

ROUNDING ERROR ANALYSIS OF MIXED PRECISION HOUSEHOLDER QR ALGORITHMS

L. MINAH YANG, ALYSON FOX, AND GEOFFREY SANDERS

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 [12] and demonstrate its practicality by applying the analysis to various Householder QR Algorithms.

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 may yield insufficient accuracy when using low precision storage and arithmetic. 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 [?] or training dense neural networks [17], 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 [?]. Modern accelerators (e.g., GPUs, Knights Landing, or Xeon Phi) are able to achieve this factor or better speedup improvements. Several examples include: (i) $2\text{--}4\times$ speedup in solving dense large linear equations [10, 11], (ii) $12\times$ speedup in training dense neural networks, and (iii) $1.2\text{--}10\times$ speedup in small batched dense matrix multiplication [1] (up to $26\times$ for batches of tiny matrices). Training deep artificial neural networks by employing lower precision arithmetic to various tasks such as multiplication [5] and storage [6] 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 [2]. 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 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.

42 this work, we develop a framework for deterministic mixed precision rounding error analysis, and
 43 explore half-precision Householder QR factorization (HQR) algorithms for data and graph analysis
 44 applications. QR factorization is known to provide a backward stable solution to the linear least
 45 squares problem and thus, is ideal for mixed precision.

46 However, additional analysis is needed as the additional round-off error will effect orthogonality,
 47 and thus the accuracy of the solution. Here, we focus on analyzing specific algorithms in a specific
 48 set of types (IEEE754 half (fp16), single (fp32, and double(fp64)), but the framework we develop
 49 could be used on different algorithms or different floating point types (such as bfloat16 in [20]).

50 This work discusses several aspects of using mixed precision arithmetic: (i) error analysis that
 51 can more accurately describe mixed precision arithmetic than existing analyses, (ii) algorithmic de-
 52 sign that is more resistant against lower numerical stability associated with lower precision types,
 53 and (iii) an example where mixed precision implementation performs as sufficiently as double-
 54 precision implementations. Our key findings are that the new mixed precision error analysis pro-
 55 duces tighter error bounds, that some block QR algorithms by Demmel et al. [8] are able to operate
 56 in low precision more robustly than non-block techniques, and that some small-scale benchmark
 57 graph clustering problems can be successfully solved with mixed precision arithmetic.

58 **2. Background: Build up to rounding error analysis for inner products.** In this
 59 section, we introduce the basic motivations and tools for mixed precision rounding error analysis
 60 needed for the *QR factorization*. A matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ for $m \geq n$ can be written as

$$61 \quad \mathbf{A} = \mathbf{Q}\mathbf{R}, \quad \mathbf{Q} \in \mathbb{R}^{m \times m}, \quad \mathbf{R} \in \mathbb{R}^{m \times n},$$

62 where \mathbf{Q} is orthogonal, $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}_{m \times m}$, and \mathbf{R} is upper trapezoidal. The above formulation is a
 63 *full* QR factorization, whereas a more efficient *thin* QR factorization results in $\mathbf{Q}_1 \in \mathbb{R}^{m \times n}$ and
 64 $\mathbf{R}_1 \in \mathbb{R}^{n \times n}$, that is

$$65 \quad \mathbf{A} = \mathbf{Q}\mathbf{R} = [\mathbf{Q}_1 \quad \mathbf{Q}_2] \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0}_{m-n \times n} \end{bmatrix} = \mathbf{Q}_1 \mathbf{R}_1.$$

66 If \mathbf{A} is full rank then the columns of \mathbf{Q}_1 are orthonormal (i.e. $\mathbf{Q}_1^\top \mathbf{Q}_1 = \mathbf{I}_{n \times n}$) and \mathbf{R}_1 is upper
 67 triangular. In many applications, computing the *thin* decomposition requires less computation and
 68 is sufficient in performance. While important definitions are stated explicitly in the text, Table 1
 69 serves to establish basic notation.

Symbol(s)	Definition(s)	Section(s)
\mathbf{x}, \mathbf{A}	Vector, matrix	2
\mathbf{Q}	Orthogonal factor $\mathbf{A} \in \mathbb{R}^{m \times n}$: m -by- m (full) or m -by- n (thin)	2
\mathbf{R}	Upper triangular or trapezoidal factor of $\mathbf{A} \in \mathbb{R}^{m \times n}$: m -by- n (full) or n -by- n (thin)	2
$\text{fl}(\mathbf{x}), \hat{\mathbf{x}}$	Quantity \mathbf{x} calculated from floating point operations	2.1
b, t, μ, η	Base/precision/mantissa/exponent bits	2.1
k	Number of successive FLOPs	2.1
$u^{(q)}$	Unit round-off for precision t_q and base b_q : $\frac{1}{2}b_q^{1-t_q}$	2.1
$\delta^{(q)}$	Quantity bounded by: $ \delta^{(q)} < u^{(q)}$	2.1
$\gamma_k^{(q)}, \theta_k^{(q)}$	$\frac{ku^{(q)}}{1-ku^{(q)}}$, Quantity bounded by: $ \theta_k^{(q)} \leq \gamma_k^{(q)}$	2.1

TABLE 1
Basic definitions

Subsection 2.1 introduces basic concepts for rounding error analysis, and Subsection 2.2 exemplifies the need for mixed precision rounding error analysis using the inner product.

2.1. Basic rounding error analysis of floating point operations. We use and analyze the 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

$$(2.1) \quad y = \pm \mu \times b^{\eta-t},$$

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, the sign, significand, and exponent identifies a unique number within that system. 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 (FLOPs). We adopt the rounding error analysis tools described in [12], which allow a relatively simple framework for formulating error bounds for complex linear algebra operations. A short analysis of FLOPs (see Theorem 2.2 [12]) shows that the relative error is controlled by the unit round-off, $u := \frac{1}{2}b^{1-t}$ in uniform precision settings. In mixed precision settings we denote the higher precision unit round-off with $u^{(h)}$ (h for high) and the lower precision unit round-off with $u^{(l)}$ (l for low).

Name	b	t	# of exponent bits	η_{\min}	η_{\max}	unit round-off u
fp16 (IEEE754 half)	2	11	5	-15	16	4.883e-04
fp32 (IEEE754 single)	2	24	8	-127	128	5.960e-08
fp64 (IEEE754 double)	2	53	11	-1023	1024	1.110e-16

TABLE 2
IEEE754 formats and their primary attributes.

Let ‘op’ be any basic operation from the set $\text{OP} = \{+, -, \times, \div\}$ and let $x, y \in \mathbb{R}$. 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:

$$(2.2) \quad \text{fl}(x \text{ op } y) = (1 + \delta)(x \text{ op } y), \quad |\delta| \leq u, \quad \text{op} \in \{+, -, \times, \div\}.$$

We use (2.2) as a building block in accumulating errors from successive FLOPs. For example, consider computing $x + y + z$, where $x, y, z \in \mathbb{R}$ with a machine that can only compute one operation at a time. 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,

$$(2.3) \quad \text{fl}(x + y + z) = (1 + \tilde{\delta})(1 + \delta)(x + y) + (1 + \tilde{\delta})z.$$

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.

103 LEMMA 2.1 (Lemma 3.1 [12]). Let $|\delta_i| < u$ and $\rho_i \in \{-1, +1\}$, for $i = 1, \dots, k$ and $ku < 1$.
 104 Then,

$$105 \quad (2.4) \quad \prod_{i=1}^k (1 + \delta_i)^{\rho_i} = 1 + \theta_k, \quad \text{where} \quad |\theta_k| \leq \frac{ku}{1 - ku} =: \gamma_k.$$

106 We also use

$$107 \quad \tilde{\gamma}_k = \frac{cku}{1 - cku},$$

108 where $c > 0$ is a small integer and further extend this to θ so that $|\tilde{\theta}_k| \leq \tilde{\gamma}_k$.

109 In other words, θ_k represents the accumulation of rounding errors from k successive operations, and
 110 it is bounded by γ_k . Allowing θ_k 's to be any arbitrary value within the corresponding γ_k bounds
 111 further aids in keeping a clear, simple error analysis. Applying this lemma to our example of adding
 112 three numbers results in

$$113 \quad (2.5) \quad \text{fl}(x + y + z) = (1 + \tilde{\delta})(1 + \delta)(x + y) + (1 + \tilde{\delta})z = (1 + \theta_2)(x + y) + (1 + \theta_1)z.$$

114 Since $|\theta_1| \leq \gamma_1 < \gamma_2$, we can further simplify (2.5) to

$$115 \quad (2.6) \quad \text{fl}(x + y + z) = (1 + \tilde{\theta}_2)(x + y + z), \quad \text{where} \quad |\tilde{\theta}_2| \leq \gamma_2,$$

116 at the cost of a slightly larger upper bound. Typically, error bounds formed in the fashion of (2.6)
 117 are converted to relative errors in order to put the error magnitudes in perspective. The relative
 118 error bound for our example is

$$119 \quad |(x + y + z) - \text{fl}(x + y + z)| \leq \gamma_2 |x + y + z|$$

120 when we assume $x + y + z \neq 0$.

121 Although Lemma 2.1 requires $ku < 1$, we actually need $ku < \frac{1}{2}$ to maintain a meaningful
 122 relative error bound as this assumption implies $\gamma_k < 1$ and guarantees a relative error below 100%.
 123 Since higher precision floating points have smaller unit round-off values, they can tolerate more
 124 successive FLOPs than lower precision floating points before reaching $\gamma_m = 1$. Table 3 shows the
 125 maximum number of successive floating point operations that still guarantees a relative error below
 100% for various floating point types. Thus, accumulated rounding errors in lower precision types

precision	$\tilde{k} = \arg \max_k (\gamma_k \leq 1)$
FP16	512
FP32	$\approx 4.194\text{e}06$
FP64	$\approx 2.252\text{e}15$

TABLE 3
Upper limits of meaningful relative error bounds in the $\gamma^{(k)}$ notation.

126 can lead to an instability with fewer operations in comparison to higher precision types and prompts
 127 us to evaluate whether existing algorithms can be naively adapted for mixed precision arithmetic.

2.2. Rounding Error Example for the Inner Product. We now consider computing the inner product of two vectors to clearly illustrate how this situation restricts rounding error analysis in fp16. An error bound for an inner product of m -length vectors is

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

as shown in [12]. 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 a forward error: $|\|\mathbf{x}\|_2^2 - \text{fl}(\|\mathbf{x}\|_2^2)| \leq \gamma_m \|\mathbf{x}\|_2^2$. Since vectors of length m accumulate rounding errors that are bounded by γ_m , the dot products of vectors computed in fp16 already face a 100% relative error bound in the worst-case scenario ($\gamma_{512}^{\text{fp16}} = 1$).

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 `alg. 1`, which was proven to be a faithful simulation in [14], for every FLOP (multiplication and addition for the dot product). The relative error in this experiment is formulated as the LHS in Equation 2.7 divided by $|\mathbf{x}|^\top |\mathbf{y}|$ and all operations outside of calculating $\text{fl}(\mathbf{x}^\top \mathbf{y})$ are executed by casting up to fp64 and using fp64 arithmetic. Table 4 shows some statistics from computing the relative error for simulated half precision dot products of 512-length random vectors. We see that the inner products

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 4

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.

of vectors sampled from the standard normal distribution have backward relative errors that do not deviate much from the unit round-off ($\mathcal{O}(1\text{e-}4)$), whereas the vectors sampled from the uniform distribution tend to accumulate larger errors on average ($\mathcal{O}(1\text{e-}3)$). Even so, the theoretical upper error bound of 100% is far too pessimistic as the maximum relative error does not even meet 2% in this experiment. Recent work in developing probabilistic bounds on rounding errors of floating point operations (see [13, 16]) have shown that the inner product relative backward error for the conditions used for this experiment is bounded by $5.466\text{e-}2$ with probability 0.99.

Most importantly, we need error analysis that allows flexibility in precision in order to better our understanding of the impact of rounding errors on computations done on emerging hardware (i.e. GPUs) that support mixed precision. We start by introducing some additional rules from [12] that build on Lemma 2.1 in Lemma 2.2. These rules summarize how to accumulate errors represented by θ 's and γ 's in a *uniform precision* setting.

LEMMA 2.2. *For any positive integer k , let θ_k denote a quantity bounded according to $|\theta_k| \leq \frac{k_u}{1-k_u} =: \gamma_k$. The following relations hold for positive integers i, j , and nonnegative integer k .*

Algorithm 1: $\mathbf{z}^{(\text{fp16})} = \text{simHalf}(f, \mathbf{x}^{(\text{fp16})}, \mathbf{y}^{(\text{fp16})})$. Simulate function $f \in \text{OP} \cup \{\text{dot_product}\}$ in half precision arithmetic given input variables \mathbf{x}, \mathbf{y} . Function **castup** converts fp16 to fp32, and **castdown** converts fp32 to fp16 by rounding to the nearest half precision float.

Input: $\mathbf{x}^{(\text{fp16})}, \mathbf{y}^{(\text{fp16})} \in \mathbb{F}_{\text{fp16}}^m$, $f : \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}^n$
Output: $\text{fl}(f(\mathbf{x}^{(\text{fp16})}, \mathbf{y}^{(\text{fp16})})) \in \mathbb{F}_{\text{fp16}}^n$
1 $\mathbf{x}^{(\text{fp32})}, \mathbf{y}^{(\text{fp32})} \leftarrow \text{castup}([\mathbf{x}^{(\text{fp16})}, \mathbf{y}^{(\text{fp16})}])$
2 $\mathbf{z}^{(\text{fp32})} \leftarrow \text{fl}(f(\mathbf{x}^{(\text{fp32})}, \mathbf{y}^{(\text{fp32})}))$
3 $\mathbf{z}^{(\text{fp16})} \leftarrow \text{castdown}(\mathbf{z}^{(\text{fp32})})$
4 **return** $\mathbf{z}^{(\text{fp16})}$

163 *Arithmetic operations between θ_k 's:*

164 (2.8) $(1 + \theta_k)(1 + \theta_j) = (1 + \theta_{k+j}) \quad \text{and} \quad \frac{1 + \theta_k}{1 + \theta_j} = \begin{cases} 1 + \theta_{k+j}, & j \leq k \\ 1 + \theta_{k+2j}, & j > k \end{cases}$

165 *Operations on γ 's:*

166 $\gamma_k \gamma_j \leq \gamma_{\min(k,j)}, \quad \text{for } \max_{(j,k)} u \leq \frac{1}{2},$

167 $n \gamma_k \leq \gamma_{nk}, \quad \text{for } n \leq \frac{1}{uk},$

168 $\gamma_k + u \leq \gamma_{k+1},$

169 $\gamma_k + \gamma_j + \gamma_k \gamma_j \leq \gamma_{k+j}.$

171 Let us define a mixed precision setting that is similar to the TensorCore Fused Multiply-Add
172 (FMA) block but works at the level of a dot product. The main difference lies in that the FMA
173 block in TensorCore is for matrix-matrix products (level-3 BLAS) for fp16 and fp32, but our ad hoc
174 mixed precision inner product FMA defined in [Assumption 2.3](#) is a level-2 BLAS operation to work
175 on any two different precision types. Although our analysis concerns accuracy and stability and
176 leaves out timing results of various hardware, we add a general timing statement to [Assumption 2.3](#)
177 that is analogous to that of TensorCore: the mixed precision FMA inner product performs at least
178 2 times faster than the inner product in the higher precision. Note that the TensorCore FMA
179 block performs matrix-matrix multiply and accumulate up to 8 times faster than fp32, and up to 16
180 times faster than fp64 (see), and our ad hoc timing assumption is in conservative in comparison.
181 Nonetheless, this gives a vague insight into the trade-offs between speediness and accuracy from
182 some mixed precision computations. The full precision multiplication in [Assumption 2.3](#) is exact
183 when the low precision type is fp16 and the high precision type of fp32 due to their precisions
184 and exponent ranges. As a quick proof, consider $x^{(\text{fp16})} = \pm \mu_x 2^{\eta_x - 11}$, $y^{(\text{fp16})} = \pm \mu_y 2^{\eta_y - 11}$ where
185 $\mu_x, \mu_y \in [0, 2^{11} - 1]$ and $\eta_x, \eta_y \in [-15, 16]$, and note that the significand and exponent ranges for
186 fp32 are $[0, 2^{24} - 1]$ and $[-127, 128]$. Then the product in full precision is

187 $x^{(\text{fp16})} y^{(\text{fp16})} = \pm \mu_x \mu_y 2^{\eta_x + \eta_y + 2 - 24},$

188 where $\mu_x \mu_y \in [0, (2^{11} - 1)^2] \subseteq [0, 2^{24} - 1]$ and $\eta_x + \eta_y + 2 \in [-28, 34] \subseteq [-127, 128]$, and therefore

is exact. Thus, the summation and the final cast down operations are the only sources of rounding error in this inner product scheme.

ASSUMPTION 2.3. *Let l and h each denote low and high precision types with unit round-off values $u^{(l)}$ and $u^{(h)}$, where $1 \gg u^{(l)} \gg u^{(h)} > 0$. Consider an FMA operation for inner products that take vectors stored in precision l , compute products in full precision, and sum the products in precision h . Finally, the result is then cast back down to precision l . Furthermore, we expect this procedure to be approximately twice as fast as if it were done entirely in the higher precision, and about the same as if it were done entirely in the lower precision.*

We now analyze the rounding error for the inner product scheme described in [Assumption 2.3](#). Let $\mathbf{x}^{(l)}, \mathbf{y}^{(l)}$ be m -length vectors stored in some low precision float, \mathbb{F}_l , s_k be the exact k^{th} partial sum, and \hat{s}_k be s_k computed with FLOPs. Then,

$$\begin{aligned}\hat{s}_1 &= \text{fl}(\mathbf{x}[1]\mathbf{y}[1]) = \mathbf{x}[1]\mathbf{y}[1], \\ \hat{s}_2 &= \text{fl}(\hat{s}_1 + \mathbf{x}[2]\mathbf{y}[2]) = (\mathbf{x}[1]\mathbf{y}[1] + \mathbf{x}[2]\mathbf{y}[2]) (1 + \delta_1^{(h)}), \\ \hat{s}_3 &= \text{fl}(\hat{s}_2 + \mathbf{x}[3]\mathbf{y}[3]) = \left[(\mathbf{x}[1]\mathbf{y}[1] + \mathbf{x}[2]\mathbf{y}[2]) (1 + \delta_1^{(h)}) + \mathbf{x}[3]\mathbf{y}[3] \right] (1 + \delta_2^{(h)}).\end{aligned}$$

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

$$(2.9) \quad \hat{s}_m = (\mathbf{x}[1]\mathbf{y}[1] + \mathbf{x}[2]\mathbf{y}[2]) \prod_{k=1}^{m-1} (1 + \delta_k^{(h)}) + \sum_{i=3}^n \mathbf{x}[i]\mathbf{y}[i] \left(\prod_{k=i-1}^{m-1} (1 + \delta_k^{(h)}) \right).$$

Using Lemma 2.1, we further simplify and form componentwise backward errors with

$$(2.10) \quad \text{fl}(\mathbf{x}^\top \mathbf{y}) = (\mathbf{x} + \Delta \mathbf{x})^\top \mathbf{y} = \mathbf{x}^\top (\mathbf{y} + \Delta \mathbf{y}) \quad \text{for } |\Delta \mathbf{x}| \leq \gamma_{m-1}^{(h)} |\mathbf{x}|, \quad |\Delta \mathbf{y}| \leq \gamma_{m-1}^{(h)} |\mathbf{y}|.$$

If we then cast down this result, the forward errors are

$$(2.11) \quad \text{fl}(\mathbf{x}^\top \mathbf{y}) = (\mathbf{x} + \tilde{\Delta} \mathbf{x})^\top \mathbf{y} = \mathbf{x}^\top (\mathbf{y} + \tilde{\Delta} \mathbf{y}),$$

where $|\tilde{\Delta} \mathbf{x}| \leq ((1 + u^{(l)})(1 + \gamma_{m-1}^{(h)}) - 1) |\mathbf{x}|$ and $|\tilde{\Delta} \mathbf{y}| \leq ((1 + u^{(l)})(1 + \gamma_{m-1}^{(h)}) - 1) |\mathbf{y}|$.

Without any analysis, we expect that a mixed precision inner product described in [Assumption 2.3](#) should perform better than if it were computed entirely in the lower precision and worse than if it were computed entirely in the high precision. This hypothesis is proven correct since,

$$\gamma_m^{(l)} > ((1 + u^{(l)})(1 + \gamma_{m-1}^{(h)}) - 1) > \gamma_m^{(h)},$$

where the lower and upper bounds are derived from the uniform precision error bound from (2.7). Our analysis in (2.11) shows us that the most of the errors are from the higher precision summation, $\gamma_{m-1}^{(h)}$, and the cast down operation adds $u^{(l)}$. The impact of the cast down step is measured relative to the length of the vector, m , and the disparity in the two precisions, $M_{l,h} := u^{(l)}/u^{(h)}$, since these two factors determine which one out of $u^{(l)}$ and $mu^{(h)}$ is the leading order term. There are 3 cases to consider.

Case 1: ($m \ll M_{l,h}$) The leading order term is $u^{(l)}$ and the mixed precision inner product can claim its superiority in accuracy only in comparison to the inner product computed in the lower precision. Therefore, this inner product mixed precision successfully reduces the error from $mu^{(l)}$

to $u^{(l)}$ with no apparent improvements in speed.

Case 2: ($m = M_{l,h}$) Both terms are now leading order. Thus, this is still an improvement in comparison to the lower precision arithmetic as the error is reduced from $mu^{(l)}$ to $2u^{(l)}$. Comparing this to the inner product computed entirely in the higher precision shows that the error has doubled from $mu^{(h)}$ to $2mu^{(h)}$, but gained a factor of 2 speed-up in timing instead. One can argue that the loss in accuracy and the improvement in speed have essentially canceled each other out.

Case 3: ($m \gg M_{l,h}$) Now the second term $\gamma_{m-1}^{(h)}$ is the leading order term. As in the above two cases, this is an improvement in the context of the low precision accuracy since the error has been reduced from $\gamma_m^{(l)}$ to $\gamma_{m/M}^{(l)} \equiv \gamma_m^{(h)}$. Since $u^{(l)} = Mu^{(h)} \ll mu^{(h)}$, the accuracy of the mixed precision inner product procedure from [Assumption 2.3](#) is what you would expect from carrying the computation out in the higher precision. Therefore, we can expect about the same level of accuracy but a factor of 2 reduction in speed when compared to the higher precision.

The remainder of this paper focuses only on accuracy and does not discuss the impact of mixed precision computations on speed and readers should refer to timing studies such as While the analysis in the above cases establish 3 regimes of trade-offs between accuracy and speed in mixed precision computing, we can also simplify the error in [\(2.11\)](#) via,

$$(1 + u^{(l)})(1 + \gamma_{m-1}^{(h)}) - 1 \leq \gamma_{M_{l,h}+m-1}^{(h)} = \gamma_{1+(m-1)/M_{l,h}}^{(l)}, \quad M_{l,h} = u^{(l)}/u^{(h)},$$

$$(1 + u^{(l)})(1 + \gamma_{m-1}^{(h)}) - 1 \leq u^{(l)} + \gamma_{m-1}^{(h)} + \min\{u^{(l)}, \gamma_{m-1}^{(h)}\}, \quad \gamma_{m-1}^{(h)} < 1.$$

These two alternate bounds can be useful in different contexts. Nevertheless, they both indicate that the error is $\mathcal{O}(u^{(l)} + mu^{(h)} + mu^{(l)}u^{(h)})$ and are slightly larger than the original bound, on the LHS. We summarize these ways of combining γ terms of different precisions in [Lemma 2.4](#), which is a simple modification of some of the rules from [Lemma 2.2](#).

LEMMA 2.4. *For any nonnegative integers k_l , k_h and some precision q defined with respect to the unit round-off, $u^{(q)}$, define $\gamma_k^{(q)} := \frac{ku^{(q)}}{1-ku^{(q)}}$. Consider a low precision and a high precision where $1 \gg u_l \gg u_h > 0$. Then,*

$$\gamma_{k_h}^{(h)} \gamma_{k_l}^{(l)} \leq \gamma_{k_l}^{(l)}, \quad \text{for } k_h u^{(h)} \leq \frac{1}{2},$$

$$\gamma_{k_h}^{(h)} + u^{(l)} \leq \gamma_{d_1}^{(l)},$$

$$\gamma_{k_l}^{(l)} + u^{(h)} \leq \gamma_{k_l+1}^{(l)},$$

$$\gamma_{k_l}^{(l)} + \gamma_{k_h}^{(h)} + \gamma_{k_l}^{(l)} \gamma_{k_h}^{(h)} < \gamma_{k_l+d_2}^{(l)},$$

where $d_1 = \lceil (u^{(l)} + k_h u^{(h)})/u^{(l)} \rceil$ and $d_2 = \lceil k_h u^{(h)}/u^{(l)} \rceil$.

Equations [\(2.10\)](#) and [\(2.11\)](#) are crucial for our analysis in [section 4](#) since they closely resemble the TensorCore FMA block which can return a matrix product in fp16 or fp32. Consider matrices $\mathbf{A} \in \mathbb{F}_{\text{fp16}}^{p \times m}$ and $\mathbf{B} \in \mathbb{F}_{\text{fp16}}^{m \times q}$, and $\mathbf{C} = \mathbf{AB} \in \mathbb{F}_{\text{fp16}}^{p \times q}$. If $\text{fl}(\mathbf{C})$ is desired in fp16, then each component of that matrix incurs rounding errors as shown in [\(2.11\)](#) and if it is desired in fp32, the componentwise rounding error is given by [\(2.10\)](#). Similarly, we could consider other mixed precision algorithms that cast down at various points within the algorithm to take advantage of better storage properties of lower precision types. In general, error bounds in the fashion of [\(2.10\)](#) can be used before the cast down operations, and the action of the cast down is best represented by error bounds similar to [\(2.11\)](#).

We have demonstrated a need for rounding error analysis that is accurate for mixed precision procedures and analyzed the inner product in a mixed precision procedure similar to that of TensorCore. We will use this to analyze some QR factorization algorithms. In [section 3](#), we introduce various Householder QR algorithms and the general framework for the standard rounding error analysis for these algorithms which we will modify for different mixed precision assumptions in [section 4](#).

3. Algorithms and existing round-off error analyses. We introduce the Householder QR factorization algorithm (HQR) in [subsection 3.1](#) and two block variants that use HQR within the block in [subsections 3.2](#) and [3.3](#). The blocked HQR (BQR) in [subsection 3.2](#) partitions the columns of the target matrix and utilizes mainly level-3 BLAS operations and is a well-known algorithm that uses the WY representation of [\[4\]](#). In contrast, the Tall-and-Skinny QR (TSQR) in [subsection 3.3](#) partitions rows of the matrix and takes a communication-avoiding divide-and-conquer approach that can be easily parallelized (see [\[7\]](#)). We also present the crucial results in standard rounding error analysis of these algorithms that excludes any mixed precision assumptions. These building steps of round-off error analysis will be easily tweaked for various mixed precision assumptions in [section 4](#).

3.1. Householder QR (HQR). The HQR algorithm uses Householder transformations to zero out elements below the diagonal of a matrix (see [\[15\]](#)). 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, $\mathbf{P}_\mathbf{v}$, such that $\mathbf{P}_\mathbf{v}$ zeros out \mathbf{x} below the first element.*

$$(3.1) \quad \begin{aligned} \sigma &= -\text{sign}(\mathbf{x}[1])\|\mathbf{x}\|_2, \quad \mathbf{v} = \mathbf{x} - \sigma\hat{\mathbf{e}}_1, \\ \beta &= \frac{2}{\mathbf{v}^\top \mathbf{v}} = -\frac{1}{\sigma\mathbf{v}[1]}, \quad \mathbf{P}_\mathbf{v} = \mathbf{I}_m - \beta\mathbf{v}\mathbf{v}^\top. \end{aligned}$$

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

3.1.1. HQR: 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 $i = 1, 2, \dots, n$,
Step 1) Compute \mathbf{v} and β that zeros out the i^{th} column of \mathbf{A} beneath a_{ii} (see [alg. 2](#)), and
Step 2) Apply $\mathbf{P}_\mathbf{v}$ to the bottom right partition, $\mathbf{A}[i : m, i : n]$ (lines 4-6 of [alg. 3](#)).

Consider the following 4-by-3 matrix example adapted from [\[12\]](#). Let \mathbf{P}_i represent the i^{th} Householder transformation of this algorithm.

$$\begin{aligned} \mathbf{A} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} &\xrightarrow{\text{apply } \mathbf{P}_1 \text{ to } \mathbf{A}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{\text{apply } \mathbf{P}_2 \text{ to } \mathbf{P}_1\mathbf{A}} \\ &\begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{\text{apply } \mathbf{P}_3 \text{ to } \mathbf{P}_2\mathbf{P}_1\mathbf{A}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{bmatrix} = \mathbf{P}_3\mathbf{P}_2\mathbf{P}_1\mathbf{A} =: \mathbf{R} \end{aligned}$$

298 Then, the \mathbf{Q} factor for a full QR factorization is $\mathbf{Q} := \mathbf{P}_1 \mathbf{P}_2 \mathbf{P}_3$ since \mathbf{P}_i 's are symmetric, and the
 299 thin factors for a general matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ are

$$300 \quad (3.2) \quad \mathbf{Q}_{\text{thin}} = \mathbf{P}_1 \cdots \mathbf{P}_n \mathbf{I}_{m \times n} \quad \text{and} \quad \mathbf{R}_{\text{thin}} = \mathbf{I}_{m \times n}^\top \mathbf{P}_n \cdots \mathbf{P}_1 \mathbf{A}.$$

Algorithm 2: $\beta, \mathbf{v}, \sigma = \text{hh_vec}(\mathbf{x})$. Given a vector $\mathbf{x} \in \mathbb{R}^n$, return $\mathbf{v}, \beta, \sigma$ that satisfy
 $(I - \beta \mathbf{v} \mathbf{v}^\top) \mathbf{x} = \sigma \hat{\mathbf{e}}_1$ and $\mathbf{v}[1] = 1$ (see [?, 12]).

Input: $\mathbf{x} \in \mathbb{R}^m$
Output: $\mathbf{v} \in \mathbb{R}^m$, and $\sigma, \beta \in \mathbb{R}$ such that $(I - \beta \mathbf{v} \mathbf{v}^\top) \mathbf{x} = \pm \|\mathbf{x}\|_2 \hat{\mathbf{e}}_1 = \sigma \hat{\mathbf{e}}_1$
 301 **1** $\mathbf{v} \leftarrow \text{copy}(\mathbf{x})$
2 $\sigma \leftarrow -\text{sign}(\mathbf{x}[1]) \|\mathbf{x}\|_2$
3 $\mathbf{v}[1] \leftarrow \mathbf{x}[1] - \sigma$
4 $\beta \leftarrow -\frac{\mathbf{v}[1]}{\sigma}$
5 **return** $\beta, \mathbf{v}/\mathbf{v}[1], \sigma$

Algorithm 3: $\mathbf{V}, \beta, \mathbf{R} = \text{HQR2}(\mathbf{A})$. A Level-2 BLAS implementation of the Householder
 QR algorithm. Given a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ where $m \geq n$, return matrix $\mathbf{V} \in \mathbb{R}^{m \times n}$, vector
 $\beta \in \mathbb{R}^n$, and upper triangular matrix \mathbf{R} . An orthogonal matrix \mathbf{Q} can be generated from
 \mathbf{V} and β , and $\mathbf{QR} = \mathbf{A}$.

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$ where $m \geq n$.
Output: $\mathbf{V}, \beta, \mathbf{R}$
1 $\mathbf{V}, \beta \leftarrow \mathbf{0}_{m \times n}, \mathbf{0}_m$
2 **for** $i = 1 : n$ **do**
3 $\mathbf{v}, \beta, \sigma \leftarrow \text{hh_vec}(\mathbf{A}[i : \text{end}, i])$ /* Algorithm 2 */
4 $\mathbf{V}[i : \text{end}, i], \beta_i, \mathbf{A}[i, i] \leftarrow \mathbf{v}, \beta, \sigma$
5 $\mathbf{A}[i + 1 : \text{end}, i] \leftarrow \text{zeros}(m - i)$
6 $\mathbf{A}[i : \text{end}, i + 1 : \text{end}] \leftarrow \mathbf{A}[i : \text{end}, i + 1 : \text{end}] - \beta \mathbf{v} \mathbf{v}^\top \mathbf{A}[i : \text{end}, i + 1 : \text{end}]$
7 **return** $\mathbf{V}, \beta, \mathbf{A}[1 : n, 1 : n]$

302 **3.1.2. HQR: Rounding Error Analysis.** Now we present an error analysis for [alg. 3](#) by
 303 keeping track of the different operations of [alg. 2](#) and [alg. 3](#).

304 *Calculating the i^{th} Householder vector and constant.* In [alg. 3](#), the we compute the Householder
 305 vector and constant by using [alg. 2](#) to $\mathbf{A}[i : m, i]$. Let us consider zeroing out any vector $\mathbf{x} \in \mathbb{R}^m$
 306 below its first component with a Householder transformation. We first calculate σ as is implemented
 307 in line 2 of [alg. 2](#).

$$308 \quad (3.3) \quad \text{fl}(\sigma) = \hat{\sigma} = \text{fl}(-\text{sign}(\mathbf{x}[1]) \|\mathbf{x}\|_2) = \sigma + \Delta\sigma, \quad |\Delta\sigma| \leq \gamma_{m+1} |\sigma|.$$

309 Note that the backward error incurred here accounts for an inner product of a vector in \mathbb{R}^m with
 310 itself and a square root operation to get the 2-norm. Let $\tilde{\mathbf{v}}[1] \equiv \mathbf{x}[i] - \sigma$, the penultimate value
 311 $\mathbf{v}[1]$. The subtraction adds a single additional rounding error via

$$312 \quad (3.4) \quad \text{fl}(\tilde{\mathbf{v}}[1]) = \tilde{\mathbf{v}}[1] + \Delta\tilde{\mathbf{v}}[1] = (1 + \delta)(\mathbf{x}[i] - \sigma - \Delta\sigma) = (1 + \theta_{m+2})\tilde{\mathbf{v}}[1]$$

313 where the last equality is granted because the sign of σ is chosen to prevent cancellation. Since
 314 [alg. 2](#) normalizes the Householder vector so that its first component is 1, the remaining components

of \mathbf{v} are divided by $\text{fl}(\tilde{\mathbf{v}}_1)$ incurring another single rounding error. As a result, the components of \mathbf{v} computed with FLOPs have error $\text{fl}(\mathbf{v}[j]) = \mathbf{v}[j] + \Delta\mathbf{v}[j]$ where

$$(3.5) \quad |\Delta\mathbf{v}[j]| \leq \begin{cases} 0, & j = 1 \\ \gamma_{1+2(m+2)}|\mathbf{v}[j]| = \tilde{\gamma}_m|\mathbf{v}[j]|, & j = 2 : m - i + 1. \end{cases}$$

Since $1 + 2(m+2) = \mathcal{O}(m)$, we have swept that minor difference between under our use of the $\tilde{\gamma}$ notation defined in [Lemma 2.1](#). Next, we consider the Householder constant, β , as is computed in line 4 of [alg. 2](#).

$$(3.6) \quad \hat{\beta} = \text{fl}(-\tilde{\mathbf{v}}[1]/\hat{\sigma}) = -(1 + \delta) \frac{\tilde{\mathbf{v}}[1] + \Delta\tilde{\mathbf{v}}[1]}{\sigma + \Delta\sigma}$$

$$(3.7) \quad = \frac{(1 + \delta)(1 + \theta_{m+2})}{(1 + \theta_{m+1})} \beta = (1 + \theta_{3m+5}) \beta$$

$$(3.8) \quad = \beta + \Delta\beta, \text{ where } |\Delta\beta| \leq \tilde{\gamma}_m \beta.$$

We have shown (3.6) to keep our analysis simple in [section 4](#) and (3.7) and (3.8) show that the error incurred from calculating of $\|\mathbf{x}\|_2$ accounts for the vast majority of the rounding error so far. At iteration i , we replace \mathbf{x} with $\mathbf{A}[i : m, i] \in \mathbb{R}^{m-i+1}$ and the i^{th} Householder constant and vector $(\hat{\beta}_i, \mathbf{v}_i)$ both have errors bounded by $\tilde{\gamma}_{m-i+1}$.

Applying a Single Householder Transformation. Now we consider lines 4-6 of [alg. 3](#). At iteration i , we set $\mathbf{A}[i + 1 : m]$ to zero and replace $\mathbf{A}[i, i]$ with σ computed from [alg. 2](#). Therefore, we now need to calculate the errors for applying a Householder transformation to the remaining columns, $\mathbf{A}[i : m, i + 1 : n]$ with the computed Householder vector and constant. This is the most crucial building block of the rounding error analysis for any variant of HQR because the \mathbf{R} factor is formed by applying the Householder transformations to \mathbf{A} and the \mathbf{Q} factor is formed by applying them in reverse order to the identity. Both of the blocked versions in [subsection 3.2](#) and [subsection 3.3](#) also require efficient implementations of this step, although they may be implemented slightly differently. For example, BQR in [alg. 5](#) uses level-3 BLAS operations to apply multiple Householder transformations at once whereas the variant of HQR in [alg. 3](#) can only use level-2 BLAS operations to apply Householder transformations.

A Householder transformation is applied through a series of inner and outer products, since Householder matrices are rank-1 updates of the identity. That is, computing $\mathbf{P}_\mathbf{v}\mathbf{x}$ for any $\mathbf{x} \in \mathbb{R}^m$ is as simple as computing

$$(3.9) \quad \mathbf{y} := \mathbf{P}_\mathbf{v}\mathbf{x} = \mathbf{x} - (\beta\mathbf{v}^\top\mathbf{x})\mathbf{v}.$$

Let us assume that \mathbf{x} is an exact vector and there were errors incurred in forming \mathbf{v} and β . 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 $\mathbf{P}_\mathbf{v}$ to a column. In practice, \mathbf{x} would be a column in $\mathbf{A}^{(i-1)}[i + 1 : m, i + 1 : n]$, where the superscript $(i - 1)$ indicates that this submatrix of \mathbf{A} has already been transformed by $i - 1$ Householder transformations that zeroed out components below $\mathbf{A}_{j,j}$ for $j = 1 : i - 1$. We show the error for forming $\text{fl}(\hat{\mathbf{v}}^\top\mathbf{x})$ where we let $\mathbf{v}, \mathbf{x} \in \mathbb{R}^m$:

$$(3.10) \quad \text{fl}(\hat{\mathbf{v}}^\top\mathbf{x}) = (1 + \theta_m)(\mathbf{v} + \Delta\mathbf{v})^\top\mathbf{x}.$$

Set $\mathbf{w} := \beta\mathbf{v}^\top\mathbf{x}\mathbf{v}$. Then,

$$(3.11) \quad \hat{\mathbf{w}} = (1 + \theta_m)(1 + \delta)(1 + \tilde{\delta})(\beta + \Delta\beta)(\mathbf{v} + \Delta\mathbf{v})^\top\mathbf{x}(\mathbf{v} + \Delta\mathbf{v}),$$

where θ_m is from computing the inner product $\hat{\mathbf{v}}^\top \mathbf{x}$, and δ and $\tilde{\delta}$ are from multiplying β , $\text{fl}(\hat{\mathbf{v}}^\top \mathbf{x})$, and $\hat{\mathbf{v}}$ together. We can write

$$\hat{\mathbf{w}} = \mathbf{w} + \Delta \mathbf{w}, \quad |\Delta \mathbf{w}| \leq \tilde{\gamma}_m |\beta| |\mathbf{v}|^\top |\mathbf{x}| |\mathbf{v}|.$$

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

$$(3.10) \quad \text{fl}(\hat{\mathbf{P}}_{\mathbf{v}} \mathbf{x}) = \text{fl}(\mathbf{x} - \hat{\mathbf{w}}) = (1 + \delta)(\mathbf{x} - \mathbf{w} - \Delta \mathbf{w}) = \mathbf{y} + \Delta \mathbf{y} = (\mathbf{P}_{\mathbf{v}} + \Delta \mathbf{P}_{\mathbf{v}}) \mathbf{x},$$

where $|\Delta \mathbf{y}| \leq u |\mathbf{x}| + \tilde{\gamma}_m |\beta| |\mathbf{v}| |\mathbf{v}|^\top |\mathbf{x}|$. Using $\sqrt{2/\beta} = \|\mathbf{v}\|_2$, we can conclude

$$(3.11) \quad \|\Delta \mathbf{y}\|_2 \leq \tilde{\gamma}_m \|\mathbf{x}\|_2.$$

Next, we convert this to a backward error for $\mathbf{P}_{\mathbf{v}}$. Since $\Delta \mathbf{P}_{\mathbf{v}}$ is exactly $\frac{1}{\mathbf{x}^\top \mathbf{x}} \Delta \mathbf{y} \mathbf{x}^\top$, we can compute its Frobenius norm by using $\Delta \mathbf{P}_{\mathbf{v}}[i, j] = \frac{1}{\|\mathbf{x}\|_2^2} \Delta \mathbf{y}[i] \mathbf{x}[j]$,

$$(3.12) \quad \|\Delta \mathbf{P}_{\mathbf{v}}\|_F = \left(\sum_{i=1}^m \sum_{j=1}^m \left(\frac{1}{\|\mathbf{x}\|_2^2} \Delta \mathbf{y}[i] \mathbf{x}[j] \right)^2 \right)^{1/2} = \frac{\|\Delta \mathbf{y}\|_2}{\|\mathbf{x}\|_2} \leq \tilde{\gamma}_m,$$

where the last inequality is a direct application of (3.11).

Applying many successive Householder transformations. Consider applying a sequence of transformations in the set $\{\mathbf{P}_i\}_{i=1}^r \subset \mathbb{R}^{m \times m}$ to $\mathbf{x} \in \mathbb{R}^m$, where \mathbf{P}_i 's are all Householder transformations computed with $\hat{\mathbf{v}}_i$'s and $\hat{\beta}_i$'s. This is directly applicable to HQR as $\mathbf{Q} = \mathbf{P}_1 \cdots \mathbf{P}_n \mathbf{I}$ and $\mathbf{R} = \mathbf{Q}^\top \mathbf{A} = \mathbf{P}_n \cdots \mathbf{P}_1 \mathbf{A}$. Lemma 3.2 is very useful for any sequence of transformations, where each transformation has a known bound. We will invoke this lemma to prove Lemma 3.3, and use it in future sections for other sequential transformations.

LEMMA 3.2. *If $\mathbf{X}_j + \Delta \mathbf{X}_j \in \mathbb{R}^{m \times m}$ satisfies $\|\Delta \mathbf{X}_j\|_F \leq \delta_j \|\mathbf{X}_j\|_2$ for all j , then*

$$\left\| \prod_{j=1}^n (\mathbf{X}_j + \Delta \mathbf{X}_j) - \prod_{j=1}^n \mathbf{X}_j \right\|_F \leq \left(-1 + \prod_{j=1}^n (1 + \delta_j) \right) \prod_{j=1}^n \|\mathbf{X}_j\|_2.$$

LEMMA 3.3. *Consider applying a sequence of transformations $\mathbf{Q} = \mathbf{P}_r \cdots \mathbf{P}_2 \mathbf{P}_1$ onto vector $\mathbf{x} \in \mathbb{R}^m$ to form $\hat{\mathbf{y}} = \text{fl}(\hat{\mathbf{P}}_r \cdots \hat{\mathbf{P}}_2 \hat{\mathbf{P}}_1 \mathbf{x})$, where $\hat{\mathbf{P}}_k$'s are Householder transformations constructed from $\hat{\beta}_k$ and $\hat{\mathbf{v}}_k$. These Householder vectors and constants are computed via alg. 2 and the rounding errors are bounded by (3.5) and (3.8). If each transformation is computed via (3.9), then*

$$(3.13) \quad \hat{\mathbf{y}} = \mathbf{Q}(\mathbf{x} + \Delta \mathbf{x}) = (\mathbf{Q} + \Delta \mathbf{Q}) \mathbf{x} = \hat{\mathbf{Q}} \mathbf{x},$$

$$(3.14) \quad \|\Delta \mathbf{y}\|_2 \leq r \tilde{\gamma}_m \|\mathbf{x}\|_2, \quad \|\Delta \mathbf{Q}\|_F \leq r \tilde{\gamma}_m.$$

Proof. Applying Lemma 3.2 directly to \mathbf{Q} yields

$$\begin{aligned} \|\Delta \mathbf{Q}\|_F &= \left\| \prod_{j=1}^r (\mathbf{P}_j + \Delta \mathbf{P}_j) - \prod_{j=1}^r \mathbf{P}_j \right\|_F \leq \left(-1 + \prod_{j=1}^r (1 + \tilde{\gamma}_{m-j+1})^r \right) \prod_{j=1}^r \|\mathbf{P}_j\|_2 \\ &\leq -1 + (1 + \tilde{\gamma}_m)^r, \end{aligned}$$

since \mathbf{P}_j 's are orthogonal and have 2-norm, 1, and $m - j + 1 \leq m$. While we omit the details here, we can show that $(1 + \tilde{\gamma}_m)^r - 1 \leq r \tilde{\gamma}_m$ using the argument from Lemma 2.1 if $r \tilde{\gamma}_m \leq 1/2$. \square

In this error analysis, the prevailing bound for errors at various stages of forming and applying a Householder transformation is $\tilde{\gamma}_m$ where m corresponds to the dimension of the transformed vectors. In [Lemma 3.3](#), a factor of r is introduced for applying r Householder transformations to form the term $r\tilde{\gamma}_m \approx rmu$. Therefore, we can expect that the columnwise norm error for a thin QR factorization should be $\mathcal{O}(mnu)$ for a full rank matrix. In [Theorem 3.4](#), we formalize this by applying [Lemma 3.3](#) directly and also show the result of converting these columnwise bounds to matrix norm bounds.

$$\|\Delta \mathbf{R}\|_F = \left(\sum_{i=1}^n \|\Delta \mathbf{R}[:, i]\|_2^2 \right)^{1/2} \leq \left(\sum_{i=1}^n n^2 \tilde{\gamma}_m^2 \|\mathbf{A}[:, i]\|_2^2 \right)^{1/2} = n\tilde{\gamma}_m \|\mathbf{A}\|_F,$$

We gather these results into [Theorem 3.4](#).

THEOREM 3.4. *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 [alg. 3](#). Then,*

$$\begin{aligned} \hat{\mathbf{R}} &= \mathbf{R} + \Delta \mathbf{R} = \mathbf{fl}(\hat{\mathbf{P}}_n \cdots \hat{\mathbf{P}}_1 \mathbf{A}), \quad \|\Delta \mathbf{R}[:, j]\|_2 \leq n\tilde{\gamma}_m \|\mathbf{A}[:, j]\|_2, \quad \|\Delta \mathbf{R}\|_F \leq n\tilde{\gamma}_m \|\mathbf{A}\|_F \\ \hat{\mathbf{Q}} &= \mathbf{Q} + \Delta \mathbf{Q} = \mathbf{fl}(\hat{\mathbf{P}}_1 \cdots \hat{\mathbf{P}}_n \mathbf{I}), \quad \|\Delta \mathbf{Q}[:, j]\|_2 \leq n\tilde{\gamma}_m, \quad \|\Delta \mathbf{Q}\|_F \leq n^{3/2} \tilde{\gamma}_m. \end{aligned}$$

Let $\mathbf{A} + \Delta \mathbf{A} = \hat{\mathbf{Q}}\hat{\mathbf{R}}$, where $\hat{\mathbf{Q}}$ and $\hat{\mathbf{R}}$ are obtained via [Algorithm 3](#). Then the backward error is

$$\|\hat{\mathbf{Q}}\hat{\mathbf{R}} - \mathbf{A}\|_F \leq n^{3/2} \tilde{\gamma}_m \|\mathbf{A}\|_F.$$

Note that the last backward error result follows from the columnwise forward errors for $\hat{\mathbf{R}}$ and $\hat{\mathbf{Q}}$. Out of all of these different ways of measuring the error from computing a QR factorization (forward/backward errors for column/matrix norms), we will focus on $\|\hat{\mathbf{Q}} - \mathbf{Q}\|_F$, a measure of orthogonality of the \mathbf{Q} factor for the remainder of [section 3](#) and for [section 4](#). In [section 5](#), we will turn to the backward norm error, $\|\hat{\mathbf{Q}}\hat{\mathbf{R}} - \mathbf{A}\|_F$ since it can actually be computed.

The content of this section shows the standard rounding error analysis in [\[12\]](#) where some important stages are summarized in [\(3.5\)](#), [\(3.8\)](#), and [\(3.14\)](#), which we will modify to different mixed precision settings in [section 4](#). These quantities account for various forward and backward errors formed in computing essential components of HQR, namely the Householder constant and vector, as well as normwise errors of the action of applying Householder transformations. In the next sections, we present blocked variants of HQR that use [alg. 3](#).

3.2. Block HQR with partitioned columns (BQR). We refer to the blocked variant of HQR where the columns are partitioned as BQR. Note that this section relies on the WY representation described in [\[4\]](#) instead of the storage-efficient version of [\[19\]](#), even though both are known to be just as numerically stable as HQR.

3.2.1. The WY Representation. A convenient matrix representation that accumulates r Householder reflectors is known as the WY representation (see [\[4, 9\]](#)). [Lemma 3.5](#) shows how to update a rank- j update of the identity, $\mathbf{Q}^{(j)}$, with a Householder transformation, \mathbf{P} , to produce a rank- $(j+1)$ update of the identity, $\mathbf{Q}^{(j+1)}$. With the correct initialization of \mathbf{W} and \mathbf{Y} , we can build the WY representation of successive Householder transformations as shown in [Algorithm 4](#). This algorithm assumes that the Householder vectors, \mathbf{V} , and constants, β , have already been computed. Since the \mathbf{Y} factor is exactly \mathbf{V} , we only need to compute the \mathbf{W} factor.

424 LEMMA 3.5. Suppose $\mathbf{X}^{(j)} = \mathbf{I} - \mathbf{W}^{(j)}\mathbf{Y}^{(j)\top} \in \mathbb{R}^{m \times m}$ is an orthogonal matrix with $\mathbf{W}^{(j)}, \mathbf{Y}^{(j)} \in$
 425 $\mathbb{R}^{m \times j}$. Let us define $\mathbf{P} = \mathbf{I} - \beta \mathbf{v}\mathbf{v}^\top$ for some $\mathbf{v} \in \mathbb{R}^m$ and let $\mathbf{z}^{(j+1)} = \beta \mathbf{X}^{(j)}\mathbf{v}$. Then,

426
$$\mathbf{X}^{(j+1)} = \mathbf{X}^{(j)}\mathbf{P} = \mathbf{I} - \mathbf{W}^{(j+1)}\mathbf{Y}^{(j+1)\top},$$

427 where $\mathbf{W}^{(j+1)} = [\mathbf{W}^{(j)} | \mathbf{z}]$ and $\mathbf{Y}^{(j+1)} = [\mathbf{Y}^{(j)} | \mathbf{v}]$ are each m -by- $(j+1)$.

Algorithm 4: $\mathbf{W}, \mathbf{Y} \leftarrow \text{buildWY}(V, \beta)$: Given a set of householder vectors $\{\mathbf{V}[:, i]\}_{i=1}^r$ and their corresponding constants $\{\beta_i\}_{i=1}^r$, form the final \mathbf{W} and \mathbf{Y} factors of the WY representation of $\mathbf{P}_1 \cdots \mathbf{P}_r$, where $\mathbf{P}_i := \mathbf{I}_m - \beta_i \mathbf{v}_i \mathbf{v}_i^\top$

Input: $\mathbf{V} \in \mathbb{R}^{m \times r}$, $\beta \in \mathbb{R}^r$ where $m > r$.
Output: \mathbf{W}
 1 Initialize: $\mathbf{W} := \beta_1 \mathbf{V}[:, 1]$. /* \mathbf{Y} is \mathbf{V} . */
 2 **for** $j = 2 : r$ **do**
 3 $\mathbf{z} \leftarrow \beta_j [\mathbf{V}[:, j] - \mathbf{W}(\mathbf{V}[:, 1:j-1]^\top \mathbf{V}[:, j])]$
 4 $\mathbf{W} \leftarrow [\mathbf{W} \quad \mathbf{z}]$ /* Update \mathbf{W} to an m -by- j matrix. */
 5 **return** \mathbf{W}

428 In HQR, \mathbf{A} is transformed into an upper triangular matrix \mathbf{R} by identifying a Householder
 429 transformation that zeros out a column below the diagonal, then applying that Householder trans-
 430 formation to the bottom right partition. For example, the k^{th} Householder transformation finds
 431 an $m - k + 1$ sized Householder transformation that zeros out column k below the diagonal and
 432 then applies it to the $(m - k + 1)$ -by- $(n - k)$ partition of the matrix, $\mathbf{A}[k : m, k + 1 : n]$. Since
 433 the $k + 1^{\text{st}}$ column is transformed by the k^{th} Householder transformation, this algorithm must be
 434 executed serially as shown in [alg. 3](#). The highest computational burden at each iteration falls on
 435 [alg. 3](#) line 6, which requires Level-2 BLAS operations when computed efficiently.

436 In contrast, BQR replaces this step with Level-3 BLAS operations by partitioning \mathbf{A} into blocks
 437 of columns. Let $\mathbf{A} = [\mathbf{C}_1 \cdots \mathbf{C}_N]$ where $\mathbf{C}_1, \dots, \mathbf{C}_{N-1}$ are each m -by- r , and \mathbf{C}_N holds the remaining
 438 columns. The k^{th} block, \mathbf{C}_k , is transformed with HQR ([alg. 3](#)), and the WY representation of these
 439 r successive Householder transformations is constructed as in [alg. 4](#). We write the WY update as

440 (3.15)
$$\mathbf{X}_k = \mathbf{I}_m - \mathbf{W}_k \mathbf{Y}_k^\top = \mathbf{P}_k^{(1)} \cdots \mathbf{P}_k^{(r)}.$$

441 Thus far, [algs. 3](#) and [4](#) are rich in Level-2 BLAS operations. Next, $\mathbf{I} - \mathbf{Y}_k \mathbf{W}_k^\top$ is applied to
 442 $[\mathbf{C}_2 \cdots \mathbf{C}_N]$ with two Level-3 BLAS operations as shown in line 5 of [alg. 5](#). BQR performs ap-
 443 proximately $1 - \mathcal{O}(1/N)$ fraction of its FLOPs in Level-3 BLAS operations (see section 5.2.3 of
 444 [\[9\]](#)), and can reap the benefits from the accelerated block FMA feature of TensorCore. Note that
 445 BQR does require strictly more FLOPs when compared to HQR, but these additional FLOPs are
 446 negligible in standard precision and does not impact the numerical stability. A pseudoalgorithm
 447 for BQR is shown in [alg. 5](#) where we assume that n is divisible by r so that $N = \lceil n/r \rceil = n/r$
 448 to make our error analysis in [section 3.2.2](#) simple. In practice, an efficient implementation might
 449 require r to be a power of two or a product of small prime factors and result a thinner N^{th} block
 450 compared to the rest. This discrepancy is easily fixed by padding the matrix with zeros, a stan-
 451 dard procedure for standard algorithms like the Fast Fourier Transform (FFT). For any variable
 452 $x \in \{\mathbf{X}, \mathbf{W}, \mathbf{Y}, \mathbf{z}, \beta, \mathbf{v}, \mathbf{P}\}$, $x_k^{(j)}$ corresponds to the j^{th} update for the k^{th} block.

Algorithm 5: $\mathbf{Q}, \mathbf{R} \leftarrow \text{blockHQR}(\mathbf{A}, r)$: Perform Householder QR factorization of matrix \mathbf{A} with column partitions of size r .

Input: $\mathbf{A} \in \mathbb{R}^{m \times n}$, $r \in \mathbb{R}$ where $r < n$.

Output: \mathbf{Q}, \mathbf{R}

```

1  $N = \frac{n}{r}$ 
  // Let  $\mathbf{A} = [\mathbf{C}_1 \cdots \mathbf{C}_N]$  where all blocks except  $\mathbf{C}_N$  are  $m$ -by- $r$  sized.
2 for  $i = 1 : N$  do
3    $\mathbf{V}_i, \beta_i, \mathbf{C}_i \leftarrow \text{hhQR}(\mathbf{C}_i)$                                      /* Algorithm 3 */
4    $\mathbf{W}_i \leftarrow \text{buildWY}(\mathbf{V}_i, \beta_i)$                                    /* Algorithm 4 */
5    $[\mathbf{C}_{i+1} \cdots \mathbf{C}_N] \leftarrow \mathbf{V}_i (\mathbf{W}_i^\top [\mathbf{C}_{i+1} \cdots \mathbf{C}_N])$  /* update the rest: BLAS-3 */
  //  $\mathbf{A}$  has been transformed into  $\mathbf{R} = \mathbf{Q}^\top \mathbf{A}$ .
  // Now build  $\mathbf{Q}$  using level-3 BLAS operations.
6  $\mathbf{Q} \leftarrow \mathbf{I}$                                                          /*  $\mathbf{I}_m$  if full QR, and  $\mathbf{I}_{m \times n}$  if thin QR. */
7 for  $i = N : -1 : 1$  do
8    $\mathbf{Q}[(i-1)r+1 : m, (i-1)r+1 : n] \leftarrow \mathbf{W}_i (\mathbf{V}_i^\top \mathbf{Q}[(i-1)r+1 : m, (i-1)r+1 : n])$ 
9 return  $\mathbf{Q}, \mathbf{A}$ 

```

3.2.2. BQR: Rounding Error Analysis. We now present the basic structure for the rounding error analysis for [alg. 5](#), which consist of: 1)HQR, 2)building the \mathbf{W} factor, and 3) updating the remaining blocks with the WY representation. We have adapted the analysis from [12] to fit this exact variant, and denote $\hat{\mathbf{Q}}_{BQR}, \hat{\mathbf{R}}_{BQR}$ to be the outputs from [alg. 5](#). First, we analyze the error accumulated from updating $\mathbf{X}_k^{(j-1)}$ to $\mathbf{X}_k^{(j)}$, which applies a rank-1 update via the subtraction of the outer product $\hat{\mathbf{z}}_k^{(j)} \hat{\mathbf{v}}_k^{(j)\top}$. Since $\mathbf{z}_k^{(j)} = \beta_k^{(j)} \mathbf{X}_k^{(j-1)} \mathbf{v}_k^{(j)}$, this update requires a single Householder transformation on the right side in the same efficient implementation that is discussed in (3.9),

$$(3.16) \quad \hat{\mathbf{X}}_k^{(j)} = \hat{\mathbf{X}}_k^{(j-1)} - \text{fl}(\hat{\beta}_k^{(j-1)} \hat{\mathbf{X}}_k^{(j-1)} \hat{\mathbf{v}}_k^{(j-1)}) \hat{\mathbf{v}}_k^{(j)\top} = \hat{\mathbf{X}}_k^{(j-1)} (\mathbf{P}_k^{(j)} + \Delta \mathbf{P}_k^{(j)}),$$

where $\|\Delta \mathbf{P}_k^{(j)}\|_F \leq \tilde{\gamma}_{m-(k-1)r}$. Since $\hat{\mathbf{X}}_k^{(1)} = \mathbf{I} - \hat{\beta}_k^{(1)} \hat{\mathbf{v}}_k^{(1)} \hat{\mathbf{v}}_k^{(1)\top} = \mathbf{P}_k^{(1)} + \Delta \mathbf{P}_k^{(1)}$, we can travel up the recursion relation in (3.16) and use [Lemma 3.2](#) to find

$$(3.17) \quad \|\Delta \mathbf{X}_k^{(j)}\|_F \leq j \tilde{\gamma}_{m-(k-1)r}.$$

HQR within each block: line 3 of [alg. 5](#). We apply [Algorithm 3](#) to the k^{th} block, $\mathbf{C}_k^{((k-1)r)}$, which applies r more Householder transformations to columns that had been transformed by $(k-1)r$ Householder transformations in prior iterations. The upper trapezoidal factor that results from applying HQR to $\mathbf{C}_k^{((k-1)r)}$ corresponds to the $(k-1)r+1^{st}$ to kr^{th} columns of $\hat{\mathbf{R}}_{BQR}$, and applying [Lemmas 3.2](#) and [3.3](#) yields

$$\|\hat{\mathbf{R}}_{BQR}[:, j] - \mathbf{R}[:, j]\|_2 \leq r \tilde{\gamma}_m \|\mathbf{A}^{((k-1)r)}[:, j]\|_2 \leq kr \tilde{\gamma}_m \|\mathbf{A}[:, j]\|_2, \quad j = (k-1)r+1 : kr.$$

Build WY at each block: line 4 of [alg. 5](#). We now calculate the rounding errors incurred from building the WY representation when given a set of Householder vectors and constants as shown in [alg. 4](#). Since the columns of $\hat{\mathbf{Y}}_k$ are simply $\{\hat{\mathbf{v}}_k^{(j)}\}$ built in [alg. 3](#) the errors for forming these are

shown in (3.5) where m should be replaced by $m - (k - 1)r$. The Householder constants, $\hat{\beta}_k^{(j)}$ are bounded by (3.8) modified similarly. Thus, $\mathbf{z}_k^{(j)}$ is the only newly computed quantity. Using (3.5), (3.8), and (3.17), we find

$$\begin{aligned} \|\Delta \mathbf{z}_k^{(j)}\|_2 &= \|\Delta \mathbf{X}_k^{(j-1)} \hat{\beta}_k^{(j)} \hat{\mathbf{v}}_k^{(j)}\|_2 \leq \|\Delta \mathbf{X}_k^{(j-1)}\|_2 \|\hat{\beta}_k^{(j)} \hat{\mathbf{v}}_k^{(j)}\|_2 \\ &\leq \|\Delta \mathbf{X}_k^{(j-1)}\|_F \|\hat{\beta}_k^{(j)} \hat{\mathbf{v}}_k^{(j)}\|_2 \\ &\leq ((1 + (j - 1)\tilde{\gamma}_{m-(k-1)r})(1 + \tilde{\gamma}_{m-(k-1)r}) - 1) \|\beta_k^{(j)} \mathbf{v}_k^{(j)}\|_2 \\ &\leq j\tilde{\gamma}_{m-(k-1)r} \|\mathbf{z}_k^{(j)}\|_2 \end{aligned}$$

Componentwise bounds follow immediately, and are summarized in summarized in Lemma 3.6.

LEMMA 3.6. *Consider the construction of the WY representation for the k^{th} partition of matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ given a set of Householder constants and vectors, $\{\beta_k^{(j)}\}_{j=1}^r$ and $\{\mathbf{v}_k^{(j)}\}$ via alg. 4. Then,*

$$(3.18) \quad \hat{\mathbf{z}}_k^{(j)} = \mathbf{z}_k^{(j)} + \Delta \mathbf{z}_k^{(j)}, \quad |\Delta \mathbf{z}_k^{(j)}| \leq j\tilde{\gamma}_{m-(k-1)r} |\mathbf{z}_k^{(j)}|$$

$$(3.19) \quad \hat{\mathbf{v}}_k^{(j)} = \mathbf{v}_k^{(j)} + \Delta \mathbf{v}_k^{(j)}, \quad |\Delta \mathbf{v}_k^{(j)}| \leq \tilde{\gamma}_{m-(k-1)r} |\mathbf{v}_k^{(j)}|,$$

where the second bound is derived from (3.5).

Most importantly, this shows that constructing the WY update is just as numerically stable as applying successive Householder transformations (see Section 19.5 of [12]).

Update blocks to the right: line 5 of alg. 5. We now consider applying $\mathbf{X}_k := \mathbf{I} - \mathbf{W}_k \mathbf{Y}_k^\top$ to some matrix, \mathbf{B} . In practice, \mathbf{B} is the bottom right submatrix, $[\mathbf{C}_{k+1} \cdots \mathbf{C}_N][(k - 1)r + 1 : m, :]$. We can apply (3.17) directly to the columns of \mathbf{B} ,

$$(3.20) \quad \|\mathbf{fl}(\hat{\mathbf{X}}_k \mathbf{B}[:, j])\|_2 = \|\mathbf{fl}(\hat{\mathbf{X}}_k^{(r)} \mathbf{B}[:, j])\|_2 \leq r\tilde{\gamma}_{m-(k-1)r} \|\mathbf{B}[:, j]\|_2$$

A normwise bound for employing a general matrix-matrix multiplication operation is stated in section 19.5 of [12].

Multiple WY updates: line 8-9 of alg. 5. All that remains is to consider the application of successive WY updates to form the QR factorization computed with BQR denoted as \mathbf{Q}_{BQR} and \mathbf{R}_{BQR} . We can apply Lemma 3.2 directly by setting $\mathbf{X}_k := \mathbf{I} - \mathbf{W}_k \mathbf{Y}_k^\top$ and consider the backward errors for applying the sequence to a vector, $\mathbf{x} \in \mathbb{R}^m$, as we did for Lemma 3.3. Since $\mathbf{X}_k = \mathbf{P}_{(k-1)r+1} \cdots \mathbf{P}_{kr}$, is simply a sequence of Householder transformations, it is orthogonal, i.e. $\|\mathbf{X}_k\|_2 = 1$. We only need to replace with \mathbf{x} with $\mathbf{A}[:, i]$'s to form the columnwise bounds for \mathbf{R}_{BQR} , and apply the transpose to $\hat{\mathbf{e}}_i$'s to form the bounds for \mathbf{Q}_{BQR} . Then,

$$(3.21) \quad \left\| \prod_{k=1}^N (\mathbf{X}_k + \Delta \mathbf{X}_k) - \prod_{k=1}^N \mathbf{X}_k \right\|_F \leq \left(-1 + \sum_{k=1}^N (1 + r\tilde{\gamma}_{m-(k-1)r}) \right) \leq rN\tilde{\gamma}_m \equiv n\tilde{\gamma}_m,$$

$$(3.22) \quad \|\hat{\mathbf{Q}}_{BQR} - \mathbf{Q}\|_F \leq n^{3/2} \tilde{\gamma}_m.$$

Now we can form the normwise bound for the j'^{th} column of $\hat{\mathbf{Q}}_{BQR}, \hat{\mathbf{R}}_{BQR}$. If we let $k' = \lceil j'/r \rceil^{\text{th}}$, then the j'^{th} column is the result of applying $k' - 1$ WY updates and an additional HQR. Applying Lemma 3.2 yields

$$(3.23) \quad \|\Delta \mathbf{R}_{BQR}[:, j']\|_2 \leq rk'\tilde{\gamma}_m \|\mathbf{A}[:, j]\|_2$$

$$(3.24) \quad \|\Delta \mathbf{Q}_{BQR}[:, j']\|_2 \leq rk'\tilde{\gamma}_m,$$

and near orthogonality of the \mathbf{Q} factor is still achieved,

$$(3.25) \quad \|\Delta \mathbf{Q}_{BQR}\|_F = r\tilde{\gamma}_m \sum_{j=1}^n \lceil j/r \rceil = n^{3/2}\tilde{\gamma}_m.$$

The primary goal of the analysis presented in this section is to provide the basic skeleton for the standard BQR rounding error analysis to make the generalization to mixed precision settings in [section 4](#) easier. Readers should refer to [\[9, 12\]](#) for full details.

3.3. Block HQR with partitioned rows : Tall-and-Skinny QR (TSQR). 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 block QR factorization methods with row partitions, we will discuss a specific variant of TSQR which is also known as the AllReduce algorithm [\[18\]](#). In this paper, the TSQR/AllReduce algorithm refers to the most parallel variant of the block QR factorization algorithms discussed in [\[8\]](#). A detailed description and rounding error analysis of this algorithm can be found in [\[18\]](#), and we present a pseudocode for the algorithm in [alg. 6](#). 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 5](#)) show that TSQR can not only increase the speed but also outperform the traditional HQR factorization in low precisions.

3.3.1. TSQR/AllReduce Algorithm. [Algorithm 6](#) partitions the rows of a tall-and-skinny matrix, \mathbf{A} . HQR 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 $i = 1, \dots, L$. The set of matrices that are QR factorized at each level i are called $\mathbf{A}_j^{(i)}$ for $j = 1, \dots, 2^{L-i}$, where superscript (i) corresponds to the level and the subscript j indexes the row-blocks within level i . In the following sections, [alg. 6](#) (`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 [alg. 3](#) and we use [alg. 2](#) as a subroutine of `qr`.

TSQR Notation. We 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 the correct blocks from the level

554 below: $\tilde{\mathbf{Q}}_{\alpha(j), \phi(j)}^{(i+1)}$. For $j = 1, \dots, 2^{L-i}$ at level i , we need $j = 2(\alpha(j) - 1) + \phi(j)$, where $\alpha(j) = \lceil \frac{j}{2} \rceil$
555 and $\phi(j) = 2 + j - 2\alpha(j) \in \{1, 2\}$. [section 3.3.2](#) shows full linear algebra details for a single-level
556 ($L = 1, 2$ initial blocks) example. The reconstruction of \mathbf{Q} can be implemented more efficiently (see
557 [\[3\]](#)), but the reconstruction method in [alg. 6](#) is presented for a clear, straightforward explanation.

Algorithm 6: $\mathbf{Q}, \mathbf{R} = \text{tsqr}(\mathbf{A}, L)$. 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 $\mathbf{QR} = \mathbf{A}$.

```

1  $h \leftarrow m2^{-L}$  // Number of rows.
/* Split  $\mathbf{A}$  into  $2^L$  blocks. Note that level  $(i)$  has  $2^{L-i}$  blocks. */
2 for  $j = 1 : 2^L$  do
3    $\mathbf{A}_j^{(0)} \leftarrow \mathbf{A}[(j-1)h + 1 : jh, :]$ 
/* Store Householder vectors as columns of matrix  $\mathbf{V}_j^{(i)}$ , Householder
   constants as components of vector  $\beta_j^{(i)}$ , and set up the next level. */
4 for  $i = 0 : L - 1$  do
5   /* The inner loop can be parallelized. */
6   for  $j = 1 : 2^{L-i}$  do
7      $\mathbf{V}_{2j-1}^{(i)}, \beta_{2j-1}^{(i)}, \mathbf{R}_{2j-1}^{(i)} \leftarrow \text{qr}(\mathbf{A}_{2j-1}^{(i)})$ 
8      $\mathbf{V}_{2j}^{(i)}, \beta_{2j}^{(i)}, \mathbf{R}_{2j}^{(i)} \leftarrow \text{qr}(\mathbf{A}_{2j}^{(i)})$ 
9      $\mathbf{A}_j^{(i+1)} \leftarrow \begin{bmatrix} \mathbf{R}_{2j-1}^{(i)} \\ \mathbf{R}_{2j}^{(i)} \end{bmatrix}$ 
10   $\mathbf{V}_1^{(L)}, \beta_1^{(L)}, \mathbf{R} \leftarrow \text{qr}(\mathbf{A}_1^{(L)})$  // The final  $\mathbf{R}$  factor is built.
11   $\mathbf{Q}_1^{(L)} \leftarrow \text{hh\_mult}(\mathbf{V}_1^{(L)}, I_{2n \times n})$ 
/* Compute  $\mathbf{Q}^{(i)}$  factors by applying  $\mathbf{V}^{(i)}$  to  $\mathbf{Q}^{(i+1)}$  factors. */
12  for  $i = L - 1 : -1 : 1$  do
13    for  $j = 1 : 2^{L-i}$  do
14       $\mathbf{Q}_j^{(i)} \leftarrow \text{hh\_mult} \left( \mathbf{V}_j^{(i)}, \begin{bmatrix} \tilde{\mathbf{Q}}_{\alpha(j), \phi(j)}^{(i+1)} \\ \mathbf{0} \end{bmatrix} \right)$ 
15   $\mathbf{Q} \leftarrow \mathbf{I}$ ; // Construct the final  $\mathbf{Q}$  factor.
16  for  $j = 1 : 2^L$  do
17     $\mathbf{Q} \leftarrow \begin{bmatrix} \mathbf{Q} \\ \text{hh\_mult} \left( \mathbf{V}_j^{(0)}, \begin{bmatrix} \tilde{\mathbf{Q}}_{\alpha(j), \phi(j)}^{(1)} \\ \mathbf{0} \end{bmatrix} \right) \end{bmatrix}$ 
18  return  $\mathbf{Q}, \mathbf{R}$ 
```

559 **3.3.2. Single-level Example.** In the single-level version of this algorithm, we first bisect \mathbf{A}
560 into $\mathbf{A}_1^{(0)}$ and $\mathbf{A}_2^{(0)}$ and compute the QR factorization of each of those submatrices. We combine the
561 resulting upper-triangular matrices (see below) which is QR factorized, and the process is repeated:

$$562 \quad \mathbf{A} = \begin{bmatrix} \mathbf{A}_1^{(0)} \\ \mathbf{A}_2^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)} \mathbf{R}_1^{(0)} \\ \mathbf{Q}_2^{(0)} \mathbf{R}_2^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2^{(0)} \end{bmatrix} \begin{bmatrix} \mathbf{R}_1^{(0)} \\ \mathbf{R}_2^{(0)} \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2^{(0)} \end{bmatrix} \mathbf{A}_1^{(1)} = \begin{bmatrix} \mathbf{Q}_1^{(0)} & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_2^{(0)} \end{bmatrix} \mathbf{Q}_1^{(1)} \mathbf{R}.$$

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, [alg. 6](#) 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} . For simplicity, we assume that m is exactly $h2^{(l)}$ so that the initial partition yields $2^{(l)}$ blocks of equal sizes, h -by- n . Also, note that `hh_mult` refers to the action of applying multiple Householder transformations given a set of Householder vectors and constants, which can be performed by iterating line 6 of [alg. 3](#). This step can be done in a level-3 BLAS operation via a WY update if [alg. 6](#) was modified to store the WY representation at the QR factorization of each block of each level, $\mathbf{A}_j^{(i)}$.

3.3.3. TSQR: Rounding Error Analysis. The TSQR algorithm presented in [alg. 6](#) 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 , $\lfloor \log_2(\frac{m}{n}) \rfloor$ -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 [\[18\]](#) and we summarize the main results in [Theorem 3.7](#).

THEOREM 3.7. *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 [alg. 6](#) with L levels. Let us further assume that m is divisible by 2^L and $n\tilde{\gamma}_{m2^{-L}}, n\tilde{\gamma}_{2n} \ll 1$. Then, normwise error bounds for the j^{th} column ($j = 1 : n$) are*

$$(3.26) \quad \|\hat{\mathbf{R}}_{TSQR}[:, j] - \mathbf{R}[:, j]\|_2 \leq n(\tilde{\gamma}_{m2^{-L}} + L\tilde{\gamma}_{2n})\|\mathbf{A}[:, j]\|_2,$$

$$(3.27) \quad \|\hat{\mathbf{Q}}_{TSQR}[:, j] - \mathbf{Q}[:, j]\|_2 \leq n(\tilde{\gamma}_{m2^{-L}} + L\tilde{\gamma}_{2n}).$$

Note that the $n\tilde{\gamma}_{m2^{-L}}$ and $n\tilde{\gamma}_{2n}$ terms correspond to errors from applying HQR to the blocks in the initial partition and to the blocks in levels 1 through L respectively. We can easily replace these with analogous mixed precision terms and keep the analysis accurate. Both level-2 and level-3 BLAS implementations will be considered in [section 4](#).

4. Mixed precision error analysis. Let us first consider rounding errors incurred from carrying out HQR in high precision, then cast down at the very end. This could be useful in applications that require economical storage but have enough memory to carry out HQR in higher precision, or in block algorithms as will be shown in [subsections 4.1](#) and [4.2](#). Consider two floating point types \mathbb{F}_l and \mathbb{F}_h where $\mathbb{F}_l \subseteq \mathbb{F}_h$, and for all $x, y \in \mathbb{F}_l$, the exact product xy can be represented in \mathbb{F}_h . Some example pairs of $\{\mathbb{F}_l, \mathbb{F}_h\}$ include $\{\text{fp16}, \text{fp32}\}$, $\{\text{fp32}, \text{fp64}\}$, and $\{\text{fp16}, \text{fp64}\}$. Suppose that the matrix to be factorized is stored with low precision numbers, $\mathbf{A} \in \mathbb{F}_l^{m \times n}$. Casting up adds no rounding errors, so we can directly apply the analysis that culminated in [Theorem 3.4](#), and we only consider the columnwise forward error in the \mathbf{Q} factor. Then, the j^{th} column of $\hat{\mathbf{Q}}_{HQR} = \mathbf{Q} + \Delta\mathbf{Q}_{HQR}$ is bounded normwise via $\|\Delta\mathbf{Q}_{HQR}[:, j]\|_2 \leq n\tilde{\gamma}_m^h$, and incurs an extra

603 rounding error when $\mathbf{Q} \in \mathbb{F}_h^{m \times n}$ is cast down to $F_l^{m \times n}$. Consider the backward error of a cast down
 604 operation represented by a linear transformation,

605 First, consider casting down a vector $\mathbf{x}^{(h)} \in \mathbb{F}_h^m$ to \mathbb{F}_l^m . We see that

$$606 \quad (4.1) \quad \mathbf{x}^{(l)} := \text{castdown}(\mathbf{x}^{(h)}) = \mathbf{I}_l \mathbf{x}^{(h)} = (\mathbf{I} + \mathbf{E}) \mathbf{x}^{(h)} = \mathbf{x}^{(h)} + \Delta \mathbf{x},$$

607 where $|\Delta \mathbf{x}| \leq u^{(l)} |\mathbf{x}^{(h)}|$ and $\|\Delta \mathbf{x}\|_2 \leq u^{(l)} \|\mathbf{x}^{(h)}\|_2$. Then, $\mathbf{E} = \Delta \mathbf{x} \mathbf{x}^\top / \|\mathbf{x}\|_2^2$ and we can use the same
 608 argument as in (3.12) to form a backward matrix norm bound,

$$609 \quad (4.2) \quad \|\mathbf{E}\|_F \leq u^{(l)}.$$

610 Using this in Lemma 3.2 to analyze the forward norm error for the j^{th} column of the \mathbf{Q} factor
 611 computed with alg. 3 yields

$$612 \quad (4.3) \quad \|\text{castdown}(\hat{\mathbf{Q}}_{HQR}[:, j]) - \mathbf{Q}[:, j]\|_2 = \|\mathbf{I}_l \hat{\mathbf{P}}_1 \cdots \hat{\mathbf{P}}_n \hat{\mathbf{e}}_j\|_2 \leq u^{(l)} + n \tilde{\gamma}_m^{(h)}.$$

613 To convert this bound to the lower precision, we define function d ,

$$614 \quad (4.4) \quad d(m, u^{(h)}, q, u^{(l)}) := \lceil (qu^{(l)} + mu^{(h)})/u^{(l)} \rceil = \mathcal{O}(q + mu^{(h)}/u^{(l)}),$$

615 so that if $\|\hat{\mathbf{x}} - \mathbf{x}\|_2 \leq \gamma_m^{(h)}$, then $\|\text{castdown}(\hat{\mathbf{x}}) - \mathbf{x}\|_2 \leq \gamma_{d(m, u^{(h)}, q, u^{(l)})}^{(l)}$. This is a looser bound but
 616 it allows us to easily compare the errors to the uniform, low precision implementation of forming $\hat{\mathbf{x}}$.

617 We can easily apply the operator $\mathbf{I}^{(l)}$ to cast down the QR factorizations computed via BQR
 618 and TSQR to find the forward matrix norm error on the \mathbf{Q} factor as shown below,

$$619 \quad \|\hat{\mathbf{Q}}_{BQR}\|_F \leq u^{(l)} + n \tilde{\gamma}_m^{(h)} \leq \tilde{\gamma}_{d(nm, u^{(h)}, u^{(l)})}^{(l)},$$

$$620 \quad \|\hat{\mathbf{Q}}_{TSQR}\|_F \leq u^{(l)} + n(L \tilde{\gamma}_{2n}^{(h)} + \tilde{\gamma}_{m2-L}^{(h)}) \leq \tilde{\gamma}_{d(n(L2n+m2-L), u^{(h)}, 1, u^{(l)})}^{(l)}.$$

622 In the next sections, we consider performing BQR and TSQR with FLOPs within a block and/or a
 623 level in high precision, but cast down to low precision in between blocks in 4.1. Finally, we consider
 624 all 3 algorithms with an ad hoc mixed precision setting where inner products are performed in high
 625 precision and all other operations are computed in low precision in 4.2.

626 **4.1. Round down at block-level (BLAS-3).** We use (4.3) to study the errors for BQR and
 627 TSQR in the case that each block QR factorized by HQR then cast down.

4.1.1. Round down at block level: BQR. Let us consider a setting in which only M
 blocks of width r can be loaded onto memory. Then, lines 2-6 of alg. 5 can be modified via alg. 7.
 We now impose a mixed-precision setting where the inner for-loop in alg. 7 is performed in high
 precision, but the WY updates for the outer loop is stored in low precision and only M blocks is
 updated at a time due to the memory constraint. These low precision WY updates would be used
 to build the \mathbf{Q} factor serially in groups of M . Then, the j^{th} column of the \mathbf{Q} factor computed in
 this mixed-precision BQR algorithm is computed via

$$\hat{\mathbf{Q}}_{mpBQR}[:, j] = \left(\prod_{q'=1}^q (\hat{\mathbf{X}}_{(q'-1)M+1} \cdots \hat{\mathbf{X}}_{q'M} \mathbf{I}^{(l)}) \right) \hat{\mathbf{e}}_j.$$

Algorithm 7: A portion of a mixed precision BQR: modifying first for-loop in [alg. 5](#).

```

1  $q = N/M;$  /* Note that  $n = Nr = qMr$ . */
2 for  $q' = 1 : q$  do
3   if  $q' > 2$  then
4      $\lfloor$  Update  $[\mathbf{C}_{(q'-1)M+1} \cdots \mathbf{C}_{qM}]$  with WY updates from blocks  $1 : (q' - 1)M$ .
5   for  $k = 1 : M$  do
6     Apply HQR to  $\mathbf{C}_{(q'-1)M+k}$ ;
7     Form WY update for  $\mathbf{C}_{(q'-1)M+k}$ ;
8     WY update blocks to the right,  $[\mathbf{C}_{(q'-1)M+k+1} \cdots \mathbf{C}_{q'M}]$ .

```

Applying [Lemma 3.2](#), we result in bounds

$$(4.5) \quad \|\hat{\mathbf{Q}}_{mpBQR}[:, j] - \mathbf{Q}[:, j]\|_2 \leq q(u^{(l)} + Mr\tilde{\gamma}_m^{(h)})$$

$$(4.6) \quad \|\hat{\mathbf{Q}}_{mpBQR} - \mathbf{Q}\|_F \leq n^{1/2}q(u^{(l)} + Mr\tilde{\gamma}_m^{(h)}),$$

showing that q number of cast downs add $\gamma_q^{(l)}$ order errors to columnwise bounds and the matrix norm bound is derived from that. This mixed precision BQR variant still rich in level-3 BLAS operations can be implemented on TensorCore technology. Furthermore, the bounds in (4.5) and (4.6) show that the loss in precision that can occur from cast downs are linear to the number of cast downs.

4.1.2. Round down at block level: TSQR. Let us now consider a WY variant of TSQR, where all instances of `qr` (lines 6,7,9 of [alg. 6](#)) are followed by `buildWY` (see [alg. 4](#)), and all instances of `hh_mult` is replaced by a WY update (line 6 of [alg. 5](#)). We additionally impose a mixed precision assumption similar to [section 4.1.1](#), where we store all WY representations of HQR within the for-loop (lines 4-8) of [alg. 6](#) in low precision, and consider the construction of the \mathbf{Q} factor. We can assume that each $2n$ -by- n and $m2^{-L}$ -by- n size matrices can fit into memory and only introduce one cast down for each $\mathbf{Q}_j^{(i)}$ block, where $i = 1 : L - 1$ and $j = 1 : 2^{i-1}$. Let us compute lines 9-10 in the higher precision, which introduces an error of order $n\tilde{\gamma}_{2n}^{(h)}$. In levels $L - 1$ to 1, each WY update adds error $u^{(l)} + n\tilde{\gamma}_{2n}^{(h)}$, and the final construction at the 0^{th} level (line 16), the WY update adds error $u^{(l)} + n\tilde{\gamma}_{m2^{-L}}^{(h)}$. Adding the cast down operator $\mathbf{I}^{(l)}$ to the analysis in [\[18\]](#) and applying [Lemma 3.2](#) yields the bounds on $\hat{\mathbf{Q}}_{mpTSQR}$,

$$(4.7) \quad \|\hat{\mathbf{Q}}_{mpTSQR}[:, j] - \mathbf{Q}[:, j]\|_2 \leq L(u^{(l)} + n\tilde{\gamma}_{2n}^{(h)}) + \tilde{\gamma}_{m2^{-L}}$$

$$(4.8) \quad \|\hat{\mathbf{Q}}_{mpTSQR} - \mathbf{Q}\|_F \leq n^{1/2}(L(u^{(l)} + n\tilde{\gamma}_{2n}^{(h)}) + \tilde{\gamma}_{m2^{-L}}).$$

4.2. Round down at inner-product. While the previous section discussed blocked variants of HQR that can be easily adapted for the mixed precision setting specific to TensorCore's level-3 BLAS operations, we want to provide a more general mixed precision environment in this section. Recall that HQR, BQR, and TSQR all rely on Householder transformations in one way or another, and Householder transformations are essentially performed via [\(3.9\)](#). This implementation capitalizes on the rank-1 update structure of Householder transformations where the predominant share of FLOPs is spent on an inner product, and computing the Householder vector and constant also

rely heavily on inner products. Therefore, we can attribute nearly all of the computational tasks for [algs. 3, 5 and 6](#) to the inner product. In addition, the inner product is just as important in non-HQR linear algebra tools, where some examples include projections and matrix-vector, matrix-matrix multiply. Consequently, we return to the mixed precision setting described in [section 2](#), where every inner product is cast down to the lower precision as shown in [\(2.11\)](#).

4.2.1. Round down at inner product: HQR. Consider forming a Householder transformation that zeros out $\mathbf{x} \in \mathbb{R}^m$ below the i^{th} element. We need to compute σ , β , $\tilde{\mathbf{v}}_1$, and \mathbf{v} as defined in [subsection 3.1](#):

$$(4.9) \quad \text{fl}(\sigma) = \text{fl}(-\text{sign}(\mathbf{x}[1])\|\mathbf{x}\|_2) = \sigma + \Delta\sigma, \quad |\Delta\sigma| \leq (\gamma_2^{(l)} + \gamma_m^{(h)})|\sigma|,$$

$$(4.10) \quad \text{fl}(\tilde{\mathbf{v}}_1) = \tilde{\mathbf{v}}_1 + \Delta\tilde{\mathbf{v}}_1 = (1 + \delta^{(l)})(\mathbf{x}_1 - \sigma - \Delta\sigma), \quad |\Delta\tilde{\mathbf{v}}_1| \leq (\gamma_3^{(l)} + \gamma_m^{(h)})|\tilde{\mathbf{v}}_1|$$

$$(4.11) \quad \text{fl}(\beta) = \beta + \Delta\beta = (1 + \delta^{(l)}) \left(-\mathbf{v}[1]/\hat{\sigma} \right), \quad |\Delta\beta| \leq (\gamma_4^{(l)} + \tilde{\gamma}_m^{(h)})|\beta|,$$

$$(4.12) \quad \text{fl}(\mathbf{v}_j) = \mathbf{v}_j + \Delta\mathbf{v}_j \text{ where } |\Delta\mathbf{v}_j| \leq \begin{cases} 0, & j = 1 \\ (\gamma_4^{(l)} + \tilde{\gamma}_m^{(h)})|\mathbf{v}_j|, & j = 2 : m - i + 1. \end{cases}$$

Using these, we can formulate the mixed precision version of [\(3.10\)](#) where $\hat{\mathbf{y}} = \text{fl}(\mathbf{P}_\mathbf{v}\mathbf{x}) \in \mathbb{R}^m$ is computed with [\(3.9\)](#),

$$(4.13) \quad \hat{\mathbf{y}} = \mathbf{y} + \Delta\mathbf{y}, \quad \|\Delta\mathbf{y}\|_2 \leq (\gamma_7^{(l)} + \tilde{\gamma}_m^{(h)})\|\mathbf{y}\|_2.$$

Thus, a backward error can be formed using $\Delta\mathbf{P}_\mathbf{v} = \Delta\mathbf{y}\mathbf{x}^\top / \|\mathbf{x}\|_2^2$,

$$(4.14) \quad \hat{\mathbf{y}} = (\mathbf{P}_\mathbf{v} + \Delta\mathbf{P}_\mathbf{v})\mathbf{x}, \quad \|\Delta\mathbf{P}_\mathbf{v}\|_F \leq (\gamma_7^{(l)} + \tilde{\gamma}_m^{(h)}).$$

The error bounds for applying n Householder transformations to \mathbf{x} can be found using [Lemma 3.2](#),

$$(4.15) \quad \hat{\mathbf{y}} = \mathbf{Q}(\mathbf{x} + \Delta\mathbf{x}) = (\mathbf{Q} + \Delta\mathbf{Q})\mathbf{x},$$

$$(4.16) \quad \|\Delta\mathbf{y}\|_2 \leq (\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)})\|\mathbf{x}\|_2, \quad \|\Delta\mathbf{Q}\|_F \leq (\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)}),$$

where the leading order error is now $\mathcal{O}(\gamma_n^{(l)})$. The analogous mixed precision QR factorization error bounds are shown in [Theorem 4.1](#).

THEOREM 4.1. *Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ with $m \geq n$ have full rank, n . Let $\hat{\mathbf{Q}}_{mpHQR} \in \mathbb{R}^{m \times n}$ and $\hat{\mathbf{R}} \in \mathbb{R}_{mpHQR}^{n \times n}$ be the thin QR factors of \mathbf{A} obtained via [alg. 3](#) with mixed precision FLOPs where inner products are computed in precision h then cast down. All other operations are carried out in precision l . Then,*

$$\|\Delta\mathbf{R}_{mpHQR}[:, j]\|_2 \leq (\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)})\|\mathbf{A}[:, j]\|_2, \quad \|\Delta\mathbf{R}_{mpHQR}\|_F \leq (\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)})\|\mathbf{A}\|_F$$

$$\|\Delta\mathbf{Q}[:, j]_{mpHQR}\|_2 \leq (\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)}), \quad \|\Delta\mathbf{Q}_{mpHQR}\|_F \leq n^{1/2}(\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)}).$$

4.2.2. Round down at inner product: BQR. Now, we analyze [alg. 5](#) with the same mixed precision inner product. At the k^{th} block, we first apply the mixed precision HQR summarized in [Theorem 4.1](#). Next, we study updating \mathbf{W}_k^{j-1} to \mathbf{W}_k^j . Since this update is applied with a single Householder transformation to the right, we can apply [\(4.16\)](#) to form

$$(4.17) \quad \hat{\mathbf{z}}_k^{(j)} = \mathbf{z}_k^{(j)} + \Delta\mathbf{z}_k^{(j)}, \quad |\Delta\mathbf{z}_k^{(j)}| \leq (\tilde{\gamma}_j^{(l)} + j\tilde{\gamma}_{m-(k-1)}^{(h)})|\mathbf{z}_k^{(j)}|,$$

and the error for $\hat{\mathbf{v}}_k^{(j)}$ is [\(4.12\)](#) with m replaced by $m - (k - 1)r$.

4.2.3. Round down at inner product: TSQR.

5. Numerical Experiments.

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.

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.

REFERENCES

- [1] A. ABDEFATTAH, 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), May 2019, pp. 111–122, <https://doi.org/10.1109/IPDPS.2019.00022>.
- [2] J. APPEYARD AND S. YOKIM, *Programming Tensor Cores in CUDA 9*, 2017, <https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/> (accessed 2018-07-30).
- [3] G. BALLARD, J. W. DEMMEL, L. GRIGORI, M. JACQUELIN, H. DIEP NGUYEN, AND E. SOLOMONIK, *Reconstructing Householder vectors from tall-skinny QR*, vol. 85, 05 2014, pp. 1159–1170, <https://doi.org/10.1109/IPDPS.2014.120>.
- [4] C. BISCHOF AND C. VAN LOAN, *The WY Representation for Products of Householder Matrices*, SIAM Journal on Scientific and Statistical Computing, 8 (1987), pp. s2–s13, <https://doi.org/10.1137/0908009>.
- [5] M. COURBARIAUX, Y. BENGIO, AND J.-P. DAVID, *Training deep neural networks with low precision multiplications*, arXiv preprint, arXiv:1412.7024, (2014).
- [6] M. COURBARIAUX, J.-P. DAVID, AND Y. BENGIO, *Low precision storage for deep learning*, arXiv preprint arXiv:1412.7024, (2014).
- [7] J. DEMMEL, I. DUMITRIU, AND O. HOLTZ, *Fast linear algebra is stable*, Numerische Mathematik, 108 (2007), pp. 59–91, <https://doi.org/10.1007/s00211-007-0114-x>, <https://arxiv.org/abs/0612264>.
- [8] J. DEMMEL, L. GRIGORI, M. HOEMMEN, AND J. LANGOU, *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM Journal on Scientific Computing, 34 (2012), <https://doi.org/10.1137/080731992>, <https://arxiv.org/abs/0808.2664>.
- [9] G. H. GOLUB AND C. F. VAN LOAN, *Matrix computations*, JHU press, 4 ed., 2013.

- 743 [10] A. HAIDAR, A. ABDELFAH, M. ZOUNON, P. WU, S. PRANESH, S. TOMOV, AND J. DONGARRA, *The Design*
744 *of Fast and Energy-Efficient Linear Solvers: On the Potential of Half-Precision Arithmetic and Iterative*
745 *Refinement Techniques*, June 2018, pp. 586–600, https://doi.org/10.1007/978-3-319-93698-7_45.
- 746 [11] A. HAIDAR, S. TOMOV, J. DONGARRA, AND N. J. HIGHAM, *Harnessing GPU tensor cores for fast fp16 arithmetic*
747 *to speed up mixed-precision iterative refinement solvers*, in Proceedings of the International Conference
748 for High Performance Computing, Networking, Storage, and Analysis, SC '18, Piscataway, NJ, USA,
749 2018, IEEE Press, pp. 47:1–47:11, <https://doi.org/10.1109/SC.2018.00050>, [https://doi.org/10.1109/SC.](https://doi.org/10.1109/SC.2018.00050)
750 [2018.00050](https://doi.org/10.1109/SC.2018.00050).
- 751 [12] N. J. HIGHAM, *Accuracy and Stability of Numerical Methods*, 2002, <https://doi.org/10.2307/2669725>.
- 752 [13] N. J. HIGHAM AND T. MARY, *A New Approach to Probabilistic Rounding Error Analysis*, SIAM Journal on
753 Scientific Computing, 41 (2019), pp. A2815–A2835, <https://doi.org/10.1137/18M1226312>, [https://epubs.](https://epubs.siam.org/doi/10.1137/18M1226312)
754 [siam.org/doi/10.1137/18M1226312](https://doi.org/10.1137/18M1226312).
- 755 [14] N. J. HIGHAM AND S. PRANESH, *Simulating Low Precision Floating-Point Arithmetic*, SIAM Journal on Sci-
756 entific Computing, 41 (2019), pp. C585–C602, <https://doi.org/10.1137/19M1251308>, [https://epubs.siam.](https://epubs.siam.org/doi/10.1137/19M1251308)
757 [org/doi/10.1137/19M1251308](https://doi.org/10.1137/19M1251308).
- 758 [15] A. S. HOUSEHOLDER, *Unitary triangularization of a nonsymmetric matrix*, Journal of the ACM (JACM), 5
759 (1958), pp. 339–342.
- 760 [16] I. C. F. IPSEN AND H. ZHOU, *Probabilistic Error Analysis for Inner Products*, (2019), [http://arxiv.org/abs/](http://arxiv.org/abs/1906.10465)
761 [1906.10465](http://arxiv.org/abs/1906.10465), <https://arxiv.org/abs/1906.10465>.
- 762 [17] P. MICIKEVICIUS, S. NARANG, J. ALBEN, G. DIAMOS, E. ELSEEN, D. GARCIA, B. GINSBURG, M. HOUSTON,
763 O. KUCHAIEV, G. VENKATESH, AND H. WU, *Mixed precision training*, in International Conference on
764 Learning Representations, 2018, <https://openreview.net/forum?id=r1gs9JgRZ>.
- 765 [18] D. MORI, Y. YAMAMOTO, AND S. L. ZHANG, *Backward error analysis of the AllReduce algorithm for householder*
766 *QR decomposition*, Japan Journal of Industrial and Applied Mathematics, 29 (2012), pp. 111–130, [https:](https://doi.org/10.1007/s13160-011-0053-x)
767 [//doi.org/10.1007/s13160-011-0053-x](https://doi.org/10.1007/s13160-011-0053-x).
- 768 [19] R. SCHREIBER AND C. VAN LOAN, *A Storage-Efficient \$WY\$ Representation for Products of Householder*
769 *Transformations*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 53–57, [https:](https://doi.org/10.1137/0910005)
770 [//doi.org/10.1137/0910005](https://doi.org/10.1137/0910005).
- 771 [20] G. TAGLIAVINI, S. MACH, D. ROSSI, A. MARONGIU, AND L. BENIN, *A transprecision floating-point platform for*
772 *ultra-low power computing*, in 2018 Design, Automation Test in Europe Conference Exhibition (DATE),
773 March 2018, pp. 1051–1056, <https://doi.org/10.23919/DATE.2018.8342167>.