Noname manuscript No.
(will be inserted by the editor)

Tall-and-Skinny QR Factorization with Approximate Householder Reflectors on Graphics Processors

Andrés E. Tomás · Enrique S. Quintana-Ortí

the date of receipt and acceptance should be inserted later

Abstract We present a novel method for the QR factorization of large tall-and-skinny matrices on a hybrid platform equipped with a graphics processor. This approach uses an approximation technique for computing the Householder vectors during the factorization. Its main advantage over the conventional factorization is the reduced amount of data transfers between the graphics accelerator and the main memory of the host.

Our experiments show that, for tall-skinny matrices, the new approach outperforms the code in MAGMA by a large margin, while it is very competitive for square matrices when the memory transfers and CPU computations are the bottleneck of the Householder QR factorization.

Keywords QR factorization · Tall-and-skinny matrices · GPU · Householder vector · look-ahead · high-performance

1 Introduction

There exist relevant applications that require the computation of an orthonormal basis for a relatively small set of very long vectors, which form a "tall-and-skinny" (TS) matrix. This type of problem appears, among others, in the orthogonalization

Andrés E. Tomás
Dept. d'Enginyeria i Ciència dels Computadors
Universitat Jaume I, 12071 Castelló de la Plana, Spain
Dept. de Sistemes Informàtics i Computació
Universitat Politècnica de València, 46022 València, Spain
E-mail: tomasan@uji.es

Enrique S. Quintana-Ortí
Dept. d'Informàtica de Sistemes i Computadors
Universitat Politècnica de València, 46022 València, Spain
E-mail: quintana@disca.upv.es

in Krylov subspace methods [12]; in the analysis of big data applications characterized with a few descriptors only (e.g., large datasets with a few variables produced by long data acquisitions of several sensors) [2, 9]; and as a preprocessing step when computing the singular value decomposition (SVD) of a matrix [8] with many more rows than columns.

The conventional blocked QR factorization based on Householder reflectors [8], hereafter QRF-H, is not an efficient algorithm for the factorization of tall-and-skinny (TS) matrices on modern parallel processors; see, e.g., [4]. The reason is that, for matrices that have few columns but a large number of rows, the fraction of work of the QRF-H procedure that can be cast in terms of kernels from the Level-3 of BLAS (basic linear algebra subprograms [5]), as part of the highly parallel and efficient trailing update, cannot compensate the overwhelming cost of the panel factorization, which is performed via the much slower Level-1 and Level-2 BLAS.

The TSQR-H [4] algorithm follows the ideas of the incremental QRF-H [9] to obtain a method that is especially appealing for the factorization of TS matrices. In this procedure, the panel is split into small "square-like" blocks, and a QRF-H is computed for each block. These small QRF-H are then merged by pairs, using a structure-aware version of QRF-H. This merge procedure can follow a linear scheme [9] or, in parallel machines, a recursion tree [4], yielding a communication-reduction scheme with a considerable higher level of parallelism than traditional QRF-H.

The Cholesky QR factorization [14] (QRF-C) is an alternative that reduces the amount of communications but, unfortunately, often suffers from orthogonality loss. The use of mixed precision (in particular, the combination of 64-bit and 128-bit floating point arithmetic) in [19] can improve the accuracy of QRF-C, but its implementation is not efficiently supported by current hardware. A simpler solution is to operate only in standard double precision but perform a second step of QRF-C to improve the orthogonality [7, 18]. This result connects neatly with the classical "twice is enough" rule for Gram-Schmidt re-orthogonalization. However, as the number of vectors grows, the cost of QRF-C increases cubically. In addition, for very large problems the conditioning of the Gram matrix can become too large and a second "pass" may not be sufficient.

In this paper we propose a blocked QR factorization with approximate Householder vectors (QRF-AH). The current approach in QRF-H computes the dot products between pairs of panel columns after each Householder vector is generated and applied. The proposed alternative computes these dot products at the beginning of the panel factorization and updates them after the computation of each Householder vector. This update can be cheaply computed by exploiting the orthogonal relation between two consecutive column orthogonalizations but, due to rounding errors, the computed values are approximations of the actual dot products: hence the name of *approximate Householder vectors*. This alternative is similar to the column norm update used in the QR factorization with pivoting (QRP) [3]. The advantage of this technique is that it only requires to access the whole panel twice: before the panel factorization commences and once that is it completed. A hybrid CPU-GPU implementation can then take advantage of this property to transfer only two small square blocks of order b, where b stands for the algorithmic block size, instead of the whole $\hat{m} \times b$ panel for

each block of columns to be factorized. This is particularly relevant for TS matrices, as $b \ll \hat{m}$, and this transfer stands in the critical path of the algorithm.

The rest of the paper is organized as follows. Section 2 reviews the conventional blocked algorithm for the QR factorization and Section 3 presents the details of the QRF-AH algorithm. Next, Sections 4 and 5 respectively provide numerical and performance evaluations of the new method in comparison with a state-of-the-art hybrid library for CPU-GPU platforms. Finally, Section 6 summarizes the contributions of this work and suggests a few future lines of research.

2 Householder QR Factorization

The compact QR factorization of a TS matrix $A \in \mathbb{R}^{m\times n}$, with $m \gg n$, is given by

$$A = QR,$$

where $Q \in \mathbb{R}^{m\times n}$ has orthonormal columns and $R \in \mathbb{R}^{n\times n}$ is upper triangular.

Algorithm 1 (QRF-H) presents a blocked implementation of the QR factorization on a hybrid platform equipped with a CPU and a GPU accelerator. The input matrix A is assumed to be initially stored in the GPU memory. The orthonormal matrix Q is not built explicitly, but maintained implicitly as a collection of Householder vectors. In practice, these orthonormal vectors are stored in the strictly lower triangular part of the matrix A while the upper triangular factor R overwrites the corresponding entries of A. Therefore, upon completion, these data reside in the GPU memory. For simplicity we hereafter assume that the number of columns n is an integer multiple of the algorithmic block size b.

Algorithm 1 Blocked QRF-H factorization for CPU + GPU

Input: $A \in \mathbb{R}^{m\times n}$
Output: $R \in \mathbb{R}^{n\times n}$ (upper triangle of A), $V \in \mathbb{R}^{m\times n}$ (strictly lower triangle of A)
1 **for** $k = 1$ **to** n **in steps of** b
2 Define the (current) panel $A_P = A_{k:m,k:k+b-1}$
3 Define the trailing submatrix $A_S = A_{k:m,k+b:n}$
4 Send A_P to the CPU
5 Compute V and R from A_P in the CPU
6 Send V and R to the GPU
7 Compute T^{-1} from V
8 $A_S := (I + VTV^T)^T A_S$
9 **end**

In this algorithm, the QR decomposition of the panel is computed on the CPU (step or line 5) and the trailing matrix update is performed on the GPU (step 8). The QR factorization can be performed in the CPU via the LAPACK routine `xGEQRF`, which may also be blocked depending on value of b. The algorithm requires transferring the panels to the CPU ($(k-m+1)\times b$ elements per iteration, for an approximate total of $mn/2$ elements) and retrieving onto the GPU the Q and R factors for the panels

(also $mn/2$ elements in total). Due to all those transfers the algorithm is only competitive if the CPU/GPU communications and CPU computations are overlapped with the computations on the GPU. This can be achieved via a look-ahead technique which, for simplicity, is not included in the algorithm. For the details, see, e.g., [15, 17].

Algorithm 1 differs form the current implementations in LAPACK and MAGMA in the computation of the triangular matrix T used to block the Householder reflectors [13]. Concretely, those libraries compute T with the routine `xLARFT`, which is based in level-2 matrix vector multiplications, and is computed on the CPU in MAGMA. In contrast, Algorithm 1 follows [11, 10] to compute T^{-1} instead of T:

$$T^{-1}_{i,j} = \begin{cases} u_i^T u_j & \text{if } i < j \\ \frac{u_i^T u_i}{2} & \text{if } i = j \\ 0 & \text{if } i > j \end{cases}$$

This computation can be performed efficiently on the GPU via a level-3 `xSYRK` operation and a small diagonal update [10]. When operating with T^{-1}, the application of the blocked Householder transformation requires solving a triangular system instead of a triangular matrix multiplication, but this is not an important issue because both operations are rather efficient on the GPU. Computing T^{-1} on the GPU also diminishes slightly the amount of communication. More importantly, it reduces the amount of work on the CPU and yields a more efficient overlapping [16].

3 QR Factorization with Approximate Householder vectors

At each iteration of Algorithm 1, the CPU needs to retrieve the panel $A_P = A_{k:m,k:k+b-1}$ prior to computing its QR factorization in order to obtain the corresponding triangular factor R and the corresponding Householder vectors in V. The caveat is that, for TS matrices, the overhead of retrieving this panel from the GPU (memory) has to be added to the cost of factorizing the panel itself, and together they can have a significant impact on the execution time, even for a realization that includes look-ahead.

In this section, we first introduce our new techique to reduce the CPU-GPU communication overhead. Next, we explain how to integrate this communication-reducing technique into the blocked algorithm, and the section is finally closed with a brief discussion of how to avoid negative cancellation artifacts.

3.1 Panel factorization

The results of the factorization of the panel $A_P \in \mathbb{R}^{\hat{m} \times b}$, with $\hat{m} = m - k + 1$, comprise the triangular factor $\mathbb{R}^{b \times b}$, plus the Householder reflectors, given by $V \in \mathbb{R}^{\hat{m} \times b}$, that correspond to the compact representation of the orthogonal factor. Algorithm 1 computes these results using the full panel A_P, which requires the transfer of $\hat{m}b$ numbers between the GPU and the CPU. Once the panel is factorized, the factors R and V have to be sent back to the GPU, requiring a transfer of $\hat{m}b$ additional numbers.

We discuss next how to compute both R and (the top $b \times b$ part of) V in the CPU, using the data in $A_T, B = A_P^T A_P \in \mathbb{R}^{b\times}$ only. These two $b \times b$ blocks reside in the GPU and transferred to the CPU before the panel factorization commences. Once R and (part of) V are obtained in the CPU, they are sent back to the GPU, yielding the reduced transfer overhead of $4b^2$ numbers. As $m \gg b$ for TS matrices, the savings can be notable.

Consider the following partitionings of the panel to be factorized during a given iteration k:

$$A_P = \left[\, a_1 \middle| a_2 \middle| \ldots \middle| a_b \,\right] = \left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \ldots & \alpha_b \\ \hline \hat{a}_1 & \hat{a}_2 & \ldots & \hat{a}_b \end{array}\right] = \left[\begin{array}{c} A_T \\ \hline A_B \end{array}\right], \tag{1}$$

where $a_j \in \mathbb{R}^{\hat{m}}$, α_j is a scalar, and $\hat{a}_j \in \mathbb{R}^{\hat{m}-1}$, for $j = 1, 2, \ldots, b$; moreover, $A_T \in \mathbb{R}^{b\times b}$ and $A_B \in \mathbb{R}^{(\hat{m}-b)\times b}$. Also, let $B = A_P^T A_P$, with the (i, j) entry of this matrix given by $\beta_{i,j} = a_i^T a_j$, for $i, j = 1, 2, \ldots, b$; and note that the entries in the first row of B satisfy

$$\beta_{1,j} = a_1^T a_j = \alpha_1 \alpha_j + \hat{a}_1^T \hat{a}_j, \quad j = 1, 2, \ldots, b. \tag{2}$$

Consider also analogous partitionings of V as

$$V = \left[\, v_1 \middle| v_2 \middle| \ldots \middle| v_b \,\right] = \left[\begin{array}{c|c|c|c} 1 & 0 & \ldots & 0 \\ \hline \hat{v}_1 & \hat{v}_2 & \ldots & \hat{v}_b \end{array}\right] = \left[\begin{array}{c} V_T \\ \hline V_B \end{array}\right], \tag{3}$$

where $v_j \in \mathbb{R}^{\hat{m}}$, $\hat{v}_j \in \mathbb{R}^{\hat{m}-1}$, for $j = 1, 2, \ldots, b$; furthermore, $V_T \in \mathbb{R}^{b\times b}$ is unit lower triangular and $V_B \in \mathbb{R}^{(\hat{m}-b)\times b}$.

The factorization of the panel A_P proceeds columnwise left to right. From the first column (i.e., the leftfmost), the procedure needs to obtain the top-left entry of R, referred to as $\rho_{1,1}$, and the first column of V. Now, for the QR factorization, we know that

$$\rho_{1,1} = -\operatorname{sgn}(\alpha_1)\|a_1\|_2 = -\operatorname{sgn}(\alpha_1)\sqrt{\beta_{1,1}}, \tag{4}$$

where $\|\cdot\|_2$ denotes the vector 2-norm. This expression exposes that the calculation of $\rho_{1,1}$ does not require the full column a_1 as $\beta_{1,1}$ is available as part of B. In addition, the Householder reflector v_1 is given by

$$\hat{v}_1 = \hat{a}_1/\sigma_1, \tag{5}$$

with

$$\sigma_1 = \alpha_1 + \operatorname{sgn}(\alpha_1)\|a_1\|_2 = \alpha_1 + \operatorname{sgn}(\alpha_1)\sqrt{\beta_{1,1}}. \tag{6}$$

The Householder reflector v_1 has to be applied next to the second, third,...,b-th columns of the panel, as in

$$\begin{aligned} a_j &:= (I + v_1 v_1^T) a_j = a_j + v_1 (v_1^T a_j) = a_j + v_1 \omega_{1,j} \\ &= a_j + \left[\begin{array}{c} 1 \\ \hline \hat{v}_1 \end{array}\right] \omega_{1,j} = a_j + \left[\begin{array}{c} 1 \\ \hline \hat{a}_1/\rho_{1,1} \end{array}\right] \omega_{1,j}, \quad j = 2, 3, \ldots, b, \end{aligned} \tag{7}$$

where I denotes an square identity of the appropriate order. Furthermore, from (5) and (2), we derive that the scalar $\omega_{1,j}$ satisfies

$$\begin{aligned} \omega_{1,j} = v_1^T a_j &= \left[\, 1 \,\middle|\, \hat{v}_1^T \,\right] \left[\frac{\alpha_j}{\hat{a}_j}\right] \\ &= \alpha_j + \hat{v}_1^T \hat{a}_j = \alpha_j + (\hat{a}_1^T/\rho_{1,1})\hat{a}_j = \alpha_j + (\beta_{1,j} - \alpha_1\alpha_j)/\rho_{1,1}, \end{aligned} \tag{8}$$

which again only needs entries from B and A_T for its calculation.

The key to the communication-reduction property of the algorithm is to defer the update of those entries of A_B in equations (5) and (7) until the panel factorization is completed. Algorithm 2 details this procedure for computing the QR decomposition of the panel using the approximate Householder vectors.

Algorithm 2 Single vector QRF-AH factorization

Input: $A_T \in \mathbb{R}^{b\times b}$, $B \in \mathbb{R}^{b\times b}$
Output: $R \in \mathbb{R}^{b\times b}$ and $V_T \in \mathbb{R}^{b\times b}$ (respectively overwritting the upper/strictly lower triangule of A), $D \in \mathbb{R}^{b\times b}$

1 $\tilde{v}_k := v_k := \sqrt{\beta_{k,k}}, \quad k = 1, 2, \ldots, b$
2 **for** $k = 1$ **to** b
3 /* *Compute the k-th Householder reflector* */
4 $\sigma_k := a_{k,k} + \mathrm{sgn}(a_{k,k})\tilde{v}_k$
5 $\tau_k := \frac{\mathrm{sgn}(a_{k,k})\rho}{a_{k,k}\tilde{v}_k}$
6 $a_{k,k} := -\mathrm{sgn}(a_{k,k})\tilde{v}_k$
7 $a_{1:b,k} := a_{1:b,k}/\sigma_k$
8 **for** $j = k + 1$ **to** b
9 /* *Apply Householder vector to columns k+1, k+2,...,b* */
10 $\omega_{k,j} := \beta_{k,j}/a_{k,k} - a_{k,j}$
11 $a_{k:b,j} := \omega_{k:b,j} a_{k:b,k} + a_{k:b,j}$
12 **end**
13 **for** $j = k + 1$ **to** b
14 /* *Norm update* */
15 $t := \max(0, 1 - (a_{k,j}/\tilde{v}_j)^2)$
16 **if** $t(\tilde{v}_j/v_j)^2 \leq \sqrt{\epsilon}$ **exit**
17 $\tilde{v}_j := \tilde{v}_j \sqrt{t}$
18 /* *Dot product update* */
19 $\beta_{k+1:b,j} := \beta_{k+1:b,j} - a_{k,j} a_{k,k+1:b}$
20 **end**
21 **end**
22 **for** $k = 1$ **to** b
23 /* *Compute D* */
24 $\delta_{k,k} := 1/\sigma_k$
25 **for** $j = k + 1$ **to** b
26 $\delta_{j,k} := \delta_k^T \omega_j/\sigma_k$
27 **end**
28 **end**

The first step of Algorithm 2 computes the norm of each panel column ν_i and its corresponding initial approximation $\tilde{\nu}_i$ from $\beta_{i,i}$. The algorithm also maintains the original norms in ν_i, in order to leverage the same norm-update formula described in [6] for the QR with pivoting. Steps 4–7 compute the Householder vectors as defined in (5), but using the equivalent $\tilde{\nu}_i$ instead of $\sqrt{\beta_{i,i}}$. As per convention, when operating with Householder reflectors, the sign of $a_{k,k}$ is taken into account to avoid cancellation errors.

Steps 10 and 11 of Algorithm 2 apply the approximate Householder vector using (7). The values of ρ and w are stored and accumulated in a lower triangular matrix $D \in \mathbb{R}^{b\times b}$ in steps 22–28. This matrix is used to update the bottom $\hat{m} - b$ entries of A_P with the matrix multiplication $A_B := A_B D$.

Steps 15–17 update the norm approximation $\tilde{\nu}_i$ and test for severe cancellation errors using the strategy in [6]. Step 19 updates the approximations stored in B exploiting the orthogonality of Householder reflectors. Concretely, assume that H stands for the Householder transformation that introduces zeros below the diagonal in the first column of A_P and define

$$\tilde{A} = HA_P = H\left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \dots & \alpha_b \\ \hline \hat{a}_1 & \hat{a}_2 & \dots & \hat{a}_b \end{array}\right] = \left[\begin{array}{c|c|c|c} \tilde{\alpha}_1 & \tilde{\alpha}_2 & \dots & \tilde{\alpha}_b \\ \hline 0 & \tilde{a}_2 & \dots & \tilde{a}_b \end{array}\right] \tag{9}$$

Then each element of B, which corresponds to a dot product between a pair of columns of $\hat{A}$, satisfies that

$$\beta_{i,j} = \alpha_i\alpha_j + \hat{a}_i^T\hat{a}_j = \tilde{\alpha}_i\tilde{\alpha}_j + \tilde{a}_i^T\tilde{a}_j = \tilde{\alpha}_i\tilde{\alpha}_j + \tilde{\beta}_{i,j}. \tag{10}$$

Now, for the next column of the panel factorization, we will need to use the dot products $\tilde{\beta}_{i,j}$, which can be cheaply computed from the update formula

$$\tilde{\beta}_{i,j} = \beta_{i,j} - \tilde{\alpha}_i\tilde{\alpha}_j, \quad i,j = 2,3,\dots,b. \tag{11}$$

3.2 Matrix factorization with approximate vectors

Algorithm 3 (QRF-AH) presents the blocked QR factorization using approximate Householder vectors. This algorithm has the same structure as QRF-H but instead of transferring the whole panel to the CPU to compute its QR decomposition, it uses Algorithm 2 to compute R and the top $b \times b$ part of V.

Algorithm 3 Blocked QRF-AH factorization for CPU + GPU

Input: $A \in \mathbb{R}^{m\times n}$
Output: $R \in \mathbb{R}^{n\times n}$ (upper part of A), $V \in \mathbb{R}^{m\times n}$ (lower part of A)
1 **for** $k = 1$ **to** n **in steps of** b
2 Define the top part of panel $A_T = A_{k:k+b-1,k:k+b-1}$
3 Define the bottom part of panel $A_B = A_{k+b:m,k:k+b-1}$
4 Define the trailing submatrix $A_S = A_{k:m,k+b:n}$
5 $B := \left[\, A_T^T \middle| A_B^T \,\right] \left[\begin{array}{c} A_T \\ \hline A_B \end{array}\right]$

6 Send A_T and B to the CPU
7 Compute R, V_T and D from A_T and B on the CPU; see Algorithm 2
8 Send R and D to the GPU
9 $V := \left[\frac{V_T}{A_B D}\right]$
10 Compute T^{-1} from V
11 $A_S := (I + VTV^T)^T A_S$
12 **end**

Step 7 of Algorithm 3 computes the QR factorization of the panel from A_T and B. The triangular factor and approximate Householder vectors respectively overwrite the upper triangular and strictly lower triangular part of A_T. This procedure does not access A_B but instead uses the values from B. The lower part of the approximate Householder vectors, that is V_B, is computed in step 6 as a linear combination of D and the columns of A_B.

The main advantage of blocked QRF-AH over QRF-H is the reduced amount of memory transferred between the CPU and the GPU. The QRF-H algorithm requires the transfer of approximately $mn/2$ numbers while its QRF-AH counterpart transfers a total of $4nb$ numbers only where, in the target scenario, $b \leq n \ll m$.

3.3 Cancellation

When any of the columns of the panel A_P is close to being a linear combination of the rest, its correspondent diagonal element in R becomes very small. This means that the update formula in step 17 of Algorithm 2 can introduce severe cancellation errors. This condition can be detected by the test in step 16, which stops the algorithm at a given iteration, say $\hat{k}$. In this occurs case, Algorithm 2 returns the factors of a smaller QR decomposition, where the initial $\hat{k} - 1$ columns of R and V_T are still computed accurately. The QRF-AH procedure in Algorithm 3 then simply adjusts the block size $b = \hat{k} - 1$ for this iteration of the matrix factorization, applies the corresponding transformations (steps 8–11), and continues with the next panel starting at column $\hat{k}$ during the next iteration of loop k (while setting b back to the original block size). There will be no errors in the computation of the first column in a block as none of the norms is updated. Similarly, the approximation to dot products in step 19 of Algorithm 2 can also suffer from severe cancellation. However, there is no need to check for those errors as they will detected by the norm update and any previous error in that column of R will be discarded.

4 Numerical experiments

In this section, we assess the numerical behaviour of the new QRF-AH algorithm by running some numerical experiments with TS matrices specifically designed to produce a breakdown of the update formula. This stress test follows the approximation in [1] and should allow a fair comparison between the reliability of QRF-AH with that

Table 1 Numerical comparison of QRF-H and QRF-AH

	QR with exact Householder		QR with approximate Householder	
ρ	$\|\|Q^TQ - I\|\|_F$	$\frac{\|\|A-QR\|\|_F}{\|\|A\|\|_F}$	$\|\|Q^TQ - I\|\|_F$	$\frac{\|\|A-QR\|\|_F}{\|\|A\|\|_F}$
10^{-1}	9.154715×10^{-15}	9.591258×10^{-16}	9.925917×10^{-15}	7.210446×10^{-16}
10^{-2}	9.377855×10^{-15}	9.598223×10^{-16}	1.011355×10^{-14}	7.181040×10^{-16}
10^{-3}	9.244618×10^{-15}	9.601067×10^{-16}	9.984865×10^{-15}	7.195629×10^{-16}
10^{-4}	9.039978×10^{-15}	9.600339×10^{-16}	1.028484×10^{-14}	7.198046×10^{-16}
10^{-5}	9.001562×10^{-15}	9.575120×10^{-16}	9.901356×10^{-15}	7.190128×10^{-16}
10^{-6}	9.094931×10^{-15}	9.620550×10^{-16}	1.020415×10^{-14}	7.175174×10^{-16}
10^{-7}	9.013317×10^{-15}	9.592615×10^{-16}	1.024607×10^{-14}	7.210177×10^{-16}
10^{-8}	8.883850×10^{-15}	9.597328×10^{-16}	9.802340×10^{-15}	7.191490×10^{-16}
10^{-9}	9.337637×10^{-15}	9.603291×10^{-16}	9.594664×10^{-15}	7.207318×10^{-16}
10^{-10}	9.323021×10^{-15}	9.574503×10^{-16}	1.062224×10^{-14}	7.190121×10^{-16}
10^{-11}	8.877251×10^{-15}	9.572992×10^{-16}	1.014206×10^{-14}	7.190841×10^{-16}
10^{-12}	9.124937×10^{-15}	9.597074×10^{-16}	1.001635×10^{-14}	7.199647×10^{-16}
10^{-13}	8.931211×10^{-15}	9.561183×10^{-16}	1.031587×10^{-14}	7.192527×10^{-16}
10^{-14}	9.340513×10^{-15}	9.610956×10^{-16}	1.009986×10^{-14}	7.188647×10^{-16}
10^{-15}	9.570032×10^{-15}	9.615985×10^{-16}	9.988672×10^{-15}	7.203320×10^{-16}

of other alternative methods, as those in [1], whose implementation does not seem to be publicly available.

The test matrices are derived from the QR factorization of an $m \times n = 1000 \times 200$ matrix A with entries following a uniform random distribution in the interval $[0, 1)$. We then set $R_{100,100} = \rho$ in the upper triangular factor R, and multiply back Q and R to form $\tilde{A} := QR$. The parameter ρ controls the condition number of the assembled matrix, which is given by $\kappa(\tilde{A}) \approx 1/\rho$ so that, varying $\rho \in [10^{-1}, 10^{-15}]$, we obtain matrices with a condition number of up to 10^{15}.

Table 1 reports the orthogonality loss

$$||Q^TQ - I||_F \tag{12}$$

and the relative residual

$$\frac{||A - QR||_F}{||A||_F} \tag{13}$$

of the QR factorizations computed with the QRF-H and QRF-AH algorithms for matrices with different values of ρ. All the tests were performed in a Intel Xeon E5-2630 v3 processor using IEEE double-precision arithmetic. The QRF-H implementation corresponds to the `xGEQRF` routine form the Intel MKL 2019 library.

Table 1 shows that QRF-AH offers orthogonality and relative residuals quite similar to those of QRF-H. This remarkable numerical behaviour is due to the update formula breakdown detection mechanism. Specifically, in this experiment the fail-safe detection on the update only triggers once for each one of the matrices with $\rho \leq 10^{-4}$. This means that, among the $k(= \lceil n/b \rceil = \lceil 200/16 \rceil) = 13$ panel factorizations that have to be computed for each matrix, only the factorization of a single block had to be stopped early. We expect that, for real applications, the probability of an update breakdown will be even much lower.

5 Performance Evaluation

Hardware setup. In this section we compare the performance of QRF-AH and QRF-H on two distinct platforms equipped with two representative GPUs: a high end NVIDIA Tesla P100 (Pascal) and a cost-effective NVIDIA GeForce Titan X (Pascal). The P100 is paired with two Intel Xeon E5-2620 v4 processors (8+8 cores) while the Titan X is paired with an Intel Core i7-3770K CPU (4 cores).

Software setup. All codes are compiled with version 8.0 of the CUDA development environment. The optimized implementations of BLAS and LAPACK are those provided by NVIDIA cuBLAS 8.0 for the GPU and Intel MKL 2019 for the CPU. Hyper-threading is disabled in the CPUs, as suggested by the MKL library documentation. To reduce the variance of execution times, the number of threads for OpenMP and MKL is set to the number of physical cores and each thread is mapped statically to one core. The corresponding Intel C compiler 2019 was used to compile the code with the -O3 optimization level. Nevertheless, the optimizations made by the compiler are not very relevant for our study, because most of the performance-sensitive code is implemented inside the cuBLAS and MKL libraries. To avoid noise caused by other processes activity on the systems, the execution times reported next are the median values from ten executions. The execution times in all cases are very similar except for some initial, much slower executions due to dynamic libraries loading.

Input data. The input matrices are created following the same procedure described in the numerical tests in section 4, with random elements in range $[0, 1)$, and setting $\rho = 1$ so that the condition number is kept small. For brevity, for the platform with the P100 we report results in (IEEE) double precision only; the analysis using single precision on this platform offers similar conclusions. As the Titan X offers very low performance in double precision, we only employ single precision on that platform.

QRF implementations. The baseline GPU implementation of `xGEQRF` (QR factorization via Householder reflectors) is that available in the MAGMA library (version 2.2.0). Among the three variants of `xGEQRF` in MAGMA, we choose the first one as it is the only one with an LAPACK-compatible interface. Furthermore, the performance of the other two versions seems to be quite similar. The block size in QRF-AH was set to 1,024 which we found optimal for both platforms.

Look-ahead. Both the MAGMA-`xGEQRF` and our implementation of QRF-AH leverage integrate a look-ahead strategy [15, 17] to hide the cost of transfers and computations on the CPU. In rough detail, the look-ahead overlaps the transfer of the panel and the computation of its QR factorization on the CPU, with the update of the trailing submatrix (for the previous iteration) on the GPU. Unfortunately, this technique offers minor advantages for TS matrices, as for those problems, the trailing matrix often presents too few columns to attain high performance.

Evaluation. Figure 1 compares the performance of MAGMA-`xGEQRF` and our implementation of QRF-AH on both platforms. The y-axis shows the ratio between the

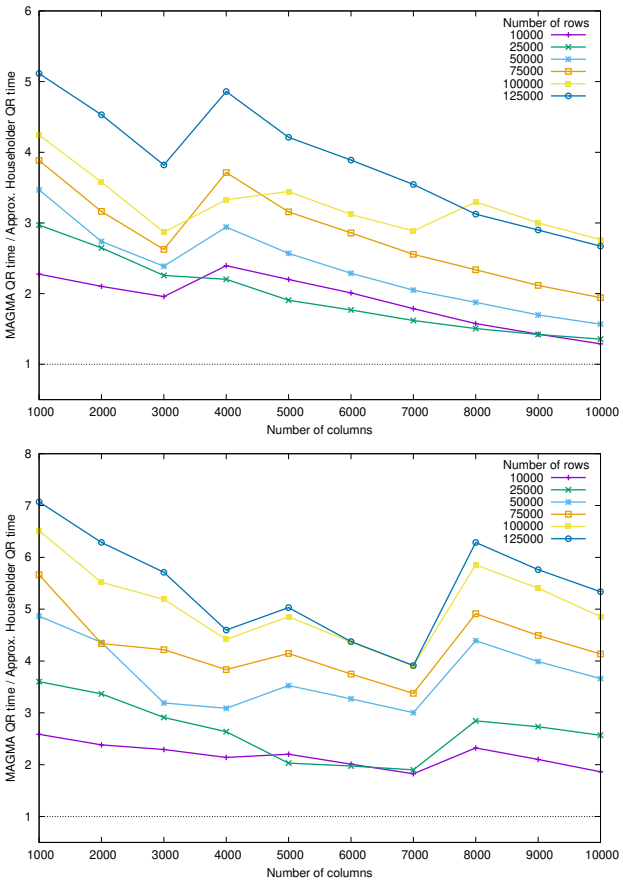

Fig. 1 Performance comparison of QRF-AH over MAGMA-`xGEQRF` for TS matrices on a P100 using double precision (top) and a Titan X using single precision (bottom)

execution time of MAGMA-`xGEQRF` divided by that of QRF-AH. Each line corresponds to a group of matrices with the same number of matirx rows (n) while the number of matrix columns (m) remains fixed as specified in the x-axis of the plot. The MAGMA implementation changes the block size when the number of columns is over 3000 in double precision and when it is over 7000 in single precision. This is the reason the plots show a small variation around those values. As expected, the performance of QRF-AH is much higher for TS matrices (up to 6 times) but this advantage diminishes as the gap between the number of columns and number of rows narrows. However, the performance drop is considerably less sharp in the Titan X platform, in part because the relative slow CPU drags the performance of MAGMA-`xGEQRF` and QRF-AH benefits from the reduced volume of communication.

Figure 2 compares the performance of MAGMA-`xGEQRF` and QRF-AH for square matrices. The effect of the reduced communications of QRF-AH is quite small in the system with the P100 because the CPU is fast enough to compute the panel factor-

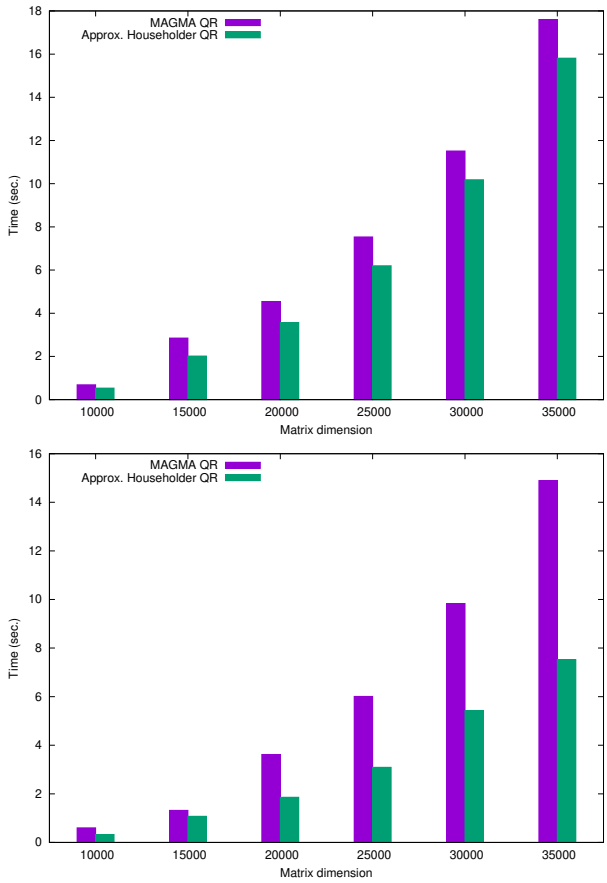

Fig. 2 Performance comparison of QRF-AH over MAGMA-`xGEQRF` for TS matrices on a P100 using double precision (top) and a Titan X using single precision (bottom)

ization and perform all the communications faster than the GPU performs the matrix update. However, the CPU in the platform with the Titan X is much slower and the look-ahead in MAGMA-`xGEQRF` is not well balanced. In that case the reduced communications of QRF-AH have a much more relevant impact and, as a result, this routine outperforms MAGMA-`xGEQRF` by a factor of two.

6 Conclusions

The new algorithm presented in this work, QRF-AH, is a variant of the conventional Householder-based QR factorization that uses an alternative formula to compute the Householder vectors. While the approximation underlying this formula can fail for very ill-conditioned matrices, these breakdowns can be easily detected and corrected by an early termination of the panel factorization.

The variant of QRF-AH with look-ahead can be efficiently implemented on a hybrid CPU-GPU system, with the CPU in charge of the panel decomposition while the rest of operations (all BLAS level-3) are performed on the GPU. The main advantage of this approach is the reduced volume of communications between CPU and GPU compared with the blocked QR implementations available in MAGMA. An additional advantage of QRF-AH is that its implementation is much simpler than other methods specifically designed for tall-skinny matrices as no custom GPU kernels are required. This also favours portability to new GPU architectures or even to different types of accelerators.

The performance of the new approach is very competitive for tall-skinny matrices, and even outperforms MAGMA for square matrices when memory transfers and CPU computations impose a strong performance bottleneck.

The stability of QRF-AH has been analyzed with matrices that explicitly enforce breakdowns. While this experiment shows that QRF-AH offers levels of orthogonality and relative error similar to those of the stable QRF-H, a theoretical analysis may help to fully understand the behaviour of the algorithm and devise future strategies for improve the accuracy of the update formula.

Acknowledgements This research was supported by the project TIN2017-82972-R from the *MINECO* (Spain), and the EU H2020 project 732631 "OPRECOMP. Open Transprecision Computing".

References

1. Ballard G, Demmel J, Grigori L, Jacquelin M, Knight N, Nguyen H (2015) Reconstructing Householder vectors from tall-skinny QR. Journal of Parallel and Distributed Computing 85:3 – 31, DOI 10.1016/j.jpdc.2015.06.003, iPDPS 2014 Selected Papers on Numerical and Combinatorial Algorithms
2. Benson AR, Gleich DF, Demmel J (2013) Direct QR factorizations for tall-and-skinny matrices in MapReduce architectures. In: 2013 IEEE International Conference on Big Data, pp 264–272, DOI 10.1109/BigData.2013.6691583
3. Businger P, Golub GH (1965) Linear least squares solutions by householder transformations. Numer Math 7(3):269–276, DOI 10.1007/BF01436084
4. Demmel J, Grigori L, Hoemmen M, Langou J (2012) Communication-optimal parallel and sequential QR and LU factorizations. SIAM J Sci Comput 34(1):206–239, DOI 10.1137/080731992
5. Dongarra J, Du Croz J, Hammarling S, Duff IS (1990) A set of level 3 basic linear algebra subprograms. ACM Trans Math Softw 16(1):1–17, DOI 10.1145/77626.79170
6. Drmač Z, Bujanović Z (2008) On the failure of rank-revealing qr factorization software – a case study. ACM Trans Math Softw 35(2):12:1–12:28, DOI 10.1145/1377612.1377616
7. Fukaya T, Nakatsukasa Y, Yanagisawa Y, Yamamoto Y (2014) CholeskyQR2: A simple and communication-avoiding algorithm for computing a tall-skinny QR factorization on a large-scale parallel system. In: 2014 5th Workshop on Lat-

est Advances in Scalable Algorithms for Large-Scale Systems, pp 31–38, DOI 10.1109/ScalA.2014.11
8. Golub G, Van Loan C (2013) Matrix Computations. Johns Hopkins Studies in the Mathematical Sciences, Johns Hopkins University Press
9. Gunter BC, van de Geijn RA (2005) Parallel out-of-core computation and updating the QR factorization. ACM Trans Math Softw 31(1):60–78, DOI 10.1145/1055531.1055534
10. Joffrain T, Low TM, Quintana-Ortí ES, Geijn Rvd, Zee FGV (2006) Accumulating householder transformations, revisited. ACM Trans Math Softw 32(2):169–179, DOI 10.1145/1141885.1141886
11. Puglisi C (1992) Modification of the Householder method based on the compact WY representation. SIAM Journal on Scientific and Statistical Computing 13(3):723–726, DOI 10.1137/0913042
12. Saad Y (2003) Iterative methods for sparse linear systems, 3rd edn. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA
13. Schreiber R, Van Loan C (1989) A storage-efficient WY representation for products of Householder transformations 10(1):53–57, DOI 10.1137/0910005
14. Stathopoulos A, Wu K (2001) A block orthogonalization procedure with constant synchronization requirements. SIAM J Sci Comput 23(6):2165–2182, DOI 10.1137/S1064827500370883
15. Strazdins P (1998) A comparison of lookahead and algorithmic blocking techniques for parallel matrix factorization. Tech. Rep. TR-CS-98-07, Department of Computer Science, The Australian National University, Canberra 0200 ACT, Australia
16. Tomás Dominguez AE, Quintana Orti ES (2018) Fast blocking of householder reflectors on graphics processors. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pp 385–393, DOI 10.1109/PDP2018.2018.00068
17. Volkov V, Demmel JW (2008) LU, QR and Cholesky factorizations using vector capabilities of GPUs. Tech. Rep. 202, LAPACK Working Note, URL http://www.netlib.org/lapack/lawnspdf/lawn202.pdf
18. Yamamoto Y, Nakatsukasa Y, Yanagisawa Y, Fukaya T (2015) Roundoff error analysis of the Cholesky QR2 algorithm. Electronic Transactions on Numerical Analysis 44:306–326
19. Yamazaki I, Tomov S, Dongarra J (2015) Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs. SIAM Journal on Scientific Computing 37(3):C307–C330, DOI 10.1137/14M0973773