A Blocked QR-Decomposition for the Parallel Symmetric Eigenvalue Problem

T. Auckenthaler[a,*], T. Huckle[a], R. Wittmann[a]

[a] *Fakultät für Informatik, Technische Universität München, D-85748 Garching, Germany*

Abstract

In this paper we present a new stable algorithm for the parallel QR-decomposition of "tall and skinny" matrices. The algorithm has been developed for the dense symmetric eigensolver ELPA, whereat the QR-decomposition of tall and skinny matrices represents an important substep. Our new approach is based on the fast but unstable CholeskyQR algorithm [1]. We show the stability of our new algorithm and provide promising results of our MPI-based implementation on a BlueGene/P and a Power6 system.

Keywords: QR-decomposition, eigenvalue and eigenvector computation, two-step tridiagonalization, parallelization

1. Introduction

Finding the eigenvalues and eigenvectors of a symmetric matrix is a widespread problem in linear algebra. If the symmetric matrix is dense, the most suited proceeding is to (1) reduce the matrix to symmetric tridiagonal form, (2) solve the tridiagonal eigensystem and (3) if eigenvectors are desired, transform the eigenvectors of the tridiagonal eigensystem back to those of the dense matrix. Thereby, the reduction to tridiagonal form is the bottleneck of the whole algorithm in terms of runtime [2]. A detailed analysis of the parallel tridiagonalization can be found in [3]. There are two major reasons limiting the performance. On the one hand, a huge amount of operations ($\sim 50\%$) has to be performed using memory bound matrix-vector operations (BLAS2), limiting the sequential performance of the algorithm. On the other hand, a high number of messages limits the scalability of the parallel execution on distributed memory systems.

In our recent publications ([4], [5]) we presented a parallel symmetric eigensolver where the tridiagonalization is done in two steps leading to staggering results in both, absolute performance and scaling behavior. The implementation is meanwhile publicly available as a library under the name ELPA [6].

The two-step tridiagonalization was originally presented in [7]. At first, the full symmetric matrix of size n is reduced to symmetric banded form with (semi)bandwidth b. In a second step the banded matrix is reduced to tridiagonal form. Except of lower order terms, the first step

*Corresponding author
Email addresses: auckenth@in.tum.de (T. Auckenthaler), huckle@in.tum.de (T. Huckle), wittmanr@in.tum.de (R. Wittmann)

Parallel algorithm	Flops on critical path	Messages	Comm. volume
PDGEQRF	$\frac{2mn^2}{P} + \frac{n^2}{2}\log(P)$	$2n\log(P)$	$\frac{n^2}{2}\log(P)$
TSQR	$\frac{2mn^2}{P} + \frac{2}{3}n^3\log(P)$	$\log(P)$	$\frac{n^2}{2}\log(P)$
CholeskyQR	$\frac{2mn^2}{P} + \frac{n^3}{3}\log(P)$	$\log(P)$	$\frac{n^2}{2}\log(P)$

Table 1: Out of [9]: Performance model of selected parallel QR-decomposition algorithms. $n \times m$ stands for the size of the matrix, P for the number of parallel processes. PDGEQRF represents the ScaLAPACK QR-decomposition, based on the classical Householder approach.

can entirely be done using matrix-matrix operations (BLAS3). The second step uses BLAS2 but requires asymptotically less operations compared to step one ($6n^2b$ vs. $4/3n^3$).

Algorithm 1 outlines the band reduction of a symmetric matrix A. The algorithm consists

Algorithm 1 Symmetric band reduction

1: **for** $block\_col = 1 \rightarrow (n/b - 1)$ **do**
2: $\quad col \leftarrow (block\_col - 1) * b + 1$
3: $\quad Q, R \leftarrow QRDecomposition(A_{(col+b:n,col:col+b-1)})$
4: $\quad A_{(col+b:n,col:col+b-1)} \leftarrow R$
5: $\quad A_{(col+b:n,col+b:n)} \leftarrow Q^T A_{(col+b:n,col+b:n)} Q$
6: **end for**

of n/b iterations. In each iteration a QR-decomposition on a block-column of width b has to be performed (line 3). Afterwards the block-column is substituted with the matrix R (line 4) and the orthogonal matrix Q is applied to the rest of the matrix A (line 5). The most flops ($4/3n^3$) are spent on the update of the trailing matrix. However, the QR-decomposition gets increasingly important for massively parallel execution. In this paper we describe further optimizations of the symmetric band reduction which were achieved by means of a new algorithmic approach for the parallel QR-decomposition.

There exists related work which tackles the parallel QR-decomposition of tall and skinny matrices too. In [8, 9] Demmel et al. propose an algorithm named Tall Skinny QR (TSQR). The approach consists of first performing a local QR-decomposition on each process (assuming a 1D parallel data layout). Afterwards, the resulting matrices R_i are reduced to a final matrix R using any form of reduction tree (e.g., a binary tree). In [10] Dongarra et al. present an implementation thereof, experimenting with different kinds of reduction trees. In [11] a QR-decomposition, based on the Cholesky factorization, is proposed. CholeskyQR computes R using the Cholesky decomposition of A^TA since $A^TA = R^TQ^TQR = R^TR$. Q can then be computed with $Q = AR^{-1}$. CholeskyQR is suited very well for parallel computation since it requires only one synchronization point. However, it is not numerically stable if A is ill conditioned [1]. A good comparison regarding parallel performance and numerical stability of the cited algorithms can be found in [9]. Table 1 summarizes this work.

Our approach is mathematically equivalent to CholeskyQR and shares its good performance characteristics. However, our algorithm is extended by a concept of adaptive blocking to guarantee numerical stability for ill conditioned matrices. Furthermore, we compute Householder transformations instead of an orthogonal matrix Q.

The rest of this paper is organized as follows. Sect. 2 presents the classical Householder QR-decomposition as it is used within ScaLAPACK [12] and the ELPA eigensolver. In Sect.

3 we introduce our new approach, which we will call blocked Householder-CholeskyQR in the following. We derive the algorithm and analyze its accuracy. Sect. 4 provides performance and accuracy results of the algorithm and, finally, Sect. 5 summarizes our work.

2. Householder QR-decomposition

A common problem in numerical algorithms is the decomposition of a matrix $A \in \mathbb{R}^{n \times m}$ into a product of matrices $Q \in \mathbb{R}^{n \times n}$ and $R \in \mathbb{R}^{n \times m}$, where R is upper triangular and Q is orthogonal. The most common methods to achieve this decomposition are based on Householder transformations.

A Householder matrix of order n is defined as:

$$H = I - y \cdot \tau \cdot y^T \in \mathbb{R}^{n \times n} \text{ with } y \in \mathbb{R}^n \text{ and } \tau = \frac{2}{\|y\|_2^2} \tag{1}$$

It can be shown that Householder matrices are symmetric and orthogonal. Let v be a vector of size n which shall be transformed to a vector $x \in \mathbb{R}^n$ where the entries from 2 to n are set to zero. This vector may be a column of our matrix that we want to decompose. Algorithm 2,

Algorithm 2 HouseGen: Generation of Householder transformation

$$\begin{aligned}
\beta &= \|v\|_2 \cdot \operatorname{sign}(v_{(1)}) \\
\tau &= \frac{v_{(1)} + \beta}{\beta} \\
y &= \left(1, \frac{v_{(2:n)}^T}{v_{(1)} + \beta}\right)^T \\
x &= (-\beta, 0, \ldots, 0)^T
\end{aligned}$$

then, shows how the Householder vector y and the scalar τ are computed such that the following equation holds:

$$x = H^T v = (I - y \cdot \tau \cdot y^T)v \tag{2}$$

The generated Householder vector y and its scaling factor τ can be applied to a matrix $A \in \mathbb{R}^{n \times m}$ in the following way:

$$A' = (I - y \cdot \tau \cdot y^T) \cdot A \tag{3}$$

This transformation is done by Algorithm 3.

Algorithm 3 HouseLeft: Left-sided application of Householder transformation

1: $z^T = \tau \cdot y^T \cdot A$
2: $A' = A - y \cdot z^T$

By combining the generation (Algorithm 2) and the application (Algorithm 3) of Householder transformations it is possible to reduce a matrix $A \in \mathbb{R}^{n\times m}$ with $n \geq m$ to a matrix $R \in \mathbb{R}^{n\times m}$ of the following form (see Algorithm 4):

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} \text{ with } R_1 \in \mathbb{R}^{m\times m} \text{ and upper triangular} \tag{4}$$

The orthogonal matrix Q is never computed explicitly, but is represented by the set of House-

Algorithm 4 QR-decomposition of a matrix $A \in \mathbb{R}^{n\times m}$

1: **for** $i = 1 \rightarrow m$ **do**
2: $\quad \tau, y, x \leftarrow HouseGen(A_{(i:n,i)})$
3: $\quad A_{(i:n,i+1:m)} \leftarrow HouseLeft(\tau, y, A_{(i:n,i+1:m)})$
4: $\quad A_{(i:n,i)} \leftarrow x$
5: **end for**

holder vectors y_i and scaling factors τ_i, with

$$Q^T = \prod_i (I - y_i \cdot \tau_i \cdot y_i^T), \quad i = 1, \ldots, m. \tag{5}$$

3. Blocked Householder-CholeskyQR

The aim of our new algorithm is to make the reduction to banded form less dependent on network latency and, thus, to improve the scalability. To compute the individual Householder vectors the classical Householder QR-decomposition, as presented in Sect. 2, requires at least one synchronization point for each column of the matrix. This limits the scalability of the algorithm. The basic idea of blocked Householder-CholeskyQR is to generate and apply more than one Householder transformation with a single communication operation.

It turned out that the derived algorithm shares many commonalities with CholeskyQR, since it is based on the Cholesky decomposition of $A^T A$. However, our approach is extended by a concept of adaptive blocking to guarantee numerical stability. Furthermore, we generate Householder transformations to represent the orthogonal matrix Q, such that the QR-decomposition is easily integrable into the existing reduction to banded form within ELPA. In the following we sketch the derivation of the algorithms which can also be found in [13].

3.1. Rank-k QR-decomposition

At first, we introduce some notation for the following algorithm. We define the matrix $A \in \mathbb{R}^{n\times m}$ as A_0. A_k stands for the content of the matrix after applying k Householder transformations. $Y \in \mathbb{R}^{n\times k}$ contains all generated Householder vectors y_i. Furthermore, we define $H_1, H_2, \cdots, H_k$ to be the Householder matrices corresponding to $y_1, y_2, \cdots, y_k$.

W.l.o.g. we assume to start with a matrix $A_0 \in \mathbb{R}^{n\times m}$ where the first k columns have the following form:

$$A_{0\,(1:n,1:k)} = (a_1, a_2, \cdots, a_k) = (u_1, u_2, \cdots, u_k) \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,k} \\ & \alpha_{2,2} & \cdots & \alpha_{2,k} \\ & & & \cdots \\ & & & \alpha_{k,k} \end{pmatrix}. \tag{6}$$

u_i and u_j are normalized and mutually orthogonal for $i \neq j$. Out of these vectors we want to compute the set of Householder vectors $y_1, \ldots, y_k$ and the appropriate scalars $\tau_1, \ldots, \tau_k$ and $\beta_1, \ldots, \beta_k$ as well as the content of R.

We start with the definition of the Householder vectors y_i

$$y_{i\ (1:i-1)} = 0^T_{(1:i-1)}, \quad y_{i\ (i)} = 1,$$

$$y_{i\ (i+1:n)} = \frac{A_{i-1\ (i+1:n,i)}}{A_{i-1\ (i,i)} + \beta_i}, \tag{7}$$

and the scalars τ_i and β_i from Algorithm 2

$$\beta_i = \|A_{i-1\ (i:n,i)}\|_2 \cdot \text{sign}(A_{i-1\ (i,i)}) \tag{8}$$

$$\tau_i = \frac{A_{i-1\ (i,i)} + \beta_i}{\beta_i}. \tag{9}$$

A possible decomposition into Q and R is shown in Equation (6) since $(u_1, u_2, \cdots, u_k)$ is orthogonal and the matrix containing $\alpha_{i,j}$ is upper triangular. If the Householder transformations are defined as it has been done in Algorithm 2, then R has the following form:

$$R = \begin{bmatrix} -s_1\alpha_{1,1} & -s_1\alpha_{1,2} & \cdots & -s_1\alpha_{1,k} \\ & -s_2\alpha_{2,2} & \cdots & -s_2\alpha_{2,k} \\ & & & \cdots \\ & & & -s_k\alpha_{k,k} \end{bmatrix}, \tag{10}$$

where s_i is the sign of $A_{i-1\ (i,i)}$.

For the first row of R we can easily show that

$$R_{1,j} = (H_1 \cdot A_0)_{1,j} = -s_1\alpha_{1,j}. \tag{11}$$

After the application of the first Householder transformation $A_1 = H_1 A_0$ has the following form:

$$A_{1\ (1:n,1:k)} = H_1(a_1, a_2, \cdots, a_k) = (\hat{u}_1, \hat{u}_2, \cdots, \hat{u}_k) \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \cdots & \alpha_{1,k} \\ & \alpha_{2,2} & \cdots & \alpha_{2,k} \\ & & & \cdots \\ & & & \alpha_{k,k} \end{pmatrix}, \tag{12}$$

where $\hat{u}_i = H_1 u_i$. Furthermore, we can show that $\hat{u}_1 = -s_1 e_1$ and $\hat{u}_{i\ (1)} = 0$ for $i > 1$.

If we now look at the matrix $A_{1\ (2:n,2:k)}$, we come upon the same pattern as for A_0. But now we have to deal with a QR-decomposition of size $(n-1) \times (k-1)$ instead of $n \times k$. As for $R_{(1,1:k)}$ we can now determine $R_{(2,2:k)}$. Therefore, Equation (10) is true by induction.

To compute the coefficients $\alpha_{i,j}$ of the matrix R we need all combinations of dot products $D = A^T A$, where $D_{(i,j)} = a_i^T a_j$. In addition we need all signs s_i to get the final content of R. In Algorithm 5 we show how to compute all the $\alpha_{i,j}$ out of D. Please note that this algorithm corresponds to a Cholesky decomposition of $A^T A$.

After computing the coefficients $\alpha_{i,j}$, we need an update strategy to compute A_i out of A_{i-1} without any further synchronization requirements. Therefore, line 1 of Algorithm 3 can be generalized to

Algorithm 5 Computation of $\alpha_{i,j}$ (Cholesky decomposition)

1: **for** $i = 1 \to k$ **do**
2: $\quad \alpha_{i,i} \leftarrow \sqrt{D_{(i,i)}}$
3: $\quad$ **for** $j = i + 1 \to k$ **do**
4: $\quad\quad \alpha_{i,j} \leftarrow \frac{D_{(i,j)}}{\alpha_{i,i}}$
5: $\quad\quad$ **for** $l = i + 1 \to j$ **do**
6: $\quad\quad\quad D_{(l,j)} \leftarrow D_{(l,j)} - \alpha_{i,j}\alpha_{i,l}$
7: $\quad\quad$ **end for**
8: $\quad$ **end for**
9: **end for**

$$\begin{aligned} z_{i\,(j)} &= \tau_i \cdot y_i^T \cdot A_{i-1\,(i:n,j)} \\ &= A_{i-1\,(i,j)} + \frac{A_{i-1\,(i:n,i)}^T \cdot A_{i-1\,(i:n,j)}}{\beta_i}, \end{aligned} \tag{13}$$

and, thus, using Equation (7),

$$\begin{aligned} A_{i\,(i+1:n,j)} =& A_{i-1\,(i+1:n,j)} - z_{i\,(j)} \cdot \frac{A_{i-1\,(i+1:n,i)}}{A_{i-1\,(i,i)} + \beta_i} \\ =& A_{i-1\,(i+1:n,j)} - \rho_{i,j} A_{i-1\,(i+1:n,i)}, \text{ with} \end{aligned} \tag{14}$$

$$\rho_{i,j} = \frac{A_{i-1\,(i,j)} + \frac{A_{i-1\,(i:n,i)}^T \cdot A_{i-1\,(i:n,j)}}{\beta_i}}{A_{i-1\,(i,i)} + \beta_i}. \tag{15}$$

Since

$$A_{i-1\,(i:n,i)}^T A_{i-1\,(i:n,j)} = \underbrace{(\alpha_{i,i}\hat{\hat{u}}_i)^T}_{A_{i-1\,(i:n,i)}^T} \cdot \underbrace{(\hat{\hat{u}}_i,\, \cdots,\, \hat{\hat{u}}_j) \cdot (\alpha_{i,j},\, \cdots,\, \alpha_{j,j})^T}_{A_{i-1\,(i:n,j)}} = \alpha_{i,i}\alpha_{i,j},$$

Equation (15) can be simplified to

$$\rho_{i,j} = \frac{A_{i-1\,(i,j)} + \frac{\alpha_{i,i}\alpha_{i,j}}{\beta_i}}{A_{i-1\,(i,i)} + \beta_i}, \tag{16}$$

where $\hat{\hat{u}}$ corresponds to u after applying i Householder transformations. Out of Equation (14) and Algorithm 5 we can now formulate the rank-k variant of $HouseGen$ (see Algorithm 6). The update of the matrix A in line 9 can easily be done in a blocked fashion such that cache efficiency is guaranteed.

After the generation, the Householder transformations have to be applied to the rest of the matrix. This can be done with blocked Householder transformations, e.g. compact WY transformations [14].

3.2. Accuracy analysis

Under certain conditions the presented algorithm runs into numerical problems. In the following we will analyze the relative error of the rank-k QR-decomposition and derive a criterion for the stability of the algorithm.

Algorithm 6 HouseGen, rank-k version

$D \leftarrow A^T A$
Compute $\alpha_{i,j}$ (Algorithm 5)
for $i = 1 \rightarrow k$ **do**
  $\beta_i \leftarrow \alpha_{i,i} \cdot \text{sign}(A_{i-1\ (i,i)})$
  $R_{(i,i)} \leftarrow -\beta_i$
  $\tau_i \leftarrow \frac{A_{i-1\ (i,i)}+\beta_i}{\beta_i}$
  for $j = i + 1 \rightarrow k$ **do**
    $\rho_{i,j} \leftarrow \frac{A_{i-1\ (i,j)}+\frac{\alpha_{i,i}\alpha_{i,j}}{\beta_i}}{A_{i-1\ (i,i)}+\beta_i}$
    $A_{i\ (i+1:n,j)} \leftarrow A_{i-1\ (i+1:n,j)} - \rho_{i,j} A_{i-1\ (i+1:n,i)}$
    $R_{(i,j)} \leftarrow -\text{sign}(A_{i-1\ (i,i)}) \cdot \alpha_{i,j}$
  end for
  $y_i \leftarrow (0^T_{(1:i-1)}, 1, \frac{A_{i-1\ (i+1:n,i)}}{A_{i-1\ (i,i)}+\beta_i})^T$
end for

Let $e(x)$ be an upper bound for the numerical error when computing x. For the computation of dot products we can estimate the error with

$$e(D_{(i,i)}) \leq \sum_{l=1}^{n} (a_{i\ (l)}^2 \epsilon) \leq (a_i^T a_i)\epsilon = (\alpha_{1,i}^2 + \alpha_{2,i}^2 + \cdots + \alpha_{i,i}^2)\,\epsilon, \text{ and} \tag{17}$$

$$\begin{aligned} e(D_{(j,i)}) \leq & \sum_{l=1}^{n} (a_{i\ (l)} a_{j\ (l)} \epsilon) \leq \|a_i\| \|a_j\|\, \epsilon \\ = & \sqrt{(\alpha_{1,j}^2 + \alpha_{2,j}^2 + \cdots + \alpha_{j,j}^2)(\alpha_{1,i}^2 + \alpha_{2,i}^2 + \cdots + \alpha_{i,i}^2)}\,\epsilon, \end{aligned} \tag{18}$$

for the diagonal and non-diagonal elements of D respectively.

Out of Algorithm 5 and Equation (17) and (18) we can now derive error bounds for $\alpha_{i,i}$ and $\alpha_{j,i}$ with $j < i$:

$$e(\alpha_{i,i}^2) = e(D_{(i,i)}) + \sum_{j=1}^{i-1} e(\alpha_{j,i}^2) \tag{19}$$

$$e(\alpha_{j,i}) = \frac{e(D_{(j,i)}) + \sum_{l=1}^{j-1} (e(\alpha_{l,i}) e(\alpha_{l,j}))}{|\alpha_{j,j}|}. \tag{20}$$

For the first column of R we can estimate the error with $e(\alpha_{1,1}^2) = O(\alpha_{i,i}^2 \epsilon)$ leading to a relative error of

$$\frac{e(\alpha_{1,1}^2)}{\alpha_{1,1}^2} = O(\epsilon) \ll 1. \tag{21}$$

Considering that the relative error of all preceding diagonal elements $\alpha_{j,j}$ is bounded by $O(\epsilon)$,

we can use the fact that

$$\frac{\sqrt{(\alpha_{1,j}^2 + \alpha_{2,j}^2 + \cdots + \alpha_{j,j}^2)}}{|\alpha_{j,j}|} = O(1), \text{ and} \tag{22}$$

$$\frac{e(\alpha_{l,j})}{|\alpha_{j,j}|} = O(1) \text{ for all } l < j < i \tag{23}$$

to simplify Equation (20):

$$e(\alpha_{j,i}) = O\left(\sqrt{(\alpha_{1,i}^2 + \alpha_{2,i}^2 + \cdots + \alpha_{i,i}^2)}\,\epsilon\right) + \sum_{l=1}^{j-1} O(e(\alpha_{l,i})). \tag{24}$$

This result, in turn, allows us to simplify Equation (19):

$$e(\alpha_{i,i}^2) = O((\alpha_{1,i}^2 + \alpha_{2,i}^2 + \cdots + \alpha_{i,i}^2)\,\epsilon) \tag{25}$$

Please note that we assumed the maximal blocking factor k to be a constant.

To guarantee similar accuracy compared to the non-blocked QR-decomposition, we limit the relative error by claiming

$$\frac{\alpha_{1,i}^2 + \alpha_{2,i}^2 + \cdots + \alpha_{i-1,i}^2}{\alpha_{i,i}^2} \leq \epsilon_{\text{fallback}}. \tag{26}$$

Finally, we can set a criterion for the numerical stability of generating and applying the i-th Householder transformation: the i-th Householder vector is regarded as "stable" if Householder transformation $i-1$ is stable and Equation (26) is fulfilled. We set $\epsilon_{\text{fallback}}$ to 1 to avoid any substantial accuracy losses.

Using this criterion we can define the maximal number of columns to decompose in one step.

4. Benchmark results

In this section we provide performance results on a BlueGene/P with 16384 cores and a Power6 system with 6624 cores. Both systems are located at the Rechenzentrum Garching (RZG).

All performance tests are carried out within the symmetric eigensolver ELPA. To benchmark the new QR-decomposition we solve an eigenproblem of size $n = 27069$, resulting from a matrix in quantum chemistry (Poly27069). Within the eigensolver the QR-decomposition is called on many different sized tall and skinny matrices, as can be seen from Algorithm 1. We compare the newly developed blocked Householder-CholeskyQR, using two different blocking factors (rank-2 and rank-16), to the ScaLAPACK routine PDGEQRF (Netlib implementation compiled on top of IBM's ESSL) and the original ELPA code using an own implementation of the classical Householder QR-decomposition.

The ELPA solver uses a 2D block-cyclic distribution of the input matrix with a block-size of 16. The intermediate bandwidth b of the two-step tridiagonalization is set to 32. As can be seen from Algorithm 1, in each iteration the QR-decomposition is called on a b-wide panel of the matrix. This means that, resulting from the block-cyclic data distribution, only 2 process columns of the 2D processor grid are effectively involved in the QR-decomposition. During our

	BlueGene/P [#cores]				Power6 [#cores]			
	512	1024	2048	4096	256	512	1024	2048
Full to band	53.6s	33.0s	22.5s	17.0s	18.0s	12.9s	8.0s	6.1s
thereof QR	10.8s	9.1s	6.3s	5.9s	3.5s	3.2s	3.1s	3.0s
	20.1%	27.6%	28.0%	34.7%	19.4%	24.8%	38.8%	49.2%

Table 2: Absolute and relative effort of the QR-decompositions during the reduction to banded form of Poly27069, using ELPA with the classical Householder QR-decomposition.

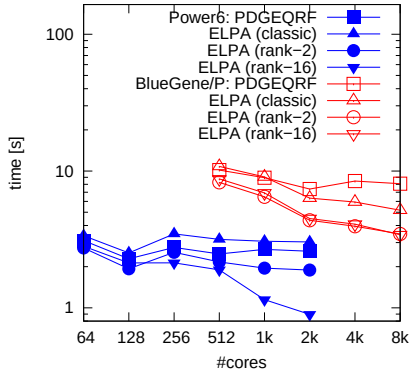

Figure 1: Accumulated runtime of the QR-decompositions during the reduction to banded form of Poly27069.

benchmarks we use core counts from 64 till up to 8192. For square numbers we use a processor grid which is a perfect square, e.g. 8×8 for 64 cores. For non square core counts we first refine in the row dimension, e.g. 32×16 for 512 cores. All computations are carried out using one MPI-process per core.

At first, we examine the importance of the QR-decomposition during the reduction to banded form. Table 2 shows the runtime of the reduction to banded form of Poly27069 and, thereof, the time required for QR-decompositions. We can see that the QR-decomposition gets increasingly relevant with higher numbers of processes. While this part of the algorithm requires about 20% of the time on 512 cores of the BlueGene/P and on 256 cores of the Power6 system, this ratio grows to 35% on 4096 BlueGene/P cores and to 49% on 2048 cores of the Power6. The results clearly show that the QR-decomposition is the limiting factor of the reduction to banded form if high scalability is required. The bottleneck QR-decomposition is more pronounced on the Power6. This may be explained with the higher single-core performance of the Power6 which implies that network latency issues become even more relevant.

In Figure 1 we compare different implementations of the QR-decomposition (PDGEQRF, ELPA unblocked, ELPA rank-2, and ELPA rank-16). The plot shows the accumulated runtime of all QR-decompositions during the reduction to banded form of Poly27069. Due to the tininess of the problems in relation to the number of processes, the scaling is expectably poor. However, blocked Householder-CholeskyQR leads to noticeable speedups, whereas the unblocked algorithms don't scale at all.

In Figure 2 we show detailed timings for the different QR-decomposition calls during the band reduction of Poly27069. The timings are plotted for the runs with the highest core count on each system. The reduction to banded form of Poly27069 consists of $n/b - 1 = 845$ iterations,

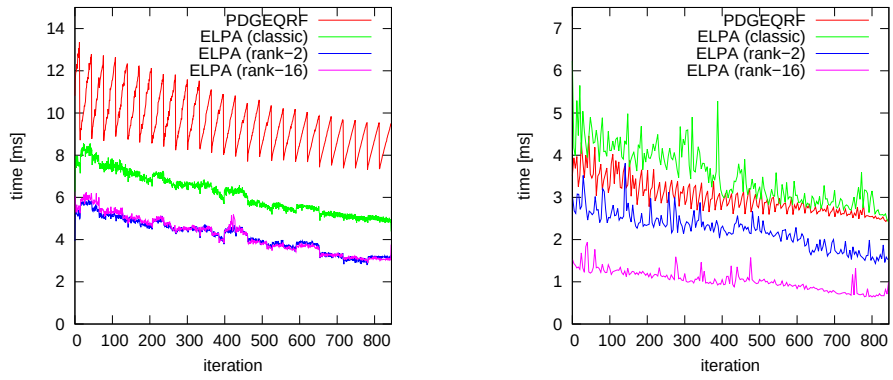

Figure 2: Timings of the occurring QR-decompositions during the reduction to banded form of Poly27069. In iteration i a QR-decomposition of a matrix of size $(n - ib - b) \times 32$ is performed. Left: BlueGene/P with 8192 cores (ELPA rank-16 and ELPA rank-2 show the best performance (overlapping), followed by ELPA classic and PDGEQRF), right: Power6 with 2048 cores (ELPA rank-16 shows the best performance, followed by ELPA rank-2, PDGEQRF, and ELPA classic).

whereas in each iteration i a matrix of size $(n - ib - b) \times 32$ has to be decomposed. The gap between the individual variants seems to be rather constant over the iterations of the algorithm. This means, in other words, that the speedup of blocked Householder-CholeskyQR is higher for smaller matrices. Again, the blocked QR-decomposition seems to be more profitable on the Power6, compared to the BlueGene/P.

For the presented performance results we disabled the fallback mechanism and ignored the accuracy of the results. In the next plots (Figure 3 and 4) we will analyze accuracy issues and fallback statistics.

For the measurements in Figure 3 we construct ill conditioned matrices of the following form. We construct a matrix A by multiplying Q' and R'. Q' is an arbitrary orthogonal matrix. R' is upper triangular and has entries of 1 on the diagonal. The values above the diagonal are set to "odiag/diag" which ranges from 1e-08 to 1e+08 during the test. We compare runtime and accuracy of the results for different values of odiag/diag and use the orthogonality $(I - Q^T Q)$ and the residual error $(A - QR)$ of the result as measures for the accuracy. In Figure 3 we compare blocked Householder-CholeskyQR with no fallback (left), blocked Householder-CholeskyQR with $\epsilon_{\text{fallback}} = 1$ (middle), and the ScaLAPACK QR-decomposition PDGEQRF (right). As expected, the ScaLAPACK QR-factorization produces accurate results for all examined matrices. On the other hand the blocked QR-decomposition with no fallback is very fast. However, if odiag/diag is larger than 1, the algorithm gets numerically unstable. With the conservative choice of $\epsilon_{\text{fallback}} = 1$ we profit from both, the efficiency of the blocked execution if the matrix is well conditioned and the numerical stability of the classical Householder QR-decomposition if the matrix is ill conditioned.

In the next plot we investigate how often such fallbacks to lower blockings occur for real matrices. In Figure 4 we can see blocking statistics during the reduction to banded form of Poly27069 (right) and a random matrix of size 27069 (left). The maximal blocking is set to 16. For both matrices the algorithm can perform full blocking most of the time. There were no observable differences regarding accuracy between the classical solver and the solver using blocked Householder-CholeskyQR. These are very promising results. However, more testing

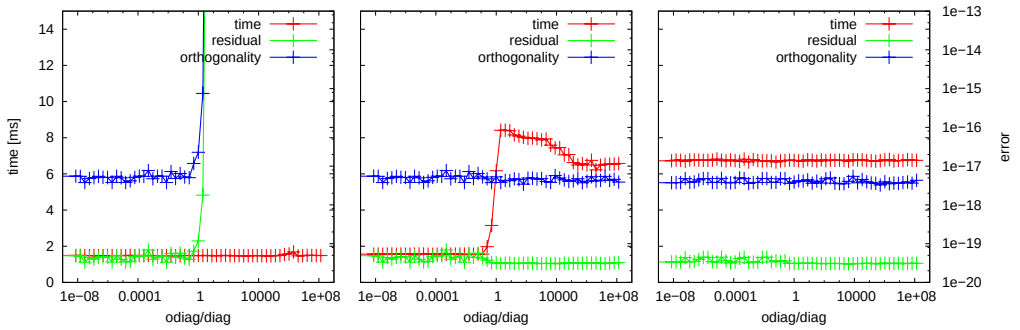

Figure 3: Runtime and accuracy while computing the QR-decomposition of $A = QR$, where R has the following structure: values of odiag/diag above the diagonal and values of 1 on the diagonal. Left: blocked Householder-CholeskyQR with no fallback (runtime continuously at 2ms, residual and orthogonality deteriorate for $odiag/diag > 1$), middle: blocked Householder-CholeskyQR with $\epsilon_{\text{fallback}} = 1$ (runtime switches from 2ms to 6ms at $odiag/diag \sim 1$, residual and orthogonality continuously at 1e-19 and 1e-17 respectively), right: ScaLAPACK QR-decomposition (runtime, residual and orthogonality at 6ms, 1e-19 and 1e-17 respectively).

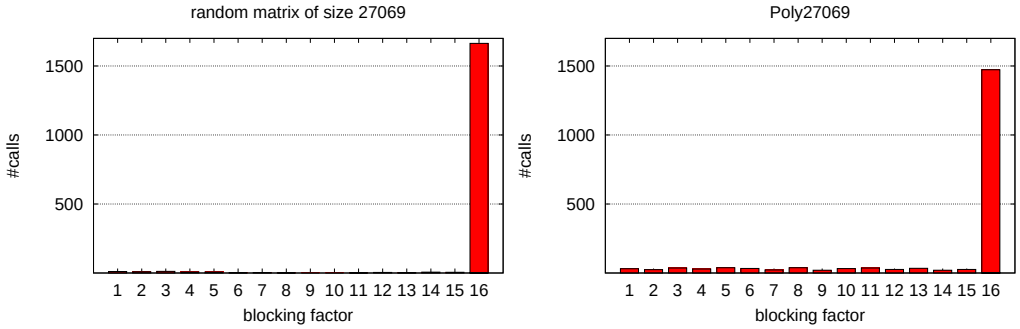

Figure 4: Blocking statistics of blocked Householder-CholeskyQR during the reduction to banded form. The maximal blocking factor was set to 16. $\epsilon_{\text{fallback}}$ was set to 1. Left: random matrix of size 27069, right: Poly27069.

with a broader set of matrices has still to be done.

5. Conclusion

We have presented a new blocking strategy for the parallel QR-decomposition which significantly reduces the synchronization requirements. The algorithm has been developed for the parallel symmetric eigensolver ELPA, but can be used wherever a scalable QR-decomposition of tall and skinny matrices is required. Moreover, the implementation can be used as a basic building block for the QR-decomposition of general rectangular matrices.

In each step, the algorithm generates and applies k Householder transformations instead of one. This reduces the number of messages for the execution on a distributed memory system. We have defined a stability criterion for the algorithm. If this criterion is not fulfilled, we switch back to lower blocking such that numerical stability is guaranteed.

Benchmarks on a BlueGene/P and a Power6 system have shown that the new implementation clearly outperforms the QR-decompositions in ScaLAPACK (routine PDGEQRF) and ELPA.

Acknowledgment

The authors would like to thank Rainer Johanni for fruitful comments on accuracy issues and the Rechenzentrum Garching for providing their computing infrastructure.

References

[1] A. Stathopoulos, K. Wu, A block orthogonalization procedure with constant synchronization requirements, SIAM J. Sci. Comput 23.

[2] E. Breitmoser, A. G. Sunderland, A performance study of the PLAPACK and ScaLAPACK eigensolvers on HPCx for the standard problem, in: Technical Report from the HPCx Consortium, 2003.

[3] K. Stanley, Execution time of symmetric eigensolvers, Ph.D. thesis, University of California at Berkeley (1997).

[4] T. Auckenthaler, V. Blum, H.-J. Bungartz, T. Huckle, R. Johanni, L. Krämer, B. Lang, H. Lederer, P. Willems, Parallel solution of partial symmetric eigenvalue problems from electronic structure calculations, Parallel Computing.

[5] T. Auckenthaler, H.-J. Bungartz, T. Huckle, L. Krämer, B. Lang, P. R. Willems, Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem (2010).

[6] ELPA documentation, `http://elpa-lib.fhi-berlin.mpg.de`, accessed 09-14-2012.

[7] C. Bischof, B. Lang, X. Sun, Parallel tridiagonalization through two-step band reduction, in: Proc. Scalable High-Performance Computing Conf., IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 23–27.

[8] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-optimal parallel and sequential QR and LU factorizations, SIAM J. Sci. Comput. 34 (1) (2012) 206–239. doi:10.1137/080731992.

[9] J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-avoiding parallel and sequential QR and LU factorizations, Technical Report No. UCB/EECS-2008-89.

[10] J. Dongarra, M. Faverge, T. Herault, J. Langou, Y. Robert, Hierarchical QR factorization algorithms for multi-core cluster systems, in: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 607–618. doi:10.1109/IPDPS.2012.62.

[11] W. Gander, Algorithms for the QR decomposition, Seminar für Angewandte Mathematik: Research report, 1980.

[12] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. C. Whaley, ScaLAPACK Users' Guide, SIAM, Philadelphia, PA, 1997.

[13] T. Auckenthaler, Highly scalable eigensolvers for petaflop applications, Ph.D. thesis, Technische Universität München (2012).

[14] R. Schreiber, C. van Loan, A storage-efficient WY representation for products of Householder transformations, SIAM J. Sci. Stat. Comput. 10 (1989) 53–57. doi:10.1137/0910005.