Mixed-Precision analysis of Householder QR Algorithms

L. Minah Yang, Alyson Fox, and Geoffrey Sanders

December 12, 2019

Abstract

Although mixed precision arithmetic has recently garnered interest for training dense neural networks, many other applications could benefit from the speed-ups and lower storage if applied appropriately. The growing interest in employing mixed precision computations motivates the need for rounding error analysis that properly handles behavior from mixed precision arithmetic. We present a framework for mixed precision analysis that builds on the foundations of rounding error analysis presented in [1] and demonstrate its practicality by applying the analysis to various Householder QR Algorithms. In addition, we present successful results from using mixed precision QR factorization for some small-scale benchmark problems in graph clustering.

1 Introduction

The accuracy of a numerical algorithm depends on several factors, including numerical stability and well-conditionedness of the problem, both of which may be sensitive to rounding errors, the difference between exact and finite-precision arithmetic. Low precision floats use fewer bits than high precision floats to represent the real numbers and naturally incur larger rounding errors. Therefore, error attributed to round-off may have a larger influence over the total error when using low precision, and some standard algorithms that are in wide use may no longer be numerically stable when using half precision floating arithmetic and storage. However, many applications exist that would benefit from the use of lower precision arithmetic and storage that are less sensitive to floating-point round off error, such as clustering or ranking graph algorithms [2] or training dense neural networks [3], to name a few.

Many computing applications today require solutions quickly and often under low size, weight, and power constraints (low SWaP), e.g., sensor formation, etc. Computing in low-precision arithmetic offers the ability to solve many problems with improvement in all four parameters. Utilizing mixed-precision, one can achieve similar quality of computation as high-precision and still achieve speed, size, weight, and power constraint improvements. There have been several recent demonstrations of computing using half-precision arithmetic (16 bits) achieving around half an order to an order of magnitude improvement of these categories in comparison to double precision (64 bits). Trivially, the size and weight of memory required for a specific problem is $4\times$. Additionally, there exist demonstrations that the power consumption improvement is similar [4]. Modern accelerators (e.g., GPUs, Knights Landing, or Xeon Phi) are able to achieve this factor or better speedup

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by the LLNL-LDRD Program under Project No. 17-SI-004, LLNL-JRNL-795525-DRAFT.

improvements. Several examples include: (i) 2-4× speedup in solving dense large linear equations [5, 6], (ii) 12× speedup in training dense neural networks, and (iii) 1.2-10× speedup in small batched dense matrix multiplication [7] (up to 26× for batches of tiny matrices). Training deep artificial neural networks by employing lower precision arithmetic to various tasks such as multiplication [8] and storage [9] can easily be implemented on GPUs and are already a common practice in data science applications.

The low precision computing environments that we consider are *mixed precision* settings, which are designed to imitate those of new GPUs that employ multiple precision types for certain tasks. For example, Tesla V100's Tensor Cores perform matrix-multiply-and-accumulate of half precision input data with exact products and single precision (32 bits) summation accumulate [10]. The existing rounding error analyses are built within what we call a *uniform precision* setting, which is the assumption that all arithmetic operations and storage are performed via the same precision. In this work, we develop a framework for deterministic mixed-precision rounding error analysis, and explore half-precision Householder QR factorization (HQR) algorithms for data and graph analysis applications. QR factorization is known to provide a backward stable solution to the linear least squares problem and thus, is ideal for mixed-precision. However, additional analysis is needed as the additional round-off error will effect orthogonality, and thus the accuracy of the solution. Here, we focus on analyzing specific algorithms in a specific set of types (IEEE754 half, single, and double), but the framework we develop could be used on different algorithms or different floating point types (such as fp16 or bfloat [11]).

This work discusses several aspects of using mixed-precision arithmetic: (i) error analysis that can more accurately describe mixed-precision arithmetic than existing analyses, (ii) algorithmic design that is more resistant against lower numerical stability associated with lower precision types, and (iii) an example where mixed-precision implementation performs as sufficiently as double-precision implementations. Our key findings are that the new mixed-precision error analysis produces tighter error bounds, that some block QR algorithms by Demmel et al. [12] are able to operate in low precision more robustly than non-block techniques, and that some small-scale benchmark graph clustering problems can successfully solved with mixed-precision arithmetic.

1.1 Preliminaries

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

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

$\mathbf{Q}$ is orthogonal, $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}_{m\times m}$, and $\mathbf{R}$ is upper-trapezoidal, $\mathbf{R}_{ij} = 0$ for $i > j$. The above formulation is a *full* QR factorization, whereas a more efficient *thin* QR factorization results in $\mathbf{Q} \in \mathbb{R}^{m\times n}$ and $\mathbf{R} \in \mathbb{R}^{n\times n}$, that is

$$\mathbf{A} = \mathbf{QR} = \begin{bmatrix}\mathbf{Q}_1 & \mathbf{Q}_2\end{bmatrix} \begin{bmatrix}\mathbf{R}_1 \\ \mathbf{0}_{m-n\times n}\end{bmatrix} = \mathbf{Q}_1\mathbf{R}_1.$$

Here, $\mathbf{Q}_1\mathbf{R}_1$ is the *thin* QR factorization, where the columns of $\mathbf{Q}_1$ are orthonormal, and $\mathbf{R}_1$ is upper-triangular. In many applications, computing the *thin* decomposition requires less computation and is sufficient in performance. While important definitions are stated explicitly in the text, Table 1 serves to establish basic notation.

In Section 2, we will give an overview of the modern developments in hardware that motivates rounding error analysis that supports multiple precision types, and we will present a set of error

analysis tools. The HQR factorization algorithm and a mixed-precision rounding error analysis of its implementation is discussed in Section 3. In Section 4, we present the TSQR algorithm as well as numerical experiments that show that TSQR can be useful in low precision environments. Section 5 explores the use of low and mixed precision QR algorithms as subroutines for an application: spectral clustering.

Symbol(s)	Definition(s)	Section(s)
$\mathbf{Q}$	Orthogonal factor of matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$: m-by-m (full) or m-by-n (thin)	1
$\mathbf{R}$	Upper triangular or trapezoidal factor of matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$: m-by-n (full) or n-by-n (thin)	1
$\mathbf{A}^{(k)}$	Matrix $\mathbf{A}$ after k Householder transformations.	3.1
$\mathrm{fl}(\mathbf{x})$, $\hat{\mathbf{x}}$	Quantity $\mathbf{x}$ calculated from floating point operations	2
b, t, μ, η	Base/precision/mantissa/exponent bits	2
`Inf`	Values outside the range of representable numbers	3.1.2
k	Number of FLOPs	2
u_q	Unit round-off for precision t and base b: $\frac{1}{2}b^{1-t}$	2
δ_q	Quantity bounded by: $\lvert\delta_q\rvert < u_q$	2
$\gamma_q^{(k)}$, $\theta_q^{(k)}$	$\frac{ku_q}{1-ku_q}$, Quantity bounded by: $\lvert\theta_q^{(k)}\rvert \leq \gamma_q^{(k)}$	2
$\mathbf{x}$, $\mathbf{A}$	Vector, matrix	2
m, n	Number of rows, columns of matrix, or length of vector	1
i, j	Row, column index of matrix or vector	3.1
$\lVert\mathbf{x}\rVert_2$, $\lVert\mathbf{A}\rVert_2$	Vector operator 2-norm	3
$\lvert c\rvert$, $\lvert\mathbf{x}\rvert$,$\lvert\mathbf{A}\rvert$	Absolute value of constant, all elements of vector, matrix	3
$\mathbf{x}_i$, $\hat{e}_i$	i^{th} element of vector $\mathbf{x}$, cardinal vector	3.1, 3
$\mathbf{A}[a:b,:]$, $\mathbf{A}[:,c:d]$	Rows a to b, columns c to d of matrix $\mathbf{A}$	3.1
$\mathbf{0}_{m\times n}$, $\mathbf{I}_n$	m-by-n zero matrix, n-by-n identity matrix	1
$\mathbf{I}_{m\times n}$	$[\mathbf{I}_n \quad \mathbf{0}_{n\times(m-n)}]^\top$	4
$\mathbf{P}_\mathbf{v}$, $\mathbf{P}_i$	Householder transformation define by $\mathbf{v}$, i^{th} Householder transformation in HQR	3.1
u_s, u_p, u_w	Unit round-off for sum, product, and storage (write)	2.2

Table 1: Basic definitions

2 Floating Point Numbers and Error Analysis Tools

2.1 Representation of Real Numbers

We use and analyze the typical IEEE 754 Standard floating point number systems. Let $\mathbb{F} \subset \mathbb{R}$ denote the space of some floating point number system with base $b \in \mathbb{N}$, precision $t \in \mathbb{N}$, significand $\mu \in \mathbb{N}$, and exponent range $[\eta_{\min}, \eta_{\max}] \subset \mathbb{Z}$. Then every element y in $\mathbb{F}$ can be written as

$$y = \pm\mu \times b^{\eta-t}, \tag{1}$$

where μ is any integer in $[0, b^t - 1]$ and η is an integer in $[\eta_{\min}, \eta_{\max}]$. While base, precision, and exponent range are fixed and define a floating point number system, the sign, significand, and

exponent identifies a single number within that system:

Name	b	t	# of exponent bits	$\eta_{\min}$	$\eta_{\max}$	unit round-off 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`

Although 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. We adopt the rounding error analysis tools described in [1], which allow a relatively simple framework for formulating error bounds for complex linear algebra operations. A short analysis of floating point operations (see Theorem 2.2 [1]) shows that the relative error is controlled by the unit round-off, $u := \frac{1}{2}b^{1-t}$.

Let 'op' be any basic operation between 2 floating point numbers from the set $\text{OP} = \{+, -, \times, \div\}$. The true value $(x \text{ op } y)$ lies in $\mathbb{R}$, and it is rounded using some conversion to a floating point number, $\text{fl}(x \text{ op } y)$, admitting a rounding error. The IEEE 754 Standard requires *correct rounding*, which rounds the exact solution $(x \text{ op } y)$ to the closest floating point number and, in case of a tie, to the floating point number that has a mantissa ending in an even number. *Correct rounding* gives us an assumption for the error model where a single basic floating point operation yields a relative error, δ, bounded in the following sense:

$$\text{fl}(x \text{ op } y) = (1+\delta)(x \text{ op } y), \quad |\delta| \leq u, \quad \text{op} \in \{+, -, \times, \div\}. \tag{2}$$

We use Equation 2 as a building block in accumulating errors from successive floating point operations (FLOPs). For example, consider computing $x + y + z$, where $x, y, z \in \mathbb{R}$. Assume that the machine can only compute one operation at a time. We take the convention of computing the left-most operation first. Then there is a rounding error in computing $\hat{s_1} := \text{fl}(x+y) = (1+\delta)(x+y)$, and another rounding error in computing $\hat{s_2} := \text{fl}(\hat{s_1} + z) = (1+\tilde{\delta})(\hat{s_1} + z)$, where $|\delta|, |\tilde{\delta}| < u$. Then,

$$\text{fl}(x + y + z) = (1+\tilde{\delta})(1+\delta)(x+y) + (1+\tilde{\delta})z. \tag{3}$$

Multiple successive operations introduce multiple rounding error terms, and keeping track of all errors is challenging. Lemma 2.1 introduces a convenient and elegant bound that simplifies accumulation of rounding error.

Lemma 2.1 (Lemma 3.1 [1]). *Let $|\delta_i| < u$ and $\rho_i \in \{-1, +1\}$, for $i = 1, \cdots, k$ and $ku < 1$. Then,*

$$\prod_{i=1}^{k}(1+\delta_i)^{\rho_i} = 1 + \theta^{(k)}, \qquad \textit{where} \qquad |\theta^{(k)}| \leq \frac{ku}{1-ku} =: \gamma^{(k)}. \tag{4}$$

In other words, $\theta^{(k)}$ represents the accumulation of rounding errors from k successive operations, and it is bounded by $\gamma^{(k)}$. Allowing $\theta^{(k)}$'s to be any arbitrary value within the corresponding $\gamma^{(k)}$ bounds further aids in keeping a clear, simple error analysis. Applying this lemma to our example of adding three numbers results in

$$\text{fl}(x + y + z) = (1+\tilde{\delta})(1+\delta)(x+y) + (1+\tilde{\delta})z = (1+\theta^{(2)})(x+y) + (1+\theta^{(1)})z. \tag{5}$$

Since $|\theta^{(1)}| \leq \gamma^{(1)} < \gamma^{(2)}$, we can further simplify Equation 5 to

$$\text{fl}(x + y + z) = (1+\tilde{\theta}^{(2)})(x+y+z), \quad \text{where} \quad \tilde{\theta}^{(2)} \leq \gamma^{(2)}. \tag{6}$$

Typically, error bounds formed in the fashion of Equation 6 are converted to relative errors in order to put the error magnitudes in perspective. In our example, for nonzero $(x + y + z)$, we have:

$$\frac{|(x + y + z) - \text{fl}(x + y + z)|}{|x + y + z|} \leq \gamma^{(2)}. \tag{7}$$

Although Lemma 2.1 only requires $ku < 1$, we actually need $ku < \frac{1}{2}$, which implies $\gamma^{(k)} < 1$, in order to maintain a meaningful relative error bound. While this assumption, $\gamma^{(k)} < 1$, is easily satisfied by fairly large k in higher precision floating point numbers, it is a problem even for small k in lower precision floating point numbers. Table 2 shows the maximum value of k that still guarantees a relative error below 100% ($\gamma^{(k)} < 1$). Thus, accumulated rounding errors in lower

precision	$\tilde{k} = \text{argmax}^{(k)}(\gamma^{(k)} \leq 1)$
half	`512`
single	$\approx$ `4.194e06`
double	$\approx$ `2.252e15`

Table 2: Upper limits of meaningful relative error bounds in the $\gamma^{(k)}$ notation.

precision types lead to instability with fewer operations in comparison to higher precision types. As k represents the number of FLOPs, this constraint restricts low-precision floating point operations to smaller problem sizes and lower complexity algorithms.

To clearly illustrate how this situation restricts rounding error analysis in half precision, we now consider performing the dot product of two vectors. A forward error bound for dot products is

$$\frac{|\mathbf{x}^\top \mathbf{y} - \text{fl}(\mathbf{x}^\top \mathbf{y})|}{|\mathbf{x}|^\top |\mathbf{y}|} \leq \gamma^{(m)}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^m, \tag{8}$$

where details and proof for this statement can be found in Section 3.1 of [1]. 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 $|\mathbf{x}^\top \mathbf{x}| = |\mathbf{x}|^\top |\mathbf{x}| = \|\mathbf{x}\|_2^2$, which leads to

$$\left| \frac{\|\mathbf{x}\|_2^2 - \text{fl}(\|\mathbf{x}\|_2^2)}{\|\mathbf{x}\|_2^2} \right| \leq \gamma_p^{(d+2)}. \tag{9}$$

Since vectors of length m accumulate rounding errors that are bounded by $\gamma^{(m)}$, the worst-case relative error bound for a dot product of vectors of length 512 is already at 100% ($\gamma_{\text{half}}^{(512)} = 1$).

We present a simple numerical experiment that shows that the standard deterministic error bound is too pessimistic and cannot be practically used to approximate rounding error for half-precision arithmetic. In this experiment, we generated 2 million random half-precision vectors of length 512 from two random distributions: the standard normal distribution, $N(0, 1)$, and the uniform distribution over $(0, 1)$. Half precision arithmetic was simulated by calling Algorithm 1 for every multiplication and summation step required in calculating the dot product, $\text{fl}(\mathbf{x}^\top \mathbf{y})$.

Algorithm 1: $\mathbf{z}_{\text{half}} = \texttt{simHalf}(f, \mathbf{x}_{\text{half}}, \mathbf{y}_{\text{half}})$ Simulate function $f \in \text{OP} \cup \{\texttt{dot\_product}\}$ in half precision arithmetic given input variables $\mathbf{x}, \mathbf{y}$. Function `castup` converts half precision floats to single precision floats, and `castdown` converts single precision floats to half precision floats by rounding to the nearest half precision float.

Input: $\mathbf{x}_{\text{half}}, \mathbf{y}_{\text{half}} \in \mathbb{F}_{\text{half}}^m$, $f : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^n$
Output: $\text{fl}(f(\mathbf{x}_{\text{half}}, \mathbf{y}_{\text{half}})) \in \mathbb{F}_{\text{half}}^n$
1 $\mathbf{x}_{\text{single}}, \mathbf{y}_{\text{single}} \leftarrow \texttt{castup}([\mathbf{x}_{\text{half}}, \mathbf{y}_{\text{half}}])$
2 $\mathbf{z}_{\text{single}} \leftarrow \text{fl}(f(\mathbf{x}_{\text{single}}, \mathbf{y}_{\text{single}}))$
3 $\mathbf{z}_{\text{half}} \leftarrow \texttt{castdown}(\mathbf{z}_{\text{single}})$
4 **return** $\mathbf{z}_{\text{half}}$

The casting up step is exact since all half precision numbers can be exactly represented in single precision, $\mathbb{F}_{\text{half}} \subset \mathbb{F}_{\text{single}}$; the second step incurs a rounding error from a single precision arithmetic operation; and the casting down step incurs a rounding error from casting down to half precision. Note that using Algorithm 1 for any operation in OP results in simulating half precision arithmetic, whereas using it with the dot product results in simulating mixed precision arithmetic instead. The relative error in this experiment is formulated as the left hand side of the inequality in Equation 8, where all operations outside of calculating $\text{fl}(\mathbf{x}^\top \mathbf{y})$ are executed by casting up to double precision format and using double precision arithmetic. Table 3 shows statistics from computing the relative error for simulated half precision dot products of 512-length random vectors. We see that the inner

Random Distribution	Average	Standard deviation	Maximum
Standard normal	`1.627e-04`	`1.640e-04`	`2.838e-03`
Uniform $(0, 1)$	`2.599e-03`	`1.854e-03`	`1.399e-02`

Table 3: Statistics from dot product backward relative error in for 512-length vectors stored in half-precision and computed in simulated half-precision from 2 million realizations.

products of vectors sampled from the standard normal distribution have backward relative errors that do not deviate much from the unit round-off (`4.883e-4`), whereas the vectors sampled from the uniform distribution tend to accumulate larger errors. Even so, the theoretical upper error bound of 100% is too pessimistic, and it is difficult to predict the kind of results this experiment shows. Recent work in developing probabilistic bounds on rounding errors of floating point operations have shown that the inner product relative backward error for the conditions used for this experiment is bounded by `5.466e-2` with probability 0.99.

Most importantly, no rounding error bounds (deterministic or probabilistic) allow flexibility in the precision types used for different operations. This restriction is the biggest obstacle in gaining an understanding of rounding errors to expect from computations done on emerging hardware that support mixed-precision such as GPUs that employ mixed-precision arithmetic. In this paper, we extend the rounding error analysis framework established in [1] to mixed-precision arithmetic operations.

Lemma 2.2 shows rules from Lemma 3.3 in [1] that summarize how to accumulate errors represented by θ's and γ's.

Lemma 2.2. *For any positive integer k, let $\theta^{(k)}$ denote a quantity bounded according to $|\theta^{(k)}| \leq \frac{ku}{1-ku} =: \gamma^{(k)}$. The following relations hold for positive integers i, j, and nonnegative integer k.*

Arithmetic operations between $\theta^{(k)}$'s:

$$(1+\theta^{(k)})(1+\theta^{(j)}) = (1+\tilde{\theta}^{(k+j)}) \qquad \textit{and} \qquad \frac{1+\theta^{(k)}}{1+\theta^{(j)}} = \begin{cases} 1+\theta^{(k+j)}, & j \le k \\ 1+\theta^{(k+2j)}, & j > k \end{cases} \tag{10}$$

Operations on γ's:

$$\begin{aligned} \gamma^{(k)}\gamma^{(j)} &\le \gamma_{\min(\mathrm{k,j})}, \quad \textit{for } \max_{(\mathrm{j,k})}\mathrm{u} \le \frac{1}{2}, \\ n\gamma^{(k)} &\le \gamma^{(nk)}, \quad \textit{for} \quad n \le \frac{1}{uk}, \\ \gamma^{(k)} + u &\le \gamma^{(k+1)}, \\ \gamma^{(k)} + \gamma^{(j)} + \gamma^{(k)}\gamma^{(j)} &\le \gamma^{(k+j)}. \end{aligned}$$

In Lemma 2.3, we present modified versions of the rules in Lemma 2.2. This mixed-precision error analysis relies on the framework given by Lemma 2.1, which best allows us to keep a simple analysis. These relations allow us to easily accumulate errors in terms of θ's and γ's and aid in writing clear and simpler error analyses. The modifications support multiple precision types, whereas Lemma 2.2 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 2.3. *For any nonnegative integer k and some precision q, let $\theta_q^{(k)}$ denote a quantity bounded according to $|\theta_q^{(k)}| \le \frac{ku_q}{1-ku_q} =: \gamma_q^{(k)}$. 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 $c > 0$:*

$$(1+\theta_p^{(k_p)})(1+\theta_p^{(j_p)})(1+\theta_s^{(k_s)})(1+\theta_s^{(j_s)}) = (1+\theta_p^{(k_p+j_p)})(1+\theta_s^{(k_s+j_s)}), \tag{11}$$

$$\frac{(1+\theta_p^{(k_p)})(1+\theta_s^{(k_s)})}{(1+\theta_p^{(j_p)})(1+\theta_s^{(j_s)})} = \begin{cases} (1+\theta_s^{(k_s+j_s)})(1+\theta_p^{(k_p+j_p)}), & j_s \le k_s, j_p \le k_p, \\ (1+\theta_s^{(k_s+2j_s)})(1+\theta_p^{(k_p+j_p)}), & j_s \le k_s, j_p > k_p, \\ (1+\theta_s^{(k_s+j_s)})(1+\theta_p^{(k_p+2j_p)}), & j_s > k_s, j_p \le k_p, \\ (1+\theta_s^{(k_s+2j_s)})(1+\theta_p^{(k_p+2j_p)}), & j_s > k_s, j_p > k_p. \end{cases} \tag{12}$$

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 $k_s u_s = du_p + ru_s$. Alternatively, d can be defined by $d := \lfloor \frac{k_s u_s}{u_p} \rfloor$. Then

$$\gamma_s^{(k_s)}\gamma_p^{(k_p)} \le \gamma_p^{(k_p)}, \quad \textit{for } k_p u_p \le \frac{1}{2} \tag{13}$$

$$\gamma_s^{(k_s)} + u_p \le \gamma_p^{(d+2)} \tag{14}$$

$$\gamma_p^{(k_p)} + u_s \le \gamma_p^{(k_p+1)} \tag{15}$$

$$\gamma_p^{(k_p)} + \gamma_s^{(k_s)} + \gamma_p^{(k_p)}\gamma_s^{(k_s)} < \gamma_p^{(k_p+d+1)}. \tag{16}$$

A proof for Equation 16 is provided in Appendix A.3.1. We use these principles to establish a mixed-precision rounding error analysis for computing the dot product, which is crucial in many linear algebra routines such as the QR factorization.

2.2 Inner product Mixed-Precision error

We will see in Section 3 that the inner product is a building block of the HQR factorization (HQR) algorithm, which was introduced in [13]. More generally, it is used widely in most linear algebra tools such as matrix-vector multiply and projections. Thus, we will generalize classic round-off error analysis of inner products to algorithms that may employ different precision types to different operations. Specifically, we consider performing an inner product with the storage precision, u_w, being lower than the summation precision, u_s. This choice was made to provide a more accurate rounding error analysis of mixed precision floating point operations present 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 (see Section 3.1, [1]), the single precision accumulator immensely reduces the error in comparison to using only half-precision operations. This increase in accuracy combined with its speedy performance motivates 1)to study how to best utilize mixed-precision arithmetic in algorithms and 2) to develop more accurate error analyses appropriate for mixed-precision algorithms.

Lemma 2.4 and Corollary 2.5 present two mixed-precision forward error bounds for inner products, which show a tighter bound than the existing error bounds. In both cases, we assume storage in the lowest precision with round-off value, u_w, and summation performed with a higher precision with round-off value, u_s, and let $d \approx mu_s/u_w$, where m is the length of the vectors. Although there are additional differing assumptions in these two lemmas, results from both show a strong dependence on d.

Lemma 2.4. *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 vectors stored in w precision. If an inner product performs multiplications in precision p and addition of the products using precision s, then*

$$\mathrm{fl}(\mathbf{x}^\top \mathbf{y}) = (\mathbf{x} + \mathbf{\Delta x})\mathbf{y} = \mathbf{x}(\mathbf{y} + \mathbf{\Delta y}), \tag{17}$$

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

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

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

Corollary 2.5 presents another mixed-precision forward error bound for mixed-precision inner products with additional constraints. Here, we assume that the vectors are being stored in a lower precision than the precision types being used for multiplications and additions. This scenario is similar to how TensorCore technology works in GPUs.

Corollary 2.5. *In addition to the assumptions in Lemma 2.4, assume $1 \gg u_w \gg u_s > 0$, and thus for any two numbers x, y in $\mathbb{F}_w$, their product xy is in $\mathbb{F}_s$. Let $\mathbf{x}, \mathbf{y} \in \mathbb{F}_w^m$ be two arbitrary vectors stored in w precision. If an inner product performs multiplications in full precision and addition of the products using precision s, then*

$$\mathrm{fl}(\mathbf{x}^\top \mathbf{y}) = (\mathbf{x} + \mathbf{\Delta x})\mathbf{y} = \mathbf{x}(\mathbf{y} + \mathbf{\Delta y}), \tag{18}$$

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

Proofs for Lemma 2.4 and Corollary 2.5 are shown in Appendix A.3.1. The analyses for these two differ only in the type of mixed-precision arithmetic performed within the inner product subroutine, and the difference is revealed to result in either $\gamma_w^{(d+1)}$ or $\gamma_w^{(d+2)}$. 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 HQR algorithm presented in the following sections.

3 Mixed-Precision HQR Factorization

The HQR algorithm uses Householder transformations to zero out elements below the diagonal of a matrix. We present this as zeroing out all but the first element of some vector, $\mathbf{x} \in \mathbb{R}^m$.

Lemma 3.1. *Given vector $\mathbf{x} \in \mathbb{R}^m$, there exist Householder vector, $\mathbf{v}$, and Householder transformation matrix, $\mathbf{P_v}$, such that $\mathbf{P_v}$ zeros out $\mathbf{x}$ below the first element.*

$$\begin{aligned} \sigma &= -\operatorname{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2, \quad \mathbf{v} = \mathbf{x} - \sigma\hat{e_1}, \\ \beta &= \frac{2}{\mathbf{v}^\top\mathbf{v}} = -\frac{1}{\sigma\mathbf{v}_1}, \qquad \mathbf{P_v} = \mathbf{I}_m - \beta\mathbf{v}\mathbf{v}^\top. \end{aligned} \tag{19}$$

The transformed vector, $\mathbf{P_v}\mathbf{x}$, has the same 2-norm as $\mathbf{x}$ since Householder transformations are orthogonal: $\mathbf{P_v}\mathbf{x} = \sigma\hat{\mathbf{e}_1}$. In addition, $\mathbf{P_v}$ is symmetric and orthogonal, $\mathbf{P_v} = \mathbf{P_v}^\top = \mathbf{P_v}^{-1}$, and therefore, $\mathbf{P_v}^2 = \mathbf{I}$.

3.1 HQR Factorization Algorithm

Given $\mathbf{A} \in \mathbb{R}^{m\times n}$ and Lemma 3.1, HQR is done by repeating the following processes until only an upper triangle matrix remains. For $i = 1, 2, \cdots, n$,

Step 1) Compute $\mathbf{v}$ and β that zeros out the i^{th} column of $\mathbf{A}$ beneath a_{ii}, and

Step 2) Apply $\mathbf{P_v}$ to the bottom right partition, $\mathbf{A}[i:m, i:n]$.

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

$$\mathbf{A} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{\text{apply } \mathbf{P}_1 \text{ to } \mathbf{A}^{(0)} := \mathbf{A}} \left[\begin{array}{c|cc} \times & \times & \times \\ \hline 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{array}\right] \xrightarrow{\text{apply } \mathbf{P}_2 \text{ to } (\mathbf{A}^{(1)} := \mathbf{P}_1\mathbf{A})}$$

$$\left[\begin{array}{cc|c} \times & \times & \times \\ 0 & \times & \times \\ \hline 0 & 0 & \times \\ 0 & 0 & \times \end{array}\right] \xrightarrow{\text{apply } \mathbf{P}_3 \text{ to } (\mathbf{A}^{(2)} := \mathbf{P}_2\mathbf{P}_1\mathbf{A})} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{bmatrix}$$

Since the final matrix $\mathbf{P}_3\mathbf{P}_2\mathbf{P}_1\mathbf{A}$ is upper-triangular, this result 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 $\mathbf{Q}$ as

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

where the last equality results from the symmetric property of $\mathbf{P}_i$'s.

Returning to the general case, we have

$$\mathbf{Q}_{\text{full}} = \mathbf{P}_1 \cdots \mathbf{P}_n \quad \text{and} \quad \mathbf{R}_{\text{full}} = \mathbf{Q}^\top \mathbf{A} = \mathbf{P}_n \cdots \mathbf{P}_1 \mathbf{A}, \tag{20}$$

for the orthogonal factor in a full QR factorization, and

$$\mathbf{Q}_{\text{thin}} = \mathbf{P}_1 \cdots \mathbf{P}_n \mathbf{I}_{m \times n} \quad \text{and} \quad \mathbf{R}_{\text{thin}} = \mathbf{I}_{m \times n}^\top \mathbf{Q}^\top \mathbf{A} = \mathbf{I}_{m \times n}^\top \mathbf{P}_n \cdots \mathbf{P}_1 \mathbf{A}. \tag{21}$$

3.1.1 HQR Factorization 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 approach is much less costly than forming $\mathbf{P}_\mathbf{v}$, and then performing matrix-vector or matrix-matrix multiplications. For some $\mathbf{P}_\mathbf{v} = \mathbf{I} - \beta \mathbf{v}\mathbf{v}^\top$, we result in the following computation:

$$\mathbf{P}_\mathbf{v} \mathbf{x} = (\mathbf{I} - \beta \mathbf{v}\mathbf{v}^\top)\mathbf{x} = \mathbf{x} - (\beta \mathbf{v}^\top \mathbf{x})\mathbf{v}. \tag{22}$$

The routine in Equation 22 is used in forming $\mathbf{R}$ and $\mathbf{Q}$. Given a vector $\mathbf{x} \in \mathbb{R}^m$, Algorithm 2 calculates the Householder constant, β, and Householder vector, $\mathbf{v}$, that zero out $\mathbf{x}$ below the first element and also returns σ. Algorithm 7 is the HQR algorithm where information necessary to build $\mathbf{Q}$ is returned instead of explicitly forming $\mathbf{Q}$; the Householder vector and constant at the k^{th} step are stored as the k^{th} column of matrix $\mathbf{V} \in \mathbb{R}^{m \times n}$ and the k^{th} element of vector $\boldsymbol{\beta} \in \mathbb{R}^n$.

Finally, the $\mathbf{Q}$ factor can be built using Algorithm 3. While this algorithm shows how to left multiply $\mathbf{Q}$ to any input matrix $\mathbf{B}$ given $\mathbf{V}$ and $\boldsymbol{\beta}$, putting in $\mathbf{B} \equiv I_{m \times n}$ will yield $\mathbf{Q}_{\text{thin}}$.

Algorithm 2: β, $\mathbf{v}$, σ = `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 $\mathbf{v}_1 = 1$, (see [14, 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 β, $\mathbf{v}$, σ

3.1.2 Normalization of Householder Vectors

Equation 19 gives a single Householder transformation matrix $\mathbf{P}_{\mathbf{v}'}$ for all $\mathbf{v}'$ in Span($\mathbf{v}$), which allows for many different ways of normalizing the Householder vectors as well as the choice of not normalizing them. However, this equivalence ($\mathbf{P}_\mathbf{v} \equiv \mathbf{P}_{\mathbf{v}'}$ for all $\mathbf{v}' \in$ Span($\mathbf{v}$)) is not guaranteed due to rounding errors when using floating point numbers and operations. When using high precision floating point numbers such as double-precision floats, rounding errors that accumulate from the normalization of Householder vectors rarely and barely contribute to the overall stability of

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

$\mathbf{V}, \boldsymbol{\beta} \leftarrow \mathbf{0}_{m\times n}, \mathbf{0}_m$
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: $\mathbf{QB} \leftarrow$ `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 $\mathbf{P}_i := \mathbf{I} - \boldsymbol{\beta}_i\mathbf{v}_i\mathbf{v}_i^\top$

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**
  $\mathbf{B}_i \leftarrow \mathbf{B}_i - \boldsymbol{\beta}_i\mathbf{v}_i(\mathbf{v}_i^\top\mathbf{B}_i)$
return $\mathbf{B}$

the HQR algorithm performed. In contrast, 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. As a result, picking a normalization scheme for $\mathbf{v}$ is important in low-precision calculations. Some methods and reasons for the normalization of $\mathbf{v}$ are as follows:

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

LINPACK and its successor LAPACK are benchmark software libraries for performing numerical linear algebra [14]. The LAPACK implementation of the HQR factorization uses the first method of normalizing via setting $\mathbf{v}_1$ to 1 and is shown in Algorithm 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.

3.2 Rounding Error Analysis

We present an error analysis for the HQR factorization where all inner products are performed with mixed-precision, and all other calculations are done in the storage precision, w.

Assumption 3.2 lays out the generalized mixed-precision inner product we will be using over and over again in the remainder of this paper.

Assumption 3.2. *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, and we can assume $1 \gg u_w \gg u_s$ and $u_p \in (0, u_w]$. Within the inner product subroutine, products are done in precision p, summation is done in precision s, and the result stored in precision w. All operations other than inner products are done in the storage precision, w.*

3.2.1 Error analysis for forming Householder Vector and Constant

Calculating the Householder vector and constant is a major routine for the HQR factorization.

Error analysis for v In this section, we show how to bound the error when employing the mixed precision dot product procedure for Algorithm 2. We begin by extending the inner-product error shown in Lemmas 2.4 and 2.5 to the 2-norm error.

Lemma 3.3 (2-norm round-off error)**.** *Consider a mixed-precision scheme as is outlined in Assumption 3.2. Let $\mathbf{x} \in \mathbb{F}_w^m$ be an arbitrary n-length vector stored in w precision. The forward error bound for computing the 2-norm of $\mathbf{v}$ is*

$$\mathrm{fl}(\|\mathbf{x}\|_2) = (1 + \theta_w^{(d+z+1)})\|\mathbf{x}\|_2, \tag{23}$$

where $|\theta_w^{(d+z+1)}| \leq \gamma_w^{(d+z+1)}|\mathbf{x}|$ 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 $\sigma = -\mathrm{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2$ is exactly the same as that for the 2-norm, i.e.,

$$\mathrm{fl}(\sigma) = \hat{\sigma} = \mathrm{fl}(-\mathrm{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2) = \sigma + \Delta\sigma, \quad |\Delta\sigma| \leq \gamma_{\mathrm{w}}^{(\mathrm{d+z+1})}|\sigma|. \tag{24}$$

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

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

and

$$\mathrm{fl}(\mathbf{v}_i) = \hat{\mathbf{v}_i} = \mathrm{fl}\left(\frac{\mathbf{x}_i}{\hat{\mathbf{v}_1}}\right) = (1 + \delta_w)\frac{\mathbf{x}_i}{\tilde{\mathbf{v}}_1 + \mathbf{\Delta}\tilde{\mathbf{v}}_1} = (1 + \theta_w^{(1+2(d+z+2))})\tilde{\mathbf{v}}_i.$$

The above equalities (as opposed to inequalities) 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:

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

where $z = 1$ or $z = 2$, depending on which mixed-precision inner product procedure was used. These two results are formalized in Lemma 3.4 below.

Lemma 3.4. *Given $\mathbf{x} \in \mathbb{R}^m$, consider the constructions of $\beta \in \mathbb{R}$ and $\mathbf{v} \in \mathbb{R}^m$ such that $\mathbf{P_v x} = \sigma \hat{e_1}$ (see Lemma 3.1) by using Algorithm 2. Then the forward error of forming $\mathbf{v}$ and β with the floating point arithmetic with the mixed-precision scheme outlined in Assumption 3.2 are*

$$\|\hat{\mathbf{v}}\|_2 = (1 + \theta_w^{(1+2(d+z+2))})\|\mathbf{v}\|_2 \qquad \text{and} \qquad \hat{\beta} = (1 + \theta_w^{(3d+3z+5)})\beta,$$

where $z \in \{1, 2\}$ and $d = \lfloor \frac{(m-1)u_s}{u_w} \rfloor$.

3.2.2 Applying a Single Householder Transformation

A Householder transformation is applied through a series of inner and outer products, since Householder matrices are rank-1 updates of the identity.

For some $\mathbf{P_v} = I - \beta \mathbf{v}\mathbf{v}^\top$, we result in $\mathbf{P_v x} = (I - \beta \mathbf{v}\mathbf{v}^\top)\mathbf{x} = \mathbf{x} - (\beta \mathbf{v}^\top \mathbf{x})\mathbf{v}$.

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 $\mathbf{P_v x} = \sigma \hat{e}_1$. As a result, the only error lies in the first element, σ, and that is shown in Equation 24. Note that the normalization choice of $\mathbf{v}$ does not impact the Householder transformation matrix ($\mathbf{P_v}$) nor its action on $\mathbf{x}$, $\mathbf{P_v x}$.

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 with floating point operations. The errors incurred from computing $\mathbf{v}$ and β need to be included in addition to the new rounding errors accumulating from the action of applying $\mathbf{P_v}$ to a column.

We show the error for forming $\text{fl}\left(\hat{\mathbf{v}}^\top \mathbf{x}\right)$ first:

$$\text{fl}\left(\hat{\mathbf{v}}^\top \mathbf{x}\right) = (1 + \theta_w^{(d+z)})(\mathbf{v} + \Delta\mathbf{v})^\top \mathbf{x}.$$

Where $\theta_w^{(d+z)}$ is incurred from the action of a dot product,

$$\begin{aligned} \text{fl}\left(\hat{\mathbf{v}}^\top \mathbf{x}\right) &= (1 + \theta_w^{(d+z)})(1 + \theta_w^{(1+2(d+z+2))})\mathbf{v}^\top \mathbf{x}, \\ &= (1 + \theta_w^{(3d+3z+5)})\mathbf{v}^\top \mathbf{x}. \end{aligned}$$

Now we can form $\text{fl}(\mathbf{w})$,

$$\hat{\mathbf{w}} = (1 + \theta_w^{(2)})(\beta + \Delta\beta)(1 + \theta_w^{(3d+3z+5)})\mathbf{v}^\top \mathbf{x}\mathbf{w}.$$

Here, $\theta_w^{(2)}$ results from multiplying $\hat{\beta}$ and $\mathbf{v}^\top \mathbf{x}$ to $\mathbf{w}$,

$$\begin{aligned} \hat{\mathbf{w}} &= (1 + \theta_w^{(2)})(1 + \theta_w^{(3d+3z+5)})\beta(1 + \theta_w^{(3d+3z+5)})\mathbf{v}^\top \mathbf{x}\mathbf{w}, \\ &= (1 + \theta_w^{(6d+6z+12)})\mathbf{w}. \end{aligned}$$

Finally, we can add in the vector subtraction operation and complete the rounding error analysis of applying a Householder transformation to any vector:

$$\text{fl}(\mathbf{P_v x}) = \text{fl}(\mathbf{x} - \hat{\mathbf{w}}) = (1 + \delta_w)(1 + \theta_w^{(6d+6z+12)})\mathbf{w}, \tag{25}$$

$$= (1 + \theta_w^{(6d+6z+13)})\mathbf{P_v x}, \tag{26}$$

$$= (\mathbf{P_v} + \mathbf{\Delta P_v})\mathbf{x}, \qquad \|\mathbf{\Delta P_v}\|_F \leq \gamma_w^{(6d+6z+13)}. \tag{27}$$

Details behind the matrix norm error bound in Equation 27 are shown in A.3. Constructing both $\mathbf{Q}$ and $\mathbf{R}$ relies 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 HQR factorization.

3.2.3 HQR Factorization Forward Error Analysis

Consider a thin QR factorization where $\mathbf{A} \in \mathbb{R}^{m\times n}$ for $m \geq n$, we have $\mathbf{Q} \in \mathbb{R}^{m\times n}$ and $\mathbf{R} \in \mathbb{R}^{n\times n}$. The pseudo-algorithm in Section 3 shows that each succeeding Householder transformation is applied to a smaller lower right submatrix each time.

Instead of continuing with a componentwise analysis of how accumulated rounding errors are distributed by HQR, we transition into normwise error analyses. To do this, we use the analysis from the preceding section (summarized in Equation 27) to implicitly form the matrix norm error of the Householder transformation matrix, $\mathbf{P_v}$. Then, we use the result of Lemma 3.7 in [1] to get a normwise bound on the perturbation effect of multiple matrix multiplications. This result is summarized in Theorem 3.5, and the proof is detailed extensively in A.3.

Theorem 3.5. *Let $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$ have full rank, n. Let $\hat{\mathbf{Q}} \in \mathbb{R}^{m\times n}$ and $\hat{\mathbf{R}} \in \mathbb{R}^{n\times n}$ be the thin QR factors of $\mathbf{A}$ obtained via the HQR algorithm with a mixed-precision scheme as is outlined in Assumption 3.2. Let $d = \lfloor \frac{(m-1)u_s}{u_w} \rfloor$, and $z = 1$ or $z = 2$. Then we have normwise forward error bounds*

$$\hat{\mathbf{R}} = \mathbf{R} + \mathbf{\Delta R} = \hat{\mathbf{P}}_n \cdots \hat{\mathbf{P}}_1 \mathbf{A}, \tag{28}$$

$$\hat{\mathbf{Q}} = \mathbf{Q} + \mathbf{\Delta Q} = \hat{\mathbf{P}}_1 \cdots \hat{\mathbf{P}}_n \mathbf{I}, \tag{29}$$

where

$$\|\mathbf{\Delta Q}\|_F \leq n^{3/2} \tilde{\gamma}_w^{(6d+6z+13)}, \tag{30}$$

and for column j in $\{1, \cdots, n\}$,

$$\|\mathbf{\Delta R}[:, j]\|_2 \leq j \tilde{\gamma}_w^{(6d+6z+13)} \|\mathbf{A}[:, j]\|_2. \tag{31}$$

We also form a backward error. Let $\mathbf{A} + \mathbf{\Delta A} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$, where $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$ are obtained via Algorithm 7. Then,

$$\|\mathbf{\Delta A}\|_F \leq n^{3/2} \tilde{\gamma}_w^{(6d+6z+13)} \|\mathbf{A}\|_F. \tag{32}$$

3.2.4 HQR Comparison to Uniform Precision Analysis

The mixed-precision segments of the analysis behind Theorem 3.5 derive from the mixed-precision inner product scheme outlined in Assumption 3.2 and are propagated to form the error bounds for a single Householder transformation as is shown in Equation 27. All steps to form the error bounds in Theorem 3.5 from the error bound for a single Householder transformation (Equation 27) directly follow the analyses in Section 19.3 of [1]. In these steps, we generalize the single Householder transformation error bound,

$$\text{fl}(\mathbf{P_v x}) = (\mathbf{P_v} + \mathbf{\Delta P_v})\mathbf{x}, \qquad \|\mathbf{\Delta P_v}\|_F \leq \epsilon, \tag{33}$$

for some small quantity $0 < \epsilon \ll 1$, and propagate it through the for-loop in Algorithm 7. This process then results in forward error bound coefficients $n\epsilon$ or $n^{3/2}\epsilon$. Since this ϵ value remains

constant, the rounding error analysis for both mixed-precision and uniform-precision schemes are essentially the same with different values for ϵ. The uniform precision equivalent of Equation 27 is shown in Equation 34,

$$\text{fl}(\mathbf{P_v x}) = (\mathbf{P_v} + \mathbf{\Delta P_v})\mathbf{x}, \qquad \|\mathbf{\Delta P_v}\|_F \leq \tilde{\gamma}^{(m)}, \tag{34}$$

which is derived in detail in [1]. Therefore, we only need to compare $\gamma^{(6d+6z+13)}$ against $\gamma^{(cm)}$, where c is a small integer. Although d relies on both m and the precisions w and s, we can generally assume that $cm \gg (6d + 6z + 13)$ in most mixed-precision settings. Therefore, the new bounds in Theorem 3.5 are much tighter than the existing ones and more accurately describe the kind of rounding error accumulated in mixed-precision computational settings.

4 Tall-and-Skinny QR

Some important problems that require QR factorizations of overdetermined systems include least squares problems, eigenvalue problems, low rank approximations, as well as other matrix decompositions. Although Tall-and-Skinny QR (TSQR) broadly refers to row-block QR factorization methods, we will discuss a specific variant of TSQR which is also known as the AllReduce algorithm [15]. In this paper, the TSQR/AllReduce algorithm refers to the most parallel variant of all row-block QR factorization algorithms discussed in [12]. A detailed description and rounding error analysis of this algorithm can be found in [15], and we present a pseudocode for the algorithm in Algorithm 4. Our initial interest in this algorithm came from its parallelizable nature, which is particularly suitable to implementation on GPUs. Additionally, our numerical simulations (discussed in Section 4.3) show that TSQR can not only increase the speed but also outperform the traditional HQR factorization in low precisions.

4.1 TSQR/AllReduce Algorithm

Algorithm 4 takes a tall-and-skinny matrix, $\mathbf{A}$, and organizes it into row-blocks. HQR factorization is performed on each of those blocks, and pairs of $\mathbf{R}$ factors are combined to form the next set of $\mathbf{A}$ matrices to be QR factorized. This process is repeated until only a single $\mathbf{R}$ factor remains, and the $\mathbf{Q}$ factor is built from all of the Householder constants and vectors stored at each level. The most gains from parallelization can be made in the initial level where the maximum number of independent HQR factorizations occur. Although more than one configuration of this algorithm may be available for a given tall-and-skinny matrix, the number of nodes available and the shape of the matrix eliminate some of those choices. For example, a 1600-by-100 matrix can be partitioned into 2, 4, 8, or 16 initial row-blocks but may be restricted by a machine with only 4 nodes, and a 1600-by-700 matrix can only be partitioned into 2 initial blocks. Our numerical experiments show that the choice in the initial partition, which directly relates to the recursion depth of TSQR, has an impact in the accuracy of the QR factorization.

We refer to *level* as the number of recursions in a particular TSQR implementation. An L-level TSQR algorithm partitions the original matrix into 2^L submatrices in the initial or 0^{th} level of the algorithm, and 2^{L-i} QR factorizations are performed in level i for $i = 1, \cdots, L$. The set of matrices that are QR factorized at each level i are called $\mathbf{A}_j^{(i)}$ for $j = 1, \cdots, 2^{L-i}$, where superscript (i) corresponds to the level and the subscript j indexes the row-blocks within level i. In the following sections, Algorithm 4 (`tsqr`) will find a TSQR factorization of a matrix $A \in \mathbb{R}^{m\times n}$ where $m \gg n$.

The inline function `qr` refers to Algorithm 7, `hh_mult` is Algorithm 3, and we use Algorithm 2 as a subroutine of `qr`.

4.1.1 TSQR Notation

We will introduce new notation due to the multi-level nature of the TSQR algorithm. In the final task of constructing $\mathbf{Q}$, $\mathbf{Q}_j^{(i)}$ factors are aggregated from each block at each level. Each $\mathbf{Q}_j^{(i)}$ factor from level i is partitioned such that two corresponding $\mathbf{Q}^{(i-1)}$ factors from level $i-1$ can be applied to them. The partition (approximately) splits $\mathbf{Q}_j^{(i)}$ into two halves, $[\tilde{\mathbf{Q}}_{j,1}^{(i)\top}\tilde{\mathbf{Q}}_{j,2}^{(i)\top}]^\top$. The functions $\alpha(j)$ and $\phi(j)$ are defined such that $\mathbf{Q}_j^{(i)}$ is applied to $\tilde{\mathbf{Q}}_{\alpha(j),\phi(j)}^{(i+1)}$. For $j = 1, \cdots, 2^{L-i}$ at level i, we need $j = 2(\alpha(j)-1)+\phi(j)$, where $\alpha(j) = \lceil \frac{j}{2} \rceil$ and $\phi(j) = 2 + j - 2\alpha(j)$. Section 4.1.2 shows full linear algebra details for a single-level ($L = 1$, 2 initial blocks) example. The reconstruction of $\mathbf{Q}$ can be implemented more efficiently (see [16]), but the reconstruction method in Algorithm 4 is presented for a clear, straightforward explanation.

4.1.2 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 , i.e., $\mathbf{A}_1^{(1)} = \begin{bmatrix}\mathbf{R}_1^{(0)}\\ \mathbf{R}_2^{(0)}\end{bmatrix}$, which is QR factorized, and the process is repeated:

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

The $\mathbf{R}$ factor of $\mathbf{A}_1^{(1)}$ is the final $\mathbf{R}$ factor of the QR factorization of the original matrix, $\mathbf{A}$. However, the final $\mathbf{Q}$ still needs to be constructed. Bisecting $\mathbf{Q}_1^{(1)}$ into two submatrices, i.e. $\tilde{\mathbf{Q}}_{1,1}^{(1)}$ and $\tilde{\mathbf{Q}}_{1,2}^{(1)}$, allows us to write and compute the product more compactly,

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

More generally, Algorithm 4 takes a tall-and-skinny matrix $\mathbf{A}$ and level L and finds a QR factorization by initially partitioning $\mathbf{A}$ into 2^L row-blocks and includes the building of $\mathbf{Q}$.

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

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$ where $m \gg n$, $L \leq \lfloor \log_2 \left(\frac{m}{n}\right) \rfloor$, and 2^L is the initial number of blocks.
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} blocks. */
for $j = 1 : 2^L - 1$ **do**
  $\mathbf{A}_j^{(0)} \leftarrow \mathbf{A}[(j-1)h + 1 : jh, :]$
$\mathbf{A}_{2^L}^{(0)} \leftarrow \mathbf{A}[(2^L - 1)h : m, :]$ // Last block may have more rows.
/* Store Householder vectors as columns of matrix $\mathbf{V}_j^{(i)}$, Householder constants as components of vector $\boldsymbol{\beta}_j^{(i)}$, and set up the next level. */
for $i = 0 : L - 1$ **do**
  /* The inner loop can be parallelized. */
  for $j = 1 : 2^{L-i}$ **do**
    $\mathbf{V}_{2j-1}^{(i)}, \boldsymbol{\beta}_{2j-1}^{(i)}, \mathbf{R}_{2j-1}^{(i)} \leftarrow \texttt{qr}(\mathbf{A}_{2j-1}^{(i)})$
    $\mathbf{V}_{2j}^{(i)}, \boldsymbol{\beta}_{2j}^{(i)}, \mathbf{R}_{2j}^{(i)} \leftarrow \texttt{qr}(\mathbf{A}_{2j}^{(i)})$
    $\mathbf{A}_j^{(i+1)} \leftarrow \begin{bmatrix} \mathbf{R}_{2j-1}^{(i)} \\ \mathbf{R}_{2j}^{(i)} \end{bmatrix}$
/* At the bottom-most level, get the final $\mathbf{R}$ factor. */
$\mathbf{V}_1^{(L)}, \boldsymbol{\beta}_1^{(L)}, \mathbf{R} \leftarrow \texttt{qr}(\mathbf{A}_1^{(L)})$
$\mathbf{Q}_1^{(L)} \leftarrow \texttt{hh\_mult}(\mathbf{V}_1^{(L)}, I_{2n \times n})$
/* Compute $\mathbf{Q}^{(i)}$ factors by applying $\mathbf{V}^{(i)}$ to $\mathbf{Q}^{(i+1)}$ factors. */
for $i = L - 1 : -1 : 1$ **do**
  for $j = 1 : 2^{L-i}$ **do**
    $\mathbf{Q}_j^{(i)} \leftarrow \texttt{hh\_mult}\left(\mathbf{V}_j^{(i)}, \begin{bmatrix} \tilde{\mathbf{Q}}_{\alpha(j),\phi(j)}^{(i+1)} \\ \mathbf{0}_{n,n} \end{bmatrix}\right)$
/* At the top-most level, construct the final $\mathbf{Q}$ factor. */
$\mathbf{Q} \leftarrow []$;
for $j = 1 : 2^L$ **do**
  $\mathbf{Q} \leftarrow \begin{bmatrix} \mathbf{Q} \\ \texttt{hh\_mult}\left(\mathbf{V}_j^{(0)}, \begin{bmatrix} \tilde{\mathbf{Q}}_{\alpha(j),\phi(j)}^{(1)} \\ O_{\tilde{h},n} \end{bmatrix}\right) \end{bmatrix}$
return $\mathbf{Q}$, $\mathbf{R}$

4.2 TSQR Rounding Error Analysis

The TSQR algorithm presented in Algorithm 4 is a divide-and-conquer strategy for the QR factorization that uses the HQR within the subproblems. Divide-and-conquer methods can naturally be implemented in parallel and accumulate less rounding errors. For example, the single-level TSQR

decomposition of a tall-and-skinny matrix, $\mathbf{A}$ requires 3 total HQRs of matrices of sizes $\lfloor \log_2(\frac{m}{n}) \rfloor$-by-$n$, $\lceil \log_2(\frac{m}{n}) \rceil$-by-$n$, and $2n$-by-n. The single-level TSQR strictly uses more FLOPs, but the dot product subroutines may accumulate smaller rounding errors (and certainly have smaller upper bounds) since they are performed on shorter vectors, and lead to a more accurate solution overall. These concepts are elucidated in [15], where the rounding error analysis of TSQR is shown in detail in [15]. We summarize the main results from [15] in Theorem 4.1.

Theorem 4.1. *Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ with $m \geq n$ have full rank, n, and $\hat{\mathbf{Q}} \in \mathbb{R}^{m \times n}$ and $\hat{\mathbf{R}} \in \mathbb{R}^{n \times n}$ be the thin QR factors of $\mathbf{A}$ obtained via Algorithm 4. Then we have normwise forward error bounds*

$$\hat{\mathbf{A}} = \mathbf{A} + \mathbf{\Delta A} = \mathbf{Q}(\mathbf{R} + \mathbf{\Delta R}),$$
$$\hat{\mathbf{Q}} = \mathbf{Q} + \mathbf{\Delta Q},$$

where

$$\|\mathbf{\Delta R}\|_F, \|\mathbf{\Delta A}\|_F \leq \left[n\tilde{\gamma}_{\frac{m}{2^L}} + (1 + n\tilde{\gamma}_{\frac{m}{2^L}}) \left\{ (1 + n\tilde{\gamma}_{2n})^L - 1 \right\} \right] \|\mathbf{A}\|_F, \text{ and} \tag{35}$$

$$\|\mathbf{\Delta Q}\|_F \leq \sqrt{n} \left[(1 + n\tilde{\gamma}_{\frac{m}{2^L}})(1 + n\tilde{\gamma}_{2n})^L - 1 \right]. \tag{36}$$

Furthermore, if we assume $n\tilde{\gamma}_{\frac{m}{2^L}}, n\tilde{\gamma}_{2n} \ll 1$, the coefficient for $\|\mathbf{A}\|_F$ in Equations 35 can be approximated as

$$\left[n\tilde{\gamma}_{\frac{m}{2^L}} + (1 + n\tilde{\gamma}_{\frac{m}{2^L}}) \left\{ (1 + n\tilde{\gamma}_{2n})^L - 1 \right\} \right] \simeq n\tilde{\gamma}_{\frac{m}{2^L}} + Ln\tilde{\gamma}_{2n}, \tag{37}$$

and the right hand side of Equation 36 can be approximated as

$$\sqrt{n} \left[(1 + n\tilde{\gamma}_{\frac{m}{2^L}})(1 + n\tilde{\gamma}_{2n})^L - 1 \right] \simeq \sqrt{n} \left(n\tilde{\gamma}_{\frac{m}{2^L}} + Ln\tilde{\gamma}_{2n} \right). \tag{38}$$

We can also form a backward error, where $\mathbf{A} + \mathbf{\Delta A}_{TSQR} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$, and both $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$ are obtained via Algorithm 4. Then,

$$\|\mathbf{\Delta A}_{TSQR}\|_F = \|\mathbf{Q\Delta R} + \mathbf{\Delta Q}\hat{\mathbf{R}}\|_F \simeq \sqrt{n} \left(n\tilde{\gamma}_{\frac{m}{2^L}} + Ln\tilde{\gamma}_{2n} \right) \|\mathbf{A}\|_F. \tag{39}$$

In Section 3.2.4, the steps of the HQR algorithm resulted in an error bound of $\mathcal{O}(\epsilon)$, where the constant is some function with respect to n and where ϵ is the error bound for a single Householder transformation, described in Equation 33 . Similarly, the analysis behind Theorem 4.1 can be generalized via defining ϵ_1 to be the error bound for a single Householder transformation corresponding to the vector length at the initial level 0, $\frac{m}{2^L}$, and defining ϵ_2 to be the error bound for a Householder transformation corresponding to vector length in all deeper levels, $2n$. This generalization leads to the error bound coefficients

$$n\epsilon_1 + Ln\epsilon_2 \quad \text{for} \quad \|\mathbf{\Delta Q}\|_F, \|\mathbf{\Delta A}_{\text{TSQR}}\|_F, \tag{40}$$

$$\sqrt{n}(n\epsilon_1 + Ln\epsilon_2) \quad \text{for} \quad \|\mathbf{\Delta R}\|_F, \|\mathbf{\Delta A}\|_F. \tag{41}$$

In a uniform-precision setting, these correspond to

$$\epsilon_1 = \tilde{\gamma}^{(\frac{m}{2^L})} \quad \text{and} \quad \epsilon_2 = \tilde{\gamma}^{(2n)}, \tag{42}$$

and in the mixed-precision setting outlined in Assumption 3.2, they correspond to

$$\epsilon_1 = \gamma_w^{(6d_1+6z+13)}, \quad \text{and } \epsilon_2 = \gamma_w^{(6d_2+6z+13)}, \tag{43}$$

where $d_1 := \lfloor (\frac{m}{2^L} - 1)\frac{u_s}{u_w} \rfloor$ and $d_2 := \lfloor \frac{(2n-1)u_s}{u_w} \rfloor$ respectively. In both settings, we see that increasing L may decrease ϵ_1, but may still increase the overall bounds; the larger L still could have an adverse effect on the coefficients in Theorem 4.1. This trade-off is precisely the balance between the sizes of initial blocks and the number of levels in the TSQR algorithm, and an optimal TSQR scheme would ideally minimize ϵ_1 and ϵ_2 with the choice of L. These error bounds are studied in detail in the following section.

4.2.1 HQR and TSQR error bound comparison

We compare the error bounds for HQR and TSQR algorithms.

Uniform precision comparison Consider the larger error bounds in the uniform precision equivalents of Theorems 3.5 and 4.1, which are the bounds of $\mathbf{\Delta Q}$ and $\mathbf{\Delta A}$. In order for the a meaningful TSQR error bound to outperform the bound for the HQR algorithm, we need integers $m, n > 0$, and $L \geq 0$ such that,

$$1 \gg n^{3/2}\gamma^{(m)} \gg n^{3/2}(\gamma^{(\frac{m}{2^L})} + L\gamma^{(2n)}).$$

If we assume $\frac{m}{2^L} = 2n$, the HQR bound is $\frac{L+1}{2^L}$ larger than the bound for TSQR with L levels. For example, in single precision, a HQR of a 2^{15}-by-2^6 matrix results in an upper bound relative backward error ($\|\mathbf{A} - \hat{\mathbf{Q}}\hat{\mathbf{R}}\|_F / \|\mathbf{A}\|_F$) of $\approx$`1.002`, but a TSQR with $L = 8$ is bounded by $\approx$`3.516e-02`. This case exemplifies a situation in which stability is not guaranteed in HQR, but the method is stable when using TSQR, even in the worst-case. Now consider some 2^{20}-by-2^{12} matrix and QR factorizations performed with double precision. The error bound for HQR is `1.686e-7`, whereas the error bound for TSQR with 12 levels is `5.351e-10`. In general, we can conjecture that values of L that can make $m2^{-L}$ and $2Ln$ much smaller than m, should produce a TSQR that outperforms HQR in worst-case scenarios, at least in uniform precision settings. However, the range of matrix sizes that TSQR can accommodate decreases as L grows larger. Figure 1 shows the matrix sizes HQR, 2-level TSQR, and 4-level TSQR can accommodate as well as their respective error bounds.

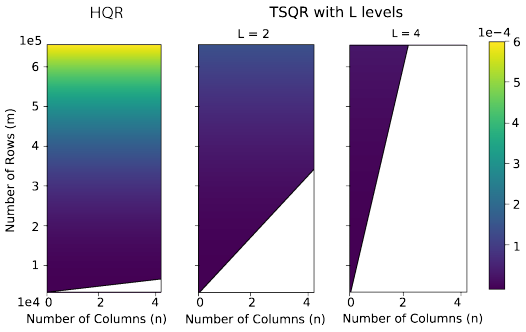

Figure 1: Non-white space indicates allowable matrix sizes for each scheme, and color map represents error bounds for $\|\mathbf{\Delta Q}\|_F$ for uniform precision error analysis when using double precision arithmetic.

Mixed precision comparison Consider a mixed-precision setting such as in Assumption 3.2, and we assume $u_p = u_w$, so that $z = 2$. In order for the a meaningful TSQR error bound to outperform the bound for the HQR algorithm, we now need integers $m, n > 0$, and $L \geq 0$ such that

$$1 \gg n^{3/2}\gamma_w^{(6d+25)} \gg n^{3/2}(\gamma_w^{(6d_1+25)} + L\gamma_w^{(6d_2+25)}),$$

where $d = \lfloor \frac{(m-1)u_s}{u_w} \rfloor$, $d_1 = \lfloor (\frac{m}{2^L} - 1)\frac{u_s}{u_w} \rfloor$, and $d_2 = \lfloor \frac{(2n-1)u_s}{u_w} \rfloor$.

In contrast to the analysis for uniform precision settings, large L values do not necessarily reduce the

error bounds of TSQR. While large L can imply $m \gg m2^{-L} + 2Ln$, it does not always lead to $d \gg d_1 + Ld_2$. Although the theoretical error bounds do not give a clear indication of the worst-case performances of HQR and TSQR in mixed-precision settings, TSQR outperformed HQR on ill-conditioned matrices within our numerical simulations.

These experiments are discussed in detail in the next section.

4.3 Numerical Experiment

In Section 4.2.1, we theorized that conditions exist where TSQR could outperform HQR and that these conditions were hard to identify in mixed-precision settings. An empirical comparison of these two QR factorization algorithms in double precision can be found in [15], where they conclude that deeper TSQR tends to produce more accurate QR factorizations than HQR. However, using TSQR with deep levels (large L) can actually start to perform worse than TSQR with shallower levels (smaller L), since deeper levels require more FLOPs. We instead focused on comparing HQR and TSQR performances in a mixed-precision setting. Our numerical simulations show that TSQR can still outperform HQR in low, mixed-precision settings in practice even though the theoretical bounds do not guarantee stability. Our empirical results do not behave as the theoretical bounds suggest, and even show opposite trends at times. This discrepancy highlights the shortcomings of deterministic error bounds that are too pessimistic.

We used Julia v1.0.4 for all of the numerical simulations. This programming language allows half precision storage as well as `castup` and `castdown` operations to and from single and double precisions, but has no half precision arithmetic. Therefore, we relied on using Algorithm 1 for $f \in \text{OP} \cup \{\texttt{dot\_product}\}$ to simulate half and mixed-precision arithmetic operations. For HQR, we created a mixed-precision version of the LAPACK routine xGEQRF, where the dot product subroutine was approximated by $\text{fl}(\mathbf{x}_{\text{half}}^\top \mathbf{y}_{\text{half}})$ with $\texttt{simHalf}(\texttt{dot\_product}, \mathbf{x}_{\text{half}}, \mathbf{y}_{\text{half}})$ to simulate the mixed-precision setting described in Assumption 3.2 with $u_p = 0$ (which implies $z = 1$), and we used Algorithm 1 on all other basic operations in OP to simulate half/storage precision arithmetic. This HQR was then used as a subroutine of TSQR as well. There are cases where the rounding will differ between the mixed-precision setting and the way we mimic it, i.e., basic operations that are meant to be in half/storage precision arithmetic, but are instead casted up to single and back down, as the tiebreaker within correct rounding may lead to different results than true half/storage precision arithmetic. All in all, our experiments nearly replicated the mixed-precision setting we assumed for the error analysis in Sections 3.2 and 4.2.

Following example from [15], we used m-by-n random matrices, $\mathbf{A}_\alpha$, constructed via

$$\mathbf{A}_\alpha = \mathbf{Q}'(\alpha\mathbf{E} + \mathbf{I})/\|\mathbf{Q}'(\alpha\mathbf{E} + \mathbf{I})\|_F, \tag{44}$$

where $\mathbf{Q}' \in \mathbb{R}^{m\times n}$ is a random orthogonal matrix and $\mathbf{E} \in \mathbb{R}^{n\times n}$ is the matrix of 1's. The random orthogonal matrix $\mathbf{Q}'$ is generated by taking a QR factorization of an iid 4000-by-100 matrix sampled from $Unif(0, 1)$, and we used the built-in QR factorization function in Julia. By construction, $\mathbf{A}_\alpha$ has 2-norm condition number $n\alpha+1$. By varying α from `1e-4` to `1`, we varied the condition number from 1.1 to 101, and we generated 10 samples for each value of α.

We generated random matrices of size 4000-by-100 using Equation 44 and computed their HQR and TSQR for $L = 1, \cdots, 6$ in a mixed-precision setting that simulates Assumption 3.2 with $z = 1$. The relative backward error, $\|\hat{\mathbf{Q}}\hat{\mathbf{R}} - \mathbf{A}\|_F/\|\mathbf{A}\|_F$, was computed by casting up $\hat{\mathbf{Q}}$, $\hat{\mathbf{R}}$, and $\mathbf{A}$ to double precision to compute the Frobenius norms. Note that the mixed-precision HQR error bounds $n\tilde{\gamma}_w^{(6d+6z+13)}$ and $n^{3/2}\tilde{\gamma}_w^{(6d+6z+13)}$ for $m = 4000$ and $n = 100$ are `0.936` and `9.364` respectively, and

the mixed-precision TSQR bounds for $L = 1, \cdots, 5$ are even larger, which indicates that our error bounds do not guarantee stability.

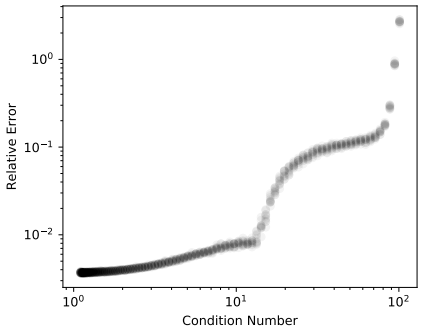

Figure 2: HQR errors for matrices with varying condition numbers.

Figure 2 shows the backward errors of mixed precision HQR increasing as the theoretical condition numbers of the generated random matrices increase, and these errors correspond to the error data on the vertical axis, $L = 0$, of Figure 3. In addition to the errors from HQR, Figure 3 shows the errors from mixed precision TSQR of levels varying from $L = 1$ to $L = 5$, where each line represents the errors of HQR and variants of TSQR calculated from the same random test matrix. Figure 3 reveals two different trends for the errors as we deepen the complexity of the QR algorithm from HQR to TSQR with 5 levels. One trend occurs for matrices with smaller condition numbers, where HQR and all levels of TSQR are stable, but deepening the levels of TSQR worsens the errors. The other trend occurs for matrices with higher condition numbers, where single-level and 2-level TSQR yield smaller errors than HQR. In these cases, TSQR with 3 or more levels have errors similar to or worse than 2-level TSQR, but those errors tend to not rise above the HQR errors. These results suggests that TSQR can significantly outperform HQR even in mixed-precision settings, and particularly when HQR is unstable due to larger condition numbers. Although this experiment focused on condition numbers, identifying other properties that point to better performance of TSQR than HQR can further broaden the potential use of mixed-precision TSQR in applications.

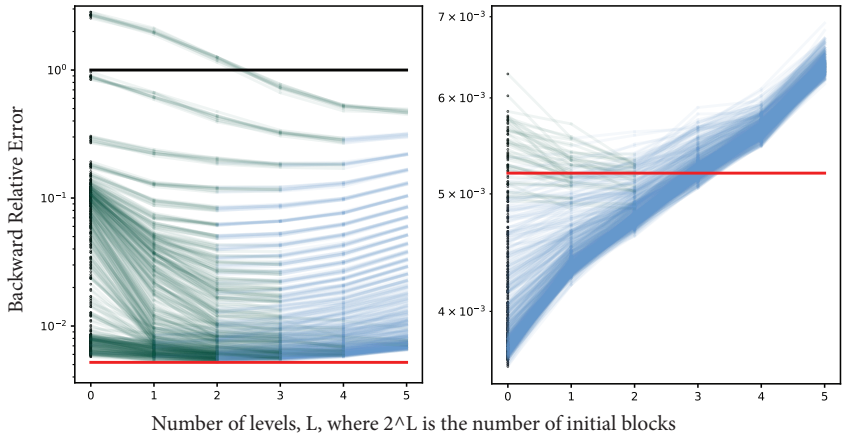

Figure 3: Left plot shows the relative error of QR factorization for matrices with condition numbers ranging from 5.3 to 101, and the right plot shows the errors for matrices with condition numbers ranging from 1.1 to 5.3.

5 Applications

Many applications in scientific computing typically employ double precision when lower precision may actually be sufficient. Due to the advances in processors, FLOPs are now considered free, causing bandwidth and storage to be the computational bottleneck. With the emergence of new technology that supports low precision and the need to reduce bandwidth and storage concerns, interest in mixed-precision algorithms has reemerged.

Since low and mixed precision settings benefit from speed-up and reduced storage, applications that process large amounts of data are potential candidates for this research. Here, we discuss our results from applying our mixed-precision HQR as a subroutine of an iterative eigensolver in the context of spectral clustering.

Graph partitioning A graph is defined by a set of nodes and a set of edges between the nodes. Partitioning, or clustering, is a task that seeks communities within a graph such that nodes within a community are *more similar* to each other than to nodes outside of that community. In datasets where the true communities are known, we can use pairwise-precision and pairwise-recall (see [17]) which are defined in Definition 5.1) to evaluate the accuracy of a clustering task.

Definition 5.1. Pairwise-precision and pairwise-recall are measured by checking for every pair of nodes if the pair is classified into the same cluster (positive), or else (negative).

$$\text{Precision} = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Positive}}, \qquad \text{Recall} = \frac{\#\text{True Positive}}{\#\text{True Positive} + \#\text{False Negative}}. \tag{45}$$

5.1 Spectral Graph Clustering

Some spectral clustering methods utilize identifying k dominant eigenvectors of a similarity matrix of a graph, which then can be used to identify k clusters. Another potential use of iterative eigensolvers for spectral clustering is in identifying the second smallest eigenvalue and its eigenvector pair, called the Fiedler value and vector. In addition, many eigenproblems outside of spectral clustering only require finding a few eigen pairs. This family of problems tends to admit tall-and-skinny matrix structures and could utilize TSQR as well. We will use subspace iteration, a variant of the power method defined in Algorithm 5 that uses a QR factorization of a tall-and-skinny matrix at each iteration and that quickly converges to the dominant eigenvectors. Although we only experimented with comparing mixed-precision HQR to uniform precision HQR, TSQR could also be used in this application.

5.1.1 Subspace Iteration

Subspace iteration is a modification of the power method, which computes an invariant subspace with dimension $p > 1$ (see [18]). A variant of this algorithm is shown below in Algorithm 5.

This algorithm is an iterative method with two possible stopping criteria: 1) the maximum number of iterations to complete before exiting the loop is declared as max_iter, or 2) if the eigenspace error is smaller than τ, then exit the loop. In practice, we added a third stopping criterion in the case that the declared τ value was too small, which would force an exit from the loop when the eigenspace error began to increase.

Algorithm 5: $\mathbf{Q} = \texttt{subIter}(\mathbf{A}, \texttt{max\_iter}, \tau, k)$. Find orthogonal basis (given by columns of output matrix Q) of an invariant subspace of the input adjacency matrix, A.

Input: Adjacency matrix $\mathbf{A} \in \{0,1\}^{m \times m}$ where $m \geq n$, `max_iter`, the maximum number of iterations, τ the threshold for the eigenspace error, and k, the suspected number of clusters.

Output: $\mathbf{Q}$

Initialize $\mathbf{Y} \in \mathbb{R}^{m \times k}$, a random matrix. // Y would likely be full-rank.

$\mathbf{Q}, \mathbf{R} \leftarrow \texttt{qr}(\mathbf{Y})$ **for** $i = 1, \cdots,$ `max_iter` **do**

  $\mathbf{Y} \leftarrow \mathbf{AQ}$

  if $\frac{\|\mathbf{Y} - \mathbf{QQ}^\top \mathbf{Y}\|_2}{\|\mathbf{Y}\|_2} < \tau$ **then**

    exit loop. // $\|\mathbf{Y} - \mathbf{QQ}^\top \mathbf{Y}\|_2$ is the eigenspace error.

  $\mathbf{Q}, \mathbf{R} \leftarrow \texttt{qr}(\mathbf{Y})$

return $\mathbf{Q}$

5.1.2 Density-based Spatial clustering of Applications with Noise (DBSCAN)

DBSCAN is a density-based spatial clustering algorithm introduced in [19] and is widely used in practice. This algorithm only requires input data, location of nodes, and two parameters, radius of neighborhoods and minimum number of points required to form a dense region. The two parameters for the DBSCAN algorithm were tuned to provide the best result, given that we used the same set of parameters for the entire experiment.

5.2 Experiment Details and Results

Our main goal in this experiment was to test if the eigenspaces identified by lower precision HQR could produce sufficient graph partitioning. We used subspace iteration (Algorithm 5) to identify eigenspaces, DBSCAN to partition the embedding of the nodes onto these eigenspaces, and precision and recall to evaluate clustering performances. We used a static graph of 5000 nodes with 19 known true partitions for the Graph Challenge [17], which are derived from block stochastic matrices. The graphs we used were undirected and unweighted; the only elements in the adjacency matrices were 0's and 1's, which can easily be represented in half, single, and double precision floats. For $i = 1, \cdots, 10$, let $\mathbf{Y}_{\text{half},i} \in \mathbb{F}_{\text{half}}^{5000 \times 19}$ be the half precision storage of the i^{th} random matrix. Since any half precision float can be exactly represented in single and double precisions, $\mathbf{Y}_{\text{half},i}$'s can be easily cast up to single and double precisions, $\mathbf{Y}_{\text{single},i}$ and $\mathbf{Y}_{\text{double},i}$. We performed mixed-precision HQR within subspace iteration initialized by $\mathbf{Y}_{\text{half},i}$'s, and uniform-precision HQR for subspace iteration initialized by $\mathbf{Y}_{\text{single},i}$'s and $\mathbf{Y}_{\text{double},i}$'s. For trial $i = 1, \cdots, 10$, we repeated the following steps.

Step 1. Identify an orthogonal basis of dimension 19 (number of known true partitions) with subspace iteration using the appropriate HQR routine for $\mathbf{Y}_{\text{half},i}$, $\mathbf{Y}_{\text{single},i}$ and $\mathbf{Y}_{\text{double},i}$.

Step 2. Apply DBSCAN to the output matrices of previous step to cluster most nodes into communities and the remaining nodes as outliers.

Step 3. Measure clustering performances of DBSCAN on the three different precision subspace iteration embeddings using precision and recall.

Subspace Iteration Results Figure 4 shows the eigenspace error, $\|\mathbf{Y}-\mathbf{Q}\mathbf{Q}^\top\mathbf{Y}\|_2/\|\mathbf{Y}\|_2$, from the subspace iteration step of one trial. The stopping criteria, τ, were set to $5u_{\text{single}}$ and $5u_{\text{double}}$ for the uniform precision HQRs and $5u_{\text{half}}$ for the mixed-precision HQR. The solid lines are plotted to show the unit roundoff values. The uniform precision implementations of subspace iterations reached their stopping criterion set by τ, and the mixed-precision implementation fluctuated close to but never dipped below τ. The convergence rate was approximately the same across the three different implementations, which suggests that the lower precision routines (mixed-precision HQR or uniform single precision HQR) can be used as a preconditioner for the double precision solution, and if paired with appropriate hardware could lead to increased computational efficiency. In addition, if double precision eigenspace error is not necessary to achieve sufficient clustering results, we can simply use the lower precision HQR subspace iterations as full eigensolvers.

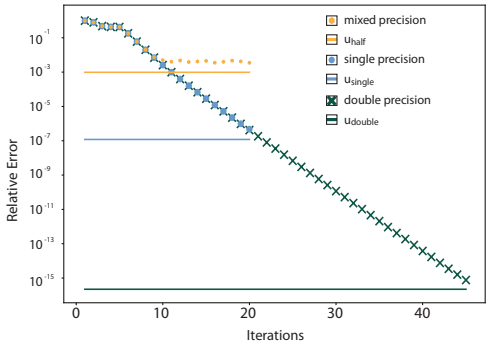

Figure 4: Eigenspace Error for subspace iteration with using double-, single-, and half- precision traditional Householder QR factorizations.

Clustering Results Table 4 shows the worst-case precision and recall results from the 10 trials for each subspace iteration implementation. The DBSCAN algorithm and the calculation of precision and recall were computed in double precision, and the variance in precision and recall values for these 10 trials were in the range of `1e-6`. Subspace iteration that employs lower precision HQR results in a suboptimal solution to the basis, which has a larger loss in orthogonality when compared to the solution from subspace iteration that uses higher precision HQR. However, clustering results show minimal difference in the precision and recall and suggests that a lower precision HQR within subspace iteration can still lead to a sufficiently accurate clustering.

HQR Scheme	Mixed Precision	Single Precision	Double Precision
Prec / Recall	`0.9822 / 0.9393`	`0.9817 / 0.9407`	`0.9822 / 0.9405`

Table 4: Minimum (worst-case) precision and recall for 10 trials on graph with 5000 nodes and 19 true clusters.

6 Conclusion

Though the use of lower precision naturally reduces the bandwidth and storage needs, the development of GPUs to optimize low precision floating point arithmetic have accelerated the interest in half precision and mixed-precision algorithms. Loss in precision, stability, and representable range offset for those advantages, but these shortcomings may have little to no impact in some applications. It may even be possible to navigate around those drawbacks with algorithmic design.

The existing rounding error analysis cannot accurately bound the behavior of mixed-precision arithmetic. We have developed a new framework for mixed-precision rounding error analysis and

applied it to HQR, a widely used linear algebra routine, and implemented it in an iterative eigensolver in the context of spectral clustering. The mixed-precision error analysis builds from the inner product routine, which can be applied to many other linear algebra tools as well. The new error bounds more accurately describe how rounding errors are accumulated in mixed-precision settings. We also found that TSQR, a communication-avoiding, easily parallelizable QR factorization algorithm for tall-and-skinny matrices, can outperform HQR in mixed-precision settings for ill-conditioned, extremely overdetermined cases, which suggests that some algorithms are more robust against lower precision arithmetic. As QR factorizations of tall-and-skinny matrices are common in spectral clustering, we experimented with introducing mixed-precision settings into graph partitioning problems. In particular, we applied DBSCAN to the spectral basis of a graph identified via subspace iteration that used our simulated mixed-precision HQR, which yielded clustering results tantamount to results from employing double-precision entirely.

Although this work is focused on QR factorizations and applications in spectral clustering, the mixed precision round-off error analysis can be applied to other tasks and applications that can benefit from employing low precision computations. While the emergence of technology that support low precision floats combats issues dealing with storage, now we need to consider how low precision affects stability of numerical algorithms.

Future work is needed to test larger, more ill-conditioned problems with different mixed-precision settings, and to explore other divide-and-conquer methods like TSQR that can harness parallel capabilities of GPUs while withstanding lower precisions.

A Proofs

A.1 Lemma 2.3 (Equation 16)

Proof. We wish to round up to the lower precision, p, since $1 \gg u_p \gg u_s$. Recall that $d := \lfloor k_s u_s / u_p \rfloor$ and $r \leq \lfloor u_p / u_s \rfloor$, and note $k_p u_p + k_s u_s = (k_p + d)u_p + r u_s \leq (k_p + d + 1)u_p$. Then,

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

□

A.2 Inner Products

A.2.1 Lemma 2.4

Let δ_p and δ_s be rounding error incurred from products and summations. They 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. Then,

$$\begin{aligned}
\hat{s_1} &= \text{fl}(\mathbf{x}_1\mathbf{y}_1) = \mathbf{x}_1\mathbf{y}_1(1+\delta_{p,1}),\\
\hat{s_2} &= \text{fl}(\hat{s_1} + \mathbf{x}_2\mathbf{y}_2),\\
&= [\mathbf{x}_1\mathbf{y}_1(1+\delta_{p,1}) + \mathbf{x}_2\mathbf{y}_2(1+\delta_{p,2})]\,(1+\delta_{s,1}),\\
\hat{s_3} &= \text{fl}(\hat{s_2} + \mathbf{x}_3\mathbf{y}_3),\\
&= ([\mathbf{x}_1\mathbf{y}_1(1+\delta_{p,1}) + \mathbf{x}_2\mathbf{y}_2(1+\delta_{p,2})]\,(1+\delta_{s,1}) + \mathbf{x}_3\mathbf{y}_3(1+\delta_{p,3}))\,(1+\delta_{s,2}).
\end{aligned}$$

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

$$\hat{s_m} = (\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2)(1+\delta_{p,1})\prod_{k=1}^{m-1}(1+\delta_{s,k}) + \sum_{i=3}^{n}\mathbf{x}_i\mathbf{y}_i(1+\delta_{p,i})\left(\prod_{k=i-1}^{m-1}(1+\delta_{s,k})\right), \tag{46}$$

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

Using Lemma 2.1, we further simplify:

$$\text{fl}(\mathbf{x}^\top\mathbf{y}) = \hat{s_m} = (1+\theta_p^{(1)})(1+\theta_s^{(m-1)})\mathbf{x}^\top\mathbf{y} = (\mathbf{x}+\Delta\mathbf{x})^\top\mathbf{y} = \mathbf{x}^\top(\mathbf{y}+\Delta\mathbf{y})$$

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

By using Lemma 2.3, Equation 16, we can bound the perturbations componentwise. Let $d := \lfloor\frac{(m-1)u_s}{u_p}\rfloor$ such that $(m-1)u_s = du_p + ru_s$. Then,

$$|\Delta\mathbf{x}| \le \gamma_p^{(d+2)}|\mathbf{x}| \qquad \text{and} \qquad |\Delta\mathbf{y}| \le \gamma_p^{(d+2)}|\mathbf{y}|.$$

Furthermore, these bounds lead to a forward error result as shown in Equation 47,

$$|\mathbf{x}^\top\mathbf{y} - \text{fl}(\mathbf{x}^\top\mathbf{y})| \le \gamma_p^{(d+2)}|\mathbf{x}|^\top|\mathbf{y}|. \tag{47}$$

A.2.2 Corollary 2.5

This proof follows similarly to the proof for Lemma 2.4. 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. The error for a general m-length vector dot product is then

$$\hat{s_m} = (\mathbf{x}_1\mathbf{y}_1 + \mathbf{x}_2\mathbf{y}_2)\prod_{k=1}^{m-1}(1+\delta_{s,k}) + \sum_{i=3}^{n}\mathbf{x}_i\mathbf{y}_i\left(\prod_{k=i-1}^{m-1}(1+\delta_{s,k})\right). \tag{48}$$

Using Lemma 2.1, we further simplify,

$$\text{fl}(\mathbf{x}^\top\mathbf{y}) = \hat{s_m} = (1+\theta_s^{(m-1)})\mathbf{x}^\top\mathbf{y} = (\mathbf{x}+\Delta\mathbf{x})^\top\mathbf{y} = \mathbf{x}^\top(\mathbf{y}+\Delta\mathbf{y}).$$

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

By using Lemma 2.3 equation 16, we can bound the perturbations componentwise. Let $d := \lfloor\frac{(m-1)u_s}{u_p}\rfloor$ such that $(m-1)u_s = du_p + ru_s$.

$$|\Delta\mathbf{x}| \le \gamma_p^{(d+1)}|\mathbf{x}| \qquad \text{and} \qquad |\Delta\mathbf{y}| \le \gamma_p^{(d+1)}|\mathbf{y}|$$

Furthermore, these bounds lead to a forward error result as shown in Equation 49,

$$|\mathbf{x}^\top\mathbf{y} - \text{fl}(\mathbf{x}^\top\mathbf{y})| \le \gamma_p^{(d+1)}|\mathbf{x}|^\top|\mathbf{y}|. \tag{49}$$

A.3 Proof for Mixed-Precision HQR result

Here, we show a few results that are necessary for the proof for Theorem 3.5. Lemma A.1 shows normwise results for a single mixed-precision Householder transformation performed on a vector, and Lemma A.2 builds on Lemma A.1 to show normwise results for multiple mixed-precision Householder transformations on a vector. We build column-wise results for HQR based on these lemmas and then compute the matrix norms at the end.

Lemma A.1. *Let $\mathbf{x} \in \mathbb{R}^m$ and consider the computation of $\mathbf{y} = \hat{\mathbf{P}}_v\mathbf{x} = \mathbf{x} - \hat{\beta}\hat{\mathbf{v}}\hat{\mathbf{v}}^\top\mathbf{x}$, where $\hat{\mathbf{v}}$ has accumulated error shown in Lemma 3.4. Then, the computed $\hat{\mathbf{y}}$ satisfies*

$$\hat{\mathbf{y}} = (\mathbf{P} + \mathbf{\Delta P})\mathbf{x}, \quad \|\mathbf{\Delta P}\|_F \leq \gamma_w^{(6d+6z+13)}, \tag{50}$$

where $\mathbf{P} = \mathbf{I} - \beta\mathbf{v}\mathbf{v}^\top$ is a Householder transformation.

Proof. Recall that the computed $\hat{\mathbf{y}}$ accumulates component-wise error shown in Equation 27. Even though we do not explicitly form $\mathbf{P}$, forming the normwise error bound for this matrix makes the analysis simple. First, recall that any matrix $\mathbf{A}$ with rank r has the following relations between its 2-norm and Frobenius norm, $\|\mathbf{A}\|_2 \leq \|\mathbf{A}\|_F \leq \sqrt{r}\|\mathbf{A}\|_2$. Then, we have

$$\|\mathbf{y}\|_2 = \|\mathbf{P}\mathbf{x}\|_2 \leq \|\mathbf{P}\|_2\|\mathbf{x}\|_2 = \|\mathbf{x}\|_2, \tag{51}$$

since $\mathbf{P}$ is orthogonal and $\|\mathbf{P}\|_2 = 1$. We now transition from the componentwise error to normwise error for $\mathbf{\Delta y}$, and write $\tilde{z} = 6d + 6z + 13$:

$$\|\mathbf{\Delta y}\|_2 = \left(\sum_{i=1}^{m} \mathbf{\Delta y}_i^2\right)^{1/2} \leq \gamma_w^{(\tilde{z})} \left(\sum_{i=1}^{m} \mathbf{y}_i^2\right)^{1/2} = \gamma_w^{(\tilde{z})}\|\mathbf{y}\|_2 \tag{52}$$

Combining Equations 51 and 52, we find

$$\frac{\|\mathbf{\Delta y}\|_2}{\|\mathbf{x}\|_2} \leq \gamma_w^{(\tilde{z})}. \tag{53}$$

Now, notice that $\mathbf{\Delta P}$ is exactly $\frac{1}{\mathbf{x}^\top\mathbf{x}}\mathbf{\Delta y}\mathbf{x}^\top$; thus,

$$(\mathbf{P} + \mathbf{\Delta P})\mathbf{x} = (\mathbf{P} + \frac{1}{\mathbf{x}^\top\mathbf{x}}\mathbf{\Delta y}\mathbf{x}^\top)\mathbf{x} = \mathbf{P}\mathbf{x} + \frac{\mathbf{x}^\top\mathbf{x}}{\mathbf{x}^\top\mathbf{x}}\mathbf{\Delta y} = \mathbf{y} + \mathbf{\Delta y}$$

We can compute the Frobenius norm of $\mathbf{\Delta P}$ by using $\mathbf{\Delta P}_{ij} = \frac{1}{\|\mathbf{x}\|_2^2}\mathbf{\Delta y}_i\mathbf{x}_j$.

$$\|\mathbf{\Delta P}\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{m}\left(\frac{1}{\|\mathbf{x}\|_2^2}\mathbf{\Delta y}_i\mathbf{x}_j\right)^2\right)^{1/2} = \frac{\|\mathbf{\Delta y}\|_2}{\|\mathbf{x}\|_2}$$

Finally, using Equation 53, we find $\|\mathbf{\Delta P}\|_F \leq \gamma_w^{(\tilde{z})}$. □

Lemma A.2. *Consider applying a sequence of transformations in the set $\{\mathbf{P}_j\}_{j=1}^r \subset \mathbb{R}^{m\times m}$ to $\mathbf{x} \in \mathbb{R}^m$, where $\mathbf{P}_j$'s are all Householder transformations and where we will assume that $r\gamma_w^{(\tilde{z})} < \frac{1}{2}$. Let $\mathbf{y} = \mathbf{P}_r\mathbf{P}_{r-1}\cdots\mathbf{P}_1\mathbf{x} = \mathbf{Q}^\top\mathbf{x}$. Then, $\hat{\mathbf{y}} = (\mathbf{Q} + \mathbf{\Delta Q})^\top\mathbf{x}$, where*

$$\|\mathbf{\Delta Q}\|_F \leq r\gamma_w^{(\tilde{z})}, \quad \|\mathbf{\Delta y}\|_2 \leq r\gamma_w^{(\tilde{z})}\|\mathbf{y}\|_2. \tag{54}$$

In addition, if we let $\hat{\mathbf{y}} = \mathbf{Q}^\top(\mathbf{x} + \mathbf{\Delta x})$*, then*

$$\|\mathbf{\Delta x}\|_2 \leq r\gamma_w^{(\tilde{z})}\|\mathbf{x}\|_2. \tag{55}$$

Proof. As was for the proof for Lemma A.1, we know $\mathbf{\Delta Q}^\top = \frac{1}{\|\mathbf{x}\|_2^2}\mathbf{\Delta y x}^\top$. Recall that the HQR factorization applies a series of Householder transformations on $\mathbf{A}$ to form $\mathbf{R}$, and applies the same series of Householder transformations in reverse order to $\mathbf{I}$ to form $\mathbf{Q}$. Therefore, it is appropriate to assume that $\mathbf{x}$ is exact in this proof, and we form a forward bound on $\hat{\mathbf{y}}$. However, we can still easily switch between forward and backward errors in the following way:

$$\begin{aligned}\hat{\mathbf{y}} = \mathbf{y} + \mathbf{\Delta y} = \mathbf{Q}^\top(\mathbf{x} + \mathbf{\Delta x}) = (\mathbf{Q} + \mathbf{\Delta Q})^\top\mathbf{x},\\ \mathbf{\Delta y} = \mathbf{\Delta Q}^\top\mathbf{x} = \mathbf{Q}^\top\mathbf{\Delta x}.\end{aligned}$$

In addition, we can switch between $\mathbf{\Delta y}$ and $\mathbf{\Delta x}$ by using the fact that $\|\mathbf{Q}\|_2 = 1$.

Error bound for $\|\mathbf{\Delta y}\|_2$ We will first find $\|\mathbf{\Delta Q}\|_2$, where this is NOT the forward error from forming $\mathbf{Q}$ with Householder transformations, but rather a backward error in accumulating Householder transformations. From Lemma A.1, we have$\|\mathbf{\Delta P}\|_F \leq \gamma_w^{(\tilde{z})} = \gamma_w^{(\tilde{z})}\|\mathbf{P}\|_2$ for any Householder transformation $\mathbf{P} \in \mathbb{R}^{m\times m}$, where $\tilde{z} = 6d + 6z + 13$ and $d = \lfloor\frac{(m-1)u_s}{u_w}\rfloor$, $z \in \{1, 2\}$. Therefore, this applies to the sequence of $\mathbf{P}_i$'s that form $\mathbf{Q}$ as well.

We will now use Lemma 3.7 from [1] to bound $\mathbf{\Delta Q}$:

$$\begin{aligned}\mathbf{\Delta Q}^\top = \left(\hat{\mathbf{Q}} - \mathbf{Q}\right)^\top = \prod_{i=r}^{1}(\mathbf{P}_i + \mathbf{\Delta P}_i) - \prod_{i=r}^{1}\mathbf{P}_i,\\ \|\mathbf{\Delta Q}\|_F = \|\mathbf{\Delta Q}^\top\|_F = \left\|\prod_{i=r}^{1}(\mathbf{P}_i + \mathbf{\Delta P}_i) - \prod_{i=r}^{1}\mathbf{P}_i\right\|_F,\\ \leq \left(\prod_{i=r}^{1}(1 + \gamma_w^{(\tilde{z})}) - 1\right)\prod_{i=r}^{1}\|\mathbf{P}_i\|_2 = \prod_{i=r}^{1}(1 + \gamma_w^{(\tilde{z})}) - 1.\end{aligned}$$

The last equality results from the orthogonality of Householder matrices.

Consider the constant, $(1 + \gamma_w^{(\tilde{z})})^r - 1$. From the very last rule in Lemma 2.2, we can generalize the following:

$$(1 + \gamma_w^{(\tilde{z})})^r = (1 + \gamma_w^{(\tilde{z})})^{r-2}(1 + \gamma_w^{(\tilde{z})})(1 + \gamma_w^{(\tilde{z})}) \leq (1 + \gamma_w^{(\tilde{z})})^{r-2}(1 + \gamma_w^{(2\tilde{z})}) \leq \cdots \leq (1 + \gamma_w^{(r\tilde{z})}).$$

So, our quantity of interest can be bound by $(1 + \gamma_w^{(\tilde{z})})^r - 1 \leq \gamma_w^{(r\tilde{z})}$.

Now we will use the following equivalent algebraic inequalities to get the final result.

$$0 < a < b < 1 \Leftrightarrow 1 - a > 1 - b \Leftrightarrow \frac{1}{1-a} < \frac{1}{1-b} \Leftrightarrow \frac{a}{1-a} < \frac{b}{1-b} \tag{56}$$

In addition, we assume $r\gamma_w^{(\tilde{z})} < \frac{1}{2}$, such that

$$\begin{aligned}(1+\gamma_w^{(\tilde{z})})^r - 1 \leq \gamma_w^{(r\tilde{z})} &= \frac{r\tilde{z}u_w}{1-r\tilde{z}u_w} \qquad \text{(by definition)} \\ &\leq \frac{r\gamma_w^{(\tilde{z})}}{1-r\gamma_w^{(\tilde{z})}}, \text{ since } r\tilde{z}u_w < r\gamma_w^{(\tilde{z})} \qquad \text{(by Equation 56)} \\ &\leq 2r\gamma_w^{(\tilde{z})} \qquad \left(\text{since } r\gamma_w^{(\tilde{z})} < \frac{1}{2} \text{ implies } \frac{1}{1-r\gamma_w^{(\tilde{z})}} < 2\right) \\ &= r\tilde{\gamma}_w^{(\tilde{z})},\end{aligned}$$

where $\tilde{\gamma^{(m)}} := \frac{cmu}{1-cmu}$ for some small integer, c. If we had started with $(1+\tilde{\gamma}_w^{(\tilde{z})})^r - 1$, we can still find $r\tilde{\gamma}_w^{(\tilde{z})}$ assuming that $2c$ is still a small integer. In conclusion, we have

$$(1+\gamma_w^{(\tilde{z})})^r - 1 \leq r\tilde{\gamma}_w^{(\tilde{z})},$$

which results in the bound for $\mathbf{\Delta Q}$ as shown in Equation 54, $\|\mathbf{\Delta Q}\|_2 \leq \|\mathbf{\Delta Q}\|_F \leq r\gamma_w^{(\tilde{z})}$.

Next, we bound $\|\mathbf{\Delta y}\|_2 = \|\mathbf{\Delta Q x}\|_2 \leq \|\mathbf{\Delta Q}\|_2\|\mathbf{x}\|_2 \leq r\tilde{\gamma}_w^{(\tilde{z})}\|\mathbf{x}\|_2$.

Bound for $\|\mathbf{\Delta x}\|_2$ We use the above result,

$$\|\mathbf{\Delta x}\|_2 = \|\mathbf{Q\Delta y}\|_2\| \leq \|\mathbf{Q}\|_2\|\mathbf{\Delta y}\|_2 = \|\mathbf{\Delta y}\|_2 \leq r\tilde{\gamma}_w^{(\tilde{z})}\|\mathbf{x}\|_2. \tag{57}$$

While $r\gamma^{(k)} = r\frac{ku}{1-ku} < \frac{rku}{1-rku} = \gamma^{(rk)}$ holds true when $r > 0$ and $rku < 1$ are satisfied, the strict inequality implies that $r\gamma^{(k)}$ is a tighter bound than $\gamma^{(rk)}$. However, $\gamma^{(rk)}$ is easier to work with using the rules in Lemma 2.1. □

A.3.1 Proof for Theorem 3.5

First, we use Lemma A.2 directly on columns of $\mathbf{A}$ and $\mathbf{I}_{m\times n}$ to get a result for columns of $\hat{\mathbf{R}}$ and $\hat{\mathbf{Q}}$. We will use the maximum number of transformations and the length of the longest vector on to which we perform a Householder transformation, that is, n transformations of vectors of length m. For j in $\{1, \cdots, n\}$, the j^{th} column of $\mathbf{R}$ and $\mathbf{Q}$ are the results of j Householder transformations on $\mathbf{A}$ and $\mathbf{I}$:

$$\|\mathbf{\Delta Q}[:,j]\|_2 \leq j\tilde{\gamma}_w^{(\tilde{z})}\|\hat{e}_j\|_2 < \tilde{\gamma}_w^{(j\tilde{z})}, \tag{58}$$

$$\|\mathbf{\Delta R}[:,j]\|_2 \leq j\tilde{\gamma}_w^{(\tilde{z})}\|\mathbf{A}[:,j]\|_2 < \tilde{\gamma}_w^{(j\tilde{z})}\|\mathbf{A}[:,j]\|_2. \tag{59}$$

Finally, we relate columnwise 2-norms to matrix Frobenius norms. It is straightforward to see the result for the $\mathbf{Q}$ factor,

$$\|\mathbf{\Delta Q}\|_F = \left(\sum_{j=1}^{n}\|\mathbf{\Delta Q}[:,j]\|_2^2\right)^{1/2} \leq \left(\sum_{j=1}^{n}(j\tilde{\gamma}_w^{(\tilde{z})})^2\|\hat{e}_j\|_2^2\right)^{1/2} \leq n^{3/2}\tilde{\gamma}_w^{(\tilde{z})}. \tag{60}$$

Note that we bound $\sum_{j=1}^{n} j^2$ by n^3, but the summation is actually exactly $\frac{n(n+1)(2n+1)}{6}$. Therefore, a tighter bound would replace $n^{3/2}$ with $\left(\frac{n(n+1)(2n+1)}{6}\right)^{1/2}$.

We can bound the $\mathbf{R}$ factor in a similar way,

$$\|\boldsymbol{\Delta}\mathbf{R}\|_F = \left(\sum_{j=1}^{n} \|\boldsymbol{\Delta}\mathbf{R}[:,j]\|_2^2\right)^{1/2} \leq \left(\sum_{j=1}^{n} (j\tilde{\gamma}_w^{(\tilde{z})})^2 \|\mathbf{A}[:,j]\|_2^2\right)^{1/2} \leq n\tilde{\gamma}_w^{(\tilde{z})}\|\mathbf{A}\|_F. \tag{61}$$

Obtaining the backward error from the HQR factorization,

$$\boldsymbol{\Delta}\mathbf{A} = \mathbf{A} - \hat{\mathbf{Q}}\hat{\mathbf{R}} = \mathbf{A} - \mathbf{Q}\hat{\mathbf{R}} + \mathbf{Q}\hat{\mathbf{R}} - \hat{\mathbf{Q}}\hat{\mathbf{R}} = \mathbf{Q}\boldsymbol{\Delta}\mathbf{R} + \boldsymbol{\Delta}\mathbf{Q}\hat{\mathbf{R}}.$$

A columnwise result for $\hat{\mathbf{A}}$ is shown by

$$\begin{aligned}
\|\boldsymbol{\Delta}\mathbf{A}[:,j]\|_2 &= \|(\mathbf{Q}\boldsymbol{\Delta}\mathbf{R} + \boldsymbol{\Delta}\mathbf{Q}\hat{\mathbf{R}})[:,j]\|_2, \\
&\leq \|\mathbf{Q}\boldsymbol{\Delta}\mathbf{R}[:,j]\|_2 + \|\boldsymbol{\Delta}\mathbf{Q}\hat{\mathbf{R}}[:,j]\|_2, \\
&\leq \|\boldsymbol{\Delta}\mathbf{R}[:,j]\|_2 + \|\boldsymbol{\Delta}\mathbf{Q}\|_2\|\hat{\mathbf{R}}[:,j]\|_2, \\
&\leq \|\boldsymbol{\Delta}\mathbf{R}[:,j]\|_2 + \|\boldsymbol{\Delta}\mathbf{Q}\|_F\|(\mathbf{R} + \boldsymbol{\Delta}\mathbf{R})[:,j]\|_2, \\
&\leq j\tilde{\gamma}_w^{(\tilde{z})}\|\mathbf{A}[:,j]\|_2 + n^{3/2}\tilde{\gamma}_w^{(\tilde{z})}\|(\mathbf{Q}^\top\mathbf{A} + \boldsymbol{\Delta}\mathbf{R})[:,j]\|_2, \\
&\leq j\tilde{\gamma}_w^{(\tilde{z})}\|\mathbf{A}[:,j]\|_2 + n^{3/2}\tilde{\gamma}_w^{(\tilde{z})}\left(\|\mathbf{A}[:,j]\|_2 + \|\boldsymbol{\Delta}\mathbf{R}[:,j]\|_2\right), \\
&\leq \left(j\tilde{\gamma}_w^{(\tilde{z})} + n^{3/2}\tilde{\gamma}_w^{(\tilde{z})}(1 + j\tilde{\gamma}_w^{(\tilde{z})})\right)\|\mathbf{A}[:,j]\|_2, \\
&= n^{3/2}\tilde{\gamma}_w^{(\tilde{z})}\|\mathbf{A}[:,j]\|_2,
\end{aligned}$$

where we assume $n\tilde{\gamma}_w^{(\tilde{z})} \ll 1$ and where the last equality sweeps all non-leading order terms into the arbitrary constant c within the definition of $\tilde{\gamma}$,

$$\|\boldsymbol{\Delta}\mathbf{A}\|_F = \left(\sum_{j=1}^{n} \|\boldsymbol{\Delta}\mathbf{A}[:,j]\|_2^2\right)^{1/2} \leq \left(\sum_{j=1}^{n} (n^{3/2}\tilde{\gamma}_w^{(\tilde{z})})^2 \|\mathbf{A}[:,j]\|_2^2\right)^{1/2} \leq n^{3/2}\tilde{\gamma}_w^{(\tilde{z})}\|\mathbf{A}\|_F. \tag{62}$$

References

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

[2] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, pp. 395–416, Dec 2007.

[3] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed precision training," in *International Conference on Learning Representations*, 2018.

[4] M. Fagan, J. Schlachter, K. Yoshii, S. Leyffer, K. Palem, M. Snir, S. M. Wild, and C. Enz, "Overcoming the power wall by exploiting inexactness and emerging COTS architectural features: Trading precision for improving application quality," in *2016 29th IEEE International System-on-Chip Conference (SOCC)*, pp. 241–246, Sep. 2016.

[5] A. Haidar, A. Abdelfattah, M. Zounon, P. Wu, S. Pranesh, S. Tomov, and J. Dongarra, *The Design of Fast and Energy-Efficient Linear Solvers: On the Potential of Half-Precision Arithmetic and Iterative Refinement Techniques*, pp. 586–600. June 2018.

[6] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham, "Harnessing GPU tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, SC '18, (Piscataway, NJ, USA), pp. 47:1–47:11, IEEE Press, 2018.

[7] A. Abdelfattah, S. Tomov, and J. Dongarra, "Fast batched matrix multiplication for small sizes using half-precision arithmetic on GPUs," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 111–122, May 2019.

[8] M. Courbariaux, Y. Bengio, and J.-P. David, "Training deep neural networks with low precision multiplications," *arXiv preprint, arXiv:1412.7024*, 2014.

[9] M. Courbariaux, J.-P. David, and Y. Bengio, "Low precision storage for deep learning," *arXiv preprint arXiv:1412.7024*, 2014.

[10] J. Appleyard and S. Yokim, "Programming Tensor Cores in CUDA 9," 2017.

[11] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benini, "A transprecision floating-point platform for ultra-low power computing," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1051–1056, March 2018.

[12] 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.

[13] A. S. Householder, "Unitary triangularization of a nonsymmetric matrix," *Journal of the ACM (JACM)*, vol. 5, no. 4, pp. 339–342, 1958.

[14] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. J. Dongarra, J. Du Croz, S. Hammarling, A. Greenbaum, A. McKenney, and D. Sorensen, *LAPACK Users' Guide (Third Ed.)*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1999; also available online from http://www.netlib.org.

[15] 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.

[16] G. Ballard, J. W. Demmel, L. Grigori, M. Jacquelin, H. Diep Nguyen, and E. Solomonik, "Reconstructing Householder vectors from tall-skinny QR," vol. 85, pp. 1159–1170, 05 2014.

[17] E. Kao, V. Gadepally, M. Hurley, M. Jones, J. Kepner, S. Mohindra, P. Monticciolo, A. Reuther, S. Samsi, W. Song, D. Staheli, and S. Smith, "Streaming graph challenge: Stochastic block partition," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–12, Sep. 2017.

[18] Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and van der Vorstm H., *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 1st ed., 2000.

[19] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *Kdd*, vol. 96, pp. 226–231, 1996.