On aggressive early deflation in parallel variants of the QR algorithm

By

Bo Kågström, Daniel Kressner, and Meiyue Shao

UMINF-10/13

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
Sweden

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

* 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 and by the Swedish Foundation for Strategic Research under grant A3 02:128.

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 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 7) plays a crucial role in attaining good performance, see [6, 10, 17] for details.

Algorithm 1 *Multishift Hessenberg QR Algorithm with AED*

1. **WHILE** not converged
2. Perform AED on the $n_{\mathsf{win}} \times n_{\mathsf{win}}$ trailing principle submatrix.
3. Apply the accumulated orthogonal transformation to the corresponding off-diagonal blocks.
4. **IF** enough eigenvalues have been deflated in step 2
5. **GOTO** step 2.
6. **END IF**
7. Perform a multishift QR sweep with undeflatable eigenvalues from Step 2 as shifts.
8. Check for negligible subdiagonal elements.
9. **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

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

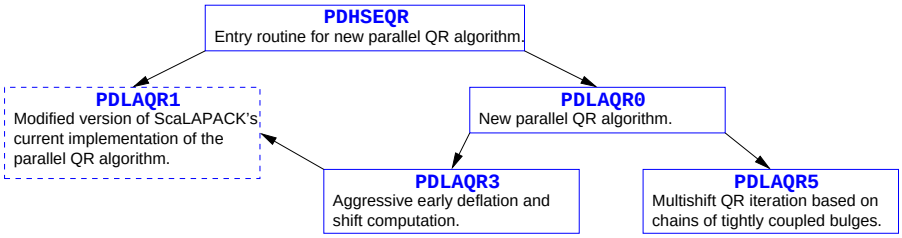
where $s \in \mathbb{R}^{n_{\mathsf{win}}}$ is the so called spike, created from the subdiagonal entry contained in H_{32}. The eigenvalues of T are checked subsequently for convergence and possibly deflated. The eigenvalue (or 2×2 block) in the bottom right corner of T can be deflated if the magnitude of the last component (or the last two components) of the spike is negligibly small. Undeflatable eigenvalue 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`,

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



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 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`*)*

1. **WHILE** not converged
2. 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.
3. Apply the accumulated orthogonal transformations to the corresponding off-diagonal blocks in parallel.
4. **IF** enough eigenvalues have been deflated
5. **GOTO** step 2.
6. **END IF**
7. Compute the eigenvalues of a trailing submatrix.
8. Perform a pipelined QR sweep with the eigenvalues computed in step 7 as shifts.
9. Check for negligible subdiagonal elements.
10. **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 has a non-negligible positive impact on the total execution time, 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

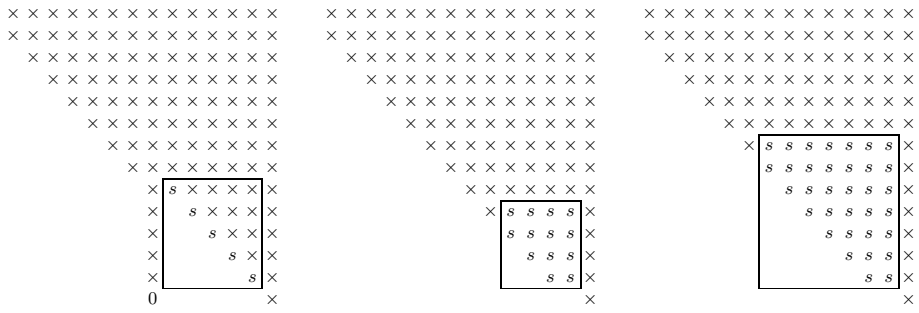
values of n_{win} returned by `IPARMQ` are 96, 192 and 384, which is much larger than if we chose `NS*3/2`. Based on the observation that the optimal AED window size does not depend strongly on the number of shifts used in the QR sweeps, we prefer to stick to large n_{win} rather than using `NS*3/2`. 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` $\times$ `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`)



(1) Using undeflatable eigenvalues

(2) Using eigenvalues of an `NS` $\times$ `NS` window

(3) Using eigenvalues of an $(n_{\mathsf{win}}+1) \times (n_{\mathsf{win}}+1)$ window

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

The new implementation of `PDLAQR1` turns out to require much less 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 results is given by

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

$n = 220\sqrt{p}$, which fits the observations reasonably well and has been incorporated in our implementation.

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%. 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 take utility of 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.

References

1. B. Adlerborn, B. Kågström and D. Kressner, Parallel variants of the multishift QZ algorithm with advanced deflation techniques, B. Kågström et al. (eds): PARA 2006, Lecture Notes in Computer Science, LNCS 4699, pp. 117–126, 2007.
2. 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 User's Guide*, 3rd ed., SIAM, Philadelphia, PA, 1999.
3. Z. Bai and J. W. Demmel, On a block implementation of Hessenberg multishift QR iteration, *Intl. J. of High Speed Comput.*, 1:97-112, 1989.
4. Z. Bai and J. W. Demmel, On swapping diagonal blocks in real Schur form, *Linear Algebra Appl.*, 186:73-95, 1993.
5. 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.
6. 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.
7. 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.
8. 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.
9. G. 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.
10. 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. (An earlier version appeared as LAPACK Working Note #216, 2009).
11. R. Granat, B. Kågström and D. Kressner, Parallel eigenvalue reordering in real Schur forms, *Concurrency and Computation: Practice and Experience*, 21(9):1225-1250, 2009.
12. GOTO-BLAS – high-performance BLAS by Kazushige Goto. See http://www.tacc.utexas.edu/tacc-projects/#blas.
13. G. Henry and R. van de Geijn, Parallelizing the QR algorithm for the nonsymmetric algebraic eigenvalue problem: Myths and reality. *SIAM J. Sci. Comput.*, 17:870-883, 1997.

14. 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.
15. D. Kressner. *Numerical Methods for General and Structured Eigenvalue Problems*, volume 46 of *Lecture Notes in Computational Science and Engineering*. Springer, Heidelberg, 2005.
16. D. Kressner, The effect of aggressive early deflation on the convergence of the QR algorithm. *SIAM J. Matrix Anal. Appl.*, 30(2):805-821, 2008.
17. B. Lang. Effiziente Orthogonaltransformationen bei der Eigen- und Singulärwertzerlegung. Habilitationsschrift, 1997.
18. D. S. Watkins. *The matrix eigenvalue problem: GR and Krylov subspace methods*. SIAM, Philadelphia, PA, 2007.
19. David S. Watkins, Francis's algorithm. *Amer. Math. Monthly*, 2010. To appear.