MIXED-PRECISION ANALYSIS OF 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 [10] and demonstrate its practicality by applying the analysis to various Householder QR Algorithms. In addition, we present successful results from using mixed precision QR factorization for some small-scale benchmark problems in graph clustering.

1. Introduction. The accuracy of a numerical algorithm depends on several factors, including numerical stability and well-conditionedness of the problem, both of which may be sensitive to rounding errors, the difference between exact and finite-precision arithmetic. Low precision floats use fewer bits than high precision floats to represent the real numbers and naturally incur larger rounding errors. Therefore, error attributed to round-off may have a larger influence over the total error when using low precision, and some standard algorithms 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 [16] or training dense neural networks [14], 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 [7]. 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 [8, 9], (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 [4] and storage [5] can easily be implemented on GPUs and are already a common practice in data science applications.

The low precision computing environments that we consider are *mixed precision* settings, which are designed to imitate those of new GPUs that employ multiple precision types for certain tasks. For example, Tesla V100's Tensor Cores perform matrix-multiply-and-accumulate of half precision input data with exact products and single precision (32 bits) summation accumulate [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 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.

this work, we develop a framework for deterministic mixed-precision rounding error analysis, and explore half-precision Householder QR factorization (HQR) algorithms for data and graph analysis applications. QR factorization is known to provide a backward stable solution to the linear least squares problem and thus, is ideal for mixed-precision. However, additional analysis is needed as the additional round-off error will effect orthogonality, and thus the accuracy of the solution. Here, we focus on analyzing specific algorithms in a specific set of types (IEEE754 half (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 [15]).

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. [6] 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

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

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

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

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

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

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

where μ is any integer in $[0, b^t - 1]$ and η is an integer in $[\eta_{\min}, \eta_{\max}]$. While base, precision, and exponent range are fixed and define a floating point number,the sign, significand, and exponent identifies a unique number within that system. Although operations we use on $\mathbb{R}$ cannot be replicated exactly due to the finite cardinality of $\mathbb{F}$, we can still approximate the accuracy of analogous floating point operations (FLOPs). We adopt the rounding error analysis tools described in [10], which allow a relatively simple framework for formulating error bounds for complex linear algebra

Symbol(s)	Definition(s)	Section(s)
$\mathbf{x}$, $\mathbf{A}$	Vector, matrix	2
$\mathbf{Q}$	Orthogonal factor of matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$: m-by-m (full) or m-by-n (thin)	2
$\mathbf{R}$	Upper triangular or trapezoidal factor of matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$: 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	Unit round-off for precision t and base b: $\frac{1}{2}b^{1-t}$	2.1
δ^q	Quantity bounded by: $\lvert\delta_q\rvert < u_q$	2.1
γ_k^q, θ_k^q	$\frac{ku^q}{1-ku^q}$, Quantity bounded by: $\lvert\theta_k^q\rvert \le \gamma_k^q$	2.1

TABLE 1
Basic definitions

operations. A short analysis of FLOPs (see Theorem 2.2 [10]) shows that the relative error is controlled by the unit round-off, $u := \frac{1}{2}b^{1-t}$.

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`

Let 'op' be any basic operation from the set $\mathrm{OP} = \{+, -, \times, \div\}$ 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. For example, consider computing $x+y+z$, where $x, y, z \in \mathbb{R}$ with a machine that can only compute one operation at a time. Then, there is a rounding error in computing $\hat{s_1} := \mathrm{fl}(x+y) = (1+\delta)(x+y)$, and another rounding error in computing $\hat{s_2} := \mathrm{fl}(\hat{s_1} + z) = (1+\tilde{\delta})(\hat{s_1} + z)$, where $|\delta|, |\tilde{\delta}| < u$. Then,

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

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

LEMMA 2.1 (Lemma 3.1 [10]). *Let $|\delta_i| < u$ and $\rho_i \in \{-1, +1\}$, for $i = 1, \cdots, k$ and $ku < 1$.*

Then,

$$\prod_{i=1}^{k}(1+\delta_i)^{\rho_i} = 1+\theta_k, \qquad \textit{where} \qquad |\theta_k| \le \frac{ku}{1-ku} =: \gamma_k. \tag{2.4}$$

We also use

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

where $c > 0$ is a small integer.

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

$$\mathrm{fl}(x+y+z) = (1+\tilde{\delta})(1+\delta)(x+y) + (1+\tilde{\delta})z = (1+\theta_2)(x+y) + (1+\theta_1)z. \tag{2.5}$$

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

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

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

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

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 floating points have smaller unit round-off values, they can tolerate more successive FLOPs than lower precision floating points before reaching $\gamma_m = 1$. Table 2 shows the maximum number of successive floating point operations that still guarantees a relative error below 100% for various floating point types. Thus, accumulated rounding errors in lower precision types

precision	$\tilde{k} = \arg\max_k(\gamma_k \le 1)$
FP16	`512`
FP32	$\approx$ `4.194e06`
FP64	$\approx$ `2.252e15`

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

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

$$|\mathbf{x}^\top \mathbf{y} - \mathrm{fl}(\mathbf{x}^\top \mathbf{y})| \leq \gamma_m |\mathbf{x}|^\top |\mathbf{y}|, \quad \mathbf{x}, \mathbf{y} \in \mathbb{R}^m \tag{2.7}$$

as shown in [10]. 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 ($\gamma_{512}^{\mathrm{fp16}} = 1$).

We present a simple numerical experiment that shows that the standard deterministic error bound is too pessimistic and cannot be practically used to approximate rounding error for half-precision arithmetic. In this experiment, we generated 2 million random half-precision vectors of length 512 from two random distributions: the standard normal distribution, $N(0, 1)$, and the uniform distribution over $(0, 1)$. Half precision arithmetic was simulated by calling alg. 1, which was proven to be a faithful simulation in [12], for every FLOP (multiplication and addition for the dot product). The relative error in this experiment is formulated as the LHS in Equation 2.7 divided by $|\mathbf{x}|^\top |\mathbf{y}|$ and all operations outside of calculating $\mathrm{fl}(\mathbf{x}^\top \mathbf{y})$ are executed by casting up to fp64 and using fp64 arithmetic. Table 3 shows some statistics from computing the relative error for simulated half precision dot products of 512-length random 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 [11, 13]) have shown that the inner product relative backward error for the conditions used for this experiment is bounded by `5.466e-2` with probability 0.99.

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

Input: $\mathbf{x}^{\mathrm{fp16}}, \mathbf{y}^{\mathrm{fp16}} \in \mathbb{F}_{\mathrm{fp16}}^m$, $f : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}^n$

Output: $\mathrm{fl}(f(\mathbf{x}^{\mathrm{fp16}}, \mathbf{y}^{\mathrm{fp16}})) \in \mathbb{F}_{\mathrm{fp16}}^n$

$\mathbf{x}^{\mathrm{fp32}}, \mathbf{y}^{\mathrm{fp32}} \leftarrow$ `castup`$([\mathbf{x}^{\mathrm{fp16}}, \mathbf{y}^{\mathrm{fp16}}])$

$\mathbf{z}^{\mathrm{fp32}} \leftarrow \mathrm{fl}(f(\mathbf{x}^{\mathrm{fp32}}, \mathbf{y}^{\mathrm{fp32}}))$

$\mathbf{z}^{\mathrm{fp16}} \leftarrow$ `castdown`$(\mathbf{z}^{\mathrm{fp32}})$

return $\mathbf{z}^{\mathrm{fp16}}$

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

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

TABLE 3

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

We start by introducing some additional rules from [10] 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. These relations aid in writing clear and simpler error analyses. Regardless of the specific details of a mixed-precision setting, a rounding error analysis for mixed-precision arithmetic must support at least two different precision types. Thus, Lemma 2.3 allows low and high precision types and is a simple modification of Lemma 2.2. The rules for θ allows us to keep track of the two precision types separately and the rules we present for γ were chosen to be useful for casting down to the lower of the two precisions, a pertinent procedure in our mixed-precision analysis in the later sections.

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 i, j, and nonnegative integer k. Arithmetic operations between θ_k's:*

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

Operations on γ's:

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

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

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

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

LEMMA 2.3. *For any nonnegative integer k and some precision q, let θ_k^q denote a quantity bounded according to $|\theta_k^q| \leq \frac{ku^q}{1-ku^q} =: \gamma_k^q$. The following relations hold for two precisions l (low) and h (high), positive integers, j_l,j_h, non-negative integers k_l, and k_h, and $c > 0$:*

$$(1+\theta_{k_l}^l)(1+\theta_{j_l}^l)(1+\theta_{k_h}^h)(1+\theta_{j_h}^h) = (1+\theta_{k_l+j_l}^l)(1+\theta_{k_h+j_h}^h), \tag{2.9}$$

$$\frac{(1+\theta_{k_l}^l)(1+\theta_{k_h}^h)}{(1+\theta_{j_l}^l)(1+\theta_{j_h}^h)} = \begin{cases} (1+\theta_{k_h+j_h}^h)(1+\theta_{k_l+j_l}^l), & j_h \leq k_h, j_l \leq k_l, \\ (1+\theta_{k_h+2j_h}^h)(1+\theta_{k_l+j_l}^l), & j_h \leq k_h, j_l > k_l, \\ (1+\theta_{k_h+j_h}^h)(1+\theta_{k_l+2j_l}^l), & j_h > k_h, j_l \leq k_l, \\ (1+\theta_{k_h+2j_h}^h)(1+\theta_{k_l+2j_l}^l), & j_h > k_h, j_l > k_l. \end{cases} \tag{2.10}$$

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

$$\gamma_{k_h}^h \gamma_{k_l}^l \leq \gamma_{k_l}^l, \quad \text{for } k_l u^l \leq \frac{1}{2} \tag{2.11}$$

$$\gamma_{k_h}^h + u^l \leq \gamma_{d+2}^l \tag{2.12}$$

$$\gamma_{k_l}^l + u^h \leq \gamma_{k_l+1}^l \tag{2.13}$$

$$\gamma_{k_l}^l + \gamma_{k_h}^h + \gamma_{k_l}^l \gamma_{k_h}^h < \gamma_{k_l+d+1}^l. \tag{2.14}$$

We use these principles to establish a mixed-precision rounding error analysis for computing the dot product, which is crucial in many linear algebra routines such as the QR factorization. Let us define an ad hoc mixed-precision setting that is similar to the TensorCore Fused Multiply-Add (FMA) block but works at the level of a dot product. While the FMA blocks are for matrix-matrix products (level-3 BLAS), we consider a FMA for vector inner products (level-2 BLAS) as defined in Assumption 2.4.

ASSUMPTION 2.4. *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.*

3. Algorithms and existing round-off error analyses.

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

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

$$\begin{aligned} \sigma &= -\operatorname{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2, \quad \mathbf{v} = \mathbf{x} - \sigma \hat{e_1}, \\ \beta &= \frac{2}{\mathbf{v}^\top \mathbf{v}} = -\frac{1}{\sigma \mathbf{v}_1}, \qquad \mathbf{P_v} = \mathbf{I}_m - \beta \mathbf{v}\mathbf{v}^\top. \end{aligned} \tag{3.1}$$

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

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 [10]. Let $\mathbf{P_i}$ represent the i^{th} Householder transformation of this algorithm.

$$\mathbf{A} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{\text{apply } \mathbf{P_1} \text{ to } \mathbf{A}} \left[\begin{array}{c|cc} \times & \times & \times \\ \hline 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{array}\right] \xrightarrow{\text{apply } \mathbf{P_2} \text{ to } \mathbf{P_1 A}}$$

$$\left[\begin{array}{cc|c} \times & \times & \times \\ 0 & \times & \times \\ \hline 0 & 0 & \times \\ 0 & 0 & \times \end{array}\right] \xrightarrow{\text{apply } \mathbf{P_3} \text{ to } \mathbf{P_2P_1A}} \left[\begin{array}{ccc} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{array}\right] = \mathbf{P_3P_2P_1A} =: \mathbf{R}$$

Then, the $\mathbf{Q}$ factor for a full QR factorization is $\mathbf{Q} := \mathbf{P_1P_2P_3}$ since $\mathbf{P_i}$'s are symmetric, and the thin factors for a general matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ are

$$\mathbf{Q}_{\text{thin}} = \mathbf{P_1} \cdots \mathbf{P_n}\mathbf{I}_{m\times n} \quad \text{and} \quad \mathbf{R}_{\text{thin}} = \mathbf{I}^{\top}_{m\times n}\mathbf{P_n} \cdots \mathbf{P_1A}. \tag{3.2}$$

Algorithm 2: β, $\mathbf{v}$, $\sigma = \texttt{hh\_vec}(\mathbf{x})$. Given a vector $\mathbf{x} \in \mathbb{R}^n$, return $\mathbf{v}$, β, σ that satisfy $(I - \beta\mathbf{v}\mathbf{v}^{\top})\mathbf{x} = \sigma\hat{e_1}$ and $\mathbf{v}_1 = 1$ (see [2, 10]).

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

$\mathbf{v} \leftarrow \texttt{copy}(\mathbf{x})$
$\sigma \leftarrow -\text{sign}(\mathbf{x}_1)\|\mathbf{x}\|_2$
$\mathbf{v}_1 \leftarrow \mathbf{x}_1 - \sigma$
$\beta \leftarrow -\frac{\mathbf{v}_1}{\sigma}$
return β, $\mathbf{v}/\mathbf{v}_1$, σ

Algorithm 3: $\mathbf{V}$, $\boldsymbol{\beta}$, $\mathbf{R} = \texttt{householder\_qr}(A)$. 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 $\mathbf{R}$. An orthogonal matrix $\mathbf{Q}$ can be generated from $\mathbf{V}$ and $\boldsymbol{\beta}$, and $\mathbf{QR} = \mathbf{A}$.

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

$\mathbf{V}, \boldsymbol{\beta} \leftarrow \mathbf{0}_{m\times n}, \mathbf{0}_m$
for $i = 1 : n$ **do**
  $\mathbf{v}, \beta, \sigma \leftarrow \text{hh\_vec}(\mathbf{A}[i : \text{end}, i])$
  $\mathbf{V}[i : \text{end}, i]$, $\boldsymbol{\beta}_i$, $\mathbf{A}[i, i] \leftarrow \mathbf{v}, \beta, \sigma$
  $\mathbf{A}[i + 1 : \text{end}, i] \leftarrow \text{zeros}(m - i)$
  $\mathbf{A}[i : \text{end}, i + 1 : \text{end}] \leftarrow \mathbf{A}[i : \text{end}, i + 1 : \text{end}] - \beta\mathbf{v}\mathbf{v}^{\top}\mathbf{A}[i : \text{end}, i + 1 : \text{end}]$
return $\mathbf{V}$, $\boldsymbol{\beta}$, $\mathbf{A}[1 : n, 1 : n]$

3.2. Block HQR with partitioned columns.

3.3. Block HQR with partitioned rows : Tall-and-Skinny QR (TSQR).

3.4. Round down at the end of the factorization.

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

3.6. Round down at inner-product level (BLAS-2).

4. Numerical Experiments.

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

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

[6] 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.

[7] M. Fagan, J. Schlachter, K. Yoshii, S. Leyffer, K. Palem, M. Snir, S. M. Wild, and C. Enz, *Overcoming the power wall by exploiting inexactness and emerging COTS architectural features: Trading precision for improving application quality*, in 2016 29th IEEE International System-on-Chip Conference (SOCC), Sep. 2016, pp. 241–246, https://doi.org/10.1109/SOCC.2016.7905477.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

[16] U. von Luxburg, *A tutorial on spectral clustering*, Statistics and Computing, 17 (2007), pp. 395–416, https://doi.org/10.1007/s11222-007-9033-z, https://doi.org/10.1007/s11222-007-9033-z.