Vol.2009-HPC-121 No.18
2009/8/5

ブロックハウスホルダー QR 分解の並列計算における自動チューニング手法の検討

深谷 猛[†1] 山本有作[†1] 張 紹良[†1]

行列計算を並列化する場合，行列ベクトル積や行列乗算などの BLAS ルーチンを並列化する方法と，それらのルーチンをコールする階層で並列化する方法が考えられる．また，行列をブロックに分割して計算を行うことが一般的となっている．そのため，ユーザーは並列化方法とブロック分割法の両者のチューニングを行う必要があるが，自由度が非常に大きいため，効果的なチューニングをすることが難しい．そこで，本稿ではハウスホルダー QR 分解を対象として，自動チューニング手法の検討を行う．

A Study on Automatic Tuning for Parallel Computation of the Blocked Housseholder QR Decomposition

TAKESHI FUKAYA,[†1] YUSAKU YAMAMOTO[†1]
AND SHAO-LIANG ZHANG[†1]

In matrix computation, we can parallelize an algorithm by two ways: parallelization of BLAS routines such as matrix-vector multiplication, and parallelization in algorithm levels where BLAS routines are called. In addition, blocking techniques are widely used for matrix computations. Therefore we have many choices when tuning our programs for parallel computers. But it is very difficult for general users to tune their programs effectively. In this paper, we discuss an approach to automatic tuning the algorithm of the blocked Householder QR decomposition.

†1 名古屋大学大学院工学研究科計算理工学専攻
Department of Computational Science and Engineering, Graduate School of Engineering, Nagoya University

1. はじめに

近年，計算機の高性能化とともにアーキテクチャの複雑化が著しく，高性能計算を行うには対象の計算機に合わせてプログラムをチューニングすることが必須となっている．しかし，性能パラメータの組合せが膨大であったり，ハードウェアに関する専門知識や十分な経験が必要であったりと，一般のユーザが自力で効果的なチューニングを行うことは困難である．そこで，性能チューニング機構付きのソフトウェアの開発や汎用的なチューニング技術の研究が注目されている．

著者らは，基本的な行列計算の一つである QR 分解を対象にして，動的計画法を用いた機械的な性能チューニング手法の開発を行ってきた.[1)] 先の研究では主にシングルコア上での計算を対象としていたが，その続きとして，本稿では共有メモリ型並列計算機上のでの並列計算のための自動チューニング手法の提案と性能評価を行う．

行列計算を並列化する場合，一般的に，行列ベクトル積や行列乗算などの基本ルーチン（以下，BLAS：Basic Linear Algebra Subprograms）[2)] レベルでの並列化と，それらのルーチンをコールする上位階層レベル（アルゴリズムレベル）での並列化が考えられる．アルゴリズムレベルでは粗粒度の並列化が可能で，近年，TSQR アルゴリズムなどの AllReduce アルゴリズム[3)4)] が注目されている．このように，並列計算の場合，シングルコア上での計算に比べて選択肢が膨大となり，人手によるチューニングはより一層困難となるため，自動チューニングの必要性が増す．

本稿では，基本的な行列計算の一つであるハウスホルダー QR 分解[5)] の並列計算を対象とした自動チューニング手法の検討を行う．ハウスホルダー QR 分解を並列化する場合，行列を列方向にブロック分割することが一般的である．このとき，縦長の部分行列の QR 分解を繰り返し行うことになるが，その際に並列版の BLAS を用いることと TSQR アルゴリズムで並列化することが可能である．したがって，それらの二種類の並列化方法の選択と，それを考慮してのブロック分割方法の選択をチューニングとして行うことになる．そのようなチューニングを機械的に行うことを目指して，本稿では，まずアルゴリズムを一般化して，チューニングすべきパラメータを明らかにする．次に，パラメータの決定問題をある種の最適化問題として定式化する．その後，実用面を考慮して最適化問題の解法を考える．

Vol.2009-HPC-121 No.18
2009/8/5

2. ブロックハウスホルダー QR 分解と並列計算

2.1 ハウスホルダー QR 分解

任意の $m \times n$ $(m \geq n)$ 行列 A を，$m \times m$ の直交行列 Q と $m \times n$ の上三角行列 R を用いて，

$$A \rightarrow QR,$$

と分解することは QR 分解と呼ばれ[5]，最小二乗問題[6] や情報検索[7] などの様々な問題に用いられる基本的な行列分解計算の一つである．

QR 分解の数値計算では，計算精度の面からハウスホルダー QR 分解[5] が用いられることが多い．この手法では，スカラー t とベクトル $\boldsymbol{y}$ を用いて

$$H := I - t\boldsymbol{y}\boldsymbol{y}^T,$$

と書かれるハウスホルダー変換を

$$H_n \cdots H_2 H_1 A \rightarrow R,$$

と繰り返し A に作用させ R を求める．また同時に，Q はハウスホルダー変換の積として，

$$Q = H_1 H_2 \cdots H_n,$$

と，陰的に求められる．

2.2 ブロックハウスホルダー QR 分解

ハウスホルダー変換を行列に作用させる計算は，行列ベクトル積などの Level-2 BLAS[2] で実行されるため，メモリアクセスのボトルネックの影響が大きく，計算機の性能を十分に引き出すことが困難である．そのため，ブロック化[5] と呼ばれる手法を用いて実装することが一般的である．ブロック化されたハウスホルダー QR 分解（ブロックハウスホルダー QR 分解）では，

(1) 行列を列方向にブロック分割する．
(2) ブロック内のみで QR 分解を計算する．
(3) ブロック内の分解で得られた複数個のハウスホルダー変換を行列の形にまとめる．
(4) 行列形式で他のブロックの計算を行う．

と，いう流れで計算を進める．複数個のハウスホルダー変換を行列の形にまとめるには，余分な計算コストが生じるが，行列形式で他のブロックの計算を行う際に，行列乗算などの Level-3 BLAS[2] の利用が可能になる．このトレードオフを考慮して，適切なブロック分割を行うと，ブロック化をしない場合に比べて，非常に高速に計算することができる．

ブロックハウスホルダー QR 分解アルゴリズムの例として，LAPACK[8] のハウスホルダー

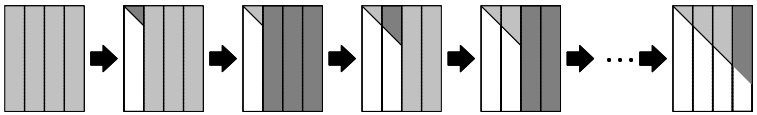

図 1 ブロックハウスホルダー QR 分解

Algorithm 1 Blocked Householder QR(A, Q, R, b)

Input: 分解対象の行列 A，ブロック幅 b

Output: $A = QR$ を満たす直交行列 Q，上三角行列 R.

1: $A \rightarrow [A_1|A_2|\cdots|A_{\mathrm{NB}}]$. // A を幅 b で NB 個のブロックに分割
2: **for** $i = 1$ to NB **do**
3: $A_i \rightarrow Q_i R_i$. // 通常の QR 分解
4: $Q_i^T [A_{i+1}|\cdots|A_{\mathrm{NB}}] \rightarrow [A_{i+1}|\cdots|A_{\mathrm{NB}}]$.
5: **end for**
6: $R = [R_1|R_2|\cdots|R_{\mathrm{NB}}]$.
7: $Q = Q_1 Q_2 \cdots Q_{\mathrm{NB}}$.

QR 分解ルーチン（DGEQRF）などで広く用いらている，固定幅でブロック化[5][8] したアルゴリズムを Algorithm 1 に示す．また，このアルゴリズム全体のイメージ図を**図 1** に示す．図中で色の濃い部分が更新された部分を意味している．また，これ以外にも，再帰的にブロック分割[9] を行う手法や，異なるブロック幅を用いてブロック分割する手法[10] などもある．

2.3 TSQR 分解

近年，非常に縦長 $(m \gg n)$ の行列に対して，行列を再帰的に行方向に分割して計算を行う TSQR（Tall Skinny QR）アルゴリズム[3][4][11][12] が注目されている．このアルゴリズムを Algorithm 2 に示す．また，再帰レベルが 2 の場合のイメージを**図 2** に示す．TSQR アルゴリズム内では，上三角行列を二つ縦に並べた特殊な構造の行列の QR 分解があり，得られる直交行列（˜をつけて区別している）も特殊な構造を持つことが知られている．また，TSQR アルゴリズムにより得られた Q を使って，行列を更新する（以後，TS-Update と呼ぶ）計算は，**図 3** のように行われる．

このアルゴリズムでは，再帰の各レベルの QR 分解や Q の作用の計算を，それぞれ完全に独立して行うことが可能となっている．ただし，アルゴリズム全体の計算量は，従来のものよりも増加している．

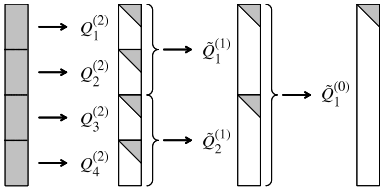

図 2 TSQR

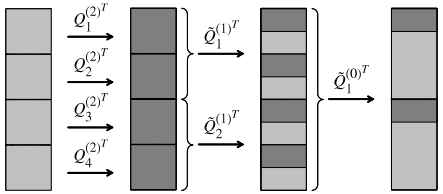

図 3 TSQR に伴う行列の更新（TS-Update）

Algorithm 2 TSQR(A, Q, R, d)

Input: 分解対象の行列 A，再帰レベル d

Output: $A = QR$ を満たす直交行列 Q，上三角行列 R.

1: **if** $d = 0$ **then**
2: $A \to QR$. // 通常の QR 分解
3: **else**
4: $A \to \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$ // 行方向に分割
5: TSQR($A_1, Q_1, R_1, d-1$). // 再帰呼び出し
6: TSQR($A_2, Q_2, R_2, d-1$). // 再帰呼び出し
7: $\begin{bmatrix} R_1 \\ R_2 \end{bmatrix} \to \tilde{Q}R$.
8: $Q = \begin{bmatrix} Q_1 & O \\ O & Q_2 \end{bmatrix} \tilde{Q}$.
9: **end if**

2.4 並 列 計 算

ハウスホルダー QR 分解の並列計算について考える．

QR 分解や LU 分解などの行列計算のほとんどは，行列ベクトル積や行列乗算などの BLAS ルーチンから成り立っている．そのため，BLAS ルーチンを並列化することが第一に挙げられる．特に，Level-3 BLAS のルーチンは効果的に並列化されていることが多く，これを用いるのは非常に実用的である．ハウスホルダー QR 分解の場合，すでに述べたように，列方向にブロック分割することで，行列乗算の利用が可能である．

一方で，TSQR アルゴリズムでは，縦長の行列を行方向にブロック分割し，小さいサイズの QR 分解を並列計算することが可能である．これは，BLAS レベルで並列化するよりも粗粒度の並列計算となる．したがって，行列を列方向に分割すると縦長の行列が生じるので，この部分に対して TSQR アルゴリズムを適用するのは，有効な並列化の手法となり得る．

表 1 計算機環境

項目	条件
CPU	Intel Xeon X5355 (2.66GHz) 8CPU 搭載
メモリ	16GB
OS	Red Hat Linux
BLAS	Intel Math Kernel Library ver. 9.0
コンパイラ	Intel icc ver. 9.1 （オプション -O3）

つまり，ハウスホルダー QR 分解を並列化する場合，

- 並列化された BLAS を使う，
- TSQR アルゴリズムを使う，

という二種類の選択肢がある．これらを計算機環境と行列サイズに応じて，適切に使い分けることで，より効率的に並列計算を行うことが期待できる．しかし，これらの手法の使い分けと，それを考慮した行列のブロック分割法の決定をユーザーの手で行うことは非常に難しい．

3. 数 値 実 験

前節で述べたように，ハウスホルダー変換の並列計算では，列方向のブロック分割方法と各ブロックの計算の並列化方法（並列版 BLAS の利用，または TSQR）を変えることができるので，これによって計算時間が変化することが予想できる．そこで，本節では数値実験を通して，性能変化を確認する．

3.1 実験方法と計算環境

サイズの異なる，$[-0.5, 0.5]$ の乱数を要素とする行列に対して，列方向のブロック幅を $b = 32, 64, 128, 256, 512$ と変えて，固定幅でブロック分割し，すべてのブロックの計算を (1) 並列化された BLAS を利用，(2)TSQR アルゴリズムを利用，して計算した場合の計算時間を測定する．なお，TSQR アルゴリズムの再帰レベルは全 CPU を使う値とした．

利用した計算環境は**表 1** に示した通りである．なお，TSQR アルゴリズムは OpenMP を用いて並列実装した．CPU 数が 8 なので，TSQR の再帰レベルは 3 とした．

3.2 実 験 結 果

実験結果を**表 2** に示す．表中で，BLAS は並列版 BLAS を用いた場合の，TSQR は TSQR アルゴリズムを用いた場合の実行時間である．また，各サイズにおいて最も短い場合を太字にしている．

表 2 QR 分解の実行時間（秒）の変化

	ブロック幅	32	64	128	256	512	1024
8000 × 1000	BLAS	2.59	1.84	1.30	1.26	1.47	1.48
	TSQR	0.99	0.75	**0.71**	0.81	0.98	–
8000 × 2000	BLAS	10.1	6.01	3.86	3.39	3.30	3.60
	TSQR	4.44	3.07	**2.74**	3.03	3.82	–
8000 × 4000	BLAS	35.2	20.8	12.3	10.0	9.02	**8.83**
	TSQR	17.8	11.8	10.6	11.2	14.4	–
8000 × 8000	BLAS	116	64.3	39.6	27.7	23.3	**21.4**
	TSQR	62.0	42.9	36.1	38.7	44.9	–
16000 × 1000	BLAS	5.28	3.81	2.91	2.77	2.99	3.54
	TSQR	1.91	1.31	**1.18**	1.27	1.49	1.58
16000 × 2000	BLAS	21.3	12.6	8.90	7.74	7.12	7.18
	TSQR	8.65	5.57	**4.53**	4.61	5.41	6.48
16000 × 4000	BLAS	78.9	46.6	28.3	22.2	19.4	19.8
	TSQR	35.8	22.3	**17.3**	**17.3**	20.2	25.1
16000 × 8000	BLAS	295	163	98.9	71.6	61.1	**55.5**
	TSQR	134	83.0	64.8	62.7	74.3	93.7
32000 × 1000	BLAS	12.4	8.01	6.50	5.96	6.23	7.59
	TSQR	3.96	2.63	**2.28**	2.38	2.73	2.98
32000 × 2000	BLAS	47.8	28.5	19.4	16.2	16.1	16.9
	TSQR	17.9	11.0	8.77	**8.40**	9.23	10.7
32000 × 4000	BLAS	177	104	66.5	48.9	45.8	42.6
	TSQR	75.5	44.2	33.1	**30.9**	33.8	38.7
32000 × 8000	BLAS	694	385	220	168	139	126
	TSQR	292	170	125	**112**	124	145

3.3 考　　察

表 2 の結果から分かるように，行列サイズによって計算時間が最少になる場合のブロック幅と並列化方法は異なっている．全体の傾向としては，行列が正方行列に近づく（m を固定した場合，n が大きくなる）ほど，ブロック幅を大きくして並列版の BLAS を用いる場合が性能が良くなっている．逆に，行列が縦長（$m \gg n$）になるほど，TSQR が効果的となっている．

4. 自動チューニング手法の検討

本節では，ハウスホルダー QR 分解の並列計算のための自動チューニング手法の検討を行う．具体的には，まず並列化された BLAS を用いる場合と TSQR アルゴリズムを用いる場合の両者を含むように，アルゴリズムを一般化し，チューニングすべきパラメータを明らかにする．その後，パラメータの決定問題をある種の最適化問題に定式化する．そして，最適化問題の解放について考える．

Algorithm 3 Blocked Householder QR with TSQR(A, Q, R, B, D)

Input: 分解対象の行列 A，ブロック幅 B，TSQR の再帰レベル D.

Output: $A = QR$ を満たす直交行列 Q，上三角行列 R.

1: $A \rightarrow [A_1|A_2|\cdots|A_{\mathrm{NB}}]$.　// A を B に従って NB 個のブロックに分割.
2: **for** $i = 1$ to NB **do**
3:　TSQR(A_i, Q_i, R_i, d_i).
4:　TS-Update($\bar{A}, Q_i, d_i$).　// $\bar{A} := [A_{i+1}|\cdots|A_{\mathrm{NB}}]$.
5: **end for**
6: $R = [R_1|R_2|\cdots|R_{\mathrm{NB}}]$.
7: $Q = Q_1 Q_2 \cdots Q_{\mathrm{NB}}$.

4.1 アルゴリズムとパラメータ

列方向のブロック分割と TSQR アルゴリズムを併用したアルゴリズムの一般形とチューニングすべきパラメータについて説明する．

列方向のブロック分割では，各ブロックのブロック幅を等しくする必要性はない．そのため，列方向のブロック分割を決定するためのパラメータは，

$$B := \{b_1, b_2, \cdots, b_{\mathrm{NB}}\} \qquad (s.t.\ 0 < b_i < n,\ b_1 + b_2 + \cdots + b_{\mathrm{NB}} = n),$$

といった数列の形で表現される．ここで，b_i は i 番目のブロックの幅，NB はブロックの総数である．なお，ブロック分割をしないアルゴリズムは $b_1 = b_2 = \cdots = b_n = 1$ の場合に相当する．

一方，各ブロックの QR 分解には TSQR アルゴリズムの適用が可能であるが，再帰レベルは各ブロックに対して個別に設定できる．そのため，TSQR アルゴリズムの再帰レベルも，

$$D := \{d_1, d_2, \cdots, d_{\mathrm{NB}}\} \qquad (s.t.\ 0 \le d_i \le d_{\max}),$$

といった数列の形で表現される．なお，TSQR アルゴリズムを用いない場合は再帰レベルが 0 の場合に相当する．また，再帰レベルの上限 $d_{\max}$ は使用可能な CPU 数と行列サイズによって決定される．

この二種類のパラメータを用いて，列方向のブロック分割と TSQR アルゴリズムを併用したアルゴリズムの一般形を Algorithm 3 に示す．

4.2 定 式 化

前節で述べたパラメータの決定問題の定式化を行う．

まず，

- $T_{\text{total}}(m,n,B,D)$：$m \times n$ 行列に対してパラメータ B, D で Algorithm 3 を実行する時間，
- $T_{\text{tsqr}}(m,n,d)$：$m \times n$ 行列に対して再帰レベル d の TSQR アルゴリズムを実行する時間，
- $T_{\text{tsup}}(m,l,n,d)$：$m \times n$ 行列に対して，幅 l の行列の QR 分解で得られた Q の作用を，再帰レベル d の TS-Update アルゴリズムで実行する時間，

と定義する．

このとき，パラメータの決定問題は，

$$T^{\text{best}}_{\text{total}}(m,n) = \min_{B,D} T_{\text{total}}(m,n,B,D), \tag{1}$$

と定式化できる．

一方で，

$$\begin{aligned} T_{\text{total}}(m,n,B,D) &= \sum_{i=1}^{\text{NB}} \Big\{ T_{\text{tsqr}}(m_i,b_i,d_i) + T_{\text{tsup}}(m_i,n_i,b_i,d_i) \Big\} \\ &= T_{\text{tsqr}}(m,b_1,d_1) + T_{\text{tsup}}(m,n-b_1,b_1,d_1) \\ &\quad + T_{\text{total}}(m-b_1,n-b_1,B',D'), \end{aligned}$$

と書ける．ただし，$B' := \{b_2,\cdots,b_{\text{NB}}\}, D' := \{d_2,\cdots,d_{\text{NB}}\}$ である．また，実装時は値が更新される部分のみを計算するため，行列のサイズが小さくなっていくので m_i, n_i という表記をした．

このことから，式 (1) は，

$$\begin{aligned} \min_{B,D} T_{\text{total}}(m,n,B,D) &= \min_{b_1} \Big[\min_{d_1} \big\{ T_{\text{tsqr}}(m,b_1,d_1) + T_{\text{tsup}}(m,b_1,n-b_1,d_1) \big\} \\ &\qquad + \min_{B',D'} T_{\text{total}}(m-b_1,n-b_1,B',D') \Big] \\ &= \min_{l} \Big[T^{\text{best}}_{\text{ts}}(m,l,n-l) + T^{\text{best}}_{\text{total}}(m-l,n-l) \Big], \end{aligned}$$

と変形できるので，最終的に式 (1) は，

$$T^{\text{best}}_{\text{total}}(m,n) = \min_{l} \Big[T^{\text{best}}_{\text{ts}}(m,l,n-l) + T^{\text{best}}_{\text{total}}(m-l,n-l) \Big], \tag{2}$$

と，ベルマン方程式[13] と呼ばれる形に帰着できる．なお，

$$T^{\text{best}}_{\text{ts}}(m,l,n) = \min_{d} \Big\{ T_{\text{tsqr}}(m,l,d) + T_{\text{tsup}}(m,l,n,d) \Big\}, \tag{3}$$

とした．

以上のように，パラメータの決定は式 (3) と式 (2) で表わされる二段階の最適化問題として定式化ができる．

4.3 解　　法

以下では，使用可能な CPU 数を 2^{NP}，対象とする行列の最大のサイズを $M \times N$ $(M \geq N)$ とする．

第 一 段 階

式 (3) の最適化問題の解法について述べる．

まず，式 (3) の最適化問題を解く際に必要となる，T_{tsqr} と T_{tsup} の値の求め方について述べる．

- $T_{\text{qr}}(m,n,p)$：$m \times n$ 行列の QR 分解を p 並列の BLAS を使って実行する時間，
- $T_{\text{spqr}}(n,p)$：幅 n の TSQR 内で生じる特殊な構造（上三角行列を縦に二つ並べた）の行列の QR 分解を p 並列の BLAS を使って実行する時間，
- $T_{\text{up}}(m,l,n,p)$：幅 l の行列の QR 分解で得られた Q による，$m \times n$ 行列の更新を p 並列の BLAS で実行する時間，
- $T_{\text{spup}}(l,n,p)$：幅 l の特殊な構造の行列の QR 分解で得られた $\tilde{Q}$ による，幅 n の行列の更新を p 並列の BLAS で実行する時間，

といった値を導入して，

$$T_{\text{tsqr}}(m,l,d) = T_{\text{qr}}(\frac{m}{2^d},l,2^{\text{NP}-d}) + \sum_{i=0}^{d-1} T_{\text{spqr}}(l,2^{\text{NP}-i}),$$

$$T_{\text{tsup}}(m,l,n,d) = T_{\text{up}}(\frac{m}{2^d},l,n,2^{\text{NP}-d}) + \sum_{i=0}^{d-1} T_{\text{spup}}(l,n,2^{\text{NP}-i}),$$

といった形で，T_{tsqr} と T_{tsup} を書き下す．ここでは，TSQR アルゴリズムの各レベルでブロック（レベル d のとき 2^d 個）に全 CPU を均等に分配して，各ブロック内では分配された個数で BLAS を並列実行することで，全 CPU を使うことにしている．

すると，T_{up} と T_{spup} の値は，DGEMM [⋆1]や DTRMM [⋆2]といった複数個の BLAS ルーチンの実行時間の和である．したがって，BLAS の性能予測モデルを用いることで予測可能である．また，T_{qr} と T_{spqr} の値も，サンプルデータからの補間や，さらに細分化して BLAS の実行時間の和に帰着させることなどにより予測可能である．したがって，これらの四種類の値を予測し，それらを足し合わせることで T_{tsqr} と T_{tsup} の値の予測が可能である．

次に，予測した T_{tsqr} と T_{tsup} の値を用いて，式 (3) の最適化問題を解くことを考える．今回は，TSQR アルゴリズムの性質からパラメータ d の範囲は，

$$0 \leq d \leq \min\left(\text{NP}, \log_2 \frac{m}{l}\right), \tag{4}$$

とする．これより，

$$1 \leq n \leq N, \quad n \leq m \leq M, \quad 1 \leq l \leq n,$$

を満たす (n, m, l) の全ての組合せについて，個別に，(4) の範囲の各 d について，$T_{\text{tsqr}} + T_{\text{tsup}}$ の予測値を計算し，最小化する d とそのときの $T_{\text{ts}}^{\text{best}}[m, l, n]$ を求めれば良い．このコストは，使用可能な CPU 数が高々，数十から数百程度であることを考慮すれば，

$$O(MN^2),$$

と見積もることができる．実際は，m や n を一定のサイズで離散化したり，関数近似などを用いることで，よりコストを抑えて解くことが考えられる．

第 二 段 階

式 (3) の最適化問題の結果を使って，式 (2) の最適化問題の解法について述べる．目的関数が式 (2) のような形の場合，動的計画法[13] を用いて解くことが可能である．そのアルゴリズムを Algorithm 4 に示す．このアルゴリズムでは，式 (3) の最適化問題の結果と最適化済みのサイズの小さい場合の結果を使って，順にサイズの大きな場合の最適化を行う．

Algorithm 4 の実行コストは，

$$O(MN^2),$$

と見積もれる．今回も第一段階と同様，m や n を一定サイズで離散化することで，コストを抑えることが考えられる．

4.4 実装に向けて

今回，検討した方法を実装する場合の手順の一例を挙げる．まず，事前に，

⋆1 一般行列同士の積を計算する BLAS ルーチン
⋆2 一般行列と三角行列の積を計算する BLAS ルーチン

Algorithm 4 動的計画法

Input: $M, N, T_{\text{ts}}^{\text{best}}$
Output: $B[m, n]$

```
for m = 1 to M do
    0 → T_total^best[m, 0].
end for
for n = 1 to N do
    for m = n to M do
        ∞ → T_total^best[m, n].
        for l = 1 to n do
            T_ts^best[m, l, n − l] + T_total^best[m − l, n − l] → t.
            if t < T_total^best[m, n] then
                t → T_total^best[m, n].
                l → B[m, n].
            end if
        end for
    end for
end for
```

(1) 性能予測モデルのために，BLAS ルーチンなどの実行時間のサンプリングを行う．
(2) 式 (3) の最適化問題を解き，TSQR の再帰レベルに関するデータベース（$D[m, l, n]$）を構築する．
(3) 式 (2) の最適化問題を解き，列方向のブロック分割に関するデータベース（$B[m, n]$）を構築する．

といったことを行う．

そして，実際に QR 分解の計算を実行する場合は，

(1)
- $b_1 = B[m, n]$,
- $b_2 = B[m - b_1, n - b_1]$,
- $b_3 = B[m - (b_1 + b_2), n - (b_1 + b_2)]$,
- ⋮

(2)
- $d_1 = D[m, b_1, n - b_1]$,
- $d_2 = D[m - b_1, b_2, n - (b_1 + b_2)]$,
- $d_3 = D[m - (b_1 + b_2), b_3, n - (b_1 + b_2 + b_3)]$

$\vdots$

と，繰り返しそれぞれのデータベースを参照してパラメータを決定し，Algorithm 3 を使って計算する．

5. おわりに

本稿では，ブロックハウスホルダー QR 分解を共有メモリ型の並列計算機を用いて並列計算する場合の自動チューニング手法について検討した．アルゴリズムを並列化する場合，列方向のブロック分割を行い，各ブロック計算を BLAS レベルで並列化をする方法と，TSQR アルゴリズムを用いて並列化する方法があるが，数値実験からも分かるように，行列サイズによって適切なブロック幅と適切な並列化方法は異なっている．

そこで，本稿では自動チューニング手法の検討として，

(1) 二つの並列化方法を含む形でのアルゴリズムの一般化とパラメータの設定
(2) パラメータの決定問題を最適化問題として定式化
(3) 最適化問題の解法の検討

を行った．

本稿での検討に基づいた具体的な実装例と性能評価の結果については研究会の当日に報告する．

謝辞 日頃からご議論いただいている名古屋大学大学院工学研究科計算理工学専攻の張研究室の皆様，自動チューニングの研究に関して有益な助言をくださった自動チューニング研究会の皆様に感謝いたします．なお，本研究は科学研究費補助金特定領域研究「i-explosion」（課題番号 21013014），科学研究費補助金基盤研究（B）（課題番号 21300013），科学研究費補助金基盤研究（A）（課題番号 20246027），および科学研究費補助金基盤研究（C）（課題番号 21560065）の補助を受けています．

参考文献

1) Fukaya, T., Yamamoto, Y. and Zhang, S., L.: A Dynamic Programming Approach to Optimizing the Blocking Strategy for the Householder QR Decomposition, *Proceedings of IEEE Cluster 2008*, pp.402–410 (2008).
2) Dongarra, J., DuCroz, J., Hammarling, S. and Duff, I.: A Set of Level 3 Basic Linear Algebra Subprograms, *ACM Transactions on Mathematical Software (TOMS)*, Vol.16, pp.1–17 (1990).
3) Langou, J.: AllReduce algorithms: application to Householder QR factorization, *Proceedings of the 2007 International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applicatipns*, pp.103–106 (2007).
4) Demmel, J., Grigori, L., Hoemmen, M. and Langou, J.: Communication-avoiding parallel and sequential QR factorizations, Technical report, Electrical Engineering and Computer Sciences, University of California Berkeley (2008).
5) Golub, G. and VanLoan, C.: *Matrix Computations*, Johns Hopkins University Press, 3 edition (1996).
6) 森　正武：数値解析，共立出版株式会社，2 edition (2002).
7) Demmel, J.W.: *Applied Numerical Linear Algebra*, SIAM, Philadelphia (1997).
8) Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S. and Sorensen, D.: *LAPACK User's Guide*, SIAM (1992).
9) Elmroth, E. and Gustavson, F.: Applying Recursion to Serial and Parallel QR Factorization Leads to Better Performance, *IBM Journal of Research and Development*, Vol.44, p.605 (2000).
10) Bischof, C. and Lacroute, P.: An Adaptive Blocking Strategy for Matrix Factorizations, *Proceedings of the Joint International Conference on Vector and Parallel Processing*, pp. 210–221 (1990).
11) 森　大介，山本有作，張　紹良：ハウスホルダー QR 分解のための AllReduce アルゴリズムの性能と精度，情報処理学会研究報告　2008-HPC-117，Vol.2008, No.99, pp. 25–29 (2008).
12) 村上　弘：ハウスホルダ型直交変換算法の多段階化によるブロック化と並列分散処理，情報処理学会研究報告　2008-HPC-117，Vol.2008, No.99, pp.19–24 (2008).
13) Bellman, R.: *Dynamic Programming*, Dover Publications (2003).