Communication-Avoiding Krylov Subspace Methods in Theory and Practice

Erin Carson

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-179
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-179.html

August 5, 2015

Acknowledgement

I acknowledge under the Dept. of Defense, Air Force Office of Scientific Research, NDSEG Fellowship, 32 CFR 168a, as well as support from the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Numbers DE-SC0004938, DE-SC0003959, and DE-SC0010200; from the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, X-Stack program under Award Numbers DE-SC0005136, DE-SC0008699, DE-SC0008700, and AC02-05CH11231; from DARPA Award Number HR0011-12-2-0016, as well as contributions from Microsoft (award 024263), Intel (award 024894), UC Discovery (award DIG07-10227), and from affiliates National Instruments, Nokia, NVIDIA, Oracle, Samsung, and MathWorks.

Communication-Avoiding Krylov Subspace Methods in Theory and Practice

by

Erin Claire Carson

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

and the Designated Emphasis

in

Computational and Data Science and Engineering

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor James W. Demmel, Chair
Professor Armando Fox
Professor Ming Gu

Summer 2015

Communication-Avoiding Krylov Subspace Methods in Theory and Practice

Abstract

Communication-Avoiding Krylov Subspace Methods in Theory and Practice

by

Erin Claire Carson

Doctor of Philosophy in Computer Science

with a Designated Emphasis in Computational and Data Science and Engineering

University of California, Berkeley

Professor James W. Demmel, Chair

Advancements in the field of high-performance scientific computing are necessary to address the most important challenges we face in the 21st century. From physical modeling to large-scale data analysis, engineering efficient code at the extreme scale requires a critical focus on reducing communication – the movement of data between levels of memory hierarchy or between processors over a network – which is the most expensive operation in terms of both time and energy at all scales of computing. Achieving scalable performance thus requires a dramatic shift in the field of algorithm design, with a key area of innovation being the development of *communication-avoiding* algorithms.

Solvers for sparse linear algebra problems, ubiquitous throughout scientific and mathematical applications, often limit application performance due to a low computation/communication ratio. Among iterative methods, Krylov subspace methods are the most general and widely-used. To alleviate performance bottlenecks, much prior work has focused on the development of communication-avoiding Krylov subspace methods, which can offer asymptotic performance improvements over a set number of iterations.

In finite precision, the convergence and stability properties of classical Krylov methods are not necessarily maintained by communication-avoiding Krylov methods. Depending on the parameters used and the numerical properties of the problem, these communication-avoiding variants can exhibit slower convergence and decreased accuracy compared to their classical counterparts, making it unclear when communication-avoiding Krylov subspace methods are suitable for use in practice.

Until now, the literature on communication-avoiding Krylov methods lacked a detailed numerical stability analysis, as well as both theoretical and practical comparisons with the stability and convergence properties of standard implementations. In this thesis, we address this major challenge to the practical use of communication-avoiding Krylov subspace methods. We extend a number of theoretical results and algorithmic techniques developed for classical Krylov subspace methods to communication-avoiding Krylov subspace methods and identify constraints under which these methods are competitive in terms of both achieving asymptotic speedups and meeting application-specific numerical requirements.

To my fellow Drs. Carson, Bernard and Scott.

Contents

List of Figures

List of Tables

Acknowledgments

Thank you to my advisors James Demmel and Armando Fox, my committee member Ming Gu, and my collaborators, colleagues, and fellow beboppers, including Grey Ballard, Aydin Buluc, Jong-Ho Byun, Razvan Carbunescu, Orianna DeMasi, Aditya Devarakonda, Michael Driscoll, Marquita Ellis, Andrew Gearhart, Evangelos Georganas, Pieter Ghysels, Laura Grigori, Mark Hoemmen, Nick Knight, Penporn Koanantakool, Marghoob Mohiyuddin, Hong Diep Nguyen, Rebecca Roelofs, Cindy Rubio Gonzalez, Oded Schwartz, Harsha Simhadri, Edgar Solomonik, Brian Van Straalen, Sam Williams, and Kathy Yelick. I would also like to thank my family, my friends, and my dog Blue for their continuous love and support during my graduate career.

I acknowledge Government support under and awarded by the Department of Defense, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a, as well as support from the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Numbers DE-SC0004938, DE-SC0003959, and DE-SC0010200; from the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research, X-Stack program under Award Numbers DE-SC0005136, DE-SC0008699, DE-SC0008700, and AC02-05CH11231; from DARPA Award Number HR0011-12-2-0016, as well as contributions from Microsoft (award 024263), Intel (award 024894), UC Discovery (award DIG07-10227), and from affiliates National Instruments, Nokia, NVIDIA, Oracle, Samsung, and MathWorks.

Chapter 1

Introduction

1.1 The Importance of Computational Science

The field of computational science has revolutionized the way science and engineering are done in the world today. Alongside theoretical and experimental science, computational science is now widely regarded as an essential 'third pillar' of scientific exploration, which will become increasingly important in driving scientific and technological progress. Computational science has enabled the construction and validation of new theoretical models of phenomena that are impossible to study in an experimental setting, for example, studying dark energy and the origin of the universe. Using computational models and simulations can also reduce the cost and time required to design and test products and procedures, from airplanes to medical treatments to renewable energy sources.

The field of computational science is a multidisciplinary area, combining research in computer science, applied mathematics, and domains in the natural, physical, and social sciences. Research in this field focuses on how to build models efficiently under constraints of limited computational power and computer memory as well as how to determine if the model and simulation are accurate enough to be reliable.

As computational models and simulations are increasingly being used for decision-making purposes, and have become an indispensable tool in design and manufacturing processes and for scientific research as well as policy-making, it is increasingly important to obtain reliable and accurate results for realistic physical systems at scale. Running the extreme-scale simulations needed to address the most important challenges we face in the 21st century, including climate modeling, combustion/efficient fuel design, computer-aided design of vehicles and airplanes, design of medical devices and diagnostic procedures, and genomic sequencing, will require major advances in both computer hardware and software [8, 65].

1.2 The Need for Communication-Avoiding Algorithms

On modern computers, the time to complete a floating point operation can be orders of magnitude less than the time to perform communication, i.e., move data between levels of the memory hierarchy or between parallel processors over a network. This gap is expected to grow in future systems [8, 65]. Traditionally, algorithm performance has been gauged in terms of computational complexity; conventional wisdom tells us that the fewer arithmetic or logical operations an algorithm performs, the faster it will be. On today's computer architectures, this notion is far from accurate. Instead, *communication complexity* is a much better indication of both runtime and energy cost.

To be effective in enabling scientific discovery and analysis, many scientific computing applications require simulations at increasingly large scales. This necessitates performing computations that do not fit in cache and/or require running on large parallel machines with multiple processors, which in turn necessitates costly communication. As the size of the problem and the number of processors is increased, these algorithms suffer a rapid degradation of performance. Some computations, including those to study climate change and the origin of the universe, are so large they must be run on peta- or exascale supercomputers. In these cases, time spent communicating data is not just a hindrance to performance, but is prohibitive in terms of both time and energy cost.

Engineering efficient code at the extreme scale thus requires a paradigm shift in the design of high-performance algorithms: to achieve efficiency, one must focus on strategies which minimize data movement rather than minimize arithmetic operations to fully exploit the computational power of large-scale systems. We call this a *communication-avoiding* approach to algorithm design.

1.3 Iterative Linear Algebra at Scale

Algorithms for solving linear systems $Ax = b$ and eigenvalue problems $Ax = \lambda x$ can be classified as either *direct* methods, which perform a fixed number of steps to obtain a solution, or *iterative* methods, which repeatedly refine a candidate solution until the given stopping criteria are met. Iterative methods are commonly used when the matrix is too large and sparse to be solved using direct methods, when only a partial answer is required, or when accuracy less than machine precision is acceptable. In this thesis, we focus on iterative methods for solving sparse linear systems and eigenvalue problems, which constitute the core computational kernels in a wide variety of application areas, including materials science [150], biology [203], structural engineering [6], and combustion and energy modeling [25], as well as areas such as computational statistics [75, 76], Markov modeling [20], unconstrained optimization [169], image recognition [40], and computational finance [15].

The most general and flexible class of iterative methods is Krylov subspace methods (KSMs) [153, 154, 87, 123, 124, 180, 119, 17]. These methods are based on vector projection

onto expanding subspaces, where, in each iteration k, the solution update is chosen from the expanding Krylov subspace

$$\mathcal{K}_k(A,\, v) = \text{span}\{v,\, Av,\, A^2v, ...,\, A^{k-1}v\}, \tag{1.1}$$

where v is a starting vector chosen by the algorithm or input by the user. Many variants of Krylov subspace methods exist, each with different minimization properties and varying storage requirements (see, e.g., [153]).

Classical implementations of KSMs require one or more sparse matrix-vector multiplications (SpMVs) and one or more inner product operations in each iteration. These computational kernels are both communication-bound on modern computer architectures. To perform an SpMV, each processor must communicate some entries of the source vector it owns to other processors in the parallel algorithm, and in the sequential algorithm the matrix A must be read from slow memory in the common case when A is too large to fit in fast memory. Inner products involve a global reduction in the parallel algorithm, and a number of reads and writes to slow memory in the sequential algorithm (depending on the size of the vectors and the size of the fast memory).

Researchers have thus sought to remove the communication bottleneck in KSMs by reorganizing the algorithms to minimize data movement. These variants are referred to as communication-avoiding KSMs (CA-KSMs), or s-step KSMs; we will use these terms interchangeably. We summarize the relevant literature in Section 2.4. The motivating insight is that temporal locality can be improved by unrolling the iteration loop a factor of $O(s)$ times and breaking the dependencies between the SpMVs and vector operations in each iteration. For a fixed number of iterations, this theoretically allows for an $O(s)$ reduction in communication cost. It has been shown that such communication-avoiding formulations can result in significant speedups per iteration for many problems [127, 190].

Improving the speed per iteration, however, is not sufficient to guarantee speedups in a practical setting. When we consider the overall performance of iterative methods in practice, we must consider both

1. the *time per iteration*, which depends on the required kernels and the particular machine parameters, as well as the matrix structure and partition, and,

2. the *number of iterations required for convergence*, i.e., until the stopping criteria are met.

This second quantity depends on the conditioning and eigenvalue distribution of the system as well as round-off error in finite precision. In practice, round-off errors resulting from finite precision computation can heavily influence the rate of convergence; for example, in the Lanczos method for solving the symmetric eigenproblem, large round-off errors can lead to the loss of orthogonality between the computed vectors, which causes a delay in the convergence of some eigenvalue estimates.

With these two quantities in mind, we can describe the total time required for an iterative method as

$$\text{Total time} = (\text{time per iteration}) \times (\text{number of iterations}). \tag{1.2}$$

It has long been recognized (see, e.g., [49, 118]) that in finite precision, the convergence and stability properties of classical Krylov methods are not necessarily maintained by communication-avoiding Krylov methods. As the parameter s is increased to reduce communication costs and so time per iteration, the rate of convergence can slow, and the attainable accuracy can decrease. Lost accuracy can be a significant problem depending on the needs of the application, and as is clear from (1.2), slow convergence can negate potential benefits from the communication-avoiding approach.

Thus despite any potential per-iteration performance gain, this decrease in convergence rate and attainable accuracy can limit the practical applicability of CA-KSMs. To achieve $O(s)$ speedups using CA-KSMs, we must not only reduce the time per iteration by $O(s)$, but we must also ensure a convergence rate close to that of the classical method in finite precision arithmetic. Further, depending on application requirements, we must ensure that any decrease in attainable accuracy does not result in an unacceptably inaccurate solution. Another challenge is the extension of techniques for improving numerical properties often used in conjunction with classical KSMs, including reorthogonalization, residual replacement, and preconditioning, to CA-KSMs in a way that still admits an $O(s)$ reduction in data movement. Overcoming these challenges to the practical use of communication-avoiding Krylov subspace methods is the goal of this thesis.

1.4 Thesis Contributions

Despite nearly 30 years of study, the literature on communication-avoiding Krylov methods still lacks a detailed numerical stability analysis, as well as both theoretical and practical comparisons with the stability and convergence properties of classical implementations. In this thesis, we address the stability, performance, and implementation challenges described above, with the goal of increasing the practicality and accessibility of CA-KSMs for the general scientific computing community. We extend a number of theoretical results and algorithmic techniques developed for classical KSMs to CA-KSMs, and identify constraints under which CA-KSMs are competitive in solving practical problems.

After reviewing preliminaries and detailing the communication and computation costs of the kernels used in this thesis, we demonstrate in Chapter 4 that classical Lanczos-based Krylov methods can be transformed into variants which perform asymptotically less communication. However, when finite arithmetic is used, the communication-avoiding methods are no longer mathematically equivalent to their classical counterparts; Chapter 5 is dedicated to explaining where and how these differences arise from a theoretical perspective. Our results show that the conditioning of the s-step bases plays an important role in influencing finite-precision behavior. Working from this observation, in Chapter 6 we develop a number of techniques that can improve finite-precision stability and convergence in CA-KSMs.

Chapter 7 details interesting performance optimizations for certain specific cases of matrix structure and in Chapter 8, we show that CA-KSMs can achieve significant speedups (up to $6\times$) over classical KSMs for large-scale problems in various scientific domains. Some work presented in this thesis is adapted from published work, as indicated in bullet points below.

The primary contributions of this thesis include:

- The development of new communication-avoiding Krylov methods, including a non-symmetric Lanczos variant, biconjugate gradient (BICG), conjugate gradient squared (CGS), biconjugate gradient stabilized (BICGSTAB), upper and lower Lanczos bidiagonalization for solving singular value problems, and the least-squares QR (LSQR) method for solving least squares problems (Chapter 4; adapted from [35], [16], [27], and [34]);

- Completion of the first complete rounding error analysis of the CA-Lanczos method, including derivation of upper bounds on the loss of normality of and orthogonality between the computed Lanczos vectors as well as a recurrence for the loss of orthogonality, which identifies conditions under which CA-KSMs are as stable as their classical counterparts (Section 5.1; adapted from [32]);

- Proof that the bounds on eigenvalue and eigenvector convergence rates in finite precision Lanczos given by Paige [141] can be extended to CA-Lanczos assuming a bound on the maximum condition number of the precomputed s-step Krylov bases, which shows that if one can maintain modest condition numbers of the precomputed s-step Krylov bases throughout the iterations, then the finite precision behavior of the CA-Lanczos method will be similar to that of classical Lanczos (Section 5.1; adapted from [32]);

- Numerical experiments confirming the validity of our bounds on convergence of Ritz values in CA-Lanczos for two different bases and various values of s, which illustrate the impact of the basis condition number on our error bounds and suggest methods for improving convergence and accuracy through the inexpensive monitoring of quantities generated during the iterations (Section 5.1.6);

- The derivation of Lanczos-type matrix recurrences governing CA-CG and CA-BICG in finite precision arithmetic, which demonstrate the algorithm's relationship to classical (BI)CG and allows us to give upper bounds on the norm of the updated residual in finite precision CA-(BI)CG in terms of the residual norm of exact GMRES applied to a perturbed matrix, multiplied by an amplification factor (Section 5.2; adapted from the Technical Report [29]);

- The derivation of a computable bound on the deviation of the true and updated residuals in the CA-CG and CA-BICG methods in finite precision, which can be iteratively updated within the method without asymptotically increasing communication or computation (Section 5.3; adapted from [31]);

- The development and implementation of an implicit residual replacement strategy, based on the derived computable maximum attainable accuracy bounds in Section 5.3, which can maintain agreement between the true and updated residuals to within the order of machine precision in CA-CG and CA-BICG, yielding improvements of up to 7 orders of magnitude improvement in accuracy for little additional cost (Section 6.1; adapted from [31]);

- The development and analysis of communication-avoiding deflated CG, and the demonstration that deflation-based preconditioning can be applied in communication-avoiding formulations of Lanczos-based Krylov methods such as CA-CG while maintaining an $O(s)$ reduction in communication cost (Section 6.2; adapted from [36]);

- The identification of a number of techniques that can be used to improve stability and convergence properties in CA-KSMs while still admitting a communication-avoiding approach, including selective reorthogonalization, look-ahead, extended/variable precision, dynamic basis parameter updates, and variable basis dimensions (Sections 6.3-6.7).

- The derivation of a parallel 'blocking covers' approach for increasing temporal locality in matrix powers computations, which allows for communication-avoidance with matrices that can be split into the sum of dense and low-rank components, and application of this approach to hierarchical semiseparable matrices (see [19]) (Section 7.1; adapted from [112]);

- The development of the streaming matrix powers optimization, which, by interleaving SpMV and orthogonalization operations, can further reduce communication in sequential methods with implicitly-represented matrices (e.g., stencils with constant coefficients) and also enables write-avoiding Krylov methods which further reduce writes to the lowest level of the memory hierarchy, which may be much more expensive than reads in some technologies (Section 7.2);

- The development of a method to improve the partitioning of nonsymmetric matrices using hypergraphs (see [37]) to avoid communication in matrix powers computations, which uses a randomized algorithm to estimate the approximate hyperedge size and selectively drop hyperedges once they become too large, reducing both hypergraph construction and partitioning time (Section 7.3);

- Weak and strong scaling experiments for distributed-memory CA-CG, which show potential speedups of up to $6\times$ over classical CG for large-scale PDE solves on a model problem run on a supercomputer with up to 4096 MPI processes, and identification of the space in which the communication-avoiding approach is most beneficial in terms of performance (Section 8.2); and

- The implementation and optimization of CA-BICGSTAB as a high performance, distributed-memory bottom solver for geometric multigrid, which enabled speedups of

over 4× on synthetic benchmarks and over 2.5× in real combustion and cosmology applications on a supercomputer with up to 4096 MPI processes (Section 8.3; adapted from [190]).

Chapter 2

Preliminaries

2.1 Notation and Definitions

We begin with a discussion of the notation and definitions used throughout this thesis. We generally use n to denote the dimension of a square matrix, p (without a subscript) to denote the number of parallel processors, and ϵ to denote the unit round-off of the machine. All logarithms are taken to be base 2. Unless otherwise specified, all norms can be taken to be 2-norms. We use a superscript '+' to denote the Moore-Penrose pseudoinverse, i.e., $Y^+ = (Y^T Y)^{-1} Y^T$.

We use standard asymptotic notation in discussing computation and communication complexity of algorithms. Specifically, $f(n) = O(g(n))$ means that for sufficiently large n, there exists a constant $c > 0$ such that $|f(n)| \leq c|g(n)|$; $f(n) = \Omega(g(n))$ means that for sufficiently large n there exists a constant $c > 0$ such that $|f(n)| \geq c|g(n)|$; and $f(n) = \Theta(g(n))$ means that both $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

We also use $f(n) \ll g(n)$ to mean that for every $c > 0$, $|f(n)| \leq c|g(n)|$ for sufficiently large n. This notation will be used somewhat informally to hide constants which are small relative to $g(n)$; i.e., $f(n) \ll g(n)$ informally implies that $f(n)$ is insignificant compared to $g(n)$. We also emphasize that our asymptotic notation does not imply that the results require n to approach infinity to be correct. While the asymptotic analyses provide better estimates for larger parameter values, they have proved meaningful and useful in algorithm design for even modestly sized problems.

In this work, all asymptotic expressions should be interpreted as univariate, despite appearances. For example, $f = O(sn)$ asserts that for every infinite sequence $((s_i, n_i) : i \in \{1, 2, \ldots\})$ of parameter pairs satisfying certain constraints, there exist constants $c > 0$ and $N \in \{1, 2, \ldots\}$ such that $f \leq c s_i n_i$ for all $i \geq N$. The given constraints will restrict the set of sequences of parameter tuples, e.g., $s \ll n$, which should be interpreted as the univariate expression $s_i \ll n_i$.

2.2 Theoretical Performance Model

In this work, we base our algorithms on a theoretical model of performance. We model a parallel machine as a homogeneous, fully-connected network of p processors, each with a local main memory of size $\hat{M}$ and a cache of size M. We assume that processors communicate via point-to-point (P2P) messages and collective synchronizations.

In addition to the number of processors p, our machine model has four main parameters: α represents a latency cost incurred by every message, β represents a bandwidth cost linear in the message size, γ represents the time to perform a floating point operation on local data, and ν represents the time to transfer a word of data between main memory and cache. Each processor executes asynchronously and can send or receive at most one message at a time; passing an n-word message takes $\alpha + \beta n$ seconds on both the sending and receiving processors. We assume that processors can only operate on local data and can execute each flop at the peak rate $1/\gamma$.

In our model the bandwidth and latency costs of an MPI_Allreduce global collective of size w are $w \log p$ and $\log p$, respectively. There is no notion of distance on the network, and we assume the network has unbounded buffer capacity.

While this simple model's assumptions are not all realistic, similar models are widely used to analyze communication costs on distributed-memory machines (see, e.g., [42]). One could refine this model to obtain, e.g., the LogP model [55], which distinguishes between network latency, software overhead, and network injection bandwidth (blurred between our α and β terms), allows overlap of communication and computation, and introduces constraints on message size and network congestion.

We model the cost of an algorithm in terms of four costs: F represents the number of floating-point operations performed locally, W represents the number of words moved from one parallel processor to another, S represents the number of messages sent/received by a processor, and $\hat{W}$ represents the number of words moved between slow and fast memory. In terms of these parameters, the time T for an algorithm to execute is bounded by

$$\max(\gamma F, \beta W, \alpha S, \nu \hat{W}) \leq T \leq \gamma F + \beta W + \alpha S + \nu \hat{W}.$$

For some purposes in this work, we will relax this model and consider only parallel costs; i.e., we assume that each processor has unbounded local memory and ignore the cost of data movement from main memory to cache. These assumptions may be unrealistic when the neglected sequential costs are nontrivial. However, when considering large p and small local problems, when performance is dominated by interprocessor communication, we expect the sequential costs would not significantly increase our models' estimated costs. If a processor performs F flops, and sends/receives S messages containing a total of W words, we then model its worst-case runtime as

$$T = \gamma F + \beta W + \alpha S. \tag{2.1}$$

This is a poor cost model for certain programs, like one where the p processors relay a value from processor 1 to processor 2 to processor 3 ... to processor p: each processor

sends/receives at most 2 words, but the actual runtime grows linearly in p. To count correctly in such situations, one can consider the runtime along critical paths; e.g., in a program activity graph [198]. For our algorithms here, we will only consider certain parallelizations where one processor is always the slowest, so we can simply count F, S, W for that processor to bound the total runtime.

Throughout the thesis, we informally use this performance model to reason about algorithm tradeoffs and justify our communication-avoiding approach. This performance model is applied more concretely in Section 6.2.4, where we use this model to predict runtimes and show tradeoffs in our communication-avoiding deflated CG method for a model problem with various parameters, and in Section 7.1.4, where we use this model to predict runtimes and show tradeoffs in our communication-avoiding matrix powers kernel for hierarchical semiseparable matrices (see, e.g., [19]).

2.3 Classical Krylov Subspace Methods

Krylov subspace methods (KSMs) are some of the most commonly used algorithms for solving large, sparse linear systems and for computing a few dominant eigenvalues/vectors of large, sparse matrices. Because of their success and ubiquity in numerous application domains, KSMs have been named one of the 'Top 10 Algorithms' of the 21st century [53]. There is no shortage of introductory texts which describe Krylov Subspace Methods, including those of Saad [153, 154], Greenbaum [87], Meurant [123, 124], van der Vorst [180], Liesen and Strakoš [119], as well as the 'Templates' book [17]. The study of Krylov methods began with the work of Lanczos [115, 116] and Hestenes and Stiefel [100] in the 1950's and began to gain traction as a field in the 1970's. For a history of KSMs and related topics, we direct the curious reader to the book of Liesen and Strakoš [119], which contains an overview in Chapter 1 and 'Historical Notes' throughout.

Krylov subspace methods work by iteratively constructing one or more bases for Krylov subspaces, which are defined in terms of a matrix A and vector v by

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

We focus our discussion here on Lanczos and Lanczos-based Krylov methods, which are our main concern in this thesis. Given a symmetric matrix $A \in \mathbb{R}^{n \times n}$ and a starting vector $v_1 \in \mathbb{R}^n$ with unit 2-norm, i steps of the Lanczos method [115] theoretically produce the orthonormal matrix $V_i = [v_1, \ldots, v_i]$ and the symmetric tridiagonal matrix $T_i \in \mathbb{R}^{i \times i}$ such that

$$AV_i = V_i T_i + \beta_{i+1} v_{i+1} e_i^T.$$

When $i = n$, if T_n exists (i.e., no breakdown occurs), then the eigenvalues of T_n are the eigenvalues of A. In practice, some of the eigenvalues of T_i are good approximations to the eigenvalues of A when $i \ll n$, which makes the Lanczos method attractive as an iterative procedure. Many KSMs, including those for solving linear systems and least squares

problems, are based on the Lanczos method. In this work, we specifically consider symmetric and nonsymmetric Lanczos, conjugate gradient (CG), biconjugate gradient (BICG), conjugate gradient squared (CGS), biconjugate gradient stabilized (BICGSTAB), Lanczos bidiagonalization, and least-squares QR (LSQR).

For the solution of linear systems, Krylov subspaces are used as expanding search spaces, from which the approximate solution x_i or the approximate eigenvalue and eigenvectors are computed via projection. A general projection method for solving the linear system $Ax = b$ extracts an approximate solution x_i from an affine subspace $x_1 + \mathcal{K}_i(A, r_1)$ of dimension i, where $r_1 = b - Ax_1$ denotes the initial residual, by imposing the Petrov-Galerkin condition $b - Ax_i \perp \mathcal{L}_i$, where $\mathcal{L}_i$ is another subspace (the 'constraint space') of dimension i. We can also write this in terms of the correction to the approximate solution in each iteration. Let $\delta_i = x_i - x_1$. Then given subspaces $\mathcal{K}_i$ and $\mathcal{L}_i$, each projection step can be described by

$$\text{Find } \delta_i \in \mathcal{K}_i \text{ such that } r_1 - A\delta_i \perp \mathcal{L}_i.$$

The choice of $\mathcal{L}_i$ distinguishes various iterative techniques; most notably, we have an orthogonal projection method if $\mathcal{L}_i = \mathcal{K}_i$, and an oblique projection method otherwise. For example, given a basis $V_i = [v_1, \ldots, v_i]$ for $\mathcal{K}_i(A, r_1)$ such that $AV_i = V_iT_i + w_ie_i^T$ with $V_i^TV_i = I$ and T_i tridiagonal, the approximate solution obtained from an orthogonal projection method onto $\mathcal{K}_i$ is given by

$$x_i = x_1 + V_iy_i \text{ where } y_i = T_i^{-1}(\|r_1\|_2e_1),$$

and thus the method proceeds to refine the approximate solution x_i by progressively solving $T_iy_i = \|r_1\|_2e_1$. This example describes the conjugate gradient (CG) method, which is based on the symmetric Lanczos process. In the case of non-Hermitian matrices A, one approach is to relax the orthogonality constraint and instead generate two sets of mutually biorthogonal basis vectors, i.e., $V_i = [v_1, \ldots, v_i]$ and $\tilde{V}_i = [\tilde{v}_1, \ldots, \tilde{v}_i]$ where $v_\ell^T\tilde{v}_j \neq 0$ if $\ell = j$ and $v_\ell^T\tilde{v}_j = 0$ if $\ell \neq j$. This is the approach taken in the non-symmetric Lanczos method, which was described by Lanczos [115, 116]. The corresponding biconjugate gradient (BICG) solver of Fletcher [69] implements the projection process

$$x_i \in x_1 + \mathcal{K}_i(A, r_1) \text{ and } r_i \perp \mathcal{K}_i(A^H, \tilde{r}_1),$$

where $\tilde{r}_1$ is an arbitrary nonzero vector. In Section 2.5, we review the stability and convergence properties of classical KSMs and discuss how finite precision affects these properties, which is of utmost importance when considering use of these methods in practice. We first turn to a discussion of the cost per iteration in classical KSMs in terms of our communication model in Section 2.2.

Speaking in terms of projection processes, in each iteration, classical Krylov methods increase the Krylov subspace $\mathcal{K}_i$ by one dimension by computing a new basis vector and subsequently orthogonalize this new basis vector against the subspace $\mathcal{L}_i$. In the case of linear equation solvers, the approximate solution is then updated imposing the Petrov-Galerkin

condition $b - Ax_i \perp \mathcal{L}_i$. In terms of linear algebra operations, computing the new basis vector to expand the Krylov subspace requires one or more sparse matrix-vector multiplications (SpMVs), and performing the orthogonalization against $\mathcal{L}_i$ (and for linear solvers, updating the approximate solution) requires one or more inner products and vector operations.

These computational kernels, SpMVs and inner products, are both communication-bound computations on modern computer architectures. To perform an SpMV, each processor must communicate entries of the source vector it owns to other processors in the parallel algorithm, and in the sequential algorithm, the matrix A must be read from slow memory (when it is too large to fit in cache, the most interesting case). Inner products involve a global reduction (see [153, §11.4]) in the parallel algorithm, and a number of reads and writes to slow memory in the sequential algorithm (depending on the size of the vectors and the size of the fast memory). We discuss the costs of SpMVs and inner products in more detail in Chapter 3. We first turn to a discussion of how CA-KSMs can reduce the communication cost of classical KSMs.

2.4 Avoiding Communication in Krylov Subspace Methods

The communication cost of classical KSMs can be reduced by computing iterations in blocks of s at a time, splitting the iteration loop into an outer loop which iterates over blocks and an inner loop that iterates within each block. Methods that use this block restructuring of classical KSMs are termed *s-step* Krylov subspace methods or *communication-avoiding* Krylov subspace methods (CA-KSMs). Again speaking in terms of projection processes, in each outer loop, CA-KSMs increase the dimension of $\mathcal{K}$ by a factor of $\Theta(s)$ by computing $\Theta(s)$ new basis vectors and then, in case of Lanczos-based CA-KSMs, compute a Gram matrix to encode inner products between these vectors. Within the corresponding inner loop iterations, vector updates are performed implicitly by updating the coordinates of the usual length-n vectors in the basis represented by the new $\Theta(s)$ basis vectors, with the scalars required for orthogonalization against $\mathcal{L}$ computed using the encoded inner products.

In terms of linear algebra operations, the $\Theta(s)$ basis vectors required to expand the Krylov subspace by s dimensions can be computed by a *communication-avoiding* matrix powers computation, which we discuss below in Section 3.2. Under certain constraints, these $\Theta(s)$ basis vectors can be computed for the same communication cost as computing a single basis vector. The Gram matrix can be computed for the same latency cost as a single inner product, with the caveat that we now increase the number of inner products computed in s iterations from $\Theta(s)$ to $\Theta(s^2)$. Thus CA-KSMs allow for the computation of s iterations for the same asymptotic communication cost as one classical KSM iteration. Depending on the input matrix, algorithm, and computing platform, the communication bottleneck could be the sparse linear algebra, dense linear algebra, or both. When it is just one, say the dense linear algebra (orthogonalization), this can simplify the implementation (see Chapter 8).

There is a wealth of literature related to s-step KSMs and the idea of avoiding communication. Early related work discussed below has been nicely summarized by Hoemmen in Table 1.1 in [102]. The first known related approach in the literature is Forsythe's s-dimensional optimum gradient method, which is a steepest descent method. This same technique was first used in Krylov methods with the conjugate gradient method of Van Rosendale [182]. Van Rosendale gives a variant of the parallel conjugate gradient method which minimizes inner product data dependencies with the goal of exposing more parallelism.

The term "s-step" Krylov methods was first used by Chronopoulos and Gear, who developed the s-step CG method [48, 49]. Over the next decade or so, Chronopoulos and others developed s-step variants of Orthomin and GMRES [50], Arnoldi and Symmetric Lanczos [108, 109], MINRES, GCR, and Orthomin [47, 52], Nonsymmetric Lanczos [110], and Orthodir [51]. Walker used s-step bases as a method for improving stability in GMRES by replacing the modified Gram-Schmidt orthogonalization process with Householder QR [184]. We emphasize that the use of the overloaded term 's-step methods' here differs from other works, e.g., [56, 106] and [84, §9.2.7], in which the term 's-step methods' is used to refer to a type of restarted Lanczos procedure.

Early on, much work in s-step Krylov methods was limited to using monomial bases (i.e., $[v, Av, A^2v, \ldots]$) for the precomputed $\Theta(s)$-dimensional Krylov subspaces, and it was found that convergence often could not be guaranteed for $s > 5$. This motivated research into the use of other more well-conditioned bases for the Krylov subspaces. Hindmarsh and Walker used a scaled (normalized) monomial basis to improve convergence [101], but only saw minimal improvement. Joubert and Carey implemented a scaled and shifted Chebyshev basis which provided more accurate results [104]. Chebyshev bases have also been used by de Sturler and van der Vorst [59, 61] in an s-step GMRES variant. Bai et al.[14] and Erhel [68] have also used the Newton basis with improved convergence in s-step GMRES. In [52], Chronopoulos and Swanson use Gram Schmidt to orthonormalize the direction vectors in each s-step block, which allowed for use of up to $s = 16$ with a monomial basis. Their goal was to be able to increase s as this exposes more potential parallelism.

Most of these early works did not exploit the potential communication avoidance in computing the s Krylov basis vectors, which can reduce communication cost by a factor of $O(s)$ for well-partitioned sparse matrices. (We point out that under certain conditions, the blocking of vector operations alone can still provide significant speedups.) The methods of Joubert and Carey [104] and Toledo [175] both include a communication-avoiding matrix powers optimization for stencil matrices, but Hoemmen et al. (see, e.g., [63, 102, 127]) were the first to describe and implement the communication-avoiding matrix powers kernel for general sparse matrices. Our derivations most closely follow their work. The growing cost of communication in large-scale sparse methods has created a recent resurgence of interest in the implementation, optimization, and development of CA-KSMs and preconditioners for CA-KSMs; see, e.g., [197, 89, 122, 194, 195].

There are many alternative approaches to reducing communication in KSMs which differ from this approach, including reducing synchronizations, allowing asynchronous iterations [18], using block Krylov methods to exploit locality, inexact Krylov methods [24, 178,

158], and using alternative methods such as Chebyshev Iteration. For a good overview, see [102, §1.6].

A large number of approaches involve enabling overlap between communication and computation and/or reducing the number of synchronization points per iteration. In [90], Gropp presents an asynchronous variant of the conjugate gradient method (CG) with two global synchronization points per iteration that can be overlapped with the matrix-vector multiplication and application of the preconditioner, respectively. Overlapping techniques for solving least squares problems with Krylov subspace methods on massively parallel distributed memory machines are presented in [200]. The Arnoldi method with delayed reorthogonalization (ADR) described in [98] mixes work from the current, previous, and subsequent iterations to avoid extra synchronization points due to reorthogonalization. This approach is currently implemented in the SLEPc library[99]. Some approaches to overlapping communication and computation to reduce synchronization combine or rearrange a fixed number of synchronizations into a single synchronization; see, e.g., [199, 91]. We stress that this differs from our approach; most notably, speedups are limited to a constant factor (depending on the number of synchronizations combined into a single synchronization) whereas speedups for CA-KSMs are theoretically asymptotic in s.

There has also been recent success in the development of *pipelined* Krylov methods, which alleviate the performance bottleneck of KSMs by overlapping communication and computation. In [77], a pipelined version of GMRES is presented, where the authors overlap nonblocking reductions (to compute dot products needed in later iterations) with matrix-vector multiplications. This resulted in speedups and improved scalability on distributed-memory machines. An analogous pipelined version of CG is presented in [79], and the pipelining approach is discussed further in [9]. We consider it an open problem to determine if the pipelined and s-step approaches are compatible.

2.5 Numerical Properties of Krylov Subspace Methods

As discussed in Section 2.3, in exact arithmetic, the Lanczos method generates a sequence of orthonormal basis vectors (in the case of nonsymmetric Lanczos, two sequences of biorthonormal basis vectors). This is what, in the case of symmetric Lanczos at least, guarantees similarity of A and T_i after at most n steps. In finite precision, (bi)orthogonality no longer necessarily holds. This can result in multiple approximations in T_i to the original eigenvalues of A, which delays convergence to other eigenvalues.

The orthogonality properties for CG are equivalent to the optimality property

$$\|x - x_i\|_A = \min_{z \in x_1 + \mathcal{K}_i(A, r_1)} \|x - z\|_A,$$

where $\|v\|_A = (v^T A v)^{1/2}$ and A is symmetric positive definite (SPD). In other words, the CG method selects at each iteration the approximate solution that minimizes the error in

the A-norm. It is well-known (see, e.g., [84]) that in exact arithmetic, after i iterations of CG, the error is (loosely) bounded by

$$\|x - x_i\|_A \leq 2\|x - x_1\|_A \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^i ,$$

where $\kappa(A) = \lambda_n/\lambda_1$ where $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of A. More accurate bounds based on Gauss quadrature connections can be obtained which show that the convergence rate of CG depends in a complicated way on how well the eigenvalues of A are approximated by T_i. Thus if multiple approximations to some eigenvalues of A appear in T_i due to finite precision error, this can cause convergence to be delayed.

Lanczos and others recognized early on that rounding errors could cause Krylov methods to deviate from ideal theoretical behavior. Since then, various efforts have been devoted to analyzing, explaining, and improving finite precision KSMs. A thorough overview of the literature can be found in Meurant and Strakoš [125]. Widely considered to be the most significant development is the series of papers by Paige [138, 139, 140, 141]. Paige's analysis succinctly describes how rounding errors propagate through the algorithm to impede orthogonality. These results were developed to give theorems which link the loss of orthogonality to convergence of the computed eigenvalues [141]. A number of strategies for maintaining orthogonality among the Lanczos vectors were inspired by the analysis of Paige, including selective reorthogonalization [147] and partial reorthogonalization [157].

Another important result is due to Greenbaum, who performed a backward-like error analysis which showed that finite precision Lanczos and CG behave very similarly to the exact algorithms applied to any of a certain class of larger matrices [85]. Further explanation and examples are due to Greenbaum and Strakoš [88]. Paige has shown a similar type of augmented stability for the Lanczos process [142], and these results have recently been extended to the nonsymmetric case [143]. There are many other analyses, including some more recent results due to Wülling [192] and Zemke [202]. The concept that the delay of convergence in finite precision CG is determined by numerical rank-deficiencies of the constructed Krylov subspaces was born from the work of Greenbaum [85], Greenbaum and Strakoš [88], and Notay [135]. The reader is directed to the work of Golub and Strakoš [82], Strakoš and Tichý [171], and Meurant [124] for analysis and discussion of the connections of CG to Gauss quadrature in finite precision. Results on convergence in the finite precision classical BICG method are given by Tong and Ye [176].

In addition to convergence delay, another ill effect of finite precision error in KSMs for solving linear systems is the loss of attainable accuracy. Finite precision roundoff in updates to the approximate solution x_i and the residual r_i in each iteration can cause the 'updated residual' r_i and the 'true residual' $b - Ax_i$ to grow further and further apart. If this deviation grows large, it can limit the *maximum attainable accuracy*, i.e., the accuracy with which we can solve $Ax = b$ on a computer with unit round-off ϵ. Analyses of maximum attainable accuracy in CG and other classical KSMs are given by Greenbaum [86], van der Vorst and Ye [181], Sleijpen, van der Vorst, and Fokkema [162], Sleijpen, van der Vorst, and

Modersitzki [163], Björck, Elfving, and Strakoš [22], and Gutknecht and Strakoš [96]. One important result of these analyses is the insight that loss of accuracy can be caused at a very early stage of the computation, which can not be corrected in later iterations.

The works mentioned here have served to greatly improve our understanding of the behavior of KSMs in practice. A primary goal of this thesis is the extension of a subset of these results to CA-KSMs.

2.6 Algorithm Notation and Terminology

As avoiding communication is our primary motivation, we generally use the terminology 'CA-KSM' to refer to the methods described here, although the rounding error results presented in Chapter 5 hold for s-step formulations in general, regardless of whether we implement them in a communication-avoiding way (i.e., whether we compute the Krylov bases using sequential SpMVs or a communication-avoiding matrix powers kernel). We use the term 'classical Krylov method' or 'classical KSM' to refer to the usual formulation and implementation of Krylov subspace methods, which consist of a single iteration loop interleaving SpMV and orthogonalization operations.

We use the parameter s to denote the factor by which the iterations are blocked in the CA-KSMs; i.e., for each outer loop iteration, we compute a Krylov subspace of dimension $O(s)$, and then perform s inner loop iterations, updating the coefficients of the iteration vectors in the computed $O(s)$-dimensional Krylov basis. In CA-KSMs, we generally index iterations by $sk + j$, where k is the outer loop iteration and j is the inner loop iteration. Corresponding iterations in the classical Krylov methods are indexed by i.

We also overload the variable k, and use it to denote the number of vectors we compute in a matrix powers kernel computation to be discussed in Chapter 3: given a matrix A and a starting vector y, the matrix powers kernel computes, e.g., $[y, Ay, A^2y, \ldots, A^ky]$, which is a $(k + 1)$-dimensional basis for a Krylov subspace. (In general, we allow for computation of any polynomial basis, not just the monomial basis). We do not reuse the variable s here, since, depending on the method and the specific starting vector, taking s iterations at a time may not require a basis of dimension exactly $s + 1$. We may need to execute a matrix powers computation to compute a basis of dimension, e.g., s, $s - 1$, $2s + 1$, $2s - 1$, or $2s + 2$. Furthermore, the value of k that gives the best performance in the matrix powers computation (which depends on the tradeoff between bandwidth/latency and flops/latency) might be different than the best choice of s for computing the Gram matrix, which may also be different than the largest s that still allows for numerical stability. There is nothing preventing us from running multiple calls to the matrix powers kernel to compute the required $O(s)$-dimension basis.

Within the context of a CA-KSM, we refer to the output of the matrix powers computation as an 's-step basis' or 's-step bases'. This is to distinguish them from, e.g., the Lanczos basis, which is composed of the length-n vectors produced in each iteration (which are recoverable in each outer loop by a change-of-basis operation using the s-step bases). We

refer to the matrix whose columns consist of the computed s-step bases in each outer loop as the 's-step basis matrix', although all columns are not necessarily linearly independent.

Chapter 3

Communication-Avoiding Kernels

Depending on the specific solver, machine architecture, and problem size and nonzero structure, the communication bottleneck in KSMs may either be computing a basis for the Krylov subspace, or the subsequent (dense) vector operations, like inner products. In this chapter, we discuss in depth the communication and computation costs of both naïve and communication-avoiding variants of these kernels in terms of our performance model in Section 2.2. Section 3.1 focuses on these costs for orthogonalization operations, which in the case of Lanczos-based KSMs and CA-KSMs, involves one or more (blocked) inner products. On large-scale parallel machines, such computations are performed using an MPI_Allreduce, which is often the dominant cost in Krylov solvers.

In Section 3.2, we present both naive and communication-avoiding algorithms for computing matrix powers and give their complexity in terms of latency, bandwidth, and computation costs, which are determined by the adjacency graph describing the nonzeros in the input matrix A. We write these costs for general sparse matrices and give an example for stencil matrices, which are common in the physical sciences, and explain the implication of these costs for potential performance gains.

Our great level of detail in Section 3.2 serves many purposes throughout this thesis. The algorithms presented in this section, namely the parallel 'PA0' and 'PA1' and the sequential 'SA0' and 'SA1', are referred to throughout Chapter 4 in comparing communication costs of classical KSMs and CA-KSMs. The presentation of these algorithms also serves to formally define constraints on sparsity (i.e., that the input matrix A is *well-partitioned*) and to establish the notation that we use to describe our data-sparse matrix powers kernel for hierarchical semiseparable (HSS) matrices in Section 7.1 and our streaming matrix powers algorithms which are presented in Section 7.2. Our detailed complexity counts for the algorithms 'PA0' and 'PA1' enable a quantitative comparison of classical and communication avoiding algorithms in subsequent sections. In Section 6.2, We use these results to model performance tradeoffs in deflated versions of CG and CA-CG in Section 6.2 and in our data-sparse matrix powers kernel applied to HSS matrices in Section 7.1.

Perhaps most importantly from a numerical standpoint, in Section 3.2.5 we discuss the numerical challenges in computing bases for Krylov subspaces in finite precision. The choice

Table 3.1: Upper bounds on costs of Gram matrix construction

Parallel (per processor)	
F	w^2n/p
W	$w^2 \log_2 p$
S	$\log_2 p$
Sequential	
F	w^2n
W	wn
S	$\lceil wn/M \rceil$

of polynomials used in basis construction greatly affects the conditioning of the computed bases. As we demonstrate through numerical experiments in Chapter 4 and through theoretical results in Chapter 5, the conditioning of the s-step bases has profound implications for convergence and accuracy in CA-KSMs.

3.1 Block Inner Products

Both classical KSMs and CA-KSMs incur communication costs due to orthogonalization performed in each iteration, usually involving a series of inner products. In this section, we describe, for both parallel and sequential implementations, the asymptotic costs of computing inner products between w dense vectors, i.e., given a matrix $\mathcal{Y}$ with n rows and w columns, we discuss the cost of computing the w-by-w matrix $\mathcal{Y}^T\mathcal{Y}$. We will refer to $\mathcal{Y}^T\mathcal{Y}$ as a Gram matrix, although in the case $w = 1$, this is simply a scalar quantity.

In the parallel case, we assume that $\mathcal{Y}$ is distributed rowwise across p processors. Computing the Gram matrix in this case requires a costly global synchronization (an MPI_Allreduce). In the sequential case, computation of the Gram matrix involves one or more reads from slow memory, the number of which depends on the relative size of $\mathcal{Y}$ and the fast memory size M. The asymptotic computation, bandwidth, and latency costs for both the parallel (per processor) and sequential case are summarized in Table 3.1.

The strategy in CA-KSMs is to block together multiple inner products into one Gram matrix computation to reduce communication costs. As an example, let us consider the parallel case. For classical Lanczos-based KSMs, s iterations require $\Theta(s)$ inner products of size $w = \Theta(1)$. In parallel, this incurs $\Theta(s \log_2 p)$ in both bandwidth and latency costs. For the communication-avoiding versions, s iterations requires $\Theta(1)$ block inner product of size $w = \Theta(s)$, so the bandwidth and latency costs are $\Theta(s^2 \log_2 p)$ and $\Theta(\log_2 p)$, respectively. The CA-KSM strategy of blocking together inner products thus leads to an s-fold decrease in latency at the cost of increasing the bandwidth and computation costs by a factor of $O(s)$.

In the case of Arnoldi-based KSMs (like GMRES), the communication-avoiding versions perform block orthogonalization by computing a (thin) QR factorization of a tall, skinny

matrix. Using the Tall-Skinny QR (TSQR) algorithm in [62], this leads to an s-fold decrease in latency in the parallel case as well as an s-fold decrease in latency and bandwidth in the sequential case [102].

3.2 The Matrix Powers Kernel

The computational complexity of classical sparse matrix-vector multiplication (SpMV) is $O(1)$. Thus any algorithm that uses classical SpMV, such as classical Krylov subspace methods, will necessarily be communication-bound. This means we need a different kernel with higher computational intensity to have a hope of reducing communication. We call this the communication-avoiding matrix powers kernel.

More concretely, our goal is to compute, given a matrix A, vector y, and desired dimension $k+1$, a basis for the Krylov subspace

$$\mathcal{K}_{k+1}(A, y) = \text{span}(y, Ay, \ldots, A^k y),$$

which we store in columns of the matrix $\mathcal{Y} = [y_0, \ldots, y_k] = [\rho_0(A)y, \ldots, \rho_k(A)y]$, where $\rho_j(z)$ is a polynomial of degree j, satisfying a three-term recurrence

$$\begin{aligned} &\rho_0(z) = 1, \quad \rho_1(z) = (z - \hat{\alpha}_0)\rho_0(z)/\hat{\gamma}_0, \quad \text{and} \\ &\rho_j(z) = ((z - \hat{\alpha}_{j-1})\rho_{j-1}(z) - \hat{\beta}_{j-2}\rho_{j-2}(z))/\hat{\gamma}_{j-1} \quad \text{for } j > 1. \end{aligned} \tag{3.1}$$

We note that this gives the recurrence

$$A[y_0, \ldots, y_{k-1}, 0_{n,1}] = [y_0, \ldots, y_k] \begin{bmatrix} \hat{\alpha}_0 & \hat{\beta}_0 & & & 0 \\ \hat{\gamma}_0 & \hat{\alpha}_1 & \ddots & & \vdots \\ & \hat{\gamma}_1 & \ddots & \hat{\beta}_{k-2} & 0 \\ & & \ddots & \hat{\alpha}_{k-1} & 0 \\ & & & \hat{\gamma}_{k-1} & 0 \end{bmatrix},$$

which we will exploit to derive communication-avoiding Krylov methods in Section 4; we discuss the choice of the polynomials $\rho_j(z)$ in Section 3.2.5.

In a naïve approach, the matrix $\mathcal{Y}$ could be computed by a sequence of k SpMV operations and $O(k^2)$ vector operations, depending on the polynomial recurrence. In the sequential algorithm, this requires reading the matrix A from slow memory to fast memory $\Theta(k)$ times, assuming A is stored explicitly and exceeds the fast memory size. In parallel, this requires k rounds of messages in order to distribute vector entries ("ghost zones").

Hoemmen et al. (see, e.g., [63, 102, 127]) were the first to make use of a communication-avoiding matrix powers kernel optimization for general sparse matrices, which reduces communication cost by a factor of $O(k)$ for well-partitioned matrices by fusing together a sequence of k SpMV operations into one kernel invocation. This kernel is used in the CA-KSM

to compute an $O(k)$-dimensional Krylov basis in each outer loop iteration. Depending on the nonzero structure of A (more precisely, of $\{A^j\}_{j=1}^k$ for some k), this enables communication-avoidance in both serial and parallel implementations as described in the sections below. In this section, we review classical and communication-avoiding versions of both parallel and sequential algorithms for matrix powers computations.

In this section, we review the communication-avoiding matrix powers kernel algorithms and compare them with the classical approach (repeated SpMVs) for both the sequential and parallel case. These algorithms for general graphs were first presented in [63] and also appear in [16, 102, 126]. We note that matrix powers computations constitute a core kernel not only in Krylov subspaces methods, but in a variety of applications including steepest descent and PageRank.

We distinguish sparse matrix representations based on whether the nonzero values A_{ij} and positions of those nonzeros (i, j) of the matrix A are stored *explicitly* or *implicitly*. In the explicit case, storage of the matrix requires a memory footprint of $\Omega(\text{nnz}(A))$; implicit storage means that the representation exploits additional assumptions about the matrix which may eliminate much of all of the memory needed to store it. A common example of the implicit case are stencil matrices, e.g., the discrete Laplacian on a regular grid.

We briefly mention known lower bounds on data movement and synchronization for matrix powers computations. If A is a stencil on an $N \times \cdots \times N$ d-dimensional mesh, the results of Hong and Kung [103] can be used to show that $\Omega(kN^d/M^{1/d})$ words must be moved between fast memory (of size M) and slow memory in the sequential algorithm. The results of Christ et al. [46] extend this lower bound to a class of directed acyclic graphs (DAGs) that admit a more general type of geometric embedding. Parallel extensions of this result have been discovered more recently by, e.g., Scquizzato and Silvestri [156] and Solomonik et al. [164].

We now review the notation that will be used to describe the matrix powers algorithms presented by Demmel et al. [63].

3.2.1 The Layered Graph

To give algorithms that hold for general sparse matrices, we will describe the problem in terms of graph notation. Let $\text{nnz}(A) = \{(i, j) : A_{ij} \neq 0\}$. Let $G = (V, E)$ denote the *layered graph* of A, which represents the dependencies in computing $y^{(j)} = \rho_j(A)y$ for every $j \in \{1, \ldots, k\}$. We denote element i of $y^{(j)}$ by $y_i^{(j)}$. We have

$$V = \{y_i^{(j)} : 1 \leq i \leq n, 0 \leq j \leq k\}.$$

We denote vertices belonging to the same *level* by $V^{(j)} = \{y_i^{(j)} : 1 \leq i \leq n\}$. The edges in G representing dependencies are due to both the non-zero structure of A and the number of

terms in the polynomial recurrence for generating basis vectors. We have

$$E = \left\{ (y_{i1}^{(j+1)}, y_{i2}^{(j)}) : 0 \leq j \leq k, (i1, i2) \in \text{nnz}(A) \right\} \\ \cup \left\{ (y_i^{(j+u')}, y_i^{(j)}) : 1 \leq d' \leq u, 0 \leq j \leq k - u', 1 \leq i \leq n \right\},$$

where the value of u depends on family of polynomials used in constructing the basis vectors; assuming a three-term recurrence, u will be at most 2 (for the monomial basis, e.g., $u = 0$), although we could in general allow for up to $u = k$.

3.2.2 Parallel Matrix Powers Algorithms

Parallel variants of matrix powers, for both structured and general sparse matrices, are described in [126], which summarizes most of [63] and elaborates on the implementation in [127]. We review two of the parallel matrix powers algorithms from [63, 102, 126], referred to by the authors as PA0, the naïve algorithm for computing k SpMV operations, and PA1, a communication-avoiding variant. To simplify the discussion, we ignore cancellation, i.e., we assume $\text{nnz}(\rho_j(A)) \subseteq \text{nnz}(\rho_{j+1}(A))$ and every entry of $y^{(j)}$ is treated as nonzero for all $j \geq 0$.

Let $R(X)$ denote the *reachability* of set X, i.e., the set of vertices reachable from $X \subseteq V$ including X. Let $\Pi = \{I_1, \ldots, I_p\}$ be a p-way partition of $\{1, \ldots, n\}$. For $m \in \{1, \ldots, p\}$, let $V_m = \{y_i^{(j)} : i \in I_m \wedge 1 \leq j \leq k\}$. Similarly, let $V_m^{(j)} = \{y_i^{(j)} : i \in I_m\}$. We also denote $R^{(j)}(X) = R(X) \cap V^{(j)}$, $R_m(X) = R(X) \cap V_m$, and $R_m^{(j)}(X) = R(X) \cap V_m^{(j)}$. Finally, we $R^T(X)$ denote the reachable set of the transpose graph G^T, i.e., G with all the edges reversed.

For both algorithms, we initially distribute A and y row-wise so that processor m owns rows indexed by I_m, and we let A_i denote the ith row of A. We assume that each processor has local copies of the coefficients describing the recurrence (3.1).

The classical approach to computing matrix powers is shown in Algorithm 1. In this method, for each SpMV, processor m synchronizes with other processors and exchanges neighboring ghost zones before computing the entries of the current vector for which they are responsible. This results in k rounds of messages.

A communication-avoiding variant is shown in Algorithm 2. Here, processor m fetches the non-empty sets $R_\ell^{(0)}(V_m)$ for $\ell \neq m$, all remote $y^{(0)}$ vertices reachable from the vector entries for which they are responsible, in a single round of messages. We note that in PA1, processors may need to communicate entries of A when $k > 1$; in PA0 matrix entries never need to be communicated/replicated.

Upper bounds on the costs of PA0 and PA1 for processor m are shown in Table 3.2. These costs are precise but opaque; as an example, in Table 3.3 we show these costs for a $(2b+1)^d$-point stencil on an $N \times \cdots \times N$ d-dimensional mesh ($n = N^d$) partitioned into $p = \rho^d$ subcubes, where we assume that $w = N/\rho$ is an integer so that each of the p processors owns a subcube and that $k = O(w/b)$. For examples of these costs for other variations of stencil

Algorithm 1 PA0: code for processor m

1: **for** $j = 1$ to k **do**
2: **for** each processor $\ell \neq m$ **do**
3: Send vector entries $R_m^{(j-1)}(V_\ell^{(j)})$ to processor ℓ
4: Receive vector entries $R_\ell^{(j-1)}(V_m^{(j)})$ from processor ℓ
5: **end for**
6: Compute vector entries $V_m^{(j)} \backslash R^T(y^{(j-1)} \backslash V_m)$
7: Wait for receives to finish
8: Compute vector entries $V_m^{(j)} \cap R^T(y^{(j-1)} \backslash V_m)$
9: **end for**

Algorithm 2 PA1: code for processor m

1: **for** each processor $\ell \neq m$ **do**
2: Send vector entries $R_m^{(0)}(V_\ell)$ to processor ℓ
3: Send matrix rows $\{A_i : y_i^{(j)} \in R_m(P_\ell) \backslash y^{(0)}\}$ to processor ℓ
4: Receive vector entries $R_\ell^{(0)}(V_m)$ from processor ℓ
5: Receive matrix rows $\{A_i : y_i^{(j)} \in R_\ell(P_m) \backslash y^{(0)}\}$ from processor ℓ
6: **end for**
7: Compute vector entries $(R(V_m) \backslash y^{(0)}) \backslash R^T(y^{(0)} \backslash V_m)$
8: Wait for receives to finish
9: Compute vector entries $(R(V_m) \backslash y^{(0)}) \cap R^T(y^{(0)} \backslash V_m)$

computations see the tables given in [63]. Note that in both tables, the variable c denotes the storage cost of a nonzero in words. In the case that the entries of A are not stored explicitly, which is often the case for stencil matrices, we can model costs by taking $c = 0$. For any processor $m \in \{1, \ldots, p\}$, we bound the number of arithmetic operations F, the number of words sent and received W, and the number of messages sent and received S. For PA1, we split W into two terms, W and W_A. The quantity W_A is a cost that can be amortized over many successive calls to PA1 with the same matrix A.

The computational cost of each vector element varies depending on the polynomial recurrence used (we have assumed a three-term recurrence here). The SpMV cost does not vary with j and depends only on the sparsity of the rows A_i, i.e., computing $A_i y^{(j)}$ requires $2 \cdot \text{nnz}(A_i) - 1$ flops. We also bound the memory capacity M_Y needed for storing vector entries and M_A needed for rows of A (in the case that A is implicit, $c = 0$ implies $M_A = O(1)$ for both algorithms).

Comparing PA0 and PA1 in Table 3.2, we see that the advantage of PA1 over PA0 is that it may send fewer messages between processors: whereas PA0 requires k rounds of messages, PA1 requires only one, assuming that $|\{\ell \neq m : R_\ell^{(0)}(V_m^{(1)}) \neq \emptyset\}| = |\{\ell \neq m : R_\ell(V_m) \neq \emptyset\}|$ and $|\{\ell \neq m : R_m^{(0)}(V_\ell^{(1)}) \neq \emptyset\}| = |\{\ell \neq m : R_m(V_\ell) \neq \emptyset\}|$ (i.e., the number of other processors each processor must communicate with is the same in PA0 and PA1). Thus if the

Table 3.2: Upper bounds on costs of PA0 and PA1 for each processor m for general matrices

PA0	
F	$2 \cdot \sum_{y_i^{(j)} \in V_m \backslash y^{(0)}} \text{nnz}(A_i) + \min\{u, j\}$
W	$k\left(\sum_{\ell \neq m} \lvert R_\ell^{(0)}(V_m^{(1)})\rvert + \lvert R_m^{(0)}(V_\ell^{(1)})\rvert\right)$
S	$k\left(\lvert\{\ell \neq m : R_\ell^{(0)}(V_m^{(1)}) \neq \emptyset\}\rvert + \lvert\{\ell \neq m : R_m^{(0)}(V_\ell^{(1)}) \neq \emptyset\}\rvert\right)$
M_Y	$\lvert R(V_m^{(1)}) \backslash V_m\rvert + (k+1)\lvert I_m\rvert$
M_A	$c \cdot \sum_{i \in I_m} \text{nnz}(A_i)$
PA1	
F	$2 \cdot \sum_{y_i^{(j)} \in R(V_m) \backslash y^{(0)}} \text{nnz}(A_i) + \min\{u, j\}$
W	$\sum_{\ell \neq m} \lvert R_\ell^{(0)}(V_m)\rvert + \lvert R_m^{(0)}(V_\ell)\rvert + W_A$
W_A	$c \cdot \sum_{\ell \neq m}\left(\sum_{i:y_i^{(j)} \in R_\ell(V_m) \backslash y^{(0)}} \text{nnz}(A_i) + \sum_{i:y_i^{(j)} \in R_m(V_\ell) \backslash y^{(0)}} \text{nnz}(A_i)\right)$
S	$(\lvert\{\ell \neq m : R_\ell(V_m) \neq \emptyset\}\rvert + \lvert\{\ell \neq m : R_m(V_\ell) \neq \emptyset\}\rvert)$
M_Y	$\lvert R(V_m)\rvert$
M_A	$c \cdot \max_{\ell=1}^{p} \sum_{i:y_i^{(j)} \in R(V_m) \backslash y^{(0)}} \text{nnz}(A_i)$

number of other processors that processor m must communicate with is the same for both algorithms, PA1 obtains a k-fold latency savings.

This savings comes at the cost of extra flops (the number of which depends on the relative sizes of $V_m \backslash y^{(0)}$ and $R(V_m) \backslash y^{(0)}$) and a potentially greater number of words moved (depending on the relative sizes of $R_\ell^{(0)}(V_m^{(1)})$ and $R_\ell^{(0)}(V_m)$ and the extra cost W_A), as processors may perform redundant computations to avoid communication.

Comparing the example costs of PA0 and PA1 for stencil computations in Table 3.3, we see that under the assumption $k = O(w/b)$ PA1 gives a factor $\Theta(k)$ decrease in S at the cost of increasing the other four costs by a factor of $O(1)$ (except possibly W in the explicit case, which increases by a factor of $O(1 + cb^d)$ due to W_A, the cost of distributing additional rows of A when $k > 1$.

For both algorithms, the data layout optimization problem of minimizing the costs in Table 3.2 is often treated as a graph or hypergraph partitioning problem. In Section 7.3, we extend existing hypergraph models for parallel SpMV to PA1.

3.2.3 Sequential Matrix Powers Algorithms

Let $G = (V, E)$ with these terms defined as in the parallel case. We still use p to denote the number of partitions of G, although we can now think of these as being cache blocks rather than subdomains local to a certain processor. Let $N(X)$ denote the neighbors of a vertex subset X, i.e., the vertices reachable from some $x \in X$ by paths of length 1 in G. We distinguish between the case where A has implicit nonzero values and positions and the

Table 3.3: Upper bounds on costs of PA0 and PA1 for each processor for a $(2b+1)^d$-point stencil on a $N \times \cdots \times N$ d-dimensional mesh with $n = N^d$, partitioned into $p = \rho^d$ subcubes with integer edge length $w = N/\rho$

PA0	
F	$2w^d \cdot \sum_{j=1}^{k} (2b+1)^d + \min\{u, j\}$
W	$2k(\min\{N, w+2b\}^d - w^d)$
S	$2k(\min\{\rho, 1+2\lceil b/w \rceil\}^d - 1)$
M_Y	$kw^d + \min\{N, w+2b\}^d$
M_A	$c(2b+1)^d w^d$
PA1	
F	$2 \cdot \sum_{j=1}^{k} \left((2b+1)^d + \min\{u, j\}\right) \cdot \min\{N, w+2b(j-1)\}^d$
W	$2(\min\{N, w+2bk\}^d - w^d) + W_A$
W_A	$2c(2b+1)^d(\min\{N, w+2b(k-1)\}^d - w^d)$
S	$2(\min\{\rho, 1+2\lceil bk/w \rceil\}^d - 1)$
M_Y	$\sum_{j=0}^{k} \min\{N, w+2bj\}^d$
M_A	$c(2b+1)^d \cdot \min\{N, w+2b(k-1)\}^d$

three cases where A has explicit values and/or positions. We note that the algorithm SA1 may not yield an asymptotic communication savings in the implicit case, which we discuss further below. In the algorithms below, 'read A' means 'read the explicit values and/or positions of A from slow memory to fast memory'. It is assumed that A does not fit in fast memory (of size M).

The classical approach to sequential matrix powers computation, called 'SA0', is shown in Algorithm 3. As in the parallel case, we assume a row-wise computation of the SpMVs, although the given algorithm could be adapted to work with any SpMV routine. We can see in Algorithm 3 that some elements of A must be read multiple times, resulting in an $O(k \cdot \text{nnz})$ bandwidth cost. Further, poor temporal locality in vector accesses can increase the cost of reading the vectors from $O(kn)$ to $O(k \cdot \text{nnz})$.

Algorithm 3 SA0

1: **for** $j = 1$ to k **do**
2: **for** $\ell \in \{1, \ldots, p\}$ **do**
3: Load vector entries $N(V_\ell^{(j)})$
4: Load matrix rows $\{A_i : i \in I_\ell\}$
5: Compute vector entries $V_\ell^{(j)}$
6: Store vector entries $V_\ell^{(j)}$
7: **end for**
8: **end for**

A communication-avoiding version, 'SA1', is shown in Algorithm 4. The variant we show here is essentially a sequential execution of PA1. This method computes the Krylov basis vectors in a block row-wise fashion, attempting to reduce the cost of reading A at the price of redundant computation and additional bandwidth cost. We note that there exist other variants of a communication-avoiding sequential matrix powers computation which avoid all redundant computation; see [63, 102, 126].

Algorithm 4 SA1

1: **for** $\ell \in \{1, \ldots, p\}$ **do**
2: Load vector entries $R(V_\ell) \cap y^{(0)}$
3: Load matrix rows $\{A_i : y_i^{(j)} \in R(V_\ell) \backslash y^{(0)}\}$
4: Compute vector entries $R(V_\ell) \backslash y^{(0)}$
5: Store vector entries $V_\ell \backslash y^{(0)}$
6: **end for**

Table 3.4 compares the complexity of SA0 and SA1. Again, these costs are precise but opaque; as an example, in Table 3.5 we show these costs for a $(2b+1)^d$-point stencil on an $N \times \cdots \times N$ d-dimensional mesh ($n = N^d$) partitioned into $p = \rho^d$ subcubes, where we assume that $w = N/\rho$ is an integer so that each of the p cache blocks corresponds to a subcube and that $k = O(w/b)$. For examples of these costs for other stencil computations see the tables given in [63]. As before, F denotes the number of arithmetic operations performed, W denotes the number of words read from and written to slow memory, and S denotes the number of messages in which these words were moved. The quantities $M_{Y,\text{slow}}$ and $M_{A,\text{slow}}$ denote the total memory needed to store the vectors Y and the matrix A in slow memory (main memory), respectively, and $M_{Y,\text{fast}}$ and $M_{A,\text{fast}}$ denote the same but for fast memory (cache). In the implicit case, for both algorithms, we would have $M_A = O(1)$ and neglect A's contribution to W and, for SA1, to S.

The potential savings of SA1 over SA0 are similar to the parallel case. Again, we would like to compare costs componentwise for the same rowwise partition, but this is now complicated by different fast memory requirements. Looking at example costs for stencil computations, shown in Table 3.5, we see that $M_{Y,\text{fast}}$ is greater for SA1 than for SA0 by a factor of $O(k/u)$, so when $u = \Theta(1)$, it is possible to pick the number of blocks p to be smaller for SA0 by a factor of $O(k)$. In this case, the latency costs of SA0 and SA1 are comparable. In the explicit case with $u = \Theta(1)$, SA1 achieves a $\Theta(k)$-fold decrease in the component of the bandwidth cost due to reading A, which is the main benefit, but no savings due to reading and writing vectors. In the implicit case with $u = \Theta(1)$, the bandwidth costs of the two approaches are comparable. For longer recurrences, i.e., $u = \Theta(k)$, the number of blocks can not be reduced in SA0 and in this case SA1 exhibits a $\Theta(k)$-fold decrease in latency cost in addition to a $\Theta(k)$-fold savings in bandwidth.

Table 3.4: Upper bounds on costs of SA0 and SA1 for general matrices

SA0	
F	$2 \cdot \sum_{\ell=1}^{p} \sum_{y_i^{(j)} \in V_\ell \setminus y^{(0)}} \text{nnz}(A_i) + \min\{u, j\}$
W	$\sum_{j=1}^{k} \sum_{q=1}^{p} \lvert N(V_\ell^{(j)}) \rvert + \lvert I_\ell \rvert + c \cdot \sum_{i \in I_\ell} \text{nnz}(A_i)$
S	$k \cdot \sum_{\ell=1}^{p} 2 + \lvert r : V_r \cap N(V_\ell) \neq \emptyset \rvert$
$M_{Y,\text{slow}}$	$(k+1)n$
$M_{Y,\text{fast}}$	$\max_{\ell=1}^{p} \lvert N(V_\ell^{(k)}) \rvert + \lvert I_\ell \rvert$
$M_{A,\text{slow}}$	$c \cdot \text{nnz}(A)$
$M_{A,\text{fast}}$	$c \cdot \max_{\ell=1}^{p} \sum_{i \in I_\ell} \text{nnz}(A_i)$
SA1	
F	$2 \cdot \sum_{\ell=1}^{p} \sum_{y_i^{(j)} \in R(V_\ell) \setminus y^{(0)}} \text{nnz}(A_i) + \min\{u, j\}$
W	$\sum_{j=1}^{k} \sum_{q=1}^{p} \lvert R(V_\ell) \setminus y^{(0)} \rvert + \lvert V_\ell \setminus y^{(0)} \rvert + c \cdot \sum_{i : y_i^{(j)} \in R(V_\ell) \setminus y^{(0)}} \text{nnz}(A_i)$
S	$\sum_{\ell=1}^{p} 2 + \lvert r : V_r \cap R(V_\ell) \neq \emptyset \rvert$
$M_{Y,\text{slow}}$	$(k+1)n$
$M_{Y,\text{fast}}$	$\max_{\ell=1}^{p} \lvert R(V_\ell) \rvert$
$M_{A,\text{slow}}$	$c \cdot \text{nnz}(A)$
$M_{A,\text{fast}}$	$c \cdot \max_{\ell=1}^{p} \sum_{i : y_i^{(j)} \in R(V_\ell) \setminus y^{(0)}} \text{nnz}(A_i)$

3.2.4 Tradeoffs in Computation, Data Movement, and Synchronization in Parallel Matrix Powers Computations

In Solomonik et al. [165], tradeoffs are derived between three basic costs of a parallel algorithm: the network latency cost S, the number of words moved W, and the number of local floating point operations F. These tradeoffs give lower bounds on the execution time which are dependent on the problem size but independent of the number of processors. That is, lower bounds on the parallel execution time can be determined for any algorithm computed by a system composed of any number of homogeneous components, each with associated computational, communication, and synchronization costs. By considering the maximum work and data moved over any execution path during the parallel computation (rather than the total communication volume), one can obtain new insights into the characteristics of parallel schedules for algorithms with non-trivial dependency structures.

In [165], these bounds have been applied to describe the strong scaling limit of a number of algorithms in terms of these three quantities, including computation of an $O(s)$-dimensional Krylov basis for $(2b+1)^d$-point stencils on a d-dimensional mesh. In particular, it is shown that for any parallel execution of this computation, for some $k \in \{1, \ldots, s\}$,

$$F = \Omega(b^d \cdot k^d \cdot s), \quad W = \Omega(b^d k^{d-1} \cdot s), \quad \text{and } S = \Omega(s/k),$$

Table 3.5: Upper bounds on costs of SA0 and SA1 for a $(2b+1)^d$-point stencil on a $N \times \cdots \times N$ d-dimensional mesh with $n = N^d$, partitioned into $p = \rho^d$ subcubes with integer edge length $w = N/\rho$

SA0	
F	$2n \cdot \sum_{j=1}^{k} (2b+1)^d + \min\{u, j\}$
W	$p \cdot \sum_{j=1}^{k} \left(\min\{N, w+2b\}^d + (1 + \max\{0, \min\{u, j\} - 1\})w^d + c(2b+1)^d w^d\right)$
S	$kp(1 + \min\{\rho, 1 + 2\lceil b/w \rceil\}^d)$
$M_{Y,\text{slow}}$	$(k+1)n$
$M_{Y,\text{fast}}$	$\min\{N, w+2b\}^d + (1 + \max\{0, u-1\})w^d$
$M_{A,\text{slow}}$	$c \cdot \text{nnz}(A)$
$M_{A,\text{fast}}$	$c(2b+1)^d w^d$
SA1	
F	$2p \cdot \sum_{j=1}^{k} \left((2b+1)^d + \min\{u, j\}\right) \min\{N, w + 2b(j-1)\}^d$
W	$p(\min\{N, w+2bk\}^d + kw^d + c(2b+1)^d \min\{N, w+2b(k-1)\}^d)$
S	$p(1 + \min\{\rho, 1 + 2\lceil bk/w \rceil\}^d)$
$M_{Y,\text{slow}}$	$(k+1)n$
$M_{Y,\text{fast}}$	$\sum_{j=0}^{k} \min\{N, w+2bj\}^d$
$M_{A,\text{slow}}$	$c \cdot \text{nnz}(A)$
$M_{A,\text{fast}}$	$c(2b+1)^d \min\{N, w+2b(k-1)\}^d$

and furthermore,

$$F \cdot S^d = \Omega(b^d \cdot s^{d+1}) \quad \text{and} \quad W \cdot S^{d-1} = \Omega(b^d \cdot s^d).$$

An s-dimensional Krylov basis can be computed by s/k invocations of PA1 (Algorithm 2). Under the assumption of well-partitioning, i.e., $k = O((n/p)^{1/d}/b)$, for PA1 we have

$$F = O(b^d \cdot k^d \cdot s) \quad W = O(b^d k^{d-1} \cdot s), \quad \text{and } S = O(s/k),$$

and thus PA1 attains the lower bounds and lower bound tradeoffs.

Again, there is no obstacle to running multiple calls to the matrix powers kernel to compute the $O(s)$-dimension basis required by the CA-KSM. The optimal choice of k to use in matrix powers computations may differ from the optimal choice of s, depending on the relative bandwidth/latency and flops/latency costs. In fact, for a number of test cases, Yamazaki et al. [196] found that best k parameter for the matrix powers kernel was always less than the optimal s parameter.

3.2.5 Numerical Aspects of Computing Krylov Bases

Achieving numerical stability for CA-KSMs is more challenging than for classical KSMs. Avoiding communication necessitates that we compute the s-step bases using the matrix

powers kernel upfront in each outer loop, without performing orthogonalization operations in between successive SpMVs. This can cause the s-step bases to quickly become ill-conditioned, which as we will see in Chapter 5, has profound implications for convergence and accuracy.

The most straightforward reorganizations of KSMs to CA-KSMs give identical results in exact arithmetic, but (depending on the input matrix) may completely fail to converge because of numerical instability. This is not surprising, since the monomial basis $[y, Ay, \ldots, A^k y]$ converges to the eigenvector of the dominant eigenvalue of A as k grows, and so form an increasingly ill-conditioned basis of the Krylov subspace. This loss of linear independence can cause the underlying Krylov methods to break down or stagnate, and thus fail to converge. This was observed by early researchers in these methods (this related work is described in Section 2.4), who proposed using different, better conditioned polynomial bases $[y, \rho_1(A)y, \ldots, \rho_k(A)y]$, whose computation was discussed in Section 3.2. Newton and Chebyshev polynomials, discussed further below, are two commonly-used choices for constructing the s-step bases.

As previous authors have demonstrated (e.g., [102, 14, 149]), we have found that changing the polynomial basis improves stability and convergence properties; see numerical experiments in Section 4. We note that this is not enough to guarantee numerical stability comparable to the classical algorithm in all cases. In Section 6.1, we develop a generalization of the residual replacement approach of [181] to improve the correlation between independent recurrences for updating the solution vector and its residual. Our results show that in many cases, the approach of combining well-conditioned polynomial bases and residual replacement strategies can make CA-KSMs reliable.

We now review strategies for using Newton and the Chebyshev polynomials to improve conditioning in computing Krylov subspace bases, as well as a matrix equilibration technique.

3.2.5.1 Newton

This approach follows that of [14, 102, 149]. The (scaled) Newton polynomials are defined by the recurrence coefficients

$$\hat{\alpha}_j = \theta_j, \qquad \hat{\beta}_j = 0, \qquad \hat{\gamma}_j = \sigma_j \tag{3.2}$$

where the σ_j are *scaling factors* and the *shifts* θ_j are chosen to be eigenvalue estimates. After choosing s shifts, we permute them according to a Leja ordering (see, e.g., [151]). Informally, the Leja ordering recursively selects each θ_j to maximize a certain measure on the set $\{\theta_i\}_{i \leq j} \subset \mathbb{C}$; this helps avoid repeated or nearly-repeated shifts. Real matrices may have complex eigenvalues, so the Newton basis may introduce complex arithmetic; to avoid this, we use the modified Leja ordering [102], which exploits the fact that complex eigenvalues of real matrices occur in complex conjugate pairs. The modified Leja ordering means that for every complex shift $\theta_j \notin \mathbb{R}$ in the sequence, its complex conjugate $\overline{\theta_j}$ is adjacent, and the complex conjugate pairs (θ_j, θ_{j+1}) are permuted so that $\Im(\theta_j) > 0$. This leads to the

recurrence coefficients

$$\hat{\alpha}_j = \Re(\theta_j), \qquad \hat{\beta}_j = \begin{cases} -\Im(\theta_j)^2 & \theta_j = \overline{\theta_{j+1}} \wedge \Im(\theta_j) > 0 \\ 0 & \text{otherwise} \end{cases}, \qquad \hat{\gamma}_j = \sigma_j \tag{3.3}$$

Both Leja orderings can be computed efficiently. For CA-KSMs, choosing σ_j is challenging; we cannot simply scale each basis vector by its Euclidean norm as it is generated since this eliminates our communication savings. In our experiments, we pick all scaling factors $\sigma_j = 1$ (i.e., no scaling), instead relying entirely on matrix equilibration (described in a paragraph later in this section), to keep the spectral radius $\rho(A)$ close to 1.

3.2.5.2 Chebyshev

The Chebyshev basis requires two parameters, complex numbers d and c, where $d \pm c$ are the foci of a bounding ellipse for the spectrum of A. The scaled, shifted, and rotated Chebyshev polynomials $\{\tilde{\tau}_j\}_{j\geq 0}$ can then be written as

$$\tilde{\tau}_j(z) \coloneqq \tau_j((d-z)/c)/\tau_j(d/c) =: \tau_j((d-z)/c)/\sigma_j \tag{3.4}$$

where the Chebyshev polynomials (of the first kind) $\{\tau_j\}_{j\geq 0}$ are

$$\begin{aligned} &\tau_0(z) \coloneqq 1, \quad \tau_1(z) \coloneqq z, \quad \text{and} \\ &\tau_j(z) \coloneqq 2z\tau_{j-1}(z) - \tau_{j-2}(z) \quad \text{for } j > 1 \end{aligned} \tag{3.5}$$

substituting (3.4) into (3.5), we obtain

$$\begin{aligned} &\tilde{\tau}_0(z) = 1, \quad \tilde{\tau}_1(z) = \sigma_0(d-z)/(c\sigma_1), \quad \text{and} \\ &\tilde{\tau}_j(z) = 2\sigma_{j-1}(d-z)\tilde{\tau}_{j-1}(z)/(c\sigma_j) - \sigma_{j-2}\tilde{\tau}_{j-2}(z)/\sigma_j \quad \text{for } j > 1 \end{aligned} \tag{3.6}$$

Extracting coefficients, we obtain

$$\hat{\alpha}_j = d, \qquad \hat{\beta}_j = -c\sigma_j/(2\sigma_{j+1}), \qquad \hat{\gamma}_j = \begin{cases} -c\sigma_1/\sigma_0 & j = 0 \\ -c\sigma_{j+1}/(2\sigma_j) & j > 0 \end{cases} \tag{3.7}$$

Note that the transformation $z \to (d-z)/c$ maps ellipses with foci $f_{1,2} = d \pm c$ to ellipses with foci at ∓ 1, especially the line segment (f_1, f_2) to $(-1, 1)$. If A is real, then the ellipse is centered on the real axis; thus $d \in \mathbb{R}$, so c is either real or imaginary. In the former case, arithmetic will be real. In the latter case ($c \in i\mathbb{R}$), we avoid complex arithmetic by replacing $c \coloneqq c/i$. This is equivalent to rotating the ellipses 90°.

For real matrices, [104] also gives a three-term recurrence, where

$$\hat{\alpha}_j = d, \qquad \hat{\beta}_j = c^2/(4g), \qquad \hat{\gamma}_j = \begin{cases} 2g & j = 0 \\ g & j > 0 \end{cases} \tag{3.8}$$

It is assumed that the spectrum is bounded by the rectangle $\{z : |\Re(z) - d| \leq a, |\Im(z)| \leq b\}$ in the complex plane, where $a \geq 0$, $b \geq 0$, and d are real. Here we choose $c = \sqrt{a^2 - b^2}$, $g = \max\{a, b\}$. Note that for real-valued matrices, recurrence (3.8) is usually sufficient for capturing the necessary spectral information.

3.2.5.3 Matrix Equilibration

Successively scaling (i.e., normalizing) the Krylov vectors as they are generated increases the numerical stability of the basis generation step. As discussed in [102], successively scaling the basis vectors (e.g., by their Euclidean norms) is not possible in the CA variants, as it reintroduces the global communication requirement between SpMV operations. For CA-KSMs for solving linear systems $Ax = b$, an alternative is to perform matrix equilibration. For nonsymmetric matrices, this involves applying diagonal row and column scalings, D_r and D_c, such that each row and column of the equilibrated matrix $D_r A D_c$ has norm one. This is performed once offline before running the CA-KSM. Results in [102] indicate that this technique is an effective alternative to the successively scaled versions of the bases.

Chapter 4

New Communication-Avoiding Krylov Subspace Methods

In this chapter, we present the derivation of a number of new communication-avoiding Krylov subspace methods. In Section 4.1, we derive a communication-avoiding variant of a general nonsymmetric Lanczos method, based on the "BIOC" variant of nonsymmetric Lanczos of Gutknecht (see [94]), which also appears in [16]. We then derive nonsymmetric Lanczos-based CA-KSMs for solving linear systems including biconjugate gradient (CA-BICG) in Section 4.2, conjugate gradient squared (CA-CGS) in Section 4.3, and biconjugate gradient stabilized (CA-BICGSTAB) in Section 4.4, which have been adapted from the article [35] and technical report [34]. We also present a number of new communication avoiding variants of the upper and lower Lanczos bidiagonalization procedures, which can be used for singular value problems and form the basis for the least-squares QR method (LSQR) for solving least-squares problems, adapted from the technical report [27].

All communication-avoiding variants are mathematically, but not numerically, equivalent to the standard implementations, in the sense that after every s steps, they produce a solution identical to that of the conventional algorithm in exact arithmetic. We focus here on two-term recurrence versions of each method. As the sections in this chapter are notation-heavy, we first begin by establishing the notation that will be used throughout the derivations.

Notation The idea behind all communication-avoiding Krylov methods is to generate upfront bases for the Krylov subspaces containing the vector iterates for the next s steps. After a block orthogonalization step, which involves either computing a Gram matrix to store inner products of all generated basis vectors or computing a TSQR factorization of the generated basis vectors, s iterations are carried out by updating the coordinates of the length-n iterates in the generated bases, rather than updating the length-n iterates themselves.

All generated bases for the Krylov subspaces are denoted with calligraphic letters and have a subscript denoting the outer loop iteration, or the block of s steps they are used for.

We assume that these bases are computed by a three-term recurrence satisfying

$$\begin{aligned} &\rho_0(z) = 1, \quad \rho_1(z) = (z - \hat{\alpha}_0)\rho_0(z)/\hat{\gamma}_0, \quad \text{and} \\ &\rho_j(z) = ((z - \hat{\alpha}_{j-1})\rho_{j-1}(z) - \hat{\beta}_{j-2}\rho_{j-2}(z))/\hat{\gamma}_{j-1} \quad \text{for } j > 1. \end{aligned} \tag{4.1}$$

where $\rho_j(z)$ is a polynomial of degree j. We note that our derivations could be generalized to use polynomials satisfying longer (up to s-term) recurrences, although three-term recurrences are most commonly used. Thus the matrix recurrence for generation of the basis vectors can be written

$$A\underline{\mathcal{Y}}_k = \mathcal{Y}_k\mathcal{B}_k$$

where $\mathcal{B}_k$ is of the form

$$\begin{bmatrix} \hat{\alpha}_0 & \hat{\beta}_0 & & & 0 \\ \hat{\gamma}_0 & \hat{\alpha}_1 & \ddots & & \vdots \\ & \hat{\gamma}_1 & \ddots & \hat{\beta}_{i-2} & 0 \\ & & \ddots & \hat{\alpha}_{i-1} & 0 \\ & & & \hat{\gamma}_{i-1} & 0 \end{bmatrix} \in \mathbb{C}^{(i+1)\times(i+1)}. \tag{4.2}$$

As shown above, an underlined calligraphic letter indicates that all entries in the last column of the matrix are set to 0. When needed, a superscript is added to $\mathcal{B}_k$ to indicate which basis vectors the entries in $\mathcal{B}_k$ were used to generate (e.g., the above would become $\mathcal{B}_k^{(\mathcal{Y})}$). The dimension of $\mathcal{B}_k^{(\mathcal{Y})}$ will depend on the number of columns in $\mathcal{Y}_k$, i.e., i in (4.2) is one less than the number of columns in $\mathcal{Y}_k$.

Prime symbols are used to denote the length-$O(s)$ coordinate vectors. For example, if some length-n vector $v \in \mathcal{Y}_k$, then v' is the length-$O(s)$ vector such that $v = \mathcal{Y}_k v'$. In all derivations, we use the convention that iterations are 1-indexed.

4.1 Nonsymmetric Lanczos

Given an $n \times n$ matrix A and starting vectors y_1 and $\tilde{y}_1$ such that $\|y_1\|_2 = \|\tilde{y}_1\|_2 = 1$ and $(\tilde{y}_1, y_1) \neq 0$, i steps of the Lanczos process produce the decompositions

$$AY_i = Y_iT_i + \gamma_i y_{i+1}e_i^T \quad \text{and} \quad A^H\tilde{Y}_i = \tilde{Y}_i\tilde{T}_i + \tilde{\gamma}_i\tilde{y}_{i+1}e_i^T \tag{4.3}$$

where $Y_i = [y_1, \ldots, y_i]$ and T_i is an $i \times i$ tridiagonal matrix (similarly for $\tilde{Y}_i$ and $\tilde{T}_i$). The biorthogonal matrices Y_{i+1} and $\tilde{Y}_{i+1}$ are bases for the Krylov subspaces $\mathcal{K}_{i+1}(A, y_1)$ and $\mathcal{K}_{i+1}(A^H, \tilde{y}_1)$, resp. The eigenvalues of T_i are called Petrov values (or Ritz values for SPD A), and are useful as approximations for the eigenvalues of A.

In this section, we present a communication-avoiding version of the Lanczos biorthogonalization method which uses a pair of coupled two-term recurrences, referred to by Gutknecht

as the "BIOC" variant of nonsymmetric Lanczos (see, e.g., [94]). Keeping with this naming convention, we call our method CA-BIOC. This first appeared in Section 8.2.2 of the Acta Numerica paper [16]. We note that in the case that A is SPD, elimination of certain computations from BIOC and CA-BIOC gives algorithms for symmetric Lanczos and communication-avoiding symmetric Lanczos, resp. The BIOC algorithm, shown in Algorithm 5, is governed by two coupled two-term recurrences (rather than the single three-term recurrence (4.3)) which can be written

$$\begin{aligned} Y_i = V_i U_i \quad & \tilde{Y}_i = \tilde{V}_i \tilde{U}_i \\ AV_i = Y_i L_i + \gamma_i y_{i+1} e_i^T \quad & A^H \tilde{V}_i = \tilde{Y}_i \tilde{L}_i + \tilde{\gamma}_i \tilde{y}_{i+1} e_i^T \end{aligned} \tag{4.4}$$

where V_i and $\tilde{V}_i$ are biconjugate with respect to A, i.e., $(\tilde{v}_\ell, Av_j) \neq 0$ if $\ell = j$ and $(\tilde{v}_\ell, Av_j) = 0$ if $\ell \neq j$, and as before, Y_i and $\tilde{Y}_i$ are biorthogonal. The matrices

$$L_i = \begin{pmatrix} \phi_1 & & & \\ \gamma_1 & \phi_2 & & \\ & \ddots & \ddots & \\ & & \gamma_{i-1} & \phi_i \end{pmatrix} \quad \text{and} \quad U_i = \begin{pmatrix} \psi_1 & & & \\ 1 & \psi_2 & & \\ & \ddots & \ddots & \\ & & 1 & \psi_i \end{pmatrix} \tag{4.5}$$

are the LU factors of T_i in (4.3). Although BIOC breakdown can occur if the (pivot-free) LU factorization of T_i does not exist, the four bidiagonal matrices in (4.4) are preferable to the two tridiagonal matrices in (4.3) for a number of reasons [146]. Derivations for a communication-avoiding three-term recurrence variant of symmetric Lanczos can be found in [102].

Algorithm 5 Classical BIOC

Input: $n \times n$ matrix A and length-n starting vectors $y_1, \tilde{y}_1$ such that $\|y_1\|_2 = \|\tilde{y}_1\|_2 = 1$, and $\delta_1 = (\tilde{y}_1, y_1) \neq 0$.

Output: Matrices Y_i, $\tilde{Y}_i$, T_i, $\tilde{T}_i$, and vectors $y_{i+1}, \tilde{y}_{i+1}$ satisfying (4.3)

$v_1 = \tilde{v}_1 = y_1$
for $i = 1, 2, \ldots,$ until convergence **do**
  $\hat{\delta}_i = (\tilde{v}_i, Av_i)$
  $\phi_i = \hat{\delta}_i / \delta_i$, $\tilde{\phi}_i = \overline{\phi}_i$
  Choose $\gamma_i, \tilde{\gamma}_i \neq 0$
  $y_{i+1} = (Av_i - \phi_i y_i)/\gamma_i$
  $\tilde{y}_{i+1} = (A^H \tilde{v}_i - \tilde{\phi}_i \tilde{y}_i)/\tilde{\gamma}_i$
  $\delta_{i+1} = (\tilde{y}_{i+1}, y_{i+1})$
  $\psi_i = \overline{\tilde{\gamma}}_i \delta_{i+1} / \hat{\delta}_i$, $\tilde{\psi}_i = \overline{\gamma_i \delta_{i+1} / \hat{\delta}_i}$
  $v_{i+1} = y_{i+1} - \psi_i v_i$
  $\tilde{v}_{i+1} = \tilde{y}_{i+1} - \tilde{\psi}_i \tilde{v}_i$
end for

Now, suppose we want to perform blocks of $s \geq 1$ iterations at once. That is, we wish to calculate $[v_{sk+2}, \ldots, v_{sk+s+1}]$, $[\tilde{v}_{sk+2}, \ldots, \tilde{v}_{sk+s+1}]$, $[y_{sk+2}, \ldots, y_{sk+s+1}]$, and $[\tilde{y}_{sk+2}, \ldots, \tilde{y}_{sk+s+1}]$, given $\{v_{sk+1}, \tilde{v}_{sk+1}, y_{sk+1}, \tilde{y}_{sk+1}\}$, for $k \geq 0$.

By induction on lines {6, 7, 10, 11} of classical BIOC (Algorithm 5), it can be shown that, for iterations $sk + j$, $j \in \{1, \ldots, s + 1\}$, the vector iterates satisfy

$$\begin{aligned} v_{sk+j}, y_{sk+j} &\in \mathcal{K}_{s+1}(A, v_{sk+1}) + \mathcal{K}_s(A, y_{sk+1}), \\ \tilde{v}_{sk+j}, \tilde{y}_{sk+j} &\in \mathcal{K}_{s+1}(A^H, \tilde{v}_{sk+1}) + \mathcal{K}_s(A^H, \tilde{y}_{sk+1}), \end{aligned} \tag{4.6}$$

where we exploit the nesting of the Krylov bases, i.e., $\mathcal{K}_j(A, v_{sk+1}) \subseteq \mathcal{K}_{j+1}(A, v_{sk+1})$ (and similarly for other starting vectors). Then to compute iterates $sk+2$ through $sk+s+1$, we will use the Krylov basis matrices

$$\begin{aligned} \mathcal{V}_k &= [\rho_0(A)v_{sk+1}, \rho_1(A)v_{sk+1}, \ldots, \rho_s(A)v_{sk+1}], & \text{span}(\mathcal{V}_k) &= \mathcal{K}_{s+1}(A, v_{sk+1}), \\ \tilde{\mathcal{V}}_k &= [\rho_0(A^H)\tilde{v}_{sk+1}, \rho_1(A^H)\tilde{v}_{sk+1}, \ldots, \rho_s(A^H)\tilde{v}_{sk+1}], & \text{span}(\tilde{\mathcal{V}}_k) &= \mathcal{K}_{s+1}(A^H, \tilde{v}_{sk+1}), \\ \mathcal{Y}_k &= [\rho_0(A)y_{sk+1}, \rho_1(A)y_{sk+1}, \ldots, \rho_{s-1}(A)y_{sk+1}], & \text{span}(\mathcal{Y}_k) &= \mathcal{K}_s(A, y_{sk+1}), \\ \tilde{\mathcal{Y}}_k &= [\rho_0(A^H)\tilde{y}_{sk+1}, \rho_1(A^H)\tilde{y}_{sk+1}, \ldots, \rho_{s-1}(A^H)\tilde{y}_{sk+1}], & \text{span}(\tilde{\mathcal{Y}}_k) &= \mathcal{K}_s(A^H, \tilde{y}_{sk+1}), \end{aligned} \tag{4.7}$$

where $\rho_j(z)$ is a polynomial of degree j satisfying (4.1). Assuming A is well partitioned, these matrices can be computed in a communication-avoiding way using the 'PA1' or 'SA1' algorithms described in Section 3.2.

Using the defined Krylov matrices (4.7) and the relations (4.6), we can represent components of the BIOC iterates in $\mathbb{C}^n$ by their coordinates in the Krylov bases, that is, subspaces of $\mathbb{C}^n$ of dimension at most $2s+1$. We introduce coefficient vectors $\{v'_{k,j}, \tilde{v}'_{k,j}, y'_{k,j}, \tilde{y}'_{k,j}\}$, each of length $2s+1$, to represent the length n vectors $\{v_{sk+j}, \tilde{v}_{sk+j}, y_{sk+j}, \tilde{y}_{sk+j}\}$ for $j \in \{1, \ldots, s+1\}$. That is,

$$\begin{aligned} v_{sk+j} &= [\mathcal{V}_k, \mathcal{Y}_k]v'_{k,j}, & y_{sk+j} &= [\mathcal{V}_k, \mathcal{Y}_k]y'_{k,j}, \\ \tilde{v}_{sk+j} &= [\tilde{\mathcal{V}}_k, \tilde{\mathcal{Y}}_k]\tilde{v}'_{k,j}, & \tilde{y}_{sk+j} &= [\tilde{\mathcal{V}}_k, \tilde{\mathcal{Y}}_k]\tilde{y}'_{k,j}, \end{aligned} \tag{4.8}$$

where the base cases for these recurrences are given by

$$v'_{k,1} = \tilde{v}'_{k,1} = [1, 0_{1,2s}]^T, \quad \text{and} \quad y'_{k,1} = \tilde{y}'_{k,1} = [0_{1,s+1}, 1, 0_{1,s-1}]^T. \tag{4.9}$$

Then, the iterate updates (lines {6, 7, 10, 11}) in the Krylov basis become

$$[\mathcal{V}_k, \mathcal{Y}_k]y'_{k,j+1} = (A[\mathcal{V}_k, \mathcal{Y}_k]v'_{k,j} - \phi_{sk+j}[\mathcal{V}_k, \mathcal{Y}_k]y'_{k,j})/\gamma_{sk+j}, \tag{4.10}$$

$$[\tilde{\mathcal{V}}_k, \tilde{\mathcal{Y}}_k]\tilde{y}'_{k,j+1} = (A^H[\tilde{\mathcal{V}}_k, \tilde{\mathcal{Y}}_k]\tilde{v}'_{k,j} - \tilde{\phi}_{sk+j}[\tilde{\mathcal{V}}_k, \tilde{\mathcal{Y}}_k]\tilde{y}'_{k,j})/\tilde{\gamma}_{sk+j}, \tag{4.11}$$

$$[\mathcal{V}_k, \mathcal{Y}_k]v'_{k,j+1} = [\mathcal{V}_k, \mathcal{Y}_k]y'_{k,j+1} + \psi_{sk+j}[\mathcal{V}_k, \mathcal{Y}_k]v'_{k,j}, \quad \text{and} \tag{4.12}$$

$$[\tilde{\mathcal{V}}_k, \tilde{\mathcal{Y}}_k]\tilde{v}'_{k,j+1} = [\tilde{\mathcal{V}}_k, \tilde{\mathcal{Y}}_k]\tilde{y}'_{k,j+1} + \tilde{\psi}_{sk+j}[\tilde{\mathcal{V}}_k, \tilde{\mathcal{Y}}_k]\tilde{v}'_{k,j}. \tag{4.13}$$

for $j \in \{1, \ldots, s\}$.

Next, we represent the multiplications by A and A^H (lines 6 and 7) in the new coordinates, in order to manipulate (4.10) and (4.11). We first note that the recurrence (4.1) for generating the matrices $\mathcal{V}_k$, $\tilde{\mathcal{V}}_k$, $\mathcal{Y}_k$, and $\tilde{\mathcal{Y}}_k$ can be written in matrix form as

$$\begin{aligned} A\underline{\mathcal{V}}_k &= \mathcal{V}_k \mathcal{B}_k^{(\mathcal{V})}, \quad & A\underline{\mathcal{Y}}_k &= \mathcal{Y}_k \mathcal{B}_k^{(\mathcal{Y})}, \\ A^H\underline{\tilde{\mathcal{V}}}_k &= \tilde{\mathcal{V}}_k \mathcal{B}_k^{(\tilde{\mathcal{V}})}, & A^H\underline{\tilde{\mathcal{Y}}}_k &= \tilde{\mathcal{Y}}_k \mathcal{B}_k^{(\tilde{\mathcal{Y}})}, \end{aligned} \tag{4.14}$$

where $\underline{\mathcal{V}}_k$, $\underline{\tilde{\mathcal{V}}}_k$, $\underline{\mathcal{Y}}_k$, and $\underline{\tilde{\mathcal{Y}}}_k$ are the same as $\mathcal{V}_k$, $\tilde{\mathcal{V}}_k$, $\mathcal{Y}_k$, and $\tilde{\mathcal{Y}}_k$, respectively, but with the last column set to 0, and the $\mathcal{B}_k$'s are of the form (4.2) with $i = s$ for $\mathcal{B}_k^{(\mathcal{V})}$ and $\mathcal{B}_k^{(\tilde{\mathcal{V}})}$ and $i = s-1$ for $\mathcal{B}_k^{(\mathcal{Y})}$ and $\mathcal{B}_k^{(\tilde{\mathcal{Y}})}$.

To simplify notation, we let $\mathcal{Z}_k = [\mathcal{V}_k, \mathcal{Y}_k]$, $\tilde{\mathcal{Z}}_k = [\tilde{\mathcal{V}}_k, \tilde{\mathcal{Y}}_k]$, $\underline{\mathcal{Z}}_k = [\underline{\mathcal{V}}_k, \underline{\mathcal{Y}}_k]$, $\underline{\tilde{\mathcal{Z}}}_k = [\underline{\tilde{\mathcal{V}}}_k, \underline{\tilde{\mathcal{Y}}}_k]$,

$$\mathcal{B}_k^{(\mathcal{Z})} = \begin{bmatrix} \mathcal{B}_k^{(\mathcal{V})} & 0 \\ 0 & \mathcal{B}_k^{(\mathcal{Y})} \end{bmatrix}, \quad \text{and} \quad \mathcal{B}_k^{(\tilde{\mathcal{Z}})} = \begin{bmatrix} \mathcal{B}_k^{(\tilde{\mathcal{V}})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{Y}})} \end{bmatrix}. \tag{4.15}$$

This allows us to write

$$A\underline{\mathcal{Z}}_k = \mathcal{Z}_k \mathcal{B}_k^{(\mathcal{Z})} \quad \text{and} \quad A\underline{\tilde{\mathcal{Z}}}_k = \tilde{\mathcal{Z}}_k \mathcal{B}_k^{(\tilde{\mathcal{Z}})}. \tag{4.16}$$

We need to perform multiplication of A with each v_{sk+j} (likewise for A^H and $\tilde{v}_{sk+j}$) for $j \in \{1, \ldots, s\}$. By (4.6) and (4.8),

$$\begin{aligned} Av_{sk+j} &= A\mathcal{Z}_k v'_{k,j} = A\underline{\mathcal{Z}}_k v'_{k,j} = \mathcal{Z}_k \mathcal{B}_k^{(\mathcal{Z})} v'_{k,j}, \\ A^H\tilde{v}_{sk+j} &= A^H\tilde{\mathcal{Z}}_k \tilde{v}'_{k,j} = A^H\underline{\tilde{\mathcal{Z}}}_k \tilde{v}'_{k,j} = \tilde{\mathcal{Z}}_k \mathcal{B}_k^{(\tilde{\mathcal{Z}})} \tilde{v}'_{k,j}. \end{aligned} \tag{4.17}$$

We now substitute (4.8) and (4.17) into classical BIOC. The vector updates in each of lines 6, 7, 10, and 11 are now expressed as a linear combination of the columns of the Krylov basis matrices. We can match coefficients on the right- and left-hand sides to obtain the governing recurrences

$$y'_{k,j+1} = (\mathcal{B}_k^{(\mathcal{Z})} v'_{k,j} - \phi_{sk+j} y'_{k,j})/\gamma_{sk+j}, \tag{4.18}$$

$$\tilde{y}'_{k,j+1} = (\mathcal{B}_k^{(\tilde{\mathcal{Z}})} \tilde{v}'_{k,j} - \tilde{\phi}_{sk+j} \tilde{y}'_{k,j})/\tilde{\gamma}_{sk+j}, \tag{4.19}$$

$$v'_{k,j+1} = y'_{k,j+1} + \psi_{sk+j} v'_{k,j}, \tag{4.20}$$

$$\tilde{v}'_{k,j+1} = \tilde{y}'_{k,j+1} + \tilde{\psi}_{sk+j} \tilde{v}'_{k,j}, \tag{4.21}$$

for $j \in \{1, \ldots, s\}$.

We also need scalar quantities $\hat{\delta}_{sk+j}$ and δ_{sk+j} which are computed from dot products involving the vector iterates. We represent these dot products (lines 3 and 8) in the new basis, using the Gram(-like) matrix

$$G_k = \tilde{\mathcal{Z}}_k^H \mathcal{Z}_k \tag{4.22}$$

of size $(2s+1) \times (2s+1)$, which can be computed with one Allreduce operation (see Section 3.1). We can then write the required inner products as

$$(\tilde{y}_{sk+j+1}, y_{sk+j+1}) = (\tilde{y}'_{k,j+1}, G_k y'_{k,j+1}) \tag{4.23}$$

$$(\tilde{v}_{sk+j}, Av_{sk+j}) = (\tilde{v}'_{k,j}, G_k \mathcal{B}_k^{(\mathcal{Z})} v'_{k,j}) \tag{4.24}$$

In our BIOC formulation, we have allowed freedom in choosing the values γ_{sk+j}. It is common to choose starting vectors $\|y_1\|_2 = \|\tilde{y}_1\|_2$ and choose γ_{sk+j} and $\tilde{\gamma}_{sk+j}$ such that $\|y_{sk+j+1}\|_2 = \|\tilde{y}_{sk+j+1}\|_2 = 1$, i.e.,

$$\gamma_{sk+j} = \sqrt{(Av_{sk+j} - \phi_{sk+j} y_{sk+j}, Av_{sk+j} - \phi_{sk+j} y_{sk+j})}$$
$$\tilde{\gamma}_{sk+j} = \sqrt{(A^H \tilde{v}_{sk+j} - \tilde{\phi}_{sk+j} \tilde{y}_{sk+j}, A^H \tilde{v}_{sk+j} - \tilde{\phi}_{sk+j} \tilde{y}_{sk+j})}$$

In this case, CA-BIOC also requires computing the matrices

$$G_k^{(\mathcal{Z})} = \mathcal{Z}_k^H \mathcal{Z}_k \quad \text{and} \quad G_k^{(\tilde{\mathcal{Z}})} = \tilde{\mathcal{Z}}_k^H \tilde{\mathcal{Z}}_k \tag{4.25}$$

in each outer loop. Note these matrices could be computed simultaneously with G_k. Then we can compute γ_{sk+j} and $\tilde{\gamma}_{sk+j}$ by

$$\|Av_{sk+j} - \phi_{sk+j} y_{sk+j}\|_2 = \sqrt{(\mathcal{B}_k^{(\mathcal{Z})} v'_{k,j} - \phi_{sk+j} y'_{k,j}, G_k^{(\mathcal{Z})} (\mathcal{B}_k^{(\mathcal{Z})} v'_{k,j} - \phi_{sk+j} y'_{k,j}))}, \tag{4.26}$$

$$\left\| A^H \tilde{v}_{sk+j} - \tilde{\phi}_{sk+j} \tilde{y}_{sk+j} \right\|_2 = \sqrt{(\mathcal{B}_k^{(\tilde{\mathcal{Z}})} \tilde{v}'_{k,j} - \tilde{\phi}_{sk+j} \tilde{y}'_{k,j}, G_k^{(\tilde{\mathcal{Z}})} (\mathcal{B}_k^{(\tilde{\mathcal{Z}})} \tilde{v}'_{k,j} - \tilde{\phi}_{sk+j} \tilde{y}'_{k,j}))} \tag{4.27}$$

in each inner loop iteration with no communication.

We now assemble the CA-BIOC method in Algorithm 6.

In parallel, the only communication in CA-BIOC occurs in the outer loop, in lines 3 and 4; the inner loop lines operate on operands of size $O(ns/p + s^2)$, which we assume fit locally on each of the p processors. Therefore, communication does not occur in the inner loop, and CA-BIOC can take s steps per $O(1)$ rounds of messages. In contrast, classical BIOC can only take 1 step per $O(1)$ rounds of messages; this is an $\Theta(s)$-fold decrease in latency for well-partitioned A.

In order to compute $s > 1$ vectors at one time, we can use the 'PA1' algorithm described in Section 3.2.2. Using PA1, each processor requires additional source vector entries (the ghost zones) and must perform redundant computation. If A is well partitioned, then the ghost zones grow slowly with respect to s; thus the additional bandwidth and computation are both lower order terms. We also note that in CA-BIOC, each processor must send $\Theta(s^2 \log_2 p)$ words to perform the matrix multiplication in line 5, while the equivalent dot products in s steps of classical BIOC only require $\Theta(s \log_2 p)$ words moved per processor (see Section 3.1). Construction of G_k brings the computational cost of the dense operations to $O(s^2 n/p)$, a factor of $O(s)$ larger than the classical algorithm. Under the reasonable

Algorithm 6 Communication-Avoiding BIOC

Input: $n \times n$ matrix A and length-n starting vectors $y_1, \tilde{y}_1$ such that $\|y_1\|_2 = \|\tilde{y}_1\|_2 = 1$, and $\delta_1 = (\tilde{y}_1, y_1) \neq 0$.

Output: Matrices $Y_{sk+s}, \tilde{Y}_{sk+s}, T_{sk+s}, \tilde{T}_{sk+s}$ and vectors $y_{sk+s+1}, \tilde{y}_{sk+s+1}$ satisfying (4.3)

1: $v_1 = \tilde{v}_1 = y_1$
2: **for** $k = 0, 1, \ldots,$ until convergence **do**
3: Compute $\mathcal{Z}_k = [\mathcal{V}_k, \mathcal{Y}_k]$ and $\tilde{\mathcal{Z}}_k = [\tilde{\mathcal{V}}_k, \tilde{\mathcal{Y}}_k]$ according to (4.7)
4: Compute G_k according to (4.22)
5: Assemble $\mathcal{B}_k^{(\mathcal{Z})}$ and $\mathcal{B}_k^{(\tilde{\mathcal{Z}})}$ such that (4.16) holds
6: Initialize $\{v'_{k,1}, \tilde{v}'_{k,1}, y'_{k,1}, \tilde{y}'_{k,1}\}$ according to (4.9)
7: **for** $j = 1$ to s **do**
8: $\hat{\delta}_{sk+j} = (\tilde{v}'_{k,j}, G_k \mathcal{B}_k^{(\mathcal{Z})} v'_{k,j})$
9: $\phi_{sk+j} = \hat{\delta}_{sk+j}/\delta_{sk+j}, \quad \tilde{\phi}_{sk+j} = \overline{\phi}_{sk+j}$
10: Choose $\gamma_{sk+j}, \tilde{\gamma}_{sk+j} \neq 0$
11: $y'_{k,j+1} = (\mathcal{B}_k^{(\mathcal{Z})} v'_{k,j} - \phi_{sk+j} y'_{k,j})/\gamma_{sk+j}$
12:
13: $\tilde{y}'_{k,j+1} = (\mathcal{B}_k^{(\tilde{\mathcal{Z}})} \tilde{v}'_{k,j} - \tilde{\phi}_{sk+j} \tilde{y}'_{k,j})/\tilde{\gamma}_{sk+j}$
14:
15: $\delta_{sk+j+1} = (\tilde{y}'_{k,j+1}, G_k y'_{k,j+1})$
16: $\psi_{sk+j} = \overline{\tilde{\gamma}}_{sk+j} \delta_{sk+j+1}/\hat{\delta}_{sk+j} \quad \tilde{\psi}_{sk+j} = \overline{\gamma_{sk+j} \delta_{sk+j+1}/\hat{\delta}_{sk+j}}$
17: $v'_{k,j+1} = y'_{k,j+1} + \psi_{sk+j} v'_{k,j}$
18:
19: $\tilde{v}'_{k,j+1} = \tilde{y}'_{k,j+1} + \tilde{\psi}_{sk+j} \tilde{v}'_{k,j}$
20:
21: **end for**
22: Recover $\{v_{sk+j+1}, \tilde{v}_{sk+j+1}, y_{sk+j+1}, \tilde{y}_{sk+j+1}\}$ for $j \in \{1, \ldots, s\}$ according to (4.8)
23: **end for**

assumption that the number of nonzeros of A per processor is greater than $O(sn/p)$, the SpMV computations dominate dense operations in the classical algorithm, so the additional cost of computing G_k is not a limiting factor.

In serial, CA-BIOC moves data on lines 3, 4, and 22. We are most concerned with the data movement in reading A (line 3), since it is common that A requires far more storage than the vectors on which it operates. If A is well partitioned, then using the algorithm SA1 described in Section 3.2.3, line 3 reads A $2 + o(1)$ times, whereas s iterations of classical BIOC read A $2s$ times. This is an s-fold savings in latency and bandwidth for reading A. All three lines (3, 4, and 22) involve reading the vector iterates of length n, asymptotically the same communication cost as the classical algorithm. Similar to the parallel case, we perform a factor of $O(s)$ more computation in the dense operations, which is a lower-order term if the number of nonzeros in A is greater than $O(sn)$.

4.2 Biconjugate Gradient

We now derive a communication-avoiding version of the biconjugate gradient method (BICG) for solving nonsymmetric linear systems $Ax = b$, which we will call CA-BICG. We first review BICG (Algorithm 7) and demonstrate its relation to BIOC. Here and in other sections in this Chapter we assume here that no breakdown occurs in the two-sided nonsymmetric Lanczos process; we discuss the extension of the look-ahead technique for avoiding breakdown situations to CA-KSMs in Section 6.4. The BICG method starts with an initial solution guess x_1 and corresponding residuals $r_1 = \tilde{r}_1 = b - Ax_1$. In each iteration i, the solution x_{i+1} is updated by a vector selected from $\mathcal{K}_i(A, r_1)$ such that $(b - Ax_1) * \mathcal{K}_i(A^H, \tilde{r}_1)$, the so-called Petrov-Galerkin condition.

The solution can then be written as $x_{i+1} = x_1 + Y_i c_i$, where Y_i and $\tilde{Y}_i$ are the matrices of biorthogonal basis vectors of $\mathcal{K}_i(A, r_1)$ and $\mathcal{K}_i(A^H, \tilde{r}_1)$, respectively, produced by BIOC with starting guess $y_1 = \tilde{y}_1 = r_1$.

Enforcing the Petrov-Galerkin condition gives $c_i = T_i^{-1} e_1$. Then, using 4.4 and 4.5,

$$\begin{aligned} x_{i+1} &= x_1 + Y_i T_i^{-1} e_1 \\ &= x_1 + Y_i U_i^{-1} L_i^{-1} e_1 \\ &= x_1 + V_i L_i^{-1} e_1 \\ &= -\frac{\phi_i}{\gamma_i} x_i - \frac{1}{\gamma_i} v_i \end{aligned} \tag{4.28}$$

The updates to r_{i+1} then become

$$r_{i+1} = r_1 - AV_i L_i^{-1} e_1 = -\frac{\phi_i}{\gamma_i} r_i + \frac{1}{\gamma_i} A v_i$$

and similarly for $\tilde{r}_{i+1}$.

To meet the BICG consistency condition that $\chi(0) = 1$ for the BICG residual vectors, we must set the scalars $\gamma_i = -\phi_i = -(\tilde{v}_i, Av_i)/(\tilde{r}_i, r_i)$. Similar relations hold for the left residual vectors, i.e., $\tilde{\gamma}_i = -\tilde{\phi}_i$. One can therefore obtain BICG by running BIOC with $y_1 = r_1$, $\gamma_i = -\phi_i$, $\tilde{\gamma}_i = -\tilde{\phi}_i$, and with the additional computation of x_{i+1} as in (4.28).

Then, given our CA-BIOC algorithm, it is possible to derive a CA-BICG algorithm in a way analogous to the classical method outlined above. A derivation that goes directly from CA-BIOC to CA-BICG can be found in Section 8.3.2 of the Acta Numerica paper [16]. In this section, we will derive the CA-BICG method from scratch using the classical BICG method as a starting point. This derivation has been adapted from our work in [35].

By induction on lines {5, 6, 7, 10, 11} of classical BICG, it can be shown that, given $k \geq 0, s > 0$, the five vector iterates of classical BICG satisfy

$$\begin{aligned} p_{sk+j}, r_{sk+j} &\in \mathcal{K}_{s+1}(A, p_{sk+1}) + \mathcal{K}_s(A, r_{sk+1}), \\ \tilde{p}_{sk+j}, \tilde{r}_{sk+j} &\in \mathcal{K}_{s+1}(A^H, \tilde{p}_{sk+1}) + \mathcal{K}_s(A^H, \tilde{r}_{sk+1}), \quad \text{and} \\ x_{sk+j} - x_{sk+1} &\in \mathcal{K}_s(A, p_{sk+1}) + \mathcal{K}_{s-1}(A, r_{sk+1}), \end{aligned} \tag{4.29}$$

Algorithm 7 Classical Biconjugate Gradient (BICG)

Input: $n \times n$ matrix A, length-n vector b, and initial approximation x_1 to $Ax = b$
Output: Approximate solution x_{i+1} to $Ax = b$ with updated residual r_{i+1}

$p_1 = r_1 = b - Ax_1$
Choose $\tilde{r}_1$ arbitrarily such that $\delta_1 = (\tilde{r}_1, r_1) \neq 0$, and let $\tilde{p}_1 = \tilde{r}_1$
for $i = 1, 2, \ldots,$ until convergence **do**
    $\alpha_i = \delta_i/(\tilde{p}_i, Ap_i)$
    $x_{i+1} = x_i + \alpha_i p_i$
    $r_{i+1} = r_i - \alpha_i A p_i$
    $\tilde{r}_{i+1} = \tilde{r}_i - \overline{\alpha_i} A^H \tilde{p}_i$
    $\delta_{i+1} = (\tilde{r}_{i+1}, r_{i+1})$
    $\beta_i = \delta_{i+1}/\delta_i$
    $p_{i+1} = r_{i+1} + \beta_i p_i$
    $\tilde{p}_{i+1} = \tilde{r}_{i+1} + \overline{\beta_i} \tilde{p}_i$
end for

for $j \in \{1, \ldots, s+1\}$.

Then to perform iterations $sk+1$ to $sk+s+1$, we define the matrices whose columns span these Krylov subspaces,

$$\begin{aligned}
\mathcal{P}_k &= [\rho_0(A)p_{sk+1}, \rho_1(A)p_{sk+1}, \ldots, \rho_s(A)p_{sk+1}], & \text{span}(\mathcal{P}_k) &= \mathcal{K}_{s+1}(A, p_{sk+1}), \\
\tilde{\mathcal{P}}_k &= [\rho_0(A^H)\tilde{p}_{sk+1}, \rho_1(A^H)\tilde{p}_{sk+1}, \ldots, \rho_s(A^H)\tilde{p}_{sk+1}], & \text{span}(\tilde{\mathcal{P}}_k) &= \mathcal{K}_{s+1}(A^H, \tilde{p}_{sk+1}), \\
\mathcal{R}_k &= [\rho_0(A)r_{sk+1}, \rho_1(A)r_{sk+1}, \ldots, \rho_{s-1}(A)r_{sk+1}], & \text{span}(\mathcal{R}_k) &= \mathcal{K}_s(A, r_{sk+1}), \\
\tilde{\mathcal{R}}_k &= [\rho_0(A^H)\tilde{r}_{sk+1}, \rho_1(A^H)\tilde{r}_{sk+1}, \ldots, \rho_{s-1}(A^H)\tilde{r}_{sk+1}], & \text{span}(\tilde{\mathcal{R}}_k) &= \mathcal{K}_s(A^H, \tilde{r}_{sk+1}),
\end{aligned} \tag{4.30}$$

where $\rho_j(z)$ is a polynomial of degree j satisfying the three-term recurrence (4.1). These matrices can be computed in a communication-avoiding way using the 'PA1' or 'SA1' algorithms described in Section 3.2. Note that the s-step bases in (4.30) span the same subspaces as the s-step bases for CA-BIOC in (4.7) given the appropriate choice of starting vector and scalars described at the beginning of this section.

We can then write the recurrence for generation of columns of the basis matrices in matrix form as

$$\begin{aligned}
A[\underline{\mathcal{P}}_k, \underline{\mathcal{R}}_k] &= [\mathcal{P}_k, \mathcal{R}_k] \begin{bmatrix} \mathcal{B}_k^{(\mathcal{P})} & 0 \\ 0 & \mathcal{B}_k^{(\mathcal{R})} \end{bmatrix} \\
A[\underline{\tilde{\mathcal{P}}}_k, \underline{\tilde{\mathcal{R}}}_k] &= [\tilde{\mathcal{P}}_k, \tilde{\mathcal{R}}_k] \begin{bmatrix} \mathcal{B}_k^{(\tilde{\mathcal{P}})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{R}})} \end{bmatrix},
\end{aligned}$$

where $\underline{\mathcal{P}}_k$, $\underline{\mathcal{R}}_k$, $\underline{\tilde{\mathcal{P}}}_k$ and $\underline{\tilde{\mathcal{R}}}_k$ are the same as $\mathcal{P}_k$, $\mathcal{R}_k$, $\tilde{\mathcal{P}}_k$ and $\tilde{\mathcal{R}}_k$, respectively, but with the last columns set to 0, and the $\mathcal{B}_k$'s are of the form given in (4.2) with $i = s$ for $\mathcal{B}_k^{(\mathcal{P})}$ and $\mathcal{B}_k^{(\tilde{\mathcal{P}})}$ and $i = s-1$ for $\mathcal{B}_k^{(\mathcal{R})}$ and $\mathcal{B}_k^{(\tilde{\mathcal{R}})}$.

To simplify notation, we let $\mathcal{Y}_k = [\mathcal{P}_k, \mathcal{R}_k]$, $\tilde{\mathcal{Y}}_k = [\tilde{\mathcal{P}}_k, \tilde{\mathcal{R}}_k]$, $\underline{\mathcal{Y}}_k = [\underline{\mathcal{P}}_k, \underline{\mathcal{R}}_k]$, $\underline{\tilde{\mathcal{Y}}}_k = [\underline{\tilde{\mathcal{P}}}_k, \underline{\tilde{\mathcal{R}}}_k]$,

$$\mathcal{B}_k^{(\mathcal{Y})} = \begin{bmatrix} \mathcal{B}_k^{(\mathcal{P})} & 0 \\ 0 & \mathcal{B}_k^{(\mathcal{R})} \end{bmatrix}, \quad \text{and} \quad \mathcal{B}_k^{(\tilde{\mathcal{Y}})} = \begin{bmatrix} \mathcal{B}_k^{(\tilde{\mathcal{P}})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{R}})} \end{bmatrix}.$$

This allows us to write

$$A\underline{\mathcal{Y}}_k = \mathcal{Y}_k \mathcal{B}_k^{(\mathcal{Y})} \quad \text{and} \quad A\underline{\tilde{\mathcal{Y}}}_k = \tilde{\mathcal{Y}}_k \mathcal{B}_k^{(\tilde{\mathcal{Y}})}. \tag{4.31}$$

Using the Krylov matrices (4.30) and equations in (4.29), we represent components of the BICG iterates in $\mathbb{C}^n$ by their coordinates in the Krylov bases, that is, subspaces of $\mathbb{C}^n$ of dimension at most $2s+1$. We introduce vectors $\{p'_{k,j}, \tilde{p}'_{k,j}, r'_{k,j}, \tilde{r}'_{k,j}, x'_{k,j}\}$ each of length $2s+1$ to represent the coordinates of $\{p_{sk+j}, \tilde{p}_{sk+j}, r_{sk+j}, \tilde{r}_{sk+j}, x_{sk+j} - x_{sk+1}\}$, respectively, in the generated bases, i.e.,

$$[p_{sk+j}, r_{sk+j}, x_{sk+j} - x_{sk+1}] = \mathcal{Y}_k[p'_{k,j}, r'_{k,j}, x'_{k,j}] \quad \text{and} \quad [\tilde{p}_{sk+j}, \tilde{r}_{sk+j}] = \tilde{\mathcal{Y}}_k[\tilde{p}'_{k,j}, \tilde{r}'_{k,j}], \tag{4.32}$$

for $j \in \{1, \ldots, s+1\}$, where the base cases for these recurrences are given by

$$p'_{k,1} = \tilde{p}'_{k,1} = [1, 0_{1,2s}]^T, \quad r'_{k,1} = \tilde{r}'_{k,1} = [0_{1,s+1}, 1, 0_{1,s-1}]^T, \quad \text{and} \quad x'_{k,1} = 0_{2s+1,1}. \tag{4.33}$$

Next, we represent the multiplications by A and A^H (lines 6 and 7) in the new basis. Note that since we will update for $j \in \{1, \ldots, s\}$, we only perform multiplication of A with p_{sk+j} for these values of j (likewise for A^H and $\tilde{p}_{sk+j}$). Therefore we can write

$$\begin{aligned} Ap_{sk+j} &= A\mathcal{Y}_k p'_{k,j} = A\underline{\mathcal{Y}}_k p'_{k,j} = \mathcal{Y}_k \mathcal{B}_k^{(\mathcal{Y})} p'_{k,j} \quad \text{and} \\ A^H \tilde{p}_{sk+j} &= A^H \tilde{\mathcal{Y}}_k \tilde{p}'_{k,j} = A^H \underline{\tilde{\mathcal{Y}}}_k \tilde{p}'_{k,j} = \tilde{\mathcal{Y}}_k \mathcal{B}_k^{(\tilde{\mathcal{Y}})} \tilde{p}'_{k,j}, \end{aligned} \tag{4.34}$$

for $j \in \{1, \ldots, s\}$. We substitute (4.32) and (4.34) into classical BICG. Each of lines 5, 6, 7, 10, and 11 is now expressed as a linear combination of the columns of the Krylov matrices, and we can match coordinates on the right- and left-hand sides to obtain the recurrences, for $j \in \{1, \ldots, s\}$,

$$x'_{k,j+1} = x'_{k,j} + \alpha_{sk+j} p'_{k,j} \tag{4.35}$$

$$r'_{k,j+1} = r'_{k,j} - \alpha_{sk+j} \mathcal{B}_k^{(\mathcal{Y})} p'_{k,j} \tag{4.36}$$

$$\tilde{r}'_{k,j+1} = \tilde{r}'_{k,j} - \overline{\alpha_{sk+j}} \mathcal{B}_k^{(\tilde{\mathcal{Y}})} \tilde{p}'_{k,j} \tag{4.37}$$

$$p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j} p'_{k,j} \tag{4.38}$$

$$\tilde{p}'_{k,j+1} = \tilde{r}'_{k,j+1} + \overline{\beta_{sk+j}} \tilde{p}'_{k,j}. \tag{4.39}$$

We also need scalar quantities α_{sk+j}, β_{sk+j}, and δ_{sk+j+1}, which are computed from dot products involving the BICG iterates. We define the Gram-like matrix

$$G_k = \tilde{\mathcal{Y}}_k^H \mathcal{Y}_k, \tag{4.40}$$

Algorithm 8 Communication-Avoiding BICG (CA-BICG)

Input: $n \times n$ matrix A, length-n vector b, and initial approximation x_1 to $Ax = b$
Output: Approximate solution x_{sk+s+1} to $Ax = b$ with updated residual r_{sk+s+1}

1: $p_1 = r_1 = b - Ax_1$
2: Choose $\tilde{r}_1$ arbitrarily such that $\delta_1 = (\tilde{r}_1, r_1) \neq 0$, and let $\tilde{p}_1 = \tilde{r}_1$
3: **for** $k = 0, 1, \ldots,$ until convergence **do**
4: Compute $\mathcal{Y}_k = [\mathcal{P}_k, \mathcal{R}_k]$ and $\tilde{\mathcal{Y}}_k = [\tilde{\mathcal{P}}_k, \tilde{\mathcal{R}}_k]$ according to (4.30)
5: Compute G_k according to (4.40)
6: Assemble $\mathcal{B}_k^{(\mathcal{Y})}$ and $\mathcal{B}_k^{(\tilde{\mathcal{Y}})}$ such that (4.31) holds
7: Initialize $\{p'_{k,1}, \tilde{p}'_{k,1}, r'_{k,1}, \tilde{r}'_{k,1}, x'_{k,1}\}$ according to (4.33)
8: **for** $j = 1$ to s **do**
9: $\alpha_{sk+j} = \delta_{sk+j}/(\tilde{p}'_{k,j}, G_k \mathcal{B}_k^{(\mathcal{Y})} p'_{k,j})$
10: $x'_{k,j+1} = x'_{k,j} + \alpha_{sk+j} p'_{k,j}$
11: $r'_{k,j+1} = r'_{k,j} - \alpha_{sk+j} \mathcal{B}_k^{(\mathcal{Y})} p'_{k,j}$
12: $\tilde{r}'_{k,j+1} = \tilde{r}'_{k,j} - \overline{\alpha_{sk+j}} \mathcal{B}_k^{(\tilde{\mathcal{Y}})} \tilde{p}'_{k,j}$
13: $\delta_{sk+j+1} = (\tilde{r}'_{k,j+1}, G_k r'_{k,j+1})$
14: $\beta_{sk+j} = \delta_{sk+j+1}/\delta_{sk+j}$
15: $p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j} p'_{k,j}$
16: $\tilde{p}'_{k,j+1} = \tilde{r}'_{k,j+1} + \overline{\beta_{sk+j}} \tilde{p}'_{k,j}$
17: **end for**
18: Recover iterates $\{p_{sk+s+1}, \tilde{p}_{sk+s+1}, r_{sk+s+1}, \tilde{r}_{sk+s+1}, x_{sk+s+1}\}$ according to (4.32)
19: **end for**

which can be computed with one Allreduce operation as described in Section 3.1. The above equation, along with (4.34), allows us to write inner products in lines 4 and 8 as

$$(\tilde{r}_{sk+j+1}, r_{sk+j+1}) = (\tilde{\mathcal{Y}}_k \tilde{r}'_{k,j+1}, \mathcal{Y}_k r'_{k,j+1}) = (\tilde{r}'_{k,j+1}, G_k r'_{k,j+1}) \quad \text{and} \tag{4.41}$$

$$(\tilde{p}_{sk+j}, A p_{sk+j}) = (\tilde{\mathcal{Y}}_k \tilde{p}'_{k,j}, \mathcal{Y}_k \mathcal{B}_k^{(\mathcal{Y})} p'_{k,j}) = (\tilde{p}'_{k,j}, G_k \mathcal{B}_k^{(\mathcal{Y})} p'_{k,j}), \tag{4.42}$$

for $j \in \{1, \ldots, s\}$.

Now we assemble the CA-BICG method from (4.35)-(4.39) and (4.40)-(4.42), shown in Algorithm 8.

The communication costs of CA-BICG versus classical BICG are the same as discussed for the CA-BIOC versus classical BIOC algorithms in the previous section. In parallel, the only communication in CA-BICG occurs in the outer loop, in lines 4 and 5; the inner loop lines operate on operands of size $O(ns/p + s^2)$, which we assume fit locally on each of the p processors. Therefore, CA-BICG can take s steps per $O(1)$ rounds of messages whereas classical BICG can only take 1 step per $O(1)$ rounds of messages; this is an $\Theta(s)$-fold decrease in latency for well-partitioned A.

We can use the 'PA1' algorithm described in Section 3.2.2 to compute the bases in line 4. Using PA1, each processor requires additional source vector entries (the ghost zones) and

must perform redundant computation. If A is well partitioned, the additional bandwidth and computation are both lower order terms. We also note that in CA-BICG, each processor must send $\Theta(s^2 \log_2 p)$ words to perform the matrix multiplication in line 5, while the equivalent dot products in s steps of classical BICG only require $\Theta(s \log_2 p)$ words moved per processor (see Section 3.1). Construction of G_k brings the computational cost of the dense operations to $O(s^2 n/p)$, a factor of $O(s)$ larger than the classical algorithm. Under the reasonable assumption that the number of nonzeros of A per processor is greater than $O(sn/p)$, the SpMV computations dominate dense operations in the classical algorithm, so the additional cost of computing G_k is not a limiting factor.

In serial, CA-BICG moves data on lines 4, 5, and 18. As in CA-BIOC, we are most concerned with the data movement in reading A (line 4), since it is common that A requires far more storage than the vectors on which it operates. If A is well partitioned, then using the algorithm SA1 described in Section 3.2.3, line 4 reads A $2 + o(1)$ times, whereas s iterations of classical BICG read A $2s$ times. This is an s-fold savings in latency and bandwidth for reading A. All three lines (4, 5, and 18) involve reading the vector iterates of length n, asymptotically the same communication cost as the classical algorithm. Similar to the parallel case, we perform a factor of $O(s)$ more computation in the dense operations, which is a lower-order term if the number of nonzeros in A is greater than $O(sn)$.

4.2.1 Numerical Experiments

We present convergence results for two matrices which exemplify the dependence of typical numerical behavior on the chosen basis and s value. In all tests, the right-hand side b was constructed such that the true solution x is the n-vector with components $x_i = 1/\sqrt{n}$.

cdde Our first test problem comes from the constant-coefficient convection diffusion equation

$$\begin{aligned} -\Delta u + 2p_1 u_x + 2p_2 u_y - p_3 u_y &= f \quad \text{in} \quad [0,1]^2 \\ u &= g \quad \text{on} \quad \partial[0,1]^2 \end{aligned}$$

This problem is widely used for testing and analyzing numerical solvers [13]. We discretized the PDE using centered finite difference operators on a 512×512 grid with $(p_1, p_2, p_3) = (25, 600, 250)$, resulting in a nonsymmetric matrix with $n = 262K$, $\text{nnz}(A) = 1.3M$, and condition number ~ 5.5. To select the Leja points, we used the convex hull of the known spectrum, given in [13]. Fig. 4.1 shows the selected Leja points and resulting basis condition number $\text{cond}(\mathcal{Y}_k) = \|\mathcal{Y}_k^+\|_2 \cdot \|\mathcal{Y}_k\|_2$ for the monomial, Newton, and Chebyshev bases, for basis lengths up to $s = 32$. Fig. 4.2 shows convergence results for CA-BICG for the different bases, for $s \in \{4, 8, 16\}$.

xenon1 The xenon1 matrix is an nonsymmetric matrix from the University of Florida Sparse Matrix Collection [57]. This matrix is used in analyzing elastic properties of crystalline

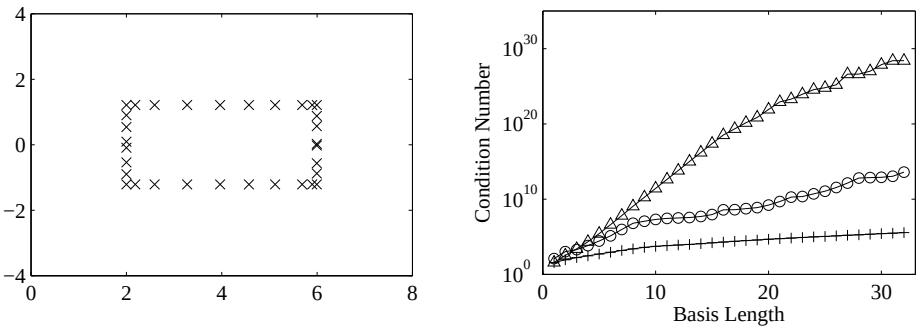

Figure 4.1: Basis properties for cdde. Left: computed Leja points ($\times$) of cdde, plotted in the complex plane. Right: basis condition number growth rate. The x-axis denotes basis length s and the y-axis denotes the basis condition number for monomial ($\triangle$), Newton ($\circ$), and Chebyshev ($+$) bases. Leja points on the left were used to generate the bases shown in the right plot.

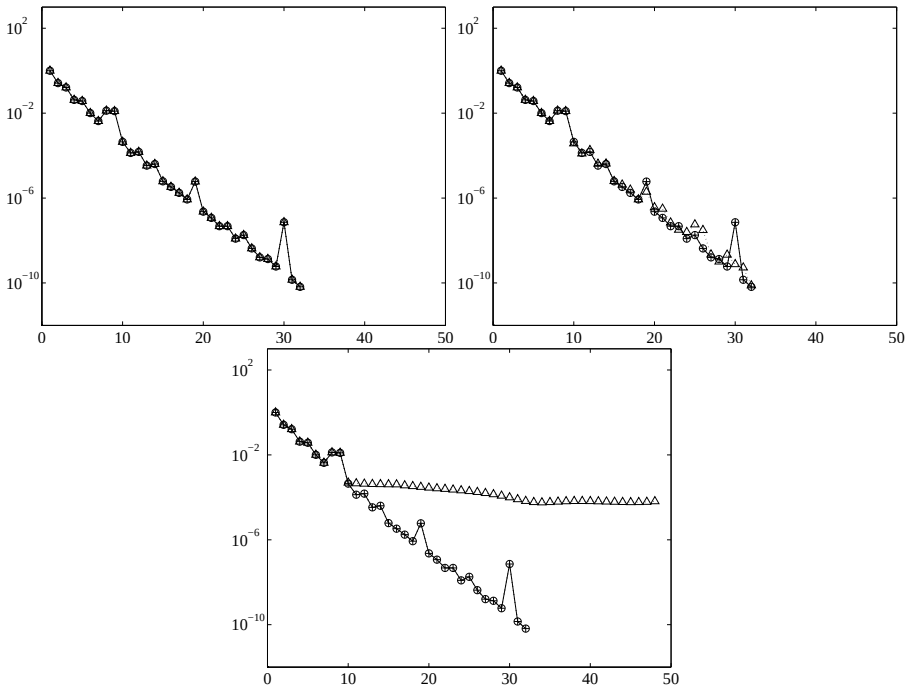

Figure 4.2: Convergence for cdde matrix for (CA-)BICG with various s values: $s = 4$ (top left), $s = 8$ (top right), $s = 16$ (bottom). The x-axis denotes iteration number, and the y-axis denotes the 2-norm of the (normalized) true residual (i.e., $\|b - Ax_m\|_2 / \|b\|_2$). Each plot shows convergence for the classical algorithm ($-$), as well as for CA-BICG using the monomial ($\triangle$), Newton ($\circ$), and Chebyshev ($+$) bases.

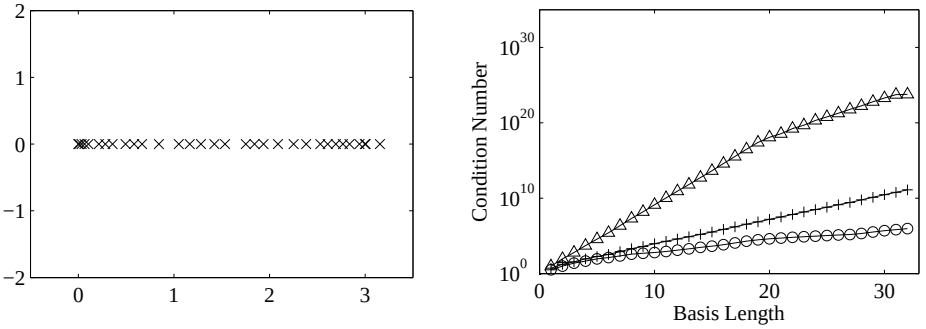

Figure 4.3: Basis properties for xenon1. Left: computed Leja points ($\times$) of xenon1, plotted in the complex plane. Right: basis condition number growth rate. The x-axis denotes basis length s and the y-axis denotes the basis condition number for monomial ($\triangle$), Newton ($\circ$), and Chebyshev ($+$) bases. Leja points on left were used to generate the bases shown in the right plot.

compounds [57]. Here, $n = 48.6K$ and $\text{nnz}(A) = 1.2M$. This test case is less well-conditioned than the first, with condition number $\sim 1.1 \cdot 10^5$ after performing matrix equilibration as a preprocessing step as described in Section 3.2.5.3 and [102]. The xenon1 matrix has all real eigenvalues. Therefore, Leja points (and thus basis parameters) were selected based only on estimates of the maximum and minimum eigenvalues obtained from ARPACK (MATLAB `eigs`). Fig. 4.3 shows the generated Leja points and resulting basis condition numbers for the monomial, Newton, and Chebyshev bases, for basis lengths up to $s = 32$. Fig. 4.4 shows convergence results for CA-BICG for the different bases, for $s \in \{4, 8, 16\}$.

For CA-BICG with the monomial basis, we generally see the convergence rate decrease (relative to the classical method) as s grows larger. This is due to roundoff error in computation of the basis vectors in the matrix powers kernel, which results in a larger perturbation to the finite precision Lanczos recurrence that determines the rate of convergence (see, e.g., [176]). At some point, when s becomes large, CA-BICG with the monomial basis will fail to converge due to (near) numerical rank deficiencies in the generated Krylov bases. For both of our test matrices, CA-BICG with the monomial basis fails to converge to the desired tolerance when $s = 16$.

CA-BICG with the Newton and Chebyshev bases, however, are able to maintain convergence closer to that of the classical method, even for s as large as 16. This is especially evident if we have a well-conditioned matrix, as in Figure 4.2 (cdde). For this matrix, CA-BICG with both the Newton and Chebyshev bases converges in the same number of iterations as the classical method, for all tested s values.

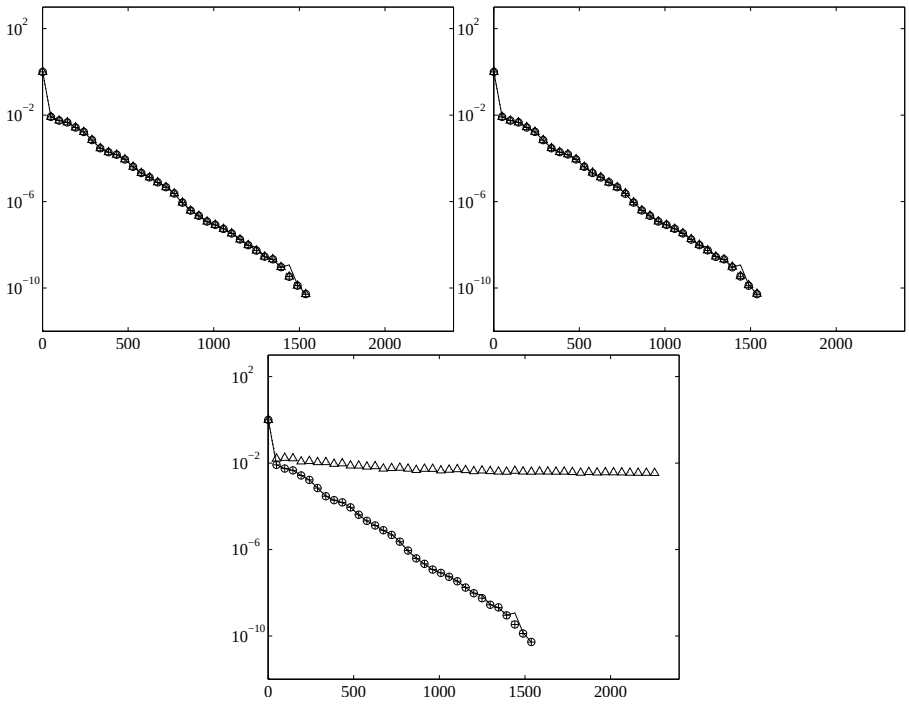

Figure 4.4: Convergence for xenon1 matrix for (CA-)BICG with various s values: $s = 4$ (top left), $s = 8$ (top right), $s = 16$ (bottom). The x-axis denotes iteration number, and the y-axis denotes the 2-norm of the (normalized) true residual (i.e., $\|b - Ax_m\|_2 / \|b\|_2$). Each plot shows convergence for the classical algorithm ($-$), as well as for CA-BICG using the monomial ($\triangle$), Newton ($\circ$), and Chebyshev ($+$) bases.

4.3 Conjugate Gradient Squared

The multiplication by A^H and corresponding left Krylov vectors (residuals $\tilde{r}$) only contribute to the BICG solution of $Ax = b$ through the scalar coefficients α_i and β_i. BICG does not exploit the reduction in magnitude of the left residuals unless we solve a dual system. In an effort to get faster convergence, or when A^H might not be available, for instance when the linear system A is the Jacobian in a (matrix-free) Newton-Krylov method, one might demand a transpose-free method such as conjugate gradient squared (CGS) [166], a quasi-orthogonal residual (QOR) method derived from BICG (another QOR method), that avoids the multiplication by A^H.

CGS respects the mutual biorthogonality of the two Krylov spaces (and so the Lanczos coefficients are the same as BICG, in exact arithmetic), however, the polynomials representing the CGS residuals are the squares of those in BICG. In BICG, the residual computed in iteration i can be expressed as

$$r_{i+1} = \chi_i(A) r_1,$$

where χ_i is a polynomial of degree i with $\chi_i(0) = 1$. Similarly, the direction vector p_{i+1} can be written

$$p_{i+1} = \pi_i(A)r_1,$$

where π_i is a polynomial of degree i. Since $\tilde{r}_{i+1}$ and $\tilde{p}_{i+1}$ are computed by the same recurrences,

$$\tilde{r}_{i+1} = \chi_i(A^T)\tilde{r} \quad \text{and} \quad \tilde{p}_{i+1} = \pi_i(A^T)\tilde{r},$$

and then the scalars can be computed, e.g.,

$$\alpha_i = \frac{(\chi_i(A^T)\tilde{r}, \chi(A)r_1)}{(\pi_i(A^T)\tilde{r}, A\pi(A)r_1)} = \frac{(\tilde{r}, \chi_i^2(A)r_1)}{(\tilde{r}, \pi_i^2(A)r_1)}.$$

These squared polynomials can then be used to avoid keeping track of the left Lanczos vectors. However, CGS actually interprets the squared BICG residuals as the true residuals, and updates the solution accordingly. This heuristic decision was motivated by the observation that the BICG polynomials typically reduce the norm of the residual, and so one would hope that applying the BICG polynomial again (to an already-reduced residual) might reduce it further. But, because it squares the polynomials, CGS might have more irregular convergence than BICG. As a side effect, larger intermediate quantities in CGS could worsen local roundoff, leading to a (faster) deviation between the updated residual and the true residual.

We will use the variant of classical CGS given in the textbook [153], which we reproduce in Algorithm 9. Like classical BICG, classical CGS has two communication-bound kernels: SpMV and inner products. Again, we replace SpMV with the matrix powers kernel, and inner products with a Gram-like matrix and vector. In the case of CGS we need $6s$ total Krylov basis vectors to complete s iterations. First, we convert classical CGS (Algorithm 9) to an s-step method. Similarly to CGS, we can determine the dependencies for steps $sk+1$ through $sk+s+1$ by considering dependencies in Algorithm 9. By induction, we can show that, given $k \geq 0, s > 0$, the vector iterates of classical CGS satisfy

$$p_{sk+j}, u_{sk+j}, r_{sk+j}, x_{sk+j} - x_{sk+1} \in \mathcal{K}_{2s+1}(A, p_{sk+1}) + \mathcal{K}_{2s}(A, u_{sk+1}) + \mathcal{K}_{2s-1}(A, r_{sk+1}), \quad (4.43)$$

for $j \in \{1, \ldots, s+1\}$.

Then to find iterates $sk+2$ to $sk+s+1$, we use the Krylov matrices $\mathcal{P}_k$, $\mathcal{U}_k$, and $\mathcal{R}_k$, where

$$\begin{aligned}
\mathcal{P}_k &= [\rho_0(A)p_{sk+1}, \rho_1(A)p_{sk+1}, \ldots, \rho_{2s}(A)p_{sk+1}], & \text{span}(\mathcal{P}_k) &= \mathcal{K}_{2s+1}(A, p_{sk+1}), \\
\mathcal{U}_k &= [\rho_0(A)u_{sk+1}, \rho_1(A)u_{sk+1}, \ldots, \rho_{2s-1}(A)u_{sk+1}], & \text{span}(\mathcal{U}_k) &= \mathcal{K}_{2s}(A, u_{sk+1}), \\
\mathcal{R}_k &= [\rho_0(A)r_{sk+1}, \rho_1(A)r_{sk+1}, \ldots, \rho_{2s-2}(A)r_{sk+1}], & \text{span}(\mathcal{R}_k) &= \mathcal{K}_{2s-1}(A, r_{sk+1}),
\end{aligned} \quad (4.44)$$

where $\rho_j(z)$ is a polynomial of degree j satisfying the three-term recurrence (4.1). Assuming A is well partitioned, these matrices can be computed in a communication-avoiding way using the 'PA1' or 'SA1' algorithms described in Section 3.2.

Algorithm 9 Classical Conjugate Gradient Squared (CGS)

Input: $n \times n$ matrix A, length-n vector b, and initial approximation x_1 to $Ax = b$
Output: Approximate solution x_{i+1} to $Ax = b$ with updated residual r_{i+1}

$p_1 = u_1 = r_1 = b - Ax_1$
Choose $\tilde{r}$ arbitrarily such that $\delta_1 = (\tilde{r}, r_1) \neq 0$
for $i = 1, 2, \ldots,$ until convergence **do**
    $\alpha_i = \delta_i / (\tilde{r}, Ap_i)$
    $q_i = u_i - \alpha_i A p_i$
    $x_{i+1} = x_i + \alpha_i (u_i + q_i)$
    $r_{i+1} = r_i - \alpha_i A (u_i + q_i)$
    $\delta_{i+1} = (\tilde{r}, r_{i+1})$
    $\beta_i = \delta_{i+1} / \delta_i$
    $u_{i+1} = r_{i+1} + \beta_i q_i$
    $p_{i+1} = u_{i+1} + \beta_i (q_i + \beta_i p_i)$
end for

We can then write the recurrence for generation of columns of the basis matrices in matrix form as

$$A[\underline{\mathcal{P}}_k, \underline{\mathcal{U}}_k, \underline{\mathcal{R}}_k] = [\mathcal{P}_k, \mathcal{U}_k, \mathcal{R}_k] \begin{bmatrix} \mathcal{B}_k^{(\mathcal{P})} & 0_{2s+1,2s} & 0_{2s+1,2s-1} \\ 0_{2s,2s+1} & \mathcal{B}_k^{(\mathcal{U})} & 0_{2s,2s-1} \\ 0_{2s-1,2s+1} & 0_{2s-1,2s} & \mathcal{B}_k^{(\mathcal{R})} \end{bmatrix},$$

where $\underline{\mathcal{P}}_k$, $\underline{\mathcal{U}}_k$, and $\underline{\mathcal{R}}_k$ are the same as $\mathcal{P}_k$, $\mathcal{U}_k$, and $\mathcal{R}_k$, respectively, but with the last columns set to 0, and the $\mathcal{B}_k$'s are of the form given in (4.2) with $i = 2s$ for $\mathcal{B}_k^{(\mathcal{P})}$, $i = 2s-1$ for $\mathcal{B}_k^{(\mathcal{U})}$, and $i = 2s-2$ for $\mathcal{B}_k^{(\mathcal{R})}$.

To simplify notation, we let $\mathcal{Y}_k = [\mathcal{P}_k, \mathcal{U}_k, \mathcal{R}_k]$, $\underline{\mathcal{Y}}_k = [\underline{\mathcal{P}}_k, \underline{\mathcal{U}}_k, \underline{\mathcal{R}}_k]$, and

$$\mathcal{B}_k = \begin{bmatrix} \mathcal{B}_k^{(\mathcal{P})} & 0_{2s+1,2s} & 0_{2s+1,2s-1} \\ 0_{2s,2s+1} & \mathcal{B}_k^{(\mathcal{U})} & 0_{2s,2s-1} \\ 0_{2s-1,2s+1} & 0_{2s-1,2s} & \mathcal{B}_k^{(\mathcal{R})} \end{bmatrix}.$$

This allows us to write

$$A\underline{\mathcal{Y}}_k = \mathcal{Y}_k \mathcal{B}_k. \tag{4.45}$$

Using the Krylov matrices (4.44) and equations in (4.43), we can then represent the n components of the CGS iterates by their $6s$ coordinates in the Krylov bases stored in $\mathcal{Y}_k$. We introduce coordinate vectors $\{p'_{k,j}, u'_{k,j}, r'_{k,j}, x'_{k,j}\}$ to represent the length-n vectors $\{p_{sk+j}, u_{sk+j}, r_{sk+j}, x_{sk+j} - x_{sk+1}\}$, such that

$$[p_{sk+j}, u_{sk+j}, r_{sk+j}, x_{sk+j} - x_{sk+1}] = \mathcal{Y}_k [p'_{k,j}, u'_{k,j}, r'_{k,j}, x'_{k,j}], \tag{4.46}$$

for $j \in \{1, \ldots, s+1\}$, where the base cases for these recurrences are given by

$$\begin{aligned} p'_{k,1} &= [1, 0_{1,6s-1}]^T, \\ u'_{k,1} &= [0_{1,2s+1}, 1, 0_{1,4s-2}]^T, \\ r'_{k,1} &= [0_{1,4s+1}, 1, 0_{1,2s-2}]^T, \quad \text{and} \\ x'_{k,1} &= 0_{6s,1}. \end{aligned} \tag{4.47}$$

Next, we represent the multiplications of p_{sk+j}, u_{sk+j}, and auxiliary vector q_{sk+j} by A in the new basis. Note that since we will update for $j \in \{1, \ldots, s\}$, we only perform multiplication of A with p_{sk+j} and u_{sk+j} for these values of j. Note also that $q_{sk+j} \in \mathcal{Y}_k$ for $j \in \{1, \ldots, s\}$. We can therefore write

$$Ap_{sk+j} = A\mathcal{Y}_k p'_{k,j} = A\underline{\mathcal{Y}}_k p'_{k,j} = \mathcal{Y}_k \mathcal{B}_k p'_{k,j}, \tag{4.48}$$

$$\begin{aligned} Au_{sk+j} &= A\mathcal{Y}_k u'_{k,j} = A\underline{\mathcal{Y}}_k u'_{k,j} = \mathcal{Y}_k \mathcal{B}_k u'_{k,j} \quad \text{and,} \\ Ad_{sk+j} &= A\mathcal{Y}_k d'_{k,j} = A\underline{\mathcal{Y}}_k d'_{k,j} = \mathcal{Y}_k \mathcal{B}_k d'_{k,j}, \end{aligned} \tag{4.49}$$

for $j \in \{1, \ldots, s\}$.

We substitute (4.46) and (4.48) into classical CGS. Each vector update is now expressed as a linear combination of the columns of the Krylov matrices, and we can match coordinates on the right and left hand sides to obtain the recurrences, for $j \in \{1, \ldots, s\}$,

$$q'_{k,j} = u'_{k,j} - \alpha_{sk+j} \mathcal{B}_k p'_{k,j} \tag{4.50}$$

$$x'_{k,j+1} = x'_{k,j} + \alpha_{sk+j}(u'_{k,j} + q'_{k,j}) \tag{4.51}$$

$$r'_{k,j+1} = r'_{k,j} - \alpha_{sk+j} \mathcal{B}_k (u'_{k,j} + q'_{k,j}) \tag{4.52}$$

$$u'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j} q'_{k,j} \tag{4.53}$$

$$p'_{k,j+1} = u'_{k,j+1} + \beta_{sk+j}(q'_{k,j} + \beta_{sk+j} p'_{k,j}) \tag{4.54}$$

We also need scalar quantities α_{sk+j}, δ_{sk+j+1}, and β_{sk+j}, which are computed from dot products involving the CGS iterates. We represent these dot products in the new basis, using the Gram matrix G_k and vector g as

$$[G_k, g] = \mathcal{Y}_k^H [\mathcal{Y}_k, \tilde{r}], \tag{4.55}$$

which can be computed with one Allreduce as described in Section 3.1. The formulas for the required inner products are then easily derived by using (4.45) and (4.48). We have

$$\begin{aligned} (\tilde{r}, Ap_{sk+j}) &= (\tilde{r}, \mathcal{Y}_k \mathcal{B}_k p'_{k,j}) = (g, \mathcal{B}_k p'_{k,j}) \quad \text{and} \\ (\tilde{r}, r_{sk+j+1} &= (\tilde{r}, \mathcal{Y}_k r'_{k,j+1}) = (g, r'_{k,j+1}). \end{aligned}$$

We now assemble the CA-CGS method shown in Algorithm 10.

Analogous to CA-BICG, the only communication in parallel CA-CGS occurs in lines 4 and 5. Using PA1 and block computation of inner products (see Chapter 3), CA-CGS can

Algorithm 10 Communication-Avoiding CGS (CA-CGS)

Input: $n \times n$ matrix A, length-n vector b, and initial approximation x_1 to $Ax = b$
Output: Approximate solution x_{sk+s+1} to $Ax = b$ with updated residual r_{sk+s+1}

1: $p_1 = u_1 = r_1 = b - Ax_1$
2: Choose $\tilde{r}$ arbitrarily such that $\delta_1 = (\tilde{r}, r_1) \neq 0$
3: **for** $k = 0, 1, \ldots,$ until convergence **do**
4: Compute $\mathcal{P}_k$, $\mathcal{U}_k$, and $\mathcal{R}_k$ according to (4.44)
5: Compute G_k and g according to (4.55)
6: Assemble $\mathcal{B}_k$ such that (4.45) holds
7: Initialize $\{p'_{k,1}, u'_{k,1}, r'_{k,1}, x'_{k,1}\}$ according to (4.47)
8: **for** $j = 1$ to s **do**
9: $\alpha_{sk+j} = \delta_{sk+j}/(g, \mathcal{B}_k p'_{k,j})$
10: $q'_{k,j} = u'_{k,j} - \alpha_{sk+j}\mathcal{B}_k p'_{k,j}$
11: $x'_{k,j+1} = x'_{k,j} + \alpha_{sk+j}(u'_{k,j} + q'_{k,j})$
12: $r'_{k,j+1} = r'_{k,j} - \alpha_{sk+j}\mathcal{B}_k(u'_{k,j} + q'_{k,j})$
13: $\delta_{sk+j+1} = (g, r'_{k,j+1})$
14: $\beta_{sk+j} = \delta_{sk+j+1}/\delta_{sk+j}$
15: $u'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j}q'_{k,j}$
16: $p'_{k,j+1} = u'_{k,j+1} + \beta_{sk+j}(q'_{k,j} + \beta_{sk+j}p'_{k,j})$
17: **end for**
18: Recover iterates $\{p_{sk+s+1}, u_{sk+s+1}, r_{sk+s+1}, x_{sk+s+1}\}$ according to (4.46)
19: **end for**

reduce parallel latency by a factor of $\Theta(s)$, albeit at the cost of increasing bandwidth by a small factor. We refer to the above discussion for CA-BICG, except noting that CA-CGS does not require multiplication by A^H, and instead requires $2s$ multiplications by A for each of three vectors per s iterations; this could be accomplished by calling PA1 (see Section 3.2.2) with three right-hand sides.

In serial, CA-CGS moves data on lines 4, 5, and 18. Again, this is analogous to CA-BICG; this formulation enables use of SA1 (see Section 3.2.3), which reduces serial latency and bandwidth by a factor of $\Theta(s)$.

Although there are instances where CGS outperforms other nonsymmetric solvers (see the experiments in [131]), it is often the case that its convergence behavior is much more erratic than methods like BICGSTAB, and is thus less commonly used in practice.

4.4 Biconjugate Gradient Stabilized

As mentioned, the CGS method uses the squares of the BICG residual polynomials. This can lead to irregular convergence, which causes accumulation of roundoff error. Lanczos-type product methods (LTPMs) (see [94]) attempt to improve on the convergence behavior of CGS

by using a different polynomial recurrence for the left basis; that is, instead of computing residuals of the form $r_{i+1} = \chi_i^2(A)r_1$, we allow for residuals of the form

$$r_{i+1} = \psi_i(A)\chi_i(A)r_1,$$

where ψ_i is a polynomial of degree i. Many LTPMs, including the biconjugate gradient stabilized method (BICGSTAB) of van der Vorst [179] and its variants, choose the polynomials ψ_i recursively at each step with the goal of smoothing or stabilizing convergence. recurrence is preferable for performance reasons.

In the case of BICGSTAB, the polynomials ψ_i are constructed via a two-term recurrence, which amounts to extending the Krylov space by one dimension (a new basis vector), and taking a steepest descent step in that direction. This is a local one-dimensional minimization, which should result in a smoother convergence curve and avoid possible overflow conditions in CGS. However, an issue with BICGSTAB is that if the input data is all real, the stabilizing polynomial will have only real zeros. Such a polynomial will not reduce error components in the direction of eigenvectors corresponding to eigenvalues with large imaginary components (relative to their real components). Matrices with such a spectrum are also more susceptible to a *minimization breakdown* in BICGSTAB, a new breakdown condition.

These two drawbacks to BICGSTAB are addressed in the literature by many newer LTPMs like BICGSTAB2 [93], which uses two-dimensional residual smoothing, and the BICGSTAB(L) [160] method, which extends the ideas of BICGSTAB2 to use L-dimensional smoothing. We note that BICGSTAB(L) is a promising direction for future work, especially once combined with the communication-avoiding tall-skinny QR kernel (TSQR), as described in [102]. We conjecture that our communication-avoiding approach generalizes, and that the flexibility to choose a polynomial basis in computing the s-step bases could accelerate the computation of the ψ_i polynomials, when the recurrence coefficients are known in advance.

We present the version of BICGSTAB from [153] in Algorithm 11, from which we will derive CA-BICGSTAB. The derivation presented here has been adapted from the work in [35]. Like classical BICG, classical BICGSTAB has two communication-bound kernels: SpMV and inner products. Again, we replace SpMV with the matrix powers kernel, and inner products with a Gram-like matrix and vector; see Chapter 3. Because BICGSTAB expands the underlying Krylov space by two dimensions each iteration, we need twice as many Krylov basis vectors to complete s iterations. First, we convert classical BICGSTAB (Algorithm 11) to an s-step method. Similarly to BICG, we can determine the dependencies for steps $sk+1$ through $sk+s+1$ by inspection of Algorithm 11. By induction on lines {8, 9, and 12} of classical BICGSTAB, we can show that, given $k \geq 0, s > 0$, the vector iterates of classical BICGSTAB satisfy

$$p_{sk+j}, r_{sk+j}, x_{sk+j} - x_{sk+1} \in \mathcal{K}_{2s+1}(A, p_{sk+1}) + \mathcal{K}_{2s}(A, r_{sk+1}), \tag{4.56}$$

for $j \in \{1, \ldots, s+1\}$.

Then to find iterates $sk+2$ to $sk+s+1$, we use the Krylov matrices $\mathcal{P}_k$ and $\mathcal{R}_k$, where

$$\begin{aligned} \mathcal{P}_k &= [\rho_0(A)p_{sk+1}, \rho_1(A)p_{sk+1}, \ldots, \rho_{2s}(A)p_{sk+1}], & \text{span}(\mathcal{P}_k) &= \mathcal{K}_{2s+1}(A, p_{sk+1}), \\ \mathcal{R}_k &= [\rho_0(A)r_{sk+1}, \rho_1(A)r_{sk+1}, \ldots, \rho_{2s-1}(A)r_{sk+1}], & \text{span}(\mathcal{R}_k) &= \mathcal{K}_{2s}(A, r_{sk+1}), \end{aligned} \tag{4.57}$$

Algorithm 11 Classical Biconjugate Gradient Stabilized (BICGSTAB)

Input: $n \times n$ matrix A, length-n vector b, and initial approximation x_1 to $Ax = b$
Output: Approximate solution x_{i+1} to $Ax = b$ with updated residual r_{i+1}

$p_1 = r_1 = b - Ax_1$
Choose $\tilde{r}$ arbitrarily such that $\delta_1 = (\tilde{r}, r_1) \neq 0$
for $i = 1, 2, \ldots,$ until convergence **do**
    $\alpha_i = \delta_i / (\tilde{r}, Ap_i)$
    $x_{i+1} = x_i + \alpha_i p_i$
    $d_i = r_i - \alpha_i A p_i$
    $\omega_i = \frac{(d_i, Ad_i)}{(Ad_i, Ad_i)}$
    $x_{i+1} = x_{i+1} + \omega_i d_i$
    $r_{i+1} = d_i - \omega_i A d_i$
    $\delta_{i+1} = (\tilde{r}, r_{i+1})$
    $\beta_i = (\delta_{i+1}/\delta_i) \cdot (\alpha_i/\omega_i)$
    $p_{i+1} = r_{i+1} + \beta_i p_i - \beta_i \omega_i A p_i$
end for

where $\rho_j(z)$ is a polynomial of degree j satisfying the three-term recurrence (4.1). Assuming A is well partitioned, these matrices can be computed in a communication-avoiding way using the 'PA1' or 'SA1' algorithms described in Section 3.2 with two right-hand sides.

We can then write the recurrence for generation of columns of the basis matrices in matrix form as

$$A[\underline{\mathcal{P}}_k, \underline{\mathcal{R}}_k] = [\mathcal{P}_k, \mathcal{R}_k] \begin{bmatrix} \mathcal{B}_k^{(\mathcal{P})} & 0 \\ 0 & \mathcal{B}_k^{(\mathcal{R})} \end{bmatrix},$$

where $\underline{\mathcal{P}}_k$ and $\underline{\mathcal{R}}_k$ are the same as $\mathcal{P}_k$ and $\mathcal{R}_k$, respectively, but with the last columns set to 0, and the $\mathcal{B}_k$'s are of the form given in (4.2) with $i = 2s$ for $\mathcal{B}_k^{(\mathcal{P})}$ and $i = 2s - 1$ for $\mathcal{B}_k^{(\mathcal{R})}$.

To simplify notation, we let $\mathcal{Y}_k = [\mathcal{P}_k, \mathcal{R}_k]$, $\underline{\mathcal{Y}}_k = [\underline{\mathcal{P}}_k, \underline{\mathcal{R}}_k]$, and

$$\mathcal{B}_k = \begin{bmatrix} \mathcal{B}_k^{(\mathcal{P})} & 0 \\ 0 & \mathcal{B}_k^{(\mathcal{R})} \end{bmatrix}.$$

This allows us to write

$$A\underline{\mathcal{Y}}_k = \mathcal{Y}_k \mathcal{B}_k. \tag{4.58}$$

Using the Krylov matrices (4.57) and equations in (4.56), we can then represent the n components of the BICGSTAB iterates by their $4s + 1$ coordinates in the Krylov bases defined by $\mathcal{P}_k$ and $\mathcal{R}_k$. We introduce coordinate vectors $\{p'_{k,j}, r'_{k,j}, x'_{k,j}\}$ to represent vectors $\{p_{sk+j}, r_{sk+j}, x_{sk+j} - x_{sk+1}\}$, such that

$$[p_{sk+j}, r_{sk+j}, x_{sk+j} - x_{sk}] = \mathcal{Y}_k [p'_{k,j}, r'_{k,j}, x'_{k,j}], \tag{4.59}$$

for $j \in \{1, \ldots, s+1\}$, where the base cases for these recurrences are given by

$$p'_{k,1} = [1, 0_{1,4s}]^T, \quad r'_{k,1} = [0_{1,2s+1}, 1, 0_{1,2s-1}]^T, \quad \text{and} \quad x'_{k,1} = 0_{4s+1,1}. \tag{4.60}$$

Next, we represent the multiplications of p_{sk+j} and auxiliary vector d_{sk+j} by A in the new basis. Note that since we will update for $j \in \{1, \ldots, s\}$, we only perform multiplication of A with p_{sk+j} and d_{sk+j} for these values of j. Note also that $d_{sk+j} \in \mathcal{Y}_k$ for $j \in \{1, \ldots, s\}$. We can therefore write

$$\begin{aligned} Ap_{sk+j} &= A\mathcal{Y}_k p'_{k,j} = A\underline{\mathcal{Y}}_k p'_{k,j} = \mathcal{Y}_k \mathcal{B}_k p'_{k,j} \quad \text{and} \\ Ad_{sk+j} &= A\mathcal{Y}_k d'_{k,j} = A\underline{\mathcal{Y}}_k d'_{k,j} = \mathcal{Y}_k \mathcal{B}_k d'_{k,j}, \end{aligned} \tag{4.61}$$

for $j \in \{1, \ldots, s\}$.

We substitute (4.59) and (4.61) into classical BICGSTAB. Each of lines 5, 6, 8, 9, and 12 is now expressed as a linear combination of the columns of the Krylov matrices, and we can match coordinates on the right and left hand sides to obtain the recurrences, for $j \in \{1, \ldots, s\}$,

$$x'_{k,j+1} = x'_{k,j} + \alpha_{sk+j} p'_{k,j}, \tag{4.62}$$

$$d'_{k,j} = r'_{k,j} - \alpha_{sk+j} \mathcal{B}_k p'_{k,j}, \tag{4.63}$$

$$x'_{k,j+1} = x'_{k,j+1} + \omega_{sk+j} d'_{k,j}, \tag{4.64}$$

$$r'_{k,j+1} = d'_{k,j} - \omega_{sk+j} \mathcal{B}_k d'_{k,j}, \tag{4.65}$$

$$p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j} p'_{k,j} - \beta_{sk+j} \omega_{sk+j} \mathcal{B}_k p'_{k,j}. \tag{4.66}$$

$$\tag{4.67}$$

We also need scalar quantities α_{sk+j}, ω_{sk+j}, β_{sk+j}, and δ_{sk+j+1}, which are computed from dot products involving the BICGSTAB iterates. We represent these dot products in the new basis, using the Gram matrix G_k and vector g as

$$[G_k, g] = \mathcal{Y}_k^H [\mathcal{Y}_k, \tilde{r}], \tag{4.68}$$

which can be computed with one Allreduce operation; see Section 3.1. The formulas for the scalar quantities can be easily derived by using (4.58) and (4.61), similar to the derivations for CA-BICG and CA-CGS. Now we assemble the CA-BICGSTAB method shown in Algorithm 12.

Analogous to the other methods in this chapter, the only communication in parallel CA-BICGSTAB occurs in lines 4 and 5. Using PA1 in Section 3.2.2, CA-BICGSTAB can thus reduce parallel latency by a factor of $\Theta(s)$, albeit at the cost of increasing bandwidth by a small factor. We refer to the above discussion for CA-BICG, except noting that CA-BICGSTAB does not require multiplication by A^H, and instead requires $2s$ multiplications by A for each of the two vectors per s iterations; this could be accomplished by calling PA1 (see Section 3.2.2) with two right-hand sides.

In serial, CA-BICGSTAB moves data on lines 4, 5, and 19. Again, this is analogous to CA-BICG; use of SA1 in Section 3.1 reduces the serial latency and bandwidth by a factor of $\Theta(s)$.

Algorithm 12 Communication-Avoiding BICGSTAB (CA-BICGSTAB)

Input: $n \times n$ matrix A, length-n vector b, and initial approximation x_1 to $Ax = b$
Output: Approximate solution x_{sk+s+1} to $Ax = b$ with updated residual r_{sk+s+1}

$p_1 = r_1 = b - Ax_1$
Choose $\tilde{r}$ arbitrarily such that $\delta_1 = (\tilde{r}, r_1) \neq 0$
for $k = 0, 1, \ldots,$ until convergence **do**
  Compute $\mathcal{P}_k$ and $\mathcal{R}_k$ according to (4.57)
  Compute G_k and g according to (4.68)
  Assemble $\mathcal{B}_k$ such that (4.58) holds
  Initialize $\{p'_{k,1}, r'_{k,1}, x'_{k,1}\}$ according to (4.60)
  for $j = 1$ to s **do**
    $\alpha_{sk+j} = \delta_{sk+j}/(g, \mathcal{B}_k p'_{k,j})$
    $x'_{k,j+1} = x'_{k,j} + \alpha_{sk+j} p'_{k,j}$
    $d'_{k,j} = r'_{k,j} - \alpha_{sk+j} \mathcal{B}_k p'_{k,j}$
    $\omega_{sk+j} = \frac{(d'_{k,j}, G_k \mathcal{B}_k d'_{k,j})}{(\mathcal{B}_k d'_{k,j}, G_k \mathcal{B}_k d'_{k,j})}$
    $x'_{k,j+1} = x'_{k,j+1} + \omega_{sk+j} d'_{k,j}$
    $r'_{k,j+1} = d'_{k,j} - \omega_{sk+j} \mathcal{B}_k d'_{k,j}$
    $\delta_{sk+j+1} = (g, r'_{k,j+1})$
    $\beta_{sk+j} = (\delta_{sk+j+1}/\delta_{sk+j}) \cdot (\alpha_{sk+j}/\omega_{sk+j})$
    $p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j} p'_{k,j} - \beta_{sk+j} \omega_{sk+j} \mathcal{B}_k p'_{k,j}$
  end for
  Recover iterates $\{p_{sk+s+1}, r_{sk+s+1}, x_{sk+s+1}\}$ according to (4.59)
end for

4.4.1 Numerical Experiments

We present convergence results for two matrices, the same used to test CA-BICG in Section 4.2.1, which exemplify the dependence of typical numerical behavior on the chosen basis and s value. Note that the Krylov bases computed in each outer loop are of length $2s$. In all tests, the right-hand side b was constructed such that the true solution x is the n-vector with components $x_i = 1/\sqrt{n}$.

cdde Figure 4.5 shows convergence results for CA-BICGTAB for the different bases, for $s \in \{4, 8, 16\}$.

xenon1 Figure 4.6 shows convergence results for CA-BICGTAB for the different bases, for $s \in \{4, 8, 16\}$.

As in the CABICG tests in Section 4.2.1, using the monomial basis we generally see the convergence rate decrease (relative to the classical method) as s grows larger due to roundoff error in computation of the basis vectors in the matrix powers kernel. For example, in Fig. 6.1 for CA-BICGSTAB, we see a decrease in convergence rate of the monomial basis from $s = 4$

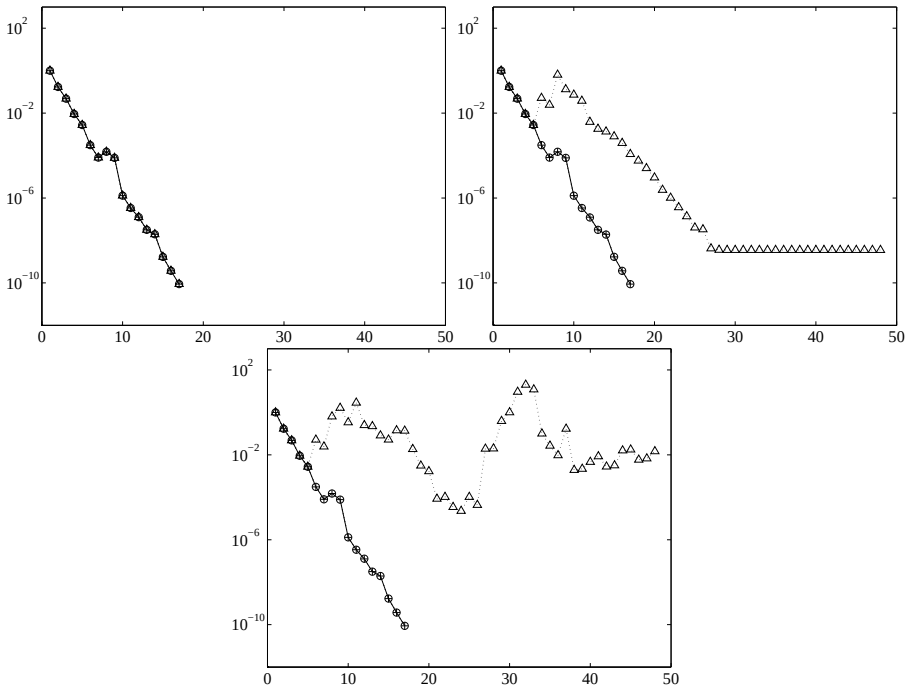

Figure 4.5: Convergence for cdde matrix for (CA-)BICGTAB with various s values: $s = 4$ (top left), $s = 8$ (top right), $s = 16$ (bottom). The x-axis denotes iteration number, and the y-axis denotes the 2-norm of the (normalized) true residual (i.e., $\|b - Ax_m\|_2 / \|b\|_2$). Each plot shows convergence for the classical algorithm ($-$), as well as for CA-BICGTAB using the monomial ($\triangle$), Newton ($\circ$), and Chebyshev ($+$) bases.

to $s = 8$. Again, for both of our test matrices, CA-BICGSTAB with the monomial basis fails to converge to the desired tolerance when $s = 16$.

As with CA-BICG, CA-BICGSTAB with the Newton and Chebyshev bases maintains convergence closer to that of the classical method, even for s as large as 16 (again, note that for CA-BICGSTAB, $s = 16$ requires basis lengths of 32). As observed in Fig. 6.4 (xenon1), for CA-BICGSTAB, the dependence of convergence rate on s is less predictable for the Newton and Chebyshev bases, although we still observe eventual convergence of the true residual to the desired tolerance. This is likely due to the large condition number of xenon1.

We tested for convergence of the true residual to a level of 10^{-10} (i.e., $\|b - Ax_m\|_2 / \|b\|_2 \leq 10^{-10}$). For both test matrices, for CA-BICGSTAB with the monomial basis and $s = 8$, we observe that the norm of the true residual stagnates and never reaches the desired level despite convergence of the computed residual. This effect is due to floating point roundoff error, which causes discrepancy between the true and computed residuals. As the computed residual converges to zero, the updates to the true residual rapidly become negligible—thus

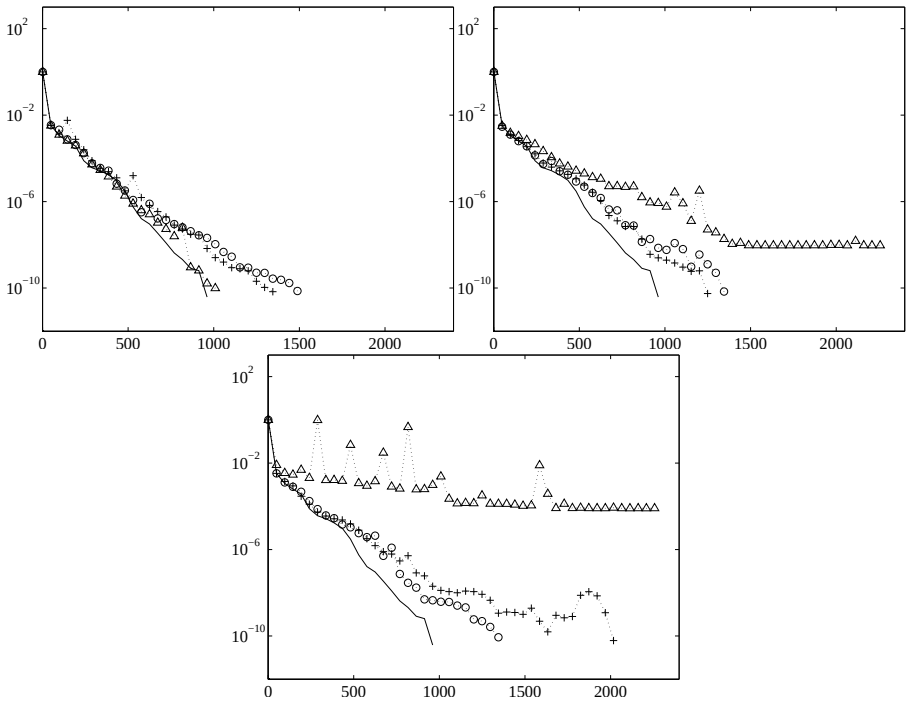

Figure 4.6: Convergence for xenon1 matrix for (CA-)BICGSTAB with various s values: $s = 4$ (top left), $s = 8$ (top right), $s = 16$ (bottom). The x-axis denotes iteration number, and the y-axis denotes the 2-norm of the (normalized) true residual (i.e., $\|b - Ax_m\|_2 / \|b\|_2$). Each plot shows convergence for the classical algorithm ($-$), as well as for CA-BICGSTAB using the monomial ($\triangle$), Newton ($\circ$), and Chebyshev ($+$) bases.

the method may fail to converge to the desired tolerance. The level where stagnation occurs is the *maximum attainable accuracy*.

This phenomenon has been observed before for the classical implementations, and is known to especially plague two-sided KSMs, which can have arbitrarily large iterates [96]. We have observed that this problem is also present in the CA-KSM variants, and is in fact exacerbated the larger the s value. Residual replacement methods have been used effectively to combat this problem in the BICG method [181]. Later in Section 5.3, we perform an equivalent analysis of floating point roundoff error in CA-KSMs and derive an analogous residual replacement strategy which can significantly improve accuracy without asymptotically affecting the communication or computation cost of CA-KSMs.

4.5 Lanczos Bidiagonalization

In this section, we present three approaches to developing communication-avoiding variants of the Lanczos bidiagonalization procedure. Each of the three approaches are used to give both

communication-avoiding upper and lower bidiagonalization routines. The least squares QR solver (LSQR) of Paige and Saunders [144] is based on the Lanczos lower bidiagonalization method, and we use this to derive two potential CA-LSQR methods based on two of our approaches to communication-avoiding lower bidiagonalization.

4.5.1 The Bidiagonalization Algorithm

We first review classical algorithms for reduction of a matrix to both upper bidiagonal and lower bidiagonal form. The original procedure given by Golub and Kahan gives the procedure as a reduction to upper bidiagonal form [81]. With some slight modifications, Paige and Saunders [144] showed that a similar procedure could be used to produce a reduction to lower bidiagonal form, and that this formulation was more amenable to solving the full-rank least squares problem $\min \|Ax - b\|_2$. This observation forms the basis for the LSQR algorithm. Both methods are connected in that they both produce the same sequence of vectors V_i that would be produced by the symmetric Lanczos method applied to A^TA.

Let A be an m-by-n matrix and b be a length-m vector. After i iterations, the Lanczos upper bidiagonalization procedure produces the m-by-i matrix $P_i = [p_1, p_2, \ldots, p_i]$ and the n-by-i matrix $V_i = [v_1, v_2, \ldots, v_i]$ such that

$$\begin{aligned} V_i(\theta_1 e_1) &= A^T b \\ AV_i &= P_i R_i \\ A^T P_i &= V_i R_i^T + \theta_{i+1} v_{i+1} e_i^T, \end{aligned} \tag{4.69}$$

where

$$R_i = \begin{bmatrix} \rho_1 & \theta_2 & & & \\ & \rho_2 & \theta_3 & & \\ & & \ddots & \ddots & \\ & & & \rho_{i-1} & \theta_i \\ & & & & \rho_i \end{bmatrix}, \tag{4.70}$$

and in exact arithmetic, $P_i^T P_i = I$ and $V_i^T V_i = I$. The algorithm of Golub and Kahan for reduction to upper bidiagonal form is shown in Algorithm 13. Note that here and in the remainder of this paper bars over variables denote intermediate quantities which are yet to be normalized. We note that one can formulate the upper bidiagonalization algorithm as the Lanczos reduction to tridiagonal form. Letting

$$Z = [z_1, z_2, \ldots, z_{2i}] \equiv \begin{bmatrix} 0 & p_1 & 0 & p_2 & \ldots & 0 & p_i \\ v_1 & 0 & v_2 & 0 & \ldots & v_i & 0 \end{bmatrix},$$

$$\tilde{A} \equiv \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix}, \quad \text{and} \quad \tilde{T} \equiv \begin{bmatrix} 0 & \rho_1 & & & & & \\ \rho_1 & 0 & \theta_2 & & & & \\ & \theta_2 & 0 & \rho_2 & & & \\ & & \rho_2 & 0 & \ddots & & \\ & & & \ddots & \ddots & \theta_i & \\ & & & & \theta_i & 0 & \rho_i \\ & & & & & \rho_i & 0 \end{bmatrix},$$

the procedure in Algorithm 13 is mathematically equivalent to

$$\tilde{A}Z = Z\tilde{T} + \theta_{i+1} z_{2i+1} e_{2i}^T,$$

with $Z^H Z = I_{2i}$ and $Z^H z_{2i+1} = 0$. This means that, in exact arithmetic, i steps of the upper bidiagonalization procedure applied to A with starting vector v_1 produces the same information as $2i$ steps of symmetric Lanczos applied to cyclic matrix $\tilde{A}$ with starting vector z_1 as defined above.

Algorithm 13 Lanczos Reduction to Upper Bidiagonal Form

Input: m-by-n matrix A and length-n starting vector b

Output: Matrices V_i, P_i and vector v_{i+1} satisfying (4.69) and matrix R_i satisfying (4.70)

1: $\theta_1 = \|A^T b\|_2$, $v_1 = A^T b/\theta_1$, $\bar{p}_1 = Av_1$, $\rho_1 = \|\bar{p}_1\|_2$, $p_1 = \bar{p}_1/\rho_1$
2: **for** $i = 1, 2, \ldots$ until convergence **do**
3: $\quad \bar{v}_{i+1} = A^T p_i - \rho_i v_i$
4: $\quad \theta_{i+1} = \|\bar{v}_{i+1}\|_2$
5: $\quad v_{i+1} = \bar{v}_{i+1}/\theta_{i+1}$
6: $\quad \bar{p}_{i+1} = Av_{i+1} - \theta_{i+1} p_i$
7: $\quad \rho_{i+1} = \|\bar{p}_{i+1}\|_2$
8: $\quad p_{i+1} = \bar{p}_{i+1}/\rho_{i+1}$
9: **end for**

We will also consider reduction to *lower* bidiagonal form for the purpose of easy connection to the LSQR method of Paige and Saunders [144]. Again, A is an m-by-n matrix and b is a length-m vector. After i iterations, the Lanczos lower bidiagonalization procedure produces the m-by-$(i+1)$ matrix $U_{i+1} \equiv [u_1, u_2, \ldots, u_{i+1}]$ and the n-by-i matrix $V_i \equiv [v_1, v_2, \ldots, v_i]$ such that

$$\begin{aligned} U_{i+1}(\beta_1 e_1) &= b \\ AV_i &= U_{i+1} B_i \\ A^T U_{i+1} &= V_i B_i^T + \alpha_{i+1} v_{i+1} e_{i+1}^T, \end{aligned} \tag{4.71}$$

where

$$B_i = \begin{bmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_i \\ & & & \beta_{i+1} \end{bmatrix}, \tag{4.72}$$

and in exact arithmetic, $U_{i+1}^T U_{i+1} = I$ and $V_i^T V_i = I$.

Again in this case, we can formulate the lower bidiagonalization algorithm as the Lanczos reduction to tridiagonal form. Here we define

$$Z = [z_1, z_2, \ldots, z_{2i}] \equiv \begin{bmatrix} 0 & v_1 & 0 & v_2 & \ldots & 0 & v_i \\ u_1 & 0 & u_2 & 0 & \ldots & u_i & 0 \end{bmatrix},$$

$$\tilde{A} \equiv \begin{bmatrix} 0 & A^T \\ A & 0 \end{bmatrix}, \quad \text{and} \quad \tilde{T} \equiv \begin{bmatrix} 0 & \alpha_1 & & & & & \\ \alpha_1 & 0 & \beta_2 & & & & \\ & \beta_2 & 0 & \alpha_2 & & & \\ & & \alpha_2 & 0 & \ddots & & \\ & & & \ddots & \ddots & \beta_i & \\ & & & & \beta_i & 0 & \alpha_i \\ & & & & & \alpha_i & 0 \end{bmatrix},$$

and then Algorithm 14 is mathematically equivalent to

$$\tilde{A}Z = Z\tilde{T} + \beta_{i+1} z_{2i+1} e_{2i}^T,$$

with $Z^H Z = I_{2i}$ and $Z^H z_{2i+1} = 0$. Then in exact arithmetic, i steps of the lower bidiagonalization procedure applied to A with starting vector u_1 produces the same information as $2i$ steps of symmetric Lanczos applied to cyclic matrix $\tilde{A}$ with starting vector z_1 as defined above.

4.5.2 Communication-Avoiding Lanczos Bidiagonalization

There are at least three ways to derive communication-avoiding variants of Algorithms 13 and 14, each with associated pros and cons. The correct method to choose will depend on the structure and conditioning of the matrix, the requirements of the particular application, and machine-specific parameters such as cache size and relative latency/bandwidth cost. We describe the three potential communication-avoiding variants in subsections below.

4.5.2.1 Equivalent Form of CA-Lanczos

As discussed in Section 4.5, i steps of either bidiagonalization procedure in Algorithm 13 or 14 will produce the same information as $2i$ steps of symmetric Lanczos applied to the

Algorithm 14 Lanczos Reduction to Lower Bidiagonal Form

Input: m-by-n matrix A and length-n starting vector b

Output: Matrices V_i, U_{i+1} and vector v_{i+1} satisfying (4.71) and matrix B_i satisfying (4.72)

1: $\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
2: **for** $i = 1, 2, \ldots$ until convergence **do**
3: $\quad \bar{u}_{i+1} = Av_i - \alpha_i u_i$
4: $\quad \beta_{i+1} = \|\bar{u}_{i+1}\|_2$
5: $\quad u_{i+1} = \bar{u}_{i+1}/\beta_{i+1}$
6: $\quad \bar{v}_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$
7: $\quad \alpha_{i+1} = \|\bar{v}_{i+1}\|_2$
8: $\quad v_{i+1} = \bar{v}_{i+1}/\alpha_{i+1}$
9: **end for**

appropriately defined cyclic matrix $\tilde{A}$ and appropriately chosen starting vector z_1. Therefore one can simply use an existing version of CA-Lanczos (available in, e.g., [102, 16, 28]) run on input $\tilde{A}$ and z_1, and recover the bidiagonalization matrices, either P_i, V_i, and R_i for upper bidiagonalization, or U_{i+1}, V_i, and B_i for lower bidiagonalization, from Z and $\tilde{T}$.

This method is simple and allows us to use an existing communication-avoiding method. The drawback is that the system is now twice the size, and extra work and storage will be required unless the Lanczos method is modified to optimize for the block non-zero structure of the matrix/vectors. We would also need a communication-avoiding matrix powers kernel (see Section 3.2) capable of alternating factors of A and A^T.

4.5.2.2 Forming Krylov Bases

By introducing auxiliary quantities, another variant can be derived that works by building s-step Krylov bases with AA^T and A^TA. The benefit here is that other polynomial bases can be used in order to improve numerical properties (e.g., Newton or Chebyshev). The drawbacks are that this requires computing bases with powers of AA^T and A^TA, which squares the condition number of A. Also, in order to satisfy the recurrences, we need $4s + 1$ basis vectors in each iteration, which doubles the number of SpMVs per s iterations versus the classical method (this is assuming we form and store A^TA and AA^T offline; otherwise the number of SpMVs required is $8s + 2$ and we again need a communication-avoiding matrix powers kernel which alternates factors of A and A^T). We note that this is equivalent (in exact arithmetic) to the method described in the previous subsection, but takes nonzero blocks into account and uses auxiliary quantities.

We derive this method below for both upper and lower bidiagonalization procedures. As before, in the communication-avoiding algorithms, we will switch from indexing iterations by i to indexing iterations by $sk + j$, where s is the iteration blocking parameter, k is the outer iteration index, and j in the inner iteration index.

Reduction to Upper Bidiagonal Form Assume we are beginning iteration $sk+1$ of Algorithm 13, where $k \in \mathbb{N}$ and $0 < s \in \mathbb{N}$, so that v_{sk+1} and p_{sk+1} have just been computed. Recall that

$$p_{sk+j} \in \mathcal{K}_{s+1}(AA^T, p_{sk+1}) + \mathcal{K}_s(AA^T, Av_{sk+1}) \quad \text{and}$$
$$v_{sk+j} \in \mathcal{K}_s(A^TA, v_{sk+1}) + \mathcal{K}_s(A^TA, A^Tp_{sk+1}),$$

for $j \in \{1, \ldots, s+1\}$.

We define basis matrices whose columns span these subspaces as follows. Let $\mathcal{V}_k$ be a basis for $\mathcal{K}_s(A^TA, v_{sk+1})$, $\tilde{\mathcal{V}}_k$ a basis for $\mathcal{K}_s(AA^T, Av_{sk+1})$, $\mathcal{P}_k$ a basis for $\mathcal{K}_{s+1}(AA^T, p_{sk+1})$ and $\tilde{\mathcal{P}}_k$ a basis for $\mathcal{K}_s(A^TA, A^Tp_{sk+1})$. These could be computed by the PA1 or SA1 algorithms described in Section 3.2. Assuming these polynomial bases are generated using a three-term recurrence, we can write the recurrence relations

$$(AA^T)[\underline{\mathcal{P}}_k, \underline{\tilde{\mathcal{V}}}_k] = [\mathcal{P}_k, \tilde{\mathcal{V}}_k] \begin{bmatrix} \mathcal{B}_k^{(\mathcal{P})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{V}})} \end{bmatrix} \quad \text{and}$$
$$(A^TA)[\underline{\mathcal{V}}_k, \underline{\tilde{\mathcal{P}}}_k] = [\mathcal{V}_k, \tilde{\mathcal{P}}_k] \begin{bmatrix} \mathcal{B}_k^{(\mathcal{V})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{P}})} \end{bmatrix},$$

where $\underline{\mathcal{P}}_k$, $\underline{\tilde{\mathcal{V}}}_k$, $\underline{\mathcal{V}}_k$, and $\underline{\tilde{\mathcal{P}}}_k$ are the same as $\mathcal{P}_k$, $\tilde{\mathcal{V}}_k$, $\mathcal{V}_k$, and $\tilde{\mathcal{P}}_k$, resp., but with the last column set to 0, and the $\mathcal{B}_k$ matrices are tridiagonal matrices of the form (4.2) with $i = s$ for $\mathcal{B}_k^{(\mathcal{P})}$ and $i = s-1$ for $\mathcal{B}_k^{(\mathcal{V})}$, $\mathcal{B}_k^{(\tilde{\mathcal{V}})}$, and $\mathcal{B}_k^{(\tilde{\mathcal{P}})}$. Note that the entries $\hat{\alpha}_j$, $\hat{\gamma}_j$, and $\hat{\beta}_j$ of the $\mathcal{B}_k$'s can be set differently depending on whether we are constructing polynomials in AA^T or A^TA. Also note that these recurrence coefficients could be refined with each new outer loop, as we discuss further in Section 6.6. See Section 3.2.5 for guidelines on setting these entries such that the basis condition number is improved.

To simplify notation, we will define $\mathcal{Y}_k = [\mathcal{P}_k, \tilde{\mathcal{V}}_k]$, $\underline{\mathcal{Y}}_k = [\underline{\mathcal{P}}_k, \underline{\tilde{\mathcal{V}}}_k]$, $\mathcal{Z}_k = [\mathcal{V}_k, \tilde{\mathcal{P}}_k]$, $\underline{\mathcal{Z}}_k = [\underline{\mathcal{V}}_k, \underline{\tilde{\mathcal{P}}}_k]$, and

$$\mathcal{B}_k^{(\mathcal{Y})} = \begin{bmatrix} \mathcal{B}_k^{(\mathcal{P})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{V}})} \end{bmatrix}, \qquad \mathcal{B}_k^{(\mathcal{Z})} = \begin{bmatrix} \mathcal{B}_k^{(\mathcal{V})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{P}})} \end{bmatrix}.$$

This lets us rewrite the recurrences as

$$(AA^T)\underline{\mathcal{Y}}_k = \mathcal{Y}_k\mathcal{B}_k^{(\mathcal{Y})} \quad \text{and} \quad (A^TA)\underline{\mathcal{Z}}_k = \mathcal{Z}_k\mathcal{B}_k^{(\mathcal{Z})}.$$

The recurrences do not give us a way to represent multiplication by A and A^T in these new bases, which are necessary to perform updates to the coordinate vectors $v'_{k,j+1}$ and $p'_{k,j+1}$. The recurrences do however give ways to multiply by AA^T and A^TA, and we introduce

auxiliary quantities to make use of this. Let

$$\begin{aligned}
\tilde{p}_{sk+j+1} \equiv A^T p_{sk+j+1} &= A^T(Av_{sk+j+1} - \theta_{sk+j+1}p_{sk+j})/\rho_{sk+j+1} \\
&= \big((A^TA)v_{sk+j+1} - \theta_{sk+j+1}\tilde{p}_{sk+j}\big)/\rho_{sk+j+1}, \quad \text{and} \\
\tilde{v}_{sk+j+1} \equiv Av_{sk+j+1} &= A(A^Tp_{sk+j} - \rho_{sk+j}v_{sk+j})/\theta_{sk+j+1} \\
&= \big((AA^T)p_{sk+j} - \rho_{sk+j}\tilde{v}_{sk+j}\big)/\theta_{sk+j+1}.
\end{aligned}$$

Then vector updates can then be written

$$\begin{aligned}
\bar{v}_{sk+j+1} &= \tilde{p}_{sk+j} - \rho_{sk+j}v_{sk+j}, \\
\tilde{v}_{sk+j+1} &= \big((AA^T)p_{sk+j} - \rho_{sk+j}\tilde{v}_{sk+j}\big)/\theta_{sk+j+1}, \quad \text{and} \\
\bar{p}_{sk+j+1} &= \tilde{v}_{sk+j+1} - \theta_{sk+j+1}p_{sk+j}
\end{aligned}$$

for $j \in \{1, \ldots, s\}$, and

$$\tilde{p}_{sk+j+1} = \big((A^TA)v_{sk+j+1} - \theta_{sk+j+1}\tilde{p}_{sk+j}\big)/\rho_{sk+j+1}$$

for $j \in \{1, \ldots, s-1\}$ ($\tilde{p}_{sk+s+1}$ is not needed). As before, $v_{sk+j+1} = \bar{v}_{sk+j+1}/\theta_{sk+j+1}$ and $p_{sk+j+1} = \bar{p}_{sk+j+1}/\rho_{sk+j+1}$. Note that $\tilde{v}_{sk+j+1} \in \mathcal{Y}_k$ for $j \in \{1, \ldots, s\}$ and $\tilde{p}_{sk+j+1} \in \mathcal{Z}_k$ for $j \in \{1, \ldots, s-1\}$, so no additional basis vectors are required to represent updates to these auxiliary quantities. The classical version of this modified upper bidiagonalization algorithm is given in Alg. 15.

Algorithm 15 Lanczos Upper Bidiagonalization with Auxiliary Quantities

Input: m-by-n matrix A and length-n starting vector b

Output: Matrices V_i, P_i and vector v_{i+1} satisfying (4.69) and matrix R_i satisfying (4.70)

$\theta_1 = \|A^Tb\|_2$, $v_1 = A^Tb/\theta_1$, $\bar{p}_1 = Av_1$, $\rho_1 = \|\bar{p}_1\|_2$, $p_1 = \bar{p}_1/\rho_1$
$\tilde{v}_1 = Av_1$, $\tilde{p}_1 = A^Tp_1$
for $i = 1, 2, \ldots$ until convergence **do**
    $\bar{v}_{i+1} = \tilde{p}_i - \rho_i v_i$
    $\theta_{i+1} = \|\bar{v}_{i+1}\|_2$
    $v_{i+1} = \bar{v}_{i+1}/\theta_{i+1}$
    $\tilde{v}_{i+1} = (AA^Tp_i - \rho_i\tilde{v}_i)/\theta_{i+1}$
    $\bar{p}_{i+1} = \tilde{v}_{i+1} - \theta_{i+1}p_i$
    $\rho_{i+1} = \|\bar{p}_{i+1}\|_2$
    $p_{i+1} = \bar{p}_{i+1}/\rho_{i+1}$
    $\tilde{p}_{i+1} = (A^TAv_{i+1} - \theta_{i+1}\tilde{p}_i)/\rho_{i+1}$
end for

We can then represent v_{sk+j+1}, $\tilde{v}_{sk+j+1}$, p_{sk+j+1}, and $\tilde{p}_{sk+j+1}$ by their coordinates $v'_{k,j+1}$, $\tilde{v}'_{k,j+1}$, $p'_{k,j+1}$, and $\tilde{p}'_{k,j+1}$, resp., in $\mathcal{Y}_k$ and $\mathcal{Z}_k$, i.e.,

$$\begin{aligned} v_{sk+j+1} &= \mathcal{Z}_k v'_{k,j+1}, \\ \tilde{v}_{sk+j+1} &= \mathcal{Y}_k \tilde{v}'_{k,j+1}, \text{ and} \\ p_{sk+j+1} &= \mathcal{Y}_k p'_{k,j+1}, \quad \text{for } j \in \{1,\dots,s\}, \text{ and} \\ \tilde{p}_{sk+j+1} &= \mathcal{Z}_k \tilde{p}'_{k,j+1} \quad \text{for } j \in \{1,\dots,s-1\}. \end{aligned} \tag{4.73}$$

Note that using (4.73), in each new outer loop we initialize the coordinate vectors to $p'_{k,1} = e_1$, $v'_{k,1} = e_1$, $\tilde{p}'_{k,1} = e_{s+1}$, and $\tilde{v}'_{k,1} = e_{s+2}$, and update them in each iteration by the formulas

$$\begin{aligned} \bar{v}'_{k,j+1} &= \tilde{p}'_{k,j} - \rho_{sk+j} v'_{k,j}, \\ v'_{k,j+1} &= \bar{v}'_{k,j+1}/\theta_{sk+j+1}, \\ \tilde{v}'_{k,j+1} &= \left(\mathcal{B}_k^{(\mathcal{Y})} p'_{k,j} - \rho_{sk+j} \tilde{v}'_{k,j}\right)/\theta_{sk+j+1}, \\ \bar{p}'_{k,j+1} &= \tilde{v}'_{k,j+1} - \theta_{sk+j+1} p'_{k,j}, \quad \text{and} \\ p'_{k,j+1} &= \bar{p}'_{k,j+1}/\rho_{sk+j+1}, \end{aligned}$$

for $j \in \{1,\dots,s\}$, and

$$\tilde{p}'_{k,j+1} = \left(\mathcal{B}_k^{(\mathcal{Z})} v'_{k,j+1} - \theta_{sk+j+1} \tilde{p}'_{k,j}\right)/\rho_{sk+j+1},$$

for $j \in \{1,\dots,s-1\}$.

Now, it remains to determine how to compute the inner products θ_{sk+j+1} and ρ_{sk+j+1}. We can write

$$\begin{aligned} \theta_{sk+j+1} &= (\bar{v}_{sk+j+1}^T \bar{v}_{sk+j+1})^{1/2} \\ &= ((\mathcal{Z}_k \bar{v}'_{k,j+1})^T (\mathcal{Z}_k \bar{v}'_{k,j+1}))^{1/2} \\ &= (\bar{v}_{k,j+1}'^T \mathcal{Z}_k^T \mathcal{Z}_k \bar{v}'_{k,j+1})^{1/2} \end{aligned} \tag{4.74}$$

and

$$\begin{aligned} \rho_{sk+j+1} &= (\bar{p}_{sk+j+1}^T \bar{p}_{sk+j+1})^{1/2} \\ &= ((\mathcal{Y}_k \bar{p}'_{k,j+1})^T (\mathcal{Y}_k \bar{p}'_{k,j+1}))^{1/2} \\ &= (\bar{p}_{k,j+1}'^T \mathcal{Y}_k^T \mathcal{Y}_k \bar{p}'_{k,j+1})^{1/2}. \end{aligned} \tag{4.75}$$

Defining the Gram matrices

$$G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k \qquad \text{and} \qquad G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k,$$

which can be computed with one Allreduce operation (see Section 3.1), we can compute (4.74) and (4.75) by the formulas

$$\theta_{sk+j+1} = (\bar{v}_{k,j+1}'^T G_k^{(\mathcal{Z})} \bar{v}'_{k,j+1})^{1/2} \quad \text{and} \quad \rho_{sk+j+1} = (\bar{p}_{k,j+1}'^T G_k^{(\mathcal{Y})} \bar{p}'_{k,j+1})^{1/2}.$$

The resulting communication-avoiding version of Algorithm 15 is shown in Algorithm 16.

Algorithm 16 Communication-Avoiding Lanczos Upper Bidiagonalization with Auxiliary Quantities

Input: m-by-n matrix A and length-n starting vector b

Output: Matrices V_{sk+s}, P_{sk+s} and vector v_{sk+s+1} satisfying (4.69) and matrix R_{sk+s} satisfying (4.70)

$\theta_1 = \|A^T b\|_2$, $v_1 = A^T b/\theta_1$, $\bar{p}_1 = Av_1$, $\rho_1 = \|\bar{p}_1\|_2$, $p_1 = \bar{p}_1/\rho_1$
$\tilde{v}_1 = Av_1$, $\tilde{p}_1 = A^T p_1$
for $k = 0, 1, \ldots$ until convergence **do**
    Compute $\mathcal{V}_k$, a basis for $\mathcal{K}_s(A^TA, v_{sk+1})$, $\tilde{\mathcal{V}}_k$, a basis for $\mathcal{K}_s(AA^T, Av_{sk+1})$, $\mathcal{P}_k$, a basis for $\mathcal{K}_{s+1}(AA^T, p_{sk+1})$, and $\tilde{\mathcal{P}}_k$, a basis for $\mathcal{K}_s(A^TA, A^Tp_{sk+1})$. Let $\mathcal{Y}_k = [\mathcal{P}_k, \tilde{\mathcal{V}}_k]$, $\mathcal{Z}_k = [\mathcal{V}_k, \tilde{\mathcal{P}}_k]$.
    $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k \qquad G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$
    $v'_{k,1} = e_1$, $p'_{k,1} = e_1$, $\tilde{v}'_{k,1} = e_{s+2}$, $\tilde{p}'_{k,1} = e_{s+1}$.
    for $j = 1, \ldots, s$ **do**
        $\bar{v}'_{k,j+1} = \tilde{p}'_{k,j} - \rho_{sk+j} v'_{k,j}$
        $\theta_{sk+j+1} = \left(\bar{v}'^T_{k,j+1} G_k^{(\mathcal{Z})} \bar{v}'_{k,j+1}\right)^{1/2}$
        $v'_{k,j+1} = \bar{v}'_{k,j+1}/\theta_{sk+j+1}$
        $\tilde{v}'_{k,j+1} = (\mathcal{B}_k^{(\mathcal{Y})} p'_{k,j} - \rho_{sk+j}\tilde{v}'_{k,j})/\theta_{sk+j+1}$
        $\bar{p}'_{k,j+1} = \tilde{v}'_{k,j+1} - \theta_{sk+j+1} p'_{k,j}$
        $\rho_{sk+j+1} = \left(\bar{p}'^T_{k,j+1} G_k^{(\mathcal{Y})} \bar{p}'_{k,j+1}\right)^{1/2}$
        $p'_{k,j+1} = \bar{p}'_{k,j+1}/\rho_{sk+j+1}$
        if $j < s$ **then**
            $\tilde{p}'_{k,j+1} = (\mathcal{B}_k^{(\mathcal{Z})} v'_{k,j+1} - \theta_{sk+j+1}\tilde{p}'_{k,j})/\rho_{sk+j+1}$
        end if
    end for
    Recover $\{v_{sk+j+1}, p_{sk+j+1}\}$ for $j \in \{1, \ldots, s\}$ according to (4.73)
end for

Reduction to Lower Bidiagonal Form Now assume we are beginning iteration $sk+1$ of Algorithm 14, where $k \in \mathbb{N}$ and $0 < s \in \mathbb{N}$, so that u_{sk+1} and v_{sk+1} have just been computed. Recall that

$$u_{sk+j} \in \mathcal{K}_s(AA^T, Av_{sk+1}) + \mathcal{K}_s(AA^T, u_{sk+1}) \quad \text{and}$$
$$v_{sk+j} \in \mathcal{K}_{s+1}(A^TA, v_{sk+1}) + \mathcal{K}_s(A^TA, A^Tu_{sk+1}),$$

for $j \in \{1, \ldots, s+1\}$.

We then define basis matrices whose columns span the desired subspaces as follows. Let $\mathcal{U}_k$ be a basis for $\mathcal{K}_s(AA^T, u_{sk+1})$, $\tilde{\mathcal{V}}_k$ a basis for $\mathcal{K}_s(AA^T, Av_{sk+1})$, $\mathcal{V}_k$ a basis for $\mathcal{K}_{s+1}(A^TA, v_{sk+1})$ and $\tilde{\mathcal{U}}_k$ a basis for $\mathcal{K}_s(A^TA, A^Tu_{sk+1})$. As for the upper bidiagonalization case, these could be computed by the PA1 or SA1 algorithms described in Section 3.2.

Assuming these polynomial bases are generated using a three-term recurrence, we can write the recurrence relations

$$(AA^T)[\underline{\mathcal{U}}_k, \underline{\tilde{\mathcal{V}}}_k] = [\mathcal{U}_k, \tilde{\mathcal{V}}_k] \begin{bmatrix} \mathcal{B}_k^{(\mathcal{U})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{V}})} \end{bmatrix} \quad \text{and}$$
$$(A^TA)[\underline{\mathcal{V}}_k, \underline{\tilde{\mathcal{U}}}_k] = [\mathcal{V}_k, \tilde{\mathcal{U}}_k] \begin{bmatrix} \mathcal{B}_k^{(\mathcal{V})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{U}})} \end{bmatrix},$$

where $\underline{\mathcal{U}}_k$, $\underline{\tilde{\mathcal{V}}}_k$, $\underline{\mathcal{V}}_k$, and $\underline{\tilde{\mathcal{U}}}_k$ are the same as $\mathcal{U}_k$, $\tilde{\mathcal{V}}_k$, $\mathcal{V}_k$, and $\tilde{\mathcal{U}}_k$, resp., but with the last column set to 0, and the $\mathcal{B}_k$ matrices are tridiagonal matrices of the form given in (4.2) with $i = s$ for $\mathcal{B}_k^{(\mathcal{V})}$ and $i = s-1$ for $\mathcal{B}_k^{(\mathcal{U})}$, $\mathcal{B}_k^{(\tilde{\mathcal{V}})}$, and $\mathcal{B}_k^{(\tilde{\mathcal{U}})}$. As before the entries $\hat{\alpha}_j$, $\hat{\gamma}_j$, and $\hat{\beta}_j$ can be different depending on whether we are constructing polynomials in AA^T or A^TA and could be refined with each new outer loop.

To simplify notation, we will define $\mathcal{Y}_k = [\mathcal{U}_k, \tilde{\mathcal{V}}_k]$, $\underline{\mathcal{Y}}_k = [\underline{\mathcal{U}}_k, \underline{\tilde{\mathcal{V}}}_k]$, $\mathcal{Z}_k = [\mathcal{V}_k, \tilde{\mathcal{U}}_Z]$, $\underline{\mathcal{Z}}_k = [\underline{\mathcal{V}}_k, \underline{\tilde{\mathcal{U}}}_k]$, and

$$\mathcal{B}_k^{(\mathcal{Y})} \equiv \begin{bmatrix} \mathcal{B}_k^{(\mathcal{U})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{V}})} \end{bmatrix}, \qquad \mathcal{B}_k^{(\mathcal{Z})} \equiv \begin{bmatrix} \mathcal{B}_k^{(\mathcal{V})} & 0 \\ 0 & \mathcal{B}_k^{(\tilde{\mathcal{U}})} \end{bmatrix}.$$

This lets us rewrite the recurrences as

$$(AA^T)\underline{\mathcal{Y}}_k = \mathcal{Y}_k \mathcal{B}_k^{(\mathcal{Y})} \quad \text{and}$$
$$(A^TA)\underline{\mathcal{Z}}_k = \mathcal{Z}_k \mathcal{B}_k^{(\mathcal{Z})}.$$

Note that these are the same recurrences used in the communication-avoiding upper bidiagonalization method of the previous subsection, but with different definitions of $\mathcal{Y}_k$, $\mathcal{Z}_k$, $\tilde{\mathcal{Y}}_k$, and $\tilde{\mathcal{Z}}_k$. Again, we introduce auxiliary quantities. Let

$$\begin{aligned} \tilde{u}_{sk+j+1} \equiv A^T u_{sk+j+1} &= A^T(Av_{sk+j} - \alpha_{sk+j}u_{sk+j})/\beta_{sk+j+1} \\ &= \big((A^TA)v_{sk+j} - \alpha_{sk+j}\tilde{u}_{sk+j}\big)/\beta_{sk+j+1}, \quad \text{and} \\ \tilde{v}_{sk+j+1} \equiv Av_{sk+j+1} &= A(A^Tu_{sk+j+1} - \beta_{sk+j+1}v_{sk+j})/\alpha_{sk+j+1} \\ &= \big((AA^T)u_{sk+j+1} - \beta_{sk+j+1}\tilde{v}_{sk+j}\big)/\alpha_{sk+j+1}. \end{aligned}$$

Then the vector updates become

$$\begin{aligned} \bar{u}_{sk+j+1} &= (\tilde{v}_{sk+j} - \alpha_{sk+j}u_{sk+j}), \\ \tilde{u}_{sk+j+1} &= \big((A^TA)v_{sk+j} - \alpha_{sk+j}\tilde{u}_{sk+j}\big)/\beta_{sk+j+1}, \quad \text{and} \\ \bar{v}_{sk+j+1} &= (\tilde{u}_{sk+j} - \beta_{sk+j+1}v_{sk+j}), \end{aligned}$$

for $j \in \{1, \ldots, s\}$, and

$$\tilde{v}_{sk+j+1} = \big((AA^T)u_{sk+j+1} - \beta_{sk+j+1}\tilde{v}_{sk+j}\big)/\alpha_{sk+j+1},$$

Algorithm 17 Lanczos Lower Bidiagonalization with Auxiliary Quantities

Input: m-by-n matrix A and length-n starting vector b

Output: Matrices V_i, U_{i+1} and vector v_{i+1} satisfying (4.71) and matrix B_i satisfying (4.72)

$\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
$\tilde{u}_1 = A^T u_1$, $\tilde{v}_1 = A v_1$
for $i = 1, 2, \ldots$ until convergence **do**
    $\bar{u}_{i+1} = \tilde{v}_i - \alpha_i u_i$
    $\beta_{i+1} = \|\bar{u}_{i+1}\|_2$
    $u_{i+1} = \bar{u}_{i+1}/\beta_{i+1}$
    $\tilde{u}_{i+1} = (A^T A v_i - \alpha_i \tilde{u}_i)/\beta_{i+1}$
    $\bar{v}_{i+1} = \tilde{u}_{i+1} - \beta_{i+1} v_i$
    $\alpha_{i+1} = \|\bar{v}_{i+1}\|_2$
    $v_{i+1} = \bar{v}_{i+1}/\alpha_{i+1}$
    $\tilde{v}_{i+1} = (A A^T u_{i+1} - \beta_{i+1}\tilde{v}_i)/\alpha_{i+1}$
end for

for $j \in \{1, \ldots, s-1\}$. As before, $u_{sk+j+1} = \bar{u}_{sk+j+1}/\beta_{sk+j+1}$ and $v_{sk+j+1} = \bar{v}_{sk+j+1}/\alpha_{sk+j+1}$. The classical version of this modified lower bidiagonalization algorithm is given in Algorithm 17.

Note that in Algorithm 17, $\tilde{u}_{sk+j+1} \in \mathcal{Z}_k$ for $j \in \{0, \ldots, s\}$, and $\tilde{v}_{sk+j+1} \in \mathcal{Y}_k$ for $j \in \{0, \ldots, s-1\}$. Then we can represent u_{sk+j+1}, $\tilde{u}_{sk+j+1}$, v_{sk+j+1}, and $\tilde{v}_{sk+j+1}$ by their coordinates $u'_{k,j+1}$, $\tilde{u}'_{k,j+1}$, $v'_{k,j+1}$, and $\tilde{v}'_{k,j+1}$, resp., in $\mathcal{Y}_k$ and $\mathcal{Z}_k$, i.e.,

$$\begin{aligned} u_{sk+j+1} &= \mathcal{Y}_k u'_{k,j+1}, \\ \tilde{u}_{sk+j+1} &= \mathcal{Z}_k \tilde{u}'_{k,j+1}, \quad \text{and} \\ v_{sk+j+1} &= \mathcal{Z}_k v'_{k,j+1} \quad \text{for } j \in \{1, \ldots, s\}, \quad \text{and} \\ \tilde{v}_{sk+j+1} &= \mathcal{Y}_k \tilde{u}'_{k,j+1} \quad \text{for } j \in \{1, \ldots, s-1\}. \end{aligned} \tag{4.76}$$

Note that using (4.76), in each new outer loop we initialize the coordinate vectors to $u'_{k,1} = e_1$, $v'_{k,1} = e_1$, $\tilde{u}'_{k,1} = e_{s+2}$, and $\tilde{v}'_{k,1} = e_{s+1}$, and update them in each iteration by the formulas

$$\begin{aligned} \bar{u}'_{k,j+1} &= \tilde{v}'_{k,j} - \alpha_{sk+j} u'_{k,j}, \\ u'_{k,j+1} &= \bar{u}'_{k,j+1}/\beta_{sk+j+1}, \\ \tilde{u}'_{k,j+1} &= \big(\mathcal{B}_k^{(\mathcal{Z})} v'_{k,j} - \alpha_{sk+j}\tilde{u}'_{k,j}\big)/\beta_{sk+j+1}, \\ \bar{v}'_{k,j+1} &= \tilde{u}'_{k,j+1} - \beta_{sk+j+1} v'_{k,j}, \quad \text{and} \\ v'_{k,j+1} &= \bar{v}'_{k,j+1}/\alpha_{sk+j+1}, \end{aligned}$$

for $j \in \{1, \ldots, s\}$, and

$$\tilde{v}'_{k,j+1} = \big(\mathcal{B}_k^{(\mathcal{Y})} u'_{k,j+1} - \beta_{sk+j+1}\tilde{v}'_{k,j}\big)/\alpha_{sk+j+1},$$

for $j \in \{1, \ldots, s-1\}$.

Now, it only remains to determine how to compute the inner products β_{sk+j+1} and α_{sk+j+1}. We can write

$$\begin{aligned}\beta_{sk+j+1} &= (\bar{u}_{sk+j+1}^T \bar{u}_{sk+j+1})^{1/2} \\ &= ((\mathcal{Y}_k \bar{u}'_{k,j+1})^T (\mathcal{Y}_k \bar{u}'_{k,j+1}))^{1/2} \\ &= (\bar{u}_{k,j+1}'^T \mathcal{Y}_k^T \mathcal{Y}_k \bar{u}'_{k,j+1})^{1/2} \end{aligned} \tag{4.77}$$

and

$$\begin{aligned}\alpha_{sk+j+1} &= (\bar{v}_{sk+j+1}^T \bar{v}_{sk+j+1})^{1/2} \\ &= ((\mathcal{Z}_k \bar{v}'_{k,j+1})^T (\mathcal{Z}_k \bar{v}'_{k,j+1}))^{1/2} \\ &= (\bar{v}_{k,j+1}'^T \mathcal{Z}_k^T \mathcal{Z}_k \bar{v}'_{k,j+1})^{1/2}. \end{aligned} \tag{4.78}$$

Defining the Gram matrices

$$G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k \qquad \text{and} \qquad G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k,$$

which can be computed with one Allreduce operation (see Section 3.1), equations (4.77) and (4.78) become

$$\beta_{sk+j+1} = (\bar{u}_{k,j+1}'^T G_k^{(\mathcal{Y})} \bar{u}'_{k,j+1})^{1/2} \quad \text{and} \quad \alpha_{sk+j+1} = (\bar{v}_{k,j+1}'^T G_k^{(\mathcal{Z})} \bar{v}'_{k,j+1})^{1/2}.$$

The resulting communication-avoiding version of Algorithm 17 is shown in Algorithm 18.

4.5.2.3 Alternating Matrix Powers

Another communication-avoiding variant can be derived which builds two coupled Krylov bases, where basis vectors are computed by alternating between multiplication by A and by A^T. We still need to obtain $4s+1$ basis vectors in order to take s steps of the algorithm, but in this case we do not need to construct or multiply by $A^T A$ and AA^T. We do, however, need a communication-avoiding matrix powers kernel capable of alternating factors of A and A^T. It is less clear how to choose polynomial basis parameters (entries of $\mathcal{B}_k$) for computing matrix powers in this case. Numerical and performance comparisons between these versions and the communication-avoiding versions discussed in Section 4.5.2.2 remains future work. Note, as before, in both algorithms derived below we show recovery of all iteration vectors after each inner loop, although only the last vectors are needed to begin the next outer loop.

Reduction to Upper Bidiagonal Form We use coupled recurrences to generate bases for v_{sk+j+1} and p_{sk+j+1}. Recall that for $j \in \{1, \ldots, s+1\}$,

$$\begin{aligned} &p_{sk+j} \in \mathcal{K}_{s+1}(AA^T, p_{sk+1}) + \mathcal{K}_s(AA^T, Av_{sk+1}) \quad \text{and} \\ &v_{sk+j} \in \mathcal{K}_s(A^T A, v_{sk+1}) + \mathcal{K}_s(A^T A, A^T p_{sk+1}). \end{aligned}$$

Algorithm 18 Communication-Avoiding Lanczos Lower Bidiagonalization with Auxiliary Quantities

Input: m-by-n matrix A and length-n starting vector b

Output: Matrices V_{sk+s}, U_{sk+s+1} and vector v_{sk+s+1} satisfying (4.71) and matrix B_{sk+s} satisfying (4.72)

$\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
$\tilde{u}_1 = A^T u_1$, $\tilde{v}_1 = A v_1$
for $k = 0, 1, \ldots$ until convergence **do**
    Compute $\mathcal{U}_k$, a basis for $\mathcal{K}_s(AA^T, u_{sk+1})$, $\tilde{\mathcal{V}}_k$, a basis for $\mathcal{K}_s(AA^T, Av_{sk+1})$, $\mathcal{V}_k$, a basis for $\mathcal{K}_{s+1}(A^TA, v_{sk+1})$, and $\tilde{\mathcal{U}}_k$, a basis for $\mathcal{K}_s(A^TA, A^Tu_{sk+1})$. Let $\mathcal{Y}_k = [\mathcal{U}_k, \tilde{\mathcal{V}}_k]$, $\mathcal{Z}_k = [\mathcal{V}_k, \tilde{\mathcal{U}}_k]$.
    $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k \qquad G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$
    $u'_{k,1} = e_1$, $v'_{k,1} = e_1$, $\tilde{u}'_{k,1} = e_{s+2}$, $\tilde{v}'_{k,1} = e_{s+1}$.
    for $j = 1, \ldots, s$ **do**
        $\bar{u}'_{k,j+1} = \tilde{v}'_{k,j} - \alpha_{sk+j} u'_{k,j}$
        $\beta_{sk+j+1} = \left(\bar{u}'^T_{k,j+1} G_k^{(\mathcal{Y})} \bar{u}'_{k,j+1} \right)^{1/2}$
        $u'_{k,j+1} = \bar{u}'_{k,j+1} / \beta_{sk+j+1}$
        $\tilde{u}'_{k,j+1} = (\mathcal{B}_k^{(\mathcal{Z})} v'_{k,j} - \alpha_{sk+j} \tilde{u}'_{k,j}) / \beta_{sk+j+1}$
        $\bar{v}'_{k,j+1} = \tilde{u}'_{k,j+1} - \beta_{sk+j+1} v'_{k,j}$
        $\alpha_{sk+j+1} = \left(\bar{v}'^T_{k,j+1} G_k^{(\mathcal{Z})} \bar{v}'_{k,j+1} \right)^{1/2}$
        $v'_{k,j+1} = \bar{v}'_{k,j+1} / \alpha_{sk+j+1}$
        if $j < s$ **then**
            $\tilde{v}'_{k,j+1} = (\mathcal{B}_k^{(\mathcal{Y})} u'_{k,j+1} - \beta_{sk+j+1} \tilde{v}'_{k,j}) / \alpha_{sk+j+1}$
        end if
    end for
    Recover $\{u_{sk+j+1}, v_{sk+j+1}\}$ for $j \in \{1, \ldots, s\}$ according to (4.76)
end for

Then assume that we have dimension $n \times (2s+1)$ matrix $\mathcal{Z}_k$ whose first $2s$ columns are a basis for $\mathcal{K}_s(A^TA, v_{sk+1}) + \mathcal{K}_s(A^TA, A^Tp_{sk+1})$ and whose last column is 0 and dimension $m \times (2s+1)$ matrix $\mathcal{Y}_k$ whose columns form a basis for $\mathcal{K}_{s+1}(AA^T, p_{sk+1}) + \mathcal{K}_s(AA^T, Av_{sk+1})$. Then there exist vectors $v'_{k,j}$ and $p'_{k,j}$ such that

$$v_{sk+j} = \mathcal{Z}_k v'_{k,j} \quad \text{and} \quad p_{sk+j} = \mathcal{Y}_k p'_{k,j},$$

for $j \in \{1, \ldots, s+1\}$.

Assume that $\mathcal{Z}_k$ and $\mathcal{Y}_k$ satisfy the recurrences

$$A\mathcal{Z}_k = \mathcal{Y}_k \mathcal{B}_k \quad \text{and} \quad A^T \underline{\mathcal{Y}}_k = \mathcal{Z}_k \tilde{\mathcal{B}}_k,$$

where $\underline{\underline{\mathcal{Y}}}_k$ is the same as $\mathcal{Y}_k$, but with the last two columns set to 0, and the matrices $\mathcal{B}_k$ and $\tilde{\mathcal{B}}_k$ are dimension $(2s+1)\times(2s+1)$ and are of the forms

$$\mathcal{B}_k = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_1 & & & 0 \\ \hat{\gamma}_1 & \hat{\alpha}_2 & \ddots & & \vdots \\ & \hat{\gamma}_2 & \ddots & \hat{\beta}_{2s-1} & 0 \\ & & \ddots & \hat{\alpha}_{2s} & 0 \\ & & & \hat{\gamma}_{2s} & 0 \end{bmatrix} \quad \text{and} \quad \tilde{\mathcal{B}}_k = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_1 & & & 0 & 0 \\ \hat{\gamma}_1 & \hat{\alpha}_2 & \ddots & & \vdots & \vdots \\ & \hat{\gamma}_2 & \ddots & \hat{\beta}_{2s-2} & 0 & 0 \\ & & \ddots & \hat{\alpha}_{2s-1} & 0 & 0 \\ & & & \hat{\gamma}_{2s-1} & 0 & 0 \\ 0 & 0 & 0 & \dots & 0 & 0 \end{bmatrix}. \tag{4.79}$$

Given $z_1 = v_{sk+1}$ and $y_1 = p_{sk+1}$, by the above recurrences the columns of $\mathcal{Z}_k$ and $\mathcal{Y}_k$ are generated by computing

$$\begin{aligned} y_2 &= (Az_1 - \hat{\alpha}_1 y_1)/\hat{\gamma}_1, \\ z_2 &= (A^T y_1 - \hat{\alpha}_1 z_1)/\hat{\gamma}_1, \quad \text{and} \\ y_{\ell+1} &= (Az_\ell - \hat{\alpha}_\ell y_\ell - \hat{\beta}_{\ell-1} y_{\ell-1})/\hat{\gamma}_\ell \quad \text{for } \ell \in \{2,\dots,2s\}, \\ z_{\ell+1} &= (A^T y_\ell - \hat{\alpha}_\ell z_\ell - \hat{\beta}_{\ell-1} z_{\ell-1})/\hat{\gamma}_\ell \quad \text{for } \ell \in \{2,\dots,2s-1\}, \end{aligned}$$

where z_ℓ and v_ℓ denote the ℓth columns of $\mathcal{Z}_k$ and $\mathcal{Y}_k$, respectively. Note that above, the coefficients $\hat{\alpha}_\ell$, $\hat{\beta}_\ell$, and $\hat{\gamma}_\ell$ could be different for computation of $y_{\ell+1}$ and $z_{\ell+1}$.

We can then write

$$\begin{aligned} \bar{v}_{sk+j+1} = \mathcal{Z}_k \bar{v}'_{k,j+1} &= A^T p_{sk+j} - \rho_{sk+j} v_{sk+j} \\ &= A^T \underline{\underline{\mathcal{Y}}}_k p'_{k,j} - \rho_{sk+j} \mathcal{Z}_k v'_{k,j} \\ &= \mathcal{Z}_k \tilde{\mathcal{B}}_k p'_{k,j} - \rho_{sk+j} \mathcal{Z}_k v'_{k,j} \end{aligned}$$

and

$$\begin{aligned} \bar{p}_{sk+j+1} = \mathcal{Y}_k \bar{p}'_{k,j+1} &= A v_{sk+j+1} - \theta_{sk+j+1} p_{sk+j} \\ &= A \mathcal{Z}_k v'_{k,j+1} - \theta_{sk+j+1} \mathcal{Y}_k p'_{k,j} \\ &= \mathcal{Y}_k \mathcal{B}_k v'_{k,j+1} - \theta_{sk+j+1} \mathcal{Y}_k p'_{k,j}. \end{aligned}$$

Therefore in the inner loop we can update

$$\begin{aligned} \bar{v}'_{k,j+1} &= \tilde{\mathcal{B}}_k p'_{k,j} - \rho_{sk+j} v'_{k,j} \qquad \text{and} \\ \bar{p}'_{k,j+1} &= \mathcal{B}_k v'_{k,j+1} - \theta_{sk+j+1} p'_{k,j}, \end{aligned}$$

and recover the iteration vectors by

$$v_{sk+j+1} = \mathcal{Z}_k v'_{k,j+1} \quad \text{and} \quad p_{sk+j+1} = \mathcal{Y}_k p'_{k,j+1}, \tag{4.80}$$

for $j \in \{1, \ldots, s\}$.

The scalars required for normalization can be computed by

$$\theta_{sk+j+1} = \|\bar{v}_{sk+j+1}\|_2 = \left(\bar{v}'^T_{k,j+1}(\mathcal{Z}_k^T \mathcal{Z}_k)\bar{v}'_{k,j+1}\right)^{1/2} = \left(\bar{v}'^T_{k,j+1} G_k^{(\mathcal{Z})} \bar{v}'_{k,j+1}\right)^{1/2} \quad \text{and}$$
$$\rho_{sk+j+1} = \|\bar{p}_{sk+j+1}\|_2 = \left(\bar{p}'^T_{k,j+1}(\mathcal{Y}_k^T \mathcal{Y}_k)\bar{p}'_{k,j+1}\right)^{1/2} = \left(\bar{p}'^T_{k,j+1} G_k^{(\mathcal{Y})} \bar{p}'_{k,j+1}\right)^{1/2},$$

with $G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$ and $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k$. We can then update

$$v'_{k,j+1} = \bar{v}'_{k,j+1}/\theta_{sk+j+1} \quad \text{and}$$
$$p'_{k,j+1} = \bar{p}'_{k,j+1}/\rho_{sk+j+1}.$$

The resulting communication-avoiding upper bidiagonalization algorithm is shown in Algorithm 19.

Algorithm 19 Communication-Avoiding Upper Bidiagonalization with Alternating Matrix Powers

Input: m-by-n matrix A and length-n starting vector b

Output: Matrices V_{sk+s}, P_{sk+s} and vector v_{sk+s+1} satisfying (4.69) and matrix R_{sk+s} satisfying (4.70)

$\theta_1 = \|A^T b\|_2$, $v_1 = A^T b/\theta_1$, $\bar{p}_1 = Av_1$, $\rho_1 = \|\bar{p}_1\|_2$, $p_1 = \bar{p}_1/\rho_1$
for $k = 0, 1, \ldots$ until convergence **do**
  Compute $\mathcal{Z}_k$ and $\mathcal{Y}_k$ such that $A\mathcal{Z}_k = \mathcal{Y}_k \mathcal{B}_k$ and $A^T \underline{\mathcal{Y}}_k = \mathcal{Z}_k \tilde{\mathcal{B}}_k$.
  $G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$, $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k$
  $v'_{k,1} = e_1$, $p'_{k,1} = e_1$
  for $j = 1, \ldots, s$ **do**
    $\bar{v}'_{k,j+1} = \tilde{\mathcal{B}}_k p'_{k,j} - \rho_{sk+j} v'_{k,j}$
    $\theta_{sk+j+1} = \left(\bar{v}'^T_{k,j+1} G_k^{(\mathcal{Z})} \bar{v}'_{k,j+1}\right)^{1/2}$
    $v'_{k,j+1} = \bar{v}'_{k,j+1}/\theta_{sk+j+1}$
    $\bar{p}'_{j+1} = \mathcal{B}_k v'_{j+1} - \theta_{sk+j+1} p'_j$
    $\rho_{sk+j+1} = \left(\bar{p}'^T_{k,j+1} G_k^{(\mathcal{Y})} \bar{p}'_{k,j+1}\right)^{1/2}$
    $p'_{k,j+1} = \bar{p}'_{k,j+1}/\rho_{sk+j+1}$
  end for
  Recover $\{v_{sk+j+1}, p_{sk+j+1}\}$ for $j \in \{1, \ldots, s\}$ according to (4.80)
end for

Reduction to Lower Bidiagonal Form Now we derive a version of lower bidiagonalization using coupled recurrences to generate bases for u_{sk+j+1} and v_{sk+j+1}. Recall that for

$j \in \{1, \ldots, s+1\}$,

$$u_{sk+j} \in \mathcal{K}_s(AA^T, Av_{sk+1}) + \mathcal{K}_s(AA^T, u_{sk+1}) \quad \text{and}$$
$$v_{sk+j} \in \mathcal{K}_{s+1}(A^TA, v_{sk+1}) + \mathcal{K}_s(A^TA, A^Tu_{sk+1}).$$

Then assume that we have dimension $m \times (2s+1)$ matrix $\mathcal{Y}_k$ whose first $2s$ columns are a basis for $\mathcal{K}_s(AA^T, Av_{sk+1}) + \mathcal{K}_s(AA^T, u_{sk+1})$ and whose last column is 0 and dimension $n \times (2s+1)$ matrix $\mathcal{Z}_k$ whose columns form a basis for $\mathcal{K}_{s+1}(A^TA, v_{sk+1}) + \mathcal{K}_s(A^TA, A^Tu_{sk+1})$. Then there exist vectors $u'_{k,j}$ and $v'_{k,j}$ such that

$$u_{sk+j} = \mathcal{Y}_k u'_{k,j} \quad \text{and} \quad v_{sk+j} = \mathcal{Z}_k v'_{k,j},$$

for $j \in \{1, \ldots, s+1\}$.

Assume that $\mathcal{Y}_k$ and $\mathcal{Z}_k$ satisfy the recurrences

$$A\underline{\underline{\mathcal{Z}}}_k = \mathcal{Y}_k \mathcal{B}_k \quad \text{and} \quad A^T\mathcal{Y}_k = \mathcal{Z}_k \tilde{\mathcal{B}}_k,$$

where $\underline{\underline{\mathcal{Z}}}_k$ is the same as $\mathcal{Z}_k$, but with the last two columns set to 0, and the matrices $\mathcal{B}_k$ and $\tilde{\mathcal{B}}_k$ are dimension $(2s+1) \times (2s+1)$ and are of the forms

$$\tilde{\mathcal{B}}_k = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_1 & & & 0 \\ \hat{\gamma}_1 & \hat{\alpha}_2 & \ddots & & \vdots \\ & \hat{\gamma}_2 & \ddots & \hat{\beta}_{2s-1} & 0 \\ & & \ddots & \hat{\alpha}_{2s} & 0 \\ & & & \hat{\gamma}_{2s} & 0 \end{bmatrix} \quad \text{and} \quad \mathcal{B}_k = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_1 & & & 0 & 0 \\ \hat{\gamma}_1 & \hat{\alpha}_2 & \ddots & & \vdots & \vdots \\ & \hat{\gamma}_2 & \ddots & \hat{\beta}_{2s-2} & 0 & 0 \\ & & \ddots & \hat{\alpha}_{2s-1} & 0 & 0 \\ & & & \hat{\gamma}_{2s-1} & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 \end{bmatrix}. \tag{4.81}$$

Given u_{sk+1} and v_{sk+1}, by the recurrence, the columns of $\mathcal{Y}_k$ and $\mathcal{Z}_k$ are generated by computing

$$\begin{aligned} y_2 &= (Az_1 - \hat{\alpha}_1 y_1)/\hat{\gamma}_1, \\ z_2 &= (A^T y_1 - \hat{\alpha}_1 z_1)/\hat{\gamma}_1, \quad \text{and} \\ y_{\ell+1} &= (Az_\ell - \hat{\alpha}_\ell y_\ell - \hat{\beta}_{\ell-1} y_{\ell-1})/\hat{\gamma}_\ell \quad \text{for } \ell \in \{2, \ldots, 2s-1\}, \\ z_{\ell+1} &= (A^T y_\ell - \hat{\alpha}_\ell z_\ell - \hat{\beta}_{\ell-1} z_{\ell-1})/\hat{\gamma}_\ell \quad \text{for } \ell \in \{2, \ldots, 2s\}, \end{aligned}$$

where z_ℓ and v_ℓ denote the ℓth columns of $\mathcal{Z}_k$ and $\mathcal{Y}_k$, respectively. Note that above, as in the upper bidiagonalization case, the coefficients $\hat{\alpha}_\ell$, $\hat{\beta}_\ell$, and $\hat{\gamma}_\ell$ can be different for computation of $y_{\ell+1}$ and $z_{\ell+1}$.

Then

$$\begin{aligned} \bar{u}_{sk+j+1} = \mathcal{Y}_k \bar{u}'_{k,j+1} &= Av_{sk+j} - \alpha_{sk+j} u_{sk+j} \\ &= A\underline{\underline{\mathcal{Z}}}_k v'_{k,j} - \alpha_{sk+j} \mathcal{Y}_k u'_{k,j} \\ &= \mathcal{Y}_k \mathcal{B}_k v'_{k,j} - \alpha_{sk+j} \mathcal{Y}_k u'_{k,j} \end{aligned}$$

and

$$\begin{aligned}\bar{v}_{sk+j+1} = \mathcal{Z}_k \bar{v}'_{k,j+1} &= A^T u_{sk+j+1} - \beta_{sk+j+1} v_{sk+j} \\ &= A^T \mathcal{Y}_k u'_{k,j+1} - \beta_{sk+j+1} \mathcal{Z}_k v'_{k,j} \\ &= \mathcal{Z}_k \tilde{\mathcal{B}}_k u'_{k,j+1} - \beta_{sk+j+1} \mathcal{Z}_k v'_{k,j}.\end{aligned}$$

Therefore in the inner loop we can update

$$\begin{aligned}\bar{u}'_{k,j+1} &= \mathcal{B}_k v'_{k,j} - \alpha_{sk+j} u'_{k,j} \qquad \text{and} \\ \bar{v}'_{k,j+1} &= \tilde{\mathcal{B}}_k u'_{k,j+1} - \beta_{sk+j+1} v'_{k,j},\end{aligned}$$

and recover the iteration vectors by

$$u_{sk+j+1} = \mathcal{Y}_k u'_{k,j+1} \quad \text{and} \quad v_{sk+j+1} = \mathcal{Z}_k v'_{k,j+1}, \tag{4.82}$$

for $j \in \{1, \ldots, s\}$.

The scalars required for normalization can be computed by

$$\begin{aligned}\beta_{sk+j+1} = \|\bar{u}_{sk+j+1}\|_2 &= \left(\bar{u}'^T_{k,j+1} (\mathcal{Y}_k^T \mathcal{Y}_k) \bar{u}'_{k,j+1}\right)^{1/2} = \left(\bar{u}'^T_{k,j+1} G_k^{(\mathcal{Y})} \bar{u}'_{k,j+1}\right)^{1/2} \quad \text{and} \\ \alpha_{sk+j+1} = \|\bar{v}_{sk+j+1}\|_2 &= \left(\bar{v}'^T_{k,j+1} (\mathcal{Z}_k^T \mathcal{Z}_k) \bar{v}'_{k,j+1}\right)^{1/2} = \left(\bar{v}'^T_{k,j+1} G_k^{(\mathcal{Z})} \bar{v}'_{k,j+1}\right)^{1/2},\end{aligned}$$

where $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k$ and $G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$. We then update

$$\begin{aligned}u'_{k,j+1} &= \bar{u}'_{k,j+1} / \beta_{sk+j+1} \quad \text{and} \\ v'_{k,j+1} &= \bar{v}'_{k,j+1} / \alpha_{sk+j+1}.\end{aligned}$$

The resulting communication-avoiding lower bidiagonalization algorithm is shown in Algorithm 20.

4.6 Least-Squares QR

Paige and Saunders [144] showed that the quantities generated by the lower bidiagonalization procedure in Algorithm 14 can be used to solve the least-squares problem $\min \|b - Ax\|_2$. We briefly review the rationale behind the least squares QR (LSQR) algorithm given by Paige and Saunders. For some vector y_i, define the quantities

$$\begin{aligned}x_i &= V_i y_i, \\ r_i &= b - A x_i, \quad \text{and} \\ t_{i+1} &= \beta_1 e_1 - B_i y_i.\end{aligned}$$

Algorithm 20 Communication-Avoiding Lower Bidiagonalization with Alternating Matrix Powers

Input: m-by-n matrix A and length-n starting vector b

Output: Matrices V_{sk+s}, U_{sk+s+1} and vector v_{sk+s+1} satisfying (4.71) and matrix B_{sk+s} satisfying (4.72)

$\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
for $k = 0, 1, \ldots$ until convergence **do**
    Compute $\mathcal{Z}_k$ and $\mathcal{Y}_k$ such that $A\underline{\underline{\mathcal{Z}}}_k = \mathcal{Y}_k\mathcal{B}_k$ and $A^T\mathcal{Y}_k = \mathcal{Z}_k\tilde{\mathcal{B}}_k$.
    $G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T\mathcal{Z}_k$, $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T\mathcal{Y}_k$
    $v'_{k,1} = e_1$, $u'_{k,1} = e_1$
    for $j = 1, \ldots, s$ **do**
        $\bar{u}'_{k,j+1} = \mathcal{B}_k v'_{k,j} - \alpha_{sk+j}u'_{k,j}$
        $\beta_{sk+j+1} = \left(\bar{u}'^T_{k,j+1}G_k^{(\mathcal{Y})}\bar{u}'_{k,j+1}\right)^{1/2}$
        $u'_{k,j+1} = \bar{u}'_{k,j+1}/\beta_{sk+j+1}$
        $\bar{v}'_{k,j+1} = \tilde{\mathcal{B}}_k u'_{k,j+1} - \beta_{sk+j+1}v'_{k,j}$
        $\alpha_{sk+j+1} = \left(\bar{v}'^T_{k,j+1}G_k^{(\mathcal{Z})}\bar{v}'_{k,j+1}\right)^{1/2}$
        $v'_{k,j+1} = \bar{v}'_{k,j+1}/\alpha_{sk+j+1}$
    end for
    Recover $\{u_{sk+j+1}, v_{sk+j+1}\}$ for $j \in \{1, \ldots, s\}$ according to (4.82)
end for

Since for the lower bidiagonalization procedure we have $U_{i+1}(\beta_1 e_1) = b$ and $AV_i = U_{i+1}B_i$, it follows that $r_i = U_{i+1}t_{i+1}$, and since U_{i+1} is orthonormal, this suggests choosing y_i such that $\|t_{i+1}\|_2$ is minimized, which gives the least-squares problem $\min \|\beta_1 e_1 - B_i y_i\|_2$.

This problem is solved by updating the QR factorization of B_i in each iteration, given by

$$Q_i[B_i \quad \beta_1 e_1] = \begin{bmatrix} R_i & f_i \\ & \tilde{\phi}_{i+1} \end{bmatrix},$$

where R_i is the upper bidiagonal matrix produced by Algorithm 13. (Coincidentally, this factorization provides a link between the two bidiagonalization procedures; see [144]). Above, Q_i is the product of a series of plane rotations, i.e., $Q_i = Q_{i,i+1}\cdots Q_{2,3}Q_{1,2}$. We then have

$$x_i = V_i R_i^{-1} f_i = D_i f_i,$$

where the columns of D_i can be found successively by forward substitution on the system $R_i^T D_i^T = V_i^T$. This gives

$$\begin{aligned} d_i &= (1/\rho_i)(v_i - \theta_i d_{i-1}) \quad \text{and} \\ x_i &= x_{i-1} + \phi_i d_i, \end{aligned}$$

where $d_0 = x_0 = 0$.

The QR factorization is determined by constructing the ith plane rotation $Q_{i,i+1}$ to operate on rows i and $i+1$ of the transformed $[B_i \quad \beta_1 e_1]$ and eliminate β_{i+1}. This recurrence relation can be written

$$\begin{bmatrix} c_i & s_i \\ s_i & -c_i \end{bmatrix} \begin{bmatrix} \bar{\rho}_i & 0 & \bar{\phi}_i \\ \beta_{i+1} & \alpha_{i+1} & 0 \end{bmatrix} = \begin{bmatrix} \rho_i & \theta_{i+1} & \phi_i \\ 0 & \bar{\rho}_{i+1} & \bar{\phi}_{i+1} \end{bmatrix},$$

where $\bar{\rho}_1 = \alpha_1$, $\bar{\phi}_1 = \beta_1$, and c_i and s_i are the elements of $Q_{i,i+1}$. Note that s without a subscript still denotes the iteration blocking factor. In the algorithm, vectors $w_i = \rho_i d_i$ are computed instead of d_i. As in the previous section, quantities with bars denote intermediate variables.

Thus, the LSQR algorithm proceeds as follows. One begins by setting

$$\bar{\phi}_1 = \beta_1, \quad \bar{\rho}_1 = \alpha_1, \quad w_1 = v_1, \quad \text{and} \quad x_1 = 0_{n,1},$$

and proceeds with the Lanczos lower bidiagonalization process (Algorithm 14). In each iteration, after β_{i+1}, α_{i+1}, and v_{i+1} have been computed via the bidiagonalization process, one updates

$$\begin{aligned} \rho_i &= (\bar{\rho}_i^2 + \beta_{i+1}^2)^{1/2}, \\ c_i &= \bar{\rho}_i / \rho_i, \\ s_i &= \beta_{i+1} / \rho_i, \\ \theta_{i+1} &= s_i \alpha_{i+1}, \\ \bar{\rho}_{i+1} &= -c_i \alpha_{i+1}, \\ \phi_i &= c_i \bar{\phi}_i, \\ \bar{\phi}_{i+1} &= s_i \bar{\phi}_i, \\ x_{i+1} &= x_i + \frac{\phi_i}{\rho_i} w_i, \quad \text{and} \\ w_{i+1} &= v_{i+1} - \frac{\theta_{i+1}}{\rho_i} w_i. \end{aligned}$$

The resulting algorithm is shown in Algorithm 21. Any of the communication-avoiding variants of the lower bidiagonalization algorithm given in Section 4.5.2 can be adapted to give a communication-avoiding version of LSQR. In Algorithm 23 we show a CA-LSQR method based on the implementation in Algorithm 18. For reference, we give the intermediate step in obtaining this new method, a classical LSQR algorithm which uses auxiliary quantities, in Algorithm 22. In Algorithm 24, we give a CA-LSQR method using the alternating matrix powers approach of the bidiagonalization in Algorithm 20. Note that the LSQR method, and thus our CA-LSQR methods, do not involve a vector representing the residual. The norm of the residual, however, is available for free: $\|r_i\| = \bar{\phi}_{i+1} = \beta_1 s_i s_{i-1} \cdots s_1$.

Algorithm 21 Least-Squares QR (LSQR)

Input: m-by-n matrix A and length-n starting vector b
Output: Approximate solution x_{i+1} to $\min \|b - Ax\|_2$

1: $\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
2: $\bar{u}_1 = A^T u_1$, $\bar{v}_1 = A v_1$
3: $\bar{\phi}_1 = \beta_1$, $\bar{\rho}_1 = \alpha_1$, $w_1 = v_1$, $x_1 = 0_{n,1}$
4: **for** $i = 1, 2, \ldots$ until convergence **do**
5: $\quad \bar{u}_{i+1} = A v_i - \alpha_i u_i$
6: $\quad \beta_{i+1} = \left(\bar{u}_{i+1}^T \bar{u}_{i+1}\right)^{1/2}$
7: $\quad u_{i+1} = \bar{u}_{i+1}/\beta_{i+1}$
8: $\quad \bar{v}_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$
9: $\quad \alpha_{i+1} = \left(\bar{v}_{i+1}^T \bar{v}_{i+1}\right)^{1/2}$
10: $\quad v_{i+1} = \bar{v}_{i+1}/\alpha_{i+1}$
11: $\quad \rho_i = (\bar{\rho}_i^2 + \beta_{i+1}^2)^{1/2}$
12: $\quad c_i = \bar{\rho}_i/\rho_i, \qquad s_i = \beta_{i+1}/\rho_i$
13: $\quad \theta_{i+1} = s_i \alpha_{i+1}, \qquad \bar{\rho}_{i+1} = -c_i \alpha_{i+1}$
14: $\quad \phi_i = c_i \bar{\phi}_i, \qquad \bar{\phi}_{i+1} = s_i \bar{\phi}_i$
15: $\quad x_{i+1} = x_i + (\phi_i/\rho_i)\, w_i$
16: $\quad w_{i+1} = v_{i+1} - (\theta_{i+1}/\rho_i)\, w_i$
17: **end for**

4.7 Conclusions and Future Work

In this Chapter, we derived a number of new CA-KSMs, including a general nonsymmetric Lanczos method, based on the 'BIOC' variant of nonsymmetric Lanczos of Gutknecht (see [94]), nonsymmetric Lanczos-based CA-KSMs for solving linear systems including CA-BICG, CA-CGS, and CA-BICGSTAB, and a number of new communication avoiding variants of the upper and lower Lanczos bidiagonalization procedures, which form the basis for the CA-LSQR method. The communication-avoiding versions enable $\Theta(s)$-fold savings in serial and parallel latency and serial bandwidth.

The CA-KSMs that we derived in this Chapter are based on coupled two-term recurrences. For these methods, communication can be further reduced by using a variant of the communication-avoiding matrix powers kernel (Section 3.2) that does k sparse matrix-matrix multiplications (SpMMs) instead of k SpMVs in order to simultaneously compute more than one Krylov subspace as required in each outer loop. Such an optimized kernel could also be used to create CA-Block KSM versions of our methods, to allow for solving multiple systems simultaneously.

There are a number of Krylov methods for non-Hermitian systems for which a communication-avoiding variant does not yet exist. The BICGSTAB(L) method [160] is a promising direction for future work, especially once combined with the communication-avoiding tall-skinny QR kernel, as described in [102]. Also of interest are the 'Multiple Lanczos' (ML)

Algorithm 22 LSQR with Auxiliary Quantities

Input: m-by-n matrix A and length-n starting vector b
Output: Approximate solution x_{i+1} to $\min \|b - Ax\|_2$

$\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$
$\tilde{u}_1 = A^T u_1$, $\tilde{v}_1 = A v_1$
$\bar{\phi}_1 = \beta_1, \quad \bar{\rho}_1 = \alpha_1, \quad w_1 = v_1, \quad x_1 = 0_{n,1}$
for $i = 1, 2, \ldots$ until convergence **do**
    $\bar{u}_{i+1} = \tilde{v}_i - \alpha_i u_i$
    $\beta_{i+1} = \left(\bar{u}_{i+1}^T \bar{u}_{i+1}\right)^{1/2}$
    $u_{i+1} = \bar{u}_{i+1}/\beta_{i+1}$
    $\tilde{u}_{i+1} = (A^T A v_i - \alpha_i \tilde{u}_i)/\beta_{i+1}$
    $\bar{v}_{i+1} = \tilde{u}_{i+1} - \beta_{i+1} v_i$
    $\alpha_{i+1} = \left(\bar{v}_{i+1}^T \bar{v}_{i+1}\right)^{1/2}$
    $v_{i+1} = \bar{v}_{i+1}/\alpha_{i+1}$
    $\tilde{v}_{i+1} = (A A^T u_{i+1} - \beta_{i+1} \tilde{v}_i)/\alpha_{i+1}$
    $\rho_i = (\bar{\rho}_i^2 + \beta_{i+1}^2)^{1/2}$
    $c_i = \bar{\rho}_i/\rho_i, \qquad s_i = \beta_{i+1}/\rho_i$
    $\theta_{i+1} = s_i \alpha_{i+1}, \qquad \bar{\rho}_{i+1} = -c_i \alpha_{i+1}$
    $\phi_i = c_i \bar{\phi}_i, \qquad \bar{\phi}_{i+1} = s_i \bar{\phi}_i$
    $x_{i+1} = x_i + (\phi_i/\rho_i)\, w_i$
    $w_{i+1} = v_{i+1} - (\theta_{i+1}/\rho_i)\, w_i$
end for

variants of BICG and BICGSTAB, which use multiple left shadow residuals [201], induced dimension reduction (IDR) methods and IDR(s) variants [167, 187], and quasi-minimal residual (QMR) methods [71, 72]. Of course, it is also of interest to develop efficient preconditioners for these methods; we discuss progress and ongoing efforts in Section 6.8.

While we have focused on algorithms rather than implementation details in this Chapter, many non-trivial implementation decisions are required for the kernels used in CA-KSMs. The savings in practice depend on the size and nonzero structure of the linear system A, the machine parameters for each level of the memory/network hierarchy, and the value of s. Optimizations required for performance are thus both are machine and matrix dependent, which makes auto-tuning and code generation an attractive approach. There are many ongoing efforts with this goal; see, e.g., [26, 114]. By analytical and empirical study of the convergence and performance properties of CA-KSMs, we can identify problems and machines for which CA-KSMs are competitive in solving practical problems, allowing the design of effective auto-tuners and integration of CA-KSM codes into existing frameworks.

The numerical experiments for CA-BICG and CA-BICGSTAB presented in this chapter demonstrate how the convergence rate and attainable accuracy of CA-KSMs can differ from their classical counterparts. It is further evident that the extent to which they differ depends heavily on the choice of polynomials used in constructing the s-step bases and the blocking

Algorithm 23 Communication-Avoiding LSQR (CA-LSQR)

Input: m-by-n matrix A and length-n starting vector b

Output: Approximate solution x_{sk+s+1} to $\min \|b - Ax\|_2$

$\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$

$\tilde{u}_1 = A^T u_1$, $\tilde{v}_1 = A v_1$

$\bar{\phi}_1 = \beta_1$, $\bar{\rho}_1 = \alpha_1$, $w_1 = v_1$, $x_1 = 0_{n,1}$

for $k = 0, 1, \ldots$ until convergence **do**

  Compute $\mathcal{U}_k$, a basis for $\mathcal{K}_s(AA^T, u_{sk+1})$, $\tilde{\mathcal{U}}_k$, a basis for $\mathcal{K}_s(A^T A, A^T u_{sk+1})$, $\mathcal{V}_k$, a basis for $\mathcal{K}_{s+1}(A^T A, v_{sk+1})$, and $\tilde{\mathcal{V}}_k$, a basis for $\mathcal{K}_s(AA^T, Av_{sk+1})$. Let $\mathcal{Y}_k = [\mathcal{U}_k, \tilde{\mathcal{V}}_k]$, $\mathcal{Z}_k = [\mathcal{V}_k, \tilde{\mathcal{U}}_k]$.

  $G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T \mathcal{Y}_k, \qquad G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T \mathcal{Z}_k$

  $u'_{k,1} = e_1$, $v'_{k,1} = e_1$, $\tilde{u}'_{k,1} = e_{s+2}$, $\tilde{v}'_{k,1} = e_{s+1}$.

  $w'_{k,1} = [0_{1,2s+1}, 1]^T$, $x'_{k,1} = 0_{2s+2,1}$

  for $j = 1, \ldots, s$ **do**

    $\bar{u}'_{k,j+1} = \tilde{v}'_{k,j} - \alpha_{sk+j} u'_{k,j}$

    $\beta_{sk+j+1} = \left(\bar{u}'^T_{k,j+1} G_k^{(\mathcal{Y})} \bar{u}'_{k,j+1} \right)^{1/2}$

    $u'_{k,j+1} = \bar{u}'_{k,j+1} / \beta_{sk+j+1}$

    $\tilde{u}'_{k,j+1} = (\mathcal{B}_k^{(\mathcal{Z})} v'_{k,j} - \alpha_{sk+j} \tilde{u}'_{k,j}) / \beta_{sk+j+1}$

    $\bar{v}'_{k,j+1} = \tilde{u}'_{k,j+1} - \beta_{sk+j+1} v'_{k,j}$

    $\alpha_{sk+j+1} = \left(\bar{v}'^T_{k,j+1} G_k^{(\mathcal{Z})} \bar{v}'_{k,j+1} \right)^{1/2}$

    $v'_{k,j+1} = \bar{v}'_{k,j+1} / \alpha_{sk+j+1}$

    $\tilde{v}'_{k,j+1} = (\mathcal{B}_k^{(\mathcal{Y})} u'_{k,j+1} - \beta_{sk+j+1} \tilde{v}'_{k,j}) / \alpha_{sk+j+1}$

    $\rho_{sk+j} = (\bar{\rho}_{sk+j}^2 + \beta_{sk+j+1}^2)^{1/2}$

    $c_{sk+j} = \bar{\rho}_{sk+j} / \rho_{sk+j}, \qquad s_{sk+j} = \beta_{sk+j+1} / \rho_{sk+j}$

    $\theta_{sk+j+1} = s_{sk+j} \alpha_{sk+j+1}, \qquad \bar{\rho}_{sk+j+1} = -c_{sk+j} \alpha_{sk+j+1}$

    $\phi_{sk+j} = c_{sk+j} \bar{\phi}_{sk+j}, \qquad \bar{\phi}_{sk+j+1} = s_{sk+j} \bar{\phi}_{sk+j}$

    $x'_{k,j+1} = x'_{k,j} + (\phi_{sk+j} / \rho_{sk+j})\, w'_{k,j}$

    $w'_{k,j+1} = [v'^T_{j+1}, 0]^T - (\theta_{sk+j+1} / \rho_{sk+j})\, w'_{k,j}$

  end for

  Recover $\{u_{sk+j+1}, v_{sk+j+1}\}$ for $j \in \{1, \ldots, s\}$ according to (4.76)

  $x_{sk+s+1} = [\mathcal{Z}_k, w_{sk+1}] x'_{k,s+1} + x_{sk+1}$

  $w_{sk+s+1} = [\mathcal{Z}_k, w_{sk+1}] w'_{k,s+1}$

  $[u_{sk+2}, \ldots, u_{sk+s+1}] = \mathcal{Y}_k [u'_{k,2}, \ldots, u'_{k,s+1}]$

end for

Algorithm 24 CA-LSQR with Alternating Matrix Powers

Input: m-by-n matrix A and length-n starting vector b

Output: Approximate solution x_{sk+s+1} to $\min \|b - Ax\|_2$

$\beta_1 = \|b\|_2$, $u_1 = b/\beta_1$, $\bar{v}_1 = A^T u_1$, $\alpha_1 = \|\bar{v}_1\|_2$, $v_1 = \bar{v}_1/\alpha_1$

$\bar{\phi}_1 = \beta_1$, $\bar{\rho}_1 = \alpha_1$, $w_1 = v_1$, $x_1 = 0_{n,1}$

for $k = 0, 1, \ldots$ until convergence **do**

  Compute $\mathcal{Z}_k$ and $\mathcal{Y}_k$ such that $A\underline{\underline{\mathcal{Z}}}_k = \mathcal{Y}_k\mathcal{B}_k$ and $A^T\mathcal{Y}_k = \mathcal{Z}_k\tilde{\mathcal{B}}_k$.

  $G_k^{(\mathcal{Z})} = \mathcal{Z}_k^T\mathcal{Z}_k, \quad G_k^{(\mathcal{Y})} = \mathcal{Y}_k^T\mathcal{Y}_k$

  $v'_{k,1} = e_1$, $u'_{k,1} = e_1$

  $w'_{k,1} = [0_{1,2s+1}, 1]^T$, $x'_{k,1} = 0_{2s+2,1}$

  for $j = 1, \ldots, s$ **do**

    $\bar{u}'_{k,j+1} = \mathcal{B}_k v'_{k,j} - \alpha_{sk+j} u'_{k,j}$

    $\beta_{sk+j+1} = \left(\bar{u}'^T_{k,j+1} G_k^{(\mathcal{Y})} \bar{u}'_{k,j+1}\right)^{1/2}$

    $u'_{k,j+1} = \bar{u}'_{k,j+1}/\beta_{sk+j+1}$

    $\bar{v}'_{k,j+1} = \tilde{\mathcal{B}}_k u'_{k,j+1} - \beta_{sk+j+1} v'_{k,j}$

    $\alpha_{sk+j+1} = \left(\bar{v}'^T_{k,j+1} G_k^{(\mathcal{Z})} \bar{v}'_{k,j+1}\right)^{1/2}$

    $v'_{k,j+1} = \bar{v}'_{k,j+1}/\alpha_{sk+j+1}$

    $\rho_{sk+j} = (\bar{\rho}_{sk+j}^2 + \beta_{sk+j+1}^2)^{1/2}$

    $c_{sk+j} = \bar{\rho}_{sk+j}/\rho_{sk+j}, \qquad s_{sk+j} = \beta_{sk+j+1}/\rho_{sk+j}$

    $\theta_{sk+j+1} = s_{sk+j}\alpha_{sk+j+1}, \qquad \bar{\rho}_{sk+j+1} = -c_{sk+j}\alpha_{sk+j+1}$

    $\phi_{sk+j} = c_{sk+j}\bar{\phi}_{sk+j}, \qquad \bar{\phi}_{sk+j+1} = s_{sk+j}\bar{\phi}_{sk+j}$

    $x'_{k,j+1} = x'_{k,j} + (\phi_{sk+j}/\rho_{sk+j})\, w'_{k,j}$

    $w'_{k,j+1} = [v'^T_{k,j+1}, 0]^T - (\theta_{sk+j+1}/\rho_{sk+j})\, w'_{k,j}$

  end for

  Recover $\{u_{sk+j+1}, v_{sk+j+1}\}$ for $j \in \{1, \ldots, s\}$ according to (4.82)

  $x_{sk+s+1} = [\mathcal{Z}_k, w_{sk+1}]x'_{k,s+1} + x_{sk+1}$

  $w_{sk+s+1} = [\mathcal{Z}_k, w_{sk+1}]w'_{k,s+1}$

end for

parameter s. In the next chapter, we extend a number of bounds on the stability and convergence properties of finite precision KSMs to finite precision CA-KSMs. Our theoretical results explain the role of basis conditioning on the finite precision behavior of CA-KSMs.

Chapter 5

Communication-Avoiding Krylov Subspace Methods in Finite Precision

Recall from the discussion in Section 2.3 that roundoff error has two discernible effects in KSMs - loss of accuracy and delay of convergence. Although CA-KSMs are mathematically equivalent to their classical counterparts in exact arithmetic, their finite precision behavior may differ significantly. It has been observed that the behavior of CA-KSMs deviates further from that of the classical method as s increases and the severity of this deviation is heavily influenced by the polynomials used for the s-step Krylov bases (see, e.g., [14, 35, 102, 104]).

As this can hinder usability of these methods in practice, a better understanding of the numerical properties of CA-KSMs is necessary. In this Chapter, we extend many results on convergence and accuracy for finite precision KSMs to finite precision CA-KSMs. We give bounds on the accuracy and convergence of eigenvalues in the finite precision CA-Lanczos method, a bound for the residual norm computed by the finite precision CA-(BI)CG algorithm, and a computable bound for the maximum attainable accuracy in finite precision CA-(BI)CG.

By sacrificing some tightness, we show that these bounds can all be written in the same form as the corresponding bounds for the classical method multiplied by an amplification factor that depends on the condition number of the s-step bases generated at the beginning of each outer loop. The bounds in this form are valuable for giving insight into the behavior of finite precision CA-KSMs. These bounds theoretically confirm the importance of basis conditioning on CA-KSM behavior and explain why the use of Newton and Chebyshev bases enable CA-KSMs to behave numerically very closely to the corresponding classical KSMs. In fact, if one can maintain a modest upper bound on the condition number of the s-step bases throughout the iteration, then one can with certainty expect close to the same convergence rate and attainable accuracy as the classical method.

The remainder of this chapter is outlined as follows. In Section 5.1, which has been adapted from [32], we present the first complete rounding error analysis of the CA-Lanczos method. We provide upper bounds on the normality of and orthogonality between the computed Lanczos vectors as well as a recurrence for the loss of orthogonality. We use this

analysis to extend the results of Paige for classical Lanczos to the CA-Lanczos method. Our analysis here of CA-Lanczos very closely follows Paige's rounding error analysis for classical Lanczos [140], and the theorems on accuracy and convergence of eigenvalues presented here follow from [141]. In Section 5.1.6 we show numerical experiments for a diagonal test problem which confirm the validity of our bounds and demonstrate the impact of the basis condition number on our error bounds. Our results suggest potential methods for improving convergence and accuracy through the inexpensive monitoring of quantities generated during the iterations.

In Section 5.2, we extend the analysis of Tong and Ye [176], to the CA-(BI)CG algorithm (see Section 4.2), which gives a bound for the residual norm computed by the finite precision CA-(BI)CG algorithm in terms of the residual norm of exact GMRES applied to a perturbed matrix, multiplied by an amplification factor. In the process, we construct the equivalent matrix form of the finite precision CA-(BI)CG recurrence, obtained by translating results from the s-step blocks to columns in the recurrence. This works also appears in the technical report [29].

Finally, in Section 5.3, we prove a computable bound for the maximum attainable accuracy (i.e., the deviation of the true and updated residual) in finite precision CA-(BI)CG, and show that this bound can be iteratively updated without asymptotically increasing computation or communication costs. In the subsequent chapter, we use this maximum attainable accuracy bound in a communication-avoiding residual replacement strategy for improving agreement between residuals. This work has been adapted from [31].

5.1 Analysis of the CA-Lanczos Method

Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$ and a starting vector $v_1 \in \mathbb{R}^n$ with unit 2-norm, i steps of the Lanczos method [115] theoretically produce the orthonormal matrix $V_i = [v_1, \ldots, v_i]$ and the symmetric tridiagonal matrix $T_i \in \mathbb{R}^{i\times i}$ such that

$$AV_i = V_iT_i + \beta_{i+1}v_{i+1}e_i^T. \tag{5.1}$$

When $i = n$, if T_n exists (i.e., no breakdown occurs), then the eigenvalues of T_n are the eigenvalues of A. In practice, some of the eigenvalues of T_i are good approximations to the eigenvalues of A when $i \ll n$, which makes the Lanczos method attractive as an iterative procedure. Many Krylov subspace methods (KSMs), including those for solving linear systems and least squares problems, are based on the Lanczos method. Such Lanczos-based methods are the core components in many applications.

Although theoretically the Lanczos process described by (5.1) produces an orthogonal basis and a tridiagonal matrix similar to A after n steps, these properties need not hold in finite precision. The most important work in the finite precision analysis of classical Lanczos is a series of papers published by Paige [138, 139, 140, 141]. Paige's analysis succinctly describes how rounding errors propagate through the algorithm to impede orthogonality. These

results were developed to give theorems which link the loss of orthogonality to convergence of the computed eigenvalues [141].

In this Section, we present the first complete rounding error analysis of the CA-Lanczos method. We provide upper bounds on the normality of and orthogonality between the computed Lanczos vectors as well as a recurrence for the loss of orthogonality. We use this analysis to extend the results of Paige for classical Lanczos to the CA-Lanczos method. Our analysis here of CA-Lanczos very closely follows Paige's rounding error analysis for classical Lanczos [140], and the theorems on accuracy and convergence of eigenvalues presented here follow from [141]. The derived bounds are very similar to those of Paige for classical Lanczos, but with the addition of an amplification term which depends on the maximum condition number of the Krylov bases computed at the start of each block of s steps. We show here that, based on restrictions on the size of this condition number, the same theorems of Paige apply to the s-step case.

Our results confirm that the conditioning of the Krylov bases plays a large role in determining finite precision behavior. Our analysis shows that if one can maintain modest condition numbers of the computed s-step Krylov bases throughout the iterations, the accuracy and convergence of eigenvalues in the CA-Lanczos method will be similar to those produced by classical Lanczos. This indicates that under certain restrictions, the CA-Lanczos method may be suitable for many practical problems.

The remainder of this section is outlined as follows. In Section 5.1.1, we present related work in the analysis of finite precision Lanczos. In Section 5.1.2, we review a variant of the Lanczos method and derive the corresponding CA-Lanczos method. In Section 5.1.3, we first state our main result in Theorem 1 and comment on its interpretation; the rest of the section is devoted to its proof. Sections 5.1.4 and 5.1.5 use the results of Paige [141] to give results on the accuracy and rate of convergence of the computed eigenvalues, respectively. We demonstrate these bounds through a number of numerical experiments in Section 5.1.6. Section 5.1.7 concludes with a discussion of future work. We note that this section has been adapted from work published in [32].

5.1.1 Related Work

Lanczos and others recognized early on that rounding errors could cause the Lanczos method to deviate from its ideal theoretical behavior. Since then, various efforts have been devoted to analyzing, explaining, and improving the finite precision Lanczos method.

Widely considered to be the most significant development is the series of papers by Paige discussed above. Another important result is due to Greenbaum, who performed a backward-like error analysis which showed that finite precision Lanczos and CG behave very similarly to the exact algorithms applied to any of a certain class of larger matrices [85]. Further explanation and examples are due to Greenbaum and Strakoš [88]. Paige has shown a similar type of augmented stability for the Lanczos process [142], and these results have recently been extended to the nonsymmetric case [143]. There are many other analyses of

Algorithm 25 Classical Lanczos

Input: n-by-n real symmetric matrix A and length-n starting vector v_1 such that $\|v_1\|_2 = 1$
Output: Matrices V_i and T_i and vector v_{i+1} satisfying (5.1)

1: $u_1 = Av_1$
2: **for** $i = 1, 2, \ldots$ until convergence **do**
3: $\quad \alpha_i = v_i^T u_i$
4: $\quad w_i = u_i - \alpha_i v_i$
5: $\quad \beta_{i+1} = \|w_i\|_2$
6: $\quad v_{i+1} = w_i / \beta_{i+1}$
7: $\quad u_{i+1} = Av_{i+1} - \beta_{i+1} v_i$
8: **end for**

the behavior of various KSMs in finite precision, including some more recent results due to Wülling [192] and Zemke [202]; for a thorough overview of the literature see [124, 125].

A number of strategies for maintaining orthogonality among the Lanczos vectors were inspired by the analysis of Paige, including selective reorthogonalization [147] and partial reorthogonalization [157]. Recently, Gustafsson et al. have extended such reorthogonalization strategies for classical Lanczos to the s-step case [92].

Section 5.1.2 of the present work includes the derivation of a new version of the CA-Lanczos method, equivalent in exact arithmetic to the variant used by Paige [140]. It uses a two-term recurrence like BIOC (see Section 4.1), but is restricted to the symmetric case and uses a different starting vector.

5.1.2 The CA-Lanczos Method

In our analysis we use the same variant of Lanczos used by Paige in [140] to allow easy comparison of results. Note that for simplicity, we assume no breakdown occurs, i.e., $\beta_{i+1} \neq 0$ for $i < n$, and thus breakdown conditions are not discussed here. We now give a derivation of CA-Lanczos, obtained from classical Lanczos in Algorithm 25. We note that, while in previous chapters, we indexed the length-$O(s)$ coordinate vectors by their inner loop iteration (e.g., v'_j), we now index these same vectors by both their outer and inner loop iterations (e.g., $v'_{k,j}$), as we need to distinguish them in the analysis.

After k blocks of s steps, consider iteration $i = sk+1$ where $k \in \mathbb{N}$ and $0 < s \in \mathbb{N}$. Using

$$v_{sk+1} \in \mathcal{K}_1(A, v_{sk+1}) \quad \text{and} \quad u_{sk+1} \in \mathcal{K}_1(A, u_{sk+1}),$$

it follows by induction on lines 6 and 7 of Algorithm 25 that for $j \in \{1, \ldots, s+1\}$,

$$\begin{aligned} v_{sk+j} &\in \mathcal{K}_s(A, v_{sk+1}) + \mathcal{K}_s(A, u_{sk+1}) \quad \text{and} \\ u_{sk+j} &\in \mathcal{K}_{s+1}(A, v_{sk+1}) + \mathcal{K}_{s+1}(A, u_{sk+1}), \end{aligned} \tag{5.2}$$

where $\mathcal{K}_\ell(A,x) = \text{span}\{x, Ax, \ldots, A^{\ell-1}x\}$ denotes the Krylov subspace of dimension ℓ of matrix A with respect to vector x. Since $\mathcal{K}_j(A,x) \subseteq \mathcal{K}_\ell(A,x)$ for $j \leq \ell$,

$$v_{sk+j}, u_{sk+j} \in \mathcal{K}_{s+1}(A, v_{sk+1}) + \mathcal{K}_{s+1}(A, u_{sk+1}) \quad \text{for} \quad j \in \{1, \ldots, s+1\}.$$

Note that since $u_1 = Av_1$, if $k = 0$ we have

$$v_j, u_j \in \mathcal{K}_{s+2}(A, v_1) \quad \text{for} \quad j \in \{1, \ldots, s+1\}.$$

For $k > 0$, we then define the 'basis matrix' $\mathcal{Y}_k = [\mathcal{V}_k, \mathcal{U}_k]$, where $\mathcal{V}_k$ and $\mathcal{U}_k$ are size n-by-$(s+1)$ matrices whose first j columns form bases for $\mathcal{K}_j(A, v_{sk+1})$ and $\mathcal{K}_j(A, u_{sk+1})$ respectively, for $j \in \{1, \ldots, s+1\}$. For $k = 0$, we define $\mathcal{Y}_0$ to be a size n-by-$(s+2)$ matrix whose (ordered) columns form a basis for $\mathcal{K}_{s+2}(A, v_1)$. Then by (5.2), we can represent v_{sk+j} and u_{sk+j}, for $j \in \{1, \ldots, s+1\}$, by their coordinates (denoted with primes) in $\mathcal{Y}_k$, i.e.,

$$v_{sk+j} = \mathcal{Y}_k v'_{k,j} \quad \text{and} \quad u_{sk+j} = \mathcal{Y}_k u'_{k,j}. \tag{5.3}$$

Note that for $k = 0$ the coordinate vectors $v'_{k,j}$ and $u'_{k,j}$ have length $s+2$ with zero elements beyond the j-th and $j+1$-st, respectively, and for $k > 0$, the coordinate vectors have length $(2s+2)$ with appropriate zero elements. We can write a similar equation for auxiliary vector w_{sk+j}, i.e., $w_{sk+j} = \mathcal{Y}_k w'_{k,j}$ for $j \in \{1, \ldots, s\}$. We also define the Gram matrix $G_k = \mathcal{Y}_k^T \mathcal{Y}_k$, which is size $(s+2)$-by-$(s+2)$ for $k = 0$ and $(2s+2)$-by-$(2s+2)$ for $k > 0$. Using this matrix the inner products in lines 3 and 5 can be written

$$\alpha_{sk+j} = v_{sk+j}^T u_{sk+j} = v_{k,j}'^T \mathcal{Y}_k^T \mathcal{Y}_k u'_{k,j} = v_{k,j}'^T G_k u'_{k,j} \quad \text{and} \tag{5.4}$$

$$\beta_{sk+j+1} = (w_{sk+j}^T w_{sk+j})^{1/2} = (w_{k,j}'^T \mathcal{Y}_k^T \mathcal{Y}_k w'_{k,j})^{1/2} = (w_{k,j}'^T G_k w'_{k,j})^{1/2}. \tag{5.5}$$

We assume that the bases are generated via polynomial recurrences represented by matrix $\mathcal{B}_k$, which is in general upper Hessenberg but often tridiagonal in practice. The recurrence can thus be written in matrix form as

$$A\underline{\mathcal{Y}}_k = \mathcal{Y}_k \mathcal{B}_k. \tag{5.6}$$

For $k = 0$, $\mathcal{B}_k$ is size $(s+2)$-by-$(s+2)$ and $\underline{\mathcal{Y}}_0$ is obtained by replacing the last column of $\mathcal{Y}_0$ by a zero column, i.e., $\underline{\mathcal{Y}}_0 = \left[\mathcal{Y}_0[I_{s+1}, 0_{s+1,1}]^T, 0_{n,1}\right]$. For $k > 0$, $\mathcal{B}_k$ is size $(2s+2)$-by-$(2s+2)$ and $\underline{\mathcal{Y}}_k$ is obtained by replacing columns $s+1$ and $2s+2$ of $\mathcal{Y}_k$ by zero columns, i.e., $\underline{\mathcal{Y}}_k = \left[\mathcal{V}_k[I_s, 0_{s,1}]^T, 0_{n,1}, \mathcal{U}_k[I_s, 0_{s,1}]^T, 0_{n,1}\right]$. Note that $\mathcal{B}_k$ has zeros in column $s+2$ when $k = 0$ and zeros in columns $s+1$ and $2s+2$ for $k > 0$. Therefore, using (5.3) and (5.6),

$$Av_{sk+j+1} = \mathcal{Y}_k \mathcal{B}_k v'_{k,j+1} \quad \text{for} \quad j \in \{1, \ldots, s\}. \tag{5.7}$$

Thus, to compute iterations $sk+2$ through $sk+s+1$ in CA-Lanczos, we first generate a basis matrix $\mathcal{Y}_k$ such that (5.6) holds and compute G_k from $\mathcal{Y}_k$. These operations can be performed for the communication cost of one iteration of the classical method (see Chapter 3).

Algorithm 26 CA-Lanczos

Input: n-by-n real symmetric matrix A and length-n starting vector v_1 such that $\|v_1\|_2 = 1$
Output: Matrices V_{sk+s} and T_{sk+s} and vector v_{sk+s+1} satisfying (5.1)

1: $u_1 = Av_1$
2: **for** $k = 0, 1, \ldots$ until convergence **do**
3: Compute $\mathcal{Y}_k$ with change of basis matrix $\mathcal{B}_k$ according to (5.7)
4: Compute $G_k = \mathcal{Y}_k^T \mathcal{Y}_k$
5: $v'_{k,1} = e_1$
6: **if** $k = 0$ **then**
7: $u'_{0,1} = \mathcal{B}_0 e_1$
8: **else**
9: $u'_{k,1} = e_{s+2}$
10: **end if**
11: **for** $j = 1, 2, \ldots, s$ **do**
12: $\alpha_{sk+j} = v'^T_{k,j} G_k u'_{k,j}$
13: $w'_{k,j} = u'_{k,j} - \alpha_{sk+j} v'_{k,j}$
14: $\beta_{sk+j+1} = (w'^T_{k,j} G_k w'_{k,j})^{1/2}$
15: $v'_{k,j+1} = w'_{k,j} / \beta_{sk+j+1}$
16: $v_{sk+j+1} = \mathcal{Y}_k v'_{k,j+1}$
17: $u'_{k,j+1} = \mathcal{B}_k v'_{k,j+1} - \beta_{sk+j+1} v'_{k,j}$
18: $u_{sk+j+1} = \mathcal{Y}_k u'_{k,j+1}$
19: **end for**
20: **end for**

Then updates to the length-n vectors can be performed by updating instead the length-$(2s+2)$ coordinates for those vectors in $\mathcal{Y}_k$. Inner products and multiplications with A become smaller operations which can be performed locally, as in (5.4), (5.5), and (5.7). The complete CA-Lanczos algorithm is presented in Algorithm 26. Note that in Algorithm 26 we show the computation of the length-n vectors by the change of basis operation (5.3) in each inner iteration (lines 16 and 18) for clarity, although these vectors play no part in the inner loop iteration updates. In practice, the basis change operation (5.3) can be performed on a block of coordinate vectors at the end of each outer loop to recover v_{sk+j} and u_{sk+j} if needed, for $j \in \{2, \ldots, s+1\}$.

5.1.3 The CA-Lanczos Method in Finite Precision

Throughout our analysis, we use a standard model of floating point arithmetic where we assume the computations are carried out on a machine with relative precision ϵ (see [83]). The analysis is first-order in ϵ, ignoring higher-order terms, which have negligible effect on our results. We also ignore underflow and overflow. Following Paige [140], we use the ϵ symbol to represent the relative precision as well as terms whose absolute values are bounded

by the relative precision.

We will model floating point computation using the following standard conventions (see, e.g., [83, §2.4]): for vectors $u, v \in \mathbb{R}^n$, matrices $A \in \mathbb{R}^{n\times m}$ and $G \in \mathbb{R}^{n\times n}$, and scalar α,

$$\begin{aligned}
fl(u - \alpha v) =& u - \alpha v - \delta w, && |\delta w| \leq (|u| + 2|\alpha v|)\epsilon,\\
fl(v^T u) =& (v + \delta v)^T u, && |\delta v| \leq n\epsilon |v|,\\
fl(Au) =& (A + \delta A)u, && |\delta A| \leq m\epsilon |A|,\\
fl(A^T A) =& A^T A + \delta E, && |\delta E| \leq n\epsilon |A^T||A|, \quad \text{and}\\
fl(u^T(Gv)) =& (u + \delta u)^T (G + \delta G)v, && |\delta u| \leq n\epsilon |u|, |\delta G| \leq n\epsilon |G|,
\end{aligned}$$

where $fl()$ represents the evaluation of the given expression in floating point arithmetic and terms with δ denote error terms. We decorate quantities computed in finite precision arithmetic with hats, e.g., if we are to compute the expression $\alpha = v^T u$ in finite precision we get $\hat{\alpha} = fl(v^T u)$.

We first prove the following lemma which will be useful in our analysis.

Lemma 1. *Assume we have a full rank matrix $Y \in \mathbb{R}^{n\times r}$, where $n \geq r$. Let Y^+ denote the pseudoinverse of Y, i.e., $Y^+ = (Y^T Y)^{-1} Y^T$. Then for any vector $x \in \mathbb{R}^r$,*

$$\| |Y||x| \|_2 \leq \| |Y| \|_2 \|x\|_2 \leq \Gamma \|Yx\|_2,$$

where $\Gamma = \left\|Y^+\right\|_2 \left\||Y|\right\|_2 \leq \sqrt{r}\left\|Y^+\right\|_2 \left\|Y\right\|_2$.

Proof. We have

$$\| |Y||x| \|_2 \leq \| |Y| \|_2 \|x\|_2 \leq \| |Y| \|_2 \|Y^+ Yx\|_2 \leq \| |Y| \|_2 \|Y^+\|_2 \|Yx\|_2 = \Gamma \|Yx\|_2.$$

We note that the term Γ can be thought of as a type of condition number for the matrix Y. In the analysis, we will apply the above lemma to the computed 'basis matrix' $\hat{\mathcal{Y}}_k$. We assume throughout that the generated bases $\hat{\mathcal{U}}_k$ and $\hat{\mathcal{V}}_k$ are numerically full rank. That is, all singular values of $\hat{\mathcal{U}}_k$ and $\hat{\mathcal{V}}_k$ are greater than $\epsilon n \cdot 2^{\lfloor \log_2 \theta_1 \rfloor}$ where θ_1 is the largest singular value of A. The results of this section are summarized in the following theorem:

Theorem 1. *Assume that Algorithm 26 is implemented in floating point with relative precision ϵ and applied for m steps to the n-by-n real symmetric matrix A with at most N nonzeros per row, starting with vector v_1 with $\|v_1\|_2 = 1$. Let $\sigma \equiv \|A\|_2$, $\theta\sigma = \||A|\|_2$ and $\tau_k\sigma = \||\mathcal{B}_k|\|_2$, where $\mathcal{B}_k$ is defined in (5.6). Let $\Gamma_k = \|\hat{\mathcal{Y}}_k^+\|_2 \||\hat{\mathcal{Y}}_k|\|_2$, where the superscript '+' denotes the Moore-Penrose pseudoinverse, i.e., $\hat{\mathcal{Y}}_k^+ = (\hat{\mathcal{Y}}_k^T \hat{\mathcal{Y}}_k)^{-1} \hat{\mathcal{Y}}_k^T$, and let*

$$\bar{\Gamma}_k = \max_{\ell \in \{0,\ldots,k\}} \Gamma_\ell \geq 1 \quad \textit{and} \quad \bar{\tau}_k = \max_{\ell \in \{0,\ldots,k\}} \tau_\ell. \tag{5.8}$$

Then for all $i \in \{1, \ldots, m\}$, $\hat{\alpha}_i$, $\hat{\beta}_{i+1}$, *and* $\hat{v}_{i+1}$ *will be computed such that*

$$A\hat{V}_m = \hat{V}_m \hat{T}_m + \hat{\beta}_{m+1}\hat{v}_{m+1}e_m^T - \delta\hat{V}_m, \tag{5.9}$$

where

$$\hat{V}_m = [\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_m] \tag{5.10}$$

$$\delta\hat{V}_m = [\delta\hat{v}_1, \delta\hat{v}_2, \ldots, \delta\hat{v}_m] \tag{5.11}$$

$$\hat{T}_m = \begin{bmatrix} \hat{\alpha}_1 & \hat{\beta}_2 & & \\ \hat{\beta}_2 & \ddots & \ddots & \\ & \ddots & \ddots & \hat{\beta}_m \\ & & \hat{\beta}_m & \hat{\alpha}_m \end{bmatrix} \tag{5.12}$$

with

$$\|\delta\hat{v}_i\|_2 \le \epsilon_1\sigma, \tag{5.13}$$

$$\hat{\beta}_{i+1}|\hat{v}_i^T\hat{v}_{i+1}| \le \epsilon_0\sigma, \tag{5.14}$$

$$|\hat{v}_{i+1}^T\hat{v}_{i+1} - 1| \le \epsilon_0/2, \quad \textit{and} \tag{5.15}$$

$$\left|\hat{\beta}_{i+1}^2 + \hat{\alpha}_i^2 + \hat{\beta}_i^2 - \|A\hat{v}_i\|_2^2\right| \le 2i(3\epsilon_0 + 2\epsilon_1)\sigma^2, \tag{5.16}$$

where

$$\epsilon_0 \equiv 2\epsilon(n+11s+15)\bar{\Gamma}_k^2 \quad \textit{and} \quad \epsilon_1 \equiv \epsilon\big((N+2s+5)\theta + (4s+9)\bar{\tau}_k + (10s+16)\big)\bar{\Gamma}_k. \tag{5.17}$$

Furthermore, if R_m *is the strictly upper triangular matrix such that*

$$\hat{V}_m^T\hat{V}_m = R_m^T + \operatorname{diag}(\hat{V}_m^T\hat{V}_m) + R_m, \tag{5.18}$$

then

$$\hat{T}_m R_m - R_m\hat{T}_m = \hat{\beta}_{m+1}\hat{V}_m^T\hat{v}_{m+1}e_m^T + \delta R_m, \tag{5.19}$$

where δR_m *is upper triangular with elements* η *such that*

$$\begin{aligned} |\eta_{1,1}| &\le \epsilon_0\sigma, \ \ \textit{and for } \ell \in \{2, \ldots, m\}, \\ |\eta_{\ell,\ell}| &\le 2\epsilon_0\sigma, \\ |\eta_{\ell-1,\ell}| &\le (\epsilon_0 + 2\epsilon_1)\sigma, \ \ \textit{and} \\ |\eta_{t,\ell}| &\le 2\epsilon_1\sigma, \textit{ for } t \in \{1, \ldots, \ell-2\}. \end{aligned} \tag{5.20}$$

Comments This generalizes Paige [140] as follows. The bounds in Theorem 1 give insight into how orthogonality is lost in the finite precision CA-Lanczos algorithm. Equation (5.13) bounds the error in the columns of the resulting perturbed Lanczos recurrence. How far the Lanczos vectors can deviate from unit 2-norm is given in (5.15), and (5.14) bounds how far adjacent vectors are from being orthogonal. The bound in (5.16) describes how close the columns of $A\hat{V}_m$ and $\hat{T}_m$ are in size. Finally, (5.19) can be thought of as a recurrence for the loss of orthogonality between Lanczos vectors, and shows how errors propagate through the iterations.

One thing to notice about the bounds in Theorem 1 is that they depend heavily on the term $\bar{\Gamma}_k$, which is a measure of the conditioning of the computed s-step Krylov bases. This indicates that if $\bar{\Gamma}_k$ is controlled in some way to be near constant, i.e., $\bar{\Gamma}_k = O(1)$, the bounds in Theorem 1 will be on the same order as Paige's analogous bounds for classical Lanczos [140], and thus we can expect orthogonality to be lost at a similar rate. The bounds also suggest that for the s-step variant to have any use, we must have $\bar{\Gamma}_k = o(\epsilon^{-1/2})$. Otherwise there can be no expectation of orthogonality.

We have sacrificed some tightness in the bounds in Theorem 1 in favor of simplified notation. Particularly, the use of $\bar{\Gamma}_k$ as defined in (5.8) in our bounds is likely to result in a large overestimate of the error (see Section 5.1.3.2 for numerical experiments). This causes our bounds for the $s = 1$ case to be larger than those of Paige for classical Lanczos. To obtain tighter bounds, one could instead use an alternative definition for $\bar{\Gamma}_k$; we discuss this and show numerical experiments in Section 5.1.3.2.

5.1.3.1 Proof of Theorem 1

The remainder of this section is dedicated to the proof of Theorem 1. To simplify the exposition, we have omitted some intermediate steps in the algebra; for the curious reader, a much longer analysis including the intermediate steps can be found in [29]. Also, in the interest of reducing overbearing subscripts, we use the indexing $i \equiv sk + j$ for quantities produced by both classical and CA-Lanczos (namely, the elements of the tridiagonal $\hat{T}_m$ and the length-n iteration vectors). We first proceed toward proving (5.15).

In finite precision, the Gram matrix construction in line 4 of Algorithm 26 becomes

$$\hat{G}_k = fl(\hat{\mathcal{Y}}_k^T \hat{\mathcal{Y}}_k) = \hat{\mathcal{Y}}_k^T \hat{\mathcal{Y}}_k + \delta G_k, \quad \text{where} \quad |\delta G_k| \leq \epsilon n |\hat{\mathcal{Y}}_k^T||\hat{\mathcal{Y}}_k|, \tag{5.21}$$

and line 14 of Algorithm 26 becomes $\hat{\beta}_{i+1} = fl\big(\, fl(\hat{w}_{k,j}'^T \hat{G}_k \hat{w}'_{k,j})^{1/2}\big)$. Let

$$d = fl(\hat{w}_{k,j}'^T \hat{G}_k \hat{w}'_{k,j}) = (\hat{w}_{k,j}'^T + \delta \hat{w}_{k,j}'^T)(\hat{G}_k + \delta \hat{G}_{k,w_j})\hat{w}'_{k,j},$$

where

$$|\delta \hat{w}'_{k,j}| \leq \epsilon(2s+2)|\hat{w}'_{k,j}| \quad \text{and} \quad |\delta \hat{G}_{k,w_j}| \leq \epsilon(2s+2)|\hat{G}_k|. \tag{5.22}$$

Remember that in the above equation we have ignored all terms of second order in ϵ. Now, we let $c = \hat{w}_{k,j}'^T \delta G_k \hat{w}'_{k,j} + \hat{w}_{k,j}'^T \delta \hat{G}_{k,w_j} \hat{w}'_{k,j} + \delta \hat{w}_{k,j}'^T \hat{G}_k \hat{w}'_{k,j}$, where

$$|c| \leq \epsilon(n+4s+4)\Gamma_k^2 \|\hat{\mathcal{Y}}_k \hat{w}'_{k,j}\|_2^2. \tag{5.23}$$

We can then write $d = \left\|\hat{\mathcal{Y}}_k \hat{w}'_{k,j}\right\|_2^2 (1 + c/\left\|\hat{\mathcal{Y}}_k \hat{w}'_{k,j}\right\|_2^2)$ and

$$\hat{\beta}_{i+1} = fl(\sqrt{d}) = \sqrt{d} + \delta\beta_{i+1} = \|\hat{\mathcal{Y}}_k \hat{w}'_{k,j}\|_2 \left(1 + \frac{c}{2\|\hat{\mathcal{Y}}_k \hat{w}'_{k,j}\|_2^2}\right) + \delta\beta_{i+1}, \tag{5.24}$$

where

$$|\delta\beta_{i+1}| \leq \epsilon\sqrt{d} = \epsilon\|\hat{\mathcal{Y}}_k \hat{w}'_{k,j}\|_2. \tag{5.25}$$

The coordinate vector $\hat{v}'_{k,j+1}$ is computed as

$$\hat{v}'_{k,j+1} = fl(\hat{w}'_{k,j}/\hat{\beta}_{i+1}) = (\hat{w}'_{k,j} + \delta\tilde{w}'_{k,j})/\hat{\beta}_{i+1}, \tag{5.26}$$

where

$$|\delta\tilde{w}'_{k,j}| \leq \epsilon|\hat{w}'_{k,j}|. \tag{5.27}$$

The corresponding Lanczos vector $\hat{v}_{i+1}$ (as well as $\hat{u}_{i+1}$) are recovered by a change of basis: in finite precision, we have

$$\hat{v}_{i+1} = fl(\hat{\mathcal{Y}}_k \hat{v}'_{k,j+1}) = \left(\hat{\mathcal{Y}}_k + \delta\hat{\mathcal{Y}}_{k,v_{j+1}}\right)\hat{v}'_{k,j+1}, \quad |\delta\hat{\mathcal{Y}}_{k,v_{j+1}}| \leq \epsilon(2s+2)|\hat{\mathcal{Y}}_k|, \tag{5.28}$$

and

$$\hat{u}_{i+1} = fl(\hat{\mathcal{Y}}_k \hat{u}'_{k,j+1}) = \left(\hat{\mathcal{Y}}_k + \delta\hat{\mathcal{Y}}_{k,u_{j+1}}\right)\hat{u}'_{k,j+1}, \quad |\delta\hat{\mathcal{Y}}_{k,u_{j+1}}| \leq \epsilon(2s+2)|\hat{\mathcal{Y}}_k|. \tag{5.29}$$

We can now prove (5.15) in Theorem 1. Using (5.24)-(5.26) and (5.28), as well as the bounds in (5.21)-(5.23), (5.28), (5.29), and Lemma 1, we obtain

$$\left|\hat{v}_{i+1}^T \hat{v}_{i+1} - 1\right| \leq \epsilon(n+8s+12)\Gamma_k^2 \leq \epsilon_0/2, \tag{5.30}$$

where ϵ_0 is defined in (5.17). This thus satisfies the bound (5.15).

We now proceed toward proving (5.14). Line 12 in Algorithm 26 is computed as

$$\hat{\alpha}_i = fl(\hat{v}_{k,j}'^T \hat{G}_k \hat{u}'_{k,j}) = (\hat{v}_{k,j}'^T + \delta\hat{v}_{k,j}'^T)(\hat{G}_k + \delta\hat{G}_{k,u_j})\hat{u}'_{k,j},$$

where $|\delta\hat{v}'_{k,j}| \leq \epsilon(2s+2)|\hat{v}'_{k,j}|$ and $|\delta\hat{G}_{k,u_j}| \leq \epsilon(2s+2)|\hat{G}_k|$. Expanding the above equation using (5.21), we obtain

$$\hat{\alpha}_i = \hat{v}_{k,j}'^T \hat{\mathcal{Y}}_k^T \hat{\mathcal{Y}}_k \hat{u}'_{k,j} + \hat{v}_{k,j}'^T \delta G_k \hat{u}'_{k,j} + \hat{v}_{k,j}'^T \delta\hat{G}_{k,u_j} \hat{u}'_{k,j} + \delta\hat{v}_{k,j}'^T \hat{G}_k \hat{u}'_{k,j},$$

and since by (5.28) and (5.29), $\hat{v}_i^T \hat{u}_i = \hat{v}_{k,j}'^T \hat{\mathcal{Y}}_k^T \hat{\mathcal{Y}}_k \hat{u}'_{k,j} + \hat{\mathcal{Y}}_k^T \delta\hat{\mathcal{Y}}_{k,u_j} \hat{u}'_{k,j} + \delta\hat{\mathcal{Y}}_{k,v_j}^T \hat{\mathcal{Y}}_k \hat{u}'_{k,j}$, we can write

$$\hat{\alpha}_i = \hat{v}_i^T \hat{u}_i + \delta\hat{\alpha}_i, \tag{5.31}$$

where $\delta\hat{\alpha}_i = \delta\hat{v}_{k,j}'^T \hat{G}_k \hat{u}'_{k,j} + \hat{v}_{k,j}'^T (\delta G_k + \delta\hat{G}_{k,u_j} - \hat{\mathcal{Y}}_k^T \delta\hat{\mathcal{Y}}_{k,u_j} - \delta\hat{\mathcal{Y}}_{k,v_j}^T \hat{\mathcal{Y}}_k)\hat{u}'_{k,j}$. Using Lemma 1 along with the bounds in (5.21)-(5.22) and (5.28)-(5.30),

$$|\delta\hat{\alpha}_i| \leq \epsilon(n+8s+8)\Gamma_k^2 \left\|\hat{u}_i\right\|_2, \tag{5.32}$$

and then using (5.31) and the bounds in (5.30) and (5.32), we obtain

$$|\hat{\alpha}_i| \leq \Big(1+\epsilon\big((3/2)n+12s+14\big)\Gamma_k^2\Big)\|\hat{u}_i\|_2. \tag{5.33}$$

In finite precision, line 13 of Algorithm 26 is computed as

$$\hat{w}'_{k,j} = \hat{u}'_{k,j} - \hat{\alpha}_i\hat{v}'_{k,j} - \delta w'_{k,j}, \quad \text{where} \quad |\delta w'_{k,j}| \leq \epsilon(|\hat{u}'_{k,j}| + 2|\hat{\alpha}_i\hat{v}'_{k,j}|). \tag{5.34}$$

Multiplying both sides of (5.34) by $\hat{\mathcal{Y}}_k$ gives $\hat{\mathcal{Y}}_k\hat{w}'_{k,j} = \hat{\mathcal{Y}}_k\hat{u}'_{k,j} - \hat{\alpha}_i\hat{\mathcal{Y}}_k\hat{v}'_{k,j} - \hat{\mathcal{Y}}_k\delta w'_{k,j}$, and multiplying each side by its own transpose and using (5.28) and (5.29),

$$\begin{aligned}\|\hat{\mathcal{Y}}_k\hat{w}'_{k,j}\|_2^2 = \|\hat{u}_i\|_2^2 - 2\hat{\alpha}_i\hat{u}_i^T\hat{v}_i + \hat{\alpha}_i^2\|\hat{v}_i\|_2^2 \\ - 2(\delta\hat{\mathcal{Y}}_{k,u_j}\hat{u}'_{k,j} - \hat{\alpha}_i\delta\hat{\mathcal{Y}}_{k,v_j}\hat{v}'_{k,j} + \hat{\mathcal{Y}}_k\delta w'_{k,j})^T(\hat{u}_i - \hat{\alpha}_i\hat{v}_i),\end{aligned}$$

where we have used $\hat{\mathcal{Y}}_k\hat{u}'_{k,j} - \hat{\alpha}_i\hat{\mathcal{Y}}_k\hat{v}'_{k,j} = \hat{u}_i - \hat{\alpha}_i\hat{v}_i + O(\epsilon)$. Now, using (5.31) and rearranging the above we obtain

$$\begin{aligned}\|\hat{\mathcal{Y}}_k\hat{w}'_{k,j}\|_2^2 + \hat{\alpha}_i^2 - \|\hat{u}_i\|_2^2 = \hat{\alpha}_i^2(\|\hat{v}_i\|_2^2 - 1) + 2\hat{\alpha}_i\delta\hat{\alpha}_i \\ - 2(\delta\hat{\mathcal{Y}}_{k,u_j}\hat{u}'_{k,j} - \hat{\alpha}_i\delta\hat{\mathcal{Y}}_{k,v_j}\hat{v}'_{k,j} + \hat{\mathcal{Y}}_k\delta w'_{k,j})^T(\hat{u}_i - \hat{\alpha}_i\hat{v}_i).\end{aligned}$$

Using Lemma 1 and the bounds in (5.28), (5.29), (5.30), (5.32), (5.33), and (5.34), we can then write the bound

$$\|\hat{\mathcal{Y}}_k\hat{w}'_{k,j}\|_2^2 + \hat{\alpha}_i^2 - \|\hat{u}_i\|_2^2 \leq \epsilon(3n+40s+56)\Gamma_k^2\|\hat{u}_i\|_2^2. \tag{5.35}$$

Given the above, we can also write

$$\|\hat{\mathcal{Y}}_k\hat{w}'_{k,j}\|_2^2 \leq \|\hat{\mathcal{Y}}_k\hat{w}'_{k,j}\|_2^2 + \hat{\alpha}_i^2 \leq \big(1+\epsilon(3n+40s+56)\Gamma_k^2\big)\|\hat{u}_i\|_2^2, \tag{5.36}$$

and using this with (5.23), (5.24), and (5.25),

$$|\hat{\beta}_{i+1}| \leq \big(1+\epsilon(2n+22s+31)\Gamma_k^2\big)\big\|\hat{u}_i\big\|_2. \tag{5.37}$$

Now, rearranging (5.26) and using (5.28), we can write

$$\hat{\beta}_{i+1}\hat{v}_{i+1} \equiv \hat{\mathcal{Y}}_k\hat{w}'_{k,j} + \delta w_i, \tag{5.38}$$

where $\delta w_i = \hat{\mathcal{Y}}_k\delta\tilde{w}'_{k,j} + \delta\hat{\mathcal{Y}}_{k,v_{j+1}}\hat{w}'_{k,j}$, and using Lemma 1 and the bounds in (5.27), (5.28), and (5.36),

$$\|\delta w_i\|_2 \leq \epsilon(2s+3)\Gamma_k\|\hat{u}_i\|_2. \tag{5.39}$$

We premultiply (5.38) by $\hat{v}_i^T$ and use (5.28), (5.29), (5.31), and (5.34) to obtain

$$\hat{\beta}_{i+1}\hat{v}_i^T\hat{v}_{i+1} = -\delta\hat{\alpha}_i - \hat{\alpha}_i(\|\hat{v}_i\|_2^2 - 1) - \hat{v}_i^T(\delta\hat{\mathcal{Y}}_{k,u_j}\hat{u}'_{k,j} - \hat{\alpha}_i\delta\hat{\mathcal{Y}}_{k,v_j}\hat{v}'_{k,j} + \hat{\mathcal{Y}}_k\delta w'_{k,j} - \delta w_i),$$

and using Lemma 1 and the bounds in (5.30), (5.26), (5.28), (5.29), (5.32), (5.33), (5.34), and (5.39), we can write the bound

$$\left|\hat{\beta}_{i+1}\hat{v}_i^T\hat{v}_{i+1}\right| \leq 2\epsilon(n+11s+15)\Gamma_k^2\|\hat{u}_i\|_2. \tag{5.40}$$

This is a start toward proving (5.14). We will return to the above bound once we later prove a bound on $\|\hat{u}_i\|_2$. Our next step is to analyze the error in each column of the finite precision CA-Lanczos recurrence. First, we note that we can write the finite precision recurrence for computing the s-step bases (line 3 in Algorithm 26) as

$$A\underline{\hat{\mathcal{Y}}}_k = \hat{\mathcal{Y}}_k\mathcal{B}_k + \delta E_k. \tag{5.41}$$

If the basis is computed by repeated matrix-vector products,

$$|\delta E_k| \leq \epsilon\big((3+N)|A||\underline{\hat{\mathcal{Y}}}_k| + 4|\hat{\mathcal{Y}}_k||\mathcal{B}_k|\big), \tag{5.42}$$

where N is the maximum number of nonzeros per row over all rows of A (see, e.g., [31]).

In finite precision, line 17 in Algorithm 26 is computed as

$$\hat{u}'_{k,j} = \mathcal{B}_k\hat{v}'_{k,j} - \hat{\beta}_i\hat{v}'_{k,j-1} + \delta u'_{k,j},\ |\delta u'_{k,j}| \leq \epsilon\big((2s+3)|\mathcal{B}_k||\hat{v}'_{k,j}| + 2|\hat{\beta}_i\hat{v}'_{k,j-1}|\big). \tag{5.43}$$

Then with Lemma 1, (5.28), (5.29), (5.41), and (5.43), we can write

$$\hat{u}_i = A\hat{v}_i - \hat{\beta}_i\hat{v}_{i-1} + \delta u_i, \tag{5.44}$$

where

$$\delta u_i = \hat{\mathcal{Y}}_k\delta u'_{k,j} - (A\delta\hat{\mathcal{Y}}_{k,v_j} - \delta\hat{\mathcal{Y}}_{k,u_j}\mathcal{B}_k + \delta E_k)\hat{v}'_{k,j} + \hat{\beta}_i(\delta\hat{\mathcal{Y}}_{k,v_{j-1}} - \delta\hat{\mathcal{Y}}_{k,u_j})\hat{v}'_{k,j-1},$$

and with the bounds in (5.28), (5.29), (5.37), (5.42), and (5.43),

$$\|\delta u_i\|_2 \leq \epsilon(N+2s+5)\,\|\,|A|\,\|_2\Gamma_k + \epsilon(4s+9)\,\|\,|\mathcal{B}_k|\,\|_2\Gamma_k + \epsilon(4s+6)\,\|\hat{u}_{i-1}\|_2\Gamma_k.$$

We will now introduce and make use of the quantities $\sigma \equiv \|A\|_2$, $\theta \equiv \|\,|A|\,\|_2/\sigma$, and $\tau_k \equiv \|\,|\mathcal{B}_k|\,\|_2/\sigma$. Note that the quantity $\|\,|\mathcal{B}_k|\,\|_2$ depends on the choice of polynomials used in constructing the Krylov bases. For the monomial basis, $\|\,|\mathcal{B}_k|\,\|_2 = 1$. For bases based on the spectrum of A, including Newton and Chebyshev bases, we expect that $\|\,|\mathcal{B}_k|\,\|_2 \lesssim \|\,|A|\,\|_2$ as long as the bases are constructed using sufficiently accurate spectral estimates. Using this notation the bound above can be written

$$\big\|\delta u_i\big\|_2 \leq \epsilon\Big(\big((N+2s+5)\theta + (4s+9)\tau_k\big)\sigma + (4s+6)\|\hat{u}_{i-1}\|_2\Big)\Gamma_k. \tag{5.45}$$

Now, manipulating (5.38) with (5.28), (5.29), and (5.34), we have

$$\hat{\beta}_{i+1}\hat{v}_{i+1} = \hat{u}_i - \hat{\alpha}_i\hat{v}_i - \delta\hat{\mathcal{Y}}_{k,u_j}\hat{u}'_{k,j} + \hat{\alpha}_i\delta\hat{\mathcal{Y}}_{k,v_j}\hat{v}'_{k,j} - \hat{\mathcal{Y}}_k\delta w'_{k,j} + \delta w_i,$$

and substituting in the expression for $\hat{u}_i$ in (5.44),

$$\hat{\beta}_{i+1}\hat{v}_{i+1} = A\hat{v}_i - \hat{\alpha}_i\hat{v}_i - \hat{\beta}_i\hat{v}_{i-1} + \delta\hat{v}_i, \tag{5.46}$$

where $\delta\hat{v}_i = \delta u_i - \delta\hat{\mathcal{Y}}_{k,u_j}\hat{u}'_{k,j} + \hat{\alpha}_i\delta\hat{\mathcal{Y}}_{k,v_j}\hat{v}'_{k,j} - \hat{\mathcal{Y}}_k\delta w'_{k,j} + \delta w_i$. Using Lemma 1, along with (5.27), (5.28), (5.29), (5.33), (5.34), and (5.39),

$$\|\delta\hat{v}_i\|_2 \leq \|\delta u_i\|_2 + \epsilon(6s+10)\Gamma_k\|\hat{u}_i\|_2,$$

and using (5.45), this bound becomes

$$\|\delta\hat{v}_i\|_2 \leq \epsilon\Big(\big((N+2s+5)\theta + (4s+9)\tau_k\big)\sigma + (6s+10)\|\hat{u}_i\|_2 + (4s+6)\|\hat{u}_{i-1}\|_2\Big)\Gamma_k. \tag{5.47}$$

We now have everything we need to write the finite-precision CA-Lanczos recurrence in its familiar matrix form. Let $\hat{V}_i$, $\delta\hat{V}_i$, and $\hat{T}_i$ be defined as in (5.10), (5.11), and (5.12), respectively. Then (5.46) in matrix form gives

$$A\hat{V}_i = \hat{V}_i\hat{T}_i + \hat{\beta}_{i+1}\hat{v}_{i+1}e_i^T - \delta\hat{V}_i. \tag{5.48}$$

Thus (5.47) gives a bound on the error in the columns of the finite precision CA-Lanczos recurrence. We will return to (5.47) to prove (5.13) once we bound $\|\hat{u}_i\|_2$.

Now, we examine the possible loss of orthogonality in the vectors $\hat{v}_1, \ldots, \hat{v}_{i+1}$. We define the strictly upper triangular matrix R_i of dimension i-by-i ($(sk+j)$-by-$(sk+j)$) with elements $\rho_{\ell,t}$, for $\ell, t \in \{1, \ldots, i\}$, such that $\hat{V}_i^T\hat{V}_i = R_i^T + \mathrm{diag}(\hat{V}_i^T\hat{V}_i) + R_i$. For notational purposes, we also define $\rho_{i,i+1} \equiv \hat{v}_i^T\hat{v}_{i+1}$. Multiplying (5.48) on the left by $\hat{V}_i^T$ and equating the right hand side by its own transpose, we obtain

$$\begin{aligned}\hat{T}_i(R_i^T + R_i) - (R_i^T + R_i)\hat{T}_i = \hat{\beta}_{i+1}(\hat{V}_i^T\hat{v}_{i+1}e_i^T - e_i\hat{v}_{i+1}^T\hat{V}_i) + \hat{V}_i^T\delta\hat{V}_i - \delta\hat{V}_i^T\hat{V}_i \\ + \mathrm{diag}(\hat{V}_i^T\hat{V}_i)\cdot\hat{T}_i - \hat{T}_i\cdot\mathrm{diag}(\hat{V}_i^T\hat{V}_i).\end{aligned}$$

Now, let $M_i \equiv \hat{T}_iR_i - R_i\hat{T}_i$, which is upper triangular and has dimension i-by-i. By definition,

$$\begin{aligned}m_{1,1} =& -\hat{\beta}_2\rho_{1,2}, \qquad m_{i,i} = \hat{\beta}_i\rho_{i-1,i}, \quad \text{and} \\ m_{\ell,\ell} =& \hat{\beta}_\ell\rho_{\ell-1,\ell} - \hat{\beta}_{\ell+1}\rho_{\ell,\ell+1}, \quad \text{for} \quad \ell \in \{2, \ldots, i-1\}.\end{aligned}$$

Therefore $M_i = \hat{\beta}_{i+1}\hat{V}_i^T\hat{v}_{i+1}e_i^T + \delta R_i$, where δR_i has elements satisfying

$$\begin{aligned}\eta_{1,1} =& -\hat{\beta}_2\rho_{1,2}, \quad \text{and for } \ell \in \{2, \ldots, i\}, \\ \eta_{\ell,\ell} =& \hat{\beta}_\ell\rho_{\ell-1,\ell} - \hat{\beta}_{\ell+1}\rho_{\ell,\ell+1}, \\ \eta_{\ell-1,\ell} =& \hat{v}_{\ell-1}^T\delta\hat{v}_\ell - \delta\hat{v}_{\ell-1}^T\hat{v}_\ell + \hat{\beta}_\ell(\hat{v}_{\ell-1}^T\hat{v}_{\ell-1} - \hat{v}_\ell^T\hat{v}_\ell), \ \text{and} \\ \eta_{t,\ell} =& \hat{v}_t^T\delta\hat{v}_\ell - \delta\hat{v}_t^T\hat{v}_\ell, \quad \text{where } t \in \{1, \ldots, \ell-2\}.\end{aligned} \tag{5.49}$$

To simplify notation, we introduce the quantities

$$\bar{u}_i = \max_{\ell\in\{1,\dots,i\}} \|\hat{u}_\ell\|_2, \quad \bar{\Gamma}_k = \max_{\ell\in\{0,\dots,k\}} \Gamma_\ell, \quad \text{and} \quad \bar{\tau}_k = \max_{\ell\in\{0,\dots,k\}} \tau_\ell.$$

Using this notation and (5.30), (5.37), (5.40), and (5.47), the quantities in (5.49) can be bounded by

$$\begin{aligned}
|\eta_{1,1}| &\le 2\epsilon(n+11s+15)\bar{\Gamma}_k^2\bar{u}_i, \text{ and for } \ell \in \{2,\dots,i\},\\
|\eta_{\ell,\ell}| &\le 4\epsilon(n+11s+15)\bar{\Gamma}_k^2\bar{u}_i,\\
|\eta_{\ell-1,\ell}| &\le 2\epsilon\bigg(\Big(\big((N+2s+5)\theta+(4s+9)\bar{\tau}_k\big)\sigma+(10s+16)\bar{u}_i\Big)\bar{\Gamma}_k+\Big(n+8s+12\Big)\bar{\Gamma}_k^2\bar{u}_i\bigg),\\
|\eta_{t,\ell}| &\le 2\epsilon\Big(\big((N+2s+5)\theta+(4s+9)\bar{\tau}_k\big)\sigma+(10s+16)\bar{u}_i\Big)\bar{\Gamma}_k, \text{ where } t\in\{1,\dots,\ell-2\}.
\end{aligned} \tag{5.50}$$

The above is a start toward proving (5.20). We return to this bound later, and now shift our focus towards proving a bound on $\|\hat{u}_i\|_2$. To proceed, we must first find a bound for $|\rho_{i-2,i}|$. We know the $(1,2)$ element of M_i is

$$\eta_{1,2} = \hat{\alpha}_1\rho_{1,2} - \hat{\alpha}_2\rho_{1,2} - \hat{\beta}_3\rho_{1,3},$$

and for $\ell > 2$, the $(\ell-1,\ell)$ element is

$$\eta_{\ell-1,\ell} = \hat{\beta}_{\ell-1}\rho_{\ell-2,\ell} + (\hat{\alpha}_{\ell-1}-\hat{\alpha}_\ell)\rho_{\ell-1,\ell} - \hat{\beta}_{\ell+1}\rho_{\ell-1,\ell+1}.$$

Then, defining $\xi_\ell \equiv (\hat{\alpha}_{\ell-1}-\hat{\alpha}_\ell)\hat{\beta}_\ell\rho_{\ell-1,\ell} - \hat{\beta}_\ell\eta_{\ell-1,\ell}$ for $\ell\in\{2,\dots,i\}$, we have

$$\hat{\beta}_\ell\hat{\beta}_{\ell+1}\rho_{\ell-1,\ell+1} = \hat{\beta}_{\ell-1}\hat{\beta}_\ell\rho_{\ell-2,\ell} + \xi_\ell = \xi_\ell + \xi_{\ell-1} + \ldots + \xi_2.$$

This, along with (5.33), (5.37), (5.40), and (5.50) gives

$$\begin{aligned}
\hat{\beta}_i\hat{\beta}_{i+1}|\rho_{i-1,i+1}| &\le \sum_{\ell=2}^{i}|\xi_\ell| \le \sum_{\ell=2}^{i}(|\hat{\alpha}_{\ell-1}|+|\hat{\alpha}_\ell|)|\hat{\beta}_\ell\rho_{\ell-1,\ell}| + |\hat{\beta}_\ell||\eta_{\ell-1,\ell}|\\
&\le 2\epsilon(i-1)\Big(\big((N+2s+5)\theta+(4s+9)\bar{\tau}_k\big)\sigma+(10s+16)\bar{u}_i\Big)\bar{\Gamma}_k\bar{u}_i\\
&\quad + 2\epsilon(i-1)\cdot 3\big(n+10s+14\big)\bar{\Gamma}_k^2\bar{u}_i^2.
\end{aligned} \tag{5.51}$$

Rearranging (5.44) gives $\hat{u}_i - \delta u_i = A\hat{v}_i - \hat{\beta}_i\hat{v}_{i-1}$, and multiplying each side by its own transpose (and ignoring all terms of second order in ϵ), we obtain

$$\hat{u}_i^T\hat{u}_i - 2\hat{u}_i^T\delta u_i = \|A\hat{v}_i\|_2^2 + \hat{\beta}_i^2\|\hat{v}_{i-1}\|_2^2 - 2\hat{\beta}_i\hat{v}_i^TA\hat{v}_{i-1}. \tag{5.52}$$

Rearranging (5.46) gives $A\hat{v}_{i-1} = \hat{\beta}_i\hat{v}_i + \hat{\alpha}_{i-1}\hat{v}_{i-1} + \hat{\beta}_{i-1}\hat{v}_{i-2} - \delta\hat{v}_{i-1}$, and premultiplying this expression by $\hat{\beta}_i\hat{v}_i^T$, we get

$$\hat{\beta}_i\hat{v}_i^TA\hat{v}_{i-1} = \hat{\beta}_i^2 + \delta\hat{\beta}_i, \tag{5.53}$$

where, using the bounds in (5.30), (5.33), (5.37), (5.40), (5.47), and (5.51),

$$|\delta\hat{\beta}_i| \leq \epsilon\big(2i-1\big)\Big(\big((N+2s+5)\theta+(4s+9)\bar{\tau}_k\big)\sigma+(10s+16)\bar{u}_i\Big)\bar{\Gamma}_k\bar{u}_i \\ +\epsilon\big(2i-1\big)\cdot 3\big(n+10s+14\big)\bar{\Gamma}_k^2\bar{u}_i^2. \tag{5.54}$$

Adding $2\hat{u}_i^T\delta u_i$ to both sides of (5.52) and using (5.53), we obtain

$$\|\hat{u}_i\|_2^2 = \|A\hat{v}_i\|_2^2+\hat{\beta}_i^2\big(\|\hat{v}_{i-1}\|_2^2-2\big)+\delta\tilde{\beta}_i, \tag{5.55}$$

where $\delta\tilde{\beta}_i = -2\delta\hat{\beta}_i + 2\hat{u}_i^T\delta u_i$, and, using the bounds in (5.45) and (5.54),

$$|\delta\tilde{\beta}_i| \leq 4i\epsilon\big((N+2s+5)\theta+(4s+9)\bar{\tau}_k\big)\sigma\bar{\Gamma}_k\bar{u}_i \\ +2\epsilon\Big(\big(2i-1\big)\big(3(n+10s+14)\bar{\Gamma}_k^2+(10s+16)\bar{\Gamma}_k\big)+(4s+6)\bar{\Gamma}_k\Big)\bar{u}_i^2. \tag{5.56}$$

Now, using (5.55), and since $\hat{\beta}_i^2 \geq 0$, we can write

$$\|\hat{u}_i\|_2^2 \leq \|\hat{u}_i\|_2^2+\hat{\beta}_i^2 \leq \sigma^2\|\hat{v}_i\|_2^2+\hat{\beta}_i^2\big(\|\hat{v}_{i-1}\|_2^2-1\big)+|\delta\tilde{\beta}_i|. \tag{5.57}$$

Let $\mu \equiv \max\big\{\bar{u}_i,\sigma\big\}$. We can now put the bounds in terms of ϵ_0 and ϵ_1, which are defined in (5.17). Then, using (5.57), along with the bounds in (5.30), (5.37), and (5.56),

$$\|\hat{u}_i\|_2^2 \leq \sigma^2+2i(3\epsilon_0+2\epsilon_1)\mu^2. \tag{5.58}$$

We consider the two possible cases for μ. First, if $\mu=\sigma$, then

$$\|\hat{u}_i\|_2^2 \leq \sigma^2\left(1+2i(3\epsilon_0+2\epsilon_1)\right).$$

Otherwise, we have the case $\mu=\bar{u}_i$. Since the bound in (5.58) holds for all $\|\hat{u}_i\|_2^2$, it also holds for $\bar{u}_i^2=\mu^2$, and thus, ignoring terms of second order in ϵ,

$$\mu^2 \leq \sigma^2+2i(3\epsilon_0+2\epsilon_1)\mu^2 \leq \sigma^2+2i(3\epsilon_0+2\epsilon_1)\sigma^2 \leq \sigma^2\left(1+2i(3\epsilon_0+2\epsilon_1)\right),$$

and, plugging this in to (5.58), we get

$$\|\hat{u}_i\|_2^2 \leq \sigma^2\left(1+2i(3\epsilon_0+2\epsilon_1)\right). \tag{5.59}$$

In either case we obtain the same bound on $\|\hat{u}_i\|_2^2$, so (5.59) holds.

Taking the square root of (5.59), we have

$$\|\hat{u}_i\|_2 \leq \sigma\left(1+i(3\epsilon_0+2\epsilon_1)\right), \tag{5.60}$$

and substituting (5.60) into (5.40), (5.47), and (5.50), we prove the bounds (5.14), (5.13), and (5.20) in Theorem 1, respectively, assuming that $i(3\epsilon_0+2\epsilon_1) \ll 1$.

The only remaining inequality to prove is (5.16). We first multiply both sides of (5.38) by their own transposes and then add $\hat{\alpha}_i^2 - \|\hat{u}_i\|_2^2$ to both sides to obtain

$$\hat{\beta}_{i+1}^2\|\hat{v}_{i+1}\|_2^2 + \hat{\alpha}_i^2 - \|\hat{u}_i\|_2^2 = \|\hat{\mathcal{Y}}_k\hat{w}'_{k,j}\|_2^2 + \hat{\alpha}_i^2 - \|\hat{u}_i\|_2^2 + 2\delta w_i^T\hat{\mathcal{Y}}_k\hat{w}'_{k,j}.$$

We then substitute in (5.55) on the left hand side, subtract $\hat{\beta}_{i+1}^2$ from both sides, and rearrange to obtain

$$\begin{aligned}\hat{\beta}_{i+1}^2 + \hat{\alpha}_i^2 + \hat{\beta}_i^2 - \|A\hat{v}_i\|_2^2 = \|\hat{\mathcal{Y}}_k\hat{w}'_{k,j}\|_2^2 + \hat{\alpha}_i^2 - \|\hat{u}_i\|_2^2 + 2\delta w_i^T\hat{\mathcal{Y}}_k\hat{w}'_{k,j} + \hat{\beta}_i^2(\|\hat{v}_{i-1}\|_2^2 - 1)\\ - \hat{\beta}_{i+1}^2(\|\hat{v}_{i+1}\|_2^2 - 1) + \delta\tilde{\beta}_i.\end{aligned}$$

Finally, using (5.30), (5.35), (5.37), (5.39), and (5.56), we arrive at the bound

$$\left|\hat{\beta}_{i+1}^2 + \hat{\alpha}_i^2 + \hat{\beta}_i^2 - \|A\hat{v}_i\|_2^2\right| \leq 2i(3\epsilon_0 + 2\epsilon_1)\sigma^2,$$

which proves (5.16) and thus completes the proof of Theorem 1.

5.1.3.2 Numerical Experiments

We give a brief example to illustrate the bounds in (5.13), (5.14), (5.15), and (5.16). We run CA-Lanczos (Algorithm 26) in double precision with $s = 4, 8$, and 12 on a 2D Poisson matrix with $n = 256$, $\|A\|_2 = 7.93$, using a random starting vector (the same random starting vector was used for all tests).

For comparison, Figure 5.1 shows the results for classical Lanczos using the bounds derived by Paige [140]. In Figure 5.1, in the top left plot, the blue curve gives the measured value of orthogonality, $|\hat{\beta}_{i+1}\hat{v}_i^T\hat{v}_{i+1}|$, and the black curve plots the upper bound, $2(n+4)\|A\|_2\epsilon$. In the top right plot, the blue curve gives the measured value of normality, $|\hat{v}_{i+1}^T\hat{v}_{i+1} - 1|$, and the black curve plots the upper bound, $(n+4)\epsilon$. In the bottom left plot (labeled 'Colbound' in the plot title), the blue curve gives the measured value of the bound analogous to (5.13) for $\|\delta\hat{v}_i\|_2$, and the black curve plots the upper bound, $\epsilon(7 + 5\||A|\|_2)$. In the bottom right plot (labeled 'Diffbound' in the plot title), the blue curve gives the measured value of the bound analogous to (5.16), and the black curve plots the upper bound, $4i\epsilon(3(n+4)\|A\|_2 + (7 + 5\||A|\|_2))\|A\|_2$. All 'measured' values were computed in quad precision.

The results for CA-Lanczos are shown in Figures 5.2−5.10. The same tests were run for three different s values for three different basis choices (see Section 3.2.5): monomial (Figures 5.2, 5.5, and 5.8), Newton (Figures 5.3, 5.6, and 5.9), and Chebyshev (Figures 5.4, 5.7, and 5.10). For each of the four plots in each Figure, the blue curves give the measured (in quad precision) values of the quantities on the left hand sides of (5.14) (top left), (5.15) (top right), (5.13) (bottom left), and (5.16) (bottom right). The cyan curves give the maximum of the measured values so far. The red curves give the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1, and the blacks curves give the upper bounds on the right hand sides of (5.14), (5.15), (5.13), and (5.16), respectively.

We see from Figures 5.2−5.10 that the upper bounds given in Theorem 1 are valid. In particular, we can also see that the shape of the red curve plotting $\bar{\Gamma}_k^2$ gives a good indication of the shape of the cyan curves which show the maximum measured values up to the current iteration. However, from, e.g., Figure 5.5 for the monomial basis, we see that if the basis has a high condition number, as does the monomial basis here, the upper bound can be a very large overestimate quantitatively, leading to bounds that are not particularly indicative of the magnitude of the measured values. We will now show how we can use a better definition of $\bar{\Gamma}_k$ to obtain tighter, more descriptive bounds.

Tighter Bounds There is an easy way to improve the bounds by using a different definition of $\bar{\Gamma}_k$ to upper bound quantities in the proof of Theorem 1. Note that all quantities which we have bounded by $\bar{\Gamma}_k$ in Section 5.1.3 are of the form $\|\,|\hat{\mathcal{Y}}_k||x|\,\|_2/\|\hat{\mathcal{Y}}_k x\|_2$. While the use of $\bar{\Gamma}_k$ as defined in Theorem 1 shows how the bounds depend on the conditioning of the computed Krylov bases, we can obtain tighter and more descriptive bounds for (5.15) and (5.14) by instead using the definition

$$\bar{\Gamma}_{k,j} \equiv \max_{x\in\{\hat{w}'_{k,j},\hat{u}'_{k,j},\hat{v}'_{k,j},\hat{v}'_{k,j-1}\}} \frac{\|\,|\hat{\mathcal{Y}}_k||x|\,\|_2}{\|\hat{\mathcal{Y}}_k x\|_2}. \tag{5.61}$$

For the bound in (5.13), we can use the definition

$$\bar{\Gamma}_{k,j} \equiv \max\Big\{\frac{\|\,|\hat{\mathcal{Y}}_k||\mathcal{B}_k||\hat{v}'_{k,j}|\,\|_2}{\|\,|\mathcal{B}_k|\,\|_2\|\hat{\mathcal{Y}}_k\hat{v}'_{k,j}\|_2}, \max_{x\in\{\hat{w}'_{k,j},\hat{u}'_{k,j},\hat{v}'_{k,j},\hat{v}'_{k,j-1}\}}, \frac{\|\,|\hat{\mathcal{Y}}_k||x|\,\|_2}{\|\hat{\mathcal{Y}}_k x\|_2}\Big\}, \tag{5.62}$$

and for the bound in (5.16), we can use the definition

$$\bar{\Gamma}_{k,j} \equiv \max\Big\{\bar{\Gamma}_{k,j-1}, \frac{\|\,|\hat{\mathcal{Y}}_k||\mathcal{B}_k||\hat{v}'_{k,j}|\,\|_2}{\|\,|\mathcal{B}_k|\,\|_2\|\hat{\mathcal{Y}}_k\hat{v}'_{k,j}\|_2}, \max_{x\in\{\hat{w}'_{k,j},\hat{u}'_{k,j},\hat{v}'_{k,j},\hat{v}'_{k,j+1}\}}, \frac{\|\,|\hat{\mathcal{Y}}_k||x|\,\|_2}{\|\hat{\mathcal{Y}}_k x\|_2}\Big\}. \tag{5.63}$$

The value in (5.63) is monotonically increasing since the bound in (5.51) is a sum of bounds from previous iterations.

In Figures 5.11−5.19 we plot bounds for the same problem, bases, and s values as Figures 5.2−5.10, but using the new definitions of $\bar{\Gamma}_{k,j}$. Comparing Figures 5.11−5.19 to Figures 5.2−5.10, we see that these bounds are better both quantitatively, in that they are tighter, and qualitatively, in that they better replicate the shape of the curves for the measured normality and orthogonality values. The exception is for the plots of bounds in (5.16) (bottom right plots), for which there is not much difference qualitatively. It is also clear that the new definitions of $\bar{\Gamma}_k$ correlate well with the size of the measured values (i.e., the shape of the blue curve closely follows the shape of the red curve). Note that, unlike the definition of $\bar{\Gamma}_k$ in Theorem 1, using the definitions in (5.61)−(5.63) do not require the assumption of linear independence of the basis vectors.

Although these new bounds can not be computed a priori, the right hand sides of (5.61), (5.62), and (5.63) can be computed within each inner loop iteration for the cost of at most one

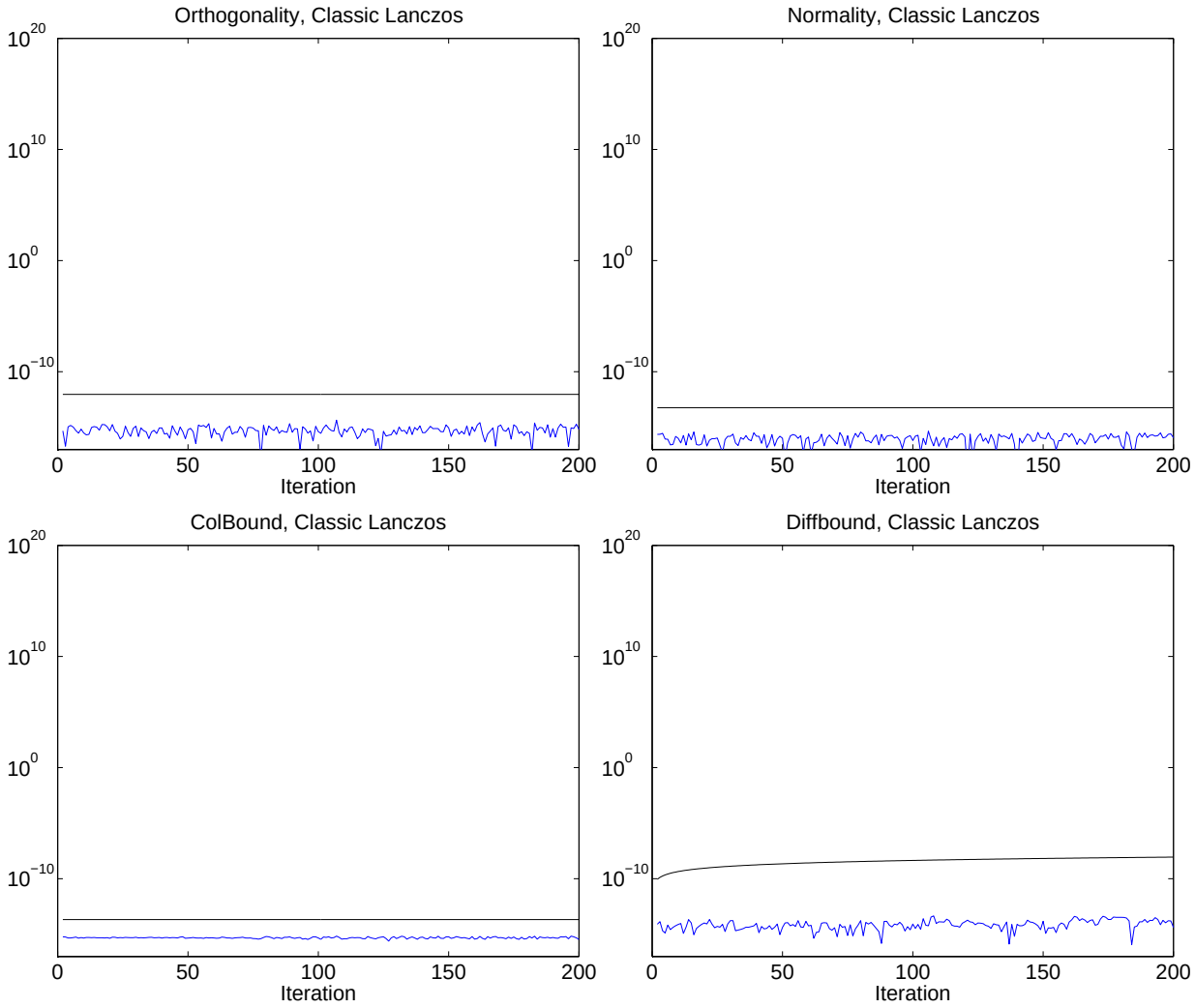

Figure 5.1: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for classical Lanczos on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision) and black lines show the upper bounds, taken from [140].

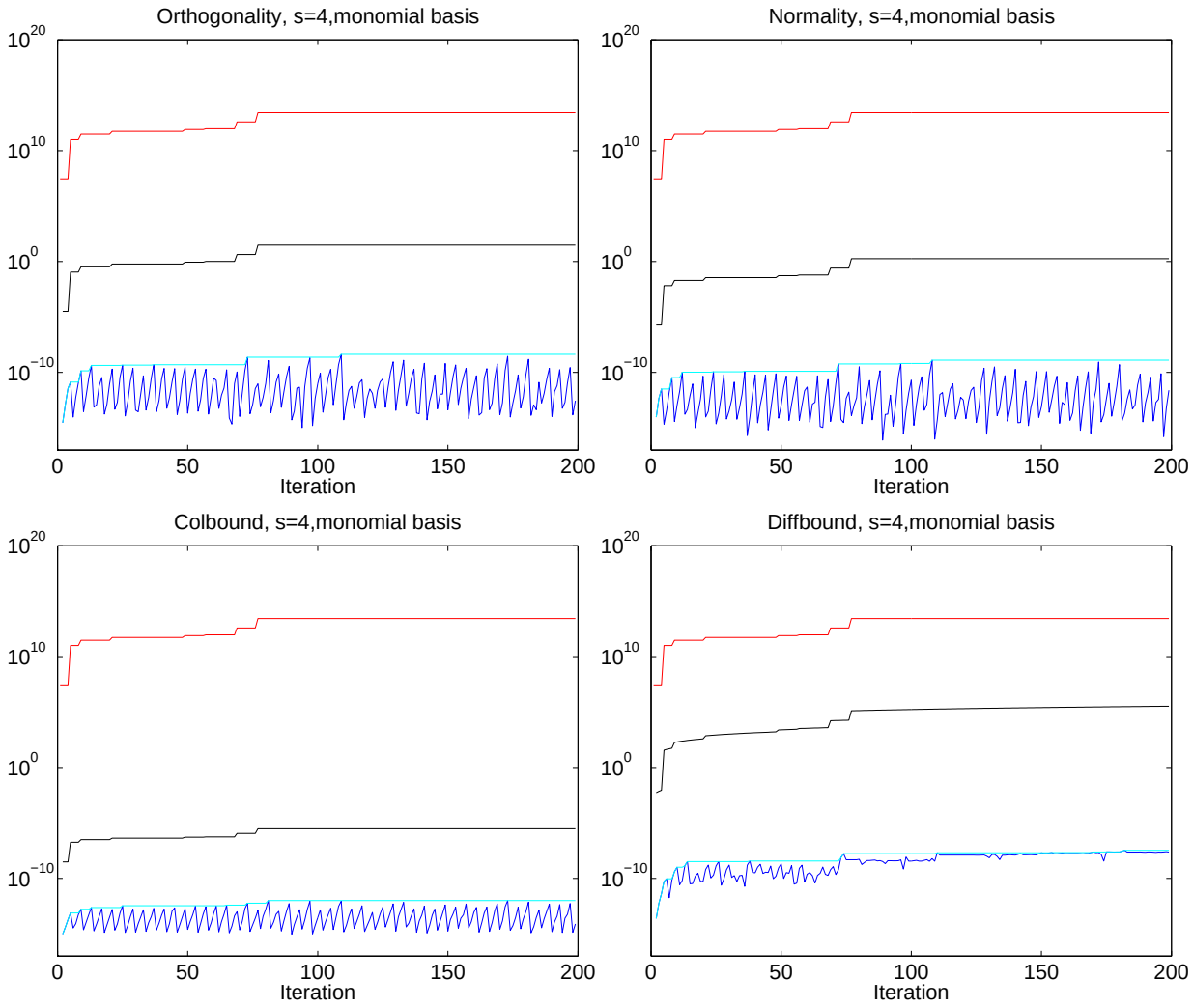

Figure 5.2: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the monomial basis and $s = 4$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), cyan lines show the maximum of the actual values obtained over all previous iterations, black lines show the upper bounds from Theorem 1, and red lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1.

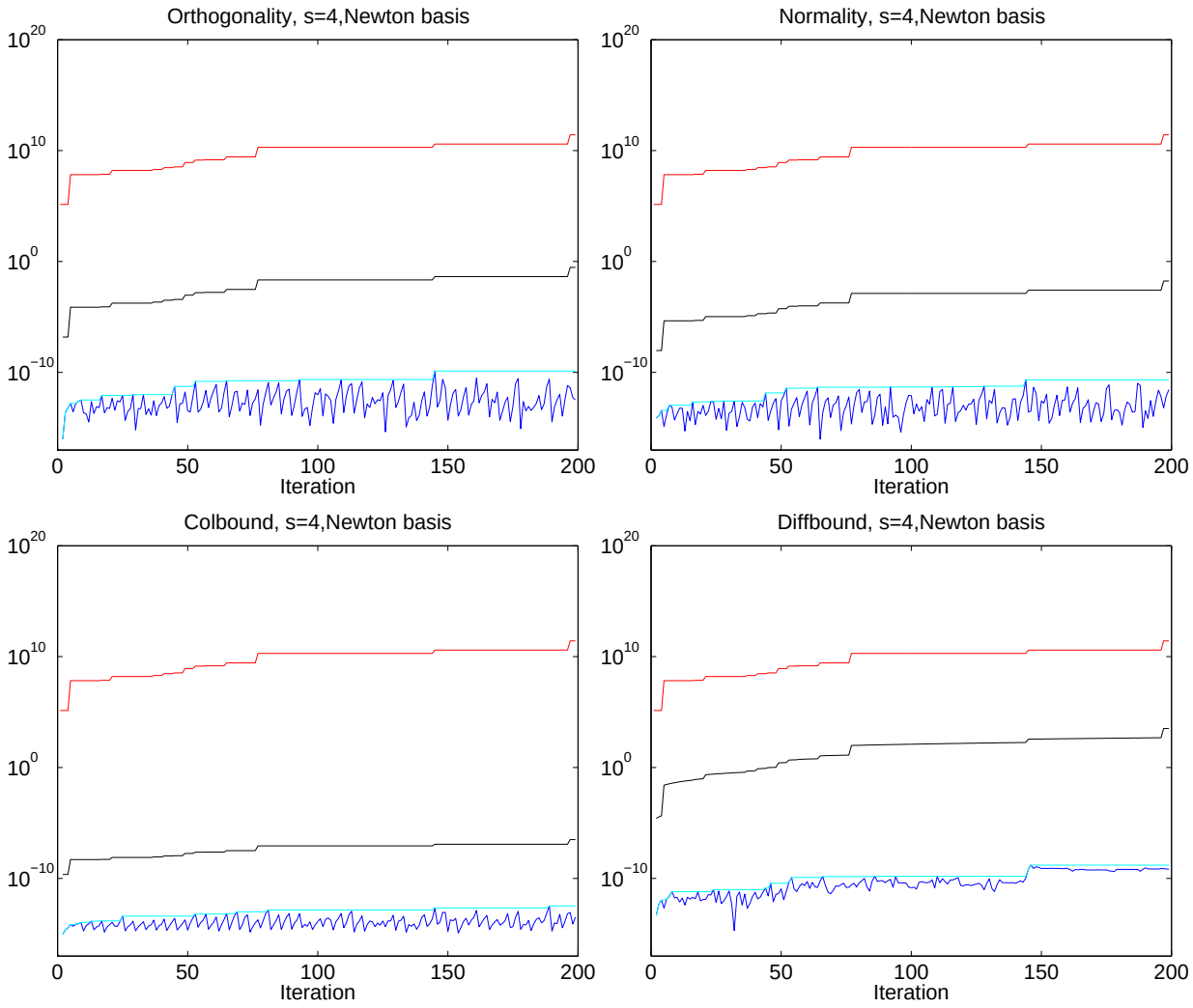

Figure 5.3: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Newton basis and $s = 4$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), cyan lines show the maximum of the actual values obtained over all previous iterations, black lines show the upper bounds from Theorem 1, and red lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1.

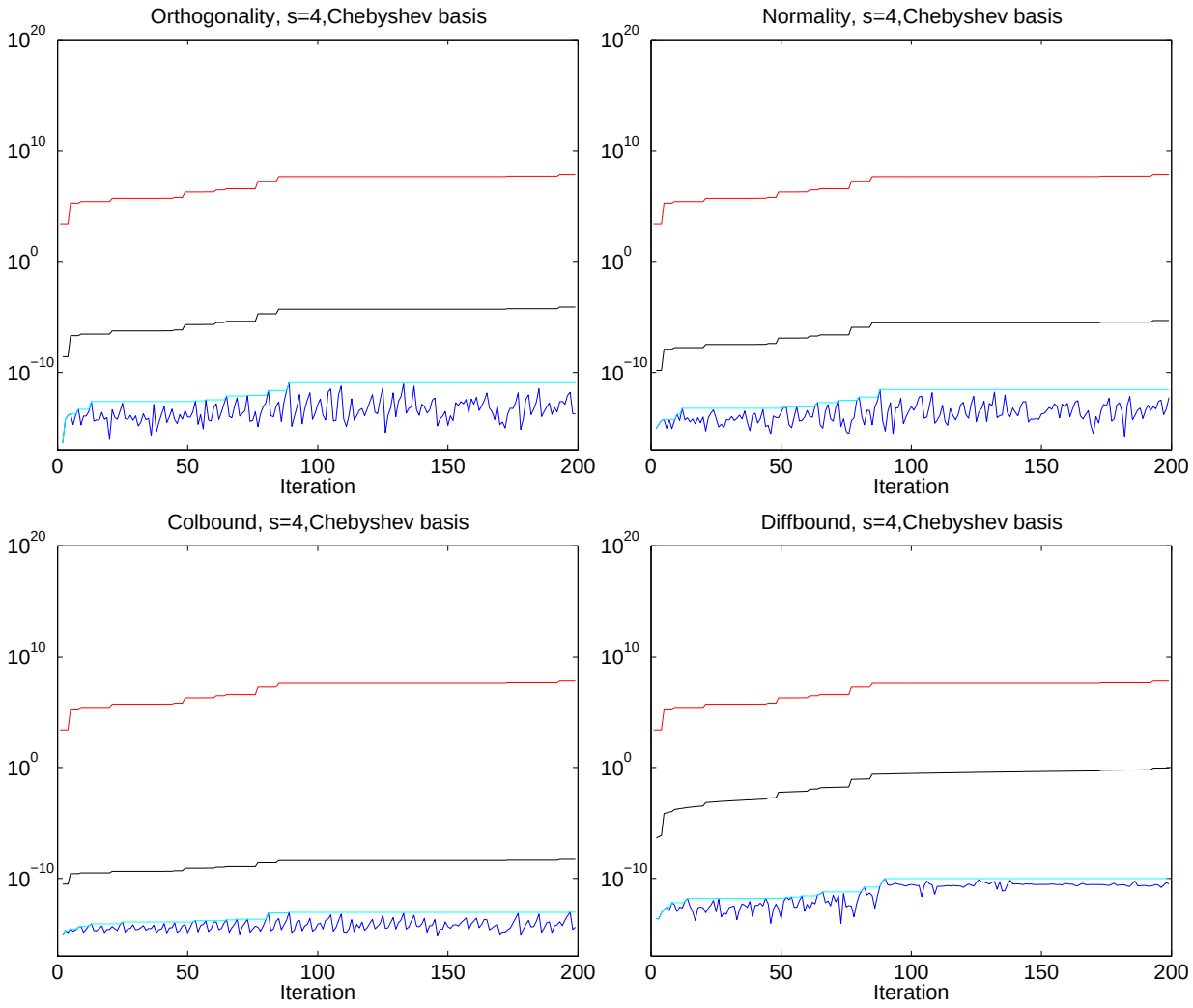

Figure 5.4: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Chebyshev basis and $s = 4$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), cyan lines show the maximum of the actual values obtained over all previous iterations, black lines show the upper bounds from Theorem 1, and red lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1.

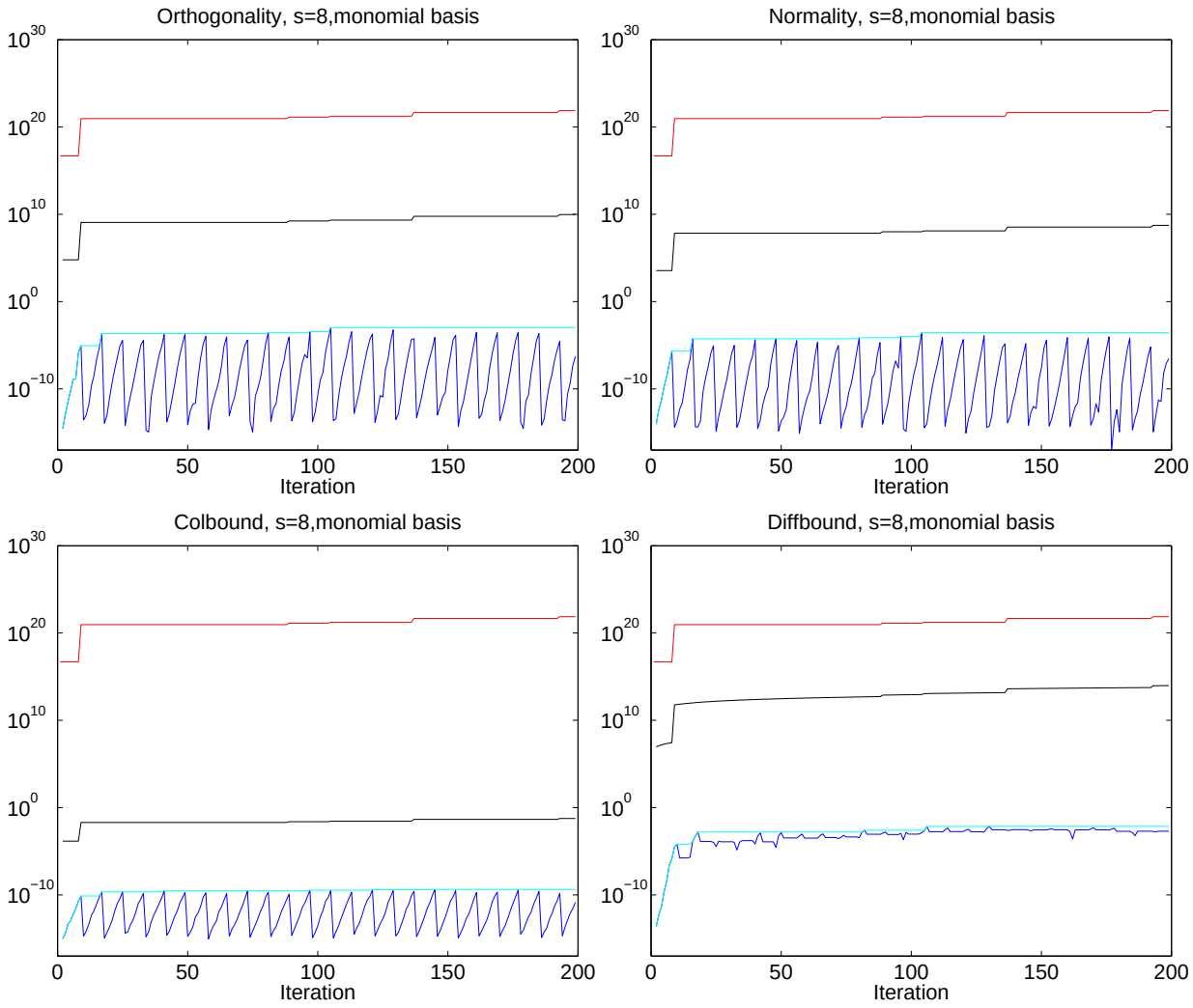

Figure 5.5: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the monomial basis and $s = 8$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), cyan lines show the maximum of the actual values obtained over all previous iterations, black lines show the upper bounds from Theorem 1, and red lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1.

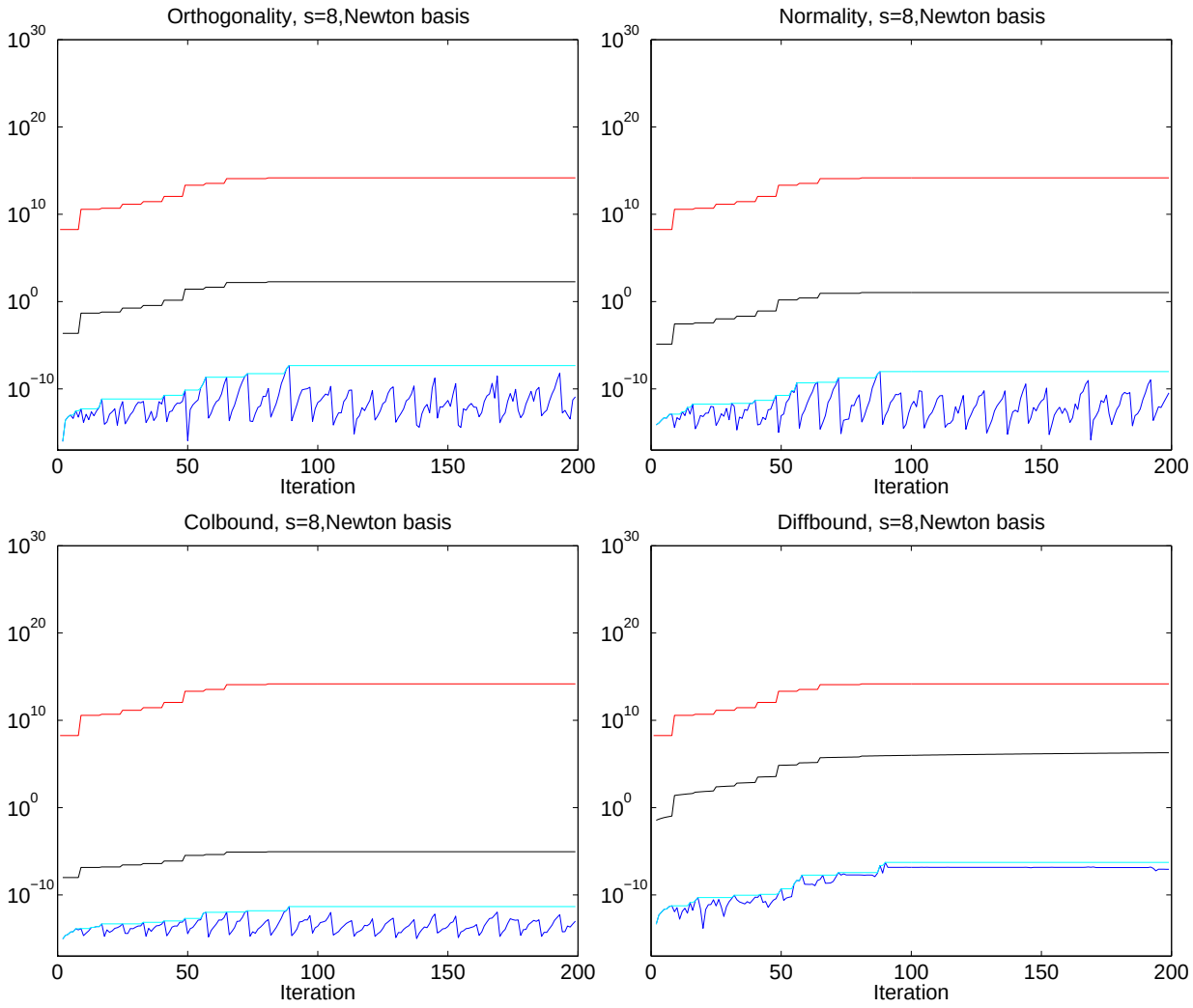

Figure 5.6: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Newton basis and $s = 8$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), cyan lines show the maximum of the actual values obtained over all previous iterations, black lines show the upper bounds from Theorem 1, and red lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1.

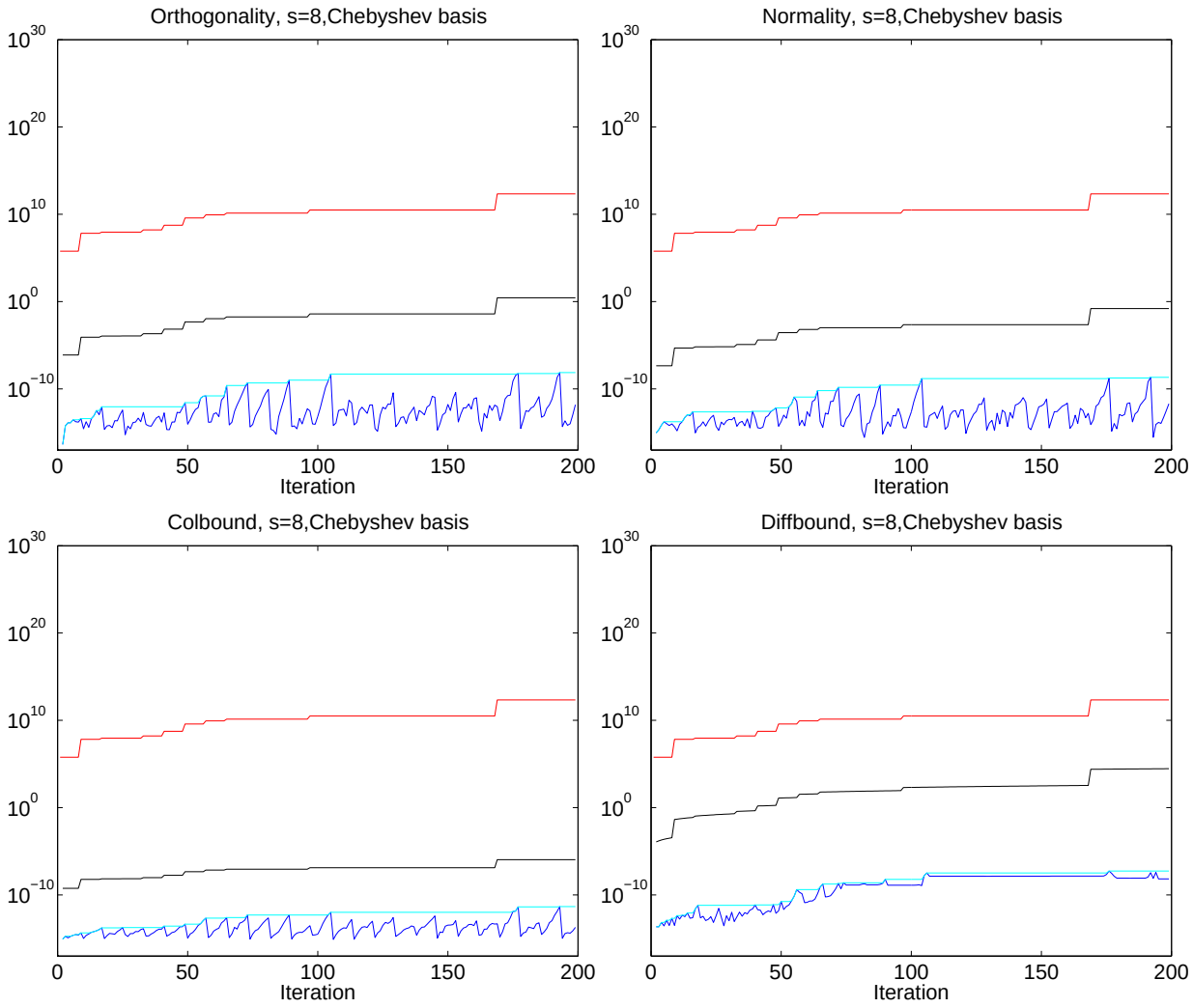

Figure 5.7: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Chebyshev basis and $s = 8$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), cyan lines show the maximum of the actual values obtained over all previous iterations, black lines show the upper bounds from Theorem 1, and red lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1.

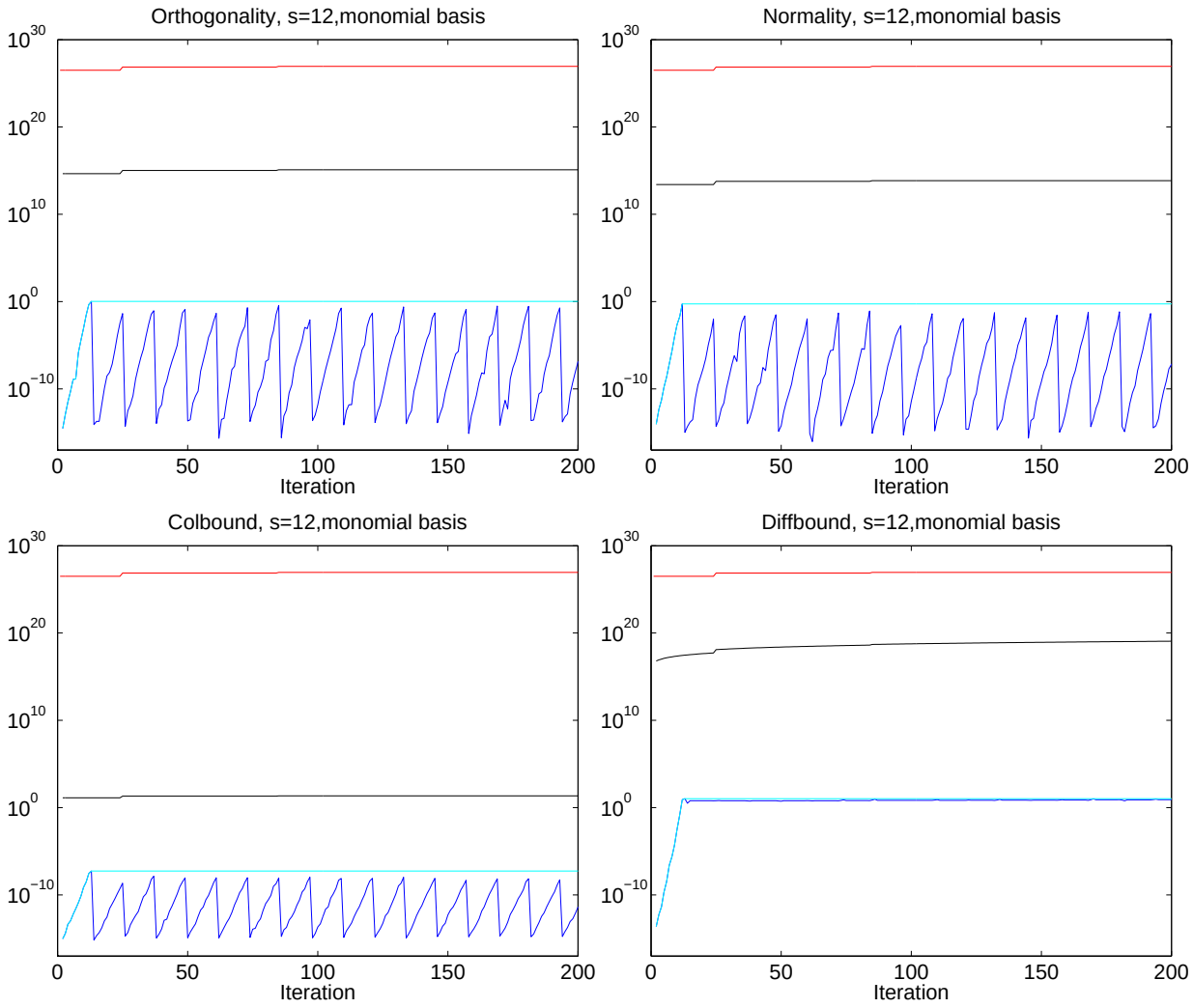

Figure 5.8: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the monomial basis and $s = 12$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), cyan lines show the maximum of the actual values obtained over all previous iterations, black lines show the upper bounds from Theorem 1, and red lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1.

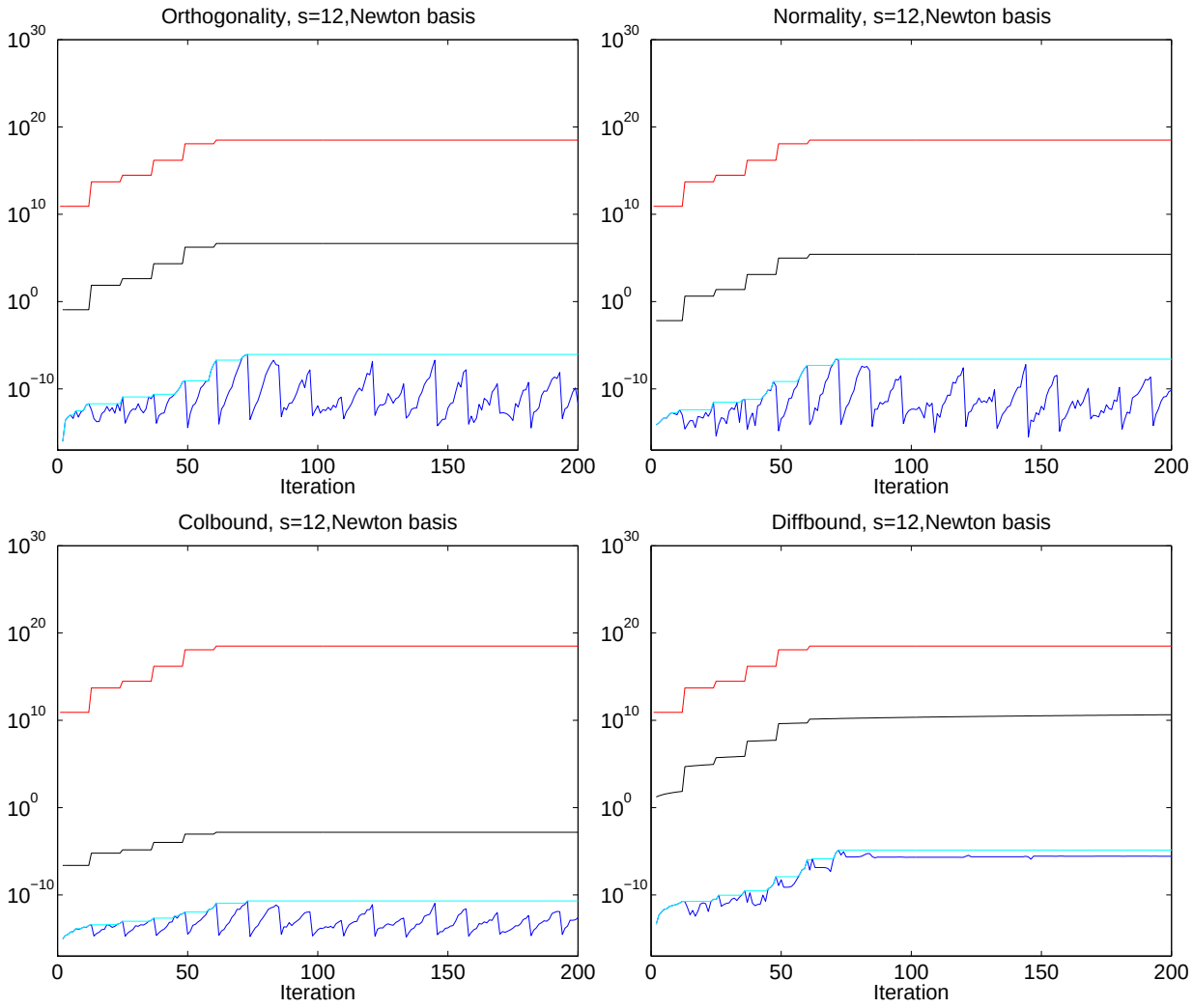

Figure 5.9: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Newton basis and $s = 12$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), cyan lines show the maximum of the actual values obtained over all previous iterations, black lines show the upper bounds from Theorem 1, and red lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1.

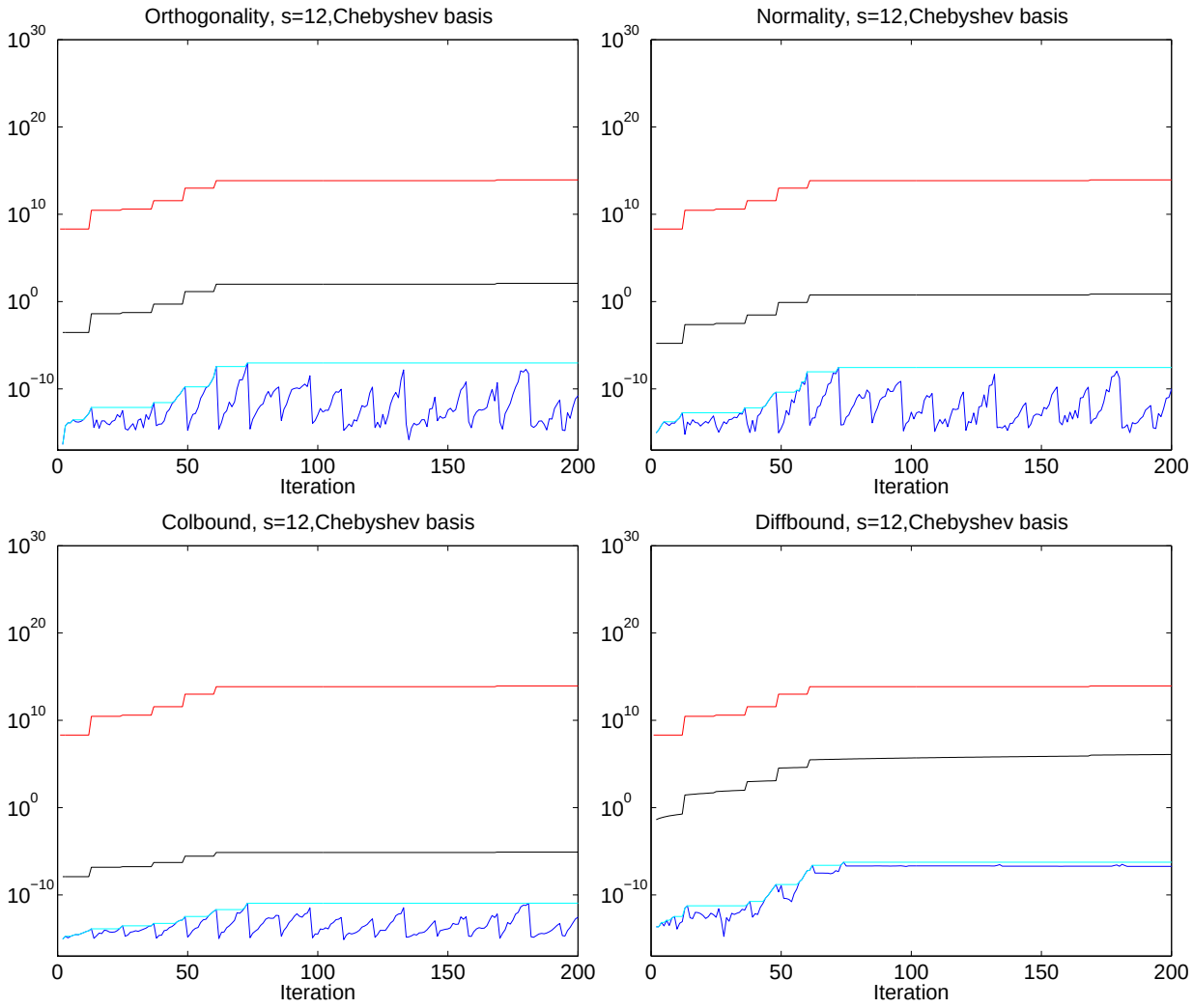

Figure 5.10: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Chebyshev basis and $s = 12$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), cyan lines show the maximum of the actual values obtained over all previous iterations, black lines show the upper bounds from Theorem 1, and red lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1.

extra reduction per outer loop. This extra cost comes from the need to compute $|\hat{\mathcal{Y}}_k|^T|\hat{\mathcal{Y}}_k|$, although this could potentially be performed simultaneously with the computation of $\hat{G}_k$ (line 4 in Algorithm 26). This means that meaningful bounds could be cheaply estimated during the iterations. Implementing a scheme to improve numerical properties using this information remains future work; some potential ideas are discussed further in Sections 6.7 and 6.5.

5.1.4 Accuracy of Eigenvalues

Theorem 1 is in the same form as Paige's equivalent theorem for classical Lanczos [141], except our definitions of ϵ_0 and ϵ_1 are about a factor $\bar{\Gamma}_k^2$ larger (assuming $s \ll n$). This amplification term, which can be bounded in terms of the maximum condition number of the computed s-step Krylov bases, has significant consequences for the algorithm as we will see in the next two sections. The equivalent forms of our theorem and Paige's theorem allow us to immediately apply his results from [141] to the s-step case; the only thing that changes in the s-step case are the values of ϵ_0 and ϵ_1.

In this and the subsequent section, we reproduce the theorems of Paige and discuss their application to the CA-Lanczos method. *Note that we claim no contribution to the analysis techniques used here.* In fact, much of the text in the following sections is taken verbatim from Paige [141], with only the notation changed to match the algorithms in Section 5.1.2.

Our contribution is showing that the theorems of Paige also apply to the CA-Lanczos method under the assumption that (5.64) (and thus also (5.65)) holds. Also note that the text in the paragraphs labeled 'Comments', which discusses the meaning of the results for the s-step case, is our own.

We note that many of the bounds stated slightly differ from those given by Paige in [141]. We suspect that the bounds in [141] were obtained using $\epsilon_0 < 1/100$ rather than the specified $\epsilon_0 < 1/12$, the former being the value used by Paige in his earlier work [138]. Such changes are indicated by footnotes and carried through the remainder of the analysis, resulting in different constants than those in [141]; the fundamental results and conclusions remain unchanged.

Assumptions In order to make use of Paige's analysis [141], we must make the similar assumptions that

$$\hat{\beta}_{i+1} \neq 0 \text{ for } i \in \{1, \ldots, m\}, \quad m(3\epsilon_0 + 2\epsilon_1) \leq 1, \text{ and } \epsilon_0 < \frac{1}{12}. \tag{5.64}$$

These assumptions are used throughout the analysis. Note that (5.64) means that in order to guarantee the applicability of Paige's results for classical Lanczos to the CA-Lanczos case, we must have

$$\bar{\Gamma}_k^2 < \Big(24\epsilon(n+11s+15)\Big)^{-1} = O\big(1/(n\epsilon)\big). \tag{5.65}$$

Since the bounds that will be presented, as well as the bounds in Theorem 1, are not tight, this condition on $\bar{\Gamma}_k^2$ may be overly restrictive in practice. In paragraphs labeled 'Comments',

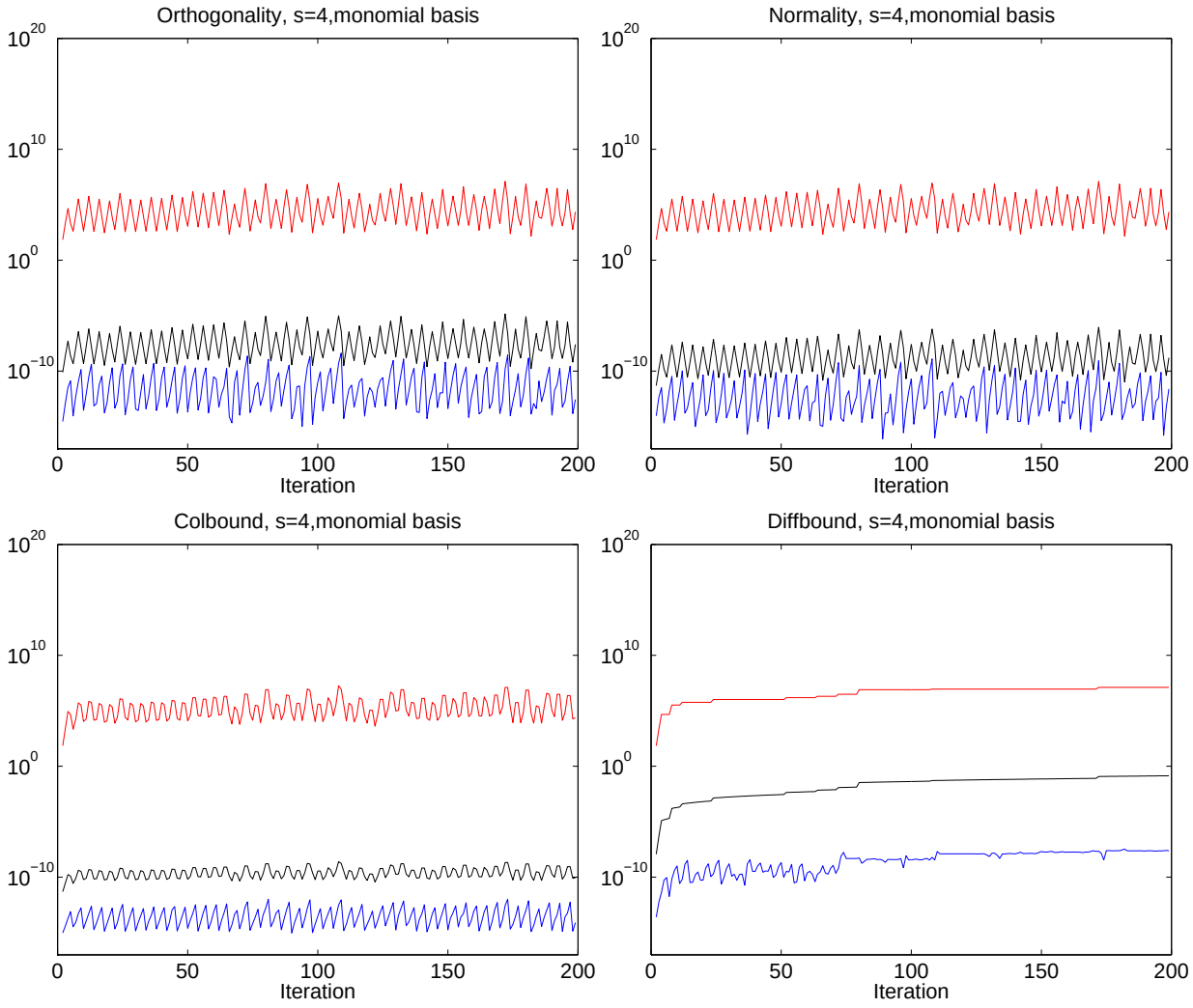

Figure 5.11: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the monomial basis and $s = 4$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), black lines show the upper bounds from Theorem 1 computed using the appropriate definition of $\bar{\Gamma}_k$ in (5.61)-(5.63), and red lines show the appropriate value of $\bar{\Gamma}_k^2$ as defined in (5.61)-(5.63).

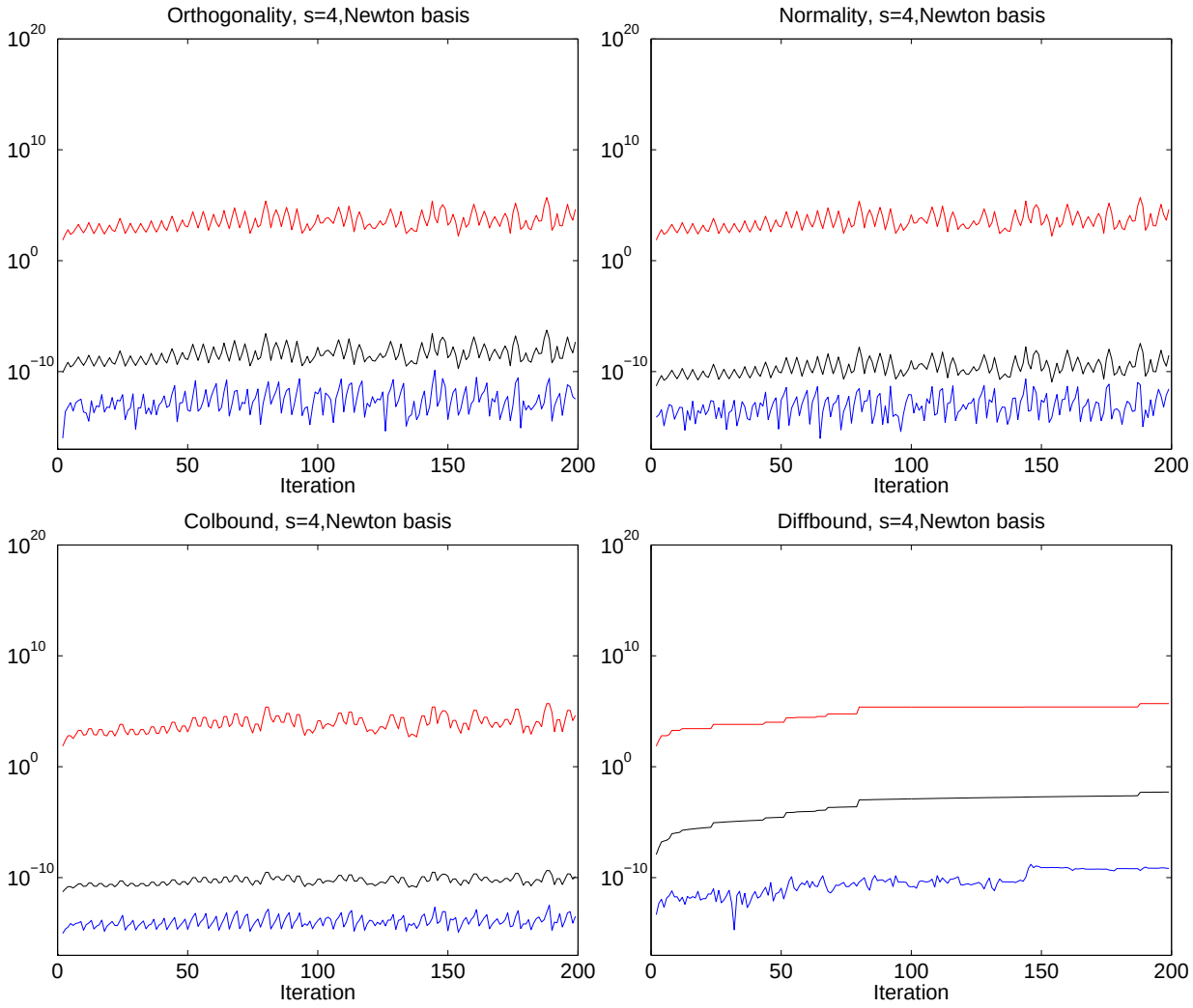

Figure 5.12: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Newton basis and $s = 4$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), black lines show the upper bounds from Theorem 1 computed using the appropriate definition of $\bar{\Gamma}_k$ in (5.61)-(5.63), and red lines show the appropriate value of $\bar{\Gamma}_k^2$ as defined in (5.61)-(5.63).

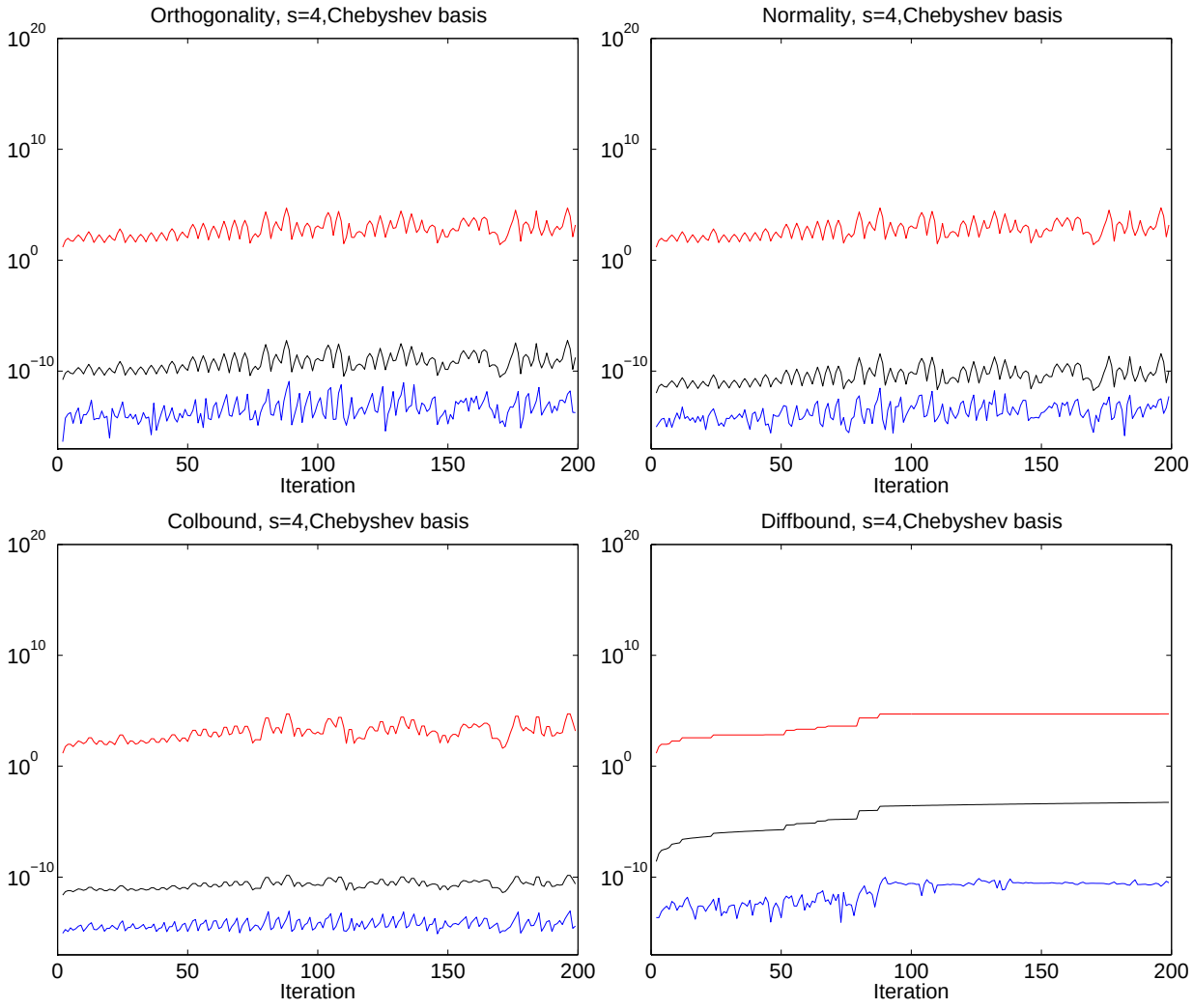

Figure 5.13: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Chebyshev basis and $s = 4$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), black lines show the upper bounds from Theorem 1 computed using the appropriate definition of $\bar{\Gamma}_k$ in (5.61)-(5.63), and red lines show the appropriate value of $\bar{\Gamma}_k^2$ as defined in (5.61)-(5.63).

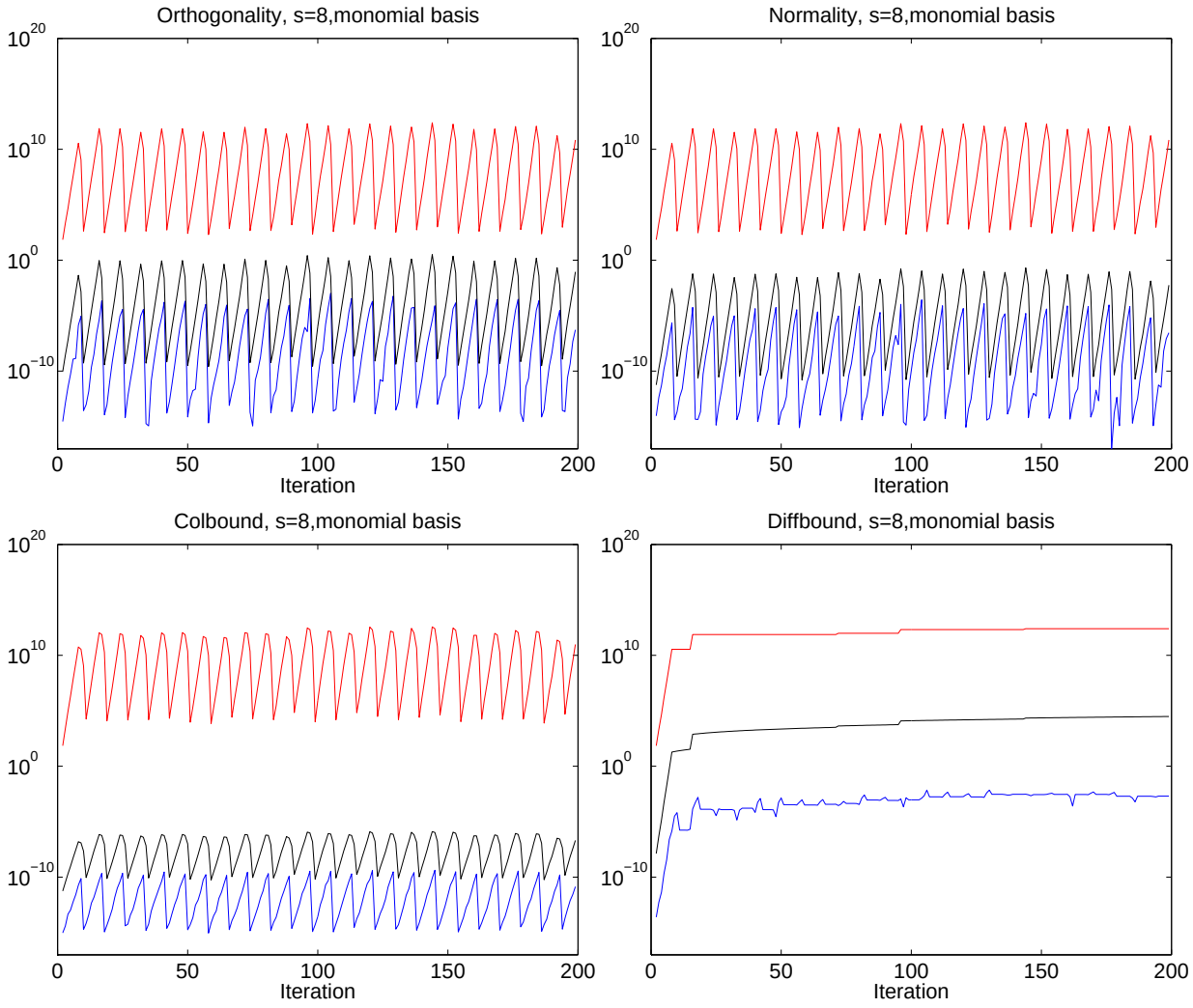

Figure 5.14: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the monomial basis and $s = 8$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), black lines show the upper bounds from Theorem 1 computed using the appropriate definition of $\bar{\Gamma}_k$ in (5.61)-(5.63), and red lines show the appropriate value of $\bar{\Gamma}_k^2$ as defined in (5.61)-(5.63).

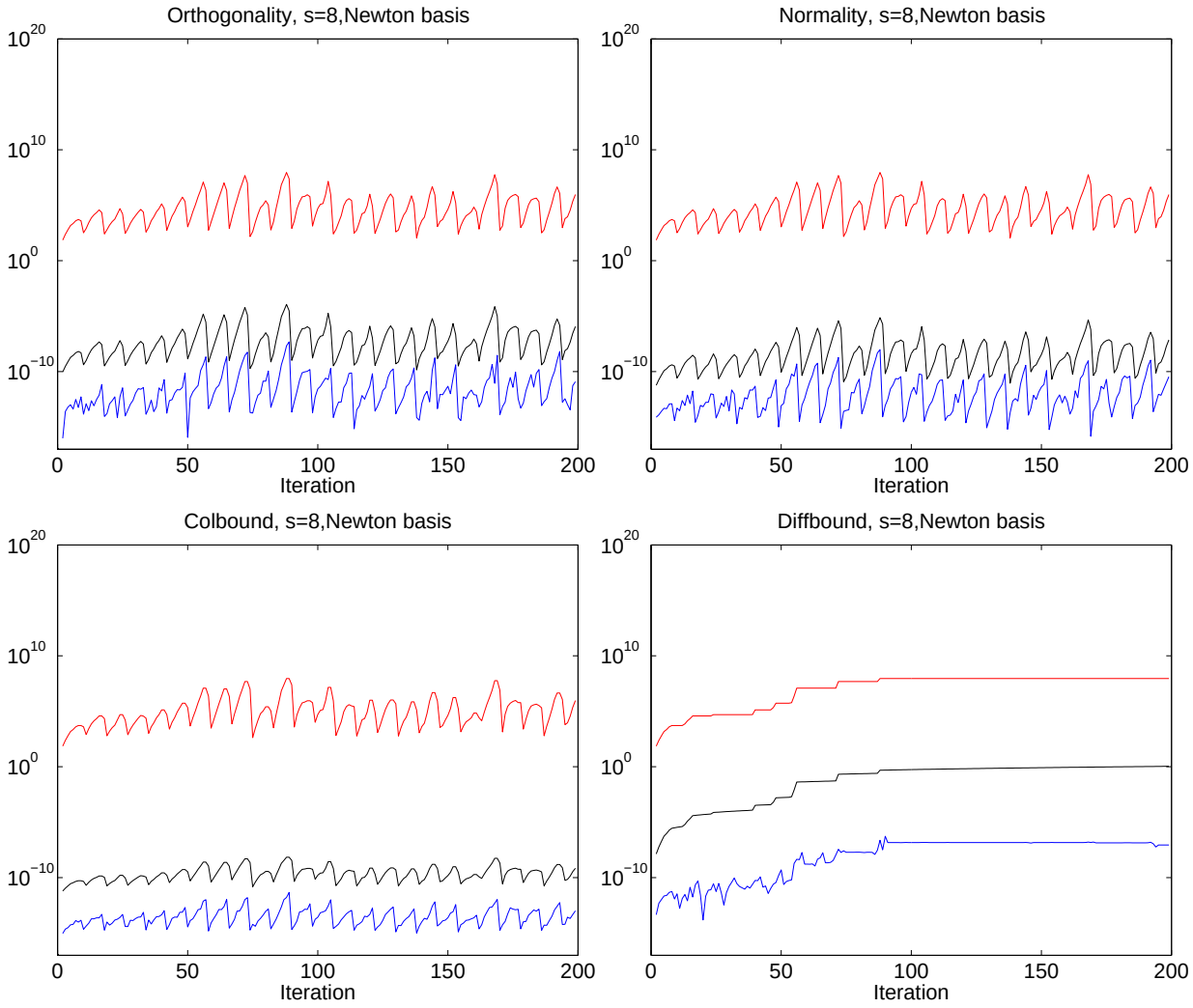

Figure 5.15: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Newton basis and $s = 8$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), black lines show the upper bounds from Theorem 1 computed using the appropriate definition of $\bar{\Gamma}_k$ in (5.61)-(5.63), and red lines show the appropriate value of $\bar{\Gamma}_k^2$ as defined in (5.61)-(5.63).

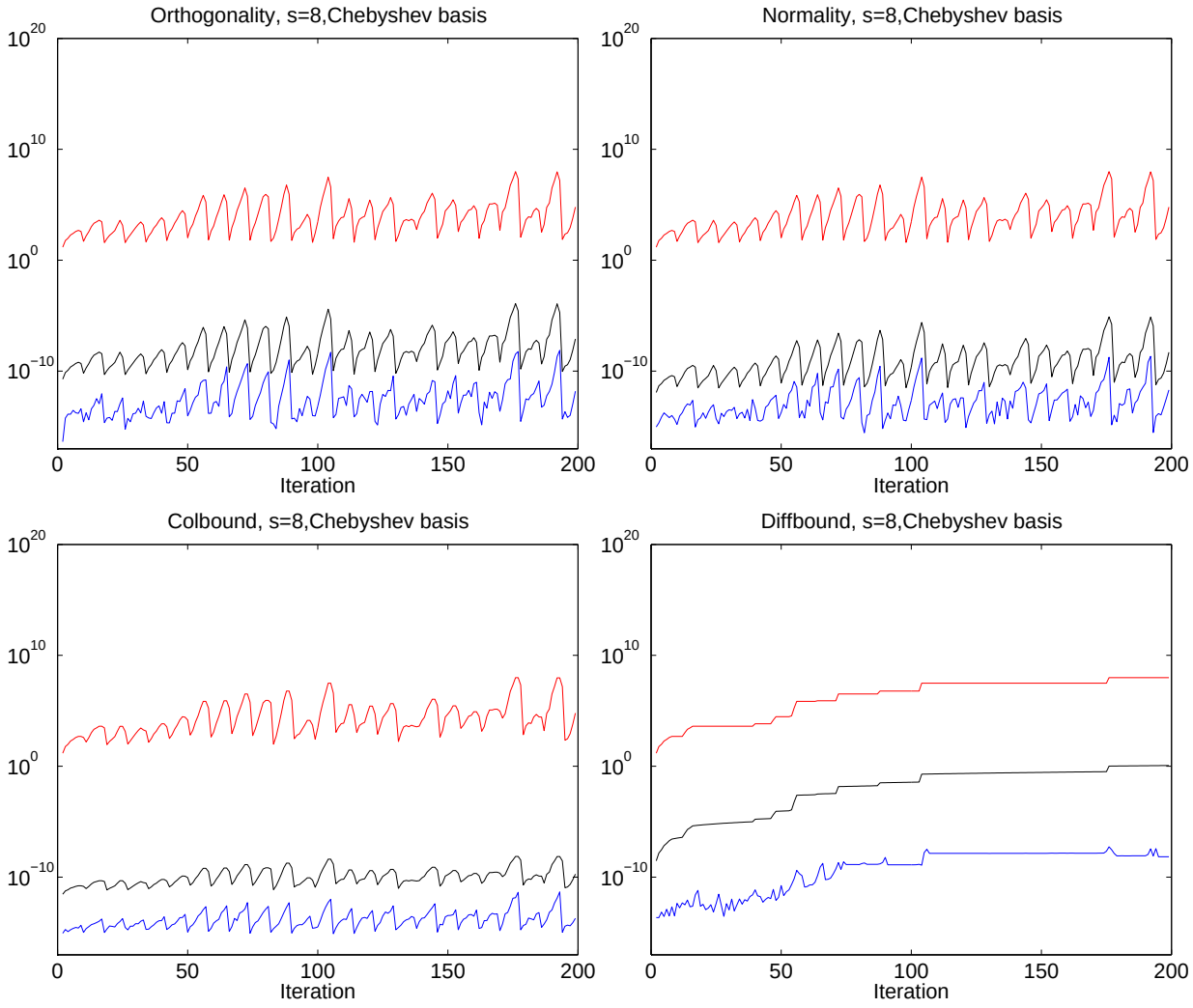

Figure 5.16: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Chebyshev basis and $s = 8$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), black lines show the upper bounds from Theorem 1 computed using the appropriate definition of $\bar{\Gamma}_k$ in (5.61)-(5.63), and red lines show the appropriate value of $\bar{\Gamma}_k^2$ as defined in (5.61)-(5.63).

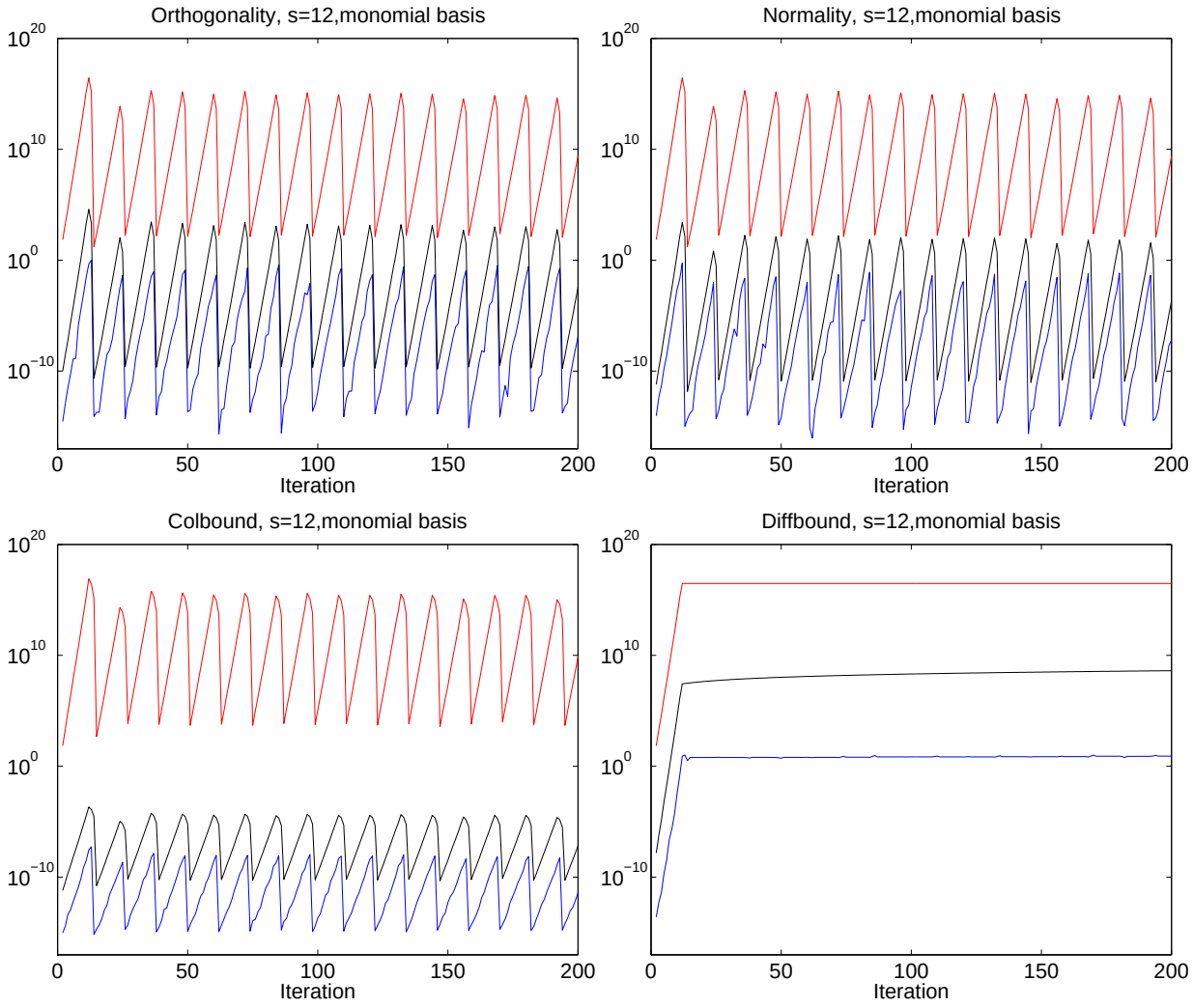

Figure 5.17: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the monomial basis and $s = 12$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), black lines show the upper bounds from Theorem 1 computed using the appropriate definition of $\bar{\Gamma}_k$ in (5.61)-(5.63), and red lines show the appropriate value of $\bar{\Gamma}_k^2$ as defined in (5.61)-(5.63).

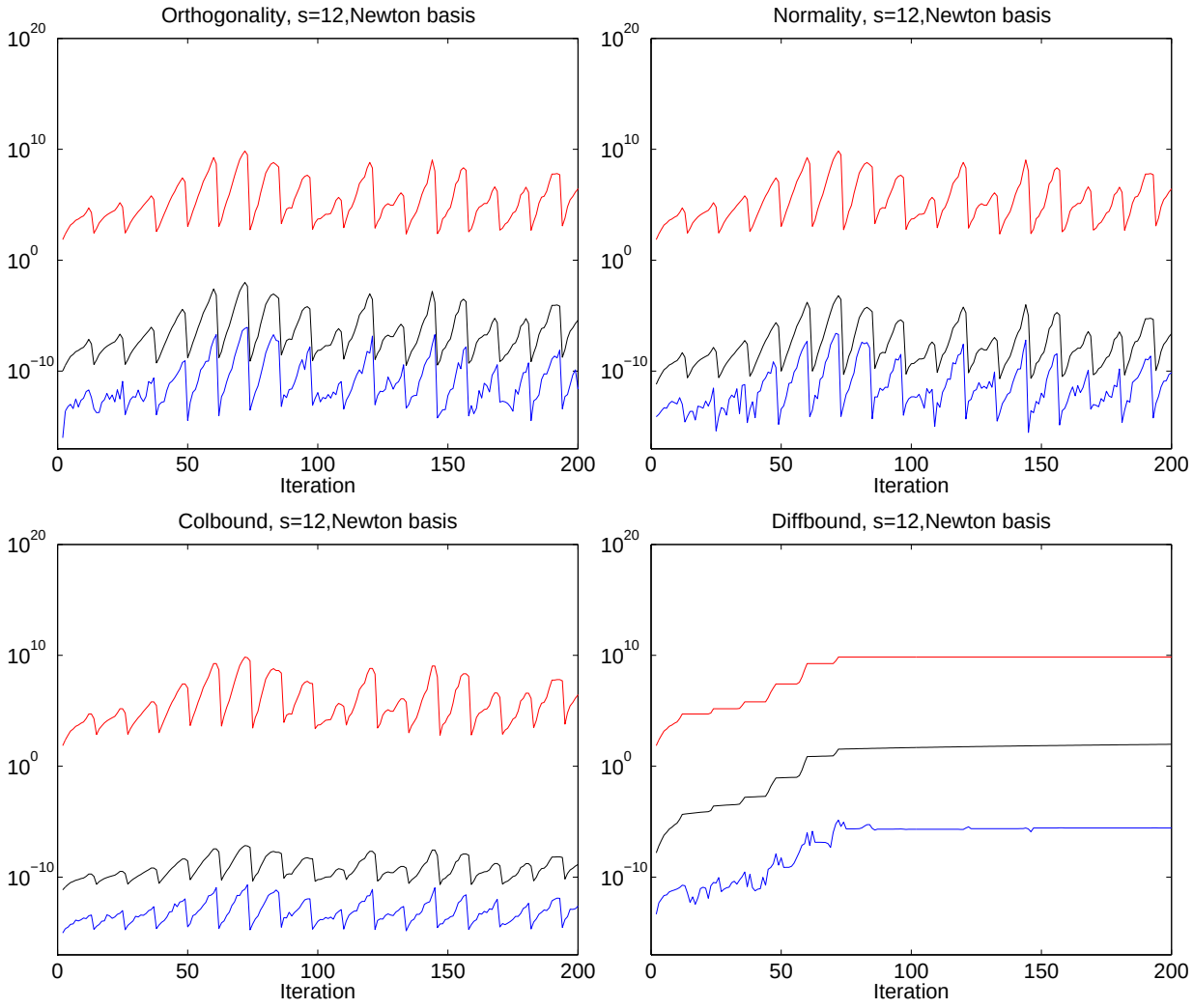

Figure 5.18: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Newton basis and $s = 12$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), black lines show the upper bounds from Theorem 1 computed using the appropriate definition of $\bar{\Gamma}_k$ in (5.61)-(5.63), and red lines show the appropriate value of $\bar{\Gamma}_k^2$ as defined in (5.61)-(5.63).

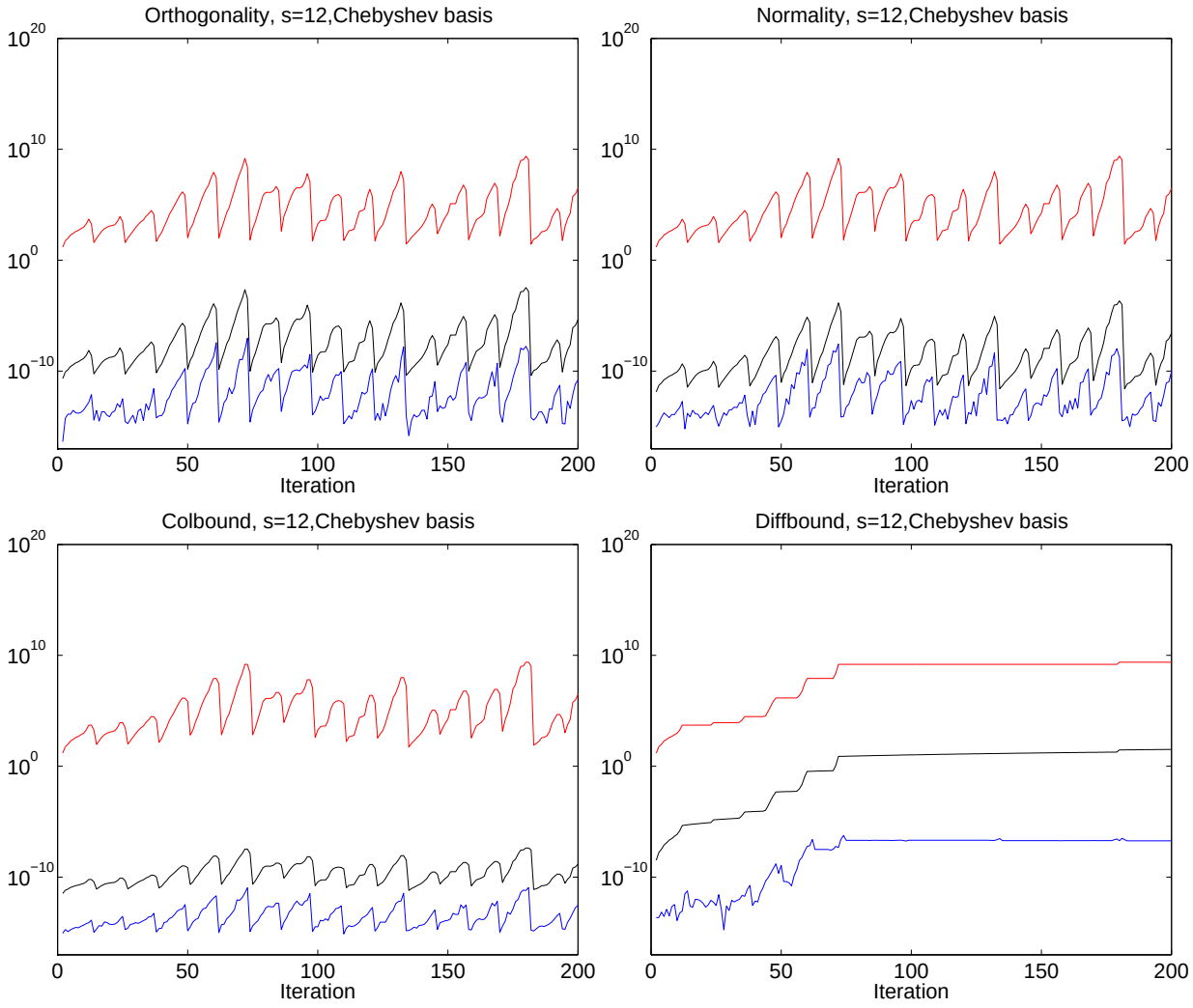

Figure 5.19: Orthogonality (top left), Normality (top right), recurrence column error (bottom left), and difference in recurrence column size (bottom right) for CA-Lanczos using the Chebyshev basis and $s = 12$ on a 2D Poisson problem with $n = 256$ with a random starting vector. Blue lines show the actual values (computed in quad precision), black lines show the upper bounds from Theorem 1 computed using the appropriate definition of $\bar{\Gamma}_k$ in (5.61)-(5.63), and red lines show the appropriate value of $\bar{\Gamma}_k^2$ as defined in (5.61)-(5.63).

we comment on what happens to the bounds and analysis in the case that $\bar{\Gamma}_k^2$ exceeds this value, i.e., at least one computed s-step basis is ill-conditioned. As stated previously, we also assume that no underflow or overflow occurs, and that all s-step Krylov bases are numerically full rank.

Using (5.20), it can be shown that

$$\|\delta R_m\|_F^2 \leq \sigma^2\Big(\big(5m-4\big)\epsilon_0^2 + 4(m-1)\epsilon_0\epsilon_1 + 2m(m-1)\epsilon_1^2\Big) \tag{5.66}$$

where subscript F denotes the Frobenius norm. If we define

$$\epsilon_2 \equiv \sqrt{2}\max\left(6\epsilon_0, \epsilon_1\right), \tag{5.67}$$

then (5.66) gives

$$\|\delta R_m\|_F \leq m\sigma\epsilon_2. \tag{5.68}$$

Let the eigendecomposition of $\hat{T}_m$ be

$$\hat{T}_m Q^{(m)} = Q^{(m)}\,\mathrm{diag}\big(\mu_i^{(m)}\big), \tag{5.69}$$

for $i \in \{1,\ldots,m\}$, where the orthonormal matrix $Q^{(m)}$ has i^{th} column $q_i^{(m)}$ and (ℓ,i) element $\eta_{\ell,i}^{(m)}$, and the eigenvalues are ordered

$$\mu_1^{(m)} > \mu_2^{(m)} > \cdots > \mu_m^{(m)}.$$

Note that it is assumed that the decomposition (5.69) is computed exactly. If $\mu_i^{(m)}$ is an approximation to an eigenvalue λ_i of A, then the corresponding approximate eigenvector is $z_i^{(m)}$, the ith column of

$$Z^{(m)} \equiv \hat{V}_m Q^{(m)}. \tag{5.70}$$

We now review some properties of $\hat{T}_m$. Let $\nu_i^{(m)}$, for $i \in \{1,\ldots,m-1\}$, be the eigenvalues of the matrix obtained by removing the $(t+1)$st row and column of $\hat{T}_m$, ordered so that

$$\mu_1^{(m)} \geq \nu_1^{(m)} \geq \mu_2^{(m)} \geq \cdots \geq \nu_{m-1}^{(m)} \geq \mu_m^{(m)}.$$

It was shown in [174] that

$$\left(\eta_{t+1,i}^{(m)}\right)^2 = \prod_{\ell=1,\ell\neq i}^{m} \delta_\ell(t+1,i,m) \tag{5.71}$$

$$\delta_\ell(t+1,i,m) \equiv \begin{cases} \frac{\mu_i^{(m)}-\nu_\ell^{(m)}}{\mu_i^{(m)}-\mu_\ell^{(m)}} & \ell = 1,2,\ldots,i-1 \\ \frac{\mu_i^{(m)}-\nu_{\ell-1}^{(m)}}{\mu_i^{(m)}-\mu_\ell^{(m)}} & \ell = i+1,\ldots,m \end{cases}$$

$$0 \leq \delta_\ell(t+1,i,m) \leq 1, \ell = 1,\ldots,i-1,i+1,\ldots,m. \tag{5.72}$$

If we apply $\hat{T}_m$ to the r^{th} eigenvector of $\hat{T}_t$, where $1 \leq r \leq t < m$,

$$\hat{T}_m \begin{bmatrix} q_r^{(t)} \\ 0_{m-t,1} \end{bmatrix} = \begin{bmatrix} \mu_r^{(t)} q_r^{(t)} \\ \hat{\beta}_{t+1} \eta_{t,r}^{(t)} e_1 \end{bmatrix} \tag{5.73}$$

and from [188],

$$\delta_{t,r} \equiv \hat{\beta}_{t+1} |\eta_{t,r}^{(t)}| \geq \min_i |\mu_i^{(m)} - \mu_r^{(t)}|. \tag{5.74}$$

Definition 1. *[141, Definition 1] We say that an eigenvalue $\mu_r^{(t)}$ of $\hat{T}_t$* has stabilized to within $\delta_{t,r}$ *if, for every $m > t$, we know there is an eigenvalue of $\hat{T}_m$ within $\delta_{t,r}$ of $\mu_r^{(t)}$. We will say $\mu_r^{(t)}$ has* stabilized *when we know it has stabilized to within $\gamma(m+1)^{\omega}\sigma\epsilon_2$ where γ and ω are small positive constants.*

From (5.74), we can see that after t steps, $\mu_r^{(t)}$ has necessarily stabilized to within $\delta_{t,r}$. Multiplying (5.73) by $q_i^{(m)T}$, $i \in \{1, \ldots, m\}$, gives

$$\big(\mu_i^{(m)} - \mu_r^{(t)}\big) q_i^{(m)T} \begin{bmatrix} q_r^{(t)} \\ 0_{m-t,1} \end{bmatrix} = \hat{\beta}_{t+1} \eta_{t+1,i}^{(m)} \eta_{t,r}^{(t)}. \tag{5.75}$$

Another result is obtained by applying eigenvectors of $\hat{T}_m$ to each side of (5.19). Multiplying (5.19) on the left by $q_\ell^{(m)T}$ and on the right by $q_i^{(m)}$ for some $i, \ell \in \{1, \ldots, m\}$, and using (5.69) and (5.70), we obtain

$$\big(\mu_\ell^{(m)} - \mu_i^{(m)}\big) q_\ell^{(m)T} R_m q_i^{(m)} = \hat{\beta}_{m+1} z_\ell^{(m)T} \hat{v}_{m+1} \eta_{m,i}^{(m)} + \epsilon_{\ell,i}^{(m)}, \tag{5.76}$$

where $\epsilon_{\ell,i}^{(m)} \equiv q_\ell^{(m)T} \delta R_m q_i^{(m)}$, and

$$|\epsilon_{\ell,i}^{(m)}| \leq m\sigma\epsilon_2, \tag{5.77}$$

which follows from (5.68). Taking $i = \ell$, the left hand side of (5.76) is zero, and we get

$$z_i^{(m)T} \hat{v}_{m+1} = -\frac{\epsilon_{i,i}^{(m)}}{\hat{\beta}_{m+1} \eta_{m,i}^{(m)}}, \tag{5.78}$$

and thus by (5.74), $z_i^{(m)}$ is almost orthogonal to $\hat{v}_{m+1}$ if we have not yet obtained a small eigenvalue interval about $\mu_i^{(m)}$, the eigenvector approximation $z_i^{(m)}$ does not have a small norm, and $\bar{\Gamma}_k$, and thus ϵ_0 and ϵ_2, are small.

Definition 2. *[141, Definition 2] We will say that an eigenpair (μ, z)* represents an eigenpair of A to within δ *if we know that $\|Az - \mu z\|/\|z\| \leq \delta$.*

Thus if (μ, z) represents an eigenpair of A to within δ, then (μ, z) is an exact eigenpair of A perturbed by a matrix whose 2-norm is no greater than δ, and if μ is the Rayleigh quotient of A with z, then the perturbation will be taken symmetric.

Multiplying (5.9) on the right by $q_i^{(m)}$, we get

$$A\hat{V}_m q_i^{(m)} = \hat{V}_m \hat{T}_m q_i^{(m)} + \hat{\beta}_{m+1}\hat{v}_{m+1} e_m^T q_i^{(m)} + \delta \hat{V}_m q_i^{(m)}.$$

Using (5.69) and (5.70), this can be written

$$Az_i^{(m)} - \mu_i^{(m)} z_i^{(m)} = \hat{\beta}_{m+1}\eta_{m,i}^{(m)}\hat{v}_{m+1} + \delta \hat{V}_m q_i^{(m)}. \tag{5.79}$$

Now, using the above and (5.13), (5.15), and (5.74), if λ_ℓ are the eigenvalues of A, then

$$\min_\ell |\lambda_\ell - \mu_i^{(m)}| \le \frac{\|Az_i^{(m)} - \mu_i^{(m)} z_i^{(m)}\|}{\|z_i^{(m)}\|} \le \frac{\delta_{m,i}(1+\epsilon_0) + m^{1/2}\sigma\epsilon_1}{\|z_i^{(m)}\|}, \tag{5.80}$$

and if

$$\|z_i^{(m)}\| \approx 1, \tag{5.81}$$

then $(\mu_i^{(m)}, z_i^{(m)})$ represents an eigenpair of A to within about $\delta_{m,i}$. Unfortunately, one can not expect (5.81) to hold in finite precision.

From (5.70) and (5.18), we see that

$$\|z_i^{(m)}\|^2 - 1 = 2q_i^{(m)T} R_m q_i^{(m)} + q_i^{(m)T} \mathrm{diag}(\hat{V}_m^T \hat{V}_m - I_m) q_i^{(m)}, \tag{5.82}$$

where by (5.15), the last term on the right has magnitude bounded by $\epsilon_0/2$.

Using (5.70), we can write $\hat{V}_t^T = Q^{(t)} Z^{(t)T}$, and multiplying on the right by $\hat{v}_{t+1}$,

$$\hat{V}_t^T \hat{v}_{t+1} = Q^{(t)} b_t, \quad \text{where } b_t = Z^{(t)T}\hat{v}_{t+1}. \tag{5.83}$$

Using (5.78), we have $e_r^T b_t = -\epsilon_{r,r}^{(t)}/(\hat{\beta}_{t+1}\eta_{t,r}^{(t)})$, and by (5.71) and (5.75), this gives

$$q_i^{(m)T} R_m q_i^{(m)} = -\sum_{t=1}^{m-1} \eta_{t+1,i}^{(m)} \sum_{r=1}^{t} \frac{\epsilon_{r,r}^{(t)}}{\hat{\beta}_{t+1}\eta_{t,r}^{(t)}} q_i^{(m)T} \begin{bmatrix} q_r^{(t)} \\ 0_{m-t,1} \end{bmatrix} \tag{5.84}$$

$$= -\sum_{t=1}^{m-1} \left(\eta_{t+1,i}^{(m)}\right)^2 \sum_{r=1}^{t} \frac{\epsilon_{r,r}^{(t)}}{\mu_i^{(m)} - \mu_r^{(t)}} \tag{5.85}$$

$$= -\sum_{t=1}^{m-1}\sum_{r=1}^{t} \left(\frac{\epsilon_{r,r}^{(t)}}{\mu_i^{(m)} - \mu_{c(r)}^{(m)}} \cdot \prod_{\substack{\ell=1 \\ \ell\neq i \\ \ell\neq c(r)}}^{m} \delta_\ell(t+1, i, m) \right). \tag{5.86}$$

From (5.82), under the assumptions in (5.64), $\|z_i^{(m)}\|$ will be significantly different from unity only if the right hand sides of these last three numbered equations are large. In this case (5.84) shows there must be a small $\delta_{t,r} = \hat{\beta}_{t+1}|\eta_{t,r}^{(t)}|$, and some $\mu_r^{(t)}$ has therefore stabilized. Equation (5.85) shows that some $\mu_r^{(t)}$ must be close to $\mu_i^{(m)}$, and combining this with (5.84) we will show that at least one such $\mu_r^{(t)}$ has stabilized. Finally from (5.86), we see that there is at least one $\mu_{c(r)}^{(m)}$ close to $\mu_i^{(m)}$, so that $\mu_i^{(m)}$ cannot be a well-separated eigenvalue of $\hat{T}_m$. Conversely, if $\mu_i^{(m)}$ is a well-separated eigenvalue of $\hat{T}_m$, then (5.81) holds, and if $\mu_i^{(m)}$ has stabilized, then it and $z_i^{(m)}$ are a satisfactory approximation to an eigenpair of A.

Note that if the assumptions in (5.64) do not hold, $\|z_i^{(m)}\|$ can be significantly differ from unity if $|q_i^{(m)T} R_m q_i^{(m)}|$ is large and/or if $\epsilon_0/2$ is large (e.g., due to a large $\bar{\Gamma}_k^2$; see (5.17)). If $\|z_i^{(m)}\|$ is significantly different from unity and $\epsilon_0/2$ is large, we can not necessarily draw meaningful conclusions about the eigenvalues of $\hat{T}_m$ via (5.84), (5.85), and (5.86) based on the size of $\|z_i^{(m)}\|$.

We will now quantify these claims. We first seek an upper bound on $|q_i^{(m)T} R_m q_i^{(m)}|$. From (5.68) and (5.77),

$$\sum_{r=1}^{t} \left(\epsilon_{r,r}^{(t)}\right)^2 \leq \sum_{r=1}^{t} \sum_{c=1}^{t} \left(\epsilon_{r,c}^{(t)}\right)^2 = \|\delta R_t\|_F^2 \leq t^2 \sigma^2 \epsilon_2^2, \tag{5.87}$$

and using the Cauchy-Schwarz inequality,

$$\Big(\sum_{r=1}^{t} |\epsilon_{r,r}^{(t)}|\Big)^2 \leq \sum_{r=1}^{t} \left(\epsilon_{r,r}^{(t)}\right)^2 \sum_{r=1}^{t} 1 \leq t^3 \sigma^2 \epsilon_2^2. \tag{5.88}$$

Similarly, using (5.86) and the bound in (5.72),

$$|q_i^{(m)T} R_m q_i^{(m)}| \leq \frac{m^{5/2} \sigma \epsilon_2}{(5/2) \min_{\ell \neq i} |\mu_i^{(m)} - \mu_\ell^{(m)}|}. \tag{5.89}$$

This bound is weak, but it shows that if

$$\min_{\ell \neq i} |\mu_i^{(m)} - \mu_\ell^{(m)}| \geq m^{5/2} \sigma \epsilon_2, \tag{5.90}$$

then from (5.89), $|q_i^{(m)T} R_m q_i^{(m)}| \leq 2/5$, and substituting this into (5.82),

$$\big|\|z_i^{(m)}\|^2 - 1\big| \leq 2\big|q_i^{(m)T} R_m q_i^{(m)}\big| + \frac{\epsilon_0}{2} \leq \frac{4}{5} + \frac{\epsilon_0}{2}. \tag{5.91}$$

Thus with the condition that $\epsilon_0 = 2\epsilon(n + 11s + 15)\bar{\Gamma}_k^2 < 1/12$ (see (5.64)), we can then guarantee that

$$0.39 < \|z_i^{(m)}\| < 1.4, \tag{5.92}$$

[1]Note that these bounds differ from those given by Paige in [141], which are $0.42 < \|z_i^{(m)}\| < 1.4$; we suspect that the bounds in [141] were obtained using $\epsilon_0 < 1/100$ rather than the specified $\epsilon_0 < 1/12$, the former being the value used by Paige in his earlier work [138].

which has implications for (5.80).

Comments Note that we could slightly loosen the bound (5.64) on ϵ_0 and still carry through much of the preceding analysis, although in (5.91) we have assumed that $\epsilon_0/2 < 1/5$. If we instead have $\epsilon_0/2 \geq 1/5$, we get the trivial bound $0 \leq \|z_i^{(m)}\|^2$. This bound is not useful because in the worst case, $z_i^{(m)}$ is the 0-vector, which indicates either breakdown of the method or rank-deficiency of some $\hat{\mathcal{Y}}_k$.

Note that from (5.70) and (5.15),

$$\left| \sum_{i=1}^{m} \|z_i^{(m)}\|_2^2 - m \right| \leq \frac{m\epsilon_0}{2}.$$

It was also proven in [138] that if $\mu_i^{(m)}, \ldots, \mu_{i+c}^{(m)}$ are $c+1$ eigenvalues of $\hat{T}_m$ which are close to each other but separate from the rest, then

$$\sum_{\ell=i}^{i+c} \|z_\ell^{(m)}\|^2 \approx c+1. \tag{5.93}$$

This means that it is possible to have several close eigenvalues of $\hat{T}_m$ corresponding to one simple eigenvalue of A. In this case, the columns of $Z_c \equiv [z_i^{(m)}, \ldots, z_{i+c}^{(m)}]$ will all correspond to one eigenvector z of A having $z^T z = 1$. We now state another result.

Lemma 2. *Let $\hat{T}_m$ and $\hat{V}_m$ be the result of m steps of the CA-Lanczos method with* (5.17) *and* (5.67)*, and let R_m be the strictly upper triangular matrix defined in* (5.18)*. Then for each eigenpair $(\mu_i^{(m)}, q_i^{(m)})$ of $\hat{T}_m$, there exists a pair of integers (r,t) with $0 \leq r \leq t < m$ such that*

$$\delta_{t,r} \equiv \hat{\beta}_{t+1}|\eta_{t,r}^{(t)}| \leq \psi_{i,m} \quad \textit{and} \quad |\mu_i^{(m)} - \mu_r^{(t)}| \leq \psi_{i,m},$$

where

$$\psi_{i,m} \equiv \frac{m^2\sigma\epsilon_2}{\left|\sqrt{3}\, q_i^{(m)T} R_m q_i^{(m)}\right|}.$$

Proof. For $r \leq t < m$ we define, using (5.75),

$$\gamma_{r,t} \equiv (\hat{\beta}_{t+1}\eta_{t,r}^{(t)})^{-1} q_i^{(m)T} \begin{bmatrix} q_r^{(t)} \\ 0_{m-t,1} \end{bmatrix} = \frac{\eta_{t+1,i}^{(m)}}{\mu_i^{(m)} - \mu_r^{(t)}}. \tag{5.94}$$

Using this notation and (5.84), we can write

$$q_i^{(m)T} R_m q_i^{(m)} = -\sum_{t=1}^{m-1} \eta_{t+1,i}^{(m)} \sum_{r=1}^{t} \gamma_{r,t}\epsilon_{r,r}^{(t)} \equiv -e^T C \bar{q},$$

where above, e is the vector with every element unity, C is an $(m-1)$-by-$(m-1)$ upper triangular matrix with (r,t) element $\gamma_{r,t}\epsilon^{(t)}_{r,r}$, and $\bar{q}$ contains the last $m-1$ elements of $q_i^{(m)}$. Letting E be the $(m-1)$-square matrix with (r,t) element $\epsilon^{(t)}_{r,r}$ and combining this with (5.87) gives

$$|q_i^{(m)T} R_m q_i^{(m)}| \leq \|e^T C\|_2 \|\bar{q}\|_2 \leq \|C\|_2 \|e\|_2 \leq m^{1/2}\|C\|_F \leq m^{1/2} \max_{r\leq t<m} |\gamma_{r,t}| \cdot \|E\|_F. \quad (5.95)$$

Using (5.87), we can write

$$\|E\|_F^2 = \sum_{t=1}^{m-1}\sum_{r=1}^{t} \left(\epsilon^{(t)}_{r,r}\right)^2 \leq \sigma^2\epsilon_2^2 \sum_{t=1}^{m-1} t^2 \leq \frac{\sigma^2\epsilon_2^2 m^3}{3},$$

and thus we can take the square root above and substitute into (5.95) to get

$$|q_i^{(m)T} R_m q_i^{(m)}| \leq \frac{m^2\sigma\epsilon_2|\gamma_{r,t}|}{\sqrt{3}},$$

where we take the r and t giving the maximum value of $|\gamma_{r,t}|$. For this r and t, substituting in the expression for $\gamma_{r,t}$ from (5.94) into the bound above, rearranging, and using $\eta^{(m)}_{t+1,i} \leq 1$ then gives the desired results

$$\delta_{t,r} = \hat{\beta}_{t+1}|\eta^{(t)}_{t,r}| \leq \frac{m^2\sigma\epsilon_2}{\left|\sqrt{3}\, q_i^{(m)T} R_m q_i^{(m)}\right|} \quad \text{and} \quad (5.96)$$

$$|\mu_i^{(m)} - \mu_r^{(t)}| \leq \frac{m^2\sigma\epsilon_2}{\left|\sqrt{3}\, q_i^{(m)T} R_m q_i^{(m)}\right|}. \quad (5.97)$$

□

Comments For classical Lanczos, these bounds show that if $\|z_i^{(m)}\|_2$ is significantly different from unity, then for some $t < m$ there is an eigenvalue of $\hat{T}_t$ which has stabilized and is close to $\mu_i^{(m)}$ [141]. For the CA-Lanczos case, the same holds with the assumptions in (5.64). These assumptions are necessary because otherwise, for CA-Lanczos, $\|z_i^{(m)}\|$ can significantly differ from unity if $|q_i^{(m)T} R_m q_i^{(m)}|$ is large and/or if $\epsilon_0/2$ is large (due to a large $\bar{\Gamma}_k^2$, see (5.17)). If $\|z_i^{(m)}\|$ is much different from unity and $\epsilon_0/2$ is large, we can not necessarily say that there is an eigenvalue of $\hat{T}_t$ which has stabilized to within a meaningful bound even if $|q_i^{(m)T} R_m q_i^{(m)}|$ is small.

Theorem 2. *If, with the conditions of Lemma 2, an eigenvalue $\mu_i^{(m)}$ of $\hat{T}_m$ produced by CA-Lanczos is stabilized so that*

$$\delta_{m,i} \equiv \hat{\beta}_{m+1}|\eta^{(m)}_{m,i}| \leq \sqrt{3}m^2\sigma\epsilon_2, \quad (5.98)$$

and $\epsilon_0 < 1/12$*, then for some eigenvalue* λ_c *of* A*,*

$$|\lambda_c - \mu_i^{(m)}| \le (m+1)^3 \sigma \epsilon_2. \tag{5.99}$$

Proof. Suppose (5.98) holds.
(i) If

$$|q_i^{(m)T} R_m q_i^{(m)}| \le \frac{3}{8} - \frac{\epsilon_0}{2}, \tag{5.100}$$

then by (5.15) and (5.82) we have

$$\|z_i^{(m)}\| \ge \sqrt{1 - \left(2\left|q_i^{(m)T} R_m q_i^{(m)}\right| + \frac{\epsilon_0}{2}\right)} \ge \sqrt{\frac{1}{4} + \frac{\epsilon_0}{2}} \ge \frac{1}{2}. \tag{5.101}$$

Substituting (5.17) and (5.67) into (5.80), we obtain

$$\min_\ell |\lambda_\ell - \mu_i^{(m)}| \le \frac{\delta_{m,i}(1+\epsilon_0) + \sqrt{m}\sigma\epsilon_1}{\|z_i^{(m)}\|} \le \sigma\epsilon_2 \left(\frac{13\sqrt{3}}{6} \cdot m^2 + \sqrt{2m}\right).$$

Since $m \ge 1$, $m^3 \ge m^2$ and $m \ge \sqrt{m}$, and it follows that

$$(m+1)^3 \;=\; m^3 + 3m^2 + 3m + 1 \;>\; 4m^2 + 3\sqrt{m} \;\ge\; \frac{13\sqrt{3}}{6} m^2 + \sqrt{2m}.$$

From this it follows that (5.99) holds.

In the other case that (5.100) is false, take $\ell = 1$ and write

$$t_1 = m, \quad r_1 = i. \tag{5.102}$$

(ii) In this case we know from (5.96) and (5.97) that there exist positive integers $r_{\ell+1}$ and $t_{\ell+1}$ with

$$r_{\ell+1} \le t_{\ell+1} < t_\ell \tag{5.103}$$

such that

$$\max\left(\delta_{t_{\ell+1}, r_{\ell+1}}, |\mu_{r_\ell}^{t_\ell} - \mu_{r_{\ell+1}}^{t_{\ell+1}}|\right) \le \frac{t_\ell^2 \sigma \epsilon_2}{\sqrt{3}\left(\frac{3}{8} - \frac{\epsilon_0}{2}\right)} \le \frac{t_\ell^2 \sigma \epsilon_2}{\sqrt{3}\left(\frac{1}{3}\right)} \le \sqrt{3} t_\ell^2 \sigma \epsilon_2. \tag{5.104}$$

If the equivalent of (5.100), and thus (5.101), holds for $(r_{\ell+1}, t_{\ell+1})$, i.e.,

$$|q_{r_{\ell+1}}^{(t_{\ell+1})T} R_{t_{\ell+1}} q_{r_{\ell+1}}^{(t_{\ell+1})}| \le 3/8 - \epsilon_0/2,$$

then using (5.80), for some eigenvalue λ_c of A,

$$|\lambda_c - \mu_{r_{\ell+1}}^{(t_{\ell+1})}| \le 2\left(\sqrt{3} t_\ell^2 \sigma \epsilon_2 (1+\epsilon_0) + \sqrt{t_{\ell+1}}\sigma\epsilon_1\right) \le \left(\frac{13\sqrt{3} t_\ell^2}{6} + \sqrt{2t_{\ell+1}}\right)\sigma\epsilon_2,$$

which gives

$$\begin{aligned}
|\lambda_c - \mu_i^{(m)}| &\leq |\lambda_c - \mu_{r_{\ell+1}}^{(t_{\ell+1})}| + \sum_{p=1}^{\ell} |\mu_{r_p}^{(t_p)} - \mu_{r_{p+1}}^{(t_{p+1})}| \\
&\leq \left(\frac{13\sqrt{3}t_\ell^2}{6} + \sqrt{2t_{\ell+1}} + \sqrt{3}\sum_{p=1}^{\ell} t_p^2 \right) \sigma\epsilon_2 \\
&\leq \left(\frac{13\sqrt{3}m^2}{6} + \sqrt{2m} + \frac{\sqrt{3}m(m+1)(2m+1)}{6} \right) \sigma\epsilon_2 \\
&\leq (m+1)^3 \sigma\epsilon_2,
\end{aligned}$$

as required by (5.99), where the penultimate inequality follows from (5.102) and (5.103). If the equivalent of (5.100) does not hold, then replace ℓ by $\ell + 1$ and return to (ii).

We see that $\hat{T}_1 = \hat{\alpha}_1$, $q_1^{(1)} = 1$, so that $z_1^{(1)} = v_1$ satisfies (5.101), proving that we must encounter an $(r_{\ell+1}, t_{\ell+1})$ pair satisfying (5.101), which completes the proof. □

Comments As in [141], the bound (5.99) is not tight, and thus should in no way be considered an indication of the maximum attainable accuracy. In practice we can still observe convergence of the eigenvalues of T_m to eigenvalues of A with larger $\bar{\Gamma}_k^2$ than allowed by $\epsilon_0 < 1/12$. The constraint $\epsilon_0 \leq 1/12$ comes from the proof of the theorem above, which requires that $3/8 - \epsilon_0/2 \geq 1/3$. We believe that the restriction on the size of ϵ_0 could be loosened by a constant factor by changing the form of the right hand side of [141, Equation 3.37] such that meaningful bounds are still obtained. This remains future work.

The following shows that if (5.98) holds we have an eigenvalue with a superior error bound to (5.99) and that we also have a good eigenvector approximation.

Corollary 1. *If* (5.98) *holds, then for the final* (r, t) *pair in Theorem 2,* $(\mu_r^{(t)}, \hat{V}_t q_r^{(t)})$ *is an exact eigenpair for a matrix within* $6t^2\sigma\epsilon_2$ *of* A*.*

Proof. From Theorem 2, if there is an i, $1 \leq i \leq m$ such that (5.98) holds, then there exist r and t, $1 \leq r \leq t \leq m$ such that

$$\delta_{t,r} \leq \sqrt{3}t^2\sigma\epsilon_2 \quad \text{and} \quad \|z_r^{(t)}\| \geq \frac{1}{2},$$

and both $\mu_r^{(t)}$ and $\mu_i^{(m)}$ are close to the same eigenvalue of A. It follows from (5.79) that

$$(A + \delta A_r^{(t)}) z_r^{(t)} = \mu_r^{(t)} z_r^{(t)}, \text{ with} \tag{5.105}$$

$$\delta A_r^{(t)} \equiv -(\hat{\beta}_{t+1} \eta_{t,r}^{(t)} \hat{v}_{t+1} + \delta V_t q_r^{(t)}) \frac{z_r^{(t)T}}{\|z_r^{(t)}\|^2}, \text{ and}$$

$$\|\delta A_r^{(t)}\| \leq \left(|\delta_{t,r}| \cdot \|\hat{v}_{t+1}\| + \|\delta \hat{V}_t\| \cdot \|q_r^{(t)}\| \right) \frac{1}{\|z_r^{(t)}\|}$$

$$\leq 2 \left(\sqrt{3} t^2 \sigma \epsilon_2 (1 + \epsilon_0) + \sqrt{t} \sigma \epsilon_1 \right) \tag{5.106}$$

$$\leq 6 t^2 \sigma \epsilon_2, ^2 \tag{5.107}$$

where we have used (5.13), (5.15), and (5.67). So, $z_r^{(t)}$, which lies in the range of $\hat{V}_r$, is an exact eigenvector of a matrix close to A, and $\mu_r^{(t)}$ is the corresponding exact eigenvalue. □

As in the classical Lanczos case, the above is also the result we obtain for an eigenvalue of $\hat{T}_m$ produced by CA-Lanczos that is stabilized and well-separated.

Paige showed that one can also consider the accuracy of the $\mu_i^{(m)}$ as Rayleigh quotients [141]. With no rounding errors, $\mu_i^{(m)}$ is the Rayleigh quotient of A with $z_i^{(m)}$ and this gives the best bound from (5.79) and (5.80) with $\epsilon = 0$, i.e., in exact arithmetic. Here (5.78) and (5.79) can be combined to give

$$z_i^{(m)T} A z_i^{(m)} - \mu_i^{(m)} z_i^{(m)T} z_i^{(m)} = -\epsilon_{i,i}^{(m)} + z_i^{(m)T} \delta \hat{V}_m q_i^{(m)}, \tag{5.108}$$

so if $\|z_i^{(m)}\| \approx 1$, then $\mu_i^{(m)}$ is close to the Rayleigh quotient

$$\varrho_i^{(m)} = z_i^{(m)T} A z_i^{(m)} / z_i^{(m)T} z_i^{(m)}.$$

If (5.90) holds, then $\|z_i^{(m)}\| > 0.39$, and thus using (5.108) and the bounds in (5.13), (5.67), and (5.77),

$$|\varrho_i^{(m)} - \mu_i^{(m)}| \leq 9 m \sigma \epsilon_2.$$

If $\|z_i^{(m)}\|$ is small, then it is unlikely that $\mu_i^{(m)}$ will be very close to $\varrho_i^{(m)}$, since a small $z_i^{(m)}$ will probably be inaccurate due to rounding errors. The equation (5.93) suggests that at least one of a group of close eigenvalues will have corresponding $\|z_i^{(m)}\| \gtrsim 1$. In fact, using (5.85), (5.88), and an argument similar to that used in Theorem 2, it can be shown that every $\mu_i^{(m)}$ lies within $m^{5/2} \sigma \epsilon_2$ of a Rayleigh quotient of A, and so with (5.17) and (5.67), all the $\mu_i^{(m)}$ lie in the interval

$$\lambda_{\min} - m^{5/2} \sigma \epsilon_2 \leq \mu_i^{(m)} \leq \lambda_{\max} + m^{5/2} \sigma \epsilon_2. \tag{5.109}$$

This differs from the bound on the distance of $\mu_i^{(m)}$ from an eigenvalue of A in (5.99), which requires that $\mu_i^{(m)}$ has stabilized.

We emphasize that whatever the size of $\delta_{m,i}$, the eigenvalue $\mu_i^{(m)}$ of $\hat{T}_m$ with eigenvector $q_i^{(m)}$ has necessarily stabilized to within $\delta_{m,i} \equiv \hat{\beta}_{m+1}|e_m^T q_i^{(m)}|$. If $\mu_i^{(m)}$ is a separated eigenvalue of $\hat{T}_m$ so that (5.90) holds, then it follows from (5.79), (5.80), and (5.92) that the eigenpair $(\mu_i^{(m)}, \hat{V}_m q_i^{(m)})$ represents an eigenpair of A to within

$$3\big(\delta_{m,i} + \sqrt{m}\sigma\epsilon_1\big). \tag{5.110}$$

On the other hand, if $\mu_i^{(m)}$ is one of a close group of eigenvalues of $\hat{T}_m$, so that (5.90) does not hold, then we have found a good approximation to an eigenvalue of A. In this case either $(\mu_i^{(m)}, \hat{V}_m q_i^{(m)})$ represents an eigenpair of A to within (5.110) (see [141]), or there exists $1 \le r \le t < m$ such that

$$\max\Big(\delta_{t,r}, |\mu_i^{(m)} - \mu_r^{(t)}|\Big) \le \sqrt{3}m^2\sigma\epsilon_2, \tag{5.111}$$

as from Lemma 2. Then, it follows from Theorem 2 that $\mu_i^{(m)}$ is within $\big((m+1)^3 + \sqrt{3}m^2\big)\sigma\epsilon_2$ of an eigenvalue of A. The $\delta_{m,i}$ and $\mu_i^{(m)}$ can be computed from $\hat{T}_m$ efficiently, and these results show how we can obtain intervals from them which are known to contain eigenvalues of A, whether $\delta_{m,i}$ is large or small.

5.1.5 Convergence of Eigenvalues

Theorem 2 showed that, assuming (5.64) holds, if an eigenvalue of $\hat{T}_m$ has stabilized to within $\sqrt{3}m^2\sigma\epsilon_2$, then it is within $(m+1)^3\sigma\epsilon_2$ of an eigenvalue of A, regardless of how many other eigenvalues of $\hat{T}_m$ are close, and Corollary 1 showed we had an eigenpair of a matrix within $6m^2\sigma\epsilon_2$ of A. It is now shown that, assuming (5.64), eigenvalues do stabilize to this accuracy using the CA-Lanczos method, and we can specify how quickly this occurs.

In [138], it was shown that at least one eigenvalue of $\hat{T}_m$ must have stabilized by iteration $m = n$. This is based on (5.78), which indicates that significant loss of orthogonality implies stabilization of at least one eigenvalue. Using (5.77) and (5.83), if at step $m \le n$

$$\delta_{\ell,i} \equiv \hat{\beta}_{\ell+1}|\eta_{\ell,i}^{(\ell)}| \ge \sqrt{3}m^2\sigma\epsilon_2, \quad \text{where } 1 \le i \le \ell < m, \tag{5.112}$$

then we have

$$\|R_m\|_F^2 \le \frac{1}{3m^4}\sum_{t=1}^{m-1} t^3 \le \frac{1}{12}.$$

Let $\sigma_1 \ge \cdots \ge \sigma_m$ be the singular values of $\hat{V}_m$. A result of Rump [152, Lemma 2.2] states that given a matrix $X \in \mathbb{R}^{n\times m}$, if $\|I - X^T X\|_2 \le \alpha < 1$, then $\sqrt{1-\alpha} \le \sigma_i(X) \le \sqrt{1+\alpha}$, for $i \in \{1, \ldots, m\}$. Using the above bound with (5.15), (5.18), and (5.64), we have $\|I - \hat{V}_m^T \hat{V}_m\|_2 < 1$, and then with $\alpha = 2/\sqrt{12} + 1/24$, we apply Lemma 2.2 from [152] to

obtain the bounds

$$0.61 < \sigma_i(\hat{V}_m) < 1.3,^3 \quad \text{for} \quad i \in \{1, \ldots, m\}. \tag{5.113}$$

Note that if (5.112) does not hold, then we already have an eigenpair of a matrix close to A. If we now consider the $q_i^{(m)}$ that minimizes $\delta_{m,i}$ for $\hat{T}_m$, we see from (5.77), (5.78), and (5.83) that

$$\|\hat{\beta}_{m+1}\eta_{m,i}^{(m)}\hat{V}_m^T\hat{v}_{m+1}\| \leq m^{3/2}\sigma\epsilon_2. \tag{5.114}$$

Theorem 3. *For the CA-Lanczos method, if $n(3\epsilon_0 + \epsilon_1) \leq 1$ and $\epsilon_0 < 1/12$, then at least one eigenvalue of $\hat{T}_n$ must be within $(n+1)^3\sigma\epsilon_2$ of an eigenvalue of the $n \times n$ matrix A, and there exist $r \leq t \leq n$ such that $(\mu_r^{(t)}, z_r^{(t)})$ is an exact eigenpair of a matrix within $6t^2\sigma\epsilon_2$ of A.*

Proof. If (5.112) does not hold for $m = n$, then an eigenvalue has stabilized to that accuracy before $m = n$. Otherwise, (5.112) holds for $m = n$, so from (5.113), $\hat{V}_n$ is nonsingular, and then (5.114) shows that for the smallest $\delta_{n,i}$ of $\hat{T}_n$,

$$\delta_{n,i} \leq \frac{n^{3/2}\sigma\epsilon_2}{0.6} \leq \sqrt{3}n^2\sigma\epsilon_2$$

since $\|\hat{V}_m^T\hat{v}_{m+1}\| \geq \sigma_m\|\hat{v}_{m+1}\| > 0.6$ from (5.113) and (5.15). So at least one eigenvalue must have stabilized to within $\sqrt{3}m^2\sigma\epsilon_2$ by iteration $m = n$, and from Theorem 2 this eigenvalue must be within $(n+1)^3\sigma\epsilon_2$ of an eigenvalue of A. Furthermore, Corollary 1 shows that there is an exact eigenpair $(\mu_r^{(t)}, z_r^{(t)})$, $r \leq t \leq n$, of a matrix within $6t^2\sigma\epsilon_2$ of A. □

This shows that the CA-Lanczos algorithm gives at least one eigenvalue of A to high accuracy by iteration $m = n$, assuming restrictions on the sizes of ϵ_0 and ϵ_1.

We now extend Paige's results to specify how quickly we can expect to find eigenvalues and eigenvectors of A using the CA-Lanczos method in practice. We first consider the Krylov sequence on which the Lanczos algorithm and several other methods are based. For symmetric A, one way of using m steps of the Krylov sequence is to form an $n \times m$ matrix V whose columns span the range of

$$[v_1, Av_1, \ldots, A^{m-1}v_1] \tag{5.115}$$

and use the eigenvalues of

$$V^TAVq = \mu V^TVq \tag{5.116}$$

as approximations to some of the eigenvalues of A. The Lanczos algorithm with full reorthogonalization forms Krylov subspaces for a matrix very close to A with the eigenvalues of T being very close to those of (5.116) [141]. We now show how CA-Lanczos without full reorthogonalization parallels these results.

[3] Again, these bounds differ from those given in [141], which are $0.41 < \sigma_i(\hat{V}_m) < 1.6$; we suspect that the bounds in [141] were obtained by squaring both sides of the bound and using $\epsilon_0 < 1/100$ rather than $\epsilon_0 < 1/12$, the former being the value used in [138].

Theorem 4. *For m iterations of the CA-Lanczos method, with* (5.17), (5.67), *and m such that* (5.112) *holds, the m Lanczos vectors (columns of $\hat{V}_m$) span a Krylov subspace of a matrix within $(3m)^{1/2}\sigma\epsilon_2$ of A.*

Proof. From (5.9),

$$A\hat{V}_m = \hat{V}_m\hat{T}_m + \hat{\beta}_{m+1}\hat{v}_{m+1}e_m^T + \delta\hat{V}_m = \hat{V}_{m+1}\hat{T}_{m+1,m} + \delta\hat{V}_m,$$

where $\hat{T}_{m+1,m}$ is the matrix of the first m columns of $\hat{T}_{m+1}$. Then with (5.13), (5.67), and (5.113),

$$\begin{aligned}
(A+\delta A_m)\hat{V}_m &= \hat{V}_{m+1}\hat{T}_{m+1,m}, \text{ with} \\
\delta A_m &\equiv -\delta\hat{V}_m(\hat{V}_m^T\hat{V}_m)^{-1}\hat{V}_m^T, \quad \text{and} \\
\|\delta A_m\|_F &= \text{trace}(\delta A_m \delta A_m^T)^{1/2} \\
&= \left(\sum_{i=1}^{m} |(\delta\hat{V}_m(\hat{V}_m^T\hat{V}_m)^{-1}\delta\hat{V}_m^T)_{i,i}|\right)^{1/2} \\
&\leq (3m)^{1/2}\sigma\epsilon_2.
\end{aligned}$$

□

Comments This is analogous to the result of Paige for classical Lanczos: until an eigenvalue of $\hat{T}_{m-1}$ has stabilized, i.e., while (5.112) holds, the vectors $\hat{v}_1, \ldots, \hat{v}_{m+1}$ computed correspond to an exact Krylov sequence for the matrix $A + \delta A_m$. As a result of this, and since it follows from (5.79) that $(A+\delta A_r^{(t)})z_r^{(t)} = \mu_r^{(t)} z_r^{(t)}$, if we assume that the s-step bases generated in each outer loop are conditioned such that (5.64) holds, then the CA-Lanczos algorithm can be thought of as a numerically stable way of computing a Krylov sequence, at least until the corresponding Krylov subspace contains an exact eigenvector of a matrix within $6m^2\sigma\epsilon_2$ of A.

When $\hat{T}_m$ and $\hat{V}_m$ are used to solve the eigenproblem of A, if we follow (5.115) and (5.116), we want the eigenvalues and eigenvectors of $\hat{T}_m$ to be close to those of

$$\hat{V}_m^T A\hat{V}_m q = \mu\hat{V}_m^T\hat{V}_m q, \quad \text{where } q^T q = 1, \tag{5.117}$$

as would be the case with classical Lanczos with full reorthogonalization. If (5.112) holds, then the range of $\hat{V}_m$ is close to what we expect from the Lanczos method with full reorthogonalization, and thus the eigenvalues of (5.117) would be close (how close depends on the value of ϵ_2) to those obtained using full reorthogonalization.

Theorem 5. *If $\hat{V}_m$ comes from the s-step Lanczos method with* (5.17) *and* (5.67), *and* (5.112) *holds, then for any μ and q which satisfy* (5.117), *$(\mu, \hat{V}_m q)$ is an exact eigenpair for a matrix within $\left(2\delta + 2m^{1/2}\sigma\epsilon_2\right)$ of A, where*

$$\eta \equiv e_m^T q, \quad \delta \equiv \hat{\beta}_{m+1}|\eta|.$$

Proof. Define

$$r \equiv A\hat{V}_m q - \mu \hat{V}_m q. \tag{5.118}$$

Then

$$r = \hat{V}_m(\hat{T}_m - \mu I)q + \hat{\beta}_{m+1}\eta \hat{v}_{m+1} + \delta \hat{V}_m q,$$

where we have used (5.9). Since from (5.117), $\hat{V}_m^T r = 0$,

$$(\hat{T}_m - \mu I)q = -(\hat{V}_m^T \hat{V}_m)^{-1}\hat{V}_m^T(\hat{\beta}_{m+1}\eta \hat{v}_{m+1} + \delta \hat{V}_m q), \text{ and} \tag{5.119}$$

$$r = P_m(\hat{\beta}_{m+1}\eta \hat{v}_{m+1} + \delta \hat{V}_m q), \tag{5.120}$$

where $P_m = I - \hat{V}_m(\hat{V}_m^T \hat{V}_m)^{-1}\hat{V}_m^T$ is the projector orthogonal to the range of $\hat{V}_m$. Using (5.112) and (5.114), we can bound

$$\|\hat{V}_m^T \hat{v}_{m+1}\| \leq \frac{m^{3/2}\sigma\epsilon_2}{\sqrt{3}m^2\sigma\epsilon_2} \leq (3m)^{-1/2}. \tag{5.121}$$

We can also write

$$\|P_m \hat{v}_{m+1}\|^2 = \hat{v}_{m+1}^T P_m \hat{v}_{m+1} = \hat{v}_{m+1}^T \hat{v}_{m+1} - \hat{v}_{m+1}^T \hat{V}_m(\hat{V}_m^T \hat{V}_m)^{-1}\hat{V}_m^T \hat{v}_{m+1},$$

and then using (5.15), (5.113), and (5.121),

$$1 - \frac{\epsilon_0}{2} - \frac{1}{m} \leq 1 - \frac{\epsilon_0}{2} - \left(\frac{1}{\sqrt{3m}} \cdot \frac{1}{0.6^2} \cdot \frac{1}{\sqrt{3m}}\right) \leq \|P_m \hat{v}_{m+1}\|^2 \leq 1 + \frac{\epsilon_0}{2}. \tag{5.122}$$

Using (5.13), (5.67), (5.120), and (5.122), we can write the bound

$$\|r\| \leq \|P_m \hat{v}_{m+1}\| \cdot |\hat{\beta}_{m+1}\eta| + \|P_m\|\|\delta \hat{V}_m q\| \leq (1 + \epsilon_0)\,\delta + \frac{\sqrt{m}\sigma\epsilon_2}{\sqrt{2}}.$$

Then by (5.118),

$$(A - \delta A)\hat{V}_m q = \mu \hat{V}_m q, \quad \text{where} \quad \delta A \equiv \frac{r q^T \hat{V}_m^T}{\|\hat{V}_m q\|^2},$$

with

$$\|\delta A\|_F = \frac{\|r\|}{\|\hat{V}_m q\|} \leq \frac{1}{0.6}\left((1 + \epsilon_0)\,\delta + \frac{m^{1/2}\sigma\epsilon_2}{\sqrt{2}}\right) \leq 2\delta + 2m^{1/2}\sigma\epsilon_2, \tag{5.123}$$

where we have used (5.113). □

Since from (5.117), $\hat{V}_m^T r = 0$,

$$(\hat{T}_m - \mu I)q = -(\hat{V}_m^T \hat{V}_m)^{-1} \hat{V}_m^T (\hat{\beta}_{m+1} \eta \hat{v}_{m+1} + \delta \hat{V}_m q). \tag{5.124}$$

Then ordering the eigenvalues of $\hat{T}_m$ such that $\delta_{m,1} \geq \delta_{m,2} \geq \cdots \geq \delta_{m,m}$, and assuming (5.112) holds for $\ell = m$, then for any eigenpair of (5.117), (5.124) gives, using (5.13), (5.67), (5.113), and (5.114),

$$\|\hat{T}_m q - \mu q\| \leq \left(2 + \frac{3m\delta}{\delta_{m,m}}\right) m^{1/2} \sigma \epsilon_2. \tag{5.125}$$

From this we can write

$$\left(2 + \frac{3m\delta}{\delta_{m,m}}\right) m^{1/2} \sigma \epsilon_2 \geq \|\hat{T}_m q - \mu q\|_2 \geq \min_i |\mu_i^{(m)} - \mu|.$$

Then, from (5.112), $\delta_{m,m} \geq \sqrt{3} m^2 \sigma \epsilon_2$, and thus

$$|\mu_x^{(m)} - \mu| \equiv \min_i |\mu_i^{(m)} - \mu| \leq 2m^{1/2} \sigma \epsilon_2 + \frac{\sqrt{3}\delta}{\sqrt{m}}. \tag{5.126}$$

Then, for any $t > m$,

$$\hat{T}_t \begin{bmatrix} q \\ 0_{t-m,1} \end{bmatrix} = \begin{bmatrix} \hat{T}_m q \\ \hat{\beta}_{m+1} \eta e_1 \end{bmatrix},$$

and together with (5.125),

$$\min_i |\mu_i^{(t)} - \mu| \leq 2m^{1/2} \sigma \epsilon_2 + \delta \left(1 + \frac{3}{m}\right)^{1/2}. \tag{5.127}$$

Equations (5.126) and (5.127) can then be combined to give

$$\min_i |\mu_i^{(t)} - \mu_x^{(m)}| \leq 4m^{1/2} \sigma \epsilon_2 + 4\delta.$$

Thus, assuming ϵ_2 is small enough, an eigenvalue of $\hat{T}_m$ close to μ has stabilized to about 4δ, where μ is within 2δ of an eigenvalue of A (see [141, Theorem 4.3]).

It can also be shown that for each $\mu_i^{(m)}$ of $\hat{T}_m$,

$$\min_{\mu \text{ in } (5.117)} |\mu - \mu_i^{(m)}| \leq 2m^{1/2} \sigma \epsilon_2 + \frac{\sqrt{3}\delta_{m,i}}{\sqrt{m}}.$$

This means that when $(\mu_i^{(m)}, \hat{V}_m q_i^{(m)})$ represents an eigenpair of A to within about $\delta_{m,i}$, there is a μ of (5.117) within about $\delta_{m,i}$ of $\mu_i^{(m)}$, assuming $m \geq 3$.

Thus, assuming no breakdown occurs and the size of $\bar{\Gamma}_k$ satisfies (5.64), these results say the same thing for the CA-Lanczos case as in the classical Lanczos case: *until an eigenvalue has stabilized, the CA-Lanczos algorithm behaves very much like the error-free Lanczos process, or the Lanczos algorithm with reorthogonalization.*

5.1.6 Numerical Experiments

In this section, we perform a variety of numerical experiments to demonstrate the proven bounds for CA-Lanczos. We first illustrate our two definitions for $\bar{\Gamma}_k$ and the implications these values have for the convergence of Ritz values. The test problem we will use for all tests is a diagonal matrix with $n = 100$ with eigenvalues evenly spaced between $\lambda_{\min} = 0.1$ and $\lambda_{\max} = 100$ (note that this is the 'Strakoš' matrix test problem originally from [170] with parameters $n = 100$, $\lambda_1 = 0.1$, $\lambda_n = 100$, and $\rho = 1.0$). We use a random starting vector with entries uniformly distributed between $[0, 1]$; the same starting vector is used in all tests.

In each test, the method was run for 100 iterations. At iteration $m = 100$, we compute the Ritz values (eigenvalues of $\hat{T}_m$) and plot these along with the true eigenvalues of A. We also plot the bounds on the interval containing the Ritz values given by (5.109), i.e., $\lambda_{\min} - m^{5/2}\sigma\epsilon_2 \leq \mu_i^{(m)} \leq \lambda_{\max} + m^{5/2}\sigma\epsilon_2$. For reference, in Figure 5.20, we show the results for the classical Lanczos method. The Ritz values are plotted as blue dots, the eigenvalues of A are plotted as red tick marks, and the dashed vertical lines mark the bounds $\lambda_{\min} - m^{5/2}\sigma\epsilon_2$ and $\lambda_{\max} + m^{5/2}\sigma\epsilon_2$. We see here that Paige's analysis leads to tight bounds on the possible range of Ritz values for this test problem. It is also worth noting that from the scatter plot, we can see that many of the extremal eigenvalues are well approximated by the Ritz values produced by classical Lanczos, although some eigenvalues in the interior of the spectrum are not well approximated by any Ritz value.

In Figures 5.21, 5.22, and 5.23, we show the same data for CA-Lanczos tests for both the monomial and Chebyshev bases, with $s = 2$, $s = 4$, and $s = 12$, respectively. In the top row of each plot, we show data illustrating the impact of the basis condition number on our error bounds in this section; for CA-Lanczos with the monomial basis (left) and the Chebyshev basis (right), the blue lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1 and the red lines show the value of $\bar{\Gamma}_{k,j}^2$ as defined in (5.61) which allow for tighter bounds (note that the results of Sections 5.1.4 and 5.1.5 only depend on assumptions about value of ϵ_0 in (5.15), so we need not consider the other definitions of $\bar{\Gamma}_k$ in (5.62) and (5.63)). The black dotted lines in these plots show the maximum allowable value of $\bar{\Gamma}_k^2$ such that Paige's results hold for the CA-Lanczos case, as defined in (5.65), i.e., $\bar{\Gamma}_k^2 < (24\epsilon(n + 11s + 15))^{-1}$. Of the two scatter plots shown in each figure, the topmost corresponds to the monomial basis and the other corresponds to the Chebyshev basis. Again, the dashed vertical lines mark the bounds $\lambda_{\min} - m^{5/2}\sigma\epsilon_2$ and $\lambda_{\max} + m^{5/2}\sigma\epsilon_2$, where ϵ_2 is computed using the corresponding value of $\bar{\Gamma}_{k,j}^2$ (the red lines) in iteration $m = 100$.

In Figure 5.21 for CA-Lanczos with $s = 2$, we see that $\bar{\Gamma}_{k,j}^2$ remains well below the black dotted line for both the monomial and Chebyshev bases, and the bounds on the range of Ritz values are tight. As s is increased to $s = 4$ we see from Figure 5.22 that the value of $\bar{\Gamma}_{k,j}^2$ becomes larger for the monomial basis, and this results in looser, but still valid bounds as indicated by the dashed vertical lines in the corresponding scatter plot. For $s = 12$ with the monomial basis (Figure 5.23) however, we see from the top left plot that $\bar{\Gamma}_{k,j}^2$ exceeds the black dotted line, and thus the assumptions in (5.65) no longer hold, which indicates

that the bounds on the range of Ritz values are no longer reliable. Indeed we can see from the corresponding scatter plot that Ritz values appear outside of these bounds. The $\bar{\Gamma}^2_{k,j}$ values for the Chebyshev basis for $s = 4$ and $s = 12$ remain relatively smaller, indicating that the Chebyshev basis remains well-conditioned in these cases, and thus the bounds on the possible range of Ritz values shown in the scatter plots remain relatively tight.

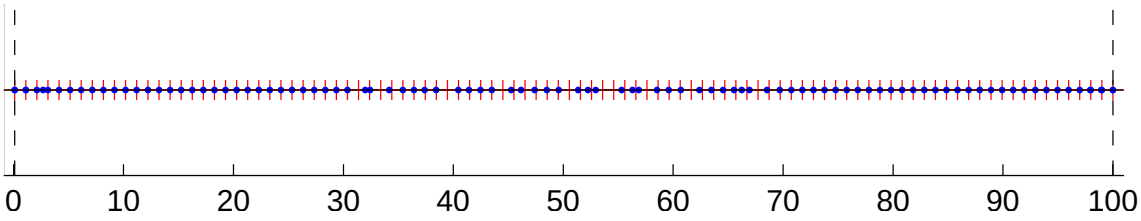

Figure 5.20: Ritz values and bounds for classical Lanczos after 100 iterations for the 'Strakoš' problem. Blue dots shows Ritz value computed after 100 iterations of classical Lanczos, red tick marks denote the true eigenvalues of A, and dashed vertical lines show the bounds on the range of computed Ritz values given by Paige [141].

For the same test problem, we show the convergence of Ritz values to eigenvalues of A in Figures 5.24-5.27. For each test, we show three plots. On the left, for each iteration m we plot values $|\hat{\beta}_{m+1}\eta^{(m)}_{m,i}|$ for each $i \in \{1, \ldots, m\}$, which are the error bounds in exact arithmetic, i.e., $\min_c |\lambda_c - \mu^{(m)}_i| \leq |\hat{\beta}_{m+1}\eta^{(m)}_{m,i}|$ (this can be seen by taking $\epsilon = 0$ in (5.80)). The middle plots show the error bounds on $\min_c |\lambda_c - \mu^{(m)}_i|$ in finite precision: from (5.110) and (5.111), we have that

$$\min_c |\lambda_c - \mu^{(m)}_i| \leq \max\left(3(|\hat{\beta}_{m+1}\eta^{(m)}_{m,i}| + \sqrt{m}\sigma\epsilon_1), ((m+1)^3 + \sqrt{3}m^2)\sigma\epsilon_2\right), \tag{5.128}$$

where for classical Lanczos plots the values of ϵ_1 and ϵ_2 used are those given by Paige [141] and for the CA-Lanczos plots the values of ϵ_1 and ϵ_2 are computed using the tighter definition of $\bar{\Gamma}^2_{k,j}$ given in (5.61). On the right, we show the actual values of $\min_c |\lambda_c - \mu^{(m)}_i|$ for each Ritz value in each iteration. Values reported as zeroes by Matlab were flushed to the value 10^{-19} in the plots in order to be visible on the log scale.

Figure 5.24 shows these plots for the classical Lanczos method, using the bounds of Paige [141]. In Figures 5.25, 5.26, and 5.27, we show the same plots for the CA-Lanczos method with $s = 2$, $s = 4$, and $s = 12$, respectively. In each figure for CA-Lanzos, we show results for both the monomial basis (top row) and the Chebyshev basis (bottom row).

The first thing to notice in Figure 5.24 is that, as mentioned by Paige, even for the classical Lanczos method the finite precision bounds should not be interpreted as an indication of attainable accuracy. In fact the convergence of the Ritz values to eigenvalues of A more closely follows the exact arithmetic error bounds. Looking at Figures 5.25- 5.27, we can see that this is also the case for CA-Lanczos, and in fact, as $\bar{\Gamma}^2_{k,j}$ becomes larger, the finite precision bounds are even worse indicators of actual accuracy. As long as $\bar{\Gamma}^2_{k,j}$ remains below

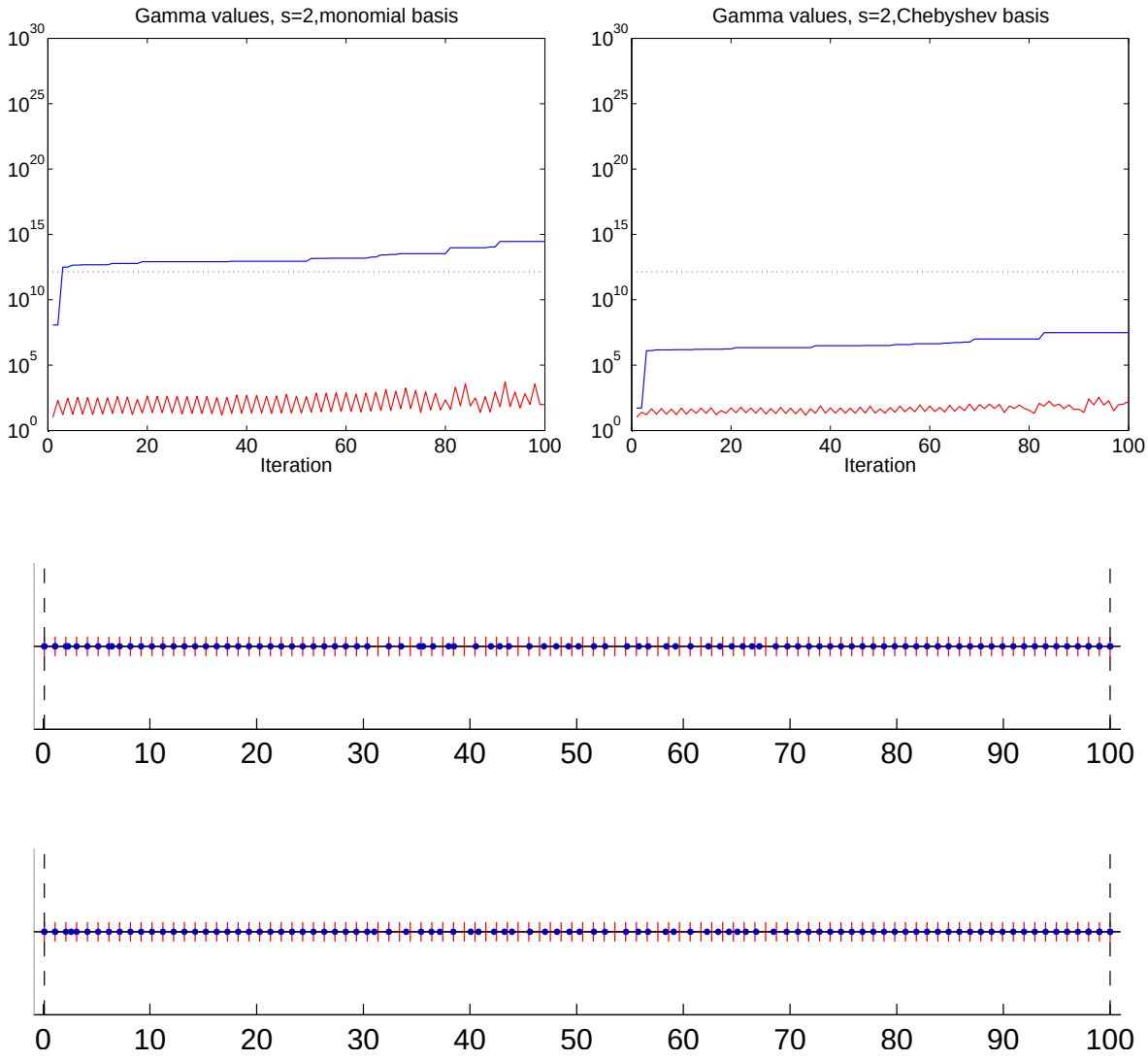

Figure 5.21: Ritz values and bounds for CA-Lanczos with $s = 2$ after 100 iterations for the 'Strakoš' problem. Plots in the top row show the values of $\bar{\Gamma}_k^2$ as defined in Theorem 1 (blue line) and the values of $\bar{\Gamma}_{k,j}^2$ as defined in (5.61) (red line) throughout the iterations for the monomial basis (left) and the Chebyshev basis (right). The black dotted lines plot the bound in (5.65). Scatter plots below these plots show the corresponding Ritz values computed at iteration $m = 100$ for the monomial basis (topmost) and the Chebyshev basis (bottommost). Blue dots shows Ritz value computed after 100 iterations, red tick marks denote the true eigenvalues of A, and dashed vertical lines show the bounds on the range of computed Ritz values given by (5.109).

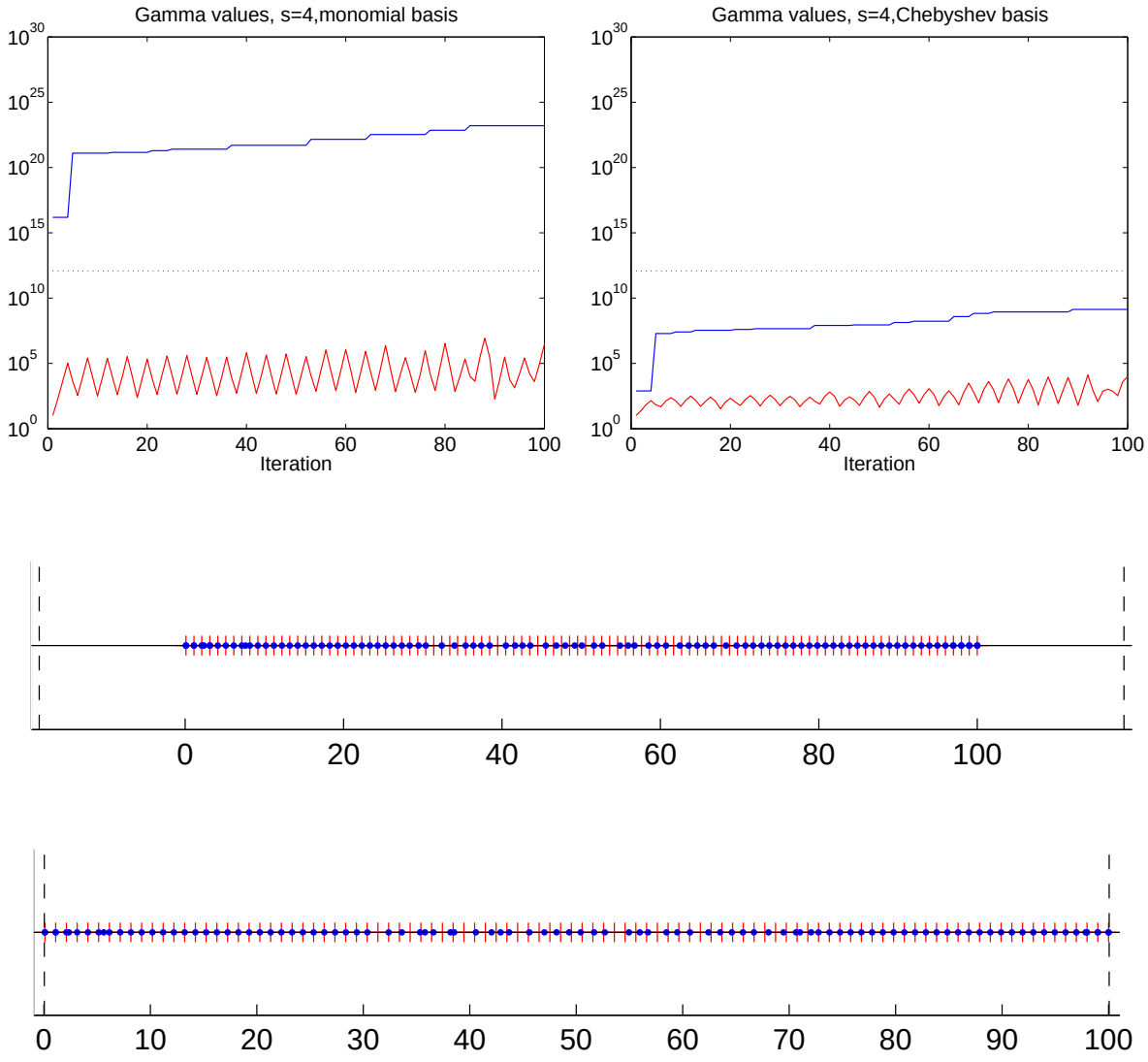

Figure 5.22: Ritz values and bounds for CA-Lanczos with $s = 4$ after 100 iterations for the 'Strakoš' problem. Plots in the top row show the values of $\bar{\Gamma}_k^2$ as defined in Theorem 1 (blue line) and the values of $\bar{\Gamma}_{k,j}^2$ as defined in (5.61) (red line) throughout the iterations for the monomial basis (left) and the Chebyshev basis (right). The black dotted lines plot the bound in (5.65). Scatter plots below these plots show the corresponding Ritz values computed at iteration $m = 100$ for the monomial basis (topmost) and the Chebyshev basis (bottommost). Blue dots shows Ritz value computed after 100 iterations, red tick marks denote the true eigenvalues of A, and dashed vertical lines show the bounds on the range of computed Ritz values given by (5.109).

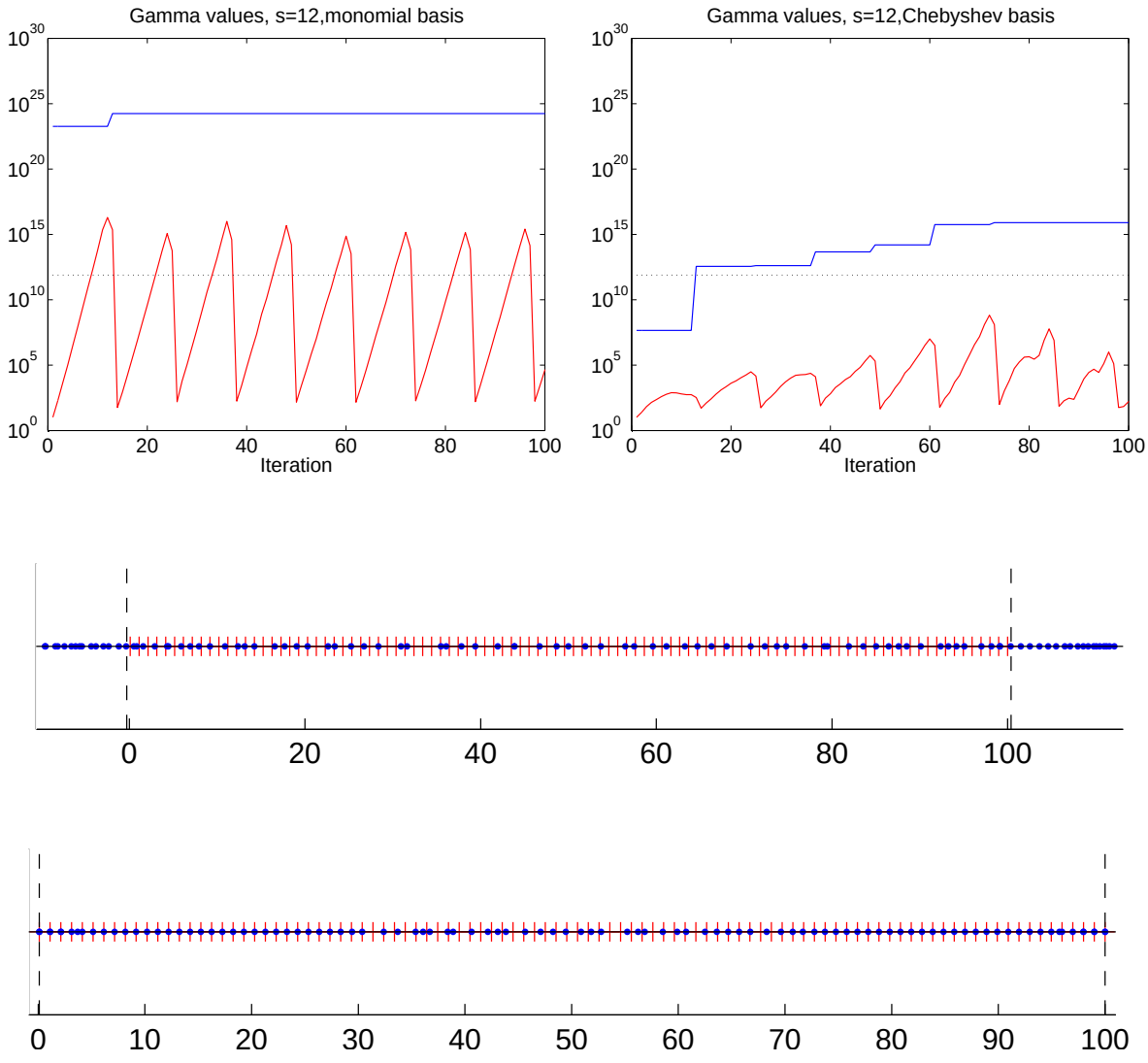

Figure 5.23: Ritz values and bounds for CA-Lanczos with $s = 12$ after 100 iterations for the 'Strakoš' problem. Plots in the top row show the values of $\bar{\Gamma}_k^2$ as defined in Theorem 1 (blue line) and the values of $\bar{\Gamma}_{k,j}^2$ as defined in (5.61) (red line) throughout the iterations for the monomial basis (left) and the Chebyshev basis (right). The black dotted lines plot the bound in (5.65). Scatter plots below these plots show the corresponding Ritz values computed at iteration $m = 100$ for the monomial basis (topmost) and the Chebyshev basis (bottommost). Blue dots shows Ritz value computed after 100 iterations, red tick marks denote the true eigenvalues of A, and dashed vertical lines show the bounds on the range of computed Ritz values given by (5.109).

the level indicated by (5.65), which is the case for all CA-Lanczos tests except the monomial basis with $s = 12$ in Figure 5.27, the exact arithmetic error bounds generally give a better estimate of accuracy of the computed Ritz values.

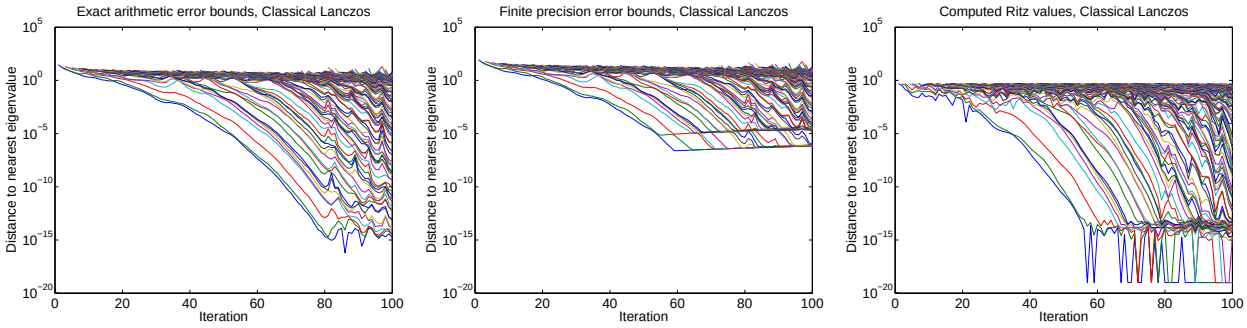

Figure 5.24: Ritz value convergence and convergence bounds for classical Lanczos for the 'Strakoš' problem. The left plot shows the exact error bounds $|\hat{\beta}_{m+1}\eta^{(m)}_{m,i}|$ for each $i \in \{1, \ldots, m\}$, the middle plot shows the finite precision error bounds as given by Paige [141], and the right plot shows the the actual values of $\min_c |\lambda_c - \mu^{(m)}_i|$ for each Ritz value in each iteration, where values measured as 0 have been flushed to 10^{-19}.

This same data is visualized in another way in Figure 5.28. In this figure, plots on the left show lower bounds on the number of Ritz values that have converged to within a given tolerance according to the exact arithmetic error bounds. That is, for each iteration, we plot $|\{|\hat{\beta}_{m+1}\eta^{(m)}_{m,i}| \leq \text{tol} : 1 \leq i \leq m\}|$. Plots on the right show the actual number of Ritz values that have converged to within a given tolerance, i.e., for each iteration we plot $|\{\min_c |\lambda_c - \mu^{(m)}_i| \leq \text{tol} : 1 \leq i \leq m\}|$. Note that this may count multiple copies of a single eigenvalue.

We plot results for two different tolerance values, $\text{tol} = 10^{-12}$ (solid lines) and $\text{tol} = 10^{-8}$ (dashed lines), for the classical Lanczos method ('CL', black) and the CA-Lanczos method with the monomial basis ('CA-M', blue) and the Chebyshev basis ('CA-C', red). We have generated these same plots for $s = 2$ (top row), $s = 4$ (middle row), and $s = 12$ (bottom row) (note that the plotted black lines for classical Lanczos are the same in each row). For $s = 2$, we can see from plots on the right that CA-Lanczos with both the monomial and Chebyshev bases finds about the same number of eigenvalues to high accuracy after 100 iterations. For $s = 4$, the CA-Lanczos method with both the monomial and Chebyshev basis find about the same number of eigenvalues to the accuracy of 10^{-8} as the classical method. However, use of the monomial basis prohibits the computation of any eigenvalues to high accuracy in this case, whereas the CA-Lanczos method with the Chebyshev basis still finds about as many eigenvalues to high accuracy as the classical method. For the higher value $s = 12$, the Chebyshev basis finds almost as many eigenvalues to the accuracy of 10^{-8} as the classical Lanczos method, and finds about a third of those found by the classical Lanczos method to high accuracy. Use of the monomial basis with $s = 12$ results in no eigenvalues found to either tolerance level.

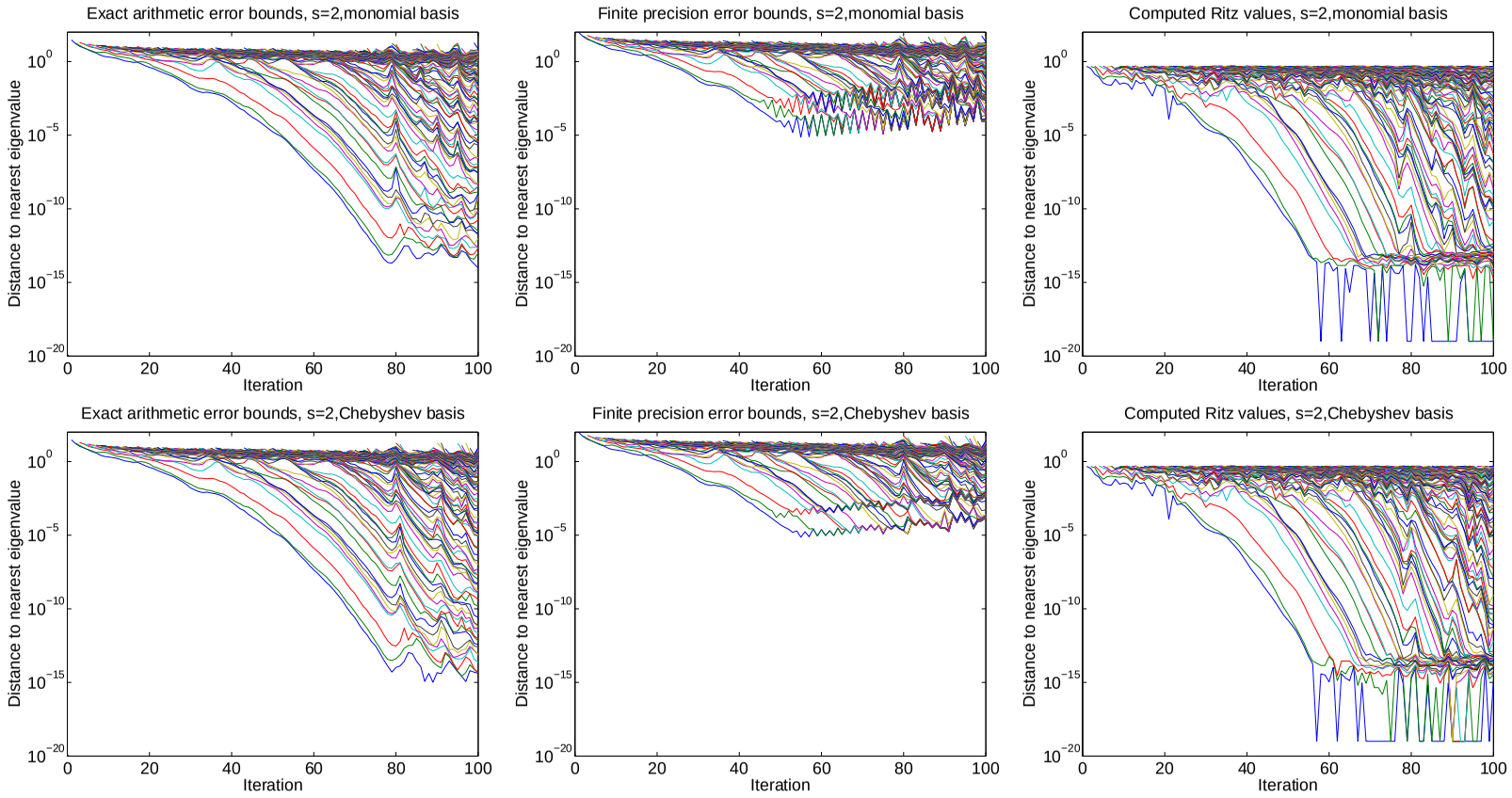

Figure 5.25: Ritz value convergence and convergence bounds for CA-Lanczos with $s = 2$ for the 'Strakoš' problem, for the monomial basis (top row) and the Chebyshev basis (bottom row). The left plots show the exact error bounds $|\hat{\beta}_{m+1}\eta_{m,i}^{(m)}|$ for each $i \in \{1, \ldots, m\}$, the middle plots show the finite precision error bounds as given in (5.128), and the right plot shows the the actual values of $\min_c |\lambda_c - \mu_i^{(m)}|$ for each Ritz value in each iteration, where values measured as 0 have been flushed to 10^{-19}.

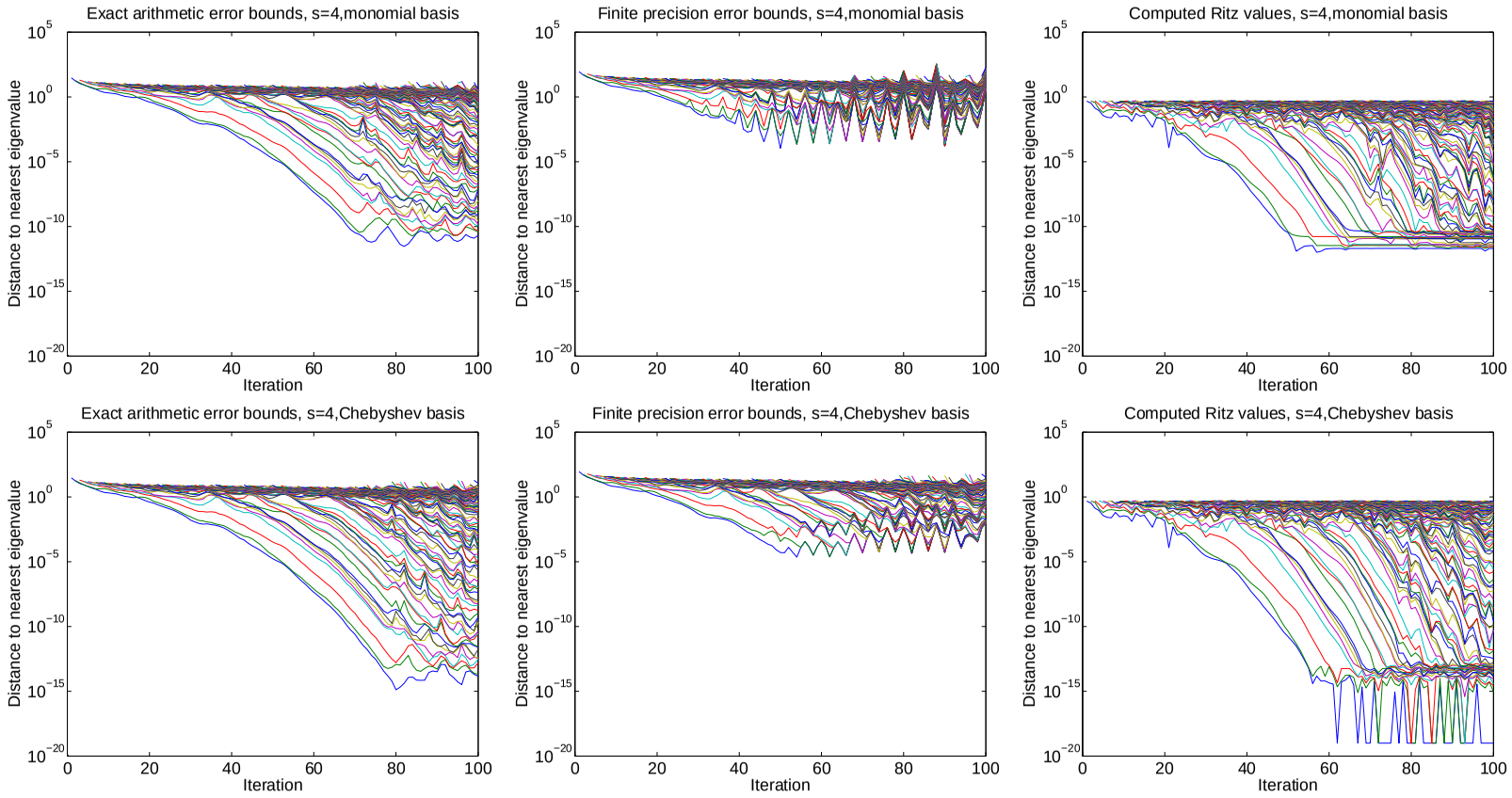

Figure 5.26: Ritz value convergence and convergence bounds for CA-Lanczos with $s = 4$ for the 'Strakoš' problem, for the monomial basis (top row) and the Chebyshev basis (bottom row). The left plots show the exact error bounds $|\hat{\beta}_{m+1}\eta_{m,i}^{(m)}|$ for each $i \in \{1, \ldots, m\}$, the middle plots show the finite precision error bounds as given in (5.128), and the right plot shows the the actual values of $\min_c |\lambda_c - \mu_i^{(m)}|$ for each Ritz value in each iteration, where values measured as 0 have been flushed to 10^{-19}.

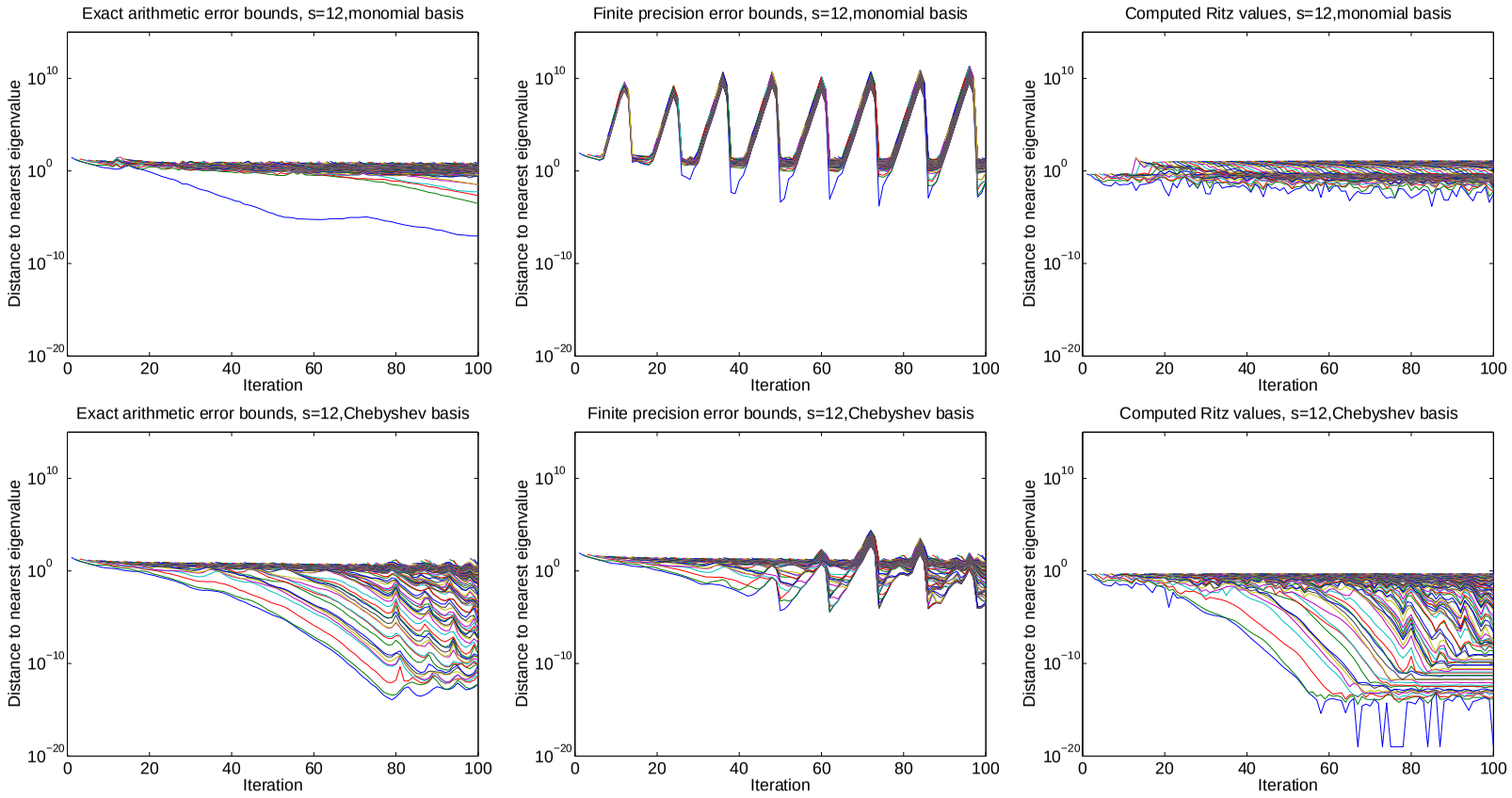

Figure 5.27: Ritz value convergence and convergence bounds for CA-Lanczos with $s = 12$ for the 'Strakoš' problem, for the monomial basis (top row) and the Chebyshev basis (bottom row). The left plots show the exact error bounds $|\hat{\beta}_{m+1}\eta^{(m)}_{m,i}|$ for each $i \in \{1, \ldots, m\}$, the middle plots show the finite precision error bounds as given in (5.128), and the right plot shows the the actual values of $\min_c |\lambda_c - \mu^{(m)}_i|$ for each Ritz value in each iteration, where values measured as 0 have been flushed to 10^{-19}.

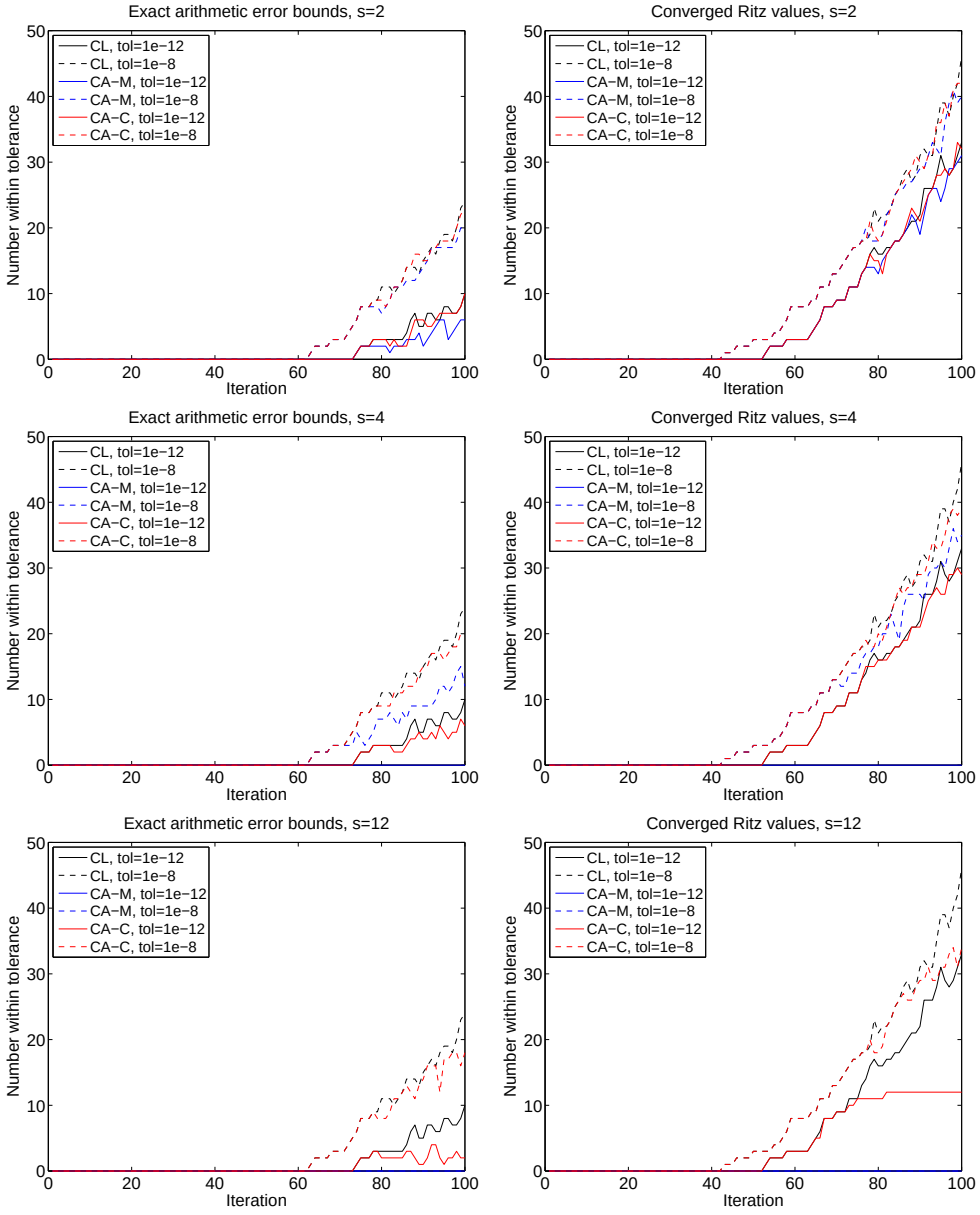

Figure 5.28: Number of converged Ritz values and lower bounds for the 'Strakoš' problem. Plots on the left show lower bounds on the number of Ritz values that have converged to within a given tolerance according to the exact arithmetic error bounds, i.e., $|\{|\hat{\beta}_{m+1}\eta_{m,i}^{(m)}| \leq \text{tol} : 1 \leq i \leq m\}|$. Plots on the right show the actual number of Ritz values that have converged to within a given tolerance, i.e., $|\{\min_c |\lambda_c - \mu_i^{(m)}| \leq \text{tol} : 1 \leq i \leq m\}|$. Results are shown for two different tolerance values, $\text{tol} = 10^{-12}$ (solid lines) and $\text{tol} = 10^{-8}$ (dashed lines), for the classical Lanczos method ('CL', black) and the CA-Lanczos method with the monomial basis ('CA-M', blue) and the Chebyshev basis ('CA-C', red). For CA-Lanczos we show tests for $s = 2$ (top row), $s = 4$ (middle row), and $s = 12$ (bottom row).

We show one more experiment, again using the same matrix and starting vector, in order to demonstrate the connection between the loss of orthogonality among the Lanczos vectors and the convergence of Ritz values in the CA-Lanczos method. Recall that by (5.78), the loss of orthogonality of the $(m+1)$-st Lanczos vector can be expressed as $z_i^{(m)T}\hat{v}_{m+1} = -\epsilon_{i,i}^{(m)}/\hat{\beta}_{m+1}\eta_{m,i}^{(m)}$, where $\epsilon_{i,i}^{(m)} \leq m\sigma\epsilon_2$. Figure 5.29 shows results for the classical Lanczos method. For each iteration m, the red line shows the maximum loss of orthogonality, $\max_i |z_i^{(m)T}\hat{v}_{m+1}|$, and the blue line shows $\min_i |\hat{\beta}_{m+1}\eta_{m,i}^{(m)}|$. The black solid line shows $\max_i |\hat{\beta}_{m+1}\eta_{m,i}^{(m)} z_i^{(m)T}\hat{v}_{m+1}| = \max_i |\epsilon_{i,i}^{(m)}|$, and the dashed black line plots $m\sigma\epsilon_2$, which by (5.78) and (5.77) is a bound for the solid black line. For classical Lanczos we use the value of ϵ_2 given by Paige [141]. Paige's fundamental result [141] that loss of orthogonality corresponds to convergence of a Ritz value is clearly seen in Figure 5.29.

We plot the same data for CA-Lanczos with $s = 2$, $s = 4$, $s = 8$, and $s = 12$ using both the monomial basis (Figure 5.30) and the Chebyshev basis (Figure 5.31). For the CA-Lanczos plots we use the value of ϵ_2 computed using the tighter definition of $\bar{\Gamma}_{k,j}^2$ given in (5.61). For reference, plots on the left show the corresponding values of the two definitions of $\bar{\Gamma}_k^2$ throughout the iterations; these are the same plots that appeared in Figures 5.21-5.23 (except for the case $s = 8$). As a reminder, in plots on the left, the blue lines show the value of $\bar{\Gamma}_k^2$ as defined in Theorem 1 and the red lines show the value of $\bar{\Gamma}_{k,j}^2$ as defined in (5.61), and the black dotted lines show the maximum allowable value of $\bar{\Gamma}_k^2$ as defined in (5.65), i.e., $\bar{\Gamma}_k^2 < (24\epsilon(n+11s+15))^{-1}$.

In Figure 5.30, we can see that the larger the value of $\bar{\Gamma}_{k,j}^2$ grows, the sooner orthogonality is lost, and that this corresponds to convergence of a Ritz value to a larger tolerance level. From the plots for $s = 12$, it is clear that as soon as $\bar{\Gamma}_{k,j}^2$ grows above the level indicated in (5.65), the result that loss of orthogonality implies convergence of a Ritz value no longer holds; around the same iteration where $\bar{\Gamma}_{k,j}^2$ exceeds the bound indicated by the black dotted line, orthogonality is immediately lost although no Ritz values have converged. In Figure 5.31, we see that $\bar{\Gamma}_{k,j}^2$ remains small indicating that the Chebyshev basis remains well conditioned, and thus despite the larger value of the bound $m\sigma\epsilon_2$, loss of orthogonality corresponds to convergence of a Ritz value to around the same accuracy as in the classical case and this occurs around the same iteration as for classical Lanczos.

These results give us hope towards the development of methods for ensuring good numerical properties in CA-Lanczos and other CA-KSMs by maintaining a small $\bar{\Gamma}_{k,j}$ value, which as these tests show, is usually a good indication of whether or not the results of Paige apply. As mentioned, the definition leading to tighter bounds in (5.61) can be computed within each inner loop iteration for the cost of (at most) one extra reduction per outer loop by computing $|\hat{\mathcal{Y}}_k|^T|\hat{\mathcal{Y}}_k|$, which can potentially be performed simultaneously with the computation of $\hat{G}_k$ (line 4 in Algorithm 26). Thus the value of $\bar{\Gamma}_{k,j}^2$ (the red lines) could be cheaply monitored during the iterations. Some potential ideas on how to enforce the restriction (5.65) are discussed further in Sections 6.7 and 6.5.

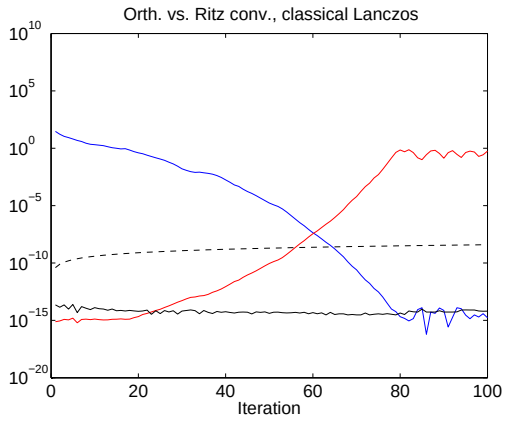

Figure 5.29: Loss of orthogonality and convergence of Ritz values for classical Lanczos for the 'Strakoš' problem. The red line shows the maximum loss of orthogonality, $\max_i |z_i^{(m)T} \hat{v}_{m+1}|$, the blue line shows $\min_i |\hat{\beta}_{m+1} \eta_{m,i}^{(m)}|$, the black solid line shows $\max_i |\hat{\beta}_{m+1} \eta_{m,i}^{(m)} z_i^{(m)T} \hat{v}_{m+1}|$, and the dashed black line plots $m\sigma\epsilon_2$.

5.1.7 Conclusions

In this section, we have presented a complete rounding error analysis of the CA-Lanczos method. The derived bounds are analogous to those of Paige for classical Lanczos [140], but also depend on a amplification factor $\bar{\Gamma}_k^2$, which depends on the condition number of the computed s-step Krylov bases.

We have further shown that the results of Paige for classical Lanczos [141] also apply to the CA-Lanczos method as long as the computed s-step bases remain well-conditioned. As in the classical Lanczos case, the upper bounds in this paper and in [30] are likely large overestimates. We stress, as did Paige, that the value of these bounds is in the *insight* they give rather than their tightness. In practice, we have observed that accurate eigenvalue estimates of A can be found with much looser restrictions than indicated by (5.64), and in some cases even in spite of a numerically rank-deficient basis.

Our analysis and extension of Paige's results confirms the observation that the conditioning of the Krylov bases plays a large role in determining finite precision behavior, and also indicates that the s-step method can be made suitable for practical use in many cases, offering both speed and accuracy. The next step is to extend the subsequent analyses of Paige, in which a type of augmented backward stability for the classical Lanczos method is proved [142]. In [12], Bai proves a result for nonsymmetric Lanczos analogous to the results of Paige, namely, that if a Ritz value is well-conditioned, convergence implies loss of orthogonality. We conjecture that our results for CA-Lanczos could be extended in the same way to the nonsymmetric CA-Lanczos method.

This analysis also inspires the development of practical techniques for improving CA-Lanczos based on our results. In Chapter 6, we discuss potential ways of controlling the

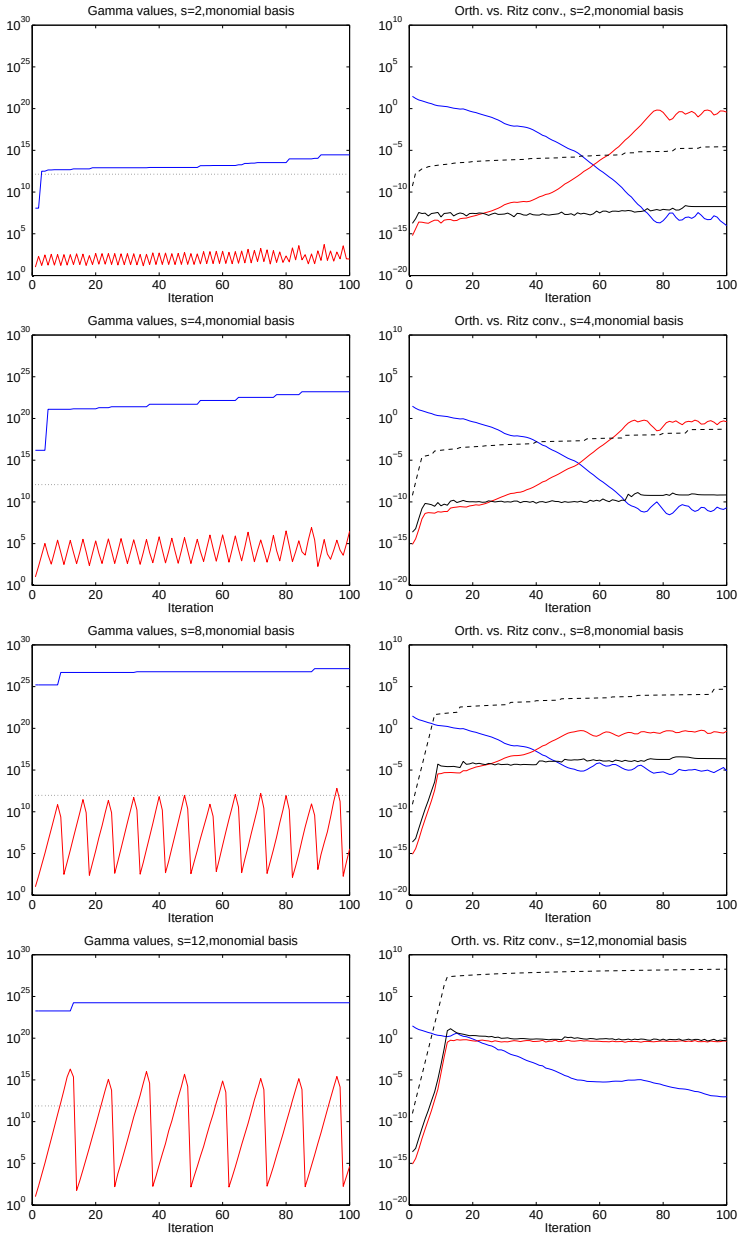

Figure 5.30: Loss of orthogonality and convergence of Ritz values for CA-Lanczos with the monomial basis for the 'Strakoš' problem, for $s = 2$ (top row), $s = 4$ (second row from the top), $s = 8$ (second row from the bottom), and $s = 12$ (bottom row). For plots on the left, blue lines show the values of $\bar{\Gamma}_k^2$ as defined in Theorem 1, red lines show the values of $\bar{\Gamma}_{k,j}^2$ as defined in (5.61), and black dotted lines plot the bound in (5.65). For plots on the right, the red line shows the maximum loss of orthogonality, $\max_i |z_i^{(m)T} \hat{v}_{m+1}|$, the blue line shows $\min_i |\hat{\beta}_{m+1} \eta_{m,i}^{(m)}|$, the black solid line shows $\max_i |\hat{\beta}_{m+1} \eta_{m,i}^{(m)} z_i^{(m)T} \hat{v}_{m+1}|$, and the dashed black line plots $m\sigma\epsilon_2$.

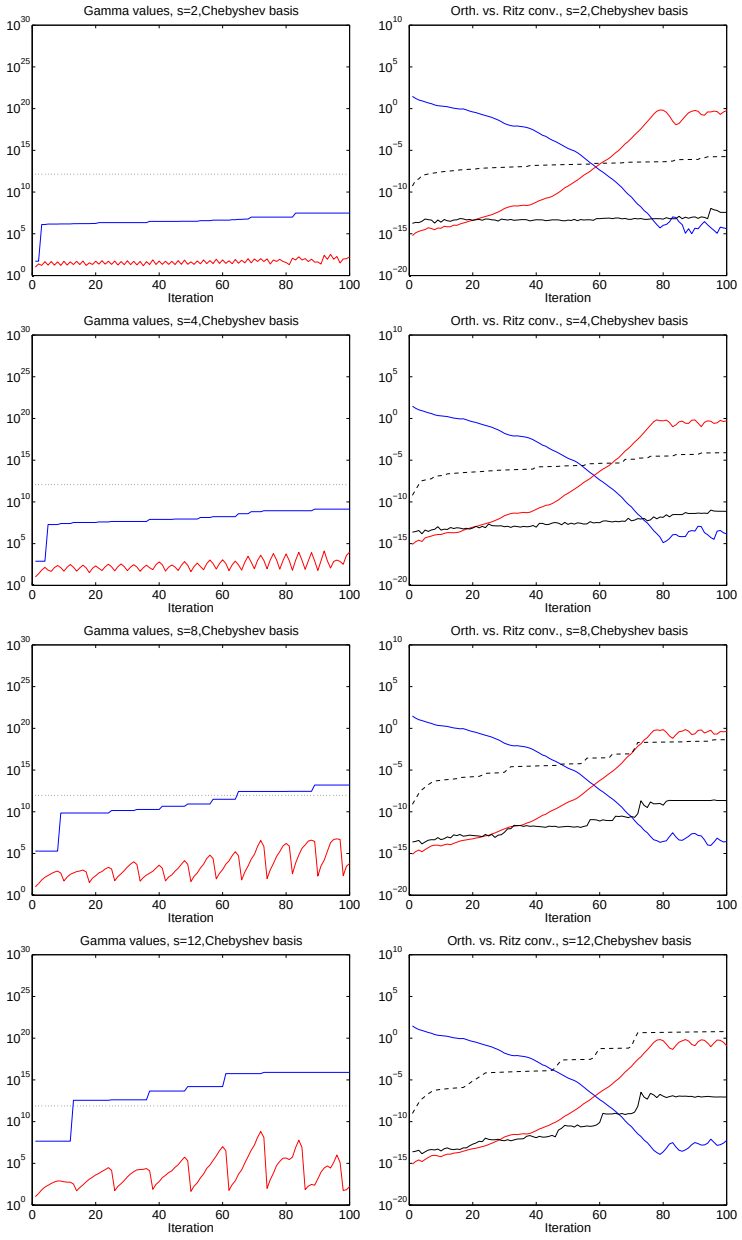

Figure 5.31: Loss of orthogonality and convergence of Ritz values for CA-Lanczos with the Chebyshev basis for the 'Strakoš' problem, for $s = 2$ (top row), $s = 4$ (second row from the top), $s = 8$ (second row from the bottom), and $s = 12$ (bottom row). For plots on the left, blue lines show the values of $\bar{\Gamma}_k^2$ as defined in Theorem 1, red lines show the values of $\bar{\Gamma}_{k,j}^2$ as defined in (5.61), and black dotted lines plot the bound in (5.65). For plots on the right, the red line shows the maximum loss of orthogonality, $\max_i |z_i^{(m)T} \hat{v}_{m+1}|$, the blue line shows $\min_i |\hat{\beta}_{m+1} \eta_{m,i}^{(m)}|$, the black solid line shows $\max_i |\hat{\beta}_{m+1} \eta_{m,i}^{(m)} z_i^{(m)T} \hat{v}_{m+1}|$, and the dashed black line plots $m\sigma\epsilon_2$.

basis conditioning such that (5.64) holds. We also discuss how our bounds could guide the use of extended or mixed precision in s-step Krylov methods; that is, rather than control the conditioning of the computed s-step base (5.64) could be satisfied by decreasing the unit roundoff ϵ using techniques either in hardware or software.

5.2 Convergence of CA-KSMs for Solving Linear Systems

In this section, we derive Lanczos-type matrix recurrences governing the CA-BICG and CA-CG methods in finite precision arithmetic, which demonstrate the relationship of these algorithms to their classical counterparts. We will use the term 'CA-(BI)CG' to indicate application to both CA-BICG and CA-CG. A derivation of the CA-BICG method appears in Algorithm 8 in Section 4.2; a derivation of the CA-CG method is deferred to Section 5.3, although CA-CG could also be obtained by taking $A = A^T$ and simplifying CA-BICG in Algorithm 8.

Using the recurrence, we extend the results of Tong and Ye for classical BICG [176] to derive an upper bound on the norm of the updated residual in finite precision CA-(BI)CG in terms of the residual norm of exact GMRES applied to a perturbed matrix, multiplied by an amplification factor. Our bound enables comparison of the finite precision CA-(BI)CG and classical BICG in terms of amplification factors. For CA-(BI)CG, the amplification factor depends heavily on the quality of s-step polynomial bases generated in each outer loop. As a side effect, by ignoring perturbation terms, we obtain an equivalent bound for the exact arithmetic case. Our bound also provides an analytical explanation for commonly-observed convergence behavior of CA-(BI)CG, namely, that convergence may occur despite (near) linear dependence among the Krylov vectors, which can occur as a result of the finite precision Lanczos process, as in the classical method, as well as from numerical rank deficiencies in the generated s-step polynomial bases. We note that this Section is adapted from the technical report [29].

5.2.1 The CA-(BI)CG Recurrence

We refer the reader to the derivation of CA-(BI)CG in Section 4.2 and to the CA-(BI)CG algorithm given in Algorithm 8. Recall that, within the inner loop of Algorithm 8, in step j of outer loop k, we update the length-$(2s+1)$ coefficients for the (BI)CG vectors as linear combinations of the columns in $\mathcal{Y}_k$ and $\tilde{\mathcal{Y}}_k$ rather than explicitly update the length-n (BI)CG vectors, as in classical (BI)CG. As before, the coefficient vectors are denoted with prime symbols (e.g., $r_{sk+j} = \mathcal{Y}_k r'_{k,j}$ for $j \in \{1, \ldots, s+1\}$). The inner iteration updates then become

$$r'_{k,j+1} = r'_{k,j} - \alpha_{sk+j} \mathcal{B}_k^{(\mathcal{Y})} p'_{k,j} \quad \text{and} \tag{5.129}$$

$$p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j} p'_{k,j}, \tag{5.130}$$

for $j \in \{1, \ldots, s\}$.

We can rearrange (5.129) and (5.130) as

$$\mathcal{B}_k^{(\mathcal{Y})} p'_{k,j} = \frac{1}{\alpha_{sk+j}} \left(r'_{k,j} - r'_{k,j+1} \right) \quad \text{for } j \in \{1, \ldots, s\}, \text{ and} \tag{5.131}$$

$$r'_{k,j} = p'_{k,j} - \beta_{sk+j-1} p'_{k,j-1} \quad \text{for } j \in \{2, \ldots, s+1\}. \tag{5.132}$$

Premultiplying (5.132) by $\mathcal{Y}_k$, and using $\mathcal{Y}_k[r'_{k,j}, p'_{k,j}] = \underline{\mathcal{Y}}_k[r'_{k,j}, p'_{k,j}]$ for $j \in \{1, \ldots, s\}$, we obtain

$$\underline{\mathcal{Y}}_k r'_{k,j} = \underline{\mathcal{Y}}_k p'_{k,j} - \beta_{sk+j-1} \underline{\mathcal{Y}}_k p'_{k,j-1} \tag{5.133}$$

for $j \in \{2, \ldots, s\}$. When $j = 1$, we have

$$\begin{aligned} \underline{\mathcal{Y}}_k r'_{k,1} &= \mathcal{Y}_{k-1} r'_{k-1,s+1} \\ &= \mathcal{Y}_{k-1} p'_{k-1,s+1} - \beta_{sk} \underline{\mathcal{Y}}_{k-1} p'_{k-1,s} \\ &= \underline{\mathcal{Y}}_k p'_{k,1} - \beta_{sk} \underline{\mathcal{Y}}_{k-1} p'_{k-1,s}, \end{aligned}$$

which gives a valid expression for the $j = 1$ case.

Now, we define $(2s+1)$-by-j matrices

$$R'_{k,j} = \left[r'_{k,1}, r'_{k,2}, \ldots, r'_{k,j} \right] \qquad \text{and} \quad P'_{k,j} = \left[p'_{k,1}, p'_{k,2}, \ldots, p'_{k,j} \right].$$

We can then write (5.133) in block form as

$$\underline{\mathcal{Y}}_k R'_{k,j} = \underline{\mathcal{Y}}_k P'_{k,j} U_{k,j} - \beta_{sk} \underline{\mathcal{Y}}_{k-1} p'_{k-1,s} e_1^T, \tag{5.134}$$

where

$$U_{k,j} = \begin{bmatrix} 1 & -\beta_{sk+1} & & \\ & 1 & \ddots & \\ & & \ddots & -\beta_{sk+j-1} \\ & & & 1 \end{bmatrix}.$$

Premultiplying (5.134) by A, we obtain

$$A\underline{\mathcal{Y}}_k R'_{k,j} = A\underline{\mathcal{Y}}_k P'_{k,j} U_{k,j} - \beta_{sk} A\underline{\mathcal{Y}}_{k-1} p'_{k-1,s} e_1^T. \tag{5.135}$$

We can also write (5.131) in block form as

$$\mathcal{B}_k^{(\mathcal{Y})} P'_{k,j} = R'_{k,j} L_{k,j} \Lambda_{k,j}^{-1} - \frac{1}{\alpha_{sk+j}} r'_{k,j+1} e_j^T, \tag{5.136}$$

where $\Lambda_{k,j} = \operatorname{diag}(\alpha_{sk+1}, \ldots, \alpha_{sk+j})$ and

$$L_{k,j} = \begin{bmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}.$$

If we premultiply (5.136) by $\mathcal{Y}_k$ and postmultiply by $U_{k,j}$, we obtain

$$\mathcal{Y}_k \mathcal{B}_k^{(\mathcal{Y})} P'_{k,j} U_{k,j} = \mathcal{Y}_k R'_{k,j} L_{k,j} \Lambda_{k,j}^{-1} U_{k,j} - \frac{1}{\alpha_{sk+j}} \mathcal{Y}_k r'_{k,j+1} e_j^T,$$

which can be written

$$A \underline{\mathcal{Y}}_k P'_{k,j} U_{k,j} = \underline{\mathcal{Y}}_k R'_{k,j} L_{k,j} \Lambda_{k,j}^{-1} U_{k,j} - \frac{1}{\alpha_{sk+j}} \mathcal{Y}_k r'_{k,j+1} e_j^T \tag{5.137}$$

since $A\underline{\mathcal{Y}}_k = \mathcal{Y}_k \mathcal{B}_k^{(\mathcal{Y})}$ and $\mathcal{Y}_k R'_{k,j} = \underline{\mathcal{Y}}_k R'_{k,j}$ for $j < s+1$. We can then combine (5.135) and (5.137) to obtain

$$A \underline{\mathcal{Y}}_k R'_{k,j} = \underline{\mathcal{Y}}_k R'_{k,j} \mathcal{T}_{k,j} - \frac{\beta_{sk}}{\alpha_{sk}} \underline{\mathcal{Y}}_{k-1} r'_{k-1,s} e_1^T - \frac{1}{\alpha_{sk+j}} \mathcal{Y}_k r'_{k,j+1} e_j^T, \tag{5.138}$$

for $j \in \{1, \ldots, s\}$, where $\mathcal{T}_{k,j} = L_{k,j} \Lambda_{k,j}^{-1} U_{k,j} + e_1 \frac{\beta_{sk}}{\alpha_{sk}} e_1^T$. Note when $k = 0$, $\frac{\beta_{sk}}{\alpha_{sk}}$ is defined to be 0.

We can now combine outer loop iterations in block form to write the CA-(BI)CG recurrence for iterations 1 through $sk + j$. Let $\underline{\mathfrak{Y}}_k = [\underline{\mathcal{Y}}_0, \ldots, \underline{\mathcal{Y}}_k]$. Let

$$\mathfrak{R}'_{k,j} = \begin{bmatrix} R'_{0,s} & & & \\ & R'_{1,s} & & \\ & & \ddots & \\ & & & R'_{k,j} \end{bmatrix}$$

and

$$T_{sk+j} = \begin{bmatrix} \frac{1}{\alpha_1} & -\frac{\beta_1}{\alpha_1} & & \\ -\frac{1}{\alpha_1} & \frac{1}{\alpha_2} + \frac{\beta_1}{\alpha_1} & \ddots & \\ & \ddots & \ddots & \frac{\beta_{sk+j-1}}{\alpha_{sk+j-1}} \\ & & -\frac{1}{\alpha_{sk+j-1}} & \frac{1}{\alpha_{sk+j}} + \frac{\beta_{sk+j-1}}{\alpha_{sk+j-1}} \end{bmatrix}.$$

Then by (5.138), we can write

$$A \underline{\mathfrak{Y}}_k \mathfrak{R}'_{k,j} = \underline{\mathfrak{Y}}_k \mathfrak{R}'_{k,j} T_{sk+j} - \frac{1}{\alpha_{sk+j}} \mathcal{Y}_k r'_{k,j+1} e_{sk+j}^T.$$

Since we can write the (BI)CG residual vectors as $R_{sk+j} = [r_1, \ldots, r_{sk+j}] = \underline{\mathfrak{Y}}_k \mathfrak{R}'_{k,j}$, and $r_{sk+j+1} = \mathcal{Y}_k r'_{k,j+1}$, we can write the above as

$$A R_{sk+j} = R_{sk+j} T_{sk+j} - \frac{1}{\alpha_{sk+j}} r_{sk+j+1} e_{sk+j}^T,$$

which gives us the same governing equation for iterations 1 through $sk + j$ as the classical (BI)CG algorithm in exact arithmetic [176]. Note that a similar relation holds for the dual Krylov vectors $\tilde{r}_{sk+j}$ and $\tilde{p}_{sk+j}$.

5.2.2 CA-(BI)CG in Finite Precision

The goal of this section is to derive a Lanczos-type recurrence for finite precision CA-(BI)CG of the form

$$A\underset{\sim}{\mathcal{Y}}_k\mathfrak{R}'_{k,j} = \underset{\sim}{\mathcal{Y}}_k\mathfrak{R}'_{k,j}T_{sk+j} - \frac{1}{\alpha_{sk+j}}\mathcal{Y}_k r'_{k,j+1}e_{sk+j}^T + \epsilon\mathfrak{d}_{k,j}$$

and upper bound the size of the error term $\epsilon\mathfrak{d}_{k,j}$. Recall that we assume a standard model of floating point arithmetic, where

$$\begin{aligned} \mathrm{fl}\left(\alpha x + y\right) &= \alpha x + y + \delta_1, && \text{where} \quad |\delta_1| \leq \epsilon 2\left|\alpha x\right| + |y| + O(\epsilon^2), \quad \text{and} \\ \mathrm{fl}\left(Ax\right) &= Ax + \delta_2, && \text{where} \quad |\delta_2| \leq \epsilon N\left|A\right|\left|x\right| + O(\epsilon^2), \end{aligned}$$

where ϵ is the machine precision unit, $x, y \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$, and N is the maximum number of nonzeros per row in A. In the remaining analysis we drop higher powers of ϵ for simplicity. For simplicity of notation, in this section we let $r'_{k,j}$, $p'_{k,j}$, $x'_{k,j}$, α_{sk+j}, r_{sk+s+1}, p_{sk+s+1}, β_{sk+j}, $\mathcal{Y}_k$, and $\mathcal{B}_k^{(\mathcal{Y})}$ be the computed quantities in finite precision CA-(BI)CG.

At the $(sk+j)$-th iteration, to compute $r'_{k,j+1}$ we first compute $\mathcal{B}_k^{(\mathcal{Y})}p'_{k,j}$ and have

$$\mathrm{fl}\left(\mathcal{B}_k^{(\mathcal{Y})}p'_{k,j}\right) = \mathcal{B}_k^{(\mathcal{Y})}p'_{k,j} + g, \quad \text{where} \quad |g| \leq \epsilon(2s+1)\left|\mathcal{B}_k^{(\mathcal{Y})}\right|\left|p'_{k,j}\right|.$$

Then

$$\begin{aligned} r'_{k,j+1} &= \mathrm{fl}\left(r'_{k,j} - \alpha_{sk+j}\cdot\mathrm{fl}\left(\mathcal{B}_k^{(\mathcal{Y})}p'_{k,j}\right)\right) \\ &= r'_{k,j} - \alpha_{sk+j}\mathcal{B}_k^{(\mathcal{Y})}p'_{k,j} - \alpha_{sk+j}g + g', \end{aligned} \tag{5.139}$$

where $\left|g'\right| \leq \epsilon\big(\left|r'_{k,j}\right| + 2\left|\alpha_{sk+j}\right|\left|\mathcal{B}_k^{(\mathcal{Y})}p'_{k,j}\right|\big)$. Let $\delta_{r'_{k,j}} = \left(\alpha_{sk+j}g + g'\right)/\left(\epsilon\left|\alpha_{sk+j}\right|\right)$. Then using (5.139) we obtain

$$\frac{1}{\alpha_{sk+j}}\left(r'_{k,j+1} - r'_{k,j}\right) = -\mathcal{B}_k^{(\mathcal{Y})}p'_{k,j} + \epsilon\delta_{r'_{k,j}},$$

where

$$\left|\delta_{r'_{k,j}}\right| \leq (2s+1)\left|\mathcal{B}_k^{(\mathcal{Y})}\right|\left|p'_{k,j}\right| + \frac{\left|r'_{k,j}\right|}{\left|\alpha_{sk+j}\right|} + 2\left|\mathcal{B}_k^{(\mathcal{Y})}p'_{k,j}\right|. \tag{5.140}$$

Similarly,

$$\begin{aligned} p'_{k,j+1} &= \mathrm{fl}\left(r'_{k,j+1} + \beta_{sk+j}p'_{k,j}\right) \\ &= r'_{k,j+1} + \beta_{sk+j}p'_{k,j} + f, \end{aligned}$$

where $|f| \leq \epsilon\left(\left|r'_{k,j+1}\right| + 2\left|\beta_{sk+j}\right|\left|p'_{k,j}\right|\right)$. Letting $\delta_{p'_{k,j+1}} = f/\epsilon$, we have

$$p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j}p'_{k,j} + \epsilon\delta_{p'_{k,j+1}},$$

where

$$\left|\delta_{p'_{k,j+1}}\right| \leq \left|r'_{k,j+1}\right| + 2\left|\beta_{sk+j}\right|\left|p'_{k,j}\right|. \tag{5.141}$$

Rearranging (5.140) and (5.141), we can write

$$\mathcal{B}_k^{(\mathcal{Y})} p'_{k,j} = \frac{1}{\alpha_{sk+j}}\left(r'_{k,j} - r'_{k,j+1}\right) + \epsilon\delta_{r'_{k,j}} \quad \text{for } j \in \{1,\ldots,s\}, \text{ and} \tag{5.142}$$

$$r'_{k,j} = p'_{k,j} - \beta_{sk+j-1}p'_{k,j-1} + \epsilon\delta_{p'_{k,j}} \quad \text{for } j \in \{2,\ldots,s+1\}, \tag{5.143}$$

and premultiplying (5.143) by $\underline{\mathcal{Y}}_k$ gives

$$\underline{\mathcal{Y}}_k r'_{k,j} = \underline{\mathcal{Y}}_k p'_{k,j} - \beta_{sk+j-1}\underline{\mathcal{Y}}_k p'_{k,j-1} + \epsilon\underline{\mathcal{Y}}_k\delta_{p'_{k,j}},$$

for $j \in \{2,\ldots,s\}$. When $j = 1$, we have

$$\begin{aligned}
\underline{\mathcal{Y}}_k r'_{k,1} &= fl(\mathcal{Y}_{k-1}r'_{k-1,s+1}) \\
&= \mathcal{Y}_{k-1}r'_{k-1,s+1} + \epsilon\phi^r_{k-1} \quad \text{and} \\
\underline{\mathcal{Y}}_k p'_{k,1} &= fl(\mathcal{Y}_{k-1}p'_{k-1,s+1}) \\
&= \mathcal{Y}_{k-1}p'_{k-1,s+1} + \epsilon\phi^p_{k-1},
\end{aligned} \tag{5.144}$$

where $\left|\phi^r_{k-1}\right| \leq (2s+1)\left|\mathcal{Y}_{k-1}\right|\left|r'_{k-1,s+1}\right|$ and $\left|\phi^p_{k-1}\right| \leq (2s+1)\left|\mathcal{Y}_{k-1}\right|\left|p'_{k-1,s+1}\right|$. Then for $j = 1$, we can write

$$\begin{aligned}
\underline{\mathcal{Y}}_k r'_{k,1} &= \mathcal{Y}_{k-1}r'_{k-1,s+1} + \epsilon\phi^r_{k-1} \\
&= \mathcal{Y}_{k-1}p'_{k-1,s+1} - \beta_{sk}\mathcal{Y}_{k-1}p'_{k-1,s} + \epsilon\mathcal{Y}_{k-1}\delta_{p'_{k-1,s+1}} + \epsilon\phi^r_{k-1} \\
&= \underline{\mathcal{Y}}_k p'_{k,1} - \epsilon\phi^p_{k-1} - \beta_{sk}\underline{\mathcal{Y}}_{k-1}p'_{k-1,s} + \epsilon\mathcal{Y}_{k-1}\delta_{p'_{k-1,s+1}} + \epsilon\phi^r_{k-1} \\
&= \underline{\mathcal{Y}}_k p'_{k,1} - \beta_{sk}\underline{\mathcal{Y}}_{k-1}p'_{k-1,s} + \epsilon\left(\mathcal{Y}_{k-1}\delta_{p'_{k-1,s+1}} + \phi^r_{k-1} - \phi^p_{k-1}\right).
\end{aligned}$$

Now, let $\Delta_{R'_{k,j}} = \left[\delta_{r'_{k,1}},\ldots,\delta_{r'_{k,j}}\right]$ and $\Delta_{P'_{k,j}} = \left[0_{2s+1},\delta_{p'_{k,2}},\ldots,\delta_{p'_{k,j}}\right]$. We can then write

$$\begin{aligned}
\underline{\mathcal{Y}}_k R'_{k,j} =& \underline{\mathcal{Y}}_k P'_{k,j}U_{k,j} - \beta_{sk}\underline{\mathcal{Y}}_{k-1}p'_{k-1,s}e_1^T + \epsilon\underline{\mathcal{Y}}_k\Delta_{P'_{k,j}} \\
&+ \epsilon\left(\mathcal{Y}_{k-1}\delta_{p'_{k-1,s+1}} + \phi^r_{k-1} - \phi^p_{k-1}\right)e_1^T \quad \text{and}
\end{aligned} \tag{5.145}$$

$$\underline{B}_k^{(\mathcal{Y})} P'_{k,j} = R'_{k,j}L_{k,j}\Lambda^{-1}_{k,j} - \frac{1}{\alpha_{sk+j}}r'_{k,j+1}e_j^T + \epsilon\Delta_{R'_{k,j}}. \tag{5.146}$$

Premultiplying (5.145) by A gives

$$\begin{aligned}
A\underline{\mathcal{Y}}_k R'_{k,j} =& A\underline{\mathcal{Y}}_k P'_{k,j}U_{k,j} - \beta_{sk}A\underline{\mathcal{Y}}_{k-1}p'_{k-1,s}e_1^T + \epsilon A\underline{\mathcal{Y}}_k\Delta_{P'_{k,j}} \\
&+ \epsilon A\left(\mathcal{Y}_{k-1}\delta_{p'_{k-1,s+1}} + \phi^r_{k-1} - \phi^p_{k-1}\right)e_1^T,
\end{aligned} \tag{5.147}$$

and premultiplying (5.146) by $\mathcal{Y}_k$ gives

$$\mathcal{Y}_k\mathcal{B}_k^{(\mathcal{Y})}P'_{k,j} = \mathcal{Y}_k R'_{k,j}L_{k,j}\Lambda_{k,j}^{-1} - \frac{1}{\alpha_{sk+j}}\mathcal{Y}_k r'_{k,j+1}e_j^T + \epsilon\mathcal{Y}_k\Delta_{R'_{k,j}} \tag{5.148}$$

for $j \in \{1,\ldots,s\}$.

Now, to write the error in CA-(BI)CG in the context of classical (BI)CG, we must account for error in computation of the s-step bases. We will write the finite precision basis computation as

$$A\underline{\mathcal{Y}}_k = \mathcal{Y}_k\mathcal{B}_k^{(\mathcal{Y})} + \epsilon\Delta_{\mathcal{Y}_k}, \tag{5.149}$$

where $|\Delta_{\mathcal{Y}_k}| \leq (3+N)|A||\underline{\mathcal{Y}}_k| + 4|\mathcal{Y}_k||\mathcal{B}_k^{(\mathcal{Y})}|$ (see (5.42)).

Using (5.149), we can write (5.148) as

$$(A\underline{\mathcal{Y}}_k - \epsilon\Delta_{\mathcal{Y}_k})P'_{k,j} = \underline{\mathcal{Y}}_k R'_{k,j}L_{k,j}\Lambda_{k,j}^{-1} - \frac{1}{\alpha_{sk+j}}\mathcal{Y}_k r'_{k,j+1}e_j^T + \epsilon\mathcal{Y}_k\Delta_{R'_{k,j}},$$

which can be rearranged to obtain

$$A\underline{\mathcal{Y}}_k P'_{k,j} = \underline{\mathcal{Y}}_k R'_{k,j}L_{k,j}\Lambda_{k,j}^{-1} - \frac{1}{\alpha_{sk+j}}\mathcal{Y}_k r'_{k,j+1}e_j^T + \epsilon\mathcal{Y}_k\Delta_{R'_{k,j}} + \epsilon\Delta_{\mathcal{Y}_k}P'_{k,j}. \tag{5.150}$$

Postmultiplying (5.150) by $U_{k,j}$ gives

$$\begin{aligned} A\underline{\mathcal{Y}}_k P'_{k,j}U_{k,j} =& \underline{\mathcal{Y}}_k R'_{k,j}L_{k,j}\Lambda_{k,j}^{-1}U_{k,j} - \frac{1}{\alpha_{sk+j}}\mathcal{Y}_k r'_{k,j+1}e_j^T \\ &+ \epsilon\big(\mathcal{Y}_k\Delta_{R'_{k,j}}U_{k,j} + \Delta_{\mathcal{Y}_k}P'_{k,j}U_{k,j}\big), \end{aligned} \tag{5.151}$$

and combining (5.151) and (5.147), we obtain

$$A\underline{\mathcal{Y}}_k R'_{k,j} = \underline{\mathcal{Y}}_k R'_{k,j}L_{k,j}\Lambda_{k,j}^{-1}U_{k,j} - \frac{1}{\alpha_{sk+j}}\mathcal{Y}_k r'_{k,j+1}e_j^T - \beta_{sk}A\underline{\mathcal{Y}}_{k-1}p'_{k-1,s}e_1^T \tag{5.152}$$

$$+ \epsilon\left(A\underline{\mathcal{Y}}_k\Delta_{P'_{k,j}} + \mathcal{Y}_k\Delta_{R'_{k,j}}U_{k,j} + \Delta_{\mathcal{Y}_k}P'_{k,j}U_{k,j}\right) \tag{5.153}$$

$$+ \epsilon A\left(\mathcal{Y}_{k-1}\delta_{p'_{k-1,s+1}} + \phi_{k-1}^r - \phi_{k-1}^p\right)e_1^T. \tag{5.154}$$

Since, using (5.149), (5.142), and (5.144), we have

$$\begin{aligned} \beta_{sk}A\underline{\mathcal{Y}}_{k-1}p'_{k-1,s}e_1^T =& \frac{\beta_{sk}}{\alpha_{sk}}\underline{\mathcal{Y}}_{k-1}r'_{k-1,s}e_1^T - \frac{\beta_{sk}}{\alpha_{sk}}(\underline{\mathcal{Y}}_k r'_{k,1} - \epsilon\phi_{k-1}^r)e_1^T \\ &+ \epsilon\beta_{sk}\mathcal{Y}_{k-1}\delta_{r'_{k-1,s}}e_1^T + \beta_{sk}\Delta_{\mathcal{Y}_{k-1}}p'_{k-1,s}e_1^T, \end{aligned}$$

we can write (5.154) as

$$A\underline{\mathcal{Y}}_k R'_{k,j} = \mathcal{Y}_k R'_{k,j}\mathcal{T}_{k,j} - \frac{\beta_{sk}}{\alpha_{sk}}\underline{\mathcal{Y}}_{k-1}r'_{k-1,s}e_1^T - \frac{1}{\alpha_{sk+j}}\mathcal{Y}_k r'_{k,j+1}e_j^T + \epsilon\Delta_{k,j},$$

where

$$\Delta_{k,j} = \left(A\underline{\mathcal{Y}}_k\Delta_{P'_{k,j}} + A\mathcal{Y}_{k-1}\delta_{p'_{k-1,s+1}}e_1^T\right) + \left(\mathcal{Y}_k\Delta_{R'_{k,j}}U_{k,j} - \beta_{sk}\mathcal{Y}_{k-1}\delta_{r'_{k-1,s}}e_1^T\right) \tag{5.155}$$

$$+ \left(\Delta_{\mathcal{Y}_k}P'_{k,j}U_{k,j} - \beta_{sk}\Delta_{\mathcal{Y}_{k-1}}p'_{k-1,s}e_1^T\right) + \left(A(\phi^r_{k-1} - \phi^p_{k-1}) - \frac{\beta_{sk}}{\alpha_{sk}}\phi^r_{k-1}\right)e_1^T. \tag{5.156}$$

Writing $\Delta_{k,j} = [\delta_{sk+1}, \ldots, \delta_{sk+j}]$, we have that the i^{th} column of $\Delta_{k,j}$, for $i \in \{1, \ldots, sk+j\}$, is

$$\delta_{sk+i} = A\underline{\mathcal{Y}}_k\delta_{p'_{k,i}} + \mathcal{Y}_k\delta_{r'_{k,i}} - \beta_{sk+i-1}\mathcal{Y}_k\delta_{r'_{k,i-1}} + \Delta_{\mathcal{Y}_k}r'_{k,i}, \tag{5.157}$$

when $i > 1$, and

$$\begin{aligned}\delta_{sk+1} =& A\mathcal{Y}_{k-1}\delta_{p'_{k-1,s+1}} + \mathcal{Y}_k\delta_{r'_{k,1}} - \beta_{sk}\mathcal{Y}_{k-1}\delta_{r'_{k-1,s}} + \Delta_{\mathcal{Y}_k}p'_{k,1} - \beta_{sk}\Delta_{\mathcal{Y}_{k-1}}p'_{k-1,s} \\ &+ A(\phi^r_{k-1} - \phi^p_{k-1}) - \frac{\beta_{sk}}{\alpha_{sk}}\phi^r_{k-1}\end{aligned} \tag{5.158}$$

for $i = 1$.

Using the inequalities $\left|\beta_{sk+j-1}p'_{k,j-1}\right| \leq \left|p'_{k,j}\right| + \left|r'_{k,j}\right| + O(\epsilon)$ and $\left|r'_{k,j-1}\right| \leq \left|r'_{k,j}\right| + \left|\alpha_{sk+j-1}\right|\left|\mathcal{B}^{(\mathcal{Y})}_k p'_{k,j-1}\right| + O(\epsilon)$, we can bound the norm of the columns as

$$\begin{aligned}\left|\delta_{sk+i}\right| \leq& \left((N+6)\left|A\right|\left|\mathcal{Y}_k\right| + (2s+8)\left|\mathcal{Y}_k\right|\left|\mathcal{B}^{(\mathcal{Y})}_k\right| + \left(\frac{1}{\left|\alpha_{sk+i}\right|} + \frac{\left|\beta_{sk+i-1}\right|}{\left|\alpha_{sk+i-1}\right|}\right)\left|\mathcal{Y}_k\right|\right)\left|r'_{k,j}\right| \\ &+ \left(2\left|A\right|\left|\mathcal{Y}_k\right| + (4s+7)\left|\mathcal{Y}_k\right|\left|\mathcal{B}^{(\mathcal{Y})}_k\right|\right)\left|p'_{k,j}\right|,\end{aligned} \tag{5.159}$$

if $i > 1$. For the $i = 1$ case, we have

$$\left|\delta_{sk+1}\right| \leq \left((N+2s+7)\left|A\right|\left|\mathcal{Y}_{k-1}\right| + (2s+8)\left|\mathcal{Y}_{k-1}\right|\left|\mathcal{B}^{(\mathcal{Y})}_{k-1}\right|\right)\left|r'_{k-1,s+1}\right| \tag{5.160}$$

$$+ \left(\frac{1}{\left|\alpha_{sk+1}\right|} + \frac{(2s+2)\left|\beta_{sk}\right|}{\left|\alpha_{sk}\right|}\right)\left|\mathcal{Y}_{k-1}\right|\left|r'_{k-1,s+1}\right|$$

$$+ \left((2N+4s+16)\left|A\right|\left|\mathcal{Y}_{k-1}\right| + (6s+22)\left|\mathcal{Y}_{k-1}\right|\left|\mathcal{B}^{(\mathcal{Y})}_{k-1}\right|\right)\left|p'_{k-1,s+1}\right| \tag{5.161}$$

We can thus write the finite precision CA-(BI)CG recurrence for iterations 1 through $sk+j$ as

$$A\mathfrak{Y}_k\mathfrak{R}'_{k,j} = \mathfrak{Y}_k\mathfrak{R}'_{k,j}T_{sk+j} - \frac{1}{\alpha_{sk+j}}\mathcal{Y}_k r'_{k,j+1}e^T_{sk+j} + \epsilon\mathfrak{d}_{k,j}, \tag{5.162}$$

where $\mathfrak{d}_{k,j} = [\Delta_{0,s-1}, \Delta_{1,s-1}, \ldots, \Delta_{k,j}]$.

5.2.2.1 Relation to Classical (BI)CG

Note that from (5.155), we can see that the two terms on the right-hand side correspond to the two terms in Tong and Ye's analysis for classical BICG, and the remaining two terms correspond to the error in computing the s-step Krylov bases and the change of basis operation. We can also see that a bound on the size of the error in each column of the finite precision recurrence depends on both the magnitude of the error in computing the s-step Krylov bases, i.e., $\|\Delta_{\mathcal{Y}_k}\|$, as well as the size and conditioning of the bases $\mathcal{Y}_k$. We elaborate on this below; we will see in Section 6.1 that this has implications for performing residual replacement.

It is also possible to follow the technique of Section 5.1 and write the bounds in (5.159) and (5.161) in the form of the equivalent bounds for classical (BI)CG, multiplied by an amplification term that depends on the conditioning of the s-step bases. As in Theorem 1, we let $\sigma \equiv \|A\|_2$, $\theta\sigma = \||A|\|_2$ and $\tau_k\sigma = \||\mathcal{B}_k^{(\mathcal{Y})}|\|_2$, and define $\Gamma_k = \|\hat{\mathcal{Y}}_k^+\|_2\|\hat{\mathcal{Y}}_k|\|_2$, where the superscript '+' denotes the Moore-Penrose pseudoinverse, i.e., $\hat{\mathcal{Y}}_k^+ = (\hat{\mathcal{Y}}_k^T\hat{\mathcal{Y}}_k)^{-1}\hat{\mathcal{Y}}_k^T$.

Using $r_{sk+j} = \mathcal{Y}_k r'_{k,j} + O(\epsilon)$ and similarly for p_{sk+j}, and again ignoring higher than first order terms in ϵ, we can use the defined quantities to bound the norm of (5.159) and (5.161) as

$$\begin{aligned}\|\delta_{sk+i}\| \leq \Gamma_k \left(\big((N+6)\theta + (2s+8)\tau_k\big)\|A\| + \frac{1}{|\alpha_{sk+i}|} + \frac{|\beta_{sk+i-1}|}{|\alpha_{sk+i-1}|} \right) \|r_{sk+i}\| \\ + \Gamma_k \Big(\big(2\theta + (4s+7)\tau_k\big)\|A\| \Big) \|p_{sk+i}\| \end{aligned} \tag{5.163}$$

if $i \in \{2, \ldots, s\}$ and

$$\begin{aligned}\|\delta_{sk+1}\| \leq \Gamma_{k-1} \left(\big((N+2s+7)\theta + (2s+8)\tau_{k-1}\big)\|A\| + \frac{1}{|\alpha_{sk+1}|} + (2s+2)\frac{|\beta_{sk}|}{|\alpha_{sk}|} \right) \|r_{sk+1}\| \\ + \Gamma_{k-1} \Big(\big((2N+4s+16)\theta + (6s+22)\tau_{k-1}\big)\|A\| \Big) \|p_{sk+1}\| \end{aligned} \tag{5.164}$$

when $i = 1$. For the classical (BI)CG method, the equivalent bound of Tong and Ye [176] is

$$\|\delta_i\| \leq \left((N+6)\theta\|A\| + \frac{1}{|\alpha_i|} + \frac{|\beta_{i-1}|}{|\alpha_{i-1}|} \right) \|r_i\| + (2N+7)\theta\|A\|\|p_i\| \tag{5.165}$$

for all $i \in \{1, \ldots, sk+j\}$.

Comparing (5.163) and (5.164) to (5.165), we can see that if all Γ_k's are not too large, then the sizes of columns of the error term in the finite precision CA-(BI)CG recurrence will be close to those of the classical (BI)CG recurrence (recall from Section 5.1.3.1 that we expect $\||\mathcal{B}_k^{(\mathcal{Y})}|\|_2 \lesssim \| \, |A| \, \|_2$). If however, at least one Γ_k is large, this will result in a large perturbation to the Lanczos recurrence and we can expect the convergence behavior of CA-(BI)CG and (BI)CG to deviate. Later in this section, we will elaborate on the importance of the basis conditioning and see the roll it plays in convergence behavior.

5.2.2.2 Diagonal Scaling

As in [176], it will be more convenient to work with a scaled version of (5.162) in subsequent sections. Let $Z_{sk+j} = [z_1, \ldots, z_{sk+j}] = \mathcal{Y}_k \mathcal{R}'_{k,j} D^{-1}_{sk+j}$ where

$$D_{sk+j} = \text{diag}\left(\left\|\mathcal{Y}_0 r'_{0,1}\right\|, \ldots, \left\|\mathcal{Y}_0 r'_{0,s}\right\|, \left\|\mathcal{Y}_1 r'_{1,1}\right\|, \ldots, \left\|\mathcal{Y}_k r'_{k,j}\right\|\right).$$

We can then write the scaled version of (5.162) as

$$AZ_{sk+j} = Z_{sk+j}\bar{T}_{sk+j} - \frac{1}{\bar{\alpha}_{sk+j}} \frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} e^T_{sk+j} + \epsilon \bar{\mathfrak{d}}_{k,j}, \tag{5.166}$$

where $\bar{T}_{sk+j} = D_{sk+j} T_{sk+j} D^{-1}_{sk+j}$,

$$\bar{\alpha}_{sk+j} = \alpha_{sk+j} \frac{\left\|\mathcal{Y}_k r'_{k,j}\right\|}{\|r_1\|} = \alpha_{sk+j} \frac{\left\|\mathcal{Y}_k r'_{k,j}\right\|}{\left\|\mathcal{Y}_0 r'_{0,1}\right\|} = e^T_{sk+j} \bar{T}^{-1}_{sk+j} e_1,$$

and

$$\bar{\mathfrak{d}}_{k,j} = \mathfrak{d}_{k,j} D^{-1}_{sk+j}.$$

5.2.3 Bounds on $\|r_{sk+j+1}\|$ for Finite Precision CA-(BI)CG

In this subsection, we upper bound the norm of the updated residual computed in iteration $sk+j$ of CA-(BI)CG. First, we will review a series of Lemmas proved by Tong and Ye [176]. The proofs shown below are nearly identical to those given by Tong and Ye [176], although we have changed the notation and indexing for consistency with our s-step formulation[4].

Lemma 3. *Assume*

$$AZ_{sk+j} = Z_{sk+j}\bar{T}_{sk+j} - \frac{1}{\bar{\alpha}_{sk+j}} \frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} e^T_{sk+j}$$

with $r_1 = \|r_1\| z_1$. *Then for any polynomial* $\rho(x) = \sum_{i=0}^{sk+j} \psi_i x^i$ *of degree* $\leq sk+j$,

$$\rho(A) z_1 = Z_{sk+j} \rho(\bar{T}_{sk+j}) e_1 + c_{sk+j} \mathcal{Y}_k r'_{k,j+1},$$

where $c_{sk+j} = (-1)^{sk+j} \psi_{sk+j} / (\alpha_1 \cdots \alpha_{sk+j} \|r_1\|)$.

[4] One lemma presented is slightly different than what appears in [176] due to a minor mathematical error that we correct.

Proof. First, we will prove by induction that for $1 \leq i \leq sk+j-1$

$$A^i Z_{sk+j} e_1 = Z_{sk+j} \bar{T}_{sk+j}^i e_1. \tag{5.167}$$

For $i = 1$, we have

$$AZ_{sk+j} e_1 = \left(Z_{sk+j} \bar{T}_{sk+j} - \frac{1}{\bar{\alpha}_{sk+j}} \frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} e_{sk+j}^T \right) e_1 = Z_{sk+j} \bar{T}_{sk+j} e_1.$$

Now, assume (5.167) holds for some $i \leq sk+j-2$. Then

$$\begin{aligned} A^{i+1} Z_{sk+j} e_1 &= A(A^i Z_{sk+j} e_1) \\ &= A(Z_{sk+j} \bar{T}_{sk+j} e_1) \\ &= \left(Z_{sk+j} \bar{T}_{sk+j} - \frac{1}{\bar{\alpha}_{sk+j}} \frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} e_{sk+j}^T \right) \bar{T}_{sk+j}^i e_1 \\ &= Z_{sk+j} \bar{T}_{sk+j} \bar{T}_{sk+j}^i e_1 = Z_{sk+j} \bar{T}_{sk+j}^{i+1} e_1, \end{aligned}$$

where we have used the fact that $e_{sk+j}^T \bar{T}_{sk+j}^i e_1 = 0$ when $i \leq sk+j-2$. Therefore the inductive hypothesis holds. Now consider the case $i = sk+j-1$. We then have

$$\begin{aligned} A^{sk+j} Z_{sk+j} e_1 &= A(Z_{sk+j} \bar{T}_{sk+j}^{sk+j-1} e_1) \\ &= \left(Z_{sk+j} \bar{T}_{sk+j} - \frac{1}{\bar{\alpha}_{sk+j}} \frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} e_{sk+j}^T \right) \left(\bar{T}_{sk+j}^{sk+j-1} e_1 \right). \end{aligned}$$

Since it can be shown that $e_{sk+j}^T \bar{T}_{sk+j}^{sk+j-1} e_1 = (-1)^{sk+j-1} \left\| \mathcal{Y}_k r'_{k,j} \right\| \left(\alpha_1 \cdots \alpha_{sk+j-1} \|r_1\| \right)^{-1}$ and $\bar{\alpha}_{sk+j} = \left\| \mathcal{Y}_k r'_{k,j} \right\| \alpha_{sk+j} / \|r_1\|$, we have

$$\begin{aligned} A^{sk+j} Z_{sk+j} e_1 &= Z_{sk+j} \bar{T}_{sk+j} \bar{T}_{sk+j}^{sk+j-1} e_1 \\ &= Z_{sk+j} \bar{T}_{sk+j}^{sk+j} e_1 + \frac{(-1)^{sk+j}}{\alpha_1 \cdots \alpha_{sk+j} \|r_1\|} \mathcal{Y}_k r'_{k,j+1}. \end{aligned}$$

The lemma follows. □

We now use this result in proving the following identity.

Lemma 4. *Assume*

$$AZ_{sk+j} = Z_{sk+j} \bar{T}_{sk+j} - \frac{1}{\bar{\alpha}_{sk+j}} \frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} e_{sk+j}^T$$

with $r_1 = \|r_1\| z_1$ *and* $\bar{\alpha}_{sk+j} = e_{sk+j}^T \bar{T}_{sk+j}^{-1} e_1$. *Assume that* $W^T \in \mathbb{R}^{(sk+j) \times n}$ *is a matrix such that* $W^T Z_{sk+j} = I$ *and* $W^T \mathcal{Y}_k r'_{k,j+1} = 0_{sk+j,1}$. *Then for any polynomial* $\rho(x)$ *of degree not exceeding* $sk+j$ *with* $\rho(0) = 1$, *we have*

$$\mathcal{Y}_k r'_{k,j+1} = \left(I - AZ_{sk+j} \bar{T}_{sk+j}^{-1} W^T \right) \rho(A) r_1.$$

Proof. First, we multiply by $\bar{T}_{sk+j}^{-1}e_1$ to get

$$AZ_{sk+j}\bar{T}_{sk+j}^{-1}e_1 = \left(Z_{sk+j}\bar{T}_{sk+j} - \frac{1}{\bar{\alpha}_{sk+j}}\frac{\underline{V}_k r'_{k,j+1}}{\|r_1\|}e_{sk+j}^T\right)\bar{T}_{sk+j}^{-1}e_1,$$

which allows us to write

$$\frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} = z_1 - AZ_{sk+j}\bar{T}_{sk+j}^{-1}e_1.$$

Now, let $\rho(x) = 1 + x\phi(x)$, with $\phi(x) = \sum_{i=0}^{sk+j-1}\psi_{i+1}x^i$ a polynomial of degree not exceeding $sk+j-1$. Then

$$\begin{aligned}
\frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} &= z_1 - AZ_{sk+j}\bar{T}_{sk+j}^{-1}e_1 + (\rho(A)z_1 - \rho(A)z_1) \\
&= -A\phi(A)z_1 - AZ_{sk+j}\bar{T}_{sk+j}^{-1}e_1 + \rho(A)z_1 \\
&= -AZ_{sk+j}\phi(\bar{T}_{sk+j})e_1 - AZ_{sk+j}\bar{T}_{sk+j}^{-1}e_1 + \rho(A)z_1 \\
&= -AZ_{sk+j}(\phi(\bar{T}_{sk+j}) + \bar{T}_{sk+j}^{-1})e_1 + \rho(A)z_1 \\
&= -AZ_{sk+j}\bar{T}_{sk+j}^{-1}\rho(\bar{T}_{sk+j})e_1 + \rho(A)z_1. \qquad (5.168)
\end{aligned}$$

By Lemma 3, recall that

$$\rho(A)z_1 = Z_{sk+j}\rho(\bar{T}_{sk+j})e_1 + c_{sk+j}\mathcal{Y}_k r'_{k,j+1},$$

and, multiplying by W^T, we have

$$\begin{aligned}
W^T\rho(A)z_1 &= W^T\left(Z_{sk+j}\rho(\bar{T}_{sk+j})e_1 + c_{sk+j}\mathcal{Y}_k r'_{k,j+1}\right) \\
&= \rho(\bar{T}_{sk+j})e_1,
\end{aligned}$$

since $W^T Z_{sk+j} = I$ and $W^T\mathcal{Y}_k r'_{k,j+1} = 0_{1,sk+j}$. Now, we can write

$$\begin{aligned}
\frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} &= -AZ_{sk+j}\bar{T}_{sk+j}^{-1}W^T\rho(A)z_1 + \rho(A)z_1 \\
&= \left(I - AZ_{sk+j}\bar{T}_{sk+j}^{-1}W^T\right)\rho(A)z_1,
\end{aligned}$$

and substituting $z_1 = r_1/\|r_1\|$, we obtain

$$\mathcal{Y}_k r'_{k,j+1} = \left(I - AZ_{sk+j}\bar{T}_{sk+j}^{-1}W^T\right)\rho(A)r_1,$$

which gives the desired result. □

The following lemma describes the construction of the basis W.

Lemma 5. *Assume that $z_1, \dots, z_{sk+j+1} \in \mathbb{R}^n$ are linearly independent and write $Z_{sk+j} = [z_1, \dots, z_{sk+j}]$. Then $W_0^T = [I_{sk+j}, 0_{1,sk+j}]\underline{Z}^+_{sk+j+1}$ has the property $W_0^T Z_{sk+j} = I$ and $W_0^T z_{sk+j+1} = 0_{1,sk+j}$. Furthermore, its spectral norm is minimal among all matrices having this property.*

Proof. By the definition of W_0, $Z^+_{sk+j+1} = [W_0, w]^T$ for some w. Since we assume that $z_1, \dots, z_{sk+j+1}$ are linearly independent,

$$[W_0, w]^T [Z_{sk+j}, z_{sk+j+1}] = Z^+_{sk+j} Z_{sk+j} = I.$$

Then $W_0^T Z_{sk+j} = I_{sk+j}$ and $W_0^T z_{sk+j+1} = 0_{1,sk+j}$.

Now, assume W is some other matrix such that $W^T Z_{sk+j} = I$ and $W^T z_{sk+j+1} = 0_{1,sk+j}$ hold. Then $W^T [Z_{sk+j}, z_{sk+j+1}] = [I_{sk+j}, 0_{1,sk+j}]$. Thus,

$$W^T Z_{sk+j+1} Z^+_{sk+j+1} = [I_{sk+j}, 0_{1,sk+j}]\underline{Z}^+_{k,j} = W_0^T$$

, and hence $\|W_0\| \leq \|W\| \cdot \left\|Z_{sk+j+1} Z^+_{sk+j+1}\right\| \leq \|W\|$. □

We can now present the main result.

Theorem 6. *Assume* (5.166) *holds and let $W_0^T = [I_{sk+j}, 0_{1,sk+j}] Z^+_{sk+j+1} \in \mathbb{R}^{(sk+j)\times n}$. If $z_1, \dots, z_{sk+j+1}$ are linearly independent, then*

$$\left\|\mathcal{Y}_k r'_{k,j+1}\right\| \leq (1 + K_{k,j}) \min_{\rho \in \mathbb{P}_{sk+j}, \rho(0)=1} \left\|\rho(A + \delta A_{k,j}) r_1\right\|, \tag{5.169}$$

where $K_{k,j} = \left\|(A Z_{sk+j} - \epsilon \bar{\mathfrak{d}}_{k,j}) \bar{T}^{-1}_{sk+j} W_0^T\right\|$ and $\delta A_{k,j} = -\epsilon \mathfrak{d}_{k,j} Z^+_{sk+j}$.

Proof. Since $z_1, \dots, z_{sk+j+1}$ are linearly independent, $Z^+_{sk+j} Z_{sk+j} = I$. Then

$$\delta A_{k,j} = -\epsilon \bar{\mathfrak{d}}_{k,j} Z^+_{sk+j} \in \mathbb{R}^{n \times n}$$

satisfies $\delta A_{k,j} Z_{sk+j} = -\epsilon \bar{\mathfrak{d}}_{k,j}$. Thus (5.166) can be written as

$$(A + \delta A_{k,j}) Z_{sk+j} = Z_{sk+j} \bar{T}_{sk+j} - \frac{1}{\bar{\alpha}_{sk+j}} \frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} e^T_{sk+j}. \tag{5.170}$$

Then, by Lemma 4, for any $\rho \in \mathbb{P}_{sk+j}$ with $\rho(0) = 1$, we obtain

$$\begin{aligned} \mathcal{Y}_k r'_{k,j+1} &= (I - (A + \delta A_{k,j}) Z_{sk+j} \bar{T}^{-1}_{sk+j} W_0^T) \cdot \rho(A + \delta A_{k,j}) r_1 \\ &= (I - (A Z_{sk+j} - \epsilon \bar{\mathfrak{d}}_{k,j}) \bar{T}^{-1}_{sk+j} W_0^T) \cdot \rho(A + \delta A_{k,j}) r_1. \end{aligned}$$

Thus, we can bound the norm of the left hand side by

$$\left\|\mathcal{Y}_k r'_{k,j+1}\right\| \leq \left(1 + \left\|(A Z_{sk+j} - \epsilon \bar{\mathfrak{d}}_{k,j}) \bar{T}^{-1}_{sk+j} W_0^T\right\|\right) \cdot \left\|\rho(A + \delta A_{k,j}) r_1\right\|.$$

Since this holds for any $\rho(x)$ with $\rho(0) = 1$, the inequality is true for the minimizing polynomial, which leads to the bound. □

Note that $\zeta_{k,j} = \min_{\rho\in\mathbb{P}_{sk+j},\rho(0)=1} \|\rho(A+\delta A_{k,j})r_1\|$ is the $(sk+j)^{\text{th}}$ residual norm of exact GMRES applied to the perturbed matrix $A+\delta A_{k,j}$, which decreases monotonically with increasing $(sk+j)$ [176].

Since we have $K_{k,j} = \left\|(AZ_{sk+j} - \epsilon\bar{\mathfrak{d}}_{k,j})\bar{T}_{sk+j}^{-1}W_0^T\right\|$, we can bound $K_{k,j}$ as

$$K_{k,j} \leq \left(\sqrt{sk+j}\left\|A\right\| + \epsilon\left\|\bar{\mathfrak{d}}_{k,j}\right\|\right)\left\|\bar{T}_{sk+j}^{-1}\right\| \cdot \left\|W_0\right\|.$$

Then, assuming $\left\|\bar{T}_{sk+j}^{-1}\right\|$ and $\left\|W_0\right\| \leq \|Z_{sk+j+1}^{+}\| \leq \|\mathfrak{Y}_k^{+}\|\|\mathfrak{R}_{k,j}^{\prime+}\|$ are bounded, $\left\|\mathcal{Y}_k r_{k,j+1}^{\prime}\right\|$ is on the order $O(\zeta_{k,j})$. We therefore expect convergence of the CA-(BI)CG residual when $K_{k,j}$ increases at a slower rate than $\zeta_{k,j}$ decreases, for all values of k. Unfortunately, as in the BICG case, we can not determine $K_{k,j}$ *a priori*, although we can make some meaningful observations based on the bound in (5.169).

Note that in the case of CG (SPD A), we have $\left\|\mathcal{Y}_k r_{k,j+1}^{\prime}\right\|_2 = \left\|r_{sk+j+1}\right\|_2 = \left\|e_{sk+j+1}^{*}\right\|_A$, where e_{sk+j}^{*} denotes the solution error $e_{sk+j}^{*} = x^{*} - x_{sk+j}$ for true solution x^{*}. Thus in this case Theorem 6 gives a bound on the error of finite precision s-step CG. It remains future work to determine under what conditions $\left\|e_{sk+j+1}^{*}\right\|_A < \left\|e_{sk+j}^{*}\right\|_A$ for CA-CG.

5.2.4 The Linearly Dependent Case

In the analysis above, we assumed linear independence among the residual vectors (which are scalar multiples of the Lanczos vectors). For many linear systems, however, convergence of classical (BI)CG in finite precision is still observed despite numerical rank deficiency of the basis. In [176] it is shown how the residual norm can be bounded absent the assumption of linear independence, which gives insight into why convergence still occurs in such cases. We will now prove similar bounds, relaxing the constraint that $z_1, \ldots, z_{sk+j+1} \in \mathbb{R}^n$ be linearly independent. Again, our analysis extends that of Tong and Ye [176] for classical BICG.

We note that in the s-step case, there are two potential causes of a rank-deficient basis. Since we have $R_{sk+j} = \mathfrak{Y}_{k,j}\mathfrak{R}_{k,j}^{\prime}$, linear dependence can occur as a result of the finite precision Lanczos process, as in the classical method, as well as from numerical rank deficiencies in the generated s-step polynomial bases $\mathcal{Y}_k$.

Given $A \in \mathbb{R}^{n\times n}$ and $C \in \mathbb{R}^{n'\times n'}$, $AE - EC = Z$ corresponds to the linear system with coefficient matrix $A \otimes I_{n'} - I_n \otimes C$. This system has a unique solution if and only if $\lambda(A) \cap \lambda(C) = \emptyset$, or, equivalently, if $\text{sep}(A, C) := \|(A \otimes I_{n'} - I_n \otimes C)^{-1}\|^{-1} > 0$, which depends on the spectral gap of A and C (see [84]).

Theorem 7. *Assume* (5.166) *holds, and let* μ *be a complex number such that* $\text{sep}(A - \mu I, \bar{T}_{sk+j}) \gg 0$. *Then*

$$\left\|\mathcal{Y}_k r_{k,j+1}^{\prime}\right\| \leq K_{k,j} \min_{\rho\in\mathbb{P}_{sk+j},\rho(0)=1} \left(\left\|\rho(\bar{T}_{sk+j})\right\| + \|\rho(A-\mu I)\|\right)\|r_1\|,$$

where

$$K_{k,j} = \frac{\sqrt{sk+j}(\operatorname{sep}(A-\mu I, \bar{T}_{sk+j}) + |\mu|) + \epsilon \|\bar{\mathfrak{d}}_{k,j}\|_F}{\operatorname{sep}(A-\mu I, \bar{T}_{sk+j})} \cdot \max\left(1, \|\rho(A-\mu I)\| \cdot \left\|\rho(\bar{T}_{sk+j})\right\|\right).$$

Proof. By (5.166),

$$(A-\mu I)Z_{sk+j} = Z_{sk+j}\bar{T}_{sk+j} - \frac{1}{\bar{\alpha}_{sk+j}} \frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} e_{sk+j}^T + \epsilon \bar{\mathfrak{d}}_{k,j} - \mu Z_{sk+j}. \tag{5.171}$$

Then since $\operatorname{sep}(A-\mu I, \bar{T}_{sk+j}) > 0$, the equation

$$(A-\mu I)E_{sk+j} = E_{sk+j}\bar{T}_{sk+j} - \epsilon \bar{\mathfrak{d}}_{k,j} + \mu Z_{sk+j} \tag{5.172}$$

has a unique solution E_{sk+j} with

$$\|E_{sk+j}\|_F \leq \frac{\|-\epsilon \bar{\mathfrak{d}}_{k,j} + \mu Z_{sk+j}\|_F}{\operatorname{sep}(A-\mu I, \bar{T}_{sk+j})} \leq \frac{\epsilon \|\bar{\mathfrak{d}}_{k,j}\|_F + |\mu|\sqrt{sk+j}}{\operatorname{sep}(A-\mu I, \bar{T}_{sk+j})}.$$

Combining (5.171) and (5.172), we can write

$$(A-\mu I)(Z_{sk+j} + E_{sk+j}) = (Z_{sk+j} + E_{sk+j})\bar{T}_{sk+j} - \frac{1}{\bar{\alpha}_{sk+j}} \frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} e_{sk+j}^T.$$

Thus, for any $\rho \in \mathbb{P}_{sk+j}$, $\rho(0) = 1$, we have, by (5.168),

$$\frac{\mathcal{Y}_k r'_{k,j+1}}{\|r_1\|} = \rho(A-\mu I)(Z_{sk+j} + E_{sk+j})e_1 - (A-\mu I)(Z_{sk+j} + E_{sk+j})\bar{T}_{sk+j}^{-1}\rho(\bar{T}_{sk+j})e_1,$$

and thus

$$\begin{aligned}\frac{\left\|\mathcal{Y}_k r'_{k,j+1}\right\|}{\|r_1\|} \leq{}& (\|Z_{sk+j}\| + \|E_{sk+j}\|)\,\|\rho(A-\mu I)\| \\ &+ \|A-\mu I\|\,(\|Z_{sk+j}\| + \|E_{sk+j}\|)\,\left\|\bar{T}_{sk+j}^{-1}\right\|\,\left\|\rho(\bar{T}_{sk+j})\right\|.\end{aligned}$$

Since

$$\|Z_{sk+j}\| + \|E_{sk+j}\| \leq \sqrt{sk+j} + \frac{\epsilon \|\bar{\mathfrak{d}}_{k,j}\|_F + |\mu|\sqrt{sk+j}}{\operatorname{sep}(A-\mu I, \bar{T}_{sk+j})},$$

we obtain the desired result. □

Note that in this case, if μ is such that $\operatorname{sep}(A-\mu I, \bar{T}_{sk+j})$ is large, the quantity $K_{k,j}$ depends heavily on $\left\|\bar{T}_{sk+j}^{-1}\right\|$. The minimizing polynomial part of the bound now depends on both $\rho(\bar{T}_{sk+j})$ and $\rho(A-\mu I)$.

5.2.5 Extensions: Perturbation Theory

We can think of (5.170) as an exact subspace relation for a perturbed A, i.e., the quantities $\mathfrak{Y}_k$, $\mathfrak{R}'_{k,j}$, and T_{sk+j} produced by the finite precision CA-(BI)CG algorithm satisfy an exact subspace recurrence (5.170) for the perturbed system $A + \delta A_{k,j}$. This means that the eigenvalues of the computed matrix T_{sk+j} generated by the s-step algorithm are among the eigenvalues of the perturbed matrix $A - \epsilon \mathfrak{d}_{k,j} \mathfrak{R}^+_{k,j} \mathfrak{Y}^+_k$. This means that the Lanczos vectors computed by the CA-(BI)CG algorithm, $\mathfrak{Y}_k \mathfrak{R}'_{k,j}$, span Krylov spaces of a matrix within $\epsilon \left\| \mathfrak{d}_{k,j} \mathfrak{R}^+_{k,j} \mathfrak{Y}^+_k \right\|$ of A. Similar observations have been made for classical finite precision Krylov methods [141, 202].

In his doctoral thesis [202], Zemke proved a number of useful properties for any perturbed Krylov decomposition of the form

$$AQ_i = Q_i C_i + q_{i+1} c_{i+1} - F_i.$$

Since (5.166) has this form, many of Zemke's results are applicable to the s-step variants. We also believe that using (5.170), the 'augmented' backward stability analysis of Paige [142] could be extended to the s-step case.

5.3 Maximum Attainable Accuracy of CA-KSMs

In each iteration i of classical Krylov methods for solving linear systems, the solution x_{i+1} and residual r_{i+1} are updated as

$$x_{i+1} = x_i + \alpha_i p_i \quad r_{i+1} = r_i - \alpha_i A p_i \tag{5.173}$$

or something similar. The above applies specifically to conjugate gradient (CG) and biconjugate gradient (BICG); similar formulas describe steepest descent, conjugate gradient squared (CGS), stabilized biconjugate gradient (BICGSTAB), and other recursively updated residual methods; for a thorough introduction to classical KSMs, see [153]. For simplicity, we restrict our discussion here to linear systems where A is real, square with dimension n, and full rank.

The accuracy of classical KSMs in finite precision is studied extensively in the literature (see, e.g., [86, 96, 125, 124, 161, 181]). Such analyses stem from the observation that x_{i+1} and r_{i+1} in (5.173) have different round-off patterns in finite precision. That is, the expression for x_{i+1} does not depend on r_{i+1}, nor does the expression for r_{i+1} depend on x_{i+1}. Therefore, computational errors made in x_{i+1} are not self-correcting. These errors can accumulate over many iterations and cause deviation of the *true residual*, $b - Ax_{i+1}$, and the *updated residual*, r_{i+1}. Writing the true residual as $b - Ax_{i+1} = r_{i+1} + (b - Ax_{i+1} - r_{i+1})$, we can bound its norm by

$$\|b - Ax_{i+1}\| \leq \|r_{i+1}\| + \|b - Ax_{i+1} - r_{i+1}\|.$$

When the updated residual r_{i+1} is much larger than $b - Ax_{i+1} - r_{i+1}$, the true residual and the updated residual will be of similar magnitude. However, as the updated residual converges,

Algorithm 27 Conjugate Gradient (CG)

Input: $n \times n$ symmetric positive definite matrix A, length-n vector b, and initial approximation x_1 to $Ax = b$

Output: Approximate solution x_{i+1} to $Ax = b$ with updated residual r_{i+1}

1: $r_1 = b - Ax_1$, $p_1 = r_1$
2: **for** $i = 1, 2, \ldots,$ until convergence **do**
3: $\quad \alpha_i = r_i^T r_i / p_i^T A p_i$
4: $\quad q_i = \alpha_i p_i$
5: $\quad x_{i+1} = x_i + q_i$
6: $\quad r_{i+1} = r_i - A q_i$
7: $\quad \beta_i = r_{i+1}^T r_{i+1} / r_i^T r_i$
8: $\quad p_{i+1} = r_{i+1} + \beta_i p_i$
9: **end for**

i.e., $\|r_{i+1}\| \to 0$, the size of the true residual depends on $\|b - Ax_{i+1} - r_{i+1}\|$. This term denotes the size of the deviation between the true and updated residuals. If this deviation grows large, it can limit the *maximum attainable accuracy*, i.e., the accuracy with which we can solve $Ax = b$ on a computer with unit round-off ϵ.

Following similar analyses for standard KSMs (e.g., [86, 125, 161, 181]), we prove bounds for the maximum attainable accuracy (i.e., the deviation of the true and computed residual) in CA-CG and CA-BICG in finite precision. We provide the first quantitative analysis of round-off error in CA-KSMs which limits the maximum attainable accuracy. We begin with a brief review of the CA-CG method. A derivation of CA-BICG appears in Section 4.2. We note that this section has been adapted from work that first appeared in [31].

5.3.1 Communication-Avoiding Conjugate Gradient

We briefly review CA-CG, shown in Algorithm 28. We chose CA-CG for simplicity, although the same general techniques used here can be applied to other CA-KSMs. For reference, CG is given in Algorithm 27. The following brief derivation is meant as a review for the familiar reader and to establish notation; we refer the unfamiliar reader to numerous other works such as [63, 102, 117, 175]. The CA-CG method can also be obtained from Algorithm 8 in Section 4.2 by setting $A = A^T$ and simplifying.

CA-CG consists of an outer loop, indexed by k, and an inner loop, which iterates over $j \in \{1, \ldots, s\}$, where we assume $s \ll n$. For clarity, we globally index iterations by $i \equiv sk + j$. Our goal is to determine the information required to compute the CG vectors p_{sk+j}, r_{sk+j}, and x_{sk+j} for $j \in \{2, \ldots, s+1\}$ and $s > 0$, given p_{sk+1}, r_{sk+1}, and x_{sk+1}. It follows from the properties of CG that

$$\begin{aligned} p_{sk+j}, r_{sk+j} &\in \mathcal{K}_{s+1}(A, p_{sk+1}) + \mathcal{K}_s(A, r_{sk+1}) \quad \text{and} \\ x_{sk+j} - x_{sk+1} &\in \mathcal{K}_s(A, p_{sk+1}) + \mathcal{K}_{s-1}(A, r_{sk+1}). \end{aligned} \tag{5.174}$$

for $j \in \{1, \ldots, s+1\}$. Then we can update the CG vectors for the next s iterations using the matrices

$$\begin{aligned} \mathcal{P}_k &= [\rho_0(A)p_{sk+1}, \ldots, \rho_s(A)p_{sk+1}], \quad \text{such that} \quad \text{span}(\mathcal{P}_k) = \mathcal{K}_{s+1}(A, p_{sk+1}) \quad \text{and} \\ \mathcal{R}_k &= [\rho_0(A)r_{sk+1}, \ldots, \rho_{s-1}(A)r_{sk+1}], \quad \text{such that} \quad \text{span}(\mathcal{R}_k) = \mathcal{K}_s(A, r_{sk+1}), \end{aligned} \tag{5.175}$$

where $\rho_j(z)$ is a polynomial of degree j satisfying the three-term recurrence

$$\begin{aligned} &\rho_0(z) = 1, \qquad \rho_1(z) = (z - \theta_0)\rho_0(z)/\gamma_0, \quad \text{and} \\ &\rho_j(z) = \left((z - \theta_{j-1})\rho_{j-1}(z) - \sigma_{j-2}\rho_{j-2}(z)\right)/\gamma_{j-1}. \end{aligned} \tag{5.176}$$

Note that this is the same as the polynomial recurrence in (4.1); we use different variables in the recurrence to avoid the use of variables decorated with hats in (4.1), as we will use hats exclusively for distinguishing quantities computed in finite-precision in this section.

Let $\mathcal{Y}_k = [\mathcal{P}_k, \mathcal{R}_k]$ and let $\underline{\mathcal{Y}}_k$ be the same as $\mathcal{Y}_k$ except with all zeros in columns $s+1$ and $2s+1$. Under certain assumptions on the nonzero structure of A, $\mathcal{Y}_k$ can be computed in each outer loop for the same asymptotic latency cost as a single SpMV (see Section 3.2 for details). Since the columns of $\mathcal{Y}_k$ satisfy (5.176), we can write

$$A\underline{\mathcal{Y}}_k = \mathcal{Y}_k\mathcal{B}_k. \tag{5.177}$$

Note that $\mathcal{B}_k$ is a $(2s+1)\times(2s+1)$ tridiagonal matrix with diagonal blocks of the form (4.2); again, we use this notation to avoid confusion with the tridiagonal matrix of Lanczos coefficients generated by CG.

By (5.174), there exist vectors $p'_{k,j}$, $r'_{k,j}$, and $x'_{k,j}$ that represent the coordinates for the CG iterates p_{sk+j}, r_{sk+j}, and $x_{sk+j} - x_{sk+1}$, respectively, in $\mathcal{Y}_k$ for $j \in \{1, \ldots, s+1\}$. That is,

$$p_{sk+j} = \mathcal{Y}_k p'_{k,j}, \quad r_{sk+j} = \mathcal{Y}_k r'_{k,j}, \quad \text{and} \quad x_{sk+j} - x_{sk+1} = \mathcal{Y}_k x'_{k,j}. \tag{5.178}$$

The CG iterate updates, for $j \in \{1, \ldots, s\}$, can then be written

$$\begin{aligned} \mathcal{Y}_k x'_{k,j+1} &= \mathcal{Y}_k x'_{k,j} + \alpha_{sk+j}\mathcal{Y}_k p'_{k,j}, \\ \mathcal{Y}_k r'_{k,j+1} &= \mathcal{Y}_k r'_{k,j} - \alpha_{sk+j}A\mathcal{Y}_k p'_{k,j}, \quad \text{and} \\ \mathcal{Y}_k p'_{k,j+1} &= \mathcal{Y}_k r'_{k,j+1} + \beta_{sk+j}\mathcal{Y}_k p'_{k,j}. \end{aligned}$$

By (5.174), $\mathcal{Y}_k p'_{k,j} = \underline{\mathcal{Y}}_k p'_{k,s}$ for $j \in \{1, \ldots, s\}$. We then use (5.177) to obtain

$$\mathcal{Y}_k r'_{k,j+1} = \mathcal{Y}_k r'_{k,j} - \alpha_{sk+j}\mathcal{Y}_k\mathcal{B}_k p'_{k,j}.$$

Therefore, in the inner loop of CA-CG, rather than update the CG vectors explicitly, we instead update their coordinates in $\mathcal{Y}_k$, i.e.,

$$\begin{aligned} x'_{k,j+1} = x'_{k,j} + \alpha_{sk+j}p'_{k,j}, \quad r'_{k,j+1} = r'_{k,j} - \alpha_{sk+j}\mathcal{B}_k p'_{k,j}, \\ \text{and} \quad p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j}p'_{k,j}. \end{aligned}$$

Algorithm 28 Communication-Avoiding CG (CA-CG)

Input: $n \times n$ symmetric positive definite matrix A, length-n vector b, and initial approximation x_1 to $Ax = b$

Output: Approximate solution x_{sk+s+1} to $Ax = b$ with updated residual r_{sk+s+1}

1: $r_1 = b - Ax_1$, $p_1 = r_1$
2: **for** $k = 0, 1, \ldots,$ until convergence **do**
3: Compute $\mathcal{Y}_k = [\mathcal{P}_k, \mathcal{R}_k]$ according to (5.175)
4: Compute $G_k = \mathcal{Y}_k^T \mathcal{Y}_k$
5: Assemble $\mathcal{B}_k$ such that (5.177) holds
6: $p'_{k,1} = [1, 0_{1,2s}]^T$, $r'_{k,1} = [0_{1,s+1}, 1, 0_{1,s-1}]^T$, $x'_{k,1} = [0_{1,2s+1}]^T$
7: **for** $j = 1$ to s **do**
8: $\alpha_{sk+j} = \left(r'^T_{k,j} G_k r'_{k,j}\right) / \left(p'^T_{k,j} G_k \mathcal{B}_k p'_{k,j}\right)$
9: $q'_{k,j} = \alpha_{sk+j} p'_{k,j}$
10: $x'_{k,j+1} = x'_{k,j} + q'_{k,j}$
11: $r'_{k,j+1} = r'_{k,j} - \mathcal{B}_k q'_{k,j})$
12: $\beta_{sk+j} = \left(r'^T_{k,j+1} G_k r'_{k,j+1}\right) / \left(r'^T_{k,j} G_k r'_{k,j}\right)$
13: $p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j} p'_{k,j}$
14: **end for**
15: Recover iterates $\{p_{sk+s+1}, r_{sk+s+1}, x_{sk+s+1}\}$ according to (5.178)
16: **end for**

Thus we can eliminate the communication cost of the SpMV with A in each iteration, and the length-$(2s+1)$ vector updates can be performed locally (parallel)/in cache (sequential) in each inner loop iteration.

We can also avoid communication in computing inner products. Letting $G_k = \mathcal{Y}_k^T \mathcal{Y}_k$, and using (5.178) and (5.177), α_{sk+j} and β_{sk+j} can be computed by

$$\alpha_{sk+j} = \left(r'^T_{k,j} G_k r'_{k,j}\right) / \left(p'^T_{k,j} G_k \mathcal{B}_k p'_{k,j}\right) \quad \text{and}$$
$$\beta_{sk+j} = \left(r'^T_{k,j+1} G_k r'_{k,j+1}\right) / \left(r'^T_{k,j} G_k r'_{k,j}\right).$$

The matrix G_k can be computed by one reduction per outer loop, the same asymptotic parallel latency cost as a single inner product (see Section 3.1). As $\mathcal{B}_k$ and G_k are dimension $(2s+1) \times (2s+1)$, α_{sk+j} and β_{sk+j} can be computed locally/in cache within the inner loop. The matrix G_k can be computed by one reduction per outer loop, the same asymptotic parallel latency cost as a single inner product (see Section 3.1). As $\mathcal{B}_k$ and G_k are dimension $(2s+1) \times (2s+1)$, α_{sk+j} and β_{sk+j} can be computed locally/in cache within the inner loop.

5.3.2 Finite Precision CA-CG

As in previous sections, we use a standard model of floating point arithmetic. Recall that

$$\text{fl}(x+y) = x + y + \delta \quad \text{with } |\delta| \leq \epsilon(|x+y|), \quad \text{and}$$

$$\mathrm{fl}(Ax) = Ax + \delta \qquad \text{with } |\delta| \leq \epsilon N_A \, |A| \, |x| \,,$$

where ϵ is the unit round-off of the machine, x, $y \in \mathbb{R}^n$, and N_A is a constant associated with the matrix-vector multiplication (e.g., the maximal number of nonzero entries in a row of A). All absolute values and inequalities are componentwise. Using this model, we can also write

$$\mathrm{fl}(y + Ax) = y + Ax + \delta \;\text{ with } |\delta| \leq \epsilon(|y + Ax| + N_A \, |A| \, |x|)$$

where, as in the remainder this analysis, we ignore higher powers of ϵ. We now perform an analysis of round-off error in the CA-CG method. Our goal is to bound the norm of the difference between the true and updated residual in terms of quantities which can be computed inexpensively in each iteration. We note that our bound also holds for CA-BICG (see Section 4.2), which has identical formulas for computing p_{sk+j+1}, r_{sk+j+1}, and $x_{sk+j+1} - x_{sk+1}$.

5.3.3 Basis Computation in Finite Precision

We begin by bounding the error in finite precision computation of $\hat{\mathcal{Y}}_k$, where, here and in the remaining analysis, we use hats to denote quantities computed in finite precision. In exact arithmetic, (5.177) holds, but in finite precision, we have $A\underline{\hat{\mathcal{Y}}}_k = \hat{\mathcal{Y}}_k \mathcal{B}_k - E_k$, where E_k represents the error due to round-off. We seek a componentwise bound on E_k. The analysis below is similar to that for the finite precision Lanczos process (see, e.g., Theorem 3.1 in [12]).

In finite precision, we can write

$$\hat{\mathcal{Y}}_k = \big[\hat{\mathcal{P}}_k, \, \hat{\mathcal{R}}_k\big] = \big[\hat{\rho}_0(A)\hat{p}_{sk+1}, \, \ldots, \, \hat{\rho}_s(A)\hat{p}_{sk+1}, \, \hat{\rho}_0(A)\hat{r}_{sk+1}, \, \ldots, \, \hat{\rho}_{s-1}(A)\hat{r}_{sk+1}\big], \tag{5.179}$$

where $\hat{\rho}_i$ is a polynomial of degree i, as defined in (5.176). In terms of starting vector y and parameters γ_i, θ_i, and σ_i (entries of $\mathcal{B}_k$), in finite precision (5.176) becomes

$$\hat{\rho}_{i+1}(A)y = \frac{1}{\gamma_i}\Big((A - \theta_i I)\hat{\rho}_i(A)y - \sigma_{i-1}\hat{\rho}_{i-1}(A)y\Big) + \nu_{y,i+1}, \tag{5.180}$$

where

$$|\nu_{y,i+1}| \leq \epsilon\Big((2 + N_A) \, |A| \, |\hat{\rho}_i(A)y| + 3 \, |\theta_i| \, |\hat{\rho}_i(A)y| + 2 \, |\sigma_{i-1}| \, |\hat{\rho}_{i-1}(A)y| \Big) / \, |\gamma_i| \,. \tag{5.181}$$

We can rearrange (5.180) to obtain

$$A\hat{\rho}_i(A)y = \gamma_i\hat{\rho}_{i+1}(A)y + \theta_i\hat{\rho}_i(A)y + \sigma_{i-1}\hat{\rho}_{i-1}(A)y - \gamma_i\nu_{y,i+1}. \tag{5.182}$$

Premultiplying (5.179) by A and combining (5.182) with (5.181) gives the desired expression, $A\underline{\hat{\mathcal{Y}}}_k = \hat{\mathcal{Y}}_k\mathcal{B}_k - E_k$, with

$$\big|E_k\big| \leq \epsilon\Big((3 + N_A)\big|A\big|\big|\underline{\hat{\mathcal{Y}}}_k\big| + 4\big|\hat{\mathcal{Y}}_k\big|\big|\mathcal{B}_k\big|\Big). \tag{5.183}$$

5.3.4 Iterate Updates in Finite Precision

We now consider the finite precision error in updating $\hat{x}'_{k,j+1}$ and $\hat{r}'_{k,j+1}$ in each inner loop iteration and performing the basis change to obtain $\hat{x}_{sk+j+1}$ and $\hat{r}_{sk+j+1}$, which occurs when $j = s$ or when $sk + j$ is the terminal iteration. In the inner loop, the formulas for updating the coordinates are

$$x'_{k,j+1} = x'_{k,j} + q'_{k,j}, \quad \text{and} \tag{5.184}$$
$$r'_{k,j+1} = r'_{k,j} - \mathcal{B}_k q'_{k,j}, \tag{5.185}$$

where $q'_{k,j} = \alpha_{sk+j} p'_{k,j}$. When (5.184) and (5.185) are implemented in finite precision, they become

$$\hat{x}'_{k,j+1} = \text{fl}\left(\hat{x}'_{k,j} + \hat{q}'_{k,j}\right) = \hat{x}'_{k,j} + \hat{q}'_{k,j} + \xi_{k,j+1}, \tag{5.186}$$
$$\text{where} \quad \left|\xi_{k,j+1}\right| \leq \epsilon\left|\hat{x}'_{k,j+1}\right|, \quad \text{and} \tag{5.187}$$
$$\hat{r}'_{k,j+1} = \text{fl}\left(\hat{r}'_{k,j} - \mathcal{B}_k\hat{q}'_{k,j}\right) = \hat{r}'_{k,j} - \mathcal{B}_k\hat{q}'_{k,j} + \eta_{k,j+1} \tag{5.188}$$
$$\text{where} \quad \left|\eta_{k,j+1}\right| \leq \epsilon\Big(\left|\hat{r}'_{k,j+1}\right| + N_{\mathcal{B}}\left|\mathcal{B}_k\right|\left|\hat{q}'_{k,j}\right|\Big), \tag{5.189}$$

and under the assumptions that we use three-term polynomials for the s-step basis computation, $N_{\mathcal{B}} = 3$. If $j < s$ and $sk + j$ is not the terminal iteration, we denote (but do not compute) the solution and updated residual by

$$x_{sk+j+1} = \hat{\mathcal{Y}}_k\hat{x}'_{k,j+1} + \hat{x}_{sk+1}, \quad \text{and} \tag{5.190}$$
$$r_{sk+j+1} = \hat{\mathcal{Y}}_k\hat{r}'_{k,j+1}. \tag{5.191}$$

At the end of each outer loop iteration (when $j = s$) and at the terminal iteration, the solution and residual are computed in finite precision, denoted $\hat{x}_{sk+j+1}$ and $\hat{r}_{sk+j+1}$, respectively. The error in computing these terms can be bounded as follows:

$$\hat{x}_{sk+j+1} = \text{fl}\left(\hat{\mathcal{Y}}_k\hat{x}'_{k,j+1} + \hat{x}_{sk+1}\right) = \hat{\mathcal{Y}}_k\hat{x}'_{k,j+1} + \hat{x}_{sk+1} + \phi_{sk+j+1} = x_{sk+j+1} + \phi_{sk+j+1}, \tag{5.192}$$
$$\text{where} \quad \left|\phi_{sk+j+1}\right| \leq \epsilon\Big(\left|\hat{x}_{sk+j+1}\right| + N_{\mathcal{Y}}\left|\hat{\mathcal{Y}}_k\right|\left|\hat{x}'_{k,j+1}\right|\Big), \quad \text{and} \tag{5.193}$$
$$\hat{r}_{sk+j+1} = \text{fl}\left(\hat{\mathcal{Y}}_k\hat{r}'_{k,j+1}\right) = \hat{\mathcal{Y}}_k\hat{r}'_{k,j+1} + \psi_{sk+j+1} = r_{sk+j+1} + \psi_{sk+j+1}, \tag{5.194}$$
$$\text{where} \quad \left|\psi_{sk+j+1}\right| \leq \epsilon N_{\mathcal{Y}}\left|\hat{\mathcal{Y}}_k\right|\left|\hat{r}'_{k,j+1}\right|, \tag{5.195}$$

and $N_{\mathcal{Y}} = 2s + 1$.

5.3.5 Deviation of the True and Updated Residuals

Using the results in previous subsections, we obtain an upper bound on the norm of the difference between the true and updated residuals at step $sk+j$. We first prove the following lemma.

Lemma 6. *Consider step $sk+j$ of the finite precision CA-CG algorithm. Let $\delta_{sk+j+1} = b - Ax_{sk+j+1} - r_{sk+j+1}$, where x_{sk+j+1} and r_{sk+j+1} are defined in* (5.190) *and* (5.191)*, respectively. Then*

$$\begin{aligned}
\left\|\delta_{sk+j+1}\right\| \leq & \left\|b - Ax_1 - r_1\right\| + \epsilon \sum_{\ell=0}^{k-1} \left(\left\|A\right\| \left\|\hat{x}_{s\ell+s+1}\right\| + N_{\mathcal{Y}} \left(\left\|A\right\| \left\| |\hat{\mathcal{Y}}_\ell| |\hat{x}'_{\ell,s+1}| \right\| + \left\| |\hat{\mathcal{Y}}_\ell| |\hat{r}'_{\ell,s+1}| \right\| \right) \right) \\
& + \epsilon \sum_{\ell=0}^{k-1} \sum_{i=1}^{s} \left(C_1 \left\|A\right\| \left\| |\underline{\hat{\mathcal{Y}}}_\ell| |\hat{x}'_{\ell,i+1}| \right\| + C_2 \left\| |\hat{\mathcal{Y}}_\ell| |\mathcal{B}_\ell| |\hat{x}'_{\ell,i+1}| \right\| + \left\| |\hat{\mathcal{Y}}_\ell| |\hat{r}'_{\ell,i+1}| \right\| \right) \\
& + \epsilon \sum_{i=1}^{j} \left(C_1 \left\|A\right\| \left\| |\underline{\hat{\mathcal{Y}}}_k| |\hat{x}'_{k,i+1}| \right\| + C_2 \left\| |\hat{\mathcal{Y}}_k| |\mathcal{B}_k| |\hat{x}'_{k,i+1}| \right\| + \left\| |\hat{\mathcal{Y}}_k| |\hat{r}'_{k,i+1}| \right\| \right).
\end{aligned}$$

where $C_1 = 7 + 2N_A$ and $C_2 = 8 + 2N_{\mathcal{B}}$.

Proof. We can write δ_{sk+j+1} as

$$\begin{aligned}
\delta_{sk+j+1} &= b - Ax_{sk+j+1} - r_{sk+j+1} \\
&= b - A(\hat{x}_{sk+1} + \hat{\mathcal{Y}}_k \hat{x}'_{k,j+1}) - \hat{\mathcal{Y}}_k \hat{r}'_{k,j+1} \\
&= b - A\hat{x}_{sk+1} - A\underline{\hat{\mathcal{Y}}}_k (\hat{x}'_{k,j} + \hat{q}'_{k,j} + \xi_{k,j+1}) - \hat{\mathcal{Y}}_k (\hat{r}'_{k,j} - \mathcal{B}_k \hat{q}'_{k,j} + \eta_{k,j+1}) \\
&= b - A(\hat{x}_{sk+1} + \underline{\hat{\mathcal{Y}}}_k \hat{x}'_{k,j}) - \hat{\mathcal{Y}}_k \hat{r}'_{k,j} - A\underline{\hat{\mathcal{Y}}}_k \xi_{k,j+1} - \hat{\mathcal{Y}}_k \eta_{k,j+1} + E_k \hat{q}'_{k,j} \\
&= \delta_{sk+j} - A\underline{\hat{\mathcal{Y}}}_k \xi_{k,j+1} - \hat{\mathcal{Y}}_k \eta_{k,j+1} + E_k \hat{q}'_{k,j} \\
&= (b - A\hat{x}_{sk+1} - \hat{r}_{sk+1}) - \sum_{i=1}^{j} \left(A\underline{\hat{\mathcal{Y}}}_k \xi_{k,i+1} + \hat{\mathcal{Y}}_k \eta_{k,i+1} - E_k \hat{q}'_{k,i} \right) \\
&= \delta_1 - \sum_{\ell=0}^{k-1} \left(A\phi_{s\ell+s+1} + \psi_{s\ell+s+1} + \sum_{i=1}^{s} \left(A\underline{\hat{\mathcal{Y}}}_\ell \xi_{\ell,i+1} + \hat{\mathcal{Y}}_\ell \eta_{\ell,i+1} - E_\ell \hat{q}'_{\ell,i} \right) \right) \\
&\quad - \sum_{i=1}^{j} \left(A\underline{\hat{\mathcal{Y}}}_k \xi_{k,i+1} + \hat{\mathcal{Y}}_k \eta_{k,i+1} - E_k \hat{q}'_{k,i} \right).
\end{aligned} \tag{5.196}$$

Using the componentwise error bounds in (5.183), (5.187), (5.189), (5.193), and (5.195),

$$\begin{aligned}
\left|\delta_{sk+j+1}\right| \leq & \left|b - Ax_1 - r_1\right| + \epsilon \sum_{\ell=0}^{k-1} \left(\left|A\right| \left|\hat{x}_{s\ell+s+1}\right| + N_{\mathcal{Y}} \left(\left|A\right| |\hat{\mathcal{Y}}_\ell| |\hat{x}'_{\ell,s+1}| + |\hat{\mathcal{Y}}_\ell| |\hat{r}'_{\ell,s+1}| \right) \right) \\
& + \epsilon \sum_{\ell=0}^{k-1} \sum_{i=1}^{s} \left((7 + 2N_A) \left|A\right| |\underline{\hat{\mathcal{Y}}}_\ell| |\hat{x}'_{\ell,i+1}| + (8 + 2N_{\mathcal{B}}) |\hat{\mathcal{Y}}_\ell| |\mathcal{B}_\ell| |\hat{x}'_{\ell,i+1}| + |\hat{\mathcal{Y}}_\ell| |\hat{r}'_{\ell,i+1}| \right) \\
& + \epsilon \sum_{i=1}^{j} \left((7 + 2N_A) \left|A\right| |\underline{\hat{\mathcal{Y}}}_k| |\hat{x}'_{k,i+1}| + (8 + 2N_{\mathcal{B}}) |\hat{\mathcal{Y}}_k| |\mathcal{B}_k| |\hat{x}'_{k,i+1}| + |\hat{\mathcal{Y}}_k| |\hat{r}'_{k,i+1}| \right),
\end{aligned} \tag{5.197}$$

where we have used $\sum_{i=1}^{j} \left|\hat{q}'_{k,i}\right| \leq (2+\epsilon) \sum_{i=1}^{j} \left|\hat{x}'_{k,i+1}\right|$. From this follows the desired bound for δ_{sk+j+1} in terms of norms. □

Assume the algorithm terminates at step $sk+j$, so $\hat{x}_{sk+j+1}$ and $\hat{r}_{sk+j+1}$ are returned. We bound the deviation of the true residual, $b - A\hat{x}_{sk+j+1}$, and the updated residual, $\hat{r}_{sk+j+1}$, in the following theorem.

Theorem 8. *Consider step $sk+j$ of finite precision CA-CG. If the algorithm terminates at step $sk+j$ and returns solution $\hat{x}_{sk+j+1}$ and updated residual $\hat{r}_{sk+j+1}$, the norm of the deviation of the true residual and the updated residual, denoted $\hat{\delta}_{sk+j+1} = b - A\hat{x}_{sk+j+1} - \hat{r}_{sk+j+1}$, can be bounded by*

$$\left\|\hat{\delta}_{sk+j+1}\right\| \leq \left\|\delta_{sk+j+1}\right\| + \epsilon\Big(\left\|A\right\|\left\|\hat{x}_{sk+1}\right\| + (1+N_{\mathcal{Y}})\left\|A\right\|\left\|\left|\hat{\mathcal{Y}}_k\right|\left|\hat{x}'_{k,j+1}\right|\right\| + N_{\mathcal{Y}}\left\|\left|\hat{\mathcal{Y}}_k\right|\left|\hat{r}'_{k,j+1}\right|\right\|\Big).$$

Proof. Using (5.192) and (5.194), we obtain

$$\begin{aligned} b - A\hat{x}_{sk+j+1} - \hat{r}_{sk+j+1} &= b - A(x_{sk+j+1} + \phi_{sk+j+1}) - (r_{sk+j+1} + \psi_{sk+j+1}) \\ &= \delta_{sk+j+1} - A\phi_{sk+j+1} - \psi_{sk+j+1}, \end{aligned}$$

and by (5.193), we can write $\|A\phi_{sk+j+1}\| \leq \epsilon \left\|A\right\| \left(\left\|\hat{x}_{sk+1}\right\| + (1+N_{\mathcal{Y}})\left\|\left|\hat{\mathcal{Y}}_k\right|\left|\hat{x}'_{k,j+1}\right|\right\|\right)$. Combining this bound with (5.195) and Lemma 6 gives the desired result. □

To obtain a tighter bound, it is beneficial to use the quantities

$$\left\|\left|\hat{\mathcal{Y}}_\ell\right|\left|\mathcal{B}_\ell\right|\left|\hat{x}'_{\ell,i+1}\right|\right\|, \quad \left\|\left|\hat{\mathcal{Y}}_\ell\right|\left|\hat{x}'_{\ell,i+1}\right|\right\|, \quad \text{and} \quad \left\|\left|\hat{\mathcal{Y}}_\ell\right|\left|\hat{r}'_{\ell,i+1}\right|\right\|$$

above, as $\left\|\hat{\mathcal{Y}}_\ell\right\|\left\|\mathcal{B}_\ell\right\|\left\|\hat{x}'_{\ell,i+1}\right\|$, $\left\|\hat{\mathcal{Y}}_\ell\right\|\left\|\hat{x}'_{\ell,i+1}\right\|$, and $\left\|\hat{\mathcal{Y}}_\ell\right\|\left\|\hat{r}'_{\ell,i+1}\right\|$, respectively, can be overestimates due to sparsity in $\hat{x}'_{\ell,i+1}$ and $\hat{r}'_{\ell,i+1}$, respectively, for $i < s$. In Section 6.1, we use the form of the bound given in Theorem 8 to derive a residual replacement strategy, which improves the accuracy of the method.

It is also of use to derive an upper bound that is less tight, but enables easy comparison of CA-CG to the classical CG method. Comparing the bound in Theorem 8 with the bound in [181], we expect the size of the deviation of the true and updated residual at iteration $sk+j$ in CA-CG to be on the same order as that in CG when $\left\|\left|\hat{\mathcal{Y}}_\ell\right|\left|\hat{x}'_{\ell,i+1}\right|\right\| \approx \left\|\hat{\mathcal{Y}}_\ell x'_{\ell,i+1}\right\|$, $\left\|\left|\hat{\mathcal{Y}}_\ell\right|\left|\hat{r}'_{\ell,i+1}\right|\right\| \approx \left\|\hat{\mathcal{Y}}_\ell r'_{\ell,i+1}\right\|$, and $\left\|\left|\hat{\mathcal{Y}}_\ell\right|\left|\mathcal{B}_\ell\right|\left|\hat{x}'_{\ell,i+1}\right|\right\| \approx \left\|\left|A\right|\left|\hat{\mathcal{Y}}_\ell\right|\left|\hat{x}'_{\ell,i+1}\right|\right\|$. This indicates that the choice of basis $\hat{\mathcal{Y}}_k$ plays a significant role in determining the accuracy of CA-CG relative to CG, as we now show.

We now take a similar approach as in Section 5.1 and show that the deviation of the true and updated residuals in CA-CG can be written in the same form as the deviation of the true and updated residuals in CG, multiplied by an amplification factor. We prove this in the following theorem.

Theorem 9. *Assume that Algorithm 27 is implemented in floating point with relative precision ϵ and applied for $sk+j$ steps to solve $Ax = b$ where A is an n-by-n real symmetric matrix with at most N_A nonzeros per row. Let $\tau_k = \||\mathcal{B}_k|\|_2/\|A\|_2$. Let $\Gamma_k = \|\hat{\mathcal{Y}}_k^+\|_2\||\hat{\mathcal{Y}}_k|\|_2$, where the superscript '+' denotes the Moore-Penrose pseudoinverse, i.e., $\hat{\mathcal{Y}}_k^+ = (\hat{\mathcal{Y}}_k^T\hat{\mathcal{Y}}_k)^{-1}\hat{\mathcal{Y}}_k^T$, and let*

$$\bar{\Gamma}_k = \max_{\ell\in\{0,\ldots,k\}} \Gamma_\ell \geq 1 \quad \textit{and} \quad \bar{\tau}_k = \max_{\ell\in\{0,\ldots,k\}} \tau_\ell. \tag{5.198}$$

Then the norm of the deviation of the true residual and the updated residual after iteration $sk + j$ can be bounded above by

$$\|\hat{\delta}_{sk+j+1}\| \leq \|b - Ax_1 - r_1\| + \epsilon\bar{\Gamma}_k N^* \sum_{i=1}^{sk+j}(1 + 2N_A)\|A\|\|\hat{x}_{i+1}\| + \|\hat{r}_{i+1}\|,$$

where $N^* = 6 + 2s(11 + 14\bar{\tau}_k)$.

Proof. By Lemma 1, we can write, ignoring higher than first order terms in ϵ,

$$\begin{aligned}
\epsilon\||\hat{\mathcal{Y}}_\ell||\hat{x}'_{\ell,i+1}|\|_2 &\leq \epsilon\Gamma_\ell\left(\|\hat{x}_{s\ell+i+1}\|_2 + \|\hat{x}_{s\ell+1}\|_2\right),\\
\epsilon\||\hat{\mathcal{Y}}_\ell||\hat{r}'_{\ell,i+1}|\|_2 &\leq \epsilon\Gamma_\ell\|\hat{r}_{s\ell+i+1}\|_2, \quad \text{and}\\
\epsilon\||\hat{\mathcal{Y}}_\ell||\mathcal{B}_\ell||\hat{x}'_{\ell,i+1}|\|_2 &\leq \epsilon\Gamma_\ell\tau_\ell\left(\|\hat{x}_{s\ell+i+1}\|_2 + \|\hat{x}_{s\ell+1}\|_2\right),
\end{aligned} \tag{5.199}$$

for $\ell \in \{0,\ldots,k\}$ and $i \in \{1,\ldots,s\}$.

Using Theorem 8 and the bound in Lemma 6, we can write

$$\begin{aligned}
\|\hat{\delta}_{sk+j+1}\| \leq&\|b - Ax_1 - r_1\|\\
&+ \epsilon\left((1+(1+N_{\mathcal{Y}})\Gamma_k)\|A\|\|\hat{x}_{sk+1}\| + (1+N_{\mathcal{Y}})\Gamma_k\|A\|\|\hat{x}_{sk+j+1}\| + N_{\mathcal{Y}}\Gamma_k\|\hat{r}_{sk+j+1}\|\right)\\
&+ \epsilon\sum_{\ell=0}^{k-1}(1+N_{\mathcal{Y}}\Gamma_\ell)\|A\|\|\hat{x}_{s(\ell+1)+1}\| + N_{\mathcal{Y}}\Gamma_\ell\|A\|\|\hat{x}_{s\ell+1}\| + N_{\mathcal{Y}}\Gamma_\ell\|\hat{r}_{s(\ell+1)+1}\|\\
&+ \epsilon\sum_{\ell=0}^{k-1}\sum_{i}^{s}(C_1 + C_2\tau_\ell)\Gamma_\ell\|A\|\left(\|\hat{x}_{s\ell+i+1}\| + \|\hat{x}_{s\ell+1}\|\right) + C_2\Gamma_\ell\|\hat{r}_{s\ell+i+1}\|\\
&+ \epsilon\sum_{i=1}^{j}(C_1 + C_2\tau_k)\Gamma_k\|A\|\left(\|\hat{x}_{sk+i+1}\| + \|\hat{x}_{sk+1}\|\right) + C_2\Gamma_\ell\|\hat{r}_{sk+i+1}\|\\
\leq&\|b - Ax_1 - r_1\|\\
&+ \epsilon\sum_{\ell=0}^{k}(1+2(1+N_{\mathcal{Y}}))\Gamma_\ell\|A\|\|\hat{x}_{s\ell+1}\| + N_{\mathcal{Y}}\Gamma_\ell\|\hat{r}_{s\ell+1}\|\\
&+ \epsilon\sum_{\ell=0}^{k}s(C_1 + C_2\tau_\ell)\Gamma_\ell\|A\|\|\hat{x}_{s\ell+1}\|
\end{aligned}$$

$$+\epsilon\sum_{\ell=0}^{k-1}\sum_{i=1}^{s}(C_1+C_2\tau_\ell)\Gamma_\ell\|A\|\|\hat{x}_{s\ell+i+1}\|+C_2\Gamma_\ell\|\hat{r}_{s\ell+i+1}\|$$
$$+\epsilon\sum_{i=1}^{j}(C_1+C_2\tau_k+1+N_{\mathcal{Y}})\Gamma_k\|A\|\|\hat{x}_{sk+i+1}\|+(C_2+N_{\mathcal{Y}})\Gamma_k\|\hat{r}_{sk+i+1}\|.$$

We can further bound this by

$$\begin{aligned}\|\hat{\delta}_{sk+j+1}\|\leq&\|b-Ax_1-r_1\|\\&+\epsilon\bar{\Gamma}_k\sum_{i=1}^{sk+j}\big(1+3(1+N_{\mathcal{Y}})+(s+1)(C_1+C_2\bar{\tau}_k)\big)\|A\|\|\hat{x}_{i+1}\|+(C_2+2N_{\mathcal{Y}})\|\hat{r}_{i+1}\|\\\leq&\|b-Ax_1-r_1\|+\epsilon\bar{\Gamma}_kN^*\sum_{i=1}^{sk+j}(1+2N_A)\|A\|\|\hat{x}_{i+1}\|+\|\hat{r}_{i+1}\|,\end{aligned}$$

with $N^*=6+2s(11+14\bar{\tau}_k)$, where we have used $N_{\mathcal{B}}=3$, $N_{\mathcal{Y}}=2s+1$, and $C_2=8+2N_{\mathcal{B}}=14$. □

We reiterate that the value of the bound in Theorem 9 is not in its tightness, but in what it tells us about the behavior of the CA-CG method compared to the CG method. The bound in Theorem 9 is the same as the bound for the CG method (see [181]) except for the amplification factor $\bar{\Gamma}_kN^*$. This indicates the important role that the conditioning of the computed s-step Krylov bases plays in increasing roundoff error which leads to loss of accuracy. Numerical results demonstrating this can be found in Section 6.1, in which we use the bounds of this section in a technique for improving maximum attainable accuracy in CA-(BI)CG at little additional cost.

5.4 Conclusions and Future Work

Understanding the finite precision behavior of CA-KSMs is crucial to making such methods usable in practical applications. In this Chapter, we have extended many results on convergence and accuracy for finite precision KSMs to finite precision CA-KSMs, including bounds on the accuracy and convergence of eigenvalues in the finite precision CA-Lanczos method, a bound for the residual norm computed by the finite precision CA-(BI)CG algorithm, and a computable bound for the maximum attainable accuracy in finite precision CA-(BI)CG.

By sacrificing some tightness, we show that these bounds can all be written in the same form as the corresponding bounds for the classical method multiplied by an amplification factor that depends on the condition number of the s-step bases generated at the beginning of each outer loop. We stress that the value of these bounds is in the *insight* they give rather than their tightness. Our analyses confirm the observation that the conditioning of the Krylov bases plays a large role in determining finite precision behavior, and also

indicates that the s-step method can be made suitable for practical use in many cases, offering both speed and accuracy. This inspires the development of practical techniques for improving s-step Lanczos based on our results. In Chapter 6, we discuss potential ways of controlling the basis conditioning such that, e.g., (5.64) holds. We also discuss how our bounds could guide the use of extended or variable precision in s-step Krylov methods; that is, rather than controlling the conditioning of the computed s-step base (5.64) could be satisfied by decreasing the unit roundoff ϵ using techniques either in hardware or software. We also in Chapter 6 develop an implicit residual replacement strategy based on the maximum attainable accuracy analysis of Section 5.3. Further investigation of these techniques as well as the development of other techniques for controlling numerical behavior based on our analyses is a fruitful direction for future work.

The next step is to extend the subsequent analyses of Paige, in which a type of augmented backward stability for the classical Lanczos method is proved [142]. In [12], Bai proves a result for nonsymmetric Lanczos analogous to the results of Paige, namely, that if a Ritz value is well-conditioned, convergence implies loss of orthogonality. We conjecture that our results for CA-Lanczos could be extended in the same way to the nonsymmetric CA-BIOC method (see Section 4.1).

Our results in this chapter explain how the finite precision behavior of CA-KSMs is determined by the conditioning of the s-step bases. This raises the question of how this information can be used to design methods for improving stability and convergence properties of these methods. In the next chapter, we use the intuition and theoretical bounds developed in this chapter to develop a number of techniques for CA-KSMs that can improve numerical properties while still allowing for an asymptotic reduction in communication.

Chapter 6

Methods for Improving Stability and Convergence

In this chapter we present a number of techniques for improving stability and convergence in finite precision CA-KSMs while maintaining asymptotic communication savings. In many cases, these techniques have been developed based on the finite precision analyses of Chapter 5. Some of these techniques, including residual replacement (Section 6.1, adapted from [31]), deflation-based preconditioning (Section 6.2, adapted from [36]), selective reorthogonalization (Section 6.3), and look-ahead (Section 6.4), are techniques that have been developed for classical Krylov methods. We extend these techniques to communication-avoiding Krylov methods, and show that they can be implemented to improve numerical properties while still maintaining asymptotic communication savings. For example, our numerical experiments in Section 6.1 show that our residual replacement scheme can improve the accuracy of CA-KSMs by up to 7 orders of magnitude for little additional cost (at most a 2% increase in the number of outer loop iterations).

To highlight the potential benefits and tradeoffs of these techniques and of the communication-avoiding kernels presented in Chapter 3, Section 6.2.4 is devoted to modeling the performance of our communication-avoiding deflated CG method for various values of s and various numbers of deflation vectors.

In Section 6.5, we discuss the use of extended/variable precision for improving CA-KSMs. Although using extended or variable precision can also have benefits for classical Krylov methods, our discussion in Section 6.5 specifically focuses on how this might be done in CA-KSMs based on the finite precision analysis in Section 5.1. Other techniques presented, including the dynamic refinement of basis parameters (Section 6.6) and variable basis size (Section 6.7) are specifically applicable to CA-KSMs and do not have a classical KSM analog. As preconditioning is integral to many uses of KSMs for solving linear systems in practice, we finish in Section 6.8 with a discussion of ongoing work in the development of communication-avoiding preconditioners for CA-KSMs.

6.1 Residual Replacement

We extend our maximum attainable accuracy analysis (discussed in Section 5.3) to devise an algorithm which tracks an upper bound on the deviation of the true and computed residual in each iteration. We demonstrate that the necessary bookkeeping can be performed in a communication-avoiding way, such that the asymptotic communication and computation costs are not increased. Following the analysis of van der Vorst and Ye [181], this leads to an implicit residual replacement scheme which improves the maximum attainable accuracy of the solution produced by CA-KSMs. This analysis extends to CA-CG- and CA-BICG-like methods, and is generally applicable to any three-term recurrence polynomial basis and any linear system. We implement and test our residual replacement scheme on a variety of matrices, demonstrating its effectiveness at maintaining agreement between the true and computed residual. We note that this section has been adapted from work that first appeared in [31].

We first discuss the residual replacement strategy of van der Vorst and Ye [181], on which our residual replacement strategy is based. In [181], at *replacement steps* $m = m_1, m_2, \ldots, m_y$, the updated residual r_m is replaced with the true residual and a *group update* is performed, i.e., the approximate solution is updated in blocks as

$$x_{i+1} = x_1 + \sum_{\ell=1}^{i} q_\ell = x_1 + (q_1 + \cdots + q_{m_1}) + \cdots + \left(q_{m_{y-1}+1} + \cdots + q_{m_y}\right).$$

This strategy has previously been suggested for use in KSMs by Neumaier [133] and was also used by Sleijpen and van der Vorst; see [161, 181] for justification of this technique. To summarize, a group update strategy ensures that the deviation of residuals remains bounded by $O(\epsilon)\|A\|\|x\|$ from the last replacement step m_y to the final iteration by reducing error in the local recurrence.

We use the same strategy as [181], i.e., when a residual replacement occurs, a group update is also performed. In iteration $sk+j+m$, where m was the last residual replacement step, we denote (but do not compute) the group solution $z_{sk+j+1+m}$ as the sum of the group solution computed in step m, z_m, and the current *partial solution*, $x_{sk+j+1} = x_{sk+1} + \mathcal{Y}_k x'_{k,j+1}$, i.e., $z_{sk+j+1+m} = z_m + x_{sk+1} + \mathcal{Y}_k x'_{k,j+1}$ (see Alg. 29).

If $sk+j+m$ is a residual replacement step, we compute $\hat{z}_{sk+j+1+m}$ and set the true residual to $\mathrm{fl}(b - A\hat{z}_{sk+j+1+m})$. As subsequent iterates cannot be represented in $\hat{\mathcal{Y}}_k$, a replacement step requires starting a new outer loop iteration, potentially before completing s iterations of the inner loop. Thus when replacement occurs we reset $k = 0$ and $x_{sk+j} = x_1 = 0_{n,1}$. Although additional outer loop iterations require additional communication, our results show that the number of replacements required is small with respect to the total number of iterations, and thus extra cost is negligible.

As described in [181], we seek to determine iterations where replacing the updated residual with the true residual does not alter the rate of convergence, based on a bound on potential

perturbation to the finite precision Lanczos recurrence. We briefly review the discussion in [181], which motivates the condition for residual replacement.

6.1.1 Selecting Residual Replacement Steps

Consider finite precision classical CG, where, in iteration i, the updated residuals $\hat{r}_{i+1}$ and search directions $\hat{p}_{i+1}$ satisfy

$$\hat{r}_{i+1} = \hat{r}_i - \alpha_i A\hat{p}_i + \eta_{i+1} \qquad \text{and} \qquad \hat{p}_{i+1} = \hat{r}_{i+1} + \beta_i \hat{p}_i + \tau_{i+1}. \tag{6.1}$$

Let e_i denote the i^{th} identity column of appropriate size. We can then write the above equations in matrix form as

$$AZ_i = Z_i \bar{T}_i - \frac{1}{\bar{\alpha}_i} \frac{\hat{r}_{i+1}}{\|\hat{r}_1\|} e_i^T + F_i, \quad \text{with} \quad Z_i = \left[\frac{\hat{r}_1}{\|\hat{r}_1\|}, \ldots, \frac{\hat{r}_i}{\|\hat{r}_i\|} \right],$$

where $\bar{T}_i$ is invertible and tridiagonal, $\bar{\alpha}_i = e_i^T \bar{T}_i^{-1} e_1$ and $F_i = [f_1, \ldots, f_i]$, with

$$f_\ell = \frac{A\tau_\ell}{\|\hat{r}_\ell\|} + \frac{1}{\alpha_\ell} \frac{\eta_{\ell+1}}{\|\hat{r}_\ell\|} - \frac{\beta_{\ell-1}}{\alpha_{\ell-1}} \frac{\eta_\ell}{\|\hat{r}_\ell\|}. \tag{6.2}$$

As discussed in Section 5.2, it has been shown by Tong and Ye [176]) that if $\hat{r}_{i+1}$ satisfies (6.1) and Z_{i+1} is full rank,

$$\|\hat{r}_{i+1}\| \leq (1 + K_i) \min_{\rho \in \mathbb{P}_i, \rho(0)=1} \|\rho(A + \Delta A_i)\hat{r}_1\|, \tag{6.3}$$

where $\mathbb{P}_i$ is the set of polynomials of degree i, $K_i = \left\|(AZ_i - F_i)\bar{T}_i^{-1}\right\| \left\|Z_{i+1}^+\right\|$ and $\Delta A_i = -F_i Z_i^+$. Therefore, regardless of how $\hat{r}_{i+1}$ is generated, if it satisfies (6.1), we can bound its norm by (6.3). Then by (6.2), we know we can replace the updated residual with the true residual without affecting the convergence rate when η_{i+1} is not too large relative to $\|\hat{r}_{i+1}\|$ and $\|\hat{r}_i\|$.

We will use $\eta_{sk+j+1+m}^{\text{RR}}$ to denote the perturbation term in CA-CG with residual replacement and group update strategies, i.e.,

$$\hat{r}_{sk+j+1+m} \equiv r_{sk+j+m} - Aq_{sk+j+m} + \eta_{sk+j+1+m}^{\text{RR}},$$

where $\eta_{sk+j+1+m}^{\text{RR}}$ is analogous to the η_{i+1} term in (6.1). Our goal will be to bound the norm of $\eta_{sk+j+1+m}^{\text{RR}}$. Comparing this bound with the norms $\|r_{sk+j+1+m}\|$ and $\|r_{sk+j+m}\|$ of the updated residuals in CA-CG with residual replacement will then allow us to select safe residual replacement steps using the following criteria:

$$\left\|\eta_{sk+j+m}^{\text{RR}}\right\| \leq \hat{\epsilon}\left\|r_{sk+j+m}\right\| \text{ and } \left\|\eta_{sk+j+1+m}^{\text{RR}}\right\| > \hat{\epsilon}\left\|r_{sk+j+1+m}\right\|. \tag{6.4}$$

Above, $\hat{\epsilon}$ is a tolerance parameter. Because $\hat{\epsilon}$ controls perturbations to the Lanczos recurrence, it should be chosen as small as possible. However, if it is too small, residual

replacement will terminate early in the iteration and the accumulated error after the last replacement can become significant [181]. The value $\hat{\epsilon} = \sqrt{\epsilon}$ has been found to balance these two constraints for standard KSMs [181].

We stress that here we have only experimented with using the same residual replacement condition used for classical CG, which is based on the finite precision Lanczos recurrence given above. Our analysis in Section 5.2 gives a finite precision CA-Lanczos recurrence analogous to that of classical Lanczos above. Equations for the columns of F_n and bounds on their size for CA-(BI)CG given in (5.157), (5.158), (5.163), and (5.164) suggest that if the condition number of the s-step basis matrix for the current outer loop, e.g., $\mathcal{Y}_k$, is very large, then this will cause large perturbations to the finite precision recurrence. This implies that one could replace the updated residual with the true residual often without affecting convergence, but only because the updated residual is likely failing to converge due to an ill-conditioned basis. In this case, when the updated residual is not converging, it is not worth our time to perform residual replacement; improving agreement between the true and updated residuals buys us nothing in terms of an accurate solution. The best we can do then, is to hope that the s-step bases are well-conditioned (i.e., that Γ_k as defined in Section 5 is small) and in this case, the replacement condition for the classical methods should be sufficient for the communication-avoiding methods.

6.1.2 Bounding the Error Term

We will now describe and analyze the error in CA-CG with residual replacement and group update strategies (see Alg. 29). Note that in this analysis, we index iterations as $sk + j + m$ where m is the last residual replacement iteration and $sk + j$ is defined as before, where we reset $k = 0$ immediately after a replacement step. The goal of this section is a bound on the perturbation term $\eta^{\text{RR}}_{sk+j+1+m}$ in the recurrence for the updated residual in CA-CG with residual replacement and group update,

$$\hat{r}_{sk+j+1+m} = r_{sk+j+m} - Aq_{sk+j+m} + \eta^{\text{RR}}_{sk+j+1+m}.$$

This will enable us to determine when we can replace the updated residual with the true residual without affecting convergence. Our bounds here are based on the bounds obtained in Section 5.3. In the present section, we use the quantities E_k, ϕ_{sk+j+1}, ψ_{sk+j+1}, $\xi_{k,j+1}$ and $\eta_{k,j+1}$ as defined in (5.183), (5.192), (5.194), (5.186), and (5.188) in Section 5.3, respectively. We denote the current group solution

$$z_{sk+j+1+m} = \begin{cases} \hat{z}_{sk+j+1+m} = \text{fl}\left(\hat{z}_m + \hat{x}_{sk+j+1}\right) = \hat{z}_m + \hat{x}_{sk+j+1} + \zeta_{sk+j+1+m}, & \text{RR step}, \\ \hat{z}_m + \hat{x}_{sk+j+1} = \hat{z}_m + x_{sk+j+1} + \phi_{sk+j+1}, & j = s, \\ \hat{z}_m + x_{sk+j+1} = \hat{z}_m + \hat{x}_{sk+1} + \hat{\mathcal{Y}}_k \hat{x}'_{k,j+1}, & 1 \leq j < s \end{cases} \tag{6.5}$$

where $\hat{z}_m$ is the approximate solution computed in the last group update in step m, $x_{sk+j+1} = \hat{x}_{sk+1} + \hat{\mathcal{Y}}_k \hat{x}'_{k,j+1}$, $\hat{x}_{sk+j+1} = x_{sk+j+1} + \phi_{sk+j+1}$, and

$$\left|\zeta_{sk+j+1+m}\right| \leq \epsilon \left|\hat{z}_{sk+j+1+m}\right| \leq \epsilon \Big(\left|\hat{z}_m\right| + \left|\hat{x}_{sk+1}\right| + \left|\hat{\mathcal{Y}}_k \hat{x}'_{k,j+1}\right| \Big). \tag{6.6}$$

We use $r_{sk+j+1+m}$ to denote the residual in CA-CG with residual replacement and group update, defined as

$$r_{sk+j+1+m} = \begin{cases} \hat{r}_{sk+j+1+m} = \mathrm{fl}\left(b - A\hat{z}_{sk+j+1+m}\right) = b - A\hat{z}_{sk+j+1+m} + \mu_{sk+j+1+m}, & \text{RR step,} \\ \hat{r}_{sk+j+1+m} = \mathrm{fl}(\hat{\mathcal{Y}}_k \hat{r}'_{k,j+1}) = \hat{\mathcal{Y}}_k \hat{r}'_{k,j+1} + \psi_{sk+j+1}, & j = s, \\ r_{sk+j+1+m} = \hat{\mathcal{Y}}_k \hat{r}'_{k,j+1}, & 1 \leq j < s, \end{cases} \tag{6.7}$$

where

$$\left|\mu_{sk+j+1+m}\right| \leq \epsilon \Big(\left|\hat{r}_{sk+j+1+m}\right| + N_A \left|A\right| \left|\hat{z}_{sk+j+1+m}\right| \Big). \tag{6.8}$$

After the last replacement step m, but before the next replacement step, we denote

$$\delta_{sk+j+1+m} = b - Az_{sk+j+1+m} - r_{sk+j+1+m}.$$

If $\hat{r}_{m+1}$ is to be computed by replacement, we compute $\hat{r}_{m+1} = \mathrm{fl}\left(b - A\hat{z}_{m+1}\right) = b - A\hat{z}_{m+1} + \mu_{m+1}$. Since $r_{m+1} = \hat{\mathcal{Y}}_0 \hat{r}'_{0,1} = \hat{r}_{m+1}$ and $z_{m+1} = \hat{z}_{m+1} + \hat{x}_1 + \underline{\hat{V}}_0 \hat{x}'_{0,1} = \hat{z}_{m+1}$, $\delta_{m+1} = b - Az_{m+1} - r_{m+1} = b - A\hat{z}_{m+1} - \hat{r}_{m+1} = -\mu_{m+1}$. We now state the main result of this section.

Theorem 10. *Consider finite precision CA-CG with residual replacement and group update (shown in Algorithm 29), where iteration $sk + j + m$ is a replacement step and m was the last replacement step. Let the quantities E_k, ϕ_{sk+j+1}, ψ_{sk+j+1}, $\xi_{k,j+1}$ and $\eta_{k,j+1}$ be defined as in* (5.183), (5.192), (5.194), (5.186), *and* (5.188) *in Section 5.3, respectively. The recurrence for the updated residual satisfies*

$$\hat{r}_{sk+j+1+m} = r_{sk+j+m} - Aq_{sk+j+m} + \eta^{RR}_{sk+j+1+m}$$

where

$$\eta^{RR}_{sk+j+1+m} = -E_k \hat{q}'_{k,j} + \hat{\mathcal{Y}}_k \eta_{k,j+1} + \delta_{sk+j+1+m} - A\phi_{sk+j+1} - A\zeta_{sk+j+1+m} + \mu_{sk+j+1+m}$$

and

$$\begin{aligned}\left\|\eta^{RR}_{sk+j+1+m}\right\| \leq & \epsilon\Big(\left\|\hat{r}_{m+1}\right\| + \left(1+2N'\right)\|A\|\left\|\hat{z}_{m+1}\right\|\Big) \\ & + \epsilon\sum_{\ell=0}^{k-1}\left(\|A\|\left\|\hat{x}_{s\ell+s+1}\right\| + C_1\|A\|\left\||\hat{\mathcal{Y}}_\ell||\hat{x}'_{\ell,s+1}|\right\| + N'\left\||\hat{\mathcal{Y}}_\ell||\hat{r}'_{\ell,s+1}|\right\|\right) \\ & + \epsilon\sum_{\ell=0}^{k-1}\sum_{i=1}^{s}\left(C_2\|A\|\left\||\hat{\mathcal{Y}}_\ell||\hat{x}'_{\ell,i+1}|\right\| + C_3\left\||\hat{\mathcal{Y}}_\ell||\mathcal{B}_\ell||\hat{x}'_{\ell,i+1}|\right\| + \left\||\hat{\mathcal{Y}}_\ell||\hat{r}'_{\ell,i+1}|\right\|\right) \\ & + \epsilon\sum_{i=1}^{j}\left(C_2\|A\|\left\||\hat{\mathcal{Y}}_k||\hat{x}'_{k,i+1}|\right\| + C_3\left\||\hat{\mathcal{Y}}_k||\mathcal{B}_k||\hat{x}'_{k,i+1}|\right\| + \left\||\hat{\mathcal{Y}}_k||\hat{r}'_{k,i+1}|\right\|\right),\end{aligned} \tag{6.9}$$

with $C_1 = 2+2N'$, $C_2 = 7+2N'$, *and* $C_3 = 8+2N'$, *where* $N' = \max(N_A, 2s+1)$.

Proof. If iteration $sk+j+m$ is a replacement step, we compute

$$\begin{aligned}\hat{z}_{sk+j+1+m} &= \mathrm{fl}(\hat{z}_{m+1} + \hat{x}_{sk+j+1}) = z_{sk+j+1+m} + \phi_{sk+j+1} + \zeta_{sk+j+1+m} \\ \hat{r}_{sk+j+1+m} &= \mathrm{fl}(b - A\hat{z}_{sk+j+1+m}) = b - A\hat{z}_{sk+j+1+m} + \mu_{sk+j+1+m} \\ &= b - Az_{sk+j+1+m} - A\phi_{sk+j+1} - A\zeta_{sk+j+1+m} + \mu_{sk+j+1+m} \\ &= r_{sk+j+1+m} + \delta_{sk+j+1+m} - A\phi_{sk+j+1} - A\zeta_{sk+j+1+m} + \mu_{sk+j+1+m} \\ &= \hat{\mathcal{Y}}_k\hat{r}'_{k,j} - \hat{\mathcal{Y}}_k\mathcal{B}_k\hat{q}'_{k,j} + \hat{\mathcal{Y}}_k\eta_{k,j+1} + \delta_{sk+j+1+m} - A\phi_{sk+j+1} - A\zeta_{sk+j+1+m} + \mu_{sk+j+1+m} \\ &= r_{sk+j+m} - Aq_{sk+j+m} + \eta^{\mathrm{RR}}_{sk+j+1+m}\end{aligned}$$

with

$$\eta^{\mathrm{RR}}_{sk+j+1+m} = -E_k\hat{q}'_{k,j} + \hat{\mathcal{Y}}_k\eta_{k,j+1} + \delta_{sk+j+1+m} - A\phi_{sk+j+1} - A\zeta_{sk+j+1+m} + \mu_{sk+j+1+m}. \tag{6.10}$$

Following the derivation of (5.196), we can write $\delta_{sk+j+1+m}$ as

$$\begin{aligned}\delta_{sk+j+1+m} = & -\mu_{m+1} - \sum_{\ell=0}^{k-1}\left(A\phi_{s\ell+s+1} + \psi_{s\ell+s+1} + \sum_{i=1}^{s}\left(A\hat{\mathcal{Y}}_\ell\xi_{\ell,i+1} + \hat{\mathcal{Y}}_\ell\eta_{\ell,i+1} - E_\ell\hat{q}'_{\ell,i}\right)\right) \\ & - \sum_{i=1}^{j}\left(A\hat{\mathcal{Y}}_k\xi_{k,i+1} + \hat{\mathcal{Y}}_k\eta_{k,i+1} - E_k\hat{q}'_{k,i}\right),\end{aligned} \tag{6.11}$$

and then

$$\begin{aligned}&\delta_{sk+j+1+m} - E_k\hat{q}'_{k,j} + \hat{\mathcal{Y}}_k\eta_{k,j+1} \\ &\quad = -\mu_{m+1} - \sum_{\ell=0}^{k-1}\left(A\phi_{s\ell+s+1} + \psi_{s\ell+s+1} + \sum_{i=1}^{s}\left(A\hat{\mathcal{Y}}_\ell\xi_{\ell,i+1} + \hat{\mathcal{Y}}_\ell\eta_{\ell,i+1} - E_\ell\hat{q}'_{\ell,i}\right)\right) \\ &\quad\quad - \sum_{i=1}^{j}A\hat{\mathcal{Y}}_k\xi_{k,i+1} - \sum_{i=1}^{j-1}\left(\hat{\mathcal{Y}}_k\eta_{k,i+1} - E_k\hat{q}'_{k,i}\right).\end{aligned} \tag{6.12}$$

Substituting (6.12) into (6.10), and using bounds in (5.183), (5.187), (5.189), (5.193), (5.195), and (6.8), as well as $\left|\hat{x}_{sk+1}\right| \leq \sum_{\ell=0}^{k-1} \left|\hat{\mathcal{Y}}_\ell\right|\left|\hat{x}'_{\ell,s+1}\right| + O(\epsilon)$, we obtain the following componentwise bound:

$$\begin{aligned}\left|\eta^{\mathrm{RR}}_{sk+j+1+m}\right| \leq & \epsilon\big(\left|\hat{r}_{m+1}\right| + (1+2N')\left|A\right|\left|\hat{z}_{m+1}\right|\big) \\ & + \epsilon\sum_{\ell=0}^{k-1}\left(\left|A\right|\left|\hat{x}_{s\ell+s+1}\right| + (2+2N')\left|A\right|\left|\hat{\mathcal{Y}}_\ell\right|\left|\hat{x}'_{\ell,s+1}\right| + N'\left|\hat{\mathcal{Y}}_\ell\right|\left|\hat{r}'_{\ell,s+1}\right|\right) \\ & + \epsilon\sum_{\ell=0}^{k-1}\sum_{i=1}^{s}\left((7+2N')\left|A\right|\left|\hat{\mathcal{Y}}_\ell\right|\left|\hat{x}'_{\ell,i+1}\right| + \left|\hat{\mathcal{Y}}_\ell\right|\Big((8+2N')\left|\mathcal{B}_\ell\right|\left|\hat{x}'_{\ell,i+1}\right| + \left|\hat{r}'_{\ell,i+1}\right|\Big)\right) \\ & + \epsilon\sum_{i=1}^{j}\left((7+2N')\left|A\right|\left|\hat{\mathcal{Y}}_k\right|\left|\hat{x}'_{k,i+1}\right| + \left|\hat{\mathcal{Y}}_k\right|\Big((8+2N')\left|\mathcal{B}_k\right|\left|\hat{x}'_{k,i+1}\right| + \left|\hat{r}'_{k,i+1}\right|\Big)\right)\end{aligned}$$

with $N' = \max(N_A, 2s+1)$. The desired bound on the norm of $\eta^{\mathrm{RR}}_{sk+j+1+m}$ follows. □

We can compute (6.9) in each iteration, and thus this bound can be used in the residual replacement condition (6.4). The following section discusses computation and communication costs associated with computing an estimate for (6.9).

6.1.3 Residual Replacement Algorithm for CA-CG

In each iteration, we will update an estimate for $\left\|\eta^{\mathrm{RR}}_{sk+j+m}\right\|$, which we denote d_{sk+j+m}. Based on (6.9), we will iteratively update the estimate $d_{sk+j+1+m}$ by

$$\begin{aligned}d_{sk+j+1+m} = & \, d_{sk+j+m} \\ & + \epsilon\Big((4+N')\Big(\|A\|\Big\|\left|\hat{\mathcal{Y}}_k\right|\left|\hat{x}'_{k,j+1}\right|\Big\| + \Big\|\left|\hat{\mathcal{Y}}_k\right|\left|\mathcal{B}_k\right|\left|\hat{x}'_{k,j+1}\right|\Big\|\Big) + \Big\|\left|\hat{\mathcal{V}}_k\right|\left|\hat{r}'_{k,j+1}\right|\Big\|\Big) \qquad (6.13) \\ & + \epsilon\begin{cases}\|A\|\Big(\|\hat{x}_{sk+s+1}\| + (2+2N')\Big\|\left|\hat{\mathcal{Y}}_k\right|\left|\hat{x}'_{k,s+1}\right|\Big\|\Big) + N'\Big\|\left|\hat{\mathcal{Y}}_k\right|\left|\hat{r}'_{k,s+1}\right|\Big\|, & j = s \\ 0, & \text{o.w.}\end{cases} \qquad (6.14)\end{aligned}$$

where we omit the factors of two due to the bound $\sum_{i=1}^{j}\left|\hat{q}'_{k,i}\right| \leq (2+\epsilon)\sum_{i=1}^{j}\left|\hat{x}'_{k,i+1}\right|$, which is pessimistic. At each replacement step m, the value of d_{m+1} is reset to $d_{m+1} = \epsilon\left(\|\hat{r}_{m+1}\| + (1+2N')\|A\|\,\|\hat{z}_{m+1}\|\right)$.

Based on (6.4), and using (6.7), we perform a residual replacement and group update step when

$$d_{sk+j+m} \leq \hat{\epsilon}\left\|\hat{\mathcal{Y}}_k\hat{r}'_{k,j}\right\|_2 \text{ and } d_{sk+j+1+m} > \hat{\epsilon}\left\|\hat{\mathcal{Y}}_k\hat{r}'_{k,j+1}\right\|_2 \text{ and } d_{sk+j+1+m} > 1.1d_{\mathrm{init}}, \qquad (6.15)$$

where the third condition is recommended in [181] to ensure that the error has nontrivially increased since the last replacement step in an effort to avoid unnecessary replacements. If this statement is true, we set $z_{sk+j+1+m} = \hat{z}_{sk+j+1+m} = \mathrm{fl}(\hat{z}_{m+1} + \hat{x}_{sk+j+1})$, and we set $r_{sk+j+1+m} = \hat{r}_{sk+j+1+m} = \mathrm{fl}(b - A\hat{z}_{sk+j+1+m})$, as in (6.5) and (6.7).

Note that we use $\left\|\hat{\mathcal{Y}}_k \hat{r}'_{k,j}\right\|_2 = (\hat{r}'^T_{k,j} G_k \hat{r}'_{k,j})^{1/2} + O(\epsilon)$ to estimate the other side of the inequalities in (6.15). Since these terms are multiplied by $\hat{\epsilon} = \epsilon^{1/2}$ in (6.15), the $O(\epsilon)$ error due to use of G_k is ignored.

A residual replacement step does incur additional costs in CA-CG. When a residual replacement occurs, we must break from the inner loop (perhaps before completing s steps) and compute $\hat{z}_{sk+j+m}$ and $\hat{r}_{sk+j+m} = \text{fl}(b - A\hat{z}_{sk+j+m})$ (the communication cost depends on the data layout/structure of A). The algorithm must then begin a new outer loop, generating new s-step bases with respect to the replaced residual.

If the number of replacements is large (i.e., we compute the true residual every iteration), this can result in a significant slowdown due to increased communication costs. Fortunately, our experimental results in Section 6.1.5 demonstrate that the number of replacements performed can be low compared to the total number of iterations; for all of our test cases, at most 2% of the total number of iterations were replacement steps. The CA-CG algorithm with residual replacement is shown in Alg. 29.

6.1.4 Avoiding Communication in Residual Replacement

To iteratively update (6.13) and (6.14) we must be able to inexpensively estimate the quantities $\left\||\hat{\mathcal{Y}}_k||\mathcal{B}_k||\hat{x}'_{k,j+1}|\right\|$, $\left\||\hat{\mathcal{Y}}_k||\hat{x}'_{k,j+1}|\right\|$, $\left\||\hat{\mathcal{Y}}_k||\hat{r}'_{k,j+1}|\right\|$, $\|A\|$, $\|\hat{r}_{m+1}\|$, $\|\hat{z}_{m+1}\|$, and $\|\hat{x}_{sk+s+1}\|$ in each inner iteration of CA-CG. We assume we have an estimate for $\|A\|$, which need only be computed once. We discuss how to obtain the remaining quantities without increasing the asymptotic computation or communication cost of CA-CG.

Let $\tilde{G}_k$ be the $(2s+1) \times (2s+1)$ matrix $\left|\hat{\mathcal{Y}}_k\right|^T \left|\hat{\mathcal{Y}}_k\right|$. Given $\tilde{G}_k$, we can compute

$$\begin{aligned}
\left\|\left|\hat{\mathcal{Y}}_k\right|\left|\hat{x}'_{k,j+1}\right|\right\|_2 &= \sqrt{\left|\hat{x}'_{k,j+1}\right|^T \tilde{G}_k \left|\hat{x}'_{k,j+1}\right|} + O(\epsilon), \\
\left\|\left|\hat{\mathcal{Y}}_k\right|\left|\hat{r}'_{k,j+1}\right|\right\|_2 &= \sqrt{\left|\hat{r}'_{k,j+1}\right|^T \tilde{G}_k \left|\hat{r}'_{k,j+1}\right|} + O(\epsilon), \text{ and} \\
\left\|\left|\hat{\mathcal{Y}}_k\right|\left|\mathcal{B}_k\right|\left|\hat{x}'_{k,j+1}\right|\right\|_2 &= \sqrt{\left|\hat{x}'_{k,j+1}\right|^T \left|\mathcal{B}_k\right|^T \tilde{G}_k \left|\mathcal{B}_k\right|\left|\hat{x}'_{k,j+1}\right|} + O(\epsilon).
\end{aligned} \tag{6.16}$$

In each outer loop, the computation of $\tilde{G}_k$ requires a single synchronization (summing a list of $O(s) \times O(s)$ matrices). As we perform an equivalent reduction to compute G_k in each outer loop of CA-CG, computing $\tilde{G}_k$ increases the communication and computation costs by no worse than a factor of two. If the reduction to obtain $\tilde{G}_k$ can be done simultaneously with that to obtain G_k, no additional latency is incurred.

The computation of quantities in (6.16) require $O(s^3)$ operations per s steps and no communication. Again, we perform equivalent operations to compute α_{sk+j} and β_{sk+j} in each inner loop of CA-CG, so the computation cost is increased by a constant factor. Note that, until now, we have not assumed a specific norm. The 2-norm allows us to compute inner products locally using $\tilde{G}_k$, and thus maintains the communication-avoiding properties of CA-CG.

Algorithm 29 CA-CG with residual replacement (CA-CG-RR)

Input: $n \times n$ symmetric positive definite matrix A, length-n vector b, and initial approximation x_1 to $Ax = b$

Output: Approximate solution x_{sk+s+1} to $Ax = b$ with updated residual r_{sk+s+1}

$z_1 = x_1$, $x_1 = 0_{n,1}$, $r_1 = p_1 = b - Az_1$, $d_1 = d_{\text{init}} = \epsilon(\|r_1\|_2 + N' \|A\|_2 \|z_1\|_2)$, $m = 0$, RR $= 0$
for $k = 0, 1, \ldots,$ until convergence **do**
  Calculate $\mathcal{Y}_k = [\mathcal{P}_k, \mathcal{R}_k]$ with starting vectors p_{sk+1+m} and r_{sk+1+m}, $G_k = \mathcal{Y}_k^T \mathcal{Y}_k$
  $p'_{k,1} = [1,\ 0_{1,2s}]^T$, $r'_{k,1} = [0_{1,s+1},\ 1,\ 0_{1,s-1}]^T$, $x'_{k,1} = [0_{1,2s+1}]^T$
  for $j \in \{1, \ldots, s\}$ **do**
    $\alpha_{sk+j} = \left(r'^T_{k,j} G_k r'_{k,j}\right) / \left(p'^T_{k,j} G_k \mathcal{B}_k p'_{k,j}\right)$
    $q'_{k,j} = \alpha_{sk+j} p'_{k,j}$
    $x'_{k,j+1} = x'_{k,j} + q'_{k,j}$
    $r'_{k,j+1} = r'_{k,j} - \mathcal{B}_k q'_{k,j}$
    $\beta_{sk+j} = \left(r'^T_{k,j+1} G_k r'_{k,j+1}\right) / \left(r'^T_{k,j} G_k r'_{k,j}\right)$
    $p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j} p'_{k,j}$
    Update $d_{sk+j+1+m}$ by (6.13).
    if condition (6.15) holds **then**
      $z_{sk+j+1+m} = z_{m+1} + x_{sk+1} + \mathcal{Y}_k x'_{k,j+1}$, $r_{sk+j+1+m} = b - Az_{sk+j+1+m}$, $x_1 = 0_{n,1}$
      $d_{\text{init}} = d_{sk+j+1+m} = \epsilon\left(\|r_{sk+j+1+m}\|_2 + (1 + 2N') \|A\|_2 \|z_{sk+j+1+m}\|_2\right)$
      RR $= 1$, $m = m + sk + j$, $k = 0$, **break**
    end if
  end for
  if RR $= 0$ **then**
    $x_{sk+s+1} = \mathcal{Y}_k x'_{k,s+1} + x_{sk+1}$, $r_{sk+s+1+m} = \mathcal{Y}_k r'_{k,s+1}$
    Update $d_{sk+s+1+m}$ by (6.14).
  end if
  $p_{sk+j+1+m} = \mathcal{Y}_k p'_{k,s+1}$, RR $= 0$
end for
return $z_{sk+j+1+m} = z_m + x_{sk+1} + \mathcal{Y}_k x'_{k,j+1}$, $r_{sk+j+1+m} = \mathcal{Y}_k r'_{k,j+1}$

Matrix	Domain	n	nnz	cond	2-norm	SPD?
cdde [13, 35]	comp. fluid dynamics	$2.6 \cdot 10^5$	$1.3 \cdot 10^6$	5.5	6.0	N
consph [57]	FEM/Spheres	$8.3 \cdot 10^4$	$6.0 \cdot 10^6$	$9.7 \cdot 10^3$	9.7	Y
thermal1 [57]	thermal	$8.3 \cdot 10^4$	$5.7 \cdot 10^5$	$3.0 \cdot 10^5$	1.9	Y
xenon1 [57]	materials	$4.9 \cdot 10^4$	$1.2 \cdot 10^6$	$3.3 \cdot 10^4$	3.2	N
G2circuit [57]	circuit simulation	$1.5 \cdot 10^5$	$7.3 \cdot 10^5$	$2.3 \cdot 10^5$	2.0	Y

Table 6.1: Matrices used in residual replacement tests. Norm and condition numbers reflect the equilibrated system. Note that cdde and xenon1 are nonsymmetric.

We must also compute $\|\hat{z}_{m+1}\|_2$ and $\|\hat{r}_{m+1}\|_2$ at every residual replacement step m. The quantity $\|\hat{r}_{m+1}\|_2$ can be computed in the next outer loop after replacement in the same way as the computation of α_{sk+j} in CA-CG, i.e., $\|\hat{r}_{m+1}\|_2 = (\hat{r}'^T_{0,1} G_0 \hat{r}'_{0,1})^{1/2} + O(\epsilon)$. The computation of $\|\hat{z}_{m+1}\|_2$ can be accomplished similarly by fusing the reduction with that of G_0 in the outer loop immediately following a residual replacement step. Again, this increases the bandwidth by a constant factor and does not increase the latency cost. Since

$$\|\hat{x}_{sk+s+1}\|_2 \leq \sum_{\ell=0}^{k} \left\|\hat{\mathcal{V}}_\ell \hat{x}'_{\ell,s+1}\right\|_2 + O(\epsilon) = \sum_{\ell=0}^{k} \sqrt{\hat{x}'^T_{\ell,s+1} G_\ell \hat{x}'_{\ell,s+1}} + O(\epsilon),$$

an estimate for this quantity can be iteratively updated in each outer loop iteration using only local quantities.

6.1.5 Numerical Experiments

We evaluated our residual replacement strategy using many matrices from the University of Florida Sparse Matrix Collection [57] and the NEP Collection [13], including the 28 matrices tested by van der Vorst and Ye [181]. We present a subset of our results, from various problem domains, which are representative of the general behavior observed. We chose to show results for matrices with large dimension n as this is where we expect CA-KSMs to be beneficial in terms of performance. The selected matrices along with relevant properties are listed in Table 6.1. We tested both symmetric positive definite (SPD) and nonsymmetric matrices, using CA-CG and CA-BICG, respectively. All matrices except cdde have real eigenvalues.

In our experiments, we compare classical (BI)CG with CA-(BI)CG, both with and without residual replacement. For CA-(BI)CG, tests were run for $s = [4, 8, 12]$, with the monomial, Newton, and Chebyshev bases (see Section 3.2.5). Coefficients for the Newton and Chebyshev bases were computed using Leja-ordered points obtained from $O(s)$ Ritz value estimates, as described in Section 3.2.5 (see also [35, 102, 149]). We used row and column scaling to equilibrate the input matrix A, preserving symmetry for the SPD case, as described in [102]. For each matrix, we selected a right hand side b such that $\|x\|_2 = 1$, $x_i = 1/\sqrt{n}$, and used the zero vector as the initial guess. All experiments were performed in double precision, i.e., $\epsilon \approx 10^{-16}$. In each test, the convergence criterion used was

		cdde		consph		thermal1		xenon1		G2circuit	
		no RR	RR	no RR	RR	no RR	RR	no RR	RR	no RR	RR
Classical		8e-15	6e-17	1e-14	1e-16	9e-14	1e-16	3e-14	9e-17	4e-14	1e-16
Monomial	4	1e-14	6e-17	1e-14	1e-16	5e-14	1e-16	2e-14	8e-17	3e-14	1e-16
	8	3e-13	7e-17	4e-12	1e-16	5e-12	1e-16	4e-12	1e-16	4e-12	1e-16
	12	9e-13	8e-17	–	–	–	–	2e-10	1e-16	3e-9	1e-16
Newton	4	2e-14	6e-17	1e-14	1e-16	5e-14	1e-16	1e-14	1e-16	2e-14	1e-16
	8	3e-14	6e-17	4e-14	1e-16	5e-14	1e-16	1e-14	1e-16	2e-14	1e-16
	12	3e-14	6e-17	6e-13	1e-16	9e-14	1e-16	4e-14	1e-16	2e-14	1e-16
Chebyshev	4	5e-15	1e-16	6e-15	1e-16	5e-14	1e-16	1e-14	1e-16	2e-14	1e-16
	8	1e-14	6e-17	3e-14	1e-16	4e-14	1e-16	1e-14	1e-16	2e-14	1e-16
	12	6e-15	6e-17	1e-13	1e-16	4e-14	1e-16	2e-14	1e-16	1e-14	1e-16

Table 6.2: Improvement in true residual 2-norm. For each test, the left column gives the 2-norm of the true residual without replacement (no RR) and the right gives the 2-norm of the true residual with replacement (RR). Dashes (-) indicate tests where the updated residual did not converge.

		cdde	consph	thermal1	xenon1	G2circuit
Classical		1.9 (1/54)	.09 (2/2164)	.09 (2/2111)	.09 (2/2226)	.07 (2/2707)
Monomial	4	1.9 (1/54)	.09 (2/2173)	.05 (1/2206)	.05 (1/2202)	.09 (2/2930)
	8	1.7 (1/58)	.52 (12/2319)	.17 (5/2929)	.20 (5/2446)	.12 (6/3779)
	12	.32 (1/310)	–	–	.03 (2/5859)	.02 (3/15901)
Newton	4	1.9 (1/54)	.09 (2/2176)	.04 (1/2226)	.05 (1/2194)	.07 (2/2930)
	8	1.9 (1/54)	.09 (2/2192)	.09 (2/2345)	.09 (2/2196)	.07 (2/2928)
	12	1.9 (1/54)	.17 (4/2296)	.08 (2/2477)	.09 (2/2299)	.07 (2/2930)
Chebyshev	4	1.9 (1/54)	.09 (2/2169)	.05 (1/2206)	.05 (1/2194)	.03 (1/2937)
	8	1.9 (1/54)	.09 (2/2179)	.05 (1/2215)	.05 (1/2194)	.07 (2/2928)
	12	1.9 (1/54)	.13 (3/2294)	.08 (2/2401)	.09 (2/2195)	.07 (2/2930)

Table 6.3: Percentage of residual replacement steps. For each test, the first number is the percentage of total iterations that were replacement steps, calculated by the fraction in parentheses, in which the numerator gives the number of replacement steps and the denominator gives total iterations. Dashes (-) indicate tests where the updated residual did not converge.

$\|r_{sk+j+1+m}\| / \|r_1\| \leq 10^{-16}$. Since $r_1 = b$ and $\|b\| \leq \|A\| \|x\|$, $\|r_{sk+j+1+m}\| / \|r_1\| \leq 10^{-16}$ implies $\|r_{sk+j+1+m}\| = O(\epsilon) \|A\| \|x\|$.

Van der Vorst and Ye [181] use $\hat{\epsilon} = 10^{-8} \approx \sqrt{\epsilon}$ and $N_A = 1$ in their experiments (with the assumption that A is very sparse). We use $\hat{\epsilon} = 10^{-8}$ for most experiments; for tests using the monomial basis with $s = 12$, we observed that using $\hat{\epsilon} = 10^{-8}$ caused restarts to occur earlier and more frequently than was desired, as the bound $\eta^{\text{RR}}_{sk+j+1+m}$ can be quite pessimistic due to the growth of $\left|\hat{\mathcal{Y}}_k\right|$ for large s values. To account for this potential overestimate, we used the value $\hat{\epsilon} = O(10^{-7})$ for all tests with the monomial basis and $s = 12$. Similarly to [181], we set $N' = 1$. While this choice of parameters produces desirable results here, these heuristics

		cdde	consph	thermal1	xenon1	G2circuit
Classical		15	375, 748	621, 1156	635, 1125	592, 998
Monomial	4	13	366, 720	642	651	591, 956
	8	10	41, 96, 170, 245, 322, 394, 482, 554, 633, 689, 722, 740	272, 494, 665, 998, 1100	215, 456, 688, 872, 1003	100, 543, 737, 970, 1279, 1477
	12	12	–	–	13, 1668	17, 33, 4298
Newton	4	13	374, 716	641	673	600, 957
	8	13	261, 602	582, 1098	590, 999	584, 949
	12	11	140, 375, 574, 717	548, 1088	520, 939	555, 919
Chebyshev	4	15	397, 727	657	701	614
	8	13	311, 651	637	658	602, 959
	12	13	193, 426, 653	576, 1098	568, 976	582, 954

Table 6.4: Iterations where residual replacement steps occur. Dashes (-) indicate tests where the updated residual did not converge.

deserve a rigorous theoretical analysis; we leave this as future work.

Although our bounds hold regardless of the bases used in constructing $\hat{\mathcal{Y}}_k$, the quality of the computed basis does affect the convergence rate of the CA-KSM; see Chapter 5. In the extreme case of a degenerate basis, the Lanczos process can break down, causing divergence of the updated residual. In such cases, any effort to maintain agreement between the true and updated residual is futile. We therefore only test the residual replacement strategy on cases where the updated residual converges.

Figs. 6.1, 6.2, 6.3, 6.4, and 6.5 show convergence of the true and updated residuals for CA-(BI)CG with and without residual replacement, for each polynomial basis. Plots in the left column show CA-(BI)CG, and plots on the right show CA-(BI)CG with the residual replacement scheme. Plots in the top, middle, and bottom rows show tests where CA-(BI)CG was run with $s = 4, 8,$ and 12, resp. For comparison, we plot classical (BI)CG and classical BI(CG) with the replacement scheme in [181].

For all plots, the x-axis is iteration number and the y-axis is the 2-norm of the quantities listed in the legend, where 'true' is the true residual, 'upd' is the updated residual, and 'd' is $d_{sk+j+1+m}/\hat{\epsilon}$, 'M','N', and 'C' denote tests with monomial, Newton, and Chebyshev bases, resp., and 'CA-(BI)CG-RR' denotes CA-(BI)CG with residual replacement.

For all tests with residual replacement (right columns), we plot the computed value of $d_{sk+j+1+m}/\hat{\epsilon}$, denoted with dash-dotted lines, where $d_{sk+j+1+m}$ is given by (6.13) and (6.14). We can see that residual replacements occur immediately after this quantity grows larger than the residual norm, as per the replacement criterion in (6.15).

Plots without residual replacement (left column) show that, as our bounds indicate, the attainable accuracy in CA-(BI)CG generally grows worse with increasing s, especially using bases where $\left|\hat{\mathcal{Y}}_k\right|$ grows quickly with s. It is worth mentioning that without residual replacement, a well-conditioned polynomial basis can allow the CA-KSM to achieve slightly better accuracy than the classical method in some cases (see, e.g., the Newton and Chebyshev

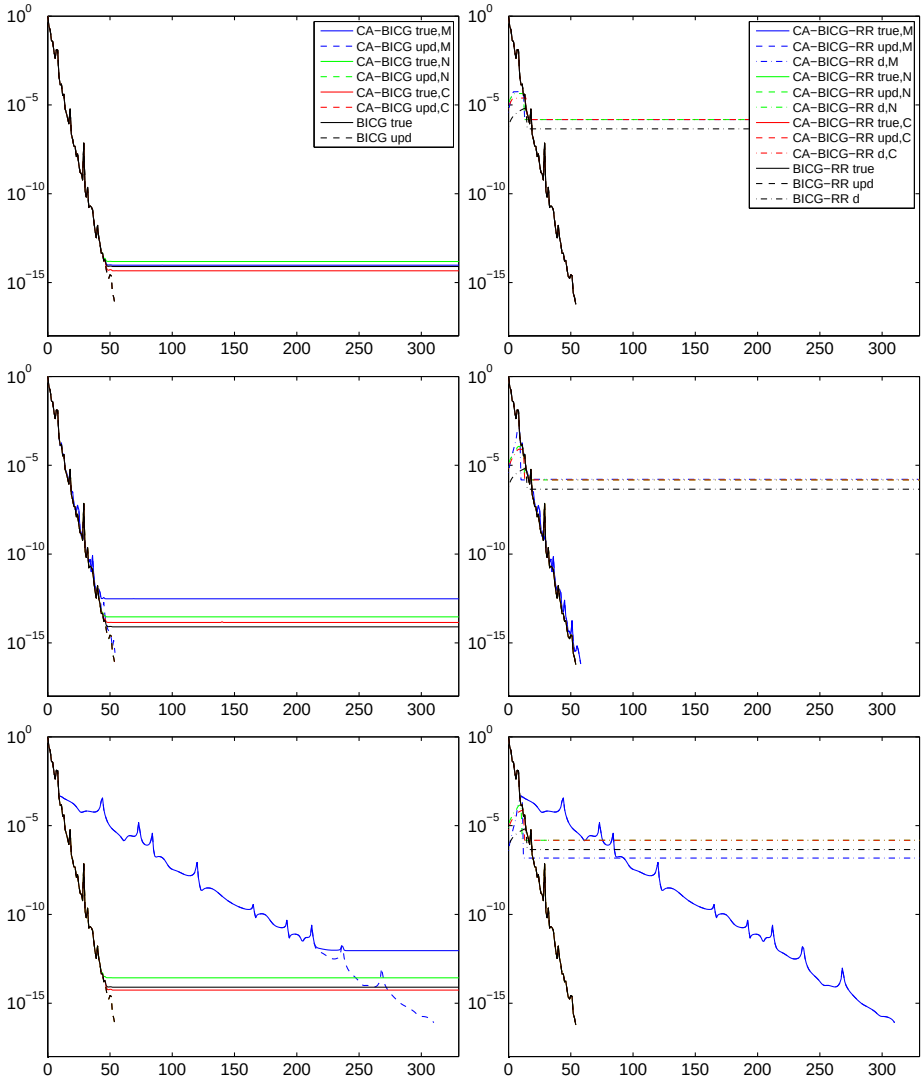

Figure 6.1: Convergence of cdde test matrix for CA-BICG without residual replacement (left) and with residual replacement (right), for $s = 4$ (top), $s = 8$ (middle), and $s = 12$ (bottom).

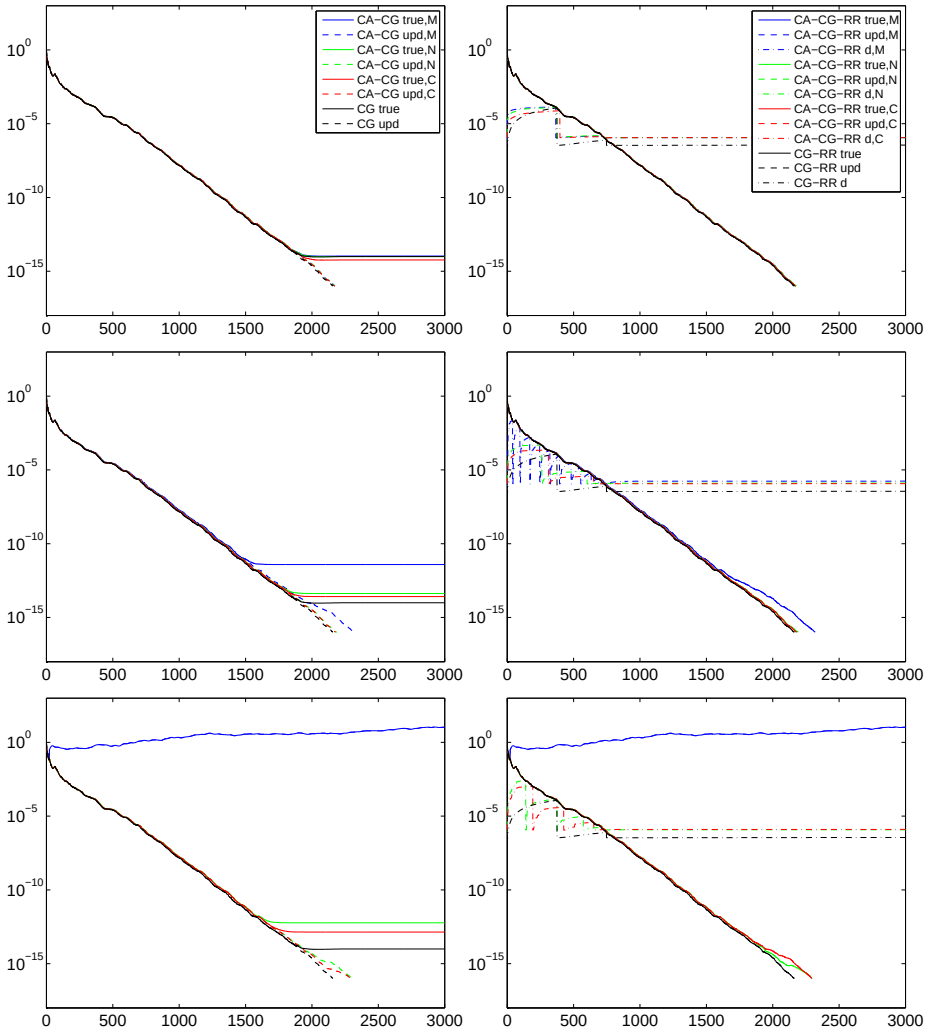

Figure 6.2: Convergence of consph test matrix for CA-CG without residual replacement (left) and with residual replacement (right), for $s = 4$ (top), $s = 8$ (middle), and $s = 12$ (bottom).

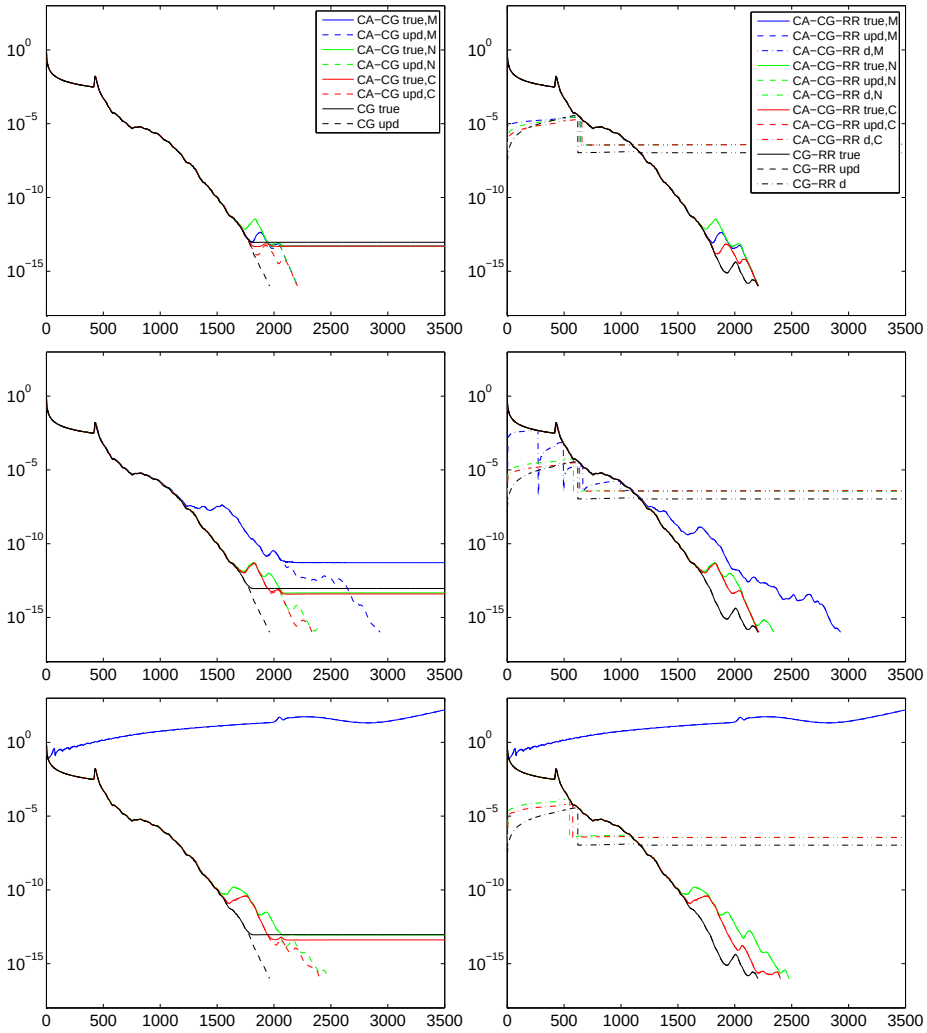

Figure 6.3: Convergence of thermal1 test matrix for CA-CG without residual replacement (left) and with residual replacement (right), for $s = 4$ (top), $s = 8$ (middle), and $s = 12$ (bottom).

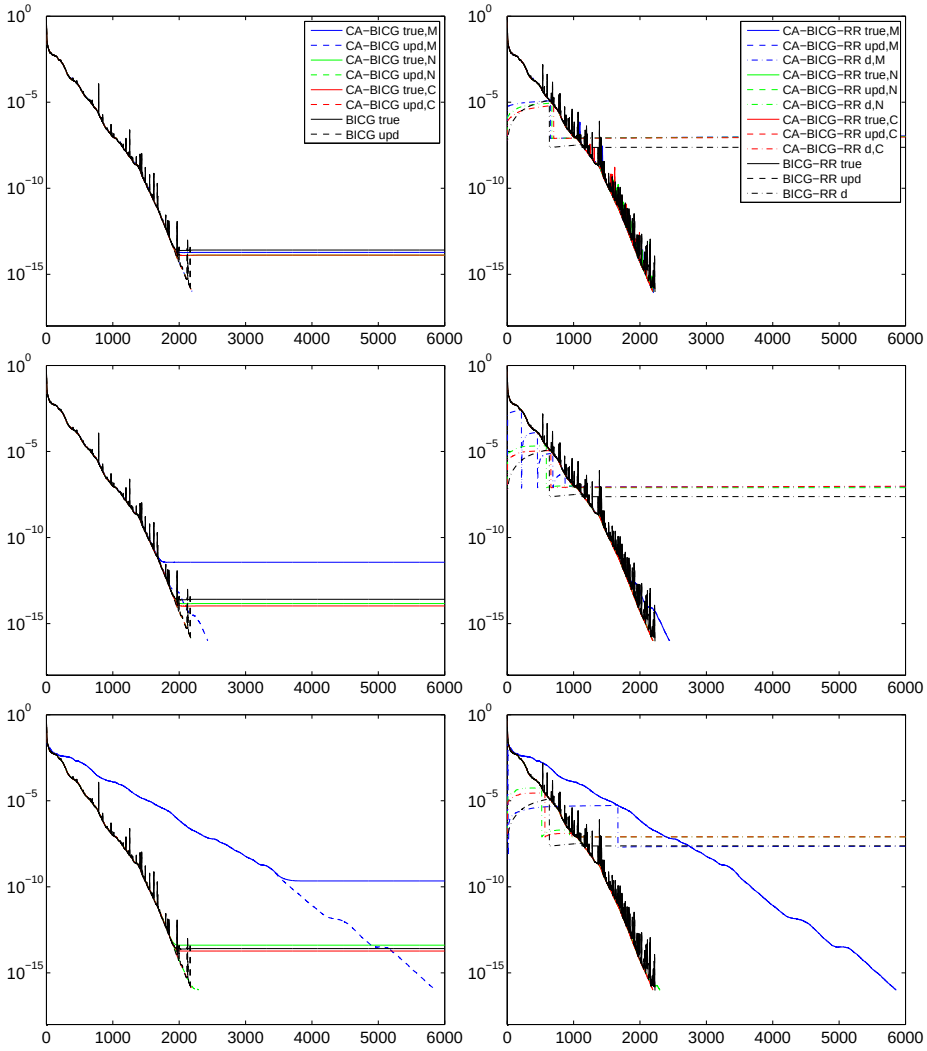

Figure 6.4: Convergence of xenon1 test matrix for CA-BICG without residual replacement (left) and with residual replacement (right), for $s = 4$ (top), $s = 8$ (middle), and $s = 12$ (bottom).

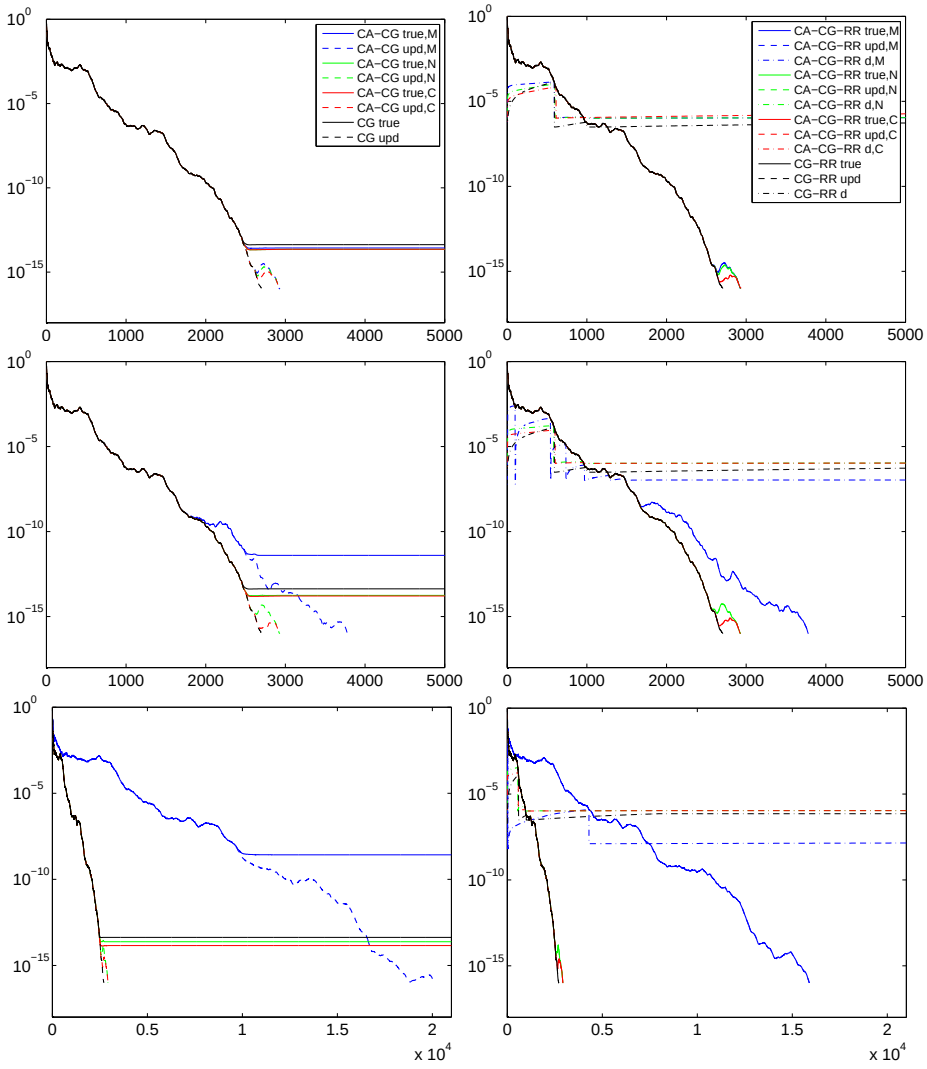

Figure 6.5: Convergence of G2circuit test matrix for CA-CG without residual replacement (left) and with residual replacement (right), for $s = 4$ (top), $s = 8$ (middle), and $s = 12$ (bottom). Note the x-axis scale differs for $s = 12$ (bottom row).

bases in Figs. 6.3 and 6.5).

The improvement in maximum attainable accuracy due to the replacement scheme for each test is summarized in Table 6.2, which gives the 2-norm of the true residual (normalized by $\|r_1\|$ as described above) for CA-(BI)CG without and with residual replacement. Our results show that, for cases where the updated residual converges to $O(\epsilon)\,\|A\|\,\|x\|$, the residual replacement scheme for CA-(BI)CG improves convergence of the true residual to within $O(\epsilon)\,\|A\|\,\|x\|$. As shown in Table 6.2, this improvement can be as large as 7 orders of magnitude.

In all cases, the updated residuals in the algorithms with replacement converge at approximately the same rate in the algorithms without replacement. We note that for tests with the monomial basis and $s = 12$, this behavior is sensitive to the chosen $\hat{\epsilon}$ value. Choosing $\hat{\epsilon}$ too low can result in replacing too frequently, as described above, which can cause slower convergence or divergence. In Fig. 6.4, we can see that, although the overall rate of convergence is unaffected, residual replacement does result in more irregular convergence for both CA-BICG and classical BICG.

The percentage of total iterations that were residual replacement iterations for each test, shown in Table 6.3 did not exceed 2% in any of our experiments. The highest number of replacements in any test was 12 (for monomial with $s = 8$ in Fig. 6.2). For reference, in Table 6.4 we give the iteration number(s) where replacements occurred in each test. Since the number of residual replacement steps is very small relative to the total number of iterations, the communication cost of performing residual replacements (adding an additional outer loop iteration) is negligible in the context of CA-(BI)CG.

6.2 Deflation-Based Preconditioning

As we've stressed, in addition to speed per iteration, the performance of iterative methods also depends on the total number of iterations required for convergence. For conjugate gradient (CG), the KSM of choice for solving symmetric positive definite (SPD) systems, it is well-known that the rate of convergence in exact arithmetic can be bounded in terms of the eigenvalue distribution. These bounds are not always tight, and additional complications arise in finite precision computations (see, e.g., [125]). Nonetheless, preconditioning techniques, wherein the system's eigenvalue distribution is altered to improve the convergence bounds, have been employed successfully in practice; for a survey of approaches, see [153].

In general, the ability to exploit temporal locality across KSM iterations is diminished by preconditioning, and the relative performance benefits of using a CA-KSM over its classical counterpart decline; see, e.g., [102]. Avoiding communication in a general preconditioned method seems to necessitate significant modifications to the algorithm. There are, however, many types of preconditioners which have been shown to be amenable to our communication-avoiding techniques, including diagonal, polynomial, incomplete factorization, and sparse approximate inverse preconditioners. This has stimulated an active area of research in designing practical preconditioners for CA-KSMs, with much recent progress. We detail the

current state of the art and future work in designing and implementing preconditioners for CA-KSMs in Section 6.8.

To add to the set of available communication-avoiding preconditioners, we propose deflation, which can be viewed as a type of preconditioning with a singular preconditioner, as a feasible technique for improving convergence in CA-KSMs. We derive communication-avoiding deflated CG (CA-D-CG), based on the deflated CG formulation (which we refer to as 'D-CG') in [155]. Our analysis shows that the additional costs of CA-D-CG over CA-CG are lower-order, which means, as in (non-deflated) CA-CG, we still expect a possible $O(s)$ speedup per s steps over the classical implementation. This motivates deflation as a promising method for improving convergence rate in CA-KSMs in practice. In this section, we give a numerical example and performance modeling results which demonstrate that choosing the number of deflation vectors, as well as the blocking factor s, result in complex, machine-dependent tradeoffs between convergence rate and time per iteration. We note that this section has been adapted from work that first appeared in [36].

6.2.1 Related Work

A review of related work related to approaches for reducing communication costs in Krylov subspace methods including CA-KSMs can be found in Section 2.4. In this section, we focus on deflation in a CA-CG variant that uses coupled two-term recurrences rather than a single three-term recurrence. Other works that consider CA-CG in particular include [48, 49, 102, 175, 182]; a brief derivation can be found in Section 5.3, with the CA-CG algorithm appearing in Algorithm 28. In the following section we derive our new CA-D-CG algorithm. Our approach here will closely follow those in Chapter 4.

Deflation and augmentation techniques have been applied to improve convergence in many KSMs since the mid 1980s; for a survey, see [159, Chapter 9]. Many approaches in the literature can be viewed as instances of more general deflation/augmentation frameworks [73, 95]. Connections have also been drawn between deflation and multilevel preconditioners; see, e.g., [173], and references therein.

In this work, we consider the case of CG, the first KSM that was modified to perform deflation [66, 134]; we note that the potential for eigenvalue deflation was also known to Rutishauser before such methods gained popularity in the literature [67]. In this work, we study the D-CG formulation as given in [155]. CG is convenient since it allows us to concretely demonstrate our algorithmic reorganization while sidestepping technical issues involving breakdown, i.e., where KSM iterates may not exist in exact arithmetic; however, we see no obstacles beyond breakdown to extending our approach to other KSMs that deflate in a similar manner.

As mentioned, there has been much recent work in the development of practical preconditioners for CA-KSMs; we defer this discussion to Section 6.8. We also note that there has been recent work in developing a high-performance deflated "pipelined" conjugate gradient method [80].

Previous work has developed efficient deflation for a CA-KSM in order to recover information lost after restarting the Arnoldi process. Wakam and Erhel [137] extended a special case of CA-GMRES [102, 127] with an adaptive augmentation approach. Both their algorithm and ours aim to reduce the frequency of global collectives incurred by deflation/augmentation compared to previous approaches. However, our applications differ: in our case we apply deflation as a more general preconditioning technique for Lanczos-based methods. We note that the algorithm presented in [137] restricts the constructed Krylov bases to Newton polynomials. It may be beneficial to extend their approach to the more general family of polynomials, which we consider here.

6.2.1.1 Deflated Conjugate Gradient

The classical CG method and CA-CG method can be found in Section 5.3 in Algorithms 27 and 28, respectively. Our CA-D-CG is based on D-CG by Saad et al. [155], shown in Algorithm 30 for reference. (As mentioned above, this was not the first appearance of deflated CG in the literature.)

We now summarize the motivation for the use of eigenvalue deflation in the CG method, as presented in [155]. It is well-known (see, e.g., [84]) that in exact arithmetic, after i iterations of CG, the error is (loosely) bounded by

$$\|x - x_i\|_A \leq 2\|x - x_1\|_A \left(\frac{\sqrt{\kappa(A)-1}}{\sqrt{\kappa(A)+1}}\right)^i,$$

with $\kappa(A) = \lambda_n/\lambda_1$ where $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of the SPD matrix A. The authors of [155] prove that for some set of linearly independent vectors $W = [w_1, w_2, \ldots, w_c]$, D-CG applied to $Ax = b$ is mathematically equivalent to CG applied to the positive semidefinite system $H^TAH\tilde{x} = H^Tb$ where $H = I - W(W^TAW)^{-1}(AW)^T$ is the A-orthogonal projection onto $W^{\perp_A}$ and $x = H\tilde{x}$. When the columns of W are exact eigenvectors associated with $\lambda_1, \ldots, \lambda_c$, the *effective* condition number (see [70]) is $\kappa_{\text{eff}}(H^TAH) = \lambda_n/\lambda_{c+1}$. When the columns of W are approximate eigenvectors associated with $\lambda_1, \ldots, \lambda_c$, one can expect $\kappa_{\text{eff}}(H^TAH) \approx \lambda_n/\lambda_{c+1}$. Thus if $\lambda_{c+1} > \lambda_1$, deflation decreases the effective condition number of the system, thus theoretically improving the bounds on (exact arithmetic) convergence rate.

We note that it is well-known that in reality, the convergence of the conjugate gradient method in exact arithmetic is much more accurately described by considering the spacing between eigenvalues of the matrix A. However, neither the bounds based on $\kappa(A)$ nor these more descriptive bounds based on eigenvalue spacing hold in finite precision [119]. However, the reduction of the effective condition number described above nonetheless remains the motivation behind deflation, and in practice can lead to an improved convergence rate in many cases. We also note that there has been recent work in developing tighter bounds on the rate of convergence in the deflated conjugate gradient method (see, e.g., [105]).

For consistency, we assume we have an initial guess x_1 such that $r_1 = b - Ax_1 \perp W$. To satisfy this initial requirement, one can choose $x_1 = x_0 + W(W^TAW)^{-1}W^Tr_0$ where x_0 is

Algorithm 30 Deflated Conjugate Gradient (D-CG)

Input: $n \times n$ symmetric positive definite matrix A, length-n vector b, initial approximation x_0 to $Ax = b$, and n-by-c matrix W of rank c

Output: Approximate solution x_{i+1} to $Ax = b$ with updated residual r_{i+1}

1: Compute and factorize $W^T AW$
2: $r_0 = b - Ax_0$, $x_1 = x_0 + W(W^T AW)^{-1} W^T r_0$, $r_1 = b - Ax_1$
3: $\mu = (W^T AW)^{-1} W^T Ar_1$, $p_1 = r_1 - W\mu$
4: **for** $i = 1, 2, \ldots$ until convergence **do**
5: $\quad \alpha_i = (r_i^T r_i)/(p_i^T Ap_i)$
6: $\quad x_{i+1} = x_i + \alpha_i p_i$
7: $\quad r_{i+1} = r_i - \alpha_i Ap_i$
8: $\quad \beta_i = (r_{i+1}^T r_{i+1})/(r_i^T r_i)$
9: $\quad$ Solve $W^T AW\mu_{i+1} = W^T Ar_{i+1}$ for μ_{i+1}
10: $\quad p_{i+1} = r_{i+1} + \beta_i p_i - W\mu_{i+1}$
11: **end for**

arbitrary and $r_0 = b - Ax_0$. Note that selection of the subspace W is out of the scope of this paper. This topic is covered extensively in the literature; see, e.g., [5, 11, 45, 60, 128, 129, 168, 186].

6.2.2 Deflated Communication-Avoiding Conjugate Gradient

We now derive CA-D-CG based on D-CG (Algorithm 30). As before, we denote iteration i in Algorithm 30 with $i = sk + j$ to distinguish inner and outer loop iterations. By induction on lines 6, 7, and 10 of Algorithm 30, we can write

$$p_{sk+j}, r_{sk+j} \in \mathcal{K}_{s+1}(A, p_{sk+1}) + \mathcal{K}_s(A, r_{sk+1}) + \mathcal{K}_{s-1}(A, W), \tag{6.17}$$

$$x_{sk+j} - x_{sk+1} \in \mathcal{K}_s(A, p_{sk+1}) + \mathcal{K}_{s-1}(A, r_{sk+1}) + \mathcal{K}_{s-2}(A, W). \tag{6.18}$$

for $j \in \{1 \ldots, s+1\}$. Deflation also requires the product Ar_{sk+j+1} in the computation of μ_{sk+j+1} in line 9. Again, by induction, we can write

$$Ar_{sk+j+1} \in \mathcal{K}_{s+2}(A, p_{sk+1}) + \mathcal{K}_{s+1}(A, r_{sk+1}) + \mathcal{K}_s(A, W) \tag{6.19}$$

for $j \in \{1, \ldots, s\}$.

As before, we define matrices $\mathcal{P}_k$ and $\mathcal{R}_k$, whose columns span the Krylov subspaces $\mathcal{K}_{s+2}(A, p_{sk+1})$ and $\mathcal{K}_{s+1}(A, r_{sk+1})$, resp. For deflation, we now also require a basis $\mathcal{W}$ for $\mathcal{K}_s(A, W)$. Note that, assuming W does not change throughout the iteration, $\mathcal{W}$ need only be computed once. For the deflated method, we now define the n-by-$(2s + 3 + cs)$ matrix

$$\begin{aligned}\mathcal{Y}_k &= [\mathcal{P}_k, \mathcal{R}_k, \mathcal{W}] \\ &= [\rho_0(A)p_{sk+1}, \ldots, \rho_{s+1}(A)p_{sk+1}, \rho_0(A)r_{sk+1}, \ldots, \rho_s(A)r_{sk+1}, \rho_0(A)W, \ldots, \rho_{s-1}(A)W].\end{aligned}$$

By (6.18), (6.17), and (6.19), we can then write, for $j \in \{1, \ldots, s+1\}$, $p_{sk+j} = \mathcal{Y}_k p'_{k,j}$, $r_{sk+j} = \mathcal{Y}_k r'_{k,j}$, and $x_{sk+j} - x_{sk+1} = \mathcal{Y}_k x'_{k,j}$, i.e., the length-$(2s+3+cs)$ vectors $p'_{k,j}$, $r'_{k,j}$, and $x'_{k,j}$ are coordinates for p_{sk+j}, r_{sk+j}, and $x_{sk+j} - x_{sk+1}$, resp., in terms of the columns of $\mathcal{Y}_k$.

As in CA-CG, we can write a recurrence for computing multiplication with A; that is, for $j \in \{1, \ldots, s\}$,

$$Ap_{sk+j} = A\mathcal{Y}_k p'_{k,j} = \mathcal{Y}_k \mathcal{B}_k p'_{k,j} \quad \text{and} \quad Ar_{sk+j+1} = A\mathcal{Y}_k r'_{k,j+1} = \mathcal{Y}_k \mathcal{B}_k r'_{k,j+1},$$

where, for the deflated method, we now define block diagonal matrix

$$B_k = \begin{bmatrix} \mathcal{B}_k^{(\mathcal{P})} & & \\ & \mathcal{B}_k^{(\mathcal{R})} & \\ & & \mathcal{B}_k^{(\mathcal{W})} \otimes I_c \end{bmatrix},$$

where $\mathcal{B}_k^{(\mathcal{P})}$, $\mathcal{B}_k^{(\mathcal{R})}$, and $\mathcal{B}_k^{(\mathcal{W})}$ are of the form (4.2) with $i = s+1$, $i = s$, and $i = s-1$, respectively. Thus multiplication by B_k in basis $\mathcal{Y}_k$ is equivalent to multiplication by A in the standard basis.

Assuming the same W is used throughout the iterations, $W^T AW$ can be precomputed and factorized offline. The small c-by-c factors of $W^T AW$ are assumed to fit in local/fast memory. If we compute the $(2s+3+cs)$-by-$(2s+3+cs)$ matrix $G_k = \mathcal{Y}_k^T \mathcal{Y}_k$, and extract the c-by-$(2s+3+cs)$ submatrix $Z_k = W^T \mathcal{Y}_k$, then we can form the right-hand side in the solve for μ_{sk+j+1} in line 9 of Algorithm 30 by $W^T Ar_{sk+j+1} = Z_k \mathcal{B}_k r'_{k,j+1}$, replacing a global reduction with a small, local operation. Note that the formulas for computing α_{sk+j} and β_{sk+j} in Algorithm 30 remain the same as in Algorithm 27. Thus, using G_k, we can compute these inner products in CA-D-CG using the same formulas as in CA-CG (lines 8 and 12 of Algorithm 28).

Similarly, the formulas for updates x_{sk+j+1} and r_{sk+j+1} are the same for D-CG and CG, so the formulas for $x'_{k,j+1}$ and $r'_{k,j+1}$ in CA-D-CG remain the same as those in CA-CG (lines 10 and 11). The formula for p_{sk+j+1} in D-CG can be written

$$\mathcal{Y}_k p'_{k,j+1} = \mathcal{Y}_k r'_{k,j+1} + \beta_{sk+j} \mathcal{Y}_k p'_{k,j} - \mathcal{Y}_k [0_{1,2s+3},\, \mu_{sk+j+1}^T,\, 0_{1,c(s-1)}]^T,$$

for $j \in \{1 \ldots, s\}$. Thus in CA-D-CG, $p'_{k,j+1}$ is updated by

$$p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j} p'_{k,j} - [0_{1,2s+3},\, \mu_{sk+j+1}^T,\, 0_{1,c(s-1)}]^T.$$

The resulting CA-D-CG method is shown in Algorithm 31.

6.2.2.1 Algorithmic Extensions

We assume in our derivation that the matrix of deflation vectors W is constant through the iterations. We could, however, extend CA-D-CG to allow for updating of W_k in (some or all) outer loop iterations k; see, e.g., [5, 11, 120, 145, 168] for example applications.

Algorithm 31 Deflated Communication-Avoiding Conjugate Gradient (CA-D-CG)

Input: $n \times n$ symmetric positive definite matrix A, length-n vector b, initial approximation x_0 to $Ax = b$, and n-by-c matrix W of rank c

Output: Approximate solution x_{i+1} to $Ax = b$ with updated residual r_{i+1}

1: Compute and factorize $W^T AW$
2: Compute $\mathcal{W}$
3: $r_0 = b - Ax_0$, $x_1 = x_0 + W(W^T AW)^{-1} W^T r_0$, $r_1 = b - Ax_1$
4: $\mu = (W^T AW)^{-1} W^T A r_1$, $p_1 = r_1 - W\mu$
5: **for** $k = 0, 1, \ldots$ until convergence **do**
6: Compute $\mathcal{P}_k$, $\mathcal{R}_k$, let $\mathcal{Y}_k = [\mathcal{P}_k, \mathcal{R}_k, \mathcal{W}]$; assemble $\mathcal{B}_k$.
7: $G_k = \mathcal{Y}_k^T \mathcal{Y}_k$; extract $Z_k = W^T \mathcal{Y}_k$
8: $p'_{k,1} = [1, 0_{1,2s+2+cs}]^T$, $r'_{k,1} = [0_{1,s+2}, 1, 0_{1,s+cs}]^T$, $x'_{k,1} = 0_{2s+3+cs,1}$
9: **for** $j = 1, \ldots, s$ **do**
10: $\alpha_{sk+j} = (r'^T_{k,j} G_k r'_{k,j}) / (p'^T_{k,j} G_k \mathcal{B}_k p'_{k,j})$
11: $x'_{k,j+1} = x'_{k,j} + \alpha_{sk+j} p'_{k,j}$
12: $r'_{k,j+1} = r'_{k,j} - \alpha_{sk+j} \mathcal{B}_k p'_{k,j}$
13: $\beta_{sk+j} = (r'^T_{k,j+1} G_k r'_{k,j+1}) / (r'^T_{k,j} G_k r'_{k,j})$
14: Solve $W^T AW \mu_{sk+j+1} = Z_k \mathcal{B}_k r'_{k,j+1}$ for μ_{sk+j+1}
15: $p'_{k,j+1} = r'_{k,j+1} + \beta_{sk+j} p'_{k,j} - [0_{1,2s+3}, \mu^T_{sk+j+1}, 0_{1,c(s-1)}]^T$
16: **end for**
17: $x_{sk+s+1} = \mathcal{Y}_k x'_{k,s+1} + x_{sk+1}$, $r_{sk+s+1} = \mathcal{Y}_k r'_{k,s+1}$, $p_{sk+s+1} = \mathcal{Y}_k p'_{k,s+1}$
18: **end for**

(Additional considerations arise when changing the operator during the iterations due to the loss of orthogonality properties [10, Chapter 12]; see also [136].) Updating W_k in outer loop k requires recomputing $\mathcal{W}_k$, a basis for $\mathcal{K}_s(A, W_k)$; this computation could potentially be fused with computation of $\mathcal{P}_k$ and $\mathcal{R}_k$ such that no extra latency cost is incurred. The quantity $W_k^T AW_k$ can be recovered from the computation of G_k, so no additional communication is required. Factorization of the c-by-c matrix $W_k^T AW_k$ can also be performed locally; note the number of deflation vectors c could be allowed to vary over outer loop iterations as well. This extension is considered future work.

6.2.3 Numerical Experiments

For numerical experiments, our goal is to show that CA-D-CG is competitive with D-CG in terms of convergence rate and that it also beats CA-CG without deflation. While the approaches are equivalent in exact arithmetic, there is no reason to expect that CA-D-CG iterates will exactly equal D-CG iterates in finite precision, given the different sets of floating-point operations performed. There are many open questions about CA-KSMs' finite precision behavior, some of which we hope to address in future work. But in lieu of theoretical results, we will rely on our practical experience that CA-KSMs' iterates deviate from their classical

counterparts as the s-step Krylov bases become ill-conditioned (increasingly, with s), and this effect can be diminished by picking different polynomial bases [35, 102, 149]. To focus on this potential instability that grows with s, we chose a test problem for which the classical methods are relatively insensitive to rounding errors. Thus, our experiments do not address the possibility that the deviation between D-CG and CA-D-CG iterates is much larger when the classical methods' convergence is highly perturbed by rounding errors.

We test the stability of our reformulation on a similar model problem to the one considered in [155], using codes written in a combination of MATLAB and C, with linear algebra routines from Intel's Math Kernel Library. We generate a discrete 2D Laplacian by `gallery('poisson',512)` in MATLAB, so A is an SPD matrix of order $n = 512^2$. We pick the right-hand side b equal to A times the vector with entries all $n^{-1/2}$. Our deflation vectors are the eigenvectors corresponding to the smallest magnitude eigenvalues, computed using MATLAB's `eigs`. Note that the study in [155] used (known) exact eigenvalues; this difference does not significantly affect the results for this test.

In Figures 6.6-6.8, we compare convergence for the model problem using D-CG and CA-D-CG with the monomial, Newton, and Chebyshev polynomial basis, resp., each for a few representative s values. We report the 2-norm of the true residual computed by $b - Ax_i$, rather than the recursively updated residual r_i, and normalize by the 2-norm of the starting residual $r_1 = b$ (i.e., the starting guess x_1 is the vector of zeros). We declare convergence after a factor of 10^8 reduction in the normalized residual 2-norm. The solid curves correspond to D-CG, and circles correspond to CA-D-CG. We deflate with $c \in \{0, 4, 8\}$ eigenvectors, plotted in black, red, and blue, respectively (when $c = 0$, D-CG is just CG and CA-D-CG is just CA-CG). Based on the formulas above, this suggests the condition number $\kappa(A) \approx 1.07 \cdot 10^5$ in the undeflated case ($c = 0$) should improve to $\approx 2.13 \cdot 10^4$ in the case $c = 4$, and to $\approx 1.25 \cdot 10^4$ when $c = 8$.

We implemented the Newton basis by choosing parameters in (4.2) as $\hat{\beta}_i = 0$, $\hat{\gamma}_i = 1$, and $\hat{\alpha}_i$ is the i-th element in a set of Leja-ordered points on the real line segment $[\lambda_{c+1}, \lambda_n]$; see, e.g., [149]. We implemented the Chebyshev basis by setting basis parameters in (4.2) as $\hat{\gamma}_i = |\lambda_n - \lambda_{c+1}|/2$ (except $\hat{\gamma}_0$, which is not divided by 2), $\hat{\alpha}_i = \lambda_{c+1} + |\lambda_n - \lambda_{c+1}|/2$, and $\hat{\beta}_i = |\lambda_n - \lambda_{c+1}|/8$. These recurrence coefficients are based on the bounding ellipse of the spectrum of A, which is, in the present case of symmetric A, an interval on the real line; see, e.g., [104]. In practice, only a few Ritz values (estimates for eigenvalues of A) need to be computed up front to sufficiently determine parameters for Newton or Chebyshev polynomials. One can also incorporate information about new Ritz values obtained as a byproduct of the iterations to improve the basis conditioning; see [149] for practical details and experiments.

Note that λ_{c+1} is used as the smallest eigenvalue in selecting Newton and Chebyshev parameters above. This is because if columns of W are exact eigenvectors of A corresponding to eigenvalues $\lambda_1, \ldots, \lambda_c$, using λ_1 as a basis parameter in computation of basis $\mathcal{W}$ can cause cancellation and can thus produce a rank-deficient basis. Although this cancellation does not occur in computation of bases $\mathcal{P}_k$ and $\mathcal{R}_k$, we used the same basis parameters chosen for $\mathcal{W}$ (i.e., using λ_{c+1}) to compute $\mathcal{P}_k$ and $\mathcal{R}_k$ for simplicity, with no ill effects.

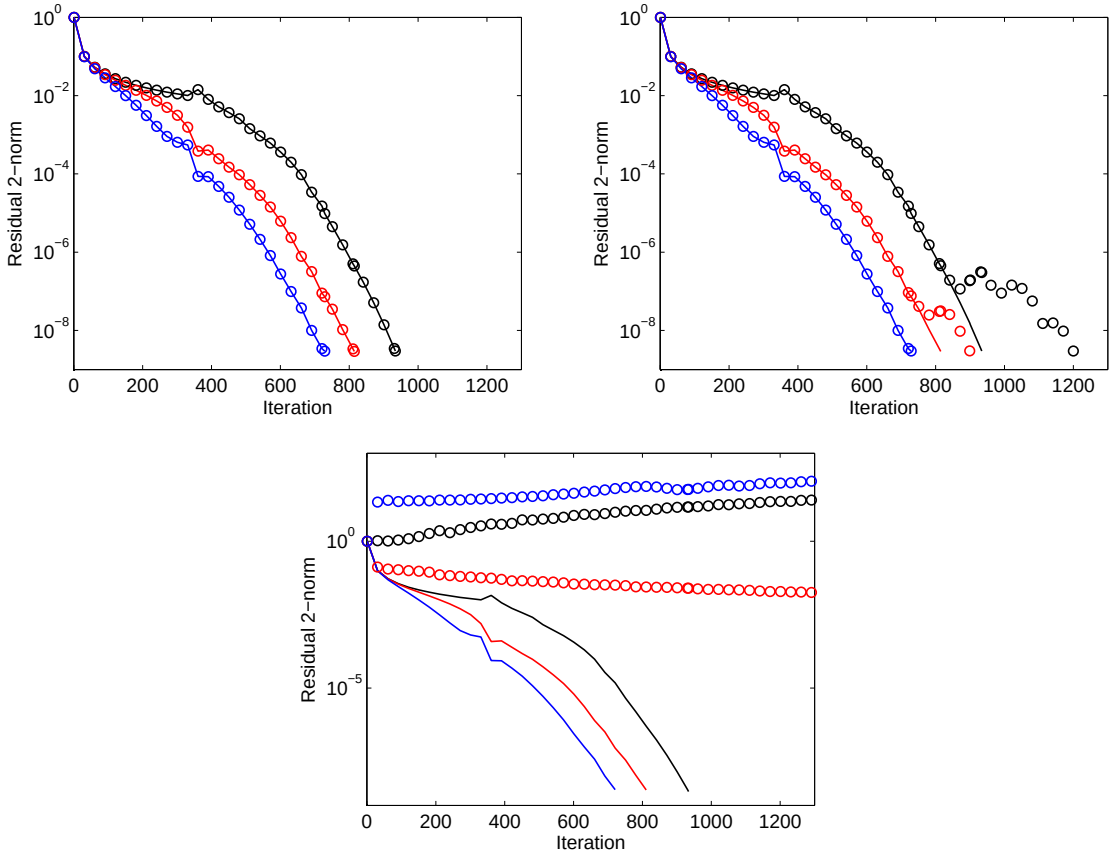

Figure 6.6: Monomial basis tests. Top: $s = 4$ (left), $s = 8$ (right), Bottom: $s = 16$. Plots show the 2-norm of the true residual, $b - Ax_i$, for tests with $c = 0$ (black), $c = 4$ (red), and $c = 8$ (blue), for both D-CG ($-$) and CA-D-CG ($\circ$) using monomial bases of size s. Note that the y-axis in the bottom plot differs.

For the monomial basis (Figure 6.6), convergence is nearly identical for $s = 4$, but we begin to see a delay in CA-D-CG convergence for $s = 8$ (top-left), and a failure to converge by $s = 16$. For the Newton basis (Figure 6.7), the two methods have similar convergence past $s = 16$; only around $s = 100$ (bottom-right) do we begin to notice a significant delay in convergence for CA-D-CG. The situation is similar for the Chebyshev basis (Figure 6.8), only the bottom-right subfigure now depicts the case $s = 220$. These results clearly demonstrate that basis choice plays an important role for CA-D-CG convergence, at least on this well-behaved model problem

In the next section, we will introduce a coarse performance model to ask about the practical benefits of values as large as $s = 220$.

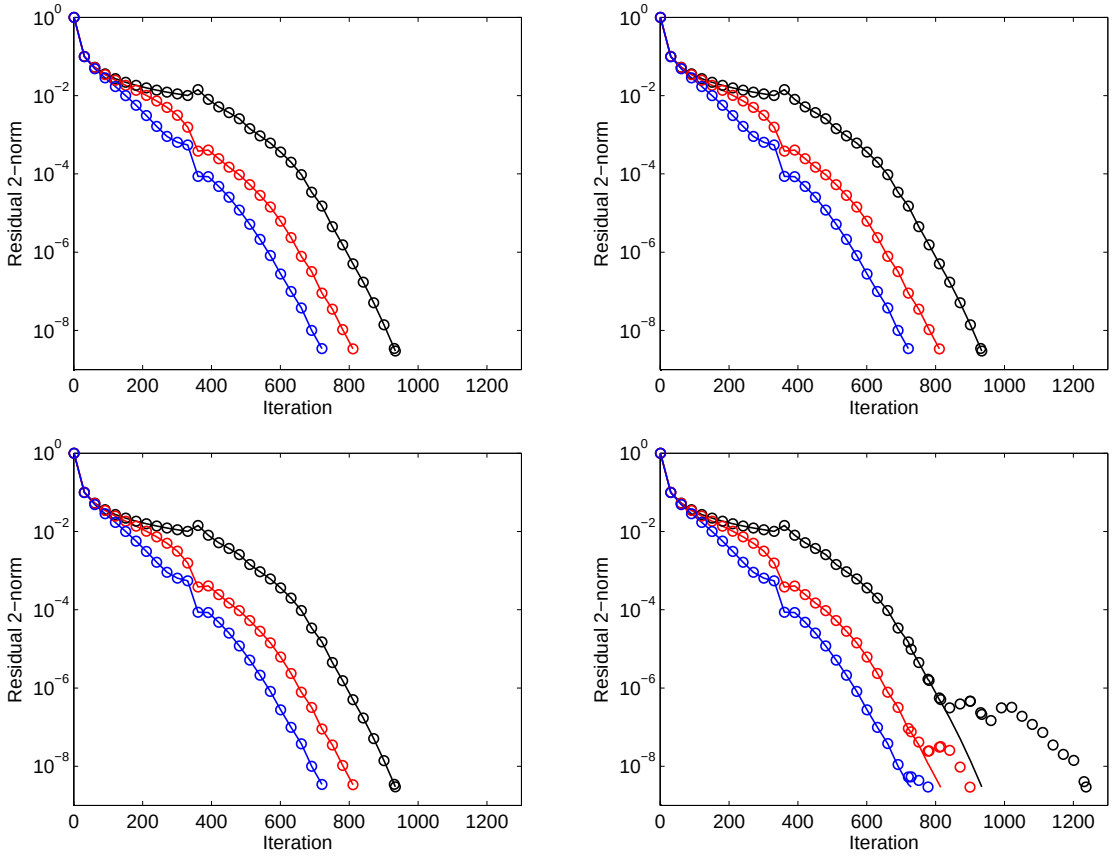

Figure 6.7: Newton basis tests. Top: $s = 4$ (left), $s = 8$ (right), Bottom: $s = 16$ (left), $s = 100$ (right). Plots show the 2-norm of the true residual, $b - Ax_i$, for tests with $c = 0$ (black), $c = 4$ (red), and $c = 8$ (blue), for both D-CG ($-$) and CA-D-CG ($\circ$) using Newton bases of size s.

6.2.4 Performance Modeling

In this section, we give a qualitative description of the performance tradeoffs between the four KSMs mentioned above — CG, CA-CG, and their deflated counterparts — on massively parallel machines. We use the theoretical machine model discussed in Section 2.2; recall that for an algorithm where a processor (on the critical path) performs F flops and sends/receives S messages containing a total of W words, we model the worst-case runtime using the equation

$$T = \gamma F + \beta W + \alpha S,$$

where α represents a latency cost incurred by every message, β represents a bandwidth cost linear in the message size, and γ represents the time to perform a floating point operation on local data. For CA-CG and CA-D-CG, we estimate the time for s inner loop iterations

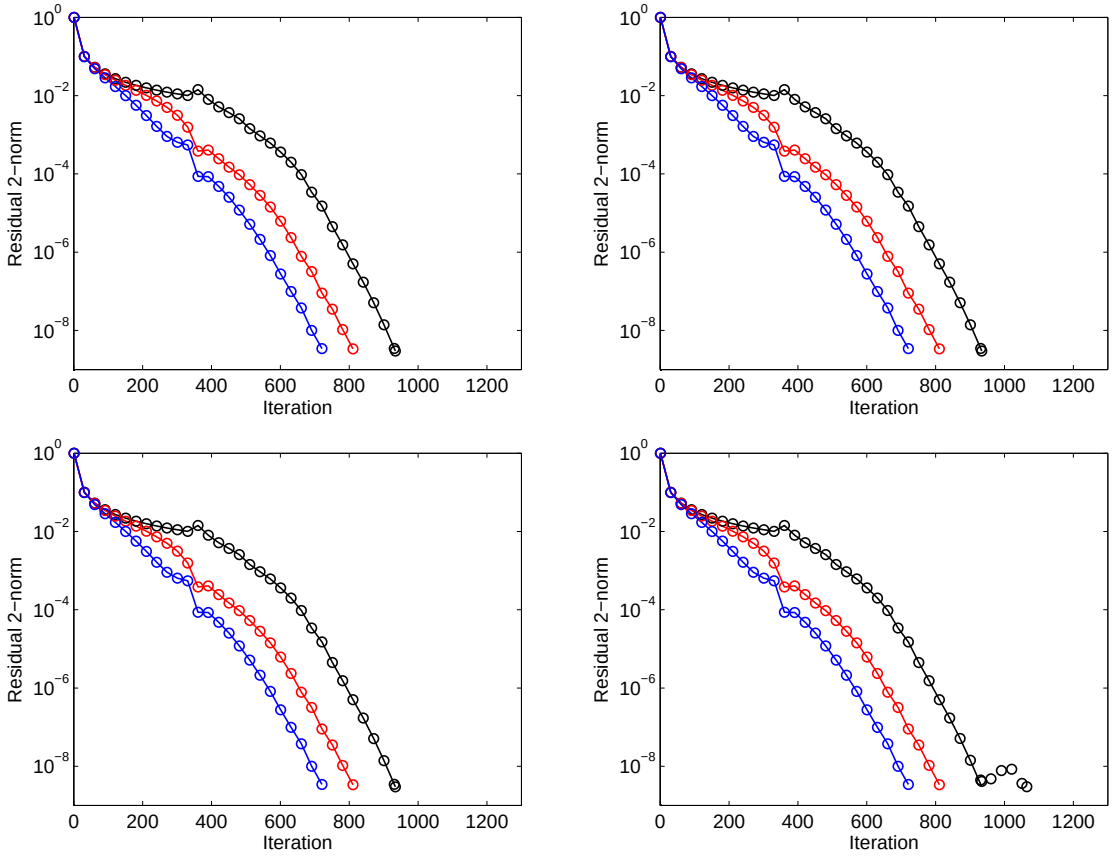

Figure 6.8: Chebyshev basis tests. Top: $s = 4$ (left), $s = 8$ (right), Bottom: $s = 16$ (left), $s = 220$ (right). Plots show the 2-norm of the true residual, $b - Ax_i$, for tests with $c = 0$ (black), $c = 4$ (red), and $c = 8$ (blue), for both D-CG ($-$) and CA-D-CG ($\circ$) using Chebyshev bases of size s.

and then divide by s to estimate time per iteration. Note that this ignores relative rates of convergence, treated in Section 6.2.3.

We were motivated to develop CA-D-CG based on the high relative cost of interprocessor communication on large-scale parallel computers. In addition to parallel implementations, CA-KSMs can avoid communication on sequential machines, namely data movement within the memory hierarchy. Indeed, the parallel and sequential approaches naturally compose hierarchically, as has been exploited in previous high-performance CA-KSM implementations [127], and we suggest the same for a future CA-D-CG implementation. However, in this section, we will restrict ourselves to a parallel model which ignores sequential communication costs to illustrate the changes in parallel communication costs. We do not claim that this model's predicted speedups are always attainable on real hardware; however, we believe that such models can help to determine feasibility of a communication-avoiding approach relative to a classical approach, as well as to efficiently explore and prune the (machine-specific)

parameter tuning space for an optimized implementation.

We consider two parallel machine models, which we call 'Exa' and 'Grid.' We use $\gamma = 1 \cdot 10^{-13}$ seconds per (double precision) flop in both cases, based on predictions for a 'node' of an exascale machine [41, 177]. This flop rate corresponds to a node with a 1024-core processor and its own memory hierarchy with 256 GB of capacity at the last level; however, as discussed above, we ignore this intranode structure. For Exa, the interconnect has parameters $\alpha = 4 \cdot 10^{-7}$ seconds per message and $\beta = 3.7 \cdot 10^{-11}$ seconds per word (4-byte double precision value). For the second machine, Grid, we replace this interconnect by the Internet (via Ethernet), using the parameters $\alpha = 10^{-1}$ and $\beta = 2.5 \cdot 10^{-8}$ given in [126]. In contrast to their predecessors, our models allow an arbitrary number of processors, in order to illustrate asymptotic scaling behavior — we do not claim that every machine configuration modeled is physically realizable.

We will study the same model problem (a 5-point 2D stencil with constant coefficients) and numbers of deflation vectors as in the numerical experiments. Because the model problem is a stencil with constant coefficients, we assume it can be represented in $O(1)$ words. We assume the $\sqrt{n}$-by-$\sqrt{n}$ mesh is partitioned across a $\sqrt{p}$-by-$\sqrt{p}$ grid of processors, so that each processor owns a contiguous $\sqrt{n/p}$-by-$\sqrt{n/p}$ subsquare, and we assume these fractions are integers; this layout minimizes communication within a factor of $\sqrt{2}$ (for this particular stencil, a diamond layout would be asymptotically optimal [21, Chapter 4.8]).

To simplify the analysis, we restrict $s \in \{1, \ldots, \sqrt{n/p}\}$ for the CA-KSMs, which means that the sparse computations only require communication with (logical) nearest neighbors; the communication-avoiding approach is correct for any s, but the latency cost rises sharply when each processor needs information from a larger neighborhood. We also simplify by using the same blocking parameter s for the sparse and dense computations; in general, one can compute the s-step Krylov bases in smaller blocks, and then compute a (larger) Gram matrix, or vice versa, i.e., constructing the Gram matrix blockwise as the s-step bases are computed. In practice, we have observed significant speedups from the Gram matrix construction alone (with no blocking of the SpMV operations) [190], and we suggest tuning the block sizes independently. Also to simplify the analysis, we will ignore the preprocessing costs of computing the deflation matrix W (not the algorithmic cost W) and computing and factorizing $W^T AW$, assuming that they can be amortized over many iterations; in practice, these costs may not be negligible, especially if the number of iterations is small.

We note that in the case of variable coefficients, i.e., the nonzeros of A are stored explicitly, we would partition A in a overlapping block rowwise fashion, as explained in [126]. For variable coefficients, we would then incur the additional, but amortizable, cost of distributing rows of A to neighboring processors as described in Section 3.2.

Under these assumptions, the number of flops, words moved, and messages required for s steps of CG, CA-CG, D-CG, and CA-D-CG are as follows:

$$\begin{aligned} \text{Flops}_{\text{CG}} &= s(19n/p + 2\log_2 p) \\ \text{Words}_{\text{CG}} &= s(4\sqrt{n/p} + 4\log_2 p) \end{aligned}$$

$$
\begin{aligned}
\text{Mess}_{\text{CG}} &= s(4\log_2 p + 4) \\
\text{Flops}_{\text{CA-CG}} &= 18(n/p)s + s(20s + 3(2s+1)(4s+1) + 10) + 12s^3 \\
&\quad + 2(n/p)(4s+1) + (n/p)(4s+3) + 36\sqrt{n/p}s^2 \\
&\quad + ((2s+1)(2s+2)(2(n/p) + \log_2 p - 1))/2 \\
\text{Words}_{\text{CA-CG}} &= 8\sqrt{n/p}s + 4s^2 + \log_2 p(2s+1)(2s+2) \\
\text{Mess}_{\text{CA-CG}} &= 2\log_2 p + 8 \\
\text{Flops}_{\text{D-CG}} &= s(30(n/p) + 2\log_2 p + c(2(n/p) + \log_2 p - 1) + 2c^2 + (n/p)(2c-1)) \\
\text{Words}_{\text{D-CG}} &= s((4+2c)\log_2 p + 8\sqrt{n/p}) \\
\text{Mess}_{\text{D-CG}} &= s(6\log_2 p + 8) \\
\text{Flops}_{\text{CA-D-CG}} &= 12(s+1)^3 + 2(n/p)(4s + 2cs + 5) + (n/p)(4s + 2cs + 7) \\
&\quad + 36\sqrt{n/p}(s+1)^2 + 18(n/p)(s+1) + s(24s + c(4s + 2cs + 5) \\
&\quad + 4(2s + cs + 3)(4s + 2cs + 5) + 12cs + 2c^2 + 36) \\
&\quad + (2s+3)(s + cs + 2)(2(n/p) + \log_2 p - 1) \\
\text{Words}_{\text{CA-D-CG}} &= 4(s+1)^2 + 8\sqrt{n/p}(s+1) + 2\log_2 p(2s+3)(s + cs + 2) \\
\text{Mess}_{\text{CA-D-CG}} &= 2\log_2 p + 8
\end{aligned}
$$

For the CA-KSMs, we do not exploit the nonzero structure of G_k and $\mathcal{B}_k$ for local matrix-vector multiplications, nor the nonzero structure of the length-$O(s)$ coefficient vectors.

For D-CG, we note that one can compute AW offline (in line 1), and avoid the SpMV Ar_{i+1} in line 9; while this may improve some constant factors by up to 2, it does not avoid the global reduction in the subsequent application of $(AW)^T$, which our performance modeling suggests is often the dominant cost.

We note that the $\log_2 p$ terms in computation and bandwidth costs can often be reduced by exploiting efficient collectives based on recursive halving/doubling approaches; see [42] for a survey. These approaches require that the number of words in the collective is at least p, which was not always true in our experiments, hence our use of simpler tree-based collectives.

We first consider weak scaling in Figure 6.9. We fix $n/p = 4^6$ and vary $p \in \{4^x : x \in \{2, \ldots, 14\}\}$. The black curves correspond to the runtime of a single iteration of the classical KSMs: CG (no markers), D-CG with $c = 4$ (square markers), and D-CG with $c = 8$ (asterisk markers) — the logarithmic dependence on p, due to the collective communications, is evident. The red curves allow us to vary the parameter $s \in \{1, \ldots, \sqrt{n/p}\}$, where $s = 1$ corresponds to the classical KSMs, and $s > 1$ corresponds to their deflated counterparts (markers mean the same as for the black curves); for comparison with the classical methods, we compute the runtime of one CA-KSM outer loop (with s inner-loop iterations) and then divide by s. On both Exa and Grid, it was beneficial to pick $s > 1$ for every problem, although the optimal s varies, as illustrated in Figure 6.11. The best speedups, i.e., the ratio of the runtime with $s = 1$ to the best runtime with $s \geq 1$, were about 55, 38, and 28 for

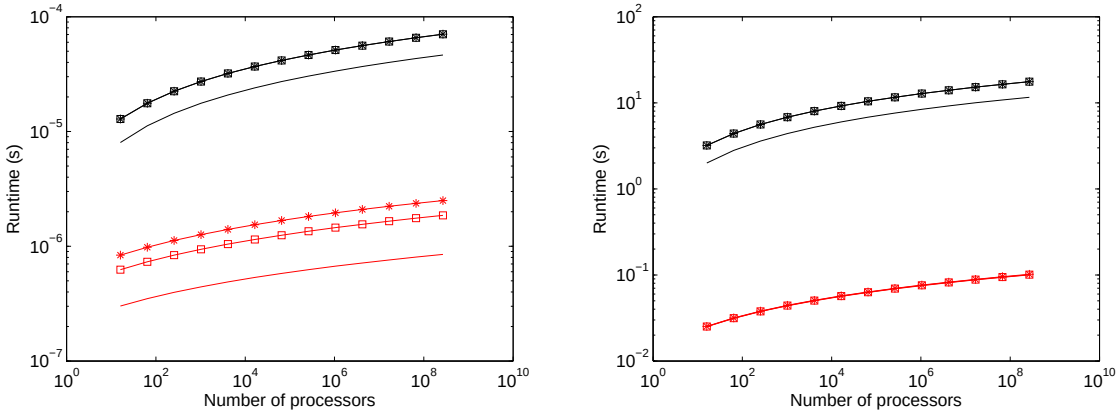

Figure 6.9: Modeled weak scaling on model problem on Exa (left) and Grid (right). Black curves correspond to the runtime of a single iteration of the classical KSMs: CG (no markers), D-CG with $c = 4$ (square markers), and D-CG with $c = 8$ (asterisk markers). Red curves correspond to the runtime of a single iteration of the CA-KSMs: CA-CG (no markers), CA-D-CG with $c = 4$ (square markers), and CA-D-CG with $c = 8$ (asterisk markers), using the optimal value of $s \in \{1, \ldots, \sqrt{n/p}\}$ for each point.

$c = 0$, 4, and 8, resp. on Exa, while the corresponding best speedups on Grid were about 116, 174, and 173.

Strong scaling plots are shown in Figure 6.10. The curves represent the same algorithms as in the previous figure, except now we use different problems for the two machines (we use the same range of p as before). Note that the red and black curves coincide for some points on the left of both plots. As the local problem size decreases, so does the range of s values over which the CA-KSMs optimize. For Exa, we fix $n = 4^{15}$, so for the largest p, e.g., the processors' subsquares are 2-by-2 and $s \in \{1, 2\}$; for Grid, we fix $n = 4^{22}$. While all tested KSMs scale when the local problem is large, the CA-KSMs are able to exploit more parallelism than the classical KSMs on both machines. In both cases, the CA-KSM runtime eventually begins to increase too. The best speedups on Exa were about 49, 42, and 31 for $c = 0$, 4, and 8, resp., while the corresponding best speedups on Grid were about 1152, 872, and 673.

Lastly, in Figure 6.11, we demonstrate the benefits of increasing the parameter s for a fixed problem and a varying number $c \in \{0, \ldots, 50\}$ of deflation vectors. The case $c = 0$ indicates the non-deflated KSMs, and is depicted separately. We plot the CA-KSMs' speedups relative to the classical KSMs, i.e., the points along the line $s = 1$. For both machines, we fix $p = 4^9$, but to illustrate the tradeoffs on both, we pick $n = 4^{14}$ for Exa, and $n = 4^{20}$ for Grid. In both cases, we see decreased relative benefits of avoiding communication as c increases, as the network bandwidth becomes saturated by the larger reductions. For Exa, for small c it is beneficial to increase s to the maximum $\sqrt{n/p}$ we consider; for Grid, however, it is never beneficial to increase s to its maximum, for any c.

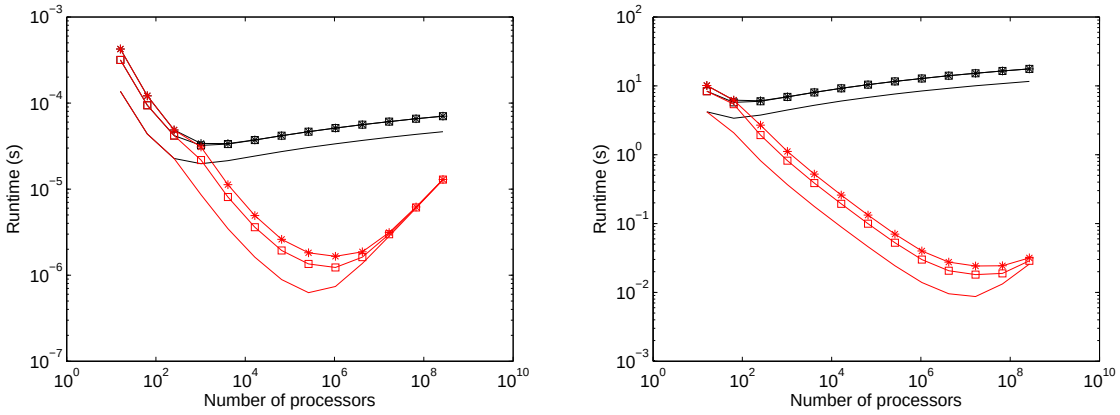

Figure 6.10: Modeled strong scaling on model problem on Exa (left) and Grid (right). Black curves correspond to the runtime of a single iteration of the classical KSMs: CG (no markers), D-CG with $c = 4$ (square markers), and D-CG with $c = 8$ (asterisk markers). Red curves correspond to the runtime of a single iteration of the CA-KSMs: CA-CG (no markers), CA-D-CG with $c = 4$ (square markers), and CA-D-CG with $c = 8$ (asterisk markers), using the optimal value of $s \in \{1, \ldots, \sqrt{n/p}\}$ for each point.

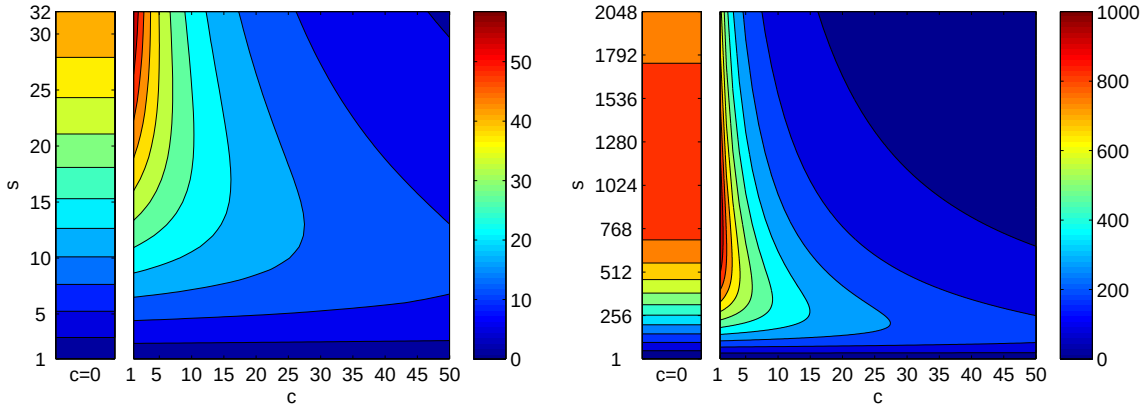

Figure 6.11: Modeled speedup per iteration on model problem, versus s and c, on Exa (left) and Grid (right).

6.2.5 Future Work and Conclusions

In this work, we have demonstrated that deflation can be performed in a communication-avoiding way, and is thus suitable for use as a preconditioner for CA-KSMs. We derived CA-D-CG, which is equivalent to D-CG in exact arithmetic but can be implemented such that parallel latency is reduced by a factor of $O(s)$ over a fixed number of iterations. Performance modeling shows predicted speedups of CA-D-CG over D-CG for a number of n/p ratios on two model architectures, for s values constrained so $s \leq \sqrt{n/p}$. We performed numerical experiments for a model problem to illustrate the benefits of deflation for convergence rate. Our results also demonstrate that by using better conditioned bases of Newton and Chebyshev polynomials, s can be made very large before convergence behavior of CA-D-CG deviates significantly from D-CG; however, for more difficult problems, to be studied in future work, we expect the practical range of s to be more restricted.

We also point out that, as in the classical case, our CA-D-CG method is mathematically equivalent to applying CA-CG (Algorithm 28) to the transformed system $H^T A H \tilde{x} = H^T b$. If we were to instead perform deflation in this way, communication-avoiding techniques related to blocking covers for linear relaxation [112, 117] and other ideas from those works could be applied in the Krylov basis computation.

The communication-avoiding reorganization applied here can also be applied to many other deflated KSMs, including adaptive deflation approaches (see, e.g., [5, 11, 120, 145, 168]), where the matrix W is allowed to change. Our future work will address these applications, as well as a distributed-memory implementation to evaluate the performance of our approaches on real parallel machines.

6.3 Selective Reorthogonalization

In practice, the Lanczos vectors computed by the Lanczos method lose orthogonality due to roundoff errors. This can cause multiple copies of converged Ritz values to appear, which delays convergence to other eigenvalues. One option to remedy this is to run the Lanczos algorithm with full reorthogonalization; if strong linear independence is maintained between the Lanczos vectors, redundant Ritz pairs can not be formed and the number of Lanczos steps does not exceed n. These desirable numerical properties come at the expense of extra computation, communication, and space requirements in order to store all previous vectors and perform the orthogonalization.

The selective orthogonalization technique of Parlett and Scott [147] achieves a middle ground between full reorthogonalization and no reorthogonalization, maintaining near orthogonality among the Lanczos vectors for a much lower cost than the full reorthogonalization method. This technique is grounded in the theory of Paige [140, 141], which says that the Lanczos vectors lose their orthogonality by developing large components in the direction of converged Ritz vectors. By monitoring the computed error bounds, it can be determined which Ritz vectors will contribute large components to the current vector $\hat{v}_{m+1}$, and $\hat{v}_{m+1}$

can then be orthogonalized against those Ritz vectors.

Selective orthogonalization can be justified as follows. A result of Paige [140] says that, for the classical Lanczos algorithm at iteration m,

$$z_i^{(m)T}\hat{v}_{m+1} = -\frac{\epsilon_{i,i}^{(m)}}{\hat{\beta}_{m+1}\eta_{m,i}^{(m)}},$$

where $\hat{v}_{m+1}$ is the current Lanczos vector, $z_i^{(m)}$ is the ith Ritz vector computed in iteration m, $\eta_{m,i}^{(m)}$ is the last element of the ith eigenvector of $\hat{T}_m$ computed in iteration m, and $\epsilon_{i,i}^{(m)} = q_i^{(m)T}\delta R_m q_i^{(m)}$ with $|\epsilon_{i,i}^{(m)}| = O(\epsilon\|A\|)$. We note that this notation is the same used in Section 5.1. Thus a small $\hat{\beta}_{m+1}\eta_{m,i}^{(m)}$, which indicates convergence of Ritz pair $(\mu_i^{(m)}, z_i^{(m)})$, causes loss of orthogonality by introducing to $\hat{v}_{m+1}$ large components of $\hat{z}_i^{(m)}$. Then to prevent this loss of orthogonality, we want to purge $\hat{v}_{m+1}$ of the *threshold* Ritz vectors corresponding to indices

$$\mathcal{L}(m+1) = \{i : |z_i^{(m)T}\hat{v}_{m+1}| \geq \sqrt{\epsilon}\}.$$

For most iterations in classical Lanczos, $|\mathcal{L}| = 0$ but when $|\mathcal{L}(m+1)| > 0$ then we orthogonalize $\hat{v}_{m+1}$ against $z_i^{(m)}$ for $i \in \mathcal{L}(m+1)$. This requires computing the $z_i^{(m)}$'s, which requires bringing in $\hat{V}_m$ from memory. In classical Lanczos,

$$z_i^{(m)T}\hat{v}_{m+1} = \frac{O(\epsilon\|A\|)}{\hat{\beta}_{m+1}\eta_{m,i}^{(m)}},$$

so we determine which indices $\mathcal{L}(m+1)$ will contain by finding the i's such that $|\hat{\beta}_{m+1}\eta_{m,i}^{(m)}| \leq \sqrt{\epsilon}\|A\|$.

We want to determine a similar criterion for the CA-Lanczos method. We can use our results for CA-Lanczos in Section 5.1, which are analogous to the results of Paige for classical Lanczos [140, 141], to develop a selective orthogonalization technique for CA-Lanczos. Similar to the results of Paige, by (5.77) and (5.78), we have

$$z_i^{(m)T}\hat{v}_{m+1} = \frac{\epsilon_{i,i}^{(m)}}{\hat{\beta}_{m+1}\eta_{m,i}^{(m)}}.$$

where $|\epsilon_{i,i}^{(m)}| = O(\epsilon\|A\|\bar{\Gamma}_k^2)$, with $\bar{\Gamma}_k = \max_{\ell\in\{0,\ldots,k\}} \|\mathcal{Y}_\ell^+\|_2\||\mathcal{Y}_\ell|\|_2$ (see Theorem 1). In this case, we will have $|z_i^{(m)T}\hat{v}_{m+1}| \geq \sqrt{\epsilon}$ when

$$|\hat{\beta}_{m+1}\eta_{m,i}^{(m)}| \leq \sqrt{\epsilon}\|A\|\bar{\Gamma}_k^2, \tag{6.20}$$

so we will purge $\hat{v}_{m+1}$ of the *threshold* Ritz vectors corresponding to indices

$$\mathcal{L}(m+1) = \{i : |\hat{\beta}_{m+1}\eta_{m,i}^{(m)}| \leq \sqrt{\epsilon}\|A\|\bar{\Gamma}_k^2\}.$$

This indicates that in order to maintain desirable numerical properties, an implementation of CA-Lanczos with selective reorthogonalization should perform, in iteration m, the reorthogonalization of $\hat{v}_{m+1}$ against all vectors $z_i^{(m)}$ for which $|\hat{\beta}_{m+1}\eta_{m,i}^{(m)}| \leq \sqrt{\epsilon}\|A\|\bar{\Gamma}_k^2$. Performing reorthogonalization incurs communication in the CA-Lanczos method as it does in the classical Lanczos method. Additionally, if reorthogonalization is triggered in the CA-Lanczos method, we must abandon the current outer loop iteration and recompute the s-step bases with the new reorthogonalized $\hat{v}_{m+1}$ vector. An example of a CA-Lanczos method with selective orthogonalization in given in Algorithm 32.

As mentioned before, for the classical method, it is often the case that $|\mathcal{L}(m+1)| = 0$. We expect the number of reorthogonalizations in the CA-Lanczos method to be similar to the classical Lanczos method when $\bar{\Gamma}_k \approx 1$. In the case then that the number of reorthogonalization steps required in CA-Lanczos is small, we can still reap the performance benefits of avoiding communication in most outer loop iterations. However, if $\bar{\Gamma}_k$ is very large (around $\epsilon^{-1/2}$), then we could have $|\mathcal{L}(m+1)| \approx m$ for all values of m. This would result in what is essentially, in terms of communication and computation cost, classical Lanczos with full reorthogonalization. This happens because now, we can lose orthogonality both because a Ritz pair has converged or because of very large roundoff errors caused by ill-conditioned s-step bases.

There are some potential solutions. One option is to use better polynomial bases in the basis construction to reduce $\bar{\Gamma}_k$. Another idea is to allow the parameter s to vary, and in each outer loop k terminating basis construction at some s such that Γ_k remains below some chosen threshold. As the term $\bar{\Gamma}_k$ tends to overestimate the actual error in many cases, it could be less expensive and still effective to artificially reduce the right-hand side of the condition (6.20). Determining the viability of these methods in practice remains future work.

We note that it is not safe to assume $\bar{\Gamma}_k \approx 1$ and simply use the same reorthogonalization condition as classical Lanczos, i.e, $|\hat{\beta}_{m+1}\eta_{m,i}^{(m)}| \leq \sqrt{\epsilon}\|A\|$. The technical report [92] tests various reorthogonalization strategies including selective orthogonalization for CA-Lanczos. In this work, the same condition that triggers selective reorthogonalization as in the classical Lanczos method is used for the CA-Lanczos method. As can be seen in Table 1 in [92], using this criteria, which doesn't take into account loss of orthogonality due to the conditioning of the s-step bases, the effectiveness of selective orthogonalization decreases as s (and thus $\bar{\Gamma}_k$) increases. Our analysis supports this result.

We note that the $|\hat{\beta}_{m+1}\eta_{m,i}^{(m)}|$ values can be computed in CA-Lanczos the same way as in the classical method. One way of doing this is to calculate all the Ritz values at each step and use a formula from Paige [138] which states that $\left(\eta_{m,i}^{(m)}\right)^2 = \chi_{m-1}(\mu_i^{(m)})/\chi'_m(\mu_i^{(m)})$ where $\chi_m(\mu)$ is the characteristic polynomial of $\hat{T}_m$ (see (5.71)). Another option is to compute a few eigenvalues of $\hat{T}_m$ at each end of the spectrum and then use inverse iteration to find the bottom elements of the corresponding eigenvectors. Recall that in a parallel communication-avoiding implementation, all processors redundantly compute the scalars that form the entries of $\hat{T}_m$, which could be stored locally if m is not too large. In this case, both options above could be implemented without increasing communication.

We note that it may also be possible to implement partial reorthogonalization in CA-KSMs. This technique, due to Simon [157], would involve monitoring of the quantity $\|\hat{V}_{sk+j}^T \hat{v}_{sk+j+1}\|$ where $\hat{V}_{sk+j} = [\hat{v}_1, \ldots, \hat{v}_{sk+j}]$. When semiorthogonality is lost, i.e., this quantity grows to $\sqrt{\epsilon}$, $\hat{v}_{sk+j}$ and $\hat{v}_{sk+j+1}$ are explicitly orthogonalized against all preceding $\hat{v}$ vectors. We leave pursuit of a communication-avoiding version of partial reorthogonalization as future work.

6.4 Look-Ahead

Look-ahead techniques may allow nonsymmetric Lanczos and nonsymmetric Lanczos-type solvers, including BICG and BICGSTAB, to continue past a Lanczos breakdown [148]. A Lanczos breakdown occurs when a vector in the expanding Krylov subspace for A is nearly numerically orthogonal to a vector in the corresponding subspace for A^H. Kim and Chronopoulos [110] have noted that for their s-step nonsymmetric Lanczos algorithm, in exact arithmetic, if breakdown occurs at iteration m in the classical algorithm, then the same breakdown will only occur in the s-step variant if m is a multiple of s. They also comment that look-ahead in the exact arithmetic s-step method amounts to ensuring that no blocked inner products between basis matrices have determinant zero. Hoemmen, however, has commented that the situation is more complicated in finite precision, as it is difficult to determine which type of breakdown occurred [102].

We consider the look-ahead nonsymmetric Lanczos algorithm of Parlett, Taylor, and Liu [148]. Using notation from our CA-BIOC nonsymmetric Lanczos method (see Section 4.1), implementation of the criteria to determine whether the current iteration will be a single step or a double step requires the inner products

$$\begin{aligned}
y_i^H y_i &= y_{k,j}'^H G_k^{(\mathcal{Z})} y_{k,j}', \\
y_{i+1}^H y_{i+1} &= y_{k,j+1}'^H G_k^{(\mathcal{Z})} y_{k,j+1}', \\
\tilde{y}_i^H \tilde{y}_i &= \tilde{y}_{k,j}'^H G_k^{(\tilde{\mathcal{Z}})} \tilde{y}_{k,j}', \\
\tilde{y}_{i+1}^H \tilde{y}_{i+1} &= \tilde{y}_{k,j+1}'^H G_k^{(\tilde{\mathcal{Z}})} \tilde{y}_{k,j+1}', \\
\tilde{y}_i^H \tilde{y}_{i+1} &= \tilde{y}_{k,j}'^H G_k^{(\tilde{\mathcal{Z}})} \tilde{y}_{k,j+1}', \\
y_i^H y_{i+1} &= y_{k,j}'^H G_k^{(\mathcal{Z})} y_{k,j+1}' \quad \text{and} \\
\tilde{y}_{i+1}^H y_i &= \tilde{y}_{k,j+1}'^H G_k y_{k,j}',
\end{aligned}$$

where $G_k = \tilde{\mathcal{Z}}_k^H \mathcal{Z}_k$ (see (4.22)), and $G_k^{(\mathcal{Z})} = \mathcal{Z}_k^H \mathcal{Z}_k$ and $G_k^{(\tilde{\mathcal{Z}})} = \tilde{\mathcal{Z}}_k^H \tilde{\mathcal{Z}}_k$ (see (4.25)). Thus we claim that performing look-ahead checks in CA-BIOC could be performed without requiring additional communication.

The only difficulty we see is that a double look-ahead step can not be performed on the last inner iteration, since $\mathcal{Z}_k$ and $\tilde{\mathcal{Z}}_k$ do not span the required subspaces to perform the update; we would need to increase the subspace by another dimension, i.e., perform another

Algorithm 32 CA-Lanczos with Selective Orthogonalization

Input: n-by-n real symmetric matrix A and length-n starting vector v_1 such that $\|v_1\|_2 = 1$
Output: Matrices V_{sk+s} and T_{sk+s} and vector v_{sk+s+1} satisfying (5.1)

$u_1 = Av_1$
for $k = 0, 1, \ldots$ until convergence **do**
  Compute $\mathcal{Y}_k$ with change of basis matrix $\mathcal{B}_k$
  Compute $G_k = \mathcal{Y}_k^T \mathcal{Y}_k$
  $v'_{k,1} = e_1$
  if $k = 0$ **then**
    $u'_{0,1} = \mathcal{B}_0 e_1$
  else
    $u'_{k,1} = e_{s+2}$
  end if
  for $j = 1, 2, \ldots, s$ **do**
    $\alpha_{sk+j} = v'^T_{k,j} G_k u'_{k,j}$
    $w'_{k,j} = u'_{k,j} - \alpha_{sk+j} v'_{k,j}$
    $\beta_{sk+j+1} = (w'^T_{k,j} G_k w'_{k,j})^{1/2}$
    if $\beta_{sk+j+1} = 0$, quit
    Find $\mathcal{L}(sk+j+1) = \{i : |\hat{\beta}_{sk+j+1} \eta^{(sk+j)}_{sk+j,i}| \leq \sqrt{\epsilon} \|A\| \bar{\Gamma}^2_k\}$
    if $|\mathcal{L}(sk+j+1)| > 0$ **then**
      $w_{sk+j} = \mathcal{Y}_k w'_{k,j}$
      for $\ell \in \mathcal{L}(m+1)$ **do**
        $w_{sk+j} = w_{sk+j} - (z^{(sk+j)T}_\ell w_{sk+j}) z^{(sk+j)}_\ell$
      end for
      $\beta_{sk+j+1} = \|w_{sk+j}\|_2$
      if $\beta_{sk+j+1} = 0$, quit
      $v_{sk+j+1} = w_{sk+j} / \beta_{sk+j+1}$
      $u_{sk+j+1} = Av_{sk+j+1} - \beta_{sk+j+1} \mathcal{Y}_k v'_{k,j}$
      Compute eigenvalues, eigenvectors, and error bounds of T_m
      Break and immediately begin next outer loop iteration
    end if
    $v'_{k,j+1} = w'_{k,j} / \beta_{sk+j+1}$
    $u'_{k,j+1} = \mathcal{B}_k v'_{k,j+1} - \beta_{sk+j+1} v'_{k,j}$
    Compute eigenvalues, eigenvectors, and error bounds of T_m
  end for
  $[v_{sk+2}, \ldots, v_{sk+s+1}] = \mathcal{Y}_k [v'_{k,2}, \ldots, v'_{k,s+1}]$
  $[u_{sk+2}, \ldots, u_{sk+s+1}] = \mathcal{Y}_k [u'_{k,2}, \ldots, u'_{k,s+1}]$
end for

multiplication by A, which is not possible using the current bases. One solution is that, if a double iteration is required on the last inner loop iteration, when $j = s$, we could abandon the current inner loop, skipping the last iteration, begin a new outer loop, and make the first step of the new inner iterations a double step. This may result in a small amount of wasted effort.

The situation is more complicated when pivots larger than 2×2 are used, i.e., we look ahead more than one step at a time. If the algorithm requires that we look ahead ℓ steps to compute the next vectors, then we can only do this in the CA-BIOC algorithm for iterations $j \leq s - \ell$. In the s-step case we are then limited to using pivots of size at most $s \times s$ unless we are willing to increase the dimension of the bases we generate in each outer loop. We leave further inquiry as future work.

6.5 Extended Precision

We stress that in infinite precision arithmetic, the CA-KSMs will produce exactly the same results as their classical counterparts, regardless of s value and regardless of polynomial bases used. This means that if we had the ability to work in arbitrary precision at little extra cost, we would be free to choose the s value based solely on matrix structure and machine architecture without worrying about numerical stability and convergence. In many cases, it may be that we can achieve these desirable results using less than infinite precision, e.g., quad precision. This would result in (at least) a doubling of the bandwidth and arithmetic cost, which might be insignificant in the latency-bound regime. We therefore seek to determine the precision needed to ensure that CA-KSMs will behave numerically like classical KSMs. If this precision can be estimated, then we can model the effect this extra precision has on performance and find the best tradeoff point.

The analysis of Section 5.1 suggests that if we use precision ϵ such that

$$\epsilon < \frac{1}{24(n + 11s + 15)\bar{\Gamma}_k^2},$$

then this will guarantee that the results of Paige apply to the s-step case. Since the bounds developed in Section 5.1 are not tight, the expression above is likely a large overestimate of the actual precision needed to achieve the desired numerical properties. This likely extends to other Lanczos-based CA-KSMs, since the computations are very similar. We note that the maximum Γ_k could be estimated using bounds on the condition numbers for various polynomial bases (see, e.g., [74, 149]).

The analysis in Section 5.1 also suggests that it is possible to use different precision for different parts of the computation. In Theorem 1, we can see that the quantity ϵ_0 is amplified by a factor of $\bar{\Gamma}_k^2$, but ϵ_1 is only amplified by a factor of $\bar{\Gamma}_k$. Therefore we can achieve the same effect numerically by using precision proportional to $\epsilon/\bar{\Gamma}_k^2$ for computations that contribute to ϵ_0 and precision proportional to $\epsilon/\bar{\Gamma}_k$ for computations that contribute to ϵ_1. To this end, future work includes extension of our numerical stability analysis for

CA-KSMs in Section 5.1 to estimate the minimum number of bits of precision needed in each kernel.

We mention that the use of extended (or limited) precision capabilities in software and hardware as a means to achieve algorithm stability while minimizing energy/bandwidth requirements is an active area of research. Specifically in the realm of CA-KSMs, Yamazaki et al. [196] have recently used a mixed-precision CholeskyQR routine to perform the TSQR factorization in a higher precision during select iterations of CA-GMRES.

6.6 Dynamically Updated Basis Parameters

As we have seen in Chapter 5, generating well-conditioned s-step bases is essential to maintaining desirable numerical properties in CA-KSMs. Determining basis parameters that will result in well-conditioned bases requires some knowledge of the spectrum of A. In practice, we might want to avoid the upfront cost of computing good estimates for eigenvalues of A to use for this purpose. One technique is to start the iterations by using the classical method or the CA method with a monomial basis (or a Newton or Chebyshev basis constructed with rough estimates) and a smaller s value, and then use information from the iterations to compute Leja points or estimate the spectral radius.

In our numerical experiments in Section 4, we have used the approach described by Philippe and Reichel [149] for constructing sets $\mathbb{S}$ from which to compute basis parameters. We first run i steps of the algorithm to find the matrix tridiagonal Lanczos matrix T_i consisting of scalars computed during the iterations, where i should be at least s ($2s$ is a common choice). Note that the first i steps can either be performed as classical iterations or as communication-avoiding iterations with either the monomial basis or another polynomial basis constructed with initial spectral estimates.

For the Chebyshev basis, after the first i steps, we set $\mathbb{S} = \text{co}(\lambda(T_i))$, the convex hull of $\lambda(T_i)$. After each outer loop iteration, one could then determine the new set of Ritz values of the now larger matrix T_{i+sk} and update $\mathbb{S} = \text{co}(\mathbb{S} \cup \lambda(T_{i+sk}))$. After the new set $\mathbb{S}$ is computed, we find the parameters of a bounding ellipse for $\mathbb{S}$, and, as described in Section 3.2.5, use these in the construction of the Chebyshev basis for the next s steps.

For construction of the Newton basis, we initialize $\mathbb{S} = \text{sp}(\lambda(T_i))$, where $\text{sp}(\lambda(T_i))$ is the convex set formed by the spokes from the centroid of $\lambda(T_i)$ to the eigenvalues $\lambda(T_i)$. Philippe and Reichel term this a "spoke set" [149]. After each outer loop, one could again determine the new set of Ritz values of the now larger matrix T_{i+sk} and update $\mathbb{S} = \mathbb{S} \cup \text{sp}(\lambda(T_{i+sk}))$.

We note that in the case that A is symmetric, $\mathbb{S}$ will be a set of points on the real line between $\lambda_1(T_i)$ and $\lambda_n(T_i)$ for both the convex hull approach and the spoke set approach. In these cases, we can simply use Leja's approach to computing Leja points on an interval on the real line. The convex hull and spoke set approaches become more important for constructing well-conditioned bases in the case that A has imaginary eigenvalues.

We also stress that except in very rare cases, we do not need very accurate Ritz values to construct well-conditioned bases. Therefore we likely only need to refine the basis parameters

after a few outer loop iterations. Determining analytically and experimentally when and how often to update basis parameters is left to future work.

6.7 Variable Basis Size

We have seen throughout Chapter 5 that the condition number of the s-step bases generated in each outer loop in CA-KSMs plays a large part in determining how their finite precision behavior differs from that of the classical method. This motivates approaches for ensuring that the s-step bases do not become too ill-conditioned in any outer loop. While using more well-conditioned polynomials for basis construction generally improves the situation, the resulting basis matrix condition number can still be too high, depending on s and the spectrum of A.

We show the CA-Lanczos method which allows for variable sizes bases in Algorithm 33. We note that this general approach could be used in any of the CA-KSMs, both those for solving eigenproblems and linear systems. In Algorithm 33, the choice of the parameter tol depends on the desired numerical behavior. Theorem 1 and (5.65) suggest that we should use tol $\approx (n\epsilon)^{-1/2}$. Depending on the application constraints, one might choose to sacrifice convergence rate for greater communication savings, or vice versa; in this case tol can be adjusted accordingly.

We note that an implementation of CA-KSMs with variable basis sizes may end up wasting work in the matrix powers kernel computation. Because vector entries required for computing the needed basis for $s_{\max}$ are exchanged with neighboring subdomains before computation begins, if we end up cutting of the computation at $s_k < s_{\max}$, we will have moved more data and performed more flops than necessary. This could be remedied during the iteration by lowering $s_{\max}$ if many successive outer loops do not use the full basis.

6.7.1 Telescoping Basis Size

Even if the chosen $s_{\max}$ is such that $\kappa(\mathcal{Y}_k) <$ tol for all k, it may still be beneficial to vary the block size s used in each outer loop for some applications. One example is in the case where we are using a CA-KSM to solve a series of linear systems, with some being much harder to solve than others in terms of the number of iterations they take. If the easiest solves take fewer than s iterations to converge, then we've wasted effort in computing the s-step bases and Gram matrix for iterations we don't execute.

This inspires the use of a "telescoping" s value, where we begin using $s = 1$ and gradually increase s in each outer loop up to $s_{\max}$ (or the largest $s_k \leq s_{\max}$ such that $\kappa(\mathcal{Y}_k) <$ tol holds). In this way, we avoid the overhead of using CA-KSMs for easier solves, and for harder solves, the cost of basis construction and block orthogonalization are amortized over many outer loops and asymptotic savings can be realized. This telescoping s approach has been implemented and proven to be beneficial when CA-KSMs are used as bottom solve routines for Geometric Multigrid methods; see Section 8.3.6.

Algorithm 33 CA-Lanczos with Variable Basis Size

Input: n-by-n real symmetric matrix A and length-n starting vector v_1 such that $\|v_1\|_2 = 1$, integer $s_{\max} > 0$, max basis condition number tol

Output: Matrices V_{sk+s} and T_{sk+s} and vector v_{sk+s+1} satisfying (5.1)

1: $u_1 = Av_1$
2: $\ell = 0$
3: **for** $k = 0, 1, \ldots$ until convergence **do**
4: Compute up to $2s_{\max}+2$ columns of $\mathcal{Y}_k$ with condition estimation; stop when $\kappa(\mathcal{Y}_k) \geq$ tol ($\mathcal{Y}_k$ will have $2s_k + 2$ columns)
5: Compute $G_k = \mathcal{Y}_k^T \mathcal{Y}_k$
6: $v'_{k,1} = e_1$
7: **if** $k = 0$ **then**
8: $u'_{0,1} = \mathcal{B}_0 e_1$
9: **else**
10: $u'_{k,1} = e_{s_k+2}$
11: **end if**
12: **for** $j = 1, 2, \ldots, s_k$ **do**
13: $\alpha_{\ell+j} = v'^T_{k,j} G_k u'_{k,j}$
14: $w'_{k,j} = u'_{k,j} - \alpha_{sk+j} v'_{k,j}$
15: $\beta_{\ell+j+1} = (w'^T_{k,j} G_k w'_{k,j})^{1/2}$
16: $v'_{k,j+1} = w'_{k,j} / \beta_{\ell+j+1}$
17: $v_{\ell+j+1} = \mathcal{Y}_k v'_{k,j+1}$
18: $u'_{k,j+1} = \mathcal{B}_k v'_{k,j+1} - \beta_{\ell+j+1} v'_{k,j}$
19: $u_{\ell+j+1} = \mathcal{Y}_k u'_{k,j+1}$
20: **end for**
21: $\ell = \ell + s_k$
22: **end for**

6.8 Preconditioning: A Discussion

Although our primary focus is on developing a practical understanding of the behavior of communication-avoiding Krylov methods in finite precision, a thesis on Krylov subspace methods would be incomplete without a discussion of preconditioning. The goal of communication-avoiding Krylov methods is to decrease the cost per iteration. Conversely, the goal of preconditioning is to decrease the total number of iterations required until convergence. These are thus two different, but potentially complementary approaches to improving overall runtime.

The basic idea behind preconditioning is to, instead running the KSM on system $Ax = b$, instead run it on a modified system that is easier to solve, such as $M^{-1}Ax = M^{-1}b$, where M is called the *preconditioner*. Note that this does not require explicitly forming $M^{-1}A$, but requires instead solving, e.g., $Mz = r$ when needed. The example given, where

M^{-1} is applied on the left, is called left preconditioning. It is also possible to use right preconditioning, where one runs the KSM on the system $AM^{-1}u = b$ and then solves $Mx = u$, or split preconditioning, where, given factored $M = M_L M_R$, one runs the KSM on the system $M_L^{-1} A M_R^{-1} u = M_L^{-1} b$ and then solves $M_R x = u$.

Selecting a good preconditioner to use, i.e., one which is inexpensive to apply and significantly increases the convergence rate, is very often problem-dependent and is considered somewhat of an art. There is a large space of research in designing efficient and effective preconditioners, which we do not hope to tackle here. In a very general sense, a good preconditioner is one where $MA \approx I$ and application of M^{-1} to a vector is inexpensive.

For CA-KSMs, the situation is even more complicated. CA-KSMs avoid communication in the matrix powers computation by exploiting locality which is made possible by sparsity in A. Even is A is very sparse, its inverse is in general dense. Thus selecting $M \approx A^{-1}$ will serve to improve the convergence rate but will introduce dependencies in the matrix powers computation which destroy locality. Thus the two goals - reducing the cost per iteration via communication-avoiding techniques and reducing the number of iterations via preconditioning - are somewhat at odds and involve complicated, problem-dependent tradeoffs.

There has nevertheless been success in designing communication-avoiding preconditioners which do not hinder asymptotic communication savings for specific cases. Preconditioned variants of CA-KSMs and potential preconditioners that can be used with them are discussed in the thesis of Hoemmen [102]. Grigori et al. developed a CA-ILU(0) preconditioner for CA-GMRES [89]. For structured problems, their method exploits a novel mesh ordering to obtain triangular factors that can be applied with less communication. There is also recent work in developing a new "underlapping" technique (as opposed to the more common "overlapping") in communication-avoiding domain decomposition preconditioners for CA-KSMs [195]. For the case of preconditioners with both sparse and low-rank components (e.g., hierarchical semiseparable matrices; see, e.g., [19]), applying the low-rank components dominates the communication cost; techniques in [102, 112] block together several applications of the low-rank components in order to amortize communication cost over several KSM iterations. Our communication-avoiding deflation technique in Section 6.2 can be viewed as preconditioning, but with a singular preconditioner. We also note that there has been recent work in developing a high-performance deflated "pipelined" conjugate gradient method [80].

We note that, although developing communication-avoiding preconditioned Krylov methods is challenging, there are many applications for which diagonal preconditioning, which incurs no additional communication, is perfectly sufficient. Further, although preconditioners are often required for KSMs for solving linear systems, preconditioners are typically not used in KSMs for solving eigenvalue problems (aside from polynomial preconditioning and shift-and-invert strategies). Thus we do not see the lack of available communication-avoiding preconditioners as prohibitive to the usability of CA-KSMs for solving eigenvalue problems.

6.8.1 Future Work

As there has been a paradigm shift towards the redesign of algorithms to avoid communication, there is also a growing trend toward the redesign of preconditioners with communication-avoidance in mind (e.g., the underlapping technique of [195]). We expect that the development of new preconditioners that avoid communication as well as modification of existing preconditioners to eliminate communication will be a fruitful area of research in coming years.

The use of variable and flexible preconditioners provides great opportunity for further optimization of communication-avoiding Krylov methods in terms of convergence rate and speed per iteration. One possibility is that the blocking parameter s can be made variable, which would allow for a few select expensive preconditioned iterations (when $s = 1$) while the majority of the iterations remain communication- avoiding (when $s > 1$). This could allow a relatively inexpensive reduction in convergence rate, leading to faster overall solves. An alternative to developing preconditioners for CA-KSMs is to use CA-KSMs themselves as inexpensive flexible inner-outer preconditioners inside classical KSMs. Further investigation remains future work.

6.9 Conclusions and Future Work

In this Chapter, we have developed a number of new techniques for improving the stability and convergence properties of CA-KSMs. Some of these techniques are extensions of techniques that were invented for classical KSMs, whereas others are new ideas applicable specifically to CA-KSMs. There is potential to develop techniques for improving numerical properties in CA-KSMs other than those discussed here. These might include restarting, other types of reorthogonalization, use of the TSQR kernel (see, e.g., [102]) for orthogonalization of the s-step bases, and the development of new preconditioners that avoid communication as well as modification of existing preconditioners to eliminate communication will be a fruitful area of research in coming years as discussed in Section 6.8.1, among others.

While the techniques discussed in this Chapter are compatible with the communication-avoiding approach (meaning we can still achieve asymptotic communication savings), the potential numerical improvements do not come for free. All these techniques will incur an additional (albeit lower order) cost in terms of either computation, bandwidth, latency, or some combination of these. Future work includes high-performance implementation of these techniques and evaluation of tradeoffs between the convergence rate and speed per iteration in the resulting method for specific applications.

In the next chapter, we turn our focus from numerical optimizations to performance optimizations based on matrix structure.

Chapter 7

Optimizations for Matrices with Special Structure

In this Chapter we present some performance optimizations that are applicable to matrices with specific nonzero structures. In Section 7.1, which has been adapted from [112], we derive a new parallel communication-avoiding matrix powers algorithm for matrices of the form $A = D+USV^H$, where D is sparse and USV^H has low rank and is possibly dense. There are many practical situations where such structures arise, including power-law graph analysis and circuit simulation. Hierarchical ($\mathcal{H}$-) matrices (e.g., [19]), common preconditioners for Krylov subspace methods, also have this form. Our primary motivation is enabling preconditioned communication-avoiding Krylov subspace methods, where the preconditioned system has hierarchical semiseparable (HSS) structure. We demonstrate that, with respect to the cost of computing k sparse matrix-vector multiplications, our algorithm asymptotically reduces the parallel latency by a factor of $O(k)$ for small additional bandwidth and computation costs. To highlight the potential benefits and tradeoffs of this approach, in Section 7.1.4 we model the performance of our communication-avoiding matrix powers computation for HSS matrices. Using problems from real-world applications, our performance model predicts up to 13× speedups on petascale machines.

In Section 7.2 we present the 'streaming' matrix powers optimization which can improve sequential performance for implicitly-represented matrices (which include stencils and stencil-like matrices). By interleaving the matrix powers computation with construction of the Gram matrix in each outer loop iteration, the data movement cost is reduced from $O(sn)$ to $O(n)$ per s steps. Write-avoiding algorithms have recently become of interest in the context of non-volatile memory [33]; use of the streaming matrix powers optimization enables write-avoiding Krylov methods for general sparse matrices.

In Section 7.3, we improve on the method of partitioning highly nonsymmetric matrices to avoid communication in computing the matrix powers kernel. The hypergraph partitioning problem has been shown to correspond exactly to minimizing the communication in SpMV [37]. To encapsulate the communication cost of k SpMVs, however, we must look at the $s-$level hyperedges, or the hyperedges corresponding to the structure of $|A|^k$.

However, computing the structure of $|A|^k$ is prohibitively expensive. To reduce this cost, we implement Cohen's algorithm for estimating the size of s steps of the transitive closure [54]. Using the approximated size of each hyperedge, we devise a dropping technique, which selectively drops hyperedges once they become too large. This can significantly reduce both the cost of building the $k-$level hyperedges and the partitioning time while still finding a near-optimal (in terms of communication) partition.

7.1 Data-Sparse Matrix Powers Kernel

Recall from Section 3.2 that given a matrix A, a vector y, and desired dimension $k+1$, we can compute a basis for the Krylov subspace $\mathcal{K}_{k+1}(A, y)$,

$$\mathcal{Y} = [\rho_0(A)y, \rho_1(A)y, \ldots, \rho_k(A)y], \tag{7.1}$$

where ρ_j is a polynomial of degree j, using the communication-avoiding matrix powers algorithms of [63]. The authors in [63] show that the communication-avoiding approach gives an $O(k)$ reduction in parallel latency cost versus computing k repeated SpMVs for a set number of iterations. This improvement is only possible if A is *well partitioned*, i.e., we can partition A such that for each processor, computing powers up to A^k involves communication only between $O(1)$ nearest neighbors. (Note that in this section, k denotes the number of SpMVs to compute rather than an outer iteration index in a CA-KSM.)

Although such advances show promising speedups for many problems, the requirement that A is well partitioned often excludes matrices with dense components, even if those components have low rank (or, data sparsity). In this Section, we derive a new parallel communication-avoiding matrix powers algorithm for matrices of the form $A = D + USV^H$, where D is well partitioned and USV^H may not be well partitioned but has low rank. Recall $y^H = \overline{y}^T$ denotes the Hermitian transpose of y. There are many practical situations where such structures arise, including power-law graph analysis and circuit simulation. Hierarchical ($\mathcal{H}$-) matrices (see, e.g., [19]), common preconditioners for Krylov subspace methods, also have this form. Our primary motivation is enabling preconditioned communication-avoiding Krylov subspace methods, where the preconditioned system has hierarchical semiseparable (HSS) structure (see, e.g., [19, 44, 185, 193]).

With respect to the cost of computing k SpMVs, our algorithm asymptotically reduces parallel latency by a factor of $O(k)$ with only small additional bandwidth and computation requirements. Using a detailed complexity analysis for an example HSS matrix, our model predicts up to 13× speedups over the standard algorithm on petascale machines.

Our approach is based on the application of a blocking covers technique [117] to communication-avoiding matrix powers algorithms described in [126] (see Section 3.2). We briefly review this work below. We note that this section has been adapted from work that first appeared in [112].

7.1.1 The Blocking Covers Technique

Hong and Kung [103] prove a lower bound on data movement for a sequential matrix powers computation on a regular mesh. Given directed graph $G = (V, E)$ representing nonzeros of A, vertex $v \in V$, and constant $k \geq 0$, let the *k-neighborhood* of v, $N^{(k)}(v)$, be the set of vertices in V such that $u \in N^{(k)}(v)$ implies there is a path of length at most k from u to v. For example, for a tridiagonal matrix with vertices $V = \{v_1, \ldots, v_n\}$, the k-neighborhood cover of a vertex v_i is the set $N^{(k)}(v_i) = \{v_{i-k} \ldots, v_i, \ldots, v_{i+k}\} \cap V$. A *k-neighborhood-cover* of G is a sequence of subgraphs $\mathcal{G} = \{G_i = (V_i, E_i)\}_{i=1}^{\ell}$, such that $\forall v \in V, \exists G_i \in \mathcal{G}$ for which $N^{(k)}(v) \subseteq V_i$ [117]. If G has a k-neighborhood cover with $O(|E|/M)$ subgraphs, each with $O(M)$ edges where M is the size of the primary memory, Hong and Kung's method reduces data movement by a factor of k over computing (7.1) column-wise. A matrix that meets these constraints is also frequently called *well partitioned* [63] (we use this terminology for the parallel case as well).

A shortcoming of Hong and Kung's method is that certain graphs with low diameter (e.g., multigrid graphs) may not have k-neighborhood covers that satisfy these memory constraints. Leiserson et al. overcome this restriction by "removing" a set $B \subseteq V$ of *blocker* vertices, chosen such that the remaining graph $V - B$ is well partitioned [117]. Let the k-neighborhood with respect to B be defined as $N_B^{(k)}(v) = \{u \in V : \exists \text{ path } u \to u_1 \to \cdots \to u_t \to v$, where $u_i \in V - B$ for $i \in \{1, \ldots, t < k\}\}$. Then a (k, r, M)-blocking cover of G is a pair $(\mathcal{G}, \mathcal{B})$, where $\mathcal{G} = \{G_i = (V_i, E_i)\}_{i=1}^{\ell}$, and $\mathcal{B} = \{B_i\}_{i=1}^{\ell}$ is a sequence of subsets of V such that: (1) $\forall i \in \{1, \ldots, \ell\}, M/2 \leq |E_i| \leq M$, (2) $\forall i \in \{1, \ldots, \ell\}, |B_i| \leq r$, (3)$\sum_{i=1}^{\ell} |E_i| = O(|E|)$, and (4) $\forall v \in V, \exists G_i \in \mathcal{G}$ such that $N_{B_i}^{(k)}(v) \subseteq V_i$ [117]. Leiserson et al. present a 4 phase sequential matrix powers algorithm that reduces the data movement by a factor of k over the standard method if the graph of A has a (k, r, M)-blocking cover that meets certain criteria. Our parallel algorithm is based on a similar approach. Our work generalizes the blocking covers approach [117], both to the parallel case and to a larger class of data-sparse matrix representations.

7.1.2 Derivation of Parallel Blocking Covers Algorithm

Recall the classical (PA0) and communication-avoiding (PA1) algorithms for computing (7.1) in Section 3.2. We will use the same notation for the layered graph and for the parallel matrix powers computation, which are defined in Sections 3.2.1 and 3.2.2, respectively. In the case that A is not well partitioned, PA0 (Algorithm 1) must communicate at every step, but now the cost of PA1 (Algorithm 2) may be much worse: when $k > 1$, every processor needs all n rows of A and $y^{(0)}$, and so there is no parallelism in computing all but the last SpMV operation. Note that when $k = 1$, PA1 degenerates to PA0.

If, however, A can be split in the form $D + USV^H$, where D is well partitioned, and U and V are $n \times r$ and S is $r \times r$ with $r \ll n$ so USV^H has low rank, we can use a generalization of the blocking covers approach [117] to recover parallelism. In this case, D has a good cover and US can be applied locally, but the application of V^H incurs global

communication. Thus, the application of V^H will correspond to the blocker vertices in our algorithm, PA1-BC, which we now derive.

We assume the polynomials ρ_j in (7.1) satisfy the three-term recurrence (4.1), i.e.,

$$\begin{aligned} &\rho_0(z) = 1, \quad \rho_1(z) = (z - \hat{\alpha}_0)\rho_0(z)/\hat{\gamma}_0, \quad \text{and} \\ &\rho_j(z) = ((z - \hat{\alpha}_{j-1})\rho_{j-1}(z) - \hat{\beta}_{j-2}\rho_{j-2}(z))/\hat{\gamma}_{j-1} \quad \text{for } j > 1. \end{aligned} \tag{7.2}$$

It is convenient to represent the polynomials $\{\rho_j\}_{j=0}^k$ by their coefficients, stored in a $(k+1)$-by-$(k+1)$ tridiagonal matrix $\mathcal{B}$ of the form (4.2) with $i = k$.

Using our splitting $A = D + USV^H$, we can write

$$y^{(j)} = ((D - \hat{\alpha}_k)y^{(j-1)} - \hat{\beta}_{j-2}y^{(j-2)} + USV^H y^{(j-1)})/\hat{\gamma}_{j-1}. \tag{7.3}$$

We obtain the following identity.

Lemma 7. *Given the additive splitting $z = z_1 + z_2$, where z_1 and z_2 need not commute,* (7.2) *can be rewritten as*

$$\begin{aligned} &\rho_0(z) = \rho_0(z_1), \quad \rho_1(z) = \rho_1(z_1) + z_2\rho_0(z)/\hat{\gamma}_0, \quad \textit{and for } j > 1, \\ &\rho_j(z) = \rho_j(z_1) + \sum\nolimits_{i=1}^{j} \rho_{i-1}^{j-i+1}(z_1)z_2\rho_{j-i}(z)/\hat{\gamma}_{j-i}. \end{aligned} \tag{7.4}$$

where $\rho_j^i(z)$ is polynomial of degree j related to $\rho_j(z)$ by reindexing the coefficients

$$(\hat{\alpha}_j, \hat{\beta}_j, \hat{\gamma}_j) \equiv (\hat{\alpha}_{i+j}, \hat{\beta}_{i+j}, \hat{\gamma}_{i+j})$$

in (7.2).

The conclusion is established by induction (see [111] for details).

Now substitute $z = A = D + USV^H = z_1 + z_2$ in (7.4), premultiply by SV^H, and postmultiply by y, to obtain

$$SV^H\rho_j(A)y = S\Big(V^H\rho_j(D)y + \sum_{i=1}^{j}(V^H\rho_{i-1}^{j-i+1}(D)U)(SV^H\rho_{j-i}(A)y/\hat{\gamma}_{j-i}\Big). \tag{7.5}$$

Let $W_i = V^H\rho_i(D)U$ for $0 \le i \le k-2$, $x_i = V^H\rho_i(D)x$ for $0 \le i \le k-1$, and $b_j = SV^H y^{(j)}$ for $0 \le i \le k-1$. We can write ρ_i^j in terms of $\rho_i = \rho_i^0$, via the following result.

Lemma 8. *There exist coefficient vectors $w_i^j \in \mathbb{C}^{i+1}$ satisfying*

$$[W_0, \ldots, W_i](w_i^j \otimes I_{r,r}) = V^H\rho_i^j(D)U \tag{7.6}$$

for $0 \le i \le k-2, 1 \le j \le k-i-1$, and they can be computed by the recurrence

$$\begin{aligned} &w_0^j = 1, \quad w_1^j = (\mathcal{B}_{2,1} - \hat{\alpha}_{i+j-1}I_{2,1})w_0^j/\hat{\gamma}_{j-1}, \quad \textit{and for } i > 1, \\ &w_i^j = ((\mathcal{B}_{i+1,i} - \hat{\alpha}_{i+j-1}I_{i+1,i})w_{i-1}^j - \hat{\beta}_{i+j-2}I_{i+1,i-1}w_{i-2}^j)/\hat{\gamma}_{i+j-1}, \end{aligned} \tag{7.7}$$

where $I_{m,n}$ denotes the leading (m,n)-submatrix of the identity and $\mathcal{B}_{m,n}$ denotes the leading (m,n)-submatrix of $\mathcal{B}$ (note that this differs from the meaning of the subscript on $\mathcal{B}$ in previous sections).

Algorithm 34 PA1-BC. Code for processor m.

1: Compute local rows of basis for $\mathcal{K}_{k-1}(D, U)$ with PA1; premultiply by local columns of V^H.
2: Compute $[W_0, \ldots, W_{k-2}]$ by an Allreduce.
3: Compute w_i^j for $0 \leq i \leq k-2$ and $1 \leq j \leq k-i-1$, via (7.7).
4: Compute local rows of basis for $\mathcal{K}_k(D, y^{(0)})$ with PA1; premultiply by local columns of V^H.
5: Compute $[x_0, \ldots, x_{k-1}]$ by an Allreduce.
6: Compute $[b_0, \ldots, b_{k-1}]$ by (7.8).
7: Compute local rows of $[y^{(0)}, \ldots, y^{(k)}]$ with PA1, modified for (7.9).

The conclusion is established by induction (see [111] for details).

Using this result, we write (7.5) as

$$b_j = S\Big(x_j + \sum\nolimits_{i=1}^{j} [W_0, \ldots, W_{i-1}] \cdot (w_{i-1}^{j-i+1} \otimes I_{r,r}) b_{j-i} / \hat{\gamma}_{j-i}\Big). \tag{7.8}$$

Ultimately we must evaluate (7.3), which we rewrite as

$$y^{(j)} = ((D - \hat{\alpha}_{j-1}) y^{(j-1)} - \hat{\beta}_{j-2} y^{(j-2)} + U b_{j-1}) / \hat{\gamma}_{j-1}. \tag{7.9}$$

This can be accomplished by applying PA1 to the following recurrence for polynomials $\rho_j(z, c)$, where $c = \{c_0, \ldots, c_{j-1}, \ldots\} = \{Ub_0, \ldots, Ub_{j-1}, \ldots\}$:

$$\begin{aligned} &\rho_0(z, c) = 1,\ \rho_1(z, c) = ((z - \hat{\alpha}_0)\rho_0(z, c) + c_0)/\hat{\gamma}_0, \quad \text{and for } j > 1, \\ &\rho_j(z, c) = ((z - \hat{\alpha}_{j-1})\rho_{j-1}(z, c) - \hat{\beta}_{j-2}\rho_{j-2}(z, c) + c_{j-1})/\hat{\gamma}_{j-1}. \end{aligned} \tag{7.10}$$

Given the notation established, we construct PA1-BC (Algorithm 34). Let $G = (N, E)$ denote the layered graph for matrix D (see Sections 3.2.1 and 3.2.2). Processor m must own

$$D_{\{i:y_i^{(1)} \in R(N_m)\}},\ U_{\{i:y_i^{(1)} \in R(N_m)\}},\ V_{\{i:y_i^{(j)} \in N_m\}},\ \text{and } R^{(0)}(N_m) = y^{(0)}_{\{i:y_i^{(0)} \in R(N_m)\}},$$

in order to compute the entries $y_i^{(j)} \in N_m$.

In exact arithmetic, PA1-BC returns the same output as PA0 and PA1. However, by exploiting the splitting $A = D + USV^H$, PA1-BC may avoid communication when A is not well partitioned. Communication occurs in calls to PA1 (Lines 1 and 4), as well as in Allreduce collectives (Lines 2 and 5). As computations in Lines 1, 2, and 3 do not depend on the input $y^{(0)}$, they need only be computed once per matrix $A = D + USV^H$, thus we assume their cost is incurred offline.

For the familiar reader, the sequential blocking covers algorithm [117] is a special case of a sequential execution of Algorithm 34, using the monomial basis, where $U = [e_i : i \in \mathcal{I}]$ and $SV^H = A_{\mathcal{I}}$, where e_i is the i-th column of the identity and $\mathcal{I} \subseteq \{1, \ldots, n\}$ are the indices of the blocker vertices. In Algorithm 34, Lines $\{1, 2, 3\}$, $\{4, 5\}$, 6, and 7 correspond to the 4 phases of the sequential blocking covers algorithm, respectively [117].

In the next section, we demonstrate the benefit of our approach on a motivating example, matrix powers with HSS matrix A.

7.1.3 Hierarchical Semiseparable Matrix Example

Hierarchical ($\mathcal{H}$-) matrices (see, e.g., [19]) are amenable to the splitting $A = D + UV^H$, where D is block diagonal and UV^H represents the off-diagonal blocks. Naturally, U and V are quite sparse and it is important to exploit this sparsity in practice. In the special case of HSS matrices, many columns of U and V are linearly dependent, and we can exploit the matrix S in the splitting USV^H to write U and V as block diagonal matrices. We review the HSS notation and the algorithm for computing $v = Ay$ given by Chandrasekaran et al. [43, §2-§3]. For any $0 \leq L \leq \lfloor \log n \rfloor$, where $\log = \log_2$, we can write A hierarchically as a perfect binary tree of depth L by recursively defining its diagonal blocks as $A = D_{0;1}$ and

$$D_{\ell-1;i} = \begin{bmatrix} D_{\ell;2i-1} & U_{\ell;2i-1}B_{\ell;2i-1,2i}V_{\ell;2i}^H \\ U_{\ell;2i}B_{\ell;2i,2i-1}V_{\ell;2i-1}^H & D_{\ell;2i} \end{bmatrix} \tag{7.11}$$

for $1 \leq \ell \leq L$, $1 \leq i \leq 2^{\ell-1}$, where $U_{0;1}, V_{0;1} = []$, and for $\ell \geq 2$,

$$U_{\ell-1;i} = \begin{bmatrix} U_{\ell;2i-1}R_{\ell;2i-1} \\ U_{\ell;2i}R_{\ell;2i} \end{bmatrix}, \qquad V_{\ell-1;i} = \begin{bmatrix} V_{\ell;2i-1}W_{\ell;2i-1} \\ V_{\ell;2i}W_{\ell;2i} \end{bmatrix}; \tag{7.12}$$

the subscript expression $\ell; i$ denotes vertex i of the 2^ℓ vertices at level ℓ.

The action of A on y, i.e., $v = Ay$, satisfies $v_{0;1} = D_{0;1}y_{0;1}$, and for $1 \leq \ell \leq L$, $1 \leq i \leq 2^\ell$, satisfies $v_{\ell;i} = D_{\ell;i}y_{\ell;i} + U_{\ell;i}f_{\ell;i}$, with $f_{1;1} = B_{1;1;2}g_{1;2}$, $f_{1;2} = B_{1;2;1}g_{1;1}$, and, for $1 \leq \ell \leq L-1$, $1 \leq i \leq 2^\ell$,

$$f_{\ell+1;2i-1} = \begin{bmatrix} R_{\ell+1;2i-1}^T \\ B_{\ell+1;2i-1,2i}^T \end{bmatrix}^T \begin{bmatrix} f_{\ell;i} \\ g_{\ell+1;2i} \end{bmatrix}, \; f_{\ell+1;2i} = \begin{bmatrix} R_{\ell+1;2i}^T \\ B_{\ell+1;2i,2i-1}^T \end{bmatrix}^T \begin{bmatrix} f_{\ell;i} \\ g_{\ell+1;2i-1} \end{bmatrix}, \tag{7.13}$$

where, for $1 \leq \ell \leq L-1$, $1 \leq i \leq 2^\ell$, $g_{\ell;i} = \begin{bmatrix} W_{\ell+1;2i-1} \\ W_{\ell+1;2i} \end{bmatrix}^H \begin{bmatrix} g_{\ell+1;2i-1} \\ g_{\ell+1;2i} \end{bmatrix}$, and $g_{L;i} = V_{L;i}^H y_{L;i}$ for $1 \leq i \leq 2^L$.

For any HSS level ℓ, we assemble the block diagonal matrices

$$U_\ell = \bigoplus_{i=1}^{2^\ell} U_{\ell;i}, \qquad V = \bigoplus_{i=1}^{2^\ell} V_{\ell;i}, \qquad D_\ell = \bigoplus_{i=1}^{2^\ell} D_{\ell;i}, \tag{7.14}$$

denoted here as direct sums of their diagonal blocks. We also define matrices S_ℓ, representing the recurrences for $f_{\ell;i}$ and $g_{\ell;i}$, satisfying

$$v = Ay = D_\ell y + U_\ell S_\ell V_\ell^H y. \tag{7.15}$$

We now discuss parallelizing the computation $v = Ay$, to generalize PA0 and PA1 to HSS matrices. Parallelization of these recurrences is natural given the perfect binary tree structure. Pseudocode for PA0-HSS can be found in [111].

7.1.3.1 PA0 for HSS Matrices

Recall from Section 3.2 that PA0 refers to the naïve algorithm for computing k SpMV operations and PA1 refers to the communication-avoiding variant. We first discuss how to modify PA0 when A is HSS, exploiting the $v = Ay$ recurrences for each $1 \leq j \leq k$; we call the resulting algorithm PA0-HSS. PA0-HSS can be seen as an HSS specialization of known approaches for distributed-memory $\mathcal{H}$-matrix-vector multiplication [113]. We assume the HSS representation of A has perfect binary tree structure to some level $L > 2$, and there are $p \geq 4$ processors with p a power of 2. For each processor $m \in \{0, 1, \ldots, p-1\}$, let L_m denote the smallest level $\ell \geq 1$ such that $p/2^\ell$ divides m. We also define the intermediate level $1 < L_p = \log(p) \leq L$ of the HSS tree; each $L_m \geq L_p$, where equality is attained when m is odd.

First, on the upsweep, each processor locally computes $V_{L_p}^H y$ (its subtree, rooted at level $L_p = \log(p)$) and then performs L_p steps of parallel reduction, until there are two processors active, and then a downsweep until level L_p, at which point each processor is active, owns $S_{L_p} V_{L_p}^H y$, and recurses into its local subtree to finally compute its rows of $v = D_L y + U_L S_L V_L y$. More precisely, we assign processor m the computations

$$f_{\ell;i} \text{ and } g_{\ell;i} \text{ for } \left\{ \ell, i : \substack{L \geq \ell \geq L_p \\ 2^\ell m/p+1 \leq i \leq 2^\ell (m+1)/p} \right\} \text{ and for } \left\{ \ell, i : \substack{L_p - 1 \geq \ell \geq L_m \\ i = 2^\ell m/p+1} \right\}$$

and D_L, U_L, and V_L are distributed contiguously block rowwise, so processor m stores blocks $D_{L_p;m+1}$, $U_{L_p;m+1}$, and $V_{L_p;m+1}$. The $R_{\ell;i}$, $W_{\ell;i}$, and $B_{\ell;i}$ matrices are distributed so that they are available for the computations in the upsweep/downsweep; we omit further details for brevity, but discuss memory requirements when we compare with PA1-HSS, below.

7.1.3.2 PA1 for HSS Matrices

The block-diagonal structure of D_ℓ, U_ℓ, and V_ℓ in (7.14) suggests an efficient parallel implementation of PA1-BC, which we present as PA1-HSS (Algorithm 35). The only parallel communication in PA1-HSS occurs in two Allgather operations, in Lines 1 and 5. This means each processor performs the *entire* upsweep/downsweep between levels 1 and L_p locally. The additional cost shows up in our complexity analysis (see Table 7.1) as a factor of p in the flops and bandwidth costs, compared to a factor of $\log(p)$ in PA0-HSS; we also illustrate this tradeoff in Section 7.1.4.

We assume the same data layout as PA0-HSS: each processor stores a diagonal block of D_{L_p}, U_{L_p}, and V_{L_p} (but only stores the smaller blocks of level L). We assume each processor is able to apply S_{L_p}. We rewrite (7.9) for the local rows, and exploit the block diagonal structure of D_{L_p} and U_{L_p}, to write

$$\begin{aligned} y_{L_p,m+1}^{(j)} = \Big(& (D_{L_p,m+1} - \hat{\alpha}_{j-1}) y_{L_p,m+1}^{(j-1)} - \hat{\beta}_{j-2} y_{L_p,m+1}^{(j-2)} \\ & + U_{L_p,m+1} (b_{j-1})_{\{mr+1,\ldots,(m+1)r\}} \Big) / \hat{\gamma}_{j-1}. \end{aligned} \tag{7.16}$$

Algorithm 35 PA1-HSS (Blocking Covers). Code for processor m.

1: Compute a basis for $\mathcal{K}_{k-1}(D_{L_p;m+1}, U_{L_p;m+1})$; premultiply by $V^H_{L_p;m+1}$.
2: Compute $[W_0, \ldots, W_{k-2}]$ by an Allgather.
3: Compute w_i^j for $0 \le i \le k-2$, and $1 \le j \le k-i-1$, via (7.7).
4: Compute a basis for $\mathcal{K}_k(D_{L_p;m+1}, y^{(0)}_{L_p;m+1})$; premultiply by $V^H_{L_p;m+1}$.
5: Compute $[x_0, \ldots, x_{k-1}]$ by an Allgather.
6: Compute $[b_0, \ldots, b_{k-1}]$ by (7.8), where $S = S_{L_p}$ is applied as described above.
7: Compute local rows of $[y^{(0)}, \ldots, y^{(k)}]$ according to (7.16).

Table 7.1: Asymptotic complexity of PA0-HSS and PA1-HSS, ignoring constant factors. 'Offline' refers to Lines 1–3 and 'Online' refers to Lines 4–7 of PA1-HSS.

Algorithm		Flops	Words moved	Messages	Memory
PA0-HSS		$kqrn/p + kqr^2 \log p$	$kqr \log p$	$k \log p$	$(kq+r)n/p + r^2 \log p$
PA1-HSS	(offline)	$kr^2 n/p + k^3$	$kr^2 p \log p$	$\log p$	$(kq+r)n/p + k(q+r)rp$
	(online)	$kqrn/p + k(k+r)^2 qp$	$kqrp \log p$	$\log p$	

Each processor locally computes all rows of $b_j = S_{L_p} V^H_{L_p} y^{(j)} = S_{L_p} \cdot z$, where z is the maximal parenthesized term in (7.8), using the HSS recurrences:

$$V^H_{L_p} y^{(j)} = z = \begin{bmatrix} g^T_{L_p,1} & \cdots & g^T_{L_p,p} \end{bmatrix}^T \mapsto \begin{bmatrix} f^T_{L_p,1} & \cdots & f^T_{L_p,p} \end{bmatrix}^T = b_j = S_{L_p} V^H_{L_p} y^{(j)}.$$

The rest of PA1-HSS is similar to PA1-BC, except that the Allreduce operations have now been replaced by Allgather operations, to exploit the block structures of V^H.

7.1.3.3 Complexity Analysis

A detailed complexity analysis of PA0-HSS and PA1-HSS can be found in [111]; we summarize the asymptotics (i.e., ignoring constant factors) in Table 7.1. We assume A is given in HSS form, as described above, where all block matrices are dense. For generality, we also assume y is dense and has n rows and q columns. For simplicity, we assume n and *HSS-rank* r are powers of 2 and leaf level $L = \log(n/r)$. Note that one could use faster Allgather algorithms (e.g., [42]) for PA1-HSS to eliminate the factor of $\log(p)$ in the number of words moved. The primary benefit of PA1-HSS over PA0-HSS is that a factor of k fewer messages are required. From the table, we can see that this comes at the cost of p times more words moved and an increase in the computational cost (the second term has a factor of p instead of $\log p$ and $(k+r)^2$ instead of r^2).

7.1.4 Performance Model

We model speedups of PA1-HSS over PA0-HSS on two machine models used by Mohiyuddin [126] – 'Peta,' an 8100 processor petascale machine, and 'Grid,' 125 terascale machines connected via the Internet. Peta has a flop rate $\gamma = 2 \cdot 10^{-11}$ s/flop, latency $\alpha = 10^{-5}$ s/message, and bandwidth $\beta = 2 \cdot 10^{-9}$ s/word, and Grid has flop rate $\gamma = 10^{-12}$ s/flop,

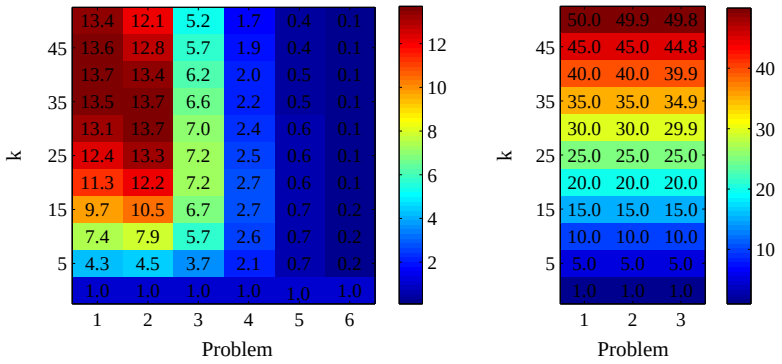

Figure 7.1: Predicted PA1-HSS speedups on Peta (left) and Grid (right). Note that p and n increase with problem number on x-axis.

latency $\alpha = 10^{-1}$ s/message, and bandwidth $\beta = 25 \cdot 10^{-9}$ s/word. Complexity counts used can be found in [111].

Timings are obtained from the performance model described in Section 2.2, which we now review. Our model is a simplified machine model, where a parallel machine consists of p processors, each able to perform arithmetic operations on their M words of *local memory*. Processors communicate point-to-point messages of $n \leq M$ contiguous words, taking $\alpha + \beta n$ seconds on both sender and receiver, over a completely connected network (no contention), and each processor can send or receive at most one message at a time. Recall that for an algorithm where a processor (on the critical path) performs F flops and sends/receives S messages containing a total of W words, we model the worst-case runtime using the equation

$$T = \gamma F + \beta W + \alpha S,$$

where α represents a latency cost incurred by every message, β represents a bandwidth cost linear in the message size, and γ represents the time to perform a floating point operation on local data. For simplicity, we do not model overlapping communication and computation.

Speedups of PA1-HSS over k invocations of PA0-HSS, for both Peta and Grid, are shown in Figure 7.1. We used parameters from the parallel HSS performance tests of Wang et al. [185], where $p = (4, 16, 64, 256, 1024, 4096)$, $n = (2.5, 5, 10, 20, 40, 80) \cdot 10^3$, $r = (5, 5, 5, 5, 6, 7)$; for example, 'Problem 1' in Figure 7.1 uses parameters $(p_1 = 4, n_1 = 2.5, r_1 = 5)$. Note that for Grid we only use the first 3 triples (p_i, n_i, r_i) since $p_{\max} = 125$.

On Peta, we see $O(k)$ speedups for smaller p and k, but as these quantities increase, the expected speedup drops. This is due to the extra multiplicative factor of p in the bandwidth cost and the extra additive factor of k^3qp in the flop cost of PA1-HSS. Since the relative latency cost is lower on Peta, the effect of the extra terms becomes apparent for large k and p. On Grid, PA0-HSS is extremely latency bound, so a $\Theta(k)$-fold reduction in latency results in a $\Theta(k)\times$ faster algorithm. This is the best we can expect. Note that many details are

abstracted in these models, which are meant to give a rough idea of asymptotic behavior. Realizing such speedups in practice remains future work.

7.1.5 Conclusions and Future Work

The new parallel communication-avoiding matrix powers algorithm for $A = D + USV^H$, where D is well partitioned and USV^H has low rank but A may not be well partitioned, allows speedups for a larger class of problems than previous algorithms [63, 126] which require well-partitioned A. This approach exploits low-rank properties of dense blocks, asymptotically reducing parallel latency cost. We demonstrate the generality of the parallel blocking covers technique by applying it to matrices with hierarchical structure. Performance models predict up to 13× speedups on petascale machines and up to $3k$ speedups on extremely latency-bound machines, despite tradeoffs in arithmetic and bandwidth cost (see 7.1). Future work includes a high-performance parallel implementation of these algorithms to verify predicted speedups, as well as integration into preconditioned communication-avoiding Krylov solvers.

In some machine learning applications, Krylov subspace methods are used on dense matrices, so the matrix vector multiplication, rather than the inner products, becomes the dominant cost in terms of communication [23]. Many efforts have thus focused on ways to reduce the cost of the matrix vector multiplication; for example, to perform Gaussian process regression, the authors in [23] reduce the cost of the matrix-vector multiplication in GMRES by using an HSS representation of the operator. The runtime of this method could potentially be further reduced by substituting a CA-GMRES method (see [102]) combined with our HSS matrix powers kernel; we leave this as potential future work.

7.2 Streaming Matrix Powers Kernel

In our CA-KSMs, the Krylov basis computation $\mathcal{V} = [Ax, \ldots, A^k x]$, which is performed in a block rowwise fashion (Section 3.2), can be interleaved with any reduction operation involving those block rows of $\mathcal{V}$, e.g., the TSQR algorithm, or computing a Gram matrix $\mathcal{V}^H\mathcal{V}$. Typically, this can only reduce communication costs by at most a constant factor, but is certainly a worthwhile optimization to consider in practice. In some cases, however, interleaving can actually lead to asymptotic (i.e., s-fold) communication savings.

Consider the case where A can be represented in $o(n)$ words of data (e.g., a stencil with constant coefficients), where we assume $n > M$, the fast (local) memory size. The Krylov basis vectors require moving $\Theta(sn)$ words of data, so the serial communication bottleneck has shifted from loading A s times ($o(sn)$ data movement) to loading/storing the Krylov basis vectors ($\Theta(sn)$ data movement). In CA-KSMs, we can reduce the serial bandwidth and latency of moving the Krylov basis vectors by a factor of $\Theta(s)$ at the cost of doubling the computational cost of computing matrix powers. We discuss the required kernel for two-term CA-BICG (see Section 4.2); the generalization to other CA-KSMs is straightforward.

In CA-CG (Algorithm 28), the Krylov basis vectors are only accessed twice after being computed: when computing G_k (line 4) and when recovering iterates (line 15). Also, observe that we can execute lines 4 and 15 reading the Krylov bases one time each. We will interleave the computations of lines 3 and 4, computing a block row of $\mathcal{Y}_k$, accumulating the product into G_k, and discarding the block row, and then repeat line 3 just before line 15, and interleave those operations analogously. In the first case, the overall memory traffic is loading the $\Theta(n)$ input vector data for the matrix powers kernel—we assume the $\Theta(s^2)$ matrix G_k fits in fast memory and can be discarded at the end of the $j = s - 1$ iteration of line 13. In the second case, we load the same amount of vector data, plus $O(s^2)$ data (the coefficient vectors (4.8)), and store the five iterates ($\Theta(n)$). Overall, this optimization reduces both bandwidth and latency terms by a factor of $\Theta(s)$. This approach can be generalized to interleaving the matrix powers kernel with other vector operations, such as the Tall-Skinny QR kernel used in CA-GMRES [127].

We call this approach a *streaming matrix powers* optimization. Although this approach requires twice as many calls to the matrix powers kernel, there still may be a practical performance gain, especially under the assumption that A can be represented with $o(n)$ words of data: from a communication standpoint, A is inexpensive to apply. In Algorithm 36, we show the first call to the streaming matrix powers kernel, i.e., for interleaving computation of the Krylov basis with construction of G_k, written using the style and notation of Section 3.2. The second call to the streaming matrix powers kernel which occurs at the end of every outer loop, shown in Algorithm 37, interleaves the Krylov basis computation with the recovery of the iteration updates (line 15 of Algorithm 28).

Algorithm 36 SA1-S-1

1: **for** $\ell \in \{1, \ldots, p\}$ **do**
2: Load vector entries $R(V_\ell) \cap y^{(0)}$
3: Load matrix rows $\{A_i : y_i^{(j)} \in R(V_\ell) \backslash y^{(0)}\}$
4: Compute vector entries $R(V_\ell) \backslash y^{(0)}$
5: Perform a rank-$|I_\ell|$ update to Gram matrix G_k using vector entries V_ℓ
6: Discard entries $V_\ell \backslash y^{(0)}$
7: **end for**

Algorithm 37 SA1-S-2

1: **for** $\ell \in \{1, \ldots, p\}$ **do**
2: Load vector entries $R(V_\ell) \cap y^{(0)}$
3: Load matrix rows $\{A_i : y_i^{(j)} \in R(V_\ell) \backslash y^{(0)}\}$
4: Compute vector entries $R(V_\ell) \backslash y^{(0)}$
5: Recover entries $\{i : i \in I_\ell\}$ of appropriate iteration vectors
6: Discard entries $V_\ell \backslash y^{(0)}$
7: **end for**

This variant should be used in the sequential algorithm for stencil-like matrices (or generally, where the cost of reading A is $o(n)$). Using the streaming technique, we can remove the communication bottleneck of reading and writing the $O(sn)$ basis vectors in each iteration, which lowers the overall communication complexity of performing s steps to $o(n)$.

For sequential algorithms, we also note that the streaming matrix powers optimization enables a *write-avoiding* implementation, with the number of writes per s steps reduced by a factor of $\Theta(s)$ versus the communication-avoiding version. This may be beneficial when writes are more costly than reads in terms of both time and energy, as is the case with many types of nonvolatile memory. We direct the reader to the technical report [33] for more details on write avoiding algorithms.

7.3 Partitioning for Matrix Powers Computations

In this section, we discuss a potential improvement to the method of partitioning matrices with highly nonsymmetric structure to avoid communication in computing the matrix powers kernel. Our method is based on the hypergraph partitioning problem. A hypergraph $H = (V, N)$ is defined as a set of vertices V and hyperedges ('nets') N, where every hyperedge $n_j \in N$ is a subset of vertices, i.e., $n_j \subseteq V$. Each vertex has a weight w_i and each hyperedge has a cost c_j. The size of a hyperedge is defined as the number of its vertices, i.e., $|n_j|$. We define a p-way partition $\Pi = [I_1, \ldots, I_p]$, where the weight of a part W_ℓ corresponds to the sum of the weights of vertices in that part, i.e., $W_\ell = \sum_{v_i \in I_\ell} w_i$. The 'connectivity set' Λ_j of n_j is defined as the set of parts $\{I_i : v \in n_j \text{ and } v \in I_i\}$, and the 'connectivity' of n_j is defined as $S_j = |\Lambda_j|$. A hyperedge is said to be 'cut' if $S_j > 1$ and 'uncut' otherwise. For a given partition, the 'cutsize' is defined as

$$\chi(\Pi) = \sum_{n_j \in N} c_j(S_j - 1).$$

Given these definitions, the hypergraph partitioning problem is the problem of finding a partition that minimizes the cutsize subject to the load-balancing constraint

$$W_\ell \leq W_{\text{average}}(1 + \epsilon) \quad \text{for} \ell \in \{1, \ldots, p\},$$

where $W_{\text{average}} = (\sum_{v_i \in V} w_i)/p$.

The hypergraph partitioning problem has been shown to correspond exactly to minimizing the communication in SpMV [37]. Each vertex partition corresponds to a parallelization of the classical SpMV computations, and the induced hyperedge cut corresponds to interprocessor communication for that parallelization. By varying the metric applied to the cut, one can exactly measure communication volume (number of words moved), or synchronization (number of messages between processors) on a distributed memory machine. Various heuristics are applied to find approximate solutions to these NP-hard partitioning problems in practice (mature software packages are available; see, e.g., [64]); additional constraints may be applied to enforce load balance requirements.

To find a good partition for performing a matrix powers computation, we must encapsulate the communication cost of k SpMVs. Recall that to reduce parallel latency in the matrix powers kernel, we want to find a rowwise partition of A. We therefore consider the 'column-net' hypergraph model, the partitioning of which produces a rowwise partition of A [37]. In the column-net model for a single SpMV, vertices correspond to matrix rows and hyperedges correspond to matrix columns. There is one vertex v_i for each row i of A and one hyperedge n_j for each column j of A, with $v_i \in n_j$ if and only if $A_{ij} \neq 0$.

In our case, we must look at the '$k-$level hyperedges', or the hyperedges corresponding to the structure of $|A|^k$. It is straightforward to prove that the problem of finding a minimum $k-$level hyperedge cut reduces to finding a communication-optimal partition for the matrix powers computation. However, just computing the hypergraph of $|A|^k$ can be a costly operation — typically more costly than the subsequent matrix powers computation — and so the cost would have to be amortized over many calls to PA1.

Another option to reduce the cost of hypergraph construction is to use Cohen's reachability estimation algorithm for estimating the size of k steps of the transitive closure [54]. This is a randomized algorithm which estimates the size of (number of vertices in) each hyperedge in $O(n \cdot \text{nnz})$ time. Cohen's original motivation was in database query size estimations, optimal matrix multiplication orderings, and in efficient memory allocation. We briefly describe this algorithm.

Let $G = (V, E)$ be a layered graph representing dependencies in the matrix powers computation (see Section 3.2.1), where there are $(k+1)n$ vertices $V = \{v_i^{(j)} : 1 \leq i \leq n, 0 \leq j \leq k\}$ and $(v_i^{(j)}, v_\ell^{(j-1)}) \in E$ if $A_{i\ell} \neq 0$ for $j \in \{2, \ldots, k\}$. We also borrow other notation from Section 3.2.1. We let $V^{(j)} = \{v_i^{(j)} : 1 \leq i \leq n\}$, $R(X)$ denote the *reachability* of set $X \subseteq V$, and let $R^{(j)}(X) = R(X) \cap V^{(j)}$. The algorithm initially assigns a length-r vector of rankings $[a_1, \ldots, a_r]$ selected uniformly at random from the interval $[0, 1]$ to each vertex $v \in V^{(0)}$. In iteration $m = \{1, \ldots, k\}$, for each vertex $v \in V^{(m)}$ we set v's ranking vector to be the coordinate-wise minima of the ranking vectors of vertices $v' \in R^{(m-1)}(v)$. At the end of k iterations, each vertex in $V^{(k)}$ has a ranking vector, and for each $v \in V^{(k)}$ we apply the estimator

$$\bar{S}(v) = \frac{r-1}{\sum_{\ell=1}^{r} a_\ell}$$

where $\bar{S}(v)$ estimates the number of nonzeros in the column of A corresponding to v. This correctly estimates the number of nonzeros with probability inversely proportional to $\sqrt{r}$.

We discuss one way that Cohen's algorithm can be applied in the context of hypergraph partitioning for a matrix powers computation. We first run Cohen's reachability estimation algorithm to estimate the size of each hyperedge in the column-net model for $|A|^k$. Let tol be a user-specified tolerance parameter. After the estimator is applied, if $\bar{S}(v) > n \cdot \text{tol}$, then this hyperedge is constructed as a 1-level column net, i.e., this hyperedge contains vertices that correspond to nonzeros in this column in A. Otherwise, this hyperedge is constructed as a k-level column net, i.e., this hyperedge contains vertices that correspond to nonzeros in this column in A^k. We note that if tol is set to 0, we have a hypergraph that corresponds

Table 7.2: Test matrices for partitioning tests. Taken from the University of Florida Sparse Matrix Collection [57]

Matrix	Application	n	nnz	% pattern symm.
arc130	material science	130	1037	76%
west0132	chemical eng.	132	413	2%
str_0	linear prog.	363	2454	0%
gre_343	dir. graph	343	1032	0%
mcca	astrophysics	180	2659	64%
rw496	Markov model	496	1859	47%
str_200	linear prog.	363	3068	1%

to the column-net model of A, and if tol = 1, we end up with a hypergraph corresponding to the column-net model of A^k. Selecting tol somewhere in between these two can allow us to build a hypergraph model that is much less costly to construct and partition, but which still captures important information about the structure of A^k. The intuition here is that if the hyperedge i corresponding to column i in A is very dense, then not only will this hyperedge be costly to compute and partition, but the corresponding column will end up being distributed among many processors and will thus require communication regardless of how it is partitioned; i.e., there is no hope to exploit locality.

We illustrate this method on a few very small matrices from the University of Florida Sparse Matrix Collection [57], details of which are listed in Table 7.2. For each problem, we partition into 4 parts and used $k = 4$. Figure 7.2 compares the size of the resulting partitioning problem ($\sum_{n_j \in N} |n_j|$) and Figure 7.3 compares the total communication volume in the resulting partition computed by the four different methods: hypergraph partitioning using 1-level column nets (labeled 'A'), k-level column nets (labeled 'A^k'), and the new "sparsified" column nets described above (labeled 'A sparse'), and graph partitioning of $A + A^T$ (labeled '$A+A^T$'). For the sparsified column nets, we used tol = 0.5. We used PaToH [38] to perform hypergraph partitioning and Metis [107] to perform graph partitioning, each using the default parameters.

We can see from these figures that the sparsified column nets can lead to partition quality comparable to that of building the full k-level column nets at a significantly reduced cost in terms of hypergraph size. When the matrix was close to being structurally symmetric, for example, arc130, we can see from Figure 7.2 that the sparsification technique resulted in the same structure as the 1-level column nets and, from Figure 7.3 that all partitioning methods were comparable in terms of resulting partition quality. This suggests that such complex partitioning techniques are only beneficial when the matrix is highly nonsymmetric in structure or has a highly nonuniform distribution of nonzeros per row. With a tridiagonal A for example, we can bound the nonzero structure of A^2 (pentadiagonal) with no computation required, as we know each row gains at most two nonzeros each time we power A. For this reason, building any k-level hyperedges isn't likely to provide a better partition than

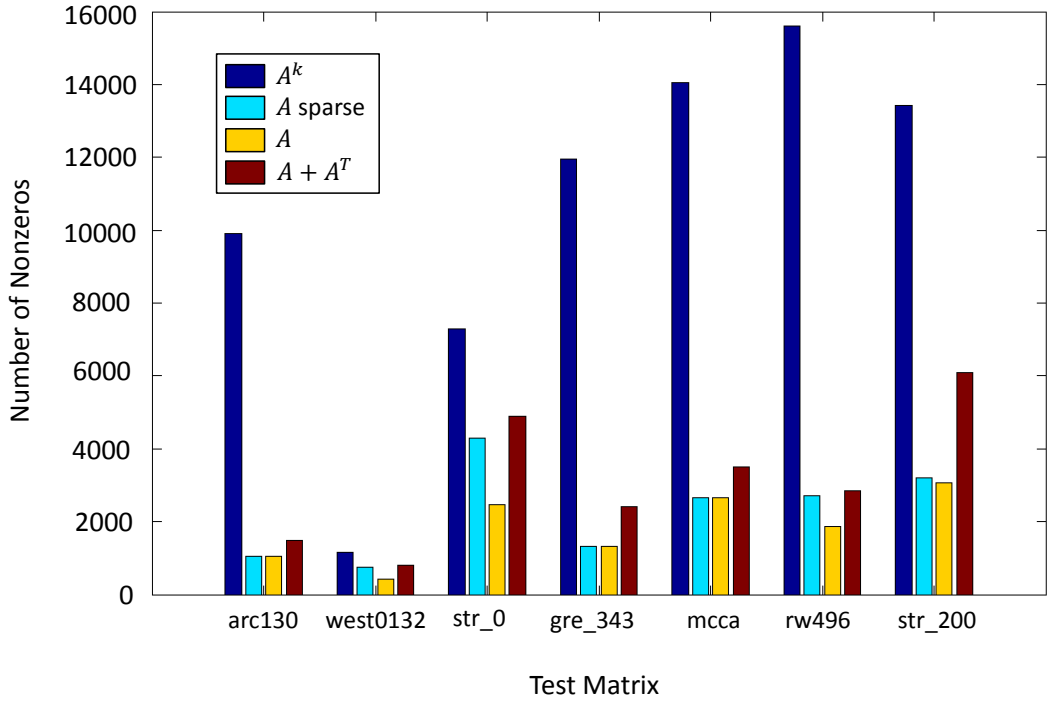

Figure 7.2: Number of nonzeros in the matrix corresponding to the constructed (hyper)graph for various test matrices and partitioning methods. In the legend, 'A' corresponds to hypergraph partitioning using 1-level column nets, 'A^k' corresponds to hypergraph partitioning using k-level column nets, 'A sparse' corresponds to hypergraph partitioning using the new "sparsified" column nets, and '$A + A^T$' corresponds to graph partitioning of $A + A^T$.

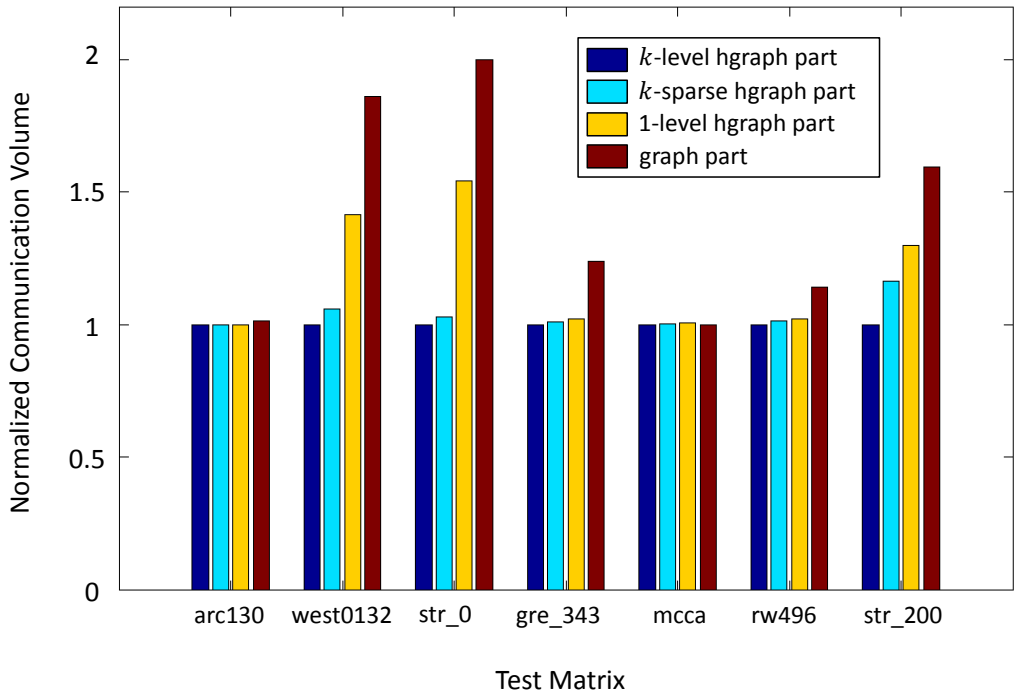

Figure 7.3: Normalized communication volume for computing matrix powers using the resulting partition for various test matrices and partitioning methods. In the legend, 'A' corresponds to hypergraph partitioning using 1-level column nets, 'A^k' corresponds to hypergraph partitioning using k-level column nets, 'A sparse' corresponds to hypergraph partitioning using the new "sparsified" column nets, and '$A + A^T$' corresponds to graph partitioning of $A + A^T$.

had we only considered the structure of A. Allowing specification of the tolerance parameter can enable us to trade off partition quality for partitioning time. For example, if we know the matrix is a stencil (i.e., structurally symmetric with about the same number of nonzeros per row/column), then we can set tol = 0 and just use the 1-level column nets. Further investigation is needed in testing larger problem sizes, gathering data on partitioning time, and determining how to set the tolerance parameter based on matrix structure. We also note that it is harder to precisely measure the computation cost of the matrix powers kernel using the hypergraph model, and thus load balancing is a difficult problem: for PA0, this is a simple function of the sparsity of A, while for PA1, it depends on the partition and nonzero fill introduced by computing successive powers of A, rather than just the sparsity of A^k.

7.4 Conclusions and Future Work

In this Chapter we have presented potential performance optimizations for matrices with specific nonzero structures, including a communication-avoiding matrix powers algorithm for matrices that can be split into sparse and low-rank parts, a streaming matrix powers optimization for implicitly-represented matrices, and a technique for reducing the cost of building and partitioning a hypergraph that encodes the dependencies in computing k repeated SpMVs. There are, of course, many other matrix structures for which optimizations and/or simplifications can be devised. The vast space of matrix structures and potential optimizations makes auto-tuning a necessary component of any software library developed for CA-KSMs.

In the next chapter, we perform weak and strong scaling studies for distributed-memory implementations of some of our methods, and comment on where we expect the most benefit of the communication-avoiding optimizations in practice. We elaborate on one particularly beneficial use of CA-KSMs: as coarse grid solve routines in geometric multigrid methods, which we show enables up to 2.5× speedups for combustion and cosmology applications.

Chapter 8

Performance and Applications

In this Chapter, we show speedups from the use of communication-avoiding Krylov methods on large-scale distributed-memory problems from various scientific domains. All tests are performed on the Hopper supercomputer, detailed in Section 8.1. We note that speedup results for CA-KSMs (specifically, CA-GMRES) in a shared-memory environment were previously obtained by Mohiyuddin et al. [127].

In Section 8.2, we perform weak and strong scaling experiments for distributed-memory CA-CG for large-scale PDE solves on Hopper. Our results provide guidance in identifying the space in which the communication-avoiding approach is most beneficial in terms of performance; we obtain up to 6× speedups in the best case.

We then in Section 8.3 discuss the implementation and optimization of a communication-avoiding CA-BICGSTAB method as a high performance, distributed-memory bottom solver for geometric multigrid. We show that the communication-avoiding approach enables speedups of over 4× on synthetic benchmarks and over 2.5× in real combustion and cosmology applications. These results were first reported in [190].

In the bottom solve application, using the classical method, the subdomain owned by each processor is so small that parallel latency overwhelmingly dominates the runtime. Also confirmed by experiments in Section 8.2, this is a great example of where the blocking of inner products in communication-avoiding Krylov methods will have the most impact on performance, that is, when the application performance is limited by expensive MPI collective communication.

8.1 Experimental Platform

All experiments in this section were performed on Hopper, a Cray XE6 at NERSC. Hopper is NERSC's first petaflop system, with a peak performance of 1.28 Petaflops/s. Hopper earned 5^{th} place on the November 2010 Top500 list.

Hopper has 6,384 compute nodes. Each compute node has two 12-core MagnyCours processors which each consist of two 6-core AMD Opteron chips. Cores within on chip share a

6MB L3 cache. Pairs of compute nodes (48 cores) are directly connected by HyperTransport to a Gemini network chip. The network chips are connected in a 3D torus with high bandwidth and low latency. The configuration of the torus results in some asymmetry in peak bandwidth depending on direction, but the programmer has no control over job placement. The internode latency is about 1.27μs (nearest nodes pair) and 3.88μs (farthest nodes pair) on a quiet network. For two cores connected to the same Gemini chip, latency is $<$1μs.

8.2 Model Problem Performance

In this section we give parallel performance results on Hopper (see Section 8.1) for a model 2D Poisson problem with the right-hand side set to the vector of all 1s. We compare the performance of the classical conjugate gradient method (CG) with the communication-avoiding conjugate gradient method (CA-CG) with various s values.

In all tests, we use flat MPI on Hopper, with 4 MPI processes per node (one per chip). For all tests with CA-CG, we use the Newton basis (see Section 3.2.5), parameters which were computed using estimates for the largest and smallest eigenvalues precomputed offline using a Jacobi-Davidson solver in SLEPc [99]. This resulted in similar convergence rates similar to classical CG for all tests; the times reported were measured by the overall solve time, i.e., including all iterations until convergence. The convergence criteria used is that the relative residual norm ($\|r_i\|_2/\|b\|_2$) is reduced to at least 10^{-10}.

The CA-CG solver is implemented as a KSM solver within the Portable, Extensible Toolkit for Scientific Computation (PETSc) framework [4]. Our implementation of CA-CG does *not* use a communication-avoiding matrix powers optimization (see Section 3.2). Because we do twice as many SpMVs and thus double the bandwidth cost of P2P communication in the matrix powers kernel, we only expect to see speedups when the problem is extremely latency-bound and the MPI collective communication dominates. An implementation which includes a communication-avoiding matrix powers kernel remains future work.

8.2.1 Strong Scaling Results

In Figure 8.1, we present strong scaling tests for CG and CA-CG with $s \in \{2, 4, 6, 8, 10\}$. The same test is performed for three different problem sizes: 512^2 (top), 1024^2 (middle), and 2048^2 (bottom) grids. We scale from 1 to 1024 nodes, which correspond to 4 and 4096 MPI processes, respectively. The same data in presented in two ways; on the left in Figure 8.1, we plot the raw timing results, and on the right, we plot the speedup relative to classical CG for each number of processors.

Looking at the plots on the left, we can see that classical CG scales well as we increase the number of processors until some point when the time starts to increase again. The CA-CG method also scales well, with about the same slope as the classical CG method. For CA-CG, however, this scaling continues for higher numbers of processors. Thus although CA-CG is

slower than classical CG for smaller numbers of processors, at some point the lines cross and CA-CG outperforms classical CG. The smaller the problem size, the smaller the number of processors where this crossover occurs.

This behavior is expected due to additional costs in CA-CG, especially since we have not used communication-avoiding matrix powers computations (see Section 3.2), which double the amount of point-to-point communication, which can be a relatively significant cost when running with small numbers of processors. We only expect the latency cost of the global collectives (inner products) to dominate the runtime once a large number of processors is used. This is where our CA-CG has the most benefit, since we have asymptotically reduced the number of global collectives performed. We also point out that since this is a strong scaling (fixed) problem, we would in practice like to use the method that gives the fastest runtime, which is CA-CG for all three tested problem sizes. It is also clear from Figure 8.1 that of the s values tested, lower s values give faster runtimes up until some number of processors at which point the lines intersect and higher s values ($s = 8$ or $s = 10$) become the winners. This is explained by the same reasoning above; as the number of processors is increased, global collectives become increasingly expensive, so it becomes increasingly beneficial to avoid them.

These results are promising, and demonstrate the benefits of CA-CG on latency-bound problems. We stress that performance of the CA-CG method can be improved even further by the implementation and use of a communication-avoiding matrix powers kernel. This remains future work.

8.2.2 Weak Scaling Results

In Figure 8.2, we present weak scaling tests for CG and CA-CG with $s \in \{2, 4, 6, 8, 10\}$. The same test is performed for three different per-process problem sizes: 16^2, 32^2, and 64^2. Again, we scale from 1 to 1024 nodes, which correspond to 4 and 4096 MPI processes, respectively.

In weak scaling tests, the runtime ideally stays constant and the number of processes and overall problem size is increased. Looking at Figure 8.2, we can see that for the smallest two per-process problem sizes, the slope of the lines for CA-CG are always lesser than for classical CG, and thus indicate closer-to-ideal behavior. For the largest per-process problem size, the classical CG method initially exhibits better scaling (lower slope), but as the number of processes and overall problem size is increased, its slope becomes greater than those of the CA-CG method. At the final data point, the best runtime was obtained from CA-CG with $s = 2$.

From Figure 8.2, we can also see that as the per-process problem size is increased, the best s value to use decreases; the fastest results were obtained by CA-CG with $s = 10$ for size 16^2, by CA-CG with $s = 4$ for 32^2, and by CA-CG with $s = 2$ for 64^2. This is as expected, since the smaller the per-process problem size, the less computation is required, and thus communication takes up a relatively greater fraction of the overall runtime. Again, we emphasize that while these results are promising, the weak scaling performance of the

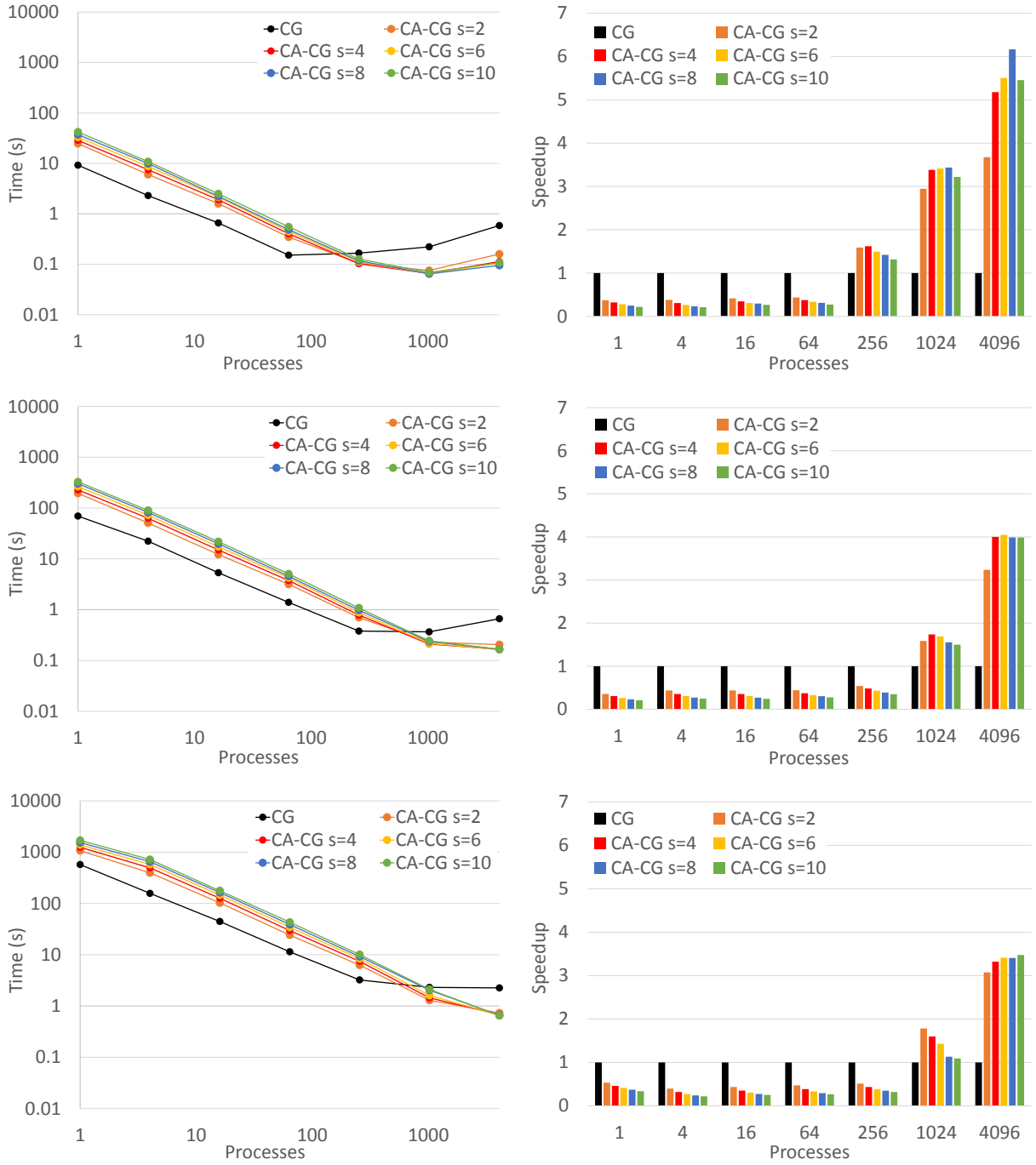

Figure 8.1: Strong scaling time and speedups

CA-CG method can be improved even further by the use of a communication-avoiding matrix powers kernel. This remains future work.

8.3 CA-KSMs as Coarse Grid Solve Routines

Our second application of communication-avoiding Krylov subspace methods is as bottom solve routines for geometric multigrid solvers within adaptive mesh refinement (AMR) applications. Krylov method bottom solvers are available as an option or as the default in a number of software packages, including BoxLib [1], Chombo [2], PETSc [4], and hypre [3].

At some level of the multigrid V-cycle, individual subdomain sizes become so small that further coarsening is impractical due to communication. At this point, there are a number of potential approaches. One approach is to repartition the data, that is, the whole problem is mapped to a subset of the processors at some level of the coarsening process [132]. The data could also be replicated by combining data with neighboring processors (see [97, 191]). Another approach is to stop coarsening at this point, solve the coarse grid problem using, e.g., a Krylov subspace method, and then begin the interpolation phase of the V-cycle. It is this approach that we focus on for the remainder of this section. (Note that the two approaches could in principle be combined.) Our results show that the communication-avoiding approach enables speedups of over 4× on synthetic benchmarks and over 2.5× in real combustion and cosmology applications. We note that this section has been adapted from work that first appeared in [190].

8.3.1 Geometric Multigrid

A wide variety of scientific applications require the solution of elliptic and/or parabolic partial differential equations. Simulations that solve time-dependent systems of equations may require multiple timesteps with many solves per timestep. These solves thus constitute a large fraction of simulation runtime. Geometric multigrid methods are amongst the most commonly used methods for solving such systems of equations.

As depicted in Figure 8.3, a multigrid V-cycle begins with the given computational domain. The first phase is restriction, where, for some number of levels, the domain is coarsened until some criterion is met. At each level, the error is smoothed using, e.g., Chebyshev iteration or Gauss-Seidel with red-black ordering, and a new right-hand side for the coarser level is constructed from the residual.

For a single uniform domain, coarsening is typically done until the domain is 2-4 points per side, at which point the coarse grid problem is solved using either a direct or iterative method (the 'bottom solve'). In the case that the domain is decomposed over multiple processes, the coarsening stops when each local subdomain reaches this size. For a large number of subdomains, this problem size may still be large enough such that the bottom solve creates a performance bottleneck. One method to alleviate this bottleneck is to agglomerate smaller subdomains into larger subdomains, continue coarsening, and then solve on a smaller number

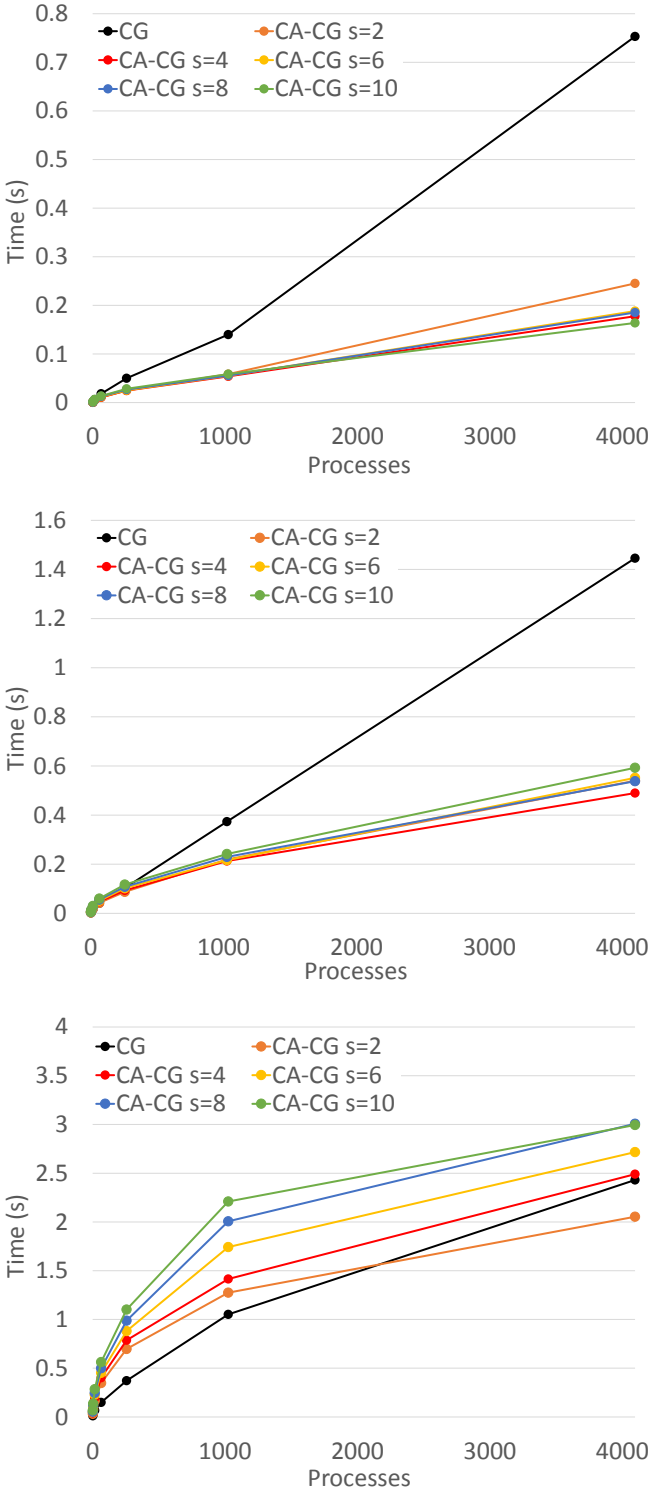

Figure 8.2: Weak scaling time

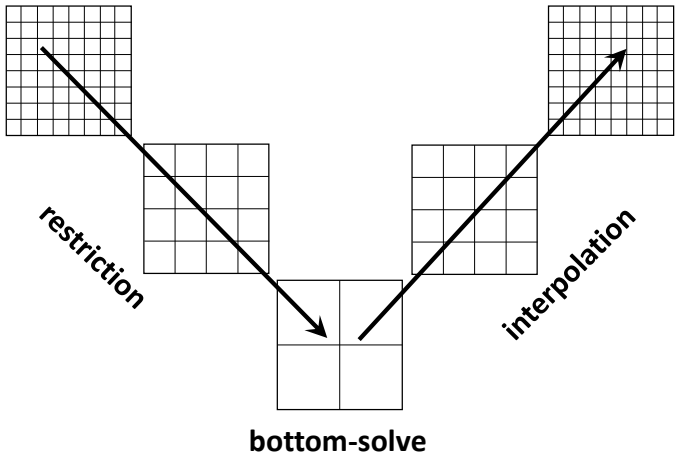

Figure 8.3: Depiction of multigrid V-cycle.

of processors. This technique is effective for uniform rectangular meshes, but may not always be feasibly for non-rectangular domains or individual adaptive mesh refinement (AMR) levels with irregular coarse/fine boundaries.

After the bottom solve on the coarsest mesh, the coarse solution is interpolated to update the solution on finer levels. Again, the error is smoothed at each level. These V-cycles are repeated until the norm of the residual on the original (finest) grid reaches some specified tolerance.

8.3.2 Related Work

A review of related work related to approaches for reducing communication costs in Krylov subspace methods can be found in Section 2.4. In [78], the authors present a communication-avoiding Chebyshev iteration that uses the matrix powers kernel (see Section 3.2). They apply this method as the smoother in a geometric multigrid solver for the Poisson equation on a regular 2D grid. In this work we do not study communication-avoiding smoothers and instead focus on the larger performance bottleneck of the bottom solve.

As previously mentioned, data repartitioning or replication are other approaches to alleviating the V-cycle communication bottleneck. These approaches also involve some amount of data movement between processors to perform the redistribution. Since usually very few

iterations of the Krylov method are needed to perform the bottom solve to sufficient accuracy, it is unclear which approach (or combination of these approaches) will win in a practical setting. Further study is warranted.

8.3.3 The miniGMG Benchmark

We describe the miniGMG benchmark of Williams et al. [189], a geometric multigrid performance benchmark, which we used for performing benchmarking and comparison of classical and communication-avoiding bottom solve routines. Experiments were first performed using the MPI+OpenMP version of miniGMG. The miniGMG benchmark is designed allow detailed timing breakdowns for different parts of the V-cycle and is designed to mimic real AMR multigrid combustion applications.

The code solves a finite volume discretization of the variable-coefficient Helmholtz equation $Lu = a\alpha u - b\nabla \cdot \beta \nabla u = f$ with $\alpha = \beta = 1.0$ and $a = b = 0.9$ on a cubic domain with periodic boundary conditions. The right hand side is a 3D triangle wave, constructed synthetically. The V-cycles are run until the norm of the residual on the fine grid is reduced by a factor of at least 10^{-10}.

The global 3D domain is partitioned into subdomains of size 64^3 which are distributed among the processes. Each MPI process has 6 threads (one Opteron chip). Piecewise constant interpolation is used between the levels and the V-cycle is terminated when each subdomain is coarsened to a size of 4^3. At this point, miniGMG uses a matrix-free BICGSTAB method to solve the coarse grid problem.

The vector and matrix operations needed to execute the BICGSTAB algorithm are grid and stencil operations in miniGMG. Inner products are implemented with pair-wise multiplications between the corresponding cells of two grids, followed by a global reduction (MPI collective communication). A matrix vector product involves a ghost zone exchange between neighbors, implemented with point-to-point (P2P) MPI communication, and application of a stencil to a grid. Specifically, MPI_Allreduce is used for global communication and MPI_ISend and MPI_Irecv are used for P2P communication.

8.3.4 Classical BICGSTAB Performance

Figure 8.4 shows a breakdown of the miniGMG solver time into time spend in the BICGSTAB bottom solve and time spent in the rest of the V-cycle, weak scaling from 8 to 4096 MPI processes on Hopper. Since each process consists of 6 cores and receives a 64^3 subdomain, this corresponds to problem sizes ranging from 128^3 over 48 cores to $N = 1024^3$ over 24,576 cores. We can see here that although the rest of the V-cycle scales very well (horizontal line is ideal), the time spent in the BICGSTAB bottom solver grows very quickly as processes are increased.

In Figure 8.5, we see that the growth in BICGSTAB time is due to an increase in time in spent in calls to MPI_Allreduce. The time spent in P2P communication is an order of magnitude smaller and computation time is relatively insignificant. Figure 8.6 provides

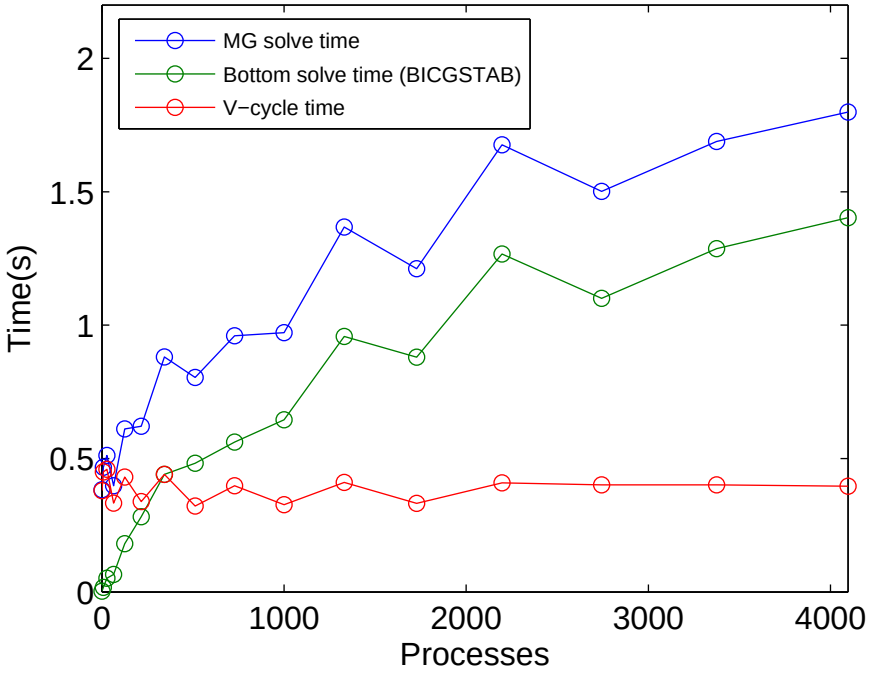

Figure 8.4: Breakdown of miniGMG time into bottom solve time and V-cycle time, weak scaling up to 4096 processes (24,576 cores) on Hopper where each process owns a 64^3 subdomain.

further explanation. This plot shows that the growth in MPI_Allreduce time is attributable to the compounding effects of higher average MPI_Allreduce time per iteration as machine scale grows and higher number of required iterations as the problem size grows. This results in an increasing number of increasingly slower iterations.

As it is infeasible, short of changing the MPI implementation, network, or job scheduler, to considerably reduce the time for an MPI_Allreduce operation, we instead focus on the use of communication-avoiding Krylov subspace methods to reduce the total number of MPI_Allreduce operations.

8.3.5 Design Space

For this application, our focus is on improving scalability for the case where we have a larger number of processors and a very small number of degrees of freedom per processor. In this case, as discussed above, the cost of the MPI collective communication dominates the runtime. As we saw in the scaling experiments of Section 8.2, this is where we expect to see the most benefit from minimizing collectives in CA-KSMs.

For smaller processor counts or larger number of degrees of freedom per processor, the bottleneck shifts from the MPI collectives to the MPI P2P and to the (on-node) computation,

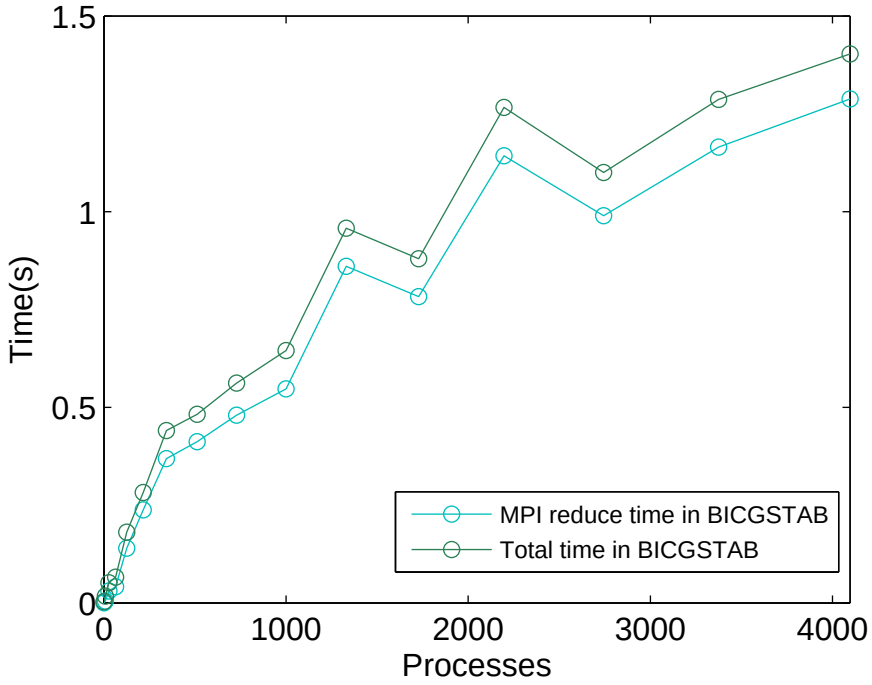

Figure 8.5: Comparison of total bottom solve time with time spent in MPI_Allreduce operations (weak scaling).

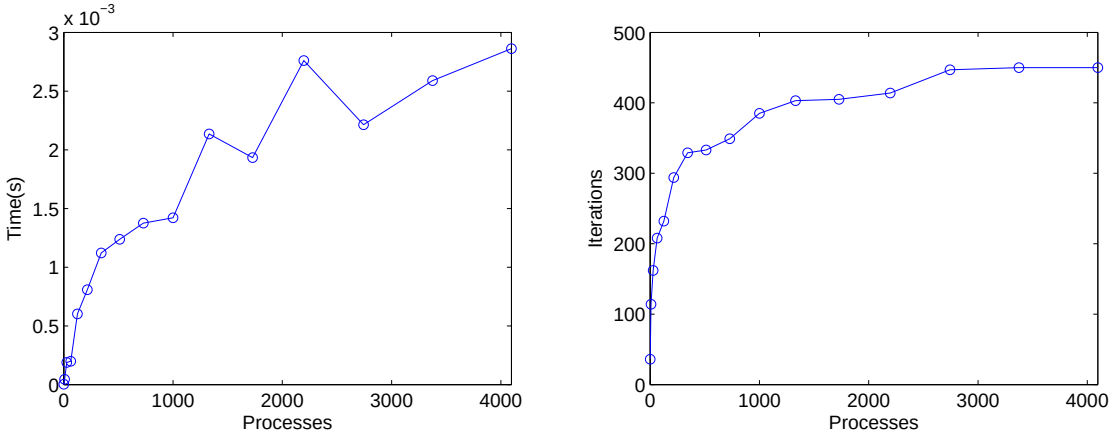

Figure 8.6: Weak scaling plot which shows that average time per iteration in MPI_Allreduce grows with machine scale (left) and total number of BICGSTAB iterations required for convergence grows with problem scale (right).

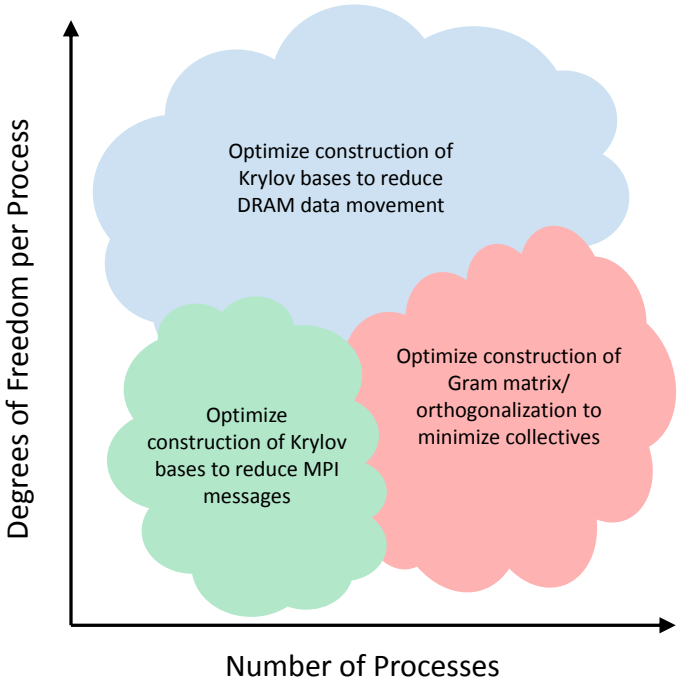

Figure 8.7: Design space for communication-avoiding optimizations in Krylov solvers

i.e., arithmetic and vertical data movement, as depicted in Figure 8.7. In these cases, other communication-avoiding approaches, like the sequential and parallel matrix powers kernels, should be considered to mitigate these costs. In general, which optimization is most beneficial depends on the problem size, nonzero structure, communication patterns, and parallel concurrency.

As an example, Figure 8.8 compares the timing breakdown of the BICGSTAB method for 4^3 degrees of freedom per process using different numbers of processes. In the legend of Figure 8.8, 'res' refers to the time to compute the initial residual $r_1 = b - Ax_1$ (line 1 in Algorithm 11) and 'appOp' refers to the time to apply the linear operator (the computation involved in application of A to p_i and d_i in lines 4 and 7 of Algorithm 11). We see that for a small number of processes, local computation dominates. But keeping the degrees of freedom per processor constant and increasing the number of processors, the MPI collectives quickly become the dominant cost.

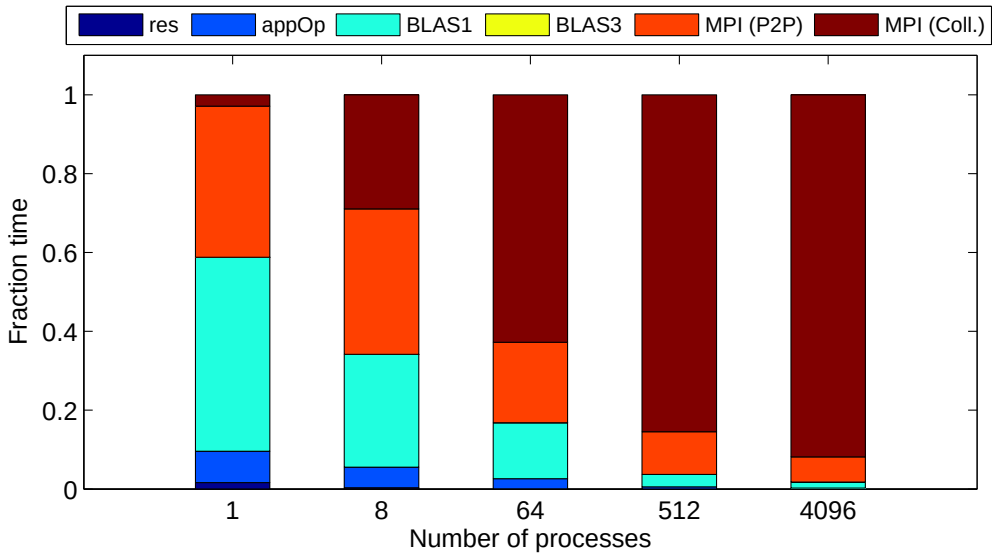

Figure 8.8: Comparison of timing breakdown for BICGSTAB solve using 1, 8, 64, 512, and 4096 MPI processes for 4^3 degrees of freedom per process.

8.3.6 CA-BICGSTAB Implementation

In order to alleviate the bottleneck caused by MPI collective communication in the coarse grid solve, we replace the BICGSTAB solver with a CA-BICGSTAB solver. See Section 4.4 for an overview of the BICGSTAB method and derivation of the CA-BICGSTAB method (Algorithm 12).

Our implementation differs from Algorithm 12 in that we add in two convergence checks; one to test the 2-norm of the intermediate residual after line 11 and one to test the 2-norm of the residual after line 14. Since we use the 2-norm, these quantities can be computed in a communication-avoiding way in each inner loop by computing

$$\|q_{sk+j}\|_2 = (d_j'^T G_k d_j')^{1/2}, \quad \text{and}$$
$$\|r_{sk+j+1}\|_2 = (r_{j+1}'^T G_k r_{j+1}')^{1/2}.$$

As discussed in Section 8.3.5, the CA-BICGSTAB method has potential performance benefits in three areas: reducing the number of MPI collectives, reducing the number of P2P messages, and reducing vertical (DRAM) data movement. The interprocess communication only occurs in lines 4 and 5 in Algorithm 12. The Krylov bases computed in line 4 can be computed either by a series of SpMV calls or by a call to the communication-avoiding matrix powers kernel (see Section 3.2). Note that it is possible to compute both bases with p and r at the same time, which halves the number of messages but doubles the size of the messages. The current infrastructure of miniGMG does not permit exchanging ghost zones or applying A to multiple vectors at a time. Therefore our implementation of CA-

BICGSTAB in miniGMG currently uses the approach of computing a series of SpMV calls, with separate computation of bases with p and r. We expect this to double the time spent in P2P communication and SpMV computation; this could be optimized in future implementations by computing bases with p and r simultaneously. While a matrix powers approach may be beneficial in some cases, it was not shown to improve performance significantly for miniGMG. Line 5 is implemented by having each process aggregate the partial sums into a matrix locally and then performing one MPI_Allreduce on a matrix of size $(4s+1) \times (4s+2)$.

As some multigrid solves are easy and others hard (in terms of iterations until convergence), our implementation uses a "telescoping" approach in which we begin using $s = 1$ and increase s by 1 in each subsequent outer loop until we reach some $s_{\max}$. In this way, easy solves do not incur the initial costs of computing the Krylov bases and performing a larger Allreduce, whereas harder solves amortize these costs and see the asymptotic benefits. Due to the local small subdomain sizes, we found that performance began to degrade using s higher than 4, so $s_{\max} = 4$ was used for miniGMG. With an s this small, a monomial basis (see Section 3.2.5) was sufficient to maintain convergence close to that of the classical method.

8.3.7 CA-BICGSTAB Performance

Figure 8.9 shows the performance of miniGMG and CA-BICGSTAB bottom solver performance compared to those using BICGSTAB as we weak scale to 4096 processes. Again, since each process uses 6 cores, the final data point corresponds to using $24,576$ cores on a 1024^3 problem. This same data plotted in terms of speedup is shown in Figure 8.10. We can see that using 4096 processes, the reduction in MPI_Allreduce calls enabled by CA-BICGSTAB results in a 4.2× speedup in the bottom solve and improves the overall multigrid solve time by nearly 2.5×.

It is also useful to look at performance in terms of degrees of freedom solved per second in the overall multigrid solve. This data is shown in Figure 8.11 for multigrid with both CA-BICGSTAB and BICGSTAB bottom solvers as we weak scale the problem. We can see from this plot that the aggregate performance of multigrid with CA-BICGSTAB improves nearly linearly with respect to multigrid with BICGSTAB.

Figure 8.12 shows a breakdown of the time spent on different operations in CA-BICGSTAB and BICGSTAB. Again, in the legend of Figure 8.12, 'res' refers to the time to compute the initial residual $r_1 = b - Ax_1$ (line 1 in Algorithm 12) and 'appOp' refers to the time to apply the linear operator (the computation involved in the $O(s)$ applications of A to vectors p_{sk+1} and r_{sk+1} in computing $\mathcal{P}_k$ and $\mathcal{R}_k$ in line 4 of Algorithm 12). As expected the sequential computation of the Krylov bases resulted in a doubling of the time spent in P2P communication and time required to apply the linear operator, although this was not enough to negate speedups gained from a reduced number of MPI collectives. The classical BICGSTAB algorithm involves $6s$ MPI_Allreduce calls per inner loop compared to one required for CA-BICGSTAB. The reason we do not see a 24× reduction in MPI collective time is that the size of the reductions has increased from one 8-byte double precision value

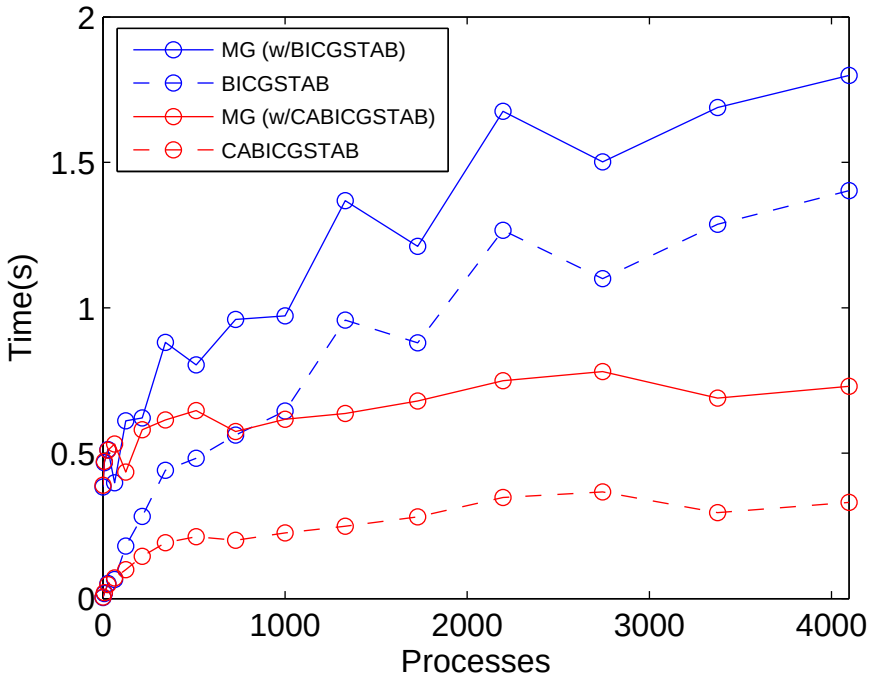

Figure 8.9: Weak scaling plot showing the timings for BICGSTAB, over multigrid time using BICGSTAB, CA-BICGSTAB, and overall multigrid time using CA-BICGSTAB.

to 2448 bytes (the size of the Gram matrix for $s = 4$). This limits the reduction in collective time to only 11.2×. A greater speedup would be obtained if collective performance were always latency-limited regardless of message size.

As previous mentioned, using $s = 4$ meant that a monomial basis was sufficient for maintaining convergence close to that of the classical method. This is demonstrated in Figure 8.13, which compares the max norm of the residual on the finest grid after each V-cycle using BICGSTAB and CA-BICGSTAB.

We finally note that our CA-BICGSTAB implementation requires $4s - 2$ additional grids to store the computed Krylov bases and thus requires additional memory. This overhead may be prohibitive for weak-scaled Krylov solves on fine grids, but is negligible in the context of multigrid bottom solvers (around 24KB per process with $s = 4$).

8.3.8 Scientific Applications

Using the miniGMG implementation of CA-BICGSTAB as a reference implementation, both C++ and Fortran versions of CA-BICGSTAB were implemented in the BoxLib AMR framework [1]. As in miniGMG, the BoxLib C++ solver uses a telescoping s value, the monomial basis, and $s_{\max} = 4$. As in the synthetic application, experiments on the bottom solves in

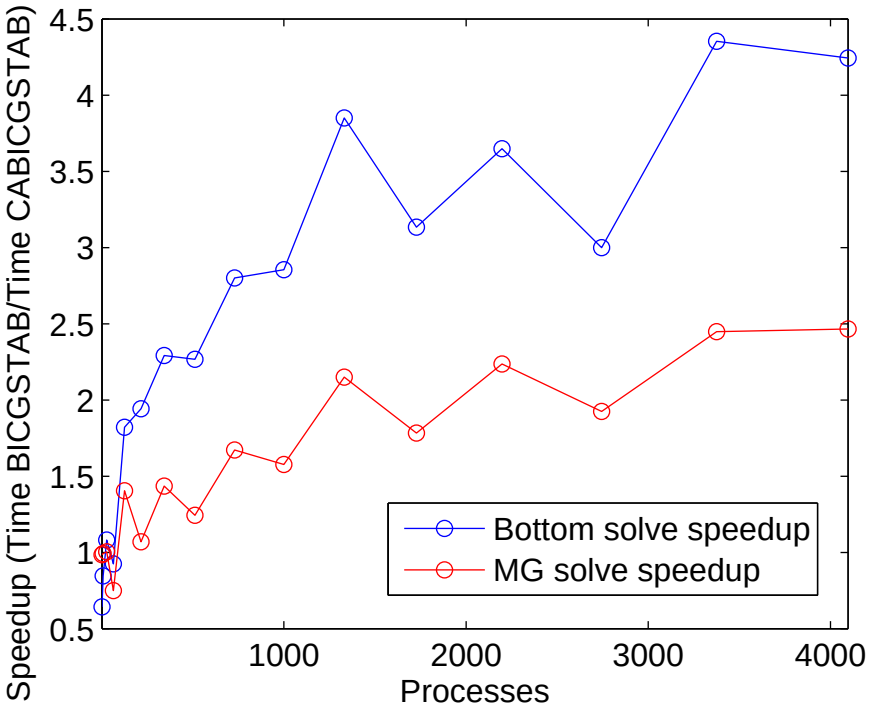

Figure 8.10: Speedups using CA-BICGSTAB with $s = 4$ over BICGSTAB in terms of both bottom solve time and overall multigrid solve time, for various numbers of processes (weak scaling problem size).

real applications showed that using s larger than 4 with the monomial basis necessitated additional V-cycles, which negate the performance advantages of a communication-avoiding bottom solver.

One difference is that, whereas miniGMG coarsened to a subdomain size of 4^3, BoxLib applications coarsen to a subdomain size of 2^3. This factor of 8 reduction in problem size can significantly reduce the number of BICGSTAB iterations. This means that we expect to see a decrease in the relative benefit of CA-BICGSTAB, which shows the best speedups when additional costs are amortized over many iterations.

In subsections below, we discuss the use of the CA-BICGSTAB bottom solver in two BoxLib applications, the Low Mach Number Combustion Code (LMC) and Nyx, a 3D N-body and gas dynamics code.

8.3.8.1 LMC

The LMC application simulates gas-phase combustion using a low Mach number model, coupled to detailed chemical reaction networks and differential diffusion [58]. Each timestep requires two different solves using multigrid on block-structured AMR grids. The first type of solve requires only a couple of iterations in the bottom solve routine. We thus focus

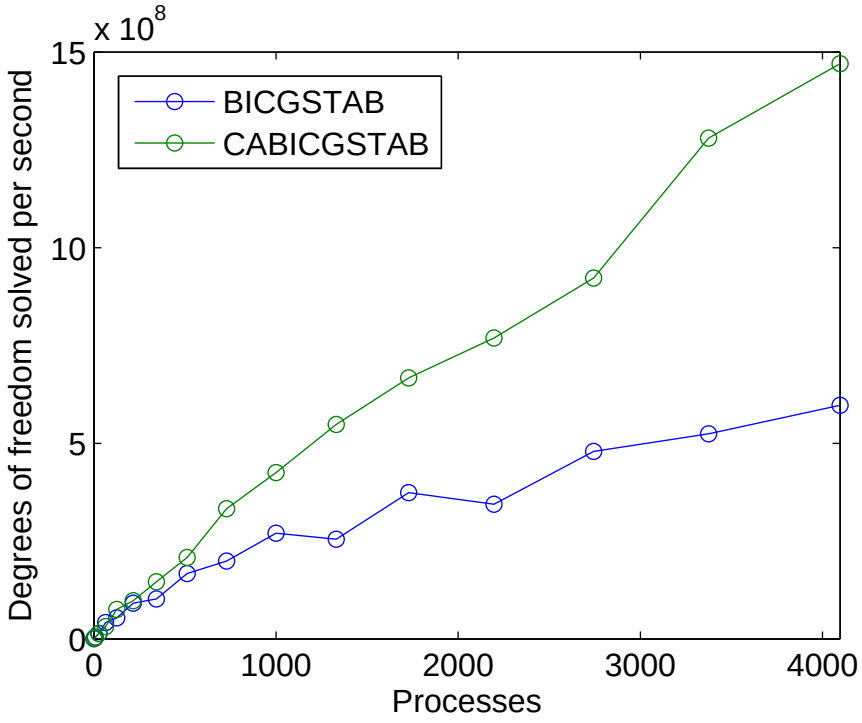

Figure 8.11: Performance in degrees of freedom solved per second (DOF/s) using multigrid with a BICGSTAB bottom solver and multigrid with a CA-BICGSTAB bottom solver.

on the second type of solve, the mac_project solve, which involves the elliptic equation $b\nabla \cdot \beta\nabla u = f$, where u represents the concentration of a chemical species, b is a constant, and β varies spatially. The discretization of this operator is a 7-point variable-coefficient stencil. These solves are more difficult than the first, often involving 10 or more V-cycles and hundreds of iterations in the bottom solve, making them ideal candidates to benefit from communication-avoiding methods.

Figures 8.14 and 8.15 show the speedup in the mac_project solve in LMC resulting from replacing the BICGSTAB bottom solve with CA-BICGSTAB for 3D and 2D problems, respectively. These are weak scaling tests run on Hopper, using both flat MPI and MPI+OpenMP with 6 threads per process. We do not run tests using MPI+OpenMP for the 2D problem, as OpenMP is not appropriate for BoxLib-based 2D AMR applications. We use initial per-process subdomain sizes of 64^3 in the 3D case and 64^2 in the 2D case. The speedup numbers are based on our measurement of the total time spent in the mac_project solve across 5 time steps.

At maximum concurrency, we see up to a 2.5× speedup in the bottom solver which corresponds to a 1.5× speedup in the 3D mac_project solve. For the 2D mac_project solve, the benefits of CA-BICGSTAB are more immediate, with a 1.5× speedup on 64 cores. We note that speedups obtained in the overall solve are mitigated by the fact that the classical

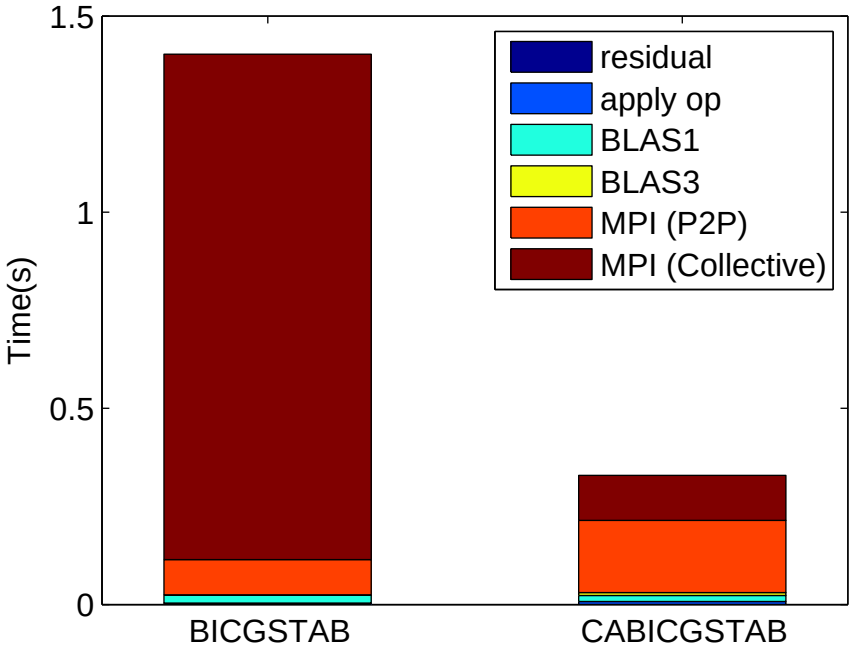

Figure 8.12: Breakdown of the net time spent across all bottom solves using $24,576$ cores for the BICGSTAB method (left) and CA-BICGSTAB with $s = 4$ (right).

algorithm constitutes less than 43% and 54% of the 3D mac_project solve time for flat MPI and MPI+OpenMP version, respectively, and less than 35% of the solve time in the 2D problem. In all cases, CA-BICGSTAB converged at a similar rate as BICGSTAB; the total number of V-cycles required was unchanged.

8.3.8.2 Nyx

Nyx is a 3D N-body and as dynamics code that uses AMR for large-scale cosmological simulations; see [7]. Nyx tracks the time evolution of a system of discrete dark matter particles gravitationally coupled to an inviscid ideal fluid in an expanding universe. The mass of the dark matter particles is deposited on the AMR grid hierarchy using a cloud-in-cell scheme and converted to a density field, which is added to the gas density. This total density defines the right-hand side of a constant-coefficient Poisson equation $b\nabla^2 u = f$ solving for the gravitational potential. The gradient of the gravitational potential, $-\nabla u$, is the gravitational force vector which is used to accelerate both the dark matter particles and the gas. The right-hand side and the gravitational potential are defined on cell centers and a 7-point discretization of the Laplacian operator is used (the stencil coefficients are modified at coarse/fine interfaces).

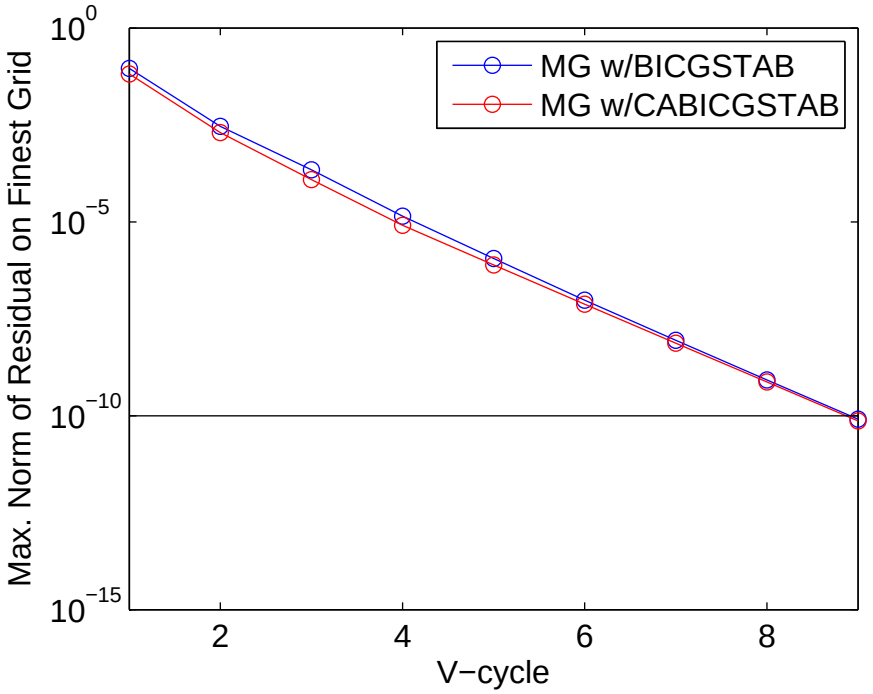

Figure 8.13: Max norm of the residual on the finest grid after each V-cycle with either the classical BICGSTAB or CA-BICGSTAB ($s = 4$) bottom solver.

Figure 8.16 shows the speedup in the gravity solve in Nyx resulting from replacing the BICGSTAB bottom solve with CA-BICGSTAB. Again, use we two programming models, both flat MPI and hybrid MPI+OpenMP. As in other tests, each process owns a 64^3 subdomain on the fine grid. The speedup numbers are based on our measurement of the total time spent in the gravity solve across 6 time steps. As for LMC, here CA-BICGSTAB converged at a similar rate as BICGSTAB; the total number of V-cycles required was unchanged.

For flat MPI, we see a 2× speedup in the bottom solver using CA-BICGSTAB at 4096 cores. However, as the bottom solver only constitutes 26% of the overall multigrid solve time in this case, the overall speedup was limited to 1.15×. In the MPI+OpenMP case, the bottom solver took as much as 41% of the total multigrid solve time, resulting in a 1.1× speedup in the overall solve corresponding to a 1.6× speedup in the bottom solver.

8.3.9 Summary and Future Work

Replacing the BICGSTAB method with CA-BICGSTAB resulted in up to a 4.2× speedup in bottom solve performance in the miniGMG benchmark and up to 2.5× in real applications. For real BoxLib applications, these bottom solver speedups led to a 1.5× improvement in overall multigrid solve time on 24,576 cores. Future work will include use of communication-

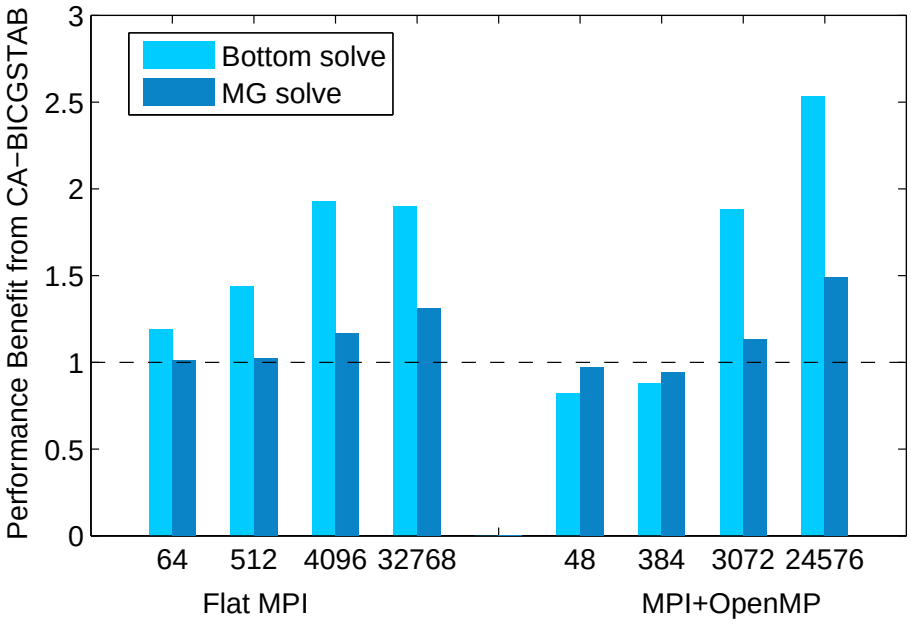

Figure 8.14: Speedup of mac_project using CA-BICGSTAB in the 3D version of LMC, as a function of the number of cores used and programming model.

avoiding bottom solvers in other applications. We expect our approach to have the most benefit (best speedups) when the bottom solve constitutes a very large percentage of overall multigrid runtime and the multigrid runtime constitutes a very large percentage of overall application runtime. Applications with such characteristics will be our target in the future.

Our tests show that although communication-avoiding Krylov methods can asymptotically reduce the number of collectives, performance is still limited by nontrivial time spent in P2P communication and a quadratic increase in size of collective communications and vector operations. Performance can also be limited since roundoff error in finite precision limits s to 4 using the monomial basis on solves in real applications. In the future, it would be beneficial to implement more well conditioned polynomial bases such as Newton or Chebyshev (see Section 3.2.5) as well as a communication-avoiding matrix powers kernel to evaluate whether these improve overall runtime.

In this section, we limited ourselves to weak scaling tests, but strong scaling tests would also be beneficial. We expect communication-avoiding Krylov methods to allow increased performance/scalability in the strong-scaled case; as one strong scales a solver, eventually a point will be reached where where the problem size per process is so small that communication (likely collectives) becomes the bottleneck.

Communication-avoiding Krylov methods expand the 'co-design' space, where the hardware design space is explored in tandem with software optimizations, by allowing hardware

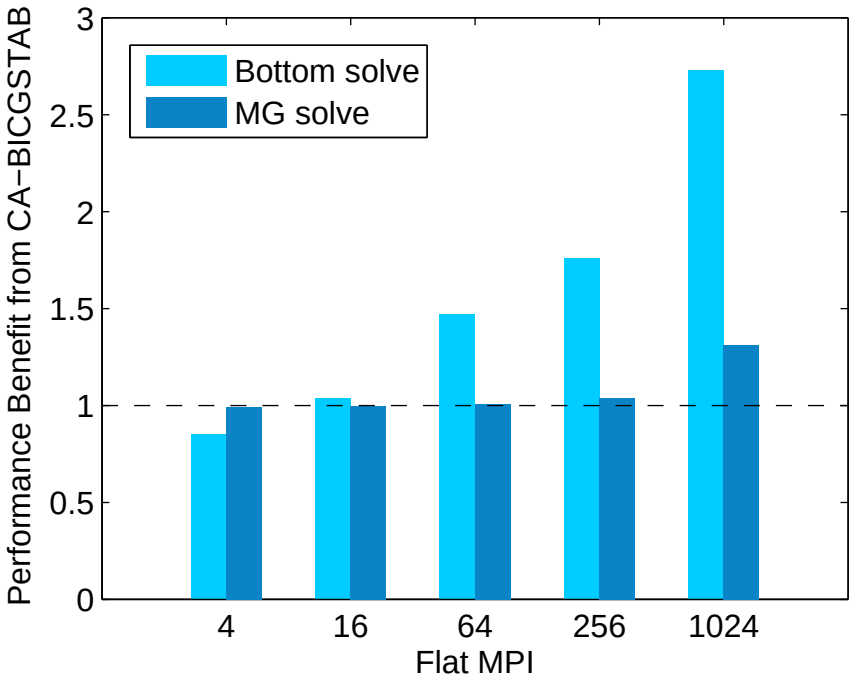

Figure 8.15: Speedup of mac_project solver speedup using CA-BICGSTAB in the 2D version of LMC, as a function of the number of cores used for the Flat MPI programming model.

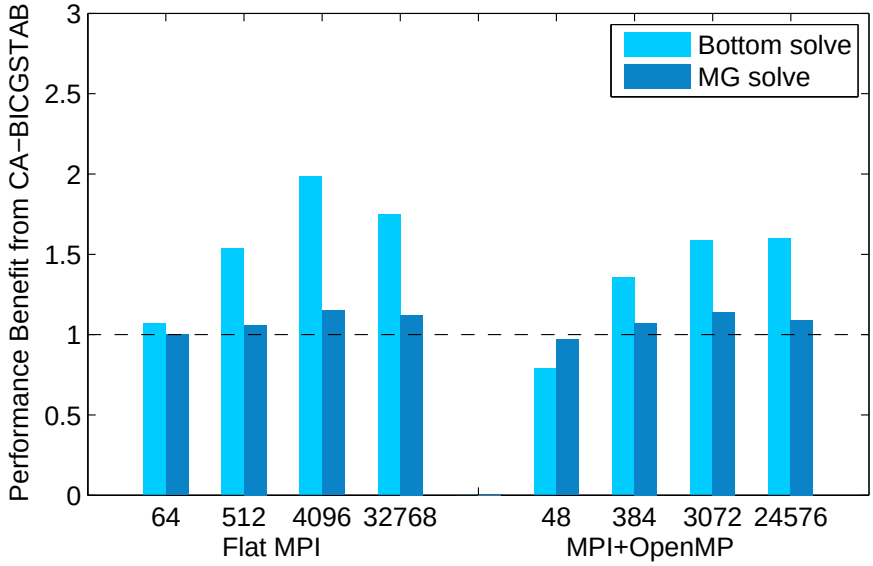

Figure 8.16: Gravity solve speedup from using CA-BICGSTAB in the Nyx application, as a function of the number of cores used and programming model.

and software designers to trade collective latency for bandwidth, i.e., we trade $O(s)$ fine-grained operations for one large coarse-grained operation that expresses more parallelism and may be more appropriate given manycore trends in processor architecture. Along these lines, future work could expand implementations to GPU and Xeon Phi processors to evaluate whether CA-KSMs can enable efficient use of those architectures.

We briefly comment on the usefulness of the performance model described in Section 2.2 in optimizing real applications. While this model is useful in terms of qualitatively comparing the communication-avoiding and classical approaches, it is not necessarily useful for parameter selection in practical applications. For example, in this section, the decision to use telescoping s values was not based solely on performance tradeoffs or numerical instability, but instead by the very few number of iterations required to meet the convergence criteria for some solves in the application. We therefore advocate taking a holistic approach, using performance models as a general guideline while considering the requirements and properties of particular application.

Chapter 9

Conclusions

Krylov subspace methods, ubiquitous throughout scientific applications, are often the most time-consuming kernels in large-scale simulations due to communication-bound performance. In this thesis, we have focused on the development and analysis of high-performance communication-avoiding Krylov subspace methods, which can achieve asymptotic reductions in data movement versus classical approaches.

In Chapter 4, we have developed many new communication-avoiding variants of Lanczos-based Krylov subspace methods, including solvers for nonsymmetric linear systems, least squares problems, and eigenvalue/singular value problems. In both sequential and parallel implementations, these variants of popular Krylov methods asymptotically reduce data movement by a factor of $\Theta(s)$ versus the classical algorithms.

In Chapter 5, we have addressed major challenges to the practical use of communication-avoiding Krylov subspace methods, identifying the conditions under which communication-avoiding Krylov methods can achieve both asymptotic speedups and maintain the numerical properties of the classical method. Our results confirm theoretically what is well-known: the conditioning of the precomputed Krylov bases plays a large role in determining finite precision behavior. In particular, if one can guarantee that the condition numbers of the precomputed s-step Krylov bases are not too large in any iteration, then the finite precision behavior of the CA-KSM will be similar to that of the classical KSM, in terms of both convergence rate and accuracy. These results can be used to provide guidance on whether or not a certain application might benefit from the use of the communication-avoiding approach, and also inspire potential techniques for improvement.

Based on theoretical bounds proved in Chapter 5, Chapter 6 has been devoted to the development of a number of efficient numerical techniques for improving the convergence and accuracy in communication-optimal Krylov methods while still achieving performance gains. Some of these techniques, including extended precision, dynamic parameter refinement, and variable basis sizes, follow from the finite precision analyses of Chapter 5. Other techniques, including residual replacement, deflation-based preconditioning, selective reorthogonalization, and look-ahead are techniques that have been developed for classical Krylov methods.

In Chapter 7, we have presented optimizations that can be applied to CA-KSMs when

the input matrix A has certain nonzero structures, including data-sparse and hierarchical matrices, stencil/implicitly represented matrices, and matrices with highly nonsymmetric nonzero structure.

Finally in Chapter 8, we have presented performance results for high-performance distributed-memory implementations of communication-avoiding Krylov methods on large-scale problems run on the Hopper supercomputer. We performed weak and strong scaling experiments for distributed-memory CA-CG on a Poisson model problem to identify the space in which the communication-avoiding approach is most beneficial in terms of performance, and discussed the implementation and optimization of the CA-BICGSTAB method as a high performance, distributed-memory bottom solver for geometric multigrid, which enabled speedups of over $4\times$ on synthetic benchmarks and over $2.5\times$ in real applications.

To conclude, we will summarize the contributions of this thesis in the context of the overall landscape of current and future research in the design and implementation of high-performance iterative solvers. To do so, we will outline three major challenges to the practical use of CA-KSMs and discuss progress and future work toward overcoming these difficulties. We finish with commentary on situations for which CA-KSMs are suitable for use in practice and when they are expected to have the greatest performance benefit.

9.1 Challenges to the Practical Use of CA-KSMs

Despite potential performance gains, CA-KSMs have faced challenges to integration into standard scientific computing libraries for a few practical reasons. In this section we address what we believe to be the three biggest challenges to the practical use of CA-KSMs. For each, we discuss how we have already overcome, or are in the process of overcoming, these obstacles.

CA-KSMs Can Be Unstable and/or Convergence is Too Delayed It is well-known that the convergence and stability properties of classical Krylov methods are not necessarily maintained by communication-avoiding Krylov methods in finite precision. Depending on the parameter s and the polynomials used in constructing the s-step bases, the rate of convergence can be slower and the attainable accuracy lower than for the classical method. Lost accuracy can be a significant problem depending on the needs of the application, and slow convergence can negate potential benefits from the communication-avoiding approach.

In this thesis, particularly in Chapter 5, we have answered many of the open questions concerning how the finite precision behavior of CA-KSMs differs from their classical counterparts. We have shown that bounds on convergence and accuracy for CA-KSMs can be written in the same form as the equivalent bounds for KSMs, with the addition of a multiplicative factor that depends solely on the conditioning of the s-step bases. This gives reassurance that, as long as well-conditioned polynomials are used for basis construction and s is chosen reasonably, the convergence and accuracy of the CA-KSM will be similar to the classical KSM for the same problem.

This thesis has also explored many potential techniques for improving the stability and convergence rate can be implemented in CA-KSMs, described in Chapter 6. Some of these follow directly from our analyses in Chapter 5 as methods for either improving the basis conditioning, e.g., variable basis sizes, or lowering the amplification factor by other means, e.g., using extra precision. Others are based on techniques originally developed for KSMs, including residual replacement, deflation, and selective reorthogonalization, which we demonstrate are compatible with our communication-avoiding approach.

Preconditioning is Incompatible with Avoiding Communication Preconditioning is considered essential for most practical uses of KSMs. As mentioned, the goals of reducing the cost per iteration via communication-avoiding techniques and reducing the number of iterations via preconditioning are somewhat at odds and involve complicated, problem-dependent tradeoffs.

There have been strides toward the integration of many preconditioners into CA-KSMs. Preconditioners which have been shown to work in the communication-avoiding setting include diagonal matrices, CA-ILU(0) for structured grids [89], sparse approximate inverse preconditioners [121], and domain decomposition preconditioners [195]. In Section 7.1 of this thesis, we have shown that it is possible to use HSS preconditioners in CA-KSMs when A is a banded matrix (and more generally, if the preconditioned system can be written as the sum of sparse and low-rank components). We have also demonstrated in Section 6.2 that it is possible to implement communication-avoiding deflation, which is like preconditioning but with a singular preconditioner.

Many commonly-used preconditioners were not designed specifically with the goal of avoiding communication in mind. In cases where the dependencies in the resulting preconditioned system are too dense, attempting to force these preconditioners into the CA-KSM mold is a futile effort. This has led some to suggest that CA-KSMs are not suitable for use in practice, as use of these preconditioners is necessary.

We advocate for an alternative view: we should instead focus on the invention of new preconditioners which allow us to balance the tradeoff between improving convergence rate and avoiding communication. Similar to the paradigm shift towards the redesign of algorithms to avoid communication, there is already a growing trend toward the redesign of preconditioners with communication-avoidance in mind; see, e.g., the underlapping technique of Boman et al. [195]. We expect that the development of new preconditioners that avoid communication, as well as modification of existing preconditioners to eliminate communication while still providing some convergence benefit, will be a fruitful area of research in coming years.

Performance and Numerical Behavior is Too Sensitive to Parameter Selection Many complex implementation decisions are required for both matrix powers kernel performance and convergence of the CA-KSMs. The optimal parameters and optimizations to apply will depend on the machine, the numerical values and structure of the system ma-

trix, and the particular method. This makes auto-tuning and code generation an attractive approach.

Most auto-tuning work has focused on optimization of kernel performance; for SpMV and matrix powers computations, this means optimizing based on matrix structure and machine parameters. There are many ongoing projects towards the development of auto-tuners and specializers for SpMV and matrix powers computations (see, e.g., [26, 39, 130, 172]). However, as we have tried to stress throughout this thesis, in the context of the overall Krylov solve we need to be concerned not only with the speed of the individual kernels but also with the rate of convergence and accuracy.

In light of our work in evaluating tradeoffs between speed per iteration and number of iterations in sparse matrix solvers, we strongly advocate for what we call '*numerical auto-tuning*'. This involves the development of an auto-tuned/specialized library for sparse iterative methods that takes numerical properties of the input matrix, right-hand side, and accuracy requirements of the application into account in additional to the usual considerations of non-zero structure/machine parameters. Such tools could automatically select an appropriate code variant that achieves both performance and stability based on matrix structure, eigenvalue estimates, and other properties of the matrix. This would lift the burden of expertise from the user, making CA-KSMs more accessible to a wider range of computational scientists.

For example, for CA-KSMs, numerical auto-tuning would involve automatically selecting the best subspace size, the best polynomial basis to use in generating the subspace, and deciding which subset of numerical optimizations to apply (e.g., residual replacement, deflation, reorthogonalization, restarting, etc.) based on the numerical properties of the problem and application-specific requirements. Of course, we still have the usual combinatorial challenges of auto-tuning the sparse matrix kernels and partitioning the input matrix.

This endeavor would require interdisciplinary collaboration between computer scientists, mathematicians, and application experts. Some challenges include understanding the behavior of non-normal matrices/operators and the convergence of iterative methods for nonsymmetric systems, the algorithmic design of new communication-avoiding methods for various preconditioners and techniques like reorthogonalization, and the study of how different optimizations interact with each other numerically and algorithmically (e.g., can we easily combine residual replacement and reorthogonalization?). There is also the challenge of predicting what effect various techniques will have on performance, and determining whether the extra flops/messages increase the convergence rate enough to be beneficial in terms of overall performance.

9.2 Are CA-KSMs Practical?

We now return to the question of when communication-avoiding Krylov subspace methods are suitable for use in practice. Here suitability requires that the communication-avoiding variants can obtain speedups over the classical implementations, i.e., that any extra imple-

mentation effort required for their use is justified. It is clear that in order to warrant use of CA-KSMs, we must have a situation where the Krylov solve is communication-bound and the Krylov solve constitutes the bottleneck within the particular application. Additionally, in our experience, CA-KSMs are good candidates for use when one or more of the following hold:

- The problem is communication-bound (and particularly latency-bound in the parallel case);
- The matrix is well-partitioned, or can be split into well-partitioned and low-rank components;
- Simple preconditioning is sufficient/the required preconditioner is amenable to communication avoidance (see Section 6.8);
- Extremal eigenvalues are known or easy to estimate; and/or
- The same system matrix will be reused over multiple solves.

There are many problems in a wide variety of science and engineering applications which fall into this category and can thus be expected to benefit from the communication-avoiding approach. Broadly speaking, these mostly involve the solution of PDEs by finite difference or finite element methods. In addition to the combustion and cosmology applications studied in Section 8.3, our approach can be applied to other complex fluid flow problems. For example, deflated CG with diagonal preconditioning is used for solving time-dependent diffusion equations in the simulation of oil and gas resevoirs [183]; we believe our deflated CA-CG method presented in Section 6.2 could be immediately applied. Other uses of the finite element method which have been shown to benefit from CA-KSMs include structural engineering problems and electromagnetic analysis [194]. Poisson's equation, which we have used in both numerical and performance tests in this thesis, is an elliptic PDE with applications in electrostatics, mechanical engineering, and theoretical physics. As shown in Section 8.2, the use of CA-KSMs for these types of problems can result in up to $6\times$ speedups. Another potential application of CA-KSMs is in circuit simulation, which involves the solution of differential-algebraic equations [194].

To conclude, it is clear that the design and implementation of iterative solvers requires a holistic approach. In selecting the right method and parameters to use for a given problem, we must consider the expected time per iteration, which depends on the matrix dimension, nonzero structure, and parameters of the specific machine to be used, as well as the numerical stability and convergence properties, which we have seen depend on numerical properties of the system matrix and the machine precision. We must also consider the performance and use of the Krylov method within the context of the overall scientific application. While CA-KSMs will not be the best solver choice for all problems and platforms, there are numerous cases in a variety of scientific domains for which the communication-avoiding approach can offer significant savings in both performance and energy.

Bibliography

[1] BoxLib website. https://ccse.lbl.gov/BoxLib.

[2] Chombo website. https://seesar.lbl.gov/ANAG/software.html.

[3] hypre website. http://computation.llnl.gov/casc/hypre/software.html.

[4] PETSc website. http://www.mcs.anl.gov/petsc/.

[5] A. Abdel-Rehim, R. Morgan, D. Nicely, and W. Wilcox. Deflated and restarted symmetric Lanczos methods for eigenvalues and linear equations with multiple right-hand sides. *SIAM J. Sci. Comput.*, 32(1):129–149, 2010.

[6] H. Adeli and R. Soegiarso. *High Performance Computing in Structural Engineering*, volume 3. CRC Press, 1998.

[7] A. Almgren, J. Bell, M. Lijewski, Z. Lukić, and E. Van Andel. Nyx: A massively parallel AMR code for computational cosmology. *The Astrophysical Journal*, 765(1):39, 2013.

[8] S. Ashby, P. Beckman, J. Chen, P. Colella, B. Collins, D. Crawford, J. Dongarra, D. Kothe, R. Lusk, P. Messina, T. Mezzacappa, P. Moin, M. Norman, R. Rosner, V. Sarkar, A. Siegel, F. Streitz, A. White, and M. Wright. The opportunities and challenges of exascale computing. Technical report, U.S. Dept. of Energy Office of Science, Advanced Scientific Computing Advisory Committee, 2010.

[9] T. Ashby, P. Ghysels, W. Heirman, and W. Vanroose. The impact of global communication latency at extreme scales on Krylov methods. In *Algorithms and Architectures for Parallel Processing*, pages 428–442. Springer, 2012.

[10] O. Axelsson. *Iterative solution methods*. Cambridge University Press, 1996.

[11] J. Baglama, D. Calvetti, G. Golub, and L. Reichel. Adaptively preconditioned GMRES algorithms. *SIAM J. Sci. Comput.*, 20(1):243–269, 1998.

[12] Z. Bai. Error analysis of the Lanczos algorithm for the nonsymmetric eigenvalue problem. *Math. Comp.*, 62(205):209–226, 1994.

[13] Z. Bai, D. Day, J. Demmel, and J. Dongarra. A test matrix collection for non-hermitian eigenvalue problems. Technical Report CS-97-355, Dept. of CS, University of Tennessee, 1997.

[14] Z. Bai, D. Hu, and L. Reichel. A Newton basis GMRES implementation. *IMA J. Numer. Anal.*, 14(4):563–581, 1994.

[15] I. Bajeux-Besnainou, W. Bandara, and E. Bura. A Krylov subspace approach to large portfolio optimization. *Journal of Economic Dynamics and Control*, 36(11):1688–1699, 2012.

[16] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz. Communication lower bounds and optimal algorithms for numerical linear algebra. *Acta Numerica*, 23:1–155, 5 2014.

[17] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Romine, and H. van der Vorst. *Templates for the solution of linear systems: building blocks for iterative methods*. SIAM, 1993.

[18] G.M. Baudet. Asynchronous iterative methods for multiprocessors. *J. ACM*, 25(2):226–244, 1978.

[19] M. Bebendorf. Hierarchical matrices. In *Hierarchical Matrices*, volume 63 of *Lecture Notes in Computational Science and Engineering*, pages 49–98. Springer Berlin Heidelberg, 2008.

[20] M. Benzi and M. Tůma. A parallel solver for large-scale Markov chains. *Applied Numerical Mathematics*, 41(1):135–153, 2002.

[21] R. Bisseling. *Parallel scientific computation*. Oxford University Press, 2004.

[22] Å. Björck, T. Elfving, and Z. Strakoš. Stability of conjugate gradient and Lanczos methods for linear least squares problems. *SIAM J. Matrix Anal. Appl.*, 19(3):720–736, 1998.

[23] S. Börm and J. Garcke. Approximating Gaussian processes with $\mathcal{H}^2$-matrices. In *Machine Learning: ECML 2007*, pages 42–53. Springer, 2007.

[24] A. Bouras and V. Frayssé. Inexact matrix-vector products in Krylov methods for solving linear systems: a relaxation strategy. *SIAM J. Matrix Anal. Appl.*, 26(3):660–678, 2005.

[25] J. Buckmaster and H. Banks. *The mathematics of combustion*, volume 2. SIAM, 1985.

[26] J. Byun, R. Lin, K. Yelick, and J. Demmel. Autotuning sparse matrix-vector multiplication for multicore. Technical Report UCB/EECS-2012-215, CS Dept., U.C. Berkeley, Nov 2012.

[27] E. Carson. Avoiding communication in the Lanczos bidiagonalization routine and associated least squares QR solver. Technical Report UCB/EECS-2015-15, EECS Department, University of California, Berkeley, Apr 2015.

[28] E. Carson and J. Demmel. Accuracy of the s-step Lanczos method for the symmetric eigenproblem. Technical Report UCB/EECS-2014-165, EECS Dept., U.C. Berkeley, Sep 2014.

[29] E. Carson and J. Demmel. Analysis of the finite precision s-step biconjugate gradient method. Technical Report UCB/EECS-2014-18, EECS Dept., U.C. Berkeley, Mar 2014.

[30] E. Carson and J. Demmel. Error analysis of the s-step Lanczos method in finite precision. Technical Report UCB/EECS-2014-55, EECS Dept., U.C. Berkeley, May 2014.

[31] E. Carson and J. Demmel. A residual replacement strategy for improving the maximum attainable accuracy of s-step Krylov subspace methods. *SIAM J. Matrix Anal. Appl.*, 35(1):22–43, 2014.

[32] E. Carson and J. Demmel. Accuracy of the s-step Lanczos method for the symmetric eigenproblem in finite precision. *SIAM J. Matrix Anal. Appl.*, 36(2):793–819, 2015.

[33] E. Carson, J. Demmel, L. Grigori, N. Knight, P. Koanantakool, O. Schwartz, and H. Simhadri. Write-avoiding algorithms. Technical Report UCB/EECS-2015-163, EECS Department, University of California, Berkeley, Jun 2015.

[34] E. Carson, N. Knight, and J. Demmel. Avoiding communication in two-sided Krylov subspace methods. Technical Report UCB/EECS-2011-93, EECS Dept., U.C. Berkeley, Aug 2011.

[35] E. Carson, N. Knight, and J. Demmel. Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods. *SIAM J. Sci. Comp.*, 35(5), 2013.

[36] E. Carson, N. Knight, and J. Demmel. An efficient deflation technique for the communication-avoiding conjugate gradient method (to appear). *ETNA*, 43, 2014.

[37] U. Catalyurek. *Hypergraph models for sparse matrix partitioning and reordering.* PhD thesis, Bilkent University, 1999.

[38] Ü. Çatalyürek and C. Aykanat. PaToH: A multilevel hypergraph partitioning tool, version 3.0. Technical Report BU-CE-9915, Computer Engineering Department, Bilkent University, 1999.

[39] B. Catanzaro, S. Kamil, Y. Lee, K. Asanovic, J. Demmel, K. Keutzer, J. Shalf, K. Yelick, and A. Fox. SEJITS: Getting productivity and performance with Selective Embedded JIT Specialization. In *Proc. 18th Int. Conf. Parallel Arch. Comp. Tech.* ACM, 2009.

[40] B. Catanzaro, B. Su, N. Sundaram, Y. Lee, M. Murphy, and K. Keutzer. Efficient, high-quality image contour detection. In *IEEE 12th International Conference on Computer Vision*, pages 2381–2388. IEEE, 2009.

[41] C. Chan, D. Unat, M. Lijewski, W. Zhang, J. Bell, and J. Shalf. Software design space exploration for exascale combustion co-design. In J. Kunkel, T. Ludwig, and H. Meuer, editors, *Supercomputing*, volume 7905 of *Lecture Notes in Comput. Sci.*, pages 196–212. Springer-Verlag, Berlin, Heidelberg, 2013.

[42] E. Chan, M. Heimlich, A. Purkayastha, and R. Van De Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007.

[43] S. Chandrasekaran, P. Dewilde, M. Gu, W. Lyons, and T. Pals. A fast solver for HSS representations via sparse matrices. *SIAM J. Matrix Anal. Appl.*, 29(1):67–81, 2006.

[44] S. Chandrasekaran, M. Gu, and T. Pals. Fast and stable algorithms for hierarchically semi-separable representations. Technical report, Dept. of Mathematics, U.C. Berkeley, 2004.

[45] A. Chapman and Y. Saad. Deflated and augmented Krylov subspace techniques. *Numer. Linear Algebra Appl.*, 4(1):43–66, 1997.

[46] M. Christ, J. Demmel, N. Knight, T. Scanlon, and K. Yelick. Communication lower bounds and optimal algorithms for programs that reference arrays - part 1. Technical Report UCB/EECS-2013-61, EECS Department, University of California, Berkeley, May 2013.

[47] A. Chronopoulos. s-Step iterative methods for (non)symmetric (in)definite linear systems. *SIAM J. Numer. Anal.*, 28(6):1776–1789, 1991.

[48] A. Chronopoulos and C. Gear. On the efficient implementation of preconditioned s-step conjugate gradient methods on multiprocessors with memory hierarchy. *Parallel Comput.*, 11(1):37–53, 1989.

[49] A. Chronopoulos and C. Gear. *s*-step iterative methods for symmetric linear systems. *J. Comput. Appl. Math*, 25(2):153–168, 1989.

[50] A. Chronopoulos and S. Kim. s-step Orthomin and GMRES implemented on parallel computers. Technical Report 90/43R, USMI, Minneapolis, MN, 1990.

[51] A. Chronopoulos and D. Kincaid. On the Odir iterative method for non-symmetric indefinite linear systems. *Numer. Lin. Alg. Appl.*, 8(2):71–82, 2001.

[52] A. Chronopoulos and C. Swanson. Parallel iterative s-step methods for unsymmetric linear systems. *Parallel Comput.*, 22(5):623–641, 1996.

[53] B. Cipra. The best of the 20th century: Editors name top 10 algorithms. *SIAM News*, 33, 2000.

[54] E. Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. System Sci.*, 55(3):441–453, 1997.

[55] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. Von Eicken. LogP: towards a realistic model of parallel computation. In *Proc. 4th Symp. Principles Pract. Parallel Program.*, pages 1–12. ACM, New York, USA, 1993.

[56] J. Cullum and W. Donath. A block Lanczos algorithm for computing the q algebraically largest eigenvalues and a corresponding eigenspace of large, sparse, real symmetric matrices. In *Proc. of the 1974 IEEE Conf. on Decision and Control*, pages 505–509. IEEE, 1974.

[57] T. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Software*, 38(1):1–25, 2011.

[58] M. Day and J. Bell. Numerical simulation of laminar reacting flows with complex chemistry. *Combustion Theory and Modelling*, 4(4):535–556, 2000.

[59] E. de Sturler. A parallel variant of GMRES(m). In *Proceedings of the 13th IMACS World Congress on Computational and Applied Mathematics. IMACS, Criterion Press*, volume 9, 1991.

[60] E. De Sturler. Truncation strategies for optimal Krylov subspace methods. *SIAM J. Numer. Anal.*, 36(3):864–889, 1999.

[61] E. De Sturler and H. van der Vorst. Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *Appl. Numer. Math.*, 18(4):441–459, 1995.

[62] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM J. Sci. Comput.*, 34:A206–A239, 2012.

[63] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in computing Krylov subspaces. Technical Report UCB/EECS-2007-123, EECS Dept., U.C. Berkeley, Oct 2007.

[64] K. Devine, E. Boman, R. Heaphy, R. Bisseling, and Ü. Çatalyürek. Parallel hypergraph partitioning for scientific computing. In *20th IEEE International Parallel and Distributed Processing Symposium: IPDPS 2006*. IEEE, 2006.

[65] J. Dongarra, J. Hittinger, J. Bell, L. Chacón, R. Falgout, M. Heroux, P. Hovland, E. Ng, C. Webster, and S. Wild. Applied mathematics research for exascale computing. Technical report, U.S. Dept. of Energy, Office of Science, Advanced Scientific Computing Research Program, 2014.

[66] Z. Dostál. Conjugate gradient method with preconditioning by projector. *Int. J. Comput. Math.*, 23(3-4):315–323, 1988.

[67] M. Engeli, T. Ginsburg, H. Rutishauser, and E. Stiefel. *Refined iterative methods for computation of the solution and the eigenvalues of self-adjoint boundary value problems*. Springer, 1959.

[68] J. Erhel. A parallel GMRES version for general sparse matrices. *Electronic Transactions on Numerical Analysis*, 3(12):160–176, 1995.

[69] R. Fletcher. Conjugate gradient methods for indefinite systems. *Lecture Notes in Math.*, 506:73–89, 1976.

[70] J. Frank and C. Vuik. On the construction of deflation-based preconditioners. *SIAM J. Sci. Comput.*, 23(2):442–462, 2001.

[71] R. Freund and N. Nachtigal. QMR: a quasi-minimal residual method for non-Hermitian linear systems. *Numerische Mathematik*, 60(1):315–339, 1991.

[72] Roland W Freund. A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. *SIAM J. Sci. Comput.*, 14(2):470–482, 1993.

[73] A. Gaul, M. Gutknecht, J. Liesen, and R. Nabben. A framework for deflated and augmented Krylov subspace methods. *SIAM J. Matrix Anal. Appl.*, 34(2):495–518, 2013.

[74] W. Gautschi. The condition of polynomials in power form. *Math. Comp*, 33(145):343–352, 1979.

[75] J. Gentle. *Computational statistics*, volume 308. Springer, 2009.

[76] J. Gentle, W. Härdle, and Y. Mori. *Handbook of computational statistics: concepts and methods*. Springer Science and Business Media, 2012.

[77] P. Ghysels, T. Ashby, K. Meerbergen, and W. Vanroose. Hiding global communication latency in the GMRES algorithm on massively parallel machines. *SIAM J. Sci. Comput.*, 35(1):C48–C71, 2013.

[78] P. Ghysels, P. Kłosiewicz, and W. Vanroose. Improving the arithmetic intensity of multigrid with the help of polynomial smoothers. *Numer. Lin. Alg. Appl.*, 19(2):253–267, 2012.

[79] P. Ghysels and W. Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Computing*, 40(7):224–238, 2014.

[80] P. Ghysels, W. Vanroose, and K. Meerbergen. High performance implementation of deflated preconditioned conjugate gradients with approximate eigenvectors. In *Householder Symposium XIX June 8-13, Spa Belgium*, page 84, 2014.

[81] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial & Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[82] G. Golub and Z. Strakoš. Estimates in quadratic formulas. *Numerical Algorithms*, 8(2):241–268, 1994.

[83] G. Golub and C. Van Loan. *Matrix computations*. Johns Hopkins University Press, Baltimore, MD, 3 edition, 1996.

[84] G. Golub and C. Van Loan. *Matrix computations*, volume 3. Johns Hopkins University Press, 2012.

[85] A. Greenbaum. Behavior of slightly perturbed Lanczos and conjugate-gradient recurrences. *Lin. Alg. Appl.*, 113:7–63, 1989.

[86] A. Greenbaum. Estimating the attainable accuracy of recursively computed residual methods. *SIAM J. Matrix Anal. Appl.*, 18:535–551, 1997.

[87] A. Greenbaum. *Iterative methods for solving linear systems*, volume 17. SIAM, 1997.

[88] A. Greenbaum and Z. Strakoš. Predicting the behavior of finite precision Lanczos and conjugate gradient computations. *SIAM J. Matrix Anal. Appl.*, 13:121–137, 1992.

[89] L. Grigori and S. Moufawad. Communication avoiding ILU(0) preconditioner. Rapport de recherche RR-8266, INRIA, March 2013.

[90] W. Gropp. Update on libraries for Blue Waters. http://jointlab.ncsa.illinois.edu/events/workshop3/pdf/presentations/Gropp-Update-on-Libraries.pdf, 2010. Bordeaux, France.

[91] T. Gu, X. Zuo, L. Zhang, W. Zhang, and Z. Sheng. An improved bi-conjugate residual algorithm suitable for distributed parallel computing. *Appl. Math. and Comput.*, 186(2):1243–1253, 2007.

[92] M. Gustafsson, J. Demmel, and S. Holmgren. Numerical evaluation of the communication-avoiding Lanczos algorithm. Technical Report ISSN 1404-3203/2012-001, Department of Information Technology, Uppsala University, Feb. 2012.

[93] M. Gutknecht. Variants of BICGSTAB for matrices with complex spectrum. *SIAM J. Sci. Comput.*, 14(5):1020–1033, 1993.

[94] M. Gutknecht. Lanczos-type solvers for nonsymmetric linear systems of equations. *Acta Numer.*, 6(1997):271–398, 1997.

[95] M. Gutknecht. Spectral deflation in Krylov solvers: a theory of coordinate space based methods. *Electron. Trans. Numer. Anal.*, 39:156–185, 2012.

[96] M. Gutknecht and Z. Strakoš. Accuracy of two three-term and three two-term recurrences for Krylov space solvers. *SIAM J. Matrix Anal. Appl.*, 22:213–229, 2000.

[97] R. Hempel and A. Schuller. Experiments with parallel multigrid algorithms using the suprenum communications subroutine library. Technical Report 141, GMD, Germany, 1988.

[98] V. Hernandez, J. Roman, and A. Tomas. Parallel Arnoldi eigensolvers with enhanced scalability via global communications rearrangement. *Parallel Comput.*, 33(7-8):521–540, 2007.

[99] V. Hernandez, J. Roman, and V. Vidal. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Trans. Math. Software*, 31(3):351–362, 2005.

[100] M. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Nat. Bur. Standards*, 49:409–436, 1952.

[101] A. Hindmarsh and H. Walker. Note on a Householder implementation of the GMRES method. Technical Report UCID-20899, Lawrence Livermore National Lab., CA., 1986.

[102] M. Hoemmen. *Communication-avoiding Krylov subspace methods*. PhD thesis, EECS Dept., U.C. Berkeley, 2010.

[103] J. Hong and H. Kung. I/O complexity: the red-blue pebble game. In *Proc. 13th Symp. Theory Comp.*, pages 326–333. ACM, 1981.

[104] W. Joubert and G. Carey. Parallelizable restarted iterative methods for nonsymmetric linear systems. Part I: theory. *Int. J. Comput. Math.*, 44(1-4):243–267, 1992.

[105] K. Kahl and H. Rittich. Analysis of the deflated conjugate gradient method based on symmetric multigrid theory. *arXiv preprint arXiv:1209.1963*, 2012.

[106] W. Karush. An iterative method for finding characteristic vectors of a symmetric matrix. *Pacific J. Math*, 1(2):233–248, 1951.

[107] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1999.

[108] S. Kim and A. Chronopoulos. A class of Lanczos-like algorithms implemented on parallel computers. *Parallel Comput.*, 17(6):763–778, 1991.

[109] S. Kim and A. Chronopoulos. An efficient Arnoldi method implemented on parallel computers. In *International Conference on Parallel Processing, vol. 3*, pages 167–170, 1991.

[110] S. Kim and A. Chronopoulos. An efficient nonsymmetric Lanczos method on parallel vector computers. *J. Comput. Appl. Math.*, 42(3):357–374, 1992.

[111] N. Knight, E. Carson, and J. Demmel. Exploiting data sparsity in parallel matrix powers computations. Technical Report UCB/EECS-2013-47, EECS Dept., UC Berkeley, May 2013.

[112] N. Knight, E. Carson, and J. Demmel. Exploiting data sparsity in parallel matrix powers computations. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waniewski, editors, *Parall. Proc. Appl. Math.*, Lecture Notes in Computer Science, pages 15–25. Springer Berlin Heidelberg, 2014.

[113] Ronald Kriemann. *Parallele Algorithmen für $\mathcal{H}$-Matrizen*. PhD thesis, Christian-Albrechts-Universität zu Kiel, 2005. in German.

[114] A. LaMielle and M. Strout. Enabling code generation within the sparse polyhedral framework. Technical Report CS-10-102, Colorado State University, Mar 2010.

[115] C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. *J. Res. Natn. Bur. Stand.*, 45(4):255–282, 1950.

[116] C. Lanczos. Solution of systems of linear equations by minimized iterations. *J. Research Nat. Bur. Standards*, 49:33–53, 1952.

[117] C. Leiserson, S. Rao, and S. Toledo. Efficient out-of-core algorithms for linear relaxation using blocking covers. *J. Comput. Syst. Sci. Int.*, 54(2):332–344, 1997.

[118] R. Leland. *The Effectiveness of Parallel Iterative Algorithms for Solution of Large Sparse Linear Systems*. PhD thesis, University of Oxford, 1989.

[119] J. Liesen and Z. Strakoš. *Krylov subspace methods: principles and analysis*. Oxford University Press, 2012.

[120] K. Meerbergen and Z. Bai. The Lanczos method for parameterized symmetric linear systems with multiple right-hand sides. *SIAM J. Matrix Anal. Appl.*, 31(4):1642–1662, 2010.

[121] M. Mehri Dehnavi, J. Demmel, and D. Fernández. Sparse approximate inverse preconditioned communication-avoiding BiCGStab solver. Technical report, Massachusetts Institute of Technology, 2014.

[122] M. Mehri Dehnavi, Y. El-Kurdi, J. Demmel, and D. Giannacopoulos. Communication-avoiding Krylov techniques on graphic processing units. *Magnetics, IEEE Transactions on*, 49(5):1749–1752, 2013.

[123] G. Meurant. *Computer solution of large linear systems*, volume 59. Elsevier Amsterdam, 1999.

[124] G. Meurant. *The Lanczos and conjugate gradient algorithms: from theory to finite precision computations*. SIAM, 2006.

[125] G. Meurant and Z. Strakoš. The Lanczos and conjugate gradient algorithms in finite precision arithmetic. *Acta Numer.*, 15:471–542, 2006.

[126] M. Mohiyuddin. *Tuning Hardware and Software for Multiprocessors*. PhD thesis, EECS Dept., U.C. Berkeley, May 2012.

[127] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick. Minimizing communication in sparse matrix solvers. In *Proc. ACM/IEEE Conference on Supercomputing*, 2009.

[128] R. Morgan. Implicitly restarted GMRES and Arnoldi methods for nonsymmetric systems of equations. *SIAM J. Matrix Anal. Appl.*, 21(4):1112–1135, 2000.

[129] R. Morgan. GMRES with deflated restarting. *SIAM J. Sci. Comput.*, 24(1):20–37, 2002.

[130] J. Morlan. Auto-tuning the matrix powers kernel with SEJITS. Master's thesis, EECS Dept., U.C. Berkeley, May 2012.

[131] N. Nachtigal, S. Reddy, and L. Trefethen. How fast are nonsymmetric matrix iterations? *SIAM J. Matrix Anal. Appl.*, 13(3):778–795, 1992.

[132] V. Naik and S. Ta'asan. Performance studies of multigrid algorithms implemented on hypercube multiprocessor systems. In M. Health, editor, *Proceedings of the Second Conference on Hypercube Multiprocessors*, pages 720–729. SIAM, 1897.

[133] A. Neumaier. Iterative regularization for large-scale ill-conditioned linear systems. Oral presentation. Numerical Linear Algebra Meeting, Oberwolfach, April 1994.

[134] R. Nicolaides. Deflation of conjugate gradients with applications to boundary value problems. *SIAM J. Numer. Anal.*, 24(2):355–365, 1987.

[135] Y. Notay. On the convergence rate of the conjugate gradients in presence of rounding errors. *Numerische Mathematik*, 65(1):301–317, 1993.

[136] Y. Notay. Flexible conjugate gradients. *SIAM J. Sci. Comput.*, 22(4):1444–1460, 2000.

[137] Désiré Nuentsa Wakam and Jocelyne Erhel. Parallelism and robustness in GMRES with the Newton basis and the deflated restarting. Rapport de recherche RR-7787, INRIA, November 2011.

[138] C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices*. PhD thesis, London University, London, UK, 1971.

[139] C. Paige. Computational variants of the Lanczos method for the eigenproblem. *IMA J. Appl. Math.*, 10(3):373–381, 1972.

[140] C. Paige. Error analysis of the Lanczos algorithm for tridiagonalizing a symmetric matrix. *IMA J. Appl. Math.*, 18(3):341–349, 1976.

[141] C. Paige. Accuracy and effectiveness of the Lanczos algorithm for the symmetric eigenproblem. *Linear Algebra Appl.*, 34:235–258, 1980.

[142] C. Paige. An augmented stability result for the Lanczos hermitian matrix tridiagonalization process. *SIAM J. Matrix Anal. Appl.*, 31(5):2347–2359, 2010.

[143] C. Paige, I. Panayotov, and J-P. Zemke. An augmented analysis of the perturbed two-sided Lanczos tridiagonalization process. *Lin. Alg. Appl.*, 447:119–132, 2014.

[144] C. Paige and M. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, 8(1):43–71, 1982.

[145] M. Parks, E. De Sturler, G. Mackey, D. Johnson, and S. Maiti. Recycling Krylov subspaces for sequences of linear systems. *SIAM J. Sci. Comput.*, 28(5):1651–1674, 2006.

[146] B. Parlett. The new qd algorithms. *Acta numerica*, 4:459–491, 1995.

[147] B. Parlett and D. Scott. The Lanczos algorithm with selective orthogonalization. *Math. Comput.*, 33(145):217–238, 1979.

[148] B. Parlett, D. Taylor, and Z. Liu. A look-ahead Lanczos algorithm for unsymmetric matrices. *Math. Comp.*, 44(169):105–124, 1985.

[149] B. Philippe and L. Reichel. On the generation of Krylov subspace bases. *Appl. Numer. Math*, 62(9):1171–1186, 2012.

[150] M. Rappaz, M. Bellet, and M. Deville. *Numerical modeling in materials science and engineering*, volume 32. Springer Science and Business Media, 2010.

[151] L. Reichel. Newton interpolation at Leja points. *BIT*, 30(2):332–346, 1990.

[152] S. Rump. Verified bounds for singular values, in particular for the spectral norm of a matrix and its inverse. *BIT Numer. Math.*, 51(2):367–384, 2011.

[153] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[154] Y. Saad. *Numerical methods for large eigenvalue problems*, volume 158. SIAM, 2011.

[155] Y. Saad, M. Yeung, J. Erhel, and F. Guyomarc'h. A deflated version of the conjugate gradient algorithm. *SIAM J. Sci. Comput.*, 21(5):1909–1926, 2000.

[156] M. Scquizzato and F. Silvestri. Communication lower bounds for distributed-memory computations. In *31st International Symposium on Theoretical Aspects of Computer Science*, page 627, 2014.

[157] H. Simon. The Lanczos algorithm with partial reorthogonalization. *Math. Comput.*, 42(165):115–142, 1984.

[158] V. Simoncini and D. Szyld. Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM J. Sci. Comput.*, 25(2):454–477, 2003.

[159] V. Simoncini and D. Szyld. Recent computational developments in Krylov subspace methods for linear systems. *Numer. Linear Algebra Appl.*, 14(1):1–59, 2007.

[160] G. Sleijpen and D. Fokkema. BiCGstab (l) for linear equations involving unsymmetric matrices with complex spectrum. *Electron. Trans. Numer. Anal.*, 1(11):2000, 1993.

[161] G. Sleijpen and H. van der Vorst. Reliable updated residuals in hybrid Bi-CG methods. *Computing*, 56(2):141–163, 1996.

[162] G. Sleijpen, H. van der Vorst, and D. Fokkema. BiCGstab (l) and other hybrid Bi-CG methods. *Numer. Algorithms*, 7(1):75–109, 1994.

[163] G. Sleijpen, H. van der Vorst, and J. Modersitzki. Differences in the effects of rounding errors in Krylov solvers for symmetric indefinite linear systems. *SIAM J. Matrix Anal. Appl.*, 22(3):726–751, 2001.

[164] E. Solomonik, E. Carson, N. Knight, and J. Demmel. Tradeoffs between synchronization, communication, and computation in parallel linear algebra computations. *ACM Trans. Parallel Comput.* (invited submission), 2014.

[165] Edgar Solomonik, Erin Carson, Nicholas Knight, and James Demmel. Tradeoffs between synchronization, communication, and computation in parallel linear algebra computations. In *Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '14, pages 307–318, New York, NY, USA, 2014. ACM.

[166] P. Sonneveld. CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *Statist. Comput.*, 10:36–52, 1989.

[167] P. Sonneveld and M. van Gijzen. IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations. *SIAM Journal on Scientific Computing*, 31(2):1035–1062, 2008.

[168] A. Stathopoulos and K. Orginos. Computing and deflating eigenvalues while solving multiple right-hand side linear systems with an application to quantum chromodynamics. *SIAM J. Sci. Comput.*, 32(1):439–462, 2010.

[169] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[170] Z. Strakoš. On the real convergence rate of the conjugate gradient method. *Lin. Alg. Appl.*, 154:535–549, 1991.

[171] Z. Strakoš and P. Tichý. On error estimation in the conjugate gradient method and why it works in finite precision computations. *Electronic Transactions on Numerical Analysis*, 13:56–80, 2002.

[172] M. Strout, A. LaMielle, L. Carter, J. Ferrante, B. Kreaseck, and C. Olschanowsky. An approach for code generation in the sparse polyhedral framework. Technical Report CS-13-109, Colorado State University, December 2013.

[173] J. Tang, S. MacLachlan, R. Nabben, and C. Vuik. A comparison of two-level preconditioners based on multigrid and deflation. *SIAM J. Matrix Anal. Appl.*, 31(4):1715–1739, 2010.

[174] R. Thompson and P. McEnteggert. Principal submatrices ii: The upper and lower quadratic inequalities. *Lin. Alg. Appl.*, 1(2):211–243, 1968.

[175] S. Toledo. *Quantitative performance modeling of scientific computations and creating locality in numerical algorithms.* PhD thesis, MIT, 1995.

[176] C. Tong and Q. Ye. Analysis of the finite precision bi-conjugate gradient algorithm for nonsymmetric linear systems. *Math. Comp.*, 69(232):1559–1576, 2000.

[177] D. Unat. Personal communication, 2013.

[178] J. van den Eshof and G. Sleijpen. Inexact Krylov subspace methods for linear systems. *SIAM J. Matrix Anal. Appl.*, 26(1):125–153, 2004.

[179] H. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 13(2):631–644, 1992.

[180] H. van der Vorst. *Iterative Krylov methods for large linear systems*, volume 13. Cambridge University Press, 2003.

[181] H. Van der Vorst and Q. Ye. Residual replacement strategies for Krylov subspace iterative methods for the convergence of true residuals. *SIAM J. Sci. Comput.*, 22(3):835–852, 1999.

[182] J. Van Rosendale. Minimizing inner product data dependencies in conjugate gradient iteration. In *Proc. IEEE Internat. Confer. Parallel Processing*, pages 44–46. IEEE, 1983.

[183] C. Vuik, A. Segal, and J. Meijerink. An efficient preconditioned CG method for the solution of a class of layered problems with extreme contrasts in the coefficients. *J. Comput. Phys.*, 152(1):385–403, 1999.

[184] H. Walker. Implementation of the GMRES method using Householder transformations. *SIAM J. Sci. Stat. Comput.*, 9:152–163, 1988.

[185] S. Wang, X.S. Li, J. Xia, Y. Situ, and M.V. de Hoop. Efficient scalable algorithms for hierarchically semiseparable matrices. *SIAM J. Sci. Comput.*, 2012. (under review).

[186] S. Wang, E. Sturler, and G. Paulino. Large-scale topology optimization using preconditioned Krylov subspace methods with recycling. *Int. J. Numer. Methods in Engrg.*, 69(12):2441–2468, 2007.

[187] P. Wesseling and P. Sonneveld. Numerical experiments with a multiple grid and a preconditioned Lanczos type method. In *Approximation methods for Navier-Stokes problems*, pages 543–562. Springer, 1980.

[188] J. Wilkinson. *The algebraic eigenvalue problem*, volume 87. Oxford Univ. Press, 1965.

[189] S. Williams, D. Kalamkar, A. Singh, A. Deshpande, B. Van Straalen, M. Smelyanskiy, A. Almgren, P. Dubey, J. Shalf, and L. Oliker. Optimization of geometric multigrid for emerging multi-and manycore processors. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 96. IEEE Computer Society Press, 2012.

[190] S. Williams, M. Lijewski, A. Almgren, B. Van Straalen, E. Carson, N. Knight, and J. Demmel. s-step Krylov subspace methods as bottom solvers for geometric multigrid. In *International Symposium on Parallel and Distributed Processing*. IEEE, 2014.

[191] D. Womble and B. Young. A model and implementation of multigrid for massively parallel computers. *Int. J. High Speed Computing*, 2(3):239–256, 1990.

[192] W. Wülling. On stabilization and convergence of clustered Ritz values in the Lanczos method. *SIAM J. Matrix Anal. Appl.*, 27(3):891–908, 2005.

[193] J. Xia, S. Chandrasekaran, M. Gu, and X. Li. Fast algorithms for hierarchically semiseparable matrices. *Numer. Linear Algebra Appl.*, 17(6):953–976, 2010.

[194] I. Yamazaki, H. Anzt, S. Tomov, M. Hoemmen, and J. Dongarra. Improving the performance of CA-GMRES on multicores with multiple GPUs. In *Proc. IEEE 28th Internat. Parallel and Distributed Processing Symposium*, pages 382–391. IEEE, 2014.

[195] I. Yamazaki, S. Rajamanickam, E. Boman, M. Hoemmen, M. Heroux, and S. Tomov. Domain decomposition preconditioners for communication-avoiding Krylov methods on a hybrid CPU/GPU cluster. In *Proc. ACM/IEEE Conference on Supercomputing*, pages 933–944. IEEE, 2014.

[196] I. Yamazaki, S. Tomov, T. Dong, and J. Dongarra. Mixed-precision orthogonalization scheme and adaptive step size for improving the stability and performance of CA-GMRES on GPUs. *High Performance Computing for Computational Science–VECPAR 2014*, pages 17–30, 2015.

[197] I. Yamazaki and K. Wu. A communication-avoiding thick-restart Lanczos method on a distributed-memory system. In *Euro-Par 2011: Parallel Processing Workshops*, pages 345–354. Springer, 2012.

[198] C-Q. Yang and B. Miller. Critical path analysis for the execution of parallel and distributed programs. In *Proc. 8th Int. Conf. Dist. Comput. Sys.*, pages 366–373. IEEE, 1988.

[199] L. Yang and R. Brent. The improved BiCGStab method for large and sparse unsymmetric linear systems on parallel distributed memory architectures. In *Fifth International Conference on Algorithms and Architectures for Parallel Processing*, pages 324–328. IEEE, 2002.

[200] T. Yang. Solving sparse least squares problems on massively distributed memory computers. In *Advances in Parallel and Distributed Computing*, pages 170–177. IEEE, 1997.

[201] M-C. Yeung and T. Chan. ML(k) BiCGSTAB: A BiCGSTAB variant based on multiple Lanczos starting vectors. *SIAM J. Sci. Comput.*, 21(4):1263–1290, 1999.

[202] J. Zemke. *Krylov subspace methods in finite precision: a unified approach*. PhD thesis, Technische Universität Hamburg-Harburg, 2003.

[203] A. Zomaya. *Parallel computing for bioinformatics and computational biology: models, enabling technologies, and case studies*, volume 55. John Wiley & Sons, 2006.