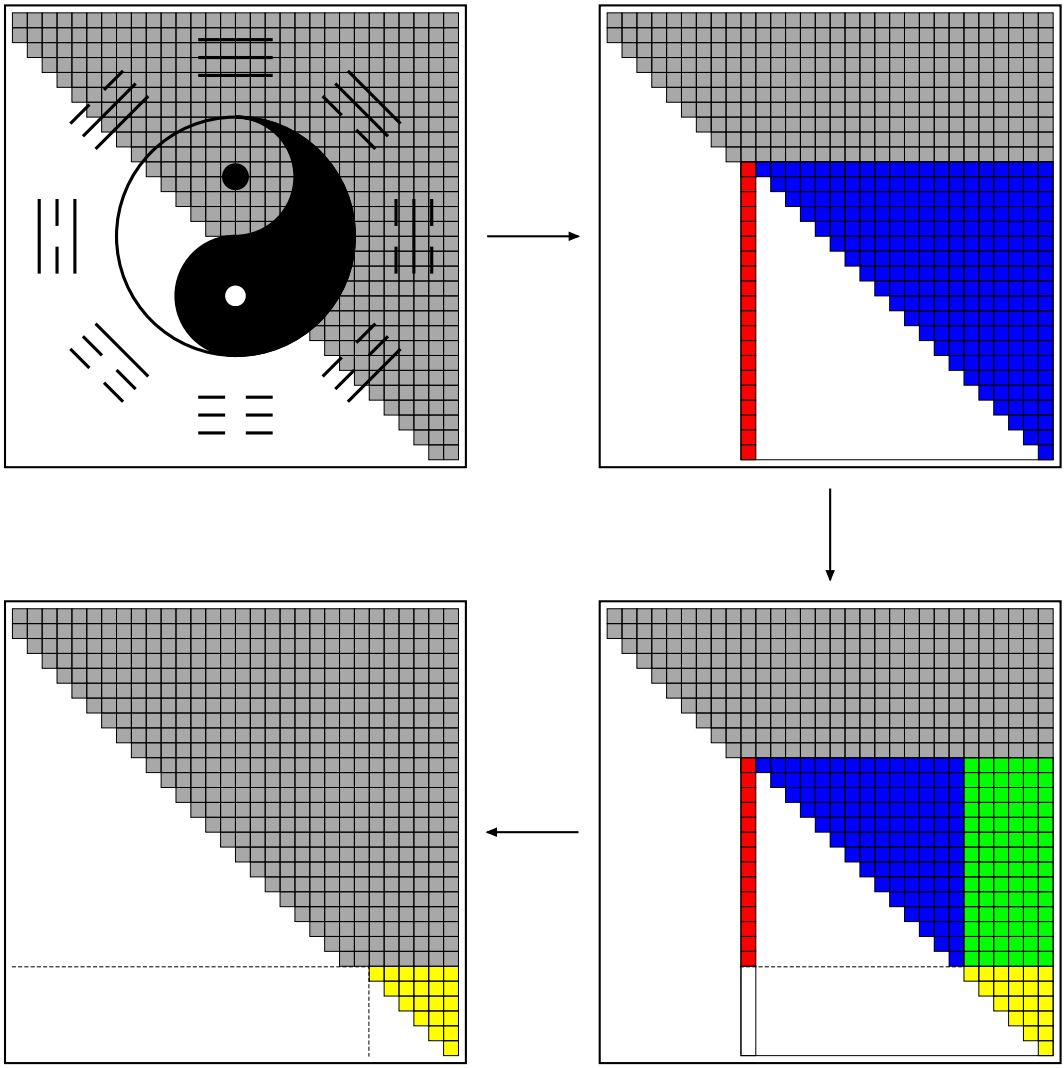

Parallel Variants and Library Software for the QR Algorithm and the Computation of the Matrix Exponential of Essentially Nonnegative Matrices

Meiyue Shao

Ph Licentiate Thesis, April 2012
Department of Computing Science
Umeå University, Sweden

Parallel Variants and Library Software for the QR Algorithm and the Computation of the Matrix Exponential of Essentially Nonnegative Matrices

Meiyue Shao

Licentiate Thesis, April 2012

DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

myshao@cs.umu.se

ISBN 978-91-7459-430-0
ISSN 0348-0542
UMINF 12.07

Printed by Print & Media, Umeå University, 2012

Abstract

This Licentiate Thesis contains contributions in two challenging topics in matrix computations: the parallel multishift QR algorithm with aggressive early deflation (AED) and the matrix exponential of essentially nonnegative matrices. They are both structure-preserving algorithms in numerical linear algebra. We focus on performance in the former problem and on accuracy in the latter one.

The solution of matrix eigenvalue problems is a fundamental topic in numerical linear algebra. The QR algorithm which computes the Schur decomposition of a matrix is by far the most important approach for solving dense nonsymmetric eigenvalue problems. Recently a novel parallel QR algorithm has been developed by incorporating some modern techniques such as small-bulge multishift and AED. The novel parallel approach significantly outperforms the pipelined QR algorithm in ScaLAPACK v1.8.0 and earlier versions. But AED becomes a computational bottleneck in the new parallel QR algorithm. We develop multilevel AED algorithms which indeed decrease the total amount of communications and further improve the performance of the parallel QR algorithm. We also identify and remedy some anomalies in the original ScaLAPACK implementation. The improved version of the new parallel QR algorithm is now available as a part of ScaLAPACK version 2.0. Both performance models and numerical experiments demonstrate the efficiency of the new approach.

The computation of the matrix exponential is also a classical topic in numerical linear algebra. The exponential of essentially nonnegative matrices (real square matrices with nonnegative off-diagonal entries) is an important special case since it has applications in continuous-time Markov processes and positive linear dynamical systems. Because of the nonnegativity, this problem is usually well-conditioned in the sense of componentwise perturbations. We derive lower and upper bounds of the matrix exponential, and combining the scaling and squaring method with aggressively truncated Taylor expansion. More precisely, we establish new à priori error estimates and use them to propose an efficient strategy to balance the scale factor and the order of expansion. This leads to new algorithms with componentwise high relative accuracy. Rounding error analyses and numerical experiments confirm the efficiency and accuracy of our algorithms.

Preface

This Licentiate Thesis consists of the following three papers:

Paper I B. Kågström, D. Kressner, and M. Shao. On Aggressive Early Deflation in Parallel Variants of the QR Algorithm[1]. *Applied Parallel and Scientific Computing (PARA 2010)*, Lecture Notes in Computer Science, Springer, LNCS 7133, pages 1–10, 2012.

Paper II R. Granat, B. Kågström, D. Kressner, and M. Shao. Parallel Library Software for the Multishift QR Algorithm with Aggressive Early Deflation. Technical Report, UMINF-12.06, April 2012.

Paper III M. Shao, W. Gao, and J. Xue. Componentwise High Relative Accuracy Algorithms for the Exponential of an Essentially Nonnegative Matrix. Technical Report, UMINF-12.04, March 2012. (submitted to *Numerische Mathematik*)

In addition to the papers included in the thesis, there are also other publications finished within the PhD studies:

X. S. Li and M. Shao. A Supernodal Approach to Incomplete LU Factorization with Partial Pivoting. *ACM Transactions on Mathematical Software*, 37(4): Article 43, 2011.

M. Shao. `PDLAQR1`: An Improved Version of the ScaLAPACK Routine `PDLAHQR`. Technical Report, UMINF-11.22, revised in April 2012.

Q. Xie and M. Shao. To Review Some Techniques in Advanced Algebra via an Elementary Problem. *Studies in College Mathematics*, to appear (in Chinese).

[1] Reprinted by permission of Springer.

Acknowledgements

First of all, I wish to thank my supervisor Professor Bo Kågström, who is also co-author of two papers in this thesis. It is you who helped me arrange everything for settling down when I arrived at Umeå. It has been great to work with you. Despite your busy schedule, you always spent a lot of time discussing my research work and providing constructive feedback. I was also largely encouraged by you for every little progress I had made. Thank you!

Secondly, I want to thank my co-supervisor Professor Daniel Kressner, who is also co-author of two papers in this contribution. Although most our discussions so far were not face-to-face, I have greatly benefited from your fruitful suggestions and comments. I look forward to the second part of my PhD studies at Lausanne. Thank you!

Thirdly, I am grateful to all members in the Numerical Linear Algebra and Parallel High Performance Computing Groups, especially to Dr. Lars Karlsson, for all kinds of assistance. Thank you very much!

Many thanks to all colleagues at the Department of Computing Science and the High Performance Computing Center North (HPC2N) for providing a good work environment and various support.

I also wish to thank all professors in the Computational Mathematics Group at Fudan University, who introduced me to numerical linear algebra and high performance computing.

Finally, I owe a great gratitude to my family. Many thanks to my parents and my girlfriend. It is impossible for me to concentrate on the research without your enduring support.

This work was conducted using the resources of the High Performance Computing Center North, `http://www.hpc2n.umu.se`. Financial support has been provided by the Swedish Research Council under grants VR7062571 and A0581501, the Swedish Foundation for Strategic Research under grant A3 02:127, UMIT Research Lab via an EU Mål 2 project, and eSSENCE, a strategic collaborative e-Science programme funded by the Swedish Research Council.

Umeå, April 2012

Meiyue Shao

Contents

Chapter 1

Introduction

This chapter includes motivations of the work presented in this thesis. Some background knowledge of the two topics is also provided.

1.1 The Parallel QR Algorithm

The solution of matrix eigenvalue problems is a fundamental topic in numerical linear algebra [5, 11] with applications in various areas of science and engineering. In this thesis we consider the *standard eigenvalue problem* (SEP)

$$Ax = \lambda x \quad (x \neq 0), \tag{1.1}$$

where A is a square $N \times N$ matrix and x is a nonzero $N \times 1$ vector. The scalar λ is called an *eigenvalue* of A and x is the corresponding *eigenvector*. We see that x is an eigenvector of A if it is mapped by A onto a multiple of itself. There are several existing methods for solving the standard eigenvalue problem based on different properties of A (e.g., symmetry or sparsity) and different demands to the knowledge of the spectrum (e.g., see [23]). We are interested in computing all eigenvalues of a dense non-Hermitian matrix. The QR algorithm [8, 9, 10, 18, 19] is by far the most popular approach for this purpose which calculates the *Schur decomposition* of A:

$$A = QTQ^*, \tag{1.2}$$

where Q is unitary and T is upper triangular. Then the eigenvalues of A are available as the diagonal entries of T. If A is real, Q and T can also be chosen as real matrices but T becomes quasi-triangular (i.e. a block triangular matrix with 1×1 and 2×2 diagonal blocks).

The QR algorithm consists two main stages:

$$A \xrightarrow{U_1} H \xrightarrow{U_2} T. \tag{1.3}$$

The first stage (known as the *Hessenberg reduction*) transforms A to an upper Hessenberg form H (i.e. $H(i,j) = 0$ for $i > j+1$). The Hessenberg reduction only requires finite number of operations and can be performed efficiently (e.g., see [3, 7, 17, 22]). The second stage (known as the *Hessenberg QR algorithm* or the *QR iteration*) further reduces H to the Schur form. This stage is iterative in nature and is usually much more difficult than the Hessenberg reduction because both the convergence rate and the algorithmic issues need to be taken care of.

Recently, some novel techniques such as *small-bulge multishift* and *aggressive early deflation* (AED) [1, 2] have been proposed. The multishift strategy reduces the number of I/O operations in a delay-and-accumulate manner; AED explores potentially deflatable eigenvalues and hence enhance the convergence rate quite a lot. Therefore these techniques dramatically improve the performance of the Hessenberg QR algorithm. A new parallel QR algorithm equipped with these modern techniques on distributed memory systems has also been developed [12] which outperforms the original ScaLAPACK's implementation of the QR algorithm [13] significantly. However, although AED can largely improve the convergence rate in the QR algorithm, it is observed that the parallel AED process becomes a computational bottleneck for a modest number of processors. In Papers I and II, we develop multilevel algorithms for AED which decrease the total execution time and at the same time perform less communications. These contributions further improve the performance of the new parallel QR algorithm. The new implementation of the parallel QR algorithm is now published as a part of ScaLAPACK version 2.0.

1.2 The Matrix Exponential of Essentially Nonnegative Matrices

The computation of matrix functions is another fundamental topic in numerical linear algebra [11, 14]. Let D be a domain in the complex plane enclosed by a Jordan curve. If D contains the spectrum of A and $f(z)$ is analytic on D, then $f(A)$ is defined as

$$f(A) = \frac{1}{2\pi \mathrm{i}} \int_{\partial D} f(z)(zI - A)^{-1} \mathrm{d}z. \tag{1.4}$$

There are many other alternative definitions of $f(A)$. For example, if $f(z)$ has the Laurent series

$$f(z) = \sum_{k=-\infty}^{\infty} a_k (z - z_0)^k \tag{1.5}$$

in D, then

$$f(A) = \sum_{k=-\infty}^{\infty} a_k (A - z_0 I)^k. \tag{1.6}$$

It is straightforward to prove from the definition of $f(A)$ that

$$f(P^{-1}AP) = P^{-1}f(A)P, \tag{1.7}$$

where P is any nonsingular matrix. This property naturally leads to the eigenvalue methods for calculating $f(A)$. For example, the high level abstraction of the Schur-Parlett algorithm [4] is:

(1) Compute the Schur decomposition $A = QTQ^*$.
(2) Compute $f(T)$ via Parlett's algorithm [21].
(3) Formulate $f(A) = Qf(T)Q^*$.

Therefore, functions of matrices closely relate to the dense eigenvalue problem (e.g., see [16]). Another class of algorithms computes $g(A)$ where $g(z)$ approximates $f(z)$ and $g(A)$ is much easier to calculate. Such kind of approximations include, e.g., truncated Taylor series and Padé approximants. Certainly, there can be other methods using some special properties of $f(z)$ or A. The work presented in this thesis is such an example.

The matrix exponential $f(A) = \mathrm{e}^A$, where $f(z) = \mathrm{e}^z$, is one of the most well-studied matrix functions, since it has many applications in physics, biology, finance and engineering. Several methods for computing the matrix exponential have been proposed, see [14, 20]. Among the existing algorithms, the scaling and squaring method,

$$\mathrm{e}^A = \left(\mathrm{e}^{A/2^k}\right)^{2^k}, \tag{1.8}$$

is considered as one of the most promising candidates for calculating the matrix exponential [20]. Usually $\mathrm{e}^{A/2^k}$ is approximated by a diagonal Padé approximation [15, 24] or a truncated Taylor series [6, 26]. It is easier to approximate $\mathrm{e}^{A/2^k}$ compared to e^A because the scaling process reduces the spectral radius and leads to a faster decay of the truncation error.

In some applications such as continuous-time Markov processes and positive linear dynamical systems, the matrix A is *essentially nonnegative* (i.e. $A(i,j) \geq 0$ for all $i \neq j$). This special structure of A leads to some nice properties in e^A. For example, the matrix exponential is always nonnegative since

$$\mathrm{e}^A = \mathrm{e}^s \sum_{k=0}^{\infty} (A - sI)^k \geq 0, \tag{1.9}$$

where $s = \min\{A(i,i)\}$. Recently a componentwise perturbation analysis for e^A where A is essentially nonnegative has been developed, see [25]. The perturbation analysis suggests that it is possible to compute all entries of e^A to high relative accuracy. We investigate such kind of algorithms in Paper III. These algorithms are variants of the scaling and squaring method. By taking advantage of the nonnegativity, we derive lower and upper bounds of e^A and establish error estimates for these bounds. The algorithms are shown to be componentwise forward stable by doing rounding error analysis and presenting results of numerical experiments.

Chapter 2

Summary of Papers

In this chapter, a brief summary of the papers is given. The topic of Papers I and II is the parallel QR algorithm for dense nonsymmetric eigenvalue problems. In Paper III, algorithms for the matrix exponential are presented.

2.1 Paper I

In this paper, we discuss a two-level approach for performing AED in a parallel environment. The lower level consists of a novel combination of AED with the pipelined QR algorithm implemented in the ScaLAPACK routine `PDLAHQR`. The new approach aims to minimize the execution time for calculating the Schur decomposition of the AED window and hence improves the performance of the higher level QR algorithm. By the data redistribution strategy, frequent communications are avoided in the deflation stage. A new shift strategy is also proposed to improve the convergence rate of the pipelined QR algorithm. These techniques significantly improve the performance of the higher level AED process as well as the entire parallel QR algorithm.

2.2 Paper II

In Paper II, we present the library software of the parallel multishift QR algorithm with AED. The algorithm is largely based on the work in [12] and Paper I. The communication avoiding algorithm via data redistribution proposed in Paper I is further developed so that the higher level AED can be performed efficiently as well. We also refine the strategy for switching between multishift QR sweeps and AED. Suggestions regarding some important tunable algorithmic parameters are provided. With these improvements, AED is no longer a computational bottleneck in the new approach. We establish a performance model which explains the scalability behavior of the new parallel QR algorithm. Finally, a lot of computational experiments demonstrate the

significant improvement we have made compared to the preliminary version of the new parallel QR algorithm in [12] as well as the original ScaLAPACK's pipelined QR approach. The new software is available as a part of ScaLAPACK version 2.0.

2.3 Paper III

In Paper III, we present componentwise forward stable algorithms for computing e^A where A is essentially nonnegative. The first approach is the scaling and squaring method with Taylor expansion

$$\left[\sum_{k=0}^{m} \frac{(A/n)^k}{k!}\right]^n \leq \mathrm{e}^A \quad (\text{for } A \geq 0). \tag{2.1}$$

Contrary to most existing approaches, we use a large scale factor n and truncate aggressively in the Taylor series. We derive a componentwise à priori error estimate and use it to propose an efficient strategy to balance the scale factor n and the order of expansion m. The scale factor is proportional to the condition number so that the algorithm is almost optimally scaled. This leads to a lower bound approach of e^A where A is essentially nonnegative. Similarly, we derive an upper bound approach using the scaling and squaring method built on a non-diagonal Padé approximation

$$\left[\sum_{k=0}^{m-2} \frac{(A/n)^k}{k!} + \frac{(A/n)^{m-1}}{(m-1)!}\left(I - \frac{A}{mn}\right)^{-1}\right]^n \geq \mathrm{e}^A \quad (\text{for } A \geq 0 \text{ and } mn > \rho(A)). \tag{2.2}$$

The corresponding componentwise error estimates are also established. Finally, we propose an interval algorithm without using interval arithmetic as well as an interpolation strategy. This novel interval approach always produce actual lower and upper bounds of e^A regardless of roundoff. Both rounding error analyses and numerical experiments confirm the forward stability of our proposed algorithms.

Chapter 3

Ongoing and Future Work

This chapter includes some possible extensions of the work presented in this theses.

3.1 Dense and Structured Eigenvalue Problems

Since most eigensolvers are essentially iterative, the deflation strategy has important impact on the convergence rate. We have already benefited from efficient approaches of AED in the parallel QR algorithm. In the future more deflation techniques, especially structure preserving strategies for some structured eigenvalue problems, would be investigated. Future work also includes investigating some applications using the software of parallel QR algorithm, for example, parallel solvers for general analytic functions of matrices. Based on the newly developed software of the parallel QR algorithm, the computation of general matrix functions for large-scale dense matrices becomes feasible.

3.2 Matrix Functions

In most existing scaling and squaring algorithms for the matrix exponential, diagonal Padé approximation instead of Taylor expansion is preferred, since the Padé approximant approximates e^A in a cheaper and more accurate manner compared to the truncated Taylor series. Using the techniques in Paper III, we are already able to establish componentwise error estimates for the scaling and squaring method using Padé approximation when A is essentially nonnegative. But how to accurately compute the Padé approximant is still a challenging problem since the nonnegativity can be hardly preserved during the calculation. Moreover, with the knowledge of the matrix exponential, it is also promising to develop theories and algorithms for its inverse problem — matrix logarithm when $\ln A$ is known in advance to be essentially nonnegative. These problems will be investigated in the future.

Bibliography

[1] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. Part I: Maintaining well-focused shifts and level 3 performance. *SIAM J. Matrix Anal. Appl.*, 23(4):929–947, 2002.

[2] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. Part II: Aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 23(4):948–973, 2002.

[3] J. Choi, J. J. Dongarra, and D. W. Walker. The design of a parallel dense linear algebra software library: Reduction to Hessenberg, tridiagonal, and bidiagonal form. *Numer. Algorithms*, 10(2):379–399, 1995.

[4] P. I. Davies and N. J. Higham. A Schur-Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

[5] J. W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[6] L. Deng and J. Xue. Accurate computation of exponentials of triangular essentially non-negative matrices. *J. Fudan University (Natural Science)*, 50(1):78–86, 2011.

[7] J. J. Dongarra, D. C. Sorensen, and S. J. Hammarling. Block reduction of matrices to condensed forms for eigenvalue computations. *J. Comput. Appl. Math.*, 27:215–227, 1989. Also as LAPACK Working Note 2.

[8] J. G. F. Francis. The QR transformation: A unitary analogue to the LR transformation — Part 1. *Comput. J.*, 4(3):265–271, 1961.

[9] J. G. F. Francis. The QR transformation — Part 2. *Comput. J.*, 4(4):332–345, 1962.

[10] G. H. Golub and F. Uhlig. The QR algorithm: 50 years later its genesis by John Francis and Vera Kublanovskaya and subsequent developments. *IMA J. Numer. Anal.*, 29(3):467–485, 2009.

[11] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, third edition, 1996.

[12] R. Granat, B. Kågström, and D. Kressner. A novel parallel QR algorithm for hybrid distributed memory HPC systems. *SIAM J. Sci. Comput.*, 32(4):2345–2378, 2010. Also as LAPACK Working Note 216.

[13] G. Henry, D. S. Watkins, and J. J. Dongarra. A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. *SIAM J. Sci. Comput.*, 24(1):284–311, 2002. Also as LAPACK Working Note 121.

[14] N. J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

[15] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Rev.*, 51(4):747–764, 2009.

[16] B. Kågström. *Methods for the Numerical Computation of Matrix Functions and the Treatment of Ill-Conditioned Eigenvalue Problems*. PhD thesis, Umeå University, 1977. Report UMINF-58.77.

[17] L. Karlsson and B. Kågström. Parallel two-stage reduction to Hessenberg form using dynamic scheduling on shared-memory architectures. *Parallel Comput.*, 37(12):771–782, 2011.

[18] D. Kressner. *Numerical Methods for General and Structured Eigenvalue Problems*, volume 46 of *Lect. Notes Comput. Sci. Eng.* Springer-Verlag, Heidelberg, 2005.

[19] V. N. Kublanovskaya. On some algorithms for the solution of the complete eigenvalue problem. *USSR Comp. Math. Math. Phys.*, 1(3):637–657, 1961.

[20] C. B. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Rev.*, 45(1):3–49, 2003.

[21] B. N. Parlett. A recurrence among the elements of functions of triangular matrices. *Linear Algebra Appl.*, 14(2):117–121, 1976.

[22] G. Quintana-Ortí and R. van de Geijn. Improving the performance of reduction to Hessenberg form. *ACM Trans. Math. Software*, 32(2):180–194, 2006.

[23] G. W. Stewart. *Matrix Algorithms, Volume II: Eigensystems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[24] R. C. Ward. Numerical computation of the matrix exponential with accuracy estimate. *SIAM J. Numer. Anal.*, 14(4):600–610, 1977.

[25] J. Xue and Q. Ye. Entrywise relative perturbation bounds for exponentials of essentially non-negative matrices. *Numer. Math.*, 110(3):393–403, 2008.

[26] J. Xue and Q. Ye. Computing exponentials of essentially non-negative matrices entrywise to high relative accuracy. *Math. Comp.*, to appear.

I

Paper I

On Aggressive Early Deflation in Parallel Variants of the QR Algorithm*

Bo Kågström[1], Daniel Kressner[2], and Meiyue Shao[1]

[1] *Department of Computing Science and HPC2N*
Umeå University, SE-901 87 Umeå, Sweden
{*bokg, myshao*}*@cs.umu.se*

[2] *Seminar for Applied Mathematics, ETH Zürich, Switzerland*
kressner@math.ethz.ch

Abstract: The QR algorithm computes the Schur form of a matrix and is by far the most popular approach for solving dense nonsymmetric eigenvalue problems. Multi-shift and aggressive early deflation (AED) techniques have led to significantly more efficient sequential implementations of the QR algorithm during the last decade. More recently, these techniques have been incorporated in a novel parallel QR algorithm on hybrid distributed memory HPC systems. While leading to significant performance improvements, it has turned out that AED may become a computational bottleneck as the number of processors increases. In this paper, we discuss a two-level approach for performing AED in a parallel environment, where the lower level consists of a novel combination of AED with the pipelined QR algorithm implemented in the ScaLAPACK routine `PDLAHQR`. Numerical experiments demonstrate that this new implementation further improves the performance of the parallel QR algorithm.

* Reprinted by permission of Springer.

On Aggressive Early Deflation in Parallel Variants of the QR Algorithm

Bo Kågström[1], Daniel Kressner[2], and Meiyue Shao[1]

[1] Department of Computing Science and HPC2N
Umeå University, S-901 87 Umeå, Sweden
{bokg,myshao}@cs.umu.se
[2] Seminar for Applied Mathematics, ETH Zürich, Switzerland
kressner@math.ethz.ch

Abstract. The QR algorithm computes the Schur form of a matrix and is by far the most popular approach for solving dense nonsymmetric eigenvalue problems. Multishift and aggressive early deflation (AED) techniques have led to significantly more efficient sequential implementations of the QR algorithm during the last decade. More recently, these techniques have been incorporated in a novel parallel QR algorithm on hybrid distributed memory HPC systems. While leading to significant performance improvements, it has turned out that AED may become a computational bottleneck as the number of processors increases. In this paper, we discuss a two-level approach for performing AED in a parallel environment, where the lower level consists of a novel combination of AED with the pipelined QR algorithm implemented in the ScaLAPACK routine `PDLAHQR`. Numerical experiments demonstrate that this new implementation further improves the performance of the parallel QR algorithm.

1 Introduction

The solution of matrix eigenvalue problems is a classical topic in numerical linear algebra, with applications in various areas of science and engineering. The QR algorithm developed by Francis and Kublanovskaya, see [9,19] for recent historic accounts, has become the de facto standard for solving nonsymmetric and dense eigenvalue problems. Parallelizing the QR algorithm has turned out to be highly nontrivial matter [13]. To our knowledge, the ScaLAPACK [5] routine `PDLAHQR` implemented nearly 10 years ago based on work by Henry, Watkins, and Dongarra [14], is the only publicly available parallel implementation of the QR algorithm. Recently, a novel parallel QR algorithm [10] has been developed, which turns out to be more than a magnitude faster compared to `PDLAHQR` for sufficiently large problems. These improvements are attained by parallelizing the multishift and aggressive early deflation (AED) techniques developed by Braman, Byers, and Mathias [6,7] for the sequential QR algorithm.

Performed after each QR iteration, AED requires the computation of the Schur form for a trailing principle submatrix (the so called AED window) that is

K. Jónasson (Ed.): PARA 2010, Part I, LNCS 7133, pp. 1–10, 2012.

relatively small compared to the size of the whole matrix. In [10], a slightly modified version of the ScaLAPACK routine `PDLAHQR` is used for this purpose. Due to the small size of the AED window, the execution time spent on AED remains negligible for one or only a few processors but quickly becomes a dominating factor as the number of processors increases. In fact, for a $100\,000 \times 100\,000$ matrix and 1024 processor cores, it was observed in [10] that 80% of the execution time of the QR algorithm was spent on AED. This provides a strong motivation to reconsider the way AED is performed in parallel. In this work, we propose to perform AED by a modification of the ScaLAPACK routine `PDLAHQR`, which also incorporates AED at this lower level, resulting in a two-level recursive approach for performing AED. The numerical experiments in Section 4 reveal that our new approach reduces the overall execution time of the parallel QR algorithm from [10] by up to 40%.

2 Overview of the QR Algorithm with AED

In the following, we assume some familiarity with modern variants of the QR algorithm and refer to [15,18] for introductions. It is assumed that the matrix under consideration has already been reduced to (upper) Hessenberg form by, e.g., calling the ScaLAPACK routine `PDGEHRD`. Algorithm 1 provides a high-level description of the sequential and parallel QR algorithm for Hessenberg matrices, using multiple shifts and AED. Since this paper is mainly concerned with AED, we will only mention that the way the shifts are incorporated in the multishift QR sweep (Step 4) plays a crucial role in attaining good performance, see [6,10,17] for details.

Algorithm 1. *Multishift Hessenberg QR Algorithm with AED*

WHILE not converged
1. Perform AED on the $n_{\mathsf{win}} \times n_{\mathsf{win}}$ trailing principle submatrix.
2. Apply the accumulated orthogonal transformation to the corresponding off-diagonal blocks.
3. **IF** enough eigenvalues have been deflated in Step 1
   GOTO Step 1.
   END IF
4. Perform a multishift QR sweep with undeflatable eigenvalues from Step 1 as shifts.
5. Check for negligible subdiagonal elements.

END WHILE

In the following, we summarize the AED technique proposed by Braman, Byers, and Mathias [7]. Given an $n \times n$ upper Hessenberg matrix H, we partition

$$H = \begin{array}{c} \\ n-n_{\mathsf{win}}-1 \\ 1 \\ n_{\mathsf{win}} \end{array} \begin{array}{c} \begin{array}{ccc} n-n_{\mathsf{win}}-1 & 1 & n_{\mathsf{win}} \end{array} \\ \begin{pmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ 0 & H_{32} & H_{33} \end{pmatrix} \end{array},$$

where n_{win} denotes the size of the AED window. Then a (real) Schur decomposition $H_{33} = VTV^T$ is performed, where V is orthogonal and T in upper quasi-triangular form. Setting

$$U = \begin{array}{c} \\ n-n_{\mathsf{win}}-1 \\ 1 \\ n_{\mathsf{win}} \end{array} \begin{array}{c} \begin{array}{ccc} n-n_{\mathsf{win}}-1 & 1 & n_{\mathsf{win}} \end{array} \\ \begin{pmatrix} I & & \\ & 1 & \\ & & V \end{pmatrix} \end{array},$$

we obtain

$$U^T H U = \begin{pmatrix} H_{11} & H_{12} & H_{13}V \\ H_{21} & H_{22} & H_{23}V \\ 0 & s & T \end{pmatrix},$$

where $s \in \mathbb{R}^{n_{\mathsf{win}}}$ is the so called spike, created from the subdiagonal entry contained in H_{32}. The eigenvalues of T are checked subsequently for convergence and possibly deflated. The eigenvalue (or 2×2 block) in the bottom right corner of T can be deflated if the magnitude of the last component (or the last two components) of the spike is negligibly small. Undeflatable eigenvalues are moved to the top left corner of T by a swapping algorithm [4,11]. After this transformation is completed, the next eigenvalue in the bottom right corner of T is treated in the same way. The orthogonal transformations for swapping eigenvalues are accumulated in an orthogonal matrix $\tilde{V} \in \mathbb{R}^{n_{\mathsf{win}} \times n_{\mathsf{win}}}$. After all eigenvalues of T have been processed, the entire matrix is reduced back to Hessenberg form and the off-diagonal blocks H_{13} and H_{23} are multiplied with the product of all involved orthogonal transformations. It is recommended to choose n_{win} somewhat larger, e.g., by 50%, than the number of shifts in the multishift QR iterations [6].

Dramatic performance gains from AED have been observed both for sequential and parallel variants of the QR algorithm. These gains can be achieved essentially no matter how the rest of the QR algorithm is implemented, in particular how many shifts are used in the multishift QR sweep [7]. In effect, any implementation of the QR algorithm may benefit from AED; a fact that we will use below to improve the ScaLAPACK routine `PDLAHQR`. A convergence analysis, partially explaining the success of AED, can be found in [16].

3 Parallel Implementation of AED

Since the main aim of this paper is to improve the parallel QR algorithm and implementation described in [10], we first recall the structure of the main routines from this implementation, see Figure 1. The entry routine is `PDHSEQR`, which branches into `PDLAQR1` for small to medium-sized matrices and `PDLAQR0` for larger ones. The cut-off point for what is considered medium-sized will be explained in the numerical experiments, see Section 4. The main purpose of `PDLAQR0` is to call `PDLAQR3` for performing AED and `PDLAQR5` for performing multishift QR iterations. The former routine invokes `PDLAQR1` for performing the Schur decomposition of the AED window. In [10], `PDLAQR1` amounts to the

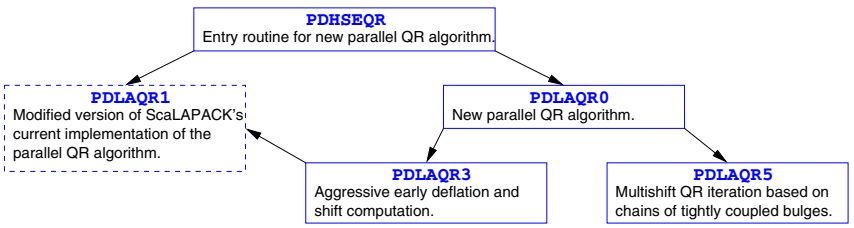

Fig. 1. Partial software structure for the parallel QR algorithm from [10]

ScaLAPACK routine PDLAHQR with minor modifications concerning the processing of 2×2 blocks in the real Schur form and the multithreaded application of small Householder reflectors. In the following, we will reconsider this choice for PDLAQR1.

3.1 Choice of Algorithm for Performing AED

A number of alternative choices are available for performing the Schur decomposition of the relatively small AED window:

- A recursive call to PDHSEQR or PDLAQR0, implementing the parallel QR algorithm with multishifts and AED.
- A call to PDLAQR1, a minor modification of ScaLAPACK's PDLAHQR.
- Assembling the AED window in local memory and a call to the sequential LAPACK [2] routine DLAHQR (or DLAQR4).

According to the numerical experiments in [10], a recursive call of PDLAQR0 may not be the optimal choice, mainly because of the fact that the way multishift QR iterations are implemented in PDLAQR0 suffers from poor scalability for relatively small matrices. ScaLAPACK's PDLAHQR achieves better scalability but does not incorporate modern developments, such as AED, and therefore suffers from poor performance. The third alternative, calling a sequential algorithm, should be used for submatrices that are too small to justify the overhead incurred by parallelization. In our experimental setup this was the case for submatrices of size 384 or smaller.

In this work, we propose to modify PDLAQR1 further and add AED to the parallel pipelined QR algorithm implemented in ScaLAPACK's PDLAHQR. Since the main purpose of PDLAQR1 is to handle small to medium-sized submatrices, a parallel implementation of AED, as in [10], will not be efficient on this level, since the size of the AED window is even smaller and does not allow for reasonable parallel performance in the Schur decomposition or the swapping of diagonal blocks. We have therefore chosen the third alternative for performing AED on the lowest level and invoke the sequential LAPACK routine DLAQR3 [8]. The accumulated orthogonal transformations returned by DLAQR3 are applied to the

off-diagonal blocks in parallel. Therefore, $O(\sqrt{p})$ processors are used for updating the off-diagonal blocks. A high-level description of the resulting procedure is given in Algorithm 2.

Algorithm 2. *Parallel pipelined QR algorithm with AED (new* `PDLAQR1`*)*

WHILE not converged
1. Copy the $(n_{\mathsf{win}}+1) \times (n_{\mathsf{win}}+1)$ trailing submatrix to local memory and perform sequential AED on an $n_{\mathsf{win}} \times n_{\mathsf{win}}$ window.
2. Apply the accumulated orthogonal transformations to the corresponding off-diagonal blocks in parallel.
3. **IF** enough eigenvalues have been deflated in Step 1
   GOTO Step 1.
   END IF
4. Compute the eigenvalues of a trailing submatrix.
5. Perform a pipelined QR sweep with the eigenvalues computed in Step 4 as shifts.
6. Check for negligible subdiagonal elements.

END WHILE

3.2 Implementation Details

In the following we discuss some implementation issues of Algorithm 2. The basis for our modification is `PDLAQR1` from [10], referred to as the *old* `PDLAQR1` in the following discussion. Following the notation established in the (Sca)LAPACK implementations of the QR algorithm, we let `NH=IHI-ILO+1` denote the dimension of the active `NH` $\times$ `NH` diagonal block and `NS` the number of shifts in the multishift QR sweep.

- In the special case when the active diagonal block is small enough, say `NH` $\leq$ 384, we copy this block to local memory and call `DLAHQR`/`DLAQR4` directly. The off-diagonal blocks are updated in parallel. This reduces communication while the required extra memory is negligible. We have observed that this modification reduces the total execution time by a non-negligible amount, especially during the final stages of the QR algorithm.
- The size of the deflation window, n_{win}, is determined by the return value of the LAPACK routine `IPARMQ`, see [8] for more details. In `PDLAHQR/PDLAQR1`, `NS` is mainly determined by the process grid and does not exceed 32. This is usually smaller than the number of shifts suggested by `IPARMQ`. Also, typical values of n_{win} returned by `IPARMQ` are 96, 192 and 384, which is much larger than if we chose `NS*3/2`. Based on the observation that the optimal AED window size does not depend strongly on the number of shifts used in the QR sweeps, we prefer to stick to large n_{win} rather than using `NS*3/2`. This increases the time spent on AED, but the overhead is compensated by fewer pipelined QR sweeps.
- The criterion for restarting another AED process rightaway, without an intermediate QR iteration, is the same as in LAPACK [8]:

1. The number of undeflatable eigenvalues is smaller than NS; or
2. the number of deflated eigenvalues is larger than $n_{\mathsf{win}} \times 14\%$.

Note that we choose the criterion in accordance with the window size suggested by IPARMQ.

– In contrast to Algorithm 1, undeflatable eigenvalues are not used as shifts in subsequent multishift QR sweep. This choice is based on numerical experiments with the following three shift strategies:
  1. Use undeflatable eigenvalues obtained from AED as shifts.
  2. Compute and use the eigenvalues of the NS × NS trailing submatrix after AED as shifts (by calling DLAHQR/DLAQR4).
  3. Compute and use some of the eigenvalues of the $(n_{\mathsf{win}} + 1) \times (n_{\mathsf{win}} + 1)$ trailing submatrix after AED as shifts (by calling DLAHQR/DLAQR4).

  An illustration of these strategies is given in Figure 2. Based on the experiments, we prefer the third strategy despite the fact that it is the computationally most expensive one. However, it provides shifts of better quality, mainly because of the larger window size, which was found to reduce the number of pipelined QR sweeps and to outweigh the increased cost for shift computation.

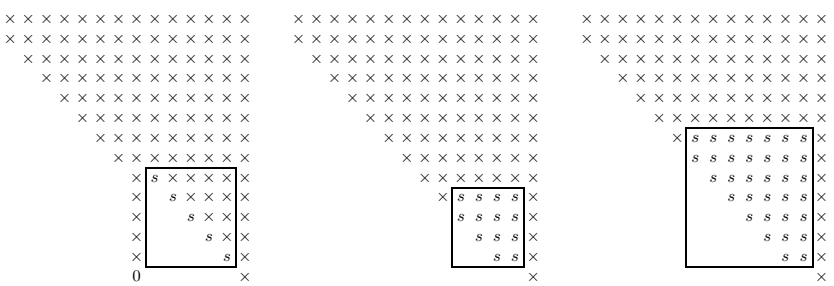

Fig. 2. Three shift strategies ($n_{\mathsf{win}} = 6$, NS=4)

– When performing AED within the new PDLAQR1, each processor receives a local copy of the trailing submatrix and calls DLAQR3 to execute the same computations concurrently. This implies redundant work performed in parallel but it reduces communication since the orthogonal transformation matrix, to be applied in parallel in subsequent updates, is readily available on each processor. A similar approach is suggested in the parallel QZ algorithm by Adlerborn et al. [1]. If the trailing submatrix is not laid out across a border of the processor mesh, we call DGEMM to perform the updates. If the trailing submatrix is located on a 2×2 processor mesh, we organize the computation and communication manually for the update. Otherwise, PDGEMM is used for updating the off-diagonal blocks.

4 Numerical Experiments

All the experiments in this section were run on the 64-bit low power Intel Xeon Linux cluster *Akka* hosted by the High Performance Computing Center North (HPC2N). Akka consists of 672 dual socket quadcore L5420 2.5GHz nodes, with 16GB RAM per node, connected in a Cisco Infiniband network. The code is compiled by the PathScale compiler version 3.2 with the flags `-O2 -fPIC -TENV:frame_pointer=ON -OPT:Olimit=0`. The software libraries OpenMPI 1.4.2, BLACS 1.1 patch3, ScaLAPACK/PBLAS 1.8.0, LAPACK 3.2.1 and GOTOBLAS2 1.13 [12] are linked with the code. No multithreaded features, in particular no mixture of OpenMP and MPI, were used. We chose `NB = 50` as the block size in the block cyclic distribution of ScaLAPACK. The test matrices are dense square matrices with entries randomly generated from a uniform distribution in [0,1]. The ScaLAPACK routine `PDGEHRD` is used to reduce these matrices initially to Hessenberg form. We only measure the time for the Hessenberg QR algorithm, i.e., the reduction from Hessenberg to real Schur form.

4.1 Improvement for `PDLAQR1`

We first consider the isolated performance of the new `PDLAQR1` compared to the old `PDLAQR1` from [10]. The sizes of the test matrices were chosen to fit the typical sizes of the AED windows suggested in [10]. Table 1 displays the measured execution time on various processor meshes.

For the sequential case (1×1 mesh), `PDLAQR1` calls the LAPACK routine `DLAQR4` directly for small matrices (see the first remark in Section 3.2). `PDLAQR0` also implements a blocked QR algorithm almost identical to the new LAPACK algorithm [8], but some algorithmic parameters (e.g., number of shifts) can be different. Since the parameters in `PDLAQR0` largely depend on the block size in the block cyclic matrix data distribution of ScaLAPACK, `PDLAQR0` can be a bit slower than LAPACK.

For determining the cross-over point for switching from `PDLAQR0` to `PDLAQR1` in the main routine `PDHSEQR`, we also measured the execution time of `PDLAQR0`.

The new implementation of `PDLAQR1` turns out to require much less execution time than the old one, with a few, practically nearly irrelevant exceptions. Also, the new `PDLAQR1` scales slightly better than `PDLAQR0`, especially when the size of matrix is not large. It is worth emphasizing that the scaling of all implementations eventually deteriorates as the number of processor increases, simply because the involved matrices are not sufficiently large to create enough potential for parallelization.

Quite naturally, `PDLAQR0` becomes faster than the new `PDLAQR1` as the matrix size increases. The dashed line in Table 1 indicates the crossover point between both implementations. A rough model of this crossover point result is given by $n = 220\sqrt{p}$, which fits the observations reasonably well and has been incorporated in our implementation.

Table 1. Execution time in seconds for old PDLAQR1 (1st line for each n), new PDLAQR1 (2nd line) and PDLAQR0 (3rd line). The dashed line is the crossover point between the new PDLAQR1 and PDLAQR0.

Matrix size	Processor mesh						
(n)	1×1	2×2	3×3	4×4	6×6	8×8	10×10
96	0.01	0.05	0.11	0.18	0.15	0.25	0.27
	0.08	0.08	0.02	0.05	0.03	0.08	0.07
	0.14	0.40	0.96	1.11	2.52	3.16	2.95
192	0.09	0.17	0.18	0.22	0.32	0.47	0.64
	0.09	0.07	0.07	0.13	0.16	0.12	0.26
	0.15	0.30	0.61	1.05	3.73	4.34	3.64
384	0.60	0.73	0.61	0.63	0.78	1.09	1.24
	0.27	0.29	0.28	0.36	0.40	0.48	0.48
	0.47	0.55	0.72	0.89	2.08	3.23	3.76
768	7.38	3.53	2.53	2.35	2.61	2.80	3.52
	3.77	2.24	1.73	1.57	1.73	2.17	2.25
	1.83	1.51	1.61	1.68	2.70	3.03	3.31
1536	133.31	20.68	13.23	11.12	9.79	10.48	13.05
	35.94	9.27	6.54	5.52	5.11	5.31	6.33
	12.34	6.61	5.63	4.86	6.26	6.76	6.84
3072	2313.61	139.05	96.73	66.06	50.64	41.82	63.22
	522.81	45.72	33.13	22.60	19.08	18.12	22.23
	80.71	30.67	21.34	15.82	15.56	15.09	14.98
6144		1049.56	623.63	351.44	231.70	199.75	227.45
		144.96	167.71	103.15	78.75	66.90	70.48
		198.54	129.58	87.07	55.40	47.61	44.07

4.2 Overall Improvement

As the main motivation for the development of the new PDLAQR1 is its application to AED within the parallel QR algorithm, we have also measured the resulting reduction of the overall execution time of PDHSEQR. From the results presented in Table 2, it is clear that PDHSEQR with the new PDLAQR1 is almost always better than the old implementation. The improvement varies between 5% and 40%. We remark that the measured execution time for the 4000×4000 problem using 64 processors is less than running the same problem on 100 processors. However, situations may occur when we prefer to solve a 4000×4000 problem using 100 processors. For example, if this is a subproblem in a large-scale computation, it would be too costly to redistribute the matrix and use only 64 of the available processors. Among the measured configurations, there is one notable exception: $n = 32000$ on a 6×6 processor grid. This is actually the only case for which PDLAQR0 is called within the AED phase, which seems to indicate that the choice of the crossover point requires some additional fine tuning.

Note that the largest AED window in all these experiments is of size 1536. According to Table 1, we expect even more significant improvements for larger matrices, which have larger AED windows.

Table 2. Execution time in seconds for old PDHSEQR (1st line for each n), new PDHSEQR (2nd line). The third lines show the relative improvement.

Processor mesh	Matrix size (n)			
	4000	8000	16000	32000
1×1	162.43			
	161.28			
	0.71%			
2×2	71.34	501.83		
	68.02	452.70		
	4.65%	9.79%		
4×4	39.18	170.75	1232.40	
	30.68	158.66	1037.93	
	22.69%	7.08%	15.78%	
6×6	35.96	123.46	617.97	3442.08
	24.62	96.23	509.38	3584.74
	31.54%	22.06%	17.57%	-4.14%
8×8	33.09	97.20	435.52	2639.32
	20.59	67.42	366.31	2016.93
	37.78%	31.64%	15.89%	24.58%
10×10	36.05	101.75	355.38	2053.16
	21.39	62.29	291.06	1646.30
	41.67%	39.58%	18.10%	19.82%

5 Summary

We have reconsidered the way AED is performed in the parallel QR algorithm [10]. A recursive approach is suggested, in which the ScaLAPACK routine PDLAHQR is combined with AED to address medium-sized problems. The focus of this work has been on minimizing the total execution time instead of how to use all the processors or how well the algorithm scales. Computational experiments demonstrate the efficiency of our approach, but also reveal potential for further improvements by a more careful fine tuning of the crossover point for switching between different implementations of the parallel QR algorithm.

Acknowledgments. We would like to thank Robert Granat, Lars Karlsson, and Åke Sandgren for their help and support.

This work was conducted using the resources of the High Performance Computing Center North (HPC2N), http://www.hpc2n.umu.se, and was supported by the Swedish Research Council under grant VR7062571, the Swedish Foundation for Strategic Research under grant A3 02:128, and UMIT Research Lab (EU Mål 2 grant).

In addition, support has been provided by eSSENCE, a collaborative e-Science programme funded by the Swedish Research Council within the framework of the strategic research areas designated by the Swedish Government.

References

1. Adlerborn, B., Kågström, B., Kressner, D.: Parallel Variants of the Multishift QZ Algorithm with Advanced Deflation Techniques. In: Kågström, B., Elmroth, E., Dongarra, J., Waśniewski, J. (eds.) PARA 2006. LNCS, vol. 4699, pp. 117–126. Springer, Heidelberg (2007)
2. Anderson, E., Bai, Z., Bischof, C.H., Blackford, S., Demmel, J.W., Dongarra, J.J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Sorensen, D.C.: LAPACK User's Guide, 3rd edn. SIAM, Philadelphia (1999)
3. Bai, Z., Demmel, J.W.: On a Block Implementation of Hessenberg Multishift QR Iteration. Intl. J. of High Speed Comput. 1, 97–112 (1989)
4. Bai, Z., Demmel, J.W.: On Swapping Diagonal Blocks in Real Schur Form. Linear Algebra Appl. 186, 73–95 (1993)
5. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J.W., Dhillon, I., Dongarra, J.J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK Users' Guide. SIAM, Philadelphia (1997)
6. Braman, K., Byers, R., Mathias, R.: The Multishift QR Algorithm. Part I: Maintaining Well-focused Shifts and Level 3 Performance. SIAM J. Matrix Anal. Appl. 23(4), 929–947 (2002)
7. Braman, K., Byers, R., Mathias, R.: The Multishift QR Algorithm. Part II: Aggressive Early Deflation. SIAM J. Matrix Anal. Appl. 23(4), 948–973 (2002)
8. Byers, R.: LAPACK 3.1 xHSEQR: Tuning and Implementation Notes on the Small Bulge Multi-shift QR Algorithm with Aggressive Early Deflation. LAPACK Working Note 187 (2007)
9. Golub, G., Uhlig, F.: The QR Algorithm: 50 Years Later Its Genesis by John Francis and Vera Kublanovskaya and Subsequent Developments. IMA J. Numer. Anal. 29(3), 467–485 (2009)
10. Granat, R., Kågström, B., Kressner, D.: A Novel Parallel QR Algorithm for Hybrid Distributed Memory HPC Systems. SIAM J. Sci. Comput. 32(4), 2345–2378 (2010) (An earlier version appeared as LAPACK Working Note 216)
11. Granat, R., Kågström, B., Kressner, D.: Parallel Eigenvalue Reordering in Real Schur Forms. Concurrency and Computat.: Pract. Exper. 21(9), 1225–1250 (2009)
12. GOTO-BLAS – High-performance BLAS by Kazushige Goto, `http://www.tacc.utexas.edu/tacc-projects/#blas`
13. Henry, G., van de Geijn, R.: Parallelizing the QR Algorithm for the Nonsymmetric Algebraic Eigenvalue Problem: Myths and Reality. SIAM J. Sci. Comput. 17, 870–883 (1997)
14. Henry, G., Watkins, D.S., Dongarra, J.J.: A Parallel Implementation of the Nonsymmetric QR Algorithm for Distributed Memory Architectures. SIAM J. Sci. Comput. 24(1), 284–311 (2002)
15. Kressner, D.: Numerical Methods for General and Structured Eigenvalue Problems. LNCSE, vol. 46. Springer, Heidelberg (2005)
16. Kressner, D.: The Effect of Aggressive Early Deflation on the Convergence of the QR Algorithm. SIAM J. Matrix Anal. Appl. 30(2), 805–821 (2008)
17. Lang, B.: Effiziente Orthogonaltransformationen bei der Eigen- und Singulärwertzerlegung. Habilitationsschrift (1997)
18. Watkins, D.S.: The Matrix Eigenvalue Problem: GR and Krylov Subspace Methods. SIAM, Philadelphia (2007)
19. Watkins, D.S.: Francis's Algorithm. Amer. Math. Monthly (2010) (to appear)

II

Paper II

Parallel Library Software for the Multishift QR Algorithm with Aggressive Early Deflation*

Robert Granat[1], Bo Kågström[1], Daniel Kressner[2], and Meiyue Shao[1]

[1] *Department of Computing Science and HPC2N*
Umeå University, SE-901 87 Umeå, Sweden
{*granat, bokg, myshao*}*@cs.umu.se*

[2] *MATHICSE, EPF Lausanne, CH-1015 Lausanne, Switzerland*
daniel.kressner@epfl.ch

Abstract: Library software implementing a parallel small-bulge multishift QR algorithm with aggressive early deflation (AED) targeting distributed memory high-performance computing systems is presented. Starting from recent developments of the parallel QR algorithm [19], we describe a number of algorithmic and implementation improvements. These include communication avoiding algorithms via data redistribution and a refined strategy for balancing between multishift QR sweeps and AED. Guidelines concerning several important tunable algorithmic parameters are provided. As a result of these improvements, AED is no longer a computational bottleneck in the parallel QR algorithm. A performance model is established to explain the scalability behavior of the new parallel QR algorithm. Numerous computational experiments confirm that our new implementation significantly outperforms previous parallel implementations of the QR algorithm. The new software is available as a part of ScaLAPACK version 2.0.

1 Introduction

The QR algorithm is the method of choice for computing all eigenvalues of a non-symmetric matrix $A \in \mathbb{R}^{n \times n}$. This paper describes a novel parallel implementation of

* Report UMINF-12.06.
The work is supported by the Swedish Research Council under grant A0581501, UMIT Research Lab via an EU Mål 2 project, and eSSENCE, a strategic collaborative e-Science programme funded by the Swedish Research Council.

the QR algorithm for distributed memory architectures. While our implementation is largely based on the algorithms described in [19], a number of additional algorithmic improvements have been made, leading to significantly reduced execution times and higher robustness.

In the following, we give a brief history of serial and parallel implementations of the QR algorithm. The algol procedure `hqr` by Martin, Petersen, and Wilkinson [27] was among the first computer implementations of the QR algorithm. A Fortran translation of this procedure was included in EISPACK [32] as routine `HQR`. The initial version of the LAPACK routine `DHSEQR` was based on work by Bai and Demmel [5]; the most notable difference to `HQR` was the use of multishift techniques to improve data locality. This routine had only seen a few minor modifications [2] until LAPACK version 3.1, when it was replaced by an implementation incorporating pipelined bulges and aggressive early deflation techniques from the works by Braman, Byers, and Mathias [11, 12]. This implementation is described in more detail in [13]. While there has been a lot of early work on parallelizing the QR algorithm, for example in [20, 30, 34, 35, 36, 38], the first publicly available parallel implementation was released only 1997 in ScaLAPACK [10] version 1.5 as routine `PDLAHQR`, based on work by Henry, Watkins, and Dongarra [21]. A complex version `PZLAHQR` of this routine was included later on [15]. In this work, we describe a new parallel implementation of the QR algorithm that aims to replace `PDLAHQR`. It might be interesting to note that all recently released high-performance linear algebra packages, such as MAGMA and PLASMA [1], ELPA [3], FLAME [9] lack adapted implementations of the QR algorithm or other nonsymmetric eigenvalue solvers.

Given a *nonsymmetric* matrix A, the parallel implementation of the eigenvalue solver in ScaLAPACK consists of the following steps. In the first optional step, the matrix is balanced, that is, an invertible diagonal matrix D is computed to make the rows and columns of $A \leftarrow D^{-1}AD$ as close as possible. In the second step, A is reduced to Hessenberg form: $H = Q_0^T A Q_0$ with an orthogonal matrix Q_0 and $h_{ij} = 0$ for $i \geq j+2$. In the third step, the QR algorithm iteratively reduces H further to real Schur form, eventually resulting in an orthogonal matrix Q such that

$$T = Q^T H Q \tag{1}$$

is quasi-upper triangular. This means that T is block upper triangular with 1×1 blocks (corresponding to real eigenvalues) and 2×2 blocks (corresponding to complex conjugate eigenvalue pairs) on the diagonal. Therefore, the Schur decomposition of A is $A = ZTZ^T$, where $Z = Q_0 Q$. The last optional step consists of computing the eigenvectors of T and performing a back transformation to obtain the eigenvectors of the original matrix A. This paper is only concerned with the reduction to real Schur form (1). In particular, we will not discuss the implementation of Hessenberg reduction, see [26, 24, 33] for recent developments in this direction.

The rest of this paper is organized as follows. Section 2 provides a summary of our implementation and the underlying algorithm, emphasizing improvements over [19]. In Section 3, we develop a performance model that provides insights into the cost of computations and communication. This model is then used to guide the choice of the

parameters in Section 4. Finally, in Section 5, we evaluate the performance of our parallel library software by a large set of numerical experiments.

2 Algorithms and Implementation

Modern variants of the Hessenberg QR algorithm usually consists two major components — multishift QR sweep and AED. A typical structure of the modern QR algorithm in a sequential or parallel setting is provided in Algorithm 1. Our parallel QR algorithm is essentially built on Algorithm 1 but there are several issues to be considered for reaching high performance. Figure 1 shows the software hierarchy of our implementation of the parallel multishift QR algorithm. Details of the algorithm and some implementation issues are discussed in the successive subsections.

Algorithm 1 *Multishift Hessenberg QR Algorithm with AED*

WHILE not converged

1. Perform AED on the $n_{\text{AED}} \times n_{\text{AED}}$ trailing principle submatrix.
2. Apply the accumulated orthogonal transformation to the corresponding off-diagonal blocks.
3. **IF** a large fraction of eigenvalues has been deflated in Step (1)

   GOTO Step (1).

   END IF
4. Perform a multishift QR sweep with n_{shift} undeflatable eigenvalues from Step (1) as shifts.
5. Check for negligible subdiagonal elements.

END WHILE

2.1 Data layout convention in ScaLAPACK

In ScaLAPACK, the $p = p_r p_c$ processors are usually arranged into a $p_r \times p_c$ grid. Matrices are distributed over the rectangular processor grid in a *2D block-cyclic layout* with block size $m_b \times n_b$ (see an example in Figure 2). The information regarding the data layout is stored in an *array descriptor* so that the mapping between entries of the global matrix and their corresponding locations in the memory hierarchy can be established. We adopt ScaLAPACK's data layout convention and require that the $n \times n$ input matrices H and Z have identical data layout with square data blocks (i.e., $m_b = n_b$). However, the processor grid need not to be square unless explicitly specified.

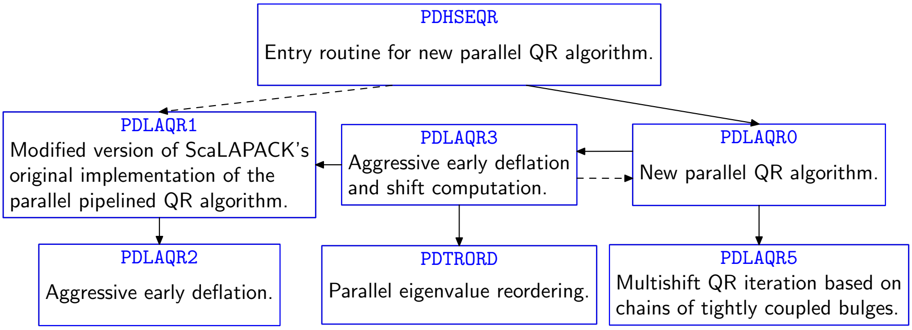

FIGURE 1: Software hierarchy of the multishift QR algorithm with AED

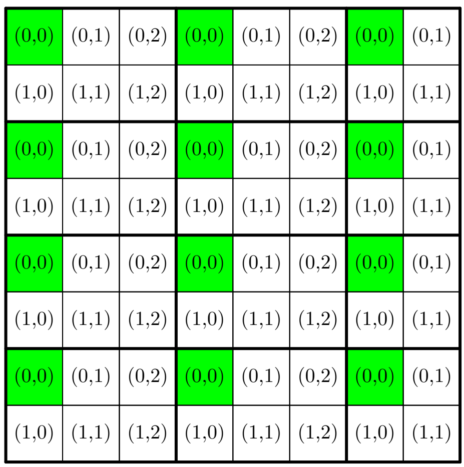

FIGURE 2: The 2D block-cyclic data layout across a 2×3 processor grid

2.2 Multishift QR sweep

The multishift QR sweep is a bulge chasing process which involves several shifts. The QR sweep applied to a Hessenberg matrix H with k shifts $\sigma_1, \sigma_2, \ldots, \sigma_k$ yields another Hessenberg matrix $Q^T HQ$ where Q is determined by the QR decomposition of the shift polynomial:

$$(H-\sigma_1 I)(H-\sigma_2 I)\cdots(H-\sigma_k I) = QR.$$

Thanks to the *Implicit Q theorem* (e.g., see [16, 17]), there is a lot of freedom regarding how to perform the QR sweep. In ScaLAPACK v1.8.0 and earlier versions, PDLAHQR uses the pipelined approach which chases a chain of loosely coupled bulges (see Figure 3(a)). Most operations in the pipelined QR algorithm have the computational intensity between level 1 and level 2 BLAS. Therefore it typically has low node performance and requires frequent communication between processors as well. To avoid this shortcoming in the pipelined QR algorithm, we use several chains of tightly

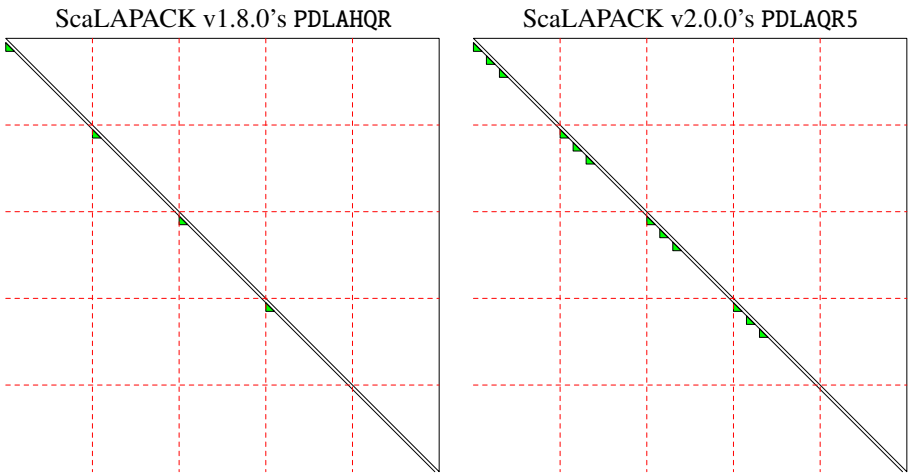

FIGURE 3: Loosely coupled shifts v.s. tightly coupled shifts. The dashed lines represent borders of the processor grid. Only parts of the matrix are displayed. Interblock bulge chasings can be performed independently and in parallel.

coupled bulges (see Figure 3(b)) to improve the node performance and reduce communication. In the new bulge chasing routine PDLAQR5, the total number of shifts (n_{shift}) used in one QR sweep is shown in Table 1, and is usually much larger than the value in the pipelined approach. These shifts are divided into several chains of tightly coupled bulges with up to $\lfloor n_b/3 \rfloor$ shifts per chain so that the length of each chain does not exceed $n_b/2$. The chains are placed on different diagonal blocks, such that $\Theta(\sqrt{p})$ chains can be chased locally and simultaneously. The corresponding off-diagonal blocks are updated by explicitly multiplying the orthogonal matrix accumulated in the diagonal chasing step. This delay-and-accumulate technique leads to level 3 computational intensity. When the chains are passing through the processor border, they are chased in an odd-even manner (see Figure 4) to avoid conflicts between different tightly coupled chains. We refer to [19] for detailed descriptions of the blocked version of the multishift bulge chasing algorithm.

2.3 Aggressive Early Deflation (AED)

Firstly, we summarize the AED technique proposed by Braman, Byers, and Mathias [12]. Let $H \in \mathbb{R}^{n\times n}$ be an upper Hessenberg matrix with partitioning

$$H = \begin{array}{c} \\ n-n_{\text{AED}}-1 \\ 1 \\ n_{\text{AED}} \end{array} \begin{array}{c} \begin{array}{ccc} n-n_{\text{AED}}-1 & 1 & n_{\text{AED}} \end{array} \\ \begin{pmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ 0 & H_{32} & H_{33} \end{pmatrix} \end{array},$$

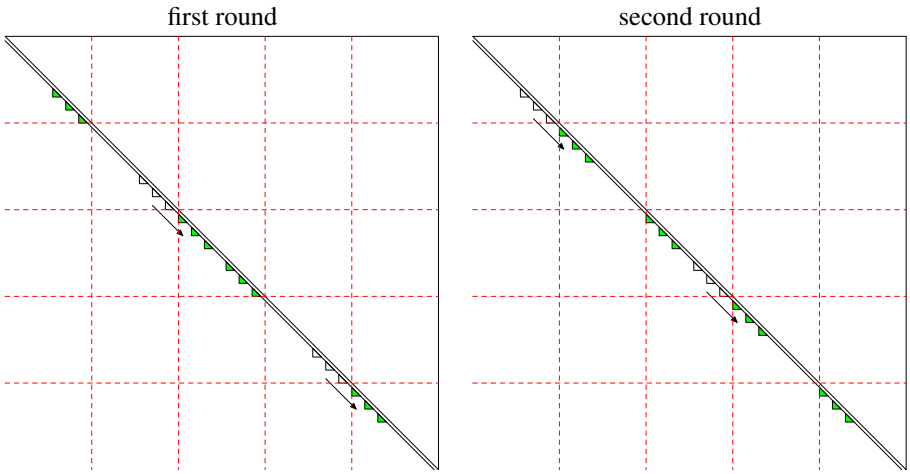

FIGURE 4: Intrablock bulge chasing. Odd-numbered chains and even-numbered chains are chased separately in two rounds.

TABLE 1: Recommended values for n_{shift} and n_{AED}.

matrix size (n)	n_{shift}	n_{AED}
<6K	see [13]	
6K–12K	256	384
12K–24K	512	768
24K–48K	1024	1536
48K–96K	2048	3072
96K–192K	4096	6144
192K–384K	8192	12288
384K–768K	16384	24576
768K–1000K	32768	49152
> 1M	$\lceil n/25 \rceil$	$3n_{\text{shift}}/2$

where H_{33} is the so called *AED window*. By computing the (real) Schur decomposition $H_{33} = VTV^T$, and applying V in a similarity transformation of H, we obtain

$$U^T H U = \begin{pmatrix} H_{11} & H_{12} & H_{13}V \\ H_{21} & H_{22} & H_{23}V \\ 0 & s & T \end{pmatrix},$$

where

$$U = \begin{array}{c} \\ n-n_{\text{AED}}-1 \\ 1 \\ n_{\text{AED}} \end{array} \begin{array}{c} \begin{array}{ccc} n-n_{\text{AED}}-1 & 1 & n_{\text{AED}} \end{array} \\ \begin{pmatrix} I & & \\ & 1 & \\ & & V \end{pmatrix} \end{array}.$$

The vector $s \in \mathbb{R}^{n_{\text{AED}}}$ is the so called *spike*, created from the first entry of H_{32}. The last diagonal entry (or 2×2 diagonal block) of T can be deflated if the magnitude of the last component (or the last two components) of the spike is negligible. Undeflatable eigenvalues are moved to the top left corner of T by a swapping algorithm [6, 18]. The orthogonal transformations for reordering eigenvalues in the Schur form of the AED window are accumulated in an $n_{\text{AED}} \times n_{\text{AED}}$ orthogonal matrix. By repeating the same procedure to all diagonal entries (or 2×2 blocks) of T, the eigenvalues of T are checked subsequently and possibly deflated. Then the entire matrix is reduced back to upper Hessenberg form and the off-diagonal blocks H_{13} and H_{23} are multiplied by $\widetilde{V}$, the product of all involved orthogonal transformations. Typically, the size of the AED window is recommended to be somewhat larger, e.g., by 50%, than the number of shifts in the multishift QR sweeps [11, 13]. The recommend values of n_{AED} are listed in Table 1.

In principle, aggressive early deflation can be incorporated into any variant of the QR algorithm. Therefore both the new multishift QR algorithm and the pipelined QR algorithm can benefit from performing AED. We discuss them separately.

2.3.1 AED in the new multishift QR algorithm

Since our algorithm is designed for solving large-scale problems, the AED window can be quite large. However, it is not reasonable to expect that executing AED locally and sequentially yields good performance. Hence, `PDLAQR3`, the routine for performing AED, requires a parallel approach.

The first and most costly step of AED is to calculate the Schur decomposition of H_{33}, the $n_{\text{AED}} \times n_{\text{AED}}$ trailing principal submatrix of H. This eigenvalue problem can be solved by either recursively using the new multishift QR algorithm (`PDLAQR0`) or using the pipelined QR algorithm (`PDLAQR1`). The choice of the solver is determined by the size of the AED window as well as the number of processors used. Since n_{AED} is relatively small compared to n, the number of available processors may be too large to facilitate all of them without causing significant communication overhead compared to the relatively small computational window. In this case we only use a subset of the processors to reduce the overhead and minimizing the execution time. We provide a more detailed discussion on this issue in Section 2.5.

In the deflation checking phase, the reordering algorithm is arranged in a blocked manner to reduce memory transfers and communications. Unlike the standard procedure, the undeflatable eigenvalues are not directly moved to the top left corner of the whole AED window. They are only reordered within an $n_b \times n_b$ computational window instead. After all eigenvalues in this $n_b \times n_b$ window are checked, the group of undeflatable eigenvalues are moved simultaneously to the top left corner of the AED window. This blocked approach increases the computational intensity and avoids the frequent communication needed when reordering each eigenvalue individually. The procedure is repeated until all eigenvalues in the AED window are checked.

The last step is to eliminate the spike and reduce the AED window back to the upper Hessenberg form. This task can be performed with the ScaLAPACK routine `PDGEHRD`. The corresponding off-diagonal blocks can be updated by explicitly multiplying the accumulated orthogonal matrix using the PBLAS routine `PDGEMM`.

2.3.2 AED in the pipelined QR algorithm

The ScaLAPACK v1.8.0 implementation of the pipelined QR algorithm is not equipped with the AED strategy. Since the pipelined QR algorithm is suitable for matrices from small to medium size, we can expect the AED window to be sufficiently small such that AED can be performed on one processor efficiently. In this case, we can make use of the LAPACK implementation of AED. Our modification of the ScaLAPACK routine `PDLAHQR` is called `PDLAQR1`. Compared to the original routine, `PDLAQR1` turns out to be both faster and more robust. The following list summarizes the most important modifications; detailed descriptions can be found in [25, 31].

- Aggressive early deflation: AED is implemented in an auxiliary routine `PDLAQR2` which copies the AED window to local memory and calls the LAPACK routine `DLAQR3` to solve the problem sequentially. To determine the parameters of AED, we use the settings of the LAPACK installation determined by `ILAENV`. However, a notable difference is that we do *not* use the undeflatable eigenvalues from the AED step as shifts in the subsequent pipelined QR sweep. Instead we recompute the eigenvalues of the trailing submatrix to improve the quality of the shifts. We have observed that this shifting strategy accelerates the convergence of the pipelined QR algorithm.

- Conventional deflation: In `PDLAHQR`, pipelined QR sweeps are performed until the very end, that is, the remaining diagonal blocks are all of size 1×1 or 2×2. In `PDLAQR1`, we use a different strategy: Once the active block is not too large (say, up to size 385×385), we copy this block to local memory and call the LAPACK routines `DLAHQR/DLAQR4` to solve the problem sequentially. This strategy significantly reduces communication overhead in the latter stages and is implemented in an auxiliary routine `PDLAQR4`.

- Avoidance of anomalies: The original ScaLAPACK routine `PDLAHQR` suffered from two anomalies, which have been removed.

  First, the routine sometimes returned 2×2 diagonal blocks containing real eigenvalues. In `PDLAQR1`, each 2×2 diagonal block contains a pair of complex conjugate eigenvalues.

  The second issue is concerned with a strategy already proposed by Francis [16]. Bulges can sometimes be introduced from the middle of an Hessenberg matrix if there are two consecutive small sub-diagonal entries. However, this turns out to be difficult to implement in a safe manner in pipelined or multishift QR sweeps. Moreover, the performance improvements gained from this strategy are usually negligible. We have therefore decided to remove it. In return, numerical stability of the solver is improved, avoiding large relative residuals of norm up to 10^{-5} that had been observed when this strategy implemented in the ScaLAPACK routine `PDLACONSB` was used.

2.4 Switching between Multishift QR and AED

In the LAPACK implementations of the QR algorithm, there are rules for balancing the cost between multishift QR sweeps and AED, see, e.g., Step (3) in Algorithm 1.

The precise meaning of "a large fraction of eigenvalues has been deflated" can be characterized by a threshold called NIBBLE. If we let n_{undflt} denote the number of undeflatable shifts in an AED step, the multishift QR sweep is skipped if

$$\frac{n_{\text{AED}} - n_{\text{undflt}}}{n_{\text{AED}}} \geq \frac{\texttt{NIBBLE}}{100}.$$

Since AED behaves differently for different matrices, this strategy automatically adjusts the choice between AED and QR sweeps based on the properties of the matrix.

For the sequential QR algorithm, the default value of NIBBLE in LAPACK is 14 which provides a good balance between multishift QR sweeps and AED. In the pipelined QR algorithm, we adopt the same default value as in the serial case, since the performance is mainly determined by the convergence rate. However, in the new parallel QR algorithm, the parallel AED process becomes substantially more expensive than the sequential AED process due to communication. As explained above, the AED process only involves a smaller trailing submatrix, leading to decreased efficiency relative to the blocked version of multishift QR sweeps, which involves the full active submatrix. To account for this efficiency decrease, we increase NIBBLE to avoid performing AED too frequently. A good choice of this threshold depends both on the size of the matrix H and the number of processors involved. We use the model $\texttt{NIBBLE} = a \cdot n^b p^c$ for this purpose, where a, b and c are machine-dependent constants. An appropriate choice of these constants can be gained from repeated runs of the program with different thresholds. It turns out that the right choice of NIBBLE becomes rather sensitive when communication is slow. (In our numerical experiments, the default values on Akka[1] and Abisko[2] are $(a,b,c) = (335,-0.44,0.5)$ and $(a,b,c) = (269,-0.47,0.55)$, respectively.) In the software we provide NIBBLE=PILAENVX(ISPEC=14) as the parameter which balances the choice between multishift QR sweeps and AED.

We need to remark that when NIBBLE > 33, it can occur that the number of undeflatable eigenvalues is less than the desired number of shifts (i.e. $n_{\text{undflt}} < n_{\text{shift}}$), due to the fact that $n_{\text{AED}} = 3n_{\text{shift}}/2$. The solution in the software is that we still use these undeflatable eigenvalues as shifts in the next QR sweep as long as $n_{\text{undflt}} \geq n_{\text{shift}}/2$. However, the condition $n_{\text{undflt}} \geq n_{\text{shift}}/2$ may also fail when NIBBLE ≥ 67. In this case we calculate the eigenvalues of the $n_{\text{shift}} \times n_{\text{shift}}$ trailing principal submatrix of H and use them as shifts. The calculation can be performed by either PDLAQR0 or PDLAQR1, just like computing the Schur decomposition in the AED step.

2.5 Avoiding communications via data redistribution

It has been observed in [25] that once the matrix is not sufficiently large, it is difficult to efficiently solve the relatively small eigenvalue problem with many processors because of heavy communications. Sometimes the execution time can even increase when increasing the number of processors. This is a typical situation when calculating the Schur decomposition of the AED window and often leads to a bottleneck in the algorithm. To resolve this problem, it is important to reduce the communication overhead when handling these relatively small submatrices.

[1] http://www.hpc2n.umu.se/node/120
[2] http://www.hpc2n.umu.se/node/724

Recent work on communication avoiding algorithms [8, 7, 22, 23] usually focuses on the design of algorithms that can attain the theoretical lower bounds of the communication cost. A basic assumption in these theoretical analyses is that the data are nearly evenly distributed over the processors. Here we propose an alternative approach, which does not rely on this assumption and is especially useful for operations involving smaller submatrices.

We first consider a simple and extreme case. Suppose there is one processor which has a large amount of local memory and very high clock speed. Then by gathering all data to this processor, the problem can be solved without further communications. Once the computation is completed, the data are scattered to their original owners. The total amount of communications does not exceed the cost of scattering and gathering regardless of the complexity of computational work. Although this simple idea does not work for large problems that cannot be stored on a single processor, it is still useful for smaller problems. For example, the AED process in the pipelined QR algorithm is implemented in such a manner since we know in advance that the AED window is always sufficiently small, such that the associated Schur decomposition can be efficiently solved sequentially. By introducing the overhead of data redistribution, the total amount of communications as well as the execution time can be reduced.

For many large problems, it is not feasible to solve them sequentially via data redistribution. For example, when solving large-scale eigenvalue problems with the new parallel QR algorithm, the AED window is also large although it is much smaller compared to the whole matrix. Very often the number of processors is more than needed for the relatively small eigenvalue problem in the AED window since the communications among the processors are heavy compared to the computational work on each processor. In this case we can choose a subset of processors instead of a single processor for performing AED. The number of processors is often chosen by minimizing the total execution time. By using the general purpose data redistribution routine `PDGEMR2D` in ScaLAPACK, the data redistribution can be done efficiently so that its overhead is negligible in practice relative to the AED process.

In the software $p_{\min}$ = `PILAENVX(ISPEC = 23)` is provided as a tunable parameter for this purpose, where $p_{\min} \times p_{\min}$ is the size of the processor grid involving AED. If $\min(p_r, p_c) > p_{\min} + 1$ we redistribute the AED window to a $p_{\min} \times p_{\min}$ processor grid and do the calculations on this subset of processors. The same strategy is also applied if we need to compute shifts after an AED step. The default value for this parameter is $p_{\min} = \lceil n_{\text{AED}}/(n_b\lceil 384/n_b\rceil)\rceil$. Or it can be roughly interpreted as $p_{\min} \approx n_{\text{AED}}/384$, i.e., each processor needs to own at least 384 columns of the AED window. The constant 384 here certainly depends on the architecture. By such a requirement of minimum work load, the ratio between computations and communication is not too small so that we have some control of the overall performance. In summary and contradictory to the standard communication avoiding approach, our technique to reducing the total execution time is to first increase the communication by doing redistribution of data to a subset $p_{\min} \times p_{\min}$ of the total number of processors p and then perform the computations on this subset much more efficiently than using the p processors and at the same time outweighing the extra data redistribution costs.

3 Performance model

In this section, we analyze the cost of computations and communications of the Schur decomposition performed by the new parallel QR algorithm. For simplicity we consider a square processor grid, i.e. $p_r = p_c = \sqrt{p}$. In addition, we assume that each processor contains reasonably many data blocks of the matrices, i.e. $\sqrt{p}\, n_b \ll n$, so that the work load is balanced. The parallel execution time consists of two main components:

$$T_p = T_a + T_c,$$

where T_a and T_c are the times for arithmetic operations and communications, respectively. The model is a worst case scenario, since overlapping communications and computations is not considered. Usually it is not harmful to neglect the communications between memory and cache lines inside one core since they are much cheaper than the communications between processors. Therefore the serial runtime can often be approximated by

$$T_a = \frac{\#(\text{flops})}{f(p)}\gamma$$

where γ is the average time for performing one floating point operation and $f(p)$ is the degree of concurrency. For the communications between two processors, we define α and β as the start-up time (or communication latency) and the time for transferring one word without latency (or reciprocal of bandwidth), respectively. The time for a single point-to-point communication is modelled as $\alpha + L\beta$ where L is the message size in words. A one-to-all broadcast or an all-to-one reduction within a scope of p processors is assumed to take $\Theta(\log p)$ steps.

In the new parallel Hessenberg QR algorithm, let k_{AED} and k_{QR} denote the number of AED steps and QR sweeps, respectively. Since some QR sweeps are skipped because the percentage of deflated eigenvalues in the AED step is larger than the threshold (`NIBBLE`), we have $k_{\text{QR}} \le k_{\text{AED}}$. Sometimes when a QR sweep is not skipped but the AED step does not provide enough shifts, we need to calculate shifts from the trailing submatrix. The number of extra calls to the parallel Schur decomposition solver in this case is denoted by k_{shift}, which naturally satisfies $k_{\text{shift}} \le k_{\text{QR}}$. Right now we do not make any further assumption for k_{AED}, k_{QR}, and k_{shift} other than

$$0 \le k_{\text{shift}} \le k_{\text{QR}} \le k_{\text{AED}},$$

since these constants heavily depend on the property of H as well as many tunable algorithmic parameters. Given the constants k_{AED}, k_{QR}, and k_{shift}, the execution time of the parallel QR algorithm is modelled as the sum of the corresponding phases, i.e.

$$T(n,p) = k_{\text{AED}} T_{\text{AED}}(n, n_{\text{AED}}, p) + k_{\text{QR}} T_{\text{QR}}(n, n_{\text{shift}}, p) + k_{\text{shift}} T_{\text{shift}}(n, n_{\text{shift}}, p), \qquad (2)$$

where T_{AED}, T_{QR}, and T_{shift} are the runtime for each phase in one QR iteration. To simplify the discussion, the window sizes n_{AED} and n_{shift} are assumed unchanged in different QR iterations. This assumption also helps to reduce the impact of load imbalance due to deflation. We will make further assumptions and estimations about T_{AED}, T_{QR}, and T_{shift}, which are stated in the discussions that follow. Tiny terms, especially lower order terms with small coefficients, are omitted in the discussions.

3.1 Estimating T_{QR}

The QR sweep is relatively simple because the computation and communication cost is well-determined by n, n_{shift} and p. Usually there are up to $\sqrt{p}$ simultaneous computational windows, one at each diagonal processor in the grid, with at most $n_b/3$ shifts in each window. If $n_{\text{shift}} > \sqrt{p}\,n_b/3$, these shifts are chased in several rounds. So we use a rough approximation $n^*_{\text{shift}} = \sqrt{p}\,n_b/3$ to represent the total amount of shifts which can be chased simultaneously in the QR sweep. Based on the assumption $\sqrt{p}\,n_b \ll n$, the overhead for the start-up and ending phases of the bulge chasing are not important. Therefore the cost of one QR sweep is roughly

$$T_{\text{QR}}(n, n_{\text{shift}}, p) = \frac{n_{\text{shift}} n}{n^*_{\text{shift}} n_b}(T_{\text{local}} + T_{\text{cross}}),$$

where T_{local} and T_{cross} represent the runtime for local and crossborder bulge chasing, respectively. Both parts require chasing the chain of bulges with $n_b/2$ steps inside the computational window, as well as updating the corresponding off-diagonal blocks. Hence the runtime for arithmetic operations is $\left(4n_b^3 + 4nn_b^2/\sqrt{p}\right)\gamma$, half of which is for accumulating the orthogonal matrix Q. The only communication cost in the local chasing phase is broadcasting the accumulated orthogonal matrix rowwise and columnwise in the processor grid, which requires $\log_2 p\left(\alpha + n_b^2\beta\right)$ runtime, i.e.,

$$T_{\text{local}} = \left(4n_b^3 + \frac{4nn_b^2}{\sqrt{p}}\right)\gamma + \log_2 p\left(\alpha + n_b^2\beta\right) \approx \frac{4nn_b^2}{\sqrt{p}}\gamma + \log_2 p\left(\alpha + n_b^2\beta\right).$$

One round crossborder chasing requires at least the same amount of communication as in one local chasing step, with some extra cost for explicitly forming the $n_b \times n_b$ computational window and exchanging data with processor neighbours for updating the off-diagonal blocks. Notice that usually there are two rounds for a crossborder chasing step, therefore we have

$$T_{\text{cross}} = 2\left[T_{\text{local}} + 3\left(\alpha + \frac{n_b^2}{4}\beta\right) + 3\left(\alpha + \frac{nn_b}{2}\beta\right)\right],$$

and then

$$\begin{aligned} T_{\text{QR}}(n, n_{\text{shift}}, p) &\approx \frac{12n^2 n_{\text{shift}} n_b}{\sqrt{p}\,n^*_{\text{shift}}}\gamma + \frac{3nn_{\text{shift}}}{n_b n^*_{\text{shift}}}(\log_2 p + 4)\alpha + \frac{3n^2 n_{\text{shift}}}{n^*_{\text{shift}}}\beta \\ &= \frac{36n^2 n_{\text{shift}}}{p}\gamma + \frac{9nn_{\text{shift}}}{\sqrt{p}\,n_b^2}(\log_2 p + 4)\alpha + \frac{9n^2 n_{\text{shift}}}{\sqrt{p}\,n_b}\beta. \end{aligned} \tag{3}$$

From this model, we can see that the cost for updating the off-diagonal blocks dominates in both the computation and communication parts, under the assumption that $\sqrt{p}\,n_b \ll n$ (or equivalently $n^*_{\text{shift}} \ll n$). As a byproduct, the performance model of a plain multishift QR algorithm without AED can also be obtained. By assuming the convergence rate as $\Theta(1)$ shifts per eigenvalue, i.e. $k_{\text{QR}} = \Theta(n/n_{\text{shift}})$ and neglecting the

cost for generating shifts, the total execution time of a plain multishift QR algorithm is

$$T(n,p) = \Theta\Big(\frac{n^3}{p}\Big)\gamma + \Theta\Big(\frac{n^2 \log p}{\sqrt{p}\, n_b^2}\Big)\alpha + \Theta\Big(\frac{n^3}{\sqrt{p}\, n_b}\Big)\beta. \tag{4}$$

The serial part indicates that the degree of concurrency is $\Theta(p)$ which is perfect in some sense. The communication cost is a bit larger than the theoretical lower bound provided in [7].

3.2 Estimating T_{AED} and T_{shift}

The AED phase requires computing the Schur decomposition of the AED window, which might contain a lot of uncertainties. Among the several possible choices of the eigensolver, we assume that the Schur decomposition is always solved by the pipelined QR algorithm with AED on a $\sqrt{p_{\text{AED}}} \times \sqrt{p_{\text{AED}}}$ processor grid — a subset of the $\sqrt{p} \times \sqrt{p}$ processor grid, so that the property $\sqrt{p_{\text{AED}}}\, n_b \ll n_{\text{AED}}$ is also valid inside the AED window. Similar assumptions regarding the choice of eigensolver and data redistribution are made for the shift calculation phase. The relationship among these parameters are typically

$$n_{\text{shift}} \approx \frac{2}{3} n_{\text{AED}} \approx \frac{1}{C_1} n \quad \text{and} \quad \frac{n_{\text{AED}}}{\sqrt{p_{\text{AED}}}} \approx \frac{n_{\text{shift}}}{\sqrt{p_{\text{shift}}}} \geq C_2,$$

where C_1 and C_2 are constants (e.g., $C_1 \approx 25$, $C_2 \approx 384$). Then the execution time for one step AED is modelled as

$$\begin{aligned} T_{\text{AED}}(n, n_{\text{AED}}, p) = {} & T_{\text{redist}}(n_{\text{AED}}, p, p_{\text{AED}}) + T_{\text{Schur}}(n_{\text{AED}}, p_{\text{AED}}) \\ & + T_{\text{reorder}}(n_{\text{AED}}, p) + T_{\text{Hess}}(n_{\text{AED}}, p) + T_{\text{update}}(n, n_{\text{AED}}, p). \end{aligned} \tag{5}$$

where the terms in the right-hand-side represent the runtime for data redistribution, Schur decomposition of the AED window, deflation checking and reordering of eigenvalues, Hessenberg reduction, and updating the off-diagonal blocks corresponding to the AED window, respectively. We estimate these terms one by one.

- T_{redist}: The general-purpose data redistribution routine `PDGEMR2D` in ScaLAPACK uses the algorithm described in [28]. Since the scheduling part is tiny compared to the communication part, the complexity of data redistribution is provided as

$$T_{\text{redist}}(n_{\text{AED}}, p, p_{\text{AED}}) = \Theta(p)\,\alpha + \Theta\Big(\frac{n_{\text{AED}}^2}{\sqrt{p\, p_{\text{AED}}}}\Big)\beta. \tag{6}$$

- T_{Schur}: The complexity of the Schur decomposition performed by `PDLAQR1` largely depends on the property of the matrix, since AED affects the convergence rate significantly. To obtain an estimate of the complexity, we assume that AED roughly reduces the number of pipelined QR sweeps by half. According to the experimental results presented in [25], this assumption usually

provides a reasonable upper bound of the runtime, although it can be overestimated. Using the model in [21], we obtain an approximate execution time

$$\widetilde{T}_{\text{Schur}}(n,p) = \frac{40n^3}{p}\gamma + \frac{3n^2}{\sqrt{p}\,n_b}(\log_2 p + 2)\alpha + \Big(\frac{3n^2\log_2 p}{\sqrt{p}} + \frac{16n^3}{pn_b}\Big)\beta, \quad (7)$$

in which the cost of accumulating Q and tiny terms are omitted. If the orthogonal matrix Q is also wanted, the arithmetic operations need to be doubled, i.e.,

$$T_{\text{Schur}}(n,p) = \widetilde{T}_{\text{Schur}}(n,p) + \frac{40n^3}{p}\gamma \quad (8)$$

$$= \Theta\Big(\frac{n^3}{p}\Big)\gamma + \Theta\Big(\frac{n^2\log p}{\sqrt{p}\,n_b}\Big)\alpha + \Theta\Big(\frac{n^2\log p}{\sqrt{p}} + \frac{n^3}{pn_b}\Big)\beta. \quad (9)$$

It is interesting to make an comparison between (9) and (4). We are able to see both solvers have ideal degree of concurrency. However, the tightly coupled shifting strategy is superior to the loosely coupled one because it communicates much less frequently. The number of messages is reduced by a factor of $\Theta(n_b)$; the average message length is increased a lot although sometimes the total amount of data transfered is also increased. Another important observation is that the serial term in T_{Schur} assumes level 3 node performance, which is far better than the actual case. Therefore in practice the pipelined QR algorithm is usually much slower than the new multishift QR for large matrices.

- T_{reorder}: Obviously, the cost for eigenvalue reordering depends on the deflation ratio. However, we are able to evaluate the upper bound for the cost — all eigenvalues are involved in the reordering. Then the performance model is almost the same as that of QR sweeps, since updating the off-diagonal blocks is the dominant operation. Notice that each eigenvalue needs to move $n_{\text{AED}}/2$ steps in average, so the overall cost for eigenvalue reordering inside the AED window is bounded by

$$T_{\text{reorder}}(n_{\text{AED}},p) \approx \frac{4n_{\text{AED}}^2 n_b}{\sqrt{p}}\gamma + \frac{2n_{\text{AED}}}{n_b}(\log_2 p + 3)\alpha + \frac{3n_{\text{AED}}^2}{2}\beta. \quad (10)$$

As a different feature compared to QR sweeps or the analysis in [18], the degree of concurrency here is $\Theta(\sqrt{p})$ instead of $\Theta(p)$ since usually there are only two computational windows for the reordering phase inside the AED window.

- T_{Hess}: The Hessenberg reduction routine `PDGEHRD` uses the parallel algorithm described in [14]. Almost all computations and communications are performed on matrix-vector and matrix-matrix multiplications. Therefore we need to model these PBLAS operations first. The level 2 operations GEMV and GER require

$$T_{\text{GEMV}}(m,n,p) \approx T_{\text{GER}}(m,n,p) \approx \frac{2mn}{p}\gamma + \log_2 p\Big(\alpha + \frac{m+n}{2\sqrt{p}}\beta\Big), \quad (11)$$

where $m \times n$ is the size of the matrix. This model can be directly generalized to multiplying two $m \times k$ and $k \times n$ matrices so long as $\min\{m,n,k\} \leq n_b$ since it is merely a "fat" level 2 operation. In the Hessenberg reduction algorithm, all level 3 operations are "fat" level 2 operations, so the cost for one GEMM operation can be modelled as

$$T_{\text{GEMM}}(m,n,n_b,p) \approx T_{\text{GEMM}}(m,n_b,n,p) \approx \frac{2mnn_b}{p}\gamma + \log_2 p\left(\alpha + \frac{(m+n)n_b}{2\sqrt{p}}\beta\right). \quad (12)$$

Using these simple models of PBLAS operations, we are able to establish a model for T_{Hess}. The level 2 part consists roughly of n matrix-vector multiplications of dimension $n \times (n-j)$ (for $j = 1, 2, \ldots, n$). Therefore the cost is

$$T_{\text{level2}} = \sum_{j=1}^{n}\left[\frac{2n(n-j)}{p}\gamma + \log_2 p\left(\alpha + \frac{2n-j}{2\sqrt{p}}\right)\right] \approx \frac{n^3}{p}\gamma + \log_2 p\left(n\alpha + \frac{3n^2}{4\sqrt{p}}\beta\right).$$

The level 3 part contains roughly n/n_b iterations with one `PDGEMM` and one `PDLARFB` per iteration. Within the jth iteration ($j = 1, 2, \ldots, n/n_b$), `PDGEMM` involves matrices of dimension $n \times n_b$ and $n_b \times (n - jn_b - n_b)$; `PDLARFB` mainly performs two parallel GEMM operations, with $\{n_b \times (n-jn_b), (n-jn_b) \times (n-jn_b)\}$ and $\{(n-jn_b) \times n_b, n_b \times (n-jn_b)\}$ matrices involved. Another sequential TRMM operation in `PDLARFB` is neglected since it only contributes lower order terms in both arithmetic and communication costs. So the cost for level 3 part is

$$\begin{aligned} T_{\text{level3}} &= \sum_{j=1}^{n/n_b}\left[\frac{2jn_b + 6(n-jn_b)}{p}n_b(n-jn_b)\gamma + \log_2 p\left(3\alpha + \frac{6n-5jn_b}{2\sqrt{p}}\beta\right)\right] \\ &\approx \frac{7n^3}{3p}\gamma + \frac{3n\log_2 p}{n_b}\alpha + \frac{7n^2\log_2 p}{4\sqrt{p}}\beta, \end{aligned}$$

and hence the execution time for Hessenberg reduction (without explicitly forming the orthogonal matrix) is

$$\widetilde{T}_{\text{Hess}}(n,p) = T_{\text{level2}} + T_{\text{level3}} \approx \frac{10n^3}{3p}\gamma + n\log_2 p\,\alpha + \frac{5n^2\log_2 p}{2\sqrt{p}}\beta. \quad (13)$$

Even if the proportion of level 3 operations is improved to 80% as suggested in [29] but not implemented in the current `PDGEHRD` yet, the estimate in (13) would not change too much since the number of messages in the level 2 part is not reduced.

Since the Householder reflections are stored in a compact form in the lower triangular part of the upper Hessenberg matrix, formulating the orthogonal matrix after Hessenberg reduction is another necessary step. This step is done by the ScaLAPACK routine `PDORMHR`, which is mainly a series of calls to `PDLARFB`. Similar to the discussion above, we are able to obtain

$$T_{\text{ORMHR}} \approx \frac{2n^3}{p}\gamma + \frac{3n\log_2 p}{n_b}\alpha + \frac{7n^2}{4\sqrt{p}}\beta.$$

Therefore the total runtime for the Hessenberg reduction process including formulating the orthogonal matrix is

$$T_{\text{Hess}}(n,p) = \widetilde{T}_{\text{Hess}} + T_{\text{ORMHR}} \approx \frac{16n^3}{3p}\gamma + n\log_2 p\,\alpha + \frac{17n^2\log_2 p}{4\sqrt{p}}\beta. \tag{14}$$

- T_{update}: The cost for updating the off-diagonal blocks with respect to the AED window is simple to analyze since it merely contains three GEMM operations. Since these GEMM operations are not "fat" level 2 operations, we need to use a model different to (12). According to [37], the execution time for a GEMM operation on a $\sqrt{p}\times\sqrt{p}$ processor grid with $m\times k$ and $k\times n$ matrices involved is

$$T_{\text{GEMM}}(m,n,k,p) \approx \frac{2mnk}{p}\gamma + \left(\frac{k}{n_b} + 2\sqrt{p}\right)\left(2\alpha + \frac{(m+n)n_b}{\sqrt{p}}\beta\right) \tag{15}$$

  if $\min\{m,n,k\} = k \gg n_b$. Then we are able to conclude that

$$T_{\text{update}}(n,n_{\text{AED}},p) \approx \frac{2nn_{\text{AED}}^2}{p}\gamma + \frac{n_{\text{AED}}}{n_b}\left(6\alpha + \frac{2nn_b}{\sqrt{p}}\beta\right). \tag{16}$$

Now we are ready to estimate the overall runtime T_{AED} by substituting n with n_{AED} in (8) and (14). We can see that T_{redist} is always negligible compared to other components. Reordering contributes with only marginal communication costs also. By merging all these estimates together, we eventually obtain

$$\begin{aligned}
T_{\text{AED}}(n,n_{\text{AED}},p) &\approx T_{\text{Schur}}(n_{\text{AED}},p_{\text{AED}}) + T_{\text{reorder}}(n_{\text{AED}},p) + T_{\text{Hess}}(n_{\text{AED}},p) + T_{\text{update}}(n,n_{\text{AED}},p) \\
&\approx \left(\frac{80n_{\text{AED}}}{p_{\text{AED}}} + \frac{4\sqrt{p}\,n_b + 16n_{\text{AED}} + 2n}{p}\right)n_{\text{AED}}^2\gamma \\
&\quad + \frac{n_{\text{AED}}^2}{n_b}\left(\frac{3(\log_2 p_{\text{AED}} + 2)}{\sqrt{p_{\text{AED}}}} + \frac{n_b\log_2 p}{n_{\text{AED}}}\right)\alpha \\
&\quad + \frac{n_{\text{AED}}}{n_b}\left(\frac{3n_{\text{AED}}n_b\log_2 p_{\text{AED}}}{\sqrt{p_{\text{AED}}}} + \frac{16n_{\text{AED}}^2}{p_{\text{AED}}} + \frac{3n_b}{2} + \frac{17n_b\log_2 p}{4\sqrt{p}} + \frac{2nn_b}{n_{\text{AED}}\sqrt{p}}\right)\beta
\end{aligned} \tag{17}$$

$$\begin{aligned}
&\le \left[\frac{120C_2^2 n}{C_1} + \frac{9n^2 n_b}{C_1^2\sqrt{p}} + \frac{3(C_1+36)n^3}{2C_1^3 p}\right]\gamma \\
&\quad + \frac{9C_2 n}{C_1 n_b}\left(\log_2\frac{3n}{2C_1C_2} + \frac{\log_2 p\,n_b}{3C_2}\right)\alpha \\
&\quad + \left[\frac{9C_2 n}{C_1}\log_2\frac{3n}{2C_1C_2} + \frac{36C_2^2 n}{C_1 n_b} + \frac{3n^2\left(9\sqrt{p} + 102\log_2 p + 8C_1\right)}{8C_1^2\sqrt{p}}\right]\beta
\end{aligned} \tag{18}$$

When n is extremely large (i.e., C_1, C_2 and n_b are all tiny enough compared to n), eventually we have

$$T_{\text{AED}} = \Theta\left(\frac{n^3}{p}\right)\gamma + \Theta(n\log n)\,\alpha + \Theta(n^2)\beta.$$

Therefore AED is asymptotically cheaper than QR sweeps. But in practice we still need to handle AED very carefully. The large constants in the lower order terms usually have significant impact on the performance if the matrix is not large enough. Similar to the analysis for T_{AED}, the cost for computing shifts can be estimated by

$$\begin{aligned} T_{\text{shift}}(n, n_{\text{shift}}, p) &\approx \widetilde{T}_{\text{Schur}}(n_{\text{shift}}, p_{\text{shift}}) \\ &\approx \frac{40n_{\text{shift}}^3}{p_{\text{shift}}}\gamma + \frac{3n_{\text{shift}}^2}{\sqrt{p_{\text{shift}}}\,n_b}(\log_2 p_{\text{shift}} + 2)\,\alpha \\ &\quad + \Big(\frac{3n_{\text{shift}}^2 \log_2 p_{\text{shift}}}{\sqrt{p_{\text{shift}}}} + \frac{16n_{\text{shift}}^3}{p_{\text{shift}} n_b}\Big)\beta \qquad (19) \\ &\le \frac{40C_2^2 n}{C_1}\gamma + \frac{6C_2 n}{C_1 n_b}\log_2\frac{n}{C_1 C_2}\,\alpha + \Big(\frac{6C_2 n}{C_1}\log_2\frac{n}{C_1 C_2} + \frac{16C_2^2 n}{C_1 n_b}\Big)\beta. \qquad (20) \end{aligned}$$

Actually, T_{shift} is not so important in the scalability analysis since it can never be worse than T_{AED}.

3.3 An overall model

To make the models discussed in previous subsections simpler and more intuitive, we assign the values of the parameters with some concrete numbers. For example, if we set $C_1 = 24$ and $C_2 = 384$, then T_{QR}, T_{AED}, and T_{shift} become

$$\begin{aligned} T_{\text{QR}}(n, n_{\text{shift}}, p) &\approx \frac{3n^3}{2p}\gamma + \frac{3n^2}{8\sqrt{p}\,n_b^2}(\log_2 p + 4)\alpha + \frac{3n^3}{8\sqrt{p}\,n_b}\beta, \\ T_{\text{AED}}(n, n_{\text{AED}}, p) &\approx \Big(737280n + \frac{n^2 n_b}{64\sqrt{p}} + \frac{5n^3}{768p}\Big)\gamma \\ &\quad + \Big[\frac{144n}{n_b}(\log_2 n - 14) + \frac{n\log_2 p}{8}\Big]\alpha \\ &\quad + \Big[144n\Big(\log_2 n - 14 + \frac{1536}{n_b}\Big) + \frac{(9\sqrt{p} + 102\log_2 p + 192)n^2}{3072\sqrt{p}}\Big]\beta, \end{aligned}$$

and

$$T_{\text{shift}}(n, n_{\text{shift}}, p) \approx 245760n\gamma + \frac{96n}{n_b}(\log_2 n - 13)\alpha + 96n\Big(\log_2 n - 13 + \frac{1024}{n_b}\Big)\beta,$$

respectively. In practice k_{AED}, k_{QR}, k_{shift} can vary a lot, but $k_{\text{AED}} = \Theta(n/n_{\text{AED}}) = \Theta(C_1)$ represents a typical convergence rate. For example, if we assume $k_{\text{AED}} = 2k_{\text{QR}} = 4k_{\text{shift}} = 64$, then an overall estimate of the new parallel multishift QR algorithm with AED is

$$\begin{aligned} T(n, p) &\approx \Big(\frac{48n^3}{p} + 5.1\times 10^7 n\Big)\gamma + \frac{12n^2\log_2 p}{\sqrt{p}\,n_b^2}\alpha + \frac{12n^3}{\sqrt{p}\,n_b}\beta, \qquad (21) \\ &= \Theta\Big(\frac{n^3}{p}\Big)\gamma + \Theta\Big(\frac{n^2\log p}{\sqrt{p}\,n_b^2}\Big)\alpha + \Theta\Big(\frac{n^3}{\sqrt{p}\,n_b}\Big)\beta, \qquad (22) \end{aligned}$$

where small order terms are neglected. T_{shift} always contributes very little in the total cost. Both QR sweeps and AED have significant serial runtime when n is not very large. However, QR sweeps dominates in the communication cost. Therefore (22) and (4) have the same asymptotic behavior although the execution time model (21) is usually smaller. AED helps to improve the convergence and never becomes a bottleneck once it is properly implemented. In practice, we sometimes use `PDLAQR0` recursively instead of `PDLAQR1` as the eigensolver in the AED and shift calculations when n is large. But since they have similar asymptotic behavior and the recursive eigensolver only contributes a lower order term (although with large coefficients) in the total cost, the computational cost is asymptotically similar to (22). Another observation from the model is that it is advisable to decrease the parameter `NIBBLE` when n is large, since AED eventually becomes much cheaper than QR sweeps. If we keep the memory load per processor (n^2/p) unchanged and let n grow, the parallel QR algorithm does not scale as well as the Hessenberg reduction algorithm. But an appropriate choice of `NIBBLE` can sometimes delay the differences for matrices which are not too large.

4 Other Implementation Issues

4.1 Calling sequence

The calling sequence of `PDHSEQR` is nearly identical with the LAPACK routine `DHSEQR`'s. Apart from the need of a descriptor for each globally distributed matrix and the leading dimension for each local matrix, the only difference is that `PDHSEQR` requires an extra integer workspace.

```
      SUBROUTINE PDHSEQR( JOB, COMPZ, N, ILO, IHI, H, DESCH, WR, WI, Z,
     $                    DESCZ, WORK, LWORK, IWORK, LIWORK, INFO )
*
*     .. Scalar Arguments ..
      INTEGER            IHI, ILO, INFO, LWORK, LIWORK, N
      CHARACTER          COMPZ, JOB
*     ..
*     .. Array Arguments ..
      INTEGER            DESCH( * ) , DESCZ( * ), IWORK( * )
      DOUBLE PRECISION   H( * ), WI( N ), WORK( * ), WR( N ), Z( * )
```

```
      SUBROUTINE DHSEQR( JOB, COMPZ, N, ILO, IHI, H, LDH, WR, WI, Z,
     $                   LDZ, WORK, LWORK, INFO )
```

```
      SUBROUTINE PDLAHQR( WANTT, WANTZ, N, ILO, IHI, A, DESCA, WR, WI,
     $                    ILOZ, IHIZ, Z, DESCZ, WORK, LWORK, IWORK,
     $                    ILWORK, INFO )
```

The calling sequence of the auxiliary routine `PDLAHQR` is also similar. Therefore, for most existing codes using `PDLAHQR`, it only requires minor changes when switching the eigensolver to `PDHSEQR`. In practice, it is advisable to call `PDHSEQR` twice — one call for the workspace query (by setting `LWORK=-1`) and another call for actually doing the computation. This follows the convention of many LAPACK/ScaLAPACK routines which require workspace. Note that multithreading (OpenMP) is not provided

TABLE 2: A list of tunable parameters

ISPEC	Name	Description	Recommended value
12	$n_{\min}$	Crossover point between PDLAQR0 and PDLAQR1	$220\min(p_r, p_c)$
13	n_{AED}	Size of the AED window	see Table 1
14	NIBBLE	Threshold for skipping a multishift QR sweep	see Section 2.4
15	n_{shift}	Number of simultaneous shifts	see Table 1
16	KACC22	The choice of how to update the off-diagonal blocks in the multishift QR sweep	use GEMM/TRMM
17	NUMWIN	Maximum number of concurrent computational windows (for both QR sweep and eigenvalue reordering)	$\min(p_r, p_c, \lceil n/n_b \rceil)$
18	WINEIG	Number of eigenvalues in each window (for eigenvalue reordering)	$\min(n_b/2, 40)$
19	WINSIZE	Computational window size (for both bulge-chasing and eigenvalue reordering)	$\min(n_b, 80)$
20	MMULT	Minimal percentage of flops for performing GEMM instead of pipelined Householder reflections when updating the off-diagonal blocks in the eigenvalue reordering routine	50
21	NCB	Width of block column slabs for row-wise update of Householder reflections in factorized form	$\min(n_b, 32)$
22	WNEICR	Maximum number of eigenvalues to bring over the block border in the eigenvalue reordering routine	same as WINEIG
23	$p_{\min}$	Size of processor grid involving AED	see Section 2.5

in the current ScaLAPACK release, since it requires further tuning and support from compilers/libraries.

4.2 Tuning parameters

In the new software for the parallel QR algorithm, tunable parameters are defined in the routine PILAENVX. They are available via the function call PILAENVX(ICTXT, ISPEC, ...) with $12 \le$ ISPEC ≤ 23. We have already discussed two of them in Sections 2.4 and 2.5. A complete list of these parameters is provided in Table 2. Some of them need to be fine tuned for different architectures.

Although a reasonable choice of n_b, the data layout block size, is important, the performance turns out not to be overly sensitive to this choice. On the one hand, n_b

should be large enough so that the local computations can achieve level 3 performance. On the other hand, it is advisable to avoid n_b being too large. A large n_b harms the load balance and increases the overhead in the start-up and ending stages of the bulge chasing process, especially when computing the Schur decomposition of the AED window. In our performance model, we always assume $n_b \ll n/\sqrt{p}$ to avoid such kind of overhead. For many architectures, $n_b \in [32, 128]$ will offer a good choice.

On many architectures, most of the recommended values in Table 2 can be used as default values. However, the parameters $n_{\min}$, $p_{\min}$ and `NIBBLE` require extra care since the performance of the AED process heavily relies on them. The values of these parameters need to be determined by performing a bunch of test runs. To determine $n_{\min}$ and $p_{\min}$, it is advisable to use the typical sizes of the AED windows (see Table 1) and run tests on different processor grids. Then the optimal values for both $n_{\min}$ and $p_{\min}$ can be chosen via examining the number of columns of H owned by each processor. `NIBBLE` should be tuned lastly, once all other parameters are fixed. Tuning `NIBBLE` is time-consuming but highly recommended, especially on older architectures with slow communication. As discussed in Section 2.4, $\texttt{NIBBLE} = a \cdot n^b p^c$ is a reasonably good model that takes into account both n and p. It is not unlikely that this model may need to be adjusted for very large-scale computations.

5 Computational Experiments

In this section, we present a subset from a large set of computational experiments that have been performed to confirm and demonstrate the improvements we have made in the parallel QR algorithm. We compare our new parallel QR algorithm in ScaLAPACK v2.0.1 with the previous implementation in [19] as well as the pipelined QR algorithm in ScaLAPACK v1.8.0. For simplicity, these solvers are denoted by S-v201, SISC and S-v180, respectively. The data layout block size $n_b = 50$ is used for all experiments. The experiments were run on the Intel Xeon Linux cluster *Akka* hosted by the High Performance Computing Center North (HPC2N). The code is compiled by the PathScale compiler version 4.0 with the optimization flags `-O3 -fPIC` and linked with the libraries OpenMPI 1.4.4, LAPACK 3.4.0 and GotoBLAS2 1.13. No multithreaded features (such as OpenMP or threaded BLAS) are used. Therefore the number of processors ($p = p_r \times p_c$) means the number of cores involved in the computation.

5.1 Random matrices

First, we consider two types of random matrices — `fullrand` and `hessrand` [19].

Matrices of the type `fullrand` are dense square matrices with all entries randomly generated from a uniform distribution in $[0, 1]$. We call the ScaLAPACK routine `PDGEHRD` to reduce them to upper Hessenberg form before applying the Hessenberg QR algorithm. Only the time for the Hessenberg QR algorithm is measured. These matrices usually have well-conditioned eigenvalues and exhibit "regular" convergence behavior.

Matrices of the type `hessrand` are upper Hessenberg matrices whose nonzero

TABLE 3: Execution time (in seconds) for `fullrand` matrices.

$p =$	$n = 4000$			$n = 8000$			$n = 16000$			$n = 32000$		
$p_r \times p_c$	S-v180	SISC	S-v201	S-v180	SISC	S-v201	S-v180	SISC	S-v201	S-v180	SISC	S-v201
1×1	834	178	114	10730	939	629						
2×2	317	87	66	2780	533	278						
4×4	136	50	39	764	205	173	6671	1220	739			
6×6	112	50	46	576	142	116	3508	754	466	∞	3163	2340
8×8	100	45	38	464	127	157	2536	506	373	∞	2979	1712
10×10	97	50	44	417	159	119	2142	457	299	∞	2401	1326

entries are randomly generated from a uniform distribution in $[0,1]$. The eigenvalues of these matrices are extremely ill-conditioned for larger n, affecting the convergence behavior of the QR sweeps [19]. On the other hand, AED often deflates a high fraction of eigenvalues in the AED window for such matrices. These properties sometimes cause erratic convergence rates.

Tables 3 and 4 show the parallel execution times of the three solvers. Both the real Schur form T and the orthogonal transformation matrix Z are calculated. We limit the total execution time (including the Hessenberg reduction) by 1000 CPU hours for each individual problem, to avoid excessive use of the computational resources. An entry ∞ corresponds to an execution time larger than 1000 CPU hours. These tables reveal that our new solver `PDHSEQR` in ScaLAPACK v2.0.1 almost always improves the performance compared to the SISC version (with only one exception). On average, the improvement is 26% for matrices of type `fullrand` and 13 times for matrices of type `hessrand`. Not surprisingly, the improvements compared to `PDLAHQR` in ScaLAPACK v1.8.0 are even more significant.

The convergence rates for `fullrand` are sufficiently regular, so that we can analyze the scalability of the parallel QR algorithm. If we fix the memory load per core to $n/\sqrt{p} = 4000$, the execution times in Table 3 satisfy

$$2T(n,p) \le T(2n,4p) \le 4T(n,p).$$

This shows that the parallel QR algorithm scales reasonably well, but not perfectly. Most importantly, this is consistent with the theoretical performance model (22). For `hessrand`, it is observed that most eigenvalues are deflated with very few (or even no) QR sweeps. Considering that the main difference of `PDHSEQR` between versions S-v201 and SISC is in the AED process, it is not surprising to see the great convergence acceleration for `hessrand`, where AED dominates the calculation. In Table 4, sometimes the execution time for S-v201 does not change too much when increasing the number of processors. This is mainly because the Schur decomposition of the AED window, which is the most expensive part of the algorithm, is performed by a constant number of processors ($p_{\min} \cdot p_{\min} \le p$) after data redistribution.

5.2 $100,000\times100,000$ matrices

The modifications proposed in this paper result into dramatic improvements for settings with very large matrices and many processors. To demonstrate this, we present the obtained execution times for $100,000\times100,000$ matrices in Table 5. Although the Hessenberg QR algorithm does not scale as well as Hessenberg reduction when fixing

TABLE 4: Execution time (in seconds) for `hessrand` matrices.

$p =$	$n = 4000$			$n = 8000$			$n = 16000$			$n = 32000$		
$p_r \times p_c$	S-v180	SISC	S-v201	S-v180	SISC	S-v201	S-v180	SISC	S-v201	S-v180	SISC	S-v201
1×1	685	317	12	6981	2050	70						
2×2	322	200	9	2464	1904	28						
4×4	163	112	28	1066	679	146	8653	2439	69			
6×6	137	84	28	768	412	83	4475	1254	72	∞	373	248
8×8	121	68	24	634	321	90	3613	719	73	∞	919	239
10×10	131	83	26	559	313	99	3549	667	76	∞	943	243

TABLE 5: Execution time (in seconds) of S-v201 for $100,000\times100,000$ matrices.

	$p = 16\times16$		$p = 24\times24$		$p = 32\times32$		$p = 40\times40$	
	`fullrand`	`hessrand`	`fullrand`	`hessrand`	`fullrand`	`hessrand`	`fullrand`	`hessrand`
Balancing	854	–	849	–	855	–	898	–
Hess. reduction	10517	–	6408	–	3877	–	2751	–
QR algorithm	9787	3601	7046	2809	6328	2336	6043	2119
k_{AED}	35	19	30	19	25	18	29	18
k_{QR}	4	0	6	0	10	0	11	0
#(shifts)/n	0.16	0	0.22	0	0.30	0	0.45	0
AED% in the QR alg.	44%	100%	44%	100%	42%	100%	44%	100%

the problem size and increasing the number of processors, the execution times of these two reduction steps are still on the same order of magnitude. With the help of the dynamic `NIBBLE` strategy, the fraction of the execution time spent on AED for `fullrand` matrices is under control. In contrast to our earlier implementation, AED is no longer a bottleneck of the whole QR algorithm. As reported in [19], it took 7 hours for our earlier implementation `PDHSEQR` to solve the $100,000\times100,000$ `fullrand` problem with 32×32 processors; 80% execution time of the QR algorithm was spent on AED. Our new version of `PDHSEQR` is able to solve the same problem in roughly 1.75 hours, which is about four times faster. The time fraction spent on AED is reduced to 42%.

5.3 Benchmark examples

Besides random matrices, we also report performance results for some commonly used benchmark matrices. For comparison, we have tested the same matrices as in [19], see Table 6. The execution times for the three solvers are listed in Tables 7–14. The conclusions are similar to those we have made for random matrices: our earlier version of `PDHSEQR` outperforms the ScaLAPACK 1.8.0 routine `PDLAHQR` by a large extent; the new `PDHSEQR` is usually even faster, especially for BBMSN and GRCAR.

In [19], it was observed that the accuracy for AF23560 is not fully satisfactory; the relative residuals $R_r = \|Q^T A Q - T\|_F / \|A\|_F$ were large for both `PDLAHQR` and `PDHSEQR`. It turns out that these large residuals are caused by an anomaly in `PDLAHQR`, which has been fixed by avoiding the use of `PDLACONSB`, see Section 2.3.2 As a result, the new `PDHSEQR` always produce $R_r \in [10^{-15}, 10^{-13}]$ for all test matrices.

6 Conclusions and Future Work

We have presented a new parallel implementation of the multishift QR algorithm with aggressive early deflation. The new routine `PDHSEQR` combines a number of techniques to improve node performance and reduce communication. Our numerical ex-

Table 6: Benchmark matrices.

ID	Name	Dimension (n)	Type/Structure
1	BBMSN [12]	n	$S_n = \begin{bmatrix} n & n-1 & n-2 & \cdots & 2 & 1 \\ 10^{-3} & 1 & 0 & & 0 & 0 \\ & 10^{-3} & 2 & & 0 & 0 \\ & & 10^{-3} & & 0 & 0 \\ & & & \ddots & n-2 & 0 \\ & & & & 10^{-3} & n-1 \end{bmatrix}$
2	AF23560 [4]	23560	Computational fluid dynamics
3	CRYG10000 [4]	10000	Material science
4	OLM5000 [4]	5000	Computational fluid dynamics
5	DW8192 [4]	8192	Electrical engineering
6	MATRAN [4]	n	Sparse random matrix
7	MATPDE [4]	n	Partial differential equations
8	GRCAR [4]	n	$G_n = \begin{bmatrix} 1 & 1 & 1 & 1 & & & & \\ -1 & 1 & 1 & 1 & 1 & & & \\ & -1 & 1 & 1 & 1 & 1 & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & -1 & 1 & 1 & 1 & 1 \\ & & & & -1 & 1 & 1 & 1 \\ & & & & & -1 & 1 & 1 \\ & & & & & & -1 & 1 \end{bmatrix}$

Table 7: Execution time (in seconds) for BBMSN.

n	$p = p_r \times p_c$	S-v180	SISC	S-v201
5000	1×1	523	6	3
10000	2×2	1401	30	9
15000	4×4	1489	62	14

Table 8: Execution time (in seconds) for AF23560.

$p = p_r \times p_c$	S-v180	SISC	S-v201
4×4	15486	2651	1438
6×6	9088	1279	823
8×8	6808	793	582
10×10	5694	662	467
12×12	5422	578	388

Table 9: Execution time (in seconds) for CRYG10000.

$p = p_r \times p_c$	S-v180	SISC	S-v201
1×1	6394	1331	1259
2×2	2123	580	497
4×4	979	236	206
6×6	731	161	130
8×8	545	128	96
10×10	496	144	89

Table 10: Execution time (in seconds) for OLM5000.

$p = p_r \times p_c$	S-v180	SISC	S-v201
1×1	426	206	190
2×2	167	98	109
4×4	76	54	49
6×6	58	52	43
8×8	46	49	42
10×10	48	51	43

Table 11: Execution time (in seconds) for DW8192.

$p = p_r \times p_c$	S-v180	SISC	S-v201
1×1	10307	1329	1309
2×2	1187	572	521
4×4	635	225	216
6×6	357	152	141
8×8	302	129	105
10×10	275	121	99

Table 12: Execution time (in seconds) for MATRAN.

$p =$	$n = 10000$			$n = 14400$			$n = 19600$		
$p_r \times p_c$	S-v180	SISC	S-v201	S-v180	SISC	S-v201	S-v180	SISC	S-v201
1×1	12429	2600	2475						
2×2	3531	1081	955						
4×4	1565	415	381	4446	1207	1076	9915	2844	2664
6×6	1118	256	227	3069	654	579	6130	1426	1301
8×8	871	189	163	2259	449	386	4615	912	807
10×10	789	189	142	1955	431	328	4046	743	662
12×12	719	194	129	1736	367	261	3483	648	502

Table 13: Execution time (in seconds) for MATPDE.

$p =$	$n = 10000$			$n = 14400$			$n = 19600$		
$p_r \times p_c$	S-v180	SISC	S-v201	S-v180	SISC	S-v201	S-v180	SISC	S-v201
1×1	12429	2600	2475						
2×2	3531	1081	955						
4×4	1565	415	381	4446	1207	1076	9915	2844	2664
6×6	1118	256	227	3069	654	579	6130	1426	1301
8×8	871	189	163	2259	449	386	4615	912	807
10×10	789	189	142	1955	431	328	4046	743	662
12×12	719	194	129	1736	367	261	3483	648	502

Table 14: Execution time (in seconds) for GRCAR.

$p =$	$n = 6000$			$n = 12000$			$n = 18000$		
$p_r \times p_c$	S-v180	SISC	S-v201	S-v180	SISC	S-v201	S-v180	SISC	S-v201
1×1	2738	1340	69						
2×2	850	645	40	8199	2734	135			
4×4	363	258	233	2499	1336	118	8171	4037	189
6×6	244	190	172	1471	849	119	4385	2172	184
8×8	217	150	132	1107	515	128	3342	1345	180
10×10	207	161	131	923	538	322	2675	1104	215

periments provide compelling evidence that PDHSEQR significantly outperforms not only the original ScaLAPACK routine PDLAHQR but also an earlier version of PDHSEQR presented in [19]. In particular, our new implementation removes a bottleneck in the aggressive early deflation strategy by reducing communication and tuning algorithmic parameters. As a result, our new version is both faster and more robust. The software is available in ScaLAPACK version 2.0.

Concerning future work, we believe to have come to a point, where it will be difficult to attain further dramatic performance improvements for parallel nonsymmetric eigensolvers, without leaving the classical framework of QR algorithms. Considering the fact that the execution times spent on Hessenberg reduction and on QR iterations are now nearly on the same level, any further improvement of the iterative part will only have a limited impact on the total execution time. The situation is quite different when shared memory many-core processors with accelerators, such as GPUs, get involved. Although efficient implementations of the Hessenberg reduction on such architectures have recently been proposed [26, 24, 33], the iterative part remains to be done. Another future challenge is to combine the message passing paradigm used in the new ScaLAPACK v2.0 software and dynamic and static scheduling on many-core nodes using multithreading.

Acknowledgements

We are grateful to Björn Adlerborn and Lars Karlsson for constructive discussions and comments on the subject, and to Åke Sandgren for support at HPC2N. We also thank Rodney James, Julien Langou as well as anonymous users from IBM for helpful feedback.

References

[1] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. *Journal of Physics: Conference Series*, 180(1):012037, 2009.

[2] M. Ahues and F. Tisseur. A new deflation criterion for the QR algorithm. LAPACK Working Note 122, 1997.

[3] T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Kraemer, B. Lang, H. Lederer, and P. R. Willems. Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations. *Parallel Comput.*, 37(12):783–794, 2011.

[4] Z. Bai, D. Day, J. Demmel, and J. Dongarra. A test matrix collection for non-Hermitian eigenvalue problems (release 1.0). Technical Report CS-97-355, Department of Computer Science, University of Tennessee, 1997. Also available online from http://math.nist.gov/MatrixMarket.

[5] Z. Bai and J. W. Demmel. On a block implementation of Hessenberg multishift QR iteration. *Intl. J. High Speed Comput.*, 1:97–112, 1989.

[6] Z. Bai and J. W. Demmel. On swapping diagonal blocks in real Schur form. *Linear Algebra Appl.*, 186:73–95, 1993.

[7] G. Ballard, J. Demmel, and I. Dumitriu. Minimizing communication for eigenproblems and the singular value decomposition. Technical Report UCB/EECS-2010-136, EECS Department, University of California, Berkeley, 2010. Also as LAPACK Working Note 237.

[8] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM J. Matrix Anal. Appl.*, 32(3):866–901, 2011.

[9] P. Bientinesi, E. S. Quintana-Ortí, and R. A. van de Geijn. Representing linear algebra algorithms in code: The FLAME application program interfaces. *ACM Trans. Math. Software*, 31(1):27–59, 2005.

[10] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. W. Demmel, I. Dhillon, J. J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.

[11] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. Part I: Maintaining well-focused shifts and level 3 performance. *SIAM J. Matrix Anal. Appl.*, 23(4):929–947, 2002.

[12] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. Part II: Aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 23(4):948–973, 2002.

[13] R. Byers. LAPACK 3.1 xHSEQR: Tuning and implementation notes on the small bulge multi-shift QR algorithm with aggressive early deflation, 2007. LAPACK Working Note 187.

[14] J. Choi, J. J. Dongarra, and D. W. Walker. The design of a parallel dense linear algebra software library: Reduction to Hessenberg, tridiagonal, and bidiagonal form. *Numer. Algorithms*, 10(2):379–399, 1995.

[15] M. R. Fahey. Algorithm 826: A parallel eigenvalue routine for complex Hessenberg matrices. *ACM Trans. Math. Software*, 29(3):326–336, 2003.

[16] J. G. F. Francis. The QR transformation: A unitary analogue to the LR transformation — Part 2. *Comput. J.*, 4(4):332–345, 1962.

[17] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, USA, third edition, 1996.

[18] R. Granat, B. Kågström, and D. Kressner. Parallel eigenvalue reordering in real Schur forms. *Concurrency and Computat.: Pract. Exper.*, 21(9):1225–1250, 2009.

[19] R. Granat, B. Kågström, and D. Kressner. A novel parallel QR algorithm for hybrid distributed memory HPC systems. *SIAM J. Sci. Comput.*, 32(4):2345–2378, 2010.

[20] G. Henry and R. Van de Geijn. Parallelizing the QR algorithm for the unsymmetric algebraic eigenvalue problem: Myths and reality. *SIAM J. Sci. Comput.*, 17:870–883, 1997.

[21] G. Henry, D. S. Watkins, and J. J. Dongarra. A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. *SIAM J. Sci. Comput.*, 24(1):284–311, 2002.

[22] M. Hoemmen. *Communication-Avoiding Krylov Subspace Methods*. PhD thesis, University of California, Berkeley, 2010.

[23] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distr. Comput.*, 64(9):1017–1026, 2004.

[24] B. Kågström, D. Kressner, E. S. Quintana-Ortí, and G. Quintana-Ortí. Blocked algorithms for the reduction to Hessenberg-triangular form revisited. *BIT*, 48(3):563–584, 2008.

[25] B. Kågström, D. Kressner, and M. Shao. On aggressive early deflation in parallel variants of the QR algorithm. In K. Jónasson, editor, *Applied Parallel and Scientific Computing (PARA 2010)*, volume 7133 of *Lecture Notes in Comput. Sci.*, pages 1–10, Berlin, 2012. Springer-Verlag.

[26] L. Karlsson and B. Kågström. Parallel two-stage reduction to Hessenberg form using dynamic scheduling on shared-memory architectures. *Parallel Comput.*, 37(12):771–782, 2011.

[27] R. S. Martin, G. Peters, and J. H. Wilkinson. Handbook Series Linear Algebra: The QR algorithm for real Hessenberg matrices. *Numer. Math.*, 14(3):219–231, 1970.

[28] L. Prylli and B. Tourancheau. Fast runtime block cyclic data redistribution on multiprocessors. *J. Parallel Distr. Comput.*, 45(1):63–72, 1997.

[29] G. Quintana-Ortí and R. van de Geijn. Improving the performance of reduction to Hessenberg form. *ACM Trans. Math. Software*, 32(2):180–194, 2006.

[30] T. Schreiber, P. Otto, and F. Hofmann. A new efficient parallelization strategy for the QR algorithm. *Parallel Comput.*, 20(1):63–75, 1994.

[31] M. Shao. `PDLAQR1`: An improved version of the ScaLAPACK routine `PDLAHQR`. Technical Report UMINF-11.22, Department of Computing Science, Umeå University, 2011.

[32] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler. *Matrix Eigensystem Routines — EISPACK Guide.*, volume 6 of *Lecture Notes in Comput. Sci.* Springer-Verlag, New York, second edition, 1976.

[33] S. Tomov, R. Nath, and J. Dongarra. Accelerating the reduction to upper Hessenberg, tridiagonal, and bidiagonal forms through hybrid GPU-based computing. *Parallel Comput.*, 36(12):645–654, 2010.

[34] R. A. van de Geijn. Storage schemes for parallel eigenvalue algorithms. In G. Golub and P. Van Dooren, editors, *Numerical Linear Algebra Digital Signal Processing and Parallel Algorithms*, pages 639–648. Springer-Verlag, 1988.

[35] R. A. van de Geijn. Deferred shifting schemes for parallel QR methods. *SIAM J. Matrix Anal. Appl.*, 14:180–194, 1993.

[36] R. A. van de Geijn and D. G. Hudson. An efficient parallel implementation of the nonsymmetric QR algorithm. In *Fourth Conference on Hypercube Concurrent Computers and Applications, Monterey, CA*, pages 697–700, 1989.

[37] R. A. van de Geijn and J. Watts. SUMMA: Scalable universal matrix multiplication algorithm. *Concurrency and Computat.: Pract. Exper.*, 9(4):255–274, 1997. Also as LAPACK Working Note 96.

[38] D. S. Watkins. Shifting strategies for the parallel QR algorithm. *SIAM J. Sci. Comput.*, 15(4):953–958, 1994.

III

Paper III

Componentwise High Relative Accuracy Algorithms for the Exponential of an Essentially Nonnegative Matrix*

Meiyue Shao[1], Weiguo Gao[2], and Jungong Xue[2]

[1] *Department of Computing Science and HPC2N*
Umeå University, SE-901 87 Umeå, Sweden
myshao@cs.umu.se

[2] *School of Mathematical Sciences and Laboratory of Mathematics for Nonlinear Science*
Fudan University, Shanghai 200433, China
{wggao, xuej}@fudan.edu.cn

Abstract: New algorithms with componentwise high relative accuracy for computing the exponentials of essentially nonnegative matrices are proposed. Our approach is to derive lower and upper bounds of the matrix exponential, and combining the scaling and squaring method with aggressively truncated Taylor expansion. More precisely, we establish new à priori error estimates and use them to propose an efficient strategy to balance the scale factor and the order of expansion. The scale factor is carefully chosen so that the algorithm is almost optimally scaled in practice. Rounding error analyses and numerical experiments demonstrate the efficiency and accuracy of the proposed algorithms.

1 Introduction

The matrix exponential is one of the most well-studied matrix functions. It has many applications in physics, biology, finance and engineering, especially related to the

* Report UMINF-12.04. Submitted to *Numerische Mathematik*.
The work of M. Shao is supported by the Swedish Research Council under grant A0581501, UMIT Research Lab via an EU Mål 2 project, and eSSENCE, a strategic collaborative e-Science programme funded by the Swedish Research Council.
The work of W. Gao is supported by the National Natural Science Foundation of China under grant 11071047, the Science and Technology Commission of Shanghai Municipality under grant 09ZR1401900 and MOE Key Laboratory of Computational Physical Sciences.
The work of J. Xue is partly supported by the National Natural Science Foundation of China under grant 10970136.

solution of dynamical systems. Several methods for computing the matrix exponential have been proposed, see [10, 15]. As a result, MATLAB's `expm` function, an excellent general purpose solver, is widely used. However, the development of high relative accuracy algorithms for particular structured matrices remain vitally important.

Among the existing algorithms, the scaling and squaring method is considered as one of the most promising candidates for calculating the matrix exponential [15]. Two main algorithmic issues in this method are 1) to choose an acceptable approximation of e^x; 2) to use the scaling and squaring technique to improve the accuracy. Mathematically, the power of a matrix is naturally relevant to its exponential. But numerically, we should avoid too many this repeated matrix multiplications. To avoid this *overscaling* problem [14] in the scaling and squaring method, most existing algorithms try to keep the scale factor as small as possible. Therefore, a lot of efforts are contributed to address the first issue. A good approximation with less computational complexity is then expected. Different types of approximations, including Taylor expansion, Padé approximation, or even more general functions have been tried to approximate the exponential function e^x.

The exponentials of essentially nonnegative matrices (real matrices with nonnegative off-diagonal entries, also known as Metzler matrices) have important applications in continuous-time Markov processes and positive linear dynamical systems. Recently, the exponential of the adjacency matrix (which is symmetric and nonnegative) is shown to be involved in the measurement analysis of networks [7]. Many theoretical results as well as algorithms taking into account the nonnegativity have been proposed [2, 4, 6, 8, 25, 26]. Our motivation is to take a new look into the overscaling issue for general essentially nonnegative matrices. By carefully reviewing the scaling and squaring process, we show that significant improvements can be attained in this case.

Based on our analysis of the scaling and squaring algorithm, we propose new efficient algorithms for essentially nonnegative matrices. The rest of this paper is organized as follows. We first show in Section 2 that the matrix exponentials of nonnegative matrices are always well-conditioned in a certain sense, and so for a wide range of essentially nonnegative matrices. In Sections 3.1 and 3.2, we derive the lower and upper bounds of the matrix exponentials of essentially nonnegative matrices, respectively. We prove the componentwise estimates of the bounds, which are based on scaling/squaring and (aggressively) truncated Taylor series. High relative accuracy algorithms for computing these bounds are also provided. In Section 3.3, an interval algorithm based on the lower and upper bounds with improved accuracy is proposed. Some special algorithmic issues regarding the essentially nonnegative matrices are discussed in Section 3.4. A rounding error analysis of the algorithms is presented in Section 4. Finally, in Section 5 we present and discuss numerical experiments.

2 Preliminaries

First we fix some notation to be used throughout the paper. Hereafter we use $\mathbb{R}_+ = \{x \in \mathbb{R} \colon x \geq 0\}$ to denote the set of all nonnegative real numbers. For any matrices A,

$B \in \mathbb{R}^{M\times N}$, we denote by $A \geq B$ if $A(i,j) \geq B(i,j)$ for all i and j. $|A|$ is the matrix with entries $|A(i,j)|$, and we let struct(A) denote the nonzero pattern of any given matrix A, i.e.,

$$\text{struct}(A) = \{(i,j)\colon A(i,j) \neq 0\}.$$

The notation

$$s(A) = \min_i A(i,i), \quad \widehat{A} = A - s(A)I, \quad \text{and} \quad \hat{\rho}(A) = \rho(\widehat{A}),$$

where $\rho(\cdot)$ is the spectral radius, are used for any $A \in \mathbb{R}^{N\times N}$ unless otherwise stated. We say that A is *essentially nonnegative* if and only if $\widehat{A} \geq 0$. We use $\text{fl}(x)$ to represent the quantity x calculated with floating point arithmetic, where x can be either a number or a matrix. Sometimes we are able to obtain true lower bounds and upper bounds of x by switching the rounding mode [16]. The corresponding bounds are denoted by $\underline{\text{fl}}(x)$ and $\overline{\text{fl}}(x)$, respectively. Then these bounds (so long as they exist) satisfy

$$\underline{\text{fl}}(x) \leq x \leq \overline{\text{fl}}(x). \tag{2.1}$$

We need to clarify that the notation we use here does not imply which rounding mode is used for attaining the result; it only indicates the property (2.1). For example, $\underline{\text{fl}}(1-x)$ can be obtained via $-\text{fl}(\text{fl}(x)-1)$ under the rounding mode to *round towards* $+\infty$.

For a given essentially nonnegative matrix A, we would like to compute every entry of e^A accurately. This is a more challenging task than only producing normwise accuracy. Normwise perturbation bounds can, for example, be found in [12, 20]. MATLAB's `expm` function, a general purpose solver, guarantees normwise backward stability in the Padé method [11]. In [4, 8], a forward normwise error analysis for the Taylor series method is provided; but the à priori truncation error estimate can be far from being useful in practice. Moreover, normwise error estimates do not deliver any information for the accuracy of small entries. To obtain such information requires additional analysis. The following theorem gives the relative perturbation of the matrix exponential with respect to the perturbation of the components of the matrix. The perturbation in the solution is also provided in the componentwise form.

Theorem 2.1. [25] *Assume A is an $N\times N$ essentially nonnegative matrix. Let $\Delta A \in \mathbb{R}^{N\times N}$ be a perturbation to A satisfying $|\Delta A(i,j)| \leq \varepsilon_1 A(i,j)$ $(i \neq j)$ and $|\Delta A(i,i)| \leq \varepsilon_2$, where $0 \leq \varepsilon_1 < 1$, $\varepsilon_2 \geq 0$. Then for any $t \geq 0$ we have*

$$\left|\mathrm{e}^{t(A+\Delta A)} - \mathrm{e}^{tA}\right| \leq \delta(t)\,\mathrm{e}^{\delta(t)}\mathrm{e}^{tA},$$

where

$$\delta(t) = t\varepsilon_2 + \left[N - 1 + \hat{\rho}(A)t\right]\frac{\varepsilon_1}{1-\varepsilon_1}.$$

Here we are interested in the case $t = 1$. We denote

$$C(A) = N - 1 + \hat{\rho}(A) \tag{2.2}$$

as the *condition number* of e^A, in the sense of relative componentwise perturbation and $\widehat{A} \geq 0$. A simple bound of $\hat{\rho}(A)$ might be helpful in understanding the conditioning of the problem.

Let $R_{\max}$ and $R_{\min}$ be the largest and smallest normalized positive finite floating point numbers, respectively, on a certain architecture. If there is no overflow in e^A, then

$$R_{\max} \geq \left\|e^A\right\|_{\max} \geq \frac{1}{N}\left\|e^A\right\|_\infty = \frac{e^{s(A)}}{N}\left\|e^{\widehat{A}}\right\|_\infty \geq \frac{e^{s(A)}}{N}\rho\left(e^{\widehat{A}}\right) = \frac{e^{s(A)}}{N}e^{\hat{\rho}(A)},$$

which implies $s(A)+\hat{\rho}(A) \leq \ln N + \ln R_{\max}$.

***Remark* 1.** For nonnegative matrices, $\hat{\rho}(A)$ cannot be too large. For example, if $A \in \mathbb{R}_+^{10000\times 10000}$ and e^A can be represented by IEEE-754 double precision floating point numbers, then $\hat{\rho}(A) \leq \rho(A) \leq \ln 10000 + \ln R_{\max} < 719$.

Therefore the matrix exponential problem is always well-conditioned for nonnegative matrices, under the assumption that e^A does not cause overflow. For essentially nonnegative matrices, the problem can become more difficult. If we further require that $e^{s(A)}$ does not cause underflow (i.e., $e^{s(A)} > R_{\min}$), the problem is still reasonably well-conditioned. Some difficult problems, e.g., when e^A overflows or $|s(A)|$ is large, are beyond the scope of this paper.

3 Algorithms

3.1 Lower bound algorithm

We know that the Maclaurin series expansion of e^x up to the m-th order is

$$T_m(x) = \sum_{k=0}^{m}\frac{x^k}{k!} = 1 + x + \cdots + \frac{x^{m-1}}{(m-1)!} + \frac{x^m}{m!}. \tag{3.1}$$

Let $R_m(x) = e^x - T_m(x)$ be the remainder term. For given positive integers m, n and an essentially nonnegative matrix A, we define

$$L_{m,n}(A) = e^{s(A)}T_m\left(\frac{\widehat{A}}{n}\right)^n. \tag{3.2}$$

Some traditional Taylor series methods (e.g., see [6, 15]) use $L_{m,n}(A)$ as an approximation of e^A, with the truncation order m determined by an à posteriori estimate

$$\frac{(\widehat{A}/n)^m}{m!}\left[I - \frac{\widehat{A}}{n(m+1)}\right]^{-1} \leq \tau \cdot T_m\left(\frac{\widehat{A}}{n}\right)^n,$$

where τ is the desired accuracy. Some other popular methods use Padé approximation instead of truncated Taylor series after the scaling phase. The scaling and squaring phase can accelerate the convergence and hence reduce the computational cost. Most

of these methods usually treat the scaling and squaring phase as a separate preprocessing step and focus on the Taylor/Padé approximation. To keep the rounding error small, the scale factor n is chosen as small as possible once it can guarantee the convergence (e.g., $\rho(A) < n$ or $\|A\| < n$). Since $\mathrm{struct}\big(L_{m,n}(A)\big) = \mathrm{struct}\big(\mathrm{e}^A\big)$ is a necessary condition for attaining componentwise accuracy, $O(N)$ matrix multiplications are necessary in some cases (e.g., when A is narrow banded), even if the desired accuracy is as modest as 10^{-1}. We try to analyze (3.2) from another point of view and derive a more efficient approach.

Recall that e^{tA} is the solution of the ordinary differential equation

$$\frac{\mathrm{d}X(t)}{\mathrm{d}t} = AX(t), \qquad X(0) = I. \tag{3.3}$$

If we apply Euler's method with step size $h = 1/n$ to (3.3), the approximated solution at $t = 1$ is $X(1) \approx (I + A/n)^n = L_{1,n}(A)$. Theoretically, the approximated solution tends to e^A by reducing the step size, i.e., $\lim_{n\to\infty}(I + A/n)^n = \mathrm{e}^A$. If we rewrite $X(1)$ as $\mathrm{e}^A = \big(\mathrm{e}^{A/n}\big)^n \approx (I + A/n)^n$, it implies that once n is large enough, the first order approximation of $\mathrm{e}^{A/n}$ can be sufficient even though it might not have any relative accuracy. Although Euler's method always suffers from rounding errors when the step size is small, it still suggests that the accuracy requirement of the approximation to $\mathrm{e}^{A/n}$ becomes lower by increasing the scale factor n. This is a subtle but important feature of the scaling and squaring method. Not surprisingly, we can expect higher order truncation in (3.1) provides better approximations to e^A. Likely with a less aggressive truncation order m, the overscaling problem in Euler's method can be avoided. We will see later that once m is appropriately chosen, this aggressively truncated Taylor series method is more efficient than the traditional Taylor series method. To gain more insight, we first prove the following lemma.

Lemma 3.1. *If $A \in \mathbb{R}^{N\times N}$ is essentially nonnegative, then for any positive integers m and n we have*

$$\lim_{n\to\infty} L_{m,n}(A) = \lim_{m\to\infty} L_{m,n}(A) = \mathrm{e}^A. \tag{3.4}$$

Furthermore, the truncation error is bounded by

$$\mathrm{e}^{s(A)/n}\frac{\widehat{A}^{m+1}}{n^m(m+1)!}L_{m,n}(A)^{\frac{n-1}{n}} \le \mathrm{e}^A - L_{m,n}(A) \le \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\mathrm{e}^A. \tag{3.5}$$

Proof. Notice that (3.4) is an immediate consequence of (3.5). Therefore, it is enough to verify (3.5) only. Firstly,

$$\mathrm{e}^A - L_{m,n}(A) = \mathrm{e}^{s(A)}\left[\mathrm{e}^{\widehat{A}} - T_m\Big(\frac{\widehat{A}}{n}\Big)^n\right] = \mathrm{e}^{s(A)}R_m\Big(\frac{\widehat{A}}{n}\Big)\sum_{k=0}^{n-1}\mathrm{e}^{\frac{k}{n}\widehat{A}}T_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1-k}.$$

Since

$$\frac{(\widehat{A}/n)^{m+1}}{(m+1)!} \le R_m\Big(\frac{\widehat{A}}{n}\Big) \le \frac{(\widehat{A}/n)^{m+1}}{(m+1)!}\mathrm{e}^{\widehat{A}/n}$$

and

$$T_m\Big(\frac{\widehat{A}}{n}\Big) \le \mathrm{e}^{\widehat{A}/n},$$

we obtain

$$\frac{\widehat{A}^{m+1}}{n^m(m+1)!}L_{m,n}(\widehat{A})^{\frac{n-1}{n}} \le \mathrm{e}^{\widehat{A}} - L_{m,n}(\widehat{A}) \le \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\mathrm{e}^{\widehat{A}},$$

which is equivalent to (3.5). □

Lemma 3.1 confirms the convergence of $L_{m,n}(A)$ and provides an estimate of the convergence rate. In most existing methods, a fixed n is chosen and m is usually large. On the other hand, if we fix m and choose a large n, it leads to the aggressively truncated Taylor series method. But so far the truncation error of each entry is still unclear. To ensure the componentwise accuracy of the truncated Taylor approximation, we need another lemma.

Lemma 3.2. [25] *Let $A \in \mathbb{R}_+^{N\times N}$. Then*

$$A\mathrm{e}^A \le [N-1+\rho(A)]\mathrm{e}^A.$$

The coefficient in the right hand side is the condition number $C(A)$ (2.2) of the matrix exponential. Actually Theorem 2.1 is derived from this lemma [25]. A direct corollary of Lemma 3.2 is that $f(A)\mathrm{e}^A \le f[C(A)]\mathrm{e}^A$ if $f(x) = \sum_{k=0}^{\infty} a_k x^k$ has nonnegative coefficients (i.e., $a_k \ge 0$ for $k = 0, 1, \ldots$) and $C(A)$ is less than the radius of convergence of $f(x)$. By combining Lemma 3.1 and Lemma 3.2 together, we obtain the componentwise truncation error estimate as follows.

Theorem 3.3. *Let $A \in \mathbb{R}^{N\times N}$ be an essentially nonnegative matrix. Then for any positive integers m and n, we have*

$$0 \le \mathrm{e}^A - L_{m,n}(A) \le \frac{C(A)^{m+1}}{n^m(m+1)!}\mathrm{e}^A. \tag{3.6}$$

Another interesting fact is that $L_{m,n}(A)$ has some monotonic properties. It is not difficult to verify that

$$L_{m,n}(A) \le L_{m+1,n}(A) \quad \text{and} \quad L_{m,n}(A) \le L_{m,2n}(A). \tag{3.7}$$

These inequalities naturally imply that the accuracy can be improved by increasing either m or n. The error bound provided in Theorem 3.3 has the same asymptotic monotonicity. We also conjecture that

$$L_{m,n}(A) \le L_{m,n+1}(A), \tag{3.8}$$

but it is currently not clear how to prove this inequality.

Theorem 3.3 offers an à priori componentwise error estimate. Although the error bound based on Lemma 3.2 is often an overestimate especially when n is small, we

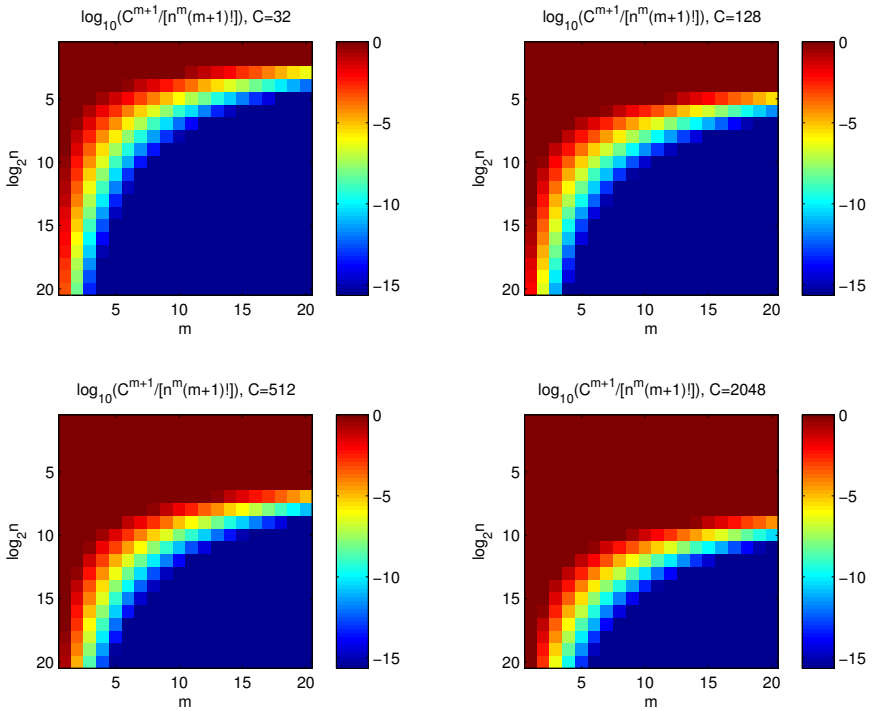

FIGURE 1: Sample values of $C(A)^{m+1}/[n^m(m+1)!]$.
The color is a one to one correspondence with $\log_{10}\left[C(A)^{m+1}/[n^m(m+1)!]\right]$.
Cool(blue) values correspond to small truncation errors, whereas warm(red) values correspond to large errors.

are still able to find out some clues about how to choose m and n. Figure 1 shows some sample values of the constant in (3.6), where $C(A) = 32$, 128, 512, and 2048, respectively. The computational cost for evaluating $L_{m,n}(A)$ as defined in (3.2) is roughly $m+\log_2 n$ matrix multiplications. If we would like to approximately minimize the computational cost, it is sensible to choose the tangent line with slope equals one and pick the point of tangency on the contour line.

***Remark* 2.** From the plots in Figure 1, the optimal value (for the worst case) of both m and $\log_2 n$ are of the magnitude 10^1 for a wide range of matrices. Preferable values on m and $\log_2 n$ are close to the boundary of the blue area.

Some similar suggestions for choosing m and n have been proposed in [15] based on normwise accuracy. Here we conclude that both m and $\log_2 n$ are still small even if componentwise accuracy is desired. For example, if $m \approx 10$ is chosen, $n \approx 2^{15}$ is usually sufficient for attaining double precision accuracy which indicates that overscaling never occurs in practice. Moreover, both Theorem 3.3 and the rounding error analysis

for repeated squaring are pessimistic [15], the true error can be even smaller.

Let τ denote the desired accuracy provided by the user. In practice, if a good estimate of $\hat{\rho}(A)$ is available (e.g., via the power method), we can apply the à priori estimate

$$\frac{C(A)^{m+1}}{n^m(m+1)!} \leq \tau \tag{3.9}$$

to choose m and n. The appropriate values can be easily found by simply enumerating over m and $\log_2 n$. Otherwise, the à posteriori error estimate

$$\frac{\widehat{A}^{m+1}}{n^m(m+1)!} T_m\Big(\frac{\widehat{A}}{n}\Big)^n \leq \frac{\tau}{1+2\tau} T_m\Big(\frac{\widehat{A}}{n}\Big)^n \tag{3.10}$$

can be used as the stopping criterion. With the extra assumption that

$$\frac{C(A)^{m+1}}{n^m(m+1)!} \leq \frac{1}{2},$$

which is reasonably easy to fulfill (since by definition, $C(A) \leq N - 1 + \ln N + \ln R_{\max} - s(A)$), we obtain

$$\frac{1}{2}\mathrm{e}^{\widehat{A}} \leq T_m\Big(\frac{\widehat{A}}{n}\Big)^n.$$

Then from (3.10) and Lemma 3.1 we conclude that

$$\mathrm{e}^{\widehat{A}} - T_m\Big(\frac{\widehat{A}}{n}\Big)^n \leq \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\mathrm{e}^{\widehat{A}} \leq \frac{2\widehat{A}^{m+1}}{n^m(m+1)!} T_m\Big(\frac{\widehat{A}}{n}\Big)^n \leq \frac{2\tau}{1+2\tau} T_m\Big(\frac{\widehat{A}}{n}\Big)^n \leq \frac{2\tau}{1+2\tau}\mathrm{e}^{\widehat{A}}.$$

Repeating the same procedure with the new bound, we eventually obtain

$$\mathrm{e}^{\widehat{A}} - T_m\Big(\frac{\widehat{A}}{n}\Big)^n \leq \frac{\widehat{A}^{m+1}}{n^m(m+1)!}\mathrm{e}^{\widehat{A}} \leq \frac{1}{1-\frac{2\tau}{1+2\tau}} \frac{\widehat{A}^{m+1}}{n^m(m+1)!} T_m\Big(\frac{\widehat{A}}{n}\Big)^n \leq \frac{\frac{\tau}{1+2\tau}}{1-\frac{2\tau}{1+2\tau}}\mathrm{e}^{\widehat{A}} = \tau\mathrm{e}^{\widehat{A}}.$$

The error bound is roughly reduced by half by the careful analysis. Sometimes this might have some potential benefit to reduce the computational cost also since either m or n can be chosen a bit smaller. Even if (3.10) is not satisfied for a given choice of (m,n), say

$$\varepsilon_{m,n} = \min\left\{ \varepsilon \colon \frac{\widehat{A}^{m+1}}{n^m(m+1)!} T_m\Big(\frac{\widehat{A}}{n}\Big)^n \leq \varepsilon T_m\Big(\frac{\widehat{A}}{n}\Big)^n \right\} > \frac{\tau}{1+2\tau},$$

it also useful to find out the possible choice of m and n. For example, we can expect $\varepsilon_{m,2n} \sim 2^{-m}\varepsilon_{m,n}$ when $\varepsilon_{m,n} < 1$ because the convergence rate provided in Lemma 3.1 is reasonably accurate. This kind of heuristics often leads to the appropriate values rapidly. The typical situation in practice is that suitable values of m and n can be found immediately after an initial guess.

***Remark* 3.** If $e^A v$ (for some vector $v \geq 0$) instead of e^A is of interest, the stopping criterion can be changed to

$$\frac{\widehat{A}^{m+1}}{n^m(m+1)!}T_m\left(\frac{\widehat{A}}{n}\right)^n v \leq \frac{\tau}{1+2\tau}T_m\left(\frac{\widehat{A}}{n}\right)^n v,$$

which might be easier to satisfy than (3.10).

***Remark* 4.** Sometimes e^A contains very tiny nonzero entries which are less than or close to $R_{\min}$. For example, when A is symmetric and banded, the magnitude of off-diagonal entries in e^A decay rapidly along with the distance to the main diagonal since e^x is smooth [3]. Therefore it is advisable to have another absolute tolerance τ_0 to avoid unfeasible accuracy requirements on these tiny entries.

The difference between true zeros and underflows can be distinguished by calculating $\mathrm{struct}[(I+B)^n]$, where $n = 2^{\lceil \log_2 N \rceil}$ and B is a binary matrix satisfying $\mathrm{struct}(B) = \mathrm{struct}(A)$.

In principle the previous discussions are valid for any choice of m and n. We propose Algorithm 1 as an example of the aggressively truncated Taylor series method, i.e., fix m and find a proper n. This strategy that m is fixed is similar to some Padé methods (e.g., [11, 21]). Compared to the traditional Taylor series method, an advantage of fixing m is that there are cheaper alternatives for evaluating $T_m(X)$ (see, e.g., [11, 26]). For example,

$$\begin{aligned} T_6(X) &= \left(I + X + \frac{X^2}{2!}\right) + X^3\left(\frac{I}{3!} + \frac{X}{4!} + \frac{X^2}{5!} + \frac{X^3}{6!}\right), \\ T_7(X) &= \left(I + \frac{X^2}{2!} + \frac{X^4}{4!} + \frac{X^6}{6!}\right) + X\left(I + \frac{X^2}{3!} + \frac{X^4}{5!} + \frac{X^6}{7!}\right). \end{aligned} \tag{3.11}$$

They both require two less matrix multiplications than the naive approach. The reformations in (3.11) are superior even taking into account the further cost for à posteriori error estimate. The upper bound of computational cost for $1 \leq m \leq 21$ is summarized in [11]. If the same idea is applied to traditional Taylor series methods, the à priori estimate of convergence rate becomes crucial; the algorithm will become very complicated too. Combining this observation with the previous discussions about balancing between m and n, it is advisable to use, e.g., $m = 9$ or 12, for IEEE double precision arithmetic. Taking into account the overscaling issue, we recommend that $m = 13$ is used so that $n = 4C(A)$ is sufficient for $N \leq 2048$. Hence we have a good control of the scale factor.

From the viewpoint of computational cost, actually there exist other advantages beyond the trick for polynomial evaluation. An upper bound of the computational cost for Algorithm 1 is $2(m + \log_2 n)N^3 = O(N^3 \log n)$ for each iteration when N is large. Once the algorithm has successfully achieved the desired accuracy, it implies that

$$\frac{[N - 1 + \hat{\rho}(A)]^{m+1}}{n^m(m+1)!} = O(\tau).$$

We have already seen in Section 2 that $\hat{\rho}(A)$ cannot grow as rapid as N does. Thus we obtain $\log n = O(\log N)$, although n can be much greater than N. Another important fact is that very typically the appropriate choice of n is predictable since the convergence rate is known.

***Remark* 5.** Usually at most three iterations are enough in Algorithm 1. Therefore the computational cost is roughly $O(N^3 \log N)$ which is not much more expensive than other $O(N^3)$ algorithms.

The complexity is especially attractive compared to that of the traditional Taylor series method when τ is large. Very often only four correct digits are of interest, i.e. $\tau = 10^{-4}$. In this case the traditional method still needs at least $O(N^4)$ operations to guarantee the convergence while the new method can accept a much smaller choice of n and is hence much cheaper.

Algorithm 1 Lower Bound Algorithm for e^A, with A essentially nonnegative

Require: $\widehat{A} \in \mathbb{R}_+^{N\times N}$, $m \in \mathbb{N}$, $\tau > \mathbf{u}$, $\tau_0 > R_{\min}$, $\texttt{MAXITER} \in \mathbb{N}$
(optional) balancing (see Section 3.4)
$k \leftarrow \left\lceil \log_2\left(N + \max\{\widehat{A}(i,i)\}\right)\right\rceil + 1$
$\varepsilon \leftarrow \tau + 1$, $\varepsilon_0 \leftarrow \varepsilon$
while $\varepsilon_0 \geq \varepsilon \geq \frac{\tau}{1+2\tau}$ **and** $k \leq \texttt{MAXITER}$ **do**
$n \leftarrow 2^k$, $\varepsilon_0 \leftarrow \varepsilon$
$L \leftarrow L_{m,n}(A)$
$W \leftarrow \dfrac{n(\widehat{A}/n)^{m+1}}{(m+1)!} L$
$\varepsilon \leftarrow \max\left\{ \dfrac{W(i,j)}{L(i,j)} : W(i,j) \geq \tau\tau_0 \right\}$
$k \leftarrow k + \dfrac{1}{m}\log_2(\varepsilon/\tau)$
end while
$E \leftarrow L$
if $\varepsilon \geq \tau$ **then**
Report failure
end if
(optional) reversed balancing

3.2 Upper bound algorithm

We have already seen that $L_{1,n}(A) = (I + A/n)^n$ can be interpreted as the approximated solution of (3.3) provided by forward Euler's method. Similarly, the backward Euler's method yields $(I - A/n)^{-n}$, which is an upper bound of e^A when A is essentially non-

negative. To generalize the upper bound to higher order methods, we define

$$\begin{aligned}\widetilde{T}_m(x) &= T_{m-2}(x) + \frac{x^{m-1}}{(m-1)!}\left(1-\frac{x}{m}\right)^{-1} \\ &= 1 + x + \cdots + \frac{x^m}{m!} + \frac{x^{m+1}}{m!\,m} + \frac{x^{m+2}}{m!\,m^2} + \cdots\end{aligned}$$

and

$$U_{m,n}(A) = \mathrm{e}^{s(A)}\widetilde{T}_m\left(\frac{\widehat{A}}{n}\right)^n. \tag{3.12}$$

Here $\widetilde{T}_m(x)$ can also be interpreted as a non-diagonal Padé approximant of e^x. The (absolute value of) corresponding remainder is denoted as

$$\widetilde{R}_m(x) = \widetilde{T}_m(x) - \mathrm{e}^x.$$

Then for an essentially nonnegative matrix A with $\hat{\rho}(A) < mn$, $\widetilde{R}_m(A) \geq 0$, we have $U_{m,n}(A) \geq \mathrm{e}^A$. Similar to the lemmas for $L_{m,n}(A)$, we are able to obtain some conclusions for $U_{m,n}(A)$.

Lemma 3.4. *If $A \in \mathbb{R}^{N\times N}$ is essentially nonnegative, then for positive integers m and n satisfying $mn > \hat{\rho}(A)$, we have*

$$\lim_{n\to\infty} U_{m,n}(A) = \lim_{m\to\infty} U_{m,n}(A) = \mathrm{e}^A. \tag{3.13}$$

Furthermore, if $m > 2$, the truncation error is bounded by

$$\mathrm{e}^{s(A)/n}\frac{\widehat{A}^{m+1}}{n^m(m+1)!\,m}\mathrm{e}^{\frac{n-1}{n}A} \leq U_{m,n}(A) - \mathrm{e}^A \leq \mathrm{e}^{s(A)/n}\frac{\widehat{A}^{m+1}}{n^m(m+1)!\,m}\left(I-\frac{\widehat{A}}{mn}\right)^{-1} U_{m,n}(A)^{\frac{n-1}{n}}. \tag{3.14}$$

The componentwise estimate is also available; but we need a different approach. Notice that for any nonnegative matrix X, we have

$$\mathrm{e}^X \leq \widetilde{T}_m(X) = \mathrm{e}^X + \widetilde{R}_m(X) \leq \mathrm{e}^X[I + \widetilde{R}_m(X)],$$

and then

$$\widetilde{T}_m\left(\frac{X}{n}\right)^n - \mathrm{e}^X \leq \left[\left(I + \widetilde{R}_m\left(\frac{X}{n}\right)\right)^n - I\right]\mathrm{e}^X \leq \left[\left(1 + \widetilde{R}_m\left(\frac{C(X)}{n}\right)\right)^n - 1\right]\mathrm{e}^X.$$

For an essentially nonnegative matrix A, it is sufficient to verify that

$$U_{m,n}(A) - \mathrm{e}^A = \mathrm{e}^{s(A)}\left[U_{m,n}(\widehat{A}) - \mathrm{e}^{\widehat{A}}\right].$$

Hence we obtain the following theorem.

Theorem 3.5. *Let $A \in \mathbb{R}^{N\times N}$ be an essentially nonnegative matrix. Then for any positive integers m and n satisfying $mn > C(A)$, we have*

$$0 \leq U_{m,n}(A) - \mathrm{e}^A \leq \left[\left(1 + \widetilde{R}_m\left(\frac{C(A)}{n}\right)\right)^n - 1\right]\mathrm{e}^A. \tag{3.15}$$

Notice that

$$\widetilde{R}_m(x) \sim \frac{x^{m+1}}{(m+1)!\,m},$$

then once $\widetilde{R}_m\big(C(A)/n\big)$ is small, we have

$$\left[\left(1+\widetilde{R}_m\Big(\frac{C(A)}{n}\Big)\right)^n - 1\right] \mathrm{e}^A \sim \frac{C(A)^{m+1}}{n^m(m+1)!\,m}\mathrm{e}^A,$$

which is similar to (3.6) in Theorem 3.3.

Similar to the lower bound, $U_{m,n}(A)$ has the monotonicity property

$$U_{m,n}(A) \geq U_{m+1,n}(A) \quad \text{and} \quad U_{m,n}(A) \geq U_{m,2n}(A), \tag{3.16}$$

which can be verified by comparing the corresponding coefficients in the Maclaurin series. In analogy to (3.8), we conjecture

$$U_{m,n}(A) \geq U_{m,n+1}(A). \tag{3.17}$$

Choosing appropriate values for m and n is similar and in some sense simpler compared to that for the lower bounds. If a good estimate of $\hat{\rho}(A)$ is available, we can apply the à priori estimate

$$\left(1+\widetilde{R}_m\Big(\frac{C(A)}{n}\Big)\right)^n - 1 \leq \tau. \tag{3.18}$$

Otherwise, we can use either

$$\frac{\widehat{A}^{m+1}}{n^m(m+1)!\,m}\Big(I-\frac{\widehat{A}}{mn}\Big)^{-1}\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n-1} \leq \frac{\tau}{1+\tau}\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^n \tag{3.19}$$

or

$$\frac{\widehat{A}^{m+1}}{n^m(m+1)!\,m}\Big(I-\frac{\widehat{A}}{mn}\Big)^{-1}\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^{n} \leq \frac{\tau}{1+\tau}\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^n \tag{3.20}$$

as an à posteriori estimate. If (3.19) is used, it is advisable to choose $n = 2^k + 1$ instead of $n = 2^k$ so that evaluating $\widetilde{T}_m(\widehat{A}/n)^{n-1}$ does not require extra cost. Both inequalities (3.19) and (3.20) imply that

$$\widetilde{T}_m\Big(\frac{\widehat{A}}{n}\Big)^n - \mathrm{e}^{\widehat{A}} \leq \tau \mathrm{e}^{\widehat{A}}.$$

A notable difference between $U_{m,n}(A)$ and $L_{m,n}(A)$ is that the matrix inverse is needed for evaluating the upper bound. Since $I-\widehat{A}/(mn)$ is an M-matrix, its inverse can be calculated accurately using GTH-type algorithms [1]. Actually by increasing the scaling, $[I-\widehat{A}/(2mn)]^{-1}$ can be safely computed even by standard Gaussian elimination without pivoting because it is always well-conditioned in the sense of componentwise perturbation (see Lemma 4.4). The subtractions on the diagonal never cause serious cancellation and can be interpreted as small relative backward errors. Both GTH-type algorithms and Gaussian elimination can adopt block variants so that most computational work is done in Level 3 BLAS. This feature is useful when the performance is important. Algorithm 2 implements an upper bound computation similar to Algorithm 1.

Algorithm 2 Upper Bound Algorithm for e^A, with A essentially nonnegative

Require: $\widehat{A} \in \mathbb{R}_+^{N\times N}$, $m \in \mathbb{N}$, $\tau > \mathbf{u}$, $\tau_0 > R_{\min}$, $\texttt{MAXITER} \in \mathbb{N}$
  (optional) balancing (see Section 3.4)
  $k \leftarrow \left\lceil \log_2\left(N + \max\{\widehat{A}(i,i)\}\right)\right\rceil + 1$
  $\varepsilon \leftarrow \tau + 1$, $\varepsilon_0 \leftarrow \varepsilon$
  while $\varepsilon_0 \geq \varepsilon \geq \frac{\tau}{1+\tau}$ **and** $k \leq \texttt{MAXITER}$ **do**
    $n \leftarrow 2^k$, $\varepsilon_0 \leftarrow \varepsilon$
    Try $U \leftarrow U_{m,n}(A)$
    if $\hat{\rho}(A) \geq mn$ **then**
      $W \leftarrow \dfrac{n(\widehat{A}/n)^{m+1}}{(m+1)!\,m}\left(I - \dfrac{\widehat{A}}{mn}\right)^{-1} U$
      $\varepsilon \leftarrow \max\left\{\dfrac{W(i,j)}{U(i,j)} : U(i,j) \geq \tau_0\right\}$
      $k \leftarrow k + \dfrac{1}{m}\log_2(\varepsilon/\tau)$
    else
      $k \leftarrow k+1$
    end if
  end while
  $E \leftarrow U$
  if $\varepsilon \geq \tau$ **then**
    Report failure
  end if
  (optional) reversed balancing

3.3 Interpolation and interval algorithms with improved accuracy

In this section, we present interpolation algorithms based on the upper and lower bounds with potentially improved accuracy. In addition, we present a novel high accuracy interval-type algorithm based on these bounds.

Sometimes both an upper bound and a lower bound of e^A or $e^A v$ (for some vector $v \geq 0$) are of interest. Certainly we are able to apply Algorithm 1 and Algorithm 2 separately to obtain these bounds. But we expect more useful outputs by merging them. Notice that $U_{m,n}(A) - L_{m,n}(A) = \left[U_{m,n}(A) - e^A\right] + \left[e^A - L_{m,n}(A)\right]$, so the difference between the upper bound and the lower bound is also componentwise small compared to e^A. Thus we obtain a simpler à posteriori error estimate if both $L_{m,n}(A)$ and $U_{m,n}(A)$ have already been calculated. In principle, any matrix X satisfying $L_{m,n}(A) \leq X \leq U_{m,n}(A)$ is a good approximation of e^A when both bounds are sufficiently accurate. We can expect an $(m+1)$-th order approximation by interpolating between $L_{m,n}(A)$ and $U_{m,n}(A)$. Similar to the proofs of Lemma 3.1 and Lemma 3.4, it is straightforward to show

$$\lim_{x\to 0} \frac{e^x - T_m(x/n)^n}{x^{m+1}} = m \lim_{x\to 0} \frac{\widetilde{T}_m(x/n)^n - e^x}{x^{m+1}} = \frac{1}{n^m(m+1)!}.$$

Therefore

$$E_{m,n}(A) = \frac{m}{m+1}U_{m,n}(A) + \frac{1}{m+1}L_{m,n}(A)$$

is an $(m+1)$-th order approximation of e^A. The interpolation can also be interpreted as a weighted average. A formal conclusion is stated in the following theorem.

Theorem 3.6. *Let $A \in \mathbb{R}^{N\times N}$ be essentially nonnegative and let m, n be positive integers satisfying $mn > C(A)$. Then*

$$0 \le E_{m,n}(A) - \mathrm{e}^A \le \frac{m}{m+1}\left[\left(1+\widetilde{R}_m\Big(\frac{C(A)}{n}\Big)\right)^n - 1 + \frac{C(A)^{m+2} - C(A)^{m+1}}{n^m(m+1)!\,m}\right]\mathrm{e}^A. \tag{3.21}$$

Proof. Without loss of generality, we can assume $A \ge 0$. From Lemma 3.1 and Lemma 3.4 we have

$$\begin{aligned} E_{m,n}(A) - \mathrm{e}^A &= \frac{m}{m+1}\left[U_{m,n}(A) - \mathrm{e}^A\right] + \frac{1}{m+1}\left[L_{m,n}(A) - \mathrm{e}^A\right] \\ &\ge \frac{1}{m+1}\left[m\widetilde{R}_m\Big(\frac{A}{n}\Big) - R_m\Big(\frac{A}{n}\Big)\right]\mathrm{e}^{\frac{n-1}{n}A}. \end{aligned}$$

Notice that

$$m\widetilde{R}_m(X) - R_m(X) = \sum_{k=1}^{\infty}\left[\frac{1}{m!\,m^{k-1}} - \frac{m+1}{(m+k)!}\right]X^{m+k} \ge 0$$

holds for any nonnegative matrix X, and therefore we obtain

$$E_{m,n}(A) - \mathrm{e}^A \ge 0.$$

On the other hand,

$$\begin{aligned} L_{m,n}(A) - \mathrm{e}^A &\le -\frac{A^{m+1}}{n^m(m+1)!} \\ &= \frac{A^{m+1}}{n^m(m+1)!}\left(\mathrm{e}^A - I\right) - \frac{A^{m+1}}{n^m(m+1)!}\mathrm{e}^A \\ &\le \frac{A^{m+2} - A^{m+1}}{n^m(m+1)!}\mathrm{e}^A. \end{aligned}$$

Therefore

$$\begin{aligned} &E_{m,n}(A) - \mathrm{e}^A \\ &\le \frac{m}{m+1}\left[\left(1+\widetilde{R}_m\Big(\frac{A}{n}\Big)\right)^n - 1 + \frac{A^{m+2} - A^{m+1}}{n^m(m+1)!\,m}\right]\mathrm{e}^A \\ &\le \frac{m}{m+1}\left[\left(1+\widetilde{R}_m\Big(\frac{C(A)}{n}\Big)\right)^n - 1 + \frac{C(A)^{m+2} - C(A)^{m+1}}{n^m(m+1)!\,m}\right]\mathrm{e}^A, \end{aligned}$$

based on the fact that

$$\left[1+\widetilde{R}_m\left(\frac{x}{n}\right)\right]^n - 1 - \frac{x^{m+1}}{n^m(m+1)!\,m}$$

has nonnegative coefficients in its Maclaurin expansion. □

***Remark* 6.** From the definition of $\widetilde{T}_m(x)$, we are able to see that the terms up to order $m-2$ are shared when evaluating $T_m(\widehat{A}/n)$ and $\widetilde{T}_m(\widehat{A}/n)$ simultaneously. This observation is also valid for some alternative approaches such as (3.11).

In applications [13] where accurate bounds are extremely important, it is advisable to switch the rounding modes during the calculation so that

$$\underline{\mathrm{fl}}[L_{m,n}(A)] \le L_{m,n}(A) \le \mathrm{e}^A \le U_{m,n}(A) \le \overline{\mathrm{fl}}[U_{m,n}(A)] \tag{3.22}$$

is guaranteed regardless of rounding errors. We will prove this property in the next section. In this case $L_{m,n}(A)$ and $U_{m,n}(A)$ need to be calculated separately and no computational cost can be saved. This approach can indeed be seen as an interval algorithm [16] but without any explicit use of interval arithmetic. The interpolation step can still be applied although $\mathrm{fl}[E_{m,n}(A)]$ is not guaranteed to be another upper bound of e^A. The complete algorithm is summarized as Algorithm 3 which provides both lower and upper bounds and a higher order approximation in between. Because of (3.22), this algorithm is more reliable in practice than Algorithm 1 and Algorithm 2 but it is also more costly.

3.4 Essentially nonnegative matrices

In the previous discussions, all definitions and conclusions are valid for essentially nonnegative matrices. The key technique is to shift the matrix to a nonnegative matrix so that some properties of completely monotonic functions and Lemma 3.2 can be applied. Some extra care must be taken during the shifting. If we compute $\mathrm{e}^{\widehat{A}}$ and $\mathrm{e}^{s(A)}$ straightforwardly, it is possible that $\mathrm{e}^{\widehat{A}}$ causes overflow but $\mathrm{e}^{s(A)}$ is small enough so that the desired result e^A is still representable with floating point numbers. Therefore shifting needs to be handled carefully when $s(A) < 0$. Since

$$\mathrm{e}^A = \left(\mathrm{e}^{A/n}\right)^n = \left(\mathrm{e}^{s(A)/n}\mathrm{e}^{\widehat{A}/n}\right)^n,$$

the shift $s(A)/n$ might be safe if n is large enough. The same observation can also be applied to $L_{m,n}(A)$ and $U_{m,n}(A)$, i.e.,

$$L_{m,n}(A) = \left[\mathrm{e}^{s(A)/n}T_m\left(\frac{\widehat{A}}{n}\right)\right]^n, \tag{3.23}$$

$$U_{m,n}(A) = \left[\mathrm{e}^{s(A)/n}\widetilde{T}_m\left(\frac{\widehat{A}}{n}\right)\right]^n. \tag{3.24}$$

By applying the scaling before shifting, we are most likely able to avoid unnecessary overflow. This trick is commonly used in computations of Markov models (e.g., see

Algorithm 3 Interval Algorithm for e^A, with A essentially nonnegative

Require: $\widehat{A} \in \mathbb{R}_+^{N\times N}$, $m \in \mathbb{N}$, $\tau > \mathbf{u}$, $\tau_0 > R_{\min}$, $\texttt{MAXITER} \in \mathbb{N}$
(optional) balancing (see Section 3.4)
$k \leftarrow \left\lceil \log_2\left(N + \max\{\widehat{A}(i,i)\}\right)\right\rceil + 1$
$\varepsilon \leftarrow \tau + 1$, $\varepsilon_0 \leftarrow \varepsilon$
$L \leftarrow A$, $U \leftarrow \infty$
while $\varepsilon_0 \geq \varepsilon \geq \tau$ **and** $k \leq \texttt{MAXITER}$ **do**
  $n \leftarrow 2^k$, $\varepsilon_0 \leftarrow \varepsilon$
  $L \leftarrow \max\left\{L, \underline{\mathrm{fl}}[L_{m,n}(A)]\right\}$
  (Try) $U \leftarrow \min\left\{U, \overline{\mathrm{fl}}[U_{m,n}(A)]\right\}$
  $\varepsilon \leftarrow \max\left\{\dfrac{U(i,j) - L(i,j)}{L(i,j)} : U(i,j) \geq \tau_0\right\}$
  $k \leftarrow k + \dfrac{1}{m}\log_2(\varepsilon/\tau)$
end while
$E \leftarrow \dfrac{1}{m+1}L + \dfrac{m}{m+1}U$
if $\varepsilon \geq \tau$ **then**
  Report failure
end if
(optional) reversed balancing

[17, 26]). Moreover, we suggest that the shifting strategies (3.23) and (3.24) are always used even for $A \geq 0$ since some entries of $e^{\widehat{A}}$ might underflow. In case that $|s(A)|$ is extremely large so that n has to be large to avoid overflow/underflow, we conclude from Section 2 that the condition number of the problem is also large. Such problems need special attention, and are not investigated in this paper.

Balancing is another commonly used technique for computing the matrix exponential, and can be applied in all the algorithms discussed in this section. It helps slightly to improve the accuracy when summing up poorly scaled numbers. Notice that balancing cannot reduce $\hat{\rho}(A)$ so it does not help reducing the condition number and computational cost. We provide balancing as an optional step. For essentially nonnegative matrices, since $\|A\|_\infty$ can be reduced by balancing, the technique can help avoiding potential overflow/underflow. Although balancing is not always safe for some eigenvalue problems [22], extensive numerical experiments in [18] show that it is unlikely to be harmful when computing matrix functions.

4 Rounding Error Analysis

In this section we provide error bounds taking into account effects of finite precision arithmetic. The bounds usually overestimate the errors in practice; their main purpose is to provide the evidence for the forward stability of the algorithms. If there are no

overflows or (gradual) underflows in the calculation, the rounding error is modeled by

$$\mathrm{fl}(a \circ b) = (a \circ b)(1+\varepsilon), \qquad |\varepsilon| \le \mathbf{u}, \tag{4.1}$$

where "$\circ$" is $+$, $-$, $\times$, or $\div$. We further assume that $N^2\mathbf{u} \ll 1$, which is a reasonable requirement in practice, so that all $O(\mathbf{u}^2)$ terms are negligible.

Firstly, the matrix polynomial $T_m(A/n)$ can be computed stably. It is not difficult to prove the following conclusion by induction.

Lemma 4.1. *If $A \in \mathbb{R}_+^{N\times N}$, then for any positive integers m and n, we have*

$$\left| \mathrm{fl}\left[T_m\Big(\frac{A}{n}\Big) \right] - T_m\Big(\frac{A}{n}\Big) \right| \le \big[(m+1)(N+2)\mathbf{u} + O(\mathbf{u}^2)\big] T_m\Big(\frac{A}{n}\Big).$$

For the scaling procedure, the following lemma is slightly modified from Theorem 4.9 in [2]. It is well-known that squaring is the most dangerous step in the whole algorithm since the error can grow quickly. The error bound for repeated squaring is almost attainable in the worst case. However, the error estimate is usually too pessimistic, especially when there are no significant cancellations during this procedure. We have already seen in Section 3 that sometimes perturbations in a nonnegative matrix B can even decay during the repeated squaring process. The full understanding of the squaring phase is still an open problem [15]. It is good to keep in mind that repeated squaring can be dangerous, but that this is not always the case.

Lemma 4.2. *Let $\Delta B \in \mathbb{R}^{N\times N}$ be the perturbation of a nonnegative matrix B satisfying $|\Delta B| \le \varepsilon B$ for some $0 \le \varepsilon < (100N)^{-1}$. Then for $n = 2^k$, we have*

$$\big|\mathrm{fl}\big[(B+\Delta B)^n\big] - B^n\big| \le \big[n(N+\varepsilon)\mathbf{u} + O(\varepsilon\mathbf{u}+\mathbf{u}^2)\big] B^n.$$

By substituting Lemma 4.1 into Lemma 4.2, we obtain the rounding error for Algorithm 1. A similar theorem can be found in [8].

Theorem 4.3. *Let A be an $N \times N$ essentially nonnegative matrix and let m, n be positive integers satisfying $\mathrm{e}^A - L_{m,n}(A) \le \tau \mathrm{e}^A$. Then*

$$\big|\mathrm{fl}[L_{m,n}(A)] - \mathrm{e}^A\big| \le \big[\tau + n(m+2)(N+2)\mathbf{u} + O(\mathbf{u}^2)\big]\mathrm{e}^A.$$

To obtain the rounding error for $U_{m,n}(A)$, the inverse of the M-matrices is the only different part compared to $L_{m,n}(A)$. Let M be an $N \times N$ nonsingular M-matrix with the Jacobi splitting $M = D - F$ where D is diagonal and F has zero diagonal entries. Then Lemma 4.4 characterizes the sensitivity of M^{-1} by $\gamma = \rho\big(D^{-1}F\big)$. Once $mn > 2\hat{\rho}(A)$, $M = I - \widehat{A}/(mn)$ is a well-conditioned nonsingular M-matrix with $\gamma < 1/2$. Moreover, the LU factorization of M satisfies $s(U) \ge 1/2$. This indicates that M^{-1} can be calculated componentwise accurately via standard Gaussian elimination. The error bound is given by Lemma 4.5.

Lemma 4.4. [24] *Assume M is an $N \times N$ nonsingular M-matrix. Let ΔM be a perturbation to M satisfying $|\Delta M| \le \varepsilon |M|$ where $0 \le \varepsilon < (1-\gamma)/(1+\gamma)$. Then we have*

$$\big|(M+\Delta M)^{-1} - M^{-1}\big| \le \left(\frac{2-\gamma}{1-\gamma} N\varepsilon + O(\varepsilon^2)\right) M^{-1}.$$

Lemma 4.5. *Let $B \in \mathbb{R}_+^{N\times N}$ satisfying $\rho(B) < 1/2$. Then the inverse of $M = I - B$ calculated by standard Gaussian elimination without pivoting satisfies*

$$\left|\mathrm{fl}\left(M^{-1}\right) - M^{-1}\right| \le \left[(15N^2+5N+2)\mathbf{u} + O(\mathbf{u}^2)\right]M^{-1}.$$

Sketch of Proof. First, we state that the LU factorization of M satisfies

$$\left|\mathrm{fl}(L)\,\mathrm{fl}(U) - M\right| \le \left[4.01N\mathbf{u} + O(\mathbf{u}^2)\right]|M|.$$

This conclusion can be proven by induction. We remark that during the proof we rely on the fact that the Schur complement satisfies

$$|M_{22}| + M_{21}M_{11}^{-1}M_{12} \le 3\left|M_{22} - M_{21}M_{11}^{-1}M_{12}\right|,$$

based on the assumption $\rho(B) < 1/2$. Then from Lemma 4.4 we obtain

$$\left|\left[\mathrm{fl}(L)\,\mathrm{fl}(U)\right]^{-1} - M^{-1}\right| \le \left[3N(4.01N+1)\mathbf{u} + O(\mathbf{u}^2)\right]M^{-1}.$$

It is well-known that the inverse of triangular M-matrices can be calculated accurately using the back substitution method. The componentwise relative error is bounded by $(N^2+N+1)\mathbf{u} + O(\mathbf{u}^2)$ (see [9, 23]). Therefore we have

$$\begin{aligned}
&\left|\mathrm{fl}\left(M^{-1}\right) - M^{-1}\right| \\
\le\ &\left|\left[\mathrm{fl}(L)\,\mathrm{fl}(U)\right]^{-1} - M^{-1}\right| + \left|\mathrm{fl}\left[\left[\mathrm{fl}(L)\,\mathrm{fl}(U)\right]^{-1}\right] - \left[\mathrm{fl}(L)\,\mathrm{fl}(U)\right]^{-1}\right| \\
\le\ &\left[(12.03N^2+3N)\mathbf{u} + 2(N^2+N+1)\mathbf{u} + O(\mathbf{u}^2)\right]M^{-1}. \qquad \square
\end{aligned}$$

Then we are able to obtain the rounding error for $U_{m,n}(A)$ similar to Theorem 4.3.

Theorem 4.6. *Let A be an $N \times N$ essentially nonnegative matrix and let m, n be positive integers satisfying $mn > 2\hat{\rho}(A)$ and $U_{m,n}(A) - \mathrm{e}^A \le \tau\mathrm{e}^A$. Then*

$$\left|\mathrm{fl}[U_{m,n}(A)] - \mathrm{e}^A\right| \le \left[\tau + n(m+15N)(N+2)\mathbf{u} + O(\mathbf{u}^2)\right]\mathrm{e}^A.$$

If true bounds of e^A are required, we need to switch the rounding modes during the calculation to achieve this goal. The rounding error can be modeled as

$$\underline{\mathrm{fl}}(a \circ b) = (a \circ b)(1-\varepsilon_1), \quad \overline{\mathrm{fl}}(a \circ b) = (a \circ b)(1+\varepsilon_2), \quad 0 \le \varepsilon_1, \varepsilon_2 \le 2\mathbf{u}.$$

We are still able to use the standard rounding model (4.1) by setting $2\mathbf{u}$ as the machine epsilon. Therefore all previous conclusions are still valid for the interval algorithm. Then we only need to verify the property (3.22). Obviously, $\underline{\mathrm{fl}}[L_{m,n}(A)] \le L_{m,n}(A)$ is easy to be satisfied by simply switching the rounding mode to *round towards* $-\infty$. For the upper bound, the tricky part is to achieve $\overline{\mathrm{fl}}\left(M^{-1}\right) \ge M^{-1}$, where $M = I - A/(mn)$ is an M-matrix. For example, GTH-type algorithms [1] do not have this property.

Firstly, $\mathrm{fl}(M)$ is obtained by $\mathrm{fl}(M) = -\overline{\mathrm{fl}}\left[\overline{\mathrm{fl}}[A/(mn) - I]\right] \le M$. Consider one step of Gaussian elimination for calculating the LU factorization

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} = \begin{bmatrix} 1 & \\ L_{21} & I \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} \\ & S_{22} \end{bmatrix},$$

where S_{22} is the Schur complement. The factorization can be computed via

$$\begin{aligned}\mathrm{fl}(L_{21}) &= -\overline{\mathrm{fl}}(|L_{21}|) = -\overline{\mathrm{fl}}\left[\frac{\overline{\mathrm{fl}}(|M_{21}|)}{\underline{\mathrm{fl}}(M_{11})}\right] \leq L_{21},\\ \mathrm{fl}(S_{22}) &= -\overline{\mathrm{fl}}\left[\overline{\mathrm{fl}}(|L_{21}|)\,\overline{\mathrm{fl}}(|M_{12}|) - \underline{\mathrm{fl}}(M_{22})\right] \leq S_{22}.\end{aligned}$$

By applying the same procedure recursively to S_{22}, we are able to conclude that the LU factorization $M = LU$ satisfies

$$\underline{\mathrm{fl}}(L) = \mathrm{fl}(L) \leq L, \qquad \underline{\mathrm{fl}}(U) = \mathrm{fl}(U) \leq U.$$

Suppose M^{-1} is computed by solving two triangular linear systems under the rounding mode to *round towards* $+\infty$, the solution satisfies

$$\overline{\mathrm{fl}}\left(M^{-1}\right) \geq \underline{\mathrm{fl}}(U)^{-1}\,\underline{\mathrm{fl}}(L)^{-1} \geq U^{-1}L^{-1} = M^{-1}.$$

The conclusion can be generalized to block LU factorization and some algorithmic variants for computing M^{-1} as long as $\mathrm{fl}(L)$ and $\mathrm{fl}(U)$ are true bounds of L and U, respectively. Therefore if the actual upper bound is important, we prefer standard Gaussian elimination rather than GTH-type algorithms. In practice all calculations can be done without switching the rounding mode (under the rounding mode to *round towards* $+\infty$).

The error bounds presented in this section confirm the componentwise accuracy of the algorithms proposed in Section 3. Although Algorithm 3 can always produce reliable solutions regardless of roundoff, the rounding error analysis still proves that the interval algorithm does not severely overestimate the error. In practice the à posteriori error is usually better than the priori error based on truncation and rounding error analysis.

5 Numerical Experiments

In this section, we present and discuss some experimental results. All experiments are tested with Algorithm 3, implemented in C99. Matrix multiplications are performed by the optimized GEMM routine from the GotoBLAS library; while other operations, such as block LU factorization and solving triangular systems, are carefully coded so that they are not ruined by the compiler. For the parameter setting, we choose $m = 7$ as the truncation order. It is relatively small compared to the recommended value ($m = 13$) so that we are able to see that "overscaling" is not too harmful in practice. The relative and absolute tolerances are set to $\tau = 1024N\mathbf{u} \approx 2.3 \times 10^{-13}N$ and $\tau_0 = R_{\min}/\mathbf{u} \approx 1.0 \times 10^{-292}$, respectively. The maximum iteration number is set to `MAXITER` $= 52$ which is more than needed. Balancing is not used in the experiments to show that our algorithm is not sensitive to bad scaling in the original matrix. The switching of rounding modes are done by the standard library function `fesetround()`. As a comparison, we also run the same tests for MATLAB's `expm` function (version R2011b) which is an excellent general purpose solver. We choose

eight test matrices which are commonly used. All entries in the exponentials can be represented by normalized floating point numbers (i.e. no overflow or (gradual) underflow).

Example 1. [6]

$$A = \begin{bmatrix} -0.01 & 10^{15} \\ 0 & -0.01 + 10^{-6} \end{bmatrix}.$$

Example 2. [5, 6]

$$A = \begin{bmatrix} -16 & 2^{60} & 2^{60} & 2^{60} \\ 0 & -16 & 2^{60} & 2^{60} \\ 0 & 0 & -1 & 2^{60} \\ 0 & 0 & 0 & -1 \end{bmatrix}.$$

Example 3. [21] *The* 10×10 *Forsythe matrix*

$$A = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 10^{-10} & & & 0 \end{bmatrix}_{10\times 10}.$$

Example 4. [26] *The* 50×50 *one-dimensional Laplacian matrix*

$$A = -T_{50} = \begin{bmatrix} -2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & -2 \end{bmatrix}_{50\times 50}.$$

Example 5. *The* 128×128 *Jordan block with zero eigenvalues*

$$A = J_{128}(0) = \begin{bmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix}_{128\times 128}.$$

Example 6. [26] *The adjacency matrix of a* 2*-nearest ring network of* 200 *nodes with four extra shortcuts* 16–30*,* 74–85*,* 90–128*,* 138–147*, modelling a small-world network. It is generated by* `smallw(200,2,0.03)`*, where* `smallw.m` *is from the MATLAB toolbox CONTEST*[1] [19].

Example 7. [26] *The two-dimensional Laplacian matrix on a* 42×42 *grid with Dirichlet boundary condition*

$$A = -T_{40,40} = -(T_{40} \otimes I_{40} + I_{40} \times T_{40}).$$

[1] CONTEST: http://www.mathstat.strath.ac.uk/research/groups/numerical_analysis/contest

TABLE 1: Componentwise relative errors of the test examples.

Matrix ID	C(A)	τ	posteriori error	$\log_2 n$	#(iter)	expm
1	≈ 1	4.5×10^{-13}	6.7×10^{-16}	2	1	1.0×10^{-2}
2	18	9.1×10^{-13}	2.5×10^{-13}	9	2	1.0×10^{0}
3	9.1	2.3×10^{-12}	8.0×10^{-14}	7	3	1.8×10^{-14}
4	≈ 51	1.1×10^{-11}	2.7×10^{-12}	10	2	2.6×10^{6}
5	127	2.9×10^{-11}	9.7×10^{-12}	11	2	9.7×10^{77}
6	≈ 203	4.5×10^{-11}	3.6×10^{-11}	9	2	1.8×10^{7}
7	≈ 1603	3.6×10^{-11}	1.7×10^{-11}	11	1	4.2×10^{6}
8	2047	4.6×10^{-10}	5.6×10^{-11}	15	2	6.0×10^{-13}

Example 8.

$$A = 1400\times J_{2048}(-1/2) = \begin{bmatrix} -700 & 1400 & & & \\ & \ddots & \ddots & & \\ & & \ddots & 1400 & \\ & & & -700 \end{bmatrix}_{2048\times 2048}.$$

The à posteriori error estimates for the testing examples are listed in Table 1. If the exact solution is not known, we use the output of Algorithm 3 as the "accurate" solution, since the à posteriori error for Algorithm 3, provided by itself, is always reliable regardless of rounding error. Algorithm 3 successfully solves all these problems to desired accuracy, while MATLAB's `expm` function only produces componentwise accuracy for two of them. In fact, for Examples 1, 2 and 6, the normwise forward errors of `expm` are also large (1.0×10^{-2}, 1.0×10^{0}, and 3.7×10^{-1}, respectively).

Example 5 is of our particular interest. The exact solution is known as

$$\left(\mathrm{e}^{A}\right)(i,j) = \begin{cases} 0, & i > j, \\ \frac{1}{(j-i)!}, & i \le j, \end{cases}$$

whose nonzero entries are the coefficients in the Maclaurin series of e^{x}. Algorithm 3 terminates successfully with $n = 2048$. Therefore the first 128 terms in the Maclaurin series of $T_7\left(x/2048\right)^{2048}$ and $\widetilde{T}_7\left(x/2048\right)^{2048}$ both agree with those terms in e^{x} within a relative error of 10^{-11}. Since $1/128! < 2.6\times10^{-216}$, the remaining terms are very tiny. This is an intuitive way to understand why our method can always produce high relative accuracy. Example 8 is a very challenging problem because the magnitude of nonzero entries in the solution vary from 10^{-304} to 10^{302}. Algorithm 3 still provides componentwise accurate bounds for this extreme case. Interestingly, `expm` also does a good job for this difficult example but fails for Example 5 which should be easier. A brief explanation is that `expm` tries to avoid "unnecessary" scaling in Example 5 and thus loses the opportunity to recover small entries in the solution. For most examples Algorithm 3 uses two iterations since the estimate of proper n based on the convergence rate is quite accurate. Example 3 requires one more iteration because the second step produces slightly larger error than the estimated one due to roundoff.

***Remark* 7.** In practice $m = 13$ is recommended as the truncation order for double precision floating point numbers. Algorithm 3 can solve most of these examples in one iteration with this setting. But the convergence rate can be better understood with a smaller truncation order since we need to predict the scale factor from an improper initial guess.

6 Conclusions

Taylor expansions are usually not the first choice for calculating the exponentials of general matrices. However, by carefully choosing the order of expansion and the scale factor, it is preferred when the matrices are nonnegative. We make use of the nonnegativity and establish accurate à priori estimates of the lower and upper bounds. We also show that the weighted average of the bounds derives a higher-order approximation. These theoretical results lead to different variations of the algorithms. Some similar techniques can also be applied to analyze the truncation error for Padé methods. But how to accurately compute the Padé approximant is still a challenging problem. Rounding error analysis ensures the stability of the proposed algorithms. The interval algorithm also provides componentwise error estimates regardless of roundoff. We tested these new algorithms with some commonly used matrices.

Acknowledgements

The authors would like to thank Bo Kågström and Daniel Kressner for fruitful discussions and constructive comments and proposals.

References

[1] A. S. Alfa, J. Xue, and Q. Ye. Accurate computation of the smallest eigenvalue of a diagonally dominant M-matrix. *Math. Comp.*, 71(237):217–236, 2002.

[2] M. Arioli, B. Codenotti, and C. Fassino. The Padé method for computing the matrix exponential. *Linear Algebra Appl.*, 240:111–130, 1996.

[3] M. Benzi and G. H. Golub. Bounds for the entries of matrix functions with applications to preconditioning. *BIT*, 39(3):417–438, 1999.

[4] B. Codenotti and C. Fassino. Error analysis of two algorithms for the computation of the matrix exponential. *Calcolo*, 29(1):1–31, 1992.

[5] P. I. Davies and N. J. Higham. A Schur-Parlett algorithm for computing matrix functions. *SIAM J. Matrix Anal. Appl.*, 25(2):464–485, 2003.

[6] L. Deng and J. Xue. Accurate computation of exponentials of triangular essentially non-negative matrices. *J. Fudan University (Natural Science)*, 50(1):78–86, 2011.

[7] E. Estrada and D. J. Higham. Network properties revealed through matrix functions. *SIAM Rev.*, 52(4):696–714, 2010.

[8] C. Fassino. *Computation of Matrix Functions*. PhD thesis, University of Pisa, 1993.

[9] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[10] N. J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.

[11] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Rev.*, 51(4):747–764, 2009.

[12] B. Kågström. Bounds and perturbation bounds for the matrix exponential. *BIT*, 17(1):39–57, 1977.

[13] R. B. Kearfott and V. Kreinovich, editors. *Applications of Interval Computations*, volume 3 of *Applied Optimization*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[14] C. S. Kenney and A. J. Laub. A Schur-Fréchet algorithm for computing the logarithm and exponential of a matrix. *SIAM J. Matrix Anal. Appl.*, 19(3):640–663, 1998.

[15] C. B. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Rev.*, 45(1):3–49, 2003.

[16] R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

[17] R. B. Sidje. Inexact uniformization and GMRES methods for large Markov chains. *Numer. Linear Algebra Appl.*, 18:947–960, 2011.

[18] M. Stadelmann. Matrixfunktionen — Analyse und Implementierung. Master's thesis, ETH Zürich, 2009.

[19] A. Taylor and D. J. Higham. CONTEST: A controllable test matrix toolbox for MATLAB. *ACM Trans. Math. Software*, 35(4):26:1–26:17, 2009.

[20] C. F. Van Loan. The sensitivity of the matrix exponential. *SIAM J. Numer. Anal.*, 14(6):971–981, 1977.

[21] R. C. Ward. Numerical computation of the matrix exponential with accuracy estimate. *SIAM J. Numer. Anal.*, 14(4):600–610, 1977.

[22] D. S. Watkins. A case where balancing is harmful. *Electron. Trans. Numer. Anal.*, 23:1–4, 2006.

[23] J. H. Wilkinson. *The Algebraic Eigenvalue Problem*. Oxford University Press, 1965.

[24] J. Xue and E. Jiang. Entrywise relative perturbation theory for nonsingular M-matrices and applications. *BIT*, 35(3):417–427, 1995.

[25] J. Xue and Q. Ye. Entrywise relative perturbation bounds for exponentials of essentially non-negative matrices. *Numer. Math.*, 110(3):393–403, 2008.

[26] J. Xue and Q. Ye. Computing exponentials of essentially non-negative matrices entrywise to high relative accuracy. *Math. Comp.*, to appear.

ISSN 0348-0542
ISBN 978-91-7459-430-0
UMINF 12.07

Department of Computing Science
Umeå University, SE-901 87 Umeå
www.cs.umu.se