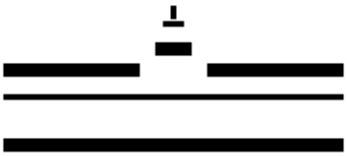


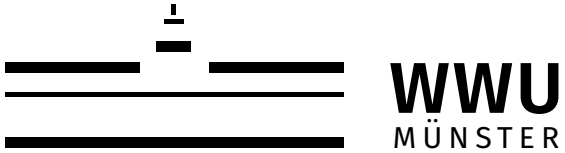
WWU
MÜNSTER

Hardware-Oriented Krylov Methods for High-Performance Computing

PhD Thesis

Nils-Arne Dreier

– 2020 –



Fach: Mathematik

Hardware-Oriented Krylov Methods for High-Performance Computing

Inaugural Dissertation

zur Erlangung des Doktorgrades der Naturwissenschaften

– Dr. rer. nat. –

im Fachbereich Mathematik und Informatik
der Mathematisch-Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms-Universität Münster

eingereicht von

Nils-Arne Dreier

aus

Bünde

– 2020 –

Dekan:	Prof. Dr. Xiaoyi Jiang Westfälische Wilhelms-Universität Münster Münster, DE
Erster Gutachter:	Prof. Dr. Christian Engwer Westfälische Wilhelms-Universität Münster Münster, DE
Zweiter Gutachter:	Laura Grigori, PhD INRIA Paris Paris, FR
Tag der mündlichen Prüfung:	08.03.2021
Tag der Promotion:	08.03.2021

Abstract

Krylov subspace methods are an essential building block in numerical simulation software. The efficient utilization of modern hardware is a challenging problem in the development of these methods. In this work, we develop Krylov subspace methods to solve linear systems with multiple right-hand sides, tailored to modern hardware in high-performance computing.

To this end, we analyze an innovative block Krylov subspace framework that allows to balance the computational and data-transfer costs to the hardware. Based on the framework, we formulate commonly used Krylov methods. For the CG and BiCGStab methods, we introduce a novel stabilization approach as an alternative to a deflation strategy. This helps us to retain the block size, thus leading to a simpler and more efficient implementation.

In addition, we optimize the methods further for distributed memory systems and the communication overhead. For the CG method, we analyze approaches to overlap the communication and computation and present multiple variants of the CG method, which differ in their communication properties. Furthermore, we present optimizations of the orthogonalization procedure in the GMRes method. Beside introducing a pipelined Gram-Schmidt variant that overlaps the global communication with the computation of inner products, we present a novel orthonormalization method based on the TSQR algorithm, which is communication-optimal and stable. For all optimized method, we present tests that show their superiority in a distributed setting.

Zusammenfassung

Krylovraummethoden stellen einen essentiellen Bestandteil numerischer Simulationssoftware dar. Die effiziente Nutzung moderner Hardware ist ein herausforderndes Problem bei der Entwicklung solcher Methoden. Gegenstand dieser Dissertation ist die Formulierung von Krylovraumverfahren zur Lösung von linearen Gleichungssystemen mit mehreren rechten Seiten, welche die Eigenschaften moderner Hardware berücksichtigen.

Dazu untersuchen wir ein innovatives Blockkrylovraum-Framework, welches es ermöglicht die Berechnungs- und Datentransferkosten der Blockkrylovraummethode an die Hardware anzupassen. Darauf aufbauend formulieren wir mehrere Krylovraummethoden. Für die CG und BiCGStab Methoden führen wir eine neuartige Stabilisierungstrategie ein, die es ermöglicht die Spaltenanzahl des Residuums beizubehalten. Diese ersetzt die bekannte Deflationstrategien und ermöglicht eine einfachere und effizientere Implementierung der Methoden.

Des Weiteren optimieren wir die Methoden bezüglich der Kommunikation auf Systemen mit verteiltem Speicher. Für die CG Methode untersuchen wir Strategien, um die Kommunikation mit Berechnungen zu überlappen. Dazu stellen wir mehrere Varianten des Algorithmus vor, welche sich durch ihre Kommunikationseigenschaften unterscheiden. Außerdem werden für die GMRes Methode optimierte Varianten der Orthonormalisierung entwickeln. Neben einem Gram-Schmidt Verfahren, welches Berechnungen und Kommunikation überlappt, präsentieren wir eine neue Methode, welche auf dem TSQR-Algorithmus aufbaut und Stabilität sowie geringe Kommunikationskosten vereint. Für alle optimierten Varianten zeigen wir numerische Tests, welche die Verbesserungen auf Systemen mit verteiltem Speicher demonstrieren.

Acknowledgments

I would like to express my deep gratitude to all people who have supported me over the last years. First, I thank Prof. Dr. Christian Engwer for giving me the opportunity to work on this topic, all the creative discussions, motivation, great guidance and for being an excellent supervisor. I thank all my colleges in our workgroup for the pleasant atmosphere, in particular I thank Liesel Sommer and Marcel Koch for proof-reading this thesis and giving useful hints for improvements. Furthermore, I thank Prof. Dr. Robert Klöfkorn for giving me the opportunity to work for a few weeks in Bergen, collecting valuable experience and enjoying the Norwegian nature. All implementations of algorithms in the thesis are based on the DUNE software framework, thus I thank all the developers of DUNE for making this project happen.

Diese Arbeit wäre ohne die bedingungslose Unterstützung meiner Eltern Kirsten und Eckhard nicht möglich gewesen. Danke für all die finanzielle und moralische Unterstützung während meiner gesamten Studienzeit.

Der größte Dank gilt meiner Frau Eileen. Danke dafür, dass es dich in meinem Leben gibt und für all den Rückhalt, die Unterstützung und Liebe, die es mir sehr erleichtert haben diese Arbeit zu verfassen.

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

1 Introduction

We can only see a short distance ahead, but we can see plenty there that needs to be done.

Alan Turing

1.1 Motivation

In the last decades, High-Performance Computing (HPC) became an essential part of science, industry and our every day life. Engineers use it to optimize the shape of cars and aircraft. Meteorologists use it to create the daily weather forecast. Physicists use it to simulate quantum mechanics which helps to understand the elements of our universe. It is widely used to simulate the global climate on large supercomputers. Petroleum engineers design offshore platforms using HPC to make them more efficient. In medical research, scientists simulate an entire heart or brain to investigate the sources of strokes and heart attacks. Other fields of application include sociology, biology and astrology. Even during the COVID-19 pandemic, HPC is used to investigate medicine that is effective to treat COVID-19 patients.

In all these applications, HPC brings great improvements. As a consequence, the need for more and more computation power has grown extremely. Figure 1.1 shows the development of the performance of the fastest 500 supercomputers in the world. It shows that the available computation performance has increased by a factor of one million over the last 17 years. Until the early 2000s, the increase of performance was due to an increase of the frequency of the processors. Since then, the frequency stagnates at approximately 2 GHz. Due to the higher power consumption and heat production at higher frequencies, it is not efficient to increase the frequency further. Hence, an increase of performance is only possible by an increase of parallelism. Another challenge

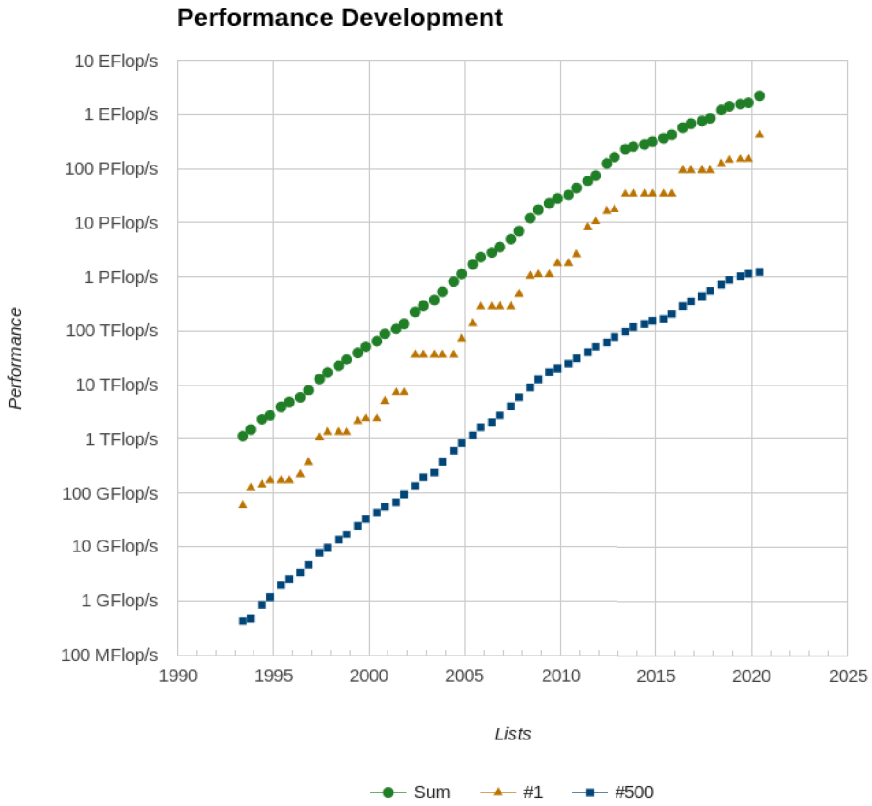


Figure 1.1: Performance development of the top 500 supercomputers. Taken from `https://top500.org` [TOP500].

in HPC is the power consumption of the over-all system. Modern supercomputers consume power in the scale of megawatts. That is comparable to a whole offshore wind turbine. Furthermore, the fault-tolerance of large computers is a problem as well. The more components are involved, the higher is the probability that components fail during the computation. Due to all these challenges, it is very important to develop software that uses the hardware efficiently.

The problem of solving large sparse linear systems is a building block in many HPC codes that consume large parts of the computation time. As direct solvers scale badly for large linear systems and consume far too much memory, iterative solvers are used on supercomputers to solve this kind of problems. Especially Krylov solvers have been approved to solve this problem. For several reasons, Krylov solvers only utilize a fraction of the peak-performance of supercomputers. This is shown in the HPCG benchmark list [TOP500]. For example, Fugaku, currently the fastest supercomputer in the world, only performed 13.366 PFlop in the HPCG benchmark where it reaches 415.53 PFlop in the LINPACK benchmark. This shows the potential for improvements.

In this thesis, we consider three aspects of this issue. First, we consider the increasing parallelism of larger machines. This parallelism appears on three levels:

1. Instruction level: The instruction sets of modern CPU contain instructions that

perform multiple floating-point operations. For example, Fused-Multiply-Add (FMA) instructions, where a multiplication is carried out together with an addition. Other examples are Single-Instruction-Multiple-Data (SIMD) instructions, where the same operation is applied on multiple data.

2. Shared memory level: Modern CPUs consist of multiple cores that work in parallel but operate on the same memory.
3. Distributed memory level: Supercomputers are build from multiple nodes that communicate over a network. Modern supercomputers have hundreds to many hundred thousand nodes. This number is expected to grow even further in the future.

All this parallelism must be exploited to use the supercomputer efficiently.

Another aspect is the so called memory-wall. The bandwidth between the memory and the CPU is limited, which hinders the CPU to exploit its full performance. This effect is often mitigated by a hierarchical cache. However, this does only work if the loaded data is reused enough. A quantity to measure the reuse of the data is the arithmetic intensity (flop per byte, Flop/B), which is a property of the used algorithm.

The last aspect is a consequence of the distributed memory parallelism. The communication costs grow if more nodes are involved in the computation. A typical communication pattern that is used in Krylov solvers is a collective communication, e.g. a global sum. This type of communication scales as $\mathcal{O}(\log(P))$, where P is the number of processors. This makes it essential to organize the communication well and overlap the communication phase with other meaningful computations.

In the literature, the data transfer between the different cache levels as well as the data movement between nodes are referred to as communication. In the present thesis, we want to strictly separate between the data movement between cache levels which could be seen as intra-node communication and the data movement between nodes which could be seen as inter-node communication.

We consider large sparse linear systems that need to be solved for multiple right-hand sides. This is a very common problem that appears in applications like inverse problems or optimization. We will see that this type of problem is quite well posed to solve or mitigate all the mentioned issues.

1.2 Related Work

The first part of this thesis is strongly inspired by the work of Frommer, Lund and Szyld [FSL17; FLS19] and the PhD thesis by Lund [Lun18]. They recently presented the block Krylov framework on which this thesis is built on.

The second part of this thesis is related to the work of Cools and Vanroose [CV17a; Coo+18; CCV19]. They presented pipelined Krylov methods that overlap the collective communication of the inner products with computation. In the field of communication-avoiding methods, Demmel et al. [Dem+08; Dem+12] as well as Carson [Car15] and

Hoemmen [Hoe10] presented several methods and ideas to avoid communication in Krylov methods. These methods fuse the communication of multiple iterations into one communication, to reduce the number of messages. Therefore, they are known as s-step Krylov methods. Also combinations of s-step Krylov methods and block Krylov methods have been proposed [CK10].

Another approach is to use multiple search directions in a Krylov method. This could be found for example in the multi preconditioning methods of Spillane [Spi16]. Another class of algorithms, that fall into this category, are the enlarged Krylov methods. The idea is to transfer the advantageous convergence properties from the block Krylov methods for multiple right-hand sides to systems with a single right-hand side. They were presented by Grigori and Tissot [GT17].

In practice, several linear algebra software frameworks provide optimized Krylov methods. For example, PETSc [Bal+97; Bal+20] provides communication-avoiding and pipelined Krylov methods, but lacks block Krylov methods. Trilinos [Tea] contains a linear algebra module that contains block Krylov methods and corresponding communication-avoiding methods. Other packages focus more on scalable preconditioners, e.g. hypre [FY02].

Software packages that focus on the solution of PDEs often only provide textbook Krylov methods, that are not explicitly optimized for high-performance computing. For example, Dune [Bas+20b; Bla+16; Bas+08b; Bas+08a], deal.II [Arn+20] and NGSolve [Sch97].

1.3 Contributions and Outline

As already mentioned, we distinguish intra- and inter-node communication. We pick up this difference to structure this thesis into two parts. Part 1 refers to the first two aspects mentioned in the motivation, i.e. the vectorization and memory-wall. In the second part, we optimize the methods further for inter-node communication, which refers to the last aspect in the description above.

In Chapter 3, we review the block Krylov framework by Frommer, Szyld and Lund and analyze its building blocks with respect to their performance on modern CPU architectures. We provide a novel view onto the set of possible *-subalgebras, based on three elementary cases and introduce a new class of *-subalgebras. Furthermore, based on the performance analysis we provide a guideline for choosing an appropriate *-subalgebra.

Based on this framework, we formulate block versions of the CG, GMRes and BiCGStab method in the Chapters 4, 5 and 6, respectively. For the CG and BiCGStab methods, we introduce a novel stabilization strategy that replaces the deflation process used in most methods in the literature. The new strategy is better suited in our context as we depend on a fixed number of columns in the block vectors.

In the second part, we optimize the methods with respect to inter-node communi-

cation. For that, we adopt the approaches by Cools and Vanroose for our block CG method in Chapter 8. This yields a novel pipelined block Krylov method that combines the advantages of both approaches.

In Chapter 9, we consider the orthogonalization procedure of the block GMRes method. We introduce a pipelined Gram-Schmidt orthogonalization and an innovative reduction-based orthogonalization and compare it with the classical Gram-Schmidt method, which is the standard in up-to-date methods. The new methods prove to perform better and are more stable than the classical Gram-Schmidt method.

All newly introduced methods are validated with numerical experiments, carried out on a modern Intel compute server or on the supercomputer PALMAII of the University of Münster.

2

A Brief Introduction to Krylov Methods

Krylov methods came up in the 1950s. Lanczos [Lan50] presented his method for solving eigenvalue problems in 1950. At the same time, Hestenes, Stiefel, et al. [HS+52] presented the Conjugate Gradient (CG) method for solving linear systems. Back then, the CG method was considered a direct method. Later, around 1975, with the development of vector computers and massive memory computers the methods became more popular as iterative methods. The term Krylov method goes back to the Russian mathematician Aleksey Nikolaevich Krylov, who presented related work in 1931 [Kry31]. Nowadays, lots of Krylov methods were developed and became an essential part of modern scientific computing. Golub and O'Leary [GO89] gave a good overview over the early developments of Krylov methods. Recommendable books about Krylov methods are written by Greenbaum [Gre97], Saad [Saa03], Hackbusch [Hac94] and Trefethen and Bau [TB97].

We start with some basic definitions. For the rest of this chapter, we consider a linear system

$$Ax^* = b, \tag{2.1}$$

where $A \in \mathcal{L}(\mathbb{R}^n, \mathbb{R}^n)$ is an invertible linear operator, $b \in \mathbb{R}^n$ is a given right-hand side and $x^* \in \mathbb{R}^n$ is the desired solution.

Definition 2.1 (Krylov space)**.** *For $k \in \mathbb{N}$ and $r \in \mathbb{R}^n$, the vector space*

$$\mathcal{K}^k(A, r) = \operatorname{span}\left(r, Ar, \ldots, A^{k-1}r\right)$$

is called the order-k Krylov space generated by A and r. The quantity

$$\nu(r, A) = \max_{k \in \mathbb{N}} \left(\dim \mathcal{K}^k(A, r) \right)$$

is called the grade of r with respect to A.

The following lemma summarizes the most important properties of the Krylov space.

Lemma 2.2 (Properties of the Krylov space)**.** *The following properties of the Krylov space hold*

- $\mathcal{K}^k(A, r) \subseteq \mathcal{K}^{k+1}(A, r)$
- $\nu(A, r) \leq 1 + \operatorname{rank}(A)$ *and* $\nu(A, r) \leq n$.
- *The vector space of polynomials $\mathbb{P}^{k-1}$ can be embedded into the Krylov space $\mathcal{K}^k(A, r)$ with the embedding*

$$\begin{aligned} \iota : \mathbb{P}^{k-1} &\to \mathcal{K}^k(A, r) \\ p &\mapsto p(A)r. \end{aligned} \tag{2.2}$$

  If $k \leq \nu(A, r)$, then ι is an isomorphism.

The objective of a Krylov method is to find an approximation $x^k \in \mathcal{K}^k(A, r^0)$ to the solution $A^{-1}b$, where $r^0 = b - Ax^0$ is the initial residual for an initial guess $x^0 \in \mathbb{R}^n$. For example, the CG method [HS+52] computes the best approximation with respect to the energy error $\|x^* - x^k\|_A$, and the GMRes method [SS86] computes the best approximation with respect to the residual norm $\|Ax^k - b\|$. To compute this approximation, it is often helpful to use an orthonormal basis of the Krylov space. This orthonormal basis can be computed with the Arnoldi process [Arn51], that is based on the Gram-Schmidt orthogonalization process. It computes an orthonormal basis $V \in \mathbb{R}^{n \times k}$ that satisfies the so-called Arnoldi relation

$$AV^k = V^k H^k + h_{k+1,k} v^{k+1} {e_k}^{\mathsf{T}}, \tag{2.3}$$

where $H^k \in \mathbb{R}^{k \times k}$ is a Hessenberg matrix, $v^{k+1} \in \mathbb{R}^n$ is the subsequent basis vector and e_k is the kth unity vector. For example, the GMRes method uses this relation to minimize the euclidean norm of the residual. If $h_{k+1,k}$ is small, H is a good approximation for the operator A restricted on the Krylov space. In the case where A is symmetric, it follows from (2.3) that H is a tridiagonal matrix. This fact is used in the CG and MINRES [PS75] methods, such that the basis V does not need to be stored explicitly. Instead the approximation is updated during the iteration. This property is called *short recurrence*.

From the definition, it is clear that the solution of the system (2.1) is contained in the Krylov space $\mathcal{K}^{\nu(A,r^0)}(A, r^0)$. Therefore, Krylov space methods that compute a best

approximation in the Krylov space, terminate after at least $\nu(A, r^0)$ steps. However, Krylov methods are usually used to compute a good approximation for the solution that is achieved before $\nu(A, r)$ iterations are performed. In general, it is not possible to provide an error estimation that ensures convergence with fewer than $\nu(A, r)$ iterations, as the following example shows.

Example 2.3. *Consider the following system*

$$A = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & 1 & 0 \\ 0 & \cdots & & 0 & 1 \\ 1 & 0 & \cdots & & 0 \end{pmatrix} \qquad b = e_1 = \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

With initial guess $x^0 = 0$, the initial residual is $r^0 = e_1$. For all $k < n$, all vectors in the Krylov space would have the last coefficient 0, as the operator A pushes the coefficients one place further. As the solution of the system is e_n, the best approximation in the Krylov space is 0. Only if $k \geq n$ the error norm can be decreased.

Therefore, to show any results about convergence rates additional assumptions are necessary. For example, there are results if the symmetric part $\frac{1}{2}(A + A^\mathsf{T})$ is positive definite which can be found in the excellent books of Greenbaum [Gre97] or Saad [Saa03]. For the CG method there exists the following famous estimation of the energy error.

Theorem 2.4 (Convergence of CG method)**.** *Let A be symmetric positive definite and $e^k = x^* - x^k$ the error of the kth CG iteration. Then the energy error of e^k can be estimated by*

$$\|e^k\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k \|e^0\|_A,$$

where $\kappa = \|A\|\|A^{-1}\|$ is the condition number of A.

The proof is geared to the one presented by Trefethen and Bau [TB97].

Proof. By Lemma 2.2, we can identify every element in the Krylov space $\mathcal{K}^k(A, r^0)$ by a polynomial of degree $k-1$. In particular, we write the kth error of the CG method as

$$\begin{aligned} e^k &= x^* - x^k \\ &= x^* - x^0 - p_{k-1}(A) r^0 \\ &= x^* - x^0 - p_{k-1}(A) A (x^* - x^0) \\ &= q_k(A) e^0, \end{aligned} \tag{2.4}$$

for a polynomial $p_{k-1} \in \mathbb{P}^{k-1}$ and $q_k(\mathrm{x}) := p_{k-1}(\mathrm{x})\mathrm{x} + 1$. As the CG methods finds the best approximation with respect to the energy norm, we conclude

$$\|e^k\|_A \leq \inf_{q_k} \|q_k(A)e^0\|_A.$$

Here the infimum is taken over all polynomials of degree k with absolute coefficient 1. For the smallest and largest eigenvalues $\lambda_{\min}$ and $\lambda_{\max}$, the polynomials that realize this infimum are given by the scaled Chebyshev polynomials

$$\widetilde{T}_k(\mathrm{x}) = \left(T_k \left(\frac{-\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} - \lambda_{\min}} \right) \right)^{-1} T_k \left(\frac{2\mathrm{x} - \lambda_{\max} - \lambda_{\min}}{\lambda_{\max} - \lambda_{\min}} \right),$$

where the kth Chebyshev polynomial T_k is defined by the recursion formula

$$\begin{aligned} T_0(\mathrm{x}) = 1, \qquad T_1(\mathrm{x}) = \mathrm{x} \\ T_{k+1}(\mathrm{x}) = 2\mathrm{x}T_k(\mathrm{x}) - T_{k-1}(\mathrm{x}) \end{aligned} \tag{2.5}$$

or directly by

$$T_k(\mathrm{x}) = \cos\left(k \arccos\left(\mathrm{x}\right)\right).$$

One can show that the scaled Chebyshev polynomials minimize the C^∞-norm on the interval $[\lambda_{\min}, \lambda_{\max}]$ in the space of polynomials with absolute coefficient 1. The C^∞-norm is bounded by

$$\|\widetilde{T}_k\|_{C^\infty} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k, \tag{2.6}$$

where $\kappa = \|A\|\|A^{-1}\| = \frac{\lambda_{\max}}{\lambda_{\min}}$ denotes the condition number of the operator A. As A is symmetric positive definite the eigenvectors $(u_i)_i$ to the eigenvalues $(\lambda_i)_i$ build an orthonormal basis of $\mathbb{R}^n$. We write the error e^0 in this basis as

$$e^0 = \sum_{i=0}^{n-1} a_i u_i.$$

Then the energy error of e^0 is given by

$$\|e^0\|_A^2 = \sum_{i=0}^{n-1} a_i^2 \lambda_i$$

and the energy error of e^k is given by

$$\begin{aligned}\|e^k\|_A^2 &= \|q_k(A)e^0\|_A^2 \\ &= \sum_{i=0}^{n-1} q_k(\lambda_i)^2 a_i^2 \lambda_i \\ &\leq \max_{i=0}^{n-1} |q_k(\lambda_i)|^2 \|e^0\|_A^2 \\ &\leq 4\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2k} \|e^0\|_A^2,\end{aligned}$$

where we used Equation (2.4) and (2.6). □

The error bound given by Theorem 2.4 is sharp, i.e. there exists data A and b such that equality holds. But for fixed data better convergence could occur. For example, if the initial residual r^0 is an eigenvector of A, then the method would converge within one iteration, as the grade of r^0 with respect to A is 1.

For the GMRes method a representation for the residual similar to Equation (2.4) can be formulated as

$$\begin{aligned}r^k &= b - Ax^k \\ &= b - Ax^0 - Ap_{k-1}(A)r^0 \\ &= r^0 - Ap_{k-1}(A)r^0 \\ &= q_k(A)r^0,\end{aligned}$$

with $q_k(\mathrm{x}) = 1 - p_{k-1}(\mathrm{x})\mathrm{x}$. From this equality, an error estimation could be derived, if the operator A is normal, i.e. diagonalizable. We review this prove for the block variant of the GMRes method in Section 5.2.

Theorem 2.4 and the theory of the proof show that the convergence behavior of Krylov space methods depend on the condition number of the operator. Therefore, it is common practice to use preconditioning. That means the Krylov method is applied on the system

$$M_L{}^{-1} A M_R{}^{-1} y = M_L{}^{-1} b,$$

for some matrices M_L, M_R for which the inverse can be applied cheaply. Once y has been found, the solution of $Ax = b$ can be found easily by computing $x = M_R{}^{-1} y$. The operators M_L, M_R are chosen to improve the condition number of the operator $M_L^{-1} A M_R{}^{-1}$ and hence to improve the convergence of the Krylov method. Often one of M_L and M_R is chosen to be the identity, resulting in so-called left or right preconditioning.

Simple preconditioners depend on iterative splitting methods like Jacobi or Gauß-

Table 2.1: Overview of commonly used Krylov methods their properties and references.

Name	Requirements	Short Recursion	Minimization	Reference
Conjugate Gradients (CG)	symmetric positive definite	yes	$\|e^k\|_A$	[HS+52]
General Minimal Residual (GMRes)	none	no	$\|r^k\|_2$	[SS86]
Biconjugate Gradients Stabilized (BiCGStab)	none	yes	none	[Van92]
Minimum Residual (MINRes)	symmetric	yes	$\|r^k\|_2$	[PS75]
Conjugate Residual (CR)	symmetric	yes	$\|r^k\|_2$	[Sti55],[EES83]
Quasi Minimal Residual (QMR)	none	yes	none	[FN91]

Seidel iteration. They split the operator into a sum of matrices

$$A = M + N,$$

where N can be easily inverted. For example, the Jacobi method chooses N as the diagonal of A and the Gauss-Seidel method chooses N as the lower triangular part of A. Then the preconditioner is given by some iterations of the fixpoint iteration

$$\begin{aligned} x^{k+1} &= N^{-1}\left(b - Mx^k\right) \\ &= x^k + N^{-1}r^k. \end{aligned}$$

Other popular preconditioners compute incomplete factorization of the operator A. These preconditioners often only affect the large eigenvalues of A. Especially on very large systems this does not reduce the condition number sufficiently, as the small eigenvalues are not affected. More sophisticated preconditioners are multi-grid methods that use restrictions of the operator A to coarser spaces and apply the simple preconditioners on that level too. Thus, all ranges of eigenvalues are affected. An alternative is to compute coarse spaces that contain the eigenvectors of the small eigenvalues. The preconditioner is then chosen as the projection onto the orthogonal complement of this coarse spaces (e.g. GenEO [Spi+14]).

Table 2.1 shows an overview of widely used Krylov methods for solving linear systems.

It shows the requirements for the operator and preconditioner as well as whether it uses a short recursion. Furthermore, the norm in which the error is minimized is given and the citation in which the method was presented.

Part 1
Block Krylov Methods

La théorie est mère de la pratique.

Louis Pasteur

3
A General Block Krylov Framework

Block Krylov methods have been developed in the 1970s and 1980s to solve linear systems with multiple right-hand sides [OLe80] or compute multiple eigenvectors [Und75]. Recently, they have been rediscovered in the context of high-performance computing to reduce the communication overhead.

The term "block" is quite overloaded in the field of numerical linear algebra. In the context of matrix structures it means that the matrix is subdivided into smaller matrices. In the context of preconditioning it often refers to the block Jacobi method that only considers the diagonal blocks of the system matrix to parallelize the preconditoning, and in the context of Krylov methods it refers to the already mentioned methods that are based on the work of O'Leary [OLe80].

We consider block Krylov methods to solve linear systems with multiple right-hand sides. Let $A \in \mathbb{R}^{n \times n}$ be an invertible linear operator and $B \in \mathbb{R}^{n \times s}$ a block vector. A linear system with multiple right-hand sides, called a *block system*, for the solution $X^* \in \mathbb{R}^{n \times s}$ is given by

$$AX^* = B, \tag{3.1}$$

which is equivalent to

$$Ax_i^* = b_i \qquad \forall i = 1, \ldots, s, \tag{3.2}$$

where x_i^* and b_i denote the ith column of X^* and B, respectively.

The basic idea of block Krylov methods is to make use of the sum of all Krylov spaces of the linear systems in Equation (3.2) to find a better approximation for the solution. O'Leary [OLe80] showed that the convergence of the block CG method is faster than that of the CG method and independent of the $s - 1$ smallest eigenvalues.

We recall this result in Theorem 4.5.

In the context of high-performance computing block Krylov methods have another advantage. During one iteration the operator (and preconditioner) is applied to block vectors in $\mathbb{R}^{n\times s}$, which is beneficial if the matrix is explicitly stored. It leads to a higher arithmetical intensity, which is crucial on modern CPUs to achieve good performance. Furthermore, it is well suited for the use of SIMD instructions if the block vectors are stored in row-major format.

Several approaches have been proposed to use the faster convergence of block Krylov methods for linear systems with a single right-hand side. Grigori and Tissot [GT17; Al +18] proposed a method where they decompose the right-hand side b, based on the domain decomposition, to obtain multiple right-hand sides which can be used to solve the original problem. This approach is also used in the PhD theses of Moufawad [Mou14], Al Daas [Al 18] and Tissot [Tis19].

Other approaches are to choose additional right-hand sides randomly (BRRHS-CG) [NY95] or to choose additional initial guesses randomly and solve all for the same right-hand side (CoopCG) [Bha+12]. In principle, these approaches are also applicable to the methods presented in this work.

One iteration of a block Krylov method has costs in order of $\mathcal{O}(s^2n + s^3)$, which could become a problem, if a lot of right-hand sides are used, i.e. s is large. To mitigate this effect, but still take advantage of the higher operational intensity of the operator and preconditioner application, we introduce a general framework of block Krylov spaces based on the work of Frommer, Szyld and Lund [FSL17; FLS19; Lun18]. That allows us to balance the information exchange between the different right-hand sides and the computational blocking overhead. Then, we provide a performance analysis of the building blocks, provide details about our implementation and present some numerical tests that approve our theory and show the advantages of the block Krylov framework.

3.1 Block Krylov Spaces

Let us start with the review of the block Krylov framework presented by Frommer, Szyld and Lund [FSL17]. Originally this framework was introduced to evaluate functions of matrices. Further development of the framework was done in the thesis by Lund [Lun18] and the paper by Frommer, Lund and Szyld [FLS19].

In the subsequent of this work all methods and algorithms are built upon this framework. The standard Krylov methods can be obtained by choosing $s = 1$, this is referred to as the non-block case. We start with the central definition of the block Krylov space.

Definition 3.1 (Block Krylov subspace)**.** *Let $\mathbb{S}$ be a *-subalgebra of $\mathbb{R}^{s\times s}$ and $R \in \mathbb{R}^{n\times s}$. The kth block Krylov space with respect to A, R and $\mathbb{S}$ is defined by*

$$\mathcal{K}_{\mathbb{S}}^{k}(A,R) = \left\{ \sum_{i=0}^{k-1} A^i R c_i \,|\, c_0,\ldots,c_{k-1} \in \mathbb{S} \right\} \subset \mathbb{R}^{n\times s}.$$

Remarks.

- A *-algebra is a vector space $\mathbb{S}$ equipped with a product and a conjunction. In particular, it means, that for all elements $s \in \mathbb{S}$ and polynomials $p \in \mathbb{P}$ the evaluation of the polynomial for that element $p(s)$ is contained in the *-algebra.
- The Cayley-Hamilton theorem yields that every *-subalgebra of $\mathbb{R}^{s\times s}$ contains an identity. See for example [Bos14]. This identity does not necessarily coincide with the identity in $\mathbb{R}^{s\times s}$. However, for the *-subalgebras $\mathbb{S}$ we consider in this work, the identities coincide $\mathbb{I}_{\mathbb{S}} = \mathbb{I}_{\mathbb{R}^{s\times s}}$. Other *-subalgebras would be pointless, as we will see later.
- We choose $\mathbb{R}^{n\times s}$ as a vector space here. In principle every vector space over some field F could be chosen. $\mathbb{S}$ is then a *-subalgebra of $F^{s\times s}$.
- For the rest of this thesis $\mathbb{S}$ denotes a *-subalgebra of $\mathbb{R}^{s\times s}$.
- The classical block Krylov methods as described by O'Leary use $\mathbb{S} = \mathbb{R}^{s\times s}$.

For the convergence theory of Krylov methods, polynomials play an important role, as we already saw in Theorem 2.4. For the convergence theory in this framework, we introduce the more generic $\mathbb{S}$-valued polynomials.

Definition 3.2. *A polynomial of the form*

$$\mathcal{P}(\mathrm{x}) = \sum_{i=0}^{k} \mathrm{x}^i \gamma_i \qquad\qquad \gamma_i \in \mathbb{S}$$

is called a $\mathbb{S}$-valued polynomial of degree k. We write $\mathbb{P}_{\mathbb{S}}^{k}$ for the space of $\mathbb{S}$-valued polynomials of degree k. Inspired by the paper of El Guennouni, Jbilou and Sadok [EJS03] we denote the product

$$\mathcal{P}(A) \circ Y = \sum_{i=0}^{k} A^i Y \gamma_i,$$

where $Y \in \mathbb{R}^{n\times s}$. With this operation, the operator $\mathcal{P}(A)$ could be considered as a linear operator on the space $\mathbb{R}^{n\times s}$. Furthermore, we define the right-sided product $\mathcal{P}\sigma \in \mathbb{P}_{\mathbb{S}}^{k}$ of a $\mathbb{S}$-valued polynomial $\mathcal{P} \in \mathbb{P}_{\mathbb{S}}^{k}$, with $\mathcal{P}(\mathrm{x}) = \sum_{i=0}^{k} \mathrm{x}^i \gamma_i$, and $\sigma \in \mathbb{S}$ as

$$(\mathcal{P}\sigma)(\mathrm{x}) = \sum_{i=0}^{k} \mathrm{x}^i \gamma_i \sigma.$$

From Definition 3.1 we find the following two lemmas immediately. The first one is

in analogy with (2.2).

Lemma 3.3. *Every element $X \in \mathcal{K}_{\mathbb{S}}^{k}(A, R)$ in the block Krylov space can be represented by a $\mathbb{S}$-valued polynomial $\mathcal{P} \in \mathbb{P}_{\mathbb{S}}^{k-1}$ of degree $k-1$*

$$X = \mathcal{P}(A) \circ R = \sum_{i=0}^{k-1} A^i R c_i \qquad\qquad c_0, \ldots, c_{k-1} \in \mathbb{S}.$$

Lemma 3.4. *If $\mathbb{S}_1$ and $\mathbb{S}_2$ are two *-subalgebras of $\mathbb{R}^{s\times s}$, with $\mathbb{S}_1 \subseteq \mathbb{S}_2$. Then*

$$\mathcal{K}_{\mathbb{S}_1}^{k}(A, R) \subseteq \mathcal{K}_{\mathbb{S}_2}^{k}(A, R)$$

holds.

Analogous to the non-block case ($s = 1$) we define the block grade of a block vector in a block Krylov space. This definition is inspired by Gutknecht and Schmelzer [GS09; Gut07].

Definition 3.5 (Block grade)**.** *In the setting of Definition 3.1 we define the block grade of R with respect of A as*

$$\nu_{\mathbb{S}}(A, R) := \min\left\{k \in \mathbb{N} \,|\, \dim \mathcal{K}_{\mathbb{S}}^{k}(A, R) = \dim \mathcal{K}_{\mathbb{S}}^{k+1}(A, R)\right\}.$$

As Gutknecht and Schmelzer show, the block grade defines the minimal k for which the solution is contained in the Krylov space. We review this result in our context.

Lemma 3.6. *We have*

$$X^* \in X^0 + \mathcal{K}_{\mathbb{S}}^{\nu_{\mathbb{S}}(A,R^0)}\left(A, R^0\right),$$

where $AX^* = B$ and $R^0 = B - AX^0$ is the residual for some initial guess $X^0 \in \mathbb{R}^{n\times s}$.

Proof. By definition of the block Krylov space we have for any $k \in \mathbb{N}$

$$\mathcal{K}_{\mathbb{S}}^{k}\left(A, R^0\right) \subseteq \mathcal{K}_{\mathbb{S}}^{k+1}\left(A, R^0\right).$$

From the definition of the block grade it follows that

$$\mathcal{K}_{\mathbb{S}}^{\nu_{\mathbb{S}}(A,R^0)}\left(A, R^0\right) = \mathcal{K}_{\mathbb{S}}^{k}\left(A, R^0\right)$$

for all $k \geq \nu_{\mathbb{S}}(A, R^0)$. As A is invertible, $\mathcal{K}_{\mathbb{S}}^{\nu_{\mathbb{S}}(A,R^0)}(A, R^0)$ is a A-invariant subspace of $\mathbb{R}^{n\times s}$

$$A\mathcal{K}_{\mathbb{S}}^{\nu_{\mathbb{S}}(A,R^0)}\left(A, R^0\right) = \mathcal{K}_{\mathbb{S}}^{\nu_{\mathbb{S}}(A,R^0)}\left(A, R^0\right).$$

By definition $\mathcal{K}_{\mathbb{S}}^{\nu_{\mathbb{S}}(A,R^0)}(A,R^0)$ contains R^0. This yields

$$R^0 \in A\mathcal{K}_{\mathbb{S}}^{\nu_{\mathbb{S}}(A,R^0)}\left(A,R^0\right).$$

As A is invertible, we can apply A^{-1} to get

$$X^* - X^0 \in \mathcal{K}_{\mathbb{S}}^{\nu_{\mathbb{S}}(A,R^0)}\left(A,R^0\right).$$

Adding X^0 completes the proof. □

This result is more of theoretical interest as the block grade in real world problems is usually quite high. In practice, often much fewer iterations are needed to reduce the residual norm sufficiently. As we will see later, a more practical relevant quantity is defined by

$$\xi_{\mathbb{S}}(A,R^0) = \min\left\{k \in \mathbb{N} \,|\, \dim \mathcal{K}_{\mathbb{S}}^k\left(A,R^0\right) < k \dim \mathbb{S}\right\}. \tag{3.3}$$

It is the iteration number in which the Krylov space does not grow by $k \dim \mathbb{S}$ dimensions in every iteration. This leads to a situation that must be treated numerically.

Next, we define a inner product on the vector space of block vectors $\mathbb{R}^{n\times s}$ that is a generalization of the scalar product.

Definition 3.7 (block inner product)**.** *A mapping $\langle\cdot,\cdot\rangle_{\mathbb{S}} : \mathbb{R}^{n\times s} \times \mathbb{R}^{n\times s} \to \mathbb{S}$ is called a block inner product if the following conditions hold for all $X, Y, Z \in \mathbb{R}^{n\times s}$ and $\gamma \in \mathbb{S}$:*

- *$\mathbb{S}$-linearity: $\langle X + Y, Z\gamma\rangle_{\mathbb{S}} = \langle X, Z\rangle_{\mathbb{S}}\gamma + \langle Y, Z\rangle_{\mathbb{S}}\gamma$*
- *symmetry: $\langle X, Y\rangle_{\mathbb{S}} = \langle Y, X\rangle_{\mathbb{S}}^{\mathsf{T}}$*
- *definiteness: $\langle X, X\rangle_{\mathbb{S}}$ is positive definite for all full rank X, and $\langle X, X\rangle_{\mathbb{S}} = 0$, if and only if $X = 0$.*
- *normality: $\operatorname{tr}\left(\langle X, Y\rangle_{\mathbb{S}}\right) = \langle X, Y\rangle_F$*

Here $\langle X, Y\rangle_F = \operatorname{tr}\left(X^{\mathsf{T}}Y\right)$ denotes the Frobenius scalar product.

Remarks. Note that the definitness condition implies that a block inner product can only be defined on *-subalgebras that contain full rank matrices. In particular, this yields that the identity of $\mathbb{S}$ is the same as the identity in $\mathbb{R}^{s\times s}$.

Definition 3.8 (normalizer)**.** *We call a map*

$$\operatorname{Norm}_{\mathbb{S}} : \mathbb{R}^{n\times s} \to \mathbb{S}$$

a normalizer or scaling quotient, if for all $X \in \mathbb{R}^{n\times s}$ there exists a Y such that

$$X = Y \operatorname{Norm}_{\mathbb{S}}(X) \qquad \text{and} \qquad \langle Y, Y\rangle_{\mathbb{S}} = \mathbb{I}.$$

Remarks.

- A normalizer can be computed by a QR factorization.
- In the work of Frommer, Lund and Szyld, the scaling quotient is only required to be defined for full rank X. We use the more restrictive definition for our stabilization strategies and to resolve breakdowns in the block Krylov methods.
- We use the Householder algorithm to compute a normalizer in our code. The Gram-Schmidt orthogonalization progress would fail for rank-deficient block vectors. However, there is work to mitigate this, for example by replacing linear dependent columns with random vectors [Soo15].
- In algorithms we write the normalizer in python style syntax

$$Y, \sigma = \mathrm{Norm}_{\mathbb{S}}(X).$$

Next, we take a look at different choices for the *-subalgebra $\mathbb{S}$. We first introduce three elementary cases, that where also considered by Frommer, Szyld and Lund [FSL17], while we use a different naming scheme.

Definition 3.9 (elementary *-subalgebras)**.** *We consider the following three elementary cases of how to treat a block system within the Krylov framework.*

1. *global: The block system is considered as one linear system. This linear system can be represented by the Kronecker system*

$$(\mathbb{I}_s \otimes A)\,\mathrm{vec}(X) = \mathrm{vec}(B)$$

or

$$\begin{pmatrix} A & & \\ & \ddots & \\ & & A \end{pmatrix} \begin{pmatrix} X_1 \\ \vdots \\ X_s \end{pmatrix} = \begin{pmatrix} B_1 \\ \vdots \\ B_s \end{pmatrix}.$$

*This choice corresponds to the *-subalgebra of multiples of the identity $\mathbb{S}_G = \mathbb{R} \cdot \mathbb{I}$ and the Frobenius inner product*

$$\langle X, Y \rangle_{\mathbb{S}_G} = \left(\sum_{i=1}^{s} {X_i}^{\mathsf{T}} Y_i \right) \mathbb{I}.$$

This method goes back to Jbilou, Messaoudi and Sadok [JMS99] for the FOM and GMRes method.

2. *parallel: All columns of the block system are considered separately, but iterations are carried out simultaneously. This corresponds to the *-subalgebra of diagonal matrices $\mathbb{S}_P = \mathrm{diag}(\mathbb{R}^s)$ and the inner block product*

$$\langle X, Y \rangle_{\mathbb{S}_P} = \mathrm{diag}\left(X^{\mathsf{T}} Y \right).$$

3. *block: The classic block Krylov case as presented by O'Leary. This corresponds to the *-subalgebra* $\mathbb{S}_B = \mathbb{R}^{s\times s}$ *and the inner product*

$$\langle X, Y\rangle_{\mathbb{S}_B} = X^\mathsf{T} Y.$$

From these three cases we compose more complex cases as the following definition shows.

Definition 3.10 (Relevant *-subalgebras)**.** *Let* $p \in \mathbb{N}$ *be a divider of* s, $q = \frac{s}{p}$ *and* $X, Y \in \mathbb{R}^{n\times s}$. *We subdivide* X, Y *column-wise into* $\mathbb{R}^{n\times p}$ *matrices*

$$X = [X_1, \ldots, X_q] \qquad\qquad Y = [Y_1, \ldots, Y_q]\,.$$

*Then we define the following *-subalgebras and corresponding block inner products:*

$$\begin{aligned}
&\textit{block-parallel:} && \mathbb{S}^p_{BP} := \operatorname{diag}\left(\left(\mathbb{R}^{p\times p}\right)^q\right) && \langle X, Y\rangle_{\mathbb{S}^p_{BP}} := \operatorname{diag}\left({X_1}^\mathsf{T} Y_1, \ldots, {X_q}^\mathsf{T} Y_q\right),\\
&\textit{block-global:} && \mathbb{S}^p_{BG} := \mathbb{I}_q \otimes \mathbb{R}^{p\times p} && \langle X, Y\rangle_{\mathbb{S}^p_{BG}} := \frac{1}{q}\sum_{i=1}^{q} \mathbb{I}_q \otimes {X_i}^\mathsf{T} Y_i,
\end{aligned}$$

where $\mathbb{I}_q$ *denotes the* q *dimensional identity matrix and* $\operatorname{diag}\left((\mathbb{R}^{p\times p})^q\right)$ *denotes the set of* $s \times s$ *matrices where only the* $p \times p$ *diagonal matrices have non-zero values.*

In principle also a global-parallel combination would be possible. As we want to make use of the advantages of the block strategy, we do not consider this in the present thesis. It would also be possible to apply the elementary case in a different order and construct parallel-block or global-block methods. The resulting *-algebras would be isomorphic to the ones of the block-parallel and block-global methods. Therefore, we restrict ourselves to the two mentioned cases.

Figure 3.1 shows a schematic representation of elements in the different *-subalgebras of $\mathbb{R}^{4\times 4}$. Same colors mean a coupling of the coefficients. White coefficients are restricted to be zero.

In the work of Frommer, Szyld and Lund the block case is called *classic*, the parallel method is called *loop-interchange* and the block-parallel case is called *hybrid*. The block-global method is not considered in their work. We chose the new naming scheme because it feels more natural, as we derive the new cases from the three elementary ones.

To get a feeling for the different *-subalgebras we look at the following example of computing the normalizer of a block vector X in the block-global case.

Example 3.11 (Normalizer in the block-global *-subalgebra)**.** *We can compute the normalizer of the block vector* $[X_1, \ldots, X_q]$ *in the block-global case by computing a QR*

(a) global $\mathbb{S}_G$

(b) parallel $\mathbb{S}_P$

(c) classic $\mathbb{S}_B$

(d) block-parallel $\mathbb{S}_{BP}^2$

(e) block-global $\mathbb{S}_{BG}^2$

Figure 3.1: Schematic representation of different *-subalgebras. Same colors mean the coefficients are coupled. White means restricted to zero. The top row shows the elementary cases. The bottom row shows the combined *-subalgebras. Inspired by [Lun18, Table 3.1].

decomposition

$$\begin{bmatrix} X_1 \\ \vdots \\ X_q \end{bmatrix} = \begin{bmatrix} \widetilde{Y}_1 \\ \vdots \\ \widetilde{Y}_q \end{bmatrix} \rho, \qquad \rho \in \mathbb{R}^{p \times p}.$$

To enforce the normalization we set the normalizer as

$$\operatorname{Norm}_{\mathbb{S}_{BG}^p}(X) = \mathbb{I}_q \otimes \frac{1}{\sqrt{q}} \rho \in \mathbb{S}_{BG}.$$

Then, the block vector $Y = \sqrt{q}\widetilde{Y}$ *is normalized as*

$$\begin{aligned} \langle Y, Y \rangle_{\mathbb{S}_{BG}^p} &= \mathbb{I}_q \otimes \sum_{i=0}^{q} \widetilde{Y}_i^{\mathsf{T}} \widetilde{Y}_i \\ &= \mathbb{I}_q \otimes \mathbb{I}_p = \mathbb{I}_s. \end{aligned}$$

The following lemma derives directly from Definitions 3.9 and 3.10.

Lemma 3.12 (Embeddings of *-subalgebras)**.** *For* $p_1, p_2 \in \mathbb{N}$*, where* p_1 *is a divisor of* p_2 *and* p_2 *is a divisor of* s*, we have the following embedding:*

$$\begin{array}{ccccccc} \mathbb{S}_P & \subseteq & \mathbb{S}_{BP}^{p_1} & \subseteq & \mathbb{S}_{BP}^{p_2} & \subseteq & \mathbb{S}_B \\ \cup\mathrm{I} & & \cup\mathrm{I} & & \cup\mathrm{I} & & \mathrm{II} \\ \mathbb{S}_G & \subseteq & \mathbb{S}_{BG}^{p_1} & \subseteq & \mathbb{S}_{BG}^{p_2} & \subseteq & \mathbb{S}_B \end{array}$$

Note that due to Lemma 3.4 we have the analog embeddings for the corresponding Krylov spaces.

As a closure of this section we consider the more generic case of a linear operator $\boldsymbol{A} \in L(\mathbb{R}^{n \times s}, \mathbb{R}^{n \times s})$, and introduce a classification for this type of operator. This is in analogy of symmetry and definiteness in the scalar case ($s = 1$).

Definition 3.13. *Let $\boldsymbol{A} \in L(\mathbb{R}^{n\times s}, \mathbb{R}^{n\times s})$ be a linear operator on $\mathbb{R}^{n\times s}$ and $\mathbb{S}$ a *-subalgebra with a block inner product $\langle\cdot,\cdot\rangle_{\mathbb{S}}$. We call $\boldsymbol{A}$*

- *block self-adjoint (BSA), if for all $X, Y \in \mathbb{R}^{n\times s}$ holds*

$$\langle \boldsymbol{A}X, Y\rangle_{\mathbb{S}} = \langle X, \boldsymbol{A}Y\rangle_{\mathbb{S}}.$$

- *block positive definite (BPD), if*
  - *a) $\boldsymbol{A}$ is BSA and for all $X \in \mathbb{R}^{n\times s}$ with full rank, $\langle X, \boldsymbol{A}X\rangle_{\mathbb{S}}$ is self-adjoint and positive definite and*
  - *b) for all rank-deficient $X \neq 0$, $\langle X, \boldsymbol{A}X\rangle_{\mathbb{S}}$ is self-adjoint, positive semi-definite and non-zero.*

Remarks. Consider the following representation of the operator $\boldsymbol{A}$ that operates on the vectorization of $\mathbb{R}^{n\times s}$,

$$\hat{\boldsymbol{A}} = \begin{bmatrix} A_{1,1} & \cdots & A_{1,s} \\ \vdots & & \vdots \\ A_{s,1} & \cdots & A_{s,s} \end{bmatrix}.$$

That means

$$(\boldsymbol{A}X)_i = \sum_{j=1}^{s} A_{ij}X_j.$$

Then we distinguish the following cases

- global ($\mathbb{S}_G$):

$\boldsymbol{A}$ is BSA	$\Leftrightarrow$	$\hat{\boldsymbol{A}}$ is symmetric
$\boldsymbol{A}$ is BPD	$\Leftrightarrow$	$\hat{\boldsymbol{A}}$ is symmetric positive definite

- parallel ($\mathbb{S}_P$):

$\boldsymbol{A}$ is BSA	$\Leftrightarrow$	$A_{i,j} = 0 \quad \forall j \neq i \in \{1,\dots,s\}$ and
		$A_{i,i}$ is symmetric $\quad \forall i \in \{1,\dots,s\}$
$\boldsymbol{A}$ is BPD	$\Leftrightarrow$	$\boldsymbol{A}$ is BSA and
		$A_{i,i}$ is positive definite $\forall i = 1,\dots,s$

- block ($\mathbb{S}_B$):

$$\begin{aligned}
\boldsymbol{A} \text{ is BSA} \quad &\Leftrightarrow \quad A_{i,j} = 0 \quad \forall j \neq i \in \{1, \dots, s\} \text{ and} \\
&\qquad A_{i,i} = A_{j,j} \quad \forall j, i \in \{1, \dots, s\} \text{ and} \\
&\qquad A_{i,i} \text{ is symmetric} \quad \forall i \neq j \in \{i, \dots, s\} \\
\boldsymbol{A} \text{ is BPD} \quad &\Leftrightarrow \quad \boldsymbol{A} \text{ is BSA and} \\
&\qquad A_{i,i} \text{ is positive definite } \forall i = 1, \dots, s
\end{aligned}$$

As we only consider block linear systems as defined by Equation (3.1), the operator $\boldsymbol{A}$ defined by

$$(\boldsymbol{A}X)_i = AX_i$$

is BSA if A is symmetric and BPD if A is symmetric positive definite for all the mentioned cases.

Finally we define orthogonally for the block inner product.

Definition 3.14 (block orthogonality)**.** *Let $X, Y \in \mathbb{R}^{n \times s}$ be two block vectors and $\boldsymbol{A}$ a BPD operator. We call X, Y*

- *$\mathbb{S}$-orthogonal if*

$$\langle X, Y \rangle_{\mathbb{S}} = 0$$

  holds.
- *$\mathbb{S}$-$\boldsymbol{A}$-orthogonal if*

$$\langle \boldsymbol{A}X, Y \rangle_{\mathbb{S}} = 0$$

  holds.

After reviewing the theoretical aspects of the block Krylov framework, we take a look at the practical parts in the next sections.

3.2 Implementation

Our implementation builds upon the SIMD interface in the C++ software framework Dune [Bas+20b; Bla+16; Bas+08b; Bas+08a]. This ensures that we make use of the SIMD capabilities of the hardware and offers a way to implement horizontal parallelism easily. SIMD data types behave like a numeric type (e.g. `double`) but process multiple values at once. The coefficients of a SIMD data type are called the *lanes*. The type of one lane is called the *scalar type* of the SIMD data type.

```
template<class T, std::size_t S, std::size_t A>
auto operator +(const LoopSIMD<T,S,A> &v,
                const LoopSIMD<T,S,A> &w) {
  LoopSIMD<T,S,A> out;
  for(std::size_t i=0; i<S; i++){
    out[i] = v[i] + w[i];
  }
  return out;
}
```

Listing 3.1: Implementation of the `operator+` of `Dune::LoopSIMD`.

Supported SIMD data types include VCL [Fog] and Vc [KL12; Kre15]. Furthermore, DUNE provides a simple fallback implementation, `Dune::LoopSIMD`, that is based on static loops and relies on compiler optimization for the exploration of SIMD instructions. In the future it is planed to support the SIMD features of the C++ standard once the parallelism TSv2[1] is merged into the standard.

The SIMD interface of DUNE unifies the usage of the SIMD specific operations that differ for different implementations. The essential components of the interface for a SIMD data type `T` are

- `Simd::lane(size_t l, T x)`: provides access to a single lane
- `Simd::lanes()`: provides access to the number of lanes (SIMD width)
- `SIMD::Scalar<T>`: the scalar data type (e.g. `double`)

Further features include the evaluation of conditional expressions and the implementation of the math functions in C++, like `min`, `max`, `sin` etc.

`Dune::LoopSIMD` is a fallback implementation in DUNE. It inherits from `std::array` and implements the SIMD interface by overloading all arithmetic operators. For example, the implementation of the `operator+` can be found in Listing 3.1. The performance gain of this data type depends on the optimizations of the compiler. In particular one problem, in which recent compilers fail, are the use of FMA operations in expressions like `a += alpha*b`, if it involves many lanes.

Another feature of `Dune::LoopSIMD` is that it can be used to concatenate another SIMD type to a large one. For example VCL only implements types with the hardware SIMD width, i.e. 4 or 8. If we want to use larger SIMD types we use `Dune::LoopSIMD` to concatenate multiple `Vec8d` or `Vec4d`. For example, `Dune::LoopSIMD<Vec8d, 4>` is a SIMD data type with 32 lanes.

To use a SIMD data type in a solver a vector type must be specified, which represents the block vector space $\mathbb{R}^{n\times s}$. This could be achieved by using the SIMD data type as the `field_type` in the `Dune::BlockVector` template. Practically, this represents a row major storage of the block vectors. For this vector type a `Dune::LinearOperator`

[1] see for example `https://en.cppreference.com/w/cpp/experimental/parallelism_2`

```
typedef Dune::LoopSIMD<double, 16> field_type;
typedef Dune::BlockVector<field_type> vector_type;
typedef Dune::BCRSMatrix<double> matrix_type;
typedef Dune::MatrixAdapter<matrix_type, vector_type,
   vector_type> linear_operator_type;
```

Listing 3.2: Setup of a linear operator type that operates on blockvectors based on a sparse matrix.

```
template<class X>
class Block{
  // the BAXPY operation x += alpha*Y*sigma
  void axpy(const scalar alpha, X& x, const X& y) const;
  Block& invert(); // inverts the block
  Block& transpose(); // transposes the block
  Block& add(const Block& other); // add another block
  Block& scale(const scalar& factor); // scale with a scalar
  // multiplication with other Block
  Block& leftmultiply(const Block& other);
  Block& rightmultiply(const Block& other);
};
```

Listing 3.3: The Block interface used to represent elements in $\mathbb{R}^{s\times s}$.

that represents the operator $A : \mathbb{R}^{n\times s} \to \mathbb{R}^{n\times s}$ can be implemented for example by a `Dune::MatrixAdatper` that takes a sparse matrix and turns it into a linear operator. A setup of the linear operator type can be seen in Listing 3.2. Preconditioners can be set up based on the same vector type.

As we want to be flexible with the choice of the *-subalgebra, the block inner product is implemented in a generic fashion. The existing solvers in Dune-ISTL [BB07] are build upon an interface called `Dune::ScalarProduct`. We extend this concept of a scalar product, i.e. the return type of the scalar product is not a scalar, but a `Block`, which is a type-erasure container that provides the functionalities shown in Listing 3.3.

This design enables us to implement the kernels for different *-subalgebras in a specialized way. The fallback implementation is the parallel case $\mathbb{S}_P$, which was the default behavior in Dune before.

We extend the interface further by a function `inormalizer(X& x)` that computes and returns the normalizer $\mathrm{Norm}_{\mathbb{S}}(X)$ and normalizes the block vector `x` with respect to the computed normalizer. The `i` as a prefix is inspired by the non-blocking MPI functions and indicates that the function returns a `Future` (see Section 7.1). In the sequential case, the normalizer is computed using the LAPACK [And+99] function `xGEQRF`, which uses Householder transformations to compute the QR decomposition. An elaborate discussion about how to compute the normalizer in the parallel case can be found in Section 7.3.

3.3 Performance Analysis

Now we look at the performance characteristics of the building blocks that are needed to build a block Krylov method. These are

- `BOP`: Applying the operator A
- `BDOT`: Compute the block inner product
- `BAXPY`: Block vector update

As already mentioned, the implementation of the normalizer relies on LAPACK in the sequential case. Therefore, we do not discuss its performance here. The performance of the preconditioner depends of course of its choice. For simplicity, we assume that the preconditioner behaves similar to `BOP`.

We assume in this section that the operator is an assembled sparse matrix in CSR format with z non-zeros. We further assume that the column index in the CSR format needs as much space in memory as the coefficient (e.g. 64-bit for `int` and `double`). This is the unit in which we denote data size. The row indices for the sparse matrix are neglected. This leads to a total memory requirement for the matrix of $2z$. Together with the input and output block vector $\beta_{\texttt{BOP}} = 2z + 2sn$ values must be transferred from the main memory to the registers. For the `BOP` operation $\omega_{\texttt{BOP}} = 2sz$ floating-point operations are necessary. Hence, we get an operational intensity of $\frac{sz}{z+sn}$. This means the operational intensity is higher (better) for more right-hand sides s or more non-zeros z.

The situation is a bit more sophisticated for the `BDOT` and `BAXPY` kernels. However, both kernels behave quite similar. Both operate on two block vectors that must be loaded from the main memory, which yields $\beta_{\texttt{BDOT}} = 2ns$. The difference between the kernels is that the `BAXPY` kernel writes one block vector back to the main memory. Therefore, we have $\beta_{\texttt{BAXPY}} = 3ns$ memory transfers. We assume that the data of the *-subalgebra element can be cached and therefore does not need to be communicated through the memory hierarchy.

The number of floating-point operations depend on the *-subalgebra. In this analysis we consider the cases $\mathbb{S}^p_{BP}$ and $\mathbb{S}^p_{BG}$ from Definition 3.10. For both, `BDOT` and `BAXPY`, the number of floating-point operations increases quadratic with p and we have $\omega_{\texttt{BDOT}} = \omega_{\texttt{BAXPY}} = 2np^2q$. This yields an arithmetic intensity of p for `BOP` and $\frac{2}{3}p$ for `BAXPY`. Table 3.1 summarizes the numerical characteristics of the kernels.

This is the great advantage of the presented framework. The parameter p can be tuned such that the arithmetic intensity matches the properties of the hardware. For many right-hand sides s and small p all kernels would be memory-bound and the costs are independent of p, as the amount of data that must be loaded does not depend on p. Therefore, the parameter p can be chosen as large such that the p^2 scaling of the kernels does not have an effect. Up to that p the faster convergence of the block method comes for free and the better arithmetical intensity of the `BOP` kernel for large s can be preserved.

Table 3.1: Performance relevant characteristics for the `BOP`, `BDOT` and `BAXPY` kernels. The columns denote the number of floating-point operations ω, amount of data loaded from main memory β and the arithmetic intensity $\frac{\omega}{\beta}$. The number of non-zeros in A are denoted by z.

	ω	β	arith. intensity
`BOP`	$2sz$	$2z + 2sn$	$\frac{sz}{z+sn}$
`BDOT`	$2np^2q$	$2ns$	p
`BAXPY`	$2np^2q$	$3ns$	$\frac{2}{3}p$

To achieve the best performance the kernels must be implemented very carefully. In particular, one must ensure a good data locality. For our implementation, we iterate over the rows of the block vectors in chunks of 4 rows. We found this number experimentally and suppose that the optimal number depends on the number of registers of the CPU and for how many cycles a FMA operation occupies the registers. Within these chunks we iterate over the rows and compute the corresponding matrix-matrix products. The implementations of the matrix-matrix products are shown in Listing 3.4. This approach was already presented by Stewart [Ste08].

3.4 Numerical Experiments

To compare the different methods in practice, we executed several tests. In this chapter, all tests are carried out on our compute server, which is an Intel Skylake-SP Xeon Gold 6148 with 377 GB main memory. To make the results as reproducible as possible, we deactivate the turbo mode. In the described setting the system has a theoretical peak performance of 76.8 GFlop/s ($= 2.4\,\text{GHz} * 32\,\text{Flop/cy}$) using one core. Measurements show a memory bandwidth of 13.34 GB/s, measured with the `daxpy` benchmark of the likwid-bench suite [THW10].

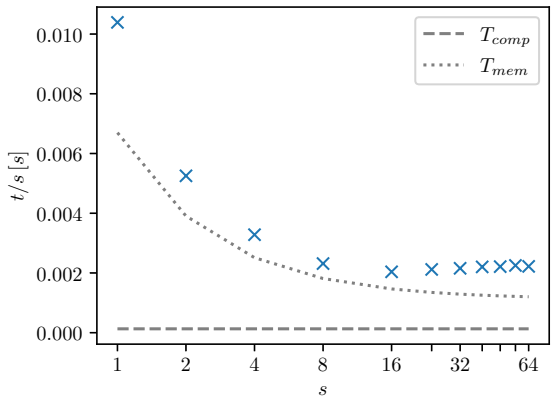
Multi-core tests are executed on 20 cores of the machine, which is one NUMA node. In this setting the frequency reduces to 2.2 GHz when using AVX-512 instructions, leading to a theoretical peak performance of 1408 GFlop/s ($= 20 * 2.2\,\text{GHz} * 32\,\text{Flop/cy}$). As the cores share the same memory connection, the memory bandwidth does not scale with the number of cores. The measured memory bandwidth with 20 cores is 98.47 GB/s, also measured with the `daxpy` benchmark.

In a first test series we compare the run-times of the building blocks `BOP`, `BAXPY` and `BDOT` for the block-parallel and block-global methods. Figure 3.2 shows the runtimes of the kernels per right-hand side in the single-core case. Figure 3.3 shows the run-time of the same kernels in the multi-core case. We plotted the execution time per right-hand side (t/s). For the `BOP` kernel we carried out the tests for different values of s. We tested two different matrix patterns. One is a very sparse one, resulting from a 2D

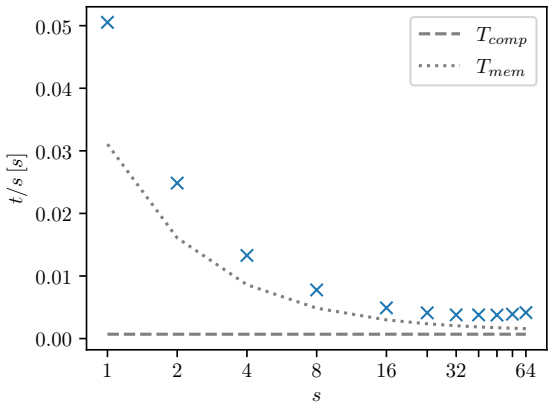
```
// computes c += a^t b
template<class SIMD, size_t ChunkSize>
void mtm(const std::array<SIMD, ChunkSize>& a,
         const std::array<SIMD, ChunkSize>& b,
         std::array<SIMD, lanes<SIMD>()>& c){
  for(size_t i=0; i<ChunkSize; ++i){
    for(size_t j=0; j<lanes<SIMD>(); ++j){
      c[j] += lane(j, a[i])*b[i];
    }
  }
}

// computes c += a b
template<class SIMD, size_t ChunkSize>
void mm(const std::array<SIMD, ChunkSize>& a,
        const std::array<SIMD, lanes<SIMD>()>& b,
        std::array<SIMD, ChunkSize>& c){
  for(size_t i=0; i<lanes<SIMD>(); ++i){
    for(size_t j=0; j<ChunkSize; ++j){
      c[j] += lane(i, a[j])*b[i];
    }
  }
}
```

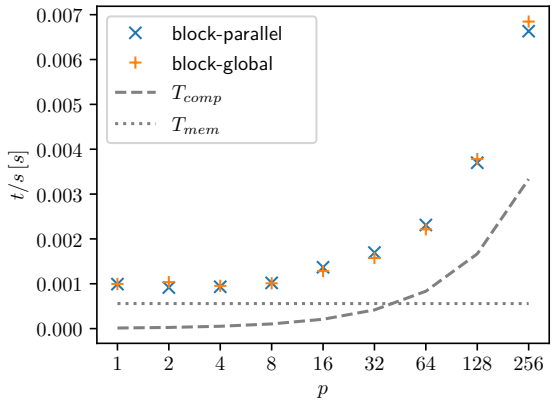
Listing 3.4: Implementation of the inner matrix-matrix products. Block vector rows are iterated in chunks of size `ChunkSize` to increase the arithmetical intensity.



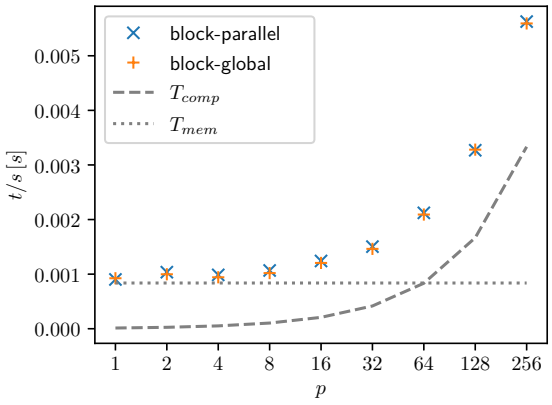
(a) `BOP` with 2D Finite-Differences matrix.



(b) `BOP` with 3D Q1-Finite-Elements matrix.



(c) `BDOT` for the block-parallel and block-global case.



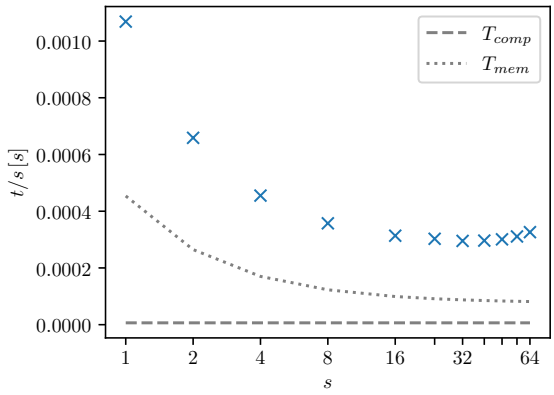
(d) `BAXPY` for the block-parallel and block-global case.

Figure 3.2: Microbenchmarks for kernels `BOP`, `BDOT` and `BAXPY` executed on one core. Crosses mark the measured data. Dotted lines mark the memory bound. Dashed lines mark the compute bound.

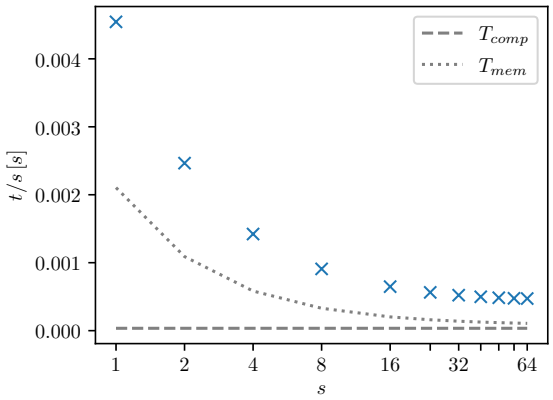
finite differences discretization of a Poisson problem on a 1000×1000 grid with $z = 5$ non-zeros per row. The other one results from a 3D Q1 finite element discretization on a $100 \times 100 \times 100$ grid with $z = 27$ non-zero coefficients per row. For the SIMD interface we used the VC library and combine it with `Dune::LoopSIMD` to assemble larger SIMD data types as described in the previous section.

For the `BDOT` and `BAXPY` kernel we used $s = 256$ and carried out the tests for different p. In the one-core test case we used $n = 500\,000$ and in the multi-core test case we used $n = 6\,000\,000$. Further numerical tests show that the run-time of these kernels scale linearly with s.

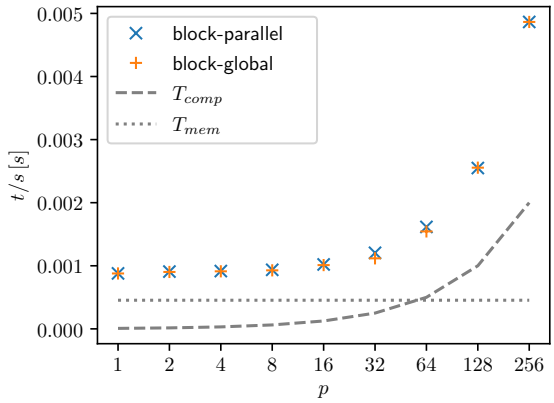
We see that the measured behavior of the kernels matches our theoretical expectation. For the `BOP` kernel it turns out that for all s the kernel is memory bound and it performs more efficient with larger s. We suppose that the slight increase of the runtime per s



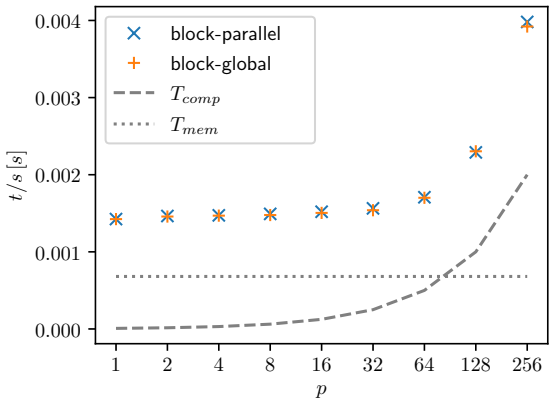
(a) `BOP` with 2D Finite-Differences matrix.



(b) `BOP` with 3D Q1-Finite-Elements matrix.



(c) `BDOT` for the block-parallel and block-global case.



(d) `BAXPY` for the block-parallel and block-global case.

Figure 3.3: Microbenchmarks for kernels `BOP`, `BDOT` and `BAXPY` executed on 20 cores. Crosses mark the measured data. Dotted lines mark the memory bound. Dashed lines mark the compute bound.

for larger s in the 2D finite differences case is due to cache effects, as fewer rows of the block vectors can be cached.

For the `BDOT` and `BAXPY` kernels we see that the runtime is memory bound for $p \lesssim 16$ in the one-core case and memory bound for $p \lesssim 64$ in the multi-core case. In particular the runtime does not depend on p in this regime. For larger p the run-time increases quadratically, as expected. The runtime of the block-global and block-parallel method does not differ. This was predicted by the theory as well. We already published similar results in [DE20].

4

Block Conjugate Gradients Method

For the solution of large sparse symmetric positive definite linear systems, the Conjugate Gradients method combined with a proper preconditioner is the method of choice. In the following, we reformulate the block Conjugate Gradients (BCG) method as proposed by O'Leary [OLe80] based on the general framework presented in the last chapter and introduce a novel adaptive stabilization technique based on the paper by Dubrulle [Dub01].

We consider A as a symmetric positive definite operator, as this is a requirement of the BCG method. Furthermore, in this chapter $M \in L(\mathbb{R}^n, \mathbb{R}^n)$ denotes a symmetric positive definite preconditioner.

4.1 Formulation of the Block Conjugate Gradients Method

The objective of the BCG method in the kth iteration is to find an approximation $X^k \in X^0 + \mathcal{K}_{\mathbb{S}}^k\left(M^{-1}A, M^{-1}R^0\right)$, which minimizes the block energy error

$$X^k = \underset{Y \in X^0 + \mathcal{K}_{\mathbb{S}}^k(M^{-1}A, M^{-1}R^0)}{\arg\min} \|Y - X^*\|_{A,F}. \tag{4.1}$$

Before characterizing this minimization property in more detail, we look at a small auxiliary lemma.

Lemma 4.1. *Let $\eta \in \mathbb{S}$ with*

$$\operatorname{tr}(\eta\sigma) = 0 \qquad \forall \sigma \in \mathbb{S}.$$

Then $\eta = 0$ must hold.

Proof. Assume that $\eta \neq 0$ and choose $\sigma = \eta^{\mathsf{T}}$. It follows

$$0 = \operatorname{tr}(\eta\sigma) = \operatorname{tr}\left(\eta\eta^{\mathsf{T}}\right) > 0.$$

Because $\eta\eta^{\mathsf{T}} \neq 0$ is a positive semi-definite matrix, for which the trace is the sum of its eigenvalues. □

With this lemma, we formulate the following theorem.

Theorem 4.2. *The minimization property* (4.1) *is equivalent to the orthogonality condition*

$$\langle R^k, X\rangle_{\mathbb{S}} = 0 \qquad \forall X \in \mathcal{K}^k_{\mathbb{S}}\left(M^{-1}A, M^{-1}R^0\right), \tag{4.2}$$

where $R^k = AX^k - B$ *is the residual for the approximation* X^k*.*

Proof. For any $X \in \mathcal{K}^k_{\mathbb{S}}(M^{-1}A, M^{-1}R^0)$, we define the coercive functional

$$\begin{aligned}\mathcal{J}_X(\varepsilon) &= \frac{1}{2}\|X^k + \varepsilon X - X^*\|^2_{A,F} \\ &= \frac{1}{2}\operatorname{tr}\left(\langle X^k + \varepsilon X - X^*, R^k + \varepsilon AX\rangle_{\mathbb{S}}\right).\end{aligned}$$

Note that the minimization of (4.1) is equivalent to the minimization of $\mathcal{J}_X$ for all $X \in \mathcal{K}^k_{\mathbb{S}}(M^{-1}A, M^{-1}R^0)$. Due to the linearity of the trace and the block inner product, the differential of $\mathcal{J}_X$ computes

$$D_\varepsilon\mathcal{J}_X(\varepsilon) = \operatorname{tr}\left(\langle R^k, X\rangle_{\mathbb{S}}\right) + \varepsilon\operatorname{tr}\left(\langle X, AX\rangle_{\mathbb{S}}\right).$$

Here we used that $\operatorname{tr}(\langle X, Y\rangle_{\mathbb{S}}) = \operatorname{tr}\left(\langle Y, X\rangle_{\mathbb{S}}{}^{\mathsf{T}}\right) = \operatorname{tr}(\langle Y, X\rangle_{\mathbb{S}})$. For the first implication, we assume that X^k is a minimizer of (4.1). Hence, we have

$$0 = D_\varepsilon\mathcal{J}_{X\sigma}(0) = \operatorname{tr}\left(\langle R^k, X\sigma\rangle_{\mathbb{S}}\right) \qquad \forall X \in \mathcal{K}^k_{\mathbb{S}}\left(M^{-1}A, M^{-1}R^0\right), \sigma \in \mathbb{S}.$$

From Lemma 4.1, we obtain

$$\langle R^k, X\rangle_{\mathbb{S}} = 0 \qquad \forall X \in \mathcal{K}^k_{\mathbb{S}}\left(M^{-1}A, M^{-1}R^0\right).$$

For the other implication, we assume that $\langle R^k, X\rangle_{\mathbb{S}} = 0$ for all $X \in \mathcal{K}^k_{\mathbb{S}}(M^{-1}A, M^{-1}R^0)$. This yields

$$D_\varepsilon\mathcal{J}_X(\varepsilon) = \varepsilon\operatorname{tr}\left(\langle X, AX\rangle_{\mathbb{S}}\right).$$

Hence, $D_\varepsilon\mathcal{J}_X(0)$ vanishes for all $X \in \mathcal{K}^k_{\mathbb{S}}(M^{-1}A, M^{-1}R^0)$. Thus, X^k is a minimizer of (4.1). □

Now we deduce the formulas for the method. The method computes an A-$\mathbb{S}$ orthogonal basis $\{P^j\}_{j=0}^k$ of the block Krylov space $\mathcal{K}_{\mathbb{S}}^{k+1}\left(M^{-1}A, M^{-1}R^0\right)$. This basis is used to update the initial guess and residual iteratively

$$X^{k+1} = X^k + P^k\lambda^k \tag{4.3}$$
$$R^{k+1} = R^k - AP^k\lambda^k. \tag{4.4}$$

To compute the coefficient $\lambda^k \in \mathbb{S}$, let an A-$\mathbb{S}$-block orthogonal basis $\{P^j\}_{j=0}^k$ be given and let X^k be a minimizer of (4.1). By Theorem 4.2, we obtain the minimizer of (4.1) for the following block Krylov space by

$$\begin{aligned} 0 &= \langle R^{k+1}, X\rangle_{\mathbb{S}} && \forall X \in \mathcal{K}_{\mathbb{S}}^{k+1}\left(M^{-1}A, M^{-1}R^0\right) \\ &= \langle R^k, X\rangle_{\mathbb{S}} - \langle AP^k\lambda^k, X\rangle_{\mathbb{S}}. \end{aligned}$$

For X we choose the basis $\{P^j\}_{j=0}^k$ and get

$$0 = \langle R^k, P^j\rangle_{\mathbb{S}} - \langle AP^k\lambda^k, P^j\rangle_{\mathbb{S}} \qquad \forall j = 0, \ldots, k.$$

Due to the A-$\mathbb{S}$-orthogonality, this yields

$$0 = \langle R^k, P^j\rangle_{\mathbb{S}} \qquad \forall j = 0, \ldots, k-1 \tag{4.5}$$

and

$$\lambda^k = \left(\langle P^k, AP^k\rangle_{\mathbb{S}}\right)^{-1}\langle P^k, R^k\rangle_{\mathbb{S}}. \tag{4.6}$$

By Theorem 4.2, Equation (4.5) holds, as X^k is a minimizer in the Krylov space $\mathcal{K}_{\mathbb{S}}^k\left(A, R^0\right)$. We use Equation (4.6) as a definition for λ^k.

The next basis vector P^{k+1} is then obtained by A-$\mathbb{S}$-orthogonalizing the preconditioned residual $M^{-1}R^{k+1}$ against the previous basis vectors, it reads

$$P^{k+1} = M^{-1}R^{k+1} - \sum_{j=0}^{k} P^j\left(\langle P^j, AP^j\rangle_{\mathbb{S}}\right)^{-1}\langle P^j, AM^{-1}R^{k+1}\rangle_{\mathbb{S}}.$$

Next, we show that for $j = 0, \ldots, k-1$, the coefficient in the orthogonalization vanishes. As A and M are symmetric and $M^{-1}AP^j \in \mathcal{K}_{\mathbb{S}}^{j+2}\left(M^{-1}A, M^{-1}R^0\right)$, we have

$$\langle P^j, AM^{-1}R^{k+1}\rangle_{\mathbb{S}} = \langle M^{-1}AP^j, R^{k+1}\rangle_{\mathbb{S}} = 0,$$

by using the orthogonality from Theorem 4.2. Hence, we get the update formula

$$P^{k+1} = M^{-1}R^{k+1} + P^k\beta^k \tag{4.7}$$

Algorithm 4.1: Block Conjugate Gradients Method

$R^0 = B - AX^0$
$P^0 = M^{-1}R^0$
$\rho^0 = \langle P^0, R^0\rangle_{\mathbb{S}}$
for $k = 0, \ldots$ until convergence **do**
  $Q^k = AP^k$
  $\alpha^k = \langle P^k, Q^k\rangle_{\mathbb{S}}$
  $\lambda^k = \left(\alpha^k\right)^{-1}\rho^k$
  $X^{k+1} = X^k + P^k\lambda^k$
  $R^{k+1} = R^k - Q^k\lambda^k$
  break if $\|R^{k+1}\| < \varepsilon_{\text{tol}}$
  $Z^{k+1} = M^{-1}R^{k+1}$
  $\rho^{k+1} = \langle Z^{k+1}, R^{k+1}\rangle_{\mathbb{S}}$
  $\beta^k = \left(\rho^k\right)^{-1}\rho^{k+1}$
  $P^{k+1} = Z^{k+1} + P^k\beta^k$
end for

with

$$\beta^k = \left(\langle P^k, AP^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-1}AP^k, R^{k+1}\rangle_{\mathbb{S}}.$$

To reduce the number of block inner products, we reformulate the coefficients λ^k and β^k as follows

$$\begin{aligned}
\lambda^k &= \left(\langle P^k, AP^k\rangle_{\mathbb{S}}\right)^{-1}\langle P^k, R^k\rangle_{\mathbb{S}} \\
&= \left(\langle P^k, AP^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-1}R^k - P^{k-1}\beta^{k-1}, R^k\rangle_{\mathbb{S}} \\
&= \left(\langle P^k, AP^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-1}R^k, R^k\rangle_{\mathbb{S}}, \qquad (4.8)\\
\beta^k &= \left(\langle P^k, AP^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-1}AP^k, R^{k+1}\rangle_{\mathbb{S}} \\
&= \left(\langle P^k, AP^k\rangle_{\mathbb{S}}\right)^{-1}\lambda^{k-\mathsf{T}}\langle R^k - R^{k+1}, M^{-1}R^{k+1}\rangle_{\mathbb{S}} \\
&= \left(\langle M^{-1}R^k, R^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-1}R^{k+1}, R^{k+1}\rangle_{\mathbb{S}}. \qquad (4.9)
\end{aligned}$$

Here, we used the orthogonality relation (4.2) and the update formula for the residual (4.4). This reduces the necessary block inner products to

$$\alpha^k = \langle P^k, AP^k\rangle_{\mathbb{S}} \qquad (4.10)$$

and

$$\rho^k = \langle M^{-1}R^k, R^k\rangle_{\mathbb{S}}. \qquad (4.11)$$

Putting together Equations (4.3), (4.4), (4.7), (4.8), (4.9), (4.10) and (4.11), we

obtain Algorithm 4.1.

In Algorithm 4.1, we choose $\|R^k\| < \varepsilon_{\text{tol}}$ as a break criteria, where we do not specify which norm is used. A natural choice would be the Frobenius norm which is the Euclidean norm on the block vector space. However, another possibility would be to choose the maximum column norm

$$\|R^k\|_\infty := \max\left\{\|R_i^k\|_2 \,|\, i = 1, \ldots, s\right\}.$$

That ensures that the residual norm of each column is smaller than ε_{tol}. In all our numerical tests, we used the latter, as this is the desired condition in most applications.

4.2 Convergence

We start with a general result that holds for all choices of *-subalgebras $\mathbb{S}$. From that result, we derive statements about the convergence in the elementary cases. These statements can be combined to obtain statements about the convergence of the combined *-subalgebras defined in Definition 3.9.

Lemma 4.3 (Generic convergence result)**.** *For the error E^k of the kth step of the BCG method, the estimation*

$$\|E^k\|_{A,F} \leq \inf_{\mathcal{Q}^k} \|\mathcal{Q}^k(M^{-1}A) \circ E^0\|_{A,F} \tag{4.12}$$

holds, where the infimum is taken over all $\mathbb{S}$-valued polynomials $\mathcal{Q}^k \in \mathbb{P}_{\mathbb{S}}^k$ of degree k with absolute coefficient $\mathbb{I}$.

Proof. We use $\mathbb{S}$-valued polynomials to represent the energy error. By Lemma 3.3, we can represent the kth error of the BCG method as

$$\begin{aligned} E^k &= X^* - X^k \\ &= X^* - X^0 - \mathcal{P}^k(M^{-1}A) \circ M^{-1}R^0 \\ &= \left(\mathbb{I} - \mathcal{P}^k(M^{-1}A)M^{-1}A\right) \circ E^0, \end{aligned}$$

for some $\mathbb{S}$-valued polynomial $\mathcal{P}^k \in \mathbb{P}_{\mathbb{S}}^{k-1}$. With $\mathcal{Q}^k(\mathrm{x}) = \mathbb{I} - \mathcal{P}^k(\mathrm{x})\mathrm{x}$ and taking the A-Frobenius norm, we obtain

$$\|E^k\|_{A,F} = \|\mathcal{Q}^k(M^{-1}A) \circ E^0\|_{A,F}.$$

As the BCG method minimizes the A-Frobenius norm in the Krylov space, we get

$$\|E^k\|_{A,F} = \inf_{\mathcal{Q}^k} \|\mathcal{Q}^k(M^{-1}A) \circ E^0\|_{A,F}$$

by taking the infimum of all $\mathcal{Q}^k \in \mathbb{P}_{\mathbb{S}}^k$ of this shape. □

The next lemma gives a concrete error bound for all *-subalgebras that we consider in this work. It is the generalization of Theorem 2.4. However, the given bound is not sharp in all cases.

Lemma 4.4. *Let $\mathbb{S}$ be a *-subalgebra of $\mathbb{R}^{s\times s}$, that contains the identity of $\mathbb{R}^{s\times s}$. Then we have*

$$\|E^k\|_{A,F} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \|E^0\|_{A,F},$$

where κ denotes the condition number of the preconditioned operator $M^{-\frac{1}{2}}AM^{-\frac{1}{2}}$.

Proof. We are using Theorem 4.3 and choose

$$\mathcal{Q}^k(\mathrm{x}) = \widetilde{T}^k(\mathrm{x})\mathbb{I}$$

in Equation (4.12), where $\widetilde{T}^k$ are the scaled Chebyshev polynomials defined by Equation (2.5), scaled with respect to the eigenvalues $\lambda_{\min}$ and $\lambda_{\max}$ of the symmetrically preconditioned operator $M^{-\frac{1}{2}}AM^{-\frac{1}{2}}$. As this operator is symmetric positive-definite, it is similar to the diagonal matrix of its eigenvalues Λ, denoted by

$$M^{-\frac{1}{2}}AM^{-\frac{1}{2}} = V\Lambda V^{-1},$$

where V is the orthonormal matrix of the eigenvectors. Similar to the proof of Theorem 2.4, we compute

$$\begin{aligned}
\|\widetilde{T}^k(M^{-1}A)\circ E^0\|_{A,F}^2 &= \mathrm{tr}\left(\left(\widetilde{T}^k(M^{-1}A)E^0\right)^{\mathsf{T}} A\widetilde{T}^k(M^{-1}A)E^0\right)\\
&= \mathrm{tr}\left(\left(\widetilde{T}^k(\Lambda)V^{-1}M^{\frac{1}{2}}E^0\right)^{\mathsf{T}}\Lambda\widetilde{T}^k(\Lambda)V^{-1}M^{\frac{1}{2}}E^0\right)\\
&= \mathrm{tr}\left(\left(V^{-1}M^{\frac{1}{2}}E^0\right)^{\mathsf{T}}\Lambda^{\frac{1}{2}}\widetilde{T}^k(\Lambda)^2\Lambda^{\frac{1}{2}}V^{-1}M^{\frac{1}{2}}E^0\right)\\
&\leq \max_{i=0}^{n}|\widetilde{T}^k(\lambda_i)|^2\,\mathrm{tr}\left({E^0}^{\mathsf{T}}AE^0\right)\\
&\leq 4\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^{2k}\|E^0\|_{A,F}^2.
\end{aligned}$$

Applying the square-root completes the proof. □

This theorem also applies for the block *-subalgebra $\mathbb{S}_B$. However, in this case the estimation can be improved. O'Leary showed the following convergence result to estimate the error of the classical BCG method.

Theorem 4.5 (Convergence of the block Conjugate Gradients Method [OLe80, Theorem 5]). *For the energy-error of the ith column in the kth iteration $\|E_i^k\|_A$ of the BCG method, the following estimation holds:*

$$\|E_i^k\|_A \leq c_1 \mu^k$$
$$\text{with } \mu = \frac{\sqrt{\kappa_s}-1}{\sqrt{\kappa_s}+1}, \kappa_s = \frac{\lambda_n}{\lambda_s} \text{ and some constant } c_1 > 0,$$

where $\lambda_1 \leq \ldots \leq \lambda_n$ denote the eigenvalues of the preconditioned matrix $M^{-\frac{1}{2}} A M^{-\frac{1}{2}}$. The constant c_1 depends on s and the initial error E^0 but not on k or i.

The proof also makes use of on Lemma 4.3 but the construction of the polynomials is much more sophisticated and technical. As we want to concentrate on the practical aspects in this work, we refer the reader to [OLe80] for the rigorous proof.

The theorem holds for the classical BCG method ($\mathbb{S}_B$). However, as the block-parallel method is only a data-parallel version of the block method the same convergence rate holds with $s = p$ for the $\mathbb{S}_{BP}^p$ method, it reads

$$\mu = \frac{\sqrt{\kappa_p}-1}{\sqrt{\kappa_p}+1}.$$

The following lemma gives us a convergence rate for the block-global method.

Lemma 4.6 (Theoretical convergence rate of the block-global method). *The theoretical convergence rate of a block-global method using $\mathbb{S}_{BG}^p$ is*

$$\hat{\mu} = \frac{\sqrt{\hat{\kappa}_p}-1}{\sqrt{\hat{\kappa}_p}+1}, \quad \text{with} \quad \hat{\kappa}_p = \frac{\lambda_n}{\lambda_{\lceil \frac{p}{q} \rceil}}.$$

Proof. A block-global method is equivalent to solve the qn-dimensional system

$$\begin{pmatrix} A & & \\ & A & \\ & & \ddots \\ & & & A \end{pmatrix} \begin{pmatrix} X_1 & \cdots & X_p \\ X_{p+1} & \cdots & X_{2p} \\ & \vdots & \\ X_{s-p+1} & \cdots & X_s \end{pmatrix} = \begin{pmatrix} B_1 & \cdots & B_p \\ B_{p+1} & \cdots & B_{2p} \\ & \vdots & \\ B_{s-p+1} & \cdots & B_s \end{pmatrix}$$

with the classical block Krylov method with p right-hand sides. The matrix of this system has the same eigenvalues as A but with q times the multiplicity. Thus, the p-smallest eigenvalue is $\lambda_{\lceil \frac{p}{q} \rceil}$. Therefore and by applying Theorem 4.5, we deduce the theoretical convergence rate. □

This result makes the block-global methods irrelevant for practical use. In particular for $q > 1$, the block-parallel method would perform better while the building blocks are similarly expensive, as we have seen in Chapter 3.

4.3 Residual Re-Orthonormalization

Algorithm 4.1 requires that $\rho^k = \langle Z^k, R^k \rangle_{\mathbb{S}}$ and $\alpha^k = \langle Q^k, P^k \rangle_{\mathbb{S}}$ are invertible. It is the case when the residual R^k has full-rank. In the scalar case, $s = 1$, this is not a problem. If the residual is rank-deficient, the linear system has been solved.

In the case, $s > 1$, however, this leads to severe problems. An interpretation of the rank deficiency of the residual is that a linear combination is converged, because there exists a vector $y \in \mathbb{R}^s$, such that

$$0 = R^k y = AX^k y - By \qquad \Leftrightarrow \qquad AX^k y = By.$$

In exact arithmetic, this case appears in the iteration $\xi_{\mathbb{S}}(A, R^0)$ defined by Equation (3.3). Initially, O'Leary [OLe80] suggested to remove dependent vectors and continue the iteration with a smaller block size. This strategy is called *deflation* and has two disadvantages from our perspective. Firstly, a numerical tolerance parameter must be introduced to check for the *numerical* rank-deficiency of the residual. Langou [Lan03] showed in his PhD thesis that a badly chosen parameter could lead to instabilities or slow down the convergence. Secondly, we want to choose the block width s as a multiple of the SIMD width to facilitate SIMD-vectorization. Deflating the system would change the block width, such that some effort is needed to handle it in our SIMD setting and we would loose the performance benefits from the exploration of the SIMD instructions.

Dubrulle [Dub01] presented multiple approaches to mitigate the stabilization issues without decreasing the block size. The most promising approach is the orthonormalization of the residual in every iteration by computing a QR decomposition. The algorithm is very elegant without preconditioning, as ρ simplifies to the identity. As we use preconditioning, this does not hold anymore, as the M-product is used to compute ρ. Therefore, we either need to make the orthonormalization with respect to the M-product, or ρ must be computed explicitly after the orthonormalization of the residual, which needs an additional global communication. We decided to use the normalizer for the orthonormalization and compute ρ explicitly thereafter, which is the reason why we defined the normalizer also for rank-deficient block vectors.

We store the transformation from the orthonormal residual $\bar{R}^k$ to the real residual in the variable $\sigma^k \in \mathbb{S}$. It can be updated as

$$\sigma^k = \gamma^k \sigma^{k-1},$$

where γ^k is the normalizer of the updated residual $\widetilde{R}^k = \bar{R}^{k-1} - Q^k \lambda^k$, i.e.

$$\widetilde{R}^k = \bar{R}^k \gamma^k.$$

Algorithm 4.2: BCG Method with Residual Re-Orthonormalization

$R^0 = B - AX^0$
$\bar{R}^0, \sigma^0 = \mathrm{Norm}_{\mathbb{S}}\left(R^0\right)$
$P^0 = M^{-1}\bar{R}^0$
$\rho^0 = \langle P^0, \bar{R}^0 \rangle_{\mathbb{S}}$
for $k = 0, \ldots$ until convergence **do**
  $Q^k = AP^k$
  $\alpha^k = \langle P^k, Q^k \rangle_{\mathbb{S}}$
  $\lambda^k = \left(\alpha^k\right)^{-1} \rho^k$
  $X^{k+1} = X^k + P^k \lambda^k \sigma^k$
  $\tilde{R}^{k+1} = \bar{R}^k - Q^k \lambda^k$
  $\bar{R}^{k+1}, \gamma^{k+1} = \mathrm{Norm}_{\mathbb{S}}\left(\tilde{R}^{k+1}\right)$
  $\sigma^{k+1} = \gamma^{k+1} \sigma^k$
  break if $\|\sigma^{k+1}\| < \varepsilon_{\mathrm{tol}}$
  $Z^{k+1} = M^{-1}\bar{R}^{k+1}$
  $\rho^{k+1} = \langle Z^{k+1}, \bar{R}^{k+1} \rangle_{\mathbb{S}}$
  $\beta^k = \left(\rho^k\right)^{-1} {\gamma^{k+1}}^{\mathsf{T}} \rho^{k+1}$
  $P^{k+1} = Z^{k+1} + P^k \beta^k$
end for

This transformation is then also used to update the solution

$$X^k = X^{k-1} + P^{k-1} \lambda^{k-1} \sigma^{k-1},$$

because the search direction P^{k-1} is obtained from the transformed residual $\bar{R}^{k-1}$. For the same reason, we must consider the normalizer in the orthogonalization coefficient β^k, because P^k and P^{k-1} are transformed with respect to σ^k and σ^{k-1}, respectively. Thus, we have

$$\beta^k = \left(\rho^{k-1}\right)^{-1} {\gamma^k}^{\mathsf{T}} \rho^k.$$

The resulting algorithm is shown in Algorithm 4.2. Note that it is not necessary to compute the real residual for checking the convergence criterion, as we have

$$\|R_i^k\| = \|Q^k \sigma_i^k\| = \|\sigma_i^k\|,$$

where σ_i^k denotes the ith column of σ^k.

As the orthonormalization is expensive, it makes sense to skip it in iterations in which it is not necessary. To specify a criterion for the adaptive orthonormalization, we define the diagonally scaled condition number.

Definition 4.7 (Diagonally scaled condition number)**.** *For a symmetric matrix $\alpha \in \mathbb{R}^{k\times k}$, we define the diagonally scaled condition number $\kappa_D(\alpha)$ as*

$$\kappa_D(\alpha) = \kappa(\delta^{-\frac{1}{2}}\alpha\delta^{-\frac{1}{2}}),$$

where $\delta = \operatorname{diag}(\alpha)$ is the diagonal of α and κ denotes the condition number.

In contrast to the condition number, the diagonally scaled condition number equals 1 for diagonal matrices. This is desirable in particular in the parallel case. As it is only a parallel version of the scalar CG, no re-orthonormalization is necessary. Using the usual condition number, in this case, could deliver high numbers if the columns are scaled differently and would lead to superfluous re-orthonormalizations. We use the diagonally scaled condition number of α^k for an indicator of the numerical rank deficiency of the residual. To check this, we evaluate

$$\eta\kappa_D(\alpha^k) > \sqrt{\varepsilon_{\text{mach}}}, \tag{4.13}$$

where η is a tuning parameter and $\varepsilon_{\text{mach}}$ is the machine precision of the used numerical type. Algorithm 4.3 shows the resulting algorithm.

This approach assumes that the diagonally scaled condition number increases continuously with the iterations. This is not clear though. However, numerical tests show that this approach works quite well. Nevertheless, there is some mathematical background missing. An alternative approach would be to roll-back one iteration if the re-orthonormalization criterion (4.13) is satisfied. This would increase the memory requirements by one block vector and leads to some overhead as some computations must be redone.

Theoretically, the normalization could add artificial directions to the orthogonal residual, if the residual is rank-deficient. In the case where this direction is already contained in the Krylov space, this has no effect, because it is orthogonal to the residual. Otherwise, this would accelerate the convergence as it enhances the Krylov subspace. Whether these new directions could be chosen more cleverly and the comparison with deflation strategies is an objective of future work.

4.4 Numerical Experiments

As a first test series, we executed test runs to approve the convergence theory developed in Section 4.2. In Figure 4.1, the convergence behavior for different block sizes p and the block-global and block-parallel case is shown. As operator we used the `thermal2` matrix from the SuiteSparse Matrix Collection [Law+02]. This matrix provides a realistic size for tests on one machine and we observed that the AMG preconditioner from the DUNE framework worked sufficiently good. For the tests we used $s = 256$ randomly generated right-hand sides.

Algorithm 4.3: BCG Method with Adaptive Residual Re-Orthonormalization

$R^0 = B - AX^0$
if $\eta > 0$ **then**
  $\bar{R}^0, \sigma^0 = \mathrm{Norm}_{\mathbb{S}}\left(R^0\right)$
else
  $\bar{R}^0 = R^0$
  $\sigma^0 = \mathbb{I}_{\mathbb{S}}$
end if
$P^0 = M^{-1}\bar{R}^0$
$\rho^0 = \langle P^0, \bar{R}^0 \rangle_{\mathbb{S}}$
for $k = 0, \ldots$ until convergence **do**
  $Q^k = AP^k$
  $\alpha^k = \langle P^k, Q^k \rangle_{\mathbb{S}}$
  $\lambda^k = \left(\alpha^k\right)^{-1} \rho^k$
  $X^{k+1} = X^k + P^k \lambda^k \sigma^k$
  $\widetilde{R}^{k+1} = \bar{R}^k - Q^k \lambda^k$
  if $\eta \kappa_D(\alpha^k) > \sqrt{\varepsilon_{\mathrm{mach}}}$ **then**
    $\bar{R}^{k+1}, \gamma^{k+1} = \mathrm{Norm}_{\mathbb{S}}\left(\widetilde{R}^{k+1}\right)$
    $\sigma^{k+1} = \gamma^{k+1}\sigma^k$
    break if $\|\sigma^{k+1}\| \leq \varepsilon_{\mathrm{tol}}$
  else
    $\bar{R}^{k+1} = \widetilde{R}^{k+1}$
    $\gamma^{k+1} = \mathbb{I}_{\mathbb{S}}$
    $\sigma^{k+1} = \sigma^k$
    break if $\|\bar{R}^{k+1}\sigma^{k+1}\| \leq \varepsilon_{\mathrm{tol}}$
  end if
  $Z^{k+1} = M^{-1}\bar{R}^{k+1}$
  $\rho^{k+1} = \langle Z^{k+1}, \bar{R}^{k+1} \rangle_{\mathbb{S}}$
  $\beta^k = \left(\rho^k\right)^{-1} {\gamma^{k+1}}^{\mathsf{T}} \rho^{k+1}$
  $P^{k+1} = Z^{k+1} + P^k \beta^k$
end for

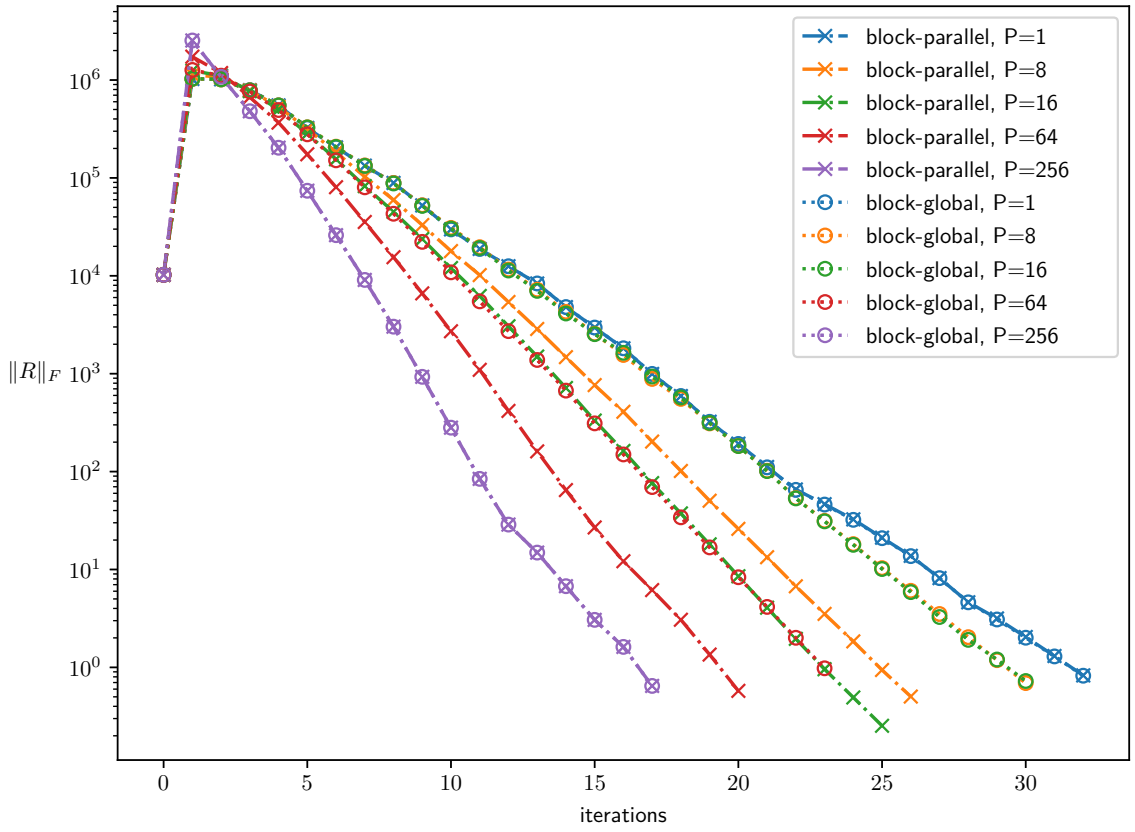


Figure 4.1: Frobenius norm of the residual vs. iterations of BCG methods. Different choices for the *-subalgebra $\mathbb{S}$ are taken into account. Dashed lines with crosses decode the block-parallel method. Dotted lines with circles decode the block-global method. The colors decode the parameter p.

The results confirm perfectly the theoretical expectations. In particular, we obtain a faster convergence (per iteration) for larger block size p. Furthermore, we see that the convergence rates in the block-global method $\mathbb{S}_{BG}^{p}$ with p up to 8 are similar to the convergence rate of the parallel method $\mathbb{S}_P$. This fits to the outcome of Lemma 4.6, as it predicted that the convergence rate depends on the $\left\lceil \frac{p}{q} \right\rceil$ smallest eigenvalue. For $p \leq 8$, we have $\left\lceil \frac{p}{q} \right\rceil = 1$. For the same reason, we see a connection of the convergence rate of the block-global method with $p = 64$, i.e. $\mathbb{S}_{BG}^{64}$ and block-parallel method with $p = 16$, i.e. $\mathbb{S}_{BP}^{16}$.

Note that the increase in the first iteration is due to the fact that the Frobenius norm of the residual is plotted, instead of the energy Frobenius norm of the error, which converges monotonically.

Further, we tested the methods behavior depending on the re-orthonormalization parameter η. Table 4.1 shows iteration and orthonormalization counts for different choices of η and different floating-point precision. The `HB/1138_bus` matrix from the SuiteSparse Matrix Collection [DH11] with a SSOR preconditioner was used with 256 randomly generated right-hand sides. The parallel method ($\mathbb{S}_P$) is shown for comparison.

Table 4.1: Iteration counts and numbers of residual re-orthonormalization for single and double precision. Blank cells indicate that the method did not converged within 1000 iterations. The parallel case is added for comparison. In all other rows the block-parallel method with $p = 64$ was used.

precision	single		double	
η	iterations	ortho.	iterations	ortho.
parallel	868	0	514	0
0				
0.1	434	10	146	3
1	344	14	137	5
10	286	30	137	5
100	229	55	104	3
1000	231	230	88	3
∞	237	238	74	75

In all other rows the block-parallel with $p = 64$ ($\mathbb{S}_{BP}^{64}$) is used. It can be seen that the number of iterations decreases if we use a higher re-orthonormalization parameter η. It can also be seen that a higher η not necessarily leads to more re-orthonormalizations. Hence, the moment of the re-orthonormalization seems to be important, a fact that can also be observed in the next experiment.

Figure 4.2 shows the convergence history of the BCG algorithm in the same setting used in Figure 4.1, for different re-orthonormalization parameters η. We used the block-parallel setting with $p = 64$. We see that the re-orthonormalization is necessary to achieve convergence and the re-orthonormalization must be reapplied during the iteration - it is not sufficient to orthonormalize the initial residual. Furthermore, we see that choosing $\eta = 1$ yields a converging method, but does not ensure optimal convergence rates. Only in cases where in iteration 0 and 1 a orthonormalization was applied the convergence rate was optimal. The additional re-orthonormalization in iteration 18 for $\eta = 10\,000$ seems to be superfluous.

Table 4.2 shows the iteration counts, the number of re-orthonormalizations and the runtime of the BCG method for the block-parallel method and different value of p for several symmetric positive definite matrices of the SuiteSparse Matrix Collection [Law+02]. We used an incomplete Cholesky preconditioner and the re-orthonormalization parameter $\eta = 10\,000$. All systems are solved for $s = 256$ randomly generated right-hand sides. The fastest runtime per matrix is marked with a green background. Missing numbers mark that no convergence was achieved within 1000 iterations. We aimed for a reduction of the residual in every column by a factor of 10^{-4}.

The result shows that the number of iterations can be reduced drastically by using a higher p. We suppose that this is due to the weak preconditioner, that mainly smooths the larger eigenvalues. For example for the `bcsstk15` matrix, the number of iterations

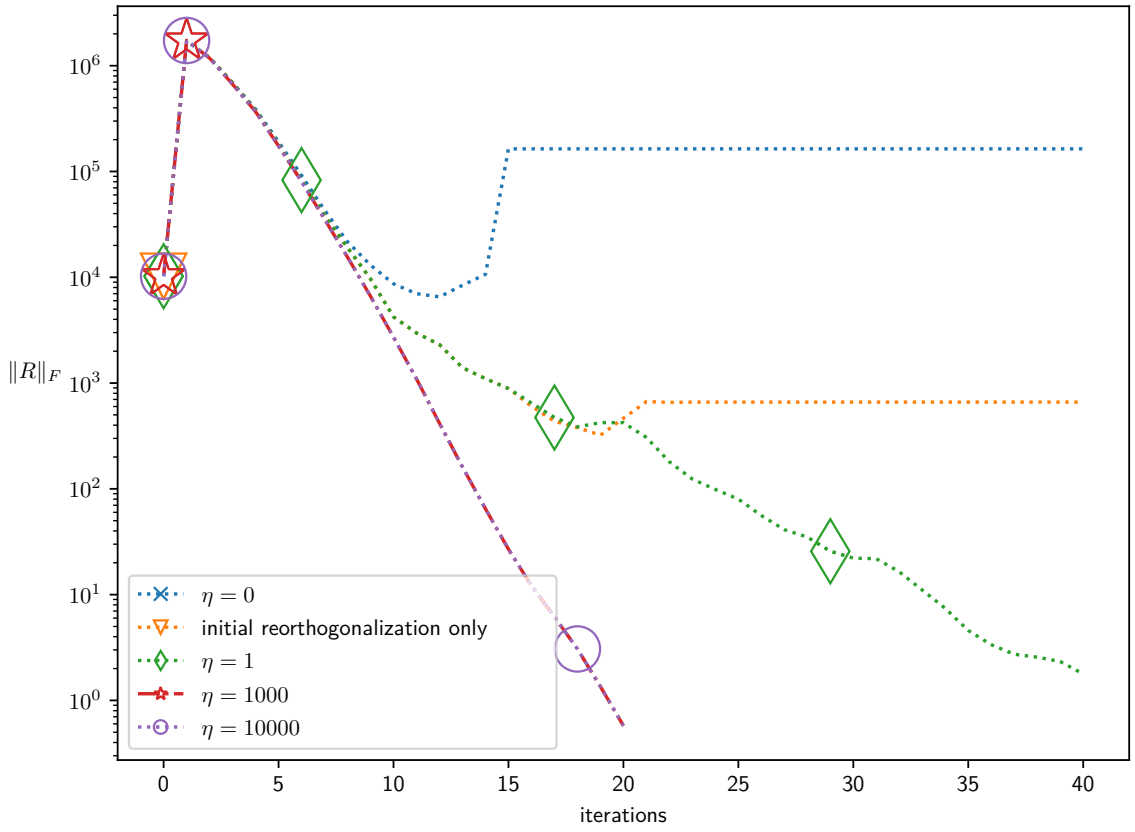


Figure 4.2: Convergence of the BCG method for different re-orthonormalization parameters. The symbols mark the iteration in which a re-orthonormalization happened. Colors decode the different values of the re-orthonormalization parameter η.

was reduced by a factor of ~ 25 by using $p = 32$ compared to the parallel case. In some cases the block Krylov methods help to achieve convergence at all, e.g. in the `bcsstk17` case, where the parallel case fails to converge within 1000 iterations, but the block methods converge within 73 iterations for the $p = 32$ case.

For the `bcsstk16` matrix the parallel method is fastest, although it need the most iterations. This is due to the re-orthonormalization costs, as the other building blocks are similar expensive for the $p = 32$ case. An improvement of the re-orthonormalization criteria is one of the goals of future work.

Table 4.2: Iteration counts, runtime and number of re-orthonormalizations for the solution of several matrices from MatrixMarket with different p. The fastest time per row is marked with a green background. Blank cells indicate that the method did not converge within 1000 iterations.

	$p = 1$			$p = 32$			$p = 256$		
Matrix	#it	#ro	t	#it	#ro	t	#it	#ro	t
bcsstk14	251	0	3.18	16	13	1.15	7	5	1.79
bcsstk15	573	0	12.24	23	12	1.97	10	5	2.41
bcsstk16	26	0	1.02	9	1	1.20	6	2	1.64
bcsstk17				73	71	14.54	16	15	8.09
bcsstk18	419	0	21.89	49	5	6.59	15	4	6.23
s1rmq4m1	85	0	3.36	16	2	1.90	9	2	2.81
s1rmt3m1	166	0	5.75	24	3	3.09	12	2	3.36
s2rmq4m1	107	0	4.24	17	3	2.02	11	2	3.33
s2rmt3m1	226	0	7.78	30	3	3.86	14	2	4.02
s3dkq4m2				217	8	190.09	65	9	138.10
s3rmq4m1	197	0	7.82	21	3	2.84	12	3	3.71
s3rmt3m1	478	0	16.80	33	5	3.03	17	5	5.13
s3rmt3m3	442	0	14.40	33	5	3.69	15	4	3.98

5
Block GMRes Method

In the previous chapter we looked at the block Conjugate Gradients method, which is the method of choice for symmetric positive definite problems. We now discuss the block GMRes (BGMRes) method [Vit90], which is a block version of the GMRes method by Saad and Schultz [SS86]. In contrast to the BCG method, it does not have any requirements on the operator. As a downside, the GMRes method can not make use of a short recurrence, i.e. the memory and arithmetically costs per iteration increase with every iteration. Nevertheless, it is one of the most important Krylov methods in practice. Note that for symmetric indefinite problems there are also block versions of the MinRes [Soo15] and conjugate residual [ZZ13] method, which are not subject of this work. Good introductions into the classical block GMRes method can be found in the monographs of Gutknecht [Gut07] and Saad [Saa03].

We will formulate the BGMRes method based on the block Krylov framework. Recently, Kubínová and Soodhalter [KS20] presented a paper that also discusses the BGMRes method in this framework and contains some results about the convergence of the method. In addition, a generalization of the Givens rotations that are used in the non-block case to triangulate the Hessenberg matrix is described. We pick up this generalization in the first section and formulate the BGMRes method. We present some simple convergence results in the second section and refer the reader to [KS20] for a more elaborate discussion. Finally, we present some numerical experiments that give further insights into the convergence behavior of the BGMRes method for different *-subalgebras.

5.1 Formulation of the Block GMRes Method

The BGMRes method is based on the block Arnoldi process [Arn51; Ruh79], which computes an orthonormal basis of the Krylov space and was originally invented to

Algorithm 5.1: Block Arnoldi Method

Let $V^0 \in \mathbb{R}^{n\times s}$ with $\langle V^0, V^0\rangle_{\mathbb{S}} = 0$ be given.
for $k = 1, \dots k_{\max}$ **do**
  $V^k = AV^{k-1}$
  for $j = 0, \dots, k-1$ **do**
    $\eta_{j,k-1} = -\langle V^j, V^k\rangle_{\mathbb{S}}$
    $V^k \leftarrow V^k + V^j\eta_{j,k-1}$
  end for
  $V^k, \eta_{k,k-1} \leftarrow \mathrm{Norm}_{\mathbb{S}}\left(V^k\right)$
end for

compute the eigenvalues of an operator. It is based on the block Gram-Schmidt orthogonalization process, see Algorithm 5.1.

The coefficients η build a block matrix $\mathcal{H} \in \mathbb{S}^{k+1\times k}$, that has block Hessenberg form, i.e. all blocks below the first off-diagonal under the diagonal are zero. The resulting basis $\mathcal{V} = \left[V^0, \dots, V^{k_{\max}}\right]$ satisfies the so called *block Arnoldi relation*

$$A\widetilde{\mathcal{V}} = \mathcal{V}\mathcal{H},$$

where $\widetilde{\mathcal{V}} = \left[V^0, \dots, V^{k_{\max}-1}\right]$.

As the normalizer is also defined for rank-deficient block vectors, we do not get a breakdown in the case where V^k is rank-deficient after the Gram-Schmidt orthonormalization. The normalization process adds additional directions to the Krylov space in this case. Theoretically the orthogonalization must be repeated to orthogonalize the additional directions to the previous block vectors. However, numerical experiments show, that this is not necessary, even if the rigorous analysis of this effect is still missing.

Algorithm 5.1 uses the modified Gram-Schmidt procedure, meaning the computation of the block inner products and the vector updates are interleaved. The modified Gram-Schmidt procedure is more stable than the classical Gram-Schmidt procedure which computes all block inner products in advance. However, as the block inner product computes multiple inner products simultaneously the stability could be affected. This could be mitigated by either using the "real" modified Gram-Schmidt that considers the columns of the block vectors individually or by doing a re-orthogonalization like presented by Björck [Bjö94]. Buhr et al. [Buh+14, Algorithm 1] presented an adaptive re-orthogonalization strategy for the Gram-Schmidt procedure, that could be applied to decide adaptively whether a re-orthonormalization is necessary.

The goal of the BGMRes method is to compute an update $U^k \in \mathcal{K}^k_{\mathbb{S}}(A, R^0)$ for the initial guess X^0 that solves the minimization problem

$$U^k = \underset{Y\in\mathcal{K}^k_{\mathbb{S}}(A,R^0)}{\arg\min} \|B - AX^0 - AY\|_F. \tag{5.1}$$

In other words, the Frobenius norm of the block residual is minimized. With help of the Arnoldi basis the update reads

$$U^k = \widetilde{\mathcal{V}}\boldsymbol{\zeta}^k,$$

where $\boldsymbol{\zeta}^k \in \mathbb{S}^k$ denote the coefficients of U^k in the basis $\widetilde{\mathcal{V}}$. Using the block Arnoldi relation, the orthonormality of $\mathcal{V}$ and a block QR decomposition $\mathcal{Q}\mathcal{R} = \mathcal{H}$, with $\mathcal{Q} \in \mathbb{S}^{k+1\times k}$ and $\mathcal{R} \in \mathbb{S}^{k\times k}$ we rewrite the minimization problem (5.1) as

$$\begin{aligned}\|B - AX^0 - AU^k\|_F &= \|R^0 - A\widetilde{\mathcal{V}}\boldsymbol{\zeta}^k\|_F \\ &= \|R^0 - \mathcal{V}\mathcal{H}\boldsymbol{\zeta}^k\|_F \\ &= \|\mathcal{V}^\mathsf{T}R^0 - \mathcal{H}\boldsymbol{\zeta}^k\|_F \\ &= \|\mathcal{Q}^\mathsf{T}\mathcal{V}^\mathsf{T}R^0 - \mathcal{R}\boldsymbol{\zeta}^k\|_F.\end{aligned}$$

This minimization problem can then be solved for $\boldsymbol{\zeta}^k$ by block-wise backward-substitution.

In the non-block case the QR decomposition of $\mathcal{H}$ is computed using Given rotations to eliminate the lower off-diagonal entries. This can be generalized for our block Krylov framework. For that, we compute a full QR decomposition of the diagonal and lower off-diagonal entry, starting with

$$Q_0\begin{pmatrix}\rho_0\\0\end{pmatrix} = \begin{pmatrix}\eta_{0,0}\\\eta_{1,0}\end{pmatrix} \qquad Q_0{}^\mathsf{T}Q_0 = \begin{pmatrix}\mathbb{I} & 0\\0 & \mathbb{I}\end{pmatrix} \qquad Q_0 \in \mathbb{S}^{2\times 2}, \rho_0 \in \mathbb{S}.$$

The lower off-diagonal element can then be eliminated by

$$\begin{pmatrix}Q_0{}^\mathsf{T} & & \\ & \mathbb{I} & \\ & & \ddots \\ & & & \mathbb{I}\end{pmatrix}\begin{pmatrix}\eta_{0,0} & & & *\\ \eta_{1,0} & \eta_{1,1} & & \\ & \eta_{2,1} & \ddots & \\ & & \ddots & \end{pmatrix} = \begin{pmatrix}\rho_0 & & & *\\ 0 & & & \\ & \eta_{2,1} & \ddots & \\ & & \ddots & \end{pmatrix}.$$

The star indicates non-zero entries in the upper triangle. This procedure is repeated to eliminate the other lower off-diagonal entries. The Q-factor of the QR decomposition

of $\mathcal{H}$ is then build by concatenating all the Q-factors of the smaller QR decompositions

$$\mathcal{Q} = \begin{pmatrix} Q_0{}^{\mathsf{T}} & & & & \\ & \mathbb{I} & & & \\ & & \mathbb{I} & & \\ & & & \ddots & \\ & & & & \mathbb{I} \end{pmatrix} \begin{pmatrix} \mathbb{I} & & & & \\ & Q_1{}^{\mathsf{T}} & & & \\ & & \mathbb{I} & & \\ & & & \ddots & \\ & & & & \mathbb{I} \end{pmatrix} \cdots$$

In the algorithm the transformation of R^0 and the QR decomposition of $\mathcal{H}$ is performed on-the-fly. The vector

$$\boldsymbol{\sigma} = \begin{pmatrix} \sigma^0 \\ \vdots \\ \sigma^{k_{\max}} \end{pmatrix} = \mathcal{Q}^{\mathsf{T}} \mathcal{V}^{\mathsf{T}} R^0$$

is updated during the iteration by

$$\begin{aligned} \sigma^0 &= \mathrm{Norm}_{\mathbb{S}} \left(M^{-1} R^0 \right), \\ \begin{pmatrix} \sigma^k \\ \sigma^{k+1} \end{pmatrix} &\leftarrow Q_i{}^{\mathsf{T}} \begin{pmatrix} \sigma^k \\ 0 \end{pmatrix}. \end{aligned}$$

The Frobenius norm of σ^{k+1} can be used to determine the residual in the k-iteration, as

$$\begin{aligned} \|R^k\|_F &= \|R^0 - A\widetilde{\mathcal{V}}\boldsymbol{\zeta}^k\|_F \\ &= \|R^0 - \mathcal{V}\mathcal{Q}\mathcal{R}\boldsymbol{\zeta}^k\|_F \\ &= \|\mathcal{Q}^{\mathsf{T}}\mathcal{V}^{\mathsf{T}}R^0 - \mathcal{R}\boldsymbol{\zeta}^k\|_F \\ &= \|\boldsymbol{\sigma} - R\boldsymbol{\zeta}^k\|_F = \|\sigma^{k+1}\|_F. \end{aligned}$$

Algorithm 5.2 shows the BGMRes algorithm. Preconditioning can be easily implemented by adapting lines 2 and 4.

5.2 Convergence

For the GMRes method we do not have a general theoretical statement about the convergence rate, like for the CG case. Rather Greenbaum, Pták and Strakoš [GPS96] showed that any non-increasing convergence curve for the GMRes method is possible. This result was recently generalized by Kubínová and Soodhalter [KS20] for the BGMRes method. A convergence theory for the BGMRes method for special classes of operators was presented by Simoncini and Gallopoulos [SG96].

As the BGMRes method minimizes the Frobenius norm of the residual in the block

Algorithm 5.2: Block GMRes Method

$R^0 = B - AX^0$
$V^0\sigma^0 = \mathrm{Norm}_{\mathbb{S}}\left(R^0\right)$
for $k = 0, \ldots, k_{\max} - 1$ **do**
    $V^{k+1} = AV^k$
    for $j = 0, \ldots, k$ **do**
        $\eta_{j,k} = -\langle V^j, V^{k+1}\rangle_{\mathbb{S}}$
        $V^{k+1} \leftarrow V^{k+1} + V^j\eta_{j,k}$
    end for
    $V^{k+1}, \gamma \leftarrow \mathrm{Norm}_{\mathbb{S}}\left(V^{k+1}\right)$
    for $j = 0, \ldots, k-1$ **do**
        $\begin{pmatrix} \eta_{j,k} \\ \eta_{j+1,k} \end{pmatrix} \leftarrow {Q_j}^{\mathsf{T}} \begin{pmatrix} \eta_{j,k} \\ \eta_{j+1,k} \end{pmatrix}$
    end for
    $Q_k \begin{pmatrix} \eta_{k,k} \\ 0 \end{pmatrix} \leftarrow \begin{pmatrix} \eta_{k,k} \\ \gamma \end{pmatrix}$ ▷ Compute QR decompostion
    $\begin{pmatrix} \sigma^k \\ \sigma^{k+1} \end{pmatrix} \leftarrow {Q_k}^{\mathsf{T}} \begin{pmatrix} \sigma^k \\ 0 \end{pmatrix}$
    break if $\|\sigma^{k+1}\| \leq \varepsilon_{\mathrm{tol}}$
end for
for $l = k_{\max} - 1, \ldots, 0$ **do** ▷ back-substitution
    $\sigma^l \leftarrow \sigma^l - \sum_{j=l+1}^{k_{\max}-1} \eta_{l,j}\sigma^j$
    $\sigma^l \leftarrow (\eta_{l,l})^{-1}\sigma^l$
end for
$X^{k_{\max}} = X^0 + \sum_{j=0}^{k_{\max}-1} V^j\sigma^j$
if $\|\sigma^{k_{\max}}\| > \varepsilon_{\mathrm{tol}}$ **then**
    restart with $X^0 \leftarrow X^{k_{\max}}$
end if

Krylov space we have

$$\|R^{k+1}\|_F \leq \|R^k\|_F.$$

This ensures a monotonic convergence of the Frobenius norm of the residual, but it cannot be ensured that the residual actually decreases, see Example 2.3.

To deduce better estimations a-priori knowledge about the operator is necessary. We use again the polynomial representation to formulate an abstract statement about the convergence rate, similar to the convergence proof of the BCG method. This statement could be used to deduce concrete estimations if further assumptions on the operator are made.

Lemma 5.1 (Abstract convergence of BGMRes method)**.** *Let $\mathbb{S}$ be a *-subalgebra of $\mathbb{R}^{s\times s}$. For the residual R^k of the kth step in the BGMRes method the following estimation holds*

$$\|R^k\|_F \leq \inf_{\mathcal{Q}^k} \|\mathcal{Q}^k(A) \circ R^0\|_F \leq \inf_{\mathcal{Q}^k} \|\mathcal{Q}^k(A)\| \|R^0\|_F.$$

The infimum is taken over all $\mathbb{S}$-valued polynomials $\mathcal{Q}^k \in \mathbb{P}^k_{\mathbb{S}}$ with absolute coefficient $\mathbb{I}$. The norm $\|\mathcal{Q}^k(A)\|$ denotes the operator norm of $\mathcal{Q}^k(A)$ in the space $\mathcal{L}(\mathbb{R}^{sn}, \mathbb{R}^{sn})$.

Proof. The BGMRes method computes the best approximation in the space $X^0 + \mathcal{K}^k_{\mathbb{S}}(A, R^0)$. As we can represent the elements in the Krylov space with $\mathbb{S}$-valued polynomials we obtain

$$\begin{aligned} R^k &= B - AX^k \\ &= B - AX^0 - AU^k \\ &= R^0 - A\mathcal{P}^{k-1}(A) \circ R^0 \\ &= \left(\mathbb{I} - A\mathcal{P}^{k-1}(A)\right) \circ R^0 \\ &= \mathcal{Q}^k(A) \circ R^0, \end{aligned} \tag{5.2}$$

where $\mathcal{Q}^k(\mathrm{x}) = \mathbb{I} - \mathrm{x}\mathcal{P}^k(\mathrm{x})$. Applying the Frobenius norm and taking the infimum completes the proof, as the Frobenius norm on $\mathbb{R}^{n\times s}$ and the Euclidean norm on the space $\mathbb{R}^{ns}$ coincide. □

The next lemma gives an example how this estimation could be used to create more concrete estimations. This is a generalization for the block Krylov framework of Theorem 3.1 in the work of Simoncini and Gallopoulos [SG96, Theorem 3.1].

Lemma 5.2. *If the operator A is diagonalizable*

$$A = V\Lambda V^{-1}, \qquad \text{with} \qquad \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$$

estimation (5.2) *can be precised as*

$$\|R^k\|_F \leq \kappa(V) \inf_{c_1 \ldots, c_k \in \mathbb{S}} \max_{i=1}^{n} \left\| \sum_{j=0}^{k} \lambda_i^j c_j \right\| \|R^0\|_F,$$

where $c_0 = \mathbb{I}$.

Proof. Let $c_1, \ldots, c_k \in \mathbb{S}$ denote the coefficients of the $\mathbb{S}$-valued polynomial $\mathcal{Q}^k$. Then

we write

$$\begin{aligned}\left\|\mathcal{Q}^k(A)\right\| &= \left\|\sum_{j=0}^{k} A^j \otimes c_i\right\| \\ &= \left\|\sum_{j=0}^{k} V\Lambda^j V^{-1} \otimes c_i\right\| \\ &= \left\|(V \otimes \mathbb{I})\left(\sum_{j=0}^{k} \Lambda^j \otimes c_i\right)\left(V^{-1} \otimes \mathbb{I}\right)\right\| \\ &\leq \kappa(V) \max_{i=1}^{n} \left\|\sum_{j=0}^{k} \lambda_i^j c_j\right\|\end{aligned}$$

□

The challenge is to choose good coefficients $c_1, \ldots, c_k \in \mathbb{S}$. If all λ^i are positive, then probably the scaled Chebyshev polynomials would yield an estimation similar to the CG case. See the recent paper of Kubínová and Soodhalter[KS20] for a detailed discussion. Further convergence results of the classical BGMRes method can be found in the paper of Simoncini and Gallopoulos [SG96].

5.3 Numerical Experiments

As the theoretical convergence results are still quite vague yet, we rely on numerical test to get an impression of the convergence behavior of the method with respect to the different *-subalgebras. As the BGMRes method minimizes the Frobenius norm of the residual we know that for the same p the block-parallel method converges faster than the block-global method. In both cases the larger the p the better the convergence rate (per iteration), cf. Lemma 3.12 and Lemma 3.4.

It is confirmed by the result presented in Figure 5.1. It shows the convergence of the BGMRes method for the `Simon/raefsky3` matrix from the SuiteSparse Matrix Collection [DH11]. The problem consists of 21 200 unknowns and originates from a computational fluid dynamics problem. We use an ILU(0) preconditioner and solve for $s = 256$ randomly generated right-hand sides until a reduction of the 2-norm of the residual for every column by a factor of 10^{-4} is reached.

We see a relation of the convergence rates of the p-block-global method and the $\frac{p}{q}$-block-parallel method, like in the BCG case. For example the convergence for $\mathbb{S}_{BG}^{128}$ ($p = 128$ block-global) and $\mathbb{S}_{BP}^{64}$ ($p = 64$ block-parallel) is almost identical. The same holds for $\mathbb{S}_{BG}^{64}$ ($p = 64$ block-global) and $\mathbb{S}_{BP}^{16}$ ($p = 16$ block-parallel). That indicates that a similar result to Lemma 4.6 could also be possible for the BGMRes method.

Note that choosing a large restart parameter in the BGMRes method is crucial for achieving good convergence. Often the choice of that parameter is limited by the memory of the machine. This means the restart length directly competes with the

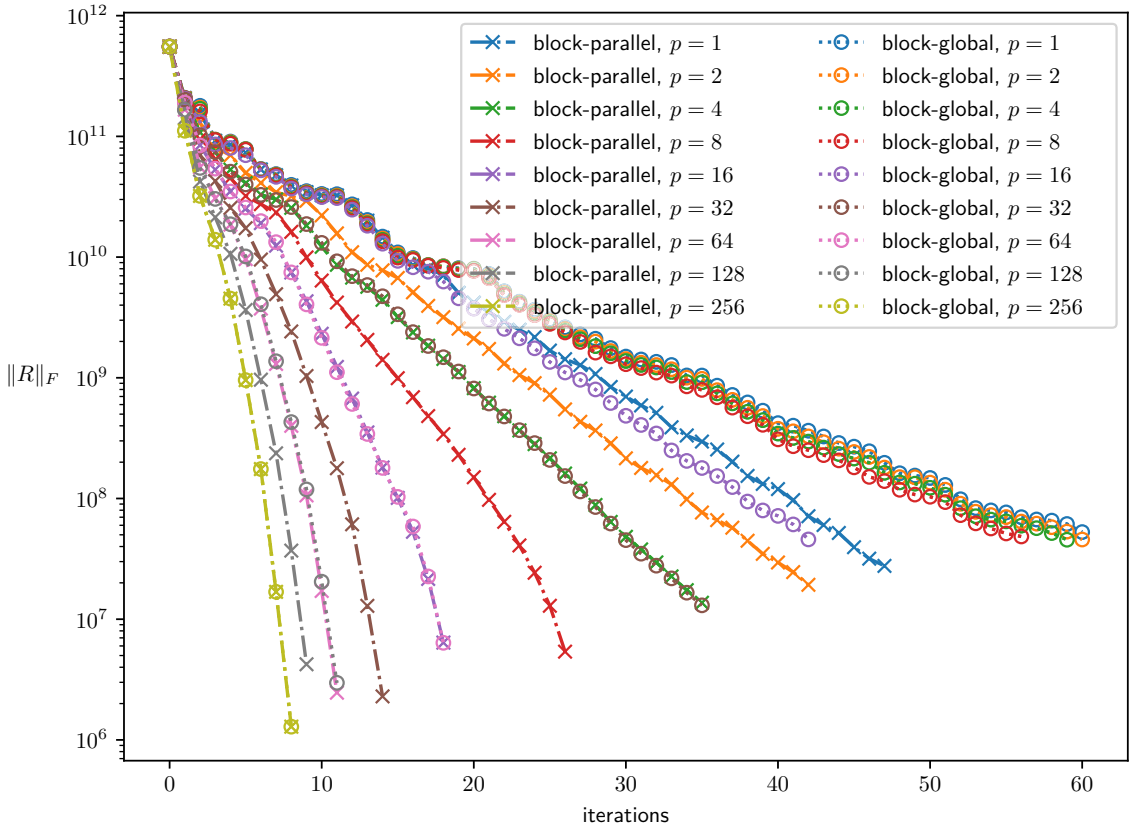


Figure 5.1: Convergence of block GMRes method. Dashed lines with crosses decode the block-parallel methods. Dotted lines with circles denote the block-global method. Colors decode the blocking parameter p.

number of right-hand sides that can be used. If this is an issue, probably the block BiCGStab method which is considered in the next chapter is a better choice to solve the problem, as its memory requirements are constant.

6
Block BiCGStab Method

As a third block Krylov method we look at the Block BiCGStab (BBiCGStab) method, a block version of the BiCGStab method presented by Van der Vorst [Van92]. This chapter is based on the paper by El Guennouni, Jbilou and Sadok [EJS03]. They deduce the block BiCGStab method, i.e. the case $\mathbb{S} = \mathbb{R}^{s\times s}$. We adapt this deduction for the block Krylov framework.

Like the GMRes method, the BiCGStab method was developed for non-symmetric problems. Unlike the GMRes method, it does not store a basis and is therefore better suited for memory limited systems. This advantage comes with the price, that no minimization property is satisfied by the approximate solution. Hence, it is difficult to develop theoretical convergence results.

As with the other methods, we reformulate the BBiCGStab method based on the block Krylov framework in the first section. In the second section we introduce a stabilization for the BBiCGStab method, similar to that for the BCG method. Up to the authors knowledge such a stabilization strategy was not presented before. Finally, we present some numerical results in section 6.3.

6.1 Formulation of the Block BiCGStab Method

Like the name suggests, the BBiCGStab method is based on the block BiCG (BBiCG) method [OLe80], but adds a stabilization step to mitigate instabilities. Another issue of the BBiCG method is that the transposed of the operator must be applied to a block vector. The BBiCG method actually solves an additional linear system with the transposed operator and computes bases $(P^i)_i$ and $(\widetilde{P}^i)_i$ of the Krylov spaces $\mathcal{K}^k\left(M^{-1}A, M^{-1}R^0\right)$ and $\mathcal{K}^k\left(M^{-\mathsf{T}}A^{\mathsf{T}}, M^{-\mathsf{T}}\widetilde{R}^0\right)$, for some block vector $\widetilde{R}^0 \in \mathbb{R}^{n\times s}$ and preconditioner $M \in \mathbb{R}^{n\times n}$. The residuals of the linear systems are projected onto the Krylov space of the other system. In the absence of rounding errors, this ensures that

the method converges in at least n iterations. However, it is not the aim to proceed as many iterations, as even a direct solve would be more efficient. The BBiCG algorithm is shown in Algorithm 6.1.

Algorithm 6.1: Block BiCG Method

$R^0 = B - AX^0$
Choose $\widetilde{R}^0$ arbitrarily with $\langle \widetilde{R}^0, R^0 \rangle_{\mathbb{S}} \neq 0$
$P^0 = M^{-1}R^0, \widetilde{P}^0 = M^{-\mathsf{T}}\widetilde{R}^0$
$\rho^0 = \langle \widetilde{R}^0, P^0 \rangle_{\mathbb{S}}$
for $k = 0, \ldots$ **do**
  $\alpha^k = \langle \widetilde{P}^k, AP^k \rangle_{\mathbb{S}}$
  $X^{k+1} = X^k + P^k (\alpha^k)^{-1} \rho^k$
  $R^{k+1} = R^k - AP^k (\alpha^k)^{-1} \rho^k$
  break if $\|R^{k+1}\| \leq \varepsilon_{\text{tol}}$
  $\widetilde{R}^{k+1} = \widetilde{R}^k - A^{\mathsf{T}} \widetilde{P}^k (\alpha^k)^{-\mathsf{T}} \rho^{k\mathsf{T}}$
  $Z^{k+1} = M^{-1}R^{k+1}, \widetilde{Z}^{k+1} = M^{-\mathsf{T}}\widetilde{R}^{k+1}$
  $\rho^{k+1} = \langle \widetilde{R}^{k+1}, Z^{k+1} \rangle_{\mathbb{S}}$
  $P^{k+1} = Z^{k+1} + P^k \left(\rho^k\right)^{-1} \rho^{k+1}$
  $\widetilde{P}^{k+1} = \widetilde{Z}^{k+1} + \widetilde{P}^k \left(\rho^k\right)^{-\mathsf{T}} \rho^{k+1\mathsf{T}}$
end for

Lemma 6.1. *The residual in the kth iteration is orthogonal to the Krylov space of the adjunct problem.*

$$\langle V, R^k \rangle_{\mathbb{S}} = 0 \qquad \forall V \in \mathcal{K}^k_{\mathbb{S}} \left(M^{-\mathsf{T}} A^{\mathsf{T}}, M^{-\mathsf{T}} \widetilde{R}^0 \right).$$

A proof can be found in the paper by O'Leary [OLe80, Lemma 1]. If the operator and preconditioner are symmetric the method is equivalent to the BCG method.

Next, we introduce some theory about orthogonality of $\mathbb{S}$-valued polynomials leading to some further properties of the BBiCG method. Based on these properties we define the BBiCGStab method, which uses enhanced polynomials that satisfy the same properties.

Lemma 6.2. *The variables in the BBiCG algorithm can be expressed in the form*

$$R^k = \mathcal{R}^k(AM^{-1}) \circ R^0$$

and

$$P^k = \mathcal{P}^k(M^{-1}A) \circ M^{-1}R^0,$$

where $\mathcal{R}, \mathcal{P} \in \mathbb{P}^k_{\mathbb{S}}$ are $\mathbb{S}$-valued polynomials of degree k defined by the recursion formulas

$$\mathcal{R}^0(\mathrm{x}) = \mathbb{I} \qquad \mathcal{R}^{k+1}(\mathrm{x}) = \mathcal{R}^k(\mathrm{x}) - \mathrm{x}\mathcal{P}^k(\mathrm{x})\lambda^k$$

and

$$\mathcal{P}^0(\mathrm{x}) = \mathbb{I} \qquad\qquad \mathcal{P}^{k+1}(\mathrm{x}) = \mathcal{R}^{k+1}(\mathrm{x}) + \mathcal{P}^k(\mathrm{x})\beta^k,$$

with $\lambda^k = \left(\alpha^k\right)^{-1}\rho^k$ *and* $\beta^k = \left(\rho^k\right)^{-1}\rho^{k+1}$.

Proof. We proof this by induction. The case $k = 0$ is trivial. Assume that the relations hold for k. Then we have

$$\begin{aligned} R^{k+1} &= R^k - AP^k\lambda^k \\ &= \mathcal{R}^k(AM^{-1}) \circ R^0 - \left(A\mathcal{P}^k(M^{-1}A) \circ M^{-1}R^0\right)\lambda^k \\ &= \mathcal{R}^k(AM^{-1}) \circ R^0 - \left(AM^{-1}\mathcal{P}^k(AM^{-1}) \circ R^0\right)\lambda^k \\ &= \left[\mathcal{R}^k - \mathrm{x}\mathcal{P}^k\lambda^k\right](AM^{-1}) \circ R^0. \end{aligned}$$

The second equation follows from the update formula of P

$$\begin{aligned} P^{k+1} &= M^{-1}R^{k+1} + P^k\beta^k \\ &= M^{-1}\mathcal{R}^{k+1}(AM^{-1}) \circ R^0 + \left(\mathcal{P}^k(M^{-1}A)\beta^k\right) \circ M^{-1}R^0 \\ &= \mathcal{R}^{k+1}(M^{-1}A) \circ M^{-1}R^0 + \left(\mathcal{P}^k(M^{-1}A)\beta^k\right) \circ M^{-1}R^0 \\ &= \left[\mathcal{R}^{k+1} + \mathcal{P}^k\beta^k\right](M^{-1}A) \circ M^{-1}R^0. \end{aligned}$$

□

Now we introduce some formality to describe orthogonal polynomials and show that the polynomials of the recursion formulas of the BBiCG method satisfy these orthogonality properties. That formalism helps us to get rid of the transposed operator that must be applied in the BBiCG method.

Definition 6.3 (Formally orthogonal polynomials)**.** *We define the* $\mathbb{S}$*-valued linear functionals* $\mathcal{C}$ *and* $\mathcal{C}^{(1)}$ *on* $\mathbb{P}_{\mathbb{S}}^k$ *for any* $\mathcal{P} \in \mathbb{P}_{\mathbb{S}}^k$ *as*

$$\begin{aligned} \mathcal{C}(\mathcal{P}) &:= \langle M^{-\mathsf{T}}\tilde{R}^0, \mathcal{P}(AM^{-1}) \circ R^0\rangle_{\mathbb{S}} \\ \mathcal{C}^{(1)}(\mathcal{P}) &:= \mathcal{C}(\mathrm{x}\mathcal{P}). \end{aligned}$$

Lemma 6.4. *For the* $\mathbb{S}$*-valued polynomials* $\mathcal{R}^k$ *and* $\mathcal{P}^k$ *from Lemma 6.2 we have*

$$\mathcal{C}(\mathcal{R}^k\mathcal{T}) = 0 \tag{6.1}$$

and

$$\mathcal{C}^{(1)}(\mathcal{P}^k\mathcal{T}) = 0 \tag{6.2}$$

for any $\mathcal{T} \in \mathbb{P}_{\mathbb{S}}^{k-1}$.

Proof. From Lemma 6.1 we know that for all $i = 0, \ldots, k-1$ it holds that

$$\langle (M^{-\mathsf{T}} A^{\mathsf{T}})^i M^{-\mathsf{T}} \widetilde{R}^0, R^k \rangle_{\mathbb{S}} = 0.$$

It follows that

$$\langle M^{-\mathsf{T}} \widetilde{R}^0, (AM^{-1})^i \mathcal{R}^k (AM^{-1}) \circ R^0 \rangle_{\mathbb{S}} = 0,$$

hence

$$\mathcal{C}(\mathrm{x}^i \mathcal{R}^k) = 0.$$

Equation (6.1) follows from the linearity of $\mathcal{C}$.

Similarly, we proof Equation (6.2). By using

$$AP^k = \left(R^k - R^{k+1}\right) \left(\lambda^k\right)^{-1}$$

we get

$$\begin{aligned}
\mathcal{C}^{(1)}(\mathrm{x}^i \mathcal{P}) &= \mathcal{C}(\mathrm{x}^{i+1} \mathcal{P}) \\
&= \langle M^{-\mathsf{T}} \widetilde{R}^0, \left(AM^{-1}\right)^{i+1} \mathcal{P}^k (AM^{-1}) \circ R^0) \rangle_{\mathbb{S}} \\
&= \langle \left(M^{-\mathsf{T}} A^{\mathsf{T}}\right)^i M^{-\mathsf{T}} \widetilde{R}^0, AP^k \rangle_{\mathbb{S}} \\
&= \langle \left(M^{-\mathsf{T}} A^{\mathsf{T}}\right)^i M^{-\mathsf{T}} \widetilde{R}^0, \left(R^k - R^{k+1}\right) \left(\lambda^k\right)^{-1} \rangle_{\mathbb{S}} \\
&= 0,
\end{aligned}$$

where we used again Lemma 6.1 and the fact that $\left(M^{-\mathsf{T}} A^{\mathsf{T}}\right)^i M^{-\mathsf{T}} \widetilde{R}^0 \in \mathcal{K}^i_{\mathbb{S}} \left(M^{-\mathsf{T}} A^{\mathsf{T}}, M^{-\mathsf{T}} \widetilde{R}^0\right)$. Equation (6.2) follows by using the linearity of $\mathcal{C}^{(1)}$. □

For the stabilization we enhance the residual and search direction by a polynomial $\mathcal{Q}^k$ as

$$\begin{aligned}
R^k &= \left[\mathcal{R}^k \mathcal{Q}^k\right] (AM^{-1}) \circ R^0 \\
P^k &= \left[\mathcal{P}^k \mathcal{Q}^k\right] (M^{-1} A) \circ M^{-1} R^0,
\end{aligned}$$

where $\mathcal{Q}^k \in \mathbb{P}^k_{\mathbb{R}}$ is a scalar polynomial recursively defined by

$$\mathcal{Q}^0(\mathrm{x}) = 1 \qquad \text{and} \qquad \mathcal{Q}^{k+1}(\mathrm{x}) = (1 - \omega^k \mathrm{x}) \mathcal{Q}^k(\mathrm{x}).$$

The $\omega^k \in \mathbb{R}$ are chosen to minimize the residual norm. By defining

$$S^k = \left[\mathcal{R}^{k+1} \mathcal{Q}^k\right] (AM^{-1}) \circ R^0,$$

we obtain the following update formulas

$$\begin{aligned}
R^{k+1} &= \left[\mathcal{R}^{k+1}\mathcal{Q}^{k+1}\right](AM^{-1}) \circ R^0 \\
&= \left[\mathcal{R}^{k+1}\mathcal{Q}^{k} - \omega^k \mathrm{x}\mathcal{R}^{k+1}\mathcal{Q}^{k}\right](AM^{-1}) \circ R^0 \\
&= S^k - \omega^k AM^{-1}S^k, \\
P^{k+1} &= \left[\mathcal{P}^{k+1}\mathcal{Q}^{k+1}\right](M^{-1}A) \circ M^{-1}R^0 \\
&= \left[\mathcal{R}^{k+1}\mathcal{Q}^{k+1} + \mathcal{P}^{k}\mathcal{Q}^{k+1}\beta^k\right](M^{-1}A) \circ M^{-1}R^0 \\
&= M^{-1}R^{k+1} + \left[\mathcal{P}^{k}\mathcal{Q}^{k} - \omega^k \mathrm{x}\mathcal{P}^{k}\mathcal{Q}^{k}\right](M^{-1}A) \circ M^{-1}R^0\beta^k \\
&= M^{-1}R^{k+1} + \left(P^k - \omega^k M^{-1}AP^k\right)\beta^k, \\
S^k &= \left[\mathcal{R}^{k+1}\mathcal{Q}^{k}\right](AM^{-1}) \circ R^0 \\
&= \left[\mathcal{R}^{k}\mathcal{Q}^{k} - \mathrm{x}\mathcal{P}^{k}\mathcal{Q}^{k}\lambda^k\right](AM^{-1}) \circ R^0 \\
&= R^k - AP^k\lambda^k.
\end{aligned}$$

Finally, we need to deduce the coefficients ω^k, λ^k and β^k. As mentioned before, the ω^k is determined by minimizing the residual $R^{k+1} = S^k - \omega^k AM^{-1}S^k$ in the Frobenius norm, which yields

$$\omega^k = \frac{\langle AM^{-1}S^k, S^k\rangle_F}{\langle AM^{-1}S^k, AM^{-1}S^k\rangle_F}.$$

The other coefficients are determined by the formal orthogonality condition for $\mathcal{R}^k$ and $\mathcal{P}^k$. We have

$$\begin{aligned}
0 &= \mathcal{C}\left(\mathcal{R}^{k+1}\mathcal{Q}^{k}\right) \\
&= \mathcal{C}\left(\mathcal{R}^{k}\mathcal{Q}^{k}\right) - \mathcal{C}\left(\mathrm{x}\mathcal{P}^{k}\mathcal{Q}^{k}\right)\lambda^k \\
&= \langle M^{-\mathsf{T}}\tilde{R}^0, R^k\rangle_{\mathbb{S}} - \langle M^{-\mathsf{T}}\tilde{R}^0, AP^k\rangle_{\mathbb{S}}\lambda^k \\
\Rightarrow \lambda^k &= \left(\langle M^{-\mathsf{T}}\tilde{R}^0, AP^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-\mathsf{T}}\tilde{R}^0, R^k\rangle_{\mathbb{S}}
\end{aligned}$$

and

$$\begin{aligned}
0 &= \mathcal{C}^{(1)}(\mathcal{P}^{k+1}\mathcal{Q}^{k}) \\
&= \mathcal{C}^{(1)}(\mathcal{R}^{k+1}\mathcal{Q}^{k}) + \mathcal{C}^{(1)}(\mathcal{P}^{k}\mathcal{Q}^{k})\beta^k \\
&= \langle M^{-\mathsf{T}}\tilde{R}^0, AM^{-1}S^k\rangle_{\mathbb{S}} + \langle M^{-\mathsf{T}}\tilde{R}^0, AP^k\rangle_{\mathbb{S}}\beta^k \\
\Rightarrow \beta^k &= -\left(\langle M^{-\mathsf{T}}\tilde{R}^0, AP^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-\mathsf{T}}\tilde{R}^0, AM^{-1}S^k\rangle_{\mathbb{S}}.
\end{aligned}$$

Now we can formulate the BBiCGStab algorithm, see Algorithm 6.2. Note that S and R as well as Q and T and V and U can share memory pairwise. Furthermore, the

Algorithm 6.2: Block BiCGStab Method

$R^0 = B - AX^0$
$P^0 = M^{-1}R^0$
Choose $M^{-\mathsf{T}}\widetilde{R}^0$ arbitrary (e.g. $M^{-\mathsf{T}}\widetilde{R}^0 = P^0$)
$V^0 = P^0$
for $k = 0, \ldots$ **do**
  $Q^k = AP^k$
  $\lambda^k = \left(\langle M^{-\mathsf{T}}\widetilde{R}^0, Q^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-\mathsf{T}}\widetilde{R}^0, R^k\rangle_{\mathbb{S}}$
  $S^k = R^k - Q^k\lambda^k$
  $Z^k = M^{-1}Q^k$
  $T^k = V^k - Z^k\lambda^k$
  $U^k = AT^k$
  $\omega^k = \frac{\langle U^k, S^k\rangle_F}{\langle U^k, U^k\rangle_F}$
  $X^{k+1} = X^k + P^k\lambda^k + \omega^k T^k$
  $R^{k+1} = S^k - \omega^k U^k$
  break if $\|R^{k+1}\| \leq \varepsilon_{\text{tol}}$
  $V^{k+1} = M^{-1}R^{k+1}$
  $\beta^k = -\left(\langle M^{-\mathsf{T}}\widetilde{R}^0, Q^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-\mathsf{T}}\widetilde{R}^0, U^k\rangle_{\mathbb{S}}$
  $P^{k+1} = V^{k+1} + \left(P^k - \omega^k Z^k\right)\beta^k$
end for

algorithm does not apply the transposed of the operator or preconditioner.

It is the subject of further investigation whether we could choose the stabilization parameter ω^k in the space $\mathbb{S}$. A naive implementation by the author did not work. Furthermore, an adaption of the BiCGStab(ℓ) method would be interesting. An approach was recently proposed by Saito, Tadano and Imakura [STI14].

6.2 Residual Re-Orthonormalization

Like in the BCG algorithm we can improve and stabilize the convergence by adding a residual re-orthonormalization approach. This helps to resolve rank-deficiencies in the residual. Unfortunately, the BBiCGStab method is not as stable as the BCG method, especially at lower precision. We transform the intermediate residual using the normalizer, such that

$$\hat{S}^k\sigma^k = S^k,$$

where σ^k is an element in the *-subalgebra $\sigma^k \in \mathbb{S}$. All other variables are transformed similarly, using the same σ^k. The transformed variables are denoted with a hat.

The recurrence coefficients $\hat{\lambda}$ and $\hat{\beta}$ are computed by using the recurrence formulas

of P and S and assuming that σ is invertible. We have

$$\begin{aligned}\hat{P}^{k+1}\sigma^{k+1} &= P^{k+1}\\ &= V^{k+1} + \left(P^k - \omega^k Z^k\right)\beta^k\\ &= \hat{V}^{k+1}\sigma^{k+1} + \left(\hat{P}^k\sigma^k + \omega^k\hat{Z}^k\sigma^k\right)\beta^k\\ &= \hat{V}^{k+1}\sigma^{k+1} + \left(\hat{P}^k + \omega^k\hat{Z}^k\right)\hat{\beta}^k\sigma^{k+1}\end{aligned}$$

where we define $\hat{\beta}^k$ by

$$\hat{\beta}^k\sigma^{k+1} = \sigma^k\beta^k.$$

This can be used to deduce a formula for $\hat{\beta}^k$

$$\begin{aligned}\sigma^k\beta^k &= \sigma^k\left(\langle M^{-\mathsf{T}}\tilde{R}^0, Q^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-\mathsf{T}}\tilde{R}^0, U^k\rangle_{\mathbb{S}}\\ &= \left(\langle M^{-\mathsf{T}}\tilde{R}^0, \hat{Q}^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-\mathsf{T}}\tilde{R}^0, \hat{U}^k\rangle_{\mathbb{S}}\sigma^{k+1}\\ \Rightarrow \hat{\beta}^k &= \left(\langle M^{-\mathsf{T}}\tilde{R}^0, \hat{Q}^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-\mathsf{T}}\tilde{R}^0, \hat{U}^k\rangle_{\mathbb{S}}.\end{aligned}$$

Here we used, that $\hat{U}^k$ is transformed with respect to the new transformation σ^{k+1}. A similar computation can be made for $\hat{\lambda}^k$ which shows that we have

$$\sigma^k\lambda^k = \hat{\lambda}^k\sigma^k.$$

Thus

$$\hat{\lambda}^k = \left(\langle M^{-\mathsf{T}}\tilde{R}^0, \hat{Q}^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-\mathsf{T}}\tilde{R}^0, \hat{R}^k\rangle_{\mathbb{S}}.$$

The stabilization coefficient $\hat{\omega}^k$ is chosen to minimize the transformed residual

$$\hat{\omega}^k = \frac{\langle\hat{U}^k, \hat{S}^k\rangle_F}{\langle\hat{U}^k, \hat{U}^k\rangle_F}.$$

As with the BCG method, we apply the re-orthonormalization adaptively. We decide on the basis of the diagonally scaled condition number $\kappa_D(\langle\hat{R}^k, \hat{R}^k\rangle_{\mathbb{S}})$ whether we re-orthogonalize the residual. This quantity was a heuristically choice. Additionally, we introduce a parameter η to be able to tune the re-orthonormalization behavior. As shown in the following numeric section, this choice works properly. The algorithm can be found in Algorithm 6.3.

We avoid computing the inverse of γ^k, because it is badly conditioned if R^k is numerically rank deficient. Therefore we apply the preconditioner directly to $\hat{S}^k$ to

Algorithm 6.3: BBiCGStab with Adaptive Residual Re-Orthonormalization

$R^0 = B - AX^0$
if $\eta \neq 0$ **then**
  $\hat{R}^0, \sigma^0 = \mathrm{Norm}_{\mathbb{S}}\left(R^0\right)$
else
  $\sigma^0 = \mathbb{I}$
end if
$\hat{P}^0 = M^{-1}\hat{R}^0$
Choose $M^{-\mathsf{T}}\widetilde{R}^0$ (e.g. $M^{-\mathsf{T}}\widetilde{R}^0 = \hat{P}^0$)
$\hat{V}^0 = \hat{P}^0$
for $k = 0, \ldots$ **do**
  $\hat{Q}^k = A\hat{P}^k$
  $\hat{\lambda}^k = \left(\langle M^{-\mathsf{T}}\widetilde{R}^0, \hat{Q}^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-\mathsf{T}}\widetilde{R}^0, \hat{R}^k\rangle_{\mathbb{S}}$
  $\hat{Z}^k = M^{-1}\hat{Q}^k$
  $\hat{S}^k = \hat{R}^k - \hat{Q}^k\hat{\lambda}^k$
  $X^{k+\frac{1}{2}} = X^k + \hat{P}\lambda^k\sigma^k$
  if $\eta\kappa_D\left(\langle\hat{R}^k, \hat{R}^k\rangle_{\mathbb{S}}\right) > \sqrt{\varepsilon_{\mathrm{mach}}}$ **then**
    $\hat{S}^k, \gamma^k \leftarrow \mathrm{Norm}_{\mathbb{S}}\left(\hat{S}^k\right)$
    $\sigma^{k+1} = \gamma^k\sigma^k$
    $\hat{T}^k = M^{-1}\hat{S}^k$
  else
    $\hat{T}^k = \hat{V}^k - \hat{Z}^k\hat{\lambda}^k$
  end if
  $\hat{U}^k = A\hat{T}^k$
  $\hat{\omega}^k = \frac{\langle\hat{U}^k, \hat{S}^k\rangle_F}{\langle\hat{U}^k, \hat{U}^k\rangle_F}$
  $X^{k+1} = X^{k+\frac{1}{2}} + \hat{\omega}^k\hat{T}^k\sigma^k$
  $\hat{R}^{k+1} = \hat{S}^k - \hat{\omega}^k\hat{U}^k$
  break if $\|R^{k+1}\| \leq \varepsilon_{\mathrm{tol}}$
  $\hat{V}^{k+1} = M^{-1}\hat{R}^{k+1}$
  $\hat{\beta}^k = -\left(\langle M^{-\mathsf{T}}\widetilde{R}^0, \hat{Q}^k\rangle_{\mathbb{S}}\right)^{-1}\langle M^{-\mathsf{T}}\widetilde{R}^0, \hat{U}^k\rangle_{\mathbb{S}}$
  $\hat{P}^{k+1} = \hat{V}^{k+1} + \left(\hat{P}^k - \hat{\omega}^k\hat{Z}^k\right)\hat{\beta}^k$
end for

Table 6.1: Convergence results of the BBiCGStab method. Gray lines indicate that no convergence was achieved within 500 iterations.

Method	p	rate	iterations	re-orthogonalizations
block-global	1	0.998	500	0
	2	0.998	500	3
	4	0.998	500	18
	8	1.023	500	95
	16	0.942	156	59
	32	0.815	59	42
	64	0.590	27	23
	128	0.428	13	10
	256	0.267	11	8
block-parallel	1	0.929	131	0
	2	0.895	90	58
	4	0.829	54	49
	8	0.708	33	30
	16	0.649	24	21
	32	0.541	18	15
	64	0.437	14	11
	128	0.352	12	9
	256	0.267	11	8

compute $\hat{T}^k$ in the case where a re-orthonormalization is performed.

6.3 Numerical Experiments

Like for the other methods we perform an experiment to compare the block-parallel and block-global methods for the BBiCGStab method. We used the same matrix and preconditioner as in the numerical experiments for the BGMRes method and a re-orthonormalization parameter of $\eta = 100$.

The convergence behavior is not very smooth for the BiCGStab method, therefore we decided to display the result in a table and not in a plot. The results are shown in Table 6.1. We see a similar behavior as for the other methods - the larger the p, the faster the convergence. The convergence rate denotes the average factor by that the residual norm is reduced per iteration. However, we do not have a relationship between the convergence of the $\frac{p}{q}$-block-parallel method and the corresponding p-block-global method. It seems that the block-global methods perform better with the BBiCGStab method.

Table 6.2 shows the number of iterations and re-orthonormalizations for the `Simon/raefsky3` problem from the SuiteSparse Matrix collection [DH11]. We used a block size of $s = 32$ and the block method $\mathbb{S}_B$. The break criteria for the

Table 6.2: Iterations of the BBiCGStab method with residual re-orthonormalization. Gray lines indicate that the method did not converged within 2000 iterations.

η	rate	iterations	re-orthogonalizations
0.000	1.036	2000	0
0.001	0.967	299	106
0.010	0.967	294	103
0.100	0.967	286	116
1.000	0.968	295	162
10.000	0.964	258	153
100.000	0.963	261	217

method was a reduction of the max-column norm by a factor of 10^{-7}. The results show that re-orthonormalization is necessary to achieve convergence, but a larger re-orthonormalization parameter η does not necessarily lead to fewer iterations.

Part 2

Communication-aware Block Krylov Methods

We cannot solve the problems using the same kind of thinking we used when we created them.

Albert Einstein

7

Challenges on Large Scale Parallel Systems

In this part of the thesis we optimize the algorithms presented in Part 1 with respect to the global communications, i.e. the inner products and orthonormalizations that must be performed during the iteration. This optimization is twofold. On the one hand we reduce the number of synchronization points, i.e. compute multiple block inner products simultaneously. On the other hand we overlap the communication for the block inner products with computation.

In the current research there are a couple of approaches to minimize the communication overhead of Krylov methods. The most popular are so called s-step Krylov methods [CG89; CK90; Hoe10]. These methods reduce the number of global communications by enlarging the Krylov space by s dimensions in every iteration. The communication needed to find the minimizer in that space and orthogonalize the basis could then be carried out chunk-wise. While decreasing the number of messages by a factor of s, these methods suffer from instabilities. A lot of investigations have been made to mitigate these instabilities. A rigorous analysis of the round-off errors in s-step methods can be found in the PhD thesis of Carson [Car15]. The techniques to stabilize the methods are quite sophisticated and require information about the spectrum of the operator. Another problem is that we often use preconditioners that realize some kind of coarse grid correction, which behave similar to a global communication. Hence, the computation of the s-step basis would take as long as s global communications, which reduces the benefits of the method. Our aim is to overlap the computation of the preconditioner with the block inner product, such that every iteration effectively only needs one global synchronization. Recently, Eller [Ell19] presented a very elaborate performance study of pipelined Krylov solvers on modern HPC hardware, showing that a clever organization of communication can yield a significant performance improvement.

Another approach to solve linear systems while reducing the communication overhead are asynchronous iterations, cf. Chazan and Miranker [CM69] and Frommer and Szyld [FS00]. The idea behind these methods is that every process iterates in its own speed and communicates the updates asynchronously. In particular these methods are non-deterministic. The strength of this method is that they can handle unreliable networks and heterogeneous architectures well, as the computation proceeds even if one node is delayed.

Finally, the block Krylov methods presented in Part 1 already reduce the communication overhead as they reduce the number of iterations. This is used by the enlarged Krylov methods, like presented by Grigori and Tissot [GT17] who make use of this advantage for single right-hand side problems.

In the first section of this chapter we look at the implementation of the non-blocking collective communication in our code. Thereafter, we introduce a benchmark for quantifying the available time during a collective communication for computations. In the last section we look at the TSQR algorithm that is used to compute a QR decomposition of a tall-skinny matrix with only one collective communication.

We use the terms “global communication”, “collective communication” and “reduction” synonymous. They all denote the communication procedure that is carried out to compute for example a global sum.

All numerical tests in this part are carried out on 128 nodes of the supercomputer PALMAII at the University of Münster. Each node is an Intel Xeon Gold 6140 18C (Skylake) with 2.3 GHz connected by a 100 Gbit/s Intel Omni Path network. We use the optimized Intel MPI library Version 2018 Update 4 in all experiments. Furthermore, we always use `I_MPI_ASYNC_PROGRESS=1`, if not stated otherwise. See Section 7.2 for a discussion about that parameter.

7.1 Implementation

The technical foundation for asynchronous collective communication was introduced with the MPI 3 standard[1]. Within this standard definition, functions for non-blocking collective communication were introduced. These functions does not only allow to overlap the communication with computation, but also to have multiple communications in progress simultaneously. In particular, the `MPI_Iallreduce` function which is needed for the asynchronous computation of inner block products. In the Dune framework, we introduced an abstraction layer for the asynchronicity which makes use of the future-concept, shown in Listing 7.1. It provides the `ready()` method for checking the status of the communication, as well as the `wait()` method to block the execution until the communication has finished. In addition, the communicated data could be obtained with the `get()` method. The `Future` object encapsulate the communicated data and

[1] `https://www.mpi-forum.org/mpi-30/`

```
template<class T>
class Future{
  bool valid();
  bool ready();
  void wait();
  T get();
};
```

Listing 7.1: The concept of a `Dune::Future`.

```
  Future<Block<X>> gamma_future = sp->idot(x,y);
  op->apply(x,z); // This computation is overlapped with communication
  Block<X> gamma = gamma_future.get();
```

Listing 7.2: Example: How to use the Future interface of the `ScalarProduct`.

the `MPI_STATUS` handle. We made this available in the DUNE framework as part of the DUNE-COMMON module.

Furthermore, we extended the `ScalarProduct` interface by a function `idot`, that returns a `Future<Block<X>>` and a function `inormalize`, computing the normalizer of a block vector that is returned in a `Future<Block<X>>`. This allows us to write concurrent code in a C++ way, without thinking too much about resource allocation and technical details. An example how to use this interface is given in Listing 7.2. For the sequential case, e.g. if no MPI is available we provide a fallback implementation that implements a Future that does nothing but hold the object.

As all the methods in this work, we plan to make the code publicly available in the DUNE-ISTL module of the DUNE framework. A prototype implementation can be found in the GitLab repository of the author[2].

7.2 Collective Communication Benchmark

To quantify the costs of the collective communication in our environment we implemented a benchmark. The benchmark is inspired by the one presented by Lawry et al. [Law+02]. As we want to overlap the communication with computation we measure how much of the communication time is available for computations. For that we proceed as follows. We measure the time that is needed for initiating a global communication and immediately waiting for it (`MPI_Wait`). We call this time the base time ($t_{\texttt{base}}$). After that, we execute the same test again, but introduce a busy wait between the initiation and the finalization of the global communication. The time we measure is called the iteration time ($t_{\texttt{iter}}$). The time we spend in the busy wait is called the work time ($t_{\texttt{work}}$). To mitigate outliers, we execute all these measurements 1000 times and take the average. We start with a work time equal to one quarter of the base

[2] https://gitlab.dune-project.org/nils.dreier/dune-common

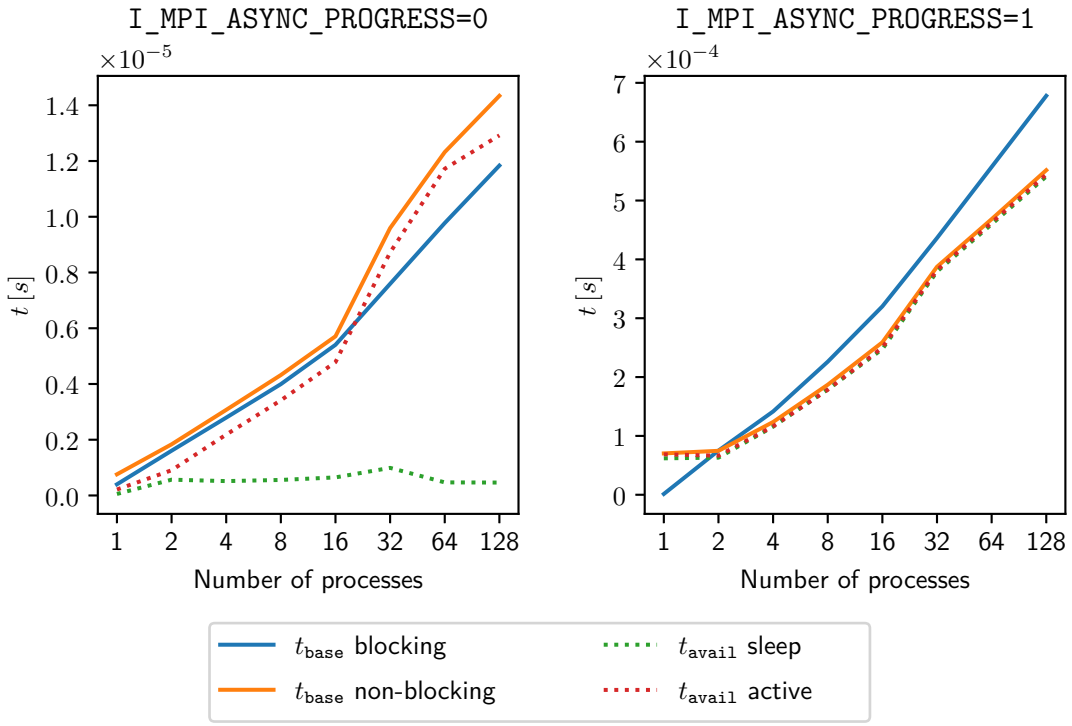


Figure 7.1: Collective communication benchmark. `I_MPI_ASYNC_PROGRESS` on vs. off. Solid lines show the duration of the communication. Dotted lines show the portion of time that can be used for computations.

time. After each repetition, the work time is doubled. We repeat this until the iteration time is larger than two times the base time. After that we determine the overhead as

$$t_{\texttt{ovhd}} = t_{\texttt{iter}} - t_{\texttt{work}},$$

and the available time for computation is

$$t_{\texttt{avail}} = t_{\texttt{base}} - t_{\texttt{ovhd}}.$$

The MPI standard only guaranties that progress in the communication is made if calls to MPI are made. However, some MPI implementations can be configured such that they proceed even if no calls are made. For Intel MPI this switch is called `I_MPI_ASYNC_PROGRESS`. To distinguish this case, we introduce two tests. In the first test `NB_sleep` we do not make any MPI calls in the busy wait. In the second case `NP_active` we call `MPI_Status` on the `MPI_Request` during the busy wait.

Figure 7.1 shows the results for our test environment for the `MPI_Iallreduce` communication. If `I_MPI_ASYNC_PROGRESS` is turned off, communication-computation overlap is only feasible in the `NB_active` case. In particular for a large number of processes most of the communication time can be used for computation. If `I_MPI_ASYNC_PROGRESS` is turned on, the `NB_active` and `NB_sleep` case do not differ,

i.e. communication computation overlap is also feasible in the `NB_sleep` case, where no MPI calls are made during the communication. In this setting 99% of the communication time can be used for computation. However, the overall time in the case where `I_MPI_ASYNC_PROGRESS` is turned on is much higher than in the other case. Notice the different scaling of the y-axes. From a practical point of view it does only make sense to activate `I_MPI_ASYNC_PROGRESS` if the overlap is exploited aggressively, because the overall communication time increase by a factor of approximately 50. An implementation of the benchmark can also be found in the GitLab repository of the author[3].

An alternative to turning `I_MPI_ASYNC_PROGRESS` on, was presented by Wittmann et al. [Wit+13]. In their approach, they spawn a dedicated thread that makes MPI calls regularly to ensure progress for non-blocking communication, even if no calls to MPI are made from the user code.

Let us now transfer these results to a hypothetical exascale machine, that consists of the same nodes as our test environment. One node in our test environment has a peak flop rate of 2.65 TFlop/s. Hence, for an exascale machine $P \approx 380\,000$ nodes are needed. From Figure 7.1 we can see that the collective communication scale like $2\log_2(P)$µs (if `I_MPI_ASYNC_PROGRESS=0`). Thus, a global reduction on this machine would take

$$2\log_2(380000)\mu\text{s} \approx 37\,\mu\text{s}.$$

Figure 7.2 shows the time that is needed for a global reduction on a hypothetical machine, as described above. It shows that the costs of a global reduction will grow for larger systems. The costs increase from petascale to exascale by $2\log_2(1000)\mu\text{s} \approx 20\,\mu\text{s}$, equally from exascale to zettascale. However, this assumes a quite optimal model. It is not clear that the scaling is still true for machines of this size, as small disturbance on one node would block the entire machine.

In particular, if the solution of a system is really time critical, e.g. weather forecasts, the only possibility to solve it within the time constraints is to use larger machines. This would probably lead to relative small systems per node for which the communication overhead plays a significant role.

7.3 The TSQR Algorithm

Before we consider the block Krylov methods in the next chapters, we look at the QR decomposition algorithm used in the distributed setting to compute the normalizer. Due to the shape of the block vectors for which the QR decomposition is computed, this algorithm is called Tall-Skinny QR (TSQR). It falls into the category of divide-

[3] `https://gitlab.dune-project.org/nils.dreier/dune-common/-/tree/master/dune/common/parallel/benchmark`

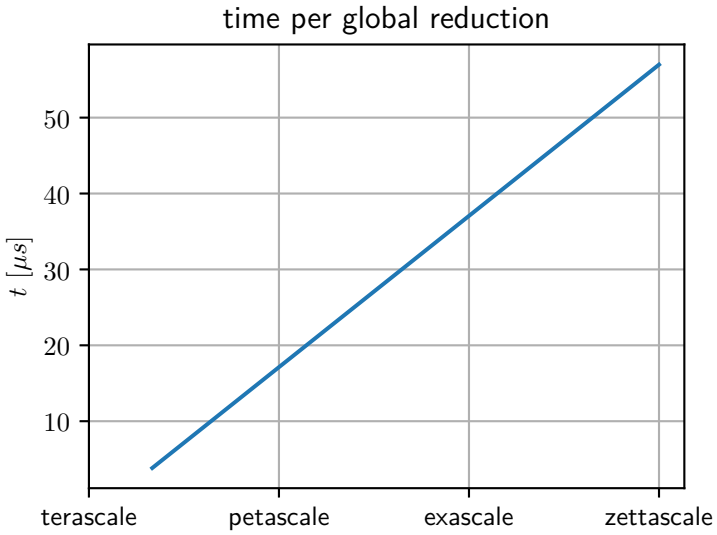


Figure 7.2: Time per global reduction vs. peak performance of the hypothetical machine. The x-axis is scaled logarithmically.

and-conquer algorithms.

The algorithm was presented by Demmel et al. [Dem+08; Dem+12]. It reduces the communication for computing a QR decomposition

$$X = Q\sigma$$

of a block vector $X \in \mathbb{R}^{n\times s}$, where $s \ll n$, $Q \in \mathbb{R}^{n\times s}$ with $Q^{\mathsf{T}}Q = \mathbb{I}$ and $\sigma \in \mathbb{R}^{s\times s}$ is upper triangular. We will review this algorithm and adapt it to our framework. We start with the definition of the $\mathbb{S}$-QR decomposition.

Definition 7.1 ($\mathbb{S}$-QR decomposition)**.** *Let $X \in \mathbb{R}^{n\times s}$ be a block vector. We call a decomposition of the type*

$$X = Q\sigma \qquad\qquad Q \in \mathbb{R}^{n\times s},$$

where $\sigma \in \mathbb{S}$ is upper triangular and

$$\langle Q, Q\rangle_{\mathbb{S}} = \mathbb{I}$$

a $\mathbb{S}$-QR decomposition.

Note that σ is a $\mathbb{S}$-normalizer of X, with respect to the normalized block vector Q. The algorithm is based on the following computation. Consider a block vector X

which is distributed onto P processes

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_P \end{bmatrix} \in \mathbb{R}^{n \times s},$$

where X_p is stored on process p. This vector can be decomposed without communication into

$$\begin{bmatrix} X_1 \\ \vdots \\ X_P \end{bmatrix} = \begin{bmatrix} Q_1\sigma_1 \\ \vdots \\ Q_P\sigma_N \end{bmatrix} = \begin{bmatrix} Q_1 & & \\ & \ddots & \\ & & Q_P \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_P \end{bmatrix}$$

where

$$X_p = Q_p\sigma_p \tag{7.1}$$

is a $\mathbb{S}$-QR decomposition for all $p = 1, \ldots, P$. The R-factors of these decompositions are gathered on one process and decomposed into

$$\begin{bmatrix} \sigma_1 \\ \vdots \\ \sigma_P \end{bmatrix} = \begin{bmatrix} \widetilde{\sigma}_1 \\ \vdots \\ \widetilde{\sigma}_P \end{bmatrix} \sigma \qquad\qquad \widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_P, \sigma \in \mathbb{S},$$

with σ is upper triangular and

$$\sum_{p=1}^{P} \widetilde{\sigma}_p^{\mathsf{T}} \widetilde{\sigma}_p = \mathbb{I}.$$

The $\widetilde{\sigma}_p$ are scattered back to the respective processes. Then the decomposition

$$X = \begin{bmatrix} Q_1\widetilde{\sigma}_1 \\ \vdots \\ Q_P\widetilde{\sigma}_N \end{bmatrix} \sigma$$

forms a $\mathbb{S}$-QR decomposition, since

$$\left\langle \begin{bmatrix} Q_1\widetilde{\sigma}_1 \\ \vdots \\ Q_P\widetilde{\sigma}_N \end{bmatrix}, \begin{bmatrix} Q_1\widetilde{\sigma}_1 \\ \vdots \\ Q_P\widetilde{\sigma}_N \end{bmatrix} \right\rangle_{\mathbb{S}} = \sum_{p=1}^{P} \widetilde{\sigma}_p^{\mathsf{T}} \langle Q_p, Q_p \rangle_{\mathbb{S}} \widetilde{\sigma}_p = \sum_{p=1}^{P} \widetilde{\sigma}_p^{\mathsf{T}} \widetilde{\sigma}_p = \mathbb{I}$$

and σ is upper triangular.

On large systems with many processes this approach can be carried out recursively.

In that case a tree hierarchy is created in the processes and the local decomposition (7.1) is computed with the same algorithm. The communication pattern is the same as for a `MPI_Allreduce`. The MPI standard allows for implementing custom reduction methods. This is sufficient for computing the R-factor, like presented in the work of Langou [Lan10]. Unfortunately, the MPI standard does not allow to define a custom function for the back scattering of the $\widetilde{\sigma}$, which would be needed to compute the Q-factor in the rank deficient case.

We will see how we can use the idea of this algorithm to create a new orthogonalization algorithm for BGMRes that is communication optimal and stable.

An alternative to the TSQR algorithm is the CholeskyQR algorithm [SW02]. It also uses only one reduction to compute the tall-skinny QR factorization of a block vector and relies on the Cholesky factorization of the block inner product

$$\lambda^{\mathsf{T}}\lambda = \langle X, X\rangle_{\mathbb{S}} \qquad\qquad \lambda \in \mathbb{S},\ \text{upper right triangular.}$$

The normalized vector could then be computed as

$$Q = X\lambda^{-\mathsf{T}}. \tag{7.2}$$

From (7.2) we see that it inverts the Cholesky factor. Therefore, it only works for full-rank X. In particular, it is unsuitable for our stabilization strategies.

8
Pipelined Block CG Method

Our BCG algorithm with adaptive re-orthonormalization, Algorithm 4.3, uses three blocking global communications per iteration – minimization, orthogonalization and convergence check/re-orthonormalization. The present chapter aims at reducing this. As a first step we fuse multiple inner products, such that the communication can be carried out simultaneously, reducing the number of synchronization points per iteration. In the second section we make use of the non-blocking features described in the last section to overlap the communication with computation. Finally, we compare all the derived variants with respect to their communication overhead.

8.1 Fusing Inner Block Products

We fuse the communication of the convergence check with the orthogonalization, i.e. the communication for η^k and ρ^k. Furthermore, the update of X could be delayed such that it can be done during this communication. These optimizations can be made without introducing any additional variables or operations. The resulting algorithm is shown in Algorithm 8.1. Arrows mark the initiation and finalization of the global communications. The algorithm is arithmetically equivalent to Algorithm 4.3. It could be slightly further improved by checking after line 25 and 26 whether the communication of η^k is already finished and then break if the convergence criterion is satisfied. This would terminate the algorithm a bit earlier.

As a next step, we reduce the number of synchronization points to one. Unfortunately, that is not possible without adding memory and computational overhead. We introduce the auxiliary variable $U^k = AZ^k$, that can be computed right after the

Algorithm 8.1: BCG with Two Reductions (2R-BCG)

$R^0 = B - AX^0$
if $\eta > 0$ **then**
  $\bar{R}^0, \sigma^0 = \mathrm{Norm}_{\mathbb{S}}\left(R^0\right)$
else
  $\bar{R}^0 = R^0$
  $\sigma^0 = \mathbb{I}$
end if
$P^0 = M^{-1}\bar{R}^0$
$\rho^0 = \langle P^0, \bar{R}^0 \rangle_{\mathbb{S}}$
for $k = 0, \ldots$ until convergence **do**
  $Q^k = AP^k$
  $\alpha^k = \langle P^k, Q^k \rangle_{\mathbb{S}}$
  $\lambda^k = \left(\alpha^k\right)^{-1} \rho^k$
  $\widetilde{R}^{k+1} = \bar{R}^k - Q^k\lambda^k$
  if $\eta\kappa_D(\alpha^k) > \sqrt{\varepsilon_{\mathrm{mach}}}$ **then**
    $\bar{R}^{k+1}, \gamma^{k+1} = \mathrm{Norm}_{\mathbb{S}}\left(\widetilde{R}^{k+1}\right)$
    $\sigma^{k+1} = \gamma^{k+1}\sigma^k$
    $\eta^{k+1} = \|\sigma^{k+1}\|$
  else
    $\bar{R}^{k+1} = \widetilde{R}^{k+1}$
    $\gamma^{k+1} = \mathbb{I}$
    $\sigma^{k+1} = \sigma^k$
    $\eta^{k+1} = \|\bar{R}^{k+1}\sigma^{k+1}\|$
  end if
  $X^{k+1} = X^k + P^k\lambda^k\sigma^k$
  $Z^{k+1} = M^{-1}\bar{R}^{k+1}$
  $\rho^{k+1} = \langle Z^{k+1}, \bar{R}^{k+1} \rangle_{\mathbb{S}}$
  break if $\eta^{k+1} \leq \varepsilon_{\mathrm{tol}}$
  $\beta^k = \left(\rho^k\right)^{-1} {\gamma^{k+1}}^{\mathsf{T}} \rho^{k+1}$
  $P^{k+1} = Z^{k+1} + P^k\beta^k$
end for

application of the preconditioner. Thus we can precompute α^{k+1} by

$$\begin{aligned}\alpha^{k+1} &= \langle P^{k+1}, Q^{k+1}\rangle_{\mathbb{S}} \\ &= \langle Z^{k+1} + P^k\beta^k, U^{k+1} + Q^k\beta^k\rangle_{\mathbb{S}} \\ &= \langle Z^{k+1}, U^{k+1}\rangle_{\mathbb{S}} + \langle Z^{k+1}, Q^k\beta^k\rangle_{\mathbb{S}} + \langle P^k\beta^k, U^{k+1}\rangle_{\mathbb{S}} + \langle P^k\beta^k, Q^k\beta^k\rangle_{\mathbb{S}} \\ &= \delta^{k+1} + \langle Z^{k+1}, Q^k\rangle_{\mathbb{S}}\beta^k + \left(\langle Z^{k+1}, Q^k\rangle_{\mathbb{S}}\beta^k\right)^{\mathsf{T}} + {\beta^k}^{\mathsf{T}}\alpha^k\beta^k \end{aligned} \tag{8.1}$$

with $\delta^{k+1} := \langle Z^{k+1}, U^{k+1}\rangle_{\mathbb{S}}$. We simplify this further by computing

$$\begin{aligned}\langle Z^{k+1}, Q^k\rangle_{\mathbb{S}} &= \langle Z^{k+1}, \bar{R}^k - \tilde{R}^{k+1}\rangle_{\mathbb{S}}\left(\lambda^k\right)^{-1} \\ &= \langle Z^{k+1}, \bar{R}^k\rangle_{\mathbb{S}}\left(\lambda^k\right)^{-1} - \langle Z^{k+1}, \tilde{R}^{k+1}\rangle_{\mathbb{S}}\left(\lambda^k\right)^{-1} \\ &= -\langle Z^{k+1}, \bar{R}^{k+1}\rangle_{\mathbb{S}}\gamma^{k+1}\left(\rho^k\right)^{-1}\alpha^k \\ &= -{\beta^k}^{\mathsf{T}}\alpha^k,\end{aligned}$$

where we used that $\langle Z^{k+1}, \bar{R}^k\rangle_{\mathbb{S}} = \langle \bar{R}^{k+1}, Z^k\rangle_{\mathbb{S}} = 0$, as $Z^k \in \mathcal{K}^k_{\mathbb{S}}\left(M^{-1}A, M^{-1}R^0\right)$ and that ρ^{k+1} is symmetric. Equation (8.1) simplifies then to

$$\alpha^{k+1} = \delta^{k+1} - {\beta^k}^{\mathsf{T}}\alpha^k\beta^k.$$

We amend Algorithm 8.1 by computing δ^{k+1} together with ρ^{k+1} right after the preconditioner application. The resulting algorithm can be found in Algorithm 8.2.

Note that, compared to Algorithm 8.1, Algorithm 8.2 needs memory for 2 more block vectors. One for the additional variable U^k and one because Z^k and Q^k can not share their memory anymore, since Q^k is updated recursively.

8.2 Overlap Computation and Communication

With the introduction of asynchronous communication techniques we can overlap the communication procedure with computations, like the operator or preconditioner application. In this section, we amend the algorithms of the last section by integrating these techniques.

All these optimizations introduce additional computational and memory overhead. The benefit of asynchronicity must compensate for this overhead, otherwise Algorithm 4.3 would perform better. Especially for very sparse matrices or many right-hand sides, the vector updates that are introduced become easily similar expensive as the operator application or the global communication. In that case, Algorithm 8.1 is probably a good choice, as it is fairly optimized and does not introduce additional computational overhead.

As a first step, we develop an algorithm based on Algorithm 8.1 that overlaps the two block inner products with the operator and preconditioner application, respectively.

Algorithm 8.2: BCG with One Reduction (1R-BCG)

$R^0 = B - AX^0$
if $\eta > 0$ **then**
  $\bar{R}^0, \sigma^0 = \mathrm{Norm}_{\mathbb{S}}\left(R^0\right)$
else
  $\bar{R}^0 = R^0$
  $\sigma^0 = \mathbb{I}$
end if
$P^0 = M^{-1}\bar{R}^0$
$\rho^0 = \langle P^0, \bar{R}^0 \rangle_{\mathbb{S}}$
$Q^0 = AP^0$
$\alpha^0 = \langle P^0, Q^0 \rangle_{\mathbb{S}}$
for $k = 0, \ldots$ until convergence **do**
  $\lambda^k = \left(\alpha^k\right)^{-1} \rho^k$
  $\widetilde{R}^{k+1} = \bar{R}^k - Q^k \lambda^k$
  if $\eta \kappa_D(\alpha^k) > \sqrt{\varepsilon_{\mathrm{mach}}}$ **then**
    $\bar{R}^{k+1}, \gamma^{k+1} = \mathrm{Norm}_{\mathbb{S}}\left(\widetilde{R}^{k+1}\right)$
    $\sigma^{k+1} = \gamma^{k+1}\sigma^k$
    $\eta^{k+1} = \|\sigma^{k+1}\|$
  else
    $\bar{R}^{k+1} = \widetilde{R}^{k+1}$
    $\gamma^{k+1} = \mathbb{I}$
    $\sigma^{k+1} = \sigma^k$
    $\eta^{k+1} = \|\bar{R}^{k+1}\sigma^{k+1}\|$
  end if
  $X^{k+1} = X^k + P^k \lambda^k \sigma^k$
  $Z^{k+1} = M^{-1}\bar{R}^{k+1}$
  $\rho^{k+1} = \langle Z^{k+1}, \bar{R}^{k+1} \rangle_{\mathbb{S}}$
  $U^{k+1} = AZ^{k+1}$
  $\delta^{k+1} = \langle Z^{k+1}, U^{k+1} \rangle_{\mathbb{S}}$
  break if $\eta^{k+1} \leq \varepsilon_{\mathrm{tol}}$
  $\beta^k = \left(\rho^k\right)^{-1} {\gamma^{k+1}}^{\mathsf{T}} \rho^{k+1}$
  $P^{k+1} = Z^{k+1} + P^k \beta^k$
  $Q^{k+1} = U^{k+1} + Q^k \beta^k$
  $\alpha^{k+1} = \delta^{k+1} - {\beta^k}^{\mathsf{T}} \alpha^k \beta^k$
end for

The approach was suggested by Gropp [Gro10] for the non-block CG method.

To do so, we introduce two additional variables $U^k = AZ^k$ and $V^k = M^{-1}Q^k$. These variables are computed during the global communication of the block inner products and then used to update Q^k and Z^k recursively

$$\begin{aligned} Q^{k+1} &= U^{k+1} + Q^k \beta^k, \\ Z^{k+1} &= M^{-1}\bar{R}^{k+1} \\ &= M^{-1}\tilde{R}^{k+1}\left(\gamma^{k+1}\right)^{-1} \\ &= M^{-1}\bar{R}^{k}\left(\gamma^{k+1}\right)^{-1} - M^{-1}Q^k\lambda^k\left(\gamma^{k+1}\right)^{-1} \\ &= Z^k\left(\gamma^{k+1}\right)^{-1} - V^k\lambda^k\left(\gamma^{k+1}\right)^{-1}. \end{aligned}$$

For the reasons mentioned in Section 4.3, it is not favorable to invert γ^{k+1}. In the iterations where a re-orthonormalization is made we rely on the direct computation of $Z^{k+1} = M^{-1}\bar{R}^{k+1}$. It means that the preconditioner is applied twice in that iteration. However, this case occurs rarely.

In honor to Gropp [Gro10], we call this algorithm *Gropp's BCG*, it is shown in Algorithm 8.3.

The next algorithm is based on the one reduction variant of our BCG method, Algorithm 8.2. We use the same technique as before and introduce auxiliary variables to resolve dependencies and precompute the operator or preconditioner application. We reuse the variable V^k and the update formula of Z^k from the previous algorithm, but update V^k recursively, too. For that we introduce the variable $W^{k+1} = M^{-1}U^{k+1}$, allowing us update V^k as

$$\begin{aligned} V^{k+1} &= M^{-1}Q^{k+1} \\ &= M^{-1}U^{k+1} + M^{-1}Q^k\beta^k \\ &= W^{k+1} + V^k\beta^k. \end{aligned}$$

The variable W^{k+1} can be computed during the communication of the block inner products. The resulting algorithm is shown in Algorithm 8.4.

If the application of the preconditioner does not suffice to hide the global communication, it could be beneficial to apply the same strategy to precompute the operator application. For that we introduce the variables $S^k = AV^k$ and $T^k = AW^k$. The update of S^k is similar to that of V^k, it reads

$$\begin{aligned} S^{k+1} &= AV^{k+1} \\ &= AW^{k+1} + AV^k\beta^k \\ &= T^{k+1} + S^k\beta^k. \end{aligned}$$

Algorithm 8.3: Gropp's BCG

$R^0 = B - AX^0$
if $\eta > 0$ **then**
  $\bar{R}^0, \sigma^0 = \mathrm{Norm}_{\mathbb{S}}\left(R^0\right)$
else
  $\bar{R}^0 = R^0$
  $\sigma^0 = \mathbb{I}$
end if
$P^0 = M^{-1}\bar{R}^0$
$Q^0 = AP^0$
$Z^0 = P^0$
$\rho^0 = \langle P^0, \bar{R}^0 \rangle_{\mathbb{S}}$
for $k = 0, \ldots$ until convergence **do**
  $\alpha^k = \langle P^k, Q^k \rangle_{\mathbb{S}}$
  $V^k = M^{-1}Q^k$
  $\lambda^k = \left(\alpha^k\right)^{-1}\rho^k$
  $\widetilde{R}^{k+1} = \bar{R}^k - Q^k\lambda^k$
  if $\eta\kappa_D(\alpha^k) > \sqrt{\varepsilon_{\mathrm{mach}}}$ **then**
    $\bar{R}^{k+1}, \gamma^{k+1} = \mathrm{Norm}_{\mathbb{S}}\left(\widetilde{R}^{k+1}\right)$
    $\sigma^{k+1} = \gamma^{k+1}\sigma^k$
    $\eta^{k+1} = \|\sigma^{k+1}\|$
    $Z^{k+1} = M^{-1}\bar{R}^{k+1}$
  else
    $\bar{R}^{k+1} = \widetilde{R}^{k+1}$
    $\gamma^{k+1} = \mathbb{I}$
    $\sigma^{k+1} = \sigma^k$
    initiate $\eta^{k+1} = \|\bar{R}^{k+1}\sigma^{k+1}\|$
    $Z^{k+1} = Z^k - V^k\lambda^k$
  end if
  $X^{k+1} = X^k + P^k\lambda^k\sigma^k$
  $\rho^{k+1} = \langle Z^{k+1}, \bar{R}^{k+1} \rangle_{\mathbb{S}}$
  $U^{k+1} = AZ^{k+1}$
  break if $\eta^{k+1} \leq \varepsilon_{\mathrm{tol}}$
  $\beta^k = \left(\rho^k\right)^{-1}{\gamma^{k+1}}^{\mathsf{T}}\rho^{k+1}$
  $P^{k+1} = Z^{k+1} + P^k\beta^k$
  $Q^{k+1} = U^{k+1} + Q^k\beta^k$
end for

Algorithm 8.4: Partially Pipelined BCG (PPBCG)

$R^0 = B - AX^0$
if $\eta > 0$ **then**
  $\bar{R}^0, \sigma^0 = \mathrm{Norm}_{\mathbb{S}}\left(R^0\right)$
else
  $\bar{R}^0 = R^0 \qquad \sigma^0 = \mathbb{I}$
end if
$Z^0 = P^0 = M^{-1}\bar{R}^0$
$\rho^0 = \langle P^0, \bar{R}^0 \rangle_{\mathbb{S}}$
$Q^0 = AP^0$
$V^0 = M^{-1}Q^0$
$\alpha^0 = \langle P^0, Q^0 \rangle_{\mathbb{S}}$
for $k = 0, \ldots$ until convergence **do**
  $\lambda^k = \left(\alpha^k\right)^{-1}\rho^k$
  $\widetilde{R}^{k+1} = \bar{R}^k - Q^k\lambda^k$
  if $\eta\kappa_D(\alpha^k) > \sqrt{\varepsilon_{\mathrm{mach}}}$ **then**
    $\bar{R}^{k+1}, \gamma^{k+1} = \mathrm{Norm}_{\mathbb{S}}\left(\widetilde{R}^{k+1}\right)$
    $\sigma^{k+1} = \gamma^{k+1}\sigma^k$
    $\eta^{k+1} = \|\sigma^{k+1}\|$
    $Z^{k+1} = M^{-1}\bar{R}^{k+1}$
  else
    $\bar{R}^{k+1} = \widetilde{R}^{k+1}$
    $\gamma^{k+1} = \mathbb{I}$
    $\sigma^{k+1} = \sigma^k$
    initiate $\eta^{k+1} = \|\bar{R}^{k+1}\sigma^{k+1}\|$
    $Z^{k+1} = Z^k - V^k\lambda^k$
  end if
  $X^{k+1} = X^k + P^k\lambda^k\sigma^k$
  $\rho^{k+1} = \langle Z^{k+1}, \bar{R}^{k+1} \rangle_{\mathbb{S}}$
  $U^{k+1} = AZ^{k+1}$
  $\delta^{k+1} = \langle Z^{k+1}, U^{k+1} \rangle_{\mathbb{S}}$
  $W^{k+1} = M^{-1}U^{k+1}$
  break if $\eta^{k+1} \leq \varepsilon_{\mathrm{tol}}$
  $\beta^k = \left(\rho^k\right)^{-1}{\gamma^{k+1}}^{\mathsf{T}}\rho^{k+1}$
  $P^{k+1} = Z^{k+1} + P^k\beta^k$
  $Q^{k+1} = U^{k+1} + Q^k\beta^k$
  $V^{k+1} = W^{k+1} + V^k\beta^k$
  $\alpha^{k+1} = \delta^{k+1} - {\beta^k}^{\mathsf{T}}\alpha^k\beta^k$
end for

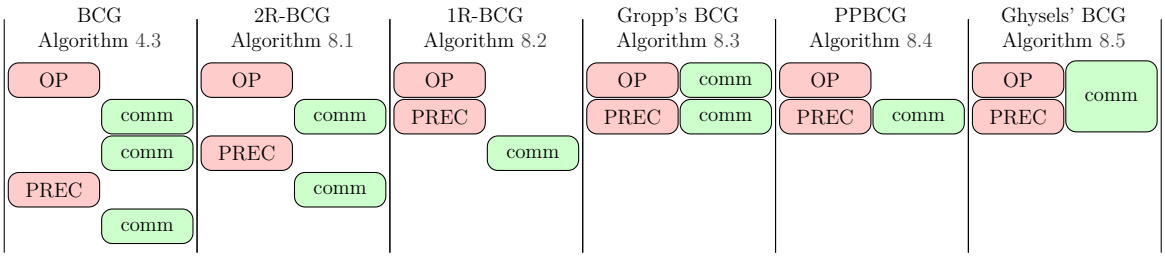


Figure 8.1: Schematic representation of the program flow in the different BCG variants. Red blocks are computational intensive procedures. Green blocks are communication. Horizontal alignment indicate concurrency.

Table 8.1: Arithmetical and communication properties of different BCG algorithms.

Method	vector storage	vector updates	synchronizations
BCG (Alg. 4.3)	4	3	3
2R-BCG (Alg. 8.1)	4	3	2
1R-BCG (Alg. 8.2)	6	4	1
Gropp's BCG (Alg. 8.3)	6	5	2 (overlapped)
PPBCG (Alg. 8.4)	8	6	1 (overlapped)
Ghysels' BCG (Alg. 8.5)	10	8	1 (overlapped)

The resulting algorithm was suggested by Ghysels and Vanroose [GV14; Ash+12] for the non-block CG. Therefore, we call the block variant Ghysels' BCG, It is shown in Algorithm 8.5.

Figure 8.1 and Table 8.1 give an overview of the Algorithms deduced in this chapter. In Figure 8.1 the schematic program flow of one iteration is shown. Red boxes represent computations, green ones represent communications. The flow direction is from top to bottom. Boxes that are placed next to each other horizontally mean, that these operations are executed simultaneously.

Table 8.1 gives an overview of the performance relevant characteristics. That is, how much memory is needed for the block vectors, how many `BAXPY` operations are performed per iterations and the number of synchronization points per iteration. We see how much overhead is introduced for optimizing the communication properties.

As the pipelining introduces additional `BAXPY` operations, it also increases the sensitivity for round-off errors. Cools et al. [Coo+18] analyzed the round-off errors for the pipelined CG method and proposed mitigation strategies [CV17a; CCV19]. These strategies include adaptive recomputation of the residual and variables or introduction of shifts in the recurrence formulas.

For the pipelined block versions of the CG method we observe a similar behavior. In principle the strategies proposed by Cools, Cornelis and Vanroose should be applicable to the $\mathbb{S}$-BCG method as well. However, the rigorous analysis and amendment of the algorithm is out of the scope of this work. Practical experiments showed that using a

Algorithm 8.5: Ghysels' BCG

$R^0 = B - AX^0$
if $\eta > 0$ **then**
  $\bar{R}^0, \sigma^0 = \mathrm{Norm}_{\mathbb{S}}\left(R^0\right)$
else
  $\bar{R}^0 = R^0 \qquad \sigma^0 = \mathbb{I}$
end if
$Z^0 = P^0 = M^{-1}\bar{R}^0$
$\rho^0 = \langle P^0, \bar{R}^0 \rangle_{\mathbb{S}}$
$U^0 = Q^0 = AP^0$
$V^0 = M^{-1}Q^0$
$S^0 = AV^0$
$\alpha^0 = \langle P^0, Q^0 \rangle_{\mathbb{S}}$
for $k = 0, \ldots$ until convergence **do**
  $\lambda^k = \left(\alpha^k\right)^{-1} \rho^k$
  $\widetilde{R}^{k+1} = \bar{R}^k - Q^k\lambda^k$
  if $\eta\kappa_D(\alpha^k) > \sqrt{\varepsilon_{\mathrm{mach}}}$ **then**
    $\bar{R}^{k+1}, \gamma^{k+1} = \mathrm{Norm}_{\mathbb{S}}\left(\widetilde{R}^{k+1}\right)$
    $\sigma^{k+1} = \gamma^{k+1}\sigma^k$
    $\eta^{k+1} = \|\sigma^{k+1}\|$
    $Z^{k+1} = M^{-1}\bar{R}^{k+1}$
    $U^{k+1} = AZ^{k+1}$
  else
    $\bar{R}^{k+1} = \widetilde{R}^{k+1}$
    $\gamma^{k+1} = \mathbb{I}$
    $\sigma^{k+1} = \sigma^k$
    initiate $\eta^{k+1} = \|\bar{R}^{k+1}\sigma^{k+1}\|$
    $Z^{k+1} = Z^k - V^k\lambda^k$
    $U^{k+1} = U^k - S^k\lambda^k$
  end if
  $X^{k+1} = X^k + P^k\lambda^k\sigma^k$
  $\rho^{k+1} = \langle Z^{k+1}, \bar{R}^{k+1} \rangle_{\mathbb{S}}$
  $\delta^{k+1} = \langle Z^{k+1}, U^{k+1} \rangle_{\mathbb{S}}$
  $W^{k+1} = M^{-1}U^{k+1}$
  $T^{k+1} = AW^{k+1}$
  break if $\eta^{k+1} \leq \varepsilon_{\mathrm{tol}}$
  $\beta^k = \left(\rho^k\right)^{-1} {\gamma^{k+1}}^{\mathsf{T}} \rho^{k+1}$
  $P^{k+1} = Z^{k+1} + P^k\beta^k$
  $Q^{k+1} = U^{k+1} + Q^k\beta^k$
  $V^{k+1} = W^{k+1} + V^k\beta^k$
  $S^{k+1} = T^{k+1} + S^k\beta^k$
  $\alpha^{k+1} = \delta^{k+1} - {\beta^k}^{\mathsf{T}}\alpha^k\beta^k$
end for

stronger preconditioner and using the residual re-orthonormalization strategy mitigates the instabilities and residual gap.

In extreme situations, it could be possible that the application of the operator and preconditioner do not suffice to overlap the entire costs of a global communication. In these situations Cornelis, Cools and Vanroose [CCV18; Coo+19] proposed a deep-pipelined CG method, that overlays multiple iterations by a global communication. This method requires further additional `BAXPY` operations and memory overhead. As we consider systems with multiple right-hand sides, the operator and preconditioner application is more expansive than in the non-block case. Therefore, we do not consider deep-pipelining for our block methods.

8.3 Numerical Experiments

To quantify the advantages of the communication optimized BCG variants we run strong scaling tests. We already presented a similar study in [Bas+20a]. The difficult part for running the tests is to choose a good problem size. The communication costs and computation costs must be well-balanced to see the differences of the algorithms. We observed three different regimes. Firstly, the computation costs are dominating the communication costs (e.g. in the sequential case or for large problems). Secondly, the communication and computation costs are similar. This is the case where the overlap of communication and computation is beneficial. In the last case the communication costs are dominating (e.g. on a large distributed system with a slow interconnect or for very small problems). In the latter case we expect the method, that uses the least global reductions, to be fastest.

With these considerations in mind we choose our test problem as a plain Poisson problem, discretized with a 5-point Finite Difference stencil on a 500×500 grid. We found this problem size experimentally by balancing the computational and communicational costs for $P = 16$ nodes. We use a thread parallel block SSOR preconditioner.

The result is shown in Figure 8.2. It shows the speedup of the iterations with respect of the sequential case ($P = 1$). We can observe the three regimes. For a smaller number of processes the BCG method (Algorithm 4.1) and 2R-BCG (Algorithm 8.1) are the fastest. This matches our expectations as these method do not add computational overhead. For a medium number of processes ($P \approx 16$) the methods that overlap communication and computation are advantageous, which are Gropp's BCG, PPBCG and Ghysels' BCG. In the scaling limit the methods that use only one global communication per iteration are fastest, which are 1R-BCG, PPBCG and Ghysels' BCG.

Note that the actual achieved speedup of $3.5\times$ for $P = 128$ nodes is pretty low. This is due to the small problem size. For $P = 128$ nodes each node has approximately 2000 DoFs. As every node consists of 36 cores, it results in approximately 55 DoFs per core. Thus, the non-optimal scaling results are due to the bad performance of the

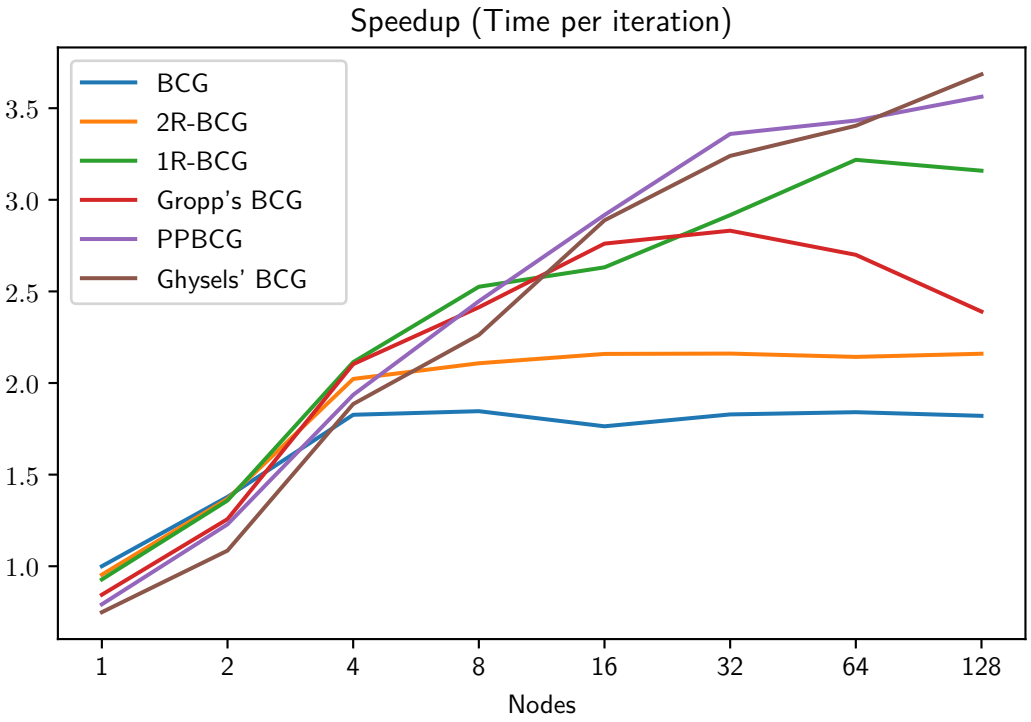


Figure 8.2: Strong scaling of the time per iteration for different CG variants. Speedup over sequential BCG.

preconditioner for this tiny problem size. However, we see that the optimization of global communication has an impact, even if the scaling is hindered by other effects.

9

Communication-optimized Block GMRes Method

In this chapter, we analyze and optimize the communication effort of the BGMRes method, introduced in Chapter 5. The orthogonalization in the Arnoldi process makes up the most communication costs. In particular, the communication overhead grows with every iteration. In the literature, the classical Gram-Schmidt method is often used in communication dominated settings. This is fairly optimal with respect to communication but suffers from instabilities for bad conditioned problems, cf. Giraud et al. [Gir+05].

Hoemmen presented in his thesis [Hoe10] a Communication-Avoiding GMRes method (CA-GMRes) that falls into the class of s-step methods. That is, the Arnoldi procedure applies the operator multiple times and the orthogonalization is carried out block-wise. In principle, this approach could be combined with all strategies presented in this chapter. However, as with all s-steps methods, the use of CA-GMRes would require the use of a stable s-step basis, which is not trivially constructed.

A pipelined method, like the one we have seen in the last chapter, was presented by Ghysels et al. [Ghy+13]. It overlaps the communication with the application of the operator and preconditioner. Nevertheless, as we consider block systems here, we concentrate on the Arnoldi procedure, as it is more expensive than in the non-block case. It should be costly enough to overlap the communication therein.

We develop and compare four variants of the BGMRes method in this chapter. All methods alter the orthogonalization algorithm used in the Arnoldi procedure. We start with the classical Gram-Schmidt orthogonalization with re-orthogonalization. Thereafter, we see how we can overlap the communication with the computation of the block inner products and block vector updates, leading to a pipelined version of the Gram-Schmidt algorithm. In addition, we develop a novel orthogonalization method

Algorithm 9.1: Modified Gram-Schmidt Orthogonalization

Input: X^{k+1} and $\mathbb{S}$-orthonormal basis $\mathcal{Q} = (Q^i)_{i=0}^k$
  $Q^{k+1} = X^{k+1}$
  for $i = 0, \ldots, k$ **do**
    $\rho_i^{k+1} = \langle Q^{k+1}, Q^i \rangle_{\mathbb{S}}$
    $Q^{k+1} \leftarrow Q^{k+1} - Q^i \rho_i^{k+1}$
  end for
  $Q^{k+1}, \rho_{k+1}^{k+1} \leftarrow \mathrm{Norm}_{\mathbb{S}}\left(Q^{k+1}\right)$
  return $Q^{k+1}, \rho_0^{k+1}, \ldots, \rho_{k+1}^{k+1}$

Algorithm 9.2: Classical($\#it$) Gram-Schmidt Orthogonalization

Input: X^{k+1} and $\mathbb{S}$-orthonormal basis $\mathcal{Q} = (Q^i)_{i=0}^k$ and a parameter $\#it \in 1, 2$
  $Q^{k+1} = X^{k+1}$
  for $j = 1, \ldots, \#it$ **do**
    for $i = 0, \ldots, k$ **do**
      $\rho_i^{k+1} = \langle Q^{k+1}, Q^i \rangle_{\mathbb{S}}$
    end for
    for $i = 0, \ldots, k$ **do**
      $Q^{k+1} \leftarrow Q^{k+1} - Q^i \rho_i^{k+1}$
    end for
  end for
  $Q^{k+1}, \rho_{k+1}^{k+1} \leftarrow \mathrm{Norm}_{\mathbb{S}}\left(Q^{k+1}\right)$
  return $Q^{k+1}, \rho_0^{k+1}, \ldots, \rho_{k+1}^{k+1}$

based on the TSQR algorithm that is optimal with respect to the communication overhead and preserves the stability of the modified Gram-Schmidt algorithm. We call this algorithm *localized Arnoldi*, as it proceeds a local orthogonalization procedure and then communicates the result with one reduction communication to obtain the global result. Finally, we present the result of a numerical test that compares all these methods.

9.1 Classical Gram-Schmidt with Re-Orthogonalization

The modified Gram-Schmidt method we used in Algorithm 5.2 needs $k + 1$ global reductions to compute the next orthogonal basis vector. Algorithm 9.1 shows the modified Gram-Schmidt method. It successively projects the new direction X^{k+1} onto the orthogonal compliments of the basis vectors Q^i. The classical Gram-Schmidt method, shown in Algorithm 9.2, does the same, but it computes the projection factors ρ_i^{k+1} in advance, such that the communication for that could be carried out together. Hence, it only needs two global communications – one for the orthogonalization and one for the normalization.

The numerical instabilities of the classical Gram-Schmidt method can be mitigated

Algorithm 9.3: Pipelined Gram-Schmidt Orthogonalization

Input: X^{k+1} and $\mathbb{S}$-orthonormal basis $\mathcal{Q} = (Q^i)_{i=0}^k$ and a parameter $1 \leq r \leq k$

```
Q^{k+1} = X^{k+1}
for i = 0, ..., r − 1 do
    ρ_i^{k+1} = ⟨Q^{k+1}, Q^i⟩_S
end for
for i = r, ..., k do
    ρ_i^{k+1} = ⟨Q^{k+1}, Q^i⟩_S
    Q^{k+1} ← Q^{k+1} − Q^{i−r} ρ_{i−r}^{k+1}
end for
for i = k − r, ..., k do
    Q^{k+1} ← Q^{k+1} − Q^i ρ_i^{k+1}
end for
Q^{k+1}, ρ_{i+1}^{k+1} ← Norm_S(Q^{k+1})
```

by repeating the orthogonalization [Hof89; Bjö94]. In Algorithm 9.2 this is implemented with the parameter $\#it$. Buhr et al. [Buh+14, Algorithm 1] proposed an algorithm that determines the number of re-orthogonalization adaptively. However, numerical experiments show that in most situations two iterations are sufficient.

9.2 Pipelined Gram-Schmidt

Especially for many right-hand sides s, the `BAXPY` and `BDOT` operations become equally expensive to or even more expensive than the application of the operator or preconditioner. In this case, it makes sense to overlap the reduction process with the computation of the `BAXPY` and `BDOT` kernels. Algorithm 9.3 shows a pipelined Gram-Schmidt method, that precomputes r inner block products before it starts to proceed the `BAXPY`s. Every reduction process of the block inner product is overlapped with r `BAXPY` operations. Figure 9.1 shows a schematic flow diagram of the methods. Arrows indicate the overlapping of communication and computation. Another advantage is that not only computation is overlapped, but also $r + 1$ global communications are computed in parallel. Hence, even in a communication dominated environment, we could expect a speedup.

The case $r = 0$ corresponds to the modified Gram-Schmidt process and the case $r = k$ corre-

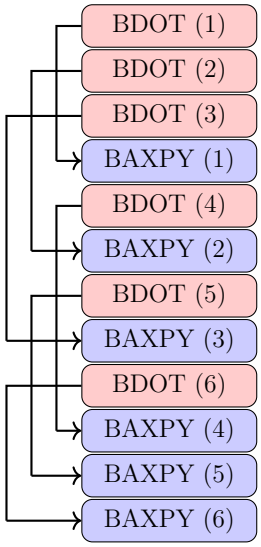


Figure 9.1: Schematic flow diagram of the pipelined Gram-Schmidt orthogonalization Algorithm 9.3 for $r = 2$. Arrows indicate overlap of communication and computation.

sponds to the classical Gram-Schmidt process. The algorithm is less stable for larger r. To find a good parameter r, one must estimate a trade-of between communication overlap and stability.

On unreliable networks, where the duration of the global communication is not predictable, one could adapt Algorithm 9.3 such that it checks in every iteration whether a global communication is complete. In that way, the parameter r adapts automatically to the network performance. The drawback is that this would lead to a non-deterministic behavior, therefore we do not pursuit this strategy further. This approach could be combined with the re-orthogonalization idea of the previous section to achieve stability.

Up to the author's knowledge, this algorithm was not presented before, albeit the basic idea is straight forward. The reason might be that this method only makes sense, if the `BAXPY` and `BDOT` operations consume a significant amount of time which is the case for BGMRes with a relatively sparse matrix or a lot of right-hand sides.

9.3 Localized Arnoldi

We now develop a orthogonalization method that is based on the recursive structure of the TSQR algorithm. Up to the authors knowledge, such algorithm was not presented before. In principle, it is also useful in the non-block case. For simplicity, we use the following notations:

- Vectors with elements in the *-subalgebra $\boldsymbol{\eta} \in \mathbb{S}^k$ are denoted by a small bold Greek letter.
- square brackets $[\ldots]$ are used to concatenate matrices or vectors, both horizontally and vertically.

Our new method relies on an extended form of the block QR decomposition. It assumes a distributed vector, i.e.

$$X = \begin{bmatrix} X_1 \\ \vdots \\ X_P \end{bmatrix},$$

where X_p is stored on processor p for $p = 1, \ldots, P$. Let us start with the definition of the $\mathbb{S}$-QR decomposition for multiple block vectors in the context of our block framework.

Definition 9.1 ($\mathbb{S}$-QR decomposition for multiple block vectors)**.** *Let $(X^i)_{i=0}^k \in \mathbb{R}^{n\times s}$, $(Q^i)_{i=0}^k \in \mathbb{R}^{n\times s}$ and $\boldsymbol{\rho}^0, \ldots, \boldsymbol{\rho}^k \in \mathbb{S}^{k+1}$, then a decomposition of the form*

$$\left[X^0 \ldots X^k\right] = \left[Q^0 \ldots Q^k\right] \left[\boldsymbol{\rho}^0 \ldots \boldsymbol{\rho}^k\right]$$

is called a $\mathbb{S}$-QR decomposition, if $\left[\boldsymbol{\rho}^0 \ldots \boldsymbol{\rho}^k\right]$ is upper triangular and Q satisfies

$$\langle Q^i, Q^j \rangle_{\mathbb{S}} = \delta_{ij}\mathbb{I} \qquad \forall i, j = 0, \ldots, k,$$

where δ_{ij} denotes the Kronecker symbol.

The next definition generalizes the $\mathbb{S}$-QR decomposition for the distributed setting.

Definition 9.2 (Distributed $\mathbb{S}$-QR decomposition)**.** *A decomposition of the form*

$$\begin{bmatrix} X_1^0 & \dots & X_1^k \\ \vdots & & \vdots \\ X_P^0 & \dots & X_P^k \end{bmatrix} = \begin{bmatrix} \left(\begin{bmatrix} P_1^0 \dots P_1^k \end{bmatrix} \begin{bmatrix} \zeta_1^0 \dots \zeta_1^k \end{bmatrix}\right) \\ \vdots \\ \left(\begin{bmatrix} P_P^0 \dots P_P^k \end{bmatrix} \begin{bmatrix} \zeta_P^0 \dots \zeta_P^k \end{bmatrix}\right) \end{bmatrix} \begin{bmatrix} \boldsymbol{\rho}^0 \dots \boldsymbol{\rho}^k \end{bmatrix}$$

is called a distributed $\mathbb{S}$-QR decomposition, if for all $l = 1, \dots, P$ holds

$$\langle P_l^i, P_l^j \rangle_{\mathbb{S}} = \delta_{ij} \mathbb{I} \tag{9.1}$$

and

$$\begin{bmatrix} \widetilde{\zeta}_1^0 & \dots & \widetilde{\zeta}_i^k \\ \vdots & & \vdots \\ \widetilde{\zeta}_P^0 & \dots & \widetilde{\zeta}_P^k \end{bmatrix} = \begin{bmatrix} \zeta_1^0 & \dots & \zeta_i^k \\ \vdots & & \vdots \\ \zeta_P^0 & \dots & \zeta_P^k \end{bmatrix} \begin{bmatrix} \boldsymbol{\rho}^0 \dots \boldsymbol{\rho}^k \end{bmatrix} \tag{9.2}$$

is a $\mathbb{S}$-QR decompostion with respect to the inner block product

$$\langle\!\langle \begin{bmatrix} \zeta_1 \\ \vdots \\ \zeta_P \end{bmatrix}, \begin{bmatrix} \boldsymbol{\tau}_1 \\ \vdots \\ \boldsymbol{\tau}_P \end{bmatrix} \rangle\!\rangle_{\mathbb{S}} := \sum_{i=1}^{P} \zeta_i^{\mathsf{T}} \boldsymbol{\tau}_i \in \mathbb{S}.$$

The vectors $\widetilde{\zeta}_p^0, \dots, \widetilde{\zeta}_p^k \in \mathbb{S}^{k+1}$ are called the local R-factors.

Remarks.

- The local parts might be rank-deficient, even if the global system $\begin{bmatrix} X^0 \dots X^k \end{bmatrix}$ has full rank. In particular, it might occur that the local R-factors $\begin{bmatrix} \widetilde{\zeta}_p^0 \dots \widetilde{\zeta}_p^k \end{bmatrix}$ are not invertible.
- Equation (9.1) needs a definition of the block inner product on the local part of the vector space. We use the definition

$$\langle P_l, Q_l \rangle_{\mathbb{S}} := \langle \begin{bmatrix} 0 \\ \vdots \\ 0 \\ P_l \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ \vdots \\ 0 \\ Q_l \\ 0 \\ \vdots \\ 0 \end{bmatrix} \rangle_{\mathbb{S}}. \tag{9.3}$$

However, as we will see, the only property that the local block inner product needs to satisfy for our theory is

$$\langle \begin{bmatrix} P_1 \\ \vdots \\ P_P \end{bmatrix}, \begin{bmatrix} Q_1 \\ \vdots \\ Q_P \end{bmatrix} \rangle_{\mathbb{S}} = \sum_{l=1}^{P} \langle P_l, Q_l \rangle_{\mathbb{S}},$$

which is true for definition (9.3) and the block inner products of Definition 3.10.

The next lemma shows that a distributed $\mathbb{S}$-QR decompostion is just a special case of the $\mathbb{S}$-QR decomposition.

Lemma 9.3. *A distributed $\mathbb{S}$-QR decomposition (Definition 9.2) is a $\mathbb{S}$-QR decomposition (Definition 9.1).*

Proof. We have to show that the columns of the matrix

$$\begin{bmatrix} \left(\begin{bmatrix} P_1^0 \dots P_1^k \end{bmatrix} \begin{bmatrix} \boldsymbol{\zeta}_1^0 \dots \boldsymbol{\zeta}_1^k \end{bmatrix}\right) \\ \vdots \\ \left(\begin{bmatrix} P_P^0 \dots P_P^k \end{bmatrix} \begin{bmatrix} \boldsymbol{\zeta}_P^0 \dots \boldsymbol{\zeta}_P^k \end{bmatrix}\right) \end{bmatrix}$$

are $\mathbb{S}$-orthonormal and that $\begin{bmatrix} \boldsymbol{\rho}^0 \dots \boldsymbol{\rho}^k \end{bmatrix}$ is upper triangular. The latter follows from the fact that (9.2) is a $\mathbb{S}$-QR decomposition. For the ith and jth column of the Q-factor it holds

$$\begin{aligned} \langle \begin{bmatrix} \begin{bmatrix} P_1^0 \dots P_1^k \end{bmatrix} \boldsymbol{\zeta}_1^i \\ \vdots \\ \begin{bmatrix} P_P^0 \dots P_P^k \end{bmatrix} \boldsymbol{\zeta}_P^i \end{bmatrix}, \begin{bmatrix} \begin{bmatrix} P_1^0 \dots P_1^k \end{bmatrix} \boldsymbol{\zeta}_1^j \\ \vdots \\ \begin{bmatrix} P_P^0 \dots P_P^k \end{bmatrix} \boldsymbol{\zeta}_P^j \end{bmatrix} \rangle_{\mathbb{S}} &= \sum_{p=1}^{P} \langle \begin{bmatrix} P_p^0 \dots P_p^k \end{bmatrix} \boldsymbol{\zeta}_p^i, \begin{bmatrix} P_p^0 \dots P_p^k \end{bmatrix} \boldsymbol{\zeta}_p^j \rangle_{\mathbb{S}} \\ &= \sum_{p=1}^{P} {\boldsymbol{\zeta}_p^i}^{\mathsf{T}} \boldsymbol{\zeta}_p^j \\ &= \langle\!\langle \boldsymbol{\zeta}^i, \boldsymbol{\zeta}^j \rangle\!\rangle_{\mathbb{S}} = \delta_{ij} \mathbb{I}, \end{aligned}$$

where we used that $\left(P_p^i\right)_{i=0}^k$ as well as $\left(\begin{bmatrix} \boldsymbol{\zeta}_1^i \\ \vdots \\ \boldsymbol{\zeta}_P^i \end{bmatrix} \right)_{i=0}^k$ are $\mathbb{S}$-orthonormal systems. □

As we want to apply this method in the Arnoldi iteration, we assume that we have given a distributed $\mathbb{S}$-QR decomposition and want to extend it by another vector X^{k+1}. Inspired by the TSQR algorithm presented in Section 7.3, the idea is that a distributed $\mathbb{S}$-QR decomposition can be extended by the direction X^{k+1} by computing a local $\mathbb{S}$-QR decomposition, followed by a global reduction of the local R-factors.

Let the Q-factor

$$\begin{bmatrix} \left(\begin{bmatrix} P_1^0 \dots P_1^k \end{bmatrix} \begin{bmatrix} \boldsymbol{\zeta}_1^0 \dots \boldsymbol{\zeta}_1^k \end{bmatrix}\right) \\ \vdots \\ \left(\begin{bmatrix} P_P^0 \dots P_P^k \end{bmatrix} \begin{bmatrix} \boldsymbol{\zeta}_P^0 \dots \boldsymbol{\zeta}_P^k \end{bmatrix}\right) \end{bmatrix}$$

of a distributed $\mathbb{S}$-QR decomposition and a distributed block vector X be given.

The aim of the *localized Arnoldi method* is to compute $P_1^{k+1}, \dots, P_P^{k+1}$, $\boldsymbol{\zeta}_1^{k+1}, \dots, \boldsymbol{\zeta}_P^{k+1}$ and $\boldsymbol{\rho}^{k+1}$ such that

$$\begin{bmatrix} X_1 \\ \vdots \\ X_P \end{bmatrix} = \begin{bmatrix} \left(\begin{bmatrix} P_1^0 \dots P_1^{k+1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\zeta}_1^0 & \dots & \boldsymbol{\zeta}_1^k & \boldsymbol{\zeta}_1^{k+1} \\ 0^\mathsf{T} & \dots & 0^\mathsf{T} & \end{bmatrix}\right) \\ \vdots \\ \left(\begin{bmatrix} P_P^0 \dots P_P^{k+1} \end{bmatrix} \begin{bmatrix} \boldsymbol{\zeta}_P^0 & \dots & \boldsymbol{\zeta}_P^k & \boldsymbol{\zeta}_P^{k+1} \\ 0^\mathsf{T} & \dots & 0^\mathsf{T} & \end{bmatrix}\right) \end{bmatrix} \boldsymbol{\rho}^{k+1}$$

and

$$\begin{aligned} \langle P_l^i, P_l^{k+1} \rangle_{\mathbb{S}} &= \delta_{i,k+1} \mathbb{I} \qquad \forall l = 1, \dots, P, \quad i = 0, \dots, k+1 \\ \langle\!\langle \boldsymbol{\zeta}^i, \boldsymbol{\zeta}^{k+1} \rangle\!\rangle_{\mathbb{S}} &= \delta_{i,k+1} \mathbb{I} \qquad \forall i = 0, \dots, k+1. \end{aligned}$$

The local Q-factors P_l^{k+1} are computed by applying any stable orthogonalization method locally. We use the modified Gram-Schmidt method in our examples. After that, the global $\mathbb{S}$-QR decomposition of the local R-factors must be computed. This can be achieved by either gather it on one master process, or perform it recursively in a reduction procedure. The latter is preferable on large scale machines. We describe the recursive procedure in the following.

The reduction procedure is performed on a tree on which every node stores its local orthogonal basis

$$\left(\begin{bmatrix} \boldsymbol{\zeta}_0^j \\ \boldsymbol{\zeta}_1^j \end{bmatrix}\right)_{j=0}^{k} \subset \mathbb{S}^{2k}.$$

In particular, every node hold a state that is extended from iteration to iteration. Hence, in every iteration of the Arnoldi method, the same tree must be used. At the beginning of every iteration, every process orthonormalizes its local part of the new block vector P_p^{k+1} to its local orthonormal basis. The local R-factor $\widetilde{\boldsymbol{\zeta}}_p^{k+1} \in \mathbb{S}^k$ is then send to its parent. During the reduction operation, every node in the reduction tree receives to vectors $\widetilde{\boldsymbol{\zeta}}_0^{k+1}, \widetilde{\boldsymbol{\zeta}}_1^{k+1}$, that are stacked on each other and then orthonormalized to its local basis. The resulting R-factor $\boldsymbol{\rho}^{k+1}$ is send to its parent. The root node does the same, but send its local orthonormal Q-factors $\boldsymbol{\zeta}_0^{k+1}, \boldsymbol{\zeta}_1^{k+1}$, back to its children. This initiates

Algorithm 9.4: Reduction Process for the Localized Arnoldi Method

Receive $\widetilde{\boldsymbol{\zeta}}_0^{k+1}$ and $\widetilde{\boldsymbol{\zeta}}_1^{k+1}$ from children
for $i = 0, \ldots, k$ **do**
$\left(\boldsymbol{\rho}^{k+1}\right)_i = \sum_{j=0}^{i} \left(\boldsymbol{\zeta}_0^i\right)_j^{\mathsf{T}} \left(\widetilde{\boldsymbol{\zeta}}_0^{k+1}\right)_j + \left(\boldsymbol{\zeta}_1^i\right)_j^{\mathsf{T}} \left(\widetilde{\boldsymbol{\zeta}}_1^{k+1}\right)_j$
$\widetilde{\boldsymbol{\zeta}}_0^{k+1} \leftarrow \widetilde{\boldsymbol{\zeta}}_0^{k+1} - \begin{bmatrix} \boldsymbol{\zeta}_0^i \\ 0 \end{bmatrix} \left(\boldsymbol{\rho}^{k+1}\right)_i$
$\widetilde{\boldsymbol{\zeta}}_1^{k+1} \leftarrow \widetilde{\boldsymbol{\zeta}}_1^{k+1} - \begin{bmatrix} \boldsymbol{\zeta}_1^i \\ 0 \end{bmatrix} \left(\boldsymbol{\rho}^{k+1}\right)_i$
end for
$\begin{bmatrix} \boldsymbol{\zeta}_0^{k+1} \\ \boldsymbol{\zeta}_1^{k+1} \end{bmatrix} \left(\boldsymbol{\rho}^{k+1}\right)_{k+1} = \begin{bmatrix} \widetilde{\boldsymbol{\zeta}}_0^{k+1} \\ \widetilde{\boldsymbol{\zeta}}_1^{k+1} \end{bmatrix}$ ▷ Normalization
if not root **then**
Send $\boldsymbol{\rho}^{k+1}$ to parent
else
Send $\boldsymbol{\zeta}_0^{k+1}$ and $\boldsymbol{\zeta}_1^{k+1}$ back to children to start back-propagation
broadcast $\boldsymbol{\rho}^{k+1}$ to all processes
end if

Algorithm 9.5: Back-Propagation for the Localized Arnoldi Method

Receive $\widetilde{\boldsymbol{\rho}}^{k+1}$ from parent
$\bar{\boldsymbol{\zeta}}_0^{k+1} = \sum_{i=0}^{k+1} \begin{bmatrix} \boldsymbol{\zeta}_0^i \\ \mathbf{0} \end{bmatrix} \left(\widetilde{\boldsymbol{\rho}}^{k+1}\right)_i$
$\bar{\boldsymbol{\zeta}}_1^{k+1} = \sum_{i=0}^{k+1} \begin{bmatrix} \boldsymbol{\zeta}_1^i \\ \mathbf{0} \end{bmatrix} \left(\widetilde{\boldsymbol{\rho}}^{k+1}\right)_i$
if not leaf **then**
Send $\bar{\boldsymbol{\zeta}}_0^{k+1}$ and $\bar{\boldsymbol{\zeta}}_1^{k+1}$ back to children
end if

the back-propagation. The resulting R-factor of the root is the global one. Every node in the tree receives the Q-factor from its parent and multiplies it with its local Q-factors. The result is then back-propagates to the children. On the leafs the globally orthogonal basis can be obtained by multiplying the received Q-factor to the locally orthogonal part of the block vector P_p^{k+1}.

Algorithm 9.4 and Algorithm 9.5 show the algorithms for the reduction and back-propagation procedure. Figure 9.2 and Figure 9.3 show the reduction and back-propagation operations, respectively. Unfortunately, the current MPI standard does not allow to define a custom back-propagation method. Therefore, we implemented the reduction pattern using Point-to-Point communications. The used tree can be either a binary tree using the MPI rank number, or can be deduced from a `MPI_Allreduce` pattern, where we use a custom reduction function to deduce the tree. A prototype implementation can be found in the gitlab repository of the author[1].

[1] `https://zivgitlab.uni-muenster.de/n_drei02/tsqr_communication_pattern`

Algorithm 9.6: Localized Arnoldi Method

$\widetilde{\boldsymbol{\zeta}}_p^0, P_p^0 = \mathrm{Norm}_{\mathbb{S}}\left(X_p^0\right)$ ▷ local

Compute local and global R-factor $\boldsymbol{\zeta}_p^0, \boldsymbol{\rho}^0 \in \mathbb{S}^1$ from $\widetilde{\boldsymbol{\zeta}}^0$ using Alg. 9.4 and 9.5

for $k = 0, \ldots, k_{\max}$ **do**

$X_p^k = \sum_{j=0}^{k} P_p^j \left(\boldsymbol{\zeta}_p^k\right)_j$ ▷ assemble global orth. vector

$P^{k+1} = AX^k$ ▷ requires neighbor communication

for $j = 0, \ldots, i$ **do** ▷ local modified Gram-Schmidt

$\left(\widetilde{\boldsymbol{\zeta}}_p^{k+1}\right)_j = \langle P_p^j, P_p^{k+1} \rangle_{\mathbb{S}}$

$P_p^{k+1} \leftarrow P_p^{k+1} - P_p^j \left(\widetilde{\boldsymbol{\zeta}}_p^{k+1}\right)_j$

end for

$\widetilde{\boldsymbol{\zeta}}_p^{k+1}, P_p^{k+1} \leftarrow \mathrm{Norm}_{\mathbb{S}}\left(P_p^{k+1}\right)$

Compute global R-factors $\boldsymbol{\zeta}_j^{k+1}, \boldsymbol{\rho}^{k+1} \in \mathbb{S}^{k+2}$ from $\widetilde{\boldsymbol{\zeta}}^{k+1}$ using Alg. 9.4 and 9.5

end for

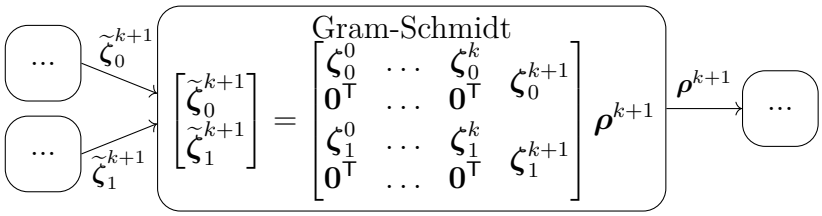


Figure 9.2: Reduction operation of the localized Arnoldi method.

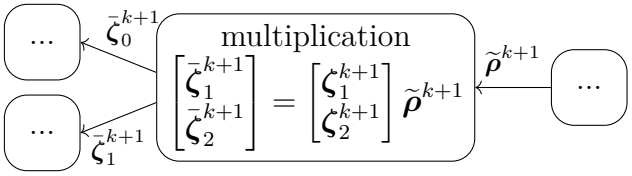


Figure 9.3: Back-propagation operation of the localized Arnoldi method.

Table 9.1: Comparison of the arithmetical complexity, number of messages and stability for different orthogonalization methods in the BGMRes method.

orthogonalization	arith. complexity	messages	stable
classical	$4kp^2q\frac{n}{P}$	$\mathcal{O}(\log(P))$	no
classical(2)	$8kp^2q\frac{n}{P}$	$\mathcal{O}(\log(P))$	yes
modified	$4kp^2q\frac{n}{P}$	$\mathcal{O}(k\log(P))$	yes
pipelined(r) hybrid	$4kp^2q\frac{n}{P}$	$\mathcal{O}(k\log(P))$ (overl.)	depend on r
localized	$6kp^2q\frac{n}{P} + \mathcal{O}\left(\log(P)k^2p^2q\right)$	$\mathcal{O}(\log(P))$	yes

Algorithm 9.6 shows the localized Arnoldi method. It uses the modified Gram-Schmidt method to compute the local orthogonalization.

One small drawback of the methods can be found in the reduction process. As it can be seen in Algorithms 9.4 and 9.5, the arithmetical complexity increases quadratically with the size k of the basis. This leads to an effort of $\mathcal{O}\left(k^2\right)$ for the reduction operation. This term could become a bottleneck for large k. Furthermore, the method computes $2(k+1)$ `BAXPY` and $k+1$ `BDOT` operations in the kth iteration. This leads to an overall effort of $6kp^2q + \mathcal{O}\left(\log(P)k^2p^2q\right)$.

Table 9.1 compares the localized Arnoldi method with the other methods developed in this chapter. Note that the localized Arnoldi strategy only proceeds one reduction pattern and the back-propagation, while the classical orthogonalization method proceeds two reductions and the broadcast of the result. From the theoretical values in Table 9.1, the localized method is competitive to the classical(2) method. However, the methods differ in the arithmetical complexity and the number of messages, as it only needs one reduction and back-propagation instead of four reductions and four broadcasts. For a BGMRes method with fewer iterations, or a low restart parameter, the localized method would perform better, while for large k the classical(2) orthogonalization method or the pipelined version would perform better.

9.4 Numerical Experiments

In this chapter, we introduced several orthogonalization methods for the BGMRes method. All resulting BGMRes methods are mathematically equivalent but differ in the numerical stability and communication efforts.

To compare all these methods, we run the same test problem from Figure 5.1, which is the `Simon/raefsky3` matrix. The result is shown in Figure 9.4. We used a block size of $p = 4$ and restart the BGMRes method after 100 iterations. For a fixed number of processors P all methods need almost the same number of iterations. Hence, stability is not an issue here. As in the previous benchmark, we can observe the same three regimes. In the sequential case, all methods take almost the same time. Just the classical(2) method is slightly slower as it proceeds the orthogonalization procedure twice, which introduces overhead. Running the test on a medium number of processors shows a

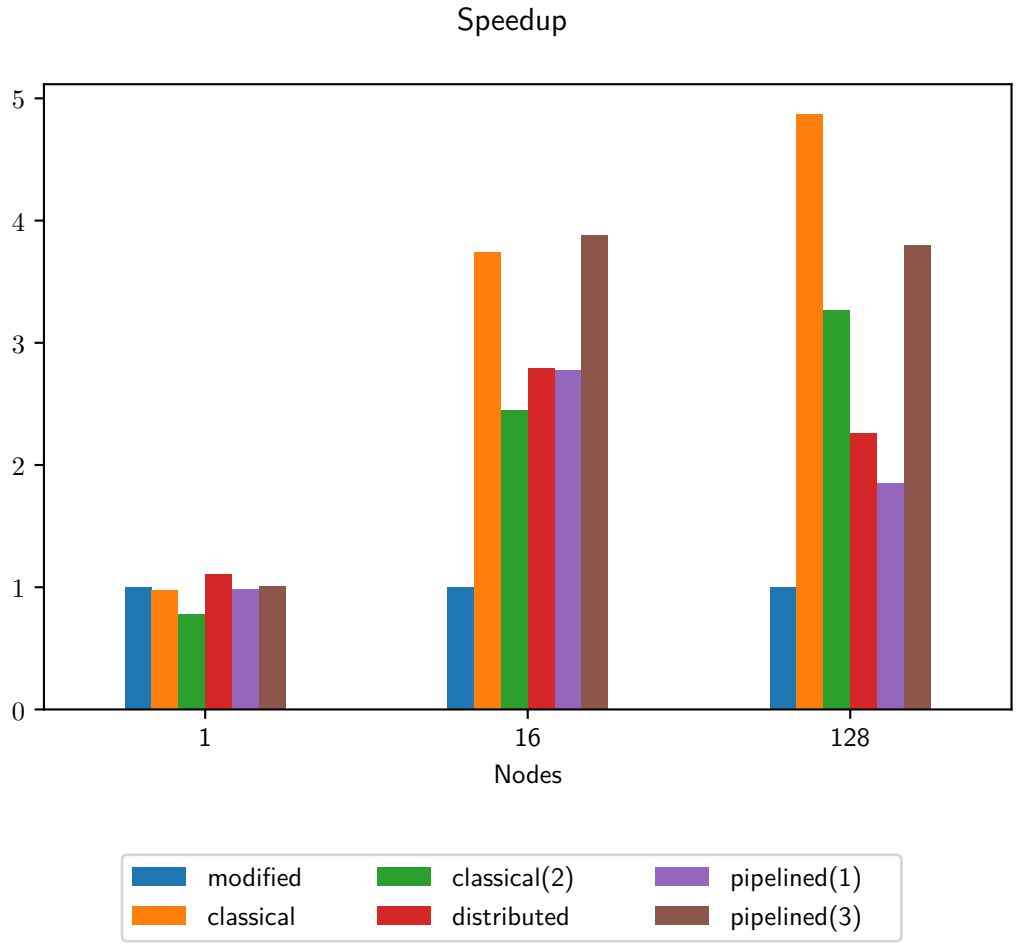


Figure 9.4: Speedup of runtime for different BGMRes variants compared to the modified method. Colors decode the orthogonalization method.

significant speedup of all the communication-optimized methods. On $P = 16$ nodes, the pipelined(3) and classical methods are fastest. They reach a speedup of almost $4\times$ over the modified Gram-Schmidt method. The other methods, classical(2), localized Arnoldi and pipelined(1) only reach a speedup of almost $3\times$.

On the scaling limit ($P = 128$) the classical method is clearly the fastest, followed by the pipelined(3) method. The classical(2) method performs significantly better than the localized methods in this regime. This could be due to the suboptimal scaling of the self-implemented reduction operation or the k^2 term in the reduction operation. Furthermore, the reduction pattern is not optimized for network hardware yet. We assume that the performance of the localized method could be significantly improved if this communication pattern would be supported by the MPI implementation. A better implementation and quantification of the performance is an objective of future work.

Overall, we see that the optimization of the orthogonalization strategy can have significant impact on the performance of the BGMRes method.

10
Pipelined Block BiCGStab Method

In this chapter, we apply the same techniques to the BBiCGStab method, as we used for the BCG method in Chapter 8. As the introduction of further recursions introduces sources of numerical instability, we do not pursuit it very aggressively. Numerical experiments showed that the mitigation of these instabilities is not trivial and is therefore left for further work.

A study of the BBiCGStab method (Algorithm 6.3) shows that it is already in a good state to overlap the communication with the preconditioner application without changing a lot. In every iteration two applications of the operator and two applications of the preconditioner are performed. Global communication is needed to compute the coefficients λ^k, β^k and ω^k. The global communication for λ^k can already be overlapped with the computation of $\hat{Z}^k = M^{-1}\hat{Q}^k$. Furthermore, the computation of ω^k and β^k can be fused. To overlap that communication, we introduce a new variable $\hat{W}^k = M^{-1}\hat{U}^k$ that can be computed during the communication, and then be used to update $\hat{V}^k$ recursively, i.e.

$$\hat{V}^k = M^{-1}\hat{R}^k = \hat{T}^k - \hat{\omega}^k\hat{W}^k.$$

Together with this communication, we can also compute the product $\langle \hat{S}^k, \hat{S}^k \rangle_{\mathbb{S}}$, which is then used to determine the residual norm for the break criterion and for the decision, whether we need to re-orthogonalize the residual in the next iteration. In that way, we overlap all communication with the two preconditioner applications. The communication optimized variant of the BBiCGStab method can be found in Algorithm 10.1. The arrows indicate which operations can overlap.

A further optimization would be to precompute the operator application, such that it can also be computed during the communication. An attempt by the author failed due to instabilities. The analysis and mitigation of these instabilities is an objective of

Algorithm 10.1: Pipelined BBiCGStab with Adaptive Residual Re-Orthonormalization

$R^0 = B - AX^0$
if $\eta > 0$ **then**
  $\hat{R}^0, \sigma^0 = \mathrm{Norm}_{\mathbb{S}}\left(R^0\right)$
else
  $\sigma^0 = \mathbb{I}$
end if
$\hat{P}^0 = M^{-1}\hat{R}^0$
$\chi^0 = \langle \hat{R}^0, \hat{R}^0 \rangle_{\mathbb{S}}$
Choose $M^{-\mathsf{T}}\widetilde{R}^0$ (e.g. $M^{-\mathsf{T}}\widetilde{R}^0 = \hat{P}^0$)
$\hat{V}^0 = \hat{P}^0$
for $k = 0, \ldots$ **do**
  $\hat{Q}^k = A\hat{P}^k$
  $\hat{\lambda}^k = \left(\langle M^{-\mathsf{T}}\widetilde{R}^0, \hat{Q}^k \rangle_{\mathbb{S}}\right)^{-1} \langle M^{-\mathsf{T}}\widetilde{R}^0, \hat{R}^k \rangle_{\mathbb{S}}$
  $\hat{Z}^k = M^{-1}\hat{Q}^k$
  $\hat{S}^k = \hat{R}^k - \hat{Q}^k\hat{\lambda}^k$
  $X^{k+\frac{1}{2}} = X^k + \hat{P}\lambda^k\sigma^k$
  if $\eta\kappa_D\left(\chi^k\right) > \sqrt{\varepsilon_{\mathrm{mach}}}$ **then**
    $\hat{S}^k, \gamma^k \leftarrow \mathrm{Norm}_{\mathbb{S}}\left(\hat{S}^k\right)$
    $\sigma^{k+1} = \gamma^k\sigma^k$
    $\hat{T} = M^{-1}\hat{S}$
  else
    $\hat{T}^k = \hat{V}^k - \hat{Z}^k\hat{\lambda}^k$
  end if
  $\chi^{k+1} = \langle \hat{S}^k, \hat{S}^k \rangle_{\mathbb{S}}$
  $\hat{U}^k = A\hat{T}^k$
  $\hat{\omega}^k = \frac{\langle \hat{U}^k, \hat{S}^k \rangle_F}{\langle \hat{U}^k, \hat{U}^k \rangle_F}$
  $\hat{\beta}^k = -\left(\langle M^{-\mathsf{T}}\widetilde{R}^0, \hat{Q}^k \rangle_{\mathbb{S}}\right)^{-1} \langle M^{-\mathsf{T}}\widetilde{R}^0, \hat{U}^k \rangle_{\mathbb{S}}$
  $\hat{W}^k = M^{-1}U^k$
  break if $\|\chi^{k+1}\|$
  $X^{k+1} = X^{k+\frac{1}{2}} + \hat{\omega}^k\hat{T}\sigma^k$
  $\hat{R}^{k+1} = \hat{S}^k - \hat{\omega}^k\hat{U}^k$
  $\hat{V}^{k+1} = \hat{T}^k - \hat{\omega}\hat{W}^k$
  $\hat{P}^{k+1} = \hat{V}^{k+1} + \left(\hat{P}^k - \hat{\omega}^k\hat{Z}^k\right)\hat{\beta}^k$
end for

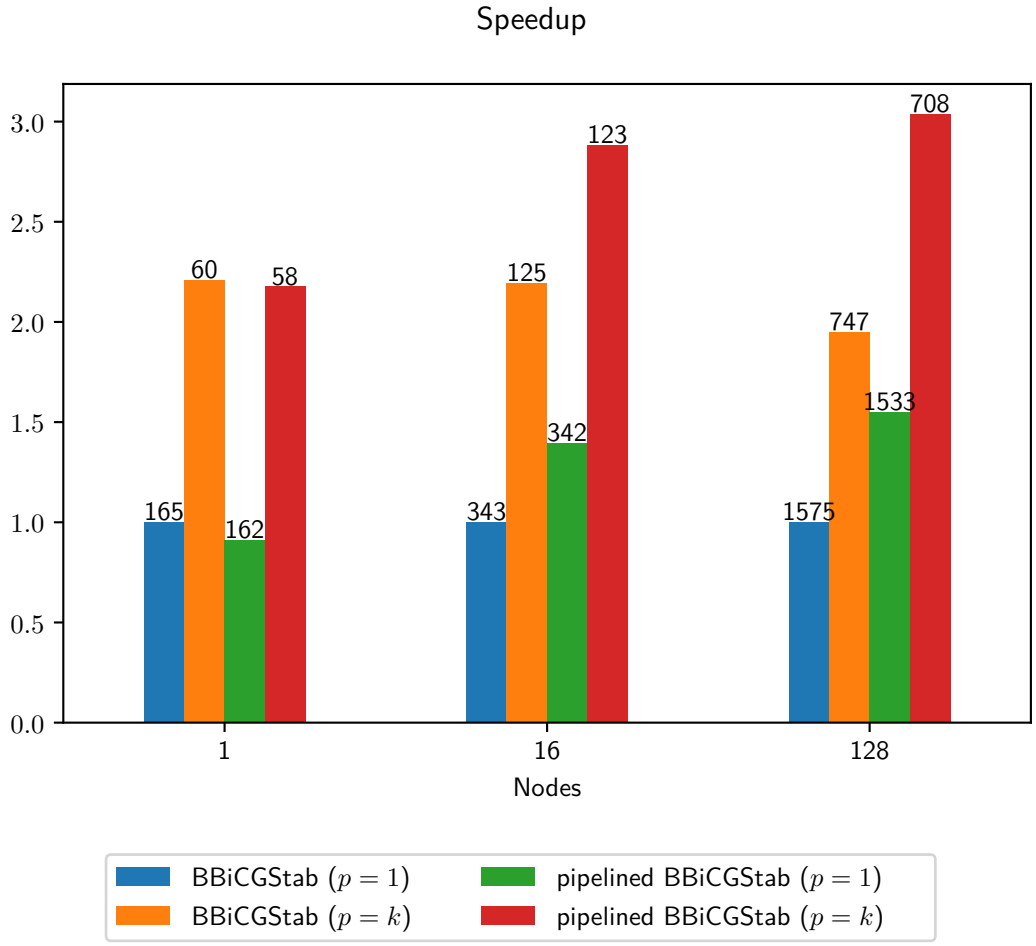


Figure 10.1: Speedup of runtime for different BiCGStab variants. Numbers indicate the number of iterations that are needed to reduce the residual by a factor of 10^{-7}.

further work. It could be based on the work of Cools and Vanroose [CV17b; Coo19], who analyzed the round-off errors for the pipelined non-block BiCGStab method.

10.1 Numerical Experiments

Although we do not spend a lot of effort in the optimization of the BBiCGStab method, we want to quantify the performance in the parallel case. We compare the BBiCGStab method (Algorithm 6.3) with its optimized version (Algorithm 10.1). For that, we use $s = 4$ right-hand sides and compare the results for the parallel ($\mathbb{S}_P$) and block ($\mathbb{S}_B$) method. The test problem is the same as in Section 6.3, i.e. the `raefsky3` matrix. We use a ILU(0) preconditioner with an additive Schwarz decomposition for the parallelization.

Figure 10.1 shows the speedup of the different versions compared to the parallel BBiCGStab method. The numbers at the top of the bars show the number of iterations the method needs to reduce the residual by a factor of 10^{-7}. As expected in the sequential case the block method is faster than the parallel method, but the optimization does

not yield any remarkable benefits. We see that the pipelined version is slightly slower in this regime as it introduces arithmetical overhead. However, the optimization of the algorithm does not affect the stability. The pipelined BBiCGStab method actually needs slightly fewer iterations than its not optimized counterparts.

In the parallel cases, the optimized versions are superior to the non-optimized ones. They are approximately 50% faster. This is due to the fewer global synchronizations they perform. The increase of iterations on larger nodes is due to the weaker preconditioning, as the domain is decomposed into more Schwarz domains. Also in this case, the pipelined BBiCGStab method needs fewer iterations than its BBiCGStab counterpart.

11
Summary and Outlook

Nothing in life is to be feared, it is only to be understood.

Marie Curie

Finally, we summarize the achievements of this thesis and give an outlook for future work and ideas how to transfer some of our approaches to other contexts.

In Chapter 3 we reviewed the block Krylov framework by Frommer, Szyld and Lund [FSL17; FLS19] and presented the novel block-global method, which turned out to be irrelevant for practice as we have seen later. We analyzed the framework and its building blocks concerning the performance on modern hardware. In particular, we considered the applicability of SIMD instructions and their arithmetical intensity. The advantage of the block Krylov framework over classical block Krylov methods is that the blocking overhead could be balanced even for a fixed large number of right-hand sides. We saw that the vector update and inner product kernels perform with constant time per right-hand side up to a certain blocking parameter. This means that the faster convergence rate comes for free in this setting.

In the following chapters, we formulated the block variants of the CG, GMRes and BiCGStab methods based on that block Krylov framework. For the block CG method, we provided a convergence analysis which gave insights into the behavior of the different block Krylov variants. In particular, we saw why the novel block-global method is inferior to the block-parallel method. Furthermore, we introduced a novel stabilization strategy which stabilizes the method and avoids the process of deflation. Deflation, which is applied in a lot of other works in the literature, is improper in our context, as we stick to the number of lanes given by the SIMD interface. For the stabilization strategy, we decided to orthonormalize the residual with respect to the euclidean block inner product. Dubrulle [Dub01] suggested in his work to orthonormalize with respect to the

inner product that is given by the preconditioner. This would simplify the algorithm but disqualifies the Householder algorithm for orthonormalization. The analysis and implementation of this approach is left for future work. All the theoretical findings about the convergence rate and the benefits of the stabilization strategy are supported by numerical tests.

In Chapter 5, we addressed the block GMRes method. Like for the block CG method, we formulated the method in the context of the block Krylov framework. To do so, we used a generalization of the Givens rotations to triangulate the Hessenberg matrix. In a numerical experiment, we observed that the convergence rates of the different block Krylov variants are similarly connected as for the CG method.

As a last block Krylov method, we considered the block BiCGStab method in Chapter 6. We provided a formulation based on orthogonal polynomials in the context of the block Krylov framework. Furthermore, we applied a similar residual re-orthonormalization strategy as for the block CG method. Numerical tests showed that also for the block BiCGStab method, the block-global method performs inferior. In future work, it would be interesting to investigate whether the stabilization coefficient, which we have chosen as a scalar in our method, could be chosen as an element of the *-subalgebra $\mathbb{S}$. Besides that, higher level stabilization BiCGStab methods like presented by Saito et al. [STI14] could be applied in our context.

The second part of the thesis is about the optimization of the block Krylov methods with respect to communication in distributed memory systems. For that, we discussed the conditions and challenges in distributed systems and created a benchmark to measure the amount of time that can be used for computation while collective communication is active. Furthermore, we reviewed the TSQR algorithm which is a key building block for block Krylov methods on distributed systems.

In the following chapter, we presented five variants of the block CG method with different properties regarding communication by applying the approaches of Ghysels and Vanroose [GV14] and Gropp [Gro10]. In the numerical tests on a medium size supercomputer, we observed three regimes in that the methods behave differently. This coincides with our theoretical expectations. In the communication dominated regime, we achieved good speedups compared to the default block CG method.

To optimize the block GMRes method with respect to communication, we considered, in contrast to many approaches in the literature, the orthogonalization method in the Arnoldi process. This makes sense in our context, as the block vector update and the inner block product are rather expensive so that overlapping it with computation already suffice to hide the whole communication costs. Moreover, we presented a novel method for the orthogonalization process that is based on the TSQR algorithm and allows an orthogonalization with only one global reduction communication. It would be interesting to examine how this method could be combined with other communication-avoiding approaches, like s-step GMRes. Another interesting subject would be to use this method for example in the MINRes method. In principle, the applicability of the

method is not restricted to block Krylov methods. In the numerical experiments, we compared the different orthogonalization methods and observed significant speed up compared to the standard block GMRes method.

We reviewed shortly the amenability of pipelining techniques to the block BiCGStab method in the last chapter of the thesis. We found that some approaches can easily be applied but others introduce too much numerical instability such that we did not further pursuit them. Thus, this would be an interesting subject for further work. However, the approaches and optimization that we have applied already yield a pretty good speedup in our numerical experiments.

In summary, we presented tailored methods to solve large sparse systems on modern super computing hardware. We proved their advantages in numerical tests on both, the node level and on a large distributed system. The author aims for the integration of the methods into the Dune-ISTL module and intends to present a merge request soon after submitting this thesis to make the results of the thesis available to the community.

In the future, we want to provide a better comparison of the new stabilization methods with deflation strategies, both experimentally and analytically. We hope that this could give better insights for choosing the re-orthonormalization parameter. Moreover, we want to improve the implementation of the reduction and back-propagation communication pattern and make it available for the community, as it could be a generic building block for more methods like the localized Arnoldi method.

In addition, we want to investigate how we could make the methods usable for a broader range of problems. One idea is to apply them in ODE solvers and compute multiple time steps simultaneously. This would lead formally to an approach like in parallel-in-time methods, which are currently very popular in the scientific computing research community. However, the gain of parallelism could also be used to apply block Krylov methods instead of distributing it over the nodes. This would not only decrease the inter-node communication, but also improve the other aspects discussed in this work.

Bibliography

[Al 18] Hussam Al Daas. "Solving linear systems arising from reservoirs modeling". Theses. Inria Paris ; Sorbonne Université, UPMC University of Paris 6, Laboratoire Jacques-Louis Lions, Dec. 2018. URL: https://hal.inria.fr/tel-01984047 (cit. on p. 15).

[Al +18] Hussam Al Daas, Laura Grigori, Pascal Hénon and Philippe Ricoux. "Enlarged GMRES for solving linear systems with one or multiple right-hand sides". In: *IMA Journal of Numerical Analysis* (2018) (cit. on p. 15).

[Kry31] Aleksey Nikolaevich Krylov. "On the numerical solution of the equation by which in technical questions frequencies of small oscillations of material systems are determined". In: *Izvestija AN SSSR (News of Academy of Sciences of the USSR), Otdel. mat. i estest. nauk* 7.4 (1931), pp. 491–539 (cit. on p. 6).

[And+99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney and D. Sorensen. *LAPACK Users' Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999 (cit. on p. 25).

[Arn+20] Daniel Arndt, Wolfgang Bangerth, Bruno Blais, Thomas C. Clevenger, Marc Fehling, Alexander V. Grayver, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Peter Munch, Jean-Paul Pelteret, Reza Rastak, Ignacio Thomas, Bruno Turcksin, Zhuoran Wang and David Wells. "The deal.II Library, Version 9.2". In: *submitted* (2020) (cit. on p. 4).

[Arn51] Walter Edwin Arnoldi. "The principle of minimized iterations in the solution of the matrix eigenvalue problem". In: *Quarterly of Applied Mathematics* 9.1 (1951), pp. 17–29. URL: https://hal.archives-ouvertes.fr/hal-01712943 (cit. on pp. 7, 46).

[Ash+12] Thomas J. Ashby, Pieter Ghysels, Wim Heirman and Wim Vanroose. "The Impact of Global Communication Latency at Extreme Scales on Krylov Methods". In: *Algorithms and Architectures for Parallel Processing*. Ed. by Yang Xiang, Ivan Stojmenovic, Bernady O. Apduhan, Guojun Wang, Koji Nakano and Albert Zomaya. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 428–442 (cit. on p. 81).

[Bal+20] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Alp Dener, Victor Eijkhout, William D. Gropp, Dmitry Karpeyev, Dinesh Kaushik, Matthew G. Knepley, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Todd Munson, Karl Rupp, Patrick Sanan, Barry F. Smith, Stefano Zampini, Hong Zhang and Hong Zhang. *PETSc Web page*. 2020. URL: https://www.mcs.anl.gov/petsc (cit. on p. 4).

[Bal+97] Satish Balay, William D. Gropp, Lois Curfman McInnes and Barry F. Smith. "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries". In: *Modern Software Tools in Scientific Computing*. Ed. by E. Arge, A. M. Bruaset and H. P. Langtangen. Birkhäuser Press, 1997, pp. 163–202 (cit. on p. 4).

[Bas+08a] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, R. Kornhuber, M. Ohlberger and O. Sander. "A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part II: Implementation and Tests in DUNE". In: *Computing* 82.2–3 (2008), pp. 121–138. URL: http://dx.doi.org/10.1007/s00607-008-0004-9 (cit. on pp. 4, 23).

[Bas+08b] P. Bastian, M. Blatt, A. Dedner, C. Engwer, R. Klöfkorn, M. Ohlberger and O. Sander. "A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part I: Abstract Framework". In: *Computing* 82.2–3 (2008), pp. 103–119. URL: http://dx.doi.org/10.1007/s00607-008-0003-x (cit. on pp. 4, 23).

[Bas+20a] Peter Bastian, Mirco Altenbernd, Nils-Arne Dreier, Christian Engwer, Jorrit Fahlke, René Fritze, Markus Geveler, Dominik Göddeke, Oleg Iliev, Olaf Ippisch, Jan Mohring, Steffen Müthing, Mario Ohlberger, Dirk Ribbrock, Nikolay Shegunov and Stefan Turek. "Exa-Dune—Flexible PDE Solvers, Numerical Methods and Applications". In: *Software for Exascale Computing: Some Remarks on the Priority Program SPPEXA*. Springer International Publishing, 2020, pp. 225–269 (cit. on p. 83).

[Bas+20b] Peter Bastian, Markus Blatt, Andreas Dedner, Nils-Arne Dreier, Christian Engwer, René Fritze, Carsten Gräser, Christoph Grüninger, Dominic Kempf, Robert Klöfkorn, et al. "The DUNE framework: basic concepts and recent developments". In: *Computers & Mathematics with Applications* (2020) (cit. on pp. 4, 23).

[Bha+12] Amit Bhaya, Pierre-Alexandre Bliman, Guilherme Niedu and Fernando Pazos. "A cooperative conjugate gradient method for linear systems permitting multithread implementation of low complexity". In: *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)* (Dec. 2012) (cit. on p. 15).

[Bjö94] Åke Björck. "Numerics of Gram-Schmidt orthogonalization". In: *Linear Algebra and Its Applications* 197 (1994), pp. 297–316 (cit. on pp. 47, 87).

[BB07] M. Blatt and P. Bastian. "The Iterative Solver Template Library". In: *Applied Parallel Computing – State of the Art in Scientific Computing*. Ed. by B. Kagström, E. Elmroth, J. Dongarra and J. Wasniewski. Berlin/Heidelberg: Springer, 2007, pp. 666–675 (cit. on p. 25).

[Bla+16] M. Blatt, A. Burchardt, A. Dedner, Ch. Engwer, J. Fahlke, B. Flemisch, Ch. Gersbacher, C. Gräser, F. Gruber, Ch. Grüninger, D. Kempf, R. Klöfkorn, T. Malkmus, S. Müthing, M. Nolte, M. Piatkowski and O. Sander. "The Distributed and Unified Numerics Environment, Version 2.4". In: *Archive of Numerical Software* 4.100 (2016), pp. 13–29. URL: http://dx.doi.org/10.11588/ans.2016.100.26526 (cit. on pp. 4, 23).

[Bos14] Siegfried Bosch. *Lineare Algebra*. Springer-Verlag, 2014 (cit. on p. 16).

[Buh+14] Andreas Buhr, Christian Engwer, Mario Ohlberger and Stephan Rave. "A Numerically Stable A Posteriori Error Estimator for Reduced Basis Approximations of Elliptic Equations." In: *11th World Congress on Computational Mechanics, WCCM 2014, 5th European Conference on Computational Mechanics, ECCM 2014 and 6th European Conference on Computational Fluid Dynamics, ECFD 2014:1407.8005*. Ed. by E. Onate XO and Huerta A. CIMNE, Barcelona, 2014, pp. 4094–4102 (cit. on pp. 47, 87).

[Car15] Erin Claire Carson. "Communication-avoiding Krylov subspace methods in theory and practice". PhD thesis. UC Berkeley, 2015 (cit. on pp. 3, 66).

[CM69] Daniel Chazan and Willard Miranker. "Chaotic relaxation". In: *Linear algebra and its applications* 2.2 (1969), pp. 199–222 (cit. on p. 67).

[CG89] Anthony T. Chronopoulos and Charles W. Gear. "s-Step iterative methods for symmetric linear systems". In: *Journal of Computational and Applied Mathematics* 25.2 (1989), pp. 153–168 (cit. on p. 66).

[CK90] Anthony T. Chronopoulos and Sung-han Kim. "s-Step Orthomin and GMRES implemented on parallel computers". In: 90/43R (1990) (cit. on p. 66).

[CK10] Anthony T. Chronopoulos and Andrey B. Kucherov. "Block s-step Krylov iterative methods". In: *Numerical Linear Algebra with Applications* 17.1 (2010), pp. 3–15 (cit. on p. 4).

[Coo19] Siegfried Cools. "Analyzing and improving maximal attainable accuracy in the communication hiding pipelined BiCGStab method". In: *Parallel Computing* 86 (2019), pp. 16–35 (cit. on p. 99).

[Coo+19] Siegfried Cools, Jeffrey Cornelis, Pieter Ghysels and Wim Vanroose. "Improving strong scaling of the conjugate gradient method for solving large linear systems using global reduction pipelining". In: *arXiv preprint arXiv:1905.06850* (2019) (cit. on p. 83).

[CCV19] Siegfried Cools, Jeffrey Cornelis and Wim Vanroose. "Numerically stable recurrence relations for the communication hiding pipelined conjugate gradient method". In: *IEEE Transactions on Parallel and Distributed Systems* (2019) (cit. on pp. 3, 81).

[CV17a] Siegfried Cools and Wim Vanroose. "Numerically Stable Variants of the Communication-hiding Pipelined Conjugate Gradients Algorithm for the Parallel Solution of Large Scale Symmetric Linear Systems". In: *arXiv preprint arXiv:1706.05988* (2017) (cit. on pp. 3, 5, 81).

[CV17b] Siegfried Cools and Wim Vanroose. "The communication-hiding pipelined BiCGStab method for the parallel solution of large unsymmetric linear systems". In: *Parallel Computing* 65 (2017), pp. 1–20 (cit. on p. 99).

[Coo+18] Siegfried Cools, Emrullah Fatih Yetkin, Emmanuel Agullo, Luc Giraud and Wim Vanroose. "Analyzing the effect of local rounding error propagation on the maximal attainable accuracy of the pipelined Conjugate Gradient method". In: *SIAM Journal on Matrix Analysis and Applications* 39.1 (2018), pp. 426–450 (cit. on pp. 3, 81).

[CCV18] Jeffrey Cornelis, Siegfried Cools and Wim Vanroose. "The communication-hiding conjugate gradient method with deep pipelines". In: *arXiv preprint arXiv:1801.04728* (2018) (cit. on p. 83).

[DH11] Timothy A. Davis and Yifan Hu. "The University of Florida Sparse Matrix Collection". In: *ACM Trans. Math. Softw.* 38.1 (Dec. 2011). URL: https://doi.org/10.1145/2049662.2049663 (cit. on pp. 42, 52, 62).

[Dem+08] James Demmel, Laura Grigori, Mark Hoemmen and Julien Langou. "Communication-avoiding parallel and sequential QR factorizations". In: *CoRR abs/0806.2159* (2008) (cit. on pp. 3, 71).

[Dem+12] James Demmel, Laura Grigori, Mark Hoemmen and Julien Langou. "Communication-optimal parallel and sequential QR and LU factorizations". In: *SIAM Journal on Scientific Computing* 34.1 (2012), pp. 206–239 (cit. on pp. 3, 71).

[DE20] Nils-Arne Dreier and Christian Engwer. "Strategies for the vectorized Block Conjugate Gradients method". In: *Numerical Mathematics and Advanced Applications-ENUMATH 2019*. Vol. 139. to appear. Springer, 2020 (cit. on p. 30).

[Dub01] Augustin A. Dubrulle. “Retooling the method of block conjugate gradients”. In: *Electronic Transactions on Numerical Analysis* 12 (2001), pp. 216–233. URL: http://etna.mcs.kent.edu/vol.12.2001/pp216-233.dir/pp216-233.pdf (cit. on pp. 31, 38, 101).

[EES83] Stanley C. Eisenstat, Howard C. Elman and Martin H. Schultz. “Variational iterative methods for nonsymmetric systems of linear equations”. In: *SIAM Journal on Numerical Analysis* 20.2 (1983), pp. 345–357 (cit. on p. 11).

[EJS03] A. El Guennouni, K. Jbilou and H. Sadok. “A block version of BiCGSTAB for linear systems with multiple right-hand sides”. In: *Electronic Transactions on Numerical Analysis* 16.129-142 (2003), p. 2 (cit. on pp. 16, 54).

[Ell19] Paul R. Eller. “Scalable non-blocking Krylov solvers for extreme-scale computing”. PhD thesis. University of Illinois at Urbana-Champaign, 2019 (cit. on p. 66).

[FY02] Robert D. Falgout and Ulrike Meier Yang. “hypre: A library of high performance preconditioners”. In: *International Conference on Computational Science*. Springer. 2002, pp. 632–641 (cit. on p. 4).

[Fog] Agner Fog. *VCL C++ vector class library*. URL: https://www.agner.org/optimize/vectorclass.pdf (cit. on p. 24).

[FN91] Roland W. Freund and Noël M. Nachtigal. “QMR: a quasi-minimal residual method for non-Hermitian linear systems”. In: *Numerische Mathematik* 60.1 (1991), pp. 315–339 (cit. on p. 11).

[FLS19] Andreas Frommer, Kathryn Lund and Daniel B. Szyld. *Block Krylov subspace methods for functions of matrices II: Modified block FOM*. Tech. rep. MATHICSE, EPFL, 2019 (cit. on pp. 3, 15, 19, 101).

[FS00] Andreas Frommer and Daniel B. Szyld. “On asynchronous iterations”. In: *Journal of computational and applied mathematics* 123.1-2 (2000), pp. 201–216 (cit. on p. 67).

[FSL17] Andreas Frommer, Daniel B. Szyld and Kathryn Lund. “Block Krylov subspace methods for functions of matrices”. In: *Electron. Trans. Numer. Anal.* 47 (2017), pp. 100–126. URL: http://etna.math.kent.edu/vol.47.2017/pp100-126.dir/pp100-126.pdf (cit. on pp. 3, 4, 15, 19, 20, 101).

[Ghy+13] Pieter Ghysels, Thomas J. Ashby, Karl Meerbergen and Wim Vanroose. “Hiding global communication latency in the GMRES algorithm on massively parallel machines”. In: *SIAM journal on scientific computing* 35.1 (2013), pp. C48–C71 (cit. on p. 85).

[GV14] Pieter Ghysels and Wim Vanroose. “Hiding global synchronization latency in the preconditioned Conjugate Gradient algorithm”. In: *Parallel Computing* 40.7 (2014), pp. 224–238. URL: https://hdl.handle.net/10067/1096860151162165141 (cit. on pp. 81, 102).

[Gir+05] Luc Giraud, Julien Langou, Miroslav Rozložník and Jasper van den Eshof. “Rounding error analysis of the classical Gram-Schmidt orthogonalization process”. In: *Numerische Mathematik* 101.1 (2005), pp. 87–100 (cit. on p. 85).

[GO89] Gene H. Golub and Dianne P. O’Leary. “Some history of the conjugate gradient and Lanczos algorithms: 1948–1976”. In: *SIAM review* 31.1 (1989), pp. 50–102 (cit. on p. 6).

[Gre97] Anne Greenbaum. *Iterative methods for solving linear systems*. Vol. 17. SIAM, 1997 (cit. on pp. 6, 8).

[GPS96] Anne Greenbaum, Vlastimil Pták and Zdenvek Strakoš. “Any nonincreasing convergence curve is possible for GMRES”. In: *SIAM journal on matrix analysis and applications* 17.3 (1996), pp. 465–469 (cit. on p. 49).

[GT17] Laura Grigori and Olivier Tissot. “Reducing the communication and computational costs of Enlarged Krylov subspaces Conjugate Gradient”. In: (2017). URL: https://hal.inria.fr/hal-01451199 (cit. on pp. 4, 15, 67).

[Gro10] W. Gropp. *Update on libraries for Blue Waters*. 2010 (cit. on pp. 78, 102).

[Gut07] Martin H. Gutknecht. “Block Krylov space methods for linear systems with multiple right-hand sides. An introduction”. en. In: *Modern mathematical models, methods and algorithms for real world systems*. Ed. by Andul Hasan Siddiqi, Iain S. Duff and Ole Christensen. . Tunbridge Wells: Anshan, 2007, pp. 420–447 (cit. on pp. 17, 46).

[GS09] Martin H. Gutknecht and Thomas Schmelzer. “The block grade of a block Krylov space”. In: *Linear Algebra and its Applications* 430.1 (2009), pp. 174–185 (cit. on p. 17).

[Hac94] Wolfgang Hackbusch. *Iterative solution of large sparse systems of equations*. Vol. 95. Springer, 1994 (cit. on p. 6).

[HS+52] Magnus R. Hestenes, Eduard Stiefel, et al. “Methods of conjugate gradients for solving linear systems”. In: *Journal of research of the National Bureau of Standards* 49.6 (1952), pp. 409–436 (cit. on pp. 6, 7, 11).

[Hoe10] Mark Hoemmen. “Communication-avoiding Krylov subspace methods”. PhD thesis. UC Berkeley, 2010 (cit. on pp. 4, 66, 85).

[Hof89] Walter Hoffmann. “Iterative algorithms for Gram-Schmidt orthogonalization”. In: *Computing* 41.4 (1989), pp. 335–348 (cit. on p. 87).

[JMS99] Khalide Jbilou, Abderrahim Messaoudi and Hassane Sadok. "Global FOM and GMRES algorithms for matrix equations". In: *Applied Numerical Mathematics* 31.1 (1999), pp. 49–63 (cit. on p. 19).

[Kre15] Matthias Kretz. "Extending C++ for explicit data-parallel programming via SIMD vector types". PhD thesis. 2015, p. 256 (cit. on p. 24).

[KL12] Matthias Kretz and Volker Lindenstruth. "Vc: A C++ library for explicit vectorization". In: *Software: Practice and Experience* 42.11 (2012), pp. 1409–1430. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.1149 (cit. on p. 24).

[KS20] Marie Kubínová and Kirk M. Soodhalter. "Admissible and attainable convergence behavior of block Arnoldi and GMRES". In: *SIAM Journal on Matrix Analysis and Applications* 41.2 (2020), pp. 464–486 (cit. on pp. 46, 49, 52).

[Lan50] Cornelius Lanczos. "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators". In: *Journal of Research of the National Bureau of Standards* 45.4 (1950), pp. 255–282 (cit. on p. 6).

[Lan03] Julien Langou. "Iterative methods for solving linear systems with multiple right-hand sides". PhD thesis. INSA Toulouse, 2003 (cit. on p. 38).

[Lan10] Julien Langou. "Computing the R of the QR factorization of tall and skinny matrices using MPI_Reduce". In: *arXiv preprint arXiv:1002.4250* (2010) (cit. on p. 73).

[Law+02] William Lawry, Christopher Wilson, Arthur B. Maccabe and Ron Brightwell. "COMB: A portable benchmark suite for assessing MPI overlap". In: *Proceedings. IEEE International Conference on Cluster Computing.* IEEE. 2002, pp. 472–475 (cit. on pp. 40, 43, 68).

[Lun18] Kathryn Lund. "A new block Krylov subspace framework with applications to functions of matrices acting on multiple vectors". PhD thesis. Temple University, 2018 (cit. on pp. 3, 15, 21).

[Mou14] Sophie Moufawad. "Enlarged Krylov Subspace Methods and Preconditioners for Avoiding Communication". PhD thesis. Université Pierre et Marie Curie, 2014 (cit. on p. 15).

[NY95] A. A. Nikishin and A. Yu. Yeremin. "Variable block CG algorithms for solving large sparse symmetric positive definite linear systems on parallel computers. I. General iterative scheme". In: *SIAM J. Matrix Anal. Appl.* 16.4 (1995), pp. 1135–1153 (cit. on p. 15).

[OLe80] Dianne P. O'Leary. "The block conjugate gradient algorithm and related methods". In: *Linear Algebra and its Applications* (1980), pp. 293–322 (cit. on pp. 14, 16, 20, 31, 37, 38, 54, 55).

[PS75] Christopher C. Paige and Michael A. Saunders. "Solution of sparse indefinite systems of linear equations". In: *SIAM journal on numerical analysis* 12.4 (1975), pp. 617–629 (cit. on pp. 7, 11).

[Ruh79] Axel Ruhe. "Implementation aspects of band Lanczos algorithms for computation of eigenvalues of large sparse symmetric matrices". In: *Mathematics of Computation* 33.146 (1979), pp. 680–687 (cit. on p. 46).

[SS86] Youcef Saad and Martin H. Schultz. "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems". In: *SIAM J. SCI. STAT. COMPUT* 7.3 (1986), pp. 856–869 (cit. on pp. 7, 11, 46).

[Saa03] Yousef Saad. *Iterative methods for sparse linear systems*. Vol. 82. SIAM, 2003 (cit. on pp. 6, 8, 46).

[STI14] Shusaku Saito, Hiroto Tadano and Akira Imakura. "Development of the Block BiCGSTAB(ℓ) method for solving linear systems with multiple right hand sides". In: *JSIAM Letters* 6 (2014), pp. 65–68 (cit. on pp. 59, 102).

[Sch97] Joachim Schöberl. "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules". In: *Computing and visualization in science* 1.1 (1997), pp. 41–52 (cit. on p. 4).

[SG96] V. Simoncini and E. Gallopoulos. "Convergence properties of block GMRES and matrix polynomials". In: *Linear Algebra and its Applications* 247 (1996), pp. 97–119 (cit. on pp. 49, 51, 52).

[Soo15] Kirk M. Soodhalter. "A block MINRES algorithm based on the band Lanczos method". In: *Numerical Algorithms* 69.3 (2015), pp. 473–494 (cit. on pp. 19, 46).

[Spi16] Nicole Spillane. "An adaptive multipreconditioned conjugate gradient algorithm". In: *SIAM journal on Scientific Computing* 38.3 (2016), A1896–A1918 (cit. on p. 4).

[Spi+14] Nicole Spillane, Victorita Dolean, Patrice Hauret, Frédéric Nataf, Clemens Pechstein and Robert Scheichl. "Abstract robust coarse spaces for systems of PDEs via generalized eigenproblems in the overlaps". In: *Numerische Mathematik* 126.4 (2014), pp. 741–770 (cit. on p. 11).

[SW02] Andreas Stathopoulos and Kesheng Wu. "A block orthogonalization procedure with constant synchronization requirements". In: *SIAM Journal on Scientific Computing* 23.6 (2002), pp. 2165–2182 (cit. on p. 73).

[Ste08] G. W. Stewart. "Block Gram–Schmidt orthogonalization". In: *SIAM Journal on Scientific Computing* 31.1 (2008), pp. 761–775 (cit. on p. 27).

[Sti55] Edvard Stiefel. “Relaxationsmethoden bester Strategie zur Lösung linearer Gleichungssysteme”. In: *Commentarii Mathematici Helvetici* 29.1 (1955), pp. 157–179 (cit. on p. 11).

[Tea] The Trilinos Project Team. *The Trilinos Project Website* (cit. on p. 4).

[Tis19] Olivier Tissot. “Iterative methods for solving linear systems on massively parallel architectures”. PhD thesis. Sorbonne Université, 2019 (cit. on p. 15).

[TOP500] *Top 500 Supercomputing Sites*. July 2020. URL: `https://top500.org` (cit. on p. 2).

[TB97] Lloyd N. Trefethen and David Bau. *Numerical linear algebra*. Vol. 50. SIAM, 1997 (cit. on pp. 6, 8).

[THW10] J. Treibig, G. Hager and G. Wellein. “LIKWID: A lightweight performance-oriented tool suite for x86 multicore environments”. In: *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*. San Diego CA, 2010 (cit. on p. 27).

[Und75] Richard Underwood. *An iterative block Lanczos method for the solution of large sparse symmetric eigenproblems*. Tech. rep. 1975 (cit. on p. 14).

[Van92] Henk A. Van der Vorst. “Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems”. In: *SIAM Journal on scientific and Statistical Computing* 13.2 (1992), pp. 631–644 (cit. on pp. 11, 54).

[Vit90] Brigitte Vital. “Etude de quelques methodes de resolution de problemes lineaires de grande taille sur multiprocesseur”. PhD thesis. Universite de Rennes I, 1990 (cit. on p. 46).

[Wit+13] Markus Wittmann, Georg Hager, Thomas Zeiser and Gerhard Wellein. “Asynchronous MPI for the Masses”. In: (2013). arXiv: `1302.4280 [cs.DC]` (cit. on p. 70).

[ZZ13] Jianhua Zhang and Jing Zhao. “A novel class of block methods based on the block AA T-Lanczos bi-orthogonalization process for matrix equations”. In: *International Journal of Computer Mathematics* 90.2 (2013), pp. 341–359 (cit. on p. 46).