Simulated Half-Precision Implementation of Blocked QR Factorization and Graph Clustering Applications

Lucia Minah Yang
NSF-MSGI Final Report
Hosting Site: Lawrence Livermore National Laboratory
Mentor: Geoffrey Sanders

Summer 2018

Abstract

We explored half-precision implementation of blocked QR algorithms with the following motivations:

1. New GPUs perform fast half-precision arithmetic (4 to 16 times as fast as double-precision).
2. QR factorization is a basic linear algebra tool useful for many physics and data analysis applications.
3. Communication-avoiding, parallelizable QR algorithms already exist for tall-and-skinny matrices.

While the standard QR algorithms are highly unstable in half-precision, our numerical simulations show that the Tall-and-Skinny QR (TSQR) algorithm can improve the backward error of QR factorization. When using subspace iteration for graph clustering applications, half-precision accuracy in forming the eigenspace is sufficient for clustering with high precision and recall. Note that all half-precision arithmetic were simulated with conversions to single-precision floats. Therefore, the results from this work are somewhat optimistic.

1 Internship Project

1.1 Introduction

1.1.1 QR Factorization

In linear algebra, matrix decomposition of some matrix A identifies two or more matrices whose product amounts to A. For example, the *full* version of the QR factorization involves an orthogonal $Q \in \mathbb{R}^{m\times m}$, and $R \in \mathbb{R}^{n\times n}$..

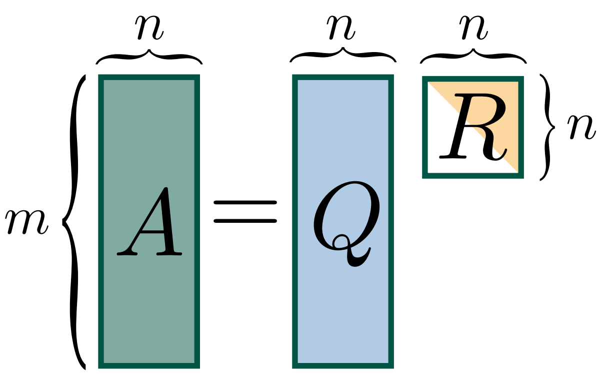

Figure 1: Tall-and-skinny A and its *thin* QR factorization.

However, we are interested in $A \in \mathbb{R}^{m\times n}$, that can be represented in a *thin* QR-form, where $Q \in \mathbb{R}^{m\times n}$ is an orthogonal matrix, and $R \in \mathbb{R}^{n\times n}$ is an upper triangular matrix such that $A = QR$. In many applications, computing the *thin* decomposition requires less computation and is sufficient in performance.

Orthogonal matrices have the property that their columns are an orthonormal set of vectors, and that their transposes are exactly their inverses ($Q^\top Q = I$). In addition, triangular matrices are easier to solve. These special properties of orthogonal Q and upper triangular R make the QR factorization a key tool in solving many types of linear algebra problems.

We will be looking at extremely overdetermined systems, where $m \gg n$. These tall-and-skinny matrices have many more rows than columns.

Finally, there are many methods of computing the QR factorization of a matrix. The three main methods are done via Householder reflections, Gram-Schmidt process, and Givens rotations. We used the Householder method, and will describe it in detail in section **??**.

1.1.2 Floating-Point Numbers

Computer representation of real numbers is expressed in a finite number of bits. Most machines use floating-point numbers, which consist of a sign bit s, k exponent bits $\{e_1, \cdots, e_k\}$, and m mantissa bits $\{b_1, \cdots, b_m\}$ in base β and minimum exponent, c_{min}. This means that ${}_i$ for $i = 1, \cdots, k$ and b_j for $j = 1, \cdots m$ are integers in the set $\{0, 1, \cdots, \beta - 1\}$. For example, a floating-point number x represented with $\{s, e_1, \cdots, e_k, b_1, \cdots, b_m\}$ has the following value:

$$x = (-1)^s \sum_{j=1}^{m} b_j \beta^{j+c}, \quad \text{where } c = e_{\text{min}} + \sum_{i=1}^{k} e_i \beta^i. \tag{1}$$

In particular, we consider the floating-point formats defined by the Institute of Electrical and Electronics (IEEE).

- half-precision (binary16)
- single-precision(binary32)
- double-precision (binary64)

The IEEE-754 standard is the most common representation for real numbers represented on computers across operating systems.

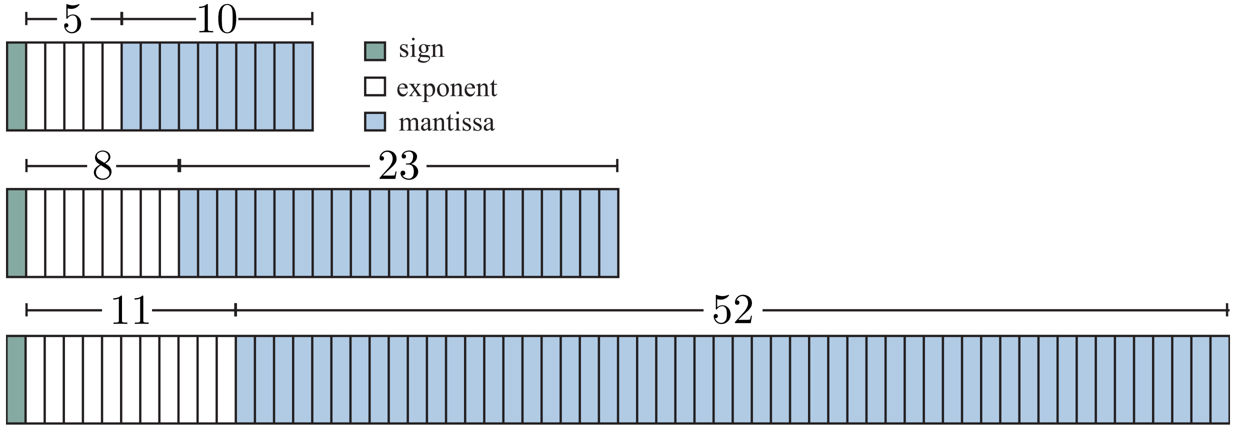

Figure 2: Layout of IEEE-754 binary 16, binary 32, and binary 64 floating numbers.

1.2 Householder Reflections and QR Factorization

We chose the Householder QR factorization method because it is numerically stable [1], and there already exist compact storage formats. Householder reflections are a special type of linear transformation that reflect about hyperplanes. A Householder transformation $P_v := I - \frac{2}{v^\top v} vv^\top$ is defined by a Householder vector v, which is orthogonal to the hyperplane. We use Householder reflections to zero-out columns of the original matrix A beneath the diagonal elements. For now, just consider zeroing out all but the first element of a vector.

Lemma 1.1. *We wish to transform $x \in \mathbb{R}^n$ to $c\hat{e_1}$, where $c \in \mathbb{R}$. The Householder vector is defined by $v := x - y$, where $y = \|x\|_2 \hat{e_1}$. The Householder reflector is*

$$P = I - \frac{2}{v^\top v} vv^\top, \tag{2}$$

and $Px = y$.

Proof. We write P and v in terms of x and y.

$$\begin{aligned} v^\top x &= x^\top x - y^\top x \\ v^\top v &= (x-y)^\top (x-y) \\ &= 2(x^\top x - x^\top y) \quad \text{(since } \|x\|_2 = \|y\|_2) \\ Px &= (I - \frac{2}{v^\top v} vv^\top)x \\ &= x - \frac{2v^\top x}{v^\top v}(x-y) \\ &= x - (x-y) = y = \|x\|_2 \hat{e_1} \end{aligned}$$

□

1.2.1 Householder QR Factorization Algorithm

Given $A \in \mathbb{R}^{m \times n}$ and Lemma 1.1, we can repeat the process of:

1. Find and store the Householder vector that would zero out the i^{th} column beneath the i^{th} element.

2. Apply the corresponding Householder transformation to the entire matrix.

3. Move to the next column.

until only an upper triangular matrix remains.

Consider the following 4-by-3 matrix example.

$$A = \left[\begin{array}{ccc} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{array}\right] \xrightarrow{P_1} \left[\begin{array}{c|cc} \times & \times & \times \\ \hline 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{array}\right] \xrightarrow{P_2} \left[\begin{array}{cc|c} \times & \times & \times \\ 0 & \times & \times \\ \hline 0 & 0 & \times \\ 0 & 0 & \times \end{array}\right] \xrightarrow{P_3} \left[\begin{array}{ccc} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{array}\right]$$

Since $R = P_3 P_2 P_1 A$, and each of the P_i's are orthogonal matrices, we result in:

$$Q = P_1 \cdots P_n, \quad \text{and} \quad R = Q^\top A. \tag{3}$$

In addition, each of the Householder transformation matrices are rank-1 updates of the identity. Therefore, when we build Q or when we update A to build R, we can use inner and outer products instead of matrix-matrix multiply. In other words, an efficient way to compute a Householder transformation defined by Householder vector v to some other matrix (or vector) B, we compute $B - \frac{2}{v^\top v} v(v^\top B)$, instead of forming the matrix P, then computing PB.

Note that Equation 2 gives a single Householder transformation matrix P for all v' in the span of v. This allows for many different ways of normalizing the Householder vectors, or with the choice of not normalizing them. For example, if we normalize Householder vectors such that the first element is 1, we can store one less number for each vector. The choice of how to normalize the Householder vectors is less important and weakly influence the overall results in single and double-precision floats. However, in half-precision where machine precision is approximately 10^{-3} and the largest number is $65,504$, a careful selection of the normalization can lead to greater stability in the Householder QR algorithm.

1.3 Tall-and-Skinny QR (TSQR) Factorization Theory and Error Bound

Of the many blocked QR factorization methods, we explored the Tall-and-Skinny QR (TSQR), otherwise known as the AllReduce algorithm [2]. A detailed description of the algorithm can be found in [2], and we present a pseudocode for the algorithm here.

Algorithm 2 will find the QR factorization of a matrix $A \in \mathbb{R}^{m \times n}$ where $m >> n$.

The inlined function qr outputs $V \in \mathbb{R}^{m\times n}$ and $R \in \mathbb{R}^{n\times n}$.The columns of V are the Householder vectors (normalized to $\sqrt{2}$) that can form the matrix $Q = H_1 \cdots H_n$. Note that $Q := Q_{\text{thin}} = Q_{\text{full}} I_{m\times n}$, and $H_i = I - v_i v_i^\top$.

Algorithm 1 is the implementation of multiplying $Q := H_1 \cdots H_n$ to another matrix or vector, when only the householder vectors to construct H_i's are given. This takes advantage of the special property of householder matrices– H_i's are rank-one updates of the identity. let $B \in \mathbb{R}^{m\times d}$. The straightforward method of computing QB costs $\mathcal{O}(m^2 d)$ where the costs of constructing Q itself is ignored. However, Algorithm 1 describes a method that is only $\mathcal{O}(mnd)$.

1.3.1 Notation

1. $I_{m\times n} := \begin{bmatrix} I_{n\times n} \\ 0_{m-n\times n} \end{bmatrix}$
2. $A[a : b, c : d]$ represents rows a to b and columns c to d of matrix A. Just the use of colon indicates all rows or all columns.
3. Define $I_{j_1,j_2} = I[:, j_1 : j_2]$. Then we can write column and row selections of a matrix as a product.
    - Column selection of a matrix A. $A[:, j_1 : j_2] = I_{j_1,j_2} A$, where $I_{j_1,j_2} = I[:, j_1 : j_2]$.
    - Row selection of a matrix A. $A[i_1 : i_2, :] = I_{i_1,i_2}^\top A$.
4. For $j = 1, \cdots \in \mathbb{N}$, define the following:
    - $\alpha(j) = \lceil \frac{j}{2} \rceil$
    - $\beta(j) = 2 + j - 2\alpha(j)$
    - or $j = 2(\alpha(j) - 1) + \beta(j)$
5. We write $Q_j^{(i)} =: \begin{bmatrix} \tilde{Q}_{j,1}^{(i)} \\ \tilde{Q}_{j,2}^{(i)} \end{bmatrix}$, where $\tilde{Q}_{j,k}^{(i)} \in \mathbb{R}^{n\times n}$ for $i = 1 : L$, and $\tilde{Q}_{j,k}^{(0)} \in \mathbb{R}^{\tilde{h}\times n}$ where $\tilde{h} \in \{h, r\}$. For more details on this part of the algorithm, look at section 1.3.3.

Algorithm 1: QB ←hh_mult(V, B): Given a set of householder vectors $\{v_i\}_{i=1}^n$ all normalized to $\sqrt{2}$, compute $H_1 \cdots H_n B$.

```
Input: $V \in \mathbb{R}^{m\times n}$ where $m >> n$. $B \in \mathbb{R}^{m\times d}$.
Output: $QB$
/* $v_i = V[i : m, i] \in \mathbb{R}^{m-(i-1)}$ and $B_i = B[i : m, i : d] \in \mathbb{R}^{(m-(i-1))\times(d-(i-1))}$.   */
for $i = 1 : n$ do
    $B_i \leftarrow B_i - v_i(v_i^\top B_i)$
return $B$
```

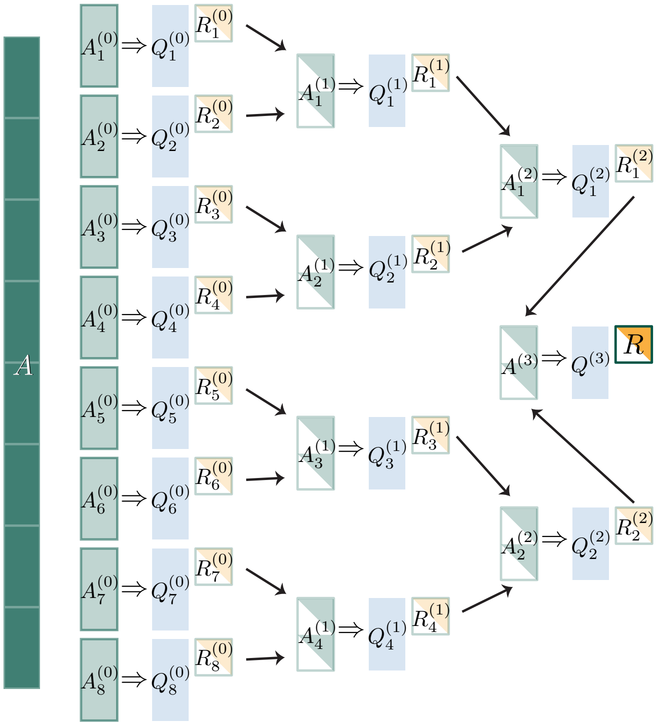
A
$A_1^{(0)} \Rightarrow Q_1^{(0)}$ $R_1^{(0)}$
$A_2^{(0)} \Rightarrow Q_2^{(0)}$ $R_2^{(0)}$
$A_3^{(0)} \Rightarrow Q_3^{(0)}$ $R_3^{(0)}$
$A_4^{(0)} \Rightarrow Q_4^{(0)}$ $R_4^{(0)}$
$A_5^{(0)} \Rightarrow Q_5^{(0)}$ $R_5^{(0)}$
$A_6^{(0)} \Rightarrow Q_6^{(0)}$ $R_6^{(0)}$
$A_7^{(0)} \Rightarrow Q_7^{(0)}$ $R_7^{(0)}$
$A_8^{(0)} \Rightarrow Q_8^{(0)}$ $R_8^{(0)}$
$A_1^{(1)} \Rightarrow Q_1^{(1)}$ $R_1^{(1)}$
$A_2^{(1)} \Rightarrow Q_2^{(1)}$ $R_2^{(1)}$
$A_3^{(1)} \Rightarrow Q_3^{(1)}$ $R_3^{(1)}$
$A_4^{(1)} \Rightarrow Q_4^{(1)}$ $R_4^{(1)}$
$A_1^{(2)} \Rightarrow Q_1^{(2)}$ $R_1^{(2)}$
$A_2^{(2)} \Rightarrow Q_2^{(2)}$ $R_2^{(2)}$
$A^{(3)} \Rightarrow Q^{(3)}$ R

Figure 3: Visualization of the TSQR factorization (AllReduce) algorithm.

1.3.2 TSQR/AllReduce Algorithm

1.3.3 Details on constructing Q

In the single-level version of this algorithm, we first bisect A into $A_1^{(0)}$ and $A_2^{(0)}$ and compute the QR factorization of each of those submatrices.

$$A = \begin{bmatrix} A_1^{(0)} \\ A_2^{(0)} \end{bmatrix} = \begin{bmatrix} Q_1^{(0)} R_1^{(0)} \\ Q_2^{(0)} R_2^{(0)} \end{bmatrix} = \begin{bmatrix} Q_1^{(0)} & 0 \\ 0 & Q_2^{(0)} \end{bmatrix} \begin{bmatrix} R_1^{(0)} \\ R_2^{(0)} \end{bmatrix} = \begin{bmatrix} Q_1^{(0)} & 0 \\ 0 & Q_2^{(0)} \end{bmatrix} Q_1^{(1)} R_1^{(1)},$$

Algorithm 2: Finds the QR factorization of a tall, skinny matrix, A.

```
Input: $A \in \mathbb{R}^{m \times n}$ where $m >> n$, $L \in \mathbb{N}$ where $2^L$ is the initial number of submatrices.
Output: $\hat{Q} \in \mathbb{R}^{m \times n}$, $\hat{R} \in \mathbb{R}^{n \times n}$ such that $\hat{Q}\hat{R} = A + \Delta A$.
$h \leftarrow \lfloor \frac{m}{2^L} \rfloor$                    // Number of rows for all but the last block.
$r \leftarrow m - (2^L - 1)h$          // Number of rows for the last block ($h \leq r < 2h$).
/* Split $A$ into $2^L$ blocks. Note that level $(i)$ has $2^{L-i}$ block        */
for $j = 1 : 2^L - 1$ do
    $A_j^{(0)} \leftarrow I_{(j-1)h, jh}^{\top} A$
$A_{2^L}^{(0)} \leftarrow I_{(2^L - 1)h, m}^{\top} A$                  // Last block may have more rows.
/* Store Householder vectors as columns of matrix $V_j^{(i)}$, and set up $A$
   for the next level.                                                          */
for $i = 0 : L - 1$ do
    for $j = 1 : 2^{L-i}$ do
        $V_{2j-1}^{(i)}, R_{2j-1}^{(i)} \leftarrow \mathrm{qr}(A_{2j-1}^{(i)})$
        $V_{2j}^{(i)}, R_{2j}^{(i)} \leftarrow \mathrm{qr}(A_{2j}^{(i)})$  // $V_j^{(i)} \in \mathbb{R}^{2n \times n}$ for $i > 0$, and $R_j^{(i)} \in \mathbb{R}^{n \times n}$ always.
        $A_j^{(i+1)} \leftarrow \begin{bmatrix} R_{2j-1}^{(i)} \\ R_{2j}^{(i)} \end{bmatrix}$
/* At the bottom-most level, get the $R$ factor.                               */
$V_1^{(L)}, R \leftarrow \mathrm{qr}(A_1^{(L)})$
$Q_1^{(L)} \leftarrow \mathrm{hh\_mult}(V_1^{(L)}, I_{2n \times n})$
/* Combine $Q$ factors from bottom-up-- look at Notation (4).                  */
for $i = L - 1 : -1 : 1$ do
    for $j = 1 : 2^{L-i}$ do
        $Q_j^{(i)} \leftarrow \mathrm{hh\_mult}\left( V_j^{(i)}, \begin{bmatrix} \tilde{Q}_{\alpha(j),\beta(j)}^{(i+1)} \\ O_{n,n} \end{bmatrix} \right)$
/* At the top-most level, construct the $Q$ factor.                             */
$Q \leftarrow []$;
for $j = 1 : 2^L$ do
    $Q \leftarrow \begin{bmatrix} Q \\ \mathrm{hh\_mult}\left( V_j^{(0)}, \begin{bmatrix} \tilde{Q}_{\alpha(j),\beta(j)}^{(1)} \\ O_{\tilde{h},n} \end{bmatrix} \right) \end{bmatrix}$
return $Q$, $R$
```

$$\text{where } Q_1^{(1)} R_1^{(1)} = \begin{bmatrix} R_1^{(0)} \\ R_2^{(0)} \end{bmatrix}$$

Whereas $R_1^{(1)}$ is the final R factor of the QR factorization of the original matrix, A, we still need to construct Q. Bisecting $Q_1^{(1)}$ into two submatrices allows us to write and compute the product more compactly.

$$Q_1^{(1)} := \begin{bmatrix} \tilde{Q}_{1,1}^{(1)} \\ \tilde{Q}_{1,2}^{(1)} \end{bmatrix} \Rightarrow \begin{bmatrix} Q_1^{(0)} & 0 \\ 0 & Q_2^{(0)} \end{bmatrix} Q_1^{(1)} = \begin{bmatrix} Q_1^{(0)} \tilde{Q}_{1,1}^{(1)} \\ Q_2^{(0)} \tilde{Q}_{1,2}^{(1)} \end{bmatrix}$$

1.3.4 Variants of TSQR

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

1.3.5 Error Bounds

The error bound for the traditional (unblocked) Householder QR algorithm is given in Theorem 18.4 of [1]. We present a part of the theorem here:

Theorem 1.2. *Let $\hat{R} \in \mathbb{R}^{m \times n}$ be the computed upper trapezoidal QR factorization of $A \in \mathbb{R}^{m \times n} (m \geq n)$ obtained via the Householder QR algorithm. Then there exists an orthogonal $Q \in \mathbb{R}^{m \times m}$ such that*

$$A + \Delta A = Q\hat{R}, \tag{4}$$

where $\|\Delta A\|_F \leq n\gamma_{cm}\|A\|_F$, and $\gamma_{cm} = \frac{cmu}{1-cmu}$ for some small integer constant c and the unit round-off, u.

This is an upper bound to the backward relative error, $n\gamma_{cm}$, and as we use lower-precision floats u increases and the bound grows larger than 1. Let's consider half-precision floats. The unit round of is: $u_{\text{binary16}} \approx 0.0009$. If we set $c = 1$ (the smallest we are allowed to do so), $\gamma_{cm} = 1$, when $m \approx 555$. Since the error bound is actually $n\gamma_{cm}$, the bound serves no purpose for matrices with more than a few hundred columns or rows.

An error bound for the parallel TSQR algorithm exists [2], and it is derived from the analyses done for Theorem 1.2 in [1]. The calculation for the bound is given in great detail in [2], and we present the punchline theorem here.

Theorem 1.3. *Let $\hat{R} \in \mathbb{R}^{m\times n}$ be the computed upper trapezoidal QR factorization of $A \in \mathbb{R}^{m\times n}(m \geq n)$ obtained via the TSQR (AllReduce) algorithm defined in Algorithm 2. Then there exists an orthogonal matrix $Q \in \mathbb{R}^{m\times n}$ such that*

$$A + \Delta A = Q\hat{R}, \quad \textit{where}$$

$$\|\Delta A\|_F \leq \left[n\gamma_{c\cdot\frac{m}{2^L}} + \left(1 + n\gamma_{c\cdot\frac{m}{2^L}}\right)\left\{(1 + n\gamma_{c\cdot 2n})^L - 1\right\}\right]\|A\|_F, \tag{5}$$

and 2^L is the number of initial blocks.

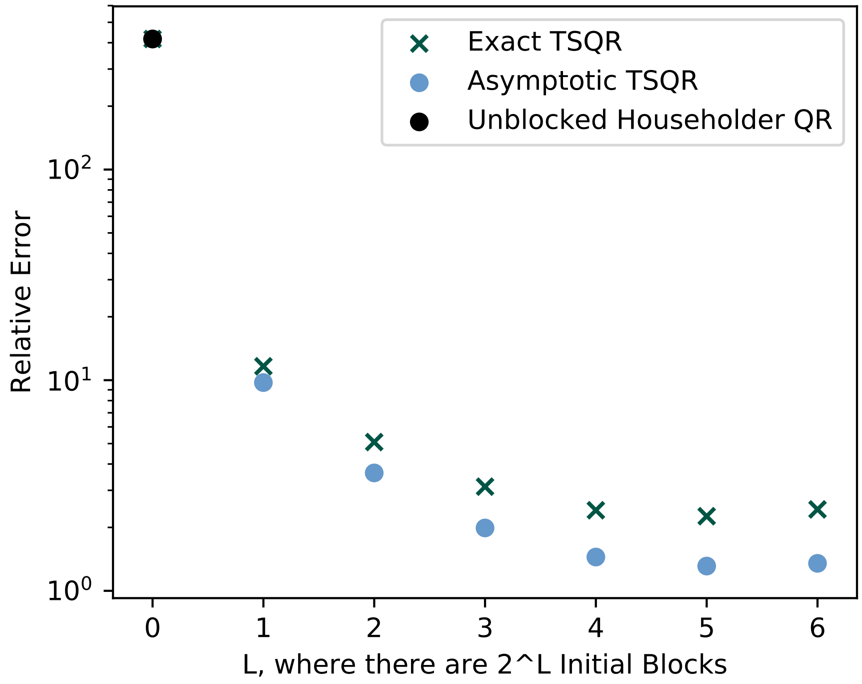

Figure 4: Error Bounds for QR Factorization Algorithms done on $A \in \mathbb{R}^{1000\times 10}$ and in half-precision.

If we can further assume that $n\gamma_{c\frac{m}{2^L}} \ll 1$ and $n\gamma_{c\cdot 2n}$, then the leading terms of the coefficient of $\|A\|_F$ in Equation 5 are $n\gamma_{c\cdot\frac{m}{2^L}} + Ln\gamma_{c\cdot 2n}$. Note that if we set $L = 0$, then we recover the bound for the unblocked Householder QR algorithm. Figure 4 shows that the TSQR error bound is tighter than that of the traditional error bounds. In addition, we can see that there is a limit to how big L should be before the error bound begins to increase again. Finally, as L increases, our approximations to the coefficient in Equation 5 becomes too optimistic as an upper bound.

Note that Figure 4 is shown for a rather small matrix with 1000 rows and 10 columns. Even though the TSQR error bounds are tighter than the error bound given by the traditional Householder QR algorithm, *large* vector and matrix operations are inherently unstable in half-precision. The next section discusses numerical simulations and results done in lower precisions.

1.4 TSQR Numerical Simulations and Results

We used Julia v0.6.4 for all of the numerical simulations. Instead of relying on the QR factorization algorithms that Julia calls from LAPACK routines, we wrote the traditional Householder QR algorithm exactly as is described in the error analysis of Theorem 1.2 in [1]. The main difference between that algorithm and the ones written for LAPACK is that whereas we normalized Householder vectors v's such that $v^\top v = 2$, those routines normalize the vectors such that their first element is 1. Unlike the numerical results shown in [2] where it is implied that the relative error tends to decrease as we increase the number of initial blocks, we found that is not necessarily the case.

There were many differences between their simulations and ours since they only experimented in double-precision, whereas we focused on experimenting in lower precisions.

For the matrix size $\mathbb{R}^{6400\times 100}$, we found that increasing the number of initial blocks in TSQR more often than not increased the backward relative error for double- and single-precisions. We did not perform enough of these simulations to get a large enough sample size and a solid conclusion, but rather moved on to doing experiments in half-precision.

Although Julia v0.6.4 allows for storage of half-precision floats, all operations done on them were done by converting to single-precision, performing single-precision arithmetic, then casting back down to half-precision. Therefore, many of our results may be more optimistic than if true half-precision arithmetic were done.

Furthermore, we attempted TSQR on matrices where their elements were picked from some random distribution. We found that TSQR performance had some correlations to the type of distributions the matrices were pulled from. We decided to focus on the relationship between the condition number of the matrix and performance of TSQR on it.

1.4.1 Constructing Test Matrices

Following example from [2], we build test matrices A_α such that α determines the condition number.

$$A_\alpha = \frac{Q'(\alpha E + I)}{\|Q'(\alpha E + I)\|_F} \tag{6}$$

Here, $Q' \in \mathbb{R}^{m\times n}$ is an orthogonal matrix generated by taking a QR factorization of a random matrix with same dimensions. The matrix $E \in \mathbb{R}^{n\times n}$ is a matrix whose entries are all 1, and α is some constant.

Lemma 1.4. *The eigenvalues of $\alpha E + I \in \mathbb{R}^{n\times n}$ are 1 with multiplicity 1, and $\alpha n + 1$ with multiplicity $(n-1)$.*

Proof. E is a singular matrix, and for any $x \in \mathbb{R}^n$, $Ex = \sum_{i=1}^n x_i$. Let $W_n = \{x \in \mathbb{R}^\ltimes : \sum_{i=1}^n x_i\}$. Without loss of generality, $x_1 = -\sum_{i=2}^n x_i$, and $W_n = \text{span}\{-\hat{e}_1 + \hat{e}_i\}_{i=2}^n$. Therefore, $\dim(W_n) = n-1$, and its basis vectors $\{-\hat{e}_1 + \hat{e}_i\}_{i=2}^n$ are the $n-1$ eigenvectors that correspond to the eigenvalue, $\lambda = 0$. Let $\lambda = n$, and $\vec{x} = \sum_{i=1}^n \hat{e}_i$. Then, $((E - \lambda I)\vec{x})_i = (1-n) + \sum_{j\neq i} 1 = 0$.

$$(\alpha E + I - \lambda I)\vec{x} = \vec{0}$$
$$(\alpha E - (\lambda - 1)I)\vec{x} = \vec{0}$$

αE has two eigenvalues, 0 and αn. Then, $\alpha E + I$ has eigenvalues 1 and $\alpha n + 1$. □

Finally, $\|Q'(\alpha E + I)\|_2 \leq \|Q\|_2 \|(\alpha E + I)\|_2 \leq \|(\alpha E + I)\|_2 = \alpha n + 1$, since Q' is an orthogonal matrix.

We worked with matrices in $\mathbb{R}^{4000\times 100}$ whose condition numbers range from 1.1 to 101. We constructed these matrices by setting $\log_{10}\alpha$ to be spaced equally from 0.001 to 1, and by picking Q from the QR factorization of matrices whose entries were picked from the uniform distribution in the range $[0, 1]$. We conducted 10 trials for each α value.

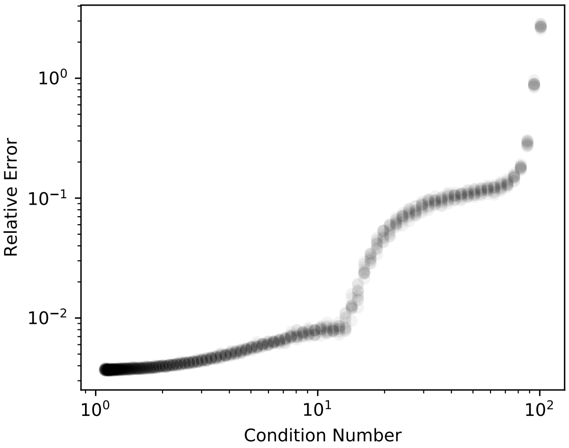

Figure 5: Backward relative error for matrices with condition numbers ranging from 1.1 to 101.

1.4.2 Results

First, we computed the relative backward error, $\frac{\|QR-A\|_F}{\|A\|_F}$, of the traditional Householder QR algorithm. Figure 5 shows positive correlation between the condition number of the matrix and the relative error of the QR factorization.

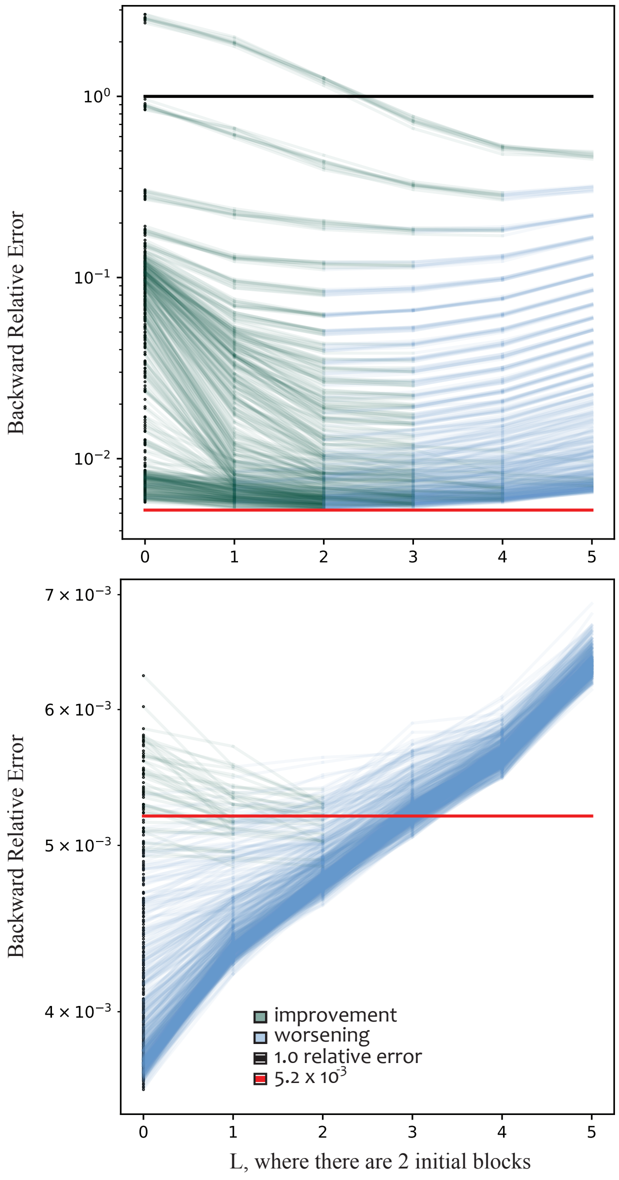

Figure 6: Top plot shows the relative error of QR factorization for matrices with condition numbers ranging from 5.3 to 101, and the bottom plot shows the errors for matrices with condition numbers ranging from 1.1 to 5.3.

Since all of the matrices were in $\mathbb{R}^{4000\times 100}$, 2^5 was the largest number of initial blocks we could start with. For each matrix, we performed TSQR with 2, 4, 8, 16, and 32 initial blocks, and computed the relative error.

Figure 6 shows the relative errors for all of these matrices. The red line on both plots marks 5.2×10^{-3}. Each line shows the relative error for TSQR with varying number of initial blocks. Green indicates reduction in error, and blue indicates increase in error. There appears to be two main trends for this experiment. Level 0 is the error for using the traditional Householder QR algorithm. When the error is low enough for the unblocked QR factorization, TSQR performs worse for these matrices. Recall that machine precision for half-precision is about 10^{-3}. This shows that the traditional QR factorization had been very good to begin with. Finally, even when TSQR is *successful* initially, we can see that too many initial blocks can become a problem as well.

Overall, this figure shows a variety of results that encourage further exploration. We have shown that TSQR can improve on certain matrices where the unblocked Householder QR algorithm was highly unstable in half-precision. Identifying which matrix properties correlate to TSQR and why can help broaden the possibility of using lower precision arithmetic for QR factorizations.

Possible future works include using true half-precision arithmetic for these numerical simulations, and tightening the error bound of the TSQR algorithm. In addition, we can seek to apply TSQR on single-precision and see how they perform with much larger matrices.

1.5 Graph Clustering Applications

We explored the efficacy of half-precision QR factorization on clustering problems. We identified eigenspaces of adjacency matrices of graphs via subspace iteration, then applied spectral clustering methods.

1.5.1 Clustering Problem

Define graph, nodes, clusters, etc?

1.5.2 Subspace Iteration

We used static graphs with known truths for the Stochastic Block Partitioning Challenge. The graphs we used were undirected and unweighted– the only elements in the adjacency matrices were 0's and 1's which can easily be represented in half-, single-, and double-precision floats.

Subspace iteration is a modification of the power method, which computes an invariant subspace with dimension $p > 1$ [4]. The simplest variation of this algorithm is shown below in Algorithm 3.

Since we were given the true clusters within the data set we simply set k to be exactly the number of clusters. The matrix product AQ can be computed efficiently since A is a sparse matrix, and note that we are always computing the QR factorization of a tall-and-skinny matrix. This algorithm is an iterative method with two possible stopping criteria:

1. The maximum number of iterations to complete before exiting the loop is declared as max_iter.
2. If the eigenspace error is smaller than τ, then exit the loop.

Algorithm 3: Find orthogonal basis (given by columns of output matrix Q) of an invariant subspace of the input adjacency matrix, A.

```
Input: Adjacency matrix $A \in \{0,1\}^{m \times m}$ where $m \geq n$, and max_iter, the maximum
       number of iterations, and $\tau$ the threshold for the eigenspace error.
Output: $Q$
Initialize $Y \in \mathbb{R}^{m \times k}$, a random matrix. // $Y$ would likely be full-rank.
$Q, R \leftarrow qr(Y)$ for $i = 1, \cdots, max\_iter$ do
    $Y \leftarrow AQ$
    if $\frac{\|Y - QQ^\top Y\|_2}{\|Y\|_2} < \tau$ then
        exit loop.
    // $\|Y - QQ^\top Y\|_2$ is the eigenspace error.
    $Q, R \leftarrow qr(Y)$
return $Q$
```

In particular, we highlight the results from performing subspace iteration on a graph with 5000 nodes and 19 true clusters. We set an initial random matrix Y in double-precision, and converted it into single- and half-precisions to use as the starting matrix. Figure 7 shows the eigenspace error at each iteration for double-, single-, and half- precision Householder QR factorization. We used the traditional, unblocked QR factorization, instead of applying TSQR. The τ values were set to $5\times$ unit round-off for each of the precisions, and the solid lines are plotted to show the unit round-off values.

The half-precision implementation approached its best performance close to 10 iterations, and continued to fluctuate near there without hitting the τ value. Nonetheless, we can see that the first 9 iterations of the subspace iteration technique yielded the same eigenspace errors for all three precisions.

The same pattern continued for single- and double- precision implementations until they reached single-precision unit round-off near 10^{-7}.

The main idea is that if we do not need an extremely low eigenspace error for the purpose of spectral clustering, we can do with lower-precision QR factorizations that require less storage and faster computation time.

Due to the random element of the initial matrix Y at the beginning of subspace iteration, there is some variability to its performance in identifying an invariant subspace. In addition, we chose the number of columns of Y to be the number of true clusters, which will likely be unknown in practice.

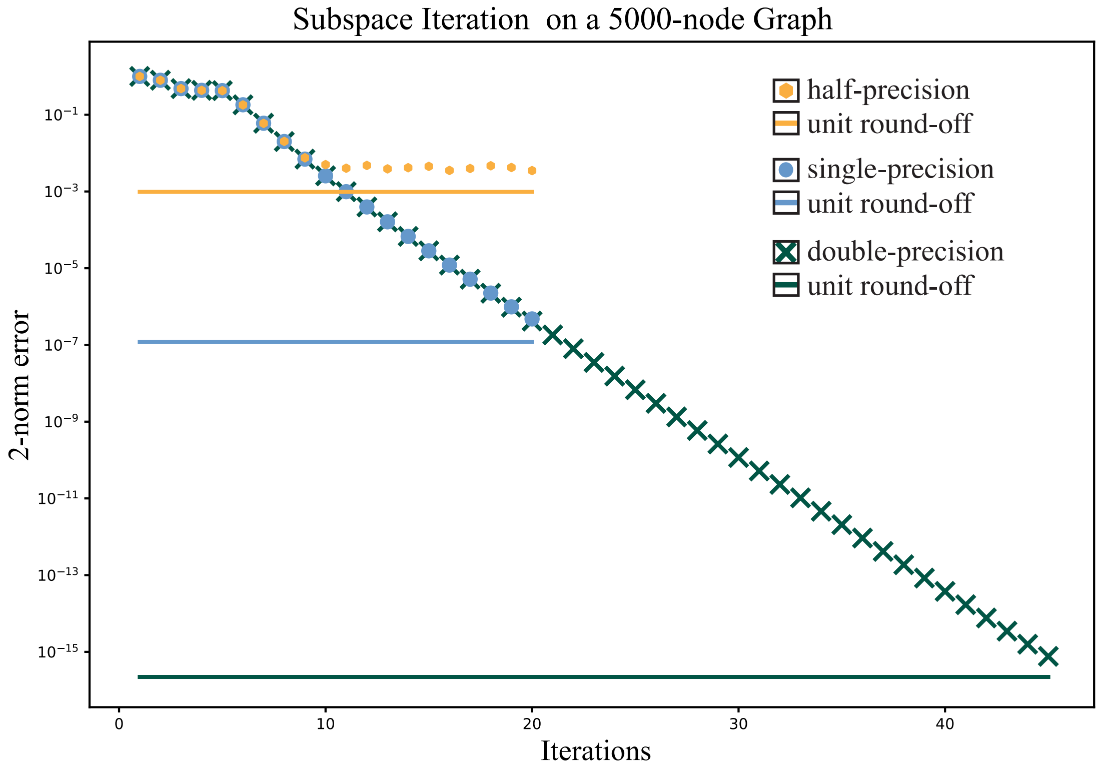

Figure 7: Eigenspace Error for subspace iteration with using double-, single-, and half- precision traiditional Householder QR factorizations.

1.5.3 Spectral Clustering

We used density-based spatial clustering of applications with nose (DBSCAN) from the Julia implementation of the SciKitLearn package. This method does not require a predetermined number of clusters as a parameter. We used precision and recall (defined below) to evaluate the quality of clustering.

Definition 1.1. Some relevant evaluation metrics of classification of clustering tasks are precision and recall. Precision is the fraction of relevant instances among the retrieved instances.

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}. \tag{7}$$

Recall is the fraction of relevant instances that have been retrieved over the total amount of relevant instances.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}. \tag{8}$$

We gathered clustering results for just one graph, although we used 10 different random matrices

	Precision	Recall
half-precision	0.9822	0.9393
single-precision	0.9817	0.9407
double-precision	0.9822	0.9405

Figure 8: Minimum precision and recall values for 10 trials of DBSCAN on graph with 5000 nodes and 19 true clusters.

as the initial Y. The variance in precision and recall values for these 10 trials in three precisions were in the range of 10^{-6}. We only show the minimum values in the table in Figure 8.

The experiment shows indiscernible difference in the DBSCAN clustering results. This suggests that a lower-precision error in the eigenspace error in subspace iteration can still lead to a sufficiently accurate clustering, and calls for further investigations into a variable precision approach to other spectral clustering methods as well.

1.6 Conclusion and Future Works

We introduced a brief summary of QR factorization [1] and commonly used floating-point numbers in section 1.1, then the Householder reflection approach of computing QR factorizations in section 1.2. A theoretical overview of the TSQR (or AllReduce [2]) algorithm, and its error bound and stability expectations are given in section 1.3, and section 1.4 describes our numerical simulations and experiments and their results. The last section, 1.5, shows simple spectral clustering applications with simulated half-precision arithmetic, and further opens the possibilities of continuing research in this area.

Our simulated half-precision arithmetic is in one way a variable precision method since all numbers are stored in half-precision but are converted into single-precision for the arithmetical operations. An exploration of true half-precision arithmetic and if it has a significant disparities with the current work would be a possible future direction. Also, a study of whether TSQR can increase stability of QR factorizations in double- and single-precision for larger matrices, a practical formulation of how to integrate half-precision QR factorizations into spectral clustering, theoretical research into stability of blocked QR algorithms with tighter error bounds in mind, and testing other applications of QR factorization in half-precision are more options.

2 Impact of Internship on My Career

My summer at Lawrence Livermore National Laboratory (LLNL) made possible by the National Science Foundation's (NSF) Mathematical Sciences Graduate Internship (MSGI) allowed me to conduct research full-time. Having spent my first two years in graduate school focused on coursework and completing preliminary exam requirements, this was a well-needed shift towards mathematical research.

The project I chose placed me in the Variational Precision Computing (VPC) project in the Center for Applied Scientific Computing (CASC) division at LLNL. The VPC project is comprised of applied mathematicians, computer scientists, and electrical engineers working together reevaluating and redesigning computational techniques at all aspects. My contribution was in the linear algebra library, whereas other areas in the project include hardware and software engineering, and more specifically data compression and new data representations. Being a part of such a large and long-term project has been a new experience for me that showed the inner-workings of a collaborative environment for technical research outside of academia.

First, I felt very welcome and comfortable with my mentor, Geoffrey Sanders, and with the rest of the math group within the VPC project. They helped me identify my goals for the summer, and offered guidance for all aspects of acclimating to working at the lab. Diversity within this group included people of color and women scientists, an uncommon feat in most STEM institutions. As a woman of color, this environment felt comfortable and I did not feel like an outsider. I was grateful to have been able to connect with women mathematicians outside of CU, and felt renewed motivations for me to contribute to the Association for Women in Mathematics (AWM) chapter at my graduate school, University of Colorado at Boulder, as well as other groups dedicated to supporting underrepresented minority groups in STEM fields. In addition, there were many lab-wide events at LLNL designed to help summer interns network with people outside of their immediate vicinity. Overall, I perceived LLNL to be an institution that cares for the personal advancement of each employee and intern, as opposed to only caring for the advancement of the institution itself. Speaking to many post-docs at the lab helped me identify what kind of resources I would desire at a workplace in a few years when I will be actively looking for jobs.

Next, I experienced how scientists across disciplines work together. Weekly and monthly project meetings encourage each member of the group to work on communicating their achievements and aspirations to others who may not be able to fully comprehend the technical details of their work, but still be able to provide a fresh view. My takeaways from this include a desire to seek collaborations outside of my department, to participate in interdisciplinary scientific organizations, and to be more open to opportunities outside of my direct path. Also, I have learned new technical skills and knowledge that has broadened my interests within applied math, and made my awareness of the field more complete. I had been trained on the more theoretical components of mathematical research, and had paid less attention to their practical computational applications. I realized that how machines store, transfer and process data are interesting mathematical questions that are relevant to my existing skills.

I am much more confident about the next few years of graduate school, which will be focused on my dissertation research. This summer experience has influenced me to think more independently about the direction of research I had been doing at school. I hope to continue collaborating on the work I contributed to this summer, and potentially return in future summers to continue research for the VPC project. This professional experience will be valuable in shaping the remainder of my graduate student years as the beginning of my career. My utmost priority will still be my dissertation research and the completion of my degree, but I am not limited to just that single project. In addition, I have widened my network of peers and collaborators.

I am very grateful for this fellowship opportunity from NSF-MSGI, and would like to thank Geoffrey Sanders and Jeffrey Hittinger for being wonderful mentors and hosts at LLNL. I hope this program continues to give students in the mathematical sciences similar opportunities as I have experienced.

References

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

[2] D. Mori, Y. Yamamoto, and S.-L. Zhang, “Backward error analysis of the allreduce algorithm for householder qr decomposition,” *Japan Journal of Industrial and Applied Mathematics*, vol. 29, pp. 111–130, Feb 2012.

[3] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, “Communication-optimal parallel and sequential qr and lu factorizations,” *SIAM Journal on Scientific Computing*, vol. 34, no. 1, pp. A206–A239, 2012.

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