ROUNDING ERROR ANALYSIS OF MIXED PRECISION BLOCK 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 [14] 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 [19], 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) 2-4$\times$ speedup in solving dense large linear equations [12, 13], (ii) 12$\times$ speedup in training dense neural networks, and (iii) 1.2-10$\times$ 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 [7] and storage [8] 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 TensorCores perform matrix products of half precision input data with exact products and single precision (32 bits) summation accumulate [3]. The existing rounding error analyses are built within what we call a *uniform precision* setting, which is the assumption that all arithmetic operations and storage are performed via the same precision. In this work, we

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.

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

However, additional analysis is needed as the additional round-off error will effect orthogonality, and thus the accuracy of the solution. Here, we focus on analyzing specific algorithms in a specific set of types (IEEE754 half (fp16), single (fp32), and double(fp64)), but the framework we develop could be used on different algorithms or different floating point types (such as bfloat16 in [22]).

This work discusses several aspects of using mixed precision arithmetic: (i) error analysis that can more accurately describe mixed precision arithmetic than existing analyses, (ii) algorithmic design that is more resistant against lower numerical stability associated with lower precision types, and (iii) an example where mixed precision implementation performs as sufficiently as double-precision implementations. Our key findings are that the new mixed precision error analysis produces tighter error bounds, that some block QR algorithms by Demmel et al. [10] are able to operate in low precision more robustly than non-block techniques, and that some small-scale benchmark graph clustering problems can be successfully solved with mixed precision arithmetic.

2. Background: Build up to rounding error analysis for inner products.

In this section, we introduce the basic motivations and tools for mixed precision rounding error analysis needed for the *QR factorization*. A matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ for $m \geq n$ can be written as

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

where an orthogonal $\mathbf{Q} \in \mathbb{R}^{m\times m}$ and an upper trapezoidal $\mathbf{R}$ form a *full* QR factorization, and $\mathbf{Q}_1 \in \mathbb{R}^{m\times n}, \mathbf{R}_1 \in \mathbb{R}^{n\times n}$ form a *thin* QR factorization. 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 triangular. In many applications, computing the *thin* decomposition requires less computation and is sufficient in performance. While important definitions are stated explicitly in the text, Table 1 serves to establish basic notation.

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
$\mathrm{fl}(\mathbf{x})$, $\hat{\mathbf{x}}$	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: $\lvert\delta^{(q)}\rvert < u^{(q)}$	2.1
$\gamma_k^{(q)}$, $\theta_k^{(q)}$	$\frac{ku^{(q)}}{1-ku^{(q)}}$, Quantity bounded by: $\lvert\theta_k^{(q)}\rvert \leq \gamma_k^{(q)}$	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 for 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, shown in Table 2. Let $\mathbb{F} \subset \mathbb{R}$ denote the space of some floating point number system with base $b \in \mathbb{N}$, precision $t \in \mathbb{N}$, significand $\mu \in \mathbb{N}$, and exponent range $[\eta_{\min}, \eta_{\max}] \subset \mathbb{Z}$. Then every element y in $\mathbb{F}$ can be written as

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

where μ is any integer in $[0, b^t - 1]$ and η is an integer in $[\eta_{\min}, \eta_{\max}]$. 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 [14], which allow a relatively simple framework for formulating error bounds for complex linear algebra operations. An analysis of FLOPs (see Theorem 2.2 [14]) 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	$\eta_{\min}$	$\eta_{\max}$	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 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, $\mathrm{fl}(x \text{ op } y)$, 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:

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

We use (2.2) as a building block in accumulating errors from successive FLOPs. Successive operations introduce multiple rounding error terms, and keeping track of all errors is challenging. Lemma 2.1 introduces a convenient and elegant bound that simplifies accumulation of rounding error.

LEMMA 2.1 (Lemma 3.1 [14]). *Let $|\delta_i| < u$, $\rho_i = \pm 1$ for $i = 1 : k$, and $ku < 1$. Then,*

$$\prod_{i=1}^{k} (1+\delta_i)^{\rho_i} = 1 + \theta_k, \qquad \text{where } |\theta_k| \le \frac{ku}{1-ku} =: \gamma_k. \tag{2.3}$$

Additionally, we define $\tilde{\theta}_k$ that satisfies $|\tilde{\theta}_k| \le \tilde{\gamma}_k$, where $\tilde{\gamma}_k = \dfrac{cku}{1-cku}$ for a small integer, $c > 0$.

In other words, θ_k represents the accumulation of rounding errors from k successive operations, and it is bounded by γ_k. In more complicated routines shown in later sections, we use the tilde notation ($\tilde{\gamma}_k$) to permit only keeping track of the leading order error terms. Applying this lemma to the computation of $x + y + z$, where $x, y, z \in \mathbb{R}$, results in

$$\mathrm{fl}(x+y+z) = (1+\delta')\left((1+\delta)(x+y) + z\right) = (1+\theta_2)(x+y) + (1+\theta_1)z, \tag{2.4}$$

where $|\delta|, |\delta'| < u$. Since $|\theta_1| \leq \gamma_1 < \gamma_2$, we can further simplify (2.4) to

$$\mathrm{fl}(x+y+z) = (1+\theta_2')(x+y+z), \quad \text{where} \quad |\theta_2'| \leq \gamma_2, \tag{2.5}$$

at the cost of a slightly larger upper bound. Note that both $|\theta_2|, |\theta_2'|$ are bounded above by γ_2. Typically, error bounds formed in the fashion of (2.5) are converted to relative errors in order to put the error magnitudes in perspective. The relative error bound for our example is

$$|(x+y+z) - \mathrm{fl}(x+y+z)| \leq \gamma_2 |x+y+z|$$

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

Although Lemma 2.1 requires $ku < 1$, we actually need $ku < \frac{1}{2}$ to maintain a meaningful relative error bound as this assumption implies $\gamma_k < 1$ and guarantees a relative error below 100%. Since higher precision types have smaller unit round-offs, they can tolerate more successive FLOPs than lower precision floating types before reaching $\gamma_m = 1$. For example, the IEEE types introduced in Table 2 meet this requirement at $1/2 = 2^{10}u^{(\text{fp16})} = 2^{23}u^{(\text{fp32})} = 2^{52}u^{(\text{fp64})}$. Thus, accumulated rounding errors in lower precision types can lead to an instability with fewer operations in comparison to higher precision types and prompts 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

$$|\mathbf{x}^\top \mathbf{y} - \mathrm{fl}(\mathbf{x}^\top \mathbf{y})| \leq \gamma_m |\mathbf{x}|^\top |\mathbf{y}|, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^m \tag{2.6}$$

as shown in [14]. 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: $\left| \|\mathbf{x}\|_2^2 - \mathrm{fl}(\|\mathbf{x}\|_2^2) \right| \leq \gamma_m \|\mathbf{x}\|_2^2$. 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 ($\gamma_{512}^{(\text{fp16})} = 1$).

A simple numerical experiment 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 fp16 vectors of length 1024 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 [16], for every FLOP (multiplication and addition for the dot product). The relative error in this experiment is formulated as the LHS in Equation 2.6 divided by $|\mathbf{x}|^\top |\mathbf{y}|$ and all operations outside of calculating $\mathrm{fl}(\mathbf{x}^\top \mathbf{y})$ are executed by casting up to fp64 and using fp64 arithmetic. Table 3 shows some statistics from computing the relative error for simulated fp16 dot products of random 1024-length vectors.

We see that the inner products of vectors sampled from the standard normal distribution have backward relative errors that do not deviate much from the unit round-off ($\mathcal{O}($`1e-4`$)$), whereas the vectors sampled from the uniform distribution tend to accumulate larger errors on average ($\mathcal{O}($`1e-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 [15, 18]) have shown that

Random Distribution	Average	Stan. Dev.	Maximum
Standard normal	1.621e-04	1.635e-04	3.204e-03
Uniform $(0,1)$	6.904e-03	3.265e-03	2.447e-02

TABLE 3
Forward error statistics from experiment of dot products computed in simulated half precision.

the inner product relative backward error for the conditions used for this experiment is bounded by `5.466e-2` with probability 0.99.

Algorithm 1: $\mathbf{z}^{(\text{fp16})} = \texttt{simHalf}(f, \mathbf{x}^{(\text{fp16})}, \mathbf{y}^{(\text{fp16})})$. Given fp16 input variables $\mathbf{x}, \mathbf{y}$, perform function $f \in \text{OP} \cup \{\texttt{dot\_product}\}$ in simulated fp16 arithmetic.

```
  Input: $\mathbf{x}^{(\text{fp16})}, \mathbf{y}^{(\text{fp16})}, f$          Output: $\mathbf{z}^{(\text{fp16})} = \text{fl}_{\text{fp16}}(f(\mathbf{x}^{(\text{fp16})}, \mathbf{y}^{(\text{fp16})}))$
1 $[\mathbf{x}^{(\text{fp32})}, \mathbf{y}^{(\text{fp32})}] \leftarrow$ castup($[\mathbf{x}^{(\text{fp16})}, \mathbf{y}^{(\text{fp16})}]$)     // Convert input vars to fp32.
2 $\mathbf{z}^{(\text{fp32})} \leftarrow \text{fl}(f(\mathbf{x}^{(\text{fp32})}, \mathbf{y}^{(\text{fp32})}))$          // Perform fp32 arithmetic.
3 $\mathbf{z}^{(\text{fp16})} \leftarrow$ castdown($\mathbf{z}^{(\text{fp32})}$)          // Convert result to fp16.
4 return $\mathbf{z}^{(\text{fp16})}$
```

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 [14] 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 $|\theta_k| \leq \frac{ku}{1-ku} =: \gamma_k$. The following relations hold for positive integers j, n and nonnegative integer k. Arithmetic operations between bounded terms, θ_k's, are:*

$$(1+\theta_k)(1+\theta_j) = (1+\theta_{k+j}) \quad \text{and} \quad \frac{1+\theta_k}{1+\theta_j} = \begin{cases} 1+\theta_{k+j}, & j \leq k \\ 1+\theta_{k+2j}, & j > k \end{cases}. \tag{2.7}$$

If $\max_{(\text{j,k})} \text{u} \leq \frac{1}{2}$ and $n \leq \frac{1}{uk}$, the operations on the bounds, γ's, are:

$$\gamma_k\gamma_j \leq \gamma_{\text{min(k,j)}}, \qquad n\gamma_k \leq \gamma_{nk},$$
$$\gamma_k + u \leq \gamma_{k+1}, \qquad \gamma_k + \gamma_j + \gamma_k\gamma_j \leq \gamma_{k+j}.$$

Note that all the rules hold when replaced by $\tilde{\gamma}$'s, but result in looser bounds.

We define two mixed precision settings that we use in section 4. In subsection 4.1, we present the block Fused Multiply-Add (bFMA) of NVIDIA's TensorCore (TC) technology, which computes matrix-matrix multiply and accumulate for 4-by-4 blocks, and incorporate it into algs. 5 and 6. Here, we introduce an ad hoc mixed precision setting which we use in subsection 4.2. This is explicitly defined in Assumption 2.3 and is a level-2 BLAS variant of the TC bFMA. Both mixed precision settings define how inner products are computed although the bFMA is only applicable to inner products within matrix products and uses fp16 and fp32 whereas our ad hoc mixed precision setting is applicable to all inner products with any two precision types.

Although our analysis concerns accuracy and stability and leaves out timing results of various hardwares, we add a general timing statement to Assumption 2.3 that is analogous to that of TC: the mixed precision FMA inner product performs at least 2 times faster than the inner product in the higher precision. Note that TCs perform matrix-matrix multiply and accumulate up to 8 times faster than fp32, and up to 16 times faster than fp64 (see), and our ad hoc timing assumption is in conservative in comparison. Nonetheless, this gives a vague insight into the trade-offs between speediness and accuracy from some mixed precision computations.

The full precision multiplication in Assumption 2.3 is exact when the low precision type is fp16 and the high precision type of fp32 due to their precisions 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 $\mu_x, \mu_y \in [0, 2^{11}-1]$ and $\eta_x, \eta_y \in [-15, 16]$, and note that the significand and exponent ranges for fp32 are $[0, 2^{24}-1]$ and $[-127, 128]$. Then the product in full precision is

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

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 and hypothesize that the guaranteed accuracy for this mixed precision inner product should be better than that of the low precision inner product and worse than that of the high precision inner product. Let $\mathbf{x}^{(l)}, \mathbf{y}^{(l)}$ be m-length vectors stored in a low precision type ($\mathbb{F}_l^m$), s_k be the exact k^{th} partial sum, and $\hat{s_k}$ be s_k computed with FLOPs. Then the first three partial sums are,

$$\hat{s}_1 = \text{fl}(\mathbf{x}[1]\mathbf{y}[1]) = \mathbf{x}[1]\mathbf{y}[1], \quad \hat{s}_2 = \text{fl}(\hat{s_1} + \mathbf{x}[2]\mathbf{y}[2]) = (\mathbf{x}[1]\mathbf{y}[1] + \mathbf{x}[2]\mathbf{y}[2])\,(1+\delta_1^{(h)}),$$
$$\hat{s}_3 = \text{fl}(\hat{s_2} + \mathbf{x}[3]\mathbf{y}[3]) = \left[(\mathbf{x}[1]\mathbf{y}[1] + \mathbf{x}[2]\mathbf{y}[2])\,(1+\delta_1^{(h)}) + \mathbf{x}[3]\mathbf{y}[3]\right](1+\delta_2^{(h)}).$$

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

$$\hat{s}_m = (\mathbf{x}[1]\mathbf{y}[1] + \mathbf{x}[2]\mathbf{y}[2]) \prod_{k=1}^{m-1}(1+\delta_k^{(h)}) + \sum_{i=3}^{n} \mathbf{x}[i]\mathbf{y}[i] \left(\prod_{k=i-1}^{m-1}(1+\delta_k^{(h)})\right). \tag{2.8}$$

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

$$\text{fl}(\mathbf{x}^\top\mathbf{y}) = (\mathbf{x}+\Delta\mathbf{x})^\top\mathbf{y} = \mathbf{x}^\top(\mathbf{y}+\Delta\mathbf{y}) \quad \text{for } |\Delta\mathbf{x}| \le \gamma_{m-1}^{(h)}|\mathbf{x}|,\ \ |\Delta\mathbf{y}| \le \gamma_{m-1}^{(h)}|\mathbf{y}|. \tag{2.9}$$

Casting down to $\mathbb{F}_l$ without underflow or overflow results in backward errors,

$$\texttt{castdown}(\text{fl}(\mathbf{x}^\top\mathbf{y})) = (\mathbf{x}+\Delta\mathbf{x}+\tilde{\Delta}\mathbf{x})^\top\mathbf{y} = \mathbf{x}^\top(\mathbf{y}+\Delta\mathbf{y}+\tilde{\Delta}\mathbf{y}), \tag{2.10}$$

where $|\Delta\mathbf{x}+\tilde{\Delta}\mathbf{x}| \le ((1+u^{(l)})(1+\gamma_{m-1}^{(h)})-1)|\mathbf{x}|$ and $|\Delta\mathbf{y}+\tilde{\Delta}\mathbf{y}| \le ((1+u^{(l)})(1+\gamma_{m-1}^{(h)})-1)|\mathbf{y}|$. Our hypothesis is indeed true since,

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

where the lower and upper bounds are derived from the uniform precision error bound in (2.6). Equation (2.10) shows us that the two larger error terms are from the higher precision summation, $\gamma^{(h)}_{m-1}$, and the cast down operation, $u^{(l)}$. We can measure the impact of the cast down step 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 $u^{(l)}$. The mixed precision inner product has a smaller worst case error bound than the bound of the low precision inner product ($mu^{(l)}$) with no apparent improvements in speed. On the other hand, $u^{(l)}$ is a larger upper bound than that of the high precision inner product ($mu^{(h)} = \frac{m}{M_{l,h}}u^{(l)}$), although it was computed approximately twice as fast. It is likely that this factor of $M_{l,h}/m$ increase in the worst case error bound is unwanted even when considering the speed-up.
Case 2: ($m = M_{l,h}$) Both terms are now leading order. This is still an improvement in comparison to the lower precision arithmetic as the error bound is reduced from $mu^{(l)}$ to $2u^{(l)}$. Comparing this to the high precision inner product shows that the error bound has doubled from $mu^{(h)}$ to $2mu^{(h)}$, but gained a factor of 2 in speed instead. One can argue that the loss in accuracy guarantee and the improvement in speed cancel each other out especially if $2mu^{(h)} \ll 1$ or if the speed-up greatly exceeds a factor of 2.
Case 3: ($m \gg M_{l,h}$) Now $\gamma^{(h)}_{m-1}$ 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 $\gamma^{(l)}_m$ to $\gamma^{(l)}_{m/M_{l,h}} \equiv \gamma^{(h)}_m$. Since $u^{(l)} = M_{l,h}u^{(h)} \ll mu^{(h)}$, the mixed precision error bound has the same *order* as the error bound from carrying the computation out in the higher precision. Therefore, we can expect about the same level of accuracy but a factor of 2 or greater reduction in speed when compared to the higher precision.

While the above cases establish 3 regimes of trade-offs between accuracy and speed in mixed precision computing, the remainder of this paper focuses only on accuracy and does not consider the impact of mixed precision computations on speed. Readers should refer to timing studies such as ..... Finally, we present alternative representations of the error bound in (2.10),

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

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

where the rules from Lemma 2.2 were directly applied. Both alternative bounds are only slightly larger than the original bound shown on the LHS and remain in the same order. The first is useful when comparing against the low or the high precision, whereas the second keeps track of the error bounds in both precisions. We summarize these ways of combining γ terms of different precisions in Lemma 2.4,

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 $\gamma^{(q)}_k := \frac{ku^{(q)}}{1-ku^{(q)}}$. Consider a low precision and a high precision where $1 \gg u^{(l)} \gg u^{(h)} > 0$, and k_l, k_h that satisfy $\max\{\gamma^{(h)}_{k_h}, \gamma^{(l)}_{k_l}\} < 1/2$. Then the following rules help us accumulate γ's of different precisions,*

$$\gamma^{(h)}_{k_h}\gamma^{(l)}_{k_l} \leq \min\{\gamma^{(h)}_{k_h}, \gamma^{(l)}_{k_l}\}, \tag{2.11}$$

$$(1+\tilde{\gamma}^{(l)}_{k_l})(1+\tilde{\gamma}^{(h)}_{k_h})-1 = \tilde{\gamma}^{(l)}_{k_l} + \tilde{\gamma}^{(h)}_{k_h}. \tag{2.12}$$

Note that (2.12) drops the term $\tilde{\gamma}_{k_l}^{(l)}\tilde{\gamma}_{k_h}^{(h)}$ since both $\tilde{\gamma}_{k_l}^{(l)}$ and $\tilde{\gamma}_{k_h}^{(h)}$ are larger than their product and this product can be swept under the small integer $c > 0$ assumption implicitly included in the tilde notation. Equations (2.9) and (2.10) are crucial for our analysis in section 4 since the two mixed precision settings add `castdown` operations at different parts of the HQR algorithms we consider. In general, error bounds in the fashion of (2.9) can be used before the cast down operations and the action of the cast down is best represented by error bounds similar to (2.10).

We have demonstrated a need for rounding error analysis that is accurate for mixed precision procedures and analyzed the inner product in an ad hoc mixed precision inner product that mimics the TensorCore bFMA. We will use this to analyze various Householder (HH) QR factorization algorithms. Algorithms and the general framework for the standard rounding error analysis for these algorithms are introduced in section 3, and both are modified to meet 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 is a well-known algorithm that uses the WY representation of [5] that utilizes mainly level-3 BLAS operations. 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 [9]). We also present the standard rounding error analysis of these algorithms which will be tweaked for various mixed precision assumptions in section 4.

3.1. Householder QR (HQR).

The HQR algorithm uses HH transformations to zero out elements below the diagonal of a matrix (see [17]). 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 a HH vector , $\mathbf{v}$, and a HH constant, β, that define the HH transformation matrix, $\mathbf{P_v} := \mathbf{I}_m - \beta\mathbf{v}\mathbf{v}^\top$, such that $\mathbf{P_v}$ zeros out $\mathbf{x}$ below the first element. The HH vector and constant are defined via*

$$\sigma = -\mathrm{sign}(\mathbf{x}[1])\|\mathbf{x}\|_2, \quad \mathbf{v} = \mathbf{x} - \sigma\hat{\mathbf{e}}_1, \text{ and } \beta = \frac{2}{\mathbf{v}^\top\mathbf{v}} = -\frac{1}{\sigma\mathbf{v}[1]}. \tag{3.1}$$

Note that $\mathbf{P_v}$ is symmetric and orthogonal, $\mathbf{P_v} = \mathbf{P_v}^\top = \mathbf{P_v}^{-1}$. As a result, the transformed vector, $\mathbf{P_v x} = \sigma\hat{\mathbf{e}_1}$, has the same 2-norm as $\mathbf{x}$.

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 $i = 1, 2, \cdots, n$,
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 $\mathbf{P_v}$ 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 [14]. Let $\mathbf{P}_i$ represent the i^{th} HH transformation of this algorithm.

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

The resulting matrix is the $\mathbf{R}$ factor, $\mathbf{R} := \mathbf{P}_3\mathbf{P}_2\mathbf{P}_1\mathbf{A}$, and 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. The thin factors for a general matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ are

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

Algorithm 2: β, $\mathbf{v}$, σ = `hhvec`($\mathbf{x}$). Given a vector $\mathbf{x} \in \mathbb{R}^m$, return $\mathbf{v} \in \mathbb{R}^m$ and $\beta, \sigma \in \mathbb{R}$ that satisfy $(I - \beta \mathbf{v}\mathbf{v}^\top)\mathbf{x} = \sigma \hat{\mathbf{e}}_1$ and $\mathbf{v}[1] = 1$ (see [2, 14]).

```
  Input: $\mathbf{x}$                                   Output: $\mathbf{v}$, $\sigma$, and $\beta$
1 $\mathbf{v} \leftarrow \texttt{copy}(\mathbf{x})$
2 $\sigma \leftarrow -\mathrm{sign}(\mathbf{x}[1])\|\mathbf{x}\|_2$
3 $\mathbf{v}[1] \leftarrow \mathbf{x}[1] - \sigma$
4 $\beta \leftarrow -\frac{\mathbf{v}[1]}{\sigma}$
5 $\mathbf{v} \leftarrow \mathbf{v}/\mathbf{v}[1]$
6 return $\beta$, $\mathbf{v}$, $\sigma$
```

Algorithm 3: $\mathbf{V}$, $\boldsymbol{\beta}$, $\mathbf{R}$ = `HQR2`($\mathbf{A}$). A Level-2 BLAS implementation of HQR. 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 $\boldsymbol{\beta} \in \mathbb{R}^n$, and upper triangular matrix $\mathbf{R}$. The orthogonal factor $\mathbf{Q}$ can be generated from $\mathbf{V}$ and $\boldsymbol{\beta}$.

```
  Input: $\mathbf{A}$                                   Output: $\mathbf{V}$,$\boldsymbol{\beta}$, $\mathbf{R}$
1 Initialize $\mathbf{V} \leftarrow \mathbf{0}_{m\times n}$, $\boldsymbol{\beta} \leftarrow \mathbf{0}_m$
2 for $i = 1 : n$ do
3     $\mathbf{v}, \beta, \sigma \leftarrow \texttt{hhvec}(\mathbf{A}[i : \mathrm{end}, i])$        /* Algorithm 2 */
4     $\mathbf{V}[i : \mathrm{end}, i]$, $\boldsymbol{\beta}_i$, $\mathbf{A}[i, i] \leftarrow \mathbf{v}, \beta, \sigma$
5     $\mathbf{A}[i + 1 : \mathrm{end}, i] \leftarrow \mathrm{zeros}(m - i)$
6     $\mathbf{A}[i : \mathrm{end}, i + 1 : \mathrm{end}] \leftarrow \mathbf{A}[i : \mathrm{end}, i + 1 : \mathrm{end}] - \beta \mathbf{v}\mathbf{v}^\top \mathbf{A}[i : \mathrm{end}, i + 1 : \mathrm{end}]$
7 return $\mathbf{V}$, $\boldsymbol{\beta}$, $\mathbf{A}[1 : n, 1 : n]$
```

3.1.2. HQR: Rounding Error Analysis. Now we present an error analysis for alg. 3 by keeping track of the different operations of alg. 2 and alg. 3.

Calculating the i^{th} HH vector and constant. In alg. 3, we compute the HH vector and constant by using alg. 2 to $\mathbf{A}[i : m, i]$. For now, consider zeroing out any vector $\mathbf{x} \in \mathbb{R}^m$ below its first component with a HH transformation. We first calculate σ as is implemented in line 2 of alg. 2.

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

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

$$\mathrm{fl}(\mathbf{v}'[1]) = \mathbf{v}'[1] + \Delta\mathbf{v}'[1] = (1 + \delta)(\mathbf{x}[i] - \sigma - \Delta\sigma) = (1 + \theta_{m+2})\mathbf{v}'[1] \tag{3.4}$$

where the last equality is granted because the sign of σ is chosen to prevent cancellation. Since alg. 2 normalizes the HH vector so that its first component is 1, the remaining components of $\mathbf{v}$ are divided by $\mathrm{fl}(\tilde{\mathbf{v}}_1)$ incurring another single rounding error. As a result, the components of $\mathbf{v}$ computed with FLOPs have error $\mathrm{fl}(\mathbf{v}[j]) = \mathbf{v}[j] + \Delta\mathbf{v}[j]$ where

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

and $|\Delta\mathbf{v}[1]| = 0$. 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 HH constant, β, as is computed in line 4 of alg. 2.

$$\hat{\beta} = \text{fl}\left(-\mathbf{v}'[1]/\hat{\sigma}\right) = -(1+\delta)\frac{\mathbf{v}'[1]+\Delta\mathbf{v}'[1]}{\sigma+\Delta\sigma} = \frac{(1+\delta)(1+\theta_{m+2})}{(1+\theta_{m+1})}\beta \tag{3.6}$$

$$= (1+\theta_{3m+5})\beta = \beta + \Delta\beta, \text{ where } |\Delta\beta| \leq \tilde{\gamma}_m\beta. \tag{3.7}$$

We have shown (3.6) to keep our analysis simple in section 4 and (3.7) 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} HH constant and vector $(\hat{\beta}_i,\mathbf{v}_i)$ both have errors bounded by $\tilde{\gamma}_{m-i+1}$.

Applying a Single HH 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 HH transformation to the remaining columns, $\mathbf{A}[i:m,i+1:n]$ with the computed HH 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 HH 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 slightly different but efficient implementations of this step. For example, BQR in alg. 5 uses level-3 BLAS operations to apply multiple HH transformations at once whereas the variant of HQR in alg. 3 can only use level-2 BLAS operations to apply HH transformations.

A HH transformation is applied through a series of inner and outer products, since HH 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

$$\mathbf{y} := \mathbf{P}_{\mathbf{v}}\mathbf{x} = \mathbf{x} - (\beta\mathbf{v}^\top\mathbf{x})\mathbf{v}. \tag{3.8}$$

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, $\mathbf{x}$ is any 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 $i-1$ HH transformations that zeroed out components below $\mathbf{A}[j,j]$ for $j = 1:i-1$. We show the error for forming $\hat{\mathbf{w}}$ where $\mathbf{w} := \beta\mathbf{v}^\top\mathbf{x}\mathbf{v}$ and $\mathbf{v},\mathbf{x} \in \mathbb{R}^m$,

$$\hat{\mathbf{w}} = \text{fl}(\hat{\beta}\ \text{fl}(\hat{\mathbf{v}}^\top\mathbf{x})\hat{v}) = (1+\theta_m)(1+\delta)(1+\delta')(\beta+\Delta\beta)(\mathbf{v}+\Delta\mathbf{v})^\top\mathbf{x}(\mathbf{v}+\Delta\mathbf{v}),$$

where θ_m is from computing the inner product $\hat{\mathbf{v}}^\top\mathbf{x}$, and δ and δ' are from multiplying β, $\text{fl}(\hat{\mathbf{v}}^\top\mathbf{x})$, and $\hat{\mathbf{v}}$. The forward error is

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

Subtracting $\hat{\mathbf{w}}$ from $\mathbf{x}$ yields the HH transformation with forward error,

$$\text{fl}(\hat{\mathbf{P}_{\mathbf{v}}}\mathbf{x}) = \text{fl}(\mathbf{x}-\hat{\mathbf{w}}) = (1+\delta)(\mathbf{x}-\mathbf{w}-\Delta\mathbf{w}) = \mathbf{y}+\Delta\mathbf{y} = (\mathbf{P}_{\mathbf{v}}+\Delta\mathbf{P}_{\mathbf{v}})\mathbf{x}, \tag{3.9}$$

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 form a normwise bound,

$$\|\Delta\mathbf{y}\|_2 \leq \tilde{\gamma}_m\|\mathbf{x}\|_2. \tag{3.10}$$

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

$$\|\Delta\mathbf{P_v}\|_F = \left(\sum_{i=1}^{m}\sum_{j=1}^{m}\left(\frac{1}{\|\mathbf{x}\|_2^2}\Delta\mathbf{y}[i]\mathbf{x}[j]\right)^2\right)^{1/2} = \frac{\|\Delta\mathbf{y}\|_2}{\|\mathbf{x}\|_2} \leq \tilde{\gamma}_m, \tag{3.11}$$

where the last inequality is a direct application of (3.10).

Applying many successive HH 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 HH transformations computed with $\tilde{\mathbf{v}}_i$'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 consecutive 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 $\hat{\mathbf{y}} = \mathrm{fl}(\hat{\mathbf{P}}_r\cdots\hat{\mathbf{P}}_2\hat{\mathbf{P}}_1\mathbf{x})$, where $\hat{\mathbf{P}}_k$'s are HH transformations constructed from $\hat{\beta}_k$ and $\hat{\mathbf{v}}_k$. These HH vectors and constants are computed via alg. 2 and the rounding errors are bounded by (3.5) and (3.7). If each transformation is computed via (3.8), then*

$$\hat{\mathbf{y}} = \mathbf{Q}(\mathbf{x}+\Delta\mathbf{x}) = (\mathbf{Q}+\Delta\mathbf{Q})\mathbf{x} = \hat{\mathbf{Q}}\mathbf{x}, \tag{3.12}$$

$$\|\Delta\mathbf{y}\|_2 \leq r\tilde{\gamma}_m\|\mathbf{x}\|_2, \quad \|\Delta\mathbf{Q}\|_F \leq r\tilde{\gamma}_m. \tag{3.13}$$

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

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

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 $r\tilde{\gamma}_m \leq 1/2$. ∎

In this error analysis, the prevailing bound for errors at various stages of forming and applying a HH 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 HH 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 a conversion of columnwise bounds to a matrix norm bound,

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

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,*

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

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. The numerical experiments in subsection 5.4 measure backward and forward errors with $\|\hat{\mathbf{Q}}\hat{\mathbf{R}} - \mathbf{A}\|_F$ and $\|\hat{\mathbf{Q}}^\top\hat{\mathbf{Q}} - \mathbf{I}\|_2$.

The content of this section shows the standard rounding error analysis in [14] where some important stages are summarized in (3.5), (3.7), and (3.13), 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 HH constant and vector, as well as normwise errors of the action of applying HH 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 [5] instead of the storage-efficient version of [21], 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 HH reflectors is known as the WY representation (see [5, 11]). Lemma 3.5 shows how to update a rank-j update of the identity, $\mathbf{Q}^{(j)}$, with a HH 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 HH transformations as shown in Algorithm 4. This algorithm assumes that the HH vectors, $\mathbf{V}$, and constants,$\boldsymbol{\beta}$, have already been computed. Since the $\mathbf{Y}$ factor is exactly $\mathbf{V}$, we only need to compute the $\mathbf{W}$ factor.

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 $\mathbf{W}^{(j)}, \mathbf{Y}^{(j)} \in \mathbb{R}^{m\times j}$. 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,*

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

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

In HQR, $\mathbf{A}$ is transformed into an upper triangular matrix $\mathbf{R}$ by identifying a HH transformation that zeros out a column below the diagonal, then applying that HH transformation to the bottom right partition. For example, the k^{th} HH transformation finds an $m-k+1$ sized HH transformation that zeros out column k below the diagonal and then applies it to the $(m-k+1)$-by-$(n-k)$ partition of the matrix, $\mathbf{A}[k:m, k+1:n]$. Since the $k+1^{st}$ column is transformed by the k^{th} HH transformation, this algorithm must be executed serially as shown in alg. 3. The highest

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

```
    Input: $\mathbf{V} \in \mathbb{R}^{m \times r}$, $\boldsymbol{\beta} \in \mathbb{R}^r$ where $m > r$.
    Output: $\mathbf{W}$
1   Initialize: $\mathbf{W} := \boldsymbol{\beta}_1 \mathbf{V}[:, 1]$.                    /* $\mathbf{Y}$ is $\mathbf{V}$. */
2   for $j = 2 : r$ do
3       $\mathbf{z} \leftarrow \boldsymbol{\beta}_j \left[\mathbf{V}[:, j] - \mathbf{W}\left(\mathbf{V}[:, 1 : j-1]^\top \mathbf{V}[:, j]\right)\right]$
4       $\mathbf{W} \leftarrow [\mathbf{W} \;\; \mathbf{z}]$          /* Update $\mathbf{W}$ to an $m$-by-$j$ matrix. */
5   return $\mathbf{W}$
```

computational burden at each iteration falls on alg. 3 line 6, which requires Level-2 BLAS operations when computed efficiently.

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

$$\mathbf{X}_k = \mathbf{I}_m - \mathbf{W}_k \mathbf{Y}_k^\top = \mathbf{P}_k^{(1)} \cdots \mathbf{P}_k^{(r)}. \tag{3.14}$$

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 $[\mathbf{C}_2 \cdots \mathbf{C}_N]$ with two Level-3 BLAS operations as shown in line 5 of alg. 5. BQR performs approximately $1 - \mathcal{O}(1/N)$ fraction of its FLOPs in Level-3 BLAS operations (see section 5.2.3 of [11]), and can reap the benefits from the accelerated block FMA feature of TensorCore. Note that BQR does require strictly more FLOPs when compared to HQR, but these additional FLOPs are negligble in standard precision and does not impact the numerical stability. A pseudoalgorithm for BQR is shown in alg. 5 where we assume that $n = Nr$ to make our error analysis in section 3.2.2 simple. In practice, an efficient implementation might require r to be a power of two or a product of small prime factors and result a thinner N^{th} block compared to the rest. This discrepancy is easily fixed by padding the matrix with zeros, a standard procedure for standard algorithms like the Fast Fourier Transform (FFT). For any variable $x \in \{\mathbf{X}, \mathbf{W}, \mathbf{Y}, \mathbf{z}, \beta, \mathbf{v}, \mathbf{P}\}$, $x_k^{(j)}$ corresponds to the j^{th} update for the k^{th} block.

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 W factor, and 3) updating the remaining blocks with the WY representation. We have adapted the analysis from [14] 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 HH transformation on the right side in the same efficient implementation that is discussed in (3.8),

$$\hat{\mathbf{X}_k^{(j)}} = \hat{\mathbf{X}}_k^{(j-1)} - \mathrm{fl}(\hat{\beta}_k^{(j-1)} \hat{\mathbf{X}}_k^{(j-1)} \hat{\mathbf{v}}_k^{(j-1)}) \hat{\mathbf{v}}_k^{(j)\top} = \hat{\mathbf{X}}_k^{(j-1)} (\mathbf{P}_k^{(j)} + \Delta \mathbf{P}_k^{(j)}), \tag{3.15}$$

Algorithm 5: $\mathbf{Q}, \mathbf{R} \leftarrow$ `blockHQR`$(\mathbf{A}, r)$: Perform HH 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     $\mathbf{V}_i, \boldsymbol{\beta}_i, \mathbf{C}_i \leftarrow$ hhQR$(\mathbf{C}_i)$                 /* Algorithm 3 */
4     $\mathbf{W}_i \leftarrow$ buildWY$(\mathbf{V}_i, \boldsymbol{\beta}_i)$                 /* Algorithm 4 */
5     $[\mathbf{C}_{i+1} \cdots \mathbf{C}_N]$ -= $\mathbf{V}_i \left(\mathbf{W}_i^\top [\mathbf{C}_{i+1} \cdots \mathbf{C}_N]\right)$    /* 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     $\mathbf{Q}[(i-1)r+1 : m, (i-1)r+1 : n]$-= $\mathbf{W}_i \left(\mathbf{V}_i^\top \mathbf{Q}[(i-1)r+1 : m, (i-1)r+1 : n]\right)$
9 return $\mathbf{Q}, \mathbf{A}$
```

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 (3.15) and use Lemma 3.2 to find

$$\|\Delta \mathbf{X}_k^{(j)}\|_F \leq j \tilde{\gamma}_{m-(k-1)r}. \tag{3.16}$$

HQR within each block: line 3 of alg. 5. We apply Algorithm 3 to the k^{th} block, $\hat{\mathbf{X}}_{k-1} \cdots \hat{\mathbf{X}}_1 \mathbf{C}_k$, which applies r more HH transformations to columns that had been transformed by $(k-1)$ WY 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

$$\|\hat{\mathbf{R}}_{BQR}[:, j] - \mathbf{R}[:, j]\|_2 \leq r \tilde{\gamma}_m \|\hat{\mathbf{X}}_{k-1} \cdots \hat{\mathbf{X}}_1^\top \mathbf{C}_k[:, j]\|_2, \quad j = (k-1)r + 1 : kr.$$

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 HH 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 HH constants, $\hat{\beta}_k^{(j)}$ are bounded by (3.7) modified similarly. Thus, $\mathbf{z}_k^{(j)}$ is the only newly computed quantity. Using (3.5), (3.7), and (3.16), we find

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

Componentwise bounds follow immediately, and are summarized in 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 HH constants and vectors, $\{\beta_k^{(j)}\}_{j=1}^{r}$ and $\{\mathbf{v}_k^{(j)}\}$ via alg. 4. Then,*

$$\hat{\mathbf{z}}_k^{(j)} = \mathbf{z}_k^{(j)} + \Delta\mathbf{z}_k^{(j)},\ \ |\Delta\mathbf{z}_k^{(j)}| \leq j\tilde{\gamma}_{m-(k-1)r}|\mathbf{z}_k^{(j)}| \tag{3.17}$$

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

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 HH transformations (see Section 19.5 of [14]).

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 (3.16) directly to the columns of $\mathbf{B}$,

$$\|\mathrm{fl}(\hat{\mathbf{X}}_k\mathbf{B}[:,j])\|_2 = \|\mathrm{fl}(\hat{\mathbf{X}}_k^{(r)}\mathbf{B}[:,j])\|_2 \leq r\tilde{\gamma}_{m-(k-1)r}\|\mathbf{B}[:,j]\|_2 \tag{3.19}$$

A normwise bound for employing a general matrix-matrix multiplication operation is stated in section 19.5 of [14].

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 HH 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,

$$\left\|\prod_{k=1}^{N}(\mathbf{X}_k + \Delta\mathbf{X}_k) - \prod_{k=1}^{N}\mathbf{X}_k\right\|_F \leq \left(-1 + \sum_{k=1}^{N}(1 + r\tilde{\gamma}_{m-(k-1)r})\right) \leq rN\tilde{\gamma}_m \equiv n\tilde{\gamma}_m, \tag{3.20}$$

$$\|\hat{\mathbf{Q}}_{BQR} - \mathbf{Q}\|_F \leq n^{3/2}\tilde{\gamma}_m. \tag{3.21}$$

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

$$\|\Delta\mathbf{R}_{BQR}[:,j']\|_2 \leq rk'\tilde{\gamma}_m\|\mathbf{A}[:,j]\|_2 \tag{3.22}$$

$$\|\Delta\mathbf{Q}_{BQR}[:,j']\|_2 \leq rk'\tilde{\gamma}_m \tag{3.23}$$

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

$$\|\Delta\mathbf{Q}_{BQR}\|_F = r\tilde{\gamma}_m\sum_{j=1}^{n}\lceil j/r\rceil = n^{3/2}\tilde{\gamma}_m. \tag{3.24}$$

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 [11, 14] 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 [20]. In this paper, the TSQR/AllReduce algorithm refers to the most parallel variant of the block QR factorization algorithms discussed in [10]. A detailed description and rounding error analysis of this algorithm can be found in [20], 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 subsection 5.4) 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 HH constants and vectors stored at each level. The most gains from parallelization can be made in the initial level where the maximum number of independent HQR factorizations occur. Although more than one configuration of this algorithm may be available for a given tall-and-skinny matrix, the number of nodes available and the shape of the matrix eliminate some of those choices. For example, a 1600-by-100 matrix can be partitioned into 2, 4, 8, or 16 initial row-blocks but may be restricted by a machine with only 4 nodes, and a 1600-by-700 matrix can only be partitioned into 2 initial blocks. Our numerical experiments show that the choice in the initial partition, which directly relates to the recursion depth of TSQR, has an impact in the accuracy of the QR factorization.

We refer to *level* as the number of recursions in a particular TSQR implementation. An L-level TSQR algorithm partitions the original matrix into $2^{(l)}$ submatrices in the initial or 0^{th} level of the algorithm, and 2^{L-i} QR factorizations are performed in level i for $i = 1, \cdots, L$. The set of matrices that are QR factorized at each level i are called $\mathbf{A}_j^{(i)}$ for $j = 1, \cdots, 2^{L-i}$, where superscript (i) corresponds to the level and the subscript j indexes the row-blocks within level i. In the following sections, 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 below: $\tilde{\mathbf{Q}}_{\alpha(j),\phi(j)}^{(i+1)}$. For $j = 1, \cdots, 2^{L-i}$ at level i, we need $j = 2(\alpha(j) - 1) + \phi(j)$, where $\alpha(j) = \lceil \frac{j}{2} \rceil$ and $\phi(j) = 2 + j - 2\alpha(j) \in \{1, 2\}$. section 3.3.2 shows full linear algebra details for a single-level ($L = 1$, 2 initial blocks) example. The reconstruction of $\mathbf{Q}$ can be implemented more efficiently (see

[4]), but the reconstruction method in alg. 6 is presented for a clear, straightforward explanation.

Algorithm 6: $\mathbf{Q},\mathbf{R} = \texttt{tsqr}(\mathbf{A}, L)$. Finds a QR factorization of a tall, skinny matrix, $\mathbf{A}$.

```
    Input: $\mathbf{A} \in \mathbb{R}^{m\times n}$ where $m \gg n$, $L \le \lfloor \log_2\left(\frac{m}{n}\right)\rfloor$, 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 HH vectors as columns of matrix $\mathbf{V}_j^{(i)}$, HH constants as components of
       vector $\boldsymbol{\beta}_j^{(i)}$, and set up the next level.                  */
 4  for $i = 0 : L-1$ do
        /* The inner loop can be parallelized.                                  */
 5      for $j = 1 : 2^{L-i}$ do
 6          $\mathbf{V}_{2j-1}^{(i)}, \boldsymbol{\beta}_{2j-1}^{(i)}, \mathbf{R}_{2j-1}^{(i)} \leftarrow \texttt{qr}(\mathbf{A}_{2j-1}^{(i)})$
 7          $\mathbf{V}_{2j}^{(i)}, \boldsymbol{\beta}_{2j}^{(i)}, \mathbf{R}_{2j}^{(i)} \leftarrow \texttt{qr}(\mathbf{A}_{2j}^{(i)})$
 8          $\mathbf{A}_j^{(i+1)} \leftarrow \begin{bmatrix} \mathbf{R}_{2j-1}^{(i)} \\ \mathbf{R}_{2j}^{(i)} \end{bmatrix}$
 9  $\mathbf{V}_1^{(L)}, \boldsymbol{\beta}_1^{(L)}, \mathbf{R} \leftarrow \texttt{qr}(\mathbf{A}_1^{(L)})$                // The final $\mathbf{R}$ factor is built.
10  $\mathbf{Q}_1^{(L)} \leftarrow \texttt{hh\_mult}(\mathbf{V}_1^{(L)}, I_{2n\times n})$
    /* Compute $\mathbf{Q}^{(i)}$ factors by applying $\mathbf{V}^{(i)}$ to $\mathbf{Q}^{(i+1)}$ factors.          */
11  for $i = L-1 : -1 : 1$ do
12      for $j = 1 : 2^{L-i}$ do
13          $\mathbf{Q}_j^{(i)} \leftarrow \texttt{hh\_mult}\left(\mathbf{V}_j^{(i)}, \begin{bmatrix} \tilde{\mathbf{Q}}_{\alpha(j),\phi(j)}^{(i+1)} \\ \mathbf{0} \end{bmatrix}\right)$
14  $\mathbf{Q} \leftarrow []$;                                   // Construct the final $\mathbf{Q}$ factor.
15  for $j = 1 : 2^L$ do
16      $\mathbf{Q} \leftarrow \begin{bmatrix} \mathbf{Q} \\ \texttt{hh\_mult}\left(\mathbf{V}_j^{(0)}, \begin{bmatrix} \tilde{\mathbf{Q}}_{\alpha(j),\phi(j)}^{(1)} \\ \mathbf{0} \end{bmatrix}\right) \end{bmatrix}$
17  return $\mathbf{Q}$, $\mathbf{R}$
```

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

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

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

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

More generally, 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 HH transformations given a set of HH 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-$n$, $\lceil \log_2(\frac{m}{n})\rceil$-by-$n$, and $2n$-by-n. The single-level TSQR strictly uses more FLOPs, but the dot product subroutines may accumulate smaller rounding errors (and certainly have smaller upper bounds) since they are performed on shorter vectors, and lead to a more accurate solution overall. These concepts are elucidated in [20] 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*

$$\|\hat{\mathbf{R}}_{TSQR}[:,j] - \mathbf{R}[:,j]\|_2 \leq n(\tilde{\gamma}_{m2^{-L}} + L\tilde{\gamma}_{2n})\|\mathbf{A}[:,j]\|_2, \tag{3.25}$$

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

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.

In this section, we consider three different mixed precision settings for the QR factorization, all of which take in a matrix $\mathbf{A}$ stored in low precision and return $\mathbf{Q}, \mathbf{R}$ both represented in low precision. First, we consider a trivial mixed precision setting where HQR, BQR, and TSQR are computed in high precision after casting up the input matrix at the beginning, and casting down the resulting high precision factors to low precision. Then in subsection 4.1, we modify BQR and TSQR to utilize level-3 BLAS operations and TensorCore bFMAs for the matrix product subroutines. Finally, we impose Assumption 2.3 in subsection 4.2 to see how a mixed precision inner product impacts HQR, BQR, and TSQR when applied in level-2 BLAS operations.

Backward error of casting down vectors. First, consider casting down a vector $\mathbf{x} \in \mathbb{F}_h^{(m)}$. The componentwise forward error is,

$$\texttt{castdown}_l(\mathbf{x}) = \mathbf{x} + \Delta\mathbf{x}, \ \ |\Delta\mathbf{x}| < u^{(l)}|\mathbf{x}|.$$

We use this to represent the backward error of a casting down a vector with a linear transformation, $\mathbf{I}^{(l)} := \mathbf{I} + \mathbf{E} \in \mathbb{R}^{m\times m}$, a diagonal perturbation of the identity. We write,

$$\mathbf{x}^{(l)} := \texttt{castdown}(\mathbf{x}^{(h)}) = \mathbf{I}^{(l)}\mathbf{x}^{(h)} = (\mathbf{I}+\mathbf{E})\mathbf{x}^{(h)} = \mathbf{x}^{(h)} + \Delta\mathbf{x}, \tag{4.1}$$

where $|\Delta\mathbf{x}| \leq u^{(l)}|\mathbf{x}^{(h)}|$ and $\|\Delta\mathbf{x}\|_2 \leq u^{(l)}\|\mathbf{x}^{(h)}\|_2$. Thus, $\mathbf{E} = \Delta\mathbf{x}\mathbf{x}^\top/\|\mathbf{x}\|_2^2$ and we can use the same argument as in (3.11) to form a backward matrix norm bound,

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

Casting down after HQR in high precision. Let us consider the trivial case of carrying out HQR in high precision and casting down at the very end. This is useful in for the analysis of mixed precision block algorithms as will be shown in subsections 4.1 and 4.2. If the two floating point types $\mathbb{F}_l$ and $\mathbb{F}_h$ satisfy $\mathbb{F}_l \subseteq \mathbb{F}_h$ and the matrix to be factorized is stored with low precision numbers, $\mathbf{A} \in \mathbb{F}_l^{m\times n}$, then casting up adds no rounding errors. Therefore, 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 rounding error when $\hat{\mathbf{Q}}_{HQR} \in \mathbb{F}_h^{m\times n}$ is cast down to $\mathbb{F}_l^{m\times n}$.

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

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

The final castdown operation increases the upper bound by $u^{(l)}$ and the size of $\mathbf{A}$ has no impact on this extra rounding error. Applying this trivial mixed precision setting to BQR and TSQR would simply increases the error bound by approximately $u^{(l)}$ all the while taking an even longer time than the high precision implementation due the extra cast down and cast up operations. Therefore, we do not analyze the rounding error analysis of this mixed precision variant of BQR and TSQR. However, we will use this mixed precision HQR as a subroutine of the mixed precision BQR and TSQR in the following section.

4.1. Round down at block-level (BLAS-3).

The mixed precision setting in this section is designed to meet the below requirements.

1. Modify Algorithms 5 and 6 to maximize level-3 BLAS operations and use TensorCore bFMAs.
2. Apply (4.3) to all instances of HQR to the error analyses for BQR and TSQR in section 3.
3. Cast down quantities at every block/level and the insertion of low precision errors $u^{(l)}$ should be somewhat correlated to the number of blocks and levels.
4. Both input and output of the various QR factorization algorithms are given in the low precision.

TensorCore's bFMA computes

$$\hat{\mathbf{D}} = \mathrm{fl}_{TC}(\mathbf{C} + \mathbf{A}\mathbf{B}), \qquad \mathbf{C},\mathbf{D} \in \mathbb{F}_{\text{fp16}}^{4\times 4} \text{ or } \mathbb{F}_{\text{fp32}}^{4\times 4}, \text{ and } \mathbf{A},\mathbf{B} \in \mathbb{F}_{\text{fp16}}^{4\times 4}, \tag{4.4}$$

and employs *full* precision products and fp32 summation accumulate. Here, the *full* precision multiplication is exact as explained in section 2. In [6], the authors investigate all four possible matrix-matrix multiplication routines in TensorCore, which depend on whether $\mathbf{C}$ and $\mathbf{D}$ are computed in fp16 or fp32. They also note that matrices larger than 4-by-4 can still be computed using this block FMA by accumulating matrix sums with $\mathbf{C} \in \mathbb{F}_{\text{fp32}}^{4\times 4}$. Suppose that we aim to compute a fp16 matrix product of two fp16 matrices, $\mathbf{X}\in\mathbb{F}^{m\times p}_{(fp16)}$, $\mathbf{Y}\in\mathbb{F}^{p\times n}_{(fp16)}$, and $\mathbf{Z}=\mathbf{X}\mathbf{Y}\in\mathbb{F}^{m\times n}_{\text{fp16}}$. We pad $\mathbf{X},\mathbf{Y}$ with zeros so that all matrix dimensions are multiples of 4 and the matrix product can be computed with the TensorCore block FMA. Let $\mathbf{Q}_{[i,j]}:=\mathbf{Q}[4(i-1)+1:4i,4(j-1)+1:4j]$ refer to the $(i,j)^{th}$ 4-by-4 block for any $\mathbf{Q}\in\{\mathbf{X},\mathbf{Y},\mathbf{Z}\}$. Then, we compute $\mathbf{Z}_{[i,j]}$ via

$$\mathbf{Z}_{[i,j]}=\sum_{k=1}^{\lceil p/4\rceil}\mathbf{X}_{[i,k]}\mathbf{Y}_{[k,j]},$$

where we use (4.4) by initializing with $\mathbf{A}^{(1)}:=\mathbf{X}_{[i,1]}$, $\mathbf{B}^{(1)}:=\mathbf{Y}_{[1,j]}$, and $\mathbf{C}^{(1)}:=\mathbf{0}_{4\times4}$ and setting $\mathbf{A}^{(k)}:=\mathbf{X}_{[i,k]}$, $\mathbf{B}^{(k)}:=\mathbf{Y}_{[k,j]}$, and $\mathbf{C}^{(k)}:=\mathbf{D}^{(k-1)}$ for $k=2:\lceil p/4\rceil$. By setting $\mathbf{C}^{(k)},\mathbf{D}^{(k)}\in\mathbb{F}^{4\times4}_{\text{fp32}}$ for $k>1$ and only casting down at the end via $\mathbf{Z}_{[i,j]}=\text{fp16}(\mathbf{D}^{(\lceil p/4\rceil)})$, we mostly employ fp32 arithmetic for a mixed precision matrix product routine whose inputs and output are in fp16. For example, take $p=8$. Then,

$$\mathbf{D}^{(1)}=\text{fl}_{TC}(\mathbf{X}_{[i,1]}\mathbf{Y}_{[1,j]}),\quad \mathbf{D}^{(2)}=\text{fl}_{TC}(\mathbf{X}_{[i,2]}\mathbf{Y}_{[2,j]}+\mathbf{D}^{(1)})\in\mathbb{F}^{4\times4}_{\text{fp32}}$$
$$\mathbf{Z}_{[i,j]}=\texttt{castdown}(\mathbf{D}^{(2)})\in\mathbb{F}^{4\times4}_{\text{fp16}}.$$

Adapting the rounding error analysis in [6] into this specific mixed precision matrix product setting yields the componentwise forward bound

$$|\mathbf{Z}-\text{fl}(\mathbf{Z})|\leq\left(u^{(\text{fp16})}+\gamma^{(\text{fp32})}_{p/4}+u^{(\text{fp16})}\gamma^{(\text{fp32})}_{p/4}\right)|\mathbf{X}||\mathbf{Y}|. \tag{4.5}$$

We denote BQR and TSQR computed via TensorCore bFMA's with `mpBQR3` and `mpTSQR3`, where the `3` represents the BLAS level-3 nature of this mixed precision setting.

4.1.1. Round down at block level: BQR. Consider the input matrix, $\mathbf{A}\in\mathbb{F}^{m\times n}_{l}$, partitioned into N blocks of r columns, $\mathbf{A}=[\mathbf{C}_1\cdots\mathbf{C}_N]$ as in subsection 3.2. Algorithm 7 shows a mixed precision variant of BQR that maximizes the use of bFMAs but uses high precision arithmetic for level-1 and 2 BLAS operations which are only a $\mathcal{O}(1/N)$ fraction of the total number of FLOPs. Each block is casted up to compute a high precision HQR and to form the WY representation. The WY representation is then casted down to low precision since the bFMAs require low precision inputs for matrix products, and the $\mathbf{R}$ factor from the high precision HQR can be casted down to return a low precision $\mathbf{R}$ factor at the very end. Since the cast down operations for the $\mathbf{R}$ factor and the WY representations occur at every block, we can expect columnwise error bound for alg. 7 to increase by approximately $Nu^{(l)}$ from the error bound for alg. 5.

Rounding Error Analysis. Since $\hat{\mathbf{W}}_k,\hat{\mathbf{Y}}_k$'s are computed with alg. 4 in high precision then cast down, the new low precision WY update is $\hat{\mathbf{X}}^{(l)}_k=\mathbf{I}-\mathbf{I}^{(l)}\hat{\mathbf{W}}_k\mathbf{I}^{(l)}\hat{\mathbf{V}}^{(\top)}_k$. Consider applying $\hat{\mathbf{X}}^{(l)}_k$ to some matrix stored in low precision, $\mathbf{B}$ using the TensorCore bFMAs. We analyze a single column $\mathbf{b}_j:=\mathbf{B}[:,j]\in\mathbb{F}^{m-(k-1)r}_{l}$ even though this operation is done on $\mathbf{B}$ as a whole. Let

$$\mathbf{I}^{(l)}\hat{\mathbf{W}}_k=(\mathbf{I}+\mathbf{E}_W)\hat{\mathbf{W}}_k,\quad \mathbf{I}^{(l)}\hat{\mathbf{Y}}_k=(\mathbf{I}+\mathbf{E}_Y)\hat{\mathbf{Y}}_k,$$

where $\mathbf{E}_W,\mathbf{E}_Y$ are diagonal and bounded componentwise by $u^{(l)}$. The rounding errors for forming $\mathbf{W}_k$ and $\mathbf{Y}_k$ remain the same since these are computed in high precision. Therefore, we first include errors introduced from casting down the WY representation and compute the matrix norm error of

Algorithm 7: $\hat{\mathbf{Q}}_{mpBQR3}, \hat{\mathbf{R}}_{mpBQR3} \leftarrow$ `mpBQR3`$(\mathbf{A}, r)$: Perform a mixed precision variant of BQR of low precision $\mathbf{A}$ with column partitions of size r. $\hat{\mathbf{Q}}_{mpBQR3}$,$\hat{\mathbf{R}}_{mpBQR3}$, and $\hat{\mathbf{A}}$ are represented in low precision. Matrix-matrix multiplication and accumulate operations in lines 10, 13, and 14 require low precision inputs but can return in either of the two precisions.

```
   Input: $\mathbf{A}$, $r$.                                         Output: $\hat{\mathbf{Q}}_{mpBQR3}$,$\hat{\mathbf{R}}_{mpBQR3}$
1  $N = \frac{n}{r}$                                                   /* Let $\mathbf{A} = [\mathbf{C}_1 \cdots \mathbf{C}_N]$. */
2  for $k = 1 : N-1$ do
3      $\mathbf{V}_k, \boldsymbol{\beta}_k, \mathbf{C}_k \leftarrow$ hhQR(castup($\mathbf{C}_k$))        /* Algorithm 3 in high precision. */
4      $\mathbf{C}_k \leftarrow$castdown ($\mathbf{C}_k$)                      /* Builds $\mathbf{R}$ factor in low precision. */
5      $\mathbf{W}_k \leftarrow$ buildWY($\mathbf{V}_k, \boldsymbol{\beta}_k$)                    /* Algorithm 4 in high precision */
6      $[\mathbf{V}_k, \mathbf{W}_k] \leftarrow$castdown($[\mathbf{V}_k, \mathbf{W}_k]$)
7      $[\mathbf{C}_{k+1} \cdots \mathbf{C}_N]$ -= $\mathbf{V}_k \left(\mathbf{W}_k^\top [\mathbf{C}_{k+1} \cdots \mathbf{C}_N]\right)$        /* returned in low precision */
8  $\mathbf{Q} \leftarrow \mathbf{I}$                    /* Build $\mathbf{Q}$: $\mathbf{I}_m$ if full QR, and $\mathbf{I}_{m\times n}$ if thin QR. */
9  for $k = N : -1 : 1$ do
       // All updates are returned in low precision.
10     $\mathbf{Q}[(k-1)r+1 : m, (k-1)r+1 : n]$-= $\mathbf{W}_k \left(\mathbf{V}_k^\top \mathbf{Q}[(k-1)r+1 : m, (k-1)r+1 : n]\right)$
11 return $\mathbf{Q}, \mathbf{A}$
```

forming $\hat{\mathbf{X}}_k^{(l)}$,

$$\begin{aligned}
\|\hat{\mathbf{X}}_k^{(l)} - \mathbf{X}_k\|_F &= \| - (\mathbf{I} + \mathbf{E}_W + \mathbf{E}_Y + \mathbf{E}_W\mathbf{E}_Y)\,\hat{\mathbf{W}}_k\hat{\mathbf{Y}}_k^\top + \mathbf{W}_k\mathbf{Y}_k^\top\|_F, \\
&\leq \left((1+\gamma_2^{(l)} + (u^{(l)})^2)r\tilde{\gamma}_{m-(k-1)r}^{(h)} + \gamma_2^{(l)} + (u^{(l)})^2\right)\|\mathbf{X}_k\|_F \\
&\leq \tilde{\gamma}_2^{(l)} + r\tilde{\gamma}_{m-(k-1)r}^{(h)} + r\tilde{\gamma}_2^{(l)}\tilde{\gamma}_{m-(k-1)r}^{(h)}.
\end{aligned}$$

Now, we consider the backward error of applying $\hat{\mathbf{X}}_k^{(l)}$ to $\mathbf{b}_j$ with the bFMA matrix product error bound from (4.5). The multiplication by $(\mathbf{I}^{(l)}\hat{\mathbf{Y}}_k)^\top$ yields backward error bounded by

$$\mathrm{fl}_{TC}((\mathbf{I}^{(l)}\hat{\mathbf{Y}}_k)^\top\mathbf{b}_j) = (\hat{\mathbf{Y}}_k + \Delta_{TC}\hat{\mathbf{Y}}_k)\mathbf{b}_j, \;\; |\Delta_{TC}\hat{\mathbf{Y}}_k| \leq u^{(l)} + \gamma^{(h)}_{\frac{m-(k-1)}{4}} + u^{(l)}\gamma^{(h)}_{\frac{m-(k-1)}{4}}|\hat{\mathbf{Y}}_k||\mathbf{b}_j|,$$

and the subsequent multiplication by $(\mathbf{I}^{(l)}\hat{\mathbf{W}}_k)$ and subtraction from $\mathbf{b}_j$ result in,

$$\begin{aligned}
&\mathrm{fl}_{TC}(\hat{\mathbf{X}}_k^{(l)}\mathbf{b}_j) = (\hat{\mathbf{X}}_k^{(l)} + \Delta^{(l)}\mathbf{X}_k)\mathbf{b}_j, \\
&\qquad |\Delta^{(l)}\mathbf{X}_k| \leq \left(\gamma_2^{(l)} + \gamma^{(h)}_{1+\frac{m-(k-2)r}{4}} + \gamma_2^{(l)}\gamma^{(h)}_{1+\frac{m-(k-2)r}{4}}\right)\left(|\mathbf{b}_j| + |\mathbf{I}^{(l)}\hat{\mathbf{W}}_k||\mathbf{I}^{(l)}\hat{\mathbf{Y}}_k|^\top|\mathbf{b}_j|\right).
\end{aligned}$$

Converting to a normwise error bound using the same logic from (3.9) and (3.10), we result in

$$\|\mathrm{fl}_{TC}(\hat{\mathbf{X}}_k^{(l)}\mathbf{b}_j) - \mathbf{X}_k\mathbf{b}_j\|_2 \leq (\tilde{\gamma}_2^{(l)} + r\tilde{\gamma}_{m-(k-1)r}^{(h)} + r\gamma_2^{(l)}\tilde{\gamma}_{m-(k-1)r}^{(h)})\|\mathbf{b}_j\|_2, \tag{4.6}$$

since the rounding errors from the bFMAs are small in comparison to the errors from casting down the WY representation built in high precision. The corresponding matrix error bound is

$$\|\mathrm{fl}_{TC}(\hat{\mathbf{X}}_k^{(l)}) - \mathbf{X}_k\|_F \leq \tilde{\gamma}_2^{(l)} + r\tilde{\gamma}_{m-(k-1)r}^{(h)} + r\tilde{\gamma}_2^{(l)}\tilde{\gamma}_{m-(k-1)r}^{(h)}. \tag{4.7}$$

We can finally compute the forward errors from implementing alg. 7. Consider the j^{th} column of the $\mathbf{Q}$ factor, which we denote with $\mathbf{q}_j := \hat{\mathbf{Q}}_{mpBQR3}[:,j]$, and let $k = \lfloor j/r \rfloor$. Invoking Lemma 3.2 with error bounds for $\mathrm{fl}_{TC}(\hat{\mathbf{X}}_k^{(l)})$'s in (4.7) results in columnwise error,

$$\|\Delta \mathbf{q}_j\|_2 \leq -1 + \prod_{k'=1}^{k}(1+\tilde{\gamma}_2^{(l)})(1+r\tilde{\gamma}_{m-(k'-1)r}^{(h)}) \tag{4.8}$$

$$\leq k\tilde{\gamma}_2^{(l)} + kr\tilde{\gamma}_m^{(h)} + k^2 r\tilde{\gamma}_2^{(l)}\tilde{\gamma}_m^{(h)}, \tag{4.9}$$

where $\Delta \mathbf{q}_j = (\mathrm{fl}_{TC}(\hat{\mathbf{X}}_1^{(l)})\cdots \mathrm{fl}_{TC}(\hat{\mathbf{X}}_k^{(l)}) - \mathbf{X}_1\cdots\mathbf{X}_k)\hat{\mathbf{e}}_j$. Summing over the columns to find a matrix norm error bound yields

$$\|\hat{\mathbf{Q}}_{mpBQR} - \mathbf{Q}\|_F \leq n^{1/2}\tilde{\gamma}_N^{(l)} + n^{(3/2)}\tilde{\gamma}_m^{(h)}, \tag{4.10}$$

where the summation of the third term in (4.9) is swept under the tilde notation in $n^{1/2}\tilde{\gamma}_N^{(l)}$. This bound shows that alg. 7 only adds $n^{1/2}\tilde{\gamma}_N^{(l)}$ order errors to the bounds in (3.24). Using that $u^{(l)} = M_{l,h}u^{(h)}$, this increase corresponds to a multiplicative factor shown below,

$$n^{1/2}\tilde{\gamma}_N^{(l)} + n^{(3/2)}\tilde{\gamma}_m^{(h)} \approx \left(1 + \frac{M_{l,h}}{rm}\right) n^{(3/2)}\tilde{\gamma}_m^{(h)}. \tag{4.11}$$

Therefore, the loss in accuracy due to mixed precision computing is relatively small when the disparity in precision ($M_{l,h}$) is small in comparison to the block size, mr. Whether this loss in accuracy in the worst-case scenario is worth the speed-ups from using mixed precision hardware is an open question that can be tackled in future research. We expect that the block size r, the dimension of the input matrix m, n, and hardware specificities will be contributing factors.

4.1.2. Round down at block level: TSQR. Unlike BQR which is rich in level-3 BLAS operations, the variant of TSQR in alg. 6 uses none. Therefore, we modify alg. 6 by replacing all instances of `hh_mult` with level-3 BLAS operations. We omit presenting the exact algorithm for mixed precision variant of TSQR in this paper, but consider computing the HQR of each block in high precision and build and store the WY representation of the HH transformations in low precision as we did in lines (3-6) of alg. 7. The low precision WY representation is then applied with TensorCore bFMAs when building the $\mathbf{Q}$ factor (lines 11-16 of alg. 6).

Rounding Error analysis. The analysis in [20] shows that each column of $\mathbf{Q}$ is transformed by n HH transformations of length $2n$ from levels $L:-1:1$, and another set of n HH transformations of length $m2^{-L}$ at level 0. Let us represent the WY representation at the j^{th} block of level i and its bFMA counterpart as $\mathbf{X}_j^{(i)}$ and $\mathrm{fl}_{TC}(\hat{\mathbf{X}}_j^{(i)})$. Then, we can use (4.7) to form backward error

$$\|\mathrm{fl}_{TC}(\hat{\mathbf{X}}_j^{(i)}) - \mathbf{X}_j^{(i)}\|_F \leq \tilde{\gamma}_2^{(l)} + n\tilde{\gamma}_{m'}^{(h)} + n\tilde{\gamma}_2^{(l)}\tilde{\gamma}_{m'}^{(h)}, \quad m' = \begin{cases} m2^{-L}, & i = 0 \\ 2n, & i = 1:L \end{cases}. \tag{4.12}$$

We can now modify the analysis in [20] by replacing $n\tilde{\gamma}_{m2^{-L}}$ and $n\tilde{\gamma}_{2n}$ with

$$(1+\tilde{\gamma}_2^{(l)})(1+n\tilde{\gamma}_{m2^{-L}}^{(h)}) - 1, \quad \text{and} \quad (1+\tilde{\gamma}_2^{(l)})(1+n\tilde{\gamma}_{2n}^{(h)}) - 1,$$

and apply Lemma 3.2. Then, the $\mathbf{Q}$ factor formed with this mixed precision variant of TSQR is denoted with $\hat{\mathbf{Q}}_{mpTSQR3}$ and its j^{th} column has rounding errors bounded by,

$$\|\hat{\mathbf{Q}}_{mpTSQR3}[:,j] - \mathbf{Q}[:,j]\|_2 \leq \tilde{\gamma}_{L+1}^{(l)} + n\left(L\tilde{\gamma}_{2n}^{(h)} + \tilde{\gamma}_{m2^{-L}}^{(h)}\right). \tag{4.13}$$

Summing up the columns for a matrix norm error bound, we result in

$$\|\hat{\mathbf{Q}}_{mpTSQR3} - \mathbf{Q}\|_F \leq n^{1/2}\tilde{\gamma}_{L+1}^{(l)} + n^{3/2}\left(L\tilde{\gamma}_{2n}^{(h)} + \tilde{\gamma}_{m2^{-L}}^{(h)}\right). \tag{4.14}$$

Once again, we convert the low precision rounding errors as a fraction of the original bound in (3.26) to quantify the impact of modifying alg. 6 to utilize bFMAs,

$$n^{1/2}\tilde{\gamma}_{L+1}^{(l)} + n^{3/2}\left(L\tilde{\gamma}_{2n}^{(h)} + \tilde{\gamma}_{m2^{-L}}^{(h)}\right) = \left(1 + \frac{M_{l,h}L}{n(2nL+m2^{-L})}\right)n^{3/2}\left(L\tilde{\gamma}_{2n}^{(h)} + \tilde{\gamma}_{m2^{-L}}^{(h)}\right). \tag{4.15}$$

Like in (4.11), the constant that represents the disparity in the two precisions, $M_{l,h}$ is compared against the original matrix size m, n and the block size specifications derived from 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 bFMA's, we want to provide a more general mixed precision environment in this section. Recall that HQR, BQR, and TSQR all rely on HH transformations in one way or another, and implementations of HH transformations are expressed by (3.8). This implementation capitalizes on the rank-1 update structure of HH transformations where the predominant share of FLOPs is spent on an inner product, and computing the HH vector and constant also rely heavily on inner products. Therefore, nearly all of the computational tasks for algs. 3, 5 and 6 are attributed to the inner product, which is important in other linear algebra tools such as projections, matrix-vector, and matrix-matrix multiply. Consequently, we return to the mixed precision setting described in Assumption 2.3, where every inner product is cast down to the lower precision as shown in (2.10). We denote HQR, BQR, and TSQR computed with the mixed precision setting of Assumption 2.3 with `mpHQR2`, `mpBQR2`, and `mpTSQR2`, where the `2` represents the mixed precision procedure computed at a level-2 BLAS operation.

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

$$\mathrm{fl}(\sigma) = \mathrm{fl}(-\mathrm{sign}(\mathbf{x}[1])\|\mathbf{x}\|_2) = \sigma + \Delta\sigma, \;\; |\Delta\sigma| \leq \left(\gamma_2^{(\mathrm{l})} + \gamma_{\mathrm{m}}^{(\mathrm{h})} + \gamma_2^{(\mathrm{l})}\gamma_{\mathrm{m}}^{(\mathrm{h})}\right)|\sigma|, \tag{4.16}$$

$$\mathrm{fl}(\mathbf{v}'[1]) = \mathbf{v}'[1] + \Delta\mathbf{v}'[1] = (1+\delta^{(l)})(\mathbf{x}[1] - \sigma - \Delta\sigma), \;\; |\Delta\mathbf{v}'[1]| \leq (\gamma_3^{(l)} + \tilde{\gamma}_m^{(h)})|\mathbf{v}'[1]| \tag{4.17}$$

$$\mathrm{fl}(\beta) = \beta + \Delta\beta = (1+\delta^{(l)})\left(-\mathbf{v}'[1]/\hat{\sigma}\right), \;\; |\Delta\beta| \leq (\gamma_8^{(l)} + \tilde{\gamma}_m^{(h)})|\beta|, \tag{4.18}$$

$$\mathrm{fl}(\mathbf{v}[j]) = \mathbf{v}[j] + \Delta\mathbf{v}[j] \text{ where } |\Delta\mathbf{v}[j]| \leq \begin{cases} 0, & j = 1 \\ (\gamma_7^{(l)} + \tilde{\gamma}_m^{(h)})|\mathbf{v}_j|, & j = 2 : m-i+1. \end{cases} \tag{4.19}$$

These bounds on $\Delta\sigma$, $\Delta\mathbf{v}'[1]$, $\Delta\beta$, and $\Delta\mathbf{v}[j]$ are computed by using the rules from Lemma 2.4 on the analysis shown in subsection 3.1. Using these, we can formulate the mixed precision version of (3.9) where $\hat{\mathbf{y}} = \mathrm{fl}(\mathbf{P}_{\mathbf{v}}\mathbf{x}) \in \mathbb{R}^m$ is implemented via (3.8). Note that the inner product $\hat{\mathbf{v}}^\top\mathbf{x}$ is computed with the mixed precision inner product scheme outlined in Assumption 2.3, and all other operations are done in the lower precision. Then, the transformed vector is bounded by

$$\hat{\mathbf{y}} = \mathbf{y} + \Delta\mathbf{y}, \;\; \|\Delta\mathbf{y}\|_2 \leq (\gamma_{25}^{(l)} + \tilde{\gamma}_m^{(h)})\|\mathbf{y}\|_2. \tag{4.20}$$

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

$$\hat{\mathbf{y}} = (\mathbf{P}_{\mathbf{v}} + \Delta\mathbf{P}_{\mathbf{v}})\mathbf{x}, \ \ \|\Delta\mathbf{P}_{\mathbf{v}}\|_F \leq (\gamma_{25}^{(l)} + \tilde{\gamma}_m^{(h)}). \tag{4.21}$$

Now, we form the error bounds for applying n HH transformations to $\mathbf{x}$ using Lemma 3.2,

$$\hat{\mathbf{z}} = \mathrm{fl}(\mathbf{P}_1 \cdots \mathbf{P}_n\mathbf{x}) = \mathbf{Q}(\mathbf{x} + \Delta\mathbf{x}) = (\mathbf{Q} + \Delta\mathbf{Q})\mathbf{x}, \tag{4.22}$$

$$\|\Delta\mathbf{y}\|_2 \leq (\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)})\|\mathbf{x}\|_2, \ \ \|\Delta\mathbf{Q}\|_F \leq (\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)}). \tag{4.23}$$

Note that we use the $\tilde{\gamma}^{(l)}$ notation, where the small integer c is now required to be $\mathcal{O}(25)$. 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 $\hat{\mathbf{Q}}_{mpHQR2} \in \mathbb{R}^{m\times n}$ and $\hat{\mathbf{R}} \in \mathbb{R}^{n\times n}_{mpHQR2}$ 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,*

$$\|\Delta\mathbf{R}_{mpHQR2}[:,j]\|_2 \leq (\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)})\|\mathbf{A}[:,j]\|_2, \ \ \|\Delta\mathbf{R}_{mpHQR2}\|_F \leq (\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)})\|\mathbf{A}\|_F$$
$$\|\Delta\mathbf{Q}[:,j]_{mpHQR2}\|_2 \leq (\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)}), \ \ \|\Delta\mathbf{Q}_{mpHQR2}\|_F \leq n^{1/2}(\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_m^{(h)}).$$

Unsurprisingly, the inner product mixed precision setting yields higher error bounds as it uses more low precision arithmetic than the settings described in subsection 4.1. In the next sections we analyze using `mpHQR2` instead of `HQR` within algs. 5 and 6.

4.2.2. Round down at inner product: BQR. Now, we analyze alg. 5 with the mixed precision inner product scheme of Assumption 2.3. At the k^{th} block, we first apply the mixed precision HQR summarized in Theorem 4.1. Next, we construct the WY representation, where we can now use (4.20) and (4.21) and Lemma 3.2 to form

$$\|\hat{\mathbf{X}}_k^{(l)} - \mathbf{X}_k\|_F = \|(\hat{\mathbf{P}}_k^{(1)} \cdots \hat{\mathbf{P}}_k^{(r)}) - (\mathbf{P}_k^{(1)} \cdots \mathbf{P}_k^{(r)}))\|_F \leq \tilde{\gamma}_r^{(l)} + r\tilde{\gamma}_m^{(h)}. \tag{4.24}$$

Then, the j^{th} column of the $\mathbf{Q}$ factor resulting from this mixed precision variant of BQR incurs rounding errors bounded by

$$\|\hat{\mathbf{Q}}_{mpBQR2}[:,j]\|_F = \|\hat{\mathbf{X}}_1 \cdots \hat{\mathbf{X}}_N\hat{\mathbf{e}}_j\|_2 \leq N\tilde{\gamma}_r^{(l)} + n\tilde{\gamma}_m^{(h)}, \tag{4.25}$$

and the matrix norm error bound is,

$$\|\hat{\mathbf{Q}}_{mpBQR2}\|_F \leq n^{1/2}N\tilde{\gamma}_r^{(l)} + n^{3/2}\tilde{\gamma}_m^{(h)} \approx \left(1 + \frac{M_{l,h}}{m}\right) n^{3/2}\tilde{\gamma}_m^{(h)}. \tag{4.26}$$

Recall that the block mixed precision variant of section 4.1.1 yielded a multiplicative factor of $(1+\frac{M_{l,h}}{rm})$ (see (4.11)). This implies that the mixed precision inner product introduces low precision error $r\times$ larger than the low precision errors incurred from casting down at the block level. However, if m is sufficiently larger than $M_{l,h}$, the mixed precision inner product can still had non-leading order error terms to the worst-case scenario.

4.2.3. Round down at inner product: TSQR.

Finally, we consider using the mixed precision inner product of Assumption 2.3 in alg. 6. This corresponds to replacing every instance of $n\tilde{\gamma}_{m'}$ for $m' \in \{2n, m2^{-L}\}$ in Theorem 3.7 with $\tilde{\gamma}_n^{(l)} + n\tilde{\gamma}_{m'}^{(h)}$. We first consider the norm errors for the j^{th} column of the $\mathbf{Q}$ factor computed by this mixed precision variant of alg. 6,

$$\|\hat{\mathbf{Q}}_{mpTSQR2}[:,j] - \mathbf{Q}[:,j]\|_2 \leq (L+1)\tilde{\gamma}_n^{(l)} + n(\tilde{\gamma}_{m2^{-L}}^{(h)} + L\tilde{\gamma}_{2n}^{(h)}). \tag{4.27}$$

Then, the matrix norm error bound is

$$\|\hat{\mathbf{Q}}_{mpTSQR2} - \mathbf{Q}\|_F \leq n^{1/2}(L+1)\tilde{\gamma}_n^{(l)} + n^{3/2}(\tilde{\gamma}_{m2^{-L}}^{(h)} + L\tilde{\gamma}_{2n}^{(h)}) \tag{4.28}$$

$$\approx \left(1 + \frac{M_{l,h}L}{m2^{-L} + 2Ln}\right) n^{3/2}(\tilde{\gamma}_{m2^{-L}}^{(h)} + L\tilde{\gamma}_{2n}^{(h)}), \tag{4.29}$$

and contributes larger low precision rounding errors than in (4.15).

5. Numerical Experiments.

We conducted several numerical experiments to confirm the validity of the error bounds formed in section 4. First, we tested algs. 3 and 5 to 7, `mpHQR2`, `mpBQR2`, and `mpTSQR2` for varying matrix sizes. Then, we tested varying block sizes in alg. 7 for a fixed matrix size, and lastly compared the performance of `mpHQR2` and `mpTSQR2`.

5.1. Varying matrix sizes.

5.2. Varying block sizes in `mpBQR3`.

5.3. Varying condition numbers and comparing `mpHQR2` against `mpTSQR2`.

5.4. Numerical Experiment.

In Section **??**, we theorized that conditions exist where TSQR could outperform HQR and that these conditions were hard to identify in mixed-precision settings. An empirical comparison of these two QR factorization algorithms in double precision can be found in [20], where they conclude that deeper TSQR tends to produce more accurate QR factorizations than HQR. However, using TSQR with deep levels (large L) can actually start to perform worse than TSQR with shallower levels (smaller L), since deeper levels require more FLOPs. We instead focused on comparing HQR and TSQR performances in a mixed-precision setting. Our numerical simulations show that TSQR can still outperform HQR in low, mixed-precision settings in practice even though the theoretical bounds do not guarantee stability. Our empirical results do not behave as the theoretical bounds suggest, and even show opposite trends at times. This discrepancy highlights the shortcomings of deterministic error bounds that are too pessimistic.

We used Julia v1.0.4 for all of the numerical simulations. This programming language allows half precision storage as well as `castup` and `castdown` operations to and from single and double precisions, but has no half precision arithmetic. Therefore, we relied on using Algorithm 1 for $f \in \text{OP} \cup \{$`dot_product`$\}$ to simulate half and mixed-precision arithmetic operations. For HQR, we created a mixed-precision version of the LAPACK routine xGEQRF, where the dot product subroutine was approximated by $\text{fl}(\mathbf{x}_{\text{half}}^\top \mathbf{y}_{\text{half}})$ with `simHalf`(`dot_product` , $\mathbf{x}_{\text{half}}, \mathbf{y}_{\text{half}}$) to simulate the mixed-precision setting described in Assumption 2.3 with $u_p = 0$ (which implies $z = 1$), and we used Algorithm 1 on all other basic operations in OP to simulate half/storage precision arithmetic. This HQR was then used as a subroutine of TSQR as well. There are cases where the rounding will differ between the mixed-precision setting and the way we mimic it, i.e., basic operations that are meant to be in half/storage precision arithmetic, but are instead casted up to single and back down, as the tiebreaker within correct rounding may lead to different results than true half/storage precision arithmetic. All in all, our experiments nearly replicated the mixed-precision setting we assumed for the error analysis in Sections **??** and 3.3.3.

Following example from [20], we used m-by-n random matrices, $\mathbf{A}_\alpha$, constructed via

$$\mathbf{A}_\alpha = \mathbf{Q}'(\alpha\mathbf{E}+\mathbf{I})/\|\mathbf{Q}'(\alpha\mathbf{E}+\mathbf{I})\|_F, \tag{5.1}$$

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

We generated random matrices of size 4000-by-100 using Equation 5.1 and computed their HQR and TSQR for $L=1,\cdots,6$ in a mixed-precision setting that simulates Assumption 2.3 with $z=1$. The relative backward error, $\|\hat{\mathbf{Q}}\hat{\mathbf{R}}-\mathbf{A}\|_F/\|\mathbf{A}\|_F$, was computed by casting up $\hat{\mathbf{Q}}$, $\hat{\mathbf{R}}$, and $\mathbf{A}$ to double precision to compute the Frobenius norms. Note that the mixed-precision HQR error bounds $n\tilde{\gamma}_w^{(6d+6z+13)}$ and $n^{3/2}\tilde{\gamma}_w^{(6d+6z+13)}$ for $m=4000$ and $n=100$ are `0.936` and `9.364` respectively, and the mixed-precision TSQR bounds for $L=1,\cdots,5$ are even larger, which indicates that our error bounds do not guarantee stability.

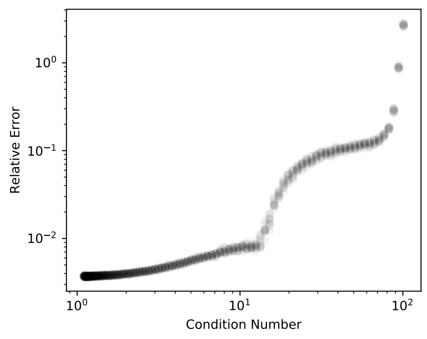

FIG. 1. *HQR errors for matrices with varying condition numbers.*

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

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.

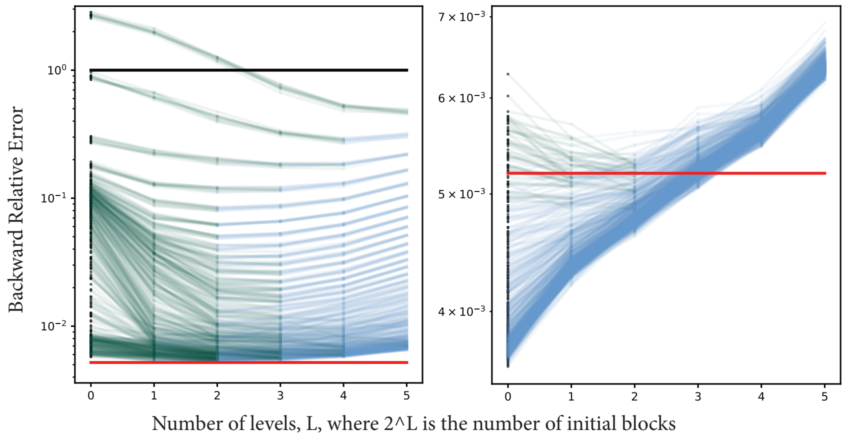

FIG. 2. *Left plot shows the relative error of QR factorization for matrices with condition numbers ranging from 5.3 to 101, and the right plot shows the errors for matrices with condition numbers ranging from 1.1 to 5.3.*

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. Abdelfattah, S. Tomov, and J. Dongarra, *Fast batched matrix multiplication for small sizes using half-precision arithmetic on GPUs*, in 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2019, pp. 111–122, https://doi.org/10.1109/IPDPS.2019.00022.

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

[3] J. Appleyard and S. Yokim, *Programming Tensor Cores in CUDA 9*, 2017, https://devblogs.nvidia.com/programming-tensor-cores-cuda-9/ (accessed 2018-07-30).

[4] 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.

[5] 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.

[6] P. Blanchard, N. J. Higham, F. Lopez, T. Mary, and S. Pranesh, *Mixed Precision Block Fused Multiply-Add : Error Analysis and Application to GPU Tensor Cores*, (2019).

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

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

[9] 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.

[10] 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.

[11] G. H. Golub and C. F. Van Loan, *Matrix computations*, JHU press, 4 ed., 2013.

[12] A. Haidar, A. Abdelfattah, M. Zounon, P. Wu, S. Pranesh, S. Tomov, and J. Dongarra, *The Design of Fast and Energy-Efficient Linear Solvers: On the Potential of Half-Precision Arithmetic and Iterative Refinement Techniques*, June 2018, pp. 586–600, https://doi.org/10.1007/978-3-319-93698-7_45.

[13] A. Haidar, S. Tomov, J. Dongarra, and N. J. Higham, *Harnessing GPU tensor cores for fast fp16 arithmetic to speed up mixed-precision iterative refinement solvers*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC '18, Piscataway, NJ, USA, 2018, IEEE Press, pp. 47:1–47:11, https://doi.org/10.1109/SC.2018.00050, https://doi.org/10.1109/SC.2018.00050.

[14] N. J. Higham, *Accuracy and Stability of Numerical Methods*, 2002, https://doi.org/10.2307/2669725.

[15] N. J. Higham and T. Mary, *A New Approach to Probabilistic Rounding Error Analysis*, SIAM Journal on Scientific Computing, 41 (2019), pp. A2815–A2835, https://doi.org/10.1137/18M1226312, https://epubs.siam.org/doi/10.1137/18M1226312.

[16] N. J. Higham and S. Pranesh, *Simulating Low Precision Floating-Point Arithmetic*, SIAM Journal on Scientific Computing, 41 (2019), pp. C585–C602, https://doi.org/10.1137/19M1251308, https://epubs.siam.org/doi/10.1137/19M1251308.

[17] A. S. Householder, *Unitary triangularization of a nonsymmetric matrix*, Journal of the ACM (JACM), 5 (1958), pp. 339–342.

[18] I. C. F. Ipsen and H. Zhou, *Probabilistic Error Analysis for Inner Products*, (2019), http://arxiv.org/abs/1906.10465, https://arxiv.org/abs/1906.10465.

[19] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, *Mixed precision training*, in International Conference on Learning Representations, 2018, https://openreview.net/forum?id=r1gs9JgRZ.

[20] D. Mori, Y. Yamamoto, and S. L. Zhang, *Backward error analysis of the AllReduce algorithm for householder QR decomposition*, Japan Journal of Industrial and Applied Mathematics, 29 (2012), pp. 111–130, https://doi.org/10.1007/s13160-011-0053-x.

[21] R. Schreiber and C. Van Loan, *A Storage-Efficient WY Representation for Products of Householder Transformations*, SIAM Journal on Scientific and Statistical Computing, 10 (1989), pp. 53–57, https://doi.org/10.1137/0910005.

[22] G. Tagliavini, S. Mach, D. Rossi, A. Marongiu, and L. Benin, *A transprecision floating-point platform for ultra-low power computing*, in 2018 Design, Automation Test in Europe Conference Exhibition (DATE), March 2018, pp. 1051–1056, https://doi.org/10.23919/DATE.2018.8342167.