縦長行列の QR 分解に対する各種アルゴリズムの比較：Oakforest-PACS 上での性能評価

深谷 猛

北海道大学 情報基盤センター

１．はじめに

Aを$m \times n$列フルランク実行列で，縦長（$m \gg n$）であるとする。本稿では，このような行列の QR 分解（Reduced QR 分解，Thin QR 分解）[1]

$$A = QR$$

の計算を考える。ただし，Qは$m \times n$列直交行列（$Q^{\mathsf{T}}Q = I_n$を満たす），Rは$n \times n$上三角行列である。縦長行列の QR 分解は，最小二乗問題の求解[2]や特異値分解の前処理[3]に加え，線形方程式や固有値問題に対する様々なアルゴリズムにおけるベクトルの直交化としての応用を持つ[4]。

行列の QR 分解を計算するためのアルゴリズムとしては，線形代数の講義で習う Gram-Schmidt の直交化のアルゴリズム[5]と，一般的な数値計算の教科書に記載されている Householder QR 分解のアルゴリズム[5]が有名である。一方，大規模分散並列環境の普及とともに，線形計算アルゴリズム中の内積計算などで必要となる集団通信（例：MPI_Allreduce）のコスト（特に，レイテンシに相当するコスト）が強スケーリング時の大きなボトルネックとなり，通信回避（Communication-Avoiding）や通信隠蔽（Communication-Hiding）の重要性が増した[6, 7]。そのような背景の下，縦長行列の QR 分解に対して，TSQR アルゴリズムが提案され[8]，その有効性が示された。また，近年，Cholesky QR 分解に基づくアルゴリズム[3]の研究も活発に行われている。

このように，現在では，縦長行列の QR 分解を計算するためのアルゴリズムとして，様々な選択肢が存在する。それぞれのアルゴリズムは，計算精度と計算時間において，異なる特徴を持っている。そこで，本稿では，縦長行列の QR 分解を計算する各種アルゴリズムを概説するとともに，Oakforest-PACS 上で性能評価を行った結果を報告する。なお，本稿で紹介する性能評価結果は，2020 年 1 月 30 日～31 日に実施した「大規模 HPC チャレンジ（Oakforest-PACS)」で得られた結果の一部である。

以下では，分散並列環境における縦長行列の QR 分解を議論するが，計算対象の行列Aは，

$$A = \begin{pmatrix} A_1 \\ \vdots \\ A_p \\ \vdots \\ A_P \end{pmatrix}$$

のように，行方向に 1 次元ブロック分割され，p番目のプロセス（プロセス数はP）がA_pを保持する状況を想定する。また，計算結果のQについても，同様の分散を想定する。一方，Rについては，少なくとも一つのプロセスがR全体を保持している（全てのプロセスが保持する必要はない）とする。なお，アルゴリズムの演算コストや通信コストの議論では，クリティカルパス上のコストをカウントするものとする。また，2 ノルムによる行列の条件数を

$$\kappa_2(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)}$$

と定義する[9]。ただし，$\sigma_{\max}(A)$と$\sigma_{\min}(A)$はそれぞれAの最大特異値と最小特異値である。Aの列ベクトルの線形従属性が強くなるほど，$\kappa_2(A)$は大きくなり，一般的に数値的な QR 分解が難しくなる（数値計算の精度が低下したり，アルゴリズムの破綻が生じたりする可能性が高まる）。なお，以下の議論や数値実験では，倍精度浮動小数点（FP64）を用いる場合を想定しており，Aが列フルランクであることを仮定しているため，$\kappa_2(A) \leq 10^{16}$程度となる。

２．様々な QR 分解アルゴリズム

様々な QR 分解アルゴリズムが存在するが，それらは，Orthogonal Triangularization 型と Triangular Orthogonalization 型に大別される[1]。Orthogonal Triangularization 型のアルゴリズムでは，

$$\hat{Q}_k \cdots \hat{Q}_2 \hat{Q}_1 A \to \begin{pmatrix} R \\ O \end{pmatrix}$$

と直交変換$\hat{Q}_1, \ldots, \hat{Q}_k \in \mathrm{R}^{m \times m}$を$A$に（左側から）作用させて，$A$を上三角行列に変形する。さらに，

$$\left(\hat{Q}_k \cdots \hat{Q}_2 \hat{Q}_1\right)^{-1} \begin{pmatrix} I_n \\ O \end{pmatrix} \to Q$$

とすることでQを得る。Orthogonal Triangularization 型アルゴリズムの代表例は，Householder 変換を用いたアルゴリズム（Householder QR）や Givens 回転を用いたアルゴリズムである。

一方，Triangular Orthogonalization 型のアルゴリズムでは，

$$A\,\hat{R}_1 \hat{R}_2 \cdots \hat{R}_k \to Q$$

と上三角行列$\hat{R}_1, \hat{R}_2, \ldots, \hat{R}_k$を$A$に（右側から）作用させて，$A$を列直交行列に変形する。さらに，

$$\left(\hat{R}_1 \hat{R}_2 \cdots \hat{R}_k\right)^{-1} \to R$$

とすることでRを得る。Triangular Orthogonalization 型アルゴリズムの代表例は，Gram-Schmidt の直交化によるアルゴリズムである。

以下では，いくつかの主要なアルゴリズムを概説する。ただし，それぞれの特徴を中心に説明するため，アルゴリズムの具体的な中身等については，示している参考文献等を参照されたい。

２．１．Householder QR 分解アルゴリズム

Householder QR 分解アルゴリズムは，直交変換として Householder 変換[5]

$$H = I - t\boldsymbol{u}\boldsymbol{u}^{\mathsf{T}}, \quad t = \frac{2}{\boldsymbol{u}^{\mathsf{T}}\boldsymbol{u}}$$

を用いた Orthogonal Triangularization 型アルゴリズムである。直交変換に基づく数値計算アルゴリズムは一般的に安定であることが多く，Householder QR アルゴリズムは，$\kappa_2(A)$に関わらず，計算精度が良いことが知られている。また，一般的な数値計算の教科書のアルゴリズムを素直に実装すると，行列ベクトル積相当の Level-2 BLAS による計算となるため，LAPACK 等では，Householder 変換のブロック化（Compact WY representation）[10]を用いた，行列積（Level-3 BLAS）中心のアルゴリズムが実装されている。基本的に，行列の列方向に計算の逐次性があり，各列に対応する計算で，列ベクトルに関する内積（や部分的な行列ベクトル積）が必要となり，分散並列計算の場合，$O(n)$回の集団通信が必要となる。そのため，大規模な分散並列計算におけ

る強スケーリングには限界がある。

２．２．TSQR アルゴリズム

TSQR アルゴリズムは，Demmel らにより提案された[8]，Orthogonal Triangularization かつ通信回避型のアルゴリズムである。アルゴリズムの主たるアイディアは，

$$\hat{Q}^{(3)}\begin{pmatrix}\hat{Q}_1^{(2)} & \\ & \hat{Q}_2^{(2)}\end{pmatrix}\begin{pmatrix}\hat{Q}_1^{(1)} & & & \\ & \hat{Q}_2^{(1)} & & \\ & & \hat{Q}_3^{(1)} & \\ & & & \hat{Q}_4^{(1)}\end{pmatrix}\begin{pmatrix}A_1\\A_2\\A_3\\A_4\end{pmatrix}\rightarrow R$$

のように，ブロック対角構造を持つ直交行列を段階的に作用させる点である。このような手順で計算する場合，最初の段階の

$$\hat{Q}_i^{(1)}A_i\rightarrow\begin{pmatrix}R_i^{(1)}\\O\end{pmatrix}$$

は，お互いに一切の通信を必要とせずに，並列に実行可能となる。その後，隣同士で得られた三角行列を（一対一通信により）送受信し，次の段階の

$$\hat{Q}_i^{(2)}\begin{pmatrix}R_i^{(1)}\\O\\R_{i+1}^{(1)}\\O\end{pmatrix}\rightarrow\begin{pmatrix}R_i^{(2)}\\O\\O\\O\end{pmatrix}$$

を計算する。ここでも，最初の段階と同様に，通信を必要とせず，並列に実行可能である。また，実際の計算では，上三角行列 2 個分に相当する$2n\times n$の密行列の QR 分解相当の演算のみで実装可能であり，さらに，三角行列の構造を配慮した実装も可能である。以下，三角行列の送受信と，三角行列の縮約計算を繰り返すことで，最終的に，元々の計算対象の行列の QR 分解のRを得ることができる。また，直交行列（の転置）を逆の順番で作用させることで，Qが得られる。なお，各段階で用いる直交行列は，一般的に Householder 変換である。したがって，TSQR アルゴリズムは，局所的な Householder QR 分解を繰り返している，と解釈することが可能である。

TSQR アルゴリズムも，直交変換を利用したアルゴリズムであるため，数値安定性に優れ，Householder QR アルゴリズムと同様に，$\kappa_2(A)$に関わらず，計算精度が良いことが知られている（より詳細には，Householder QR と異なる特徴を持つことも示されている）[11]。一方，分散並列計算において必要となる通信は，各段階の間の上三角行列の送受信であり，合計で$O(\log_2 P)$回程度の一対一通信となる（簡単のため，二分木に沿った通信を想定する）。より直感的には，TSQR アルゴリズムにおける通信は，三角行列を 2 個縦に並べた行列の QR 分解を縮約演算とする，三角行列に対する$O(1)$回の集団通信と解釈できる。これは，集団通信の回数において，Householder QR 分解アルゴリズムの$O(n)$よりも少なく（通信されるデータ量では両者は同程度），TSQR アルゴリズムが通信回避型アルゴリズムである所以である。

２．３．Gram-Schmidt の直交化に基づくアルゴリズム

Gram-Schmidt の直交化は，非常に長い歴史を持つ（1907 年出版の論文で提唱）数値計算アルゴリズムの一つである[12]。アルゴリズムの基本的な構造は，Aの列ベクトルに対するスケーリング（規格化）と，計算済み（直交化済み）のベクトルとの直交化の繰り返しである。ある列ベ

クトルに対するスケーリングは

$$A\begin{pmatrix} I_{i-1} & & \\ & s_i & \\ & & I_{n-i} \end{pmatrix}$$

と書くことができ，一方，直交化（直交補空間への射影）は

$$A\begin{pmatrix} I_{i-1} & \boldsymbol{r}_i & \\ & 1 & \\ & & I_{n-i} \end{pmatrix}$$

と書くことができる。そのため，Gram-Schmidt の直交化に基づく QR 分解アルゴリズムは，Triangular Orthogonalization 型と解釈できる。

具体的な数値計算アルゴリズムとしては，Classical Gram-Schmidt（CGS）と modified Gram-Schmidt（MGS）が有名である。両者は，数学的には同じであるが，演算順序が異なるため，数値計算的にはアルゴリズムの挙動が異なる。CGS も MGS も$\kappa_2(A)$が大きくなるにつれて，計算結果の精度（Qの直交性）が悪化することが知られており，CGS の方が MGS に比べて，精度低下の度合いが大きい。この問題を解決する手段として，再直交化（Reorthogonalization）があり，再直交化を付与した CGS（通称：CGS2）などがある。なお，再直交化は 1 回（全体で直交化は 2 回）で十分であることが示されており，「twice is enough」という有名なフレーズがある[13]。

CGS や MGS のアルゴリズムも，教科書に記載の手順を素直に実装すると，Level-2 BLAS による計算となる。ただし，CGS（や CGS2）については，ブロック化により，Level-3 BLAS 中心のアルゴリズムが構築可能である（MGS は演算順序の制約から不可能）[14]。また，分散並列計算では，Householder QR 分解と同様に，基本構造が列ごとの逐次的な処理であり，それに付随する列ベクトルの内積相当の計算のために，$O(n)$回の集団通信が必要となる。そのため，強スケーリングには不向きとなる。

２．４．Cholesky QR 分解に基づくアルゴリズム

計算対象の行列Aに対して，

$$A^{\mathsf{T}}A \to W, \quad W \to R^{\mathsf{T}}R, \quad AR^{-1} \to Q$$

とすることで，Aの QR 分解を（数学的には）得ることが可能である[3]。これを Cholesky QR 分解と呼ぶ。なお，2 番目の計算はWに対するコレスキー分解である。このアルゴリズムは，Aが縦長である場合，計算コストの主要部が，1 番目と 3 番目の計算となり，どちらも Level-3 BLAS で実行可能である。また，分散並列計算で必要となる通信は，グラム行列Wを得るための，集団通信 1 回のみである（TSQR と同様に，通信回避型アルゴリズムとみなせる）。したがって，実装が非常にシンプルである点も含めて，Cholesky QR 分解は，高性能計算に非常に適したアルゴリズムである。なお，上記の構造から，Cholesky QR 分解が Triangular Orthogonalization 型であることは明らかである。

しかし，数値計算の教科書等において，$A^{\mathsf{T}}A$のような行列を陽的に構成するアルゴリズムはタブー視されている（例：正規方程式を陽的に構成して，最小二乗問題を解く）。これは，行列の条件数が二乗になり，計算精度の悪化やアルゴリズムの破綻等につながるためである。Cholesky QR 分解でも同様であり，$\kappa_2(A)$に応じて，計算結果の精度（Qの直交性）が悪化し，実用的でないことが知られていた。

この問題に対して，近年，いくつかの進展があった。以前から，Cholesky QR 分解のアルゴリズムを繰り返すことで，Qの直交性が改善することが示唆されていた[15]が，実際に，$\kappa_2(A)$に関する現実的な条件（$\kappa_2(A) \leq O(10^8)$）の下で，1 回の再直交化を付与（CholeskyQR2 アルゴリズム）すれば，十分な精度で QR 分解が計算できる（条件付きではあるが，「twice is enough」である）ことが示され[16]，その有効性が確認された[17]。また，さらに$\kappa_2(A)$が大きい場合に対して，部分的に高精度（擬似 4 倍精度）演算を用いる混合精度型の Cholesky QR 分解アルゴリズム[18]，計算されたグラム行列Wに対するシフトを導入した Shifted Cholesky QR 分解を前処理として用いる Shifted-CholeskyQR3 アルゴリズム[19]，Aに対する部分ピボット選択付き LU 分解を前処理とする，LU-CholeskyQR 型アルゴリズム[20]などが提案されている。これらのアルゴリズムは，主要部が Cholesky QR 分解の構造となっているため，演算と通信の両面において，依然として利点を維持している。

３．各種アルゴリズムの特徴

前節で紹介した，縦長行列の QR 分解を計算する様々なアルゴリズムの特徴を整理する。具体的には，計算時間に関する項目として，演算量(#Flops)，通信回数(#Msgs)，通信データ量(#Words)をまとめる。なお，通信回数と通信データ量については，集団通信を単位とする（$\log_2 P$回の一対一通信を 1 回の集団通信相当としてカウントする)。また，計算精度に関する項目として，計算されたQの直交性（$\|Q^\top Q - I_n\|_2$）を比較する。

各アルゴリズムについて整理した結果を表 1 に示す。なお，CholQR は（再直交化なしの）Cholesky QR，CholQR2 は CholeskyQR2，S-CholQR3 は Shifted-CholeskyQR3，LU-CholQR は LU-CholeskyQR，LU-CholQR2 は LU-CholeskyQR2（LU-CholeskyQR に再直交化を付与したもの），HQR は Householder QR である（以降，同様の略記を用いる)。ϵは unit roundoff であり，本稿では，倍精度を仮定しているので，$\epsilon = 2^{-53} \simeq 10^{-16}$である。ただし，$\kappa_2(A) \leq \epsilon$は仮定している。また，$Q$の直交性に関して，解析的な結果が示されているケースについては，参考文献を付与している（参考文献の記載がないケースは，経験的に知られた情報である)。

表 1 から分かるように，いずれのアルゴリズムも演算量は$O(mn^2/P)$であるが，TSQR のみPに応じて増加する項（上三角行列の縮約のための演算）があることに注意が必要である。一方，通信回数は，非通信回避型が$O(n)$，通信回避型が$O(1)$である。LU-CholQR と LU-CholQR2 が$O(n)$なのは，部分ピボット選択付き LU 分解におけるピボット操作のためである。通信データ量については，どのアルゴリズムも$O(n^2)$である。これは，非通信回避型の場合は一度の通信量が少なく（スカラーもしくはベクトル)，通信回避型の場合は一度の通信が行列であるためである。なお，通信回数が$O(1)$のアルゴリズムに関して，通信回数（あるいは通信データ量）の比は概ね

$$\mathrm{CholQR} : \mathrm{CholQR2} : \mathrm{S-CholQR3} : \mathrm{TSQR} = 1 : 2 : 3 : 1$$

となっている。

計算精度（Qの直交性）については，CGS と CholQR が$\kappa_2^2(A)$に応じて（かつ，CholQR は途中でアルゴリズムが破綻)，MGS と LU-CholQR が$\kappa_2(A)$に応じて，それぞれ低下する。一方，それ以外のアルゴリズムは，（$\kappa_2(A) \leq \epsilon$の仮定の下では）問題ないが，CholQR2 については，$\kappa_2(A)$が大きくなると，アルゴリズムが破綻するため，注意が必要である。なお，余談であるが，CGS については，解析的には$O(\epsilon \cdot \kappa_2^{n-1}(A))$が上界となることが示されている[21]が，経験的には$O(\epsilon \cdot \kappa_2^2(A))$程度であることが知られている（MGS は$O(\epsilon \cdot \kappa_2(A))$が上界であることが示されている)。

表 1：各種 QR 分解アルゴリズムの特徴。

Algorithm	#Flops	#Msgs	#Words	Orthogonality of Q
CGS	$\frac{2mn^2}{P}+O(n^3)$	$O(n)$	$O(n^2)$	$O(\epsilon\cdot\kappa_2^2(A))$ ※$O(\epsilon\cdot\kappa_2^{n-1}(A))$ [21]
CGS2	$\frac{4mn^2}{P}+O(n^3)$	$O(n)$	$O(n^2)$	$O(\epsilon)$
MGS	$\frac{2mn^2}{P}+O(n^3)$	$O(n)$	$O(n^2)$	$O(\epsilon\cdot\kappa_2(A))$ [22]
CholQR	$\frac{2mn^2}{P}+O(n^3)$	$O(1)$	$O(n^2)$	$O(\epsilon\cdot\kappa_2^2(A))$ if $\kappa_2(A)\leq\frac{1}{\sqrt{\epsilon}}$ [16]
CholQR2	$\frac{4mn^2}{P}+O(n^3)$	$O(1)$	$O(n^2)$	$O(\epsilon)$ if $\kappa_2(A)\leq\frac{1}{\sqrt{\epsilon}}$ [16]
S-CholQR3	$\frac{6mn^2}{P}+O(n^3)$	$O(1)$	$O(n^2)$	$O(\epsilon)$ [19]
LU-CholQR	$\frac{3mn^2}{P}+O(n^3)$	$O(n)$	$O(n^2)$	$O(\epsilon\cdot\kappa_2(A))$
LU-CholQR2	$\frac{5mn^2}{P}+O(n^3)$	$O(n)$	$O(n^2)$	$O(\epsilon)$ [20]
HQR	$\frac{4mn^2}{P}+O(n^3)$	$O(n)$	$O(n^2)$	$O(\epsilon)$ [9]
TSQR	$\frac{4mn^2}{P}+O(n^3\log_2 P)$	$O(1)$	$O(n^2)$	$O(\epsilon)$ [11]

４．Oakforest-PACS 上での性能評価

Oakforest-PACS を用いて，各種 QR 分解アルゴリズムの性能を評価した結果を報告する。評価環境は表 2 の通りである。プログラムは Fortran90 で，BLAS や LAPACK の関数を必要に応じて利用する形で実装した。なお，ScaLAPACK のルーチンは使用しておらず，MPI を用いて，独自に分散並列化している。LU-CholQR 型アルゴリズムの LU 分解部分については，（本来の LU 分解の意味では必要であるが）今回のケースでは不要となるデータ通信を削除した実装を用いている。また，S-CholQR3 のシフト量は$\sqrt{m}\|A\|_F^2\times10^{-16}$とした。

表 2：性能評価環境。

システム名	Oakforest-PACS (OFP)
CPU	Intel Xeon Phi 7250 (KNL), 1.40 GHz, 68 cores
メモリモード	Flat (MCDRAM のみ使用)
インターコネクト	Intel Omni-Path
コンパイラ&オプション	Intel ifort (ver. 18.0.1) -O3 -align array64byte -qopenmp-ipo-xMIC-AVX512
数値計算ライブラリ	Intel MKL (ver. 2018.0.1) by -mkl=parallel

４．１．計算精度に関する評価

まず，各アルゴリズムの計算精度に関する評価結果を示す。テスト行列は，意図した条件数に対して，

$$\Sigma \coloneqq \mathrm{diag}\left(1, \sigma^{\frac{1}{n-1}}, \ldots, \sigma^{\frac{n-2}{n-1}}, \sigma\right), \quad \sigma \coloneqq \frac{1}{\kappa_2(A)}$$

とした上で，

$$A \coloneqq U\Sigma V^{\top}$$

と生成した。なお，$U \in R^{m\times n}, V \in R^{n\times n}$はランダムに生成した直交行列である。

評価結果を表３に示す。概ね，表１の内容と一致していることが確認できる。なお，S-CholQR3 が，$\kappa_2(A) = 10^{15}$の場合に破綻しているが，Shifted Cholesky QR による前処理が１回では不十分であったことを意味しており，２回前処理を行うことで回避可能である（Shifted Cholesky QR を２回行った後に，CholeskyQR2 を行う）。

表３：計算精度の評価（$m = 16777216, n = 128, P = 4096$）。

(a) 直交性：$\|Q^{\top}Q - I\|_F/\sqrt{n}$

$\kappa_2(A)$	1.00E+00	1.0E+03	1.0E+06	1.0E+09	1.0E+12	1.0E+15
CGS	2.06E-16	7.24E-13	1.05E-06	1.17E+00	3.39E+00	4.75E+00
CGS2	1.83E-16	1.83E-16	1.69E-16	1.60E-16	1.60E-16	1.57E-16
MGS	2.02E-16	1.01E-14	4.17E-12	2.94E-09	2.29E-06	2.34E-03
CholQR	1.67E-16	2.73E-12	1.69E-06	破綻	破綻	破綻
CholQR2	1.71E-16	2.06E-16	2.09E-16	破綻	破綻	破綻
S-CholQR3	1.63E-16	2.19E-16	2.00E-16	2.21E-16	2.08E-16	破綻
LU-CholQR	2.69E-15	2.75E-14	1.37E-11	1.06E-08	5.89E-06	6.12E-03
LU-CholQR2	1.76E-16	2.14E-16	1.91E-16	1.92E-16	2.44E-16	2.52E-16
HQR	2.92E-16	2.77E-16	2.72E-16	2.75E-16	2.71E-16	2.74E-16
TSQR	3.00E-16	2.69E-16	2.76E-16	2.54E-16	2.88E-16	2.82E-16

(b) 残差：$\|A - QR\|_F/\|A\|_F$

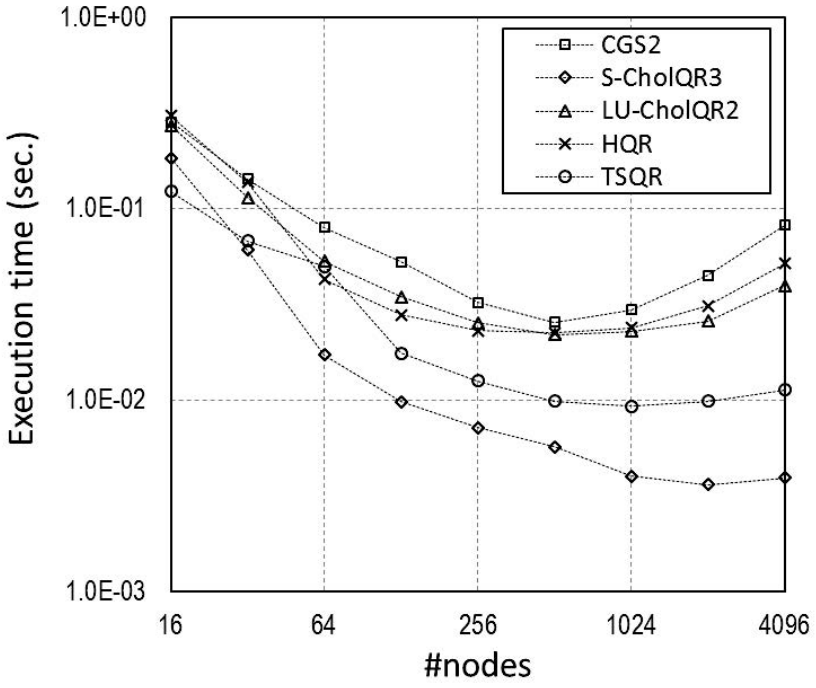
$\kappa_2(A)$	1.00E+00	1.0E+03	1.0E+06	1.0E+09	1.0E+12	1.0E+15
CGS	1.84E-16	3.75E-16	3.77E-16	3.79E-16	3.81E-16	3.81E-16
CGS2	2.33E-16	3.82E-16	3.82E-16	3.83E-16	3.84E-16	3.85E-16
MGS	1.84E-16	3.75E-16	3.77E-16	3.79E-16	3.81E-16	3.81E-16
CholQR	1.08E-16	1.74E-16	1.73E-16	破綻	破綻	破綻
CholQR2	1.23E-16	4.87E-16	5.32E-16	破綻	破綻	破綻
S-CholQR3	1.46E-16	5.31E-16	5.57E-16	5.81E-16	5.91E-16	破綻
LU-CholQR	1.59E-16	1.74E-16	1.72E-16	1.72E-16	1.72E-16	1.72E-16
LU-CholQR2	1.64E-16	4.59E-16	4.93E-16	5.24E-16	5.32E-16	5.52E-16
HQR	2.30E-16	4.88E-16	5.27E-16	5.14E-16	4.90E-16	6.45E-16
TSQR	4.05E-16	5.12E-16	5.12E-16	4.96E-16	4.66E-16	6.25E-16

４．２．計算時間に関する評価

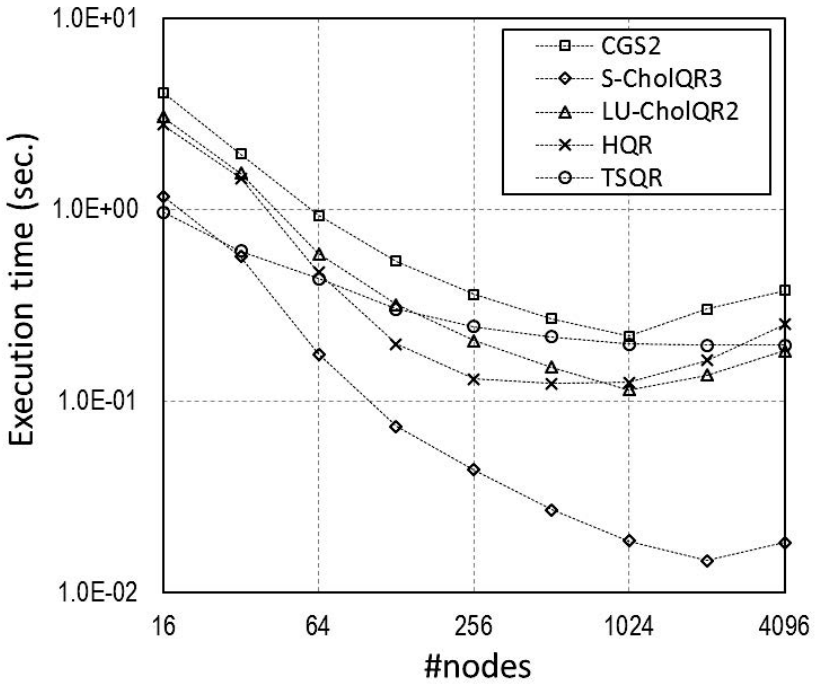
次に，各アルゴリズムの計算時間を評価した結果を報告する。以下では，$\kappa_2(A)$がそれなりに大きい場合でも精度良く計算可能な，CGS2，S-CholQR3，LU-CholQR2，HQR，TSQR を比較する。計算ノード数を 16 から 4096 まで変化（強スケーリング）させて，計算時間を測定した。なお，ノード上のプロセス数とスレッド数については，1 プロセス×64 スレッド，4 プロセス×16 スレッド，16 プロセス×4 スレッド，64 プロセス×1 スレッド，の 4 通りを試し（各設定で 5 回計測），各アルゴリズム（の各ノード数）で最も計算時間が短かった場合を評価対象としている。

$n = 64,256$の場合の結果を図 1 に示す。図 1(a)，(b)から分かるように，どちらの場合でも，S-CholQR3 が最も高速で，良好な強スケーリングを示している。4096 ノードを使用した場合において，$n = 64$では，2 番目に高速である TSQR の 2.9 倍，$n = 256$では，LU-CholQR2 の 10 倍，S-CholQR3 は高速であった。また，どちらの場合においても，16 ノードの場合に対して，4096 ノードにおいて，47 倍の高速化となっている．

次に，4096 ノードの場合における，各アルゴリズムの計算時間の内訳を図 2 に示す。なお，「Reduction Comput.」は TSQR における三角行列を 2 個並べた行列の QR 分解（縮約処理）の演算時間である。図 2 から分かるように，通信回数が$O(n)$である CGS2，LU-CholQR2，HQR では，通信時間（MPI Comm.）が大部分を占めており，これが強スケーリングを阻害する原因となっている。一方，TSQR については，通信時間は少ないが，Reduction Comput.が目立っており，特に，$n = 256$において，大きなボトルネックとなっている。この部分は，$O(n^3)$で増加するため，nが大きくなると，途端にボトルネックとなってしまう[23, 24]。これらのアルゴリズムに対して，S-CholQR3 は，通信時間と演算時間（Others）の両方が少なく，その結果，良好な性能が得られている。なお，演算時間も目立っていないのは，前処理部分の Shifted Cholesky QR 部分を含めて，Level-3 BLAS で大部分が計算可能であるという，Cholesky QR 元々の利点が維持されているからである。表 3 に関連して，$\kappa_2(A)$が非常に大きい場合，前処理（Shifted Cholesky QR）の追加が必要であると述べたが，その場合，計算時間は約 4/3 倍となる。しかし，図 1 の結果を踏まえると，今回の状況では，依然として計算時間の面で優位であることが推測できる。

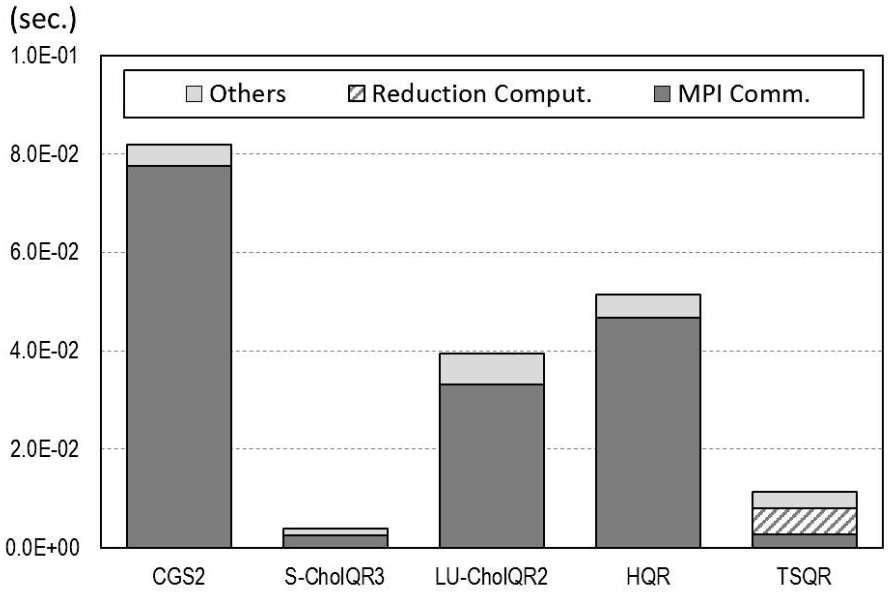


(a) $n = 64$

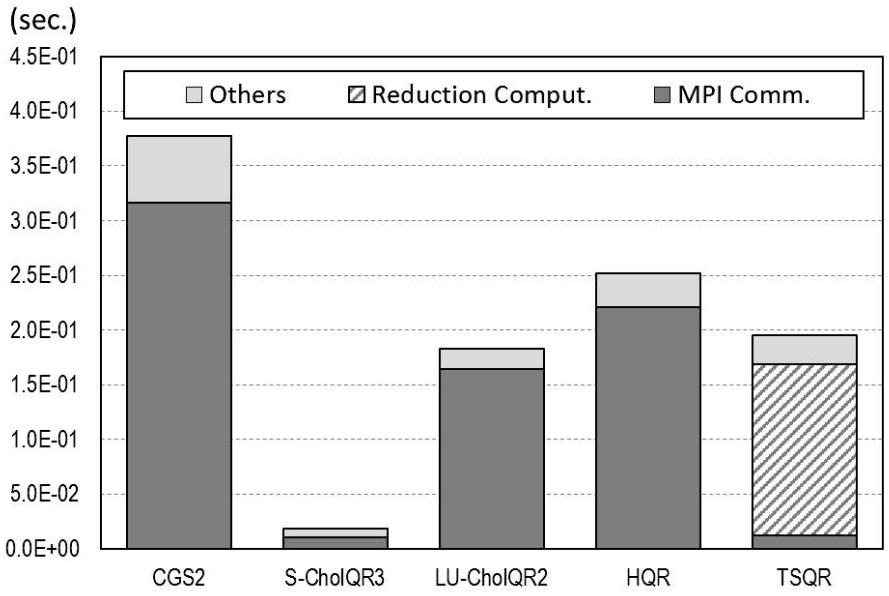


(b) $n = 256$

図 1：計算時間の評価結果（強スケーリング，$m = 16777216$）。



(a) $n = 64$



(b) $n = 256$

図 2：計算時間の内訳（$P = 4096$，$m = 16777216$）。

５．おわりに

本稿では，縦長行列の QR 分解を計算するための各種アルゴリズムを概説し，Oakforest-PACS 上で実施した性能評価結果を示しながら，アルゴリズムを比較した。計算対象の行列が比較的難しい（$\kappa_2(A)$が大きい）場合において，精度良く計算が可能なアルゴリズムの計算時間を，最大 4096 ノードまでを用いた強スケーリングで評価した結果，今回の状況では，S-CholQR3 が最も優れていることが確認された。S-CholQR3 は，演算量の面では他のアルゴリズムに劣るが，通信回避型（集団通信の回数が$O(1)$）であり，演算は行列積等の Level-3 BLAS が中心であるため，最近の高性能計算環境に非常に適している。この点が今回の評価結果につながったと考えられる。

最後に，本稿では紹介できなかったが，関連する研究をいくつか紹介する。まず，アルゴリズムの安定性や計算精度に関しては，最近の文献[25]で詳しい議論がなされている。次に，Cholesky QR 型のアルゴリズムについて，行列が縦長ではない場合を想定した，2.5 次元実装が提案されている[26]。その他にも，同期を削減した QR 分解アルゴリズムとして，タイル QR 分解アルゴリズム[27]が知られている。

謝　辞

大規模 HPC チャレンジの実施に関してお世話になりました，東京大学情報基盤センターの関係者の皆様，及び，当日の不具合対応でお世話になりました，富士通株式会社の関係者の皆様に深く感謝いたします。また，文献[19]をはじめとする，Cholesky QR 分解関連の共同研究においてお世話になっている，山本有作 教授（電気通信大学），中務佑治 准教授（University of Oxford），柳澤優香 博士（早稲田大学，文献[19]投稿時），Dr. Ramaseshan Kannan（Arup, UK）に深く感謝いたします。本研究の一部は，JSPS 科研費（課題番号：18K18058）の支援を受けています。

参　考　文　献

[1] L. N. Trefethen and I. David Bau, Numerical Linear Algebra, SIAM, 1997.

[2] Å. Björck, Numerical Methods for Least Squares Problems, SIAM, 1996.

[3] G. H. Golub and C. F. Van Loan, Matrix Computations, 3rd ed., Johns Hopkins University Press, 1996.

[4] 櫻井鉄也，松尾宇泰，片桐孝洋，数値線形代数の数理と HPC，共立出版，2018.

[5] 杉原正顯，室田一雄，線形計算の数理，岩波書店，2009.

[6] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, Minimizing communication in numerical linear algebra, SIAM J. Matrix Anal. Appl., Vol.32 (2011), pp.866-901.

[7] J. Demmel, Communication-Avoiding Algorithms for Linear Algebra, Machine Learning, and Beyond, E-NLA seminar, 2020. https://www.youtube.com/watch?v=42f0n0w2Nlg

[8] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, Communication optimal parallel and sequential QR and LU factorizations, SIAM J. Sci. Comput., Vol.34 (2012), pp.206-239.

[9] N. J. Higham, Accuracy and Stability of Numerical Algorithms, SIAM, 2002.

[10] R. Schreiber and C. Van Loan, A storage-efficient WY representation for products of Householder transformations, SIAM J. Sci. Stat. Comput., Vol.10 (1989), pp.53-57.

[11] D. Mori, Y. Yamamoto, and S.-L. Zhang, Backward error analysis of the Allreduce algorithm for Householder QR decomposition, Japan J. Indust. Appl. Math., Vol.29 (2012), pp.111-130.

[12] S. J. Leon, Å. Björck, and W. Gander, Gram-Schmidt orthogonalization: 100 years and more, Numer. Linear Algebra Appl., Vol.20 (2013), pp.492-532.

[13] B. N, Parlett, The Symmetric Eigenvalue Problem, SIAM, 1980.

[14] G. W. Stewart, Block Gram-Schmidt orthogonalization, SIAM J. Sci. Comput., Vol.31 (2008), pp.761-775.

[15] A. Stathopoulos and K. Wu, A block orthogonalization procedure with constant synchronization requirements, SIAM J. Sci. Comput., Vol.23 (2002), pp.2165-2182.

[16] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, and T. Fukaya, Roundoff error analysis of the CholeskyQR2 algorithm, Electron. Trans. Numer. Anal., Vol.44 (2015), pp.306-326.

[17] T. Fukaya, Y. Nakatsukasa, Y. Yanagisawa, and Y. Yamamoto, CholeskyQR2; A simple and communication-avoiding algorithm for computing a tall-skinny QR factorization on a large-scale parallel system, Proc. Scala'14, pp.31-38, 2014.

[18] I. Yamazaki, S. Tomov, and J. Dongarra, Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs, SIAM J. Sci. Comput., Vol.37 (2015), pp.C307-C330.

[19] T. Fukaya, R. Kannan, Y. Nakatukasa, Y. Yamamoto, and Y. Yanagisawa, Shifted Cholesky QR for computing the QR factorization of ill-conditioned matrices, SIAM J. Sci. Comput., Vol.42 (2020), pp.A477-A503.

[20] T. Terao, K. Ozaki, and T. Ogita, LU-Cholesky QR algorithms for thin QR decomposition, Parallel Comput., Vol.92 (2020), pp.102571.

[21] A. Kiełbasiński, Analiza numeryczna algorytmu ortogonalizacji Grama-Schmidta, Ser. III Mat. Stosow. II, (1974), pp.15-35 (in Polish).

[22] Å. Björck, Solving linear least squares problems by Gram-Schmidt orthogonalization, BIT, Col.7 (1967), pp.1-21.

[23] T. Fukaya, T. Imamura, and Y. Yamamoto, Performance analysis of the Householder-Type Parallel tall-skinny QR factorizations toward automatic algorithm selection, LNCS, Vol.8969 (2015), pp.269-283.

[24] T. Fukaya, An investigation into the impact of the structured QR kernel on the overall performance of the TSQR algorithm, Proc. HPC Asia 2019, pp.81-90, 2019.

[25] E. Carson, K. Lund, M. Rozložník, and S. Thomas, An overview of block Gram-Schmidt methods and their stability properties, arXiv:2010.12058v1, 2020.

[26] E. Hutter and E. Solomonik, Communication-avoiding Cholesky-QR2 for rectangular matrices, Proc. IPDPS 2019, pp.89-100, 2019.

[27] B. Hadri, H. Ltaief, E. Agullo, J. Dongarra, Tile QR factorization with parallel panel processing for multicore architectures, Proc. IPDPS 2010, pp.1-10, 2010.