Parallel Algorithms and Library Software for the Generalized Eigenvalue Problem on Distributed Memory Computer Systems

Björn Adlerborn

Licentiate Thesis

DEPARTMENT OF COMPUTING SCIENCE
UMEÅ UNIVERSITY, SWEDEN

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

adler@cs.umu.se

Copyright © 2016 by Björn Adlerborn
Except Paper I, © SIAM J. Scientific Computing, 2015
Paper II, ©Björn Adlerborn, Bo Kågström, and Lars Karlsson, 2016
Paper III, ©Björn Adlerborn, Bo Kågström, and Daniel Kressner, 2015

ISBN 978-91-7601-491-2
ISSN 0348-0542
UMINF 16.11

Printed by Print & Media, Umeå University, 2016

Abstract

We present and discuss algorithms and library software for solving the generalized non-symmetric eigenvalue problem (GNEP) on high performance computing (HPC) platforms with distributed memory. Such problems occur frequently in computational science and engineering, and our contributions make it possible to solve GNEPs fast and accurate in parallel using state-of-the-art HPC systems. A generalized eigenvalue problem corresponds to finding scalars λ and vectors x such that $Ax = \lambda Bx$, where A and B are real square matrices. A nonzero x that satisfies the GNEP equation is called an eigenvector of the ordered pair (A,B), and the scalar λ is the associated eigenvalue. Our contributions include parallel algorithms for transforming a matrix pair (A,B) to a generalized Schur form (S,T), where S is quasi upper triangular and T is upper triangular. The eigenvalues are revealed from the diagonals of S and T. Moreover, for a specified set of eigenvalues an associated pair of deflating subspaces can be computed, which typically is requested in various applications. In the first stage the matrix pair (A,B) is reduced to a Hessenberg-triangular form (H,T), where H is upper triangular with one nonzero subdiagonal and T is upper triangular, in a finite number of steps. The second stage reduces the matrix pair further to generalized Schur form (S,T) using an iterative QZ-based method. Outgoing from a one-stage method for the reduction from (A,B) to (H,T), a novel parallel algorithm is developed. In brief, a delayed update technique is applied to several partial steps, involving low level operations, before associated accumulated transformations are applied in a blocked fashion which together with a wave-front task scheduler makes the algorithm scale when running in a parallel setting. The potential presence of infinite eigenvalues makes a generalized eigenvalue problem ill-conditioned. Therefore the parallel algorithm for the second stage, reduction to (S,T) form, continuously scan for and robustly deflate infinite eigenvalues. This will reduce the impact so that they do not interfere with other real eigenvalues or are misinterpreted as real eigenvalues. In addition, our parallel iterative QZ-based algorithm makes use of multiple implicit shifts and an aggressive early deflation (AED) technique, which radically speeds up the convergence. The multi-shift strategy is based on independent chains of so called coupled bulges and computational windows which is an important source of making the algorithm scalable. The parallel algorithms have been implemented in state-of-the-art library software. The performance is demonstrated and evaluated using up to 1600 CPU cores for problems with matrices as large as 100000×100000. Our library software is described in a User Guide. The software is, optionally, tunable via a set of parameters for various thresholds and buffer sizes etc. These parameters are discussed, and recommended values are specified which should result in reasonable performance on HPC systems similar to the ones we have been running on.

Preface

The Licentiate Thesis consists of the the following three papers and an introduction including a summary of the papers.

Paper I	Björn Adlerborn, Bo Kågström, and Daniel Kressner. A parallel QZ algorithm for distributed memory HPC systems[1]. In *SIAM J. Scientific Computing*, 36(5), pages 480–503. 2015.
Paper II	Björn Adlerborn, Bo Kågström, and Lars Karlsson. Distributed One-Stage Hessenberg-Triangular Reduction with Wavefront Scheduling. *Report UMINF* 16.10. Dept. of Computing Science, Umeå University, Sweden, 2016 (to be submitted).
Paper III	Björn Adlerborn, Bo Kågström, and Daniel Kressner. PDHGEQZ User Guide. *Report UMINF* 15.14. Dept. of Computing Science, Umeå University, Sweden, 2015.

[1] Reprinted by permission of *Society for Industrial and Applied Mathematics.*

Acknowledgements

First of all, I would like to thank my supervisors Bo Kågström and Lars Karlsson, for their great enthusiasm, inspiration, and encouragement, and for providing exceptional knowledge and support, and for always making so much of their time available.

Thanks go to Meiyue Shao, Daniel Kressner and Robert Granat for fruitful discussions on parallel QZ algorithms.

Thanks also to the colleagues and staff at the High Performance Computer Center North (HPC2N), for access to the HPC systems Abisko and Akka and for their excellent user support.

I also would like thank Anna for being there for me, making me a better person, making my life valuable, and giving birth to and being such a great mother and inspiration for our dearest daughter Emelie.

Financial support has been provided by the Swedish Research Council (VR) under grant A0581501, and by eSSENCE, a strategic collaborative e-Science programme funded by the Swedish Government via VR. This work was also partly funded from the European Unions Horizon 2020 research and innovation programme under the NLAFET grant agreement No 671633.

Umeå, May 2016

Björn Adlerborn

Contents

Chapter 1

Introduction

Solving large-scale problems efficiently and effectively on modern parallel high performance computing (HPC) platforms requires both a good parallel algorithm for the considered problem as well as very good knowledge of the underlying computer architecture. To facilitate the use of such parallel HPC systems, it is important to provide various software tools so that engineers and scientists can focus on solving their applications. With long tradition, numerical software libraries that include ready to use computational routines provide a well-functioning tool for this purpose. In this way, the burden of handling the architecture issues is mainly laid on the developers of parallel algorithms and software.

Today's parallel HPC architectures are hierarchical and are getting more and more heterogeneous in several dimensions (e.g., multicore processors, accelerators, high-speed interconnect networks). In this thesis, we focus on distributed memory architectures with multicore nodes, but common for all parallel HPC systems is that they have a complex memory hierarchy that must be utilized and given special attention in order to obtain a high portion of the theoretical peak performance. How to succeed is highly dependent on what problems to solve. Some of them lead to algorithms that are straightforward to parallelize, while many have strong dependencies between computational steps and associated data flows; the latter is the case for the dense matrix computational problems studied in this thesis.

One paramount issue concerns the management of complex memory hierarchies, which aim at avoiding unnecessary data movements between memory layers defined by on chip multi-level caches and local as well as remote memory. In practice, this means that matrix elementwise computations are restructured (or new algorithms are designed) so that blocked (submatrix) operations are used as much as possible. Ideally, if most computations can be expressed as matrix-matrix operations, it makes it possible to reuse data as much as possible at the different memory layers and thereby obtain near to optimal performance. Equally important is to balance the computational load (defined by tasks) across all participating processes, keep them active and avoid them from going idle. To restructure and rebalance the computational load during execution, which even may include redundant computations, can be an efficient way of reducing communication and idle time.

In this thesis, we investigate and propose parallel algorithms for solving the generalized non-symmetric eigenvalue problem (GNEP)

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

for dense real square matrices A and B, using a two-stage method, executing on distributed memory machines. GNEPs emerge frequently, for example when solving differential-algebraic equations, in model reductions, and in the linearization of (non-linear) quadratic eigenvalue problems, and boil down to finding eigenvalues, eigenvectors and deflating subspaces of a general matrix pair (A, B). In many applications, e.g., in control system design and analysis, eigenvectors are not needed and it is enough to know a pair of deflating subspaces associated with a specified spectrum. An example is stable subspaces associated to all eigenvalues within the unit circle (or in the left complex plane).

When B is nonsingular, equation (1.1) can be transformed to a standard eigenvalue problem $Cx = \lambda Ix$ with $C = B^{-1}A$, but this is not recommended since if B is close to singular (i.e. ill-conditioned) the computation of C may affect the conditioning of other well-conditioned finite eigenvalues. Moreover, if B is a singular matrix, the GNEP has one or several infinite eigenvalues and in finite precision arithmetic there is a big risk that large finite and infinite eigenvalues are mixed up. Therefore, in practice the GNEP formulation is kept and (A, B) is treated as a matrix pair in all computations.

The two-stage method, illustrated in Figure 1 for 10×10 matrices, first reduces the matrix pair to an upper Hessenberg-triangular form (H, T), where H is an upper Hessenberg matrix (has one nonzero subdiagonal below the main diagonal) and T is an upper triangular matrix. The second stage further reduces the pair (H, T) to generalized real Schur form (S, T), where S is an upper quasi-triangular matrix, possibly with 2×2 blocks along the diagonal, and T remains upper triangular. Each 2×2 diagonal block of (S, T) corresponds to a complex conjugate pair of eigenvalues and each 1×1 block $(s_{i,i}, t_{i,i})$ corresponds to a finite eigenvalue $\lambda_i = s_{i,i}/t_{i,i}$ for $t_{i,i} \neq 0$ and to an infinite eigenvalue ∞ when $t_{i,i} = 0$. The two-stage reductions are performed using *novel parallel two-sided transformation based algorithms*, where each algorithm computes two sequences of matrix transformations that are applied to the matrix pair (A, B) from left and right, respectively.

1.1 Background

Already in 1973, Moler and Stewart [24] presented a two-stage transformation based algorithm, that follow the description above, for solving the dense generalized eigenvalue problem. In the first stage, the matrix pair (A, B) is

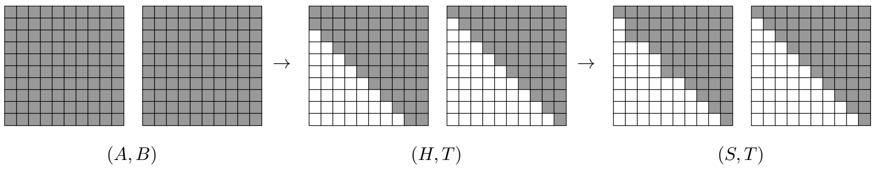

Figure 1: Reduction of a general matrix pair (A, B) to generalized real Schur form (S, T) using a two-stage method. In the first stage, (A, B) is reduced to upper Hessenberg-triangular form (H, T). In the second stage, the pair (H, T) is further reduced to (S, T) with 1×1 and 2×2 blocks along the main diagonal, corresponding to infinite or real eigenvalues and complex conjugate pair of eigenvalues, respectively.

reduced to HT form in a finite number of steps; the columns of A and B are reduced from left to right where a crucial step is to remove the undesirable fill-in caused by left and right transformations. In the second stage, the pair is further reduced to generalized real Schur form (S, T) by an iterative method, known as the *QZ algorithm*, which is a generalization and extension of the *QR algorithm*, independently proposed by Francis [15] and Kublanovskaya [23] for the standard eigenvalue problem. Throughout the years, several improvements have been proposed, and here follows a brief description of the most relevant for our study.

A cache-blocked approach was proposed by Dackland and Kågström, [12],[13], dividing the HT reduction into two separate sub-stages, first to a block HT form (H-part has several nonzero subdiagonals, algorithm is rich in matrix-matrix operations), followed by a procedure based upon Givens rotations that completes the reduction to proper upper HT form. Another one-stage cache-blocked approach using mainly level-3 BLAS, i.e. matrix-matrix operations, was proposed by Kågström, Kressner, E.S. Quintana-Ortí, and G. Quintana-Ortí [22]. The latter showed that dividing the HT reduction into two separate stages is not always better than to keep it as a single stage.

A blocked approach for the QZ algorithm, was proposed by Dackland and Kågström [14], that showed a speedup of 2–5 obtained over the unblocked algorithm. The QR algorithm, used to reduce a Hessenberg matrix to real Schur form when solving the standard eigenvalue problem, extended with improved use of several shifts and a technique for speeding up convergence, called aggressive early deflation(AED), was proposed by Braman, Byers, and Mathias [10, 11]. The multishift and AED techniques were later extended to the QZ algorithm by Kågström and Kressner [21], which greatly increased the performance when comparing with previous blocked and unblocked versions.

My master thesis, a parallel formulation for the reduction from a block HT form to proper upper HT form, together with [13] formed the first, to our knowledge, parallel reduction of a general matrix pair (A, B) to HT form, see [1]. This parallel two-staged HT reduction approach shows that although the first stage involves much more flops than the second stage, the latter dominates the overall execution time. However, the second stage scales somewhat better than the first stage.

In [2, 4], we propose a parallel solution for the complete reduction of a general matrix pair to generalized Schur form. The parallel QZ reduction stage is based on and extend [14] but, despite having a more favorable flop count, it dominates the total execution time, leaving room for improvements. However, this implementation is significantly faster than the corresponding parallel QR implementation [20], despite that the flop count for QZ is about two times more than for QR. This is explained by the preliminary use of AED, efficient multishift strategies and accumulated updates applied in a matrix-matrix manner. However, the parallel QR algorithm did later undergo a complete revision, adopting the new efficient multishift strategies and AED, resulting in an efficient parallel solver on (hybrid) distributed memory machines, see [18, 19].

The Papers I–III, in this thesis, present further novel contributions (algorithms and software) to the parallel solution of the generalized eigenvalue problem.

1.2 Data distribution

The target parallel platform is distributed memory machines, where each process has it own set of memory, and the problem is split among the participating processes. However, in practice and in most modern HPC systems, the processes are grouped into compute units which have some shared memory, leading to so called hybrid distributed memory machines. The programming model used in this thesis is message passing and any available shared memory at the CPU nodes is treated and allocated as separate memory units to each process.

Using a two-dimensional block-cyclic data distribution schema of the matrices A and B ensures that each process has a part, of roughly the same size, of the problem data and the computational workload. The P processes are logically arranged into to a $P_\mathrm{r} \times P_\mathrm{c}$ grid, not necessarily square. Moreover, the $N \times N$ matrices A and B are partitioned into $N_\mathrm{b} \times N_\mathrm{b}$ sized blocks, and scattered cyclically across the $P_\mathrm{r} \times P_\mathrm{c}$ grid, see Figure 2 for an illustration.

ScaLAPACK [9], a state-of-the-art library of high-performance linear algebra routines for parallel distributed memory machines, uses the two-dimensional block cyclic distribution schema, generalized such that the data blocks need not be square, where each distributed object is represented by two objects—a pointer to the local data and a globally defined descriptor to define the partitioning and a communication context.

ScaLAPACK is based upon, and includes, the software package BLACS for handling communication and offers point-to-point non blocking send and blocking receive, as well as broadcast send and receive and global summation

(0,0)	(0,1)	(0,2)	(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)	(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)	(2,0)	(2,1)	(2,2)
(0,0)	(0,1)	(0,2)	(0,0)	(0,1)	(0,2)
(1,0)	(1,1)	(1,2)	(1,0)	(1,1)	(1,2)
(2,0)	(2,1)	(2,2)	(2,0)	(2,1)	(2,2)

Figure 2: Block cyclic data distribution of a matrix exemplified using a 3×3 grid, where the residence for each of the 36 matrix blocks is given by its process coordinate (p_r, p_c) with $0 \leq p_r < 3 = P_r$ and $0 \leq p_c < 3 = P_c$.

routines. Our algorithms and software are designed to fit into the ScaLAPACK suite of routines, and is therefore restricted to use the communication primitives offered by BLACS. The lack of a non blocking receive limits the degree of algorithmic freedom a bit, but in general, communication via BLACS and operating on distributed objects in ScaLAPACK is straightforward.

Operations on distributed data typically requires communication, for example, when one process holds part of data that other processes need to complete an operation. In order to reduce communication, the data can be reorganized, temporarily, to a subgrid before the computation is performed. Using less processes, means that each process that participates in the computation, will own more data, and potentially reduce the need for communication. After completing the computation, data is typically restored to the original grid to continue using all allocated processes for the remaining computation. ScaLAPACK provides a redistribution routine, and is used in our software, for example, to perform parallel AED, where the AED computational window is spread over several processes. Our heuristics show that it is beneficial to use less computational power by performing the AED computation on a subgrid. The redistribution routine is efficient, but should be used with caution as it increases the memory load, and de facto decreases the core utilization leading to poor scalability in the long run.

1.3 Memory hierarchies and operations

The memory hierarchy of a typical CPU core consists of registers, levels of caches and main memory. However, all computations are performed at the very top of the hierarchy, i.e. in the registers, and an operation on data stored in the main memory requires data to be transferred and copied all the way up to the top. Once stored at the top, we should try to utilize this data as much as possible before copying another set of data to work on, as this copy procedure is time consuming; each memory level has a latency and a per item cost associated to a memory transfer. Put into practice, (cache-)blocked code is used, where the software is written in such a way that it works on optimized sized chunks of data, performing several operations on one chunk at a time, before the next chunk is addressed. As there are often several layers of cache memories, of different sizes, several layers of blocked code are also common. Instead of performing several elementwise operations, operations can often be bundled and applied in an accumulated way. A simple example is a list of numbers added to each element of a vector; instead of adding numbers individually to the vector, compute the sum once, and then add the sum to the elements of the vector. Another example is the usage of accumulated Givens rotations. A Givens rotation is an orthogonal matrix $G \in \mathbb{R}^{n \times n}$ that applied to another matrix makes a rotation with angle θ in the plane (i, j) spanned by two coordinates axes (here $j = i + 1$):

$$G_{ij}(\theta) = \left[\begin{array}{c|cc|c} I_{i-1} & & & \\ \hline & c & s & \\ & -s & c & \\ \hline & & & I_{n-j} \end{array}\right],$$

where $c^2 + s^2 = 1$ $(c = cos(\theta), s = sin(\theta))$ and I is the identity matrix of size $(i-1) \times (i-1)$ and $(n-j) \times (n-j)$, respectively. The main use of Givens rotations in numerical linear algebra is to zero out, *annihilate*, elements in vectors or matrices: given a, b find c and s such that

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix},$$

and $r > 0$ does not over- or underflow. We use this annihilation technique to reduce a matrix pair (A, B) to HT-form, where several Givens rotations are bundled, i.e. accumulated into one matrix and subsequently applied to blocks in (A, B). This bundling process enables more coarse-grained matrix-matrix operations which greatly improve the arithmetic performance compared to applying the rotations one by one.

1.4 Redundant computing

Increasing the amount of arithmetic work and let a few processes repeat and do the same work can often be beneficial from a total execution time point of view. Consider the matrix operation

$$U \cdot V,$$

where $U \in \mathbb{R}^{m\times m}, V \in \mathbb{R}^{m\times n}$. Partition $V = \left[\begin{array}{c} V_0 \\ V_1 \end{array}\right]$ and let it be distributed over a 2×1 process grid (most likely a subgrid), such that process p_0 holds V_0 and process p_1 holds V_1. We assume that U is stored on both p_0 and p_1.

To perform the operation, p_0 and p_1 need to exchange data. One approach is to let p_0 receive V_1 from p_1, perform the matrix multiplication, and then send updated V_1 back. p_1 will be idle during the computation, waiting for the updated V_1. In a non blocking receive communication environment, p_1 could however perform other tasks, that are independent of V_1. Another approach is to let p_0 and p_1 exchange data such that both have enough to perform the complete operation. This requires some extra workspace, but the upside is that the exchange can be performed, almost perfect, in parallel; both perform non blocking send of their parts of V, and enter the receive mode to receive the data, which is already on the way. Both compute $U \cdot V$, but only save the part of the product they own, that is p_1 discards updated V_0, and vice versa for p_0. This technique requires extra work but has one synchronization point less and reduce idling processes, and is successfully used in our two-stage reduction of a matrix pair (A, B) to generalized Schur form, for example, when applying bundled Givens operations.

Chapter 2

Summary of papers

In the following, a brief summary of each paper in the thesis is given.

2.1 Paper I

Paper I [5] concerns the parallel reduction of a matrix pair in Hessenberg, triangular form (H, T) to generalized real Schur form (S, T). The paper begins with an overview of the generalized Schur decomposition and the structure of the QZ algorithm before moving on to discussing the multishift and AED techniques, with focus on aspects related to the parallel algorithms and implementations. The potential presence of infinite eigenvalues in the generalized eigenvalue problem makes a fundamental difference compared to the standard one. Infinite eigenvalues need to be dealt with, i.e. identified and deflated, before other actions are taken in order to preserve them as infinite, or not having them inflicting damage to other eigenvalues, due to round off errors. A serial and a novel parallel algorithm are discussed and exemplified where the infinite eigenvalues are moved to the top-left or bottom-right corner of the matrix pair, whichever is nearest.

Performance is evaluated using several different problems, on two different parallel HPC systems. Interesting and applicable real world benchmark examples from Matrix Market [8], some with a large fraction of infinite eigenvalues, together with constructed problems of three different types demonstrate execution times for different grid configurations. The problem size n ranges from 4000 to 32000, and up to 100 cores are utilized to solve the problems in parallel, demonstrating increasing speedup as the problem size and number of cores increases. These problems are however rather small in a context of what modern HPC systems are capable of, so in order to utilize more compute power, a constructed 100000×100000 benchmark problem is solved in parallel using up to 1600 cores, and performance is evaluated and compared with the parallel solver for standard eigenvalue problems [19] for a similar problem. Even though the standard eigenvalue problem requires less than half of the number of operations to complete compared to the generalized eigenvalue problem [17], execution time ratios show that our solver takes substantially less than twice the time to complete.

2.2 Paper II

Paper II [3] concerns the parallel reduction of a general matrix pair (A, B) to Hessenberg-triangular form (H, T). Based on the sequential cache-blocked algorithm [22], this parallel formulation makes use of Givens rotations to reduce the matrix pair with a novel static wavefront scheduling algorithm. The sequential algorithm and its blocking strategy are briefly described, before moving on to a discussion on how different parts of the algorithm have been redesigned to work in parallel. Since a straightforward parallelization strategy proves to be poorly scalable, due to a high fraction of idle time among the participating processors, a scheduler is implemented with the aim to maximize process utilization and execute the shortest possible sequence of parallel steps. At each step, the scheduler select a parallel task to execute such that

- the degree of parallelism is maximized,
- tasks with more remaining work is chosen over tasks with less work.

Two different HPC systems are used to evaluate the parallel performance; weak and strong scaling is measured, visualized and discussed. Results, using up to 961 mpi-processes, indicate that our implementation scales but suffers from bottlenecks, related to synchronization points, in two of its major subroutines.

2.3 Paper III

Paper III [6] is a User Guide for the `PDHGEQZ` software; library software routines to solve the generalized eigenvalue problem for dense and real matrix pairs (A, B) in parallel on multicore HPC systems. The guide mainly describes software and parameters related to Paper I, but also includes a description of routines related to Paper II and routines from other earlier work. Installation and building instructions, for a Linux like system, are presented, followed by a software hierarchy overview of how routines are related and called.

The calling sequences for the main driver routines with input and output parameters are described in detail. Moreover, the set of tunable parameters and a description of their usage and default values are discussed. The default value for parameters may need tuning to reach the best possible performance of the `PDHGEQZ` software executing on a new target architecture, however, the defaults should give reasonable performance on systems similar to the ones we have been running on.

During the build process, internal tests are performed to make sure the software work as intended. Both sequential and parallel tests are performed, with validation of the computed results. System software requirements are listed so users can prepare their systems before the build process and installation is initiated.

Chapter 3

Future work

Our solution to the generalized eigenvalue problem provides eigenvalues and deflating subspaces, but presently does not compute eigenvectors. Given a matrix pair in generalized Schur form, LAPACK [7] offers serial routines for computing both left and right eigenvectors. Combining those with the accelerating techniques proposed by Gates et al. in [16] and our own experiences will be a good base for formulating a parallel distributed memory algorithm.

Our parallel algorithms for computing the generalized Schur form scale with the number of processors, but there is room for improvements. A great challenge is to develop novel architecture-aware algorithms that expose as much parallelism as possible in today's and future extreme-scale HPC systems. This and many other challenges will be investigated within the Horizon 2020 project *Parallel Numerical Linear Algebra for Future Extreme-Scale Systems* with acronym NLAFET, coordinated by Umeå University. The NLAFET overall aim is to enable a radical improvement in the performance and scalability of a wide range of real-world applications relying on linear algebra software for future extreme-scale systems. For more information see the NLAFET website: `http://www.nlafet.eu`

Bibliography

[1] B. Adlerborn, K. Dackland, and B. Kågström. Parallel two-stage reduction of a regular matrix pair to Hessenberg-Triangular form. In T. Sørevik, F. Manne, A. H. Gebremedhin, and R. Moe, editors, *Applied Parallel Computing, PARA 2000*, LNCS 1947, pages 92–102. Springer Berlin Heidelberg, 2000.

[2] B. Adlerborn, K. Dackland, and B. Kågström. Parallel and blocked algorithms for reduction of a regular matrix pair to Hessenberg-Triangular and generalized Schur forms. In J. Fagerholm, J. Haataja, J. Järvinen, M. Lyly, P. Råback, and V. Savolainen, editors, *Applied Parallel Computing, PARA 2002*, LNCS 2367, pages 319–328. Springer-Verlag, 2002.

[3] B. Adlerborn, L. Karlsson, and B. Kågström. Distributed One-Stage Hessenberg-Triangular Reduction with Wavefront Scheduling. *Report UMINF* 16.10, Dept. of Computing Science, Umeå University, Sweden, 2016.

[4] B. Adlerborn, B. Kågström, and D. Kressner. Parallel variants of the multishift QZ algorithm with advanced deflation techniques. In B. Kågström, E. Elmroth, J. Dongarra, and J. Waśniewski, editors, *Applied Parallel Computing, PARA 2006*, LNCS 4699, pages 117–126. Springer Berlin Heidelberg, 2006.

[5] B. Adlerborn, B. Kågström, and D. Kressner. A Parallel QZ Algorithm for distributed memory HPC-systems. *SIAM J. Sci. Comput.*, 36(5):C480–C503, 2014.

[6] B. Adlerborn, B. Kågström, and D. Kressner. PDHGEQZ User Guide. *Report UMINF* 15.12, Dept. of Computing Science, Umeå University, Sweden, 2015.

[7] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK Users' Guide.* SIAM, Philadelphia, PA, third edition, 1999.

[8] Z. Bai, D. Day, J. W. Demmel, and J. 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, Knoxville, TN, USA, March 1997. Also available online from `http://math.nist.gov/MatrixMarket`.

[9] 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 Users' Guide*. SIAM, Philadelphia, PA, 1997.

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

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

[12] K. Dackland and B. Kågström. Reduction of a Regular Matrix Pair (A, B) to Block Hessenberg Triangular Form. In J. Dongarra, K. Madsen, and J. Waśniewski, editors, *Applied Parallel Computing, PARA 1995*, LNCS 1041, pages 125–133. Springer Berlin Heidelberg, 1995.

[13] K. Dackland and B. Kågström. A ScaLAPACK-Style Algorithm for Reducing a Regular Matrix Pair to Block Hessenberg-Triangular Form. In B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski, editors, *Applied Parallel Computing, PARA 1998*, LNCS 1541, pages 95–103. Springer Berlin Heidelberg, 1998.

[14] K. Dackland and B. Kågström. Blocked algorithms and software for reduction of a regular matrix pair to generalized Schur form. *ACM Trans. Math. Software*, 25(4):425–454, 1999.

[15] J. G. F. Francis. The QR Transformation. A Unitary Analogue to the LR Transformation - Part 1. *The Computer Journal*, 4(3):265–271, 1961.

[16] M. Gates, A. Haidar, and J. Dongarra. Accelerating computation of eigenvectors in the dense nonsymmetric eigenvalue problem. In M. Daydé, O. Marques, and K. Nakajima, editors, *High Performance Computing for Computational Science, VECPAR 2014*, LNCS 8969, pages 182–191. Springer International Publishing, 2015.

[17] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 4th edition, 2012.

[18] 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.

[19] R. Granat, B. Kågström, D. Kressner, and M. Shao. Parallel library software for the multishift QR algorithm with aggressive early deflation. *ACM Trans. Math. Software*, 41(4), 2015.

[20] 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.

[21] B. Kågström and D. Kressner. Multishift variants of the QZ algorithm with aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 29(1):199–227, 2006.

[22] 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.

[23] V.N. Kublanovskaya. On some algorithms for the solution of the complete eigenvalue problem. *USSR Computational Mathematics and Mathematical Physics*, 1(3):637 – 657, 1962.

[24] C. B. Moler and G. W. Stewart. An algorithm for generalized matrix eigenvalue problems. *SIAM J. Numer. Anal.*, 10:241–256, 1973.