Low-Precision QR Factorization: Algorithms, Analysis, and Applications

L. Minah Yang, Alyson Fox, and Geoffrey Sanders

October 2018

Contents

* I am wondering ~~if~~ the ordering should change?

E.g. 3.1
3.2.1
3.2.2

Section 2
3.2.3
3.2.4
3.3
3.4 .

1 Introduction

- Higham [???] [1]
- TSQR [???] [2], TSQR Analysis [???]
- GPU Refs [???] TPUs [???]
- spectral clustering [???]

Given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, we consider performing the so-called *QR factorization*, where

$$\mathbf{A} = \mathbf{QR}, \qquad \mathbf{Q} \in \mathbb{R}^{m \times n}, \qquad \mathbf{R} \in \mathbb{R}^{n \times n},$$

and $\mathbf{Q}$ is orthogonal, $\mathbf{Q}^\top \mathbf{Q} = I$, and is upper-triangular, $\mathbf{R}_{ij} = 0$ for $i > j$.

2 Algorithms

2.1 Modern GPU Hardware

2.2 Notation

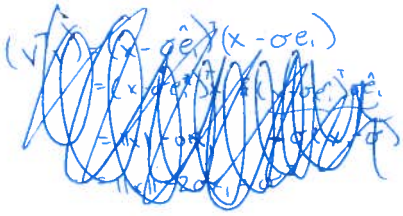
2.3 Householder QR Factorization Algorithm

The Householder QR factorization uses Householder transformations to zero out elements below the diagonal of a matrix. First, we consider the simpler task of zeroing out all but the first element of a vector, $\mathbf{x} \in \mathbb{R}^m$.

Lemma 2.1. *Given vector $\mathbf{x} \in \mathbb{R}^m$, there exist Householder vector $\mathbf{v}$ and Householder transformation matrix $\mathbf{P}_\mathbf{v}$ such that $\mathbf{P}_\mathbf{v}$ zeroes out $\mathbf{x}$ below the first element.*

$$\sigma = -\operatorname{sign}(x_1)\|\mathbf{x}\|_2, \quad \mathbf{v} = \mathbf{x} - \sigma \hat{e}_1,$$
$$\beta = \frac{2}{\mathbf{v}^\top \mathbf{v}} = -\frac{1}{\sigma v_1}, \quad \mathbf{P}_\mathbf{v} = I - \beta \mathbf{v}\mathbf{v}^\top \tag{1}$$

Notation differs... do the mean different things?



Symbol(s)	Definition(s)	Suggestion
$\mathrm{fl}(x); \hat{x}$	calculated from fp operations	
$\mathbf{x}/\mathbf{A}$	vectors/matrices	
m/n	num rows/columns in $\mathbf{A}$	
μ	mantissa	
k	num flops	
$\mathbf{x}_i$	i^{th} index of vector $\mathbf{x}$	
s, p, w	sum, product, and storage (write)	
η	exponent bits	
$\hat{e}_i$	cardinal vectors	
i/j	row/column index of a matrix or vector	
u_q	unit round-off for precision $\mathbf{Q}$	
δ_q	defined only by $\lvert\delta_q\rvert < u_q$	
$\gamma_q^{(k)}$	$\frac{ku_q}{1-ku_q}$	
$\theta_q^{(k)}$	defined only by $\lvert\theta_q^{(k)}\rvert \leq \gamma_q^{(k)}$	
$\gamma_{p,q}^{(k_p,k_q)}$	$(1+\gamma_p^{(k_p)})(1+\gamma_q^{(k_q)})-1$	
$\lvert x\rvert$; $\lVert\mathbf{x}\rVert_2$	matrix 2-norm	double bars throughout
$\cdot I_{m\times n}$	$\begin{bmatrix} I_{n\times n} \\ 0_{m-n\times n} \end{bmatrix}$	
$\mathbf{A}[a:b, c:d]$	rows $\mathbf{A}$ to b and columns c to d of matrix $\mathbf{A}$	
$\mathbf{A}[:, c:d]$	columns c to d of matrix $\mathbf{A}$	
$\hat{\mathbf{A}}$	used interchangeably with $\mathrm{fl}(\mathbf{A})$	

Table 1: Notation discrepancies and suggestions. TODO: resolve each row, comment out, and replace for an eventual notation summary table.

The resulting vector has the same 2-norm as $\mathbf{x}$ *since Householder transformations are orthogonal.*

which P_{ux} or v? unclear.

$$\mathbf{P_v x} = \sigma \hat{e_1} \tag{2}$$

Does not flow well. This is float

In addition, $\mathbf{P_v}$ *is symmetric and orthogonal* ($\mathbf{P_v} = \mathbf{P_v^\top} = \mathbf{P_v^{-1}}$)*, and therefore involutary* ($\mathbf{P_v^2} = \mathbf{I}$).

Given $\mathbf{A} \in \mathbb{R}^{m\times n}$ and Lemma 2.1, a Householder QR factorization is done by repeating the following processes. For $i = 1, 2, \cdots, n$,

Step 1) Find and store the Householder constant β_i and vector $\mathbf{v}_i$ that zeros out the i^{th} column beneath the i^{th} element, of $\hat{\mathbf{x}}_i$? unclear.

This is bolded, the others was not.

again, inconsistant notation.

Step 2) Apply the corresponding Householder transformation to the appropriate bottom right partition of the matrix,

Step 3) Move to the next column,

until only an upper triangular matrix remains.

Add to follow without supplemental math to describe steps.

* Might be useful to fill out the notation section for consistancy

Write out what you mean by this - ... i.e. you are applying P_i to A.

Consider the following 4-by-3 matrix example adapted from [1]. Let $\mathbf{P}_i$ represent the i^{th} Householder transformation of this algorithm.

$$A = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{\mathbf{P}_1} \left[\begin{array}{c|cc} \times & \times & \times \\ \hline 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{array}\right] \xrightarrow{\mathbf{P}_2} \left[\begin{array}{cc|c} \times & \times & \times \\ 0 & \times & \times \\ \hline 0 & 0 & \times \\ 0 & 0 & \times \end{array}\right] \xrightarrow{\mathbf{P}_3} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{bmatrix}$$

Since the final matrix $\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1\mathbf{A}$ is upper-triangular, this is the $\mathbf{R}$ factor of the QR decomposition. Set $\mathbf{Q}^\top := \mathbf{P}_3\mathbf{P}_2\mathbf{P}_1$. Then we can formulate $\mathbf{Q}$ via:

$$\mathbf{Q} = (\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1)^\top = \mathbf{P}_1^\top\mathbf{P}_2^\top\mathbf{P}_3^\top = \mathbf{P}_1\mathbf{P}_2\mathbf{P}_3,$$

Q is orthogonal since each P_i is orthogonal

where the last equality results from the symmetric property of $\mathbf{P}_i$'s. In addition, this is orthogonal because $\mathbf{Q}^\top = \mathbf{P}_3\mathbf{P}_2\mathbf{P}_1 = \mathbf{P}_3^\top\mathbf{P}_2^\top\mathbf{P}_1^\top = \mathbf{P}_3^{-1}\mathbf{P}_2^{-1}\mathbf{P}_1^{-1} = (\mathbf{P}_1\mathbf{P}_2\mathbf{P}_3)^{-1} = \mathbf{Q}^{-1}$, where the third equality results from the orthogonal property of $\mathbf{P}_i$'s.

Returning to the general case, we have:

$$\mathbf{Q} = \mathbf{P}_1 \cdots \mathbf{P}_n, \quad \text{and} \quad \mathbf{R} = \mathbf{Q}^\top\mathbf{A} = \mathbf{P}_n \cdots \mathbf{P}_1\mathbf{A}. \tag{3}$$

2.3.1 Implementation

The Householder transformation is implemented by a series of inner and outer products, since Householder matrices are rank-1 updates of the identity. This is much less costly than forming $\mathbf{P}_\mathbf{v}$, then performing matrix-vector or matrix-matrix multiplications. For some $\mathbf{P}_\mathbf{v} = I - \beta\mathbf{v}\mathbf{v}^\top$, we result in the following computation..

$$\mathbf{P}_\mathbf{v}\mathbf{x} = (I - \beta\mathbf{v}\mathbf{v}^\top)\mathbf{x} = \mathbf{x} - (\beta\mathbf{v}^\top\mathbf{x})\mathbf{v} \tag{4}$$

The routine shown in Equation 4 is used in forming $\mathbf{R}$ and $\mathbf{Q}$. Given a vector $\mathbf{x} \in \mathbb{R}^m$, Algorithm 1 calculates the Householder constant β and Householder vector $\mathbf{v}$, that zero out $\mathbf{x}$ below the first element, and also returns σ. Algorithm 2 is the actual Householder QR factorization algorithm where information necessary to build $\mathbf{Q}$ is returned instead of explicitly forming $\mathbf{Q}$. Finally, the $\mathbf{Q}$ factor can be built using Algorithm 3. While this algorithm shows how to left multiply $\mathbf{Q}$ constructed by $\mathbf{V}$ and β to any input matrix $\mathbf{B}$, putting in $\mathbf{B} \equiv I_{m\times n}$ will yield $\mathbf{Q}$ in the thin $\mathbf{QR}$ factorization.

never introduced — unclear — Bold?

2.3.2 Normalization of Householder Vectors

Equation 1 gives a single Householder transformation matrix $\mathbf{P}$ for all v' in Span(v). This allows for many different ways of normalizing the Householder vectors as well as the choice of not normalizing them. Some methods and reasons for normalization are as follows:

no vars, new notation? — notation? — why would (1) allow this — be more concrete.

- Set $\mathbf{v}_1$ to 1 for efficient storage of many Householder vectors.
- Set the 2-norm of $\mathbf{v}$ to $\sqrt{2}$ to always have $\beta = 1$.
- Set the 2-norm of $\mathbf{v}$ to 1 to prevent extremely large values, and to always have $\beta = 2$.

Algorithm 1: $\beta, \mathbf{v}, \sigma = \text{hh\_vec}(\mathbf{x})$. Given a vector $\mathbf{x} \in \mathbb{R}^n$, return the Householder vector $\mathbf{v}$, a Householder constant β, and σ such that $(I - \beta \mathbf{v}\mathbf{v}^\top)\mathbf{x} = \sigma(\hat{e_1})$, and $\mathbf{v}_1 = 1$.

Input: $\mathbf{x} \in \mathbb{R}^m$

Output: $\mathbf{v} \in \mathbb{R}^m$, and $\sigma, \beta \in \mathbb{R}$ such that $(I - \beta \mathbf{v}\mathbf{v}^\top)\mathbf{x} = \pm\|\mathbf{x}\|_2 \hat{e_1} = \sigma \hat{e_1}$

/* We choose the sign of sigma to avoid cancellation of $\mathbf{x}_1$ (As is the standard in LAPACK, LINPACK packages [1]). This makes $\beta > 0$. */

$\mathbf{v} \leftarrow \mathbf{x}$

$\sigma \leftarrow -\text{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2$

$\mathbf{v}_1 \leftarrow \mathbf{x}_1 - \sigma$ // This is referred to as $\tilde{\mathbf{v}}_1$ later on.

$\beta \leftarrow -\frac{\mathbf{v}_1}{\sigma}$

$\mathbf{v} \leftarrow \frac{1}{\mathbf{v}_1}\mathbf{v}$

return $\beta, \mathbf{v}, \sigma$

Algorithm 2: $\mathbf{V}, \boldsymbol{\beta}, \mathbf{R} = \text{hh\_QR}(A)$. Given a matrix $A \in \mathbb{R}^{m \times n}$ where $m \geq n$, return matrix $\mathbf{V} \in \mathbb{R}^{m \times n}$, vector $\boldsymbol{\beta} \in \mathbb{R}^n$, and upper triangular matrix $\mathbf{R}$. An orthogonal matrix $\mathbf{Q}$ can be generated from $\mathbf{V}$ and $\boldsymbol{\beta}$, and $\mathbf{QR} = \mathbf{A}$.

Input: $A \in \mathbb{R}^{m \times n}$ where $m \geq n$.

Output: $\mathbf{V}$,$\boldsymbol{\beta}$, $\mathbf{R}$

$\mathbf{V}, \boldsymbol{\beta} \leftarrow 0_{m \times n}, 0_m$

for $i = 1 : n$ **do**

  $\mathbf{v}, \beta, \sigma \leftarrow \text{hh\_vec}(\mathbf{A}[i : \text{end}, i])$

  $\mathbf{V}[i : \text{end}, i], \boldsymbol{\beta}_i, \mathbf{A}[i, i] \leftarrow \mathbf{v}, \beta, \sigma$ // Stores the Householder vectors and constants.

  /* The next two steps update A. */

  $\mathbf{A}[i+1 : \text{end}, i] \leftarrow \text{zeros}(m - i)$

  $\mathbf{A}[i : \text{end}, i+1 : \text{end}] \leftarrow \mathbf{A}[i : \text{end}, i+1 : \text{end}] - \beta \mathbf{v}\mathbf{v}^\top \mathbf{A}[i : \text{end}, i+1 : \text{end}]$

return $\mathbf{V}$, $\boldsymbol{\beta}$, $\mathbf{A}[1 : n, 1 : n]$

Algorithm 3: $\mathbf{QB} \leftarrow \text{hh\_mult}(V, \mathbf{B})$: Given a set of householder vectors $\{\mathbf{v}_i\}_{i=1}^n$ and their corresponding constants $\{\boldsymbol{\beta}_i\}_{i=1}^n$, compute $\mathbf{P}_1 \cdots \mathbf{P}_n \mathbf{B}$, where $\mathbf{P}_i := I - \beta_i \mathbf{v}_i \mathbf{v}_i^\top$

Input: $\mathbf{V} \in \mathbb{R}^{m \times n}$, $\boldsymbol{\beta} \in \mathbb{R}^n$ where $m \geq n$. $\mathbf{B} \in \mathbb{R}^{m \times d}$.

Output: $\mathbf{QB}$

/* $\mathbf{v}_i = V[i : m, i] \in \mathbb{R}^{m-(i-1)}$ and $\mathbf{B}_i = \mathbf{B}[i : \text{end}, i : \text{end}] \in \mathbb{R}^{(m-(i-1)) \times (d-(i-1))}$. */

for $i = 1 : n$ **do**

  $\mathbf{B}_i \leftarrow \mathbf{B}_i - \beta_i \mathbf{v}_i (\mathbf{v}_i^\top \mathbf{B}_i)$

return $\mathbf{B}$

The first normalizing method adds an extra rounding error to β and $\mathbf{v}$ each, whereas the remaining methods incur no rounding error in forming β— $d1$ and 2 can be represented exactly.

The LINPACK implementation of the Householder QR factorization uses CHECK! the first method of normalizing via setting $\mathbf{v}_1$ to 1, and is shown in Algorithm 1.

The normalization of Householder vectors has weak influence over the stability of Householder QR algorithm performed in higher precision floating numbers such as single and double-precision floats. However, lower precision floating point numbers with limited dynamic range may be more sensitive to the un/normalization choice. For example, if we leave the Householder vectors unnormalized while using half-precision, it is possible to accumulate `Inf`'s in inner products of "large" vectors.

2.4 Tall-and-Skinny QR

Of the many blocked QR factorization methods, we chose Tall-and-Skinny QR (TSQR), otherwise known as the AllReduce algorithm [2]. This is the most parallel version of within the class of block QR factorizations discussed in [3]. A detailed description of this algorithm can be found in [2], and we present a pseudocode for the algorithm here.

Algorithm 4 will find a QR factorization of a matrix $A \in \mathbb{R}^{m\times n}$ where $m >> n$.

The inlined function qr returns $\mathbf{V} \in \mathbb{R}^{m\times n}$ and $\mathbf{R} \in \mathbb{R}^{n\times n}$. The columns of $\mathbf{V}$ are the Householder vectors (normalized to 1) that can form the matrix $\mathbf{Q}_{\text{thin}} = \mathbf{P}_1 \cdots \mathbf{P}_n I_{m\times n}$. Note that a full $\mathbf{Q}$ can be constructed via $\mathbf{Q}_{\text{full}} = \mathbf{P}_1 \cdots \mathbf{P}_n$.

Algorithm 3 is the implementation of multiplying $\mathbf{Q} := \mathbf{P}_1 \cdots \mathbf{P}_n$ to another matrix or vector, when only the householder vectors to construct $\mathbf{P}_i$'s are given. This takes advantage of the special property of householder matrices $\mathbf{P}_i$'s are rank-one updates of the identity. Let $\mathbf{B} \in \mathbb{R}^{m\times d}$. The straightforward method of computing $\mathbf{QB}$ costs $\mathcal{O}(m^2 d)$ where the costs of constructing $\mathbf{Q}$ itself is ignored. However, Algorithm 3 describes a method that is only $\mathcal{O}(mnd)$.

2.4.1 TSQR Notation

1. For $j = 1, \cdots \in \mathbb{N}$, define the following:
   - $\alpha(j) = \lceil \frac{j}{2} \rceil$
   - $\beta(j) = 2 + j - 2\alpha(j)$
   - or $j = 2(\alpha(j) - 1) + \beta(j)$
2. We write $\mathbf{Q}_j^{(i)} =: \begin{bmatrix} \tilde{\mathbf{Q}}_{j,1}^{(i)} \\ \tilde{\mathbf{Q}}_{j,2}^{(i)} \end{bmatrix}$, where $\tilde{\mathbf{Q}}_{j,k}^{(i)} \in \mathbb{R}^{n\times n}$ for $i = 1 : L$, and $\tilde{\mathbf{Q}}_{j,k}^{(0)} \in \mathbb{R}^{\tilde{h}\times n}$ where $\tilde{h} \in \{h, r\}$.
   For more details on this part of the algorithm, look at section 2.4.2.

2.4.2 Single-level Example

In the single-level version of this algorithm, we first bisect $\mathbf{A}$ into $\mathbf{A}_1^{(0)}$ and $\mathbf{A}_2^{(0)}$ and compute the QR factorization of each of those submatrices. We combine the resulting upper-triangular matrices ($\mathbf{R}_1^{(0)}$ and $\mathbf{R}_2^{(0)}$) into $\mathbf{A}_1^{(1)}$, which we QR factorize next.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1^{(0)} \\ \mathbf{A}_2^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)}\mathbf{R}_1^{(0)} \\ \mathbf{Q}_2^{(0)}\mathbf{R}_2^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & 0 \\ 0 & \mathbf{Q}_2^{(0)} \end{bmatrix} \begin{bmatrix} \mathbf{R}_1^{(0)} \\ \mathbf{R}_2^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & 0 \\ 0 & \mathbf{Q}_2^{(0)} \end{bmatrix} \mathbf{A}_1^{(1)} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & 0 \\ 0 & \mathbf{Q}_2^{(0)} \end{bmatrix} \mathbf{Q}_1^{(1)}\mathbf{R}_1^{(1)}.$$

Whereas $\mathbf{R}_1^{(1)}$ is the final $\mathbf{R}$ factor of the QR factorization of the original matrix, $\mathbf{A}$, we still need to construct $\mathbf{Q}$. Bisecting $\mathbf{Q}_1^{(1)}$ into two submatrices ($\tilde{\mathbf{Q}}_{1,1}^{(1)}$ and $\tilde{\mathbf{Q}}_{1,1}^{(2)}$) allows us to write and compute the product more compactly.

$$\mathbf{Q} := \begin{bmatrix} \mathbf{Q}_1^{(0)} & 0 \\ 0 & \mathbf{Q}_2^{(0)} \end{bmatrix} \mathbf{Q}_1^{(1)} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & 0 \\ 0 & \mathbf{Q}_2^{(0)} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{Q}}_{1,1}^{(1)} \\ \tilde{\mathbf{Q}}_{1,2}^{(1)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)} \tilde{\mathbf{Q}}_{1,1}^{(1)} \\ \mathbf{Q}_2^{(0)} \tilde{\mathbf{Q}}_{1,2}^{(1)} \end{bmatrix}$$

2.4.3 TSQR/AllReduce Algorithm

Introduction

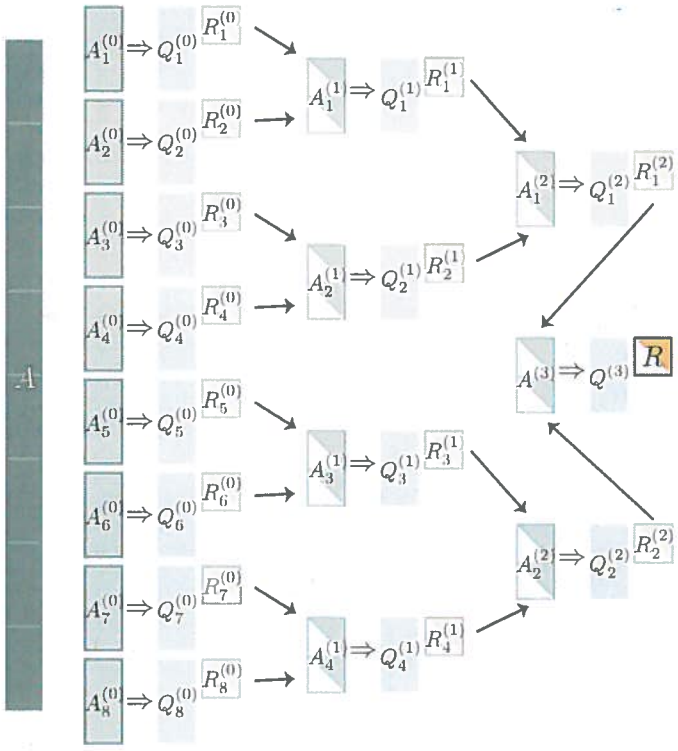


Figure 1: Visualization of the TSQR factorization (AllReduce) algorithm.

Make sure to discuss how the final R & Q in detail

2.4.4 Variants of TSQR

This is just one variation of the TSQR algorithm, which broadly refers to all blocked QR factorization algorithms that treats tall-and-skinny matrices as a single block-column. While the above algorithm is extremely parallelizable [2], there do exist other algorithms that are sequential, or that combine sequential and parallel methods [3]. Include why we are specifically studying this variant.

Algorithm 4: $\mathbf{Q}, \mathbf{R} = tsqr(\mathbf{A})$. Finds the QR factorization of a tall, skinny matrix, $\mathbf{A}$.

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$ where $m \gg n$, $L \in \mathbb{N}$ where 2^L is the initial number of submatrices.
Output: $\mathbf{Q} \in \mathbb{R}^{m \times n}$, $\mathbf{R} \in \mathbb{R}^{n \times n}$ such that $\mathbf{QR} = \mathbf{A}$.

$h \leftarrow \lfloor \frac{m}{2^L} \rfloor$ // Number of rows for all but the last block.

$r \leftarrow m - (2^L - 1)h$ // Number of rows for the last block ($h \leq r < 2h$).

/* Split $\mathbf{A}$ into 2^L blocks. Note that level (i) has 2^{L-i} block */

for $j = 1 : 2^L - 1$ **do**

  $\mathbf{A}_j^{(0)} \leftarrow I_{(j-1)h,jh}^{\top} \mathbf{A}$

$\mathbf{A}_{2^L}^{(0)} \leftarrow I_{(2^L-1)h,m}^{\top} \mathbf{A}$ // Last block may have more rows.

/* Store Householder vectors as columns of matrix $\mathbf{V}_j^{(i)}$, and set up $\mathbf{A}$ for the next level. */

for $i = 0 : L - 1$ **do**

  for $j = 1 : 2^{L-i}$ **do**

    $\mathbf{V}_{2j-1}^{(i)}, \mathbf{R}_{2j-1}^{(i)} \leftarrow \mathrm{qr}(\mathbf{A}_{2j-1}^{(i)})$

    $\mathbf{V}_{2j}^{(i)}, \mathbf{R}_{2j}^{(i)} \leftarrow \mathrm{qr}(\mathbf{A}_{2j}^{(i)})$ // $\mathbf{V}_j^{(i)} \in \mathbb{R}^{2h \times n}$ for $i > 0$, and $\mathbf{R}_j^{(i)} \in \mathbb{R}^{n \times n}$ always.

    $\mathbf{A}_j^{(i+1)} \leftarrow \begin{bmatrix} \mathbf{R}_{2j-1}^{(i)} \\ \mathbf{R}_{2j}^{(i)} \end{bmatrix}$

/* At the bottom-most level, get the $\mathbf{R}$ factor. */

$\mathbf{V}_1^{(L)}, \mathbf{R} \leftarrow \mathrm{qr}(\mathbf{A}_1^{(L)})$

$\mathbf{Q}_1^{(L)} \leftarrow \mathrm{hh\_mult}(\mathbf{V}_1^{(L)}, I_{2n \times n})$

/* Combine $\mathbf{Q}$ factors from bottom-up-- look at Notation (4). */

for $i = L - 1 : -1 : 1$ **do**

  for $j = 1 : 2^{L-i}$ **do**

    $\mathbf{Q}_j^{(i)} \leftarrow \mathrm{hh\_mult}\left(\mathbf{V}_j^{(i)}, \begin{bmatrix} \tilde{\mathbf{Q}}_{\alpha(j),\beta(j)}^{(i+1)} \\ O_{n,n} \end{bmatrix}\right)$

/* At the top-most level, construct the $\mathbf{Q}$ factor. */

$\mathbf{Q} \leftarrow []$;

for $j = 1 : 2^L$ **do**

  $\mathbf{Q} \leftarrow \begin{bmatrix} \mathbf{Q} \\ \mathrm{hh\_mult}\left(\mathbf{V}_j^{(0)}, \begin{bmatrix} \tilde{\mathbf{Q}}_{\alpha(j),\beta(j)}^{(1)} \\ O_{\tilde{h},n} \end{bmatrix}\right) \end{bmatrix}$

return Q, R

2.5 Modified Gram-Schmidt QR

2.6 Subspace Iteration

3 Analysis

3.1 Floating Point Numbers and Error Analysis Tools

We will be using floating-point operation error analysis tools developed in [1]. Let $\mathbb{F} \subset \mathbb{R}$ denote the space of some floating point number system with base β, precision t, significand/mantissa μ, and exponent range $\eta_{\text{ran}} := \{\eta_{\min}, \eta_{\min}+1, \cdots, \eta_{\max}\}$. Then every element y in $\mathbb{F}$ can be written as

$$y = \pm\mu \times \beta^{\eta - t}, \tag{5}$$

where μ is any integer in $[0, \beta^t - 1]$, and $\eta \in \eta_{\text{ran}}$. While operations we use on $\mathbb{R}$ cannot be replicated exactly due to the finite cardinality of $\mathbb{F}$, we can still approximate the accuracy of analogous floating point operations using these error analysis tools in [1].

Name	β	t	# of exponent bits	$\eta_{\min}$	$\eta_{\max}$	u
IEEE754 half	2	11	5	-15	16	4.883e-04
IEEE754 single	2	24	8	-127	128	5.960e-08
IEEE754 double	2	53	11	-1023	1024	1.110e-16

Table 2: IEEE754 formats with j exponent bits range from $1 - 2^{j-1}$ to 2^{j-1}.

A short analysis of floating point operations (cf. Theorem 2.2 [1]) shows that the relative error is controlled by the unit round-off, $u := \frac{1}{2}\beta^{1-t}$. Table 2 shows IEEE precision types described by the same parameters as in Equation 5. The true value $(x \text{ op } y)$ lies in $\mathbb{R}$ and it is rounded to the nearest floating point number, $\text{fl}(x \text{ op } y)$, admitting a rounding error. Suppose that a single basic floating-point operation yields a relative error, δ, bounded in the following sense,

$$\text{fl}(x \text{ op } y) = (1+\delta)(x \text{ op } y), \quad |\delta| \le u, \quad \text{op} \in \{+, -, \times, \div\} \tag{6}$$

We use Equation 6 as a building block in accumulating errors from k successive floating point operations in product form. Lemma 3.1 introduces new notations that simplify round-off error analyses.

Lemma 3.1 (Lemma 3.1 [1]). *Let $|\delta_i| < u$ and $\rho_i \in \{-1, +1\}$ for $i = 1, \cdots, k$, and $ku < 1$. Then,*

$$\prod_{i=1}^{k}(1+\delta_i)^{\rho_i} = 1 + \theta^{(k)} \tag{7}$$

where

$$|\theta^{(k)}| \le \frac{ku}{1-ku} =: \gamma^{(k)}. \tag{8}$$

In other words, $\theta^{(k)}$ represents the accumulation of k successive round-off errors (δ's), and it is bounded by $\gamma^{(k)}$. This notation often provides upper bounds for relative error, and requiring $\gamma^{(k)} < 1$ ensures that the error bound is meaningful. While the assumption $ku < \frac{1}{2}$ which implies

$\gamma^{(k)} < 1$, is satisfied by fairly large k in single and double precision types, it is a problem for small k in lower precision types. Table 3 shows the maximum value of k that still guarantees a relative error below 100% ($\gamma^{(k)} < 1$).

precision	u	$\bar{k} = \text{argmax}_k(\gamma^{(k)} \leq 1)$
half	4.883e-04	512
single	5.960e-08	$\approx$ 4.194e06
double	1.110e-16	$\approx$ 2.252e15

Table 3: Upper limits of validity in the $\gamma^{(k)}$ notation.

This reflects on two sources of difficulty: 1) Successive operations in lower precision types grow unstable more quickly, and 2) the upper bound given by $\gamma^{(k)}$ becomes suboptimal faster in low precision. However, error analysis within the framework given by Lemma 3.1 at least allows us to keep the analysis simple. We will use it to study variable-precision block QR factorization methods.

In Lemma 3.2, we present modified versions of relations in Lemma 3.1 in [1]. These relations allow us to easily deal with accumulated errors, and aid in writing clear and simpler error analyses. The modifications support multiple precision types, whereas [1] assumes that the same precision is used in all operations.

We distinguish between the different precision types using subscripts — these types include products (p), sums (s), and storage formats (w).

Lemma 3.2 (Mixed precision version of Lemma 3.3 from [1]). *For any nonnegative integer k and some precision q, let $\theta_q^{(k)}$ denote a quantity bounded according to $|\theta_q^{(k)}| \leq \frac{ku_q}{1-ku_q} =: \gamma_q^{(k)}$. The following relations hold for two precisions s and p, positive integers, j_s, j_p, non-negative integers k_s and k_p, and $c > 0$.*

$$(1+\theta_p^{(k_p)})(1+\theta_p^{(j_p)})(1+\theta_s^{(k_s)})(1+\theta_s^{(j_s)}) = (1+\theta_p^{(k_p+j_p)})(1+\theta_s^{(k_s+j_s)}) \tag{9}$$

$$\frac{(1+\theta_p^{(k_p)})(1+\theta_s^{(k_s)})}{(1+\theta_p^{(j_p)})(1+\theta_s^{(j_s)})} = \begin{cases} (1+\theta_s^{(k_s+j_s)})(1+\theta_p^{(k_p+j_p)}), & j_s \leq k_s, j_p \leq k_p \\ (1+\theta_s^{(k_s+2j_s)})(1+\theta_p^{(k_p+j_p)}), & j_s \leq k_s, j_p > k_p \\ (1+\theta_s^{(k_s+j_s)})(1+\theta_p^{(k_p+2j_p)}), & j_s > k_s, j_p \leq k_p \\ (1+\theta_s^{(k_s+2j_s)})(1+\theta_p^{(k_p+2j_p)}), & j_s > k_s, j_p > k_p \end{cases} \tag{10}$$

Without loss of generality, let $1 \gg u_p \gg u_s > 0$. Let d, a nonnegative integer, and $r \in [0, \lfloor \frac{u_p}{u_s} \rfloor]$ be numbers that satisfy $k_s u_s = d u_p + r u_s$. Alternatively, d can be defined by $d := \lfloor \frac{k_s u_s}{u_p} \rfloor$.

$$\gamma_s^{(k_s)} \gamma_p^{(k_p)} \leq \gamma_p^{(k_p)}, \quad \text{for } k_p u_p \leq \frac{1}{2} \tag{11}$$

$$\gamma_s^{(k_s)} + u_p \leq \gamma_p^{(d+2)} \tag{12}$$

$$\gamma_p^{(k_p)} + u_s \leq \gamma_p^{(k_p+1)} \quad \textit{(A loose bound)} \tag{13}$$

$$\gamma_p^{(k_p)} + \gamma_s^{(k_s)} + \gamma_p^{(k_p)} \gamma_s^{(k_s)} < \gamma_p^{(k_p+d+1)} \tag{14}$$

A proof for Equation 14 is shown in Appendix A.

3.2 Mixed-Precision HQR

We present an error analysis for the Householder QR factorization where all inner products are performed with mixed-precision, and all other calculations are done in the storage precision, w. Within the inner product subroutine, products are done in precision p and summation is done in precision s.

3.2.1 Inner product error

As seen from the previous section, the inner product is a building block of the Householder QR method. More generally, it is used widely in most linear algebra tools. Thus, we will generalize classic round-off error analysis of inner products to multiple precision.

Specifically, we consider performing an inner product with different floating point precision assigned to operations multiplication and addition. This is designed to provide a more accurate rounding error analysis of mixed precision floating point operations in recent GPU technologies such as NVIDIA's TensorCore. Currently, TensorCore computes the inner product of vectors stored in half-precision by employing full precision multiplications and a single-precision accumulator. As the majority of rounding errors from computing inner products occur during summation, this immensely reduces the error in comparison to using only half-precision operations. This increase in accuracy combined with its speedy performance motivates us to: 1) study how to best utilize mixed-precision arithmetic in algorithms, and 2) to develop error analysis for mixed-precision algorithms to better understand them.

Lemma 3.3. *Let w, p, and s each represent floating-point precisions for storage, product, and summation, where the varying precisions are defined by their unit round-off values denoted by u_w, u_p, and u_s. Let $\mathbf{x}, \mathbf{y} \in \mathbb{F}_w^m$ be two arbitrary m-length vectors stored in w precision. If an inner product performs multiplications in precision p, and addition of the products using precision s, then,*

$$\mathrm{fl}(\mathbf{x}^\top \mathbf{y}) = (\mathbf{x} + \Delta\mathbf{x})\mathbf{y} = \mathbf{x}(\mathbf{y} + \Delta\mathbf{y}), \tag{15}$$

where $|\Delta\mathbf{x}| \leq \gamma_{p,s}^{(1,m-1)}|\mathbf{x}|$, $|\Delta\mathbf{y}| \leq \gamma_{p,s}^{(1,m-1)}|\mathbf{y}|$ componentwise, and

$$\gamma_{p,s}^{(1,m-1)} := (1 + u_p)(1 + \gamma_s^{(m-1)}) - 1.$$

If we further assume that this result is then stored in precision w and $u_w = u_p$, then $|\Delta\mathbf{x}| \leq \gamma_w^{(d+2)}|\mathbf{x}|$ and $|\Delta\mathbf{y}| \leq \gamma_w^{(d+2)}|\mathbf{y}|$ where $d := \lfloor \frac{(m-1)u_s}{u_w} \rfloor$.

Lemma 3.4. *Let w and s each represent floating-point precisions for storage and summation, where the unit round-off values for each precision are denoted by u_w and u_s. Futhermore, assume $1 \gg u_w \gg u_s > 0$, and that for any two arbitary numbers x and y in $\mathbb{F}_w$, their product xy is in $\mathbb{F}_s$. Let $\mathbf{x}, \mathbf{y} \in \mathbb{F}_w^n$ be two arbitrary n-length vectors stored in w precision. If an inner product performs multiplications in full precision, and addition of the products using precision s, then,*

$$\mathrm{fl}(\mathbf{x}^\top \mathbf{y}) = (\mathbf{x} + \Delta\mathbf{x})\mathbf{y} = \mathbf{x}(\mathbf{y} + \Delta\mathbf{y}), \tag{16}$$

where $|\Delta x| \leq \gamma_w^{(d+1)}|x|$, $|\Delta y| \leq \gamma_w^{(d+1)}|y|$ componentwise, and $d := \lfloor \frac{(n-1)u_s}{u_w} \rfloor$.

Proofs for Lemmas 3.3 and 3.4 are shown in Appendix A. The analyses for these two lemmas differ only in the type of mixed-precision arithmetic performed within the inner product subroutine.

For the rest of this paper, we will refer to the forward error bound for the inner product as γ_w^{d+z} for $z = 1, 2$ to generalize the analysis for varying assumptions. This simplification allows us to use the same analysis for the remaining steps of the Householder QR algorithm since inner products are the only computation that use mixed-precision arithmetic.

3.2.2 Calculation and normalization of Householder Vector

An efficient algorithm for calculating $\mathbf{v}$ is shown in Algorithm 5.

Algorithm 5: Given a vector $\mathbf{x} \in \mathbb{R}^n$, return a Householder vector $\mathbf{v}$ and a Householder constant β such that $(I - \beta \mathbf{v}\mathbf{v}^\top)\mathbf{x} \in \text{Span}(\hat{e_1})$.

Input: $\mathbf{x} \in \mathbb{R}^m$

Output: $\mathbf{v} \in \mathbb{R}^m$, and $\sigma, \beta \in \mathbb{R}$ such that $(I - \beta \mathbf{v}\mathbf{v}^\top)\mathbf{x} = \pm\|\mathbf{x}\|_2 \hat{e_1} = \sigma \hat{e_1}$

/* We choose the sign of sigma to avoid cancellation of x_1 (As is the standard in LAPACK, LINPACK packages [1]). This makes $\beta > 0$. */

1 $\mathbf{v} \leftarrow \mathbf{x}$

2 $\sigma \leftarrow -\text{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2$

3 $\mathbf{v}_1 \leftarrow \mathbf{v}_1 - \sigma$

4 $\beta \leftarrow -\frac{1}{\sigma \mathbf{v}_1}$

5 **return** β, $\mathbf{v}$

The above algorithm leaves $\mathbf{v}$ unnormalized, but it is often normalized via the various methods and reasons listed below:

- Set $\mathbf{v}_1$ to 1 for efficient storage of many Householder vectors.
- Set the 2-norm of $\mathbf{v}$ to $\sqrt{2}$ to always have $\beta = 1$.
- Set the 2-norm of $\mathbf{v}$ to 1 to prevent extremely large values, and to always have $\beta = 2$.

The first normalizing method adds an extra rounding error to β and $\mathbf{v}$ each, whereas the remaining methods incur no rounding error in forming β since 1 and 2 can be represented exactly. The LINPACK implementation of the Householder QR factorization uses CHECK! the first method of normalizing via setting $\mathbf{v}_1$ to 1. Algorithm 1 shows how this convention could be carried out. The error analysis in the subsequent section assumes that there may exist errors in both β and $\mathbf{v}$ to get the worse-case scenario and to be consistent with the LINPACK implementation.

Error analysis for $\mathbf{v}$. In this section, we show how to bound the error when employing the mixed precision dot product procedure for Algorithm 1. To do so, we start with the 2-norm error and build from there.

Lemma 3.5 (2-norm error). *Let p, and s each represent floating-point precisions for storage, product, and summation, where the varying precisions are defined by their unit round-off values denoted by u_w, u_p, and u_s, and we can assume $1 \gg u_w \gg u_p, u_s$. Let $\mathbf{x} \in \mathbb{F}_w^m$ be an arbitrary n-length vector stored in w precision. If an inner product performs multiplications in precision p, and addition of the products using precision s, then,*

$$\text{fl}(\|\mathbf{x}\|_2) = (1 + \theta_w^{(d+z+1)})\|\mathbf{x}\|_2, \tag{17}$$

where $|\theta_w^{(d+z+1)}| \le \gamma_w^{(d+z+1)}|\mathbf{x}|$ *for* $z \in \{1,2\}$ *and* $d := \lfloor \frac{(m-1)u_s}{u_w} \rfloor$.

There is no error incurred in evaluating the sign of a number or flipping the sign. Therefore, the error bound for computing $\sigma = -\text{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2$ is exactly the same as that for the 2-norm.

$$\text{fl}(\sigma) = \hat{\sigma} = \text{fl}(-\text{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2) = \sigma + \Delta\sigma, \quad |\Delta\sigma| \le \gamma_w^{(d+z+1)}|\sigma| \tag{18}$$

We can now show the error for $\tilde{\mathbf{v}}_1$ and $\mathbf{v}_i$ where $i = 2, \cdots, n$. Here $\tilde{\mathbf{v}}_1$ is still the penultimate value $\mathbf{v}_1$ held ($\tilde{\mathbf{v}}_1 = \mathbf{x}_1 - \sigma$). Then the round-off errors for $\tilde{\mathbf{v}}_1$ and $\mathbf{v}_i$'s are

$$\begin{aligned}
\text{fl}(\mathbf{v}_1) = \hat{\mathbf{v}}_1 &= \tilde{\mathbf{v}}_1 + \Delta\tilde{\mathbf{v}}_1, \\
&= \text{fl}(\mathbf{x}_1 - \hat{\sigma}) = (1+\delta_w)(\sigma + \Delta\sigma) = (1+\theta_w^{(d+z+2)})\tilde{\mathbf{v}}_1 \\
\text{fl}(\mathbf{v}_i) = \hat{\mathbf{v}}_i = \text{fl}(\frac{\mathbf{x}_i}{\hat{\mathbf{v}}_1}) &= (1+\delta_w)\frac{\mathbf{x}_i}{\tilde{\mathbf{v}}_1 + \Delta\tilde{\mathbf{v}}_1} = (1+\theta_w^{(1+2(d+z+2))})\tilde{\mathbf{v}}_i.
\end{aligned}$$

The above equalities are permitted since θ values are allowed to be flexible within the corresponding γ bounds.

Error analysis for β. Now we show the derivation of round-off error for the Householder constant, β.

$$\begin{aligned}
\hat{\beta} = \text{fl}(-\frac{\hat{\mathbf{v}}_1}{\hat{\sigma}}) &= -(1+\delta_w)\frac{\tilde{\mathbf{v}}_1 + \Delta\tilde{\mathbf{v}}_1}{(\sigma + \Delta\sigma)} \\
&\le -(1+\theta_w^{(1)})\frac{(1+\theta_w^{(d+z+2)})\tilde{\mathbf{v}}_1}{(1+\theta_w^{(d+z+1)})\sigma} \\
&\le (1+\theta_w^{(d+z+3+2(d+z+1))})\beta \\
&= (1+\theta_w^{(3d+3z+5)})\beta,
\end{aligned}$$

where $z = 1$ or $z = 2$ depending on which mixed-precision inner product procedure was used.

Comparison to uniform precision analysis. In this paper, uniform precision refers to using the same precision for all floating point operations. We compare the errors for $\hat{\beta}$ and $\hat{\mathbf{v}}$ computed via the mixed-precision inner products to the errors computed while everything was done in half-precision. Without mixed-precision, the errors would be bounded by

$$\tilde{\gamma}^{(k)} := \frac{cku}{1-cku}, \tag{19}$$

and c is a small integer (c.f. Section 19.3 [1]). Let us further assume that the storage precision (u_w) in the mixed-precision analysis is half-precision. In other words, we can let $u \equiv u_w$, and directly compare $\tilde{\gamma_w}^{(m)}$ and $\gamma_w^{(3d+3z+5)}$. The integer d depends on the length of the vector, m and the precisions (u_w and u_s), and likely is a small integer. For example, if storage is done in half-precision, and summation within the inner product is done in single-precision, $d := \lfloor \frac{m-1}{8192} \rfloor$. Since both d and z are usually small integers, the errors for $\hat{\beta}$ and $\hat{\mathbf{v}}$ with mixed-precision arithmetic can be approximated by $\gamma_w^{(3d+3z+5)} \approx \tilde{\gamma_w}^{(d+z+1)}$. This is an improvement from $\tilde{\gamma_w}^{(m)}$ as

$$m \gg \lfloor \frac{m-1}{8192} \rfloor + z + 1.$$

3.2.3 Applying a Single Householder Transformation

Applying a Householder transformation is implemented by a series of inner and outer products, since Householder matrices are rank-1 updates of the identity. This is much less costly than forming $\mathbf{P_v}$, then performing matrix-vector or matrix-matrix multiplications. For some $\mathbf{P_v} = I - \beta\mathbf{v}\mathbf{v}^\top$, we result in the following computation.

$$\mathbf{P_v}\mathbf{x} = (I - \beta\mathbf{v}\mathbf{v}^\top)\mathbf{x} = \mathbf{x} - (\beta\mathbf{v}^\top\mathbf{x})\mathbf{v} \tag{20}$$

Applying $\mathbf{P_v}$ to zero out the target column of a matrix Let $\mathbf{x} \in \mathbb{R}^m$ be the target column we wish to zero out beneath the first element. Recall that we chose a specific $\mathbf{v}$ such that $\mathbf{P_v}\mathbf{x} = \sigma\hat{e}_1$. As a result, the only error lies in the first element, σ, and that is shown in Equation 18. Note that the normalization choice of $\mathbf{v}$ does not impact the Householder transformation matrix ($\mathbf{P_v}$) nor its action on $\mathbf{x}$, $\mathbf{P_v}\mathbf{x}$.

Applying $\mathbf{P_v}$ to the remaining columns of the matrix Now, let $\mathbf{x}$ and $\mathbf{v}$ have no special relationship, as $\mathbf{v}$ was constructed given some preceding column.

Set $\mathbf{w} := \beta\mathbf{v}^\top\mathbf{x}\mathbf{v}$. Note that $\mathbf{x}$ is exact, whereas $\mathbf{v}$ and β were still computed.

$$\begin{aligned}
\text{fl}(\hat{\mathbf{v}}^\top\mathbf{x}) &= (1+\theta_w^{(d+z)})(\mathbf{v}+\Delta\mathbf{v})^\top\mathbf{x} \\
&= (1+\theta_w^{(d+z)})(1+\theta_w^{(1+2(d+z+2))})\mathbf{v}^\top\mathbf{x} \\
&= (1+\theta_w^{(3d+3z+5)})\mathbf{v}^\top\mathbf{x} \\
\hat{\mathbf{w}} &= (1+\theta_w^{(2)})(\beta+\Delta\beta)(1+\theta_w^{(3d+3z+5)})\mathbf{v}^\top\mathbf{x}\mathbf{w} \\
&= (1+\theta_w^{(2)})(1+\theta_w^{(3d+3z+5)})\beta(1+\theta_w^{(3d+3z+5)})\mathbf{v}^\top\mathbf{x}\mathbf{w} \\
&= (1+\theta_w^{(6d+6z+12)})\mathbf{w} \\
\text{fl}(\mathbf{x}-\hat{\mathbf{w}}) &= (1+\delta_w)(1+\theta_w^{6d+6z+12})\mathbf{w} \\
&= (1+\theta_w^{(6d+6z+13)})\mathbf{P_v}\mathbf{x}
\end{aligned}$$

Constructing $\mathbf{Q}$ and both rely on applying Householder transformations in the above two ways: 1) to zero out below the diagonal of a target column, and 2) to update the bottom right submatrix. We now have the tools to formulate the forward error bound on $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$ calculated from the Householder QR factorization.

3.2.4 Householder QR Factorization Analysis

The pseudo-algorithm in Section 2.3 shows each succeeding Householder transformation is applied to a smaller lower right submatrix each time. Consider a thin QR factorization. Then, for $\mathbf{A} \in \mathbb{R}^{m\times n}$ for $m \geq n$, we have $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{R} \in \mathbb{R}^{n\times n}$. Everything beneath the diagonal on is set to zero.

$$\begin{aligned}
\hat{\mathbf{R}}_{ij} &= (1+\theta_w^{(r_{i,j})})\mathbf{R}_{ij} \\
\hat{\mathbf{Q}}_{ij} &= (1+\theta_w^{(q_{i,j})})\mathbf{Q}_{ij}
\end{aligned}$$

$$r_{ij} = \begin{cases} \lfloor \frac{((m-(i-1))u_s}{u_w} \rfloor + z + 1) + \sum_{k=0}^{i-1} \left(6(\lfloor \frac{(m-k)u_s}{u_w} \rfloor + z) + 13 \right), & i = j \\ \sum_{k=0}^{i-1} \left(6(\lfloor \frac{(m-k)u_s}{u_w} \rfloor + z) + 13 \right), & i < j \end{cases}$$

$$q_{ij} = \begin{cases} \sum_{k=1}^{i} \left(6(\lfloor \frac{(m-(k-1))u_s}{u_w} \rfloor + z) + 13 \right), & j \leq i < n \\ \sum_{k=1}^{j} \left(6(\lfloor \frac{(m-(k-1))u_s}{u_w} \rfloor + z) + 13 \right), & i < j < n \\ 13 + 5(\lfloor \frac{(m-(n-1))u_s}{u_w} \rfloor + z) + \sum_{k=1}^{n-1} \left(6(\lfloor \frac{(m-(k-1))u_s}{u_w} \rfloor + z) + 13 \right), & j \leq i = n \end{cases}$$

Where are proofs?

write it with cl

For values of m, n, u_s, and u_w such that $d := \lfloor \frac{mu_s}{u_w} \rfloor = \lfloor \frac{(m-(n-1))u_s}{u_w} \rfloor$, this simplifies. ~~Even~~ When $\lfloor \frac{mu_s}{u_w} \rfloor > \lfloor \frac{(m-(n-1))u_s}{u_w} \rfloor$, the same analysis can be ~~used as~~ form an upper bound,

$$r_{ij} = \begin{cases} (6i+1)d + (6i+1)z + 13i + 1, & i = j \\ i\,(6d + 6z + 13), & i < j \end{cases}$$

and.

$$q_{ij} = \begin{cases} i\,(6d + 6z + 13), & j \leq i < n \\ j\,(6d + 6z + 13), & i < j < n \\ (6i+5)d + (6i+5)z + 13i + 13, & j \leq i = n \end{cases}$$

We can further approximate to get:

$$\hat{\mathbf{R}} = \mathbf{R} + \mathbf{\Delta R} = (1 + \theta_w^{((6n+1)d+(6n+1)z+13n+1)})\mathbf{R}$$
$$\hat{\mathbf{Q}} = \mathbf{Q} + \mathbf{\Delta Q} = (1 + \theta_w^{((6n+5)d+(6n+5)z+13n+13)})\mathbf{Q}$$

← where are proofs

A backward error for $\mathbf{A}$ can be given from this. We use the mixed-precision inner product as a subroutine for this matrix-matrix multiplication.

$$\begin{aligned} \hat{\mathbf{A}} &= \mathrm{fl}(\hat{\mathbf{Q}}\hat{\mathbf{R}}) = \mathbf{A} + \mathbf{\Delta A} \\ &= (1 + \theta_w^{((12n+6)d+(12n+6)z+26n+14)})(1 + \theta_w^{(d+z)})\mathbf{A} \\ &= (1 + \theta_w^{((12n+7)d+(12n+7)z+26n+14)})\mathbf{A} \end{aligned}$$

← I don't see this right away.

implying $\left| \theta_w^{((12n+7)d+(12n+7)z+26n+14)} \right| \leq \tilde{\gamma}_w^{(10n(d+z+1))}$

This is an improvement from $\tilde{\gamma}_w^{(mn)}$, since $m \gg 10(d+z+1)$ in a TSQR setting.

I thought we were only looking at the householder for now?

This is the single precision, this got lost...

Proof. We wish to round up to the lower precision, p, since $1 \gg u_p \gg u_s$.

$$k_p u_p + k_s u_s = (k_p + d)u_p + r u_s \leq (k_p + d + 1)u_p$$

$$\begin{aligned}
\gamma_p^{(k_p)} + \gamma_s^{(k_s)} + \gamma_p^{(k_p)}\gamma_s^{(k_s)} &= \frac{k_p u_p}{1 - k_p u_p} + \frac{k_s u_s}{1 - k_s u_s} + \frac{k_p u_p}{1 - k_p u_p}\frac{k_s u_s}{1 - k_s u_s} \\
&= \frac{k_p u_p + k_s u_s - k_p k_s u_p u_s}{1 - (k_p u_p + k_s u_s) + k_p k_s u_p u_s} \\
&\leq \frac{(k_p + d + 1)u_p - k_p k_s u_p u_s}{1 - (k_p + d + 1)u_p + k_p k_s u_p u_s} \\
&< \frac{(k_p + d + 1)u_p}{1 - (k_p + d + 1)u_p} = \gamma_p^{(k_p + d + 1)}
\end{aligned}$$

□

A.2 Inner Products

A.2.1 Lemma 3.3

Let δ_p and δ_s be rounding error incurred from products and summations, and are bounded by: $|\delta_p| < u_p$ and $|\delta_s| < u_s$ following the notation in [1]. Let s_k denote the k^{th} partial sum, and let $\hat{s_k}$

denote the floating point representation of the calculated s_k.

$$\begin{aligned}
\hat{s_1} &= \text{fl}(\mathbf{x}_1\mathbf{y}_1) = \mathbf{x}_1\mathbf{y}_1(1+\delta_p^{(1)}) \\
\hat{s_2} &= \text{fl}(\hat{s_1} + \mathbf{x}_2\mathbf{y}_2) \\
&= \left[\mathbf{x}_1\mathbf{y}_1(1+\delta_p^{(1)}) + \mathbf{x}_2\mathbf{y}_2(1+\delta_p^{(2)})\right](1+\delta_s^{(1)}) \\
\hat{s_3} &= \text{fl}(\hat{s_2} + \mathbf{x}_3\mathbf{y}_3) \\
&= \left(\left[\mathbf{x}_1\mathbf{y}_1(1+\delta_p^{(1)}) + \mathbf{x}_2\mathbf{y}_2(1+\delta_p^{(2)})\right](1+\delta_s^{(1)}) + \mathbf{x}_3\mathbf{y}_3(1+\delta_p^{(3)})\right)(1+\delta_s^{(2)})
\end{aligned}$$

We can see a pattern emerging. The error for a general length m vector dot product is then:

$$\hat{s_m} = (\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2)(1+\delta_p)(1+\delta_s)^{m-1} + (1+\delta_p)\sum_{i=3}^{n}\mathbf{x}_i\mathbf{y}_i(1+\delta_s)^{m-(i-1)}, \tag{21}$$

where each occurence of δ_p and δ_s are distinct, but still bound by u_p and u_s.

Using Lemma 3.1 and that $\gamma^{(m)}$ is a monotonically increasing function with respect to m (for $mu < 1$), we further simplify.

$$\begin{aligned}
\text{fl}(\mathbf{x}^\top\mathbf{y}) = \hat{s_m} &\leq (1+\theta_p^{(1)})\left[(\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2)(1+\delta_s)^{(m-1)} + \sum_{i=3}^{n}\mathbf{x}_i\mathbf{y}_i(1+\delta_s)^{(m-(i-1))}\right] \\
&\leq (1+\theta_p^{(1)})(1+\theta_s^{(m-1)})\mathbf{x}^\top\mathbf{y} \\
&= (\mathbf{x}+\Delta\mathbf{x})^\top\mathbf{y} = \mathbf{x}^\top(\mathbf{y}+\Delta\mathbf{y})
\end{aligned}$$

Here $\Delta\mathbf{x}$ and $\Delta\mathbf{y}$ are vector perturbations.

By using Lemma 3.2 equation 14, we can bound the perturbations componentwise. Let $d := \lfloor\frac{(m-1)u_s}{u_p}\rfloor$ such that $(m-1)u_s = du_p + ru_s$.

$$\begin{aligned}
|\Delta\mathbf{x}| &\leq \gamma_p^{(d+2)}|\mathbf{x}| \\
|\Delta\mathbf{y}| &\leq \gamma_p^{(d+2)}|\mathbf{y}|
\end{aligned}$$

Furthermore, these bounds lead to a forward error result as shown in Equation 22 .

$$|\mathbf{x}^\top\mathbf{y} - \text{fl}(\mathbf{x}^\top\mathbf{y})| \leq \gamma_p^{(d+2)}|\mathbf{x}|^\top|\mathbf{y}| \tag{22}$$

While this result does not guarantee a high relative accuracy when $|\mathbf{x}^\top\mathbf{y}| \ll |\mathbf{x}|^\top|\mathbf{y}|$, high relative accuracy is expected in some special cases. For example, let $\mathbf{x} = \mathbf{y}$. Then we have exactly $|\mathbf{x}^\top\mathbf{x}| = |\mathbf{x}|^\top|\mathbf{x}| = \|\mathbf{x}\|_2^2$. This leads to

$$\left|\frac{\|\mathbf{x}\|_2^2 - \text{fl}(\|\mathbf{x}\|_2^2)}{\|\mathbf{x}\|_2^2}\right| \leq \gamma_p^{(d+2)} \tag{23}$$

A.2.2 Lemma 3.4

This proof follows similarly to the proof for Lemma 3.3. Since no error is incurred in the multiplication portion of the inner products, δ_s and δ_{st} are rounding error incurred from summations and

storage. As a result, for $i = 1, \cdots, m-1$, $\hat{s_i} \in \mathbb{F}_s$, and $\hat{s_m} \in \mathbb{F}_{st}$, incurring a rounding error into the storage precision.

$$\begin{aligned}
\hat{s_1} &= \text{fl}(\mathbf{x}_1\mathbf{y}_1) = \mathbf{x}_1\mathbf{y}_1 = s_1 \in \mathbb{F}_s \\
\hat{s_2} &= \text{fl}(\hat{s_1} + \mathbf{x}_2\mathbf{y}_2) \\
&= [\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2]\,(1+\delta_s^{(1)}) \\
\hat{s_3} &= \text{fl}(\hat{s_2} + \mathbf{x}_3\mathbf{y}_3) \\
&= \left([\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2]\,(1+\delta_s^{(1)}) + \mathbf{x}_3\mathbf{y}_3\right)(1+\delta_s^{(2)})
\end{aligned}$$

We can see a pattern emerging. The error for a general m-length vector dot product is then:

$$\hat{s_m} = (\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2)(1+\delta_s)^{m-1} + \sum_{i=3}^{n} \mathbf{x}_i\mathbf{y}_i(1+\delta_s)^{m-(i-1)}, \tag{24}$$

where each occurence of $\prod_{i=1}^{k}(1+\delta_{s,i})$ has been simplified to $(1+\delta_s)^k$.

Using Lemma 3.1 and that $\gamma^{(m)}$ is a monotonically increasing function with respect to m (for $mu < 1$), we further simplify.

$$\begin{aligned}
\text{fl}(\mathbf{x}^\top\mathbf{y}) = \hat{s_m} &\leq \left[(\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2)(1+\delta_s)^{(m-1)} + \sum_{i=3}^{n} \mathbf{x}_i\mathbf{y}_i(1+\delta_s)^{(m-(i-1))}\right] \\
&\leq (1+\theta_s^{(m-1)})\mathbf{x}^\top\mathbf{y} \\
&= (\mathbf{x}+\Delta\mathbf{x})^\top\mathbf{y} = \mathbf{x}^\top(\mathbf{y}+\Delta\mathbf{y})
\end{aligned}$$

Here $\Delta\mathbf{x}$ and $\Delta\mathbf{y}$ are vector perturbations.

By using Lemma 3.2 equation 14, we can bound the perturbations componentwise. Let $d := \lfloor \frac{(m-1)u_s}{u_p} \rfloor$ such that $(m-1)u_s = du_p + ru_s$.

$$\begin{aligned}
|\Delta\mathbf{x}| &\leq \gamma_p^{(d+1)}|\mathbf{x}| \\
|\Delta\mathbf{y}| &\leq \gamma_p^{(d+1)}|\mathbf{y}|
\end{aligned}$$

Furthermore, these bounds lead to a forward error result as shown in Equation 25 .

$$|\mathbf{x}^\top\mathbf{y} - \text{fl}(\mathbf{x}^\top\mathbf{y})| \leq \gamma_p^{(d+1)}|\mathbf{x}|^\top|\mathbf{y}| \tag{25}$$

While this result does not guarantee a high relative accuracy when $|\mathbf{x}^\top\mathbf{y}| \ll |\mathbf{x}|^\top|\mathbf{y}|$, high relative accuracy is expected in some special cases. For example, let $\mathbf{x} = \mathbf{y}$. Then we have exactly $|\mathbf{x}^\top\mathbf{x}| = |\mathbf{x}|^\top|\mathbf{x}| = \|\mathbf{x}\|_2^2$. This leads to

$$\left|\frac{\|\mathbf{x}\|_2^2 - \text{fl}(\|\mathbf{x}\|_2^2)}{\|\mathbf{x}\|_2^2}\right| \leq \gamma_p^{(d+1)} \tag{26}$$

In the case that precision st is half-precision and s is single-precision, $d = 0$ as long as $m \leq 8192$.

B GPU details

B.1 Julia Simulation

References

[1] N. J. Higham, *Accuracy and Stability of Numerical Algorithms.* Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2nd ed., 2002.

[2] 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, pp. 111–130, Feb 2012.

[3] 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.