ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

MATHICSE

Mathematics Institute of Computational Science and Engineering

School of Basic Sciences - Section of Mathematics

MATHICSE Technical Report

Nr. 30.2012 **NEW**

August 2012 (June 13. 2013)


# Optimally packed chains of bulges in multishift QR algorithms

L. Karlsson, D. Kressner, B. Lang

http://mathicse.epfl.ch

Phone: +41 21 69 37648

Address:
EPFL - SB - MATHICSE (Bâtiment MA)
Station 8 - CH-1015 - Lausanne - Switzerland

Fax: +41 21 69 32545

# Optimally packed chains of bulges in multishift QR algorithms*

Lars Karlsson† Daniel Kressner‡ Bruno Lang§

June 13, 2013

**Abstract**

The QR algorithm is the method of choice for computing all eigenvalues of a dense nonsymmetric matrix $A$. After an initial reduction to Hessenberg form, a QR iteration can be viewed as chasing a small bulge from the top left to the bottom right corner along the subdiagonal of $A$. To increase data locality and create potential for parallelism, modern variants of the QR algorithm perform several iterations simultaneously, which amounts to chasing a chain of several bulges instead of a single bulge. To make effective use of level 3 BLAS, it is important to pack these bulges as tightly as possible within the chain. In this work, we show that the tightness of the packing in existing approaches is not optimal and can be increased. This directly translates into a reduced chain length by 33% compared to the state-of-the-art LAPACK implementation of the QR algorithm. To demonstrate the impact of our idea, we have modified the LAPACK implementation to make use of the optimal packing. Numerical experiments reveal a uniform reduction of the execution time, without affecting stability or robustness.

## 1 Introduction

The real Schur decomposition of a general matrix $A \in \mathbb{R}^{n\times n}$ takes the form

$$Q^T A Q = T, \tag{1}$$

where $Q \in \mathbb{R}^{n\times n}$ is orthogonal and $T \in \mathbb{R}^{n\times n}$ is block upper triangular with each diagonal block having size $1 \times 1$ or $2 \times 2$. The $1 \times 1$ diagonal blocks correspond to real eigenvalues of $A$, while the $2 \times 2$ diagonal blocks correspond to complex conjugate pairs of eigenvalues. The goal of the QR algorithm is to compute such a Schur decomposition.

After an initial reduction of $A$ to a Hessenberg matrix $H$ [8, Section 7.4], the QR algorithm iteratively reduces $H$ further to the desired Schur form. Given $m \ll n$ shifts $\sigma_1, \ldots, \sigma_m \in \mathbb{C}$ closed under complex conjugation, a QR iteration consists of a QR factorization

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

*Financial support has been provided by the Swedish Research Council (grant VR 7062571), UMIT Research Lab via Balticgruppen, and eSSENCE, a strategic collaborative eScience programme funded by the Swedish Research Council.

†Department of Computing Science, Umeå University, Sweden, larsk@cs.umu.se

‡ANCHP, MATHICSE, EPF Lausanne, Switzerland, daniel.kressner@epfl.ch

§Department of Mathematics and Computer Science, University of Wuppertal, Germany, lang@math.uni-wuppertal.de

and the update $H \leftarrow Q^T H Q$. Equivalently, a QR iteration can be viewed as creating an $(m+1) \times (m+1)$ bulge in the top left corner of the Hessenberg matrix and chasing it downwards along the subdiagonal until the bulge disappears at the bottom right corner. This bulge chasing mechanism was already explained in Francis' original description [7] of the QR algorithm; see, e.g., [13, 18] for more recent accounts.

Originally, the number of shifts in the QR algorithm was chosen to be tiny, say $m = 1$ or $m = 2$. However, such a choice comes with the disadvantage that a single QR iteration performs relatively few operations while still accessing the whole matrix, leading to a performance that is bound by memory bandwidth. In theory, this poor ratio between floating point operations (flops) and memory accesses can be improved by increasing $m$. This has been suggested by Bai and Demmel [3], who also observed that the convergence deteriorates in finite-precision arithmetic if $m$ is chosen too large, leading to the default choice $m = 6$ in the initial LAPACK [1] implementation of the QR algorithm. This undesirable phenomenon is called shift blurring: The relation between the shifts and the bulges becomes exponentially ill-conditioned as $m$ increases [17, 14].

A numerically more suitable alternative to increasing $m$ is to chase not only one but several small bulges simultaneously. This allows for the use of a large number of shifts without suffering from shift blurring. Braman, Byers, and Mathias [4] as well as Lang [15] proposed to chase a tightly packed chain of bulges. In effect, most of the operations performed during the QR iterations can be rephrased effectively in terms of matrix–matrix multiplications. Improved data locality and being able to make use of highly optimized level 3 BLAS lead to significant performance improvements for larger matrices, despite the fact that an increased number of flops is needed. To limit this increase, it is important to pack the bulges as tightly as possible. They should, however, not start interfering with each other, which would compromise the convergence of the QR algorithm. For $m = 2$, the packing proposed in [4] arranges $n_\mathsf{b}$ bulges (and thus $2n_\mathsf{b}$ shifts) in a chain of length $3n_\mathsf{b}$. Asymptotically, such a packing leads to a flops increase of 140%.

In this paper we show that the bulges can be packed even tighter. For $m = 2$, our packing arranges $n_\mathsf{b}$ bulges in a chain of length $2n_\mathsf{b} + 1$. The impossibility to reduce the chain length further has tempted us to call this an optimally packed chain. In effect, the growth of flops mentioned above is reduced to a mere 60% and data locality is improved. As we will see, this directly translates into reduced execution times.

The parallel variant of the QR algorithm suggested by Henry, Watkins, and Dongarra [11] and implemented in ScaLAPACK also makes use of several bulges, placed sufficiently separate from each other to be able to chase them in parallel. On the process node level, this amounts to the use of a single bulge (i.e., $n_\mathsf{b} = 1$) and suffers from the poor flops per memory access ratio explained above. Recent work [9, 10] has shown that the use of several tightly packed chains of bulges in the parallel QR algorithm improves performance, often significantly. It can be expected that the use of optimally packed chains will lead to even further improvements. However, for the sake of simplicity, we keep the focus of this paper on serial implementations.

The rest of this paper is organized as follows. Section 2 briefly recalls the basic bulge chasing mechanism. In Section 3, the existing approach of packing bulges is explained. In Section 4, our optimal packing is introduced. In Section 5, we analyze the impact of this packing on the number of flops required by the QR algorithm and provide some implementation details concerned with performing updates by means of matrix–matrix multiplications. Finally, in Section 6, various numerical experiments demonstrate that the improved packing consistently reduces the execution time of the LAPACK implementation of the QR algorithm.

## 2 The bulge chasing mechanism

For the rest of the paper, we assume that the initial Hessenberg reduction of the QR algorithm has been performed and focus on the QR iterations applied to the Hessenberg matrix $H \in \mathbb{R}^{n\times n}$.

An implicit shifted QR iteration starts by choosing $m$ shifts $\sigma_1, \ldots, \sigma_m \in \mathbb{C}$ closed under complex conjugation and computes the vector

$$v = (H - \sigma_1 I) \cdots (H - \sigma_m I) e_1,$$

where $e_1$ denotes the first unit vector. Note that $v$ is zero except for the first $m+1$ entries. Using a Householder reflector, an orthogonal matrix $Q_0$ is computed such that $Q_0^T v$ becomes a scalar multiple of $e_1$. Then the update $H \leftarrow Q_0^T H Q_0$ is performed. Since $Q_0$ differs from the identity matrix only in the leading $(m+1)\times(m+1)$ principal submatrix, this update perturbs the Hessenberg form of $H$ only locally. For $m = 2$, $H$ will take the shape illustrated in Figure 1. The shaded $3 \times 3$ box in the figure contains the newly introduced entries and is

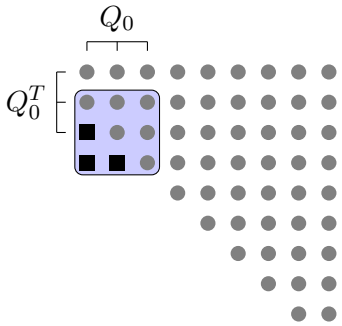


Figure 1: Illustration of the introduction of a bulge during the first step of a QR iteration.

usually referred to as the *bulge*. For general $m$, the bulge has size $(m+1)\times(m+1)$.

The rest of the QR iteration consists of chasing the bulge downwards along the first subdiagonal until it disappears at the bottom right corner. In the $j$th step of this process, a Householder reflector is used to construct an orthogonal matrix $Q_j$ that annihilates the entries below the subdiagonal in the first column of the bulge. Performing the update $H \leftarrow Q_j^T H Q_j$ effectively moves the bulge one step downwards. This is illustrated in Figure 2. In total, $n-2$ steps are needed until the bulge reaches the bottom right corner and disappears there.

**Algorithm 1** Bulge chasing

**Input:** Hessenberg matrix $H \in \mathbb{R}^{n\times n}$ perturbed by an $(m+1)\times(m+1)$ bulge starting at column $i$. Number of steps $k$ to chase the bulge.

**Output:** Updated Hessenberg matrix $H$ perturbed by an $(m+1)\times(m+1)$ bulge starting at column $i+k$.

1: **for** $j \leftarrow i, \ldots, i+k-1$ **do**
2: Construct Householder reflector $V \in \mathbb{R}^{(m+1)\times(m+1)}$ such that $V^T H(j+1 : j+m+1, j)$ is a multiple of $e_1$.
3: Update from the left: $H(j+1 : j+m+1, j : n) \leftarrow V^T H(j+1 : j+m+1, j : n)$.
4: Update from the right: $H(1 : j+m+2, j+1 : j+m+1) \leftarrow H(1 : j+m+2, j+1 : j+m+1)V$.
5: **end for**

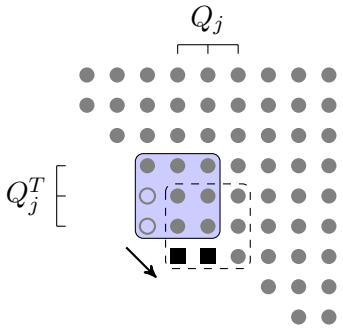


Figure 2: Illustration of a typical bulge chasing step. The first column of the bulge is reduced by a reflector, which is then applied to the corresponding rows and columns of the matrix. The net effect of these two transformations is that the bulge moves one step downwards.

Algorithm 1 gives a description of $k$ bulge chasing steps and will serve as a building block for our further developments. It requires approximately $4kmn + 2kn$ flops.

## 3 Tightly packed chains of bulges

The computational intensity of chasing a single bulge is proportional to the bulge size $m$. As explained in the introduction, increasing $m$ leads to the shift blurring phenomenon, which slows down the convergence of the QR algorithm in finite-precision arithmetic. To avoid this phenomenon and still benefit from increased computational intensity, it is preferable to use a chain of small bulges instead of one big bulge. In this section, we briefly summarize the concept of *tightly packed chains* proposed in [4].

The first part consists of introducing a chain of bulges in the top left corner of $H$. For this purpose, the first bulge is introduced and initially chased only $m+1$ (instead of $\mathcal{O}(n)$) steps using Algorithm 1. This creates sufficient room for the next bulge to be introduced. Then both bulges are chased $m+1$ steps, creating sufficient room for a third bulge, and so on. By the end of this procedure, $n_\mathsf{b}$ bulges are lined up in a chain covering $(m+1)n_\mathsf{b}$ columns. The shape of $H$ at that point is illustrated in Figure 3 for $m = 2$ and $n_\mathsf{b} = 3$.

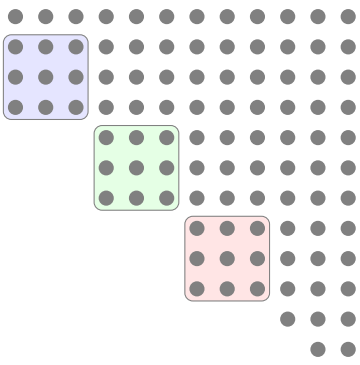

Figure 3: Illustration of a tightly packed chain of bulges introduced in the top left corner.

It is important to note that none of the bulges (except the first one) can advance another step without colliding with the bulge in front of it. This point is illustrated in Figure 4, which

shows that the collision creates one $(2m+1) \times (2m+1)$ bulge from two $(m+1) \times (m+1)$ bulges. As already pointed out, the use of such large bulges is not advisable and prone to the shift blurring phenomenon.

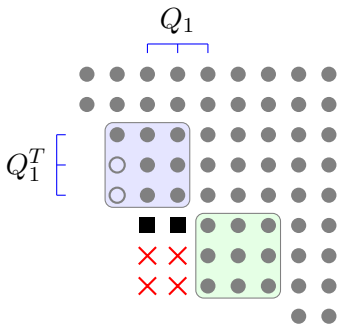


Figure 4: Illustration of the unwanted fill-in (red crosses) caused by chasing the blue bulge one step too far and letting it collide with the green bulge in front of it.

Now that the bulges are lined up in a chain, the process continues by chasing the whole chain simultaneously, see Figure 5 for an illustration. We chase the chain only $k \ll n$ steps,

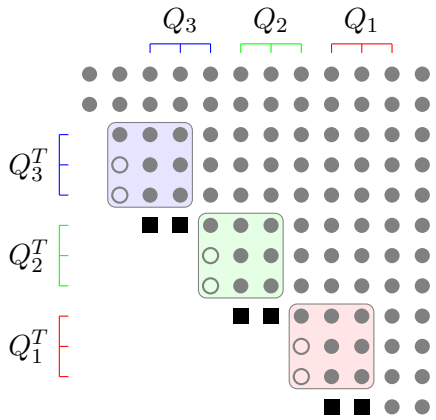


Figure 5: Illustration of chasing a tightly packed chain of bulges one step.

allowing us to restrict the updates of $H$ during the chase to a principal submatrix of size $(m+1)n_\mathsf{b} + k + 1$. In the following, we will refer to this submatrix as the *computational window*, which is the blue region denoted by $W$ in Figure 6. All orthogonal transformations applied within the window are accumulated into an orthogonal matrix $U$. This enables us to apply the transformations above and to the right of the window in terms of matrix–matrix multiplications with $U$. More specifically, the submatrices denoted by $A$ and $B$ in Figure 6 are updated via $A \leftarrow AU$ and $B \leftarrow U^T B$. At the end, the chain resides in the bottom right corner of the current computational window. The whole process is repeated by choosing a new computational window of the same size holding the chain in its top left corner. Eventually, the chain of bulges reaches the bottom right corner of the Hessenberg matrix and disappears there.

The algorithm described above performs most of its operations in terms of multiplications with the matrix $U$. The typical nonzero pattern of $U$ is illustrated in Figure 7. In particular,

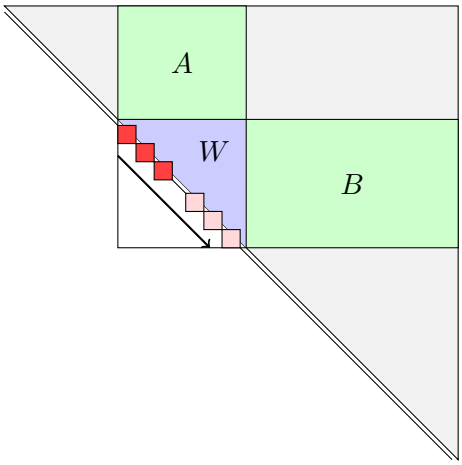


Figure 6: Illustration of the process of moving a tightly packed chain of bulges within a computational window $W$ before updating the off-diagonal submatrices $A$ and $B$ using matrix–matrix multiplications with an accumulated transformation matrix.

$U$ can be partitioned as

$$U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix}, \tag{2}$$

where $U_{12}$ is lower triangular and $U_{21}$ is upper triangular. The LAPACK implementation of the QR algorithm offers two options for exploiting the structure in multiplications with $U$: (1) completely ignore it and use one call to `xGEMM`; (2) exploit the triangular shapes of $U_{12}$ and $U_{21}$ by calls to `xGEMM` and `xTRMM`. To be advantageous, Option (2) requires a fairly well-tuned implementation of `xTRMM`, which may sometimes not be available. When using Option (1)

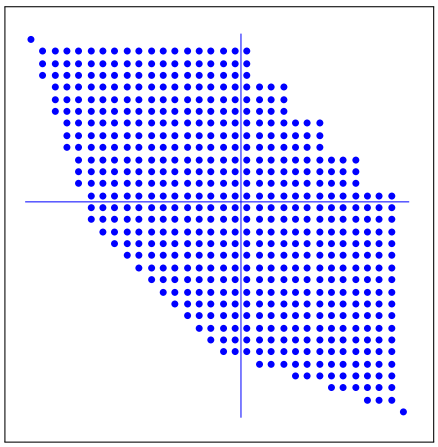

Figure 7: Sparsity pattern of the accumulated transformation matrix $U$ for a tightly packed chain with $m = 2$, $n_\mathsf{b} = 5$, and $k = (m+1)n_\mathsf{b} + 1 = 16$.

with $m = 2$, the analysis in [4] shows that the flop overhead caused by applying $U$ instead of the Householder reflectors directly is minimized when choosing $k = 3n_\mathsf{b}$.

**Remark 3.1** *When chasing a bulge one step downwards, it may happen that the updated subdiagonal element just above the bulge becomes so small that it can be safely set to zero,*

*giving rise to a so called vigilant deflation [16]. If the next bulge is moved immediately before checking for such deflations, the update from the right will engage this subdiagonal element, making it more difficult to perform the check. On the other hand, to keep the code structure simple and allow for compiler optimizations, one would like to move all bulges simultaneously and avoid to perform such checks in between. The LAPACK bulge-chasing routine* `xLAQR5` *achieves this by postponing the update of the last row for each bulge until after the deflation checks. Interestingly, this postponing of updates will be the key idea for our optimally packed chains introduced in the next section.*

# 4 Optimally packed chains of bulges

In this section, we aim at condensing the packing of the bulges in a chain. For this purpose, it is helpful to reconsider Figure 4, illustrating what prevented us from creating a more condensed packing. The reason why we cannot move the second bulge one step further is because the application of the corresponding Householder reflector from the right would mix up the second bulge with the first column of the first bulge. However, this can be prevented by annihilating the first column of the first bulge by another Householder reflector before attempting to move the second bulge. Of course, if we immediately applied all updates with the new Householder reflector then this would just correspond to moving the first bulge one step away and nothing would be gained in terms of a condensed packing. Hence, the idea is to delay parts of the updates.

At the minimum, we have to delay the update of the last row. This is shown in the left illustration of Figure 8, where a bulge is located at column 2 but its last row has not been formed yet, which is indicated by hollow squares. In each step of chasing the bulge, the last row is first formed by applying the previous Householder reflector. In its final position, when the bulge has been chased $k$ steps, the last row of the bulge is again not formed. The right illustration of Figure 8 shows this configuration after $k = 3$ steps. This idea leads to Algorithm 2, which performs bulge chasing with delayed row updates.

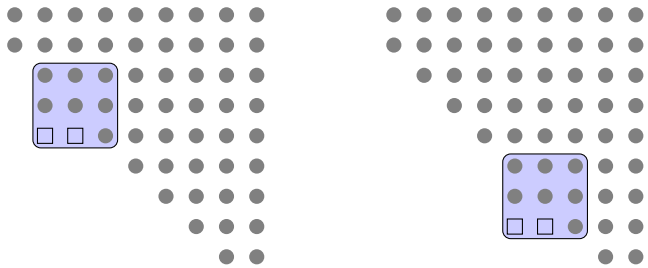

Figure 8: Illustration of a bulge with an unformed last row (*left*) and the same bulge after being moved $k = 3$ steps, again with an unformed last row (*right*).

As explained above, the delayed row updates avoid the interference of the bulges shown in Figure 4. Hence, we can achieve the packing displayed in Figure 9, where the hollow squares again denote delayed updates. Comparing this figure with Figure 3 clearly reveals the reduced length of the optimally packed chain.

When chasing such an optimally packed chain one step, we start with the first bulge and apply one step of Algorithm 2. This is repeated for the second bulge, for the third bulge,

**Algorithm 2** Bulge chasing with delayed row update

**Input:** Hessenberg matrix $H \in \mathbb{R}^{n\times n}$ perturbed by a bulge starting at column $i$. Previous Householder reflector $V$. Number of steps $k$ to chase the bulge.

**Output:** Updated Hessenberg matrix $H$ perturbed by a bulge starting at column $i+k$. Next Householder reflector $V$.

1: **for** $j \leftarrow i, \ldots, i+k-1$ **do**
2: Apply delayed update to last row: $H(j+m+1, j:j+m) \leftarrow H(j+m+1, j:j+m)V$.
3: Construct Householder reflector $\hat{V} \in \mathbb{R}^{(m+1)\times(m+1)}$ such that $\hat{V}^T H(j+1:j+m+1, j)$ is a multiple of $e_1$.
4: Update $V \leftarrow \hat{V}$.
5: Update from the left: $H(j+1:j+m+1, j:n) \leftarrow V^T H(j+1:j+m+1, j:n)$.
6: Incomplete update from the right:
$H(1:j+m+1, j+1:j+m+1) \leftarrow H(1:j+m+1, j+1:j+m+1)V$.
7: **end for**

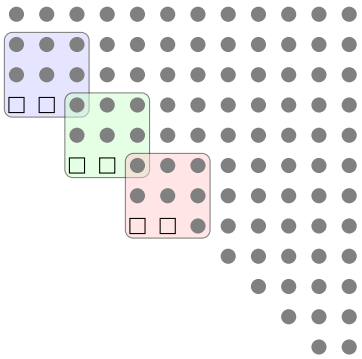

Figure 9: Illustration of an optimally packed chain of bulges introduced in the top left corner.

and so on until all bulges have been processed. Note that it is important that this order is obeyed.

This completes the basic description of the concept of optimally packed chains. As described in Section 3, a chain is chased in a computational window and the updates outside the window are performed by matrix–matrix multiplications.

**Remark 4.1** *There are several alternative ways of implementing the details of the bulge-chasing process. The method proposed by Lang [15] chases each bulge by k columns before considering the following bulge. This approach requires special treatment only every k steps. The method presented in this paper was chosen to mimic the method used in the current LAPACK implementation.*

# 5 Analysis and Implementation

## 5.1 Choice of parameters and flops

The implementation of chasing a chain of bulges depends on a number of parameters, such as the size of the bulge ($m+1$), the number of bulges in the chain ($n_\mathsf{b}$), and the number of steps ($k$) the chain is chased inside the computational window. In this section, we will discuss the choice of these parameters.

To set the stage, we first provide some analysis for tightly packed chains. For the moment, let us assume fixed $m = \mathcal{O}(1)$ and $n_\mathsf{b} \ll n$. The *chain length*, defined as the number of rows/columns occupied by the bulges, is given by $n_\mathsf{b}(m+1)$, see also Figure 3. The size of the computational window is given by $n_\mathsf{b}(m+1)+k+1$. To choose $k$ optimally, we make the following two assumptions:

1. The number of flops performed inside the computational window is negligible.
2. The structure of the orthogonal transformation matrix $U$, see Figure 7, is *not* exploited.

In particular, the second assumption implies that the updates outside the computational window require approximately

$$2n(n_\mathsf{b}(m+1)+k+1)^2$$

flops. On the other hand, chasing a chain of $n_\mathsf{b}$ bulges $k$ steps effects $kn_\mathsf{b}$ total bulge chasing steps. Hence, to obtain a minimal ratio between effort and progress, we should minimize the cost per step:

$$\frac{2n(n_\mathsf{b}(m+1)+k+1)^2}{kn_\mathsf{b}}.$$

It is straightforward to check that the minimum is attained for $k = n_\mathsf{b}(m+1)+1$. Since in total $n_\mathsf{b}n$ steps are needed to chase $n_\mathsf{b}$ bulges from the top left to the bottom right corner of the Hessenberg matrix, this optimal choice of $k$ leads to an overall cost of

$$C_\mathsf{tight} = 8n^2(n_\mathsf{b}(m+1)+1).$$

For an optimally packed chain, the chain length is given by $n_\mathsf{b}m+1$, see also Figure 9, and the computational window has size $n_\mathsf{b}m+k+2$. The resulting cost per step is

$$\frac{2n(n_\mathsf{b}m+k+2)^2}{kn_\mathsf{b}},$$

whose minimum is attained for $k = n_\mathsf{b} m + 2$, leading to an overall cost of

$$C_{\mathsf{optimal}} = 8n^2(n_\mathsf{b} m + 2).$$

To summarize, the use of an optimally packed chain reduces the flops by approximately $1/(m+1)$, amounting to 33% for the usual choice $m = 2$.

It remains to discuss the choice of $m$ and $n_\mathsf{b}$. For optimally packed chains, there is no apparent advantage of using $m > 2$. Unlike for tightly packed chains [14], an increase of $m$ does not lead to a more condensed packing. So we fix $m = 2$. The choice of $n_\mathsf{b}$ is much more subtle, regardless of the packing scheme. An optimal choice depends on the matrix size $n$ and the performance of the BLAS. It even depends on the matrix entries, as a change of $n_\mathsf{b}$ has a hard-to-predict impact on the convergence of the QR algorithm and the behavior of the aggressive early deflation technique [5] used in modern implementations. In lack of a better alternative, we suggest to follow the recommendations for tightly packed chains specified in [6, Table 2] and provided as default option in the LAPACK function `IPARMQ`.

### 5.2 Structure of orthogonal transformations

On the computing environments we have considered, exploiting the structure of $U$, as discussed in Section 3, only led to negligible speedups at best. Therefore we apply $U$ as a dense matrix in all our experiments. However, for completeness, we discuss the structure of $U$ also for optimally packed chains.

The typical nonzero pattern of the orthogonal transformation matrix $U$ obtained by chasing an optimally packed chain is illustrated in Figure 10 for the case $m = 2$, $n_\mathsf{b} = 7$, and $k = 16$. This figure should be compared to Figure 7, which illustrates the corresponding nonzero pattern for a tightly packed chain (with $n_\mathsf{b} = 5$). Note that $U$ can again be partitioned as

$$U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix},$$

where $U_{21}$ is upper triangular and $U_{12}$ is lower triangular for $m = 2$. This allows to apply $U$ by means of calls to the BLAS `xGEMM` and `xTRMM`.

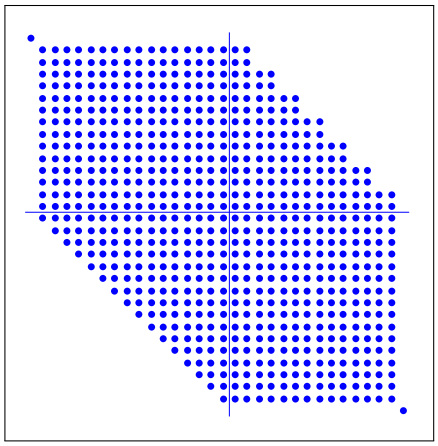

Figure 10: Typical nonzero pattern of the orthogonal transformation matrix for an optimally packed chain with $m = 2$, $n_\mathsf{b} = 7$, and $k = mn_\mathsf{b} + 2 = 16$.

**Remark 5.1** *If the triangular structure of $U_{12}$ is not expoited then the size of the matrix $U$ may be reduced as follows. Noting that the top left and bottom right entries in Figure 10 must be ones, we only need an "effective" computational window of size $n_{\mathsf{b}}m + k$, leading to an optimal value $k_{\mathsf{opt}} = n_{\mathsf{b}}m$. In fact, a similar reasoning applies to the tightly packed chains; cf. Figure 7, giving $k_{\mathsf{opt}} = n_{\mathsf{b}}(m+1)-1$. Again, the optimally packed chains will save roughly $1/(m+1)$ of the flops.*

# 6 Numerical experiments

To compare the performance of the new optimally packed chains versus the tightly packed chains currently used in LAPACK, we have modified the corresponding LAPACK routine `DLAQR5` to make use of optimally packed chains. Very few changes of the code were necessary and the overall structure of the code remains as described in [6]. Some care needed to be taken to keep the vigilant deflation strategy intact, see also Remark 3.1. Our new implementation can be downloaded from `http://anchp.epfl.ch`.

We have performed tests on two different architectures hosted by the High Performance Computing Center North (HPC2N) in Umeå (Sweden):

- *Akka* Two Intel Xeon L5420 processors at 2.5 GHz with four cores each and a total of 16 GB RAM. Each core has a private 32 KB L1 cache and pairs of cores share a 6 MB L2 cache. The code was compiled by the PathScale `pathf95` compiler version 4.0.12 using the flags `-march=auto -O3` and linked to sequential GotoBLAS2 version 1.13 and LAPACK version 3.2.2.

- *Abisko* Four AMD Opteron 6238 (Interlagos) processors at 2.6 GHz with twelve cores each and a total of 128 GB RAM. Each core has a private 16 KB L1 cache, pairs of cores, called modules, share a 2 MB L2 cache, and groups of three modules share a 6 MB L3 cache. The code was compiled by the PathScale `pathf95` compiler version 4.0.12.1 using the flags `-march=bdver1 -msse -msse2 -msse3 -mavx -O3` and linked to sequential ACML version 5.1.0 and LAPACK version 3.4.0.

In all our experiments, the full Schur form including the orthogonal factor $Q$ is computed. Both the norm of the residual and the orthogonality of $Q$ have been verified. In none of the test cases did we observe an essential difference in accuracy.

## 6.1 Experiments with random matrices

For the first set of experiments, we have generated full matrices with pseudorandom entries uniformly distributed in $[0, 1]$. The execution time spent by the LAPACK routine `DGEHRD` for reducing these matrices to Hessenberg form is not included in the figures below.

Unless otherwise noted, we have used the default parameters from the LAPACK routine `IPARMQ` with the following exception. Throughout all experiments, we explicitly switched off the exploitation of the $2 \times 2$ triangular block structure when performing matrix–matrix multiplications with the orthogonal transformation matrices $U$, see Section 5.2.

Figures 11 and 12 show the obtained execution times relative to the LAPACK implementation with the default setting for $n_{\mathsf{s}} = 2n_{\mathsf{b}}$ (number of shifts), that is, $n_{\mathsf{s}} = 64$ for $n \in [590, 3000)$ and $n_{\mathsf{s}} = 128$ for $n \in [3000, 6000)$. For both the LAPACK and the new implementation, we also show the results obtained when choosing the optimal value of $n_{\mathsf{s}} \in \{2, 4, \ldots, 170\}$. Note

that this optimized value of $n_s$ is only used in calls to `DLAQR5` by the top-level routine `DLAQR0`. The lower-level routine `DLAQR3`, which performs aggressive early deflation, also indirectly calls `DLAQR5`, but the size of the involved matrices varies and is typically much smaller. The size of the aggressive early deflation window is always set to $3n_b = 3n_s/2$, in accordance with the suggestion in [6].

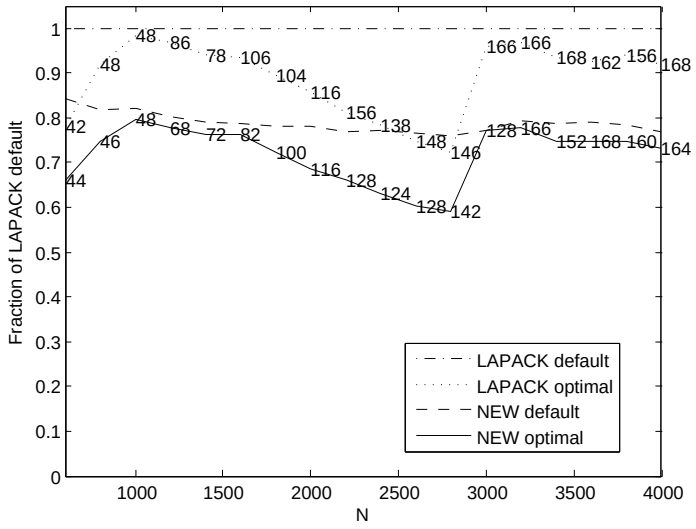


Figure 11: Execution times of the current LAPACK implementation and our new implementation using the default and optimal number of shifts. The numbers displayed on the graphs indicate the optimal number of shifts ($n_s$). The execution times are given relative to LAPACK with the default number of shifts. Results from *Akka*.

A number of observations can be made from Figures 11 and 12. Our new implementation is consistently faster than LAPACK, both for the default and the optimized choice of $n_s$; the reductions in execution time range between 14% and 23%. Although the optimized and default choices for $n_s$ may differ to a large extent, the impact of this difference on the execution times is less dramatic. This is illustrated in Figure 13, showing not only that $n_s$ strikes a balance between the time for aggressive early deflations and QR iterations but also that the target function is quite flat near the minimum.

## 6.2 Experiments with examples from the Matrix Market

Apart from random matrices, we have also tested several matrices from the benchmark collection [2] available from the Matrix Market. Throughout all experiments, the default choice of $n_s$ has been used. Figures 14 and 15 show the obtained execution times on *Akka* and *Abisko*, respectively. Note that the number in each matrix name refers to the matrix size. With two exceptions, the results reflect a consistent reduction of the execution time by 16% or more. This is quite remarkable when taking into account that the reference time by LAPACK may vary strongly for different matrices of (nearly) the same size.

The Tolosa matrices of size 2000 (tols2000) and 4000 (tols4000) behave exceptionally on *Akka*, with only 12% decrease and 9% *in*crease, respectively. A closer investigation reveals that this effect is caused by a mechanism for avoiding convergence failures, see [6, Pg. 17]. After five multishift QR iterations without any converged eigenvalues, the LAPACK routine

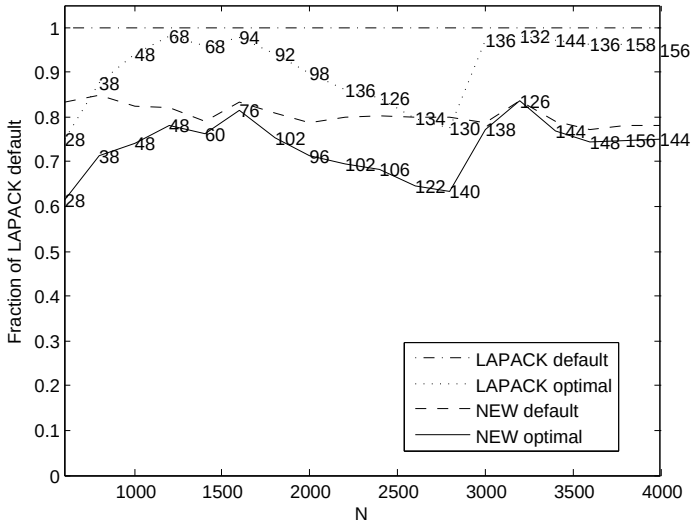


Figure 12: Execution times of the current LAPACK implementation and our new implementation using the default and optimal number of shifts. The numbers displayed on the graphs indicate the optimal number of shifts ($n_s$). The execution times are given relative to LAPACK with the default number of shifts. Results from *Abisko*.

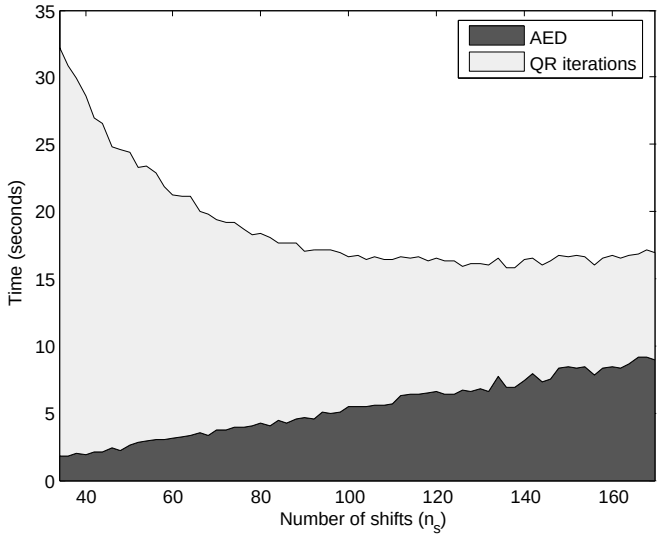


Figure 13: The optimal choice of $n_s$ is found by trading off the cost for QR iterations against the cost of aggressive early deflation (AED). This figure was obtained on *Abisko* using the new implementation on a random matrix of size $n = 3000$.

`xLAQR0` doubles the size of the aggressive early deflation window after each iteration until it cannot be increased any further due to workspace limitations. If still no eigenvalues have converged, the deflation window size is slowly decreased. While the LAPACK implementation happens to require one large deflation window for tols4000, our new implementation requires three such large windows until eigenvalues converge, leading to the increased execution time. A similar situation arises for tols2000. We believe this effect to be purely coincidental and not related to the use of optimally packed chains. Even the slightest perturbation (e.g., due to differences in roundoff error) may make it disappear or even reverse. In fact, this happens when performing the same experiments on *Abisko*.

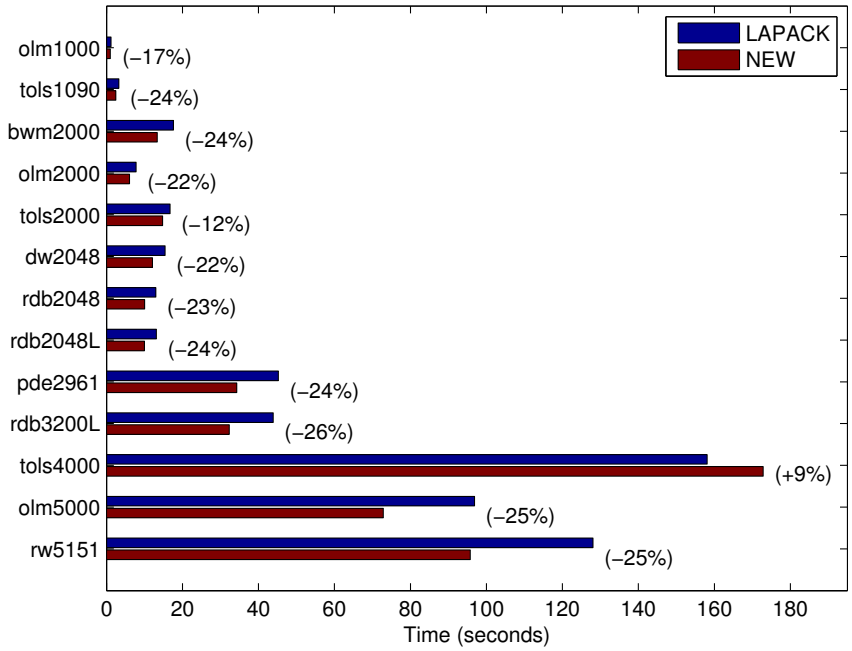


Figure 14: Comparison between LAPACK and the new implementation for a selection of examples from the Matrix Market on *Akka*.

# 7 Conclusion

We have introduced the concept of optimally packed chains of bulges in the context of multishift QR algorithms, allowing for a significant increase of the computational intensity. Numerical experiments with a modified LAPACK implementation that incorporates the proposed technique demonstrate its effectiveness. The concept of optimally packed chains is quite general and certainly carries over to other bulge-chasing algorithms, such as the multishift QZ algorithm [12] or parallel implementations of the QR algorithm.

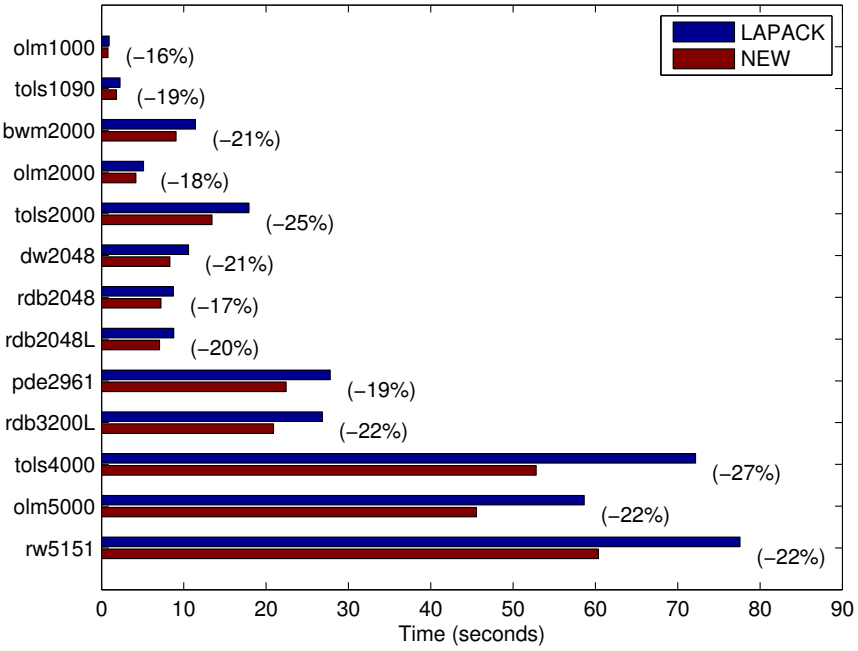


Figure 15: Comparison between LAPACK and the new implementation for a selection of examples from the Matrix Market on *Abisko*.

## 8 Acknowledgments

We thank Bo Kågström and Meiyue Shao for inspiring discussions on the subject of this article. This research was conducted using the resources of High Performance Computing Center North (HPC2N).

## References

[1] 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.

[2] 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`.

[3] Z. Bai and J. W. Demmel. On a block implementation of the Hessenberg multishift $QR$ iterations. *Internat. J. High Speed Comput.*, 1:97–112, 1989.

[4] 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.

[5] 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.

[6] 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.

[7] J. G. F. Francis. The QR transformation, parts I and II. *Computer Journal*, 4:265–271, 332–345, 1961, 1962.

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

[9] 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.

[10] R. Granat, B. Kågström, D. Kressner, and M. Shao. Parallel library software for the multishift QR algorithm with aggressive early deflation. 2012. In preparation.

[11] 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.

[12] 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. Also appeared as LAPACK working note 173.

[13] D. Kressner. *Numerical Methods for General and Structured Eigenvalue Problems*, volume 46 of *Lecture Notes in Computational Science and Engineering.* Springer-Verlag, Berlin, 2005.

[14] D. Kressner. On the use of larger bulges in the QR algorithm. *Electron. Trans. Numer. Anal.*, 20:50–63, 2005.

[15] B. Lang. Effiziente Orthogonaltransformationen bei der Eigen- und Singulärwertzerlegung. Habilitationsschrift, Bergische Universität Gesamthochschule Wuppertal, 1997.

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

[17] D. S. Watkins. Forward stability and transmission of shifts in the $QR$ algorithm. *SIAM J. Matrix Anal. Appl.*, 16(2):469–487, 1995.

[18] D. S. Watkins. *The matrix eigenvalue problem.* SIAM, Philadelphia, PA, 2007. $GR$ and Krylov subspace methods.

**Recent publications :**

**MATHEMATICS INSTITUTE OF COMPUTATIONAL SCIENCE AND ENGINEERING**
**Section of Mathematics**
**Ecole Polytechnique Fédérale**
**CH-1015 Lausanne**

**17.2012** A. ABDULLE, W. E, B. ENGQUIST, E. VANDEN-EIJNDEN:
*The heterogeneous multiscale method*

**18.2012** D. KRESSNER, M. PLESINGER, C. TOBLER:
*A preconditioned low-rank CG method for parameter-dependent Lyapunov matrix equations*

**19.2012** A. MANZONI, T. LASSILA, A. QUARTERONI, G. ROZZA:
*A reduced-order strategy for solving inverse Bayesian shape identification problems in physiological flows*

**20.2012** J. BONNEMAIN. C. MALOSSI, M. LESINIGO, S. DEPARIS, A. QUARTERONI, L. VON SEGESSER:
*Numerical simulation of left ventricular assist device implantations: comparing the ascending and the descending aorta cannulations*

**21.2012** A. ABDULLE, M. HUBER:
*Discontinuous Galerkin finite element heterogeneous multiscale method for advection-diffusion problems with multiple scales*

**22.2012** R. ANDREEV, C. TOBLER:
*Multilevel preconditioning and low rank tensor iteration for space-time simultaneous discretizations of parabolic PDES*

**23.2012** P. CHEN, A. QUARTERONI, G. ROZZA:
*Stochastic optimal Robin boundary control problems of advection-dominated elliptic equations*

**24.2012** J. BECK, F. NOBILE, L. TAMELLINI, R. TEMPONE:
*Convergence of quasi-optimal stochastic Galerkin methods for a class of PDES with random coefficients*

**25.2012** E. FAOU, F. NOBILE, C. VUILLOT:
*Sparse spectral approximations for computing polynomial functionals*

**26.2012** G. ROZZA, A. MANZONI, F. NEGRI:
*Reduction strategies for PDE-constrained optimization problems in haemodynamics*

**27.2012** C. EFFENBERGER:
*Robust successive computation of eigenpairs for nonlinear eigenvalue problems*

**28.2012** C. LUBICH, T. ROHWEDDER, R. SCHNEIDER, B. VANDEREYCKEN:
*Dynamical approximation of hierarchical Tucker and tensor-train tensors*

**29.2012** A. ABDULLE, Y. BAI:
*Adaptive reduced basis finite element heterogeneous multiscale method*

**30.2012** *new* L. KARLSSON, D. KRESSNER, B. LANG:
*Optimally packed chains of bulges in multishift QR algorithms*