Vol.2022-HPC-183 No.23
2022/3/18

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

中村 祥大[1,a)]　廣田 悠輔[1,b)]

概要：実対称行列の帯行列化を行う Bischof のアルゴリズムでは，繰り返し行われる QR 分解がその主要な処理の一つとなる．分散メモリ型並列計算機においては，QR 分解を行うアルゴリズムによっては頻繁な集団通信が必要となる．このため高並列環境では QR 分解処理が帯行列化全体の性能を低下させる要因になることが知られている．帯行列化における QR 分解を適切に実装された TSQR アルゴリズムにより行うことで高性能を実現できるという予想がなされているが，既存の帯行列化プログラムには TSQR アルゴリズムを用いたものはなく，またそのようなプログラムを作成して性能評価を行ったという報告も我々の知る限り存在しない．我々は，TSQR アルゴリズムを QR 分解に用いる帯行列化プログラムを分散メモリ型計算機向けに実装し，その性能の評価および分析を行った．行列次数 $n = 16384$，プロセス数 64 として実験したところ，TSQR 版帯行列化プログラムは他の QR 分解アルゴリズムを用いたものに比べて約 8%高速となった．また，TSQR アルゴリズムルーチンの実行時間内訳の分析により，行列要素の再分配処理が性能ボトルネックとなっていることを確認した．

NAKAMURA Shota[1,a)]　HIROTA YUSUKE[1,b)]

1. はじめに

実対称密行列 A と正整数 k が与えられるときに，

$$U^\top AU = B \tag{1}$$

を満たす $n \times n$ 直交行列 U，半帯幅 k の $n \times n$ 実対称帯行列 B を求める行列操作を帯行列化という．本研究では，n が数千から数十万程度であり，k が数十から数百程度である場合の（すなわち $k \ll n$ である場合の）帯行列化について扱う．大規模密行列を扱うため，このような帯行列化を分散メモリ型並列計算機（以下，単に並列計算機）により行うことを考える．

実対称行列の帯行列化は，特に実対称密行列固有値問題の求解において重要な役割を果たす．実対称密行列固有値問題を解くには，与えられた密行列をより固有対の計算が容易な中間形行列に相似変換し，中間形行列の固有対を求め，その結果から元の固有ベクトルの固有対を求めるという手順がよく用いられる．古典的には実対称三重対角行列を中間形に用いる方法（[1], Sec. 8.3）が用いられてきたが，近年は中間形に実対称帯行列を用いる方法 [2], [3] も用いられる．中間形に実対称帯行列を用いる方法は，実対称三重対角行列を用いる方法と比べて，中間形の固有対を求める計算量が増大するものの，アルゴリズム・実装の工夫により密行列から中間形に相似変換する際の主記憶アクセスや通信のコストを軽減できることが知られている．近年の並列計算機においては，主記憶アクセスや通信にかかる時間が計算にかかる時間に対して相対的に長くなる傾向があるため，中間形に帯行列を用いる手法の開発の重要度が増している．したがって，実対称密行列の帯行列化の高速化技術の研究は重要である．

実対称行列の帯行列化には，Bischof の帯行列化アルゴリズム [4] を用いるのが一般的である．Bischof のアルゴリズムは，

(1) panel matrix と呼ばれる縦長ブロックを QR 分解し，
(2) QR 分解結果を利用して，行列積と対称 rank-2k 更新により trailing matrix と呼ばれるブロックを更新する

というステップにより k 行 k 列ずつ行列要素を消去する操作を n/k 回繰り返し，与えられた実対称行列を帯行列化する．この繰り返しにおける QR 分解にはどのようなアルゴ

[1] 福井大学
Bunkyo, Fukui-shi, Fukui 910-8507, Japan
[a)] hb180797@g.u-fukui.ac.jp
[b)] y-hirota@u-fukui.ac.jp

リズムを用いてもよく，例えば Householder QR アルゴリズム（[1], Sec. 8.3.1），TSQR アルゴリズム [5]，Cholesky QR アルゴリズム [6] などが利用できる．

並列計算機を用いて大規模行列計算を行う場合，その行列要素を各プロセスに分散して記憶させる．このため，行列計算の途中にプロセス間で計算の途中結果を通信してやり取りする必要がある．並列計算機においては，一般的に通信時間がプログラム全体の性能を低下させる主要な要因となる．特に，多数のプロセスが同時に通信を行う集団通信は，プロセス数が多い場合にセットアップ時間が長くなり，実際に通信するデータサイズが小さい場合であっても大きく性能を阻害する要因となりえることが知られている．したがって，高い並列度の並列計算機で高性能な行列計算プログラムを実現するには，単に各プロセスに均等にタスクを割り当てるといった並列化の基本を守るだけでなく，集団通信を削減するようにプログラムを構成する必要がある．

Bischof のアルゴリズムを用いる帯行列化プログラムを並列計算機で実行する場合には，QR 分解を行う際の集団通信が問題になりえる．Bischof のアルゴリズム中で QR 分解を行う一般的な方法として，Householder QR アルゴリズムがある．Householder QR アルゴリズムでは, QR 分解対象の行列の各列に対しハウスホルダ変換を順に適用することで QR 分解を行う．Householder QR アルゴリズムを標準的な方法で実装して並列計算機で動作させると，$\mathcal{O}$(分解対象行列の列数) 回の頻繁な集団通信を必要とする．そのため，高並列環境では Householder QR アルゴリズムによる集団通信が，Bischof のアルゴリズムの全体性能に対するボトルネックとなると考えられる．このような問題を回避するため，Bischof のアルゴリズム中の QR 分解を，TSQR アルゴリズムに置き換えることが考えられている．TSQR アルゴリズムは，Householder QR アルゴリズムと比べて計算量が多いものの，集団通信を $\mathcal{O}(1)$ 回のみ行う実装が可能である．このため，このような置き換えにより，高並列環境における帯行列化全体の実行時間を短縮できるという予想がなされている [7].

しかしながら我々の知る限り，現在，広く使われる並列計算機向け帯行列化プログラムは，いずれも Bischof の帯行列化アルゴリズムまたはその派生である Wu のアルゴリズムを採用しているが，その中の QR 分解には TSQR アルゴリズムは用いられていない．例えば，ドイツの研究チームにより開発された固有値計算ライブラリ ELPA [8] に含まれる ELPA2 ソルバの帯行列化では，Householder QR アルゴリズムが用いられている．また，理化学研究所の研究チームにより開発された EigenExa [9] に含まれるソルバ eigen_sx は，このソルバの五重対角化（すなわち $k = 2$ の帯行列化）の際に Cholesky QR アルゴリズムが用いている．加えて，Bischof のアルゴリズム内で TSQR アルゴリズムを用いた QR 分解を行う際，自然に考えられる実装法を用いた場合には QR 分解の前後に新たな前/後処理が発生となる．このような追加のオーバーヘッドを考慮し，実際の性能を議論した研究は報告されていない．

そこで本研究では，QR 分解に TSQR アルゴリズムを用いる Bischof のアルゴリズムの並列計算機における性能を調査する．より具体的には，QR 分解に Householder QR アルゴリズム，TSQR アルゴリズムを用いる Bischof のアルゴリズムに基づく帯行列化プログラムをそれぞれ作成し，その実行時間を実際に並列計算機上で測定する．また，その結果をもとに QR 分解アルゴリズムの違いによる性能特性について分析を行う．

本稿の構成は以下のとおりである．第 2 節では，Bischof の帯行列化アルゴリズムの概要について述べる．また，Bischof のアルゴリズム内で必要となる QR 分解アルゴリズムを行う方法として，Householder QR アルゴリズム，TSQR アルゴリズムの概要を述べる．第 3 節では，我々の実装する帯行列化プログラムについて述べる．第 4 節では，実装した帯行列化プログラムの性能評価を行い，その結果について議論する．最終節でまとめと今後の課題について述べる．

2. 帯行列化に用いられるアルゴリズム

実対称密行列の帯行列化を行うアルゴリズムにはさまざまな手法がある．例えば，Bischof のアルゴリズムや，その改良である Wu のアルゴリズムなどがある [10]．本章では，広く用いられる帯行列化アルゴリズムである Bischof のアルゴリズムと，その中で必要となる QR 分解を行うアルゴリズムについて概要を説明する．ここでは本研究で主に用いる，Householder QR アルゴリズムと TSQR アルゴリズムについて述べる．

2.1 Bischof のアルゴルリズム

Bischof のアルゴルリズムは，$n \times n$ 実対称密行列 A を半帯幅 k の帯行列 B に変換する．本アルゴリズムの疑似コードを Alg. 1 に示す．

Bischof のアルゴリズムの中心となる処理は，A の縦長ブロック M に対する QR 分解（Alg. 1，第 4 行）と，その結果を用いた A の更新（Alg. 1，第 7 行）の n/k 回の繰り返しであり，繰り返しごとに A は k 行 k 列ずつ要素が消去され，徐々に帯行列へ変換される．この繰り返しの中で行われる QR 分解には任意の QR 分解アルゴリズムを利用可能である．QR 分解は約 n/k 回行われ，そのアルゴリズムの性能が帯行列化全体の性能に影響を及ぼす．

2.2 Householder QR アルゴリズム

QR 分解対象の行列 M を $r \times c$ 行列とする（ただし $r \geq c$）．Householder QR アルゴリズムは Householder 変換により

Algorithm 1 Bischof のアルゴリズム

Input: $n \times n$ 実対称密行列 A，半帯幅 k
Output: $U^\top AU = B$ となる直交行列 U，半帯幅 k の帯行列 B

$U \leftarrow I$
for $i = 0, 1, \ldots, n/k - 2$ **do**
  $M \leftarrow A[(i+1)k+1 : n, ik+1 : (i+1)k+1]$
  $QR \leftarrow M$ と QR 分解する
  $Q_{part} \leftarrow \begin{bmatrix} I & O \\ O & Q \end{bmatrix}$
  $U \leftarrow Q_{part}^\top U$
  $A \leftarrow Q_{part}^\top A Q_{part}$
end for
$B \leftarrow A$
return U, B

Algorithm 2 Householder QR アルゴリズム

Input: $r \times c$ 行列 M $(r \geq c)$
Output: $M = QR$ となる $r \times r$ 直交行列 Q と $r \times c$ 上三角行列 R

$Q \leftarrow I$
for $i = 1, 2, \ldots, c-1$ **do**
  $\boldsymbol{v} \leftarrow \begin{bmatrix} M_{i,i} \\ \vdots \\ M_{r,i} \end{bmatrix}$
  $\boldsymbol{v}$ の第 2 要素以降を 0 にするようなハウスホルダ変換を行う行列 H_i を求める．
  $Q_i \leftarrow \begin{bmatrix} I & O \\ O & H_i \end{bmatrix}$
  $M \leftarrow Q_i M$
  $Q \leftarrow Q Q_i^\top$
end for
$R \leftarrow M$
return Q, R

行列 M の QR 分解を行う．アルゴリズムの疑似コードを Alg. 2 に示す．M の i 列目に対し，$i+1$ 行目以降の要素を 0 にするようなハウスホルダ変換を行う行列 H_i を作成し，H_i を M に左からかけることにより，M を上三角行列 R へ変換する．また，H_i は直交行列であるため，H_i の積として Q が構成される．

2.3 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通信回避型の QR 分解アルゴリズムである．疑似コードを Alg. 3 に示す．TSQR アルゴリズムでは，Alg 3 の 1 行目に示されるように行列 M を縦に二分割し，分割後の行列 M_1, M_2 に対する QR 分解を行う．その後，それらの分解結果を用いて M に対する QR 分解を行う．分割後の行列 Q_1, Q_2 に対する QR 分解は，どのような QR 分解アルゴリズムを用いても良いが，その行列が更に分割可能である限り TSQR アルゴリズムを再帰的に適用することが可能である．

Algorithm 3 TSQR アルゴリズム

Input: $r \times c$ 行列 M $(r \geq 2c)$
Output: $M = QR$ となる $r \times r$ 直交行列 Q と $r \times c$ 上三角行列 R

1: $\begin{bmatrix} M_1 \\ M_2 \end{bmatrix} \leftarrow M$ と縦方向に 2 分割する
2: $Q_1 R_1 \leftarrow M_1$ と QR 分解する
3: $Q_2 R_2 \leftarrow M_2$ と QR 分解する
4: $\tilde{Q}R \leftarrow \begin{bmatrix} R_1 \\ R_2 \end{bmatrix}$ と QR 分解する
5: $Q \leftarrow \begin{bmatrix} Q_1 & O \\ O & Q_2 \end{bmatrix} \tilde{Q}$
6: **return** Q, R

3. Bischof のアルゴリズムの分散メモリ型並列計算機向け実装

この節では我々が実装した異なる二つの QR 分解アルゴリズム（Householder QR アルゴリズム，TSQR アルゴリズム）を用いる帯行列化プログラムについて説明する．

基本的な実装方針として，我々はプログラムのすべての機能を自作するのではなく，プログラムの下位ルーチンとして ScaLAPACK とその依存ライブラリ（PBLAS, BLACS，LAPACK）の機能を用いる．プログラムの自作部分については C99 規格の C 言語を用いて実装する．

以下では，まず帯行列化アルゴリズム中で用いる QR 分解結果の格納形式である compact WY 表現 [11] について述べる．その後，Bischof のアルゴリズム，Householder QR アルゴリズムを用いる QR 分解ルーチン, TSQR アルゴリズムを用いる QR 分解ルーチンの実装について説明する．

3.1 Compact WY 表現

Bischof のアルゴリズム（Alg. 1）では，縦長行列 M に対する QR 分解 $M = QR$ の結果を必要とする．この分解結果の Q は帯行列化対象の行列のブロックの更新などに用いられる．この行列を密行列として構成・記憶し，その密行列をブロックの更新に用いることは実行時間，主記憶使用量の面から望ましくない．より少ないメモリ量で Q を記憶し，またブロック更新の実行時間を抑制するための方法として，compact WY 表現が提案されている．この表現では，Q を密行列として陽に記憶する代わりに，

$$Q = I - YTY^\top$$

となる $r \times c$ 下三角行列 Y，$c \times c$ 上三角行列 T を記憶する．これにより，Q の保存に必要な記憶領域を $\mathcal{O}(r^2)$ から $\mathcal{O}(rc)$ へ削減できる．また，Bischof のアルゴリズムによる A の更新処理が行列の rank-$2k$ 更新により行えるようになり，実行時間の面でも有利である．

Compact WY 表現の優位性を利用するため，実装する帯行列化プログラムでは QR 分解ルーチン出力の Q を compact WY 表現を用いて保存し，それを利用する．す

Algorithm 4 Bischof のアルゴリズムの実装

Input: $n \times n$ 実対称密行列 A, 半帯幅 k
Output: 帯行列化後の行列 B, 帯行列化に用いた行列 Y_i, T_i

for $i = 0, 1, \cdots, n/k - 2$ **do**
  $A_{part} \leftarrow A[(i+1)k+1 : n, ik+1 : (i+1)k+1]$
  $A_{update} \leftarrow A[(i+1)k+1 : n, (i+1)k+1 : n]$
  $(I - Y_i T_i Y_i^\top)R \leftarrow A_{part}$ // QR 分解（Compact-WY 表現）
  $C \leftarrow A_{update} Y_i T_i$ // PDGEMM, PDSYMM
  $D \leftarrow T_i^\top Y_i^\top C/2$ // PDGEMM
  $E \leftarrow C - Y_i D$ // PDGEMM
  $A[(i+1)k+1 : n, (i+1)k+1 : n] \leftarrow A_{update} - Y_i E^\top - E Y_i^\top$ // PDSYR2K
end for
$B \leftarrow A$
return B, Y_i, T_i

なわち，QR 分解ルーチンの入力として M を受け取り，$M = (I - YTY^\top)R$ となる上三角行列 R と Q の compact WY 表現 Y, T を出力する．

3.2 Bischof のアルゴリズムの実装

我々の帯行列化プログラムは，Bischof のアルゴリズムは山本（2005）[10] に記載のあるアルゴリズムを参考に，ScaLAPACK の機能を下位ルーチンとして用いて実装されている．行列の分散には 2D block-cyclic 形式を用いる．この実装では，入力として $n \times n$ 実対称密行列 A と半帯幅 k を受け取り，出力として半帯幅 k の帯行列 B と U_i, T_i を構成する．この U_i, T_i を用いることで (1) を満たす直交行列 U を構成できる．

本プログラムの動作を Alg. 4 に示す．このアルゴリズムでは，A の縦長部分 M に対する QR 分解が n/k 回発生する．この処理には後述する QR 分解ルーチンを用いられる．また，QR 分解結果の T_i, Y_i を用いて，A の左下部分 A_{update} を更新する際には，5 回の行列行列積と 1 回の rank-$2k$ 更新が必要となる．これらの処理には ScaLAPACK の行列積ルーチンである PDGEMM, PDSYMM と rank-$2k$ 更新ルーチンである PDSYR2K を用いる．

3.3 Householder QR 版 QR 分解ルーチン

Householder QR 版 QR 分解ルーチンの動作を Alg. 5 に示す．動作は大きく分けて 2 ステップからなる．第 1 ステップとして，M を Householder QR アルゴリズムによって QR 分解する．この処理は ScaLAPACK の QR 分解ルーチンである PDGEQRF を呼び出すことで実装した．このルーチンによって，QR 分解結果の R とハウスホルダリフレクタの組 $(\boldsymbol{v}_i, \tau_i)$ $(i = 1, 2, \ldots, c-1)$ が構成される．第 2 ステップとして，得られた $(\boldsymbol{v}_i, \tau_i)$ から Y, T を構成する．これには Schreiber らのアルゴリズムを参考に実装した．アルゴリズムを Alg. 6 に示す．このアルゴリズムでは，$2(c-2)$ 回の行列ベクトル積が発生する．

この QR 分解ルーチン全体で発生する集団通信の回数について述べる．ステップ 1 では，PDGEQRF の内部で $\mathcal{O}(c)$ 回の集団通信が発生する．ステップ 2（Alg. 6）では，1，2 行目の実行に定数回の集団通信が発生する．また，3 行目以降の for ループの中で，行列ベクトル積が $2(c-2)$ 回発生するが，それぞれの実行には定数回の集団通信を行う必要がある．従って，ステップ 2 では $\mathcal{O}(c)$ 回の集団通信が発生する．以上より，Householder QR 版 QR 分解ルーチンでは，合計で $\mathcal{O}(c)$ 回の集団通信が発生する．

Algorithm 5 Householder QR 版 QR 分解ルーチン

Input: $r \times c$ 行列 M $(r \geq c)$
Output: $M = (I - YTY^\top)R$ となる Q の compact WY 表現 Y, T と $r \times c$ 上三角行列 R

1: M を Householder QR アルゴリズムによって QR 分解し，R とハウスホルダリフレクタの組 $(\boldsymbol{v}_i, \tau_i)$ $(i = 1, \ldots, c-1)$ を得る
2: Schreiber らのアルゴリズムにより $(\boldsymbol{v}_i, \tau_i)$ から Y, T を構成（後処理）
3: **return** Y, T, R

Algorithm 6 ハウスホルダリフレクタから compact WY 表現を構成する手順

Input: Q のハウスホルダリフレクタ $(\boldsymbol{v}_i, \tau_i)$ $(i = 1, 2, \ldots, c-1)$
Output: Q の compact WY 表現 Y, T

1: $Y = \boldsymbol{v}_1$
2: $T = \begin{bmatrix} \tau_i \end{bmatrix}$
3: **for** $k = 2, 3, \ldots, c-1$ **do**
4: $\quad \boldsymbol{z} \leftarrow -\tau_i T Y^\top \boldsymbol{v}_i$
5: $\quad Y \leftarrow \begin{bmatrix} Y & \boldsymbol{y}_i \end{bmatrix}$
6: $\quad T \leftarrow \begin{bmatrix} T & \boldsymbol{z} \\ \boldsymbol{0}^\top & \tau_i \end{bmatrix}$
7: **end for**
8: **return** Y, T

3.4 TSQR 版 QR 分解ルーチン

TSQR 版 QR 分解ルーチンは，Ballard（2013）[12] に記載のアルゴリズムを参考に実装されている．ルーチン全体の動作を Alg. 7 に示す．実装した QR 分解ルーチンの動作は大きく分けて 5 ステップからなる．以下，各ステップについて詳しく説明する．

第 1 ステップとして，与えられた M の保存形式を変更する．開始時点で M は親ルーチンで用いる 2D block-cyclic 形式で分散保存されている．しかしながら，以降のステップを効率よく動作させるためには M が 1D row block 形式で各プロセスへ格納されている必要がある．そのため，M を 2D block-cyclic 形式から 1D row block 形式へ再分配する．再分配は実装の簡潔さから，2 つの処理に分けて行った．各処理には 1 回の集団通信が発生する．はじめに，M を 2D block-cyclic 形式から 1D row block-cyclic 形式に変換する．この処理には，BLACS の行列コピールーチンである PDGEMR2D を用いた．その後，1D row block-cyclic 形式に再分配された M を 1D row block 形式へ再分配す

る．この処理には MPI ライブラリを利用した自作ルーチンを用いた．

第 2 ステップとして，再分配後の M に対し TSQR アルゴリズムを適用する．アルゴリズムを Alg. 8 に示す．各プロセスがこのアルゴリズムを同時に実行する．第 3 節で述べた TSQR アルゴリズムは与えられた行列を再帰的に分割し処理を行うが，M はステップ 1 ですでに分割されているため，このアルゴリズムでは各 QR 分解結果を再帰的にマージしていく．また，第 3 節で示した Alg. 3 では，各小行列に対する QR 分解結果の Q_1, Q_2 を陽に構築しているが，実装上は使用メモリ量の観点からそれを避け，代わりにハウスホルダリフレクタによって各 Q_i を表現する．今回の実装では，Q_i を構成するハウスホルダリフレクタの組 $(\boldsymbol{v}_i, \tau_i)$ を 1 つの行列 W とベクトル $\boldsymbol{x}$ の形で管理する．この QR 分解には LAPACKE ルーチンの DGEQRF を用いた．アルゴリズムの実行後，プロセス 0 に M の QR 分解結果である $\tilde{R}$ が，各プロセスに $W_{i,j}, \boldsymbol{x}_{i,j}$ が格納される．このステップでは 1 回の集団通信が発生する．

第 3 ステップとして，第 2 ステップで求めた $W_{i,j}$ から Q_{thin} を構成する．ここで，Q_{thin} とは，M の QR 分解結果における Q の最初の c 列からなる行列である．この構成には Alg. 9 を用いる．この処理に 1 回の集団通信が発生する．

第 4 ステップとして，$\tilde{R}, Q_{thin}$ から Y, T, R を構成する．はじめに，Q_{thin} を modified LU 分解する．Modified LU 分解とは，

$$Q_{thin} - \begin{bmatrix} S \\ O \end{bmatrix} = LU$$

となるような，対角要素が ± 1 の対角行列 S，$r \times c$ 単位下三角行列 L，$c \times c$ 上三角行列 U を求める操作である．この処理には Alg. 10 を用いた．このアルゴリズム中の 4 行目，S, L_0, U を求める処理は，BLAS の rank-1 更新ルーチンである DGER ルーチンなどにより実装した．また，6 行目の L_i を求める処理には，三角行列に対する方程式を解く BLAS ルーチンである DTRSM を用いた．Modified LU 分解の実行後，求めた L は各プロセス内に 1D row block 形式で分散保存される．その後，得られた S, L, U を用いて $Y = L, T = -USY_1^\top, R = S\tilde{R}$ とする．ここで，Y_1 は Y の $(1,1)$ 要素を左上とする $c \times c$ 正方行列である．第 4 ステップには合計で 1 回の集団通信が発生する．

第 4 ステップ終了時点では，構成した T, R は全要素がプロセス 0 に，Y は 1D row block 形式で各プロセスに分配されている．第 5 ステップとして，これらの行列を 2D block-cyclic 形式で再分配する．T の再分配は PDGEMR2D を 1 回用いることで実現した．また，Y, R の再分配は，ステップ 1 と逆に，始めに Y, R を自作のルーチンを用いて 1D row block-cyclic 形式に変換し，その後に PDGEMR2D を用いて 2D block-cyclic 形式へ変換することで実現した．ステップ 5 には合計で 3 回の集団通信が発生する．

各ステップでの集団通信は定数回のみであり，TSQR 版 QR 分解ルーチン全体で $\mathcal{O}(1)$ 回の集団通信が発生する．

なお，我々の帯行列化ルーチン（Alg. 4）では，帯行列化が進むにつれ行数が列数に対して十分大きくない行列の QR 分解を行う必要がある．ところが，本副節で説明した TSQR 版 QR 分解ルーチンの実装では，p プロセスで $r \times c$ 行列 M の QR 分解を行う際，$cp \leq r$ が満たされている必要がある．このため，TSQR アルゴリズムを QR 分解に用いる帯行列化ルーチンであっても，$cp \leq r$ が満たされていない場合には Householder QR 版 QR 分解ルーチンを用いている．このような状況は少数かつ次数の小さい行列の QR 分解でのみ発生するため，全体の実行時間に与える影響は僅かである．

Algorithm 7 TSQR 版 QR 分解ルーチン

Input: $r \times c$ 行列 M $(r \geq c)$
Output: $M = (I - YTY^\top)R$ となる Q の compact WY 表現 Y, T と $r \times c$ 上三角行列 R

M の要素を再分配（前処理）
M を TSQR によって QR 分解し，$\tilde{R}, W_{i,j}, \boldsymbol{x}_{i,j}$ を生成（TSQR 版 QR 分解）
$\tilde{R}, W_{i,j}$ から Q_{thin} を構成
Q_{thin} を modified LU 分解し，その結果を用いて Y, T, R を構成
Y, T, R を再分配（後処理）
return Y, T, R

Algorithm 8 TSQR アルゴリズムの実装

Input: 自身のプロセス番号 i, プロセス i が持つ M の部分行列 M_i
Output: $W_{i,j}, \boldsymbol{x}_{i,j}, \tilde{R}$

$W_{i,0}, \boldsymbol{x}_{i,0}, R \leftarrow M_i$ と QR 分解する
for $k = 1, 2, \ldots, \lceil \log p \rceil$ **do**
  if $i \equiv 0 \mod 2^k$ and $i + 2^{k-1} < p$ **then**
    $j \leftarrow i + 2^{k-1}$
    プロセス j から $\tilde{R}_i$ を受け取る
    $W_{i,k}, \boldsymbol{x}_{i,k}, \tilde{R}_i \leftarrow \begin{bmatrix} \tilde{R}_i \\ \tilde{R}_j \end{bmatrix}$ と QR 分解する
  else if $i \equiv 2^{k-1} \mod 2^k$ **then**
    プロセス $i - 2^{k-1}$ へ $\tilde{R}_i$ を送信する
  end if
end for
if $i = 0$ **then**
  $\tilde{R} \leftarrow \tilde{R}_i$
end if
return $W_{i,j}, \boldsymbol{x}_{i,j}, \tilde{R}$

4. 数値実験

本節では実装した帯行列化プログラムを並列計算機で実行し，その性能を評価・分析する．

4.1 実験条件

QR 分解に Householder QR アルゴリズムを用いる帯行

Algorithm 9 Q_{thin} の構成手順

Input: $W_{i,j}, \boldsymbol{x}_{i,j}$, プロセス数 p, 自身のプロセス番号 i
Output: Q_{thin} の部分行列 Q_i

if $i = 0$ **then**
  $\tilde{Q}_0 \leftarrow I$
end if
for $k = \lceil \log p \rceil, \lceil \log p \rceil - 1, \ldots, 1$ **do**
  if $i \equiv 0 \mod 2^k$ and $i + 2^{k-1} < p$ **then**
    $j \leftarrow i + 2^{k-1}$
    $\begin{bmatrix} \tilde{Q}_i \\ O \end{bmatrix}$ に $(W_{i,k}, \boldsymbol{x}_{i,k})$ が表現する Q を左から掛け，その結果を $\begin{bmatrix} \tilde{Q}_i \\ \tilde{Q}_j \end{bmatrix}$ と分割する
    プロセス j へ $\tilde{Q}_j$ を送信する
  else if $i \equiv 2^{k-1} \mod 2^k$ **then**
    プロセス $i - 2^{k-1}$ から $\tilde{Q}_i$ を受け取る
  end if
end for
$\begin{bmatrix} \tilde{Q}_i \\ O \end{bmatrix}$ に $(W_{i,0}, \boldsymbol{x}_{i,0})$ が表現する Q を左から掛け，その結果を Q_i に格納する
return Q_i

Algorithm 10 Modified LU 分解

Input: プロセス番号 i，$r \times c$ 行列 Q_{thin} の部分行列 Q_i
Output: $Q_{thin} - \begin{bmatrix} S \\ O \end{bmatrix} = LU$ となるような，対角要素が ± 1 の対角行列 S，$r \times c$ 単位下三角行列 L の部分行列 L_i, $c \times c$ 上三角行列 U

if $i = 0$ **then**
  $Q_0 - S = L_0 U$ となるような S, L_0, U を求める
  U を他の全プロセスへブロードキャスト
else
  プロセス 0 から U を受け取る
  $L_i \leftarrow Q_i U^{-1}$
end if
return S, L_i, U

列化ルーチン（Householder QR 版ルーチン）および TSQR 分解アルゴリズムを用いるルーチン（TSQR 版ルーチン）をそれぞれ，以下に示す複数の行列サイズとプロセス数の組み合わせに対して実行し，その実行時間を測定する．

- 行列 A の次数 $n = 16384, 32768$.
- プロセス数 $p = 1, 2, 4, 8, 16, 32, 64$.

行列 A の半帯幅 k は 32 とし，要素は乱数を用いて生成する．二次元ブロックサイクリック方式のブロックサイズ b は 60 とする．

本実験はすべて表 1 に示す並列計算機（北海道大学 Grand Chariot）を用いて行う．プロセスは 1 ノードあたり 2 プロセスを割当て，各プロセス内では 20 スレッドを使用するように設定する．

4.2 帯行列化全体の実行時間およびその内訳

二種類の帯行列化プログラムの実行時間全体とその内訳を**図 1**，**図 2** に示す．図中の各項目の意味は以下のとおりである．

- total：帯行列化ルーチン全体の実行時間.
- QR decomp：QR 分解ルーチンの実行時間.
- update：QR 分解の結果を用いた，A の更新処理の実行時間.

いずれの行列サイズでも，実行時間全体に占める QR 分解ルーチンの実行時間の割合が，プロセス数の増加に伴って増大していることがわかる．ここから，実行プロセス数が多い場合には，QR 分解にかかる時間が全体に対し無視できない時間を占めており，QR 分解の高速化が帯行列化全体の高速化に寄与すると言える．また，いずれの行列サイズでも，プロセス数の増大に対する QR 分解の高速化率は TSQR 版が Householder QR 版よりも高く，プロセス数が大きい場合には TSQR アルゴリズムによる QR 分解ルーチンがより高速になることが確認できた．一方で，行列の更新処理は両プログラムで同じ実装になっており，実行時間の差はほとんどない．結果，Householder QR 版ルーチンに対する TSQR 版ルーチンの実行時間全体の高速化率は，$n = 16384$ の場合は約 8%，$n = 32768$ の場合は約 7% と 10%未満である．そのため，TSQR 版ルーチンは Householder QR 版よりも高速に動作するものの，本実験で行った 64 プロセス以下の実行では，その差は大きくないと言える．

4.3 QR 分解に関連する実行時間の内訳

QR 分解ルーチンの実行時間内訳を**図 3**，**図 4** に示す．図中の各項目の意味は以下のとおりである．

- total：QR 分解ルーチン全体の実行時間（図 3，図 4 の QR decomp と同一）.
- QR comp：QR 分解処理そのものの実行時間．Householder QR 版ルーチンでは Householder QR アルゴリズムによる QR 分解，TSQR 版ルーチンでは TSQR アルゴリズムと modified LU 分解による Y, T, R の構成にかかる時間である.
- pre/post comp：前/後処理の実行時間．Householder QR 版ルーチンでは Schreiber らのアルゴリズムによる Y, T の構成，TSQR 版ルーチンでは入力行列の再分配，出力行列の再分配である.

実行プロセス数が 4 以上となるすべての実行条件において，前/後処理にかかる時間が QR 分解本体の実行時間よりも長くなった．また，TSQR 版ルーチンは並列度を高めるにつれ各項目の実行時間が Householder QR 版ルーチンよりも高速になった．しかしながら，これらの条件下では 16 プロセス以上での実行時に，TSQR 版ルーチンの各項目の高速化率に鈍化が見られた．以上の結果から，TSQR 版 QR 分解ルーチンはプロセス数が多い条件下では Householder QR 版 QR 分解ルーチンより高速に動作するものの，前/後処理の実行時間が依然として大きなボトルネックとなっ

表 1　実験に用いる並列計算機の各ノードの構成

CPU	Intel Xeon Gold 6148（20 コア，40 スレッド，1.536 TFLOPS）× 2 ソケット
主記憶	DDR4，192 GB × 2
コンパイラ	mpiicc (Intel C++ Compiler (icc) 19.1.3.304)
コンパイルオプション	`-std=c99 -O3 -xCORE-AVX512 -lmkl_scalapack_lp64 -lmkl_blacs_intelmpi_lp64 -mkl`

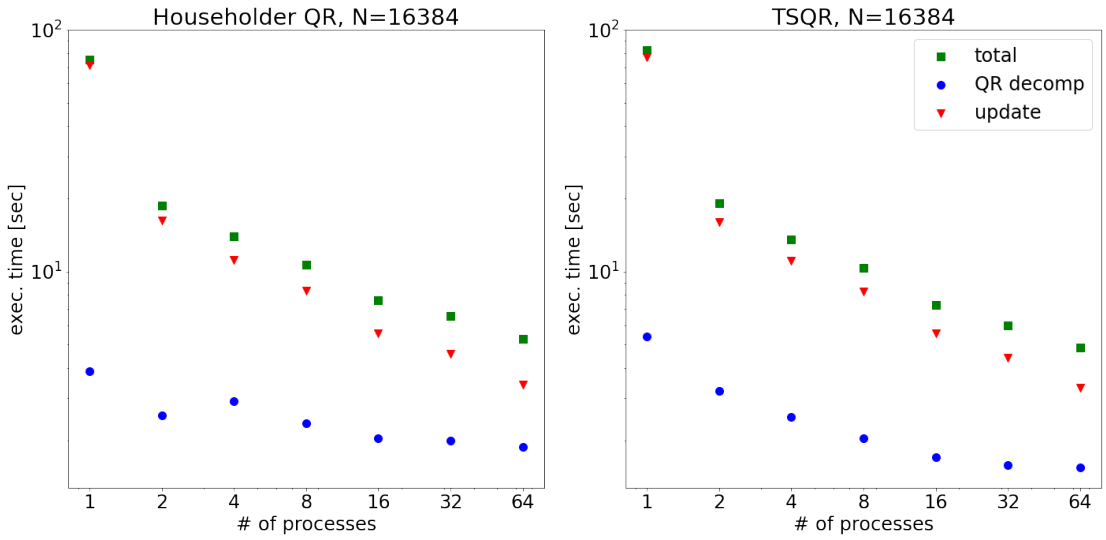

図 1　Bischof のアルゴリズム全体の実行時間とその内訳（$n = 16384$）

ていることがわかる．

5. まとめと今後の課題

本研究では異なる二つの QR 分解アルゴリズム（Householder QR アルゴリズム，TSQR アルゴリズム）を用いる帯行列化プログラムをそれぞれ実装し，分散メモリ型並列計算機の最大 32 ノード（64 プロセス）を用いて性能分析を行った．64 プロセス実行時には，TSQR アルゴリズムを用いる帯行列化プログラムは Householder QR アルゴリズムを用いるものと比べ約最大 10%高速となった．実行時間プロファイリングにより，TSQR アルゴリズムによる QR 分解の前処理および後処理が，プログラムの性能ボトルネックの一つとなっていることが確認された．

今後の課題としては，TSQR アルゴリズムによる QR 分解の前処理および後処理にかかる時間を低減するため，Bischof のアルゴリズム中で TSQR アルゴリズムを使用するのに適したデータ分散方式の検討が挙げられる．

謝辞　本研究の遂行にあたり，理化学研究所の今村俊幸チームリーダーに多くの有意義なご助言を頂きました．ここに感謝申し上げます．

本研究成果の一部は，北海道大学情報基盤センターのスーパーコンピュータを利用して得られた．本研究は JSPS 科研費 JP19H04127 の助成を受けている．

参考文献

[1] Golub, G. H. and Van Loan, C. F.: *Matrix Computations Fourth Edition*, Johns Hopkins University Press (2013).

[2] Bischof, C. H., Lang, B. and Sun, X.: Algorithm 807: The SBR Toolbox — Software for Successive Band Reduction, *ACM Transactions on Mathematical Software (TOMS)*, Vol. 26, No. 4, pp. 602–616 (2000).

[3] Bischof, C. H., Lang, B. and Sun, X.: A Framework for Symmetric Band Reduction, *ACM Transactions on Mathematical Software (TOMS)*, Vol. 26, No. 4, pp. 581–601 (2000).

[4] Bischof, C., Sun, X. and Lang, B.: Parallel tridiagonalization through two-step band reduction, *Proceedings of IEEE Scalable High Performance Computing Conference*, pp. 23–27 (1994).

[5] 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).

[6] 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，技術報告 6，東京大学情報基盤センター スーパーコンピューティングニュース (2020).

[7] Ballard, G.: Algorithmic Improvements for Dense Symmetric Tridiagonalization (talk material) (2015).

[8] Marek, A., Blum, V., Johanni, R., Havu, V., Lang, B., Auckenthaler, T., Heinecke, A., Bungartz, H.-J. and Lederer, H.: The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science, *Journal of Physics: Condensed Matter*, Vol. 26, No. 21 (2014).

[9] EigenExa (online): `https://www.r-ccs.riken.jp/labs/lpnctrt/projects/eigenexa/` (2022.02.16 最終閲覧).

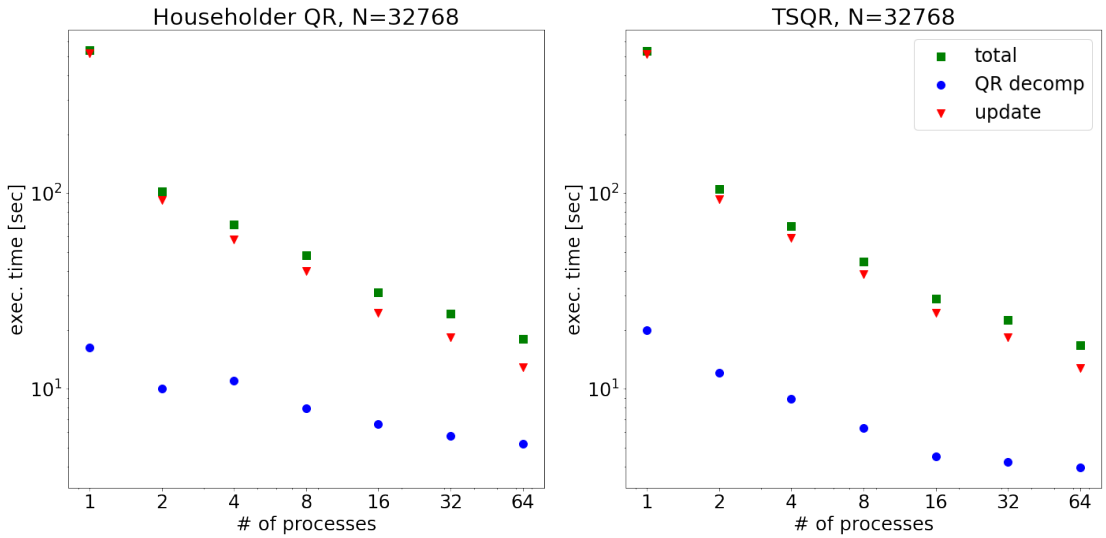

図 2　Bischof のアルゴリズム全体の実行時間とその内訳（$n = 32768$）

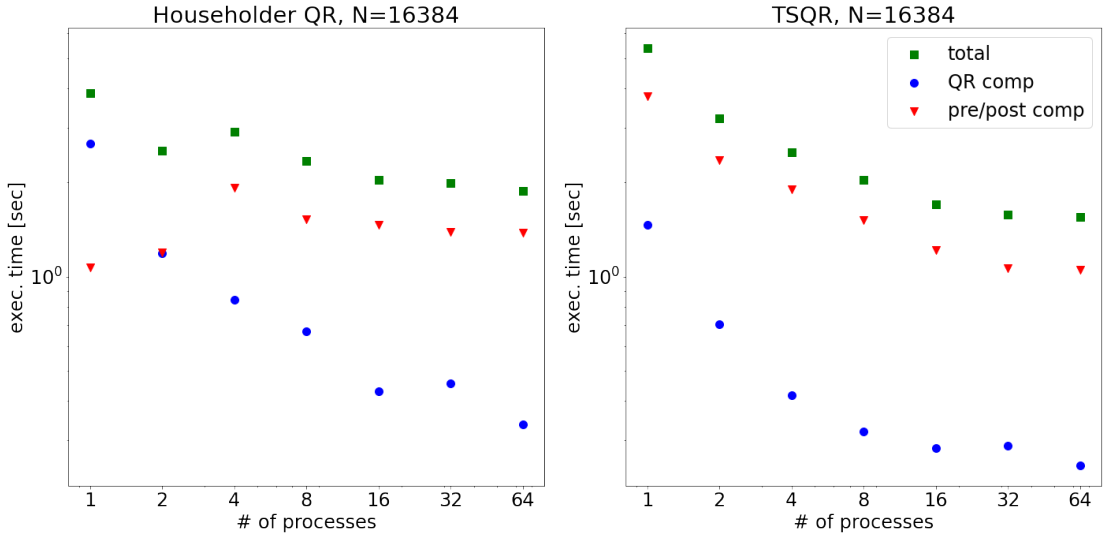

図 3　QR 分解ルーチンの実行時間内訳（$n = 16384$）

[10] 有作山本：キャッシュマシン向け対称密行列固有値解法の性能・精度評価，情報処理学会論文誌コンピューティングシステム（ACS），Vol. 46, No. SIG3(ACS8), pp. 81–91 (2005).
[11] Schreiber, R. and Van Loan, C.: A storage efficient WY representation for products of Householder transformations (1987).
[12] Ballard, G., Demmel, J., Grigori, L., Jacquelin, M., Nguyen, H. D. and Solomonik, E.: Reconstructing Householder Vectors from Tall-Skinny QR, Technical Report UCB/EECS-2013-175, EECS Department, University of California, Berkeley (2013).

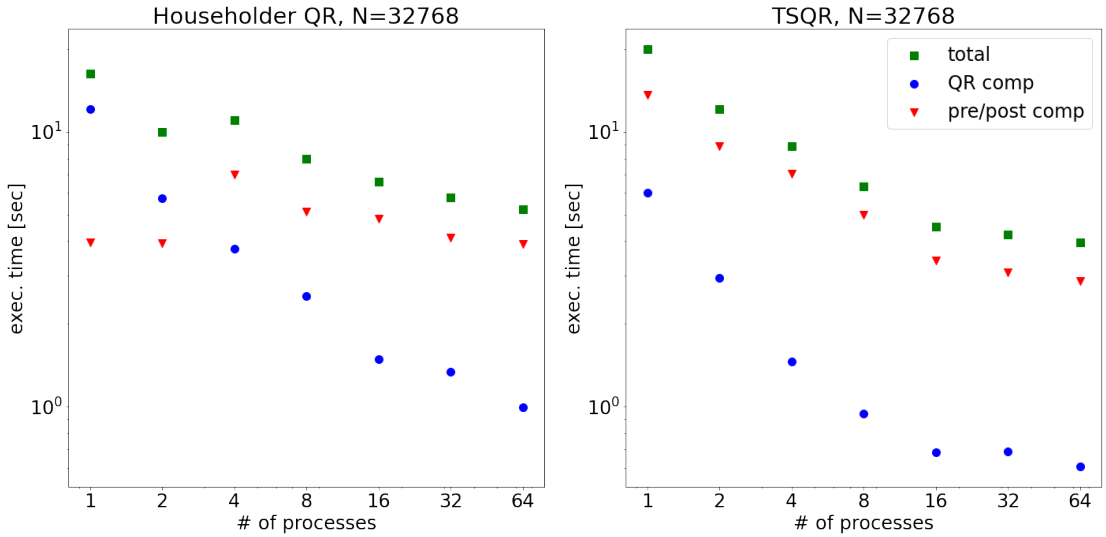

図 4　QR 分解ルーチンの実行時間内訳（$n = 32768$）