SAND2013-9366C

Reconstructing Householder Vectors from Tall-Skinny QR

October 29, 2013

Grey Ballard[1], James Demmel[2], Laura Grigori[3], Mathias Jacquelin[4], Hong Diep Nguyen[2], and Edgar Solomonik[2]

[1]Sandia National Laboratories, Livermore, USA
[2]University of California, Berkeley, Berkeley, USA
[3]INRIA Paris - Rocquencourt, Paris, France
[4]Lawrence Berkeley National Laboratory, Berkeley, USA

Abstract

The Tall-Skinny QR (TSQR) algorithm is more communication efficient than the standard Householder algorithm for QR decomposition of matrices with many more rows than columns. However, TSQR produces a different representation of the orthogonal factor and therefore requires more software development to support the new representation. Further, implicitly applying the orthogonal factor to the trailing matrix in the context of factoring a square matrix is more complicated and costly than with the Householder representation.

We show how to perform TSQR and then reconstruct the Householder vector representation with the same asymptotic communication efficiency and little extra computational cost. We demonstrate the high performance and numerical stability of this algorithm both theoretically and empirically. The new Householder reconstruction algorithm allows us to design more efficient parallel QR algorithms, with significantly lower latency cost compared to Householder QR and lower bandwidth and latency costs compared with Communication-Avoiding QR (CAQR) algorithm. As a result, our final parallel QR algorithm outperforms ScaLAPACK and Elemental implementations of Householder QR and our implementation of CAQR on the Hopper Cray XE6 NERSC system.

1 Introduction

Because of the rising costs of communication (*i.e.*, data movement) relative to computation, so-called *communication-avoiding* algorithms—ones that perform roughly the same computation as alternatives but significantly reduce communication—often run with reduced running times on today's architectures. In particular, the standard algorithm for computing the QR decomposition of a tall and skinny matrix (one with many more rows than columns) is often bottlenecked by communication costs. A more recent algorithm known as Tall-Skinny QR (TSQR) is presented in [1] (the ideas go back to [2]) and overcomes this bottleneck by reformulating the computation. In fact, the algorithm is communication optimal, attaining the lower bound for communication costs of QR decomposition (up to a logarithmic factor in the number of processors) [3]. Not only is communication reduced in theory, but the algorithm has been demonstrated to perform better on a variety of architectures, including multicore processors [4], graphics processing units [5], and distributed-memory systems [6].

The standard algorithm for QR decomposition, which is implemented in LAPACK [7], ScaLAPACK [8], and Elemental [9] is known as Householder-QR (given below as Algorithm 1). For tall and skinny matrices, the algorithm works column-by-column, computing a Householder vector and applying the corresponding transformation for each column in the matrix. When the matrix is distributed across a parallel machine,

this requires one parallel reduction per column. The TSQR algorithm (given below as Algorithm 2), on the other hand, performs only one reduction during the entire computation. Therefore, TSQR requires asymptotically less inter-processor synchronization than Householder-QR on parallel machines (TSQR also achieves asymptotically higher cache reuse on sequential machines).

Computing the QR decomposition of a tall and skinny matrix is an important kernel in many contexts, including standalone least squares problems, eigenvalue and singular value computations, and Krylov subspace and other iterative methods. In addition, the tall and skinny factorization is a standard building block in the computation of the QR decomposition of general (not necessarily tall and skinny) matrices. In particular, most algorithms work by factoring a tall and skinny panel of the matrix, applying the orthogonal factor to the trailing matrix, and then continuing on to the next panel. Although Householder-QR is bottlenecked by communication in the panel factorization, it can apply the orthogonal factor as an aggregated Householder transformation efficiently, using matrix multiplication [10].

The Communication-Avoiding QR (CAQR) [1] algorithm uses TSQR to factor each panel of a general matrix. One difficulty faced by CAQR is that TSQR computes an orthogonal factor that is implicitly represented in a different format than that of Householder-QR. While Householder-QR represents the orthogonal factor as a set of Householder vectors (one per column), TSQR computes a tree of smaller sets of Householder vectors (though with the same total number of nonzero parameters). In CAQR, this difference in representation implies that the trailing matrix update is done using the implicit tree representation rather than matrix multiplication as possible with Householder-QR. From a software engineering perspective, this means writing and tuning more complicated code. Furthermore, from a performance perspective, the update of the trailing matrix within CAQR is less communication efficient than the update within Householder-QR by a logarithmic factor in the number of processors.

Building on a method introduced by Yamamoto [11], we show that the standard Householder vector representation may be recovered from the implicit TSQR representation for roughly the same cost as the TSQR itself. The key idea is that the Householder vectors that represent an orthonormal matrix can be computed via LU decomposition (without pivoting) of the orthonormal matrix subtracted from a diagonal sign matrix. We prove that this reconstruction is as numerically stable as Householder-QR (independent of the matrix condition number), and validate this proof with experimental results.

This reconstruction method allows us to get the best of TSQR algorithm (avoiding synchronization) as well as the best of the Householder-QR algorithm (efficient trailing matrix updates via matrix multiplication). By obtaining Householder vectors from the TSQR representation, we can logically decouple the block size of the trailing matrix updates from the number of columns in each TSQR. This abstraction makes it possible to optimize panel factorization and the trailing matrix updates independently. Our resulting parallel implementation outperforms ScaLAPACK, Elemental, and our own lightly-optimized CAQR implementation on the Hopper Cray XE6 platform at NERSC. While we do not experimentally study sequential performance, we expect our algorithm will also be beneficial in this setting, due to the cache efficiency gained by using TSQR.

2 Performance cost model

In this section, we detail our algorithmic performance cost model for parallel execution on p processors. We will use the α-β-γ model which expresses algorithmic costs in terms of computation and communication costs, the latter being composed of a bandwidth cost as well as a latency cost. We assume the cost of a message of size w words is $\alpha+\beta w$, where α is the per-message latency cost and β is the per-word bandwidth cost. We let γ be the cost per floating point operation. We ignore the network topology and measure the costs in parallel, so that the cost of two disjoint pairs of processors communicating the same-sized message simultaneously is the same as that of one message. We also assume that two processors can exchange equal-sized messages simultaneously, but a processor can communicate with only one other processor at a time.

Our algorithmic analysis will depend on the costs of collective communication, particularly broadcasts,

reductions, and all-reductions, and we consider both tree-based and bidirectional-exchange (or recursive doubling/halving) algorithms [12, 13, 14]. For arrays of size $w \geq p$, the collectives scatter, gather, all-gather, and reduce-scatter can all be performed with communication cost

$$\alpha \cdot \log p + \beta \cdot \frac{p-1}{p} w \tag{1}$$

(reduce-scatter also incurs a computational cost of $\gamma \cdot ((p-1)/p)w$). Since a broadcast can be performed with scatter and all-gather, a reduction can be performed with reduce-scatter and gather, and an all-reduction can be performed with reduce-scatter and all-gather, the communication costs of those collectives for large arrays are twice that of Equation (1). For small arrays ($w < p$), it is not possible to use pipelined or recursive-doubling algorithms, which require the subdivision of the message into many pieces, therefore collectives such as a binomial tree broadcast must be used. In this case, the communication cost of broadcast, reduction, and all-reduction is $\alpha \cdot \log p + \beta \cdot w \log p$.

3 Previous Work

We distinguish between two types of QR factorization algorithms. We call an algorithm that distributes entire rows of the matrix to processors a 1D algorithm. Such algorithms are often used for tall and skinny matrices. Algorithms which distribute the matrix across a 2D grid of $p_r \times p_c$ processors are known as 2D algorithms. Many right-looking 2D algorithms for QR decomposition of nearly square matrices divide the matrix into column panels and work panel-by-panel, factoring the panel with a 1D algorithm and then updating the trailing matrix. We consider two such existing algorithms in this section: 2D-Householder-QR (using Householder-QR) and CAQR (using TSQR).

3.1 Householder-QR

We first present Householder-QR in Algorithm 1, following [15] so that each Householder vector has a unit diagonal entry. We use LAPACK [7] notation for the scalar quantities.[1] We present Algorithm 1 in Matlab-style notation as a sequential algorithm. The algorithm works column-by-column, computing a Householder vector and then updating the trailing matrix to the right. The Householder vectors are stored in an $m \times b$ lower triangular matrix Y. Note that we do not include τ as part of the output because it can be recomputed from Y.

While the algorithm works for general m and n, it is most commonly used when $m \gg n$, such as a panel factorization within a square QR decomposition. In LAPACK terms, this algorithm corresponds to `geqr2` and is used as a subroutine in `geqrf`. In this case, we also compute an upper triangular matrix T so that

$$Q = \prod_{i=1}^{n} (I - \tau_i y_i y_i^T) = I - YTY^T,$$

which allows the application of Q^T to the trailing matrix to be done efficiently using matrix multiplication. Computing T is done in LAPACK with `larft` but can also be computed from Y^TY by solving the equation $Y^TY = T^{-1} + T^{-T}$ for T^{-1} (since Y^TY is symmetric and T^{-1} is triangular, the off-diagonal entries are equivalent and the diagonal entries differ by a factor of 2) [16].

3.1.1 1D Algorithm

We will make use of Householder-QR as a sequential algorithm, but there are parallelizations of the algorithm in libraries such as ScaLAPACK [8] and Elemental [9] against which we will compare our new approach. Assuming a 1D distribution across p processors, the parallelization of Householder-QR (Algorithm 1) requires communication at lines 2 and 8, both of which can be performed using an all-reduction.

[1] However, we depart from the LAPACK code in that there is no check for a zero norm of a subcolumn.

Algorithm 1 $[Y, R] = \text{Householder-QR}(A)$

Require: A is $m \times b$

1: **for** $i = 1$ to b **do**

% *Compute the Householder vector*

2: $\alpha = A(i,i)$, $\beta = \|A(i{:}m, i)\|_2$

3: **if** $\alpha > 0$ **then**

4: $\beta = -\beta$

5: **end if**

6: $A(i,i) = \beta$, $\tau(i) = \frac{\beta - \alpha}{\beta}$

7: $A(i{+}1{:}m, i) = \frac{1}{\alpha - \beta} \cdot A(i{+}1{:}m, i)$

% *Apply the Householder transformation to the trailing matrix*

8: $z = \tau(i) \cdot [A(i, i{+}1{:}b) + A(i{+}1{:}m, i)^T \cdot A(i{+}1{:}m, i{+}1{:}b)]$

9: $A(i{+}1{:}m, i{+}1{:}b) = A(i{+}1{:}m, i{+}1{:}b) - A(i{+}1{:}m, i) \cdot z$

10: **end for**

Ensure: $A = \left(\prod_{i=1}^{n} (I - \tau_i y_i y_i^T)\right) R$

Ensure: R overwrites the upper triangle and Y (the Householder vectors) has implicit unit diagonal and overwrites the strict lower triangle of A; τ is an array of length b with $\tau_i = 2/(y_i^T y_i)$

Because these steps occur for each column in the matrix, the total latency cost of the algorithm is $2b \log p$. This synchronization cost is a potential parallel scaling bottleneck, since it grows with the number of columns of the matrix and does not decrease with the number of processors. The bandwidth cost is dominated by the all-reduction of the z vector, which has length $b - i$ at the ith step. Thus, the bandwidth cost of Householder-QR is $(b^2/2) \log p$. The computation within the algorithm is load balanced for a parallel cost of $(2mb^2 - 2b^3/3)/p$ flops.

3.1.2 2D Algorithm

In the context of a 2D algorithm, in order to perform an update with the computed Householder vectors, we must also compute the T matrix from Y in parallel. The leading order cost of computing T^{-1} from $Y^T Y$ is mb^2/p flops plus the cost of reducing a symmetric $b \times b$ matrix, $\alpha \cdot \log p + \beta \cdot b^2/2$; note that the communication costs are lower order terms compared to computing Y. We present the costs of parallel Householder-QR in the first row of Table 1, combining the costs of Algorithm 1 with those of computing T.

We refer to the 2D algorithm that uses Householder-QR as the panel factorization as 2D-Householder-QR. In ScaLAPACK terms, this algorithm corresponds to `pxgeqrf`. The overall cost of 2D-Householder-QR, which includes panel factorizations and trailing matrix updates, is given to leading order by

$$\gamma \cdot \left(\frac{6mnb - 3n^2 b}{2p_r} + \frac{n^2 b}{2p_c} + \frac{2mn^2 - 2n^3/3}{p} \right) + \beta \cdot \left(nb \log p_r + \frac{2mn - n^2}{p_r} + \frac{n^2}{p_c} \right) + \alpha \cdot \left(2n \log p_r + \frac{2n}{b} \log p_c \right).$$

The derivation of these costs is very similar to that of CAQR-HR (see Section 4.3). If we pick $p_r = p_c = \sqrt{p}$ (assuming $m \approx n$) and $b = n/(\sqrt{p} \log p)$ then we obtain the leading order costs

$$\gamma \cdot (2mn^2 - 2n^3/3)/p + \beta \cdot (mn + n^2)/\sqrt{p} + \alpha \cdot n \log p.$$

Note that these costs match those of [1, 8], with exceptions coming from the use of more efficient collectives. The choice of b is made to preserve the leading constants of the parallel computational cost. We present the costs of 2D-Householder-QR in the first row of Table 2.

3.2 Communication-Avoiding QR

In this section we present parallel Tall-Skinny QR (TSQR) [1, Algorithm 1] and Communication-Avoiding QR (CAQR) [1, Algorithm 2], which are algorithms for computing a QR decomposition that are more

communication efficient than Householder-QR, particularly for tall and skinny matrices.

3.2.1 1D Algorithm (TSQR)

We present a simplified version of TSQR in Algorithm 2: we assume the number of processors is a power of two and use a binary reduction tree (TSQR can be performed with any tree). Note also that we present a reduction algorithm rather than an all-reduction (*i.e.*, the final R resides on only one processor at the end of the algorithm). TSQR assumes the tall-skinny matrix A is distributed in block row layout so that each processor owns a $(m/p) \times n$ submatrix. After each processor computes a local QR factorization of its submatrix (line 1), the algorithm works by reducing the p remaining $n \times n$ triangles to one final upper triangular $R = Q^T A$ (lines 2–10). The Q that triangularizes A is stored implicitly as a tree of sets of Householder vectors, given by $\{Y_{i,k}\}$. In particular, $\{Y_{i,k}\}$ is the set of Householder vectors computed by processor i at the kth level of the tree. The ith leaf of tree, $Y_{i,0}$ is the set of Householder vectors which processor i computes by doing a local QR on its part of the initial matrix A.

In the case of a binary tree, every internal node of the tree consists of a QR factorization of two stacked $b \times b$ triangles (line 6). This sparsity structure can be exploited, saving a constant factor of computation compared to a QR factorization of a dense $2b \times b$ matrix. In fact, as of version 3.4, LAPACK has subroutines for exploiting this and similar sparsity structures (`tpqrt`). Furthermore, the Householder vectors generated during the QR factorization of stacked triangles have similar sparsity; the structure of the $Y_{i,k}$ for $k > 0$ is an identity matrix stacked on top of a triangle.

Because the set $\{Y_{i,k}\}$ constitutes the implicit tree representation of the orthogonal factor, it is important to note how the tree is distributed across processors since the Q factor will be either applied to a matrix (see [1, Algorithm 2]) or constructed explicitly (see Section 3.2.3). For a given $Y_{i,k}$, i indicates the processor number (both where it is computed and where it is stored), and k indicates the level in the tree. Although $0 \leq i \leq p-1$ and $0 \leq k \leq \log p$, many of the $Y_{i,k}$ are null; for example, $Y_{i,\log p} = \emptyset$ for $i > 0$ and $Y_{p-1,k} = \emptyset$ for $k > 0$. In the case of the binary tree in Algorithm 2, processor 0 computes and stores $Y_{0,k}$ for $0 \leq k \leq \log p$ and thus requires $O(b^2 \log p)$ extra memory.

The costs and analysis of TSQR are given in [1, 17]:

$$\gamma \cdot \left(\frac{2mb^2}{p} + \frac{2b^3}{3} \log p \right) + \beta \cdot \left(\frac{b^2}{2} \log p \right) + \alpha \cdot \log p.$$

We tabulate these costs in the second row of Table 1. We note that the TSQR inner tree factorizations require an extra computational cost $O(b^3 \log p)$ and a bandwidth cost of $O(b^2 \log p)$. Also note that in the context of a 2D algorithm, using TSQR as the panel factorization implies that there is no $b \times b$ T matrix to compute; the update of the trailing matrix is performed differently.

3.2.2 2D Algorithm (CAQR)

The 2D algorithm that uses TSQR for panel factorizations is known as CAQR. In order to update the trailing matrix within CAQR, the implicit orthogonal factor computed from TSQR needs to be applied as a tree in the same order as it was computed. See [1, Algorithm 2] for a description of this process, or see [18, Algorithm 4] for pseudocode that matches the binary tree in Algorithm 2. We refer to this application of implicit Q^T as Apply-TSQR-Q^T. The algorithm has the same tree dependency flow structure as TSQR but requires a bidirectional exchange between paired nodes in the tree. We note that in internal nodes of the tree it is possible to exploit the additional sparsity structure of $Y_{i,k}$ (an identity matrix stacked on top of a triangular matrix), which our implementation does via the use of the LAPACK v3.4+ routine `tpmqrt`.

Further, since A is $m \times n$ and intermediate values of rows of A are communicated, the trailing matrix update costs more than TSQR when $n > b$. In the context of CAQR on a square matrix, Apply-TSQR-Q^T is performed on a trailing matrix with $n \approx m$ columns. The extra work in the application of the inner leaves of the tree is proportional to $O(n^2 b \log(p)/\sqrt{p})$ and bandwidth cost proportional to $O(n^2 \log(p)/\sqrt{p})$. Since the cost of Apply-TSQR-Q^{T}is almost leading order in CAQR, it is desirable in practice to optimize

Algorithm 2 $[\{Y_{i,k}\}, R] = \text{TSQR}(A)$

Require: Number of processors is p and i is the processor index
Require: A is $m \times b$ matrix distributed in block row layout; A_i is processor i's block
1: $[Y_{i,0}, \bar{R}_i] = \text{Householder-QR}(A_i)$
2: **for** $k = 1$ to $\lceil \log p \rceil$ **do**
3: **if** $i \equiv 0 \mod 2^k$ and $i + 2^{k-1} < p$ **then**
4: $j = i + 2^{k-1}$
5: Receive $\bar{R}_j$ from processor j
6: $[Y_{i,k}, \bar{R}_i] = \text{Householder-QR}\left(\begin{bmatrix} \bar{R}_i \\ \bar{R}_j \end{bmatrix}\right)$
7: **else if** $i \equiv 2^{k-1} \mod 2^k$ **then**
8: Send $\bar{R}_i$ to processor $i - 2^{k-1}$
9: **end if**
10: **end for**
11: **if** $i = 0$ **then**
12: $R = \bar{R}_0$
13: **end if**
Ensure: $A = QR$ with Q implicitly represented by $\{Y_{i,k}\}$
Ensure: R is stored by processor 0 and $Y_{i,k}$ is stored by processor i

the update routine. However, the tree dependency structure complicates this manual developer or compiler optimization task.

The overall cost of CAQR is given to leading order by

$$\gamma \cdot \left(\frac{6mnb - 3n^2 b}{2p_r} + \left(\frac{4nb^2}{3} + \frac{3n^2 b}{2p_c} \right) \log p_r + \frac{6mn^2 - 2n^3}{3p} \right) +$$
$$\beta \cdot \left(\left(\frac{nb}{2} + \frac{n^2}{p_c} \right) \log p_r + \frac{2mn - n^2}{p_r} \right) + \alpha \cdot \left(\frac{3n}{b} \log p_r + \frac{4n}{b} \log p_c \right).$$

See [1] for a discussion of these costs and [17] for detailed analysis. Note that the bandwidth cost is slightly lower here due to the use of more efficient broadcasts. If we pick $p_r = p_c = \sqrt{p}$ (assuming $m \approx n$) and $b = \frac{n}{\sqrt{p} \log^2 p}$ then we obtain the leading order costs

$$\gamma \cdot \left(\frac{2mn^2 - 2n^3/3}{p} \right) + \beta \cdot \left(\frac{2mn + n^2 \log p}{\sqrt{p}} \right) + \alpha \cdot \left(\frac{7}{2} \sqrt{p} \log^3 p \right).$$

Again, we choose b to preserve the leading constants of the computational cost. Note that b needs to be chosen smaller here than in Section 3.1.2 due to the costs associated with Apply-TSQR-Q^T.

It is possible to reduce the costs of Apply-TSQR-Q^T further using ideas from efficient recursive doubling/halving collectives. See Appendix A for details.

3.2.3 Constructing Explicit Q from TSQR

In many use cases of QR decomposition, an explicit orthogonal factor is not necessary; rather, we need only the ability to apply the matrix (or its transpose) to another matrix, as done in the previous section. For our purposes (see Section 4), we will need to form the explicit $m \times b$ orthonormal matrix from the implicit tree representation.[2] Though it is not necessary within CAQR, we describe it here because it is a known algorithm (see [19, Figure 4]) and the structure of the algorithm is very similar to TSQR.

[2] In LAPACK terms, constructing (*i.e.*, generating) the orthogonal factor when it is stored as a set of Householder vectors is done with `orgqr`.

Algorithm 3 $[B] = \text{Apply-TSQR-Q}^T(\{Y_{i,k}\}, A)$

Require: Number of processors is p and i is the processor index
Require: A is $m \times n$ matrix distributed in block row layout; A_i is processor i's block
Require: $\{Y_{i,k}\}$ is the implicit tree TSQR representation of b Householder vectors of length m.

1: $B_i = \text{Apply-Householder-}Q^T(Y_{i,0}, A_i)$
2: Let $\bar{B}_i$ be the first b rows of B_i
3: **for** $k = 1$ to $\lceil \log p \rceil$ **do**
4: **if** $i \equiv 0 \mod 2^k$ and $i + 2^{k-1} < p$ **then**
5: $j = i + 2^{k-1}$
6: Receive $\bar{B}_j$ from processor j
7: $\begin{bmatrix} \bar{B}_i \\ \bar{B}_j \end{bmatrix} = \text{Apply-Householder-}Q^T \left(Y_{i,k}, \begin{bmatrix} \bar{B}_i \\ \bar{B}_j \end{bmatrix} \right)$
8: Send $\bar{B}_j$ back to processor j
9: **else if** $i \equiv 2^{k-1} \mod 2^k$ **then**
10: Send $\bar{B}_i$ to processor $i - 2^{k-1}$
11: Receive updated rows $\bar{B}_i$ from processor $i - 2^{k-1}$
12: Set the first b rows of B_i to $\bar{B}_i$
13: **end if**
14: **end for**

Ensure: $B = Q^T A$ with processor i owning block B_i, where Q is the orthogonal matrix implicitly represented by $\{Y_{i,k}\}$

Algorithm 4 presents the method for constructing the $m \times b$ matrix Q by applying the (implicit) square orthogonal factor to the first b columns of the $m \times m$ identity matrix. Note that while we present Algorithm 4 assuming a binary tree, any tree shape is possible, as long as the implicit Q is computed using the same tree shape as TSQR. While the nodes of the tree are computed from leaves to root, they will be applied in reverse order from root to leaves. Note that in order to minimize the computational cost, the sparsity of the identity matrix at the root node and the sparsity structure of $\{Y_{i,k}\}$ at the inner tree nodes is exploited. In particular, since I_b is upper triangular and $Y_{0,\lceil \log p \rceil}$ has the structure of identity stacked on top of an upper triangle, the output of the root node application in line 7 is a stack of two upper triangles. By induction, since every $Y_{i,k}$ for $k > 0$ has the same structure, all output $Q_{i,k}$ are triangular matrices. At the leaf nodes, when $Y_{i,0}$ is applied in line 13, the output is a dense $(m/p) \times b$ block.

Since the communicated matrices $\bar{Q}_j$ are triangular just as $\bar{R}_j$ was triangular in the TSQR algorithm, Construct-TSQR-Q incurs the exact same computational and communication costs as TSQR. So, we can reconstruct the unique part of the Q matrix from the implicit form given by TSQR for the same cost as the TSQR itself.

3.3 Yamamoto's Basis-Kernel Representation

The main goal of this work is to combine Householder-QR with CAQR; Yamamoto [11] proposes a scheme to achieve this. As described in Section 3.1, 2D-Householder-QR suffers from a communication bottleneck in the panel factorization. TSQR alleviates that bottleneck but requires a more complicated (and slightly less efficient) trailing matrix update. Motivated in part to improve the performance and programmability of a hybrid CPU/GPU implementation, Yamamoto suggests computing a representation of the orthogonal factor that triangularizes the panel that mimics the representation in Householder-QR.

As described by Sun and Bischof [20], there are many so-called "basis-kernel" representations of an orthogonal matrix. See also [21] for general discussion of block reflectors. For example, the Householder-

Algorithm 4 $Q = \text{Construct-TSQR-}Q(\{Y_{i,k}\})$

Require: Number of processors is p and i is the processor index
Require: $\{Y_{i,k}\}$ is computed by Algorithm 2 so that $Y_{i,k}$ is stored by processor i

```
if i = 0 then
    Q̄_0 = I_b
end if
for k = ⌈log p⌉ down to 1 do
    if i ≡ 0 mod 2^k and i + 2^{k-1} < p then
        j = i + 2^{k-1}
        [Q̄_i; Q̄_j] = Apply-Householder-Q(Y_{i,k}, [Q̄_i; 0])
        Send Q̄_j to processor j
    else if i ≡ 2^{k-1} mod 2^k then
        Receive Q̄_i from processor i − 2^{k-1}
    end if
end for
Q_i = Apply-Q-to-Triangle(Y_{i,0}, [Q̄_i; 0])
```

Ensure: Q is orthonormal $m \times b$ matrix distributed in block row layout; Q_i is processor i's block

QR algorithm computes a lower triangular matrix Y such that $A = (I - YTY_1^T)R$, so that

$$Q = I - YTY^T = I - \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} T \begin{bmatrix} Y_1^T & Y_2^T \end{bmatrix}. \tag{2}$$

Here, Y is called the "basis" and T is called the "kernel" in this representation of the square orthogonal factor Q. However, there are many such basis-kernel representations if we do not restrict Y and T to be lower and upper triangular matrices, respectively.

Yamamoto [11] chooses a basis-kernel representation that is easy to compute. For an $m \times b$ matrix A, let $A = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R$ where Q_1 and R are $b \times b$. Then define the basis-kernel representation

$$Q = I - \tilde{Y}\tilde{T}\tilde{Y}^T = I - \begin{bmatrix} Q_1 - I \\ Q_2 \end{bmatrix} \left[I - Q_1\right]^{-T} \begin{bmatrix} (Q_1 - I)^T & Q_2^T \end{bmatrix}, \tag{3}$$

where $I - Q_1$ is assumed to be nonsingular. It can be easily verified that $Q^TQ = I$ and $Q^TA = \begin{bmatrix} R \\ 0 \end{bmatrix}$; in fact, this is the representation suggested and validated by [22, Theorem 3]. Note that both the basis and kernel matrices $\tilde{Y}$ and $\tilde{T}$ are dense.

The main advantage of basis-kernel representations is that they can be used to apply the orthogonal factor (or its transpose) very efficiently using matrix multiplication. In particular, the computational complexity of applying Q^T using any basis-kernel is the same to leading order, assuming Y has the same dimensions as A and $m \gg b$. Thus, it is not necessary to reconstruct the Householder vectors; from a computational perspective, finding any basis-kernel representation of the orthogonal factor computed by TSQR will do. Note also that in order to apply Q^T with the representation in Equation (3), we need to apply the inverse of $I - Q_1$, so we need to perform an LU decomposition of the $b \times b$ matrix and then apply the inverses of the triangular factors using triangular solves.

The assumption that $I - Q_1$ is nonsingular can be dropped by replacing I with a diagonal sign matrix S chosen so that $S - Q_1$ is nonsingular [23]; in this case the representation becomes

$$QS = I - \tilde{Y}\tilde{T}\tilde{Y}^T = I - \begin{bmatrix} Q_1 - S \\ Q_2 \end{bmatrix} S \left[S - Q_1\right]^{-T} \begin{bmatrix} (Q_1 - S)^T & Q_2^T \end{bmatrix}. \tag{4}$$

Yamamoto's approach is very closely related to TSQR-HR (Algorithm 6), presented in Section 4. We compare the methods in Section 4.1.

4 New Algorithms

We first present our main contribution, a parallel algorithm that performs TSQR and then reconstructs the Householder vector representation from the TSQR representation of the orthogonal factor. We then show that this reconstruction algorithm may be used as a building block for more efficient 2D QR algorithms. In particular, the algorithm is able to combine two existing approaches for 2D QR factorizations, leveraging the efficiency of TSQR in panel factorizations and the efficiency of Householder-QR in trailing matrix updates. While Householder reconstruction adds some extra cost to the panel factorization, we show that its use in the 2D algorithm reduces overall communication compared to both 2D-Householder-QR and CAQR.

4.1 TSQR with Householder Reconstruction

The basic steps of our 1D algorithm include performing TSQR, constructing the explicit tall-skinny Q factor, and then computing the Householder vectors corresponding to Q. The key idea of our reconstruction algorithm is that performing Householder-QR on an orthonormal matrix Q is the same as performing an LU decomposition on $Q - S$, where S is a diagonal sign matrix corresponding to the sign choices made inside the Householder-QR algorithm. This claim is proved explicitly in Lemma 5.2. Informally, ignoring signs, if $Q = I - YTY_1^T$ with Y a matrix of Householder vectors, then $Y \cdot (-TY_1^T)$ is an LU decomposition of $Q - I$ since Y is unit lower triangular and TY_1^T is upper triangular.

In this section we present Modified-LU as Algorithm 5, which can be applied to any orthonormal matrix (not necessarily one obtained from TSQR). Ignoring lines 1, 3, and 4, it is exactly LU decomposition without pivoting. Note that with the choice of S, no pivoting is required since the effective diagonal entry will be at least 1 in absolute value and all other entries in the column are bounded by 1 (the matrix is orthonormal).[3] This holds true throughout the entire factorization because the trailing matrix remains orthonormal, which we prove within Lemma 5.2.

Algorithm 5 $[L, U, S] = \text{Modified-LU}(Q)$

Require: Q is $m \times b$ orthonormal matrix

1: $S = 0$
2: **for** $i = 1$ to b **do**
3: $\quad S(i,i) = -\operatorname{sgn}(Q(i,i))$
4: $\quad Q(i,i) = Q(i,i) - S(i,i)$
$\quad$ % *Scale ith column of L by diagonal element*
5: $\quad Q(i{+}1{:}m, i) = \frac{1}{Q(i,i)} \cdot Q(i{+}1{:}m, i)$
$\quad$ % *Perform Schur complement update*
6: $\quad Q(i{+}1{:}m, i{+}1{:}b) = Q(i{+}1{:}m, i{+}1{:}b) - Q(i{+}1{:}m, i) \cdot Q(i, i{+}1{:}b)$
7: **end for**

Ensure: U overwrites the upper triangle and L has implicit unit diagonal and overwrites the strict lower triangle of Q; S is diagonal so that $Q - S = LU$

Given the algorithms of the previous sections, we now present the full approach for computing the QR decomposition of a tall-skinny matrix using TSQR and Householder reconstruction. That is, in this section we present an algorithm such that the format of the output of the algorithm is identical to that of Householder-QR. However, we argue that the communication costs of this approach are much less than those of performing Householder-QR.

[3] We use the convention $\operatorname{sgn}(0) = 1$.

	Flops	Words	Messages
Householder-QR	$\frac{3mb^2}{p} - \frac{2b^3}{3p}$	$\frac{b^2}{2}\log p$	$2b\log p$
TSQR	$\frac{2mb^2}{p} + \frac{2b^3}{3}\log p$	$\frac{b^2}{2}\log p$	$\log p$
TSQR-HR	$\frac{5mb^2}{p} + \frac{4b^3}{3}\log p$	$b^2\log p$	$4\log p$

Table 1: Costs of QR factorization of tall-skinny $m \times b$ matrix distributed over p processors in 1D fashion. We assume these algorithms are used as panel factorizations in the context of a 2D algorithm applied to an $m \times n$ matrix. Thus, costs of Householder-QR and TSQR-HR include the costs of computing T.

The method, given as Algorithm 6, is to perform TSQR (line 1), construct the tall-skinny Q factor explicitly (line 2), and then compute the Householder vectors that represent that orthogonal factor using Modified-LU (line 3). The R factor is computed in line 1 and the Householder vectors (the columns of Y) are computed in line 3. An added benefit of the approach is that the triangular T matrix, which allows for block application of the Householder vectors, can be computed very cheaply. That is, a triangular solve involving the upper triangular factor from Modified-LU computes the T so that $A = (I - YTY_1^T)R$. To compute T directly from Y (as is necessary if Householder-QR is used) requires $O(nb^2)$ flops; here the triangular solve involves $O(b^3)$ flops. Our approach for computing T is given in line 4, and line 5 ensures sign agreement between the columns of the (implicitly stored) orthogonal factor and rows of R.

Algorithm 6 $[Y, T, R] = \text{TSQR-HR}(A)$

Require: A is $m \times b$ matrix distributed in block row layout

1: $[\{Y_{i,k}\}, \tilde{R}] = \text{TSQR}(A)$
2: $Q = \text{Construct-TSQR-}Q(\{Y_{i,k}\})$
3: $[Y, U, S] = \text{Modified-LU}(Q)$
4: $T = -USY_1^{-T}$
5: $R = S\tilde{R}$

Ensure: $A = (I - YTY_1^T)R$, where Y is $m \times b$ and unit lower triangular, Y_1 is top $b \times b$ block of Y, and T and R are $b \times b$ and upper triangular

On p processors, Algorithm 6 incurs the following costs (ignoring lower order terms):

1. Compute $[\{Y_{i,k}\}, R'] = \text{TSQR}(A)$

   The computational costs of this step come from lines 1 and 6 in Algorithm 2. Line 1 corresponds to a QR factorization of a $(m/p) \times b$ matrix, with a flop count of $2(m/p)b^2 - 2b^3/3$ (each processor performs this step simultaneously). Line 6 corresponds to a QR factorization of a $b \times b$ triangle stacked on top of a $b \times b$ triangle. Exploiting the sparsity structure, the flop count is $2b^3/3$; this occurs at every internal node of the binary tree, so the total cost in parallel is $(2b^3/3)\log p$.

   The communication costs of Algorithm 2 occur in lines 5 and 8. Since every $R_{i,k}$ is a $b \times b$ upper triangular matrix, the cost of a single message is $\alpha + \beta \cdot (b^2/2)$. This occurs at every internal node in the tree, so the total communication cost in parallel is a factor $\log p$ larger.

   Thus, the cost of this step is

$$\gamma \cdot \left(\frac{2mb^2}{p} + \frac{2b^3}{3}\log p\right) + \beta \cdot \left(\frac{b^2}{2}\log p\right) + \alpha \cdot \log p.$$

2. $Q = \text{Construct-TSQR-}Q(\{Y_{i,k}\})$

   The computational costs of this step come from lines 7 and 13 in Algorithm 4. Note that for $k > 0$, $Y_{i,k}$ is a $2b \times b$ matrix: the identity matrix stacked on top of an upper triangular matrix. Furthermore, $Q_{i,k}$ is an upper triangular matrix. Exploiting the structure of $Y_{i,k}$ and $Q_{i,k}$, the cost of line 7 is $2b^3/3$, which occurs at every internal node of the tree. Each $Y_{i,0}$ is a $(m/p) \times b$ lower triangular

matrix of Householder vectors, so the cost of applying them to an upper triangular matrix in line 13 is $2(m/p)b^2 - 2b^3/3$. Note that the computational cost of these two lines is the same as those of the previous step in Algorithm 2.

The communication pattern of Algorithm 4 is identical to Algorithm 2, so the communication cost is also the same as that of the previous step.

Thus, the cost of constructing Q is

$$\gamma \cdot \left(\frac{2mb^2}{p} + \frac{2b^3}{3} \log p \right) + \beta \cdot \left(\frac{b^2}{2} \log p \right) + \alpha \cdot \log p.$$

3. $[Y, U, S] = \text{Modified-LU}(Q)$

   Ignoring lines 3–4 in Algorithm 5, Modified-LU is the same as LU without pivoting. In parallel, the algorithm consists of a $b \times b$ (modified) LU factorization of the top block followed by parallel triangular solves to compute the rest of the lower triangular factor. The flop count of the $b \times b$ LU factorization is $2b^3/3$, and the cost of each processor's triangular solve is $(m/p)b^2$. The communication cost of parallel Modified-LU is only that of a broadcast of the upper triangular $b \times b$ U factor (for which we use a bidirectional-exchange algorithm): $\beta \cdot (b^2) + \alpha \cdot (2 \log p)$.

   Thus, the cost of this step is

$$\gamma \cdot \left(\frac{mb^2}{p} + \frac{2b^3}{3} \right) + \beta \cdot b^2 + \alpha \cdot 2 \log p$$

4. $T = -USY_1^{-T}$ and $R = SR'$

   The last two steps consist of computing T and scaling the final R appropriately. Since S is a sign matrix, computing US and SR' requires no floating point operations and can be done locally on one processor. Thus, the only computational cost is performing a $b \times b$ triangular solve involving Y_1^T. If we ignore the triangular structure of the output, the flop count of this operation is b^3. However, since this operation occurs on the same processor that computes the top $m/p \times b$ block of Y and U, it can be overlapped with the previous step (Modified-LU). After the top processor performs the $b \times b$ LU factorization and broadcasts the U factor, it computes only $m/p - b$ rows of Y (all other processors update m/p rows). Thus, performing an extra $b \times b$ triangular solve on the top processor adds no computational cost to the critical path of the algorithm.

Thus, TSQR-HR(A) where A is m-by-b incurs the following costs (ignoring lower order terms):

$$\gamma \cdot \left(\frac{5mb^2}{p} + \frac{4b^3}{3} \log p \right) + \beta \cdot \left(b^2 \log p \right) + \alpha \cdot (4 \log p) \,. \tag{5}$$

Note that the LU factorization required in Yamamoto's approach (see Section 3.3) is equivalent to performing Modified-LU($-Q_1$). In Algorithm 6, the Modified-LU algorithm is applied to an $m \times b$ matrix rather than to only the top $b \times b$ block; since no pivoting is required, the only difference is the update of the bottom $m - b$ rows with a triangular solve. Thus it is not hard to see that, ignoring signs, the Householder basis-kernel representation in Equation (2) can be obtained from the representation given in Equation (3) with two triangular solves: if the LU factorization gives $I - Q_1 = LU$, then $Y = \tilde{Y}U^{-1}$ and $T = UL^{-T}$. Indeed, performing these two operations and handling the signs correctly gives Algorithm 6. While Yamamoto's approach avoids performing the triangular solve on Q_2, it still involves performing both TSQR and Construct-TSQR-Q.

	Flops	Words	Messages
2D-Householder-QR	$\frac{2mn^2-2n^3/3}{p}$	$\frac{2mn+n^2/2}{\sqrt{p}}$	$n \log p$
CAQR	$\frac{2mn^2-2n^3/3}{p}$	$\frac{2mn+n^2 \log p}{\sqrt{p}}$	$\frac{7}{2}\sqrt{p}\log^3 p$
CAQR-HR	$\frac{2mn^2-2n^3/3}{p}$	$\frac{2mn+n^2/2}{\sqrt{p}}$	$6\sqrt{p}\log^2 p$

Table 2: Costs of QR factorization of $m \times n$ matrix distributed over p processors in 2D fashion. Here we assume a square processor grid ($p_r = p_c$). We also choose block sizes for each algorithm independently to ensure the leading order terms for flops are identical.

4.2 LU Decomposition of $A - R$

Computing the Householder vectors basis-kernel representation via Algorithm 6 requires forming the explicit $m \times b$ orthonormal matrix. It is possible to compute this representation more cheaply, though at the expense of losing numerical stability. Suppose A is full rank, $R = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$ is the upper triangular factor computed by TSQR, and $A - R$ is also full rank. Then $A = (I - YTY_1^T)R_1$ implies that the unique LU decomposition of $A - R$ is given by

$$A - R = Y(-TY_1^T R_1), \tag{6}$$

since Y is unit lower triangular and T, Y_1^T, and R_1 are all upper triangular. The assumption that $A - R$ is full rank can be dropped by choosing an appropriate diagonal sign matrix S and computing the LU decomposition of $A - \begin{bmatrix} SR_1 \\ 0 \end{bmatrix}$.

Not surprisingly, choosing the signs to match the choices of the Householder-QR algorithm applied to A guarantees that the matrix is nonsingular. In fact, the signs can be computed during the course of a modified LU decomposition algorithm very similar to Algorithm 5 (designed for general matrices rather than only orthonormal). This more general algorithm can also be interpreted as performing Householder-QR on A but using R to provide "hints" to the algorithm to avoid certain expensive computations; see Appendix B for more details.

The advantage of this approach is that Y can be computed without constructing Q explicitly, which is as expensive as TSQR itself. Unfortunately, this method is not numerically stable. An intuition for the instability comes from considering a low-rank matrix A. Because the QR decomposition of A is not unique, the TSQR algorithm and the Householder-QR algorithm may compute different factor pairs $(Q_{\text{TSQR}}, R_{\text{TSQR}})$ and $(Q_{\text{Hh}}, R_{\text{Hh}})$. Performing an LU decomposition using A and R_{TSQR} to recover the Householder vectors that represent Q_{Hh} is hopeless if $R_{\text{TSQR}} \neq R_{\text{Hh}}$, even in exact arithmetic. In floating point arithmetic, an ill-conditioned A corresponds to a factor R which is sensitive to roundoff error, so reconstructing the Householder vectors that would produce the same R as computed by TSQR is unstable. However, if A is well-conditioned, then this method is sufficient; one can view Algorithm 5 as applying the method to an orthonormal matrix (which is perfectly conditioned).

One method of decreasing the sensitivity of R is to employ column pivoting. That is, after performing TSQR apply QR with column pivoting to the $b \times b$ upper triangular factor R, yielding a factorization $A = Q_{\text{TSQR}}(\tilde{Q}\tilde{R}P)$, or $AP = (Q_{\text{TSQR}}\tilde{Q})\tilde{R}$, where the diagonal entries of R decrease monotonically in absolute value. We have observed experimentally that performing LU decomposition of $AP - \begin{bmatrix} S\tilde{R} \\ 0 \end{bmatrix}$ is a more stable operation than not using column pivoting, though it is still not as robust as Algorithm 6. See Section 5 for a discussion of experiments where the instability occurs (even with column pivoting).

4.3 CAQR-HR

We refer to the 2D algorithm that uses TSQR-HR for panel factorizations as CAQR-HR. Because Householder-QR and TSQR-HR generate the same representation as output of the panel factorization, the trailing matrix

Algorithm 7 $[Y, T, R] = \text{CAQR-HR}(A)$

Require: A is $m \times n$ and distributed block-cyclically on $p = p_r \cdot p_c$ processors with block size b, so that each $b \times b$ block A_{ij} is owned by processor $\Pi(i, j) = (i \bmod p_r) + p_r \cdot (j \bmod p_c)$

1: **for** $i = 0$ to $n/b - 1$ **do**

% *Compute TSQR and reconstruct Householder representation using column of p_r processors*

2: $\left[Y_{i:m/b-1,i}, T_i, R_{ii}\right] = \text{Hh-Recon-TSQR}(A_{i:m/b-1,i})$

% *Update trailing matrix using all p processors*

3: $\Pi(i, i)$ broadcasts T_i to all other processors

4: **for** $r \in [i, m/b - 1], c \in [i + 1, n/b - 1]$ **do in parallel**

5: $\Pi(r, i)$ broadcasts Y_{ri} to all processors in its row, $\Pi(r, :)$

6: $\Pi(r, c)$ computes $\tilde{W}_{rc} = Y_{ri}^T \cdot A_{rc}$

7: Allreduce $W_c = \sum_r \tilde{W}_{rc}$ so that processor column $\Pi(:, c)$ owns W_c

8: $\Pi(r, c)$ computes $A_{rc} = A_{rc} - Y_{ri} \cdot T_i^T \cdot W_c$

9: Set $R_{ic} = A_{ic}$

10: **end for**

11: **end for**

Ensure: $A = \left(\prod_{i=1}^{n} (I - Y_{:,i} T_i Y_{:,i}^T)\right) R$

update can be performed in exactly the same way. Thus, the only difference between 2D-Householder-QR and CAQR-HR, presented in Algorithm 7, is the subroutine call for the panel factorization (line 2).

Algorithm 7 with block size b, and matrix m-by-n matrix A, with $m \geq n$ and $m, n \equiv 0 \pmod{b}$ distributed on a 2D p_r-by-p_c processor grid incurs the following costs over all n/b iterations.,

1. Compute $\left[Y_{i:m/b-1,i}, T_i, R_{ii}\right] = \text{Hh-Recon-TSQR}\left(A_{i:m/b-1,i}\right)$

   Equation (5) gives the cost of a single panel TSQR factorization with Householder reconstruction. We can sum over all iterations to obtain the cost of this step in the 2D QR algorithm (line 2 in Algorithm 7),

$$\sum_{i=0}^{n/b-1} \left(\gamma \cdot \left(\frac{5(m - ib)b^2}{p_r} + \frac{4b^3}{3} \log p_r\right) + \beta \cdot \left(b^2 \log p_r\right) + \alpha \cdot (4 \log p_r)\right) =$$
$$\gamma \cdot \left(\frac{5mnb}{p_r} - \frac{5n^2 b}{2p_r} + \frac{4nb^2}{3} \log p_r\right) + \beta \cdot (nb \log p_r) + \alpha \cdot \left(\frac{4n \log p_r}{b}\right)$$

2. $\Pi(i, i)$ broadcasts T_i to all other processors

   The matrix T_i is b-by-b and triangular, so we use a bidirectional exchange broadcast. Since there are a total of n/b iterations, the total communication cost for this step of Algorithm 7 is

$$\beta \cdot (nb) + \alpha \cdot \left(\frac{2n}{b} \log p\right)$$

3. $\Pi(r, i)$ broadcasts Y_{ri} to all processors in its row, $\Pi(r, :)$

   At this step, each processor which took part in the TSQR and Householder reconstruction sends its local chunk of the panel of Householder vectors to the rest of the processors. At iteration i of Algorithm 7, each processor owns at most $\lceil \frac{m/b-i}{p_r} \rceil$ blocks Y_{ri}. Since all the broadcasts happen along processor rows, we assume they can be done simultaneously on the network. The communication along the critical path is then given by

$$\sum_{i=0}^{n/b-1}\left(\beta\cdot\left(\frac{2(m/b-i)\cdot b^2}{p_r}\right)+\alpha\cdot 2\log p_c\right)=\beta\cdot\left(\frac{2mn-n^2}{p_r}\right)+\alpha\cdot\left(\frac{2n}{b}\log p_c\right)$$

4. $\Pi(r,c)$ computes $\tilde{W}_{rc}=Y_{ri}^T\cdot A_{rc}$

   Each block $\tilde{W}_{rc}$ is computed on processor $\Pi(r,c)$, using A_{rc}, which it owns, and Y_{ri}, which it just received from processor $\Pi(r,i)$. At iteration i, processor j may own up to $\lceil\frac{m/b-i}{p_r}\rceil\cdot\lceil\frac{n/b-i}{p_c}\rceil$ blocks $\tilde{W}_{rc}$, $\Pi(r,c)=j$. Each, block-by-block multiply incurs $2b^3$ flops, therefore, this step incurs a total computational cost of

$$\gamma\cdot\left(\sum_{i=0}^{n/b-1}\frac{2(m/b-i)(n/b-i)b^3}{p}\right)=\gamma\cdot\left(\frac{mn^2-n^3/3}{p}\right)$$

5. Allreduce $W_c=\sum_r\tilde{W}_{rc}$ so that processor column $\Pi(:,c)$ owns W_c

   At iteration i of Algorithm 7, each processor j may own up to $\lceil\frac{m/b-i}{p_r}\rceil\cdot\lceil\frac{n/b-i}{p_c}\rceil$ blocks W_{rc}. The local part of the reduction can be done during the computation of W_{rc} on line 6 of Algorithm 7. Therefore, no process should contribute more than $\lceil\frac{n/b-i}{p_c}\rceil$ b-by-b blocks W_{rc} to the reduction. Using a bidirectional exchange all-reduction algorithm and summing over the iterations, we can obtain the cost of this step throughout the entire execution of the 2D algorithm:

$$\sum_{i=0}^{n/b-1}\left(\beta\cdot\left(\frac{2(n/b-i)\cdot b^2}{p_c}\right)+\alpha\cdot 2\log p_r\right)=\beta\cdot\left(\frac{n^2}{p_c}\right)+\alpha\cdot\left(\frac{2n}{b}\log p_r\right)$$

6. $\Pi(r,c)$ computes $A_{rc}=A_{rc}-Y_{ri}\cdot T_i\cdot W_c$

   Since in our case, $m\geq n$, it is faster to first multiply T_i by W_c rather than Y_{ri} by T_i. Any processor j may update up to $\lceil\frac{m/b-i}{p_r}\rceil\cdot\lceil\frac{n/b-i}{p_c}\rceil$ blocks of A_{rc} using $\lceil\frac{m/b-i}{p_r}\rceil$ blocks of Y_{ri} and $\lceil\frac{n/b-i}{p_c}\rceil$ blocks of W_c. When multiplying T_i by W_c, we can exploit the triangular structure of T, to lower the flop count by a factor of two. Summed over all iterations, these two multiplications incur a computational cost of

$$\gamma\cdot\left(\sum_{i=0}^{n/b-1}\frac{2(m/b-i)(n/b-i)b^3}{p}+\frac{(n/b-i)b^3}{p_c}\right)=\gamma\cdot\left(\frac{mn^2-n^3/3}{p}+\frac{n^2b}{2p_c}\right)$$

The overall costs of CAQR-HR are given to leading order by

$$\gamma\cdot\left(\frac{10mnb-5n^2b}{2p_r}+\frac{4nb^2}{3}\log p_r+\frac{n^2b}{2p_c}+\frac{2mn^2-2n^3/3}{p}\right)+$$
$$\beta\cdot\left(nb\log p_r+\frac{2mn-n^2}{p_r}+\frac{n^2}{p_c}\right)+\alpha\cdot\left(\frac{8n}{b}\log p_r+\frac{4n}{b}\log p_c\right).$$

If we pick $p_r=p_c=\sqrt{p}$ (assuming $m\approx n$) and $b=\frac{n}{\sqrt{p}\log p}$ then we obtain the leading order costs

$$\gamma\cdot\left(\frac{2mn^2-2n^3/3}{p}\right)+\beta\cdot\left(\frac{2mn+n^2/2}{\sqrt{p}}\right)+\alpha\cdot\left(6\sqrt{p}\log^2 p\right),$$

shown in the third row of Table 2.

Comparing the leading order costs of CAQR-HR with the existing approaches, we note again the $O(n \log p)$ latency cost incurred by the 2D-Householder-QR algorithm. CAQR and CAQR-HR eliminate this synchronization bottleneck and reduce the latency cost to be independent of the number of columns of the matrix. Further, both the bandwidth and latency costs of CAQR-HR are factors of $O(\log p)$ lower than CAQR (when $m \approx n$). As previously discussed, CAQR includes an extra leading order bandwidth cost term $(\beta \cdot n^2 \log p/\sqrt{p})$, as well as a computational cost term $(\gamma \cdot (n^2 b/p_c) \log p_r)$ that requires the choice of a smaller block size and leads to an increase in the latency cost.

Further, the Householder form allows us to decouple the block sizes used for the trailing matrix update from the width of each TSQR. This is a simple algorithmic optimization to add to our 2D algorithm, which has the same leading order costs. We show this nested blocked approach in Algorithm 8, which is very similar to Algorithm 7, and does not require any additional subroutines. While this algorithm does not have a theoretical benefit, it is highly beneficial in practice, as we will demonstrate in the performance section. We note that it is not possible to aggregate the implicit TSQR updates in this way.

Algorithm 8 $[Y, T, R] = \text{CAQR-HR-Agg}(A)$

Require: A is $m \times n$ and distributed block-cyclically on $p = p_r \cdot p_c$ processors with block size b_1, so that each $b_1 \times b_1$ block A_{ij} is owned by processor $\Pi(i,j) = (i \bmod p_r) + p_r \cdot (j \bmod p_c)$

1: **for** $i = 0$ to $n/b_2 - 1$ **do**
2: $\quad \left[Y_{i:m/b_2-1,i}, T_i, R_{ii}\right] = \text{CAQR-HR}(A_{i:m/b_2-1,i})$
3: $\quad$ Aggregate updates from line 5 of CAQR-HR into $(m - ib_2)$-by-b_2 matrix $\bar{Y}_{ri}$
4: $\quad$ **for** $r \in [i, m/b_2 - 1], c \in [i+1, n/b_2 - 1]$ **do in parallel**
5: $\quad\quad \Pi(r,c)$ computes $\tilde{W}_{rc} = \bar{Y}_{ri}^T \cdot A_{rc}$
6: $\quad\quad$ Allreduce $W_c = \sum_r \tilde{W}_{rc}$ so that processor column $\Pi(:,c)$ owns W_c
7: $\quad\quad \Pi(r,c)$ computes $A_{rc} = A_{rc} - \bar{Y}_{ri} \cdot T_i^T \cdot W_c$
8: $\quad\quad$ Set $R_{ic} = A_{ic}$
9: $\quad$ **end for**
10: **end for**

Ensure: $A = \left(\prod_{i=1}^{n} (I - Y_{:,i} T_i Y_{:,i}^T)\right) R$

5 Correctness and Stability

In order to reconstruct the lower trapezoidal Householder vectors that constitute an orthonormal matrix (up to column signs), we can use an algorithm for LU decomposition without pivoting on an associated matrix (Algorithm 5). Lemma 5.1 shows by uniqueness that this LU decomposition produces the Householder vectors representing the orthogonal matrix in exact arithmetic. Given the explicit orthogonal factor, the LU method is cheaper in terms of both computation and communication than constructing the vectors via Householder-QR.

Lemma 5.1. *Given an orthonormal $m \times b$ matrix Q, let the compact QR decomposition of Q given by the Householder-QR algorithm (Algorithm 1) be*

$$Q = \left(\begin{bmatrix} I_n \\ 0 \end{bmatrix} - YTY_1^T \right) S,$$

where Y is unit lower triangular, Y_1 is the top $b \times b$ block of Y, and T is the upper triangular $b \times b$ matrix satisfying $T^{-1} + T^T = Y^T Y$. Then S is a diagonal sign matrix, and $Q - \begin{bmatrix} S \\ 0 \end{bmatrix}$ has a unique LU decomposition given by

$$Q - \begin{bmatrix} S \\ 0 \end{bmatrix} = Y \cdot (-TY_1^T S). \tag{7}$$

Proof. Since Q is orthonormal, it has full rank and therefore has a unique QR decomposition with positive diagonal entries in the upper triangular matrix. This decomposition is $Q = Q \cdot I_n$, so the Householder-QR algorithm must compute a decomposition that differs only by sign. Thus, S is a diagonal sign matrix. The uniqueness of T is guaranteed by [20, Example 2.4].

We obtain Equation (7) by rearranging the Householder QR decomposition. Since Y is unit lower triangular and T, Y_1^T, and S are all upper triangular, we have an LU decomposition. Since Y is full column rank and T, Y_1^T, and S are all nonsingular, $Q - \begin{bmatrix} S \\ 0 \end{bmatrix}$ has full column rank and therefore has a unique LU decomposition with a unit lower triangular factor. □

Unfortunately, given an orthonormal matrix Q, the sign matrix S produced by Householder-QR is not known, so we cannot run a standard LU algorithm on $Q - \begin{bmatrix} S \\ 0 \end{bmatrix}$. Lemma 5.2 shows that by running the Modified-LU algorithm (Algorithm 5), we can cheaply compute the sign matrix S during the course of the algorithm.

Lemma 5.2. *In exact arithmetic, Algorithm 5 applied to an orthonormal $m \times b$ matrix Q computes the same Householder vectors Y and sign matrix S as the Householder-QR algorithm (Algorithm 1) applied to Q.*

Proof. Consider applying one step of Modified-LU (Algorithm 5) to the orthonormal matrix Q, where we first set $S_{11} = -\operatorname{sgn}(q_{11})$ and subtract it from q_{11}. Note that since all other entries of the first column are less than 1 in absolute value and the absolute value of the diagonal entry has been increased by 1, the diagonal entry is the maximum entry. Following the LU algorithm, all entries below the diagonal are scaled by the reciprocal of the diagonal element: for $2 \le i \le m$,

$$y_{i1} = \frac{q_{i1}}{q_{11} + \operatorname{sgn}(q_{11})}, \tag{8}$$

where Y is the computed lower triangular factor. The Schur complement is updated as follows: for $2 \le i \le m$ and $2 \le j \le b$,

$$\tilde{q}_{ij} = q_{ij} - \frac{q_{i1}q_{1j}}{q_{11} + \operatorname{sgn}(q_{11})}. \tag{9}$$

Now consider applying one step of the Householder-QR algorithm (Algorithm 1). To match the LAPACK notation, we let $\alpha = q_{11}$, $\beta = -\operatorname{sgn}(\alpha) \cdot \|q_1\| = -\operatorname{sgn}(\alpha)$, and $\tau = \frac{\beta - \alpha}{\beta} = 1 + \alpha \operatorname{sgn}(\alpha)$, where we use the fact that the columns of Q have unit norm. Note that $\beta = -\operatorname{sgn}(\alpha) = S_{11}$, which matches the Modified-LU algorithm above. The Householder vector y is computed by setting the diagonal entry to 1 and scaling the other entries of q_1 by $\frac{1}{\alpha - \beta} = \frac{1}{\alpha + \operatorname{sgn}(\alpha)}$. Thus, computing the Householder vector matches computing the column of the lower triangular matrix from Modified-LU in Equation (8) above.

Next, we consider the update of the trailing matrix: $\tilde{Q} = (I - \tau y y^T)Q$. Since Q has orthonormal columns, the dot product of the Householder vector with the jth column is given by

$$\begin{aligned} y^T q_j &= \sum_{i=1}^{m} y_i q_{ij} = q_{1j} + \sum_{i=2}^{m} \frac{q_{i1}}{\alpha + \operatorname{sgn}(\alpha)} q_{ij} \\ &= q_{1j} - \frac{q_{11}q_{1j}}{\alpha + \operatorname{sgn}(\alpha)} = \left(1 - \frac{\alpha}{\alpha + \operatorname{sgn}(\alpha)}\right) q_{1j}. \end{aligned}$$

The identity

$$(1 + \alpha \operatorname{sgn}(\alpha)) \left(1 - \frac{\alpha}{\alpha + \operatorname{sgn}(\alpha)}\right) = 1$$

implies $\tau y^T q_j = q_{1j}$. Then the trailing matrix update ($2 \le i \le m$ and $2 \le j \le b$) is given by

$$\tilde{q}_{ij} = q_{ij} - y_i(\tau y^T q_j) = q_{ij} - \frac{q_{i1}q_{1j}}{\alpha + \operatorname{sgn}(\alpha)},$$

which matches the Schur complement update from Modified-LU in Equation (9) above.

Finally, consider the jth element of first row of the trailing matrix ($j \geq 2$): $\tilde{q}_{ij} = q_{ij} - y_1(\tau y^T q_j) = 0$. Note that the computation of the first row is not performed in the Modified-LU algorithm. The corresponding row is not changed since the diagonal element is $Y_{11} = 1$. However, in the case of Householder-QR, since the first row of the trailing matrix is zero, and because the Householder transformation preserves column norms, the updated $(m-1) \times (b-1)$ trailing matrix is itself an orthonormal matrix. Thus, by induction, the rest of the two algorithms perform the same computations. □

In order to bound the error in the decomposition of a general matrix using Householder reconstruction, we will use Lemma 5.4 below as well as the stability bounds on the TSQR or "AllReduce" algorithm provided in [24]. We restate those results here.

Lemma 5.3 ([24])**.** *Let $\hat{R}$ be the computed upper triangular factor of $m \times b$ matrix A obtained via the AllReduce algorithm using a binary tree of height L ($m/2^L \geq b$). Then there exists an orthonormal matrix Q such that*

$$\|A - Q\hat{R}\|_F \leq f_1(m, b, L, \varepsilon)\|A\|_F$$

and

$$\|Q - \hat{Q}\|_F \leq f_2(m, b, L, \varepsilon),$$

where $f_1(m, b, L, \varepsilon) \simeq b\gamma_{c \cdot \frac{m}{2^L}} + Lb\gamma_{c \cdot 2b}$ and $f_2(m, b, L, \varepsilon) \simeq \left(b\gamma_{c \cdot \frac{m}{2^L}} + Lb\gamma_{c \cdot 2b}\right)\sqrt{b}$, for $b\gamma_{c \cdot \frac{m}{2^L}} \ll 1$ and $b\gamma_{c \cdot 2b} \ll 1$, where c is a small constant.

The following lemma shows that the computation performed in Algorithm 5 is more stable than general LU decomposition. Because it is performed on an orthonormal matrix, the growth factor in the upper triangular factor is bounded by 2, and we can bound the Frobenius norms of both computed factors by functions of the number of columns. Further, we can stably compute the $\tilde{T}$ factor (needed to apply a blocked Householder update) from the upper triangular LU factor, a computationally cheaper method than using the Householder vectors themselves.

Lemma 5.4. *In floating point arithmetic, given an orthonormal $m \times b$ matrix Q, Algorithm 5 computes factors S, $\tilde{Y}$, and $\tilde{T}$ such that*

$$\left\| QS - \left(\begin{bmatrix} I \\ 0 \end{bmatrix} - \tilde{Y}\tilde{T}\tilde{Y}_1^T \right) \right\|_F \leq f_3(b, \varepsilon).$$

where

$$f_3(b, \varepsilon) = 2\gamma_b \left(b^2 + \left(1 + \sqrt{2}\right) b \right)$$

and Y_1 is given by the first b rows of Y.

Proof. This result follows from the standard error analysis of LU decomposition and triangular solve with multiple right hand sides, and the properties of the computed lower and upper triangular factors.

First, we use the fact that for LU decomposition of $Q - \begin{bmatrix} S \\ 0 \end{bmatrix}$ into triangular factors $\tilde{Y}$ and $\tilde{U}$, we have (see [25, Section 9.3])

$$\left\| \left(Q - \begin{bmatrix} S \\ 0 \end{bmatrix} \right) - \tilde{Y}\tilde{U} \right\|_F \leq \gamma_b \|\tilde{Y}\|_F \|\tilde{U}\|_F.$$

Second, we compute $\tilde{T} = (\tilde{U}S)Y_1^{-T}$ using the standard algorithm for triangular solve with multiple right hand sides to obtain (see [25, Section 8.1])

$$\left\| \tilde{U} - \tilde{T}\tilde{Y}_1^T S \right\|_F = \left\| \tilde{U}S - \tilde{T}\tilde{Y}_1^T \right\|_F \leq \gamma_b \|\tilde{T}\|_F \|\tilde{Y}_1^T\|_F.$$

Finally, we bound the norms of the computed triangular factors in exact arithmetic. At each step of the Modified-LU algorithm, the trailing matrix is an orthogonal matrix (before modifying the diagonal element). Each column of $\tilde{Y}$ is computed by scaling down elements of a unit vector and setting the diagonal element to 1. Thus, the sum of squares of elements in a column is bounded by 2, and $\|\tilde{Y}\|_F \leq \sqrt{2b}$. Note that this also implies $\|\tilde{Y}_1\|_F \leq \sqrt{2b}$. Each row of $\tilde{U}$ is computed by increasing the absolute value of the diagonal element by 1, but leaving the other elements to the right of the diagonal unchanged. The sum of squares of elements in a row of a matrix with orthonormal columns is at most 1. Thus, the sum of squares of elements in a row of $\tilde{U}$ is bounded by 4, and $\|\tilde{U}\|_F \leq 2\sqrt{b}$. From [22, Theorem 13], we have $\|\tilde{T}\|_F < b + 1$.

Therefore, altogether we have

$$\begin{aligned}
\left\| QS - \left(\begin{bmatrix} I \\ 0 \end{bmatrix} - \tilde{Y}\tilde{T}\tilde{Y}_1^T \right) \right\|_F &= \left\| Q - \left(\begin{bmatrix} S \\ 0 \end{bmatrix} - \tilde{Y}\tilde{T}\tilde{Y}_1^T S \right) \right\|_F \\
&= \left\| \left(Q - \begin{bmatrix} S \\ 0 \end{bmatrix} \right) - \tilde{Y}\tilde{U} + \tilde{Y}\left(\tilde{U} - \tilde{T}\tilde{Y}_1^T S \right) \right\|_F \\
&\leq \left\| \left(Q - \begin{bmatrix} S \\ 0 \end{bmatrix} \right) - \tilde{Y}\tilde{U} \right\|_F + \|\tilde{Y}\|_F \left\| \tilde{U} - \tilde{T}\tilde{Y}_1^T S \right\|_F \\
&\leq \gamma_b \|\tilde{Y}\|_F \|\tilde{U}\|_F + \gamma_b \|\tilde{Y}\|_F \|\tilde{T}\|_F \|\tilde{Y}_1^T\|_F \\
&\leq \gamma_b \left(\sqrt{8}b + 2b(b+1) \right).
\end{aligned}$$

□

Theorem 5.5. *Let $\hat{R}$ be the computed upper triangular factor of $m \times b$ matrix A obtained via the TSQR algorithm using a binary tree of height L ($m/2^L \geq b$), and let $\tilde{Q} = I - \tilde{Y}\tilde{T}\tilde{Y}_1^T$ and $\tilde{R} = S\hat{R}$ where $\tilde{Y}$, $\tilde{T}$, and S are the computed factors obtained from the Householder reconstruction algorithm. Then*

$$\|A - \tilde{Q}\tilde{R}\|_F \leq F_1(m, b, L, \varepsilon)\|A\|_F$$

and

$$\|I - \tilde{Q}^T\tilde{Q}\|_F \leq F_2(m, b, L, \varepsilon)$$

where

$$F_1(m, b, L, \varepsilon) \simeq (b\gamma_{c\cdot\frac{m}{2^L}} + Lb\gamma_{c\cdot 2b})(1 + \sqrt{b}) + 2\gamma_b \left(b^2 + \left(1 + \sqrt{2} \right) b \right)$$

and

$$F_2(m, b, L, \varepsilon) \simeq 2(b\gamma_{c\cdot\frac{m}{2^L}} + Lb\gamma_{c\cdot 2b})\sqrt{b} + 4\gamma_b \left(b^2 + \left(1 + \sqrt{2} \right) b \right)$$

for $b\gamma_{c\cdot\frac{m}{2^L}} \ll 1$ and $b\gamma_{c\cdot 2b} \ll 1$.

Proof. From Lemmas 5.3 and 5.4, there exists an exactly orthonormal matrix Q such that

$$\begin{aligned}
\|A - \tilde{Q}\tilde{R}\|_F &= \|A - Q\hat{R} - (Q - \hat{Q})\hat{R} - (\hat{Q} - \tilde{Q}S)\hat{R}\|_F \\
&\leq \|A - Q\hat{R}\|_F + \|Q - \hat{Q}\|_F \|R\|_F + \|\hat{Q} - \tilde{Q}S\|_F \|R\|_F \\
&\leq f_1(m, b, L, \varepsilon)\|A\|_F + f_2(m, b, L, \varepsilon)\|\hat{R}\|_F + f_3(b, \varepsilon)\|\hat{R}\|_F \\
&\leq F_1(m, b, L, \varepsilon)\|A\|_F.
\end{aligned}$$

The approximation of F_1 also uses [24, Lemma 1]: $\|\hat{R}\|_F \simeq \|A\|_F$ assuming $b\gamma_{c\cdot\frac{m}{2^L}} \ll 1$ and $b\gamma_{c\cdot 2b} \ll 1$.

Further, since Q is exactly orthonormal, we have

$$\begin{aligned}
\|I - \tilde{Q}^T\tilde{Q}\|_F &= \|Q^T(Q - \tilde{Q}S) + (Q - \tilde{Q}S)^T\tilde{Q}\|_F \\
&\leq \|Q - \tilde{Q}S\|_F \left(1 + \|\tilde{Q}\|_F\right) \\
&\leq \left(\|Q - \hat{Q}\|_F + \|\hat{Q} - \tilde{Q}S\|_F\right)\left(1 + \|\tilde{Q}\|_F\right) \\
&\leq \left(f_2(m, b, L, \varepsilon) + f_3(b, \varepsilon)\right)\left(1 + \|\tilde{Q}\|_F\right) \\
&\leq F_2(m, b, L, \varepsilon).
\end{aligned}$$

□

6 Numerical experiments

In this section we present numerical results of TSQR-HR. Experiments were conducted on two representative sets of test matrices. The first set is used to check the stability of the algorithm on single panels represented by tall and skinny matrices, while the second set focuses on the factorization of full matrices panel by panel.

6.1 Tall and Skinny Matrices

In this section, we use matrices which are formed by $A = Q \cdot R_\rho$, where Q and R are computed via QR decomposition of an $m \times b$ matrix with i.i.d. entries chosen from a normal distribution. R_ρ is obtained by setting the $\lfloor \frac{n}{2} \rfloor$-th diagonal element of an upper triangular matrix R to a small parameter value ρ. This experimental setup is used to vary the condition number of the matrix and demonstrate instability of the cheaper method described in Section 4.2.

As can be seen from Table 3, for all test cases both the orthogonality and factorization errors of are of order 10^{-15} for TSQR-HR, which is close to $\epsilon = 2^{-52}$ of double precision. This demonstrates the numerical stability of this approach on tall and skinny matrices. Table 3 shows a comparison between three approaches: TSQR-HR, Yamamoto's approach described in Section 3.3, and LU($A - R$) described in Section 4.2. While Yamamoto's approach is as stable as TSQR-HR, the LU($A - R$) method provides unsatisfactory backward errors which grow with the condition number of the matrix.

6.2 Square Matrices

We now present numerical results for the QR factorization of square matrices using a panel-by-panel factorization. The matrices are generated similarly to [26], where the set is chosen from well-known anomalous matrices. Most are of size 1000-by-1000 (except the ARC130 and FS_541_1 matrices, which are respectively of order 130 and 541), with various condition numbers, some of them being very ill-conditioned (i.e. having a condition number much larger than the inverse of machine precision). We tried several panel sizes ranging from 2 to 256 columns and report only the largest errors and their associated panel widths.

Results from Table 5 show that TSQR-HR is numerically stable in terms of backward errors when computing the QR factorization of full matrices, regardless of the condition number of the matrix, as suggested by Theorem 5.5. Again, Yamamoto's approach displays similar results. To the contrary, the alternative approach of LU($A - R$) does not yield numerically stable results in practice, confirming what was observed on tall and skinny matrices. Altogether, these two sets of experiments demonstrate the numerical stability of the proposed approach on representative matrices.

		$Q-I$ (T from Algorithm 5)			Yamamoto's approach			$A-R$ (T^{-1} from Y^TY)		
ρ	κ	norm-wise	col-wise	ortho.	norm-wise	col-wise	ortho.	norm-wise	col-wise	ortho.
1e-01	5.1e+02	2.2e-15	2.7e-15	9.3e-15	2.5e-15	3.1e-15	9.2e-15	3.8e-14	1.7e-14	5.5e-15
1e-02	5.0e+03	2.3e-15	2.9e-15	1.0e-14	2.4e-15	3.1e-15	1.1e-14	3.2e-13	1.1e-13	6.2e-15
1e-03	5.0e+04	2.2e-15	2.6e-15	8.4e-15	2.6e-15	3.4e-15	1.1e-14	4.2e-12	1.7e-12	5.6e-15
1e-04	4.9e+05	2.2e-15	2.6e-15	7.7e-15	2.3e-15	2.8e-15	8.7e-15	3.8e-11	1.7e-11	5.4e-15
1e-05	5.1e+06	2.3e-15	2.9e-15	8.7e-15	3.2e-15	4.2e-15	1.0e-14	3.9e-10	1.4e-10	5.3e-15
1e-06	5.0e+07	2.3e-15	3.0e-15	9.1e-15	3.0e-15	3.9e-15	1.0e-14	3.6e-09	1.5e-09	6.1e-15
1e-07	5.0e+08	2.4e-15	3.4e-15	1.1e-14	2.7e-15	3.7e-15	9.9e-15	4.2e-08	2.1e-08	5.0e-15
1e-08	5.1e+09	2.2e-15	2.8e-15	8.6e-15	2.5e-15	3.1e-15	8.9e-15	3.8e-07	1.5e-07	5.8e-15
1e-09	5.0e+10	2.3e-15	3.1e-15	9.9e-15	3.9e-15	5.1e-15	1.3e-14	3.6e-06	2.0e-06	5.4e-15
1e-10	5.0e+11	2.1e-15	2.6e-15	7.1e-15	2.6e-15	3.4e-15	9.9e-15	3.3e-05	1.2e-05	6.3e-15
1e-11	4.9e+12	2.5e-15	3.4e-15	1.0e-14	2.4e-15	3.1e-15	1.0e-14	3.1e-04	1.2e-04	5.9e-15
1e-12	5.1e+13	2.2e-15	2.9e-15	8.5e-15	2.6e-15	3.3e-15	1.2e-14	3.7e-03	1.6e-03	5.8e-15
1e-13	5.0e+14	2.2e-15	2.7e-15	8.8e-15	3.0e-15	3.9e-15	1.0e-14	4.0e-02	1.4e-02	4.7e-15
1e-14	3.5e+15	2.3e-15	3.1e-15	1.0e-14	2.3e-15	2.9e-15	9.4e-15	2.7e-01	9.7e-02	4.9e-15
1e-15	5.0e+15	2.4e-15	3.1e-15	9.7e-15	2.8e-15	3.7e-15	9.4e-15	3.5e-01	1.3e-01	6.3e-15

Table 3: Error on tall and skinny matrices ($m = 1000, n = 200$) for three approaches. The label "norm-wise" corresponds to $\|A-QR\|_2$, "col-wise" corresponds to $\max_i \|A_i-(QR)_i\|_2$, and "ortho." corresponds to $\|I-Q^TQ\|_2$.

Id	Matrix type	Size	κ
1.	$A = 2 * rand(m) - 1$	1000-by-1000	$2.106e+03$
2.	$A = diag((10 * eps)^{\frac{i}{m}}) * rand(m)$	1000-by-1000	$7.731e+17$
3.	Golub-Klema-Stewart: $A(i,i) = \frac{1}{\sqrt{i}}, A(i,j>i) = \frac{1}{\sqrt{j}}, A(i,j<i) = 0$	1000-by-1000	$2.242e+20$
4.	Break 1 distribution: $A = gallery('randsvd', n, 1e9, 2)$	1000-by-1000	$1.00e+09$
5.	Break 9 distribution: $A = gallery('randsvd', n, 1e9, 1)$	1000-by-1000	$1.00e+09$
6.	$A = orth(rand(n)) \cdot diag([100, 10, linspace(1e-8, 1e-2, n-2)])$ $A = A \cdot orth(rand(n))$	1000-by-1000	$1.00e+10$
7.	$v = linspace(1, 1e-3, n); v(51:end) = 0;$ $A = orth(rand(n)) \cdot diag(v) \cdot orth(rand(n)) + 0.1 * v(50) * rand(n)$	1000-by-1000	$8.464e+04$
8.	$U\Sigma V^T$ with exponential distribution	1000-by-1000	$4.155e+19$
9.	The devil's stairs matrix	1000-by-1000	$2.275e+19$
10.	KAHAN matrix, a trapezoidal matrix	1000-by-1000	$5.642e+56$
11.	Matrix ARC130 from Matrix Market	130-by-130	$6.054e+10$
12.	Matrix FS_541_1 from Matrix Market	541-by-541	$4.468e+03$
13.	BAART Test problem: Fredholm integral equation of the first kind	1000-by-1000	$5.251e+18$
14.	BLUR Test problem: digital image deblurring. A is a symmetric, doubly block Toeplitz matrix	961-by-961	$3.075e+01$
15.	DERIV2 Test problem: computation of the second derivative	1000-by-1000	$1.216e+06$
16.	Matrix Ex1-CMRS	1000-by-500	$1.398e+17$
17.	Matrix Ex2-RST	1024-by-512	$3.276e+16$
18.	FOXGOOD Test problem: severely ill-posed problem	1000-by-1000	$5.742e+20$
19.	GRAVITY Test problem: 1-D gravity surveying model problem	1000-by-1000	$8.797e+20$
20.	HEAT Test problem: inverse heat equation	1000-by-1000	$1.070e+232$
21.	PARALLAX Stellar parallax problem with 28 fixed, real observations	26-by-1000	$4.629e+14$
22.	PHILLIPS Test problem: discretization of the 'famous' first-kind Fredholm integral equation deviced by D. L. Phillips	1000-by-1000	$2.641e+10$
23.	SHAW Test problem: one-dimensional image restoration model	1000-by-1000	$2.441e+21$
24.	SPIKES Test problem with a "spiky" solution	1000-by-1000	$5.796e+21$
25.	TOMO is a 2D tomography test problem	961-by-961	$1.091e+17$

Table 4: Description of the full matrices used in the experiments.

	$Q - I$ (T from Algorithm 5)			Yamamoto's approach			$A - R$ (T^{-1} from Y^TY)		
Id	norm-wise error	column-wise error	orthogonality	norm-wise error	column-wise error	orthogonality	norm-wise error	column-wise error	orthogonality
1	4.3e-15 (256)	3.7e-15 (256)	2.8e-14 (2)	4.7e-15 (256)	4.4e-15 (256)	2.8e-14 (2)	3.5e-15 (256)	3.5e-15 (256)	2.8e-14 (2)
2	2.0e-15 (256)	6.6e-15 (64)	2.6e-14 (2)	2.6e-15 (256)	7.4e-15 (256)	2.6e-14 (2)	2.7e-15 (256)	9.0e-15 (256)	3.1e-14 (2)
3	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)
4	1.0e-14 (256)	5.5e-15 (256)	2.8e-14 (2)	1.1e-14 (256)	6.3e-15 (256)	2.8e-14 (2)	4.4e-14 (256)	4.4e-14 (256)	2.8e-14 (2)
5	9.9e-15 (256)	5.1e-15 (256)	2.9e-14 (2)	1.0e-14 (256)	6.2e-15 (256)	2.8e-14 (2)	4.2e-14 (256)	3.0e-14 (256)	2.8e-14 (2)
6	1.4e-15 (256)	5.9e-15 (256)	2.8e-14 (2)	2.2e-15 (128)	6.9e-15 (256)	2.8e-14 (2)	1.4e-15 (256)	1.8e-14 (256)	2.8e-14 (2)
7	1.4e-15 (256)	4.6e-15 (64)	2.7e-14 (2)	3.5e-15 (256)	7.7e-15 (256)	2.8e-14 (2)	2.3e-15 (256)	5.8e-15 (256)	2.8e-14 (2)
8	2.0e-15 (256)	4.3e-15 (64)	2.8e-14 (2)	3.8e-15 (256)	7.0e-15 (256)	2.8e-14 (2)	1.9e-15 (256)	1.4e-14 (256)	2.8e-14 (2)
9	2.4e-15 (256)	5.0e-15 (256)	2.8e-14 (2)	2.7e-15 (256)	5.4e-15 (256)	2.8e-14 (2)	2.9e-15 (256)	1.5e-14 (256)	2.8e-14 (2)
10	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)	0.0e+00 (2)
11	8.8e-19 (16)	1.3e-15 (16)	2.1e-15 (2)	1.1e-18 (16)	1.8e-15 (16)	3.9e-15 (16)	3.9e-19 (32)	8.7e-16 (32)	1.7e-15 (2)
12	5.8e-16 (64)	1.3e-15 (2)	1.8e-15 (256)	8.4e-16 (16)	1.9e-15 (32)	2.8e-15 (64)	6.0e-16 (256)	1.3e-15 (32)	1.7e-15 (2)
13	1.6e-15 (32)	6.2e-15 (256)	2.9e-14 (2)	2.0e-15 (32)	7.3e-15 (256)	2.8e-14 (2)	3.0e-07 (256)	1.5e-06 (256)	2.8e-14 (2)
14	1.0e-15 (32)	1.5e-15 (16)	4.7e-15 (2)	1.6e-15 (256)	2.8e-15 (256)	6.8e-15 (128)	8.2e-16 (16)	1.5e-15 (16)	3.9e-15 (2)
15	2.8e-15 (256)	8.7e-15 (256)	4.6e-14 (2)	1.0e-14 (256)	1.6e-14 (256)	4.8e-14 (2)	1.0e-12 (256)	5.3e-12 (256)	5.6e-14 (2)
16	1.8e-15 (256)	5.7e-15 (256)	4.9e-14 (2)	1.7e-15 (256)	5.7e-15 (256)	4.0e-14 (2)	4.3e-07 (256)	2.3e-06 (256)	3.3e-14 (2)
17	3.7e-15 (32)	1.1e-14 (16)	5.0e-14 (2)	3.0e-15 (16)	9.8e-15 (32)	3.8e-14 (2)	1.4e-14 (256)	2.3e-14 (256)	4.5e-14 (2)
18	2.4e-15 (256)	6.4e-15 (16)	2.8e-14 (2)	3.5e-15 (256)	8.0e-15 (256)	2.8e-14 (2)	8.2e-04 (256)	5.7e-03 (256)	2.8e-14 (2)
19	2.3e-15 (256)	4.8e-15 (64)	2.9e-14 (2)	3.1e-15 (256)	7.1e-15 (256)	2.8e-14 (2)	2.1e-03 (256)	1.1e-02 (256)	2.8e-14 (2)
20	2.6e-15 (256)	4.5e-15 (256)	3.9e-14 (2)	2.6e-15 (256)	5.1e-15 (256)	4.0e-14 (2)	3.8e-02 (256)	1.7e-01 (256)	3.4e-14 (2)
21	8.8e-16 (16)	1.9e-15 (32)	2.6e-15 (16)	1.1e-15 (32)	2.3e-15 (32)	2.5e-15 (32)	1.1e-11 (32)	1.8e-10 (32)	2.3e-15 (16)
22	1.0e-15 (32)	4.9e-15 (32)	3.7e-14 (2)	1.2e-15 (256)	5.0e-15 (256)	3.1e-14 (2)	5.2e-11 (256)	1.9e-10 (256)	3.4e-14 (2)
23	2.5e-15 (256)	5.3e-15 (64)	2.8e-14 (2)	2.8e-15 (256)	7.1e-15 (256)	2.9e-14 (2)	7.0e-04 (256)	5.2e-03 (256)	2.8e-14 (2)
24	7.2e-16 (32)	3.8e-15 (256)	2.8e-14 (2)	1.2e-15 (256)	4.7e-15 (256)	2.8e-14 (2)	2.2e-03 (256)	2.6e-02 (256)	2.8e-14 (2)
25	2.2e-15 (256)	4.3e-15 (256)	2.2e-14 (2)	2.3e-15 (256)	5.0e-15 (256)	2.2e-14 (2)	2.3e-15 (256)	1.2e-14 (256)	2.3e-14 (2)

Table 5: Error of full matrices ($n = 1000$). The label "norm-wise" corresponds to $\|A - QR\|_2$, "col-wise" corresponds to $\max_i \|A_i - (QR)_i\|_2$, and "ortho." corresponds to $\|I - Q^TQ\|_2$. Descriptions of the matrices can be found in Table 4.

7 Performance

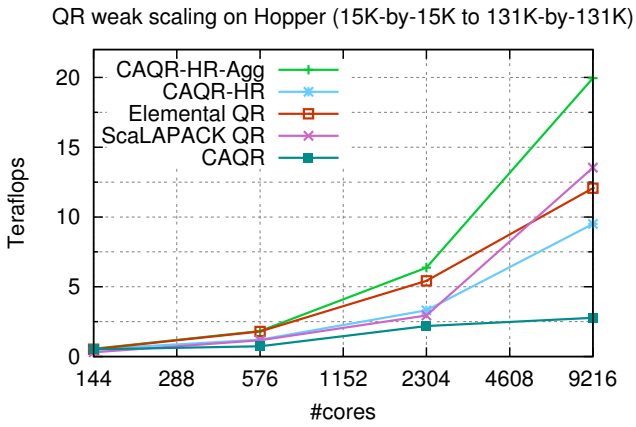
Having established the stability of our algorithm, we now analyze its experimental performance. We demonstrate that for tall and skinny matrices TSQR-HR achieves better parallel scalability than library implementations (ScaLAPACK and Elemental) of Householder-QR. Further, we show that for square matrices CAQR-HR-Agg outperforms our implementation of CAQR, and library implementations of 2D-Householder-QR.

7.1 Architecture

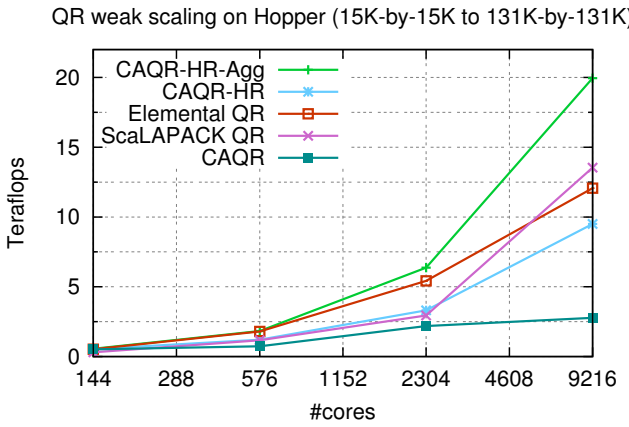
The experimental platform is "Hopper," which is a Cray XE6 supercomputer, built from dual-socket 12-core "Magny-Cours" Opteron compute nodes. We used the Cray LibSci BLAS routines. This machine is located at the NERSC supercomputing facility. Each node can be viewed as a four-chip compute configuration due to NUMA domains. Each of these four chips have six super-scalar, out-of-order cores running at 2.1 GHz with private 64 KB L1 and 512 KB L2 caches. Nodes are connected through Cray's "Gemini" network, which has a 3D torus topology. Each Gemini chip, which is shared by two Hopper nodes, is capable of 9.8 GB/s bandwidth.

7.2 Parallel Scalability

In this section, we give performance results based on a C++/MPI/LAPACK implementation of TSQR and TSQR-HR, as well as two library implementations of 2D-Householder-QR, Elemental (version 0.80) and ScaLAPACK (native LibSci installation on Hopper, October 2013). Our implementations aim to do minimal communication and arithmetic, and do not employ low-level tuning or overlap between communication and computation. All the benchmarks use one MPI process per core, despite the fact that is favorable on Hopper to use one process per socket and six threads per process. This decision was made because we observed that some of the many LAPACK routines used throughout our codes (`geqrf`, `ormqr`, `tpqrt`, `tmpqrt`, etc.) were not threaded.



(a) Tall-skinny QR performance on Cray XE6



(b) Square QR performance on Cray XE6

First, we study the performance of QR factorization of tall-skinny matrices using a 1D processor grid. Figure 1(a) gives the strong scaling performance for a matrix of size 122,880-by-32. We also tested a range of reasonable panel sizes that are not detailed here and observed similar performance trends. We observe from Figure 1(a) that TSQR-HR takes roughly twice the execution time of TSQR, which is in line with our theoretical cost analysis. Figure 1(a) also gives the time to solution of Elemental and ScaLAPACK, which both use the Householder-QR algorithm, albeit with different matrix blocking and collectives. We see that TSQR obtains a performance benefit over Householder-QR due to the lower synchronization cost and TSQR-HR preserves the scaling behavior with roughly a factor of two overhead.

Second, we study the parallel scaling of QR factorization on square matrices. In Figure 1(b), we compare our implementation of CAQR (with a simple binary tree update, no pipelining or other optimizations), CAQR-HR, and CAQR-HR-Agg, with Elemental and ScaLAPACK, which use 2D-Householder-QR. We tuned the block sizes of all the codes (the CAQR-HR-Agg required tuning two block sizes), though fewer data points were collected for larger scale runs, due to timing and allocation constraints.

Comparing the performance of CAQR-HR-Agg and CAQR-HR in Figure 1(b), we observe that significant benefit is obtained from aggregating the trailing matrix update. By combining the aggregated update and lower synchronization cost of TSQR, CAQR-HR-Agg outperforms ScaLAPACK and Elemental and achieves better parallel scalability. On the other hand, the CAQR performance is relatively poor due to the overhead of the implicit tree trailing update. We also note that for Elemental, ScaLAPACK, and all of our QR implementations, it was often better to utilize a rectangular processor grid with more rows than columns. Having more processes in each column of the processor grid accelerates the computation of each tall-skinny panel.

8 Conclusion

In this paper, we introduce a method for recovering the Householder basis-kernel representation from any matrix with orthonormal columns in a stable and efficient manner. Our method was motivated by the desire to combine the efficiency of the TSQR algorithm with that of the trailing matrix updates within Householder-QR in the context of computing the QR factorization of general matrices.

We argue using a performance cost model that the savings from combining the approaches outweigh the extra cost required to reconstruct the Householder vectors for each panel factorization, observing asymptotic improvements over both existing approaches. We also demonstrate that our algorithm is practical, attaining speed-ups of up to 1.5× for CAQR-HR-Agg over the standard 2D-Householder-QR algorithm, and we conjecture that larger speed-ups may be achieved on future architectures, which are becoming increasingly more synchronization and communication bound.

Our approach provides a promising direction for heterogenous architectures (as suggested in [11]), where

synchronization-avoidance and high granularity computation have even more pervasive effects on performance efficiency. Furthermore, because our approach recovers the standard representation of orthogonal matrices (as is used in libraries like LAPACK), we are able to re-use the existing software infrastructure and maintain performance portability.

Finally, we conjecture that the Householder reconstruction technique will enable the design of a QR algorithm which is as stable as Householder QR and reduces the bandwidth cost asymptotically compared to parallel CAQR. We aim to reduce the asymptotic bandwidth cost for QR as done by Tiskin [27], except in a more practical manner, following the communication-optimal parallel algorithm for LU [28].

9 Acknowledgements

We would like to thank Yusaku Yamamoto for sharing his slides from SIAM ALA 2012 with us. We also thank Jack Poulson for help configuring Elemental.

Solomonik was supported by a Department of Energy Computational Science Graduate Fellowship, grant number DE-FG02-97ER25308. This research used resources of the National Energy Research Scientific Computing Center (NERSC), which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. We also acknowledge DOE grants DE-SC0004938, DE-SC0005136, DE-SC0003959, DE-SC0008700, DE-SC0010200, AC02-05CH11231, and DARPA grant HR0011-12-2-0016. Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

References

[1] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential QR and LU factorizations," *SIAM Journal on Scientific Computing*, vol. 34, no. 1, pp. A206–A239, 2012.

[2] G. H. Golub, R. J. Plemmons, and A. Sameh, "Parallel block schemes for large-scale least-squares computations," in *High-speed computing: scientific applications and algorithm design*, R. B. Wilhelmson, Ed. Champaign, IL, USA: University of Illinois Press, 1988, pp. 171–179.

[3] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in numerical linear algebra," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 3, pp. 866–901, 2011.

[4] F. Song, H. Ltaief, B. Hadri, and J. Dongarra, "Scalable tile communication-avoiding QR factorization on multicore cluster systems," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–11.

[5] M. Anderson, G. Ballard, J. Demmel, and K. Keutzer, "Communication-avoiding QR decomposition for GPUs," in *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*, ser. IPDPS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 48–58.

[6] E. Agullo, C. Coti, J. Dongarra, T. Herault, and J. Langem, "QR factorization of tall and skinny matrices in a grid computing environment," in *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, 2010, pp. 1–11.

[7] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*. Philadelphia, PA, USA: SIAM, 1992.

[8] 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, and R. C. Whaley, *ScaLAPACK Users' Guide*. Philadelphia, PA, USA: SIAM, May 1997.

[9] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, "Elemental: A new framework for distributed memory dense matrix computations," *ACM Trans. Math. Softw.*, vol. 39, no. 2, pp. 13:1–13:24, Feb. 2013.

[10] R. Schreiber and C. Van Loan, "A storage-efficient WY representation for products of Householder transformations," *SIAM Journal on Scientific and Statistical Computing*, vol. 10, no. 1, pp. 53–57, 1989.

[11] Y. Yamamoto, "Aggregation of the compact WY representations generated by the TSQR algorithm," 2012, Conference talk presented at SIAM Applied Linear Algebra.

[12] A. Farley, "Broadcast time in communication networks," *SIAM Journal on Applied Mathematics*, vol. 39, no. 2, pp. 385–390, 1980.

[13] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[14] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn, "Collective communication: theory, practice, and experience," *Concurrency and Computation: Practice and Experience*, vol. 19, no. 13, pp. 1749–1783, 2007.

[15] G. Golub and C. Van Loan, *Matrix Computations*, ser. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 2012.

[16] C. Puglisi, "Modification of the Householder method based on compact WY representation," *SIAM Journal on Scientific and Statistical Computing*, vol. 13, no. 3, pp. 723–726, 1992.

[17] J. Demmel, L. Grigori, M. F. Hoemmen, and J. Langou, "Communication-optimal parallel and sequential QR and LU factorizations," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-89, Aug 2008.

[18] G. Ballard, J. Demmel, L. Grigori, M. Jacquelin, H. D. Nguyen, and E. Solomonik, "Reconstructing Householder vectors from tall-skinny QR," EECS Department, University of California, Berkeley, Tech. Rep., 2013.

[19] M. Hoemmen, "A communication-avoiding, hybrid-parallel, rank-revealing orthogonalization method," in *Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International*, 2011, pp. 966–977.

[20] X. Sun and C. Bischof, "A basis-kernel representation of orthogonal matrices," *SIAM Journal on Matrix Analysis and Applications*, vol. 16, no. 4, pp. 1184–1196, 1995.

[21] R. Schreiber and B. Parlett, "Block reflectors: Theory and computation," *SIAM Journal on Numerical Analysis*, vol. 25, no. 1, pp. 189–205, 1988.

[22] C. H. Bischof and X. Sun, "On orthogonal block elimination," Argonne National Laboratory, Argonne, IL, Tech. Rep. MCS-P450-0794, 1994.

[23] Y. Yamamoto, 2012, Personal communication.

[24] D. Mori, Y. Yamamoto, and S.-L. Zhang, "Backward error analysis of the AllReduce algorithm for Householder QR decomposition," *Japan Journal of Industrial and Applied Mathematics*, vol. 29, no. 1, pp. 111–130, 2012.

[25] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA: SIAM, 2002.

[26] J. Demmel, L. Grigori, M. Gu, and H. Xiang, "Communication avoiding rank revealing QR factorization with column pivoting," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-46, May 2013.

[27] A. Tiskin, "Communication-efficient parallel generic pairwise elimination," *Future Generation Computer Systems*, vol. 23, no. 2, pp. 179 – 188, 2007.

[28] E. Solomonik and J. Demmel, "Communication-optimal 2.5D matrix multiplication and LU factorization algorithms," in *Springer Lecture Notes in Computer Science, Proceedings of Euro-Par, Bordeaux, France*, Aug 2011.

A Improved CAQR

In this Appendix, we show how the CAQR algorithm can be improved via a more efficient trailing matrix update algorithm. The main idea of this approach is to use a recursive halving and recursive doubling approach over the columns of the trailing matrix while applying the TSQR tree. This task can be achieved via a butterfly communication network which performs the update and scatters the data, followed by an inverted butterfly network which collects the scattered data via recursive doubling. Performing the trailing matrix update in this manner also requires replicating the TSQR tree as a butterfly (Householder data replicated 2^k times at the kth level). The Householder tree data may be replicated explicitly via communication or can be computed redundantly during the factorization phase via a butterfly network TSQR (see Algorithm 9) for no extra parallel cost. We arrived at this algorithm only after implementing and evaluating CAQR with a simple binary tree network for the TSQR and the trailing matrix update. We plan to implement and study the performance of this trailing matrix update in the context of CAQR, as it alleviates important algorithmic bottlenecks on which we elaborate below.

Algorithm 9 demonstrates how TSQR can be done via butterfly tree network. The main difference between the butterfly algorithm and the binary tree TSQR is the fact that the butterfly algorithm computes and stores each $Y_{i,k}$ redundantly on 2^k processors on level k. Algorithm 10 uses the redundantly stored representation produced by Algorithm 9 to perform the trailing matrix update more efficiently. In particular, at each level of the first butterfly network in Algorithm 10, the working part of the columns of the trailing matrix are partitioned in half and scattered amongst pairs of processor rows. In the binary tree application method (Algorithm 3), half the processors were assigned work at each successive level of the tree. In our butterfly algorithm, all the processors are assigned half the work at each successive level of the butterfly. As a result, the number of columns each processor works on reduces in half at each successive level in the butterfly. We note that Algorithm 10 is different from the butterfly algorithm for the trailing matrix application presented in [19], which applies the redundant copies of $Y_{i,k}$ to the columns of the trailing matrix redundantly.

The benefit achieved by Algorithm 10 over Algorithm 3 is reflected both in communication bandwidth and computational cost. The number of words moved by each processor in Algorithm 10 is given to leading order by

$$2\sum_{i=1}^{\log p} nb/2^i = 2nb.$$

We note that this is a factor of $\log p$ smaller than the bandwidth cost associated with Algorithm 3. Further, excluding the cost of the initial application on line 1 (which also occurs in Algorithm 3), the computational

cost of the algorithm is given by

$$\sum_{i=1}^{\log p} O(nb^2/2^i) = O(nb^2),$$

which compares favorably to Algorithm 3 which has a cost of $O(nb^2 \log p)$. The reduction in bandwidth cost yielded by this algorithm leads directly to a reduction in bandwidth cost from $O(\frac{mn+n^2 \log p}{\sqrt{p}})$ to $O(\frac{mn+n^2}{\sqrt{p}})$ (where we assume $p_r = p_c = \sqrt{p}$). Further, the reduction in floating point cost makes it possible to select a block size for CAQR which is a factor of $\log p$ larger without incurring any overhead in leading order computational cost. Previously, the block size was made smaller due to the $O(n^2 b \log p)$ computational term associated with using Algorithm 3 for the trailing matrix update. Raising the block size by a factor of $\log p$ yields a reduction in the latency cost of CAQR by the same factor. The overall cost of the improved CAQR algorithm (using Algorithms 9 and 10) is given to leading order for nearly square matrices by

$$\gamma \cdot \left(\frac{2mn^2 - 2n^3/3}{p} \right) + \beta \cdot \left(\frac{2mn + 2n^2}{\sqrt{p}} \right) + \alpha \cdot \left(7\sqrt{p} \log^2 p \right).$$

Therefore, it is possible for CAQR to achieve the same asymptotic costs as CAQR-HR; however, it is not possible to aggregate the update to obtain the practical benefits harvested by CAQR-HR-Agg. Further, the reconstruction of Householder vectors requires less software engineering to be incorporated into high performance numerical linear algebra libraries.

Algorithm 9 $[\{Y_{i,k}\}, R] = \text{Butterfly-TSQR}(A)$

Require: Number of processors, p, is a power of two and i is the processor index
Require: A is $m \times b$ matrix distributed in block row layout; A_i is processor i's block
1: $[Y_{i,0}, \bar{R}_i] = \text{Householder-QR}(A_i)$
2: **for** $k = 1$ to $\log p$ **do**
3: % *Determine my neighbor in this level of the butterfly*
4: $j = 2^k \lfloor \frac{i}{2^k} \rfloor + (i + 2^{k-1} \bmod 2^k)$
5: **if** $i < j$ **then**
6: Send $\bar{R}_i$ from processor j
7: Receive $\bar{R}_j$ from processor j
8: $[Y_{i,k}, \bar{R}_i] = \text{Householder-QR}\left(\begin{bmatrix} \bar{R}_i \\ \bar{R}_j \end{bmatrix} \right)$
9: **else**
10: Receive $\bar{R}_j$ from processor j
11: Send $\bar{R}_i$ to processor j
12: $[Y_{i,k}, \bar{R}_i] = \text{Householder-QR}\left(\begin{bmatrix} \bar{R}_j \\ \bar{R}_i \end{bmatrix} \right)$
13: **end if**
14: **end for**
15: $R = \bar{R}_i$
Ensure: $A = QR$ with Q implicitly represented by $\{Y_{i,k}\}$
Ensure: R is stored redundantly on all processors, $Y_{0,0}$ is stored by processor 0, and $Y_{i,k}$ for $i > 0$ is stored redundantly on processor j for each $j + 2^{k-1} = i \bmod 2^k$

B Householder-QR with Hints

The idea in this section is to interpret computing the LU decomposition of $A - R$ (discussed in Section 4.2) as performing Householder-QR on A, but we will use the information in R as "hints" to avoid parts of the computation (particularly the communication-expensive parts).

Algorithm 10 $[B]$ = Scatter-Apply-TSQR-Q^T($\{Y_{i,k}\}, A$)

Require: Number of processors, p, is a power of two and i is the processor index
Require: A is $m \times n$ matrix distributed in block row layout; A_i is processor i's block
Require: $\{Y_{i,k}\}$ is the implicit representation of b Householder vectors computed via a butterfly TSQR.

B_i = Apply-Householder-$Q^T(Y_{i,0}, A_i)$
Let $\bar{B}_i$ be the first b rows of B_i
% *Butterfly network recursive-halving TSQR*
for $k = 1$ to $\log p$ **do**
    % *Determine my neighbor in this level of the butterfly*
    $j = 2^k \lfloor \frac{i}{2^k} \rfloor + (i + 2^{k-1} \bmod 2^k)$
    % *Subdivide columns via recursive halving*
    Let $\bar{B}_i = [\bar{B}_{i1}, \bar{B}_{i2}]$ where each block is b-by-$n/2^k$
    if $i < j$ **then**
        % *Send half my columns to my neighbor and receive half of his*
        Send $\bar{B}_{i2}$ to processor j
        Receive $\bar{B}_{j1}$ from processor j
        % *Apply the Householder vectors to stacked blocks to compute input to next level of butterfly*
        $\begin{bmatrix} \bar{B}_i \\ \bar{B}_{j1}^k \end{bmatrix}$ = Apply-Householder-$Q^T \left(Y_{i,k}, \begin{bmatrix} \bar{B}_{i1} \\ \bar{B}_{j1} \end{bmatrix} \right)$
        % *Lower rows are not acted on at further levels in butterfly, so can be returned immediately*
        Send $\bar{B}_{j1}^k$ back to processor j
    else
        Receive $\bar{B}_{j2}$ from processor j
        Send $\bar{B}_{i1}$ to processor j
        $\begin{bmatrix} \bar{B}_i \\ \bar{B}_{i2}^k \end{bmatrix}$ = Apply-Householder-$Q^T \left(Y_{i,k}, \begin{bmatrix} \bar{B}_{i2} \\ \bar{B}_{j2} \end{bmatrix} \right)$
        Receive updated rows $\bar{B}_{i1}^k$ back from processor j
    end if
end for
% *Propagate computed B back to origins via recursive doubling*
for $k = \log p$ down to 1 **do**
    % *Determine my neighbor in this level of the butterfly*
    $j = 2^k \lfloor \frac{i}{2^k} \rfloor + (i + 2^{k-1} \bmod 2^k)$
    if $i < j$ **then**
        % *Collect upper rows*
        Receive $\bar{B}_j$ from processor j
        $\bar{B}_i = [\bar{B}_i, \bar{B}_j]$
    else
        % *Collect lower rows*
        Send $\bar{B}_i$ to processor j
        $\bar{B}_i = [\bar{B}_{i1}^k, \bar{B}_{i2}^k]$
    end if
end for
Set the first b rows of B_i to $\bar{B}_i$

Ensure: $B = Q^T A$ where Q is the orthogonal matrix implicitly represented by $\{Y_{i,k}\}$

Algorithm 11 $[Y, S]$ = Householder-QR-with-Hints(A, R)

Require: A is $m \times b$, R is upper triangular such that $R = Q^T A$ for some orthogonal Q

1: $S = I$
2: **for** $i = 1$ to b **do**
   % *Compute the Householder vector to annihilate sub-diagonal entries in the ith column*
3: $\alpha = A(i, i)$
   % *Compute column norm from diagonal entry of R*
4: $\beta = R(i, i)$
5: **if** $\text{sgn}(\beta) = \text{sgn}(\alpha)$ **then**
6: $S(i, i) = -1$
7: $\beta = -\beta$
8: **end if**
9: $\tau(i) = \frac{\beta - \alpha}{\beta}$
10: $A(i + 1 : m, i) = \frac{1}{A(i,i) - \beta} \cdot A(i + 1 : m, i)$
   % *Apply the Householder transformation to the trailing matrix*
   % *Compute z from the ith row of R*
11: $z = A(i, i + 1 : n) - S(i, i) \cdot R(i, i + 1 : n)$
12: $A(i + 1 : m, i + 1 : n) = A(i + 1 : m, i + 1 : n) - A(i + 1 : m, i) \cdot z$
13: **end for**

Ensure: $A = \left(\prod_{i=1}^{n} (I - \tau_i y_i y_i^T)\right) SR$

Ensure: Y (the Householder vectors) has implicit unit diagonal and overwrites the strict lower triangle of A; τ is an array of length b with $\tau_i = 2/(y_i^T y_i)$

Let $A^{(i)}$ be the partially factored matrix whose first i columns form an upper triangular matrix. Since Y is a lower triangular matrix (the height of the nonzero part of the Householder vectors decreases as the algorithm progresses), the i^{th} row of the trailing matrix is not updated after the i^{th} Householder reflection is applied. This implies that the i^{th} row of $A^{(i)}$ is equal to the i^{th} row of the final output; that is, $A^{(i)}(i, :) = R(i, :)$.

In order to compute the Householder vector y_i, we must compute the norm of $A^{(i-1)}(i : m, i)$ and scale each entry below the diagonal by the reciprocal of the norm. Computing this norm requires a reduction, but we can avoid this calculation because the norm is given by the absolute value of the diagonal entry $R(i, i)$.

Further, after the Householder vector y_i is computed, the orthogonal reflector is applied to the trailing matrix in the form of a rank-one update. This update is given by

$$A^{(i)} = \left(I - \tau_i y_i y_i^T\right) A^{(i-1)} = A^{(i-1)} - y_i \left(\tau y_i^T A^{(i-1)}\right) = A^{(i-1)} - y_i z_i \tag{10}$$

where z_i is the i^{th} row of Z, y_i is the i^{th} column of Y, and $\tau_i = \frac{2}{y_i^T y_i}$. Note that the i^{th} entry of y_i, or $Y(i, i)$, is always 1. Then by Equation (10), we have

$$R(i, :) = A^{(i)}(i, :) = A^{(i-1)}(i, :) - y_i(i) \cdot z_i = A^{(i-1)}(i, :) - z_i$$

so that $z_i = A^{(i-1)}(i, :) - R(i, :)$. This implies that z_i can be computed via subtraction rather than from the formula $z_i = \tau y_i^T A^{(i-1)}$, thereby avoiding the matrix-vector product. Algorithm 11 shows the Householder-QR-with-Hints algorithm which incorporates the cheaper means of computing column norms and vectors z_i. Close comparison of Algorithm 11 with Algorithm 1 will show the computational savings from lines 4 and 11, where hints from R are used to compute the relevant quantities more cheaply.