Vol.2017-HPC-158 No.23
2017/3/9

縦長行列におけるタイルCAQRアルゴリズムの性能評価

高柳 雅俊[1,a)]　鈴木 智博[1]

概要：行列分解のためのタイルアルゴリズムは高い並列性をもつため，近年のマルチコア，メニーコア環境向きの手法として注目を集めている．我々はクラスタシステム上に OpenMP/MPI ハイブリッド実装によるタイル QR 分解の並列実装を行っている．今回はクラスタシステムにおける通信削減型タイル QR 分解 (タイル CAQR) の実装と，京コンピュータにおける性能評価について報告する．

Performance result of tile CAQR for tall and skinny matrices.

Masatoshi Takayanagi[1,a)]　Tomohiro Suzuki[1]

Abstract: The tile algorithm for a matrix decomposition has an ability to generate many fine-grained tasks, so it is suitable for modern multicore/manycore architectures. We implemented the tile QR decomposition algorithm in OpenMP/MPI hybrid fashion on the cluster system. In this report, we show the performance result of our communication-avoiding tile QR (tile CAQR) implementation for tall and skinny matrices on the K computer.

1. はじめに

近年，大規模化する科学技術計算を高速に行うためにスーパーコンピュータが用いられる．現在のスーパーコンピュータはマルチコアクラスタシステムが主流であり，比較的小規模な科学技術計算においても数百から数千の CPU を扱うことが日常的になりつつある．CPU の演算能力と比較してノード間通信の速度は非常に遅いので，分散メモリ型のスーパーコンピュータではアプリケーションプログラムの高速化のために通信時間を削減することが最重要な課題である．

科学技術計算に用いられる密行列の数値線形代数計算において，行列分解はさまざまな前処理に適用される重要なアルゴリズムである．通常，ユーザは各スーパーコンピュータベンダーから提供される数値計算ライブラリに含まれる LU 分解，QR 分解，Cholesky 分解などの基本的な行列分解のプログラムを利用して科学技術計算プログラムを作成する．クラスタシステム上で用いられる行列計算ライブラリに ScaLAPACK[1] があり，スーパーコンピュータユーザ以外でも netlib[2] からソースコードを入手可能である．ScaLAPACK はクラスタシステム向けの行列計算ライブラリとしてはデファクトスタンダードな位置にあるが，マルチコアアーキテクチャへの対応の遅れが指摘されており，最新のクラスタシステム向けの行列計算ライブラリが求められている．

LAPACK[3]，ScaLAPACK で使用されているブロックアルゴリズムは，パネル分解と後続行列更新を繰り返しながら行列分解を進める．後続行列更新に高速な L3 BLAS 演算が適用されるためブロックアルゴリズムは逐次アルゴリズムに比べ高い性能を発揮するが，パネル分解部分は逐次アルゴリズムで実装されており，その逐次性からマルチコアアーキテクチャの並列計算資源を有効に利用できず，大規模問題ではこの部分がボトルネックとなる．

現在のマルチコア CPU が有する並列計算資源を活かすアルゴリズムとして，行列分解に対するタイルアルゴリズム [4]，[5] が注目されている．タイルアルゴリズムは，行列を小行列（タイル）に分割し，1 または 2 タイルごとに行列分解を行う細粒度のタスクを大量に生成し，これを非同期に実行させることでマルチコア CPU の豊富な計算資

[1] 山梨大学大学院総合研究部工学域
Graduate School of Interdisciplinary Research, Faculty of Engineering, University of Yamanashi
[a)] g16dm002@yamanashi.ac.jp

源を有効に活用することが可能である．

本研究では，行列分解の一つである密行列の QR 分解を扱う．QR 分解を行う並列計算アルゴリズムとして，近年，通信削減という点から CAQR アルゴリズム [6] が提案され注目されている．我々はタイルアルゴリズムに基づく CAQR を，理化学研究所計算科学研究機構（ AICS ）の京コンピュータ上に実装し，その性能評価について報告する．タイルアルゴリズムはデータ依存のあるタスク並列プログラミングモデルであるため，性能モデルの構築が比較的困難である．今回，クラスタシステム上のタイル CAQR アルゴリズムの性能モデルを構築し，評価実験の結果と合わせてその有用性を検証する．

2. タイル QR 分解

与えられた $m \times n\,(m \geq n)$ 行列 A に対して，

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

を QR 分解と呼ぶ．ここで， Q は $m \times m$ 直交行列， R は $m \times n$ 上三角行列である．QR 分解の計算アルゴリズムとして，グラム・シュミットの直交化を用いたもの，ハウスホルダー変換を用いたものなどがあるが，本研究では数値的に安定な後者を用いる．QR 分解は最小二乗法の正規方程式の安定な解法として知られている．また，固有値分解，特異値分解の前処理としても多用されており，数値線形代数計算の重要なアルゴリズムである．QR 分解を含めた行列分解の並列化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．

2.1 ブロックアルゴリズム

行列分解のブロックアルゴリズムは，行列から切り出したブロックサイズ幅のパネルに対して分解処理を行った後，後続行列の更新を行う．パネル分解，後続行列更新を繰り返し行うことで行列分解が完了する．並列化された L3 BLAS が高い性能を発揮するので，後続行列更新に L3 BLAS が適用可能であるブロックアルゴリズムは，L2 BLAS が主要演算となる逐次アルゴリズムに比べ高い性能を発揮する．しかし，パネル分解部分は逐次アルゴリズムで実装されており，逐次性からマルチコアアーキテクチャの並列計算資源を有効に利用できず，大規模問題ではこの部分がボトルネックとなる．このような fork-join 型の並列計算モデルは，現在主流となっているマルチコア・メニーコアなどの高並列環境では計算資源を効率的に利用出来ない．

2.2 タイルアルゴリズム

タイルアルゴリズムは行列を小行列（タイル）に分割し，分解，更新処理を 1 または 2 タイルごとに行う（**図 1**）．ある程度大きな行列に対しては，データ参照の局所性を利用

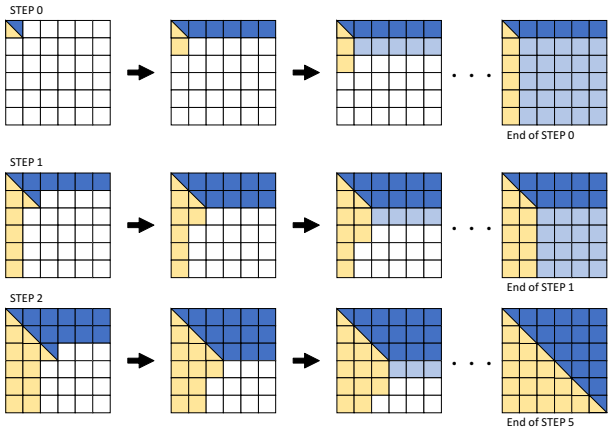

図 1 タイル QR 分解
Fig. 1 Tile QR decomposition

した高速化が期待できる一方で，タスクの粒度が大きいほど負荷不均衡が生じやすくなる．タイルアルゴリズムではこのようなトレードオフを考慮しながら適切なタイルサイズを選択することが重要である．並列計算資源に応じた量の細粒度タスクを生成し，これを非同期に実行することでマルチコア CPU の計算資源を可能な限り稼動状態にすることができ，高速な実行が期待できる．タイルアルゴリズムは近年の高並列な計算資源を有効に活用できる手法として注目されている．

式 (1) の行列 A はタイルサイズを $b \times b$ としたとき，$p \times q$ 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．

2.2.1 タイル QR 分解のカーネル

タイル QR 分解は次の 4 種類のカーネルで構成される．

- GEQRT : 対角タイル $A_{k,k}(k = 0, 1, \cdots, q-1)$ の QR 分解を行い，上三角行列 $R_{k,k}$ を生成する．直交変換 Q は陽には生成されず，compact-WY 法 [7] により単位下三角行列 $V_{k,k}$ と上三角行列 $T_{k,k}$ 2 つの変換行列が生成される．
- TSQRT : 上三角行列 $R_{k,k}(k = 0, 1, \cdots, q-1)$ とその下にあるタイル $A_{i,k}(k < i \leq p-1)$ の 2 つを組にして QR 分解を行い，$R_{k,k}$ の更新を行う．変換行列として正方行列 $V_{i,k}$ ， 上三角行列 $T_{i,k}$ が生成される．
- LARFB : GEQRT カーネルによって生成された変換行列 $V_{k,k}, T_{k,k}$ をその右タイル $A_{k,j}(k < j \leq q-1)$ に適用し更新を行う．

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

- 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 $A_{k,j}, A_{i,j}(k < i \leq p-1, k < j \leq q-1)$ に適用する．

$$\begin{bmatrix} A_{k,j} \\ A_{i,j} \end{bmatrix} \leftarrow \left[I - \begin{bmatrix} I \\ V_{i,k} \end{bmatrix} T_{i,k}^T \begin{bmatrix} I & V_{i,k}^T \end{bmatrix} \right] \begin{bmatrix} A_{k,j} \\ A_{i,j} \end{bmatrix}$$

GEQRT，TSQRT を分解カーネル， LARFB，SSRFB

を更新カーネルと呼ぶ．分解カーネルの実装はブロックアルゴリズムで行われることで高速化される．このブロック幅を内部ブロック幅と呼ぶ．

2.2.2 カーネルの依存性

前節で示した４つのカーネル実行には依存関係が存在する．ここで，タイルの上から下を i 方向，左から右を j 方向，図 1 におけるステップの方向を k 方向とする．同一タイル列の分解または更新は同時に 1 カーネルしか実行出来ない．この i 方向の逐次性を i 方向依存とよぶ．同一タイル行の更新カーネルは，最左列の分解カーネルにより変換行列が生成された後でなければ実行できない．これを j 方向依存とよぶ．ただし，同一タイル行の更新カーネルは並列実行可能である．ステップ k における全てのカーネルは，同一タイルのステップ k-1 のカーネルが終了していなければ実行できない．これを 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イルを扱うタスクから実行が可能である．

2.2.3 タスクスケジューリング

上記の依存性に従ったタスクのスケジューリングに関して複数の手法が提案されている [8]，[9]，[10]．タスクスケジューリング手法は静的スケジューリングと動的スケジューリングに大別されるが，タイルアルゴリズムの特徴である細粒度タスクの非同期実行を実現するためには動的スケジューリングが必要となる．

ステップ k のすべてのタスクが終了する前にステップ k+1 のタスクを実行することをルックアヘッドと呼び，ステップ k+h のタスクまで実行している場合，ルックアヘッドの深さは h であると言う．タイル QR では，動的スケジューリングを行うことで深いルックアヘッドが実現され，分解カーネルの i 方向依存，k 方向依存がオーバーヘッドになりにくい．

データ依存のあるタスクの動的スケジューリングを実装するために，我々はタスクキューと依存関係を管理するプログレステーブルを使用している．

2.3 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実行時間の内訳は，演算時間とノード内のデータ移動を含めた通信時間の二種類である．問題規模を固定して CPU 数を増加させると各 CPU における計算時間は減少するが，通信時間は増加する．そのため，並列化効率を向上させるためには通信時間の削減（ Communication-Avoiding ）が重要となる．

本研究ではノード間の通信時間に関するモデルとして，alpha-beta model[6] を用いる．これは通信のレイテンシを α，ネットワークバンド幅の逆数を β，通信するデータサイズを n とし，通信時間 $T_{comm} = \alpha + \beta n$ とモデル化するものである．

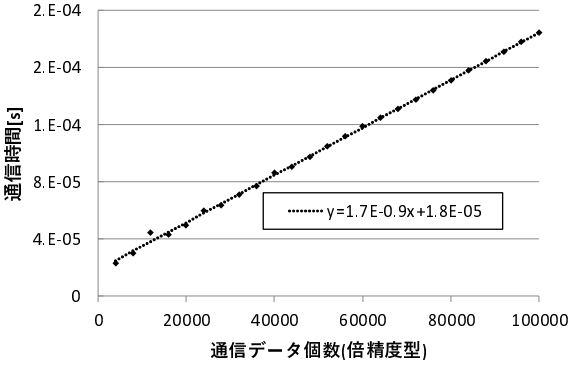

図 2　2 ノード間の MPI_Send の実行時間
Fig. 2

図 2 は京コンピュータ 2 ノード間における MPI 一対一同期通信 MPI_Send について，double 型のデータ個数を変化させ，通信時間を測定したものである．この図には最小二乗法で得られた回帰直線も示されており，これよりモデルの各パラメータは $\alpha \simeq 1.8 \times 10^{-5}$ (秒)，$\beta \simeq 1.7 \times 10^{-9}$ (秒/個) である．この結果から，通信一回を行うのに必要な立ち上がり時間が，データ一個あたりの送信コストよりも非常に大きいことが確認できる．ゆえに，通信時間を減らすためにはデータサイズよりも通信回数を減らすことが有効であることが分かる．通信回数を削減する QR 分解アルゴリズム CAQR はこのような背景の下に考案された [6]．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．

2.4 タイル CAQR アルゴリズム

行列データをタイルごとに 2D ブロックサイクリックデータ分散で配置したタイル QR 分解の実装について考える．クラスタシステム上で実装したとき，必要となる通信は 2 種類ある．ひとつ目は，分解カーネルで生成された変換行列を更新カーネルを実行するプロセスに送信する通信（ 1 対多通信）．また，TSQRT，SSRFB カーネルを実行するプロセスは，他のプロセスからカーネル実行に必要となる下部タイルのデータを受信しなければならない（ 1 対 1 通信）．このように，タイル QR をそのままクラスタシステムに実装するのでは多くの通信が必要となるため，通信の削減を行う必要がある．

タイル CAQR では $A_{i,j}$ を縦方向に再分割し，それぞれの領域（ドメイン）ごとにタイル QR 分解のステップ k を行う．**図 3** では $A_{i,j}$ を 2 つのドメインに分割している．ステップ k における分解，更新カーネルの実行完了後，各ドメインの最上タイル行の更新を行う（図の橙部）．このドメイン間にまたがる処理をマージ処理とよぶ．

ドメインに分割することでマージ操作による演算量が増えるが，ドメインごとにタイル QR 分解を行えるため，i 方向の並列度が高くなる．また，ドメイン間の通信がマー

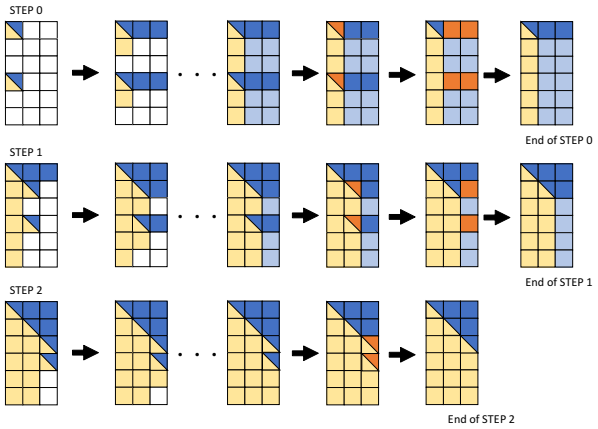

図 3　タイル CAQR 分解
Fig. 3　Tile CAQR Algorithm

ジ処理時のみになるため，通信回数の削減につながる．[6] では複数プロセスに分割した行列データを二分木的にマージするとき，プロセス間通信の回数が最小になることが示されている．

2.4.1　タイル CAQR のマージカーネル

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

- TTQRT：上三角行列 $R_{h1,k}$ とその下にある上三角行列 $R_{h2,k}$ の 2 つを組にして QR 分解を行い，$R_{h1,k}$ の更新を行う．変換行列として上三角行列 $V_{h2,k}$，上三角行列 $T_{h2,k}$ が生成される．
- 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 $A_{h1,j}, A_{h2,j}$ に適用する．

$$\begin{bmatrix} A_{h1,j} \\ A_{h2,j} \end{bmatrix} \leftarrow \left[I - \begin{bmatrix} I \\ V_{h2,k} \end{bmatrix} T^T_{h2,k} \begin{bmatrix} I & V^T_{h2,k} \end{bmatrix} \right] \begin{bmatrix} A_{h1,j} \\ A_{h2,j} \end{bmatrix}$$

3.　関連研究

行列分解のタイルアルゴリズム実装に PLASMA ライブラリ [11] がある．PLASMA は共有メモリ環境，マルチコアアーキテクチャを想定し実装されている．PLASMA ではマルチスレッドライブラリに pthread を使用しているが，我々は OpenMP を用いた並列実装を行っている．最近では， PLASMA プロジェクトでも OpenMP を使用した実装が始められている．

参考文献 [12] では共有メモリ環境において，後述するフラットバイナリツリー方式，バイナリツリー方式のタイル CAQR 分解を実装している．また，DAG(Directed Acyclic Graph) という依存関係を表すタスクグラフにもとづくタスクスケジューリングを行っている．

参考文献 [13] ではマルチコアクラスタシステム上でタイル CAQR 分解を実装している．行列データの分散方法として 2D ブロックサイクリックデータ分散を行い，ドメイン内の分解，更新タスクのスケジューリングとして Greedy や Fibonacci などを用いている．この実装はタスクスケジューリングに DAGuE というフレームワークを用いているが，我々の実装は OpenMP，MPI のみを用いた実装であり，可搬性が高い．

4.　CAQR アルゴリズムのリダクションツリー

クラスタシステム上に CAQR アルゴリズムを実装するにあたって，行列データをタイル行ごとに各プロセスに分散させる 1D ブロックサイクリックデータ分散を行った．また，今回の実装では 1 プロセスは 1 ドメインのタイル QR 分解を行う．タイル CAQR では，第 k ステップにおいて最上ドメインの第 k タイル行を持つプロセスに各ドメインのデータがマージされるので，このプロセスの負荷が高くなる．1D ブロックサイクリックデータ分散を採用することにより，各 k ステップごとに最上タイル行を保有するプロセスが変わるので，マージ処理の負荷分散が可能となる．

今回の実装ではドメイン間のマージ処理のスケジューリングとして，3 種類のスケジューリングを行った．

- フラットツリー：最上ドメインとその下のドメインの上三角部分を逐次的にマージする．
- フラットバイナリツリー：行列を 2 つに分割し，それぞれの中でフラットツリー状のマージを行う．最後に分割した 2 つのマージを行う．
- バイナリツリー：二分木によるリダクション演算で，全ドメインを 2 組ずつマージする．

各スケジューリングの概要を**図 4** に図示する．

フラットツリーは最上ドメインへのマージを逐次的に行うため木の高さが $d-1$ となり，最も時間がかかる．しかし，第 k ステップのドメイン間マージが終了する前に，第 k+1 ステップのドメイン間マージのタスクを始められるため，ある程度ドメインの幅がある場合には他の手法よりもマージ処理のスループットが高くなる．ただし，今回の我々の実装ではドメイン内分解とノード間マージの間に同期処理が行われるため，この効果は得られない．ただし，第 k ステップのドメイン間マージと，第 k+1 ステップのドメイン内分解の間に同期はない．

バイナリツリーは木の高さが $\log d$ で最も低く，マージの時間が最も短くなることが予想される．フラットバイナリツリーは二つの手法を組み合わせたもので，ドメイン分割のやり方によりフラットツリーとバイナリツリーの割合は可変であるが，今回は行列を上下二つのドメインに分割し，ドメイン内でフラットツリーを実行した後，二つの結果をマージすることにした．

ドメイン数を d とした時，各スケジューリングで第 k ステップにおけるクリティカルパス上の通信回数を**表 1** に示す．

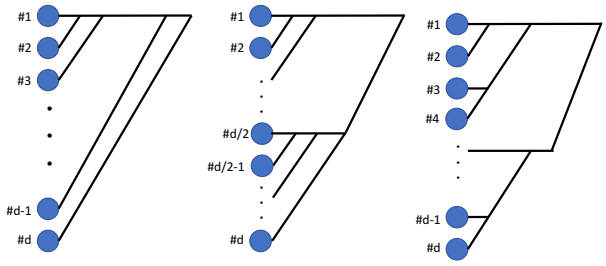

図 4　ドメイン数 d のときの各スケジューリング方法．
左から，フラットツリー，フラットバイナリツリー，バイナリツリーである．

Fig. 4 (left) flat-tree scheduling. (center) flat-binary-tree scheduling. (right) binary-tree scheduling.

表 1　各通信方式における通信回数

Table 1 The number of communication times

通信方式	通信回数
フラットツリー	$d-1$
フラットバイナリツリー	$d/2$
バイナリツリー	$\log d$

5. 性能モデル

前述のとおり，タイルアルゴリズムにおいて，タイルサイズは重要なパラメータであり，計算速度に大きな影響を与える．しかし，大規模並列環境で最適なタイルサイズを得るためにのパラメータサーチは非常にコストがかかる．今回，パラメータチューニングを行うことを一つの目的として，タイルサイズをパラメータとしたタイル CAQR の性能モデルを構築した．

CAQR アルゴリズムの全体実行時間 T_{all} は次の 3 つの実行時間の総和とする．

- ドメイン内タイル QR 分解の計算時間 T_{QR}
- ドメイン間マージの計算時間 T_{merge}
- ドメイン間の通信時間 T_{comm}

各実行時間に対するモデルを立て，全体の性能モデルを構築する．次のような状況を仮定する．

- 行列サイズ $m \times n$
- タイルサイズ $b \times b$
- タイル数 $p \times q$ $(p = \lceil m/b \rceil, q = \lceil n/b \rceil)$
- ノード数 P
- ノードあたりの CPU コア数 C
- ドメイン内タイル行数 d $(d = p/P, d \geq n, p \bmod P = 0)$

5.1 ドメイン内タイル QR 分解の実行時間

今回，ドメイン内のタイル QR 分解は動的スケジューリングで実装している．これは非同期のタスク並列プログラムであり，複数のタスクがオーバーラップするため性能モデルを構築するのは難しい．タイル QR 分解において，

表 2　各カーネルの実行時間

Table 2 Kernel execute time

カーネル	T_{SSRFB}	T_{TTQRT}	T_{TTMQR}
実行時間 [s]	0.025	0.018	0.017

もっとも支配的な計算カーネルは SSRFB カーネルである．例えば，4000×1000 の縦長行列に対して，タイルサイズ 100×100 でタイル QR 分解を実行すると，SSRFB カーネルの比率は約 80%となる．そこで今回は SSRFB カーネルの実行回数から，ドメイン内のタイル QR 分解の性能モデルを構築する．

第 k ステップにおける SSRFB の実行回数は，全 CPU コアを用いて第 i 行の SSRFB カーネルを全て実行するのにかかる回数 $\lceil (q-k-1)/C \rceil$ とタイル行数 $d - \lfloor k/P \rfloor$ の積である．

$$(d - \lfloor k/P \rfloor) \times \lceil (q-k-1)/C \rceil \tag{2}$$

よって，タイルサイズ $b \times b$ のときの SSRFB カーネル実行時間を T_{SSRFB} としたとき，ドメイン内タイル QR 分解の実行時間の性能モデルは以下のようになる．

$$T_{QR} = T_{SSRFB} \times \sum_{k=0}^{q-1} (d - \lfloor k/P \rfloor) \times \lceil q/C \rceil \tag{3}$$

5.2 ドメイン間マージの実行時間

ドメイン間マージの実行時間は計算時間 T_{merge} とドメイン間通信 T_{comm} の和である．1 回のマージ処理の実行時間は，TTQRT 実行時間を T_{TTQRT}，TTMQR 実行時間を T_{TTMQR} とすると

$$T_{TTQRT} + T_{TTMQR} \times \lceil (q-k-1)/C \rceil \tag{4}$$

である．この時，データの送受信にかかる通信時間は alpha-beta model から，

$$2 \times (\alpha + \beta b^2 q) \tag{5}$$

となる．表 1 の各実装方式の通信回数を N とすると，それぞれの実行時間は (6)，(7) となる．

$$T_{merge} = \sum_{k=0}^{q-1} (T_{TTQRT} + T_{TTMQR} \times \lceil (q-k-1)/C \rceil) N \tag{6}$$

$$T_{comm} = \sum_{k=0}^{q-1} 2(\alpha + \beta b^2 (q-k)) N \tag{7}$$

ゆえに全体の実行時間は $T_{all} = (3) + (6) + (7)$ で求められる．また，京コンピュータにおけるタイルサイズ 400 の各カーネル実行時間は**表 2** のとおりである．

6. 実験

我々のタイル CAQR 実装の性能評価を京コンピュータ上で行った．フラットツリー，フラットバイナリツリー，

バイナリツリーの 3 種類と，ScaLAPACK との比較を行った．ScaLAPACK のデータ分散は我々の実装と同様の形状である行方向の 1D ブロックサイクリックデータ分散とし，ブロック幅はパラメータサーチを行った結果から，縦横ともに 200 とした．また，我々の実装の各計算カーネルは PLASMA ライブラリのものを使用した．

6.1 weak scale

京コンピュータの 1 ノードから 4096 ノードを使用して，縦長行列の QR 分解速度測定を行った．事前の簡単なタイルサイズチューニングからタイルサイズを 400，内部ブロック幅を 100 とした．1 ノードあたりの行列サイズを 64000 × 8000 とし，各実装の並列化効率を測定した．**図 5** はその結果であり，上が実行時間，下が相対速度のグラフである．

我々が行った 3 種類の実装において，少ないノード数の時ではどの実装もよくスケールしているが，ノード数が増加するとバイナリツリーによるマージスケジューリングが最速となった．これは，ノード数が多くなるほど CAQR のマージ処理の実行回数に大きな差が出てくるためと考えられる．また，ScaLAPACK の実装と比較するとノード数が少ない時は速度差が小さいが，ノード数を増加すると，我々のバイナリツリー実装の方が速くなる．このことから，ScaLAPACK の実装より我々の実装が大規模行列に向いていると言える．

6.2 strong scale

図 6，**図 7** はそれぞれ行列サイズを 64000 × 8000 512000 × 8000 に固定し，並列化効率の測定を行った結果である．縦長行列に対して，バイナリツリーは他のスケジューリング方式と比べ強スケールしている．フラットツリーがバイナリツリーと比べ強スケールしないのは，ノード数が増加すると 1 プロセスあたりのドメイン数が小さくなりドメイン内タイル QR 分解は並列化されるが，ドメイン間のマージ処理回数が最も多いく，マージ処理の負荷が増えたためスケールしないと考えられる．ノード数が少ない時，ScaLAPACK による実装は良くスケールするが，図 7 のようにノード数が多くなると，本実装の方がスケールしている．このことから，本実装はより大規模な環境でも並列化による高速化が行われると考えられる．

6.3 性能モデルとの比較

図 8 は今回立てたバイナリツリーを用いた性能モデルと実際の計算時間の比較である．タイルサイズを 40 から 800 まで変更した時の性能モデルと，タイルサイズ 400 の実際の計測時間である．計測した範囲内の性能モデルでは，タイルサイズ 400 が最適な値と判断できる．また，タイルサイズ 400 の性能モデルと実際の計算時間の比から，本性能

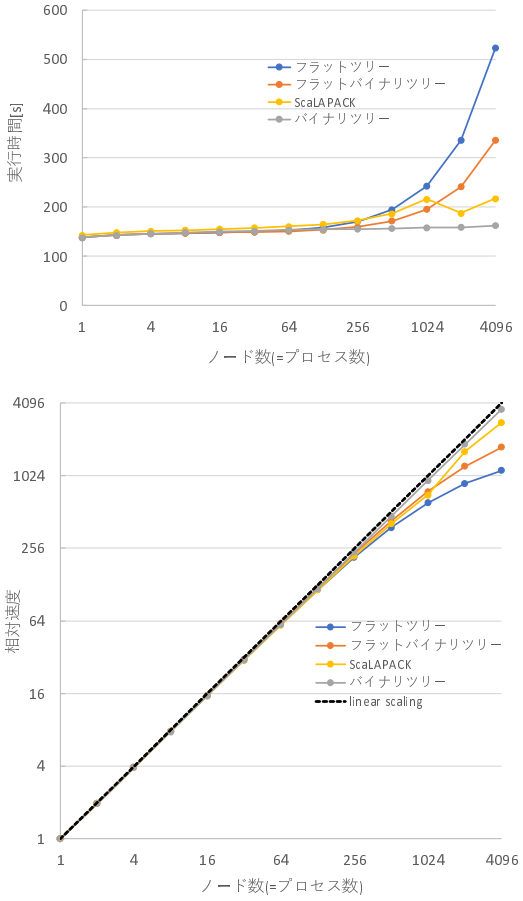

図 5 1 ノードあたり 64000 × 8000 の実行時間及び相対速度
Fig. 5 Performance result and relative rate of tile CAQR, on a 64000 × 8000 per node.

モデルでは約 90%の精度で実際の計算時間が求められる．今回，考慮に入れなかった GEQRT なども含めることができれば，性能モデルの実行時間の精度を上げることができると考えられる．

7. おわりに

行列分解におけるタイルアルゴリズムは，細粒度のタスクを大量に生成するため，近年のマルチコアアーキテクチャの並列性を活かすことができるアルゴリズムである．今回，タイル CAQR アルゴリズムの OpenMP，MPI によるハイブリッド実装を行い，クラスタシステム上での性能評価を行った．本報告では，CAQR アルゴリズムにおけるマージ処理のスケジューリング方式の比較および，ScaLAPACK による QR 分解実装との比較を行った．実験では，大規模並列環境ではタイル CAQR による実装の有用性が見られることがわかった．また，計算性能モデルをたて実際の計算時間との比較を行い，本モデルでは 90%の精度で実行時間の予測がたてられることが得られた．今後は，マージ実行時の同期を削除した動的スケジューリングの実装を検証する必要がある．

謝辞 本研究を遂行するにあたり，ご指導頂いた理化

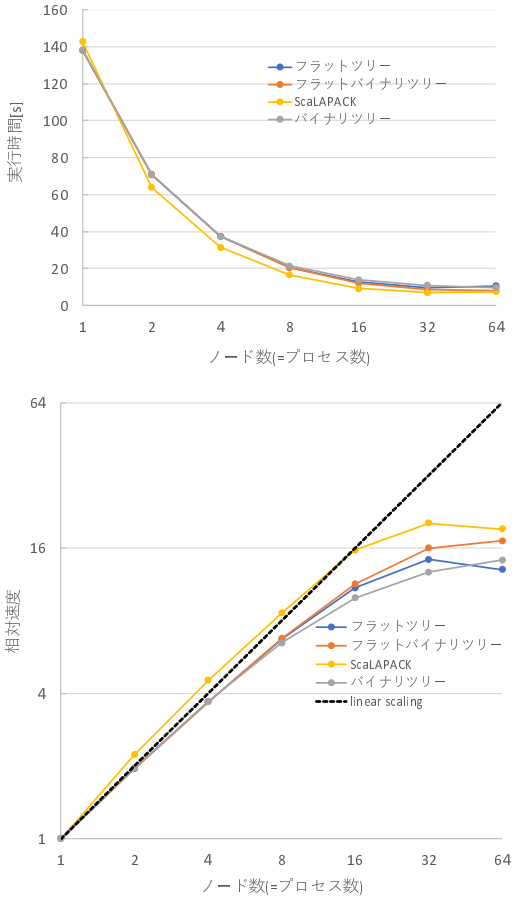

図 6　行列サイズ 64000 × 8000 の実行時間及び相対速度

Fig. 6　Performance result and relative rate of tile CAQR, on a 64000 × 8000 matrix.

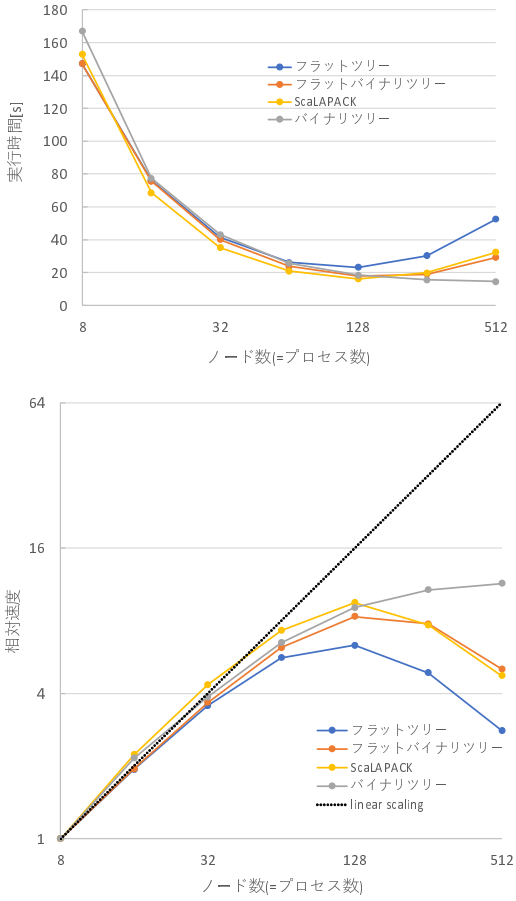

図 7　行列サイズ 512000 × 8000 の実行時間及び相対速度

Fig. 7　Performance result and relative rate of tile CAQR, on a 512000 × 8000 matrix.

学研究所 計算科学研究機構 大規模並列数値計算技術研究チームの今村俊幸チームリーダ，及び同チームの皆様に深く感謝いたします．

参考文献

[1] Dongarra, J. J.(ed.): *ScaLAPACK user's guide*, Society for Industrial and Applied Mathmatics, Philadelphia, PA, USA (1997)
[2] netlib: http://www.netlib.org/ (2017)
[3] Anderson, E., Bai,Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S. and Sorense, D.: *LAPACK's user's guide 3rd. edition*, SIAM, Philadelphia (1999).
[4] 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).
[5] Buttari, A., Langou, J., Kurzak, J. and Dongarra, J. J.: *A class of parallel tiled linear algebra algorithms for multicore architectures*, Parallel Computing, Vol. 35, pp. 38 53 (2009).
[6] 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).

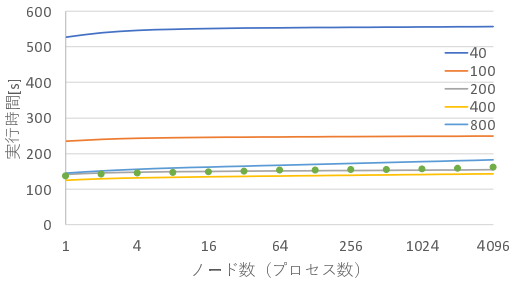

図 8　性能モデルと実行時間

Fig. 8　Performance model and execution time of tile CAQR decomposition.

[7] Schreiber, R., Loan, C. V.: *A storage-efficient WY representation for products of Householder transformations*, SIAM J. Sci. Statist. Comput., Vol. 10, No. 1, pp. 52 – 57 (1989).
[8] Kurzak, J., Ltaief, H. Dongarra, J., Badia, R. M.: *Scheduling Linear Algebra Operations on Multicore Processors.*, Technical Report, LAPACK Working note 213 (2009).
[9] Bosilca, G., Bouteiller, A., Danalis, A., Herault, T., Lemarinier, P., Dongarra, J.: *DAGuE: A generic distributed DAG engine for high performance computing.*, Technical Report, LAPACK Working note 231 (2010).
[10] Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M.,

Haidar, H., Herault, T., Kurzak, J., Langou, J., Lemarinier, P., Ltaief, H., Luszczek, P., YarKhan, A., Dongarra, J.: *Distributed-Memory Task Execution and Dependence Tracking within DAGuE and the DPLASMA Project.*, Technical Report, LAPACK Working note 232 (2010).

[11] PLASMA: https://bitbucket.org/icl/plasma/ (2017)

[12] Hadri, B., Ltaief, H., Agullo, E. and Dongarra, J.: *Enhancing Parallelism of Tile QR Factorization for Multicore Architectures*, Technical Report, LAPACK Working note 222 (2009).

[13] Dongarra, J., Faverge, M., Herault, T., Langou, J., Robert, Y.: *Hierarchical QR factorization algorithms for multi-core cluster systems*, Technical Report, Lapack Working note 257 (2011)