Vol.2014-HPC-146 No.15
2014/10/3

クラスタシステム向けタイルQR分解のタイルサイズチューニング

鈴木 智博[1,a)]

概要：タイルアルゴリズムは細粒度のタスクを多数生成できるので，マルチコア，メニーコアなどの高並列環境向け行列分解アルゴリズムとして注目されている．我々は Opteron プロセッサを搭載したクラスタシステムにタイル QR 分解を実装した．今回，この環境において実施したタイルサイズのチューニングについて報告する．

Tile Size Tuning for Tile QR Decomposition on Multi-Core Cluster Systems

Tomohiro Suzuki[1,a)]

Abstract: The tile algorithm for a matrix decomposition can generate a large amount of fine-grained tasks, so it is suited to recent multi-core/many-core architectures and attracts HPC communities attention. We implemented the tile QR decomposition algorithm on the cluster system of Opteron processors and the tile size was tuned to achieve high performance. In this report, we show the result of tile size tuning and the performance of the tile QR decomposition on the cluster system.

1. はじめに

密行列の数値線形代数において，さまざまな前処理に適用される行列分解は重要なアルゴリズムである．近年の科学技術計算の大規模化，高速化の要請に応えるために，マルチコア，メニーコアなど最新の計算資源を有効に活用できる行列分解アルゴリズムが求められている．

行列分解に対するタイルアルゴリズムは，行列を小行列に分割して処理することで細粒度のタスクを多数生成できるので，マルチコア，メニーコアなどの高並列環境向けアルゴリズムとして近年注目されている．我々はこれまでにQR 分解のタイルアルゴリズムをマルチコアクラスタシステムに実装した [15]．この実装は，OpenMP によるタスク並列プログラミングモデル，動的タスクスケジューリングなどの特徴を持つ．さらに，[17] においてノード間通信の改良を行い大幅な性能向上が得られた．

タイルアルゴリズムはタイルサイズを適切に選択することで，実行プロセス/スレッド数に対して十分な数のタスクを生成することができる．逆に，不適切なタイルサイズではプログラムの性能を発揮できないためチューニングが必要となる．今回，マルチコアクラスタ上に実装したタイル QR プログラムに対して実施したタイルサイズチューニングについて報告する．

2. タイル QR 分解

QR 分解は各種数値線形代数アルゴリズムの前処理などに多用される基本的かつ重要なアルゴリズムである．数値線形代数ライブラリ LAPACK[1] の QR 分解はブロックアルゴリズムを採用している．ブロックアルゴリズムは，行列をブロックサイズ (b) の縦ブロックに分割し，分解と後続行列更新を縦ブロック毎に繰り返すことで行列全体を分解する．後続行列更新に L3 BLAS 演算が使用できるので，L2 BLAS 演算を多用する逐次アルゴリズムと比較して高い性能を発揮することができる．

[1] 山梨大学大学院医学工学総合研究部
Interdisciplinary Graduate School of Medical and Engineering, University of Yamanashi, Takeda 4-3-11, Kofu, Yamanashi 400–8511, Japan
[a)] stomo@yamanashi.ac.jp

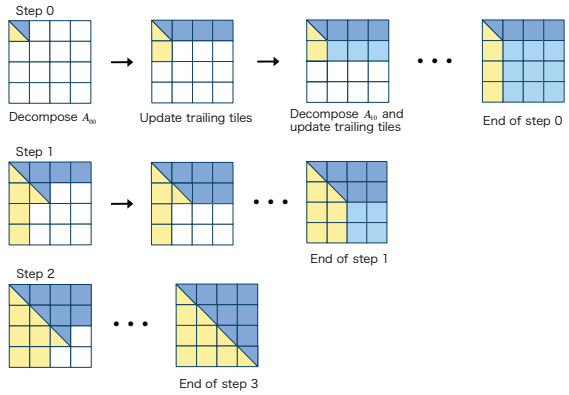

図 1　タイル QR 分解
Fig. 1　Tile QR decomposition

しかし，ブロックアルゴリズムは基本的には縦ブロックの分解と後続行列更新を順に繰り返す逐次アルゴリズムであり，fork-join 型の並列計算モデルである．fork-join 型計算モデルでは原理上必ずストールするスレッドが発生するため，近年の高並列な計算資源を効率的に利用することはできない．

行列分解に対するタイルアルゴリズム [3], [4], [5], [12] は対象とする行列を小行列（タイル）に分割し，分解，更新操作を 1 または 2 タイル毎に行う（図 **1**）．タイルサイズを適切に選択することで，実行プロセス/スレッド数に対して十分な数のタスクを生成することができるため，高並列な計算資源を有効に活用できる手法として近年注目されている．

$m \times n$ 行列 A の QR 分解は以下で定義される．

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

ただし，$m \geq n$，Q は $m \times m$ 直交行列，R は $m \times n$ 上三角行列である．以下では，タイルサイズを $b \times b$ (正方タイル) とし，行列 A は $p \times q$ 個のタイル $A_{ij} (i = 0, \ldots, p-1, j = 0, \ldots, q-1)$ から成るものとする．ただし，$p = \lceil m/b \rceil$，$q = \lceil n/b \rceil$ とする．また，各タイル内でデータが連続となるよう，タイル内で要素は列優先方式で配置され，それらのタイルが列優先方式で配置されている Column-Column Rectangular Block (CCRB) レイアウト [9] を使用を仮定する．

2.1　タイル QR 分解のカーネル

タイル QR 分解は以下の 4 つの基本計算（カーネル）で構成される．

- `GEQRT` : 対角タイル $A_{kk}\,(k = 0, \ldots, q-1)$ の QR 分解を行ない，上三角行列 R_{kk} を生成する．変換行列は，Householder 変換に基づく compact-WY 記法 [14] を使用し，単位下三角行列 V_{kk} と上三角行列 T_{kk} に格納される．各小行列の関係を以下に示す．

$$R_{kk} \leftarrow (I - V_{kk} T_{kk}^{\mathrm{T}} V_{kk}^{\mathrm{T}})\, A_{kk}$$

  R_{kk}，V_{kk} は A_{kk} の領域に上書きされる（ただし，V_{kk} の対角要素 1 は保存しない）．T_{kk} を保存するために別の領域が必要である．

- `TSQRT` : 上三角行列 $R_{kk}\,(k = 0, \ldots, p-1)$ とその下方にあるタイル $A_{ik}\,(0 \leq k < i < q)$ の組に対して QR 分解を行い，R_{kk} を更新する．変換行列は正方行列 V_{ik}，上三角行列 T_{ik} に格納される．各小行列の関係を以下に示す．

$$\begin{bmatrix} R_{kk} \\ 0 \end{bmatrix} \leftarrow \left[I - \begin{bmatrix} I \\ V_{ik} \end{bmatrix} T_{ik}^{\mathrm{T}} \begin{bmatrix} I & V_{ik}^{T} \end{bmatrix} \right] \begin{bmatrix} R_{kk} \\ A_{ik} \end{bmatrix}$$

  変換行列 V_{ik} は A_{ik} の領域に上書きされ，T_{ik} は別に確保した領域に保存される．
  `TSQRT` カーネルのように，QR 分解済みの行列の上部または下部に要素を付加した行列の QR 分解は updating QR 分解 [8] と呼ばれる．

- `LARFB` : `GEQRT` によって生成された変換行列 V_{kk}，T_{kk} をその右側タイル $A_{kj}\,(0 \leq k < j < q)$ に適用する．

$$A_{kj} \leftarrow (I - V_{kk} T_{kk}^{\mathrm{T}} V_{kk}^{T})\, A_{kj}$$

- `SSRFB` : `TSQRT` によって生成された変換行列 V_{ik}，T_{ik} をその右側タイル A_{kj}，$A_{ij}\,(0 \leq k < i < p, 0 \leq k < j < q)$ に適用する．

$$\begin{bmatrix} A_{kj} \\ A_{ij} \end{bmatrix} \leftarrow \left[I - \begin{bmatrix} I \\ V_{ik} \end{bmatrix} T_{ik}^{\mathrm{T}} \begin{bmatrix} I & V_{ik}^{T} \end{bmatrix} \right] \begin{bmatrix} A_{kj} \\ A_{ij} \end{bmatrix}$$

`GEQRT` と `TSQRT` を分解カーネル，`LARFB` と `SSRFB` を更新カーネルと呼ぶ．

2.2　カーネル実行の依存性

我々はタイル QR 分解を，前項で示した 4 つのカーネル実行をタスクとしたタスク並列モデルとして実装する．タイルの上から下を “i 方向”，左から右を “j 方向”，図 1 のステップ数に相当する方向を “k 方向” として，タイル QR 分解のタスクには 3 方向の依存関係が存在する．

同一タイル列の分解または更新カーネルは上から順に実行しなければならない．この分解カーネルと更新カーネルの i 方向の逐次性を i 方向依存と呼ぶ．

同一タイル行の更新カーネルは並列に実行することができるが，分解カーネルが変換行列を生成した後でなければ実行できない．これを j 方向依存と呼ぶ．

ステップ 0 を除いて，ステップ k のすべてのカーネルは，ステップ $k-1$ における同一タイル上のカーネルの終了後でなければ実行できない．これを k 方向依存と呼ぶ．

2.3　動的スケジューリング

タイルアルゴリズムにはタスク実行の依存性を考慮した

いくつかのスケジューリング方法が存在する [10], [11]．3 方向すべての依存性がなく実行可能であるタスクが実行待ちとなることを極力排除するために，我々はタイル QR 分解のタスクスケジューリング方式として動的スケジューリング [16] を採用する．動的スケジューリングとは，プロセス/スレッドがタスクキューから実行可能タスクの情報を取り出し，これを実行する方式であり，静的なスケジューリング方式に比べて最適に近いスケジューリングを行える可能性が高い．依存性の有無は，タイルの位置に対応するプログレステーブルで管理する．

動的スケジューリングにおける各プロセス/スレッドの動作は以下となる．

(1) タスクキューからデータの取り出し
(2) タスク（カーネル）実行
(3) プログレステーブル更新
(4) プログレステーブルの確認（依存性の確認）
(5) 実行可能なタスクをタスクキューへ投入

タイルアルゴリズムでは，タイルの位置 $\{i,j,k\}$ と，そこで実行されるカーネルが一意に対応するので，タスクキューで受け渡されるデータはタイルの位置とする．

タスクの依存性の有無は，タイルの位置 $\{i,j,k\}$ に対応したプログレステーブルにその進捗を記録することで判定することができる．タスクを実行したプロセス/スレッドは，その完了によって解消される依存性をただちにプログレステーブルに記録し，すべての方向依存が解消された位置 $\{i,j,k\}$ をタスクキューに投入する．

3. クラスタシステム向け実装

ここではタイル QR 分解のクラスタシステム向け実装 [15], [17] について述べる．実装の基本方針を以下に示す．

- タイルサイズをブロックサイズとした縦方向一次元ブロックサイクリックデータ分散
- 1 ノード 1 MPI プロセス
- OpenMP によるワークシェアリング
  - 1 ノード t スレッド
  - 通信スレッド (thread id (tid) = 0)
  - 計算スレッド (tid = $1 \cdots t-1$)
    - 動的タスクスケジューリング

クラスタシステムに対するタイル QR 分解において縦方向一次元ブロックサイクリックデータ分散を使用した場合，タイル列分解を担当するノードが生成する変換行列を他のノードへ送信するブロードキャスト通信が発生する．[15] では第 k タイル列の分解を担当するノードは，i 方向すべての分解カーネルが終了するまで送信元のままであり，他のノードからの送信は行われなかった．しかし，第 k タイル列の分解が完了する前に第 k+1 タイル列の一部の分解タスクは実行可能である．第 k タイル列の分解がある程度進んだ後では，第 k+1 タイル列を担当するノードは送信可能な変換行列データを持つが，送信できずにいる状況にある．

[17] ではブロードキャストすべきデータを持ったノードが随時通信を行えるよう改良が実施された．これにより，依存性の解消によって実行可能となる更新カーネルのタスクが増大し，大幅な性能向上が得られた．

4. タイルサイズチューニング

4.1 タイルサイズ

ブロックアルゴリズムにおいてブロックサイズが重要な性能チューニングパラメータであるのと同様に，タイルアルゴリズムにおいてはタイルサイズが性能に大きな影響を及ぼす．

正方行列 ($m = n$) のタイル QR 分解における各カーネルの実行回数は以下のとおり．

$$\texttt{GEQRT} \ : \ \sum_{k=0}^{p-1} 1 = p \tag{2}$$

$$\texttt{TSQRT} \ : \ \sum_{k=1}^{p-1} (p-k) = \frac{1}{2}(p-1) \tag{3}$$

$$\texttt{LARFB} \ : \ \sum_{k=1}^{p-1} (p-k) = \frac{1}{2}(p-1) \tag{4}$$

$$\texttt{SSRFB} \ : \ \sum_{k=1}^{p-1} (p-k)^2 = \frac{1}{6}p(2p^2-3p+1) \tag{5}$$

$$\text{Total} \ : \ \frac{1}{6}p(2p^2+3p+1) \tag{6}$$

ここで，SSRFB カーネルの実行回数の割合 (5)/(6) を α とする．例えば，$m = n = 20000$, $b = 500$ のとき $\alpha = 0.953$ である．次節の実験で示すとおり，最適なタイルサイズにおける α は常に高い値となり，SSRFB カーネルがタイル QR 分解で支配的であることが分かる．ここで「最適な」とは「測定範囲内でもっとも実行時間が短い」という意味で使用している．各カーネルの演算量は，$b^3/3$ を単位として，その重みが GEQRT : 4，TSQRT : 6，LARFB : 6，SSRFB : 12 となる [7]．α が高い値のときには SSRFB カーネルは演算量の面からも支配的である．

SSRFB カーネルは L3 BLAS ルーチンが主要演算であるから，データサイズが大きいほど性能が向上するが，タイルサイズが大きくなると α は小さくなる．つまり，SSRFB カーネルが支配的である程度にはタイルサイズは小さくなければならない．さらに，タイルアルゴリズムにおいて，タイルサイズはタスクの粒度と等価であるから，大きなタイルサイズは低い並列性と負荷不均衡をもたらす．このことから，実行スレッド数，ノード数もタイルサイズの最適化に影響することが分かる．

4.2 内部ブロックサイズ

タイル QR 分解の分解カーネルにブロックアルゴリズムを適用することを内部ブロック化と呼ぶ．内部ブロック化されたタイルアルゴリズムでは，ブロックアルゴリズム同様に内部ブロックサイズもチューニング要素となる．

内部ブロック化によりタイル QR 分解の全体性能（速度）が 15%程度向上することが経験的に知られている．前述のとおり，タイル QR 分解では SSRFB カーネルが支配的であるから，性能向上は分解カーネルではなく，SSRFB カーネルの速度向上に起因する．内部ブロック化により，一つの TRMM が複数の TRMM と GEMM に分割される．一般的に TRMM よりも GEMM は高速であるから，内部ブロック化による SSRFB カーネルの速度向上は GEMM の高速性によるものと考えられる．

5. 数値実験

タイルサイズ，内部ブロックサイズのタイル QR 分解プログラムの性能への影響を調べるために，数値実験を行った．表 **1** に使用した京都大学学術情報メディアセンター Cray XE6 の概要を示す．我々はこのシステムの最大 8 ノードを使用してプログラムを実行した．

表 **1** Cray XE6 の概要（ 1 ノード）
Table 1 Cray XE6 specifications (single node)

ノード	プロセッサ数	2
	メモリ	DDR3-1600 64GB
プロセッサ	プロセッサ名	AMD Opteron 6380
	クロック	2.5GHz
	コア数	16

表 **2** に使用したコンパイラ，ライブラリを示す．ノード

表 **2** 開発環境
Table 2 Compiler and Libraries

コンパイラ	Cray C コンパイラ 8.2.2
BLAS/LAPACK	Cray LibSci 12.1.3
MPI	Cray MPICH 6.3.0

あたりのスレッド数は $t = 32$ とした．また，コンパイルオプションを `-omp -O2`，プログラム実行時のオプションを `numactl -i all` とした．

5.1 最適パラメータ探索 1

Cray XE6 8 ノードを使用して，タイルサイズ (b)，内部ブロックサイズ (s) をパラメータとして，正方行列に対するタイル QR 分解の速度を測定した．表 **3** の範囲でパラメータを変化させ，各行列サイズで最速となった b，s を図 **2** に示す．

図 2 では，最適なタイルサイズにおける α の値も示し

表 **3** パラメータ
Table 3 Parameter list

$m = n$	7680, 15360, 23040, 30720, 38400, 46080, 53760, 61440, 69120
b	128, 256, 384, 512, 640, 768
s	32, 64, 128, 256

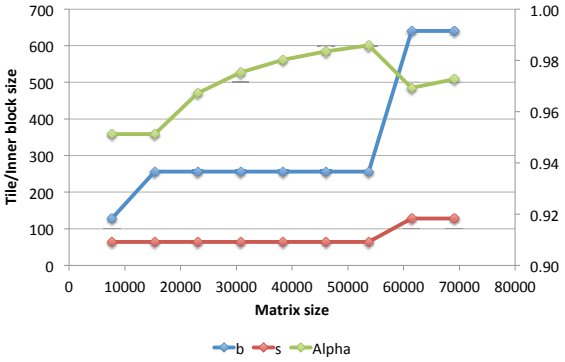

図 **2** 最適パラメータ
Fig. 2 Optimal parameters

ている（目盛りは右側）．前節で言及したように，最適なタイルサイズにおける SSRFB カーネルの実行回数の割合は 0.95 から 0.99 の高い値を示している．

パラメータ空間を粗く探索しているため，最適パラメータが連続的に変化するのかどうか判定できないが，大きな行列サイズに対しては大きなタイルサイズが最適となり，タイルサイズの大きさに応じて内部ブロックサイズも大きくなることは図 2 から分かる．以降では，内部ブロックサイズはタイルサイズに比例するものと仮定し，その係数を 0.2 とする．

5.2 最適パラメータ探索 2

次に実行スレッド数とパラメータの関係を見るために，1，2，4，8 ノードにおいて，タイル QR 分解の速度を測定した（図 **3**）．各図において，タイルサイズを $b = 80, 160, 320, 640, 1280$ とした 5 本の曲線が示されている．このとき内部ブロックサイズはそれぞれ $s = 16, 32, 64, 128, 256$ である．

これらの図から，前節同様に大きな行列サイズに対しては大きなタイルサイズが最適となること，また，ノード数が多い方が最適なタイルサイズが小さいことが分かる．ノード数 = スレッド数の増大に伴って，ストールするスレッドが少なくなるような，または，負荷不均衡が小さくなるようなタイルサイズが最適値として得られていることが分かる．

図 3 において，最適なタイルサイズが 160 から 320 になる行列サイズ，320 から 640 になる行列サイズを横軸に，ノード数を縦軸にプロットしたものが図 **4** である．ただ

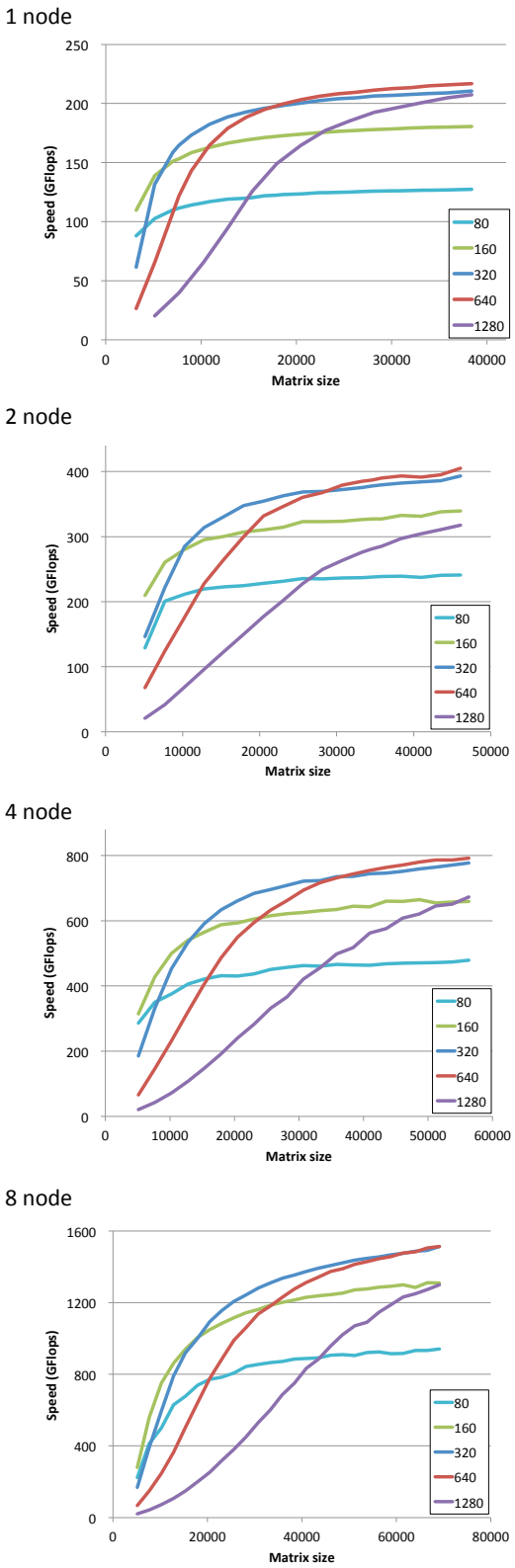

図 3　タイル QR 分解の実行時間
Fig. 3　Performance result of tile QR

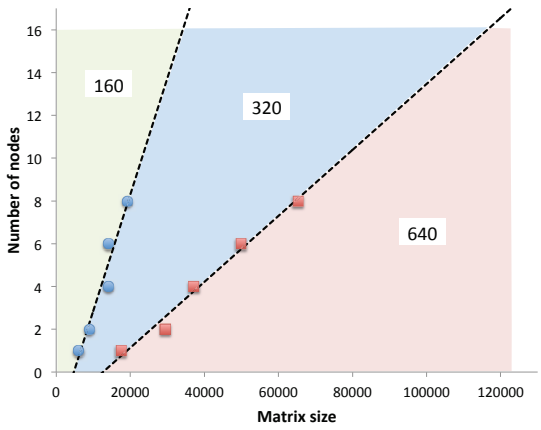

図 4　ノード数と最適タイルサイズの関係
Fig. 4　Relationship between number of nodes and optimal tile size

し，同図ではノード数 6 のデータが追加されている．図 4 から，ノード数が多ければ広い範囲でタイルサイズを一定の値としてよいこと，逆に，ノード数が少なければ行列サイズに応じてタイルサイズを変更する必要があることが分かる．また，実験の範囲では，タイルサイズが変化する行列サイズとノード数に線形性が見られる．図中の点線は 4 点から最小二乗法で得られた一次式である．

6. 関連研究

この節では本研究に関連する他の研究について言及する．

テネシー大学で開発された PLASMA[5], [13] は，タイルアルゴリズムを採用した共有メモリ環境向け密行列数値線形代数ライブラリである．本研究の実装では内部ブロック化された PLASMA のタイル QR 分解カーネルを使用している．PLASMA のタスクは QUARK と呼ばれるスケジューラにより管理される．QUARK スケジューラは逐次的記述のアルゴリズムから依存関係を解析し，タスクを動的にスケジューリングする．分散メモリ環境向けのタイルアルゴリズムライブラリには DPLASMA[2] がある．

我々の実装のスレッド並列化基盤が OpenMP なのに対し，PLASMA では POSIX threads (pthreads) である．標準スレッドライブラリとして長い歴史を持つ pthreads であるが，その素朴な API を駆使した並列プログラム開発は容易ではない．近年のクラスタシステム用の実装では，本研究と同様に，ノード内を OpenMP，ノード間を MPI で並列化するのが定石となっている．

2 節では，カーネル実行の i 方向依存について述べたが，$m \gg n$ の縦長行列ではこの依存性（逐次性）が致命的な欠点となる．縦長行列向けの QR 分解アルゴリズムに Tall Skinny QR (**TSQR**) がある [6]．TSQR では，縦長行列を縦方向に分割して 1 本のタイル列を構成し，はじめに，すべてのタイルを `GEQRT` カーネルで QR 分解する．生成さ

れた上三角行列をマージしながら，つまり，２つの上三角行列を updating QR 分解しながらタイル列全体の QR 分解を行う．はじめの分解カーネルと，その後のマージ操作の一部は並列に実行できるので i 方向の逐次性は大きく緩和される．

Communication Avoiding QR (**CAQR**)[6] は，行列を縦方向に複数の “ドメイン” に分割し，各ドメイン内で QR 分解を実行したのち，結果をドメイン間でマージするものであり，TSQR を一般化したものと考えることができる．行列をドメインに分割したことで，マージ操作とこれに伴う後続タイル更新操作の計算量が増加するが，これによる速度低下を i 方向の並列化で補うことができる．本論文では正方行列を扱っているので，j 方向に十分な数の並列タスクを生成することが可能であること，動的スケジューリングを採用していることとノード間通信を改良したことにより i 方向の逐次性が性能のボトルネックとはなっていないものと考えられる．

7. おわりに

行列分解に対するタイルアルゴリズムは細粒度のタスクを多く生成することで，高い並列性を持つ最近のマルチコア, メニーコアアーキテクチャの性能を発揮できるアルゴリズムとして注目されている．今回，タイルサイズ，内部ブロックサイズ，ノード数を性能パラメータとして, 実行時間の意味で最適なパラメータの探索を行った．内部ブロックサイズはタイルサイズに比例するという仮定の下で行った実験では，行列サイズの増加に伴って最適タイルサイズも大きくなること，今回の実験の範囲では，最適タイルサイズが変化する行列サイズとノード数に比例関係が見られることが分かった．

今後は，より多くのノードでの実行においても今回の傾向が見られるかを検証する必要がある．

謝辞　本研究は京都大学学術情報メディアセンターのスーパーコンピュータを利用して実施したものであり，同センターの「平成 25 年度プログラム高度化支援事業」による支援を受けた．また，本研究は JSPS 科研費 26400197 の助成を受けた．ここに記して謝意を表す．

参考文献

[1] 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).

[2] Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Haidar, H., Herault, T., Kurzak, J., Langou, J., Lemarinier, P., Ltaief, H., Luszczek, P., YarKhan, A. and Dongarra, J.: Distributed-Memory Task Execution and Dependence Tracking within DAGuE and the DPLASMA Project, Technical report, LAPACK Working Note 232 (2010).

[3] Bosilca, G., Bouteiller, A., Danalis, A., Herault, T., Lemarinier, P. and Dongarra, J.: DAGuE: A generic distributed DAG engine for high performance computing, Technical Report 231, LAPACK Working Note 231 (2010).

[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).

[7] Dongarra, J., Faverge, M., Herault, T., Langou, J. and Robert, Y.: Hierarchical QR factorization algorithms for multi-core cluster systems, Technical report, LAPACK Working Note 257 (2011).

[8] Golub, G. H. and Loan, C. F. V.: *Matrix Computations (3rd Ed.)*, Johns Hopkins University Press (1996).

[9] Gustavson, F., Karlsson, L. and Kågström, B.: Parallel and Cache-Efficient In-Place Matrix Storage Format Conversion, *ACM Trans. Math. Softw.*, Vol. 38, No. 3, pp. 17:1–17:32 (2012).

[10] Kurzak, J., Ltaief, H., Dongarra, J. and Badia, R. M.: Scheduling Linear Algebra Operations on Multicore Processors, *Concurrency and Computation: Practice and Experience*, Vol. 21(1), pp. 15 – 44 (2009).

[11] Kurzak, J., Buttari, A. and Dongarra, J.: Solving systems of linear equations on the CELL processor using Cholesky factorization, Technical report, Trans. Parallel Distrib. Syst (2007).

[12] Kurzak, J. and Dongarra, J. J.: QR Factorization for the CELL Processor, *Scientific Programming, Special Issue: High Performance Computing with the Cell Broadband Engine*, Vol. 17, No. 1-2, pp. 31 – 42 (2009).

[13] PLASMA: http://icl.cs.utk.edu/plasma/ (2013).

[14] Schreiber, R. and 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).

[15] Suzuki, T. and Miyashita, H.: OpenMP/MPI implementation of tile QR factorization on T2K open supercomputer, *Proceedings of IEEE 7th International Symposium on Embedded Multicore/Many-core SoCs (MCSoC-13), Special Session on Auto-Tuning for Multicore and GPU (ATMG)* (2013).

[16] 鈴木智博：タイルアルゴリズムのための動的スケジューラの OpenMP 実装，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)，No. 2013-HPC-139(13), pp. 1 – 6 (2013).

[17] 鈴木智博：クラスタシステム向けタイル QR 分解のノード間通信の改良，（投稿中）(2014).