ハイブリッドクラスタシステムにおける
通信削減タイルQR分解実装

高柳雅俊

山梨大学大学院情報機能システム工学専攻

１．はじめに

高い性能対消費電力比を有するGPUを汎用計算に用いるGPUコンピューティングが一般的になりつつある。先日発表されたTop500[1]においても，多くのスーパーコンピュータでGPUが採用されている。そのため，このような並列計算環境を支援するためのソフトウェアが強く求められている。科学技術計算の様々な場面で使用される行列計算ライブラリを例に挙げると，CPU/GPUヘテロジニアスシステム向けに開発されたMAGMAライブラリ[2]があるが，上述した並列計算環境向けのライブラリは筆者の知る限り存在しない。

本研究では，基本的な密行列計算の１つである QR 分解のヘテロジニアスクラスタシステム上での実装を行う。QR分解は，特異値分解や固有値分解で用いられ，データマイニングや画像解析など多くの分野で利用される数値計算である。これらの問題はQR分解による前処理，後処理が計算時間の大部分を占めるため，行列分解の高速化を行うことが計算全体の高速化につながる。行列分解の高速化手法としてタイルアルゴリズムが考案されている[3,4]。タイルアルゴリズムによる密行列の行列分解は，並列実行可能な細粒度のタスクを多数生成し，これらを非同期に実行することですることで，並列計算資源への負荷不均衡を減らすことが可能である。

一方，分散メモリ環境においては通信回数を削減することが重要な課題となるが，QR 分解に対しては CAQR アルゴリズムが提案されている[5]。

筆者はCPU/GPUクラスタシステムにタイルCAQRアルゴリズムを，OpenMP 4.0 の task 構文による動的タスクスケジューリングと，MPI，cuBLAS を用いて実装し高速化を行った。本研究では，その実装の詳細と性能評価について報告する。

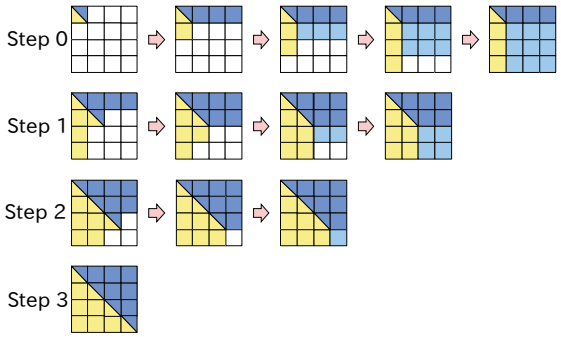
２．QR分解

本稿では，$m \times n\ (m \geq n)$の行列Aに対するQR分解を考える。QR分解の計算アルゴリズムには，グラム・シュミットの直交化や，ギブンス回転，ハウスホルダー変換を用いたものがあるが，数値的に安定なハウスホルダーQRを用いる。QR分解は固有値問題や特異値分解の前処理などに用いられており，数値線形代数計算の重要なアルゴリズムである。QR分解を含む行列分解の効率的な手法として，ブロックアルゴリズムやタイルアルゴリズムが提案されている。本研究ではタイルアルゴリズムを用いる。

２．１．タイルアルゴリズム

タイルアルゴリズムでは，行列を小行列（タイル）に分割し，カーネルと呼ばれるタスクを１または２タイルごとに実行し，各タイルの分解，更新を行う(図１)。適切なタイルサイズを選択することで，並列計算資源に対して適切な量のタスクの生成が可能である。また，各タス

クを非同期に実行することで，並列計算資源を可能な限り稼働状態にできるため，高速な実行が期待できる。



第 1 図：タイルアルゴリズム

タイル QR 分解では次の 4 種類のカーネルが使用される。ここで，行列$A(m \times n)$をタイルサイズ$b \times b$で分割したとき，行列Aは$p \times q$個のタイル$A_{i,j}$，$(i = 0,1,\cdots p-1)$，$(j = 0,1,\cdots,q-1)$からなる。ただし，$p = \lceil m/b \rceil$，$q = \lceil n/b \rceil$とする。

- GEQRT

  対角タイル$A_{k,k}$ $(k = 0,1,\cdots,q-1)$に対して QR 分解を行い，上三角行列$R_{k,k}$を生成する。このとき，直交行列Qは陽に生成されず，compact-WY 法[6]により単位下三角行列$V_{k,k}$と上三角行列$T_{k,k}$の 2 つの変換行列が生成される。

  $R_{k,k} \leftarrow \left(I - V_{k,k}T_{k,k}^{T}V_{k,k}^{T}\right)A_{k,k}$

- TSQRT

  GEQRT で生成された上三角行列$R_{k,k}$ $(k = 0,1,\cdots,q-1)$と，その下のタイル$A_{i,k}$ $(k < i < p)$の 2 つを組にして QR 分解を行い，$R_{k,k}$の更新を行う。このとき，変換行列として正方行列$V_{i,k}$と上三角行列$T_{i,k}$が生成される。

  $$\begin{pmatrix} R_{k,k} \\ 0 \end{pmatrix} \leftarrow \left(I - \begin{pmatrix} I \\ V_{i,k} \end{pmatrix} T_{i,k}^{T} \begin{pmatrix} I & V_{i,k}^{T} \end{pmatrix} \right) \begin{pmatrix} R_{k,k} \\ A_{i,k} \end{pmatrix}$$

- LARFB

  GEQRT カーネルで生成された変換行列$V_{k,k}$，$T_{k,k}$を，同一列の右タイル$A_{k,j}$ $(k < j < q)$に適用して後続行列の更新を行う。

  $A_{k,j} \leftarrow \left(I - V_{k,k}T_{k,k}^{T}V_{k,k}^{T}\right)A_{k,j}$

- SSRFB

  TSQRT カーネルによって生成された変換行列$V_{i,k}$，$T_{i,k}$をその右タイルの二組$A_{k,j}$，$A_{i,j}$ $(k < i < p,\ k < j < q)$に適用して更新を行う。

  $$\begin{pmatrix} A_{k,j} \\ A_{i,j} \end{pmatrix} \leftarrow \left(I - \begin{pmatrix} I \\ V_{i,k} \end{pmatrix} T_{i,k}^{T} \begin{pmatrix} I & V_{i,k}^{T} \end{pmatrix} \right) \begin{pmatrix} A_{k,j} \\ A_{i,j} \end{pmatrix}$$

4 種類のカーネルのうち，GEQRT，TSQRT を分解カーネル，LARFB，SSRFB を更新カーネルと呼ぶ。4 つのカーネルを繰り返し適用する事でタイル QR 分解を行う。

これら 4 種類のカーネルには実行時に依存関係が存在する。タイルの上から下をi方向，左から右をj方向，図 1 の Step をk方向とする。同一タイル列の分解または更新カーネルは同時に 1 カーネルしか実行できない。このi方向の逐次性をi方向依存とよぶ。同一タイル行に対する更

新カーネルは，最左列の分解カーネル実行後，変換行列が生成された後ならば並列に同時実行可能である。これをj方向依存とよぶ。第kステップにおけるすべてのカーネルは，適用するタイルの１つ前のステップ$k-1$のカーネルが終了していなければ実行できない。この依存関係をk方向依存とよぶ。この３種類の依存関係が解消されたタスクから実行可能である。

本研究では，OpenMP 4.0 から導入された task 構文の depend 節を用いて動的スケジューリングの実装を行う。depend 節では依存関係を以下の 3 種類で記述する。

- in　　指定されたデータが入力先
- out　　指定されたデータが出力先
- inout　指定されたデータが入出力先

データ依存のある引数を指定することで,そのデータ依存が解消されたタスクから実行される。以下にタイル QR 分解の動的スケジューリング実装の擬似コードを示す。

```
for k=0 to q-1 do
  #pragma omp task depend (inout:A_{k,k}) depend (out:V_{k,k}, T_{k,k})
  (R_{k,k}, V_{k,k}, T_{k,k})←GEQRT(A_{k,k})
  for j=k+1 to q-1 do
    #pragma omp task depend (inout:A_{k,j}) depend (in:V_{k,k}, T_{k,k})
    (A_{k,j})←LARFB(A_{k,j}, T_{k,k}, V_{k,k})
  end for
  for i=k+1 to p-1 do
    #pragma omp task depend (inout:R_{k,k}, A_{i,k}) depend (out:V_{i,k}, T_{i,k})
    (R_{k,k}, T_{i,k}, V_{i,k})←TSQRT(R_{k,k}, A_{i,k})
    for j=k+1 to q-1 do
      #pragma omp task depend (inout:A_{k,j}, A_{i,j}) depend (in:V_{i,k}, T_{i,k})
      (A_{k,j}, A_{i,j})←SSRFB(A_{k,j}, A_{i,j}, T_{i,k}, V_{i,k})
    end for
  end for
end for
```

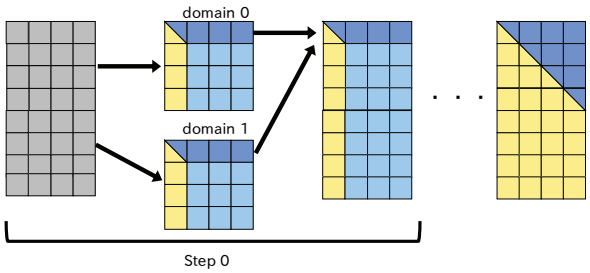
３．通信削減型アルゴリズム

分散メモリ型並列計算機におけるプログラムの実行時間は，計算時間とノード間のデータ移動にかかる通信時間の２つが大部分を占める。一般的に，ノード数を増加させるとノード間のデータ通信時間が増加するため，通信時間の削減を行うことが重要な課題である。そのため，ノード間通信の削減を行った QR 分解アルゴリズムである Communication-Avoiding QR(CAQR)が考案された[5]。本研究では，タイル化された行列データに対して CAQR アルゴリズムを実装する。

３．１．タイル CAQR アルゴリズム

タイル CAQR では，タイルに分割された行列Aを縦方向にドメインとよばれる領域に分割し，各ノードにドメインを配置する。図２では行列Aを２つのドメインに分割している。それぞれ

のドメインごとにタイル QR 分解の第kステップのタスクを実行し最上位タイルの上三角化を行う（ローカル QR 分解）。その後，各ドメインの上三角化された最上タイル行に対して QR 分解を行う。このドメイン間に対する処理をマージ処理とよぶ。マージ処理が終了すると，第kステップが終了する。ドメインに分割することでドメイン間のマージ処理による演算量が増えるが，ドメインごとに並列にローカル QR 分解を行うことが可能なため，i方向の並列性が向上する。また，CAQR ではマージ処理時のみドメイン間の通信が必要となるため，クラスタシステム上にタイル QR 分解を実装した時と比較して，通信回数は少なくなる。



第２図：タイル CAQR アルゴリズム（２ドメイン分割）

タイル CAQR ではタイル QR 分解の４カーネルに加えて，マージ処理を行う２つのカーネルが必要となる。

- TTQRT

  上三角行列$R_{h1,k}$，$R_{h2,k}$の２つを組にして QR 分解を行い，$R_{h1,k}$の更新を行う。このとき，変換行列として上三角行列$V_{h2,k}$と上三角行列$T_{h2,k}$が生成される。

$$\begin{pmatrix} R_{h1,k} \\ 0 \end{pmatrix} \leftarrow \left(I - \begin{pmatrix} I \\ V_{h2,k} \end{pmatrix} T_{h2,k}^{T} \begin{pmatrix} I & V_{h2,k}^{T} \end{pmatrix} \right) \begin{pmatrix} R_{h1,k} \\ R_{h2,k} \end{pmatrix}$$

- TTMQR

  TTQRT カーネルによって生成された変換行列$V_{h2,k}$，$T_{h2,k}$をその右タイルの二組$A_{h1,j}$，$A_{h2,j}$ $(k < j < q)$に適用して更新を行う。

$$\begin{pmatrix} A_{h1,j} \\ A_{h2,j} \end{pmatrix} \leftarrow \left(I - \begin{pmatrix} I \\ V_{h2,k} \end{pmatrix} T_{h2,k}^{T} \begin{pmatrix} I & V_{h2,k}^{T} \end{pmatrix} \right) \begin{pmatrix} A_{h1,j} \\ A_{h2,j} \end{pmatrix}$$

本研究では，タイル CAQR アルゴリズムを CPU/GPU ヘテロジニアスクラスタシステムへの実装を行う。次節では GPU 実装について詳細を示す。

４．ヘテロジニアスクラスタシステムでのタイルアルゴリズム実装

４．１．更新カーネルの GPU 実装

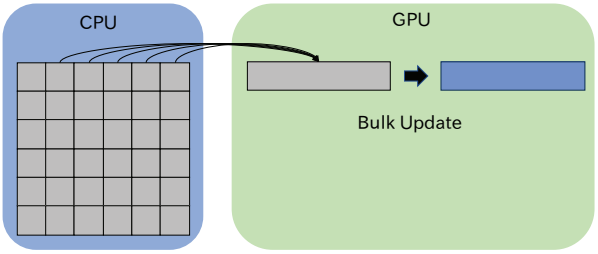
後続行列更新を行う SSRFB カーネルについて前節で述べた。以下は SSRFB カーネルを BLAS ルーチンを用いて実装したものである。ここで，Wは作業領域である。

$$W = A_{k,j} + V_{i,k}^{T} A_{i,j} \quad \text{(GEMM)}$$
$$W = T_{i,k}^{T} W \quad \text{(TRMM)}$$
$$A_{k,j} = A_{k,j} - W \quad \text{(AXPY)}$$
$$A_{i,j} = A_{i,j} - V_{i,k} W \quad \text{(GEMM)}$$

上の式から，SSRFB カーネルは L3 BLAS 演算が大部分を占めている。また，他の更新カーネルも同様に L3 BLAS が主要演算である。本研究では更新カーネルを cuBLAS[7]を用いて実装し，後続行列更新処理を GPU に割り当てる。一方，逐次処理が多い GEQRT，TSQRT，TTQRT カーネルおよび GPU の制御を CPU で行う。また，大規模な行列へ適用するために，行列データは CPU のホストメモリ上に確保し，タイルデータが必要となった時に GPU へと送受信する。そのため，更新カーネル実行の度に CPU-GPU 間で変換行列，後続行列のデータ通信が発生する。以下では本研究で実装した２種類の GPU 更新手法について説明する。

４．２．Bulk Update

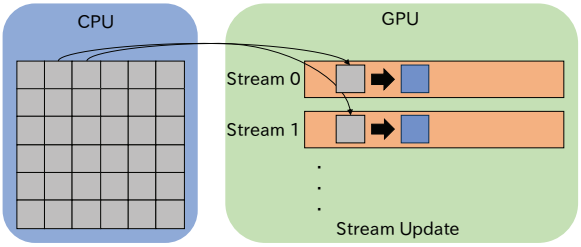
従来の GPU プログラミングでは１つのカーネル内で GPU の全スレッドを使用した並列計算を行うことで，GPU の計算性能を生かしている。そのため，GPU の全計算資源が稼働状態となるように，大きな行列に対して行列演算を行うことでより高い処理性能が得ていた。しかし，本実装におけるタイル CAQR アルゴリズムでは，GPU で処理する更新カーネルは１または２タイルのみに適用される。タイルサイズを大きくすると，更新カーネルは GPU により高速に計算されるが，分解カーネルの逐次処理に時間がかかり，GPU の処理開始が遅延しボトルネックとなってしまう。そこで，小さいタイルサイズを選択しつつ GPU の性能を発揮させる方法として Bulk Update[8-11]が提案された。Bulk Update では図のように，並列実行可能なj方向のタイルを１つの長方行列にまとめ，一度に更新カーネルを適用する。長方行列にしてまとめて更新を行うことで，GPU での更新カーネルの性能を向上できる。



第３図：Bulk Update の更新手法

４．３．Stream Update

NVIDIA 社の Kepler アーキテクチャ以降の GPU では，異なる CUDA ストリームに複数の GPU タスクが投入された時，可能ならばそれらが自動的に並列実行される。そこで図４のように，各タイル列に対応する CUDA ストリームを作成し，各タイル列の更新タスクを投入する。こうすることで並列実行可能な更新カーネルが非同期に並列実行できるため，性能向上につながる。この更新手法を Stream Update と呼ぶ[12]。Bulk Update 実装では，一度に更新タスクを適用するため，タスク粒度が大きくなり，タイルアルゴリズムの「細粒度タスクの非同期実行」という特徴と相反する。しかし，Stream Update では，細粒度のタスクのまま実行が可能であり，複数カーネルの同時実行により，計算資源の稼働率を向上させることが可能と考えられる。一方で，タイルサイズ分の小さなデータ転送が頻発する。ホスト-デバイス間の双方向データ通信は逐次実行であるため，性能を阻害する可能性がある。これら２種類の更新手法を Reedbush-H 上に実装した。



第４図：Stream Update の更新手法

５．性能評価実験

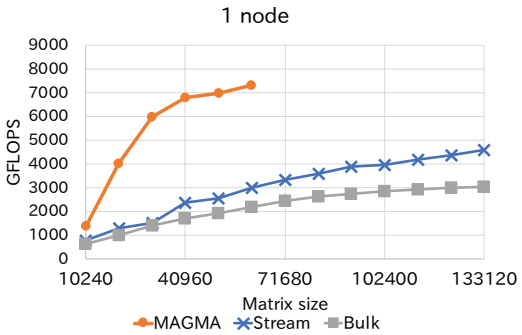
東京大学情報基盤センターの Reedbush-H 上でタイル CAQR の実装を行い，正方行列に対する QR 分解の性能評価を行った。使用した開発環境は表１のとおりである。GPU で実行される更新タスクには上記２種類の手法が実装されている。Reedbush-H 上で，２種類の実装および MAGMA の性能比較および，１ノードから 16 ノードを使用した並列化効率の性能測定を行った。タイルサイズについてはチューニングを行い，各行列サイズに対して最適なタイルサイズを選択した。

表１：使用した開発環境

コンパイラ	Intel C++ コンパイラ 17.0.1.132
MPI ライブラリ	Intel MPI 2017.1.132
BLAS	Intel MKL 17.1.132
CUDA & cuBLAS	8.0.44
MAGMA	2.2.0

５．１．シングルノード

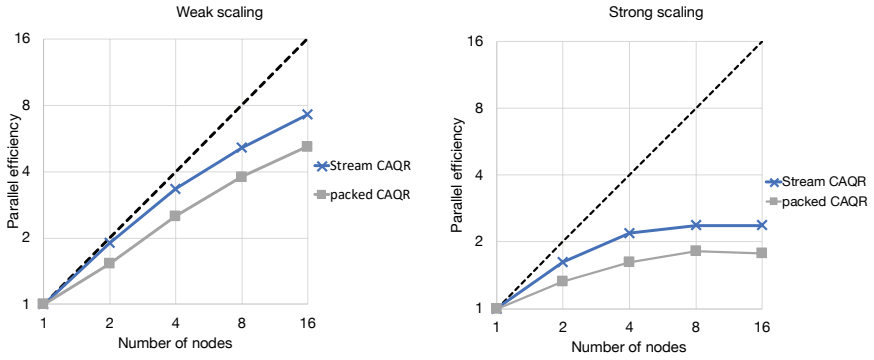
最初に，Reedbush-H の１ノードを使用して，正方行列に対して各実装の QR 分解と MAGMA ライブラリ実装の実行速度を比較した（図５）。今回行った Stream Update と Bulk Update では，Stream Update 実装の方が実行速度が速かった。これは，OpenMP と CUDA Stream による非同期実行が影響したと考えられる。Stream Update 実装は MAGMA のピーク性能の約半分の性能だが，MAGMA と比較して約２倍のサイズの正方行列に対して QR 分解を実行することができる。MAGMA は全行列データを GPU メモリ上に保存し，CPU で実行される分解タスクに必要なパネル部分をメインメモリ上に転送する方式を取る。しかし，我々の実装では個々の更新タスクに対して毎回 CPU-GPU 間の転送が必要であるため，MAGMA よりも多くのデータ転送が必要となる。そのため，MAGMA が処理できる行列サイズでは非常に大きな性能差が生じてしまう。



第５図：１ノードによるタイル CAQR 性能測定

５．２．マルチノード

Stream Update と Bulk Update の２つの手法を使用して，weak scaling の性能評価を行った。1 ノードあたりの行列サイズを81920 × 81920とし，1 ノードから 16 ノードまでの台数効果を測定した。行列データはタイル行ごとに 1D ブロックサイクリックデータ分散で各ノードに分散させている。weak scaling の測定結果を図７に示す。2 種類の実装のうち Stream Update 実装の方が高い並列化効率を示し，16 ノード使用時で理想性能の約５０％の性能となっている。これは，タイル CAQR アルゴリズムによる縦方向の並列化が影響していると考えられる。



第６図：タイル CAQR アルゴリズムの weak scaling（左図）および strong scaling（右図）

次に，Stream Update と Bulk Update の strong scaling の性能評価を行った。行列サイズを102400 × 102400の正方行列に固定し，1 ノードから 16 ノードまでの台数効果を測定した。測定結果を図８に示す。行列データの分散方法は，weak scaling と同様に 1D ブロックサイクリックデータ分散で行っている。この測定実験においても，Stream Update 実装の方が高い並列化効率を得られたが，16 ノードに注目すると十分な並列化性能とは言えない。この原因は，ノード数が増加させると，プロセス数も増加するため，ドメイン内タイル QR 分解の並列度は向上するが，1 プロセスが処理するドメイン内のタイル数が少なくなり，またドメイン間のマージ処理回数が増え，ドメインのマージ処理がボトルネックとなる。これが，並列化効率が阻害され十分な性能が得られない要因であると考えられる。

６．おわりに

今回，タイル CAQR アルゴリズムの CPU/GPU クラスタシステム実装を行い，Reedbush-H を使用した性能評価を行った。本稿では，MAGMA ライブラリとの性能比較および，クラスタシステムでの並列化効率の測定を行った。性能測定実験では，weak scaling で１６ノード使用時に理想性能の約５０％の性能が得られ，大規模並列環境でのタイル CAQR アルゴリズムの有用性が見られることがわかった。ただし，タイルアルゴリズムでは適切なタイルサイズを選択しなければ高い性能が得られず，この実験でもタイルサイズチューニングのために複数回のパラメータ探索が必要であった。そのため，事前に実行時間の予測を行うために，性能モデルによるタイルサイズチューニングが必要である。

謝辞

本研究は「東京大学情報基盤センター　平成２９年度若手・女性利用推薦課題」における採

択課題「ハイブリッドクラスタシステムにおける通信削減タイル QR 分解実装」によって行われたものです。本研究では，東京大学情報基盤センターの Reedbush-H を使用させていただきました。この場を借りてお礼申し上げます。

参考文献

[1] Top500, https://www.top500.org

[2] MAGMA, https://icl.cs.utk.edu/magma/

[3] Buttari, A. Langou, J., Kurzak, J. and Dongarra,J.：Parallel tiled QR factorization for multicore architectures, Concurrency and Computation, Practice and Experience, Vol. 20, No. 13, pp. 1573 - 1590,2008.

[4] Buttari, A., Langou, J., Kurzak, J. and Dongarra, J.:A class of parallel tiled linear algebra algorithms for multicore architectures, Parallel Computing, Vol. 35, pp. 38 - 53,2009.

[5] Demmel, J. W., Grigori, L., Hoemmen, M. and Langou, J.: Communication-avoiding parallel and sequential QR and LU factorizations: theory and practice, Technical Report UCB/EECS-2008-74, LAPACK Working note 204,2008.

[6] R. Schreiber and C. Van Loan: A storage-efficient WY representation for products of Householder transformations, SIAM J. Sci. Statist. Comput., Vol.10, Num. 1, pp. 52 - 57,1989.

[7] NVIDIA, NVIDIA Developer cuBLAS Web Page, https://developer.nvidia.com/cublas

[8] 高柳雅俊，鈴木智博:マルチ GPU 環境におけるタイル CAQR アルゴリズムの実装，日本応用数理学会 2014 年度年会（2014）

[9] 高柳雅俊，鈴木智博:CPU/GPU 混在環境における再帰的タイル QR 分解，日本応用数理学会 2015 年度年会（2015）

[10] 高柳雅俊，鈴木智博:CPU/GPU 混在環境における再帰的タイル QR 分解の動的スケジューリング実装，日本応用数理学会 第 12 回研究部会連合発表会（2016）

[11] 高柳雅俊，鈴木智博:クラスタ型ヘテロジニアス環境におけるタイル QR 分解，日本応用数理学会 2016 年度年会（2016）

[12] 高柳雅俊，鈴木智博: CPU/GPU クラスタシステムにおけるタイル QR 分解の実装と性能評価，第 23 回計算工学講演会（2018）