universität wien

Bachelorarbeit

Implementation and experimental comparison between the COMMUNICATION AVOIDING-GENERALIZED MINIMAL RESIDUAL METHOD and standard GMRES

Verfasser
Robert Ernstbrunner

angestrebter akademischer Grad
Bachelor of Science (BSc)

Wien, 2019

Studienkennzahl lt. Studienblatt:	A 033 521
Fachrichtung:	Informatik - Scientific Computing
Betreuerin / Betreuer:	Univ.-Prof. Dipl.-Ing. Dr. Wilfried Gansterer, M.Sc.

Contents

Notation

Similar notation is considered as in Hoemmen et al. [8] and Grigori et al. [7].

Linear Algebra

- Greek letters denote scalars, lower case Roman letters denote vectors (or - based on the context - dimensions), capital Roman letters denote matrices.
- Capital letters with two subscripts, e.g. '$V_{m,n}$', denote matrices with m rows and n columns.
- Capital *Black letter* letters (e.g. V, Q and R in Black letters are represented by $\mathfrak{V}$, $\mathfrak{Q}$ and $\mathfrak{R}$ resp.) denote matrices that are composed out of other matrices.
- v_k denotes the k^{th} vector in a series of vectors v_0, v_1, …, , v_k, v_{k+1}, … of equal length.
- Similarly, V_k denotes the k^{th} matrix in a sequence of matrices V_0, V_1, … , V_k, V_{k+1}, …. Generally all these matrices have the same number of rows. They may or may not have the same number of columns.
- If V is a matrix consisting of s vectors $[v_1, v_2, \dots, v_s]$, then $\underline{V} = [V, v_{s+1}]$. The underline denotes one more column at the end.
- If again, V is a matrix consisting of s vectors $[v_1, v_2, \dots, v_s]$, then $\acute{V} = [v_2, v_3, \dots, v_s]$. The acute denotes one column less at the beginning.
- As a consequence $\underline{\acute{V}}$ denotes one more column at the end and one less column at the beginning, e.g. $\underline{\acute{V}} = [\acute{V}, v_{s+1}]$.
- Depending on the context, both underline or/and acute letters can also refer to rows as well.
- $0_{m,n}$ is defined as an $m \times n$ matrix consisting of zeros, I_m denotes the $m \times m$ Identity matrix and e_k denotes the k^{th} canonical vector with the dimension depending on the context.
- All matrices and vectors are assumed to be real, if not stated otherwise.
- Matlab notation is used for addressing elements of matrices and vectors. For example, given a matrix A of size $n \times n$ and two sets of indices α and β, $A(\alpha, :)$ is a submatrix formed by the subset of the rows of A whose indices belong to α. Similarly, $A(\alpha, \beta)$ is a submatrix formed by the subset of the rows of A whose indices belong to α and the subset of the columns of A whose indices belong to β.

Terms and definitions

- **Fetching**: the movement of data. This could either be *reading*, *copying* or *sending* and *receiving* messages.
- **Ghosting**: the storage of redundant data that does not belong to a processors assigned domain.

Graph Notation

- $G(A)$ denotes the directed graph of A with $G(A) = \{V, E\}$ where $V(G(A))$ is the set of vertices of $G(A)$ and $E(G(A))$ is the set of edges of $G(A)$.
- $R(G(A), \alpha)$ denotes the set of vertices in $G(A)$ that are reachable from any vertex in the set α, including α.
- $R(G(A), \alpha, m)$ denotes the set of vertices in $G(A)$ that are reachable by paths of length at most m from any vertex in α, including α.

Abbreviations

- **MPK**: Matrix powers kernel
- **TSQR**: Tall and skinny QR factorization
- **CGS**: Classical Gram-Schmidt method
- **MGS**: Modified Gram-Schmidt method
- **BCGS**: Block Classical Gram-Schmidt method
- **MKL**: Intel Math Kernel Library
- **SPD**: symmetric positive definite

Abstract

In this thesis the communication avoiding GMRES method named CA-GMRES of Hoemmen et al. [*Communication-avoiding Krylov Subspace Methods. PhD thesis*, Berkeley, CA, USA, 2010] is examined. CA-GMRES is a Krylov Subspace method based on so called s-step methods and enables the solving of large sparse (nonsymmetric) linear systems.
Its main advantage over known standard GMRES implementations is the reduction of communication by a factor s both sequentially and in parallel, while at the same time being able to converge at the same rate as standard GMRES. It does so by allowing the basis and the restart length to be separately chosen, which adds numerical stability and improves convergence compared to other s-step methods. This often leads to significant performance gains.
The CA-GMRES method was implemented in a shared-memory environment and then compared to a standard parallel version. Moreover, the ILU(0) preconditioner was incorporated and its behavior was analyzed in order to produce additional numerical and performance results that further support the work and findings of Hoemmen et al..

1. Introduction

The costs of an algorithm consist of arithmetic computations and communication. Compared to arithmetic, communication costs are much higher and the widening CPU-memory performance gap and increased parallelism in hardware environments further necessitate the construction of communication-avoiding algorithms.
The term *communication* generally denotes the movement of data either between different processors in the parallel case or between 'fast' and 'slow' memory in the sequential case, where 'fast' and 'slow' are relative to the two levels examined in the memory hierarchy (e.g., cache and DRAM, or DRAM and disk). Communication optimal algorithms do not eliminate communication completely, but they are constructed in a way that prioritizes the reduction of communication. This often results in new challenges. E.g., in the case of CA-GMRES (Section 5), the data dependencies between the sparse matrix-vector multiplies and dot products in the Arnoldi process are broken up, which further necessitates the incorporation of additional techniques and algorithms in order to deal with numerical instabilities and ill-conditioned basis vectors.

Krylov subspace methods are iterative algorithms that can be used for solving large, sparse linear systems either in real or complex arithmetic. Many scientific computations spend much of their time in Krylov methods. Therefore, it only makes sense to try to improve these methods by reducing their communication cost. Krylov methods can be 'one-sided' or 'two-sided', have long recurrences or short recurrences, and may work for specific matrix types only (e.g., the popular Conjugate Gradient method is one-sided, has short recurrences and works for symmetric matrices only). For a more detailed description of Krylov properties see Hoemmen et al. [8]. The Generalized minimal residual method (GMRES) is a one-sided Krylov subspace method that uses long recurrences and works for general matrices. The goal of the communication-avoiding version, namely CA-GMRES, is to take s steps for the same communication cost as 1 step, which would be optimal.

The rest of this thesis is organized as follows. The second section introduces GMRES and the basic principles behind the GMRES method. The third section explains the most important kernels in the communication-avoiding version of the GMRES method. Section four gives insight into the s-step based Arnoldi process of Hoemmen et al. [8]. Section five presents the communication-avoiding GMRES method, explains how a preconditioner can be applied and addresses implementation details. Finally, sections six and seven show results for the numerical resp. performance experiments and section eight summarizes and concludes on these results.

2. GMRES

The Generalized Minimal Residual Method (GMRES) was first introduced by Saad et al. [13] and is an iterative Krylov subspace method for solving large sparse linear systems. The GMRES method starts with an initial approximate solution x_0 and initial residual $r_0 = b - Ax_0$ and finds a correction z_k at iteration k which solves the least-squares problem

$$z_k := \text{argmin}_z \|b - A(x_0 + z)\|_2 \tag{2.1}$$

where z_k is determined in the Krylov subspace

$$\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}.$$

The solution at iteration k is then formed by $x_k = x_0 + z_k$. Since $\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$ is usually ill-conditioned the Arnoldi method is incorporated to produce $k+1$ orthonormal basis vectors $\underline{Q} = [q_1, q_2, \ldots, q_k, q_{k+1}]$ with $q_1 = r_0 / \|r_0\|_2$ and a $k{+}1{\times}k$ upper Hessenberg coefficient matrix $\underline{H}$ where

$$AQ = \underline{Q}\underline{H}.$$

With these conditions z_k can be defined as $z := Qy$ such that

$$\begin{aligned} \text{argmin}_z \|b - A(x_0 + z)\|_2 &= \text{argmin}_y \|r_0 - AQy\|_2 \\ &= \text{argmin}_y \|r_0 - \underline{Q}\underline{H}y\|_2. \end{aligned}$$

Since $q_1 = r_0 / \|r_0\|_2$ and $\underline{Q}$ is orthonormal, one has

$$\begin{aligned} \text{argmin}_y \|r_0 - \underline{Q}\underline{H}y\|_2 &= \text{argmin}_y \|\underline{Q}^T r_0 - \underline{H}y\|_2 \\ &= \text{argmin}_y \|\beta e_1 - \underline{H}y\|_2 \end{aligned} \tag{2.2}$$

with $\beta = \|r_0\|_2$. $\underline{H}$ is then factored into $\underline{H} = \underline{GU}$ with square matrix $\underline{G}$ being a product of k Givens rotations, $\underline{U} = \begin{pmatrix} U \\ 0_{1,k} \end{pmatrix}$ and U being upper triangular. Also, $\underline{g}$ is defined by

$\underline{g} := \beta \underline{G}^T e_1 = \begin{pmatrix} g \\ \gamma \end{pmatrix}$, where g is a subvector and γ is the last entry in $\underline{g}$. The triangular system to solve is then given by

$$y_k := \text{argmin}_y \|g - Uy\|_2$$

The solution at iteration k is obtained by computing $x_k = x_0 + Qy_k$. Note, that the absolute value of γ is $\|b - Ax_k\|_2$, the absolute residual at iteration k.

3. Computational kernels

In this thesis *computational kernels* define parts of an algorithm with significantly high costs, relatively speaking. These costs include both arithmetic operations and communication. The following kernels make up the essential building blocks in Arnoldi(s, t) (see Section 4) and eventually CA-GMRES (see Section 5).

3.1. Matrix powers kernel

The matrix powers kernel, as described by Hoemmen et al. in [8], was not implemented in the context of this thesis. However, it is an essential part to avoid communication and therefore, will be briefly summarized here.

In its basic form, the MPK takes an $n \times n$ matrix A and a starting vector v_1 as input and produces s more vectors $Av, A^2v, \ldots, A^sv$. Since A is usually large and sparse, it makes sense to look at the graph of A as well, namely $G(A)$ in order to apply known graph algorithms. In s-step methods, the MPK replaces the sparse matrix-vector products that generate the basis for the Krylov subspace $\mathcal{K}_{s+1}(A, v) = [v, Av, A^2v, \ldots, A^sv] = [v_1, v_2, \ldots, v_{s+1}]$. One invocation of the MPK produces the same amount of basis vectors as s sparse matrix-vector products. The MPK sends a factor of $\Theta(s)$ fewer messages than s SpMV invocations and the matrix has to be read from slow to fast memory only once. In order to achieve this, the data and the workload are distributed among P processors, where each processor is assigned a part α of $A(\alpha, :)$ and $v_1(\alpha)$ with $\alpha \subseteq V(G(A))$. Then, each processor fetches $A(\eta, :)$ and $v_1(\eta)$, with $\eta = R(G(A), \alpha, s) - \alpha$ in order to compute s more vectors $v_2(\alpha), v_3(\alpha), \ldots, v_{s+1}(\alpha)$ without communication. In other words, to compute $v_2(\alpha)$, the superset $\beta = R(G(A), \alpha, 1)$ is required. To compute $v_3(\alpha)$, the set $R(G(A), \beta, 1) = R(G(A), \alpha, 2)$ must be available. In general, to compute $v_{s+1}(\alpha)$, the set $R(G(A), \alpha, s)$ must be at hand. Since

$$\alpha \subseteq R(G(A), \alpha, 1) \subseteq \ldots \subseteq R(G(A), \alpha, s-1) \subseteq R(G(A), \alpha, s)$$

it is clear that larger steps eventually lead to increasing amounts of ghosted data and floating point operations.

3.1.1. Preconditioned matrix powers kernel

Iterative Krylov methods often require a preconditioner that, when applied, fundamentally changes the MPK. In order to avoid communication, highly parallelizable preconditioners come to mind. E.g., Nuentsa et al. [11] present their parallel GMRES with a multiplicative Schwarz preconditioner. Grigori et al. [7] developed CA-ILU(0), a very interesting type of preconditioner that, at first glance, seems unfit for a parallel and communication-avoiding environment. Section 5.3.1 summarizes their work.

3.2. Tall and skinny QR

TSQR is a QR decomposition algorithm especially suited for $m \times n$ matrices, where $m \gg n$. TSQR uses a divide-and-conquer approach and, therefore, works on a reduction tree structure. The highest form of parallelism is achieved if TSQR uses a binary tree. In the purely sequential case a linear (flat) tree comes into play. Hybrid algorithms use anything in between and the best tree structure may depend on the matrix size and underlying architecture as Demmel et al. explain in [4]. The parallel TSQR with a binary tree is summarized below. For a more detailed description, as well as a description of the sequential algorithm, see Demmel et al. [4].

Parallel TSQR First, the matrix A is split up into P parts with each submatrix having size $m/P \times n$. TSQR on a binary tree then passes $(\log_2 P) + 1$ stages where any fast and accurate QR factorization can be applied for each stage. Let's assume $P = 4$, then

$$A = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix}.$$

At stage zero the QR factorization for each submatrix A_i is computed, with

$$A_0 = Q_{00}R_{00}, \qquad A_1 = Q_{10}R_{10}, \qquad A_2 = Q_{20}R_{20} \quad \text{and} \quad A_3 = Q_{30}R_{30}.$$

The successive stage merges the R-factors and computes the next QR factorizations

$$\begin{pmatrix} R_{00} \\ R_{10} \end{pmatrix} = Q_{01}R_{01} \qquad \text{and} \qquad \begin{pmatrix} R_{20} \\ R_{30} \end{pmatrix} = Q_{11}R_{11}.$$

This procedure is repeated until one last QR factorization is performed where the final R factor can be interpreted as the root of the tree. This results in the following decomposition

$$A = \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{pmatrix} = \left(\begin{array}{c|c|c|c} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline & & Q_{20} & \\ \hline & & & Q_{30} \end{array}\right) \cdot \left(\begin{array}{c|c} Q_{01} & \\ \hline & Q_{11} \end{array}\right) \cdot Q_{02} \cdot R_{02}. \tag{3.1}$$

Figure 1 shows that this approach only requires $\mathcal{O}(\log_2 P)$ messages on P processors (a factor of $\Theta(s)$ fewer messages than Householder QR or MGS). [4] show that, sequentially, the matrix is read only once, saving a factor of $\Theta(s)$ transferred data between levels of the memory hierarchy (compared to Householder QR or MGS). Of course, (3.1) is stored

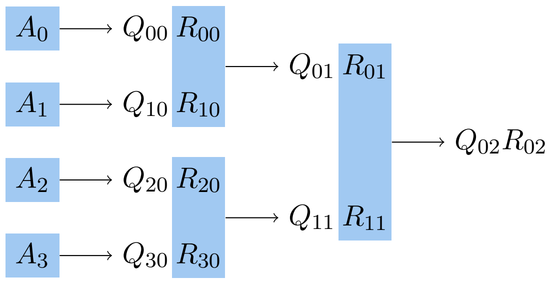


Figure 1: Parallel TSQR on a binary tree of four processors. The first subscript of the Q and R matrices indicates the sequence number for a stage and the second subscript is the stage number. The blue boxes represent the processors involved at each stage.

implicitly to save storage space.
The orthogonality of the Q factor computed by Classical or Modified Gram-Schmidt depends on the condition number of A. Householder QR, on the other hand, does not make any assumptions on $\mathcal{K}(A)$, it is *unconditionally* stable. Therefore, Householder QR is a good choice for the local QR factorizations in TSQR, which makes TSQR inherently unconditionally stable as well.

For reasons explained in Section 5.1 one might want TSQR to produce an R factor with real nonnegative diagonal entries. Demmel et al. show in [3] how to modify the usual Householder QR in a numerically stable way so that such a factor is generated. This modified Householder QR factorization can then be incorporated in order to let TSQR as well produce an R factor with real nonnegative diagonal entries.

3.3. Block Gram-Schmidt

The Gram-Schmidt process takes a set of s linearly independent basis vectors $V = [v_1, \dots, v_s]$ and creates an orthonormal basis that spans the same subspace as V. Unlike unblocked Gram-Schmidt methods, blocked Gram-Schmidt algorithms work on blocks of columns at a time instead of one column at a time. If the matrix consists of s columns, this usually requires a factor of $\Theta(s)$ fewer messages and a factor of $\Theta(s)$ fewer data transfers between levels of the memory hierarchy.
In their performance analysis with a simplified parallel model, Hoemmen et al. [8] also show that blocked classical Gram-Schmidt is superior to a blocked modified Gram-Schmidt variant in terms of the messages sent between processors. However, they state that BCGS and BMGS contain similar accuracy properties as their unblocked versions. Therefore, BCGS is not as numerically stable as the blocked MGS variant. The modified Gram-Schmidt method is often used for basis orthogonalization in the Arnoldi iteration (Section 4). Greenbaum et al. showed in [6], that it is the linear independence of the Arnoldi basis, not the orthogonality near machine precision, that is important when solving linear systems with GMRES. Therefore, the CA-GMRES algorithm (Section 5) can

make use of the classical Gram-Schmidt approach that is presented below.

BCGS with TSQR Algorithm 1 shows a version of BCGS that incorporates TSQR in order to improve vector orthogonalization. BCGS orthogonalizes the s basis vectors in V_k against all previous basis vectors in Q by computing

$$V'_k := (I - QQ^T)V_k = V_k - Q(Q^T V_k). \tag{3.2}$$

TSQR then orthogonalizes the s basis vectors with respect to each other. Combined, these two kernels do the work of updating a QR factorization with new columns. The advantage of TSQR and BCGS over unblocked MGS is that they move asymptotically less data between levels of the memory hierarchy. Unlike MGS, BCGS consists almost entirely of dense matrix-matrix operations. Also, TSQR improves orthogonality of the block columns. Therefore, if Algorithm 1 is used for solving linear systems, reorthogonalization can be omitted entirely.
Since TSQR stores its Q matrices implicitly, they must be extracted for the highly optimized dense matrix-matrix operations of BCGS. This step involves additional communication but should be still faster than directly performing the local QR-like computations that are required for the implicit representation.

Algorithm 1 BCGS with TSQR

Input: $V = [V_1, V_2, \ldots, V_M]$ where V is $n \times m$. Each V_k is $n \times m_k$, with $\sum_{k=1}^{M} m_k = m$.
Output: $Q = [Q_1, \ldots, Q_M]$, where $\mathcal{R}(Q) = \mathcal{R}(V)$ and $\mathcal{R}([Q_1, \ldots, Q_k]) = \mathcal{R}([V_1, \ldots, V_k])$
Output: $R : m \times m$ upper triangular matrix.

```
1: for k = 1 to $M$ do
2:     $R_{1:k-1,k} := [Q_1, \ldots, Q_{k-1}]^T V_k$
3:     $V'_k := V_k - [Q_1, \ldots, Q_{k-1}] R_{1:k-1,k}$
4:     Compute $V'_k = Q_k R_{kk}$ via TSQR
5: end for
```

4. CA-Arnoldi

4.1. Arnoldi Iteration

The Arnoldi iteration is a method for solving sparse nonsymmetric eigenvalue problems and was first introduced by W. Arnoldi in [1]. S steps of standard Arnoldi produce an $s+1 \times s$ upper Hessenberg matrix $\underline{H}$ and $m \times s+1$ orthonormal vectors $\underline{Q} = [q_1, q_2, \ldots, q_s, q_{s+1}]$, where

$$AQ = \underline{Q}\underline{H}. \tag{4.1}$$

In the GMRES method the columns of $\underline{Q}$ form a basis for the Krylov Subspace $\mathcal{K}_{s+1}(A, r_0)$. There are many ways to orthogonalize successive basis vectors. Modified Gram-Schmidt is often employed because it performs numerically better compared to classical Gram-Schmidt (MGS based Arnoldi is outlined in Algorithm 2). On the other hand, CGS is more suited for parallel implementations, because it provides fewer synchronization points. Walker [14] used Householder QR instead because it provides better orthogonalization than MGS. Since Householder QR requires the vectors to be available all at once, Walker produced s Monomial basis vectors first. Bai et al. [?] later improved on Walkers work by replacing the (usually ill-conditioned) Monomial basis with the Newton basis. Greenbaum et al. [6] later showed, that the loss of orthogonality caused by MGS usually does not affect the solution of the linear system at all. This gave rise to a new communication avoiding version of the Arnoldi method that will be used by CA-GMRES, namely Arnoldi(s,t).

Algorithm 2 MGS based Arnoldi iteration

Input: $n \times n$ matrix A and starting vector v of size n

Output: Orthonormal $n \times s+1$ matrix $\underline{Q} = [Q, q_{s+1}]$, and a nonsingular $s+1 \times s$ upper Hessenberg matrix $\underline{H}$ such that $AQ = \underline{Q}\underline{H}$

```
1: $\beta := \|v\|_2$, $q_1 := v/\beta$
2: for $j = 1$ to $s$ do
3:     $w_j := Aq_j$
4:     for $i = 1$ to $j$ do
5:         $h_{ij} := \langle w, q_i \rangle$
6:         $w_j := w_j - h_{ij}q_i$
7:     end for
8:     $h_{j+1,j} := \|w_j\|_2$
9:     $q_{j+1} := w_j/h_{j+1,j}$
10: end for
```

4.2. Arnoldi(s,t)

Arnoldi(s,t) was first introduced by Hoemmen et al. in [8] and is based on Walkers Householder Arnoldi method [14] and the s-step Arnoldi method of Kim and Chronopoulos [10]. Like Walkers version, Arnoldi(s,t) produces s basis vectors at once, except that, after s steps, Walkers method must restart. This makes choosing a good s difficult. If s is too short, the method may converge too slow or may not converge at all. If s is too large, the method is not numerically stable.

Arnoldi(s,t) decouples the step size from the restart length by introducing an additional parameter t. The parameter t refers to the number of iterations that produce s orthonormal basis vectors before the method must restart. Therefore, the restart length m is given by $m = s \cdot t$, where the basis length s refers to the number of *inner* iterations and t denotes the number of *outer* iterations. The s-step Arnoldi method of Kim and Chronopoulos also shares this independence property, but is not as effective in terms of avoiding communication and basis orthogonalization. Also, while Arnoldi(s,t) can use any s-step basis, Kim and Chronopoulos' algorithm is restricted to the Monomial basis only.

Hoemmen et al. [8] use the MPK to produce s basis vectors at each outer iteration and considered the Monomial, the Newton and the Chebyshev basis. Depending on the basis type, their MPK has to compute either a one-term, two-term, or three-term recurrence respectively. Among the three, the Chebyshev basis is the most expensive to compute and also reacts more sensitive to bad eigenvalue approximations than the Newton basis (see Section 4.2.2). Therefore, it is not considered here. A more detailed description of the Chebyshev basis can be found, e.g. in [8] and in Joubert and Carey [9].

4.2.1. The Monomial basis

The Monomial basis in s-step Krylov methods is given by

$$\mathcal{K}_{s+1}(A, v) = [v, Av, A^2v, \ldots, A^s v]$$

and has a change of basis matrix

$$\underline{B} = [\sigma_1 e_2, \sigma_2 e_3, \ldots, \sigma_s e_{s+1}].$$

(where $\sigma_1, \ldots, \sigma_s$ are scaling factors) that satisfies

$$AV = \underline{VB} \tag{4.2}$$

with V and $\underline{V}$ having dimensions $n \times s$ resp. $n \times s+1$ and $\underline{B}$ being an $s+1 \times s$ structurally upper Hessenberg matrix. The Monomial basis is also known as the *power method* which is an iterative method for finding the principal eigenvalue and corresponding eigenvector of a matrix by repeatedly applying a starting vector to the matrix. If the matrix and starting vector satisfy certain conditions, the basis converges to the principal eigenvector. Ideally, a basis has orthogonal basis vectors and should not converge. In theory, the converged basis is still linearly independent in exact arithmetic. In machine arithmetic, the basis vectors become inevitably dependent at some point. Therefore, other bases are considered, (e.g. Newton or Chebyshev) that provide better numerical stability.

The similarity between (4.2) and the Arnoldi relation (4.1) can be used in order to reconstruct the upper Hessenberg matrix $\underline{H}$. The vectors created from the QR factorization of $\underline{V}$ might differ from the ones created by the standard Arnoldi method by a unitary scaling (see Section 5.1 for details). For simplicity, it is assumed, that the QR factorization $\underline{V} = \underline{QR}$ produces the same unitary vectors as standard Arnoldi. From

$$\begin{aligned} AV &= \underline{VB} \\ AQR &= \underline{QRB} \\ AQ &= \underline{Q}\underline{RB}R^{-1} \end{aligned} \tag{4.3}$$

emerges

$$\underline{H} = \underline{RB}R^{-1}. \tag{4.4}$$

Since $\underline{H}$ is upper Hessenberg, $\underline{B}$ must at least be structurally upper Hessenberg as well. This is in fact the case for the Monomial basis.

4.2.2. The Newton basis

The Newton basis in s-step Krylov methods is given by

$$\mathcal{K}_{s+1}(A,v) = \left[v, (A-\theta_1 I)v, (A-\theta_2 I)(A-\theta_1 I)v, \ldots, \prod_{i=1}^{s}(A-\theta_i I)v\right]$$

and has a change of basis matrix

$$\underline{B} = \begin{pmatrix} \theta_1 & 0 & \ldots & 0 \\ \sigma_1 & \theta_2 & \ddots & \vdots \\ 0 & \sigma_2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \theta_s \\ 0 & 0 & \ldots & \sigma_s \end{pmatrix} \tag{4.5}$$

with scaling factors $\sigma_1, \ldots, \sigma_s$ and shifts $\theta_1, \ldots, \theta_s$.

Since $\underline{B}$ is structurally Hessenberg, Equation (4.4) holds for the Newton basis as well.

Hoemmen et al. [8], among many other authors, choose the shifts to be the eigenvalues of the upper Hessenberg matrix H (the *Ritz* values), because the Arnoldi iteration implicitly constructs an interpolating polynomial of the characteristic polynomial of A at these points. Reichel [12] showed that, at least for normal matrices, the condition number of the Newton basis grows sub-exponentially in the number of interpolation points, whereas the Monomial basis has exponential growth. However, poor approximations of the eigenvalues of A may lead to faster growth of the basis condition number than expected. For good approximations $A - \theta I$ may be ill-conditioned and several nearly identical shifts in a row could lead to an ill-conditioned basis. Therefore, the shifts have to be ordered in a way that makes the Newton basis as dissimilar from the Monomial basis as possible. This is accomplished by the Leja ordering.

The Leja ordering This section only provides an overview of the Leja ordering. For a more detailed description see Hoemmen et al. [8]. The Leja ordering takes as input a set of shifts $\theta_1, \ldots, \theta_s$ and orders them so that a particular measure of the 'distance' of the current shift θ_j is maximized to the previous shifts $\theta_1, \ldots, \theta_{j-1}$ (see (4.8)). These shifts may come in complex conjugate pairs and with some multiplicity μ (Hoemmen et al. [8] point out, that the Ritz values from a Krylov method may be unique in exact arithmetic, but need not be in machine arithmetic). If the ordered Ritz values have consecutive complex conjugate pairs, the Newton basis can be computed using real arithmetic only (see the next paragraph for details). Unfortunately, the Leja ordering may separate complex conjugate pairs during the ordering process. The *Modified* Leja ordering is an extension of the Leja ordering and ensures that complex conjugate pairs stay in consecutive order with the leading entry always comprising the positive imaginary part. The ordering process for both the Leja and Modified Leja orderings comprises the sets K_j for $j = 1, 2, \ldots, s$, where K_j is given by

$$K_j := \{\theta_{j+1}, \theta_{j+2}, \ldots, \theta_s\}. \tag{4.6}$$

The first shift θ_1 is chosen by

$$\theta_1 = \text{argmax}_{z \in K_0} |z|. \tag{4.7}$$

Subsequent shifts $\theta_2, \ldots, \theta_s$ are chosen using the rule

$$\theta_{j+1} = \text{argmax}_{z \in K_j} \prod_{k=0}^{j} |z - z_k|^{\mu_j} \tag{4.8}$$

where μ_j denotes the number of occurrences of z_k (i.e., the multiplicity of z_k). The product to maximize in Equation (4.8) may underflow (for shifts that are close together) or overflow (for shifts that are far apart) in machine arithmetic. In order to prevent this, Hoemmen et al. [8] suggest to scale the shifts with a capacity estimate. Scaling by a capacity estimate may still fail. In this case, Hoemmen et al. restart the computation with slightly randomly perturbed input values and repeat this process with growing perturbations until the method succeeds or an iteration limit has been reached.

Avoiding complex arithmetic Like the eigenvalues of a real matrix, the Ritz values can occur as complex conjugate pairs. The Modified Leja ordering ensures that these pairs are ordered consecutively with leading positive imaginary entries, i.e. $\theta_{j+1} := \overline{\theta}_j$ with $\Im(\theta_j) > 0$. Complex arithmetic doubles the storage and floating point operations and therefore, should be avoided. Instead of computing $v_{j+1} = (A - \theta_j I)v_j$ and $v_{j+2} = (A - \overline{\theta}_j I)v_{j+1}$ like one would normally do, Bai et al. [?] suggest that complex arithmetic can be skipped by setting

$$v_{j+1} = (A - \Re(\theta_j)I)v_j \tag{4.9}$$

and

$$v_{j+2} = (A - \Re(\theta_j)I)v_{k+1} + \Im(\theta_j)^2 v_j. \tag{4.10}$$

It can easily be shown that

$$\begin{aligned} v_{j+2} &= (A - \Re(\theta_j)I)^2 v_j + \Im(\theta_j)^2 v_j \\ &= (A - \overline{\theta}_j I)(A - \theta_j I)v_j. \end{aligned}$$

This also affects the change of basis matrix $\underline{B}$. If the Ritz values contain complex conjugate pairs, B is tridiagonal. E.g., if θ_1 through θ_s are real, with the exception of θ_j

and θ_{j+1} being a complex conjugate pair, the change of basis matrix is given by

$$\underline{B} = \begin{pmatrix} \theta_1 & 0 & \dots & \dots & \dots & 0 \\ \sigma_1 & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \Re(\theta_j) & -\Im(\theta_j)^2 & \ddots & \vdots \\ \vdots & \ddots & \sigma_j & \Re(\theta_{j+1}) & \ddots & \vdots \\ \vdots & \ddots & \ddots & \sigma_{j+1} & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \theta_s \\ 0 & \dots & \dots & \dots & 0 & \sigma_s \end{pmatrix}.$$

Performance notes Avoiding complex arithmetic in that way necessitates an extra SpMV operation in the Newton basis which, in the worst case, leads to as many floating point operations as in the Chebyshev basis (which uses a three-term recurrence). However, in their performance analysis Hoemmen et al. [8] observed that the runtime of the Newton basis was still close to the runtime of the Monomial basis.
[8] further point out, that this approach might lose accuracy when θ_{j-1} is real and θ_j and θ_{j+1} form a complex conjugate pair with $\Re(\theta_j) = \theta_{j-1}$. Then

$$\begin{aligned} v_j &= (A - \theta_{j-1}I)v_{j-1} \\ v_{j+1} &= (A - \Re(\theta_j)I)(A - \theta_{j-1}I)v_{j-1} \\ &= (A - \theta_{j-1}I)^2 v_{j-1} \end{aligned}$$

is equivalent to computing the Monomial basis with a possibly ill-conditioned matrix $A - \theta_{j-1}I$. This might occur if the Ritz values reside within an ellipse with a long vertical axis and very short horizontal axis on the complex plane.

4.2.3. The Arnoldi(s,t) algorithm

In this thesis, the main focus lies on the Newton-based CA-GMRES method. Therefore, only the Netwon-based Arnoldi(s, t) algorithm is outlined in Algorithm 3 and described below.

First inner iteration The Arnoldi(s, t) algorithm starts its first outer iteration with s steps of the standard Arnoldi method which results in an $n \times s + 1$ basis vector matrix $\underline{Q}_0$, with

$$\underline{Q}_0 = [Q_0, q_{s+1}] = [q_1, q_2, \dots, q_s, q_{s+1}] \tag{4.11}$$

and an $s + 1 \times s$ nonsingular upper Hessenberg matrix $\underline{H}_0$ that satisfies

$$AQ_0 = \underline{Q}_0 \underline{H}_0. \tag{4.12}$$

Approximations of the eigenvalues of A are obtained by computing the eigenvalues of the $s \times s$ upper submatrix of $\underline{H}_0$, namely H_0. As mentioned in Section 4.2.2, the eigenvalues

Algorithm 3 Arnoldi(s, t)

```
Input: $n \times n$ matrix $A$ and starting vector $v$ of size $n$
Output: An orthonormal $n \times st + 1$ matrix $\underline{\mathfrak{Q}} = [\mathfrak{Q}, q_{st+1}]$ and a nonsingular $st + 1 \times st$
    upper Hessenberg matrix $\underline{H}$ such that $AQ = \underline{QH}$
 1: $\beta := \|v\|_2$, $q_1 := v/\beta$
 2: for $k = 0$ to $t - 1$ do
 3:   if $k = 0$ then
 4:     Compute $\underline{Q}_0$ and $\underline{H}_0$ using Algorithm 2
 5:     Set $\underline{\mathfrak{Q}}_0 := \underline{Q}_0$ and $\underline{\mathfrak{H}}_0 := \underline{H}_0$
 6:     Compute Ritz values from $H_0$
 7:   else
 8:     Fix basis conversion matrix $\underline{B}_k$
 9:     Compute $\underline{\acute{V}}_k$
10:     $\underline{\acute{\mathfrak{R}}}_{k-1,k} := \underline{\mathfrak{Q}}_{k-1}^T \underline{\acute{V}}_k$
11:     $\underline{\acute{V}}'_k := \underline{\acute{V}}_k - \underline{\mathfrak{Q}}_{k-1} \underline{\acute{\mathfrak{R}}}_{k-1,k}$
12:     Compute QR factorization of $\underline{\acute{V}}'_k \rightarrow \underline{\acute{Q}}_k \underline{\acute{R}}_k$ using TSQR
13:     Compute $\underline{\mathfrak{H}}_{k-1,k} := -\mathfrak{H}_{k-1} \mathfrak{R}_{k-1,k} R_k^{-1} + \underline{\mathfrak{R}}_{k-1,k} \underline{B}_k R_k^{-1}$
14:     Compute $H_k := R_k B_k R_k^{-1} + \tilde{\rho}_k^{-1} b_k z_k e_s^T - h_{k-1} e_1 e_{sk}^T \mathfrak{R}_{k-1,k} R_k^{-1}$
15:     Compute $h_k := \tilde{\rho}_k^{-1} \rho_k b_k$
16:     $\underline{\mathfrak{H}}_k := \begin{pmatrix} \mathfrak{H}_{k-1} & \underline{\mathfrak{H}}_{k-1,k} \\ h_{k-1} e_1 e_{sk}^T & H_k \\ 0_{1,sk} & h_k e_s^T \end{pmatrix}$
17:   end if
18: end for
```

of H_0 are also known as the Ritz values $\theta_1, \dots, \theta_s$ and will be used as shifts for the Newton basis in successive outer iterations.

Successive inner iterations At iteration k (with $k > 0$) the last basis vector of the previous iteration is taken as the new start vector v_{sk+1} and the Newton basis vector matrix

$$\underline{\acute{V}}_k = [\acute{V}_k, v_{s(k+1)+1}] = [v_{sk+2}, \dots, v_{s(k+1)}, v_{s(k+1)+1}] \tag{4.13}$$

is computed. Then, the upper Hessenberg matrix $\mathfrak{H}_k$ must be recreated. From Relation (4.2) emerges

$$A[\underline{\mathfrak{Q}}_{k-1}, \acute{V}_k] = [\underline{\mathfrak{Q}}_{k-1}, \underline{\acute{V}}_k]\underline{\mathfrak{B}}_k \tag{4.14}$$

where $\underline{\mathfrak{B}}_k$ satisfies

$$\underline{\mathfrak{B}}_k = \begin{pmatrix} \mathfrak{H}_{k-1} & 0_{sk,s} \\ h_{k-1}e_1e_{sk}^T & \underline{B}_k \end{pmatrix} \tag{4.15}$$

with $\mathfrak{H}_0 := H_0$ and $\begin{pmatrix} \mathfrak{H}_0 \\ h_0 e_{sk}^T \end{pmatrix} := \underline{H}_0$.

Computing the QR factorization of $\acute{V}$ and $\underline{\acute{V}}$ respectively yields

$$A[\underline{\mathfrak{Q}}_{k-1}, \acute{Q}_k] \cdot \mathfrak{R}_k = [\underline{\mathfrak{Q}}_{k-1}, \underline{\acute{Q}}_k] \cdot \underline{\mathfrak{R}}_k \cdot \underline{\mathfrak{B}}_k$$

with

$$\begin{aligned} \mathfrak{R}_k &= \begin{pmatrix} I_{sk+1} & \acute{\mathfrak{R}}_{k-1,k} \\ 0_{s-1,sk+1} & \acute{R}_k \end{pmatrix}, \\ \underline{\mathfrak{R}}_k &= \begin{pmatrix} I_{sk+1} & \underline{\acute{\mathfrak{R}}}_{k-1,k} \\ 0_{s,sk+1} & \underline{\acute{R}}_k \end{pmatrix} \end{aligned}$$

where

$$\begin{aligned} \acute{\mathfrak{R}}_{k-1,k} &= \underline{\mathfrak{Q}}_{k-1}^T \acute{V}_k, \\ \underline{\acute{\mathfrak{R}}}_{k-1,k} &= \underline{\mathfrak{Q}}_{k-1}^T \underline{\acute{V}}_k \end{aligned}$$

are the interim results of the BCGS process and $\acute{R}_k$ and $\underline{\acute{R}}_k$ are the R factors in the QR factorization of V_k resp. $\underline{V}_k$.
The Arnoldi Relation (4.1) yields

$$A[\underline{\mathfrak{Q}}_{k-1}, \acute{Q}_k] = [\underline{\mathfrak{Q}}_{k-1}, \underline{\acute{Q}}_k]\underline{\mathfrak{H}}_k$$

and from the equation

$$\begin{aligned} A \quad & [\underline{\mathfrak{Q}}_{k-1}, \acute{Q}_k] \cdot \begin{pmatrix} I_{sk+1} & \acute{\mathfrak{R}}_{k-1,k} \\ 0_{s-1,sk+1} & \acute{R}_k \end{pmatrix} \\ = \quad & [\underline{\mathfrak{Q}}_{k-1}, \underline{\acute{Q}}_k] \cdot \begin{pmatrix} I_{sk+1} & \underline{\acute{\mathfrak{R}}}_{k-1,k} \\ 0_{s,sk+1} & \underline{\acute{R}}_k \end{pmatrix} \cdot \begin{pmatrix} \mathfrak{H}_{k-1} & 0_{sk,s} \\ h_{k-1}e_1e_{sk}^T & \underline{B}_k \end{pmatrix} \end{aligned} \tag{4.16}$$

the upper Hessenberg matrix $\underline{\mathfrak{H}}_k$ can be retrieved, with

$$\begin{aligned}\underline{\mathfrak{H}}_k &= \underline{\mathfrak{R}}_k \underline{\mathfrak{B}}_k \mathfrak{R}_k^{-1} \\ &= \begin{pmatrix} I_{sk+1} & \underline{\acute{\mathfrak{R}}}_{k-1,k} \\ 0_{s,sk+1} & \underline{\acute{R}}_k \end{pmatrix} \cdot \begin{pmatrix} \mathfrak{H}_{k-1} & 0_{sk,s} \\ h_{k-1} e_1 e_{sk}^T & \underline{B}_k \end{pmatrix} \cdot \begin{pmatrix} I_{sk+1} & \acute{\mathfrak{R}}_{k-1,k} \\ 0_{s-1,sk+1} & \acute{R}_k \end{pmatrix}^{-1}. \quad (4.17)\end{aligned}$$

Updating the upper Hessenberg matrix Hoemmen et al. derive in [8] how to compute Equation (4.17) efficiently. It suffices to show the results only here. However, in order to understand their meaning, some matrices have to be broken down or must be repartitioned first. This has nothing to do with additional computations or data movement, it is purely notational.

First, four different parts of the basis vector matrix V are defined by

$$\begin{aligned}\acute{V}_k &= [v_{sk+2}, \ldots, v_{s(k+1)}], \\ \underline{\acute{V}}_k &= [\acute{V}_k, v_{s(k+1)+1}], \\ V_k &= [v_{sk+1}, \acute{V}_k], \\ \underline{V}_k &= [v_{sk+1}, \underline{\acute{V}}_k].\end{aligned}$$

Then, four variations on the interim result of the BCGS process are defined by

$$\begin{aligned}\mathfrak{R}_{k-1,k} &= \mathfrak{Q}_{k-1}^T V_k, \\ \underline{\mathfrak{R}}_{k-1,k} &= \mathfrak{Q}_{k-1}^T \underline{V}_k, \\ \acute{\mathfrak{R}}_{k-1,k} &= \underline{\mathfrak{Q}}_{k-1}^T \acute{V}_k, \\ \underline{\acute{\mathfrak{R}}}_{k-1,k} &= \underline{\mathfrak{Q}}_{k-1}^T \underline{\acute{V}}_k.\end{aligned}$$

Given this notation, the R factors $\underline{\mathfrak{R}}_k$ and $\mathfrak{R}_k$ can be repartitioned into

$$\begin{aligned}\underline{\mathfrak{R}}_k &= \begin{pmatrix} I_{sk+1} & \underline{\acute{\mathfrak{R}}}_{k-1,k} \\ 0_{s,sk+1} & \underline{\acute{R}}_k \end{pmatrix} = \begin{pmatrix} I_{sk} & \underline{\mathfrak{R}}_{k-1,k} \\ 0_{s+1,sk} & \underline{R}_k \end{pmatrix}, \\ \mathfrak{R}_k &= \begin{pmatrix} I_{sk+1} & \acute{\mathfrak{R}}_{k-1,k} \\ 0_{s-1,sk+1} & \acute{R}_k \end{pmatrix} = \begin{pmatrix} I_{sk} & \mathfrak{R}_{k-1,k} \\ 0_{s,sk} & R_k \end{pmatrix}\end{aligned}$$

where R_k and $\underline{\acute{R}}_k$ are $s \times s$ matrices, $\underline{R}_k$ is $s+1 \times s+1$ and $\acute{R}_k$ is an $s-1 \times s-1$ matrix. (see Appendix C for a sketch of $\underline{\mathfrak{R}}_2$ with $s = 3$)

Since the last basis vector of the previous iteration (e.g., the last column of $\underline{\mathfrak{Q}}_{k-1}$) is orthonormal to all previous basis vectors and since it is taken as the start vector of the current iteration, it holds that

$$\underline{\mathfrak{Q}}_{k-1}^T \underline{V}_k e_1 = \underline{\mathfrak{Q}}_{k-1}^T q_{sk+1} = e_{sk+1}.$$

From that follows $\underline{\mathfrak{R}}_{k-1,k} e_1 = \mathfrak{Q}_{k-1}^T \underline{V}_k e_1 = \mathfrak{Q}_{k-1}^T q_{sk+1} = 0_{sk,1}$ resp. $\mathfrak{R}_{k-1,k} e_1 = 0_{sk,1}$ and $\underline{R}_k e_1 = 1$ resp. $R_k e_1 = 1$.

$\underline{R}_k$ is broken down into

$$\underline{R}_k = \begin{pmatrix} R_k & z_k \\ 0_{s,s} & \rho_k \end{pmatrix}$$

where z_k is $1 \times s$, ρ_k is a scalar and $\tilde{\rho}_k^{-1} := R_k^{-1}(s,s)$. $\underline{B}_k$ is broken down into

$$\underline{B}_k = \begin{pmatrix} B_k \\ b \cdot e_s^T \end{pmatrix}.$$

$\underline{\mathfrak{H}}_k$ then consists of the following parts

$$\underline{\mathfrak{H}}_k := \begin{pmatrix} \mathfrak{H}_{k-1} & \underline{\mathfrak{H}}_{k-1,k} \\ h_{k-1} e_1 e_{sk}^T & H_k \\ 0_{1,sk} & h_k e_s^T \end{pmatrix} \tag{4.18}$$

with

$$\begin{aligned} \underline{\mathfrak{H}}_{k-1,k} &:= -\mathfrak{H}_{k-1}\mathfrak{R}_{k-1,k}R_k^{-1} + \underline{\mathfrak{R}}_{k-1,k}\underline{B}_k R_k^{-1}, && (4.19) \\ H_k &:= R_k B_k R_k^{-1} + \tilde{\rho}_k^{-1} b_k z_k e_s^T - h_{k-1} e_1 e_{sk}^T \mathfrak{R}_{k-1,k} R_k^{-1}, && (4.20) \\ h_k &:= \tilde{\rho}_k^{-1} \rho_k b_k. && (4.21) \end{aligned}$$

5. CA-GMRES

The CA-GMRES algorithm as in Hoemmen et al. [8] uses all of Arnoldi(s,t) together with a Givens rotation scheme that exposes the absolute residual error $\|b - Ax_k\|_2$ at every inner iteration j. Therefore, the approximate solution x_j need not be computed in order to estimate convergence. CA-GMRES also solves the linear least-squares problem

$$y_k := argmin_y \left\|\underline{\mathfrak{H}}_k y - \beta e_1\right\|_2$$

at every outer iteration k and then computes a new approximate solution $x_k := x_0 + \mathfrak{Q}_k y_k$. The difference to GMRES(m) is, that s Givens rotations have to be applied at once. A Newton-based version of the CA-GMRES algorithm is outlined in Algorithm 4.

5.1. Scaling the first basis vector

The Monomial-based CA-GMRES method could also start without the standard Arnoldi method by generating an $n \times s+1$ Monomial basis vector matrix first and then performing a QR factorization as in Equation (4.3). The vectors created from the QR factorization might differ from the ones created by the standard Arnoldi method by a unitary scaling. Suppose that Arnoldi(s,t) and the standard Arnoldi method start with the same starting vector q_1. After st steps, Arnoldi(s,t) developed an $n \times st+1$ basis vector matrix $\underline{\mathfrak{Q}}$ and an $st+1 \times st$ upper Hessenberg matrix $\underline{\mathfrak{H}}$ and standard Arnoldi produced an $n \times st+1$ basis vector matrix $\hat{\underline{\mathfrak{Q}}}$ and an $st+1 \times st$ upper Hessenberg matrix $\hat{\underline{\mathfrak{H}}}$. $\underline{\mathfrak{Q}}$ then differs from $\hat{\underline{\mathfrak{Q}}}$ by a unitary scaling $\underline{\Theta} := \text{diag}(\theta_1,\ \theta_2,\ \ldots,\ \theta_{st},\ \theta_{st+1})$ with $\theta_j = |1|$, $\forall j$, such

Algorithm 4 Newton-GMRES(s,t)

```
Input: $n \times n$ linear system $Ax = b$ and initial guess $x_0$
 1: restart := true
 2: while restart do
 3:   $r_0 := b - Ax_0$, $\beta := \|r_0\|_2$, $q_1 := r_0/\beta$,
 4:   for $k = 0$ to $t-1$ do
 5:     if $k = 0$ then
 6:       Compute $\underline{Q}_0$ and $\underline{H}_0$ using MGS-Arnoldi
 7:       Set $\underline{\mathfrak{Q}}_0 := \underline{Q}_0$ and $\underline{\mathfrak{H}}_0 := \underline{H}_0$
 8:       Compute Ritz values from $H_0$
 9:       Reduce $\underline{H}_0$ from upper Hessenberg to upper triangular form using $s$ Givens
            rotations $G_1, G_2, \dots, G_s$. Apply the same rotations in the same order to
            $\beta e_1$, resulting in the length $s+1$ vector $\zeta_0$.
10:     else
11:       Fix basis conversion matrix $\underline{B}_k$
12:       Set $v_{sk+1} := q_{sk+1}$
13:       Compute $\underline{\acute{V}}_k$ where $v_{i+1} = (A - \theta_i I)v_i$, for $i = sk+1, \dots, sk+s$
14:       $\underline{\acute{\mathfrak{R}}}_{k-1,k} := \underline{\mathfrak{Q}}_{k-1}^T \underline{\acute{V}}_k$
15:       $\underline{\acute{V}}_k' := \underline{\acute{V}}_k - \underline{\mathfrak{Q}}_{k-1} \underline{\acute{\mathfrak{R}}}_{k-1,k}$
16:       Compute QR factorization of $\underline{\acute{V}}_k' \to \underline{\acute{Q}}_k \underline{\acute{R}}_k$ using TSQR
17:       Compute $\underline{\mathfrak{H}}_{k-1,1} := -\mathfrak{H}_{k-1}\mathfrak{R}_{k-1,k}R_k^{-1} + \underline{\mathfrak{R}}_{k-1,k}\underline{B}_k R_k^{-1}$
18:       Compute $H_k := R_k B_k R_k^{-1} + \tilde{\rho}_k^{-1} b_k z_k e_s^T - h_{k-1} e_1 e_{sk}^T \mathfrak{R}_{k-1,k} R_k^{-1}$
19:       Compute $h_k := \tilde{\rho}_k^{-1} \rho_k b_k$
20:       $\underline{\mathfrak{H}}_k := \begin{pmatrix} \mathfrak{H}_{k-1} & \underline{\mathfrak{H}}_{k-1,k} \\ h_{k-1} e_1 e_{sk}^T & H_k \\ 0_{1,sk} & h_k e_s^T \end{pmatrix}$
21:       Apply Givens rotations $G_1, \dots, G_{sk}$ in order to $\begin{pmatrix} \underline{\mathfrak{H}}_{k-1,k} \\ \underline{H}_k \end{pmatrix}$.
22:       Reduce $\underline{H}_k$ to upper triangular form using $s$ Givens rotations $G_{sk+1}, \dots,$
            $G_{s(k+1)}$. Apply the rotations in the same order to $\begin{pmatrix} \zeta_{k-1} \\ 0_{s,1} \end{pmatrix}$, resulting in the
            length $s(k+1)+1$ vector $\zeta_k$.
23:     end if
24:     Element $s(k+1)+1$ of $\zeta_k$ is the 2-norm (in exact arithmetic) of the current
          residual $r_{k+1} = b - Ax_{k+1}$ of the current solution $x_{k+1}$.
25:     if converged then
26:       restart = false, and exit for loop
27:     end if
28:   end for
29:   Use the above reduction of $\underline{\mathfrak{H}}_k$ to upper triangular form and $\zeta_k$ to solve $y_k :=$
        $\operatorname{argmin}_y \left\| \underline{\mathfrak{H}}_k y - \beta e_1 \right\|_2$
30:   Set $x_0 := x_0 + \mathfrak{Q}_k y_k$
31: end while
```

that $\hat{\underline{\mathfrak{Q}}} = \underline{\mathfrak{Q}}\underline{\Theta}$. Also, $\underline{\mathfrak{H}}$ differs from $\hat{\underline{\mathfrak{H}}}$ such that $\hat{\underline{\mathfrak{H}}} = \underline{\Theta}^T \underline{\mathfrak{H}}\Theta$, where Θ is the $st \times st$ upper left submatrix of the $st+1 \times st+1$ diagonal matrix $\underline{\Theta}$. Suppose that iteration st of standard GMRES computes an approximate solution $\hat{x} = x_0 + \hat{\mathfrak{Q}}\hat{y}$ and CA-GMRES computes $x = x_0 + \mathfrak{Q}y$. Hoemmen et al. show in [8] that the computed solutions relate only through the first direction θ_1, with $\hat{x} = x_0 + \theta_1 \mathfrak{Q}y$. I.e., if $\theta \neq 1$, the CA-GMRES solution x will differ from the standard GMRES solution $\hat{x}$. This issue can be addressed in multiple ways. Either the QR factorization used must not change direction of the first column or $\theta_1 = \langle r_0, q_1 \rangle \ / \ \beta$ must be computed which adds additional communication. A third approach is to compute the first set of basis vectors with the standard Arnoldi method (this happens naturally when the first outer iteration is started with standard Arnoldi in order to compute Ritz values for the Newton basis).

5.2. A different approach

If the Newton-based GMRES method in [5] did not converge, Erhel computed $2s$ Ritz values instead of s for better eigenvalue approximations. She then picked s of them and sorted them with the Modified Leja ordering. This improved the condition of the Newton basis and lead to (better) convergence in some cases. Hoemmen et al. [8] simply chose a smaller step size. This might give bad eigenvalue approximations and one might want to compute $2s$ Ritz values anyway. However, these $2s$ values could come in complex conjugate pairs and have to amount to s values eventually. If the $2s$ Ritz values only consist of complex conjugate pairs and s is odd, a complex conjugate pair has to be split. To address this issue and still avoid complex arithmetic, one could either compute $2s+1$ Ritz values or simply omit the positive imaginary part of the last Ritz value. Another approach might be to apply the $2s$ Ritz values to two consecutive outer iterations of Arnoldi(s,t). Since computing $2s$ Ritz values yields better approximations of the eigenvalues, one could incorporate them all as well. This slightly changes the way Equation (4.19) is computed. Remember that, in order to avoid complex arithmetic, the consecutive order of a complex conjugate pair must be preserved. In the case where θ_s is the first entry of a complex conjugate pair, the change of basis matrix $\underline{B}_{k-1}$ is connected to its consecutive change of basis matrix $\underline{B}_k$ by an additional entry right to the last Ritz value of $\underline{B}_{k-1}$ and above the first Ritz value of $\underline{B}_k$. $\underline{\mathfrak{B}}_k$ then differs from Equation (4.15) by that additional entry with

$$\underline{\mathfrak{B}}_k = \begin{pmatrix} \mathfrak{H}_{k-1} & -e_{sk} e_1^T \Im(\theta_s)^2 \\ h_{k-1} e_1 e_{sk}^T & \underline{B}_k \end{pmatrix}. \tag{5.1}$$

Equation (4.19) then changes to

$$\underline{\mathfrak{H}}_{k-1,k} := -\mathfrak{H}_{k-1}\mathfrak{R}_{k-1,k} R_k^{-1} + \underline{\mathfrak{R}}_{k-1,k} \underline{B}_k R_k^{-1} - \Im(\theta_s)^2 e_{sk} e_1^T R_k^{-1}. \tag{5.2}$$

However, this option might overcomplicate things without significant benefit.

5.3. Preconditioning

For better effectiveness GMRES is usually combined with a preconditioner. Different kinds of preconditioners exist, like left, right, and split preconditioners. In this thesis, only left preconditioners are considered. The preconditioner matrix M^{-1} is applied to the left of A, where $M \approx A$. The system for CA-GMRES to solve is then given by $M^{-1}Ax = M^{-1}b$. Precisely, the preconditioner M^{-1} must be applied to the red parts in Algorithm 4, i.e. replace $r_0 = b - Ax_0$ by $r_0 = M^{-1}(b - Ax_0)$ and $v_{i+1} = (A - \theta_i)v_i$ by $v_{i+1} = M^{-1}((A - \theta_i)v_i)$.

Scaling is a special type of preconditioning. Hoemmen et al. [8] considered two types of scaling in order to prevent rapid basis vector growth and improve numeric stability:

1. Balancing: replacing A by $A' = DAD^{-1}$ with D diagonal.
2. Equilibration: replacing A by $A' = D_r A D_c$ with D_r and D_c diagonal, so that the largest absolute value in every row and every column is approximately equal to 1.

In their experiments for solving nonsymmetric linear systems with CA-GMRES Hoemmen et al. [8] found that for practical problems, equilibration proved quite effective and almost made the basis type irrelevant. We observed mostly similar behavior after applying a standard version of the CA-ILU(0) preconditioner, which is described below.

5.3.1. CA-ILU(0) preconditioner

The CA-ILU(0) is a communication-avoiding variant of the incomplete LU factorization where the structure of L resp. U is the same as the lower resp. upper triangular part of A. The preconditioner $M = LU$ then approximates the matrix A with $A = LU + R$, where R is the residual matrix.
In addition to the matrix-vector multiplications in the MPK the CA-ILU(0) uses a forward and a backward substitution. Therefore, additional data has to be fetched and ghosted before the computations are done. This often leads to vast amounts of data transfer in the end. Grigori et al. [7] address this issue by first reordering the matrix A via nested dissection and then reducing fills further by applying their novel Alternating min-max layers algorithm. While nested dissection affects convergence, Grigori et al. state that their novel algorithm does not. Also, the LU factorization itself can be done without communication because the graphs of L and U are known prior to their computation.

5.4. Convergence metrics

CA-GMRES produces a cheap convergence metric, namely the absolute residual $\|r_{k+1}\|_2$ relative to the initial residual $\|r_0\|_2$. This metric might not be the best choice since this 'relative' residual error depends too strongly on the initial guess x_0 (e.g., if $\|x_0\|_2$ is too large $\|r_0\|$ will be large and the iteration will stop too early). Therefore, one might consider more expensive alternatives. For a fair comparison, the convergence metrics should be the same for both CA-GMRES and standard GMRES. Therefore, other metrics would only add the same amount of overhead. Since the methods are just compared here and since Hoemmen et al. use the cheap metric as well, both implemented methods do not incorporate other convergence metrics.

5.5. Implementation details

The CA-GMRES algorithm was implemented in the `C++` language and incorporates the LAPACK and BLAS routines provided by the Intel Math Kernel Library (MKL). The code layout was inspired by PETSc's KSP framework which is written in `C` but emulates object oriented design. Simple patterns like the Strategy Pattern were implemented as well.
BLAS and LAPACK functions are written in the Fortran language. Fortran itself stores matrices in column major order while C++ adopts the row major order layout. MKL provides a C-interface to most of the Fortran-style BLAS and LAPACK routines. The TSQR factorization, which is implemented in MKL as described in Section 3.2, is one of the few routines that are used here but not supported. Therefore, dense matrices are stored in column major order and the unsupported Fortran routines are called directly.
MKL provides highly optimized routines that mostly benefit from cache-management techniques (e.g., `dgemm()`). The major optimization techniques used in MKL include loop unrolling, blocking, and data prefetching.

All experiments and computations were done on the harris server. harris provides 4 sockets with 12 AMD Opteron(tm) cores per socket and one thread per core, leading to a total of 48 threads.

Differences to the implementation of Hoemmen et al. [8]

- The CA-GMRES always starts its first inner iteration with the standard Arnoldi iteration, regardless of the basis type. This is not needed in case the Monomial basis is used. Starting with standard Arnoldi slows down the algorithm but might benefit the convergence if the Monomial basis is on the edge of being unstable.

- The CA-GMRES always starts an outer iteration with the standard Arnoldi iteration. If the Newton basis is used, the first iteration must start with standard Arnoldi in order to compute Ritz values. If the Ritz values are good approximations to the eigenvalues, it might not be necessary to compute those values again in successive outer iterations. Hoemmen et al. [8] do not address techniques or provide information on when to compute new Ritz values. One might monitor the estimate of the reciprocal condition number of the basis vector matrix and compute new values in case the condition number estimate worsens significantly. Since the R factor of the QR factorization of V and V have the same condition number the complexity for computing the reciprocal condition number is rather cheap with $\mathcal{O}(s^2)$.

- Although, the implemented Modified Leja ordering algorithm uses a capacity estimate to prevent over/underflows in the product to maximize, it does not ultimately perturb the input values if over/underflows still occur like in Hoemmen et al. [8]. Also, the algorithm depicted in [8] and Appendix D has complexity $\mathcal{O}(s^3)$. Computing the Ritz values from an upper $s \times s$ Hessenberg matrix requires that many operations anyway. However, Bai et al. [2] point out that the Modfied Leja ordering can be done in $\mathcal{O}(s^2)$ as well.

- A method that computes $2s$ Ritz values instead of s (as in Erhel [5]) was not implemented.
- Neither the MPK nor the CA-ILU(0) preconditioner were fully implemented due to time constraints. Therefore, the CA-GMRES algorithm uses the highly optimized sparse matrix-vector routine that is provided by MKL. Communication could be avoided anyway by giving MKL the hint that s successive SpMV operations are coming up for the creation of the s basis vectors. This optimization routine named `mkl_sparse_set_mv_hint()` would work for the Monomial basis, but might be ruined by the additional vector operation(s) needed in the Newton basis.
- To avoid rapid growth in the length of the basis vectors the basis vectors may be scaled by their 2-norm right after they are generated. This would break the communication avoiding scheme of the MPK if it was implemented at this point. In order to avoid scaling and therefore, communication, Hoemmen et al. [8] address this issue by equilibrating the matrix first.

how to build

1. MKL and PAPI are required before building.
2. Adapt the Makefile in the source folder (e.g., change the MKL root directory, etc.).
3. type 'make -j8' to build.
4. type 'make test' for a small example.

Problems and implementation difficulties

- Memory leakage was observed, that originates from MKL. Valgrind reports that bytes are possibly lost and/or still reachable. This indicates that a library might not be freed somewhere.
- The ILU(0) preconditioner was implemented up to the point where all necessary values for the triangular solves and matrix-vector multiplications could be fetched. The permutation algorithm for minimizing fetching and ghosting was still missing as well as the actual triangular solving and matrix-vector multiplying routines.
  The implemented sparse matrix manipulation routines include matrix (and vector) row and column permutations, row and column extractions, computing $A + A^T$ for a sparse matrix A (needed by Metis, a hypergraph partitioner, which only works for undirected graphs) and a neighborhood method that uses the breadth first search algorithm and some reachability order to find the set of all reachable vertices from a subset of given vertices in a graph. All these methods reside in the SparceUtils.cpp/.hpp files and have become mere artifacts.

- CA-GMRES may fail for some matrices with the given true solution described in Section 6, especially `Watt1`. In fact, finding a start vector that produces the results in Figures 3 and 4 was not easy. The issues for the `Watt1` matrix are known and can be addressed with the techniques described in [5] and [8]. Briefly those techniques comprise the computation of $2s$ Ritz values instead of s, perturbing the input of the Modified Leja ordering and equilibrating the matrix before anything else. Equilibration could help with other matrices for which the CA-GMRES methods may fail as well.

5.6. Test matrices

For the numerical and performance experiments the following test matrices were used.

Name	Dim	nnz	Type
watt1	1856	11550	unsymmetric
sherman3	5005	20K	unsymmetric
dmat	10K	10K	diagonal
bcsstk18	12K	149K	SPD
bmwst7_1	141K	3.7M	SPD
xenon2	157K	3.9M	unsymmetric
pwtk	218K	3.7M	SPD

6. Numerical experiments

This section provides a numerical comparison between CA-GMRES and standard GMRES. Hoemmen et al. [8] showed in their numerical experiments, that for the correct choice of basis and parameters of the restart length $m = s \cdot t$ the CA-GMRES converges in no more iterations than standard GMRES (with the same restart length) for almost all test cases. This is due to the fact, that the ability to choose the s-step basis length way shorter than the restart length improves stability and performance. They suggest $s = 5$ and $t = 12$ often gave the best performance. In order to recreate their results, mainly the same parameter settings, matrices and right hand side vectors were used.

Like in [8], the true solution $\hat{x}$ is generated with

$$\hat{x}(k) = u(k) + sin(2\pi k/n) \tag{6.1}$$

where the scalar $u(k)$ is chosen from a random uniform [-1, 1] distribution. Hoemmen et al. state, that $\hat{x}$ was chosen in this way because a completely random solution is usually nonphysical, but a highly nonrandom solution (such as a vector of all ones) might be near an eigenvector of the matrix (which would result in artificially rapid convergence of the iterative method).

6.1. Key for convergence plots

The x axis of the convergence plots shows the number of iterations of the Krylov methods tested. The y axis of the convergence plots shows the 2-norm of the residual $\|b - Ax_k\|_2$ relative to the 2-norm of the initial residual $\|r_0\|_2$ at iteration k. Since standard GMRES exposes the residual at every iteration, it is presented as a continuous line. The s-step based CA-GMRES variants are depicted as points since their residuals are exposed only after every s steps.

The terms in the legend are defined as follows.

GMRES(m) Standard GMRES with restart length m (the algorithm is outlined in Appendix A). Its convergence metric is always shown as a black line.

Monomial-GMRES(s,t) Monomial-based CA-GMRES with basis length s and restart length $m = s \cdot t$. Its convergence metric is plotted as a circle shaped dot with different colors for both s values shown in the plot.

Newton-GMRES(s,t) Newton-based CA-GMRES with basis length s and restart length $m = s \cdot t$. Its convergence metric is plotted as a triangle shaped dot with different colors for both s values shown in the plot.

All plots show the standard GMRES, two versions of the Monomial-based CA-GMRES and two versions of the Newton-based CA-GMRES with different basis lengths. All methods have the same restart length, i.e., GMRES(m) always uses $m = s \cdot t$ and $s_1 \cdot t_1$ is always equal to $s_2 \cdot t_2$ in the Newton-based resp. Monomial-based CA-GMRES versions. The quality of the s-step basis for each method is shown as well. Hoemmen et al. [8] compute the 2-norm condition number for the matrix $\underline{V}_k$ at iteration k, while an estimate for the reciprocal of the infinity-norm condition number $\mathcal{K}_\infty(\underline{\acute{V}}_k)$ is used here. Among all computed estimates for $\underline{\acute{V}}_k$ the minimum and maximum values are shown in the legend.

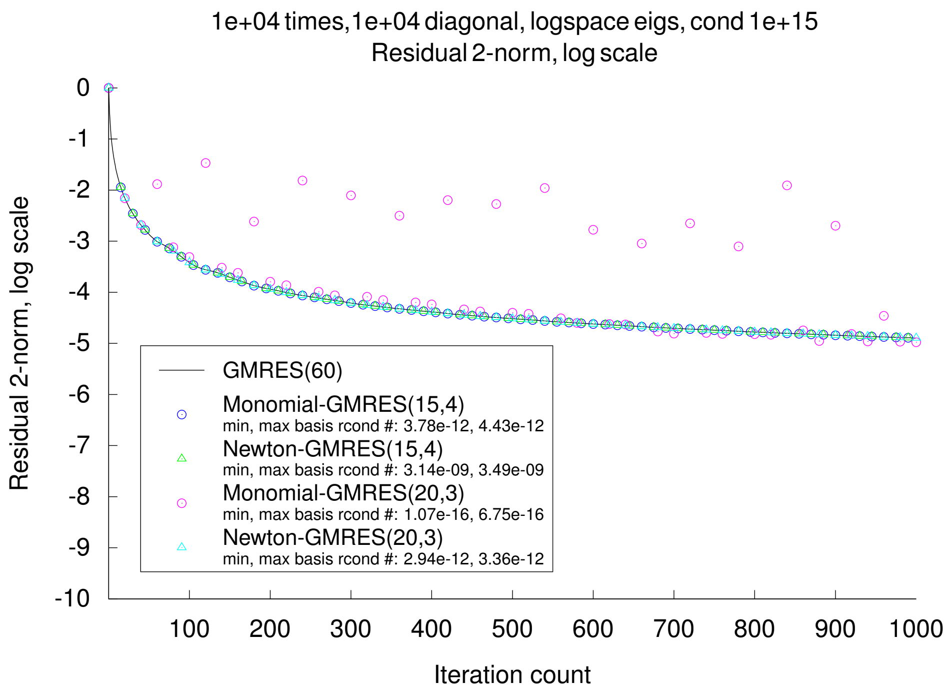


Figure 2: Convergence plot for a positive diagonal matrix with a 2-norm condition number $\mathcal{K}$ and logarithmically spaced eigenvalues between 1 and $1/\mathcal{K}$.

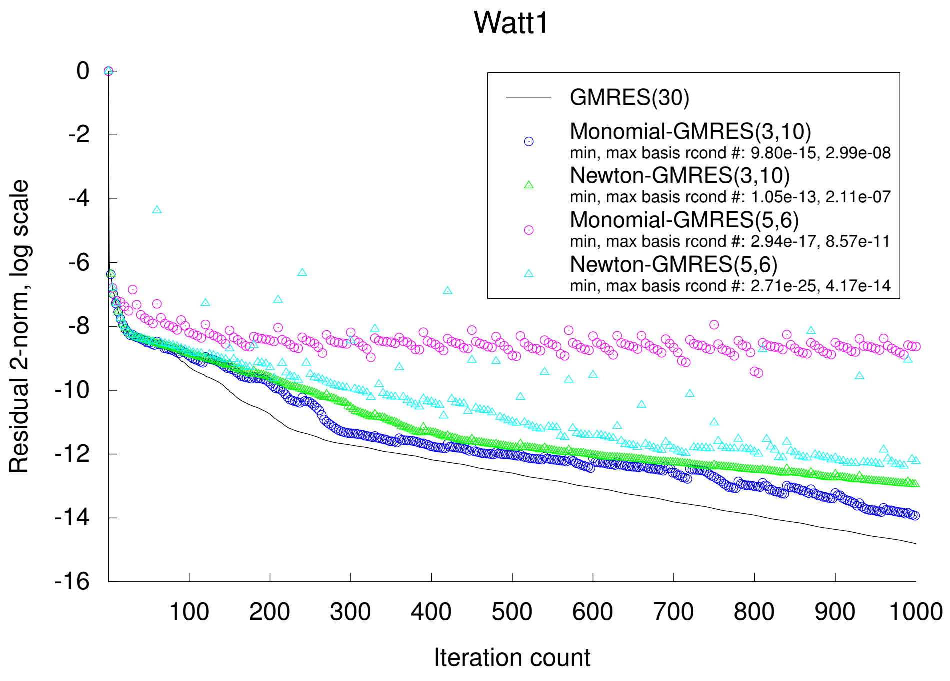


Figure 3: Without basis vector scaling

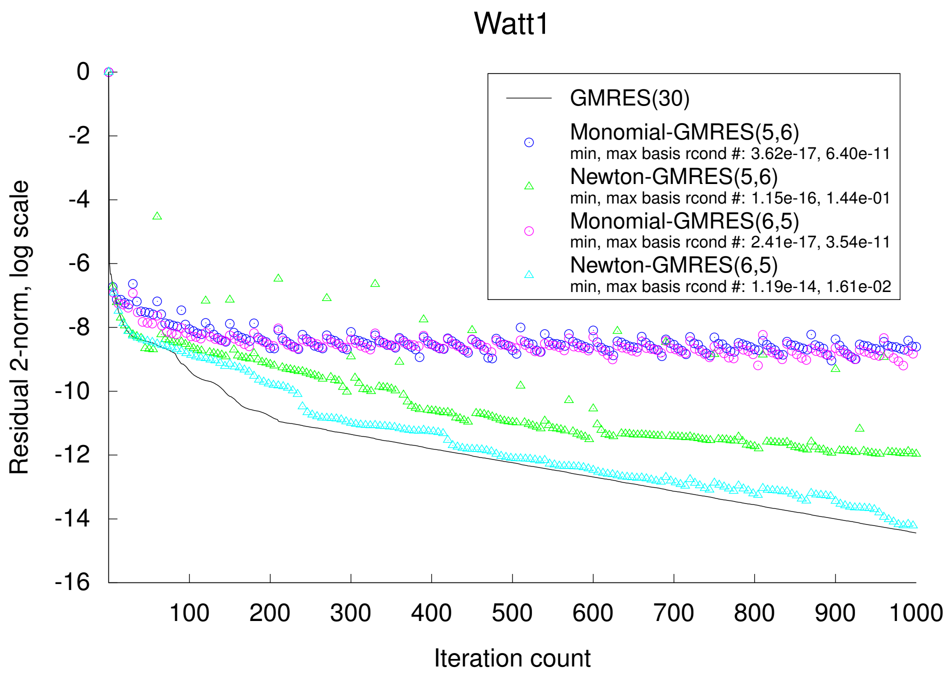


Figure 4: With basis vector scaling

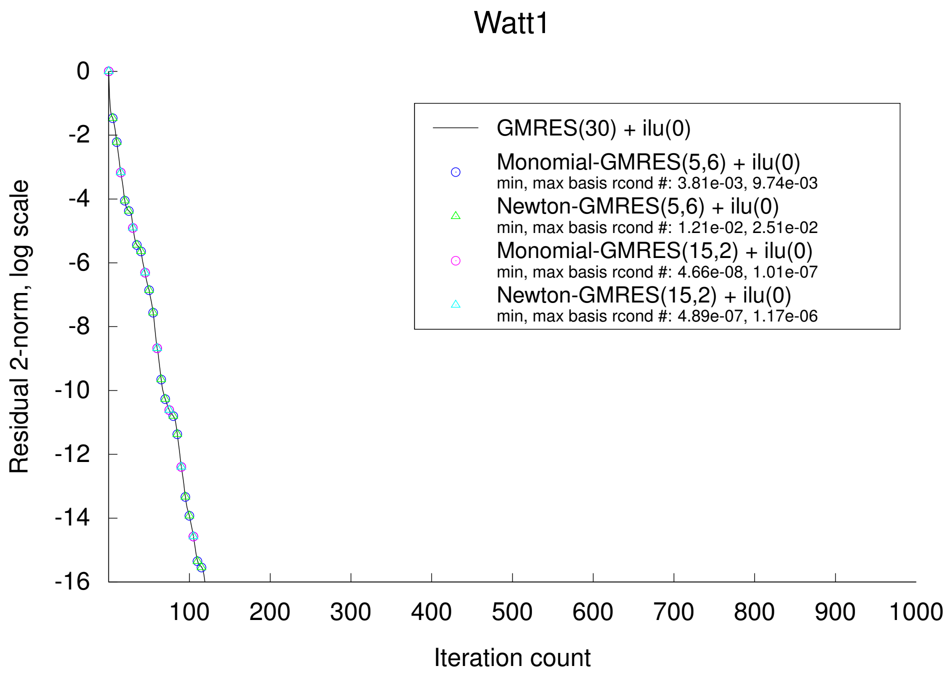


Figure 5: Without ILU(0) preconditioner

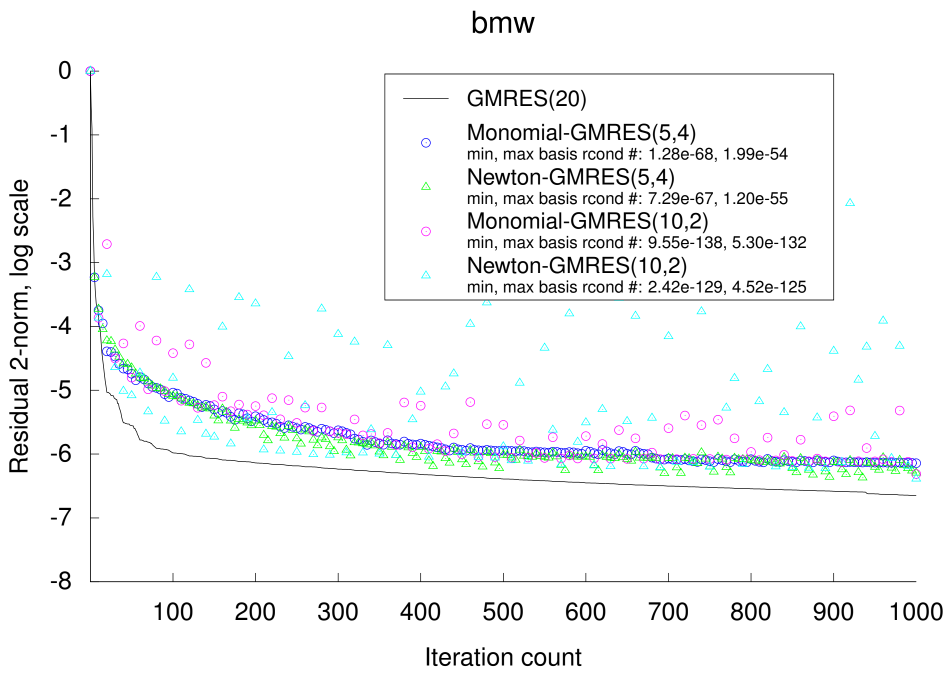


Figure 6: Without basis vector scaling

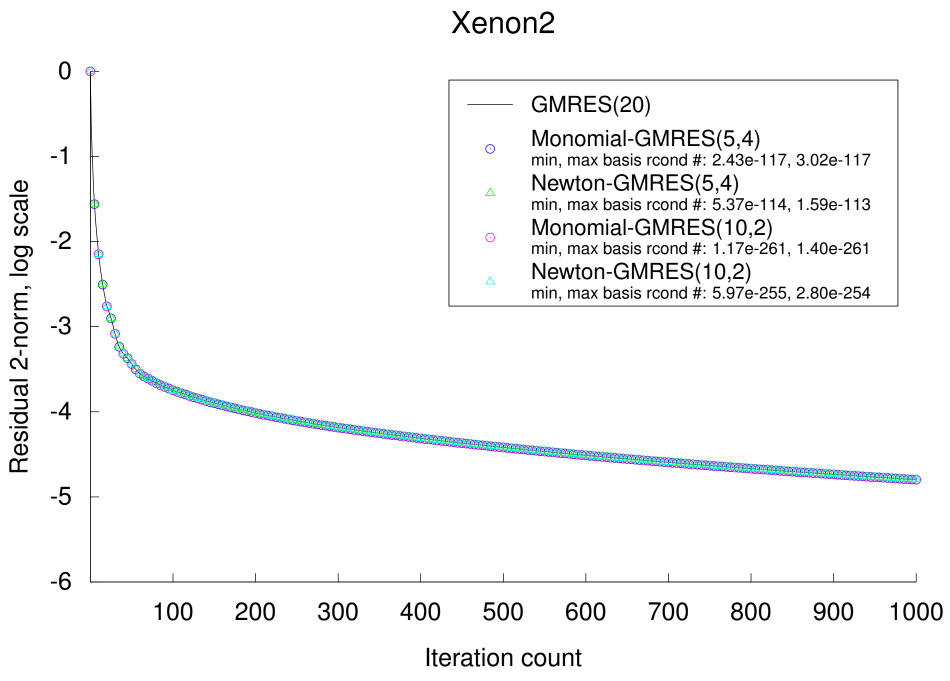


Figure 7: Without basis vector scaling

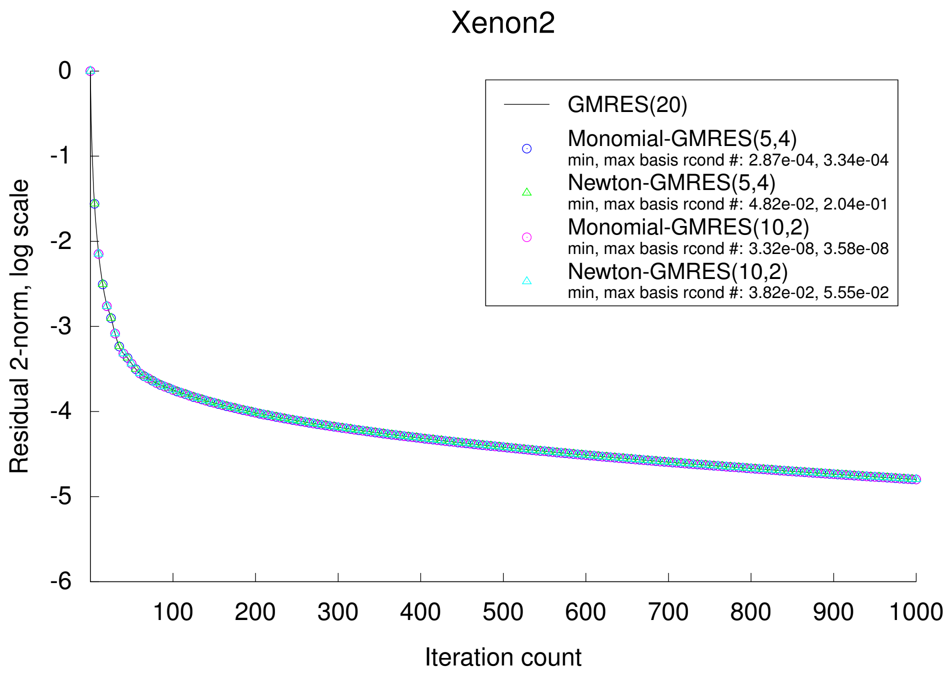
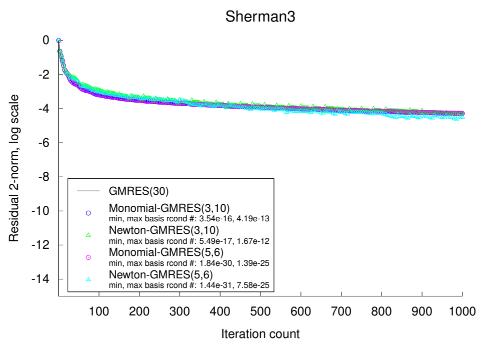
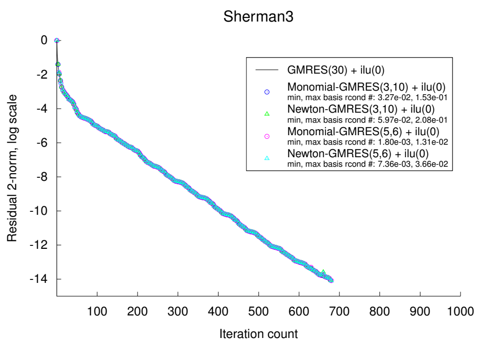


Figure 8: With basis vector scaling

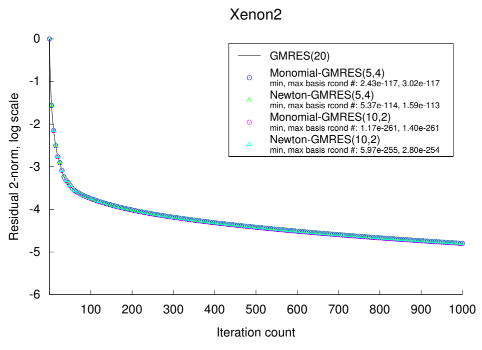


(a) Without preconditioner

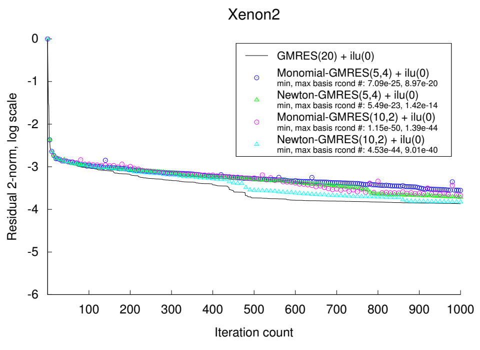


(b) With ILU(0) preconditioner

Figure 9: A case where ILU(0) is in favor of the convergence behavior for all tested methods.

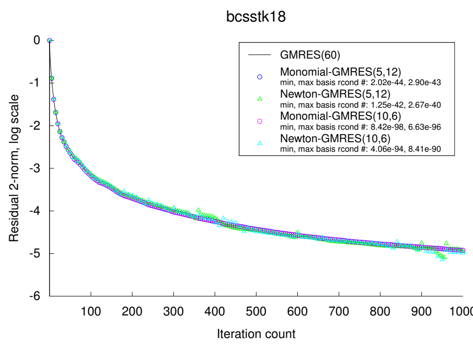


(a) Without preconditioner

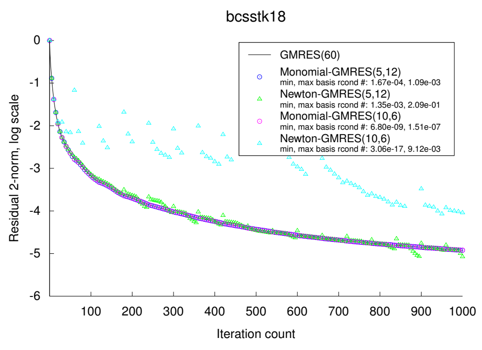


(b) With ILU(0) preconditioner

Figure 10: A case where ILU(0) adversely affects the convergence behavior for all tested methods.

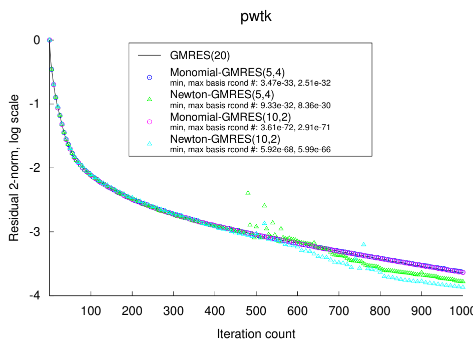


(a) Without basis vector scaling

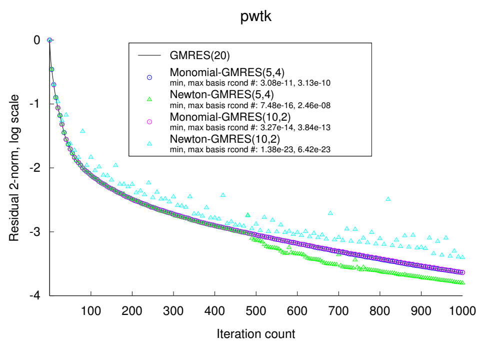


(b) With basis vector scaling

Figure 11: Scaling the basis vectors does not help to stabilize the Newton-GMRES with the short basis length $s = 5$. It even significantly magnifies the convergence instability in the Newton-GMRES with longer basis length $s = 10$.



(a) Without basis vector scaling



(b) With basis vector scaling

Figure 12: In the unscaled case in figure 12a, the Newton-GMRES with the shorter basis length $s = 5$ is slightly unstable between 500 and 600 iterations. The scaling in figure 12b fixes this, but drastically worsens the overall convergence for the Newton-GMRES with longer basis length $s = 10$.

6.2. Results for various matrices

6.2.1. Diagonal matrices

CA-GMRES was tested with $10K \times 10K$ positive diagonal matrices (`Dmat`) with three different condition numbers 1e+05, 1e+10 and 1e+15. Figure 2 shows how the Monomial basis fails to converge for larger s. Although the constructed matrices are the same as in Hoemmen et al. [8] the convergence for the Monomial basis looks slightly different here because Hoemmen et al. do not start with standard Anoldi at every outer iteration. Interestingly, the Arnoldi residual converges further after the first inner iteration (it most definitely diverges from the true residual at this point). Only the next restart cycle exposes the true and large residual in every third iteration. This phenomenon was observed by Greenbaum et al. [6] when using a Householder-based GMRES instead of the MGS-based approach. The TSQR kernel in CA-GMRES uses Householder QR factorizations and therefore, might be the cause of this behavior.

6.2.2. Matrices from applications

`Watt1` is a small but very difficult matrix with two clusters of eigenvalues centered around one and zero. This impedes the computation of a good and linearly independent Newton basis. Heommen et al. used equilibration for numeric stability and could eventually converge with the same rate as standard GMRES by using a smaller step size as well. The results presented in Figures3 and 4 show similar problems for the unequilibrated case. Scaling the basis vectors by its 2-norm was helpful in this case and a larger step size could be chosen eventually. However the most effective way was to use the ILU(0) preconditioner (Figure 5). Even the Monomial basis with a very large step size converged at the same rate as standard GMRES.

`Bmw7st_1` (Figure 6) is the next difficult matrix that Hoemmen et al. could only fix with equilibration. The matrix is badly scaled which made the equilibration process challenging as well. The `Bmw` matrix is another good example for how the CA-GMRES approach may fail. However, applying the ILU(0) preconditioner lets the method converge in one step for standard GMRES and in s steps for CA-GMRES.

`Xenon2` is a good example for when the CA-GMRES works best. Figure 7 shows the convergence plot with unscaled basis vectors. Although the reciprocal condition number is extremely close to zero the convergence rate behaves rather well. This indicates that the vectors must be still linearly independent. Scaling the basis vectors as in Figure 8 drastically improves the condition and leaves the convergence rate unchanged.

After Hoemmen et al. used equilibration on the `Xenon2` matrix, the convergence rate started to become unstable. Something similar can be observed with the ILU(0) preconditioner in Subfigure 10b.
In general, Figure 10 shows two examples for when the ILU(0) improves the convergence rate and when it doesn't. Also, Figure 12 discusses observed issues with basis vector scaling.

7. Performance experiments

This section describes performance experiments that compare the shared-memory implementation of CA-GMRES with the standard implementation GMRES(m) on the harris platform.

7.1. Key for speedup plots

The left bars show the stacked relative runtimes for the kernels used in CA-GMRES. Those kernels comprise the first inner iteration (INIT), the sparse matrix-vector multi-

plications for creating the basis vectors (SpMV), the QR factorization of V (TSQR), the orthogonalization against previous basis vectors (Block Gram-Schmidt) and the update of the upper Hessenberg Matrix (small dense operations). The right bars depict the stacked relative runtimes of the kernels used in GMRES(m). These kernels include the sparse matrix-vector products (SpMV) and the modified Gram-Schmidt process (MGS). Solving the least squares problem at every outer iteration k is not included since this kernel is the same for both methods. There is a difference on how the Givens rotations are applied. However, the cost for these Givens rotations is only marginal and therefore, negligible.
The numbers above the CA-GMRES bars indicate the speedup over standard GMRES. In some cases, the relative forward error was computed in order to verify the result of the computed solution. Here, the relative forward error is defined by

$$\frac{\|x - \hat{x}\|_2}{\|\hat{x}\|_2}$$

where x is the approximation and $\hat{x}$ is the true solution for the system.

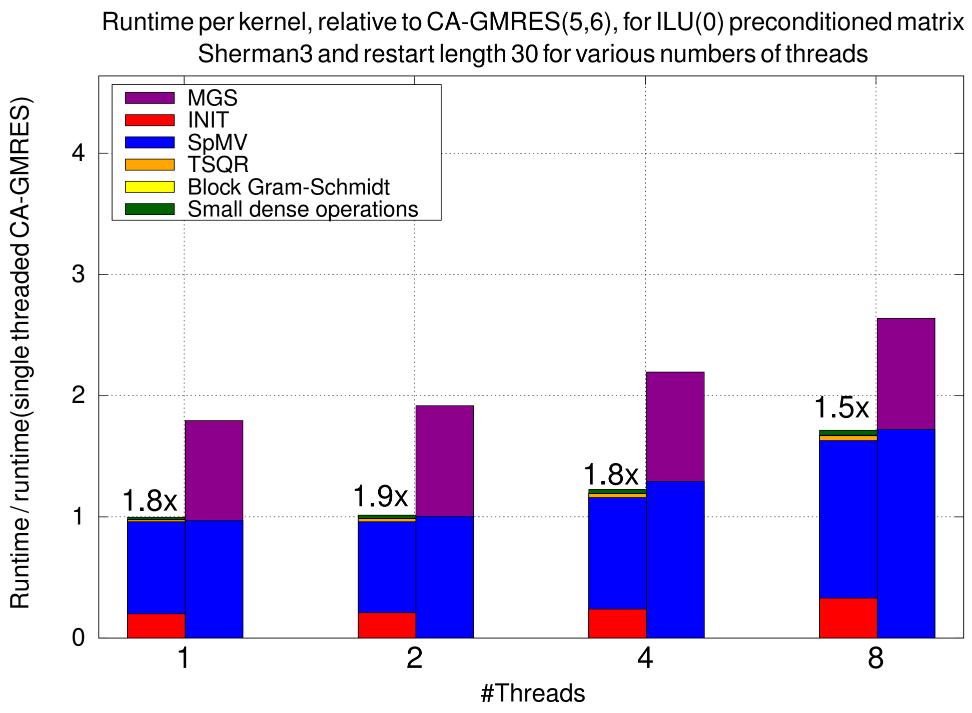


Figure 13: This plot presents the runtimes for different numbers of threads for a small sparse preconditioned matrix (`Sherman3`). The runtimes for different threads are scaled by the runtime of the single threaded CA-GMRES. The tolerance ϵ for the convergence metric was set to $1\mathrm{e}-14$. The convergence plot of this matrix is depicted in Figure 9b. CA-GMRES converged at iteration 680, GMRES(m) converged at iteration 677. The relative forward error CA-GMRES is $6.61\mathrm{e}-12$, the relative forward error for GMRES(m) is $7.97\mathrm{e}-12$. Computations were done 10 times and the average was taken.

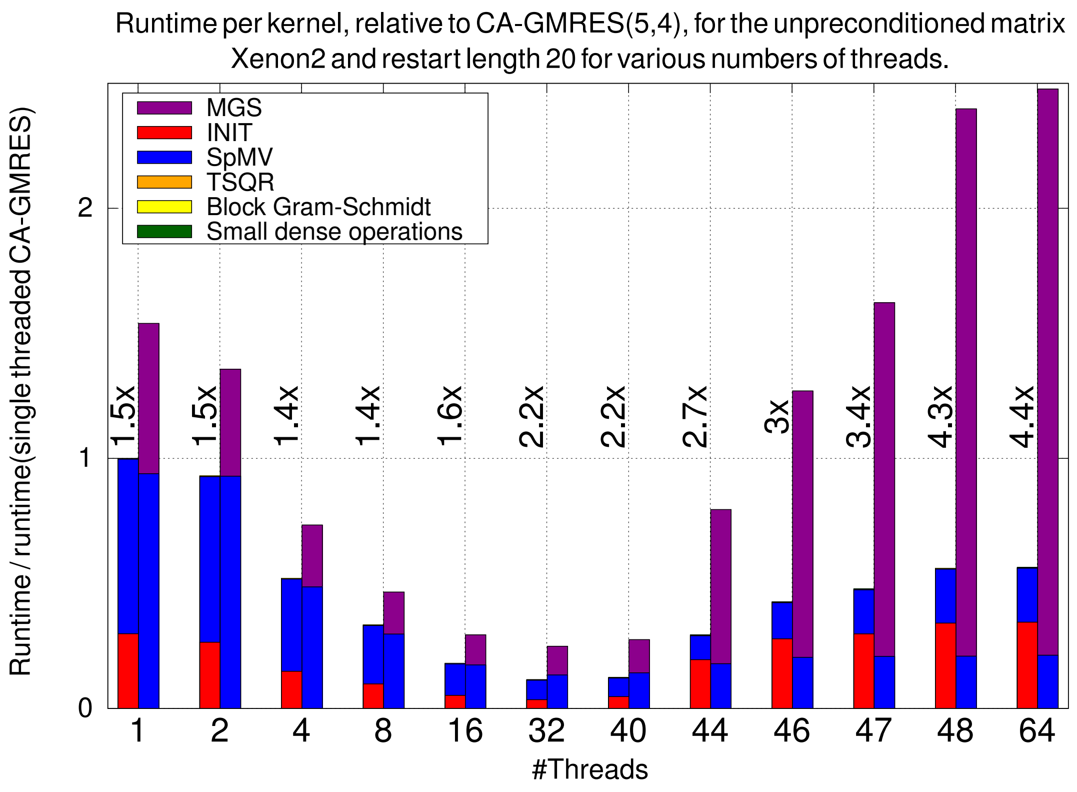


Figure 14: This plot presents the runtimes for different numbers of threads for a large sparse unpreconditioned matrix (Xenon2). The runtimes for different threads are scaled by the runtime of the single threaded CA-GMRES. The tolerance ϵ for the convergence metric was set to $5\text{e}-5$. This is too high for practical applications but it is needed in order to let the method converge in under 1000 iterations (see Figures 7,8 for convergence plots). Both methods converged at iteration 390 and both methods have a very high relative forward error with $8.3951\text{e}+23$. The computations were done 10 times for each thread and the average runtime was taken.

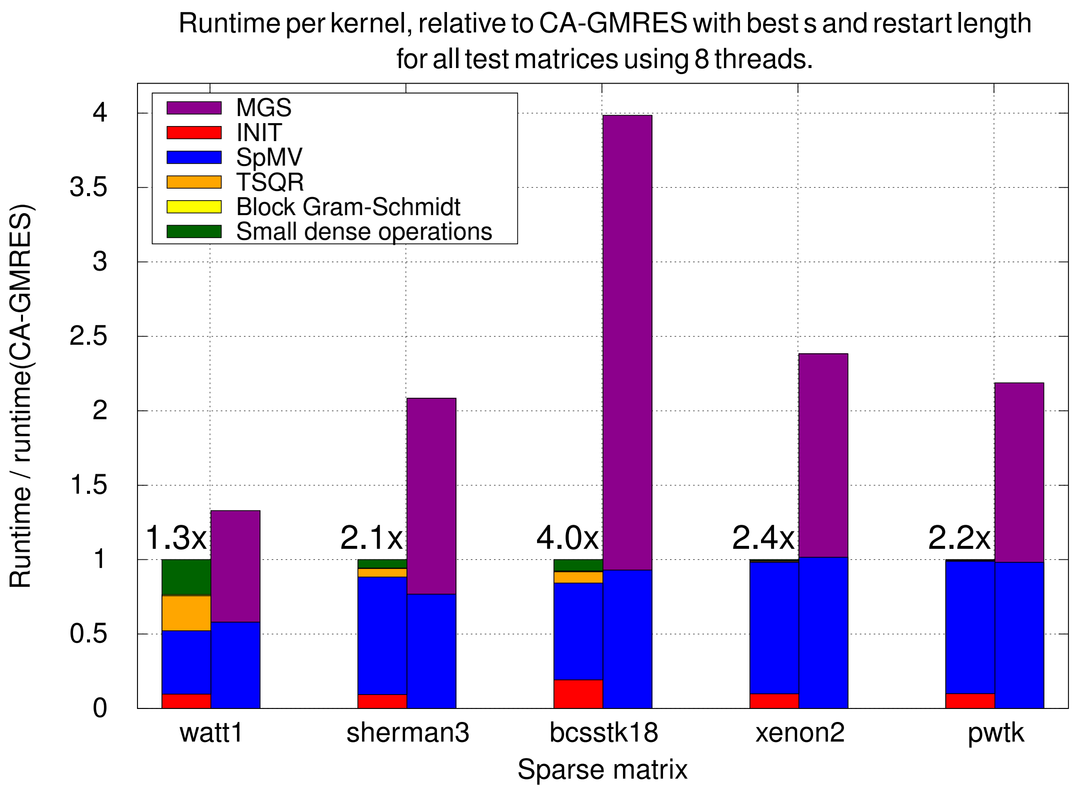


Figure 15: This plot shows the relative runtime for different matrices with optimal basis and restart lengths (not indicated) on 8 cores. The convergence metric was disabled and 1000 iterations were performed each. The matrices are presented in ascending order from left to right according to their size.

7.2. Results for various matrices

Figure 13 shows an almost constant speedup factor of around 1.8 between methods for different threads, but indicates speedups < 1 between same methods with increasing thread size. The dimension of the matrix used (`Sherman3`) is 5K and therefore, rather small. Using more than one thread is obviously not appropriate in this case.
Larger matrices like `Xenon2` with dimension 157K however do benefit from multiple threads. Figure 14 suggests around 32 to 40 threads would be optimal for a matrix of this size. Notice that, on the far right of the plot, more threads than available cores were used which further adds more overhead to the standard implementation rather than CA-GMRES.
Figure 15 shows the relative runtimes for three small matrices (`Watt1`, `Sherman3` and `Bcsstk18`) and two large ones (`Xenon2` and `Pwtk`). The different kernels in CA-GMRES are best visible in the smallest matrix, namely `Watt1`. However, even here the Block Gram-Schmidt process is too small to notice. This could be due to the fact that MKL is highly optimized for dense matrix-matrix operations. `Bcsstk18` has far more nonzeros

compared to the other small matrices which might correlate with the significant speedup for CA-GMRES over the standard implementation.
The dominant kernel in CA-GMRES for large matrices is obviously SpMV. This is the only kernel that was not optimized (i.e., replaced with the matrix powers kernel from Heommen et al. [8]). Also, as is discussed in Section 5.5, the INIT kernel may be reduced by shifting the work to the TSQR and 'small dense operations' kernels.

8. Summary and Conclusion

The numerical results show that CA-GMRES converges like standard GMRES under these conditions.

- The right basis length s is chosen.
- The right s-step basis is used.
- Preconditioners (e.g. equilibration, etc.) are used.

Basis vector scaling was done in Bai et al. [2] and Erhel [5]. The convergence plots show that scaling the basis vectors by its 2-norm may exacerbate convergence instabilities and, therefore, is not recommended. Hoemmen et al. [8] use matrix equilibration instead.
The speedup plots clearly show the absence of the matrix powers kernel, especially for large matrices. Also, restarting with standard GMRES in every outer iteration of CA-GMRES may not be optimal and leaves room for optimization.

References

[1] Walter Edwin Arnoldi. The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of Applied Mathematics*, 9(1):17 – 29, 1951.

[2] Z. BAI, D. HU, and L. REICHEL. A newton basis gmres implementation. *IMA Journal of Numerical Analysis*, 14(4):563–581, 1994.

[3] J. Demmel, M. Hoemmen, Y. Hida, and E. Riedy. Nonnegative diagonals and high performance on low-profile matrices from householder qr. *SIAM Journal on Scientific Computing*, 31(4):2832–2841, 2009.

[4] James Demmel, Laura Grigori, Mark Hoemmen, and Julien Langou. Communication-optimal parallel and sequential qr and lu factorizations. *SIAM J. Sci. Comput.*, 34(1):206–239, February 2012.

[5] Jocelyne Erhel. A parallel gmres version for general sparse matrices. *Electronic Transactions on Numerical Analysis*, 3:160–176, 1995.

[6] A. Greenbaum, M. Rozloznik, M. Rozlo Zn Ik, and Z. Strakos. Numerical behaviour of the modified gram-schmidt gmres implementation, 1997.

[7] Laura Grigori and Sophie Moufawad. Communication avoiding ilu0 preconditioner. *SIAM Journal on Scientific Computing*, 37:C217–C246, 04 2015.

[8] Mark Hoemmen. *Communication-avoiding Krylov Subspace Methods.* PhD thesis, Berkeley, CA, USA, 2010. AAI3413388.

[9] Wayne D. Joubert and Graham F. Carey. Parallelizable restarted iterative methods for nonsymmetric linear systems. part i: Theory. *International Journal of Computer Mathematics*, 44(1-4):243–267, 1992.

[10] S K. Kim and A Chronopoulos. An efficient parallel algorithm for extreme eigenvalues of sparse nonsymmetric matrices(please reference in your papers). *International Journal of High Performance Computing Applications - IJHPCA*, 6:98–111, 12 1992.

[11] D. Nuentsa Wakam and G.-A. Atenekeng Kahou. Parallel GMRES with a multiplicative schwarz preconditioner. *ARIMA*, 14:81–99, 2011.

[12] Lothar Reichel. Newton interpolation at leja points. *BIT*, 30(2):332–346, 1990.

[13] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, July 1986.

[14] H. Walker. Implementation of the gmres method using householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 9(1):152–163, 1988.

List of Algorithms

Appendices

A. GMRES(m)

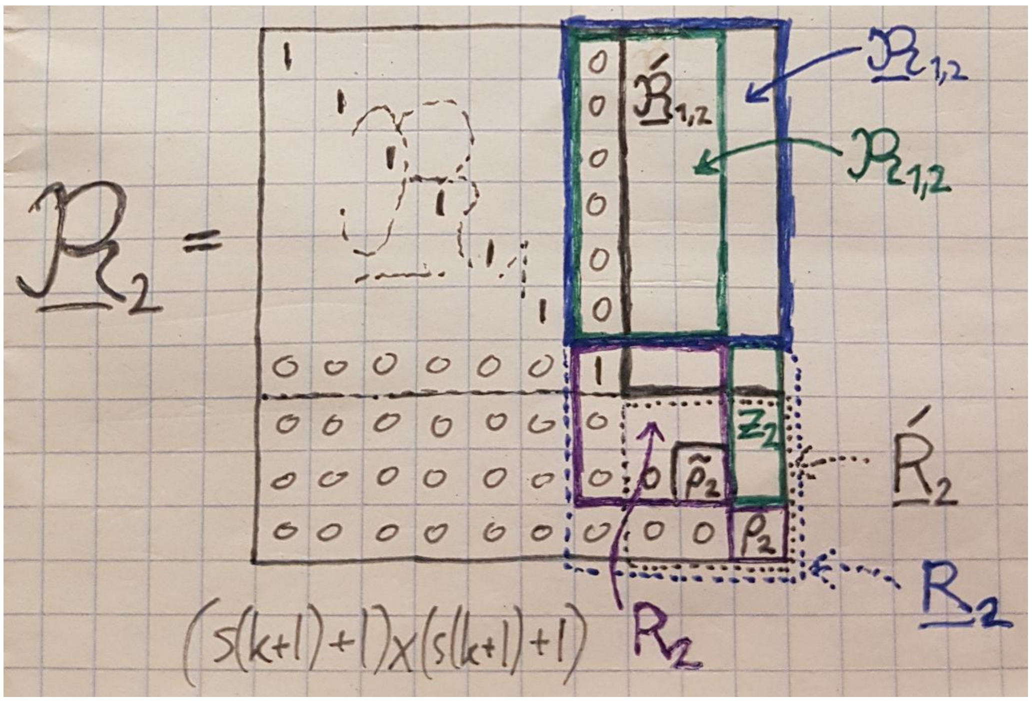
B. TODO

- describe ILU(0)
- write related work

Algorithm 5 GMRES(m)

```
Input: $n \times n$ linear system $Ax = b$ and initial guess $x_0$
 1: restart := true
 2: while restart do
 3:   $r_0 := b - Ax_0$, $\beta := \|r_0\|_2$, $q_0 := r_0/\beta$, $\underline{Q}_0 := q_0$, $\underline{H}_0 := \varnothing$
 4:   for $k = 1$ to $m$ do
 5:     Compute $q_k$ and $h_k$ using Algorithm 2
 6:     Set $\underline{Q}_k := [Q_{k-1}, q_k]$ and $\underline{H}_k := [\underline{H}_{k-1}, h_k]$
 7:     Reduce $h_k$ of $\underline{H}_k$ from upper Hessenberg to upper triangular form using $k$
          Givens rotations $G_1$, $G_2$, ..., $G_k$. Apply the same rotations in the same order
          to $\beta e_1$, resulting in the length $k + 1$ vector $\zeta_k$.
 8:     Element $k + 1$ of $\zeta_k$ is the 2-norm (in exact arithmetic) of the current residual
          $r_{k+1} = b - Ax_{k+1}$ of the current solution $x_{k+1}$.
 9:     if converged then
10:       restart = false, and exit for loop
11:     end if
12:   end for
13:   Use the above reduction of $\underline{H}_k$ to upper triangular form and $\zeta_k$ to solve $y_k :=$
        $\text{argmin}_y \|\underline{H}_k y - \beta e_1\|_2$
14:   Set $x_0 := x_0 + Q_k y_k$
15: end while
```

C. Visualizing the R matrix



$\underline{\mathcal{R}}_2 =$
$\underline{\mathcal{R}}_{1,2}$
$\mathcal{R}_{1,2}$
$\acute{\underline{\mathcal{R}}}_{1,2}$
$\mathcal{R}_1$
Z_2
$\tilde{\rho}_2$
ρ_2
$\acute{R}_2$
$\underline{R}_2$
R_2
$(s(k+1)+1) \times (s(k+1)+1)$

D. Modified Leja Ordering

Algorithm 40 Modified Leja ordering

Input: n unique shifts $z_1, \ldots, z_n$, ordered so that any complex shifts only occur consecutively in complex conjugate pairs $z_k, z_{k+1} = \overline{z_k}$, with $\Im(z_k) > 0$ and $\Im(z_{k+1}) < 0$
Input: Each shift z_k has multiplicity μ_k (a positive integer)
Output: n unique shifts $\theta_1, \ldots, \theta_n$, which are the input shifts arranged in the modified Leja order
Output: `outList`: array of indices such that for $k = 1, \ldots, n$, $\theta_k = z_{\texttt{outList}(k)}$
Output: C: estimate of the capacity of $\{z_1, \ldots, z_n\}$

```
1:  $C \leftarrow 1$                                              ▷ Initial capacity estimate
2:  Let $k$ be the least index $j$ maximizing $|z_j|$
3:  $\theta_1 \leftarrow z_k$, and outList ← $[k]$
4:  if $\Im(z_k) \neq 0$ then
5:      If $\Im(z_k) < 0$ or $k = n$, error: Input out of order
6:      $\theta_2 \leftarrow z_{k+1}$, and outList ← [outList, $k+1$]
7:      $L \leftarrow 2$
8:  else
9:      $L \leftarrow 1$
10: end if

11: while $L < n$ do
12:     $C' \leftarrow C$, and $C \leftarrow \prod_{j=1}^{L-1} |\theta_L - \theta_j|^{\mu_{\texttt{outList}(j)}/L}$ ▷ Update capacity estimate. outList($j$)
          is the multiplicity of $\theta_j$.
13:     for $j = 1, \ldots, n$ do                                 ▷ Rescale all the shifts
14:         $z_j \leftarrow z_j/(C/C')$, and $\theta_j \leftarrow \theta_j/(C/C')$
15:     end for
16:     Let $k$ be the least index $k$ in $\{1, \ldots, n\} \setminus$ outList maximizing $\prod_{j=1}^{L-1} |z_k - \theta_j|^{\mu_{\texttt{outList}(k)}}$
17:     If the above product overflowed, signal an error.
18:     If the above product underflowed, randomly perturb $z_1, \ldots, z_n$ and start over from
          scratch. Keep track of the number of times this happens, and perturb by a
          larger amount each time. Signal an error if the number of restarts exceeds a
          predetermined bound.
19:     $\theta_{L+1} \leftarrow z_k$, and outList ← [outList, $k$]
20:     if $\Im(z_k) \neq 0$ then
21:         If $\Im(z_k) < 0$ or $k = n$, error: Input out of order
22:         $\theta_{L+2} \leftarrow z_{k+1}$, and outList ← [outList, $k+1$]
23:         $L \leftarrow L + 2$
24:     else
25:         $L \leftarrow L + 1$
26:     end if
27: end while
```