsymbol{x}^{(sk+j)'}$ for $0 \leq j \leq s$, each of length $2s+1$, such that

$$\boldsymbol{q}^{(sk+j)} = \boldsymbol{Y}^{(k)}\boldsymbol{q}^{(sk+j)'}, \boldsymbol{p}^{(sk+j)} = \boldsymbol{Z}^{(k)}\boldsymbol{q}^{(sk+j)'}, \tag{2.12}$$
$$\boldsymbol{r}^{(sk+j)} = \boldsymbol{Y}^{(k)}\boldsymbol{r}^{(sk+j)'}, \boldsymbol{u}^{(sk+j)} = \boldsymbol{Z}^{(k)}\boldsymbol{r}^{(sk+j)'}, \tag{2.13}$$
$$\boldsymbol{x}^{(sk+j)} = \boldsymbol{x}^{(sk)} + \boldsymbol{Z}^{(k)}\boldsymbol{x}^{(sk+j)'}. \tag{2.14}$$

The next s steps are computed by updating these small vectors analogously to the recursive update equations in PCG. The vectors in the original basis are regained using (2.12)–(2.14) at the end of each outer iteration.

Define the "change-of-basis" matrix $\boldsymbol{B}$ of size $(2s+1) \times (2s+1)$ such that

$$\boldsymbol{A}\underline{\boldsymbol{Z}}^{(k)} = \boldsymbol{Y}^{(k)}\boldsymbol{B}, \tag{2.15}$$

with $\underline{\boldsymbol{Z}}^{(k)} = \left[\underline{\boldsymbol{P}}^{(k)}, \underline{\boldsymbol{U}}^{(k)}\right]$, where $\underline{\boldsymbol{P}}^{(k)}$ and $\underline{\boldsymbol{U}}^{(k)}$ are the same as $\boldsymbol{P}^{(k)}$ and $\boldsymbol{U}^{(k)}$ except their respective last column is all zeros. Using the definition in (2.4), the matrix $\boldsymbol{B}$ is defined as

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{B}_{s+1} & 0_{s+1,1} & 0_{s+1,s-1} & 0_{s+1,1} \\ 0_{s,s} & 0_{s,1} & \boldsymbol{B}_s & 0_{s,1} \end{bmatrix}. \tag{2.16}$$

Using this matrix, the global MV products of standard PCG can inexpensively be computed in the changed basis with $\boldsymbol{B}\boldsymbol{q}^{(sk+j)'}$.

3 P-sPCG with arbitrary basis types P-sPCG$_{\text{mon}}$, as sPCG$_{\text{mon}}$, can only use the monomial

basis. We generalize P-sPCG$_{\text{mon}}$ so that it is possible to improve its numerical stability by using other basis types, denoting our new algorithm as P-sPCG.

We define extended s-step basis matrices $\boldsymbol{S}_{ext}^{(k)}$ and $\boldsymbol{T}_{ext}^{(k)}$ which are equivalent to the MPK basis matrices in (2.1) and (2.2) for the step size $2s$, i.e.,

$$\text{span}\left(\boldsymbol{S}_{ext}^{(k)}\right) = \mathcal{K}_{2s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{r}^{(sk)}\right),$$
$$\text{span}\left(\boldsymbol{T}_{ext}^{(k)}\right) = \mathcal{K}_{2s+1}\left(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{u}^{(sk)}\right).$$

The recursive computations of the basis matrices $\boldsymbol{S}^{(k+1)}$ and $\boldsymbol{T}^{(k+1)}$ at the end of iteration k in (2.8) and (2.9) are replaced by

$$\begin{aligned} P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{r}^{(k+1)} &= P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{r}^{(k)} \\ &\quad - P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{P}^{(k)}\boldsymbol{a}^{(k)}, \\ P_l(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{u}^{(k+1)} &= P_l(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{u}^{(k)} \\ &\quad - P_l(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{P}^{(k)}\boldsymbol{a}^{(k)} \end{aligned}$$

for $l = 0, \ldots, s$. For obtaining $P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{P}^{(k)}$ and $P_l(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{M}^{-1}\boldsymbol{A}\boldsymbol{P}^{(k)}$, we use the following recursive computations, analogously to (2.10) and (2.11). For $l = 0, \ldots, s$,

$$\begin{aligned} P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{P}^{(k)} &= P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{U}^{(k)} \\ &\quad + P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}, \end{aligned} \tag{3.1}$$

$$\begin{aligned} &P_l(\boldsymbol{M}^{-1}\boldsymbol{A})(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{P}^{(k)} \\ &\quad = P_l(\boldsymbol{M}^{-1}\boldsymbol{A})(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{U}^{(k)} \\ &\quad + P_l(\boldsymbol{M}^{-1}\boldsymbol{A})(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{P}^{(k-1)}\boldsymbol{B}^{(k)}. \end{aligned} \tag{3.2}$$

When using the monomial basis, $P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{U}^{(k)}$ are simply parts of $\boldsymbol{S}_{ext}^{(k)}$. For other basis types, these matrices must be computed explicitly. We define a "change-of-basis" matrix $\boldsymbol{B}_{ext} = \boldsymbol{B}_{2s+1}$ (see (2.4)) such that

$$\boldsymbol{A}\boldsymbol{U}_{ext}^{(k)} = \boldsymbol{A}\boldsymbol{M}^{-1}\boldsymbol{R}_{ext}^{(k)} = \boldsymbol{S}_{ext}^{(k)}\boldsymbol{B}_{ext}, \tag{3.3}$$

where $\boldsymbol{R}_{ext}^{(k)}$ and $\boldsymbol{U}_{ext}^{(k)}$ are the first $2s$ columns of $\boldsymbol{S}_{ext}^{(k)}$ and $\boldsymbol{T}_{ext}^{(k)}$, respectively. The first s columns of $\boldsymbol{S}_{ext}^{(k)}\boldsymbol{B}_{ext}$ are $\boldsymbol{A}\boldsymbol{U}^{(k)} = P_0(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{U}^{(k)}$.

Since this "change-of-basis" matrix $\boldsymbol{B}_{ext}$ is not square, we define a new Matrix Powers Kernel based on a polynomial of degree l for a matrix z of size $h \times (h-1)$ that satisfies the three-term recurrence relation

$$\begin{aligned} P_0'(z) &= \boldsymbol{I}_{h,h}, \\ P_1'(z) &= 1/\gamma_0(z - \theta_0\boldsymbol{I}_{h,h-1})P_0'(z)_{1:h-1,1:h-1}, \\ P_l'(z) &= \frac{1}{\gamma_{l-1}}\left((z-\theta_{l-1}\boldsymbol{I}_{h,h-1})P_{l-1}'(z)_{1:h-1,1:h-l}\right. \\ &\quad \left.+\mu_{l-2}P_{l-2}'(z)_{1:h,1:h-l}\right), \quad l \geq 2. \end{aligned} \tag{3.4}$$

PROPOSITION 3.1. *For $l = 0, \ldots, s$,*

$$\begin{aligned} &P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{U}_{ext\,1:(2s-l)}^{(k)} = \\ &\boldsymbol{S}_{ext}^{(k)}P_l'(\boldsymbol{B}_{ext})\boldsymbol{B}_{ext\,1:(2s-l+1),1:(2s-l)}. \end{aligned} \tag{3.5}$$

Proof. By induction. $l = 0$ corresponds to (3.3). For $l = 1$, we have

$$\begin{aligned} &(1/\gamma_0)\left(\boldsymbol{A}\boldsymbol{M}^{-1} - \sigma_0\boldsymbol{I}_{n,n}\right)\boldsymbol{A}\boldsymbol{U}_{ext\,1:(2s-1)}^{(k)} \\ &= (1/\gamma_0)\left(\boldsymbol{A}\boldsymbol{U}_{ext}^{(k)} - \sigma_0\boldsymbol{R}_{ext}^{(k)}\right)\boldsymbol{B}_{ext\,1:2s,1:(2s-1)} \\ &= (1/\gamma_0)\boldsymbol{S}_{ext}^{(k)}\left(\boldsymbol{B}_{ext} - \sigma_0\boldsymbol{I}_{2s+1,2s}\right)\boldsymbol{B}_{ext\,1:2s,1:(2s-1)}. \end{aligned}$$

For $l > 1$, we replace both sides of (3.5) with the recursive definitions in (2.3) respectively (3.4), and compare the terms on both sides individually. Assuming that (3.5) holds for $l-2$ and $l-1$, the proof for most terms becomes trivial, except for

$$\begin{aligned} &\boldsymbol{A}\boldsymbol{M}^{-1}P_{l-1}(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{U}_{ext\,1:(2s-l)}^{(k)} \\ &= \boldsymbol{S}_{ext}^{(k)}\boldsymbol{B}_{ext}P_{l-1}'(\boldsymbol{B}_{ext})_{1:2s,1:(2s-l+1)} \\ &\quad \boldsymbol{B}_{ext\,1:(2s-l+1),1:(2s-l)}. \end{aligned} \tag{3.6}$$

We prove (3.6) as follows. Since $\boldsymbol{B}_{ext}$ is tridiagonal, only the first $2s-l+1$ rows of $\boldsymbol{B}_{ext\,1:(2s-l)}$ have non-zeros. Using (3.3), we replace the left-hand side of (3.6) by

$$\begin{aligned} &\boldsymbol{A}\boldsymbol{M}^{-1}P_{l-1}(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{R}_{ext\,1:(2s-l+1)}^{(k)}\boldsymbol{B}_{ext\,1:(2s-l+1),1:(2s-l)} \\ &= P_{l-1}(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{U}_{ext\,1:(2s-l+1)}^{(k)}\boldsymbol{B}_{ext\,1:(2s-l+1),1:(2s-l)}, \end{aligned}$$

since by induction, $P_l(z)z = zP_l(z)$. Using (3.5) for $l-1$, the above term becomes

$$\begin{aligned} &\boldsymbol{S}_{ext}^{(k)}P_{l-1}'(\boldsymbol{B}_{ext})\boldsymbol{B}_{ext\,1:(2s-l+2),1:(2s-l+1)} \\ &\boldsymbol{B}_{ext\,1:(2s-l+1),1:(2s-l)}. \end{aligned} \tag{3.7}$$

For $z \in \mathbb{R}^{h\times(h-1)}$, by induction $P_l'(z)z_{1:(h-l),1:(h-l)} = zP_l'(z)_{1:(h-1),1:(h-l)}$. Thus, (3.7) corresponds to the right-hand side of (3.6). ∎

We use Proposition 3.1 to obtain $P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{A}\boldsymbol{U}^{(k)}$ for (3.1), as this matrix corresponds to the first s columns of (3.5). Since $P_l'(\boldsymbol{B}_{ext})$ is applied to $\boldsymbol{S}_{ext}^{(k)}$ from the right, the explicit computation of $P_l'(\boldsymbol{B}_{ext})\boldsymbol{B}_{ext\,1:(2s+1-l),1:(2s-l)}$ for $l = 0, \ldots, s$ is required. These $\mathcal{O}(s) \times \mathcal{O}(s)$ matrices are computed once at the beginning of the solver and are also applied to $\boldsymbol{T}_{ext}^{(k)}$ to obtain $P_l(\boldsymbol{M}^{-1}\boldsymbol{A})(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{U}^{(k)}$ for computing $\boldsymbol{T}^{(k+1)}$ recursively.

P-sPCG computes $\boldsymbol{a}^{(k)}$ and $\boldsymbol{B}^{(k)}$ exactly like sPCG. The required "change-of-basis" matrix $\boldsymbol{B}$ is a submatrix

of $\boldsymbol{B}_{ext}$ consisting of the elements that are both in the first $s+1$ rows and first s columns of $\boldsymbol{B}_{ext}$.

We omit a complete algorithm listing for P-sPCG for arbitrary basis types due to space restrictions.

4 Pipelined CA-PCG (P-CA-PCG) Analogously to P-sPCG$_{\text{mon}}$ and P-sPCG, our new Pipelined CA-PCG variant P-CA-PCG uses extended s-step basis matrices to compute the basis matrices of CA-PCG recursively. This enables overlapping the global communication with MV products and preconditioner applications.

Define the extended s-step basis matrices $\boldsymbol{Y}_{ext}^{(k)}$ and $\boldsymbol{Z}_{ext}^{(k)} = \boldsymbol{M}^{-1}\boldsymbol{Y}_{ext}^{(k)}$ that span the subspaces

$$\mathcal{K}_{2s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{q}^{(sk)}\right) + \mathcal{K}_{2s}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{r}^{(sk)}\right),$$
$$\mathcal{K}_{2s+1}\left(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{p}^{(sk)}\right) + \mathcal{K}_{2s}\left(\boldsymbol{M}^{-1}\boldsymbol{A}, \boldsymbol{u}^{(sk)}\right),$$

respectively. We extend the relation in (2.15) by defining a "change-of-basis" matrix $\boldsymbol{B}_{ext}$ of size $(4s+1)\times(4s+1)$ such that

$$\boldsymbol{A}\underline{\boldsymbol{Z}}_{ext}^{(k)} = \boldsymbol{Y}_{ext}^{(k)}\boldsymbol{B}_{ext}, \tag{4.1}$$

where $\underline{\boldsymbol{Z}}_{ext}^{(k)} = \boldsymbol{M}^{-1}\underline{\boldsymbol{Y}}_{ext} = \left[\underline{\boldsymbol{P}}_{ext}^{(k)}, \underline{\boldsymbol{U}}_{ext}^{(k)}\right]$, with $\underline{\boldsymbol{P}}_{ext}^{(k)}$ and $\underline{\boldsymbol{U}}_{ext}^{(k)}$ being defined analogously to $\underline{\boldsymbol{P}}^{(k)}$ and $\underline{\boldsymbol{U}}^{(k)}$, i.e., their respective last column is zero. The matrix $\boldsymbol{B}_{ext}$ is equivalent to $\boldsymbol{B}$ in (2.16) for the step size $2s$.

Moreover, define the small vectors

$$\boldsymbol{q}_{ext}^{(sk+s)'} = \left[\boldsymbol{q}_{1:(s+1)}^{(sk+s)'}, 0_{1,s}, \boldsymbol{q}_{(s+2):(2s+1)}^{(sk+s)'}, 0_{1,s}\right]^T,$$
$$\boldsymbol{r}_{ext}^{(sk+s)'} = \left[\boldsymbol{r}_{1:(s+1)}^{(sk+s)'}, 0_{1,s}, \boldsymbol{r}_{(s+2):(2s+1)}^{(sk+s)'}, 0_{1,s}\right]^T$$

of length $4s+1$. Equivalently to (2.12) and (2.13),

$$\boldsymbol{q}^{(sk+s)} = \boldsymbol{Y}_{ext}^{(k)}\boldsymbol{q}_{ext}^{(sk+s)'}, \boldsymbol{p}^{(sk+s)} = \boldsymbol{Z}_{ext}^{(k)}\boldsymbol{q}_{ext}^{(sk+s)'}, \tag{4.2}$$
$$\boldsymbol{r}^{(sk+s)} = \boldsymbol{Y}_{ext}^{(k)}\boldsymbol{r}_{ext}^{(sk+s)'}, \boldsymbol{u}^{(sk+s)} = \boldsymbol{Z}_{ext}^{(k)}\boldsymbol{r}_{ext}^{(sk+s)'}. \tag{4.3}$$

LEMMA 4.1. *Elements $s+2+l$ to $2s+1$ and $3s+2+l$ to $4s+1$ of the vectors $P_l(\boldsymbol{B}_{ext})\boldsymbol{q}_{ext}^{(sk+s)'}$ and $P_l(\boldsymbol{B}_{ext})\boldsymbol{r}_{ext}^{(sk+s)'}$ are zero.*

Proof. By induction, using the definition of $\boldsymbol{q}_{ext}^{(sk+s)'}$ and $\boldsymbol{r}_{ext}^{(sk+s)'}$, the tridiagonality of $\boldsymbol{B}_{ext}$ and the fact that columns $2s+1$ and $4s+1$ of $\boldsymbol{B}_{ext}$ are zero. ∎

PROPOSITION 4.2. *Multiplying the search direction vectors in (4.2) with $P_l(\boldsymbol{A}\boldsymbol{M}^{-1})$ respectively $P_l(\boldsymbol{M}^{-1}\boldsymbol{A})$, for $l = 0, \ldots, s$,*

$$P_l(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{q}^{(sk+s)} = \boldsymbol{Y}_{ext}^{(k)}P_l(\boldsymbol{B}_{ext})\boldsymbol{q}_{ext}^{(sk+s)'}, \tag{4.4}$$
$$P_l(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{p}^{(sk+s)} = \boldsymbol{Z}_{ext}^{(k)}P_l(\boldsymbol{B}_{ext})\boldsymbol{q}_{ext}^{(sk+s)'}. \tag{4.5}$$

Proof. (4.5) follows from (4.4). We prove (4.4) by induction. $l=0$ corresponds to (4.2). For $l=1$, using (4.1), (4.2) and Lemma 4.1, we have

$$(1/\gamma_0)(\boldsymbol{A}\boldsymbol{M}^{-1} - \theta_0\boldsymbol{I}_{n,n})\boldsymbol{q}^{(sk+s)}$$
$$= (1/\gamma_0)\boldsymbol{Y}_{ext}^{(k)}\left(\boldsymbol{B}_{ext} - \theta_0\boldsymbol{I}_{4s+1,4s+1}\right)\boldsymbol{q}_{ext}^{(sk+s)'}.$$

For $l > 1$, we replace both sides of (4.4) with the recursive definition in (2.3), and compare the terms on both sides individually. Assuming that (4.4) holds for $l-2$ and $l-1$, the proof becomes trivial for most terms, except for

$$\begin{aligned}&\boldsymbol{A}\boldsymbol{M}^{-1}P_{l-1}(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{q}^{(sk+s)}\\ &= \boldsymbol{Y}_{ext}^{(k)}\boldsymbol{B}_{ext}P_{l-1}(\boldsymbol{B}_{ext})\boldsymbol{q}_{ext}^{(sk+s)'}.\end{aligned} \tag{4.6}$$

We prove (4.6) using (4.4) for $l-1$. From Lemma 4.1, $\boldsymbol{Y}_{ext}^{(k)}$ in (4.4) can be replaced by $\underline{\boldsymbol{Y}}_{ext}^{(k)}$ if $l < s$.

$$\begin{aligned}&\boldsymbol{A}\boldsymbol{M}^{-1}P_{l-1}(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{q}^{(sk+s)}\\ &= \boldsymbol{A}\boldsymbol{M}^{-1}\underline{\boldsymbol{Y}}_{ext}P_{l-1}(\boldsymbol{B}_{ext})\boldsymbol{q}_{ext}^{(sk+s)'}.\end{aligned} \tag{4.7}$$

Using (4.1), it follows that (4.7) coincides with the right-hand side of (4.6). ∎

(4.4) and (4.5) are the first $s+1$ columns of $\boldsymbol{Y}^{(k+1)}$ and $\boldsymbol{Z}^{(k+1)}$, respectively. Thus, the MPK can be performed on the small "change-of-basis" vector $\boldsymbol{q}_{ext}^{(sk+s)'}$ with the matrix $\boldsymbol{B}_{ext}$. Analogously, for the remaining columns of $\boldsymbol{Y}^{(k+1)}$ and $\boldsymbol{Z}^{(k+1)}$, the MPK is performed on the vector $\boldsymbol{r}_{ext}^{(sk+s)'}$. We store the resulting vectors in a "change-of-basis" s-step basis matrix

$$\boldsymbol{Y'}^{(k+1)} = \Big[P_0(\boldsymbol{B}_{ext})\boldsymbol{q}^{(sk+s)'}, \ldots, P_s(\boldsymbol{B}_{ext})\boldsymbol{q}^{(sk+s)'},$$
$$P_0(\boldsymbol{B}_{ext})\boldsymbol{r}^{(sk+s)'}, \ldots, P_s(\boldsymbol{B}_{ext})\boldsymbol{r}^{(sk+s)'}\Big]$$

of size $(4s+1)\times(2s+2)$. Note that we compute two additional vectors based on the residual with negligible cost, which are not required for initiating the global reduction operation of $\boldsymbol{G}^{(k+1)}$.

$$\left[\boldsymbol{Y}^{(k+1)}, P_s(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{r}^{(sk+s)}\right] = \boldsymbol{Y}_{ext}^{(k)}\boldsymbol{Y'}^{(k+1)}$$
$$\left[\boldsymbol{Z}^{(k+1)}, P_s(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{u}^{(sk+s)}\right] = \boldsymbol{Z}_{ext}^{(k)}\boldsymbol{Y'}^{(k+1)}$$

By recursively computing these two vectors, one less MV product and preconditioner application are required when extending the s-step basis matrices. Thus, like CA-PCG, P-CA-PCG computes $2s-1$ MV products and preconditioner applications per outer iteration.

P-CA-PCG for arbitrary basis types is outlined in Algorithm 4.1. Since we perform the MPK on the

Algorithm 4.1 P-CA-PCG (Pipelined CA-PCG)

1: $\boldsymbol{q}^{(0)} = \boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}, \boldsymbol{p}^{(0)} = \boldsymbol{u}^{(0)} = \boldsymbol{M}^{-1}\boldsymbol{r}^{(0)}$
2: $\text{span}\left(\left[\boldsymbol{Y}^{(0)}, P_s(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{r}^{(0)}\right]\right) =$
$\mathcal{K}_{s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{q}^{(0)}\right) + \mathcal{K}_{s+1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, \boldsymbol{r}^{(0)}\right)$
3: $\left[\boldsymbol{Z}^{(0)}, P_s(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{u}^{(0)}\right] =$
$\boldsymbol{M}^{-1}\left[\boldsymbol{Y}^{(0)}, P_s(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{r}^{(0)}\right]$
4: **for** $k = 0, 1, ...$ until convergence **do**
5: $\quad \boldsymbol{G}^{(k)} = \boldsymbol{Z}^{(k)^T}\boldsymbol{Y}^{(k)}$ // Finalize after line 7
6: $\quad$ Compute remaining columns of $\boldsymbol{Y}_{ext}^{(k)}$ that span
$\mathcal{K}_s\left(\boldsymbol{A}\boldsymbol{M}^{-1}, (\boldsymbol{A}\boldsymbol{M}^{-1})^{s+1}\boldsymbol{q}^{(sk)}\right) +$
$\mathcal{K}_{s-1}\left(\boldsymbol{A}\boldsymbol{M}^{-1}, (\boldsymbol{A}\boldsymbol{M}^{-1})^{s+1}\boldsymbol{r}^{(sk)}\right)$
7: $\quad$ Compute remaining columns of $\boldsymbol{Z}_{ext}^{(k)}$ that span
$\mathcal{K}_s\left(\boldsymbol{M}^{-1}\boldsymbol{A}, (\boldsymbol{M}^{-1}\boldsymbol{A})^{s+1}\boldsymbol{p}^{(sk)}\right) +$
$\mathcal{K}_{s-1}\left(\boldsymbol{M}^{-1}\boldsymbol{A}, (\boldsymbol{M}^{-1}\boldsymbol{A})^{s+1}\boldsymbol{u}^{(sk)}\right)$
8: $\quad$ Inner iterations, see Algorithm 2.4
9: $\quad \boldsymbol{x}^{(sk+s)} = \boldsymbol{x}^{(sk)} + \boldsymbol{Z}^{(k)'}\boldsymbol{x}^{(sk+s)'}$
10: $\quad \boldsymbol{q}_{ext}^{(sk+s)'} = \left[\boldsymbol{q}_{1:(s+1)}^{(sk+s)'}, 0_{1,s}, \boldsymbol{q}_{(s+2):(2s+1)}^{(sk+s)'}, 0_{1,s}\right]^T$
11: $\quad \boldsymbol{r}_{ext}^{(sk+s)'} = \left[\boldsymbol{r}_{1:(s+1)}^{(sk+s)'}, 0_{1,s}, \boldsymbol{r}_{(s+2):(2s+1)}^{(sk+s)'}, 0_{1,s}\right]^T$
12: $\quad$ Local MPK on small vectors: $\boldsymbol{Y'}^{(k+1)} =$
$\left[P_0(\boldsymbol{B}_{ext})\boldsymbol{q}^{(sk+s)'}, \ldots, P_s(\boldsymbol{B}_{ext})\boldsymbol{q}^{(sk+s)'},\right.$
$\left.P_0(\boldsymbol{B}_{ext})\boldsymbol{r}^{(sk+s)'}, \ldots, P_s(\boldsymbol{B}_{ext})\boldsymbol{r}^{(sk+s)'}\right]$
13: $\quad \left[\boldsymbol{Y}^{(k+1)}, P_s(\boldsymbol{A}\boldsymbol{M}^{-1})\boldsymbol{r}^{(sk+s)}\right] = \boldsymbol{Y}_{ext}^{(k)}\boldsymbol{Y'}^{(k+1)}$
14: $\quad \left[\boldsymbol{Z}^{(k+1)}, P_s(\boldsymbol{M}^{-1}\boldsymbol{A})\boldsymbol{u}^{(sk+s)}\right] = \boldsymbol{Z}_{ext}^{(k)}\boldsymbol{Y'}^{(k+1)}$
15: **end for**

small "change-of-basis" vectors, the overhead for using arbitrary basis types compared to the monomial basis is negligible.

5 Theoretical analysis

In this section, we theoretically analyze the performance of the s-step methods sPCG$_{\text{mon}}$/sPCG, CA-PCG and CA-PCG3 as well as our new pipelined s-step methods P-CA-PCG and P-sPCG. Moreover, we provide a comparative analysis with the pipelined PCG methods PPCG [12] and P&R-PPCG [7]. We argue why our new pipelined s-step solver P-CA-PCG is beneficial in terms of performance.

5.1 Communication reduction

All discussed communication-avoiding solvers only require one global reduction operation per s steps. Therefore, they reduce the number of global collectives by a factor of $2s$ compared to standard PCG, which performs two global synchronization points per iteration.

The pipelined s-step methods P-sPCG$_{\text{mon}}$, P-sPCG and P-CA-PCG overlap this global communication with the computation of all $\mathcal{O}(s)$ MV products and preconditioner applications of the current s steps, further reducing the impact of global communication.

5.2 Computational cost

The computational cost for each algorithm is listed in Table 5.1.

5.2.1 MV products and preconditioner applications

CA-PCG and P-CA-PCG compute $2s-1$ MV products and preconditioner applications per s steps. The remaining considered solvers only require s per s steps. Even though the amount of floating-point operations (FLOPs) for the MPK kernel is almost doubled, the additional MV products and preconditioner applications of CA-PCG and P-CA-PCG can be computed in blocks to avoid communication latency and memory sweeps of the system matrix's and the preconditioners data.

While these additional computations make CA-PCG only suitable for rather simple and inexpensive preconditioners, P-CA-PCG overlaps these computations with global communication, reducing or even eliminating this overhead if communication is expensive.

5.2.2 Other computations

Recursively updating the search direction matrix in sPCG requires $\mathcal{O}(s^2n)$ FLOPs due to the multiplication of an $n \times s$ matrix with the $s \times s$ matrix $\boldsymbol{B}^{(k)}$. P-sPCG recursively updates $\mathcal{O}(s)$ matrices based on $\boldsymbol{P}^{(k)}$ (each with s columns), thus the cost is $\mathcal{O}(s^3n)$. This makes the usually negligible local vector operations a significant part of the runtime for larger values of s. Note that this cost (column 4 in Table 5.1) is identical for P-sPCG$_{\text{mon}}$ and P-sPCG.

Contrarily, in CA-PCG, the cost of recovering the full vectors from the small "change-of-basis" vectors at the end of an outer CA-PCG iteration requires only $\mathcal{O}(sn)$ FLOPs. The cost for recovering the entire s-step basis of the next outer iteration in P-CA-PCG is therefore $\mathcal{O}(s^2n)$.

Column 5 in Table 5.1 lists the additional cost when using a basis type with maximum possible cost where all parameters $\theta^{(j)}$ and $\mu^{(j)}$ in (2.3) are non-zero, and all parameters $\gamma^{(j)} \neq 1$ (e.g., the Chebyshev basis). The recursive computation of the s-step basis matrix parts in P-sPCG and P-CA-PCG require an MPK execution on matrices with $\mathcal{O}(s)$ columns (see Propositions 3.1 and 4.2). For P-sPCG, it is executed on $n \times \mathcal{O}(s)$ matrices, making the local vector computations even more expensive. In P-CA-PCG, these computations are performed on the small $\mathcal{O}(s) \times \mathcal{O}(s)$ matrices in the changed basis before recovering the full basis matrix columns, making the cost negligible. Note that these recursive MPK computations require the result of the global reduction operation of the current (outer) iteration and can therefore not be overlapped with global communication.

5.3 Summary of (pipelined) s-step methods

Choosing an inexpensive preconditioner is crucial for the

Table 5.1: 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). Arbitrary basis type: we list the maximum possible cost where all parameters $\theta^{(j)}$ and $\mu^{(j)}$ in (2.3) are non-zero, and all parameters $\gamma^{(j)} \neq 1$.

Algorithm	#MV + #prec. a.	Remaining #FLOPs/n (beyond MV and prec. appl.)			
		Local reduct.	Vector/Matrix column computations for monomial b.	Addit. for arb. b.	Total remain. #FLOPs/n Arbitrary basis
PCG	s	$2s$	$6s$	-	8s
CA-PCG	$2s-1$	$(2s+1)^2$	$20s+6$	$10s-9$	$4s^2+34s-2$
P-CA-PCG	$2s-1$	$(2s+1)^2$	$32s^2+24s+4$	$10s-5$	$32s^2+34s-1$
sPCG	s	$2s(s+1)$	$4s^2+4s$	$10s-4$	$6s^2+16s-4$
P-sPCG	s	$2s(s+1)$	$4s^3+10s^2+6s$	$\frac{11}{6}s^3+\frac{15}{2}s^2+\frac{2}{3}s$	$\sim 6s^3+20s^2+9s$
CA-PCG3	s	$(2s+1)^2$	$8s^2+17s$	$5s-2$	$12s^2+26s-1$
PPCG	s	$2s$	$16s$	-	$18s$
P&R-PPCG	$2s$	$4s$	$16s$	-	$20s$

performance of CA-PCG due to its additional MV products and preconditioner applications. In P-CA-PCG, this overhead is overlapped with global communication, reducing or even eliminating this additional cost if communication is expensive. Thus, unlike CA-PCG, P-CA-PCG can be used efficiently with expensive preconditioners if the influence of global communication is large enough.

Regarding local vector/matrix column operations, P-CA-PCG and the s-step PCG methods require a cost of $\mathcal{O}(s^2 n)$ FLOPs quadratic in s, whereas P-sPCG requires $\mathcal{O}(s^3 n)$ FLOPs. Thus, with sufficient parallelism and the resulting strong influence of global communication, our novel method P-CA-PCG is superior to all the other (pipelined) s-step PCG variants considered.

5.4 Comparison with pipelined methods The pipelined methods PPCG and P&R-PPCG overlap global communication with one respectively two MV product(s) and preconditioner application(s) while the pipelined s-step methods overlap it with $\mathcal{O}$ (s) MV products and preconditioner applications. Assuming high communication cost, these pipelined methods thus only improve scalability if the MV product and the preconditioner application are sufficiently expensive. Moreover, PPCG and P&R-PPCG only halve the number of global collectives. Contrarily, pipelined s-step methods reduce the number of synchronization steps by a factor of $\mathcal{O}$ (s) and overlap the fewer remaining global reductions with more local work, thus giving more potential to reduce the impact of global communication. In terms of local vector computations, the pipelined methods are less expensive than all considered (pipelined) s-step methods.

6 Experimental evaluation We experimentally evaluate the runtime performance of all considered pipelined, s-step and pipelined s-step methods. Moreover, we investigate the influence of different basis types on numerical stability for these solvers.

6.1 Implementation and experimental setup We implemented the algorithms in C++ using MPI. We used Trilinos 16.0.0 [19], 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, specifically VSC-5, which has dual-socket 64-core AMD EPYC 7713 (Milan) compute nodes connected via Mellanox HDR infiniband (200 Gbit/s). Matrices of size $n \times n$ are block-row distributed, vectors of length n are distributed accordingly. The right-hand side $\boldsymbol{b}$ was chosen such that each entry of the solution vector $\boldsymbol{x}$ is $1/\sqrt{n}$. The start vector $\boldsymbol{x}^{(0)}$ is a zero vector.

We ran the algorithms on 1 to 128 nodes, with 128 processes per node. For each execution variant (different numbers of nodes, different values of s, different basis types), we show the median runtime over 10 test runs.

Jacobi or Chebyshev preconditioners were used in the experiments, since they require little or no communication and are thus suitable for scalable PCG methods. The largest and smallest eigenvalues required for the Chebyshev basis type and the Chebyshev preconditioner were estimated by executing a few iterations of standard PCG (not included in the runtimes).

When used in communication-avoiding/-hiding methods, preconditioners should not require much communication themselves. A global Incomplete Cholesky or Incomplete LU preconditioner potentially requires much communication and becomes the major communication bottleneck instead of the global scalar products. Using a local Incomplete Cholesky preconditioner on the

diagonal blocks owned solely by each rank is comparable to (but less accurate than) a Block-Jacobi preconditioner. We used a Jacobi preconditioner for the Poisson matrix as using large block sizes is not suitable for this very sparse matrix (only 7 non-zeros/row).

Generally, an expensive preconditioner (in terms of communication and/or local computation) tends to contradict the focus on the global communication of scalar products as the main bottleneck. For the stability experiments, we used a Chebyshev preconditioner, which only requires communication for MV products with the system matrix $\boldsymbol{A}$ and thus fulfills the requirement of being communication-avoiding.

6.2 Numerical stability Table 6.1 lists all SPD test matrices available in the SuiteSparse Matrix Collection [10] with a problem size between 100000 and 2000000 that converged within 10000 iterations of standard PCG with a Chebyshev preconditioner with degree 3 when executed on one node (128 processes). For the s-step and pipelined s-step methods, we used the step size $s = 10$. All 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}. Table 6.1 contains the number of iterations required to achieve the desired accuracy for each solver considered.

The s-step and pipelined s-step methods evaluate the convergence criterion only every s steps. We consider less than 20% iteration overhead or less than $s = 10$ 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.

Results with the monomial basis for the s-step methods can be found in [14, Table 2], where the same 40 matrices and the same setup were used for a comparison of existing s-step methods. We omitted results for the pipelined s-step methods with the monomial basis in Table 6.1, since with this basis P-CA-PCG only converged for two of the 40 matrices (thermomech_TC with 20 iterations and thermomech_dM with 250 iterations) and P-sPCG did not converge for any of the 40 matrices. This underlines the importance of a suitable basis type.

6.2.1 (Pipelined) s-step methods Table 6.1 shows that CA-PCG has the best numerical stability by converging for 35 out of the 40 test matrices (87.5%) using the Chebyshev basis. It even converged for 23 of the 40 test matrices when using the monomial basis, although only six of these matrices had a convergence speed similar to standard PCG (see [14]). sPCG and P-CA-PCG only converged for two matrices using the monomial basis, while P-sPCG did not converge for any matrix with this basis type.

Using the Chebyshev basis improves the stability of these algorithms immensely, with P-CA-PCG converging for 31 (more than 75%) and P-sPCG for 19 out of the 40 test matrices (almost 50%). sPCG showed the same convergence behavior as P-sPCG, converging for the same 19 test matrices using the Chebyshev basis. CA-PCG3 showed similar convergence behavior by converging for 21 matrices (see [14]).

Using the Chebyshev basis, P-CA-PCG converged for fewer matrices than CA-PCG. However, for the four matrices where CA-PCG did converge and P-CA-PCG did not, CA-PCG had an iteration overhead of at least 14%, for one of them over 40% and for one over 250%. Thus, P-CA-PCG only failed in the cases where CA-PCG was at least slightly less stable than standard PCG as well. This raises the question whether the numerical aspects of P-CA-PCG can be further improved by adapting existing strategies to improve stability and the maximum attainable accuracy in CA-PCG, as discussed, e.g., in [3, 2, 5]. We leave such an investigation for future work.

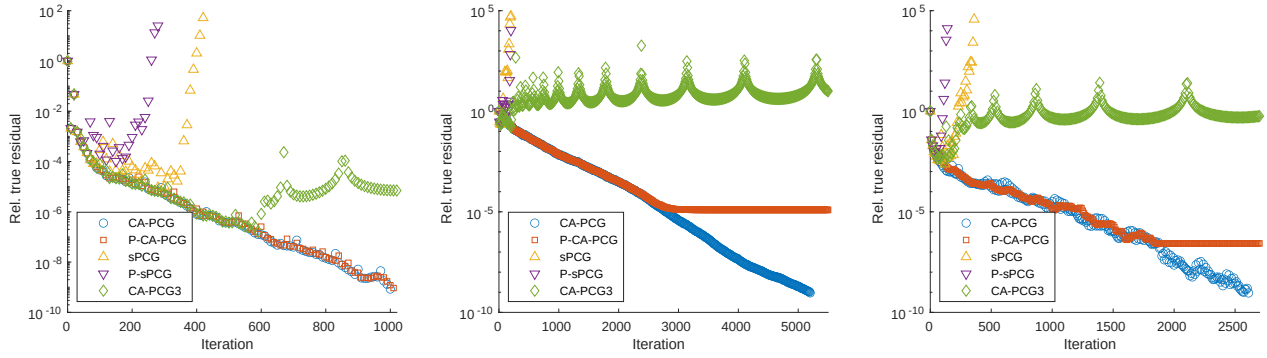
6.2.2 Pipelined methods PPCG and P&R-PPCG showed better numerical stability than the (pipelined) s-step methods. The former enhance the Krylov subspace by one additional dimension in each iteration to enable overlapping of global communication, while the latter enhance the Krylov subspace by s dimensions at once, which potentially introduces more stability issues with increasing s. As expected, P&R-PPCG was more stable than PPCG in our experiments.

6.2.3 Influence of matrix properties We estimated the condition number of the preconditioned matrix $\boldsymbol{M}^{-1}\boldsymbol{A}$ with Trilinos' Belos Standard PCG solver (terminated after the relative residual norm had been reduced by a factor of 10^{-9}). The results listed in Table 6.1 show the relation between the condition number and the convergence speed of standard PCG, i.e., the matrices with very few iterations also have a very small condition number. However, our results do not indicate an influence of the condition number on the stability of the (pipelined) s-step methods, e.g., the matrix Serena has a smaller condition number and fewer iterations than many other matrices, but no (pipelined) s-step method converged. Note that the estimates of these condition numbers are (potentially) only very coarse.

We also list the largest and smallest eigenvalue that had been estimated with a few iterations of standard PCG for each matrix (for using the Chebyshev basis). For matrices with significantly higher largest eigenvalue than most other matrices (Serena, Fault_639), no (pipelined) s-step method converged. However, other

Table 6.1: All SPD matrices from SuiteSparse Matrix Collection of problem size $100000-2000000$, where standard PCG converged within 10000 iterations. Chebyshev preconditioner (degree 3), $s = 10$, Chebyshev basis, 128 processes. If iteration count differed: left: s-step method, right: pipelined s-step method. Convergence criterion: $\|(b - \boldsymbol{A}\boldsymbol{x}^{(k)})\|_2/\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(0)}\|_2 < 10^{-9}$. Hyphen: algorithm diverged/residual stagnated before reaching desired accuracy/convergence was not achieved within 12000 iterations. Bold: $< 20\%$ iteration overhead or $< s$ extra iterations compared to standard PCG. "cond. est.": estimated condition number of the preconditioned matrix $\boldsymbol{M}^{-1}\boldsymbol{A}$ approximated with Belos' PCG implementation. "max./min. eig.": Largest and smallest eigenvalue of the preconditioned matrix estimated with a few iterations of standard PCG.

Matrix	cond. est.	max. eig.	min. eig.	PCG	(P-)CA-PCG	(P-)sPCG	CA-PCG3	PPCG	P&R-PPCG
2cubes_sphere	4	2.61	0.39	22	**30**	**30**	**30**	**22**	**22**
thermomech_TC	7	1.61	0.39	11	**20**	**20**	**20**	**11**	**11**
shipsec8	10^4	2.35	0.08	1666	**1960**/-	-	-	**1691**	**1684**
ship_003	10^5	1.62	0.01	1584	**1590**	**1590**	**1590**	**1584**	**1584**
cfd2	10^5	1.61	0.04	1731	**1750**	**1750**	**1750**	**1731**	**1731**
boneS01	10^6	1.61	0.04	787	**790**	**790**	**790**	**787**	**787**
shipsec1	10^4	1.62	0.05	909	**910**	**910**	**910**	**909**	**909**
bmw7st_1	10^6	1.62	0.07	7243	**7260**	-	**7280**	**5771**	**7243**
Dubcova3	10^3	1.61	0.05	73	**80**	**80**	**80**	**73**	**73**
bmwcra_1	10^7	1.61	0.03	2183	7890/-	-	-	-	**2487**
G2_circuit	10^4	1.61	0.04	506	**510**	**510**	**510**	**506**	**506**
shipsec5	10^5	1.62	0.08	751	**760**	**760**	**760**	**751**	**751**
thermomech_dM	4	1.61	0.39	11	**20**	**20**	**20**	**11**	**11**
pwtk	10^7	2.11	0.05	7377	-	-	-	-	**7412**
hood	10^5	1.60	0.04	1515	**1520**	**1520**	**1520**	**1515**	**1515**
offshore	10^3	1.61	0.40	178	**180**	**180**	**180**	**178**	**178**
af_0_k101	10^7	1.62	0.05	8891	**8960**	-	**8960**	**8956**	**8891**
af_1_k101	10^7	1.61	0.08	8359	**8360**	-	**8360**	**8357**	**8359**
af_2_k101	10^7	1.61	0.04	9956	**10000**/**10010**	-	-	**10004**	**9990**
af_3_k101	10^6	1.62	0.05	8076	**8110**	-	-	**8093**	**8088**
af_4_k101	10^7	1.61	0.04	9881	**9890**/**9900**	-	**9890**	**9882**	**9881**
af_5_k101	10^7	1.62	0.05	9467	**9470**	-	**9470**	**9469**	**9467**
af_shell3	10^4	1.61	0.09	993	**1000**/**1010**	-	-	**1001**	**999**
af_shell4	10^4	1.61	0.09	993	**1000**/**1010**	-	-	**1001**	**999**
af_shell7	10^4	1.61	0.09	991	**1000**/**1010**	-	-	**1001**	**998**
af_shell8	10^4	1.61	0.09	991	**1000**/**1010**	-	-	**1001**	**998**
parabolic_fem	10^4	1.61	10^{-4}	540	**540**	**540**/**550**	-	**540**	**540**
Fault_639	10^5	9.66	0.05	5414	-	-	-	-	**5453**
apache2	10^5	1.61	0.03	1554	**1560**	**1560**	-	**1554**	**1554**
Emilia_923	10^5	1.61	0.03	4564	**5200**/-	-	-	-	**4642**
audikw_1	10^6	1.61	0.07	2520	**2520**	**2520**	**2520**	**2520**	**2520**
ldoor	10^6	1.61	0.05	2764	**2770**	**2770**	**2770**	**2764**	**2764**
bone010	10^7	1.61	0.04	4308	-	-	-	-	5667
ecology2	10^7	1.61	0.03	2345	**2350**	**2350**	-	9392	**2345**
thermal2	10^6	1.61	0.04	1674	**1680**	-	-	**1684**	**1680**
Serena	10^5	50.44	0.09	570	-	-	-	666	**650**
Geo_1438	10^4	1.61	0.03	545	**550**	**550**	**550**	**545**	**545**
Hook_1498	10^5	1.61	0.04	1817	2610/-	-	-	**1830**	**1827**
Flan_1565	10^7	1.84	0.05	4469	-	-	-	**4575**	**4560**
G3_circuit	10^4	1.61	0.04	628	**630**	**630**	**630**	**628**	**628**



(a) af_shell3 (b) Emilia_923 (c) Hook_1498

Figure 6.1: Convergence histories of solver executions (Chebyshev basis) for selected matrices from Table 6.1.

matrices, where none of these methods converged, did not have such a large eigenvalue. A more thorough investigation remains future work.

Figure 6.1 shows the convergence behavior of three test matrices from Table 6.1 that were challenging for the (pipelined) s-step methods. sPCG and P-sPCG diverge after only a few iterations, while CA-PCG3 reaches higher accuracy for one of the matrices. Even though not converging for two of the three matrices, P-CA-PCG reaches higher accuracy than sPCG, P-sPCG and CA-PCG3 and stagnates instead of diverging. We omit results of other matrices that showed similar behavior due to space restrictions. These plots show that the methods differ significantly in their behavior. We believe that a thorough investigation of the convergence behavior has to be done for each method individually, which we leave for future work.

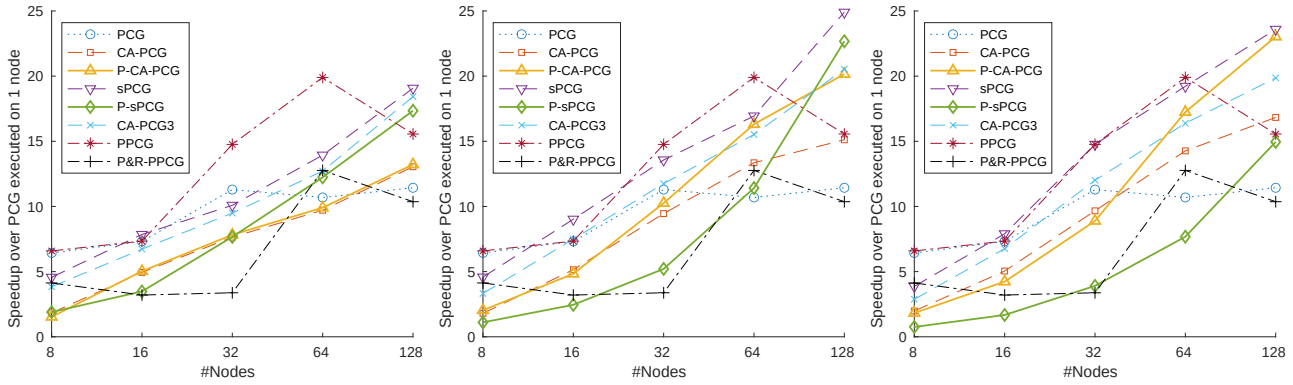
6.3 Runtime performance We show strong scaling experiments with a 3D-Poisson matrix of size $256 \times 256 \times 256$ resulting from discretizing Poisson's equation with a 7-point stencil. We used different step sizes $s = 5, 10, 15$, a Jacobi preconditioner and the Chebyshev basis. Standard PCG required 9.34126 seconds until convergence on one node (128 processes). In Figure 6.2, we show the speedup of all solver variants with different numbers of nodes over standard PCG executed on one node. 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.

All solvers required similar numbers of iterations until convergence. While standard PCG did not scale beyond 32 nodes, the s-step and pipelined s-step PCG methods also scaled for larger numbers of nodes. However, P-sPCG was only able to perform similarly to the s-step methods when using $s = 5$ and $s = 10$. For $s = 15$, the large amount of local computations (see Table 5.1) made P-sPCG the least performing method, only being faster than standard PCG for 128 nodes.

6.3.1 Performance of P-CA-PCG CA-PCG was the least performing s-step method in terms of runtime performance. Despite performing similarly to CA-PCG for $s = 5$, P-CA-PCG showed speedups similar to the other s-step methods for larger step sizes.

The reason for this behavior is depicted in Figure 6.3, which shows the average runtimes over all processes spent in the most important kernels for execution of the solvers on 64 nodes and $s = 15$. Although CA-PCG and P-CA-PCG spent more time in the MPK than the other solvers (due to the additional MV products and preconditioner applications), pipelining reduces this overhead in P-CA-PCG. Note that the time spent purely on waiting for the global reductions to finish is almost eliminated in P-CA-PCG. If the global reductions were even more expensive, larger parts of the MPK could be overlapped by global communication. Thus, P-CA-PCG has the potential to outperform sPCG and CA-PCG3 with higher parallelism.

6.3.2 Performance of P-sPCG The time spent purely for waiting on global communication results is also reduced in P-sPCG compared to sPCG (see Figure 6.3). However, the large amount of local matrix column computations cannot be overlapped with global communication. The recursive MPK computations and the local update operations of $\mathcal{O}(s)$ matrices of size $n \times s$ both involve $\mathcal{O}(s^3 n)$ cost per iteration (i.e., per s steps). Using the monomial basis, the recursive MPK computations can be omitted, as the respective matrices are part of the extended s-step basis matrices. However, the amount of local matrix column updates is the same



(a) Step size $s = 5$ (b) Step size $s = 10$ (c) Step size $s = 15$

Figure 6.2: Speedup for a 7-point 3D Poisson matrix of size $256 \times 256 \times 256$ for different numbers of nodes (128 processes per node) and different step sizes over standard PCG executed on one node (128 processes). Chebyshev basis and Jacobi preconditioner.

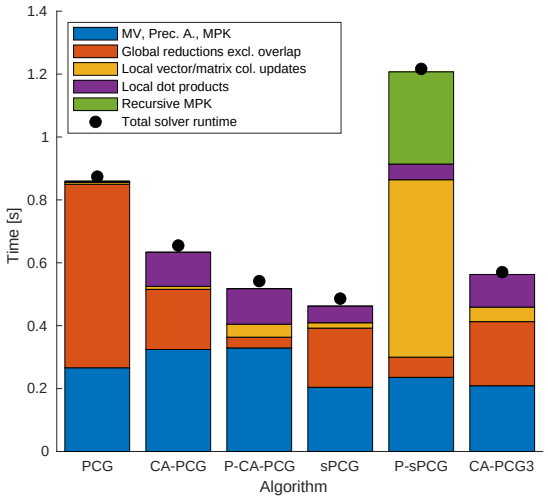


Figure 6.3: Average runtimes over all processes spent in different kernels for the data in Figure 6.2 for $s = 15$ and 64 nodes.

for P-sPCG and P-sPCG$_{\text{mon}}$.

6.3.3 Performance of pipelined methods While performing better than all (pipelined) s-step methods for smaller s, PPCG showed a similar speedup as sPCG with $s = 15$ for 8 to 64 nodes. The scalability of the (pipelined) s-step methods can be improved by increasing s while PPCG and P&R-PPCG do not provide such possibility and were not able to scale beyond 64 nodes. Increasing s potentially decreases stability. P-CA-PCG showed better numerical stability than sPCG in Table 6.1 while performing very similar to sPCG with large s and large numbers of nodes.

P&R-PPCG performed significantly worse than PPCG in our experiments due to the larger number of MV products and preconditioner applications. These computations to improve stability of PPCG decreases performance of P&R-PPCG, similarly to CA-PCG, which performs worse than the other two s-step methods. Contrarily, P-CA-PCG performs similar to sPCG despite these additional computations.

7 Conclusion and future work We generalized the existing pipelined s-step PCG method P-sPCG$_{\text{mon}}$ [16] for arbitrary basis types, denoting this new algorithm as P-sPCG, and illustrated the resulting improvement in numerical stability. Additionally, we developed P-CA-PCG, a new pipelined s-step PCG method based on the s-step method CA-PCG proposed in [18]. We show theoretically that P-CA-PCG requires much less local computation than P-sPCG and P-sPCG$_{\text{mon}}$ while retaining their communication-hiding and -avoiding properties.

In experiments with real-world problems, we showed that our novel algorithms P-sPCG and P-CA-PCG are numerically more stable than P-sPCG$_{\text{mon}}$. Strong scaling experiments with a synthetic test problem confirmed our theoretical analysis, showing clear performance advantages of P-CA-PCG over P-sPCG for larger values of s.

P-CA-PCG seems slightly less stable than CA-PCG. Beyond choosing a suitable basis type, several strategies have been proposed to improve stability in CA-PCG. In future work, we plan to investigate the applicability of these strategies to P-CA-PCG.

References

[1] E. Carson, *Communication-Avoiding Krylov Subspace Methods in Theory and Practice*, PhD thesis, EECS Department, University of California, Berkeley, 2015.

[2] E. Carson, *The adaptive s-step conjugate gradient method*, SIAM Journal on Matrix Analysis and Applications, 39 (2018), pp. 1318–1338, https://doi.org/10.1137/16M1107942.

[3] E. Carson and J. Demmel, *A residual replacement strategy for improving the maximum attainable accuracy of s-step Krylov subspace methods*, SIAM Journal on Matrix Analysis and Applications, 35 (2014), pp. 22–43, https://doi.org/10.1137/120893057.

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

[5] E. C. Carson, T. Gergelits, and I. Yamazaki, *Mixed precision s-step Lanczos and conjugate gradient algorithms*, Numerical Linear Algebra with Applications, 29 (2022), p. e2425.

[6] E. C. Carson, M. Rozložník, Z. Strakoš, P. Tichý, and M. Tůma, *The numerical stability analysis of pipelined conjugate gradient methods: Historical context and methodology*, SIAM Journal on Scientific Computing, 40 (2018), pp. A3549–A3580, https://doi.org/10.1137/16M1103361.

[7] T. Chen and E. C. Carson, *Predict-and-recompute conjugate gradient variants*, SIAM Journal on Scientific Computing, 42 (2020), pp. A3084–A3108, https://doi.org/10.1137/19m1276856, https://arxiv.org/abs/1905.01549.

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

[9] S. Cools, J. Cornelis, and W. Vanroose, *Numerically stable recurrence relations for the communication hiding pipelined conjugate gradient method*, IEEE Transactions on Parallel and Distributed Systems, 30 (2019), p. 2507–2522, https://doi.org/10.1109/TPDS.2019.2917663.

[10] T. A. Davis and Y. Hu, *The university of florida sparse matrix collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1–25, https://doi.org/10.1145/2049662.2049663.

[11] J. Demmel, M. F. Hoemmen, M. Mohiyuddin, and K. A. Yelick, *Avoiding communication in computing Krylov subspaces*, Tech. Report UCB/EECS-2007-123, EECS Department, University of California, Berkeley, 2007.

[12] P. Ghysels and W. Vanroose, *Hiding global synchronization latency in the preconditioned conjugate gradient algorithm*, Parallel Computing, 40 (2014), pp. 224–238, https://doi.org/10.1016/j.parco.2013.06.001. 7th Workshop on Parallel Matrix Algorithms and Applications.

[13] M. F. Hoemmen, *Communication-avoiding Krylov subspace methods*, PhD thesis, EECS Department, University of California, Berkeley, 2010.

[14] V. Mayer and W. N. Gansterer, *Numerical properties and scalability of s-step preconditioned conjugate gradient methods*, in Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops'25), St Louis, MO, USA, 2025, ACM, https://doi.org/10.1145/3731599.3767542.

[15] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia,PA,USA, 2nd edition ed., 2003, https://doi.org/10.1137/1.9780898718003.ch4.

[16] M. Tiwari and S. Vadhiyar, *Pipelined preconditioned conjugate gradient methods for distributed memory systems*, in 2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC), Piscataway, NJ, USA, 2020, IEEE, pp. 151–160, https://doi.org/10.1109/HiPC50609.2020.00029.

[17] M. Tiwari and S. Vadhiyar, *Pipelined preconditioned s-step conjugate gradient methods for distributed memory systems*, in 2021 IEEE International Conference on Cluster Computing (CLUSTER), Piscataway, NJ, USA, 2021, IEEE, pp. 215–225, https://doi.org/10.1109/Cluster48925.2021.00061.

[18] S. Toledo, *Quantitative Performance Modeling of Scientific Computations and Creating Locality in Numerical Algorithms*, PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 1995.

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