2014 IEEE 28th International Parallel & Distributed Processing Symposium

Reconstructing Householder Vectors from Tall-Skinny QR

Grey Ballard*, James Demmel†, Laura Grigori‡, Mathias Jacquelin§, Hong Diep Nguyen†, and Edgar Solomonik†

*Sandia National Laboratories, Livermore, USA
†University of California, Berkeley, Berkeley, USA
‡INRIA Paris - Rocquencourt, Paris, France
§Lawrence Berkeley National Laboratory, Berkeley, USA

gmballa@sandia.gov, demmel@cs.berkeley.edu, laura.grigori@inria.fr,
mjacquelin@lbl.gov, hdnguyen@cs.berkeley.edu, solomon@cs.berkeley.edu

***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. We also provide algorithmic improvements to the ScaLAPACK and CAQR algorithms.

I. 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 the trailing matrix update within CAQR is less communication efficient than the update within Householder-QR by a logarithmic factor in the number of

1530-2075/14 $31.00 © 2014 IEEE
DOI 10.1109/IPDPS.2014.120

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 and use two levels of aggregation of Householder vectors. This abstraction makes it possible to optimize panel factorization and the trailing matrix updates independently. Our resulting parallel implementation outperforms ScaLAPACK, Elemental, and a binary-tree 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.

Two other contributions of the paper include improvements to the Householder QR and CAQR algorithms for square matrices. We employ the two-level aggregation technique within Householder QR to alleviate a memory bandwidth bottleneck (see Section IV-C), and we use a more efficient trailing matrix update within CAQR that improves both the computation and communication costs of that algorithm (see Section IV-D). Both optimizations lead to significant performance improvement for the two algorithms.

II. 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, Using recursive doubling/halving or pipelined broadcast algorithms [12], [13], [14] the cost of an array broadcast of length $w \geq p$ is

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

(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).

III. 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 that 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).

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

[1] This notation is not to be confused with the communication cost model given in Equation (1).

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 $\tau_i = 2/(y_i^T y_i)$

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. 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 algorithm also performs $2mb^2/p$ flops and communicates $(b^2/2) \log p$ words.

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 I, 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

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

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 II.

B. 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.

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.

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 I. 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.

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: $[Y_{i,0}, \bar{R}_i] =$ 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: $[Y_{i,k}, \bar{R}_i] =$ 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

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 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 III-A2 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 Section IV-D for more details. Another important practical optimization for CAQR is pipelining the trailing matrix updates [19], though we do not consider this idea here as it cannot be applied with the Householder reconstruction approach.

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 IV), 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 [20, Figure 4]) and the structure of the algorithm is very similar to TSQR.

Algorithm 3 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 3 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.

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.

C. 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 III-A, 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,

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

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
1: **if** $i = 0$ **then**
2: $\quad \bar{Q}_0 = I_b$
3: **end if**
4: **for** $k = \lceil \log p \rceil$ down to 1 **do**
5: $\quad$ **if** $i \equiv 0 \mod 2^k$ and $i + 2^{k-1} < p$ **then**
6: $\quad\quad j = i + 2^{k-1}$
7: $\quad\quad \begin{bmatrix} \bar{Q}_i \\ \bar{Q}_j \end{bmatrix} = \text{Apply-Householder-}Q\left(Y_{i,k}, \begin{bmatrix} \bar{Q}_i \\ 0 \end{bmatrix}\right)$
8: $\quad\quad$ Send $\bar{Q}_j$ to processor j
9: $\quad$ **else if** $i \equiv 2^{k-1} \mod 2^k$ **then**
10: $\quad\quad$ Receive $\bar{Q}_i$ from processor $i - 2^{k-1}$
11: $\quad$ **end if**
12: **end for**
13: $Q_i = \text{Apply-}Q\text{-to-Triangle}\left(Y_{i,0}, \begin{bmatrix} \bar{Q}_i \\ 0 \end{bmatrix}\right)$
Ensure: Q is orthonormal $m \times b$ matrix distributed in block row layout; Q_i is processor i's block

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 [21], there are many so-called "basis-kernel" representations of an orthogonal matrix. For example, the Householder-QR algorithm computes a lower triangular matrix Y such that $A = (I - YTY_1^T)R$, so that

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

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

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

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

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

Yamamoto's approach is very closely related to TSQR-HR (Algorithm 5), presented in Section IV. We compare the methods in Section IV-A.

IV. 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.

A. 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 V.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 4, 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 V.2.

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

	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 I: 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.

Parallelizing this algorithm is straightforward. Since $m \geq b$ and no pivoting is required, the Modified-LU algorithm can be applied on one processor to the top $b \times b$ block. The rest of the lower triangular factor is updated with a triangular solve involving the upper triangular factor. After the upper triangular factor has been broadcast to all processors, the triangular solve is performed in parallel. Thus, the cost of Algorithm 4 is given by

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

Algorithm 4 $[L, U, S] = \text{Modified-LU}(Q)$

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.

The method, given as Algorithm 5, 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 5 $[Y, T, R] = \text{TSQR-HR}(A)$

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

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

which is roughly twice the cost of TSQR plus the cost of Modified-LU.

Note that the LU factorization required in Yamamoto's approach (see Section III-C) is equivalent to performing Modified-LU$(-Q_1)$. In Algorithm 5, 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 5.

While Yamamoto's approach avoids performing the triangular solve on Q_2, it still involves performing both TSQR and Construct-TSQR-Q. Avoiding the triangular solve saves 20% of the arithmetic of the panel factorization with Householder reconstruction, though we found in our performance experiments that the triangular solve accounts for only about 10% of the running time (mostly due to the broadcast of the triangular factor).

The main advantages of TSQR-HR over Yamamoto's algorithm are that the storage of the basis-kernel representation is more compact (since Y is unit lower triangular and T is upper triangular) and that this basis-kernel representation is backward-compatible with (Sca)LAPACK and other libraries using the compact WY representation [10], offering greater performance portability.

B. 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 update can be performed in exactly the same way. Thus, the

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

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} across proc. row $\Pi(r, :)$
6: $\Pi(r,c)$ computes $\tilde{W}_{rc} = Y_{ri}^T \cdot A_{rc}$
7: Allreduce $W_c = \sum_r \tilde{W}_{rc}$ along proc. column $\Pi(:, c)$
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: $A = \left(\prod_{i=1}^{n} (I - Y_{:,i} T_i Y_{:,i}^T)\right) R$

	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	$\frac{2mn^2 - 2n^3/3}{p}$	$\frac{2mn + n^2/2}{\sqrt{p}}$	$6\sqrt{p} \log^2 p$
Scatter-Apply CAQR	$\frac{2mn^2 - 2n^3/3}{p}$	$\frac{2mn + n^2/2}{\sqrt{p}}$	$7\sqrt{p} \log^2 p$

Table II: 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.

only difference between 2D-Householder-QR and CAQR-HR, presented in Algorithm 6, is the subroutine call for the panel factorization (line 2).

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

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

See [18] for full derivation of these costs. 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 II.

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.

C. Two-Level Aggregation

The Householder-QR algorithm attains an efficient trailing matrix update by aggregating Householder vectors into panels (the compact-WY representation). Further, it is straightforward to combine aggregated sets (panels) of Householder vectors into a larger aggregated form. While it is possible to aggregate any basis-kernel representation in this way [21, Corollary 2.8], the Householder form allows for maintaining trapezoidal structure of the basis and triangular structure of the kernel (note that Yamamoto's representation would yield a block-trapezoidal basis and block-triangular kernel). We will refer to the aggregation of sets of Householder vectors as two-level aggregation.

Given the Householder vector reconstruction technique, two-level aggregation makes it possible to decouple the block sizes used for the trailing matrix update from the width of each TSQR. Adding the second blocking parameter to achieve two-level aggregation is a simple algorithmic optimization to our 2D algorithm, and does not change the leading order interprocessor communication costs. This two-level algorithm is given in full in the technical report [18]; it calls Algorithm 6 recursively on large panels of the matrix and then performs an aggregated update on the trailing matrix. While this algorithm does not lower the interprocessor communication, it lowers the local memory-bandwidth cost associated with reading the trailing matrix from memory. This two-level aggregation is analogous to the two-level blocking technique employed in [24] for LU factorization, albeit only on a 2D grid of processors. We also implemented a 2D Householder algorithm with two-level aggregation, which employs ScaLAPACK to factor each thin panel. We refer to this algorithm as Two-Level 2D Householder. We note that ScaLAPACK could be easily modified to use this algorithm with the addition of a second algorithmic blocking factor. Both of our two-level algorithms obtained a significant performance improvement over their single-level aggregated counterparts.

D. Scatter-Apply CAQR

We also found an alternative method for improving the CAQR trailing matrix update that does not reconstruct the Householder form. A major drawback with performing the update via a binary tree algorithm is heavy load imbalance. This problem may be resolved by exploiting the fact that each column of the trailing matrix may be updated independently and subdividing the columns among more processors to balance out the work. This can be done with ideal load balance using a butterfly communication network instead of a binary tree.

Doing the CAQR trailing matrix update via a butterfly network requires storing the implicit representation of the Householder vectors redundantly. We compute the Householder vectors redundantly by doing the TSQR via a butterfly network as done in [25]. Algorithm 7 shows how the trailing

matrix update can be efficiently computed using a butterfly communication network, which effectively performs recursive halving on the columns of the trailing matrix, then recombines the computed updates via an inverse butterfly network (recursive doubling). We call this algorithm Scatter-Apply TSQR-Q^T to emphasize that its structure is analogous to performing a broadcast via a scatter-allgather algorithm, which generalizes recursive halving and doubling and lowers the asymptotic bandwidth cost of a large message broadcast over a simple binary tree broadcast by a factor of $O(\log p)$. Within the context of a 2D QR implementation, Algorithm 7 would be used for each processor column.

Algorithm 7 reduces the bandwidth and computational costs of the trailing matrix update by a factor of $O(\log p)$, since these costs are now dominated by the first level of the butterfly. The leading order costs of a CAQR algorithm which uses Scatter-Apply TSQR-Q^T for the trailing matrix update are

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

We extended Algorithm 7 to a non-power-of-two number of processes via an additional level of the butterfly, which cuts to the nearest power-of-two, though there are alternatives which could be cheaper. An interesting remaining question is whether pipelined CAQR with flat trees, such as the algorithms presented in [19] can yield the same improvement in costs as Algorithm 7.

Algorithm 7 $[B] =$ Scatter-Apply TSQR-Q$^T(\{Y_{i,k}\}, A)$

Require: No. of processors p is a power of 2 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; and $\{Y_{i,k}\}$ 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
for $k = 1$ to $\log p$ **do**
  $j = 2^k \lfloor \frac{i}{2^k} \rfloor + (i + 2^{k-1} \bmod 2^k)$
  Let $\bar{B}_i = [\bar{B}_{i1}, \bar{B}_{i2}]$ where each block is b-by-$n/2^k$
  if $i < j$ **then**
    Swap $\bar{B}_{i2}$ with $\bar{B}_{j1}$ from processor j
    $\begin{bmatrix} \bar{B}_i \\ \bar{B}^k_{j1} \end{bmatrix} =$ Apply-Householder-$Q^T \left(Y_{i,k}, \begin{bmatrix} \bar{B}_{i1} \\ \bar{B}_{j1} \end{bmatrix} \right)$
  else
    Swap $\bar{B}_{j2}$ with $\bar{B}_{i1}$ from processor j
    $\begin{bmatrix} \bar{B}_i \\ \bar{B}^k_{i2} \end{bmatrix} =$ Apply-Householder-$Q^T \left(Y_{i,k}, \begin{bmatrix} \bar{B}_{i2} \\ \bar{B}_{j2} \end{bmatrix} \right)$
  end if
end for
for $k = \log p$ down to 1 **do**
  $j = 2^k \lfloor \frac{i}{2^k} \rfloor + (i + 2^{k-1} \bmod 2^k)$
  if $i < j$ **then**
    Swap $\bar{B}^k_{j1}$ with $\bar{B}_j$ from processor j
    $\bar{B}_i = [\bar{B}_i, \bar{B}_j]$
  else
    Swap $\bar{B}_i$ with $\bar{B}^k_{i1}$ from processor j
    $\bar{B}_i = [\bar{B}^k_{i1}, \bar{B}^k_{i2}]$
  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 $\{Y_{i,k}\}$

V. 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 4). Lemma V.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 V.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

$$Q - \begin{bmatrix} S \\ 0 \end{bmatrix} = Y \cdot (-TY_1^T S). \tag{5}$$

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 [21, Example 2.4].

We obtain Equation (5) 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 V.2 shows that by running the Modified-LU algorithm (Algorithm 4), we can cheaply compute the sign matrix S during the course of the algorithm.

Lemma V.2. *In exact arithmetic, Algorithm 4 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 4) 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 $2 \leq i \leq m$,

$$y_{i1} = \frac{q_{i1}}{q_{11} + \operatorname{sgn}(q_{11})}, \tag{6}$$

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

$$\tilde{q}_{ij} = q_{ij} - \frac{q_{i1}q_{1j}}{q_{11} + \operatorname{sgn}(q_{11})}. \tag{7}$$

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 (6) 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 $(1 + \alpha \operatorname{sgn}(\alpha)) \cdot (1 - \alpha/(\alpha + \operatorname{sgn}(\alpha))) = 1$ implies $\tau y^T q_j = q_{1j}$. Then the trailing matrix update ($2 \leq i \leq m$ and $2 \leq j \leq b$) 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 (7) above.

Finally, consider the jth element of first row of the trailing matrix ($j \geq 2$): $\tilde{q}_{1j} = q_{1j} - y_1(\tau y^T q_j) = 0$. 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. ∎

The following theorem shows that not only is Algorithm 5 correct in exact arithmetic, it is a stable computation in floating point arithmetic. In particular, it proves the normwise backward errors of both the QR factorization and the orthonormality of the computed Q factor are small and that they are independent of the condition number of the input matrix. We use the notation ε to be machine precision.

Theorem V.3. *Let $\hat{R}$ be the computed upper triangular factor of $m \times b$ matrix A obtained via the TSQR algorithm with p processors using a binary tree (assuming $m/p \geq b$), 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 TSQR-HR. Then*

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

and

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

where $F_1, F_2 = O\left(\left(b^{3/2}(m/p) + b^{5/2}\log p + b^3\right)\epsilon\right)$ for $b(m/p)\epsilon \ll 1$.

Proof: We give only a sketch here; see [18] for the full proof. The proof combines the known stability results of TSQR [26] with proofs of the stability of Modified-LU and the rest of the computation. The key idea is that applying Modified-LU to an orthonormal matrix is a very stable computation: not only is the computed lower triangular factor bounded, the upper triangular factor can also be tightly bounded (by a function of the number of columns) using the properties of orthogonality. ∎

VI. 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.

The orthogonality of $\tilde{Q}$ is measured by $\|I - \tilde{Q}^T\tilde{Q}\|_2$. The stability of the factorization is also measured by normwise relative backward errors, $\|A - \tilde{Q}\tilde{R}\|_2/\|A\|_2$. We found similar results for column-wise relative errors.

A. 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 a related cheaper reconstruction method (see [18] for details).

As can be seen from Table III, 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 result demonstrates the numerical stability of this approach on tall and skinny matrices. We note that Yamamoto's approach provides similar results on these test cases.

ρ	κ	$\|A - \tilde{Q}\tilde{R}\|_2$	$\|I - \tilde{Q}^T\tilde{Q}\|_2$
1e-01	5.1e+02	2.2e-15	9.3e-15
1e-03	5.0e+04	2.2e-15	8.4e-15
1e-05	5.1e+06	2.3e-15	8.7e-15
1e-07	5.0e+08	2.4e-15	1.1e-14
1e-09	5.0e+10	2.3e-15	9.9e-15
1e-11	4.9e+12	2.5e-15	1.0e-14
1e-13	5.0e+14	2.2e-15	8.8e-15
1e-15	5.0e+15	2.4e-15	9.7e-15

Table III: Error of TSQR-HR on tall and skinny matrices ($m = 1000, b = 200$)

Matrix type	κ	$\|A - \tilde{Q}\tilde{R}\|_2$	$\|I - \tilde{Q}^T\tilde{Q}\|_2$
$A = 2 * rand(m) - 1$	$2.1e{+}03$	4.3e-15 (256)	2.8e-14 (2)
Golub-Klema-Stewart	$2.2e{+}20$	0.0e+00 (2)	0.0e+00 (2)
Break 1 distribution	$1.0e{+}09$	1.0e-14 (256)	2.8e-14 (2)
Break 9 distribution	$1.0e{+}09$	9.9e-15 (256)	2.9e-14 (2)
$U\Sigma V^T$ with exponential distribution	$4.2e{+}19$	2.0e-15 (256)	2.8e-14 (2)
The devil's stairs matrix	$2.3e{+}19$	2.4e-15 (256)	2.8e-14 (2)
KAHAN matrix, a trapezoidal matrix	$5.6e{+}56$	0.0e+00 (2)	0.0e+00 (2)
Matrix ARC130 from Matrix Market	$6.0e{+}10$	8.8e-19 (16)	2.1e-15 (2)
Matrix FS_541_1 from Matrix Market	$4.5e{+}03$	5.8e-16 (64)	1.8e-15 (256)
BAART: discretization of Fredholm integral equation of the first kind	$5.2e{+}18$	1.6e-15 (32)	2.9e-14 (2)
DERIV2: second derivative	$1.2e{+}06$	2.8e-15 (256)	4.6e-14 (2)
FOXGOOD: severely ill-posed problem	$5.7e{+}20$	2.4e-15 (256)	2.8e-14 (2)

Table IV: Errors of CAQR-HR on square matrices ($m = 1000$). The numbers in parentheses give the panel width yielding largest error.

B. 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 [27], 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 IV 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 V.3. Again, Yamamoto's approach displays similar results. Altogether, these two sets of experiments demonstrate the numerical stability of the proposed approach on representative matrices.

VII. 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 Two-Level CAQR-HR outperforms our implementation of CAQR, and library implementations of 2D-Householder-QR.

A. 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.

B. Parallel Scalability

In this section, we give performance results based on our C++/MPI/LAPACK implementations of TSQR, TSQR-HR, Two-Level CAQR-HR, CAQR, Scatter-Apply CAQR, and Two-Level 2D Householder, as well as two library implementations of 1D Householder-QR and 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.

First, we study the performance of QR factorization of tall-skinny matrices using a 1D processor grid. Figure 1 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 that TSQR-HR takes roughly twice the execution time of TSQR, which is in line with our theoretical cost analysis. Figure 1 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 and remains competitive with Householder-QR.

We collected these results by taking the best observed time over a few runs including ones where a subset of the nodes in the scheduled partition was used. We note that ScaLAPACK performance was highly variable and benefited significantly from using only a fraction of the partition. For instance, on 768 nodes the best ScaLAPACK observed performance was 3.4 ms for this problem size when using half of a 1536 node partition, but over 7 ms when using a 768 node partition. This variability could be justified by the hypothesis that using a subset of a partition on Hopper yields better locality on the network, which alleviates the latency bottleneck of the Householder-QR algorithm. This claim is supported by the fact that the performance variability of algorithms employing TSQR was smaller and much less benefit was yielded from these algorithms being executed on a subset of a partition.

Second, we study the parallel scaling of QR factorization on square matrices. In Figure 2, we compare our implemen-

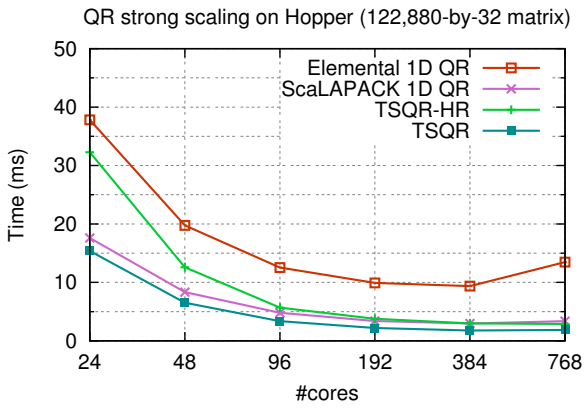

Figure 1: Tall-skinny QR performance on Cray XE6

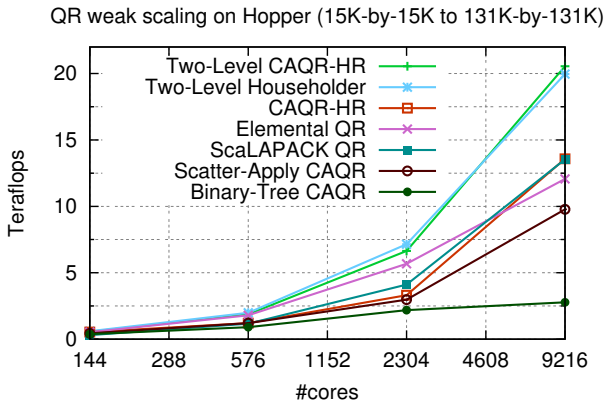

Figure 2: Square QR performance on Cray XE6

tation of CAQR with a binary tree update (no pipelining or other optimizations), Scatter-Apply CAQR, CAQR-HR, and Two-Level CAQR-HR, with Elemental and ScaLAPACK, which use 2D-Householder-QR, as well as Two-Level 2D Householder. We tuned the block sizes of all the codes (the Two-Level CAQR-HR 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 Two-Level CAQR-HR and CAQR-HR in Figure 2, we observe that significant benefit is obtained from aggregating the trailing matrix update. Similarly, we note that two-level aggregation of the 2D Householder algorithm yields a similar performance improvement (compare ScaLAPACK with Two-Level 2D Householder). On the other hand, the binary-tree CAQR performance is relatively poor due to the overhead of the implicit tree trailing update. This overhead is significantly alleviated by the Scatter-Apply TSQR-Q^T algorithm for the trailing matrix update, though the Scatter-Apply CAQR is still slower than algorithms which perform the trailing matrix update using the Householder form.

Overall, these results suggest that latency cost is not a significant overhead on this platform, though as explained in the performance analysis of the 1D algorithms, heavy latency cost contributes to performance variability. Further, other architectures such as cloud and grid environments typically have higher latency payloads. We also expect that the relative cost of messaging latency will grow in future architectures and larger scales of parallelism as the topological distance between computing elements grows. Lastly, we 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.

VIII. 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. We argue both theoretically and empirically that the method can be used to combine two existing approaches, Householder-QR and CAQR, to produce a more communication-efficient and numerically stable algorithm.

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 [28], except in a more practical manner, following the communication-optimal parallel algorithm for LU [24].

IX. 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. Ballard was supported in part by an appointment to the Sandia National Laboratories Truman Fellowship in National Security Science and Engineering, sponsored by Sandia Corporation (a wholly owned subsidiary of Lockheed Martin Corporation) as Operator of Sandia National Laboratories under its U.S. Department of Energy Contract No. DE-AC04-94AL85000. 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.

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. Langou, "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] J. Dongarra, M. Faverge, T. HéRault, M. Jacquelin, J. Langou, and Y. Robert, "Hierarchical QR factorization algorithms for multi-core clusters," *Parallel Computing*, vol. 39, no. 4-5, pp. 212–232, Apr. 2013.

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

[21] 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.

[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] 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.

[25] M. Hoemmen, "Communication-avoiding Krylov subspace methods," Ph.D. dissertation, EECS Department, University of California, Berkeley, Apr 2010.

[26] 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.

[27] 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.

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