Vol.2022-HPC-184 No.2
2022/5/11

分散並列環境上での縦長行列のQR分解に対する各種アルゴリズムの性能評価

深谷 猛[1,a)]

概要：本稿では，4 つのスーパーコンピュータシステム上で，縦長行列の QR 分解に関して，異なる特徴を持つ 4 種類の数値計算アルゴリズムの実行時間を，強スケーリングの観点で評価した結果を報告する．QR 分解の数値計算アルゴリズムとしては，教科書等で広く知られているものから，近年の研究の成果として提案された新しいものまで，様々なものが存在する．これらは，演算や通信に関して異なる特徴を持っており，通信回避型アルゴリズムはその一例である．これらのアルゴリズムの性能（実行時間）は，その特徴から，理論的な議論がある程度可能であるが，一方で，実際の計算機システム上で実行することで初めて分かる知見もあり，特に，大規模な分散並列計算では顕著である．今回の性能評価により，各アルゴリズムの特徴をより詳しく把握するとともに，今後のアルゴリズムの研究開発に向けた課題を明らかにすることを目指す．

1. はじめに

本稿では，分散並列計算機における，縦長（Tall-Skinny）行列の QR 分解を考える．QR 分解は，基本的な行列分解の一つであり，最小二乗問題の求解 [1] や特異値分解の前処理 [2] などの応用を持つ．また，縦長行列の QR 分解は，ベクトルの直交化 [3] や密行列の帯行列化 [4] と深く関係しており，線形方程式や固有値・特異値問題のための数値計算アルゴリズムを構成する道具としても利用される．

行列の QR 分解を計算する数値計算アルゴリズムとしては，Gram-Schmidt の直交化と Householder QR 分解が広く知られている [5]．一方，大規模分散並列環境では，ベクトルの内積などの計算で必要となる集団通信（例：`MPI_Allreduce`）のコスト（特にレイテンシ）が強スケーリング時のボトルネックとなることが多い．それに対して，通信回避（CA：Communication-Avoiding） [6] の重要性が指摘され，縦長行列の QR 分解に対する具体的なアルゴリズムとして，TSQR アルゴリズム [7] が提案された．また，近年，TSQR と異なる別のアプローチである，Cholesky QR 型アルゴリズムについても進展 [8], [9], [10] が報告された．

このように，現在，縦長行列の QR 分解に対して，異なる特徴を持った様々なアルゴリズムが存在している．本稿では，これらのうちの代表的な 4 種類のアルゴリズムについて，異なる 4 種類のスーパーコンピュータシステムを用いて性能評価を行い，強スケーリングの観点で実行時間を検証した結果を報告する[*1]．これにより，各アルゴリズムの特徴をより詳しく把握し，同時に，今後のアルゴリズムの研究開発に向けた課題を明らかにすることを目指す．

以下，2 節で問題設定を提示し，様々なアルゴリズムを概観する．その後，3 節で今回の性能評価で対象とするアルゴリズムの特徴を述べる．そして，4 節で性能評価の結果を報告する．最後に，5 節で本稿のまとめを述べる．

2. 問題設定と様々なアルゴリズムの概観

2.1 問題設定

縦長の行列 $A \in \mathbb{R}^{m\times n}$ $(m \gg n)$ の Thin QR 分解（Reduced QR 分解） [12].

$$A = QR \tag{1}$$

を考える．ここで，$Q \in \mathbb{R}^{m\times n}$ は列直交行列（$Q^{\top}Q = I_n$ を満たす[*2]），$R \in \mathbb{R}^{n\times n}$ は上三角行列である．また，今回の性能評価では，Q を陽的に計算する場合を想定する[*3]．

行列の 2 ノルムによる条件数は，

$$\kappa_2(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \tag{2}$$

で定義される．ここで，$\sigma_{\max}(A)$ は A の最大特異値，$\sigma_{\min}(A)$

[1] 北海道大学 情報基盤センター
[a)] fukaya@iic.hokudai.ac.jp

[*1] 記事 [11] にて，Oakforest-PACS 上で実施した同様の性能評価結果を報告している．本稿の内容は，これを拡充したものとなる．なお，本稿で提示する Oakforest-PACS における結果は，プログラムの一部修正等の後に，改めて性能評価を実施したものである．
[*2] I_n：n 次の単位行列．
[*3] 応用によっては，Q を直交行列の積などの形で陰的に求めれば十分な場合もある．

は最小特異値である．本稿では，倍精度浮動小数点（FP64）を用いる場合を想定し，A が数値的に列フルランクであることを仮定する．これは，$\kappa_2(A) \leq 10^{16}$ を仮定することを意味する．

本稿では，分散並列（MPI 並列）を想定している．その際，計算対象の行列 A は，

$$A = \begin{pmatrix} A_1 \\ \vdots \\ A_P \end{pmatrix} \tag{3}$$

のような，行方向の 1 次元ブロック分散を仮定する．ここで，P はプロセス数（並列数）である．また，計算結果の Q も同様のデータ分散を仮定する．なお，R については，少なくとも一つのプロセスが全体を保持している（全てのプロセスが保持している必要はない），とする．

2.2 様々なアルゴリズムの概観

QR 分解の数値計算アルゴリズムは，Orthogonal Triangularization 型と Triangular Orthogonalization 型に大別できる [12]．Orthogonal Triangularization 型アルゴリズムは，直交変換 $\hat{Q}_1, \ldots, \hat{Q}_k \in \mathbb{R}^{m \times m}$ により，

$$\hat{Q}_k \cdots \hat{Q}_2 \hat{Q}_1 A \to \begin{pmatrix} R \\ O \end{pmatrix}, \tag{4}$$

と A を上三角行列に変形する．さらに，

$$\left(\hat{Q}_k \cdots \hat{Q}_2 \hat{Q}_1\right)^{-1} \begin{pmatrix} I_n \\ O \end{pmatrix} \to Q, \tag{5}$$

として，Q を陽的に得る．Orthogonal Triangularization 型の代表的なアルゴリズムは Householder QR である．また，通信回避の観点で提案された TSQR アルゴリズムもこのクラスに属する．それ以外には，Givens 回転を用いるアルゴリズムや，Tile QR [13] などがある．

一方，Triangular Orthogonalization 型アルゴリズムは，上三角行列 $\hat{R}_1, \ldots, \hat{R}_k$ により，

$$A\hat{R}_1 \hat{R}_2 \cdots \hat{R}_k \to Q \tag{6}$$

と A を列直交行列に変形する．さらに，

$$\left(\hat{R}_1 \hat{R}_2 \cdots \hat{R}_k\right)^{-1} \to R, \tag{7}$$

として，R を得る．Triangular Orthogonalization 型の代表的なアルゴリズムは，Gram-Schmid の直交化に基づく種々のアルゴリズム [14] である．また，Cholesky QR 型の各アルゴリズム [8], [9], [10], [15], [16] もこのクラスとなる．

3. 評価対象のアルゴリズム

今回の性能評価では，**表 1** に挙げた 4 種類のアルゴリズムを対象とする．なお，本稿では，全プロセスにまたがる Collective 通信（集団通信）の回数が行列サイズ（主に n）に依存しない（$O(1)$ 回である）アルゴリズムを「通信回避型」と呼ぶ．表 1 に挙げた 4 種類のアルゴリズムは，計算対象の行列が悪条件（例：$\kappa_2(A) \simeq 10^{14}$）の場合でも，破綻せずに十分な精度で計算可能なものである[*4]．各アルゴリズムの計算精度に関する評価結果については，記事 [11] を参照されたい．

今回の目的は，各アルゴリズムの実行時間を強スケーリングの観点で評価することである．そこで，各アルゴリズムの演算量，通信回数，通信データ量を**表 2** に示す．表 2 に示した各値は，並列計算時のクリティカルパス上の値となっている．ここで，演算量の項目で，$\log_2 P$ が含まれる項は，集団通信のリダクションに伴う演算量（二分木に従って行うことを仮定）である．TSQR 以外のアルゴリズムでは，MPI 関数の一部として陰的に実行されるものとなる．なお，演算量は主要部分のみを記載している（例：$O(n^3)$ の項などは無視している）．また，通信回数と通信データ量は，全プロセスにまたがる集団通信を単位として算出したものである．TSQR に関しては，一連の一対一通信全体を集団通信に相当するものとみなして記載している（詳細は論文 [17] に委ねる）．

以下では，各アルゴリズムの特徴を簡単に紹介する．詳細については，記事 [11] や各参考文献等を参照されたい．

3.1 Householder QR（HQR）

Householder QR は，数値線形代数分野における代表的な直交変換の一つである，Householder 変換 [5] を用いた Orthogonal Triangularization 型の QR 分解アルゴリズムである．無条件安定で精度の良い QR 分解の計算手法として，一般的な数値計算の教科書で紹介されている．Level-3 BLAS（例：行列積）に基づいた高性能な実装方法としては，Compact WY 表現 [18] に基づいたブロック版のアルゴリズム [3] が知られている．列方向の逐次性があり，各列に対応する計算において，列ベクトルの内積等が必要となるため，分散並列計算では $O(n)$ 回の集団通信が必要となる．

3.2 再直交化付き古典 Gram-Schmidt（CGS2）

Gram-Schmidt の直交化は 100 年以上の歴史を持つアルゴリズム [14] である．アルゴリズムは，ベクトルのスケーリングと，直交化（直交補空間への射影）で構成され，どちらも右からの三角行列の作用で表記できるため，Triangular

[*4] より正確には，HQR と TSQR は，本稿の $\kappa_2(A) \leq 10^{16}$ の仮定と関係なく，無条件に安定（ゼロによる除算や負の数の平方根等によるアルゴリズムの破綻がない）である．一方，CGS2 は $\kappa_2(A) \leq 10^{16}$ の条件の下で安定である．S-CholQR3 については，厳密には，$\kappa_2(A) \simeq 10^{16}$ の場合などに破綻する可能性がある．ただし，内部の反復を増やす（例：3 回から 4 回）ことで，破綻を回避することが可能である．なお，計算精度に関しては，いずれのアルゴリズムも十分なものとなっている．

表 1 評価対象のアルゴリズムの位置づけ.

	Orthogonal Triangularization 型	Triangular Orthogonalization 型
従来型	Householder QR（HQR）	再直交化付き古典 Gram-Schmidt（CGS2）
通信回避型	TSQR	Shifted CholeskyQR3（S-CholQR3）

表 2 評価対象のアルゴリズムの演算量・通信回数・通信データ量：並列数を P として，並列計算時のクリティカルパス上の値を記載している．また，通信回数と通信データ量は全プロセスにまたがる集団通信を単位として算出している．

アルゴリズム	演算量	通信回数	通信データ量
HQR	$\frac{4mn^2}{P} + O(n^2 \log_2 P)$	$O(n)$	$O(n^2)$
CGS2	$\frac{4mn^2}{P} + O(n^2 \log_2 P)$	$O(n)$	$O(n^2)$
TSQR	$\frac{4mn^2}{P} + O(n^3 \log_2 P)$	$O(1)$	$O(n^2)$
S-CholQR3	$\frac{6mn^2}{P} + O(n^2 \log_2 P)$	$O(1)$	$O(n^2)$

Orthogonalization 型アルゴリズムに分類される．具体的な数値計算アルゴリズムとしては，古典（Classical）Gram-Schmidt（CGS）や修正（Modiied）Gram Schmidt（MGS）など，様々なバリエーションが存在する．一般的に，計算対象の行列の条件数に応じて計算精度（Q の直交性）が悪化する特徴を持ち，その解決方法として，再直交化（Reorthogonalization）が知られている．今回取り扱う CGS2 は，古典 Gram-Schmidt に再直交化を付与したアルゴリズムである．Gram-Schmidt 型アルゴリズム全般として，列方向の逐次性があり，各列に対応する計算で，ベクトルの内積等の計算が必要となる．そのため，HQR と同様，分散並列計算では $O(n)$ 回の集団通信が必要となる．なお，Level-3 BLAS の利用を目的としたブロック化の方法も存在する [19] が，計算精度に関する理論的な根拠を持つアルゴリズムが複雑で，また，一般的な実装方法が定まっていないため，今回の性能評価では取り扱わないこととする．

3.3 TSQR

TSQR は，通信回避の目的で提案されたアルゴリズム [7] である．まず，独立して，各プロセスが自分が持つ部分行列の QR 分解を計算する．その後，得られた上三角行列について，一対一通信と三角行列を上下に並べた行列の QR 分解（structured QR 分解）を繰り返すことで，最終的な R に集約する．文献 [7] で示されたアルゴリズムは，各部分の QR 分解に Householder QR（やその変種）を用いており，数学的には特殊な構造を持った直交変換による上三角化となる．そのため，Orthogonal Triangularization 型に分類され，Householder QR と同様に，安定性や計算精度に関して優れた特徴を持つ [20]．分散並列計算における通信パターンは，リダクション操作として structured QR 分解を伴った集団通信（`MPI_Reduce` に相当）と解釈できる．そのため，通信回数は $O(1)$ となる．一方，リダクション操作が QR 分解のため，$O(n^3 \log_2 P)$ の演算量が必要となり，n が大きい場合に，これが問題となることがある [17]．また，structured QR 分解の実装が TSQR 全体の性能に大きく影響を与える場合がある [21] ので，注意が必要である．

3.4 Shifted CholeskyQR3（S-CholQR3）

行列 A のグラム行列（$W = A^\top A$）のコレスキー分解（$W \to R^\top R$）を用いて，（数学的には）A の QR 分解を計算できる（$Q = AR^{-1}$）[2]．これを Cholesky QR 分解と呼び，その手順が示すように，Triangular Orthogonalization 型に属する．縦長行列の場合，計算の主要部が Level-3 BLAS の関数 2 つのみで構成されるため，単純かつ高い実行性能が期待できるアルゴリズムである．また，分散並列計算では，グラム行列の計算で集団通信を 1 回だけ必要とするため，通信回避型アルゴリズムとなる．しかし，アルゴリズムが不安定（$\kappa_2(A) \gtrsim 10^8$ で破綻）で，計算精度（Q の直交性）が条件数に応じて悪化するという欠点を持つ．この問題に対して，安定性や計算精度の改善を目的として，再直交化を付与したアルゴリズム（CholeskyQR2）[8], [9]，部分的に高精度演算を利用した混合精度型アルゴリズム [15]，部分軸選択付き LU 分解を前処理に用いたアルゴリズム（LU CholeskyQR2）[16] などが提案された．本稿で取り扱う Shifted CholeskyQR3 は，グラム行列に正の対角シフトを加えた行列のコレスキー分解を前処理として用いる手法 [10] である．このアルゴリズムは，構造としては，Cholesky QR 分解を 3 回繰り返すものに相当する．そのため，演算と通信コストが 3 倍になるが，Level-3 BLAS が中心で通信回避型である，という長所は維持される．

4. 性能評価

4.1 各アルゴリズムの実装の概要

前節で挙げた 4 種類のアルゴリズムの実装の概要を述べる．全てのプログラムは Fortran90 で実装した．できるだけ，BLAS や LAPACK の関数を利用することとし，MPI により分散並列化を行った．なお，スレッド並列化に関しては，スレッド並列版の BLAS や LAPACK を用いることとし，独自にスレッド並列化は行っていない．以下，各アルゴリズム固有の点について列挙する．

- HQR：固定幅によるブロック化を行ったアルゴリズムを実装した．ブロック幅の候補は $l = 4, 8, 16, 32, 64, 128, 256$（ただし，$l \leq n$）とした．
- CGS2：文献 [14] の 4.2 節で示されている CGS2 のアルゴリズム（スケーリングと直交化を単純に繰り返す）を実装した．前節で述べたように，ブロック化は採用していない．
- TSQR：各プロセスが最初に行う QR 分解は LAPACK の `dgeqr`（利用不可の場合は `dgeqrf`）を利用した．structured QR 分解の部分は，手動でブロック化を施した実装を採用した（詳細は文献 [21] に委ねる）．なお，structured QR 分解の部分はスレッド並列化は行われていない．また，三角行列の通信は，$n \times n$ の二次元配列のままで行う実装とした．
- S-CholQR3：グラム行列の計算は，BLAS の `dgemm` または `dsyrk` を使う 2 種類の実装を用意した．ただし，グラム行列の計算の際の通信（`MPI_Allreduce`）は，上記の 2 種類のルーチンに関わらず，$n \times n$ の二次元配列として行う実装とした．また，三角行列同士の積を計算する部分は，片方の三角行列をゼロ埋めして，BLAS の `dtrmm` を用いた．アルゴリズム中で用いるシフト量は，記事 [11] の場合と同じものを採用した（$\|A\|_F$ に基づいたシフト量）．

4.2 性能評価環境・設定

今回の性能評価は，**表 3** に挙げた，4 種類のスーパーコンピュータシステムを用いて実施した．各システムの諸元や性能評価時の設定は表 3 に記載の通りである．基本的に，コンパイルオプションや実行時の設定は，各システムのマニュアル等で推奨されているものを用いた．なお，OFP と BDEC に関しては，東京大学情報基盤センターの講習会資料 [22], [23] を参考にした．

テスト行列の生成方法は，記事 [11] と同じで，ランダムに生成した直交行列を用いる手法を用いた[*5]．行列サイズは，$m = 16777216 (= 2^{24})$ として，$n = 16, 64, 256$ の 3 ケースで実験を行った．各アルゴリズムは，同一の条件で 5 回実行時間を測定（ただし，同一のジョブ内）し，最小値を評価対象とした．なお，HQR と S-CholQR3 に関しては，候補となるブロック幅や BLAS ルーチンを全て試して，その中の最小値を評価対象とした．

今回の評価では，行列サイズを固定し，ノード数を変えて計算時間を測定した（強スケーリング）．評価で使用したノード数は，OFP と BDEC では，$8, 16, 32, 64, 128, 256, 512, 1024, 2048$ である．また，OBCX と Grand では，$8, 16, 32, 64, 128, 256$ である．なお，表 3 に記載の通り，各システムでノード当たりのプロセス配置数が異なる．

[*5] $\kappa_2(A) = 10^{14}$ としたが，今回の性能評価の対象となるアルゴリズムの計算時間は，$\kappa_2(A)$ に依存しないので，特に意味はない．

4.3 評価結果：実行時間（強スケーリング）

各システムにおける各アルゴリズムの実行時間を**図 1** に示す．なお，図 1 の各グラフの横軸はノード数（プロセス数ではない）で，全てのグラフの縦軸・横軸のスケールは同じである．

まず，各アルゴリズムの実行時間に関して，図 1 より観察できることを以下に挙げる．

- OFP：$n = 16$ の場合，ノード数に関わらず，S-CholQR3 が最速である．$n = 64$ の場合，256 ノードまでは，S-CholQR3 と TSQR が同程度で最速（もしくは TSQR が若干高速）で，512 ノード以降は S-CholQR3 が高速である．$n = 256$ の場合，64 ノードまでは同様に S-CholQR3 と TSQR が同程度，それより多いノード数では S-CholQR3 が高速である．HQR と CGS2 は，最速なアルゴリズムと比べて，n やノード数が関わらず，概ね一桁以上実行時間が長い．
- BDEC: $n = 16$ の場合, 128 ノードまでは HQR と CGS2, それ以降は S-CholQR3 が最速である．$n = 64$ の場合，128 ノードまでは HQR，それ以降は S-CholQR3 が最速である．$n = 256$ の場合，S-CholQR3 が最速であるが，2048 ノードでは HQR と大差ない．TSQR が遅く，特に，$n = 16, 64$ の場合に顕著である．
- OBCX：ほぼ全ての条件（n およびノード数）で S-CholQR3 が最速で，他のアルゴリズムと有意な差がある．ただし，$n = 64$ でノード数が 64 以下の場合においては，TSQR とあまり差がない．
- Grand：概ね，OBCX と同じ傾向である．

次に，スケーラビリティ（ノード数を増やした際の速度向上）に関して，図 1 より観察できることを以下に挙げる．

- OFP：$n = 16$ の場合，HQR と CGS2 は 128 もしくは 256 ノードの時点で速度向上が限界となっている．一方，TSQR と S-CholQR3 は 512 もしくは 1024 ノードまで速度向上が得られている．$n = 64$ の場合, 256 ノード以降で，TSQR と S-CholQR3 の速度向上が異なることが確認できる．$n = 256$ の場合，同様に，64 ノード

表 3 性能評価で用いる計算機システムの諸元と性能評価における設定.

本稿での呼称	OFP	BDEC	OBCX	Grand
システム名	Oakforest-PACS	Wisteria/BDEC-01 (Odyssey)	Oakbridge-CX	Grand Chariot
運用組織	JCAHPC	東京大学 情報基盤センター	東京大学 情報基盤センター	北海道大学 情報基盤センター
総ノード数	8,208	7,680	1,368	1,004
CPU	Intel Xeon Phi 7250 (KNL)	Fujitsu A64FX	Intel Xeon Platinum 8280 (Cascade Lake)	Intel Xeon Gold 6148 (Skylake)
周波数	1.4 GHz	2.2 GHz	2.7 GHz	2.4 GHz
コア数	68	48 （アシスタントコア 2 or 4）	28	20
CPU 数 / ノード	1	1	2	2
メモリ容量 / ノード	96 GiB （MCDRAM: 16 GiB）	32 GiB	128 GiB	384 GiB
理論演算性能 / ノード	3.046 TFLOPS	3.379 TFLOPS	4.838 TFLOPS	3.072 TFLOPS
インターコネクト	Intel Omni-Path	Tofu インターコネクト D	Intel Omni-Path	Intel Omni-Path （2 ポート / ノード）
ネットワークトポロジー	Full-bisection Fat Tree	6 次元メッシュ / トーラス	Full-bisection Fat Tree	Full-bisection Fat Tree
コンパイラ	Intel `mpiifort` （ver. 19.0.5.281）	Fujitsu `mpifrtpx` （ver. 4.7.0）	Intel `mpiifort` （ver. 19.1.3.304）	Intel `mpiifort` （ver. 19.1.3.304）
BLAS/LAPACK	Intel MKL （ver. 2019.0.5）	Fujitsu BLAS/LAPACK （ver. 1.2.34）	Intel MKL （ver. 2020.0.4）	Intel MKL （ver. 2020.0.4）
コンパイルオプション	`-mkl=parallel -O3 -ipo -qopenmp -align array64byte` -xMIC-AVX512	`-SSL2BLAMP -Kfast -Kopenmp`	`-mkl=parallel -O3 -qopenmp -axCORE-AVX512`	`-mkl=parallel -O3 -qopenmp -xCORE-AVX512`
MPI プロセス数 / ノード	1	4	2	2
スレッド数 / プロセス	64 （コア 0 と 1 を除く）	12	28	20
補足事項	MCDRAM のみ使用 （Flat モード）	libomp を使用 （デフォルト設定のまま）		

以降で，TSQR と S-CholQR3 の速度向上の差が確認できる．

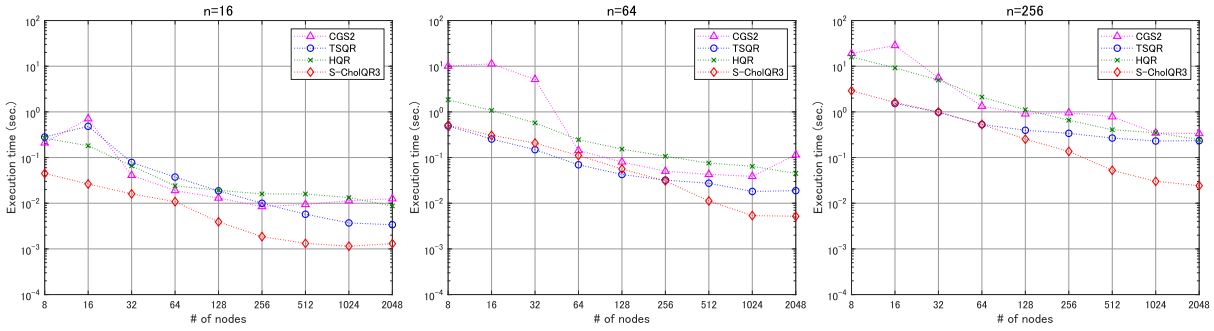
- BDEC：$n = 16$ の場合，HQR と CGS2 は 128 ノード以降で速度向上が停滞しているが，TSQR と S-CholQR3 は速度向上が続いている．$n = 64$ の場合も，同様に，256 ノード以降で，HQR，CGS2 と TSQR，S-CholQR3 の間で，速度向上の差を確認できる．$n = 256$ の場合，256 ノードで TSQR の速度向上が止まっている．HQR と CGS2 については，512 もしくは 1024 ノードで停滞している．一方，S-CholQR3 は，512 ノード以降，速度向上が負（実行時間が増加）となっている．
- OBCX：$n = 16$ の CGS2 を除いて，どのアルゴリズムも，概ね，似たような速度向上が確認できる．特に，$n = 256$ の場合は，4 種類のアルゴリズムの速度向上（グラフの傾き）がほぼ同じである．
- Grand：概ね，OBCX と同じ傾向である．

最後に，上述の観察内容を踏まえて，図 1 の結果で特に興味深い点を挙げる．

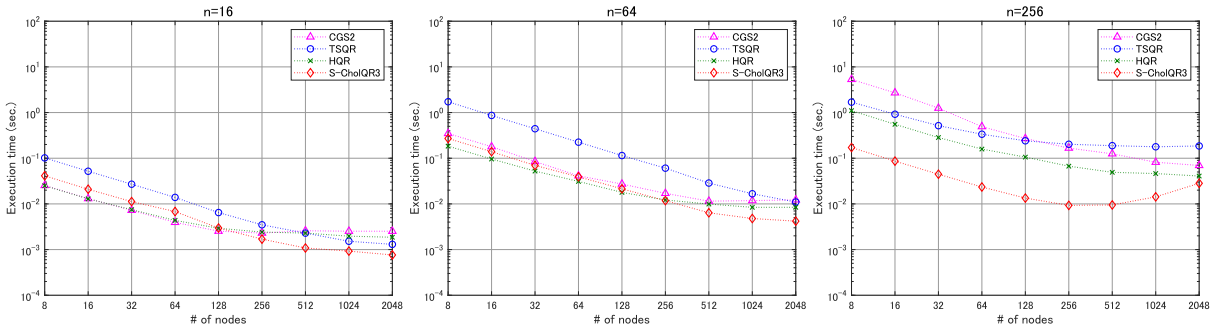
- BDEC の結果は，他の 3 システムと傾向が異なる．具体的には，TSQR が遅い，HSQR と CGS2 が速い，ノード数が増加した際に S-CholQR3 の実行時間が増加する，という点である．また，$n = 16, 64$ と $n = 256$ で，S-CholQR3 と HQR（と CGS2）との実行時間の差が大きく異なる，という点も興味深い．
- OBCX および Grand において，HQR と CGS2 の優劣が n によって異なる．$n = 16$ では CGS2 が高速，$n = 64$ では同程度，$n = 256$ では HQR が高速，となっている．

4.4 評価結果：実行時間の内訳

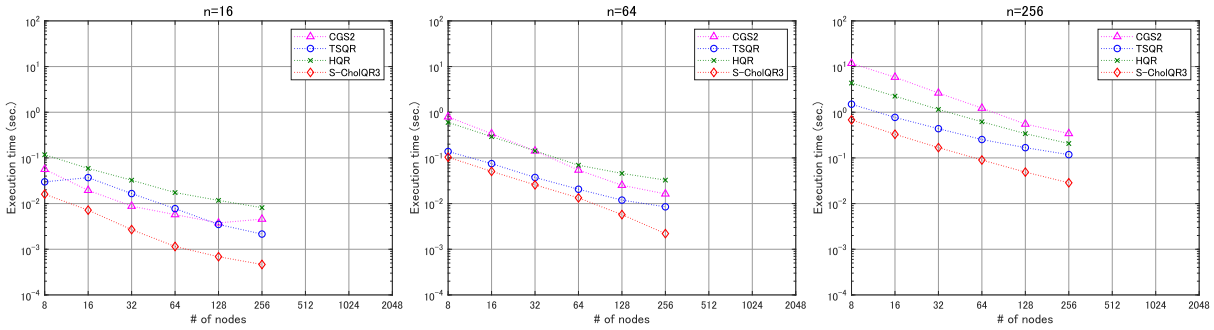
前節で示した結果を分析するために，各アルゴリズムの実行時間の内訳を調査する．**図 2** に 256 ノード使用時の各アルゴリズムの実行時間の内訳を示す．また，OFP と



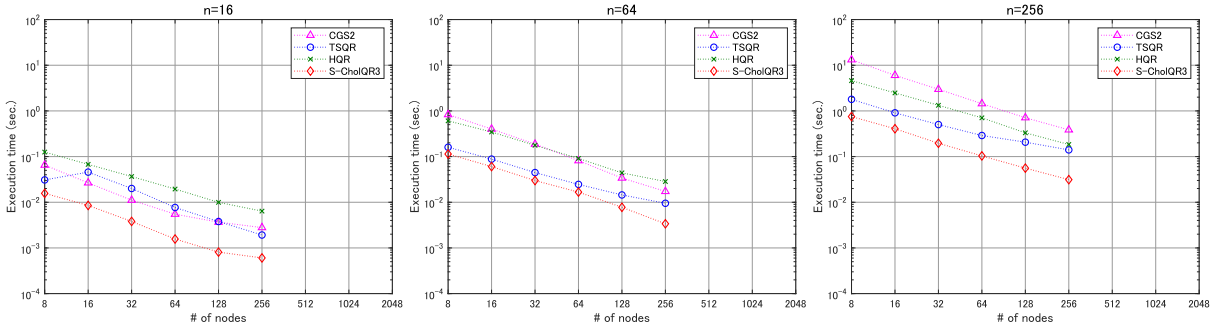
(a) Oakforest-PACS



(b) Wisteria/BDEC-01 (Odyssey)



(c) Oakbridge-CX



(d) Grand Chariot

図 1　各計算機システムにおける，各アルゴリズムの実行時間（強スケーリング）：$m = 16777216$.

BDEC については，2048 ノード使用時の内訳を**図 3** に示す．これらのグラフで，“comm.” は通信時間（MPI 関数の実行時間），“st-qr” は TSQR 内の structured QR 分解の実行時間，“other” は全体の実行時間から comm. と st-qr を引いた時間（主に演算時間に相当），である．なお，通信時間の測定では，MPI 関数ごとにはバリア同期を設定しておらず，示しているグラフの値はランク 0 で測定した値である．また，図 2 および図 3 のグラフの縦軸のスケールは統一されていないので，注意されたい．

まず，256 ノードの結果（図 2）に関する考察を述べる．

- 全体的に演算（other）の部分が多いので，256 ノードの時点では十分なスケーラビリティが得られているの

は自然な結果である．

- 通信回避型と非通信回避型で，通信時間に有意な差があることが確認できる．
- S-CholQR3（あるいは TSQR）が高速なのは，演算時間と通信時間の両方が少ないことが理由である．
- BDEC では，$n = 16, 64$ の場合に，S-CholQR3 と HQR，CGS2 の演算時間が同程度で，他の 3 システムと傾向が異なる．
- 論文 [17] の指摘の通り，$n = 256$ の場合，TSQR の st-qr のコストを無視できなくなっている．また，BDEC において特に顕著であるが，これは論文 [21] における structured QR 分解の実装の検討で，A64FX を対象としていなかったことが一因であると考えられる（今回の実装が必ずしも適当とは言えない）．
- BDEC の $n = 16, 64$ の場合に，TSQR の演算時間が大きく，他の 3 システムと異なる傾向である．これは，他の 3 システムでは LAPACK の `dgeqr` 関数を利用しているが，BDEC では `dgeqrf` を使っており，縦長行列の QR 分解のスレッド並列計算において，前者の方が効率がよい場合が多いことが一因であると思われる．
- OBCX および Grand で，HQR と CGS2 の演算時間と通信時間の比率が n によって異なっており，これが，前節の最後で挙げた挙動の原因である．演算時間については，HQR がブロック化を採用しているのに対して，CGS2 がブロック化を採用していないので，n が大きくなって，Level-3 BLAS の効率が上がったことで，HQR の演算時間が相対的に少なくなったと思われる．
  一方，通信時間に関しては，現状，理由は不明である．

次に，2048 ノード時の結果（図 3）に関する考察を述べる．

- 256 ノードの結果と比べると，通信時間の割合が増加しており，スケーラビリティが停滞しているのは自然な結果である．
- BDEC の $n = 256$ の場合で，S-CholQR3 の実行時間のほぼ全てが通信時間となっている．そのため，ノード数の増加とともに，通信時間のみが増加して，その結果として，全体の実行時間も増加したと思われる．$n = 16, 64$ の場合と比べて，演算時間の割合が極端に小さい原因は，現状，不明である．

5. おわりに

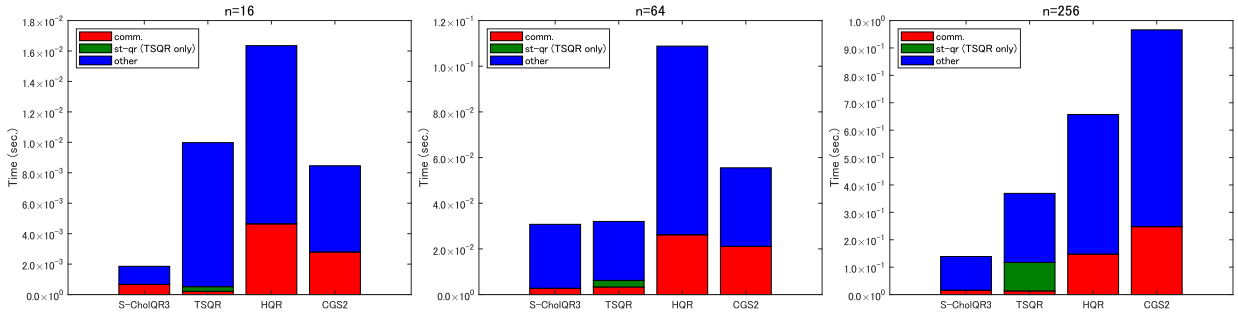
本稿では，縦長行列の QR 分解に対して，4 種類の異なるアルゴリズムの実行時間（強スケーリング）を，4 つのスーパーコンピュータシステム上で評価した結果を報告した．アルゴリズムが通信回避型かどうかで，スケーラビリティ（ノード数の増加による速度向上）の挙動が異なり，さらに，演算部分の特徴（例：Level-3 BLAS の利用効率）の影響も加味されて，アルゴリズムの実行時間の差が生じている．今回の評価の範囲内では，概ね，この両者で優れた特徴を持つ S-CholQR3 の有効性が目立った．また，よく似た構成のシステム（OBCX と Grand）では同じような傾向が確認された一方で，大きく特徴が異なる BDEC では，他のシステムと異なる傾向が多数確認された．

今後の課題・展望としては，まず，MPI 関数や BLAS/LAPACK のベンチマーク結果と関連させて，今回の結果の分析を深めることが挙げられる．さらに，その先の展望として，各アルゴリズムの性能モデル・性能予測に関する研究も考えられる．また，今回の評価では，行列の行数（m）は固定していたので，これを変化させて実験を行うことも必要である．加えて．最近は，低精度演算の活用が注目されているので，単精度等の場合における評価も興味深い．さらに，複素行列の QR 分解に対する評価も行う価値がある．その他として，プロセス数やスレッド数の設定も実行時間に影響を与えることが多々あるので，その調査も有益であると思われる．

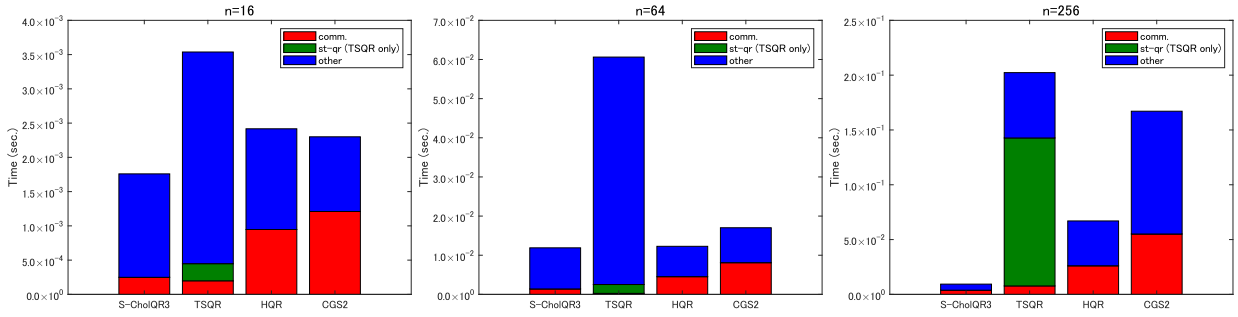
謝辞 本研究は JSPS 科研費（課題番号：JP21K11909）の助成を受けたものです．また，学際大規模情報基盤共同利用・共同研究拠点，および，革新的ハイパフォーマンス・コンピューティング・インフラの支援を受けています（課題番号：jh210044-NAH）．

参考文献

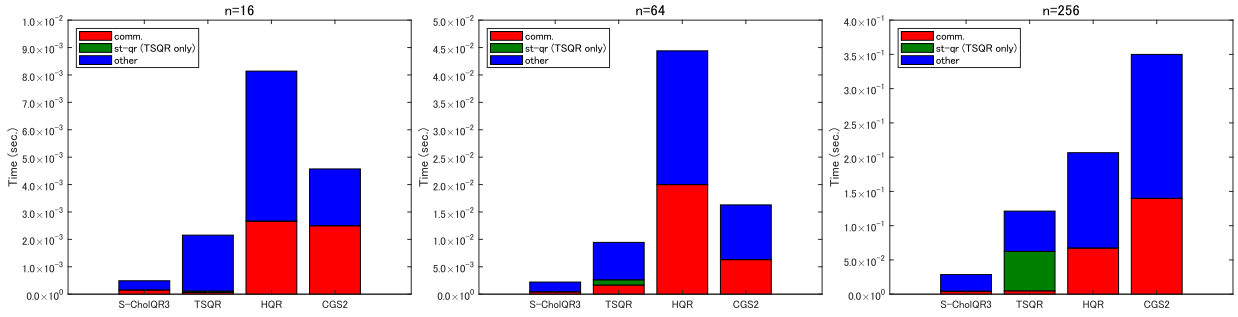
[1] Åke Björck: *Numerical Methods for Least Squares Problems*, SIAM (1996).
[2] Golub, G. and Van Loan, C.: *Matrix Computations*, Johns Hopkins University Press, 4th edition (2013).
[3] 櫻井鉄也，松尾宇泰，片桐孝洋：数値線形代数の数理と HPC，共立出版 (2018).
[4] 中村祥大，廣田悠輔：分散メモリ型並列計算機における TSQR アルゴリズムを用いた帯行列化アルゴリズムの性能分析，情報処理学会研究報告：ハイパフォーマンスコンピューティング（HPC），Vol. 2022-HPC-183, No. 23, pp. 1–9 (2022).
[5] 杉原正顯，室田一雄：線形計算の数理，岩波書店 (2009).
[6] Ballard, G., Demmel, J., Holtz, O. and Schwartz, O.: Minimizing Communication in Numerical Linear Algebra, *SIAM Journal on Matrix Analysis and Applications*, Vol. 32, No. 3, pp. 866–901 (2011).
[7] Demmel, J., Grigori, L., Hoemmen, M. and Langou, J.: Communication-optimal Parallel and Sequential QR and LU Factorizations, *SIAM Journal on Scientific Computing*, Vol. 34, No. 1, pp. A206–A239 (2012).
[8] Fukaya, T., Nakatsukasa, Y., Yanagisawa, Y. and Yamamoto, Y.: CholeskyQR2: A Simple and Communication-Avoiding Algorithm for Computing a Tall-Skinny QR Factorization on a Large-Scale Parallel System, *Proceedings of the 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ScalA '14, pp. 31–38 (2014).
[9] Yamamoto, Y., Nakatsukasa, Y., Yanagisawa, Y. and Fukaya, T.: Roundoff Error Analysis of the CholeskyQR2 Algorithm, *Electronic Transactions on Numerical Analysis*, Vol. 44, pp. 306–326 (2015).
[10] Fukaya, T., Kannan, R., Nakatsukasa, Y., Yamamoto, Y. and Yanagisawa, Y.: Shifted Cholesky QR for Computing the QR



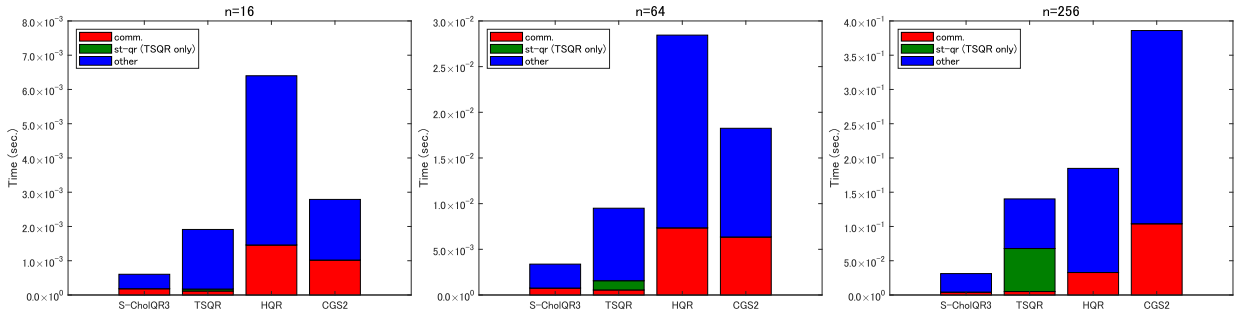
(a) Oakforest-PACS



(b) Wisteria/BDEC-01 (Odyssey)



(c) Oakbridge-CX



(d) Grand Chariot

図 2　各アルゴリズムの実行時間の内訳（256 ノード使用時）：$m = 16777216$.

Factorization of Ill-Conditioned Matrices, *SIAM Journal on Scientific Computing*, Vol. 42, No. 1, pp. A477–A503 (2020).

[11] 深谷　猛：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，スーパーコンピューティングニュース，Vol. 22, No. 6, 東京大学情報基盤センター，pp. 28–39 (2000).

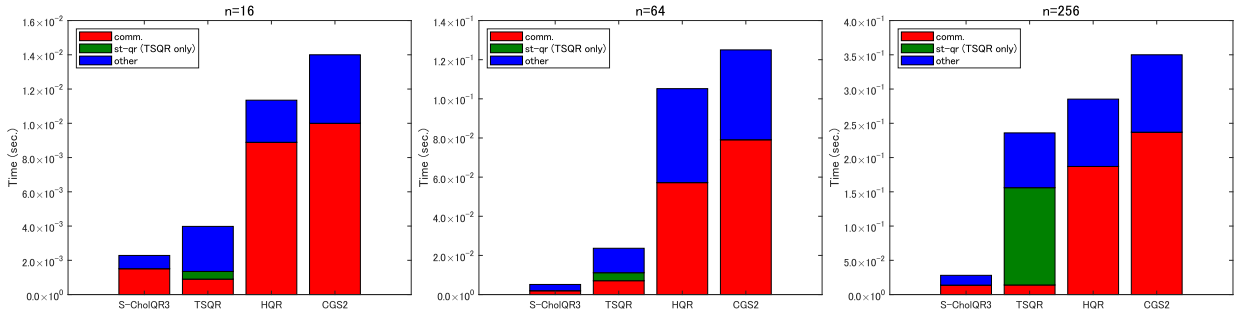
[12] Trefethen, L. N. and Bau, D.: *Numerical Linear Algebra*, SIAM (1997).

[13] Hadri, B., Ltaief, H., Agullo, E. and Dongarra, J.: Tile QR Factorization with Parallel Panel Processing for Multicore Architectures, *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pp. 1–10 (2010).

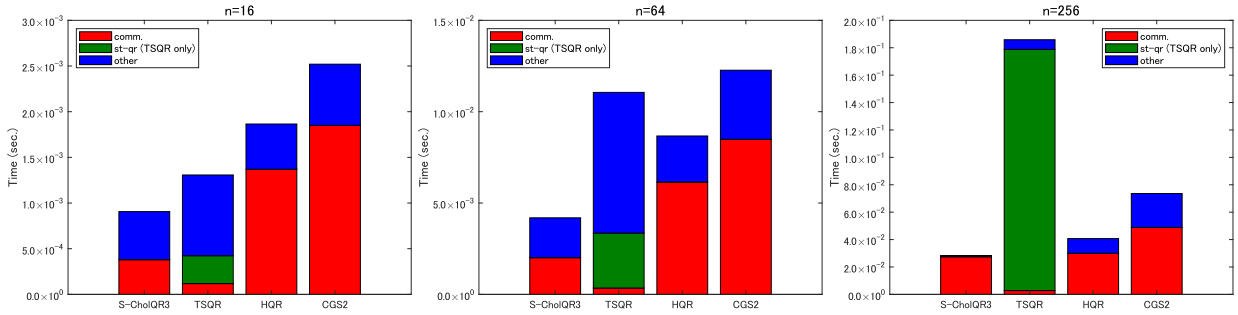
[14] Leon, S. J., Björck, r. and Gander, W.: Gram-Schmidt orthogonalization: 100 years and more, *Numerical Linear Algebra with Applications*, Vol. 20, No. 3, pp. 492–532 (2013).

[15] Yamazaki, I., Tomov, S. and Dongarra, J.: Mixed-Precision Cholesky QR Factorization and Its Case Studies on Multicore CPU with Multiple GPUs, *SIAM Journal on Scientific Computing*, Vol. 37, No. 3, pp. C307–C330 (2015).

[16] Terao, T., Ozaki, K. and Ogita, T.: LU-Cholesky QR Al-



(a) Oakforest-PACS



(b) Wisteria/BDEC-01 (Odyssey)

図 3 各アルゴリズムの実行時間の内訳（2048 ノード使用時）：$m = 16777216$.

gorithms for thin QR Decomposition, *Parallel Computing*, Vol. 92, p. 102571 (2020).

[17] Fukaya, T., Imamura, T. and Yamamoto, Y.: Performance Analysis of the Householder-Type Parallel Tall-Skinny QR Factorizations Toward Automatic Algorithm Selection, *High Performance Computing for Computational Science – VECPAR 2014*, Lecture Notes in Computer Science, pp. 269–283 (2015).

[18] Schreiber, R. and Van Loan, C.: A Storage-Efficient WY Representation for Products of Householder Transformations, *SIAM Journal on Scientific and Statistical Computing*, Vol. 10, No. 1, pp. 53–57 (1989).

[19] Carson, E., Lund, K., Rozložník, M. and Thomas, S.: Block Gram-Schmidt Algorithms and their Stability Properties, *Linear Algebra and its Applications*, Vol. 638, pp. 150–195 (2022).

[20] Mori, D., Yamamoto, Y. and Zhang, S.-L.: Backward Error Analysis of the AllReduce Algorithm for Householder QR Decomposition, *Japan Journal of Industrial and Applied Mathematics*, Vol. 29, No. 1, pp. 111–130 (2012).

[21] Fukaya, T.: An Investigation into the Impact of the Structured QR Kernel on the Overall Performance of the TSQR Algorithm, *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2019, pp. 81–90 (2019).

[22] 東京大学情報基盤センター：講習会資料「KNL 実践」, `https://www.cc.u-tokyo.ac.jp/events/lectures/112/20190213-2.pdf`.

[23] 東京大学情報基盤センター：講習会資料「Wisteria 実践」, `https://www.cc.u-tokyo.ac.jp/events/lectures/161/20210909-1.pdf`.