Vol.2010-HPC-126 No.33
2010/8/4

密行列計算アルゴリズムに対するブロック分割法の最適化と性能評価

深谷 猛[†1] 山本有作[†2] 張 紹良[†1]

高性能な行列計算を行う場合，プログラムの性能チューニングが必要不可欠である．我々は基本的な密行列計算が BLAS ルーチンを使って実行される点に着目し，チューニング済みの BLAS ルーチンを効率的に使えるようにプログラムをチューニングすることを目指す．ブロック化されたアルゴリズムにおいて，効率的に BLAS を使うためには行列のブロック分割法を最適化することが重要となる．本稿では，LU 分解のアルゴリズムをブロック化して，ブロック分割法が性能に与える影響を評価し，さらに適切な分割法を決定するための手法の検討を行う．

Perforamnce Optimization and Evaluation of the Blocking Strategy for the Dense Matrix COmputations

TAKESHI FUKAYA,[†1] YUSAKU YAMAMOTO[†2] AND SHAO-LIANG ZHANG[†1]

For high performance matrix computations, it is necessity to tune the software. Since basic dense matrix computations consist almost entirely of the BLAS routines, it is important how to tune programs for exploiting the peak performance of optimized BLAS routines. In blocked algorithm, this means how to optimize the partitioning of the target matrix. In this paper, we evaluate and discuss the blocking strategy for the blocked LU 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
†2 神戸大学大学院システム情報学研究科計算科学専攻
Department of Computational Science, Graduate School of System Informatics, Kobe University

1. はじめに

計算機の多様化・複雑化に伴い，高性能計算を行うためにはソフトウェアの性能チューニングが必要不可欠となっている．そこで，近年，我々は LU 分解や QR 分解といった基本的な密行列計算アルゴリズムに対して，その階層構造に着目した性能チューニング手法の研究を行っている[1)]．上記のような密行列計算は BLAS と呼ばれる線形計算の基本ルーチンを繰り返し使用して計算を進めるため，この基本ルーチンのチューニングをまず行い，次にそれらをコールする上位階層のチューニングを行うことで効率良くチューニングを行うことが期待できる．

すでに BLAS ルーチンのチューニングに関しては様々な先行研究[2)] が存在するので，我々は BLAS ルーチンをコールする上位階層のチューニング手法に重点をおいて研究を進めている．つまり，既にチューニングされた BLAS が提供されている場合に，その性能が十分発揮されるようにプログラムの上位階層をチューニングすることを目指す．

BLAS に基づいた行列計算ではブロック化[3)] と呼ばれる手法を用いることが一般的となっている．ブロック化では計算対象の行列をブロックに分割して，ブロック単位で計算を進める．このとき，ブロックの大きさによってコールされる BLAS ルーチンの性能が変化するため，使用する BLAS の性能特性を考慮したブロック分割を行うことが重要となる．

本稿では，基本的な行列計算の一つである，LU 分解のアルゴリズムをブロック化し，数値実験を通してブロック分割法と性能の関係を評価する．さらに，適切なブロック分割法の決定方法として，動的計画法を利用した方法を紹介する．

2. BLAS に基づく密行列計算と性能チューニング

LU 分解や QR 分解などの基本的な密行列計算アルゴリズムの大部分を占めているのは，ベクトルの内積や行列ベクトル積，あるいは行列乗算といった基本的な演算である．これらの基本演算は，BLAS（Basic Linear Algebra Subprograms）[4)] によって標準インターフェースが定められているため，BLAS のインターフェースに合わせてプログラムを作成することで，移植性の高いプログラムを効率的に作成することができる．

さらにこの構造を利用することでプログラムの性能チューニングも効率的に行うことができる．まず最初に BLAS ルーチンの性能チューニングを行い，次にチューニングされた BLAS を効率的に使えるように上位階層のプログラムのチューニングを行う．このようにチューニングを行うことで，

- BLAS ルーチンは様々な行列計算で使われるため，チューニングのコストが大きくなってもあまり問題にならない．
- 上位階層のチューニングでは，コールする BLAS ルーチンの性能特性のみを考慮すれば良い（キャッシュメモリのサイズなどのアーキテクチャの特性を直接考慮する必要がない）．

といったメリットが生じる．このチューニング方法で得られる性能は，基本演算から上位階層までの全てを同時にチューニングして得られる性能よりは劣るかもしれないが，階層ごとにチューニングを行うので，機械的なチューニングを比較的容易に実現できる期待がある．

すでに BLAS のチューニングの部分では，ATLAS[2] などの自動チューニング機構を備えた BLAS がすでにあり，一方で，新しいハードウェアが登場する場合は GOTO BLAS や Intel Math Kernel Library や CUBLAS などのチューニング済みの BLAS ライブラリが早期に提供されることが多い．このように，新しい計算環境が登場した際に，まず BLAS のチューニングがなされるという傾向を踏まえて，我々は提供された BLAS をより効率的に利用できるようにプログラムの上位階層をチューニングする手法の開発を目指している．

3. 効率的な BLAS の利用方法

本節では，チューニング済みの BLAS ルーチンを効率的に利用するための方針について述べる．前節で述べたように，BLAS は線形計算における基本演算をサブルーチン化したものであるが，それらは演算の種類によって以下の三種類に分類されている．[4]

- Level-1 BLAS : ベクトルの内積など，データの再利用性なし．
- Level-2 BLAS : 行列ベクトル積など，ベクトルの再利用は可能だが，行列の再利用は不可能．
- Level-3 BLAS : 行列乗算など，行列の再利用が可能．

近年の計算機では演算性能に対してメモリアクセス性能が相対的に劣るため，データの再利用性が高くキャッシュメモリなどを効率的に使うことができる Level-3 BLAS をできるだけ使って行列計算を行うことが重要となっている．代表的な Level-3 BLAS のルーチンとしては，行列 A, B の積 $C = \alpha AB + \beta C$ を計算する DGEMM（単精度の場合は SGEMM）があり，実装の工夫次第では計算機のピーク性能に近い性能を出すことが可能となっている．

このように BLAS を効率的に使って高性能計算を行うためには，まず Level-3 BLAS のルーチン計算の中心にすることが必要不可欠である．そこで，基本的な行列計算の教科書に載っている行列ベクトル積などの Level-2 BLAS のルーチンが中心のアルゴリズムを，ブロック化と呼ばれる手法を用いて，Level-3 BLAS を中心としたものに再構成することが一般的となっている．

ブロック版アルゴリズムでは，計算対象の行列を部分行列（以下，ブロックと呼ぶ）に分割して，ブロック単位で計算を進めることになるが，分割したブロックの大きさによって性能が異なってくる．これは，一般的に Level-3 BLAS のルーチンの性能は計算対象の行列のサイズによって大きく変化し，行列サイズが大きくなるにつれて性能が向上する傾向があるためである．また，QR 分解などの一部の計算ではブロック化に伴う演算量の増加があり，BLAS の性能と演算量の増加のトレードオフが生じることもある．

つまり，効率的に BLAS を用いて行列計算を行うためには，まずアルゴリズムを Level-3 BLAS が中心になるように再構成し，さらに，Level-3 BLAS の性能特性を考慮して，行列を適切な大きさのブロックに分割していくことが重要となっている．

4. LU 分解アルゴリズムのブロック化

本節では，本稿で対象とする部分軸選択付き LU 分解[5] のアルゴリズムについて説明する．与えられた $n \times n$ の正則行列 A に対して部分軸選択を行い，

$$A = PLU$$

となる下三角行列 L と上三角行列 U を求めることを考える．ここで P は置換行列とする．また，L の対角は 1 とする．

ブロック化により DGEMM を中心として計算を行う方法として，本稿では Algorithm 1 に示すような再帰的アルゴリズムを考える．このアルゴリズムは，再帰の各段階におけるブロック幅 l を任意に設定できるようになっているため，自由度の高いブロック分割が可能となっている．

アルゴリズム中で U_{12} を求める際に三角行列行列を係数とする線形方程式を解く必要があるが，この部分の計算もブロック化をすることで DGEMM を使うことが可能である．このアルゴリズムを Algorithm 2 に示す．このアルゴリズムでも再帰の各段階におけるブロック幅 l が任意に設定できるため，Algorithm 1 と同様に自由度の高いブロック分割が可能となっている．

5. 数 値 実 験

本節では，前節で紹介したアルゴリズムの性能がブロック幅によってどのように変化するのか，数値実験を通して確認する．

Vol.2010-HPC-126 No.33
2010/8/4

表 1 計算機環境

項目	環境 1	環境 2
CPU	Intel Xeon X5355 (2.66GHz) 1 コア使用	Intel Core i7 (2.67GHz) 1 コア使用
BLAS	Intel Math Kernel Library ver. 9.0	Intel Math Kernel Library ver. 11.0
コンパイラ	Intel icc ver. 9.1 （オプション -O3）	gcc ver. 4.1.2 （オプション -O3）

5.1 実験方法と計算環境

LU 分解（Algorithm 1）と TRSM（Algorithm 2）をそれぞれ実装し，乱数行列に対してブロック幅を変化させて計算を行い，それぞれ性能を測定した．両者ともブロック幅は d を定数として，

$$l = \begin{cases} d & (n > d) \\ n/2 & (n \leq d) \end{cases}$$

と与え，d の値は $32, 64, 128, 256, 512, 1024$(LU 分解のみ), ∞ とした．

使用した計算環境は**表 1** に示した通りである．

5.2 実 験 結 果

環境 1 と 2 において，行列サイズを 4096 と 8192 にして，ブロック幅の決定方法を変えて LU 分解の実効性能を測定した結果を**表 2** に示す．

5.3 考　　察

表 2 を見ると，まず，全てのケースで LU 分解におけるブロック幅の決定方法が重要となっていることが分かる．また，LU 分解のブロック幅が小さい場合は，TRSM のブロック幅の影響がほとんどない結果となっている．これは，LU 分解のブロック幅が小さいと，LU 分解の全体の演算量に対して TRSM が占める割合が非常に小さくなるため，TRSM のブロック幅の違いによる性能差の影響が無視できるからだと思われる．逆に，LU 分解で大きなブロック幅を採用する必要がある場合は，TRSM の演算量が LU 分解全体に対して占める割合も大きくなるため，TRSM のブロック幅も適切に設定することが必要になると考えられる．

6. 動的計画法によるブロック分割法の決定

前節の数値実験の結果からも分かるように，行列の分割法によって LU 分解の実効性能が異なってくる．そこで，本節では実際に Algorithm 1 と Algorithm 2 におけるブロック幅を決定するための方法について検討する．

Algorithm 1 および Algorithm 2 では，再帰の各段階で行列が任意のブロック幅で二分割されていく．したがって，**図 1** に示したような二分木によって分割法を表現することができる．

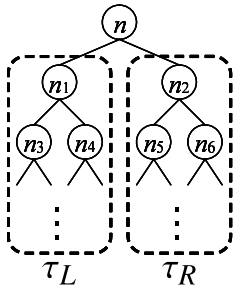

図 1 二分木による分割法の表現

ここで，τ を二分木，$\mathfrak{T}_n$ を葉の数が n の二分木の集合として，

- $T_{\mathrm{LU}}(m, n, \tau)$: $m \times n$ 行列を Algorithm 1 で τ に従ってブロック分割をして LU 分解する時間.
- $T_{\mathrm{TRSM}}(n, k, \tau)$: $n \times n$ 行列 L と $n \times k$ 行列 B に対して Algorithm 2 で τ に従ってブロック分割をして X を計算する時間.

とする．さらに，

$$T^*_{\mathrm{LU}}(m, n) = \min_{\tau \in \mathfrak{T}_n} T_{\mathrm{LU}}(m, n, \tau),$$
$$T^*_{\mathrm{TRSM}}(n, k) = \min_{\tau \in \mathfrak{T}_n} T_{\mathrm{TRSM}}(n, k, \tau)$$

とする．

すると，まず，Algorithm 2 より

$$\begin{aligned} T^*_{\mathrm{TRSM}}(n, k) &= \min_{\tau \in \mathfrak{T}_n} T_{\mathrm{TRSM}}(n, k, \tau) \\ &= \min_{\tau \in \mathfrak{T}_n} \{T_{\mathrm{TRSM}}(n_1, k, \tau_L) + T_{\mathrm{DGEMM}}(n_1, n_2, k) + T_{\mathrm{TRSM}}(n_2, k, \tau_R)\} \\ &= \min_{1 \leq n_1 \leq n-1} \{ \min_{\tau \in \mathfrak{T}_{n_1}} T_{\mathrm{TRSM}}(n_1, k, \tau) + \min_{\tau \in \mathfrak{T}_{n_2}} T_{\mathrm{TRSM}}(n_2, k, \tau) + T_{\mathrm{DGEMM}}(n_1, n_2, k)\} \end{aligned}$$

となり，最終的に

$$T^*_{\mathrm{TRSM}}(n, k) = \min_{1 \leq l \leq n-1} \{T^*_{\mathrm{TRSM}}(l, k) + T^*_{\mathrm{TRSM}}(n - l, k) + T_{\mathrm{DGEMM}}(l, n - l, k)\} \qquad (1)$$

という形の再帰方程式を得ることができる．なお，$\tau_{L(R)}$ は図 1 にあるように左（右）の部

分木に対応している．また，T_{DGEMM} は DGEMM の実行時間である．

同様の手順で，$T^*_{\mathrm{LU}}(m,n)$ についても，

$$T^*_{\mathrm{LU}}(m,n) = \min_{1\le l\le n-1}\{T^*_{\mathrm{LU}}(m,l)+T^*_{\mathrm{LU}}(m-l,n-l)+T^*_{\mathrm{TRSM}}(l,n-1)+T_{\mathrm{DGEMM}}(m-l,n-l,l)\} \quad (2)$$

という再帰方程式を得ることができる．

式 (1) および式 (2) はベルマン方程式[6] と呼ばれる再帰方程式で，動的計画法を用いて多項式時間で解けることが知られている．動的計画法では，問題の小さい場合（今回の場合は $n = 1$）から順番に結果を再利用して解を求めていくことになる．したがって，まず式 (1) に対して動的計画法を実行して $T^*_{\mathrm{TRSM}}(n,k)$ を求める．次にその結果も再利用しながら式 (2) を解いて $T^*_{\mathrm{LU}}(m,n)$ を順番に求めることになる．なお，両方の計算において，T_{DGEMM} の値は BLAS ルーチンの性能予測モデルを利用することになるので，適切な分割法を動的計画法を解いて得るためには，できるだけ精度の良い BLAS の性能予測モデルの構築が重要な課題となる．

7. おわりに

今回扱った LU 分解などの基本的な行列計算アルゴリズムはブロック化をされることが一般的だが，その際に使用する BLAS ルーチンの性能が十分でるようなブロック分割法を採用することが重要となる．LU 分解をはじめ多くの行列計算には再帰構造が存在していて，その構造に着目した形でブロック分割を考えると，動的計画法に基づいてブロック分割法の決定を行うことが可能となる．

本稿で説明した動的計画法の具体的なアルゴリズムと，そこから得られたブロック分割法の性能評価については当日に報告する．

謝辞 日頃からご議論頂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頂いている自動チューニング研究会の皆様に感謝いたします．なお，本研究は日本学術振興会特別研究員奨励費および科学研究費補助金を補助を受けている．

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) Whaley, R., Petotet, A. and Dongarra, J.: Automated empirical optimizations of software and theATLAS project, *Parallel Computings*, pp.27:3–35 (2001).
3) Dongarra, J., Duff, I., Sorensen, D. and van der Vorst, H.: *Numerical Linear Algebra for High-Performance Computers*, SIAM, 2 edition (1998).
4) 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).
5) Golub, G. and VanLoan, C.: *Matrix Computations*, Johns Hopkins University Press, 3 edition (1996).
6) Bellman, R.: *Dynamic Programming*, Dover Publications (2003).

Vol.2010-HPC-126 No.33
2010/8/4

Algorithm 1 Recursive Blocked LU decomposition

Input: $m \times n$ 行列 A.

Output: $A = PLU$ を満たす m 次置換行列 P，$m \times n$ 下三角行列 L(対角は 1), $n \times n$ 上三角行列 U.

1: **if** $n = 1$ **then**

2: $A = (a_1, \ldots, a_m)^T$ として，その中で絶対値最大の要素 a_p を探す.

3: 1 行目と p 行目を交換する置換行列を P として,

$$A = P(1, a_2/a_p, \ldots, a_1/a_p, \ldots, a_m/a_p)^T a_p.$$

4: **else**

5: A を次のように分割する.

$$A \to \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}.$$

ここで，$A_{11} : l \times l,\ A_{12} : l \times (n-l),\ A_{21} : (m-l) \times l,\ A_{22} : (m-l) \times (n-l)$.

6: アルゴリズムを再帰的に適用して前半ブロックの LU 分解を計算する.

$$\begin{pmatrix} A_{11} \\ A_{21} \end{pmatrix} \to P_1 \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} U_{11}.$$

7: 後半ブロックに対して置換を作用させる.

$$P_1^{-1} \begin{pmatrix} A_{12} \\ A_{22} \end{pmatrix} \to \begin{pmatrix} A'_{12} \\ A'_{22} \end{pmatrix}.$$

8: 前進代入により U_{12} を求める.

$$L_{11}^{-1} A'_{12} \to U_{12}.$$

9: A'_{22} の更新を行う (DGEMM).

$$A'_{22} - L_{21} U_{12} \to A''_{22}.$$

10: アルゴリズムを再帰的に適用して A''_{22} の LU 分解を計算する.

$$A''_{22} \to P_2 L_{22} U_{22}.$$

ここで，P_2 は $(m-l) \times (m-l)$ の置換行列.

11: 以前の前半ブロックに対して置換を作用させる.

$$P_2^{-1} L_{21} \to L'_{21}.$$

12: 求めるべき P, L, U は以下の通りになる.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = P_1 \begin{pmatrix} I^{(l)} & O \\ O & P_2 \end{pmatrix} \begin{pmatrix} L_{11} & O \\ L'_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ O & U_{22} \end{pmatrix}.$$

ここで，$I^{(l)}$ は l 次の単位行列.

13: **end if**

Algorithm 2 Recursive Blocked TRSM

Input: $n \times n$ 下三角行列 L(対角は 1), $n \times k$ 行列 B.

Output: $LX = B$ を満たす $n \times k$ 行列 X.

1: **if** $n = 1$ **then**

2:

$$X = (x_1, \ldots, x_k)^T = (b_1, \ldots, b_k)^T (= B).$$

3: **else**

4: L, X, B を次のように分割する.

$$L \to \begin{pmatrix} L_{11} & O \\ L_{21} & L_{22} \end{pmatrix}, \quad X \to \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}, \quad B \to \begin{pmatrix} B_1 \\ B_2 \end{pmatrix}.$$

ここで，$L_{11} : l \times l,\ L_{21} : (n-l) \times l,\ L_{22} : (n-l) \times (n-l),\ X_1, B_1 : l \times k,\ X_2, B_2 : (n-l) \times k$.

5: アルゴリズムを再帰的に用いて，X_1 を求める.

$$L_{11} X_1 = B_1.$$

6: 行列を更新する (DGEMM).

$$B_2 - L_{21} X_1 \to B'_2.$$

7: アルゴリズムを再帰的に用いて，X_2 を求める.

$$L_{22} X_2 = B'_2.$$

8: **end if**

Vol.2010-HPC-126 No.33
2010/8/4

表 2 LU 分解の実効性能 (MFLOPS)

環境 1		LU 分解の d						
サイズ=4096		32	64	128	256	512	1024	∞
TRSM の d	32	4.98	5.35	5.56	5.52	5.53	5.47	5.34
	64	4.94	5.32	5.49	5.49	5.50	5.44	5.48
	128	4.97	5.36	5.54	5.50	5.50	5.47	5.47
	256	4.95	5.32	5.50	5.48	5.39	5.39	5.35
	512	4.99	5.36	5.56	5.51	5.51	5.43	5.42
	∞	4.94	5.32	5.51	5.48	5.48	5.48	5.46

環境 1		LU 分解の d						
サイズ=8192		32	64	128	256	512	1024	∞
TRSM の d	32	5.79	6.27	6.51	6.46	6.44	6.36	6.12
	64	5.77	6.31	6.55	6.50	6.50	6.42	6.31
	128	5.79	6.27	6.50	6.45	6.45	6.39	6.34
	256	5.83	6.32	6.56	6.50	6.41	6.32	6.19
	512	5.78	6.26	6.51	6.45	6.43	6.37	6.30
	∞	5.83	6.31	6.55	6.50	6.48	6.42	6.35

環境 2		LU 分解の d						
サイズ=4096		32	64	128	256	512	1024	∞
TRSM の d	32	7.75	8.10	8.18	8.20	8.12	8.04	7.96
	64	7.75	8.10	8.17	8.20	8.14	8.08	8.03
	128	7.77	8.10	8.17	8.19	8.14	8.09	8.06
	256	7.76	8.10	8.18	8.19	8.03	7.95	7.91
	512	7.76	8.10	8.17	8.20	8.14	8.08	8.05
	∞	7.77	8.10	8.17	8.19	8.14	8.09	8.06

環境 2		LU 分解の d						
サイズ=8192		32	64	128	256	512	1024	∞
TRSM の d	32	8.12	8.54	8.76	8.78	8.75	8.68	8.54
	64	8.13	8.54	8.76	8.78	8.75	8.71	8.65
	128	8.13	8.55	8.76	8.78	8.76	8.72	8.70
	256	8.13	8.54	8.76	8.78	8.66	8.60	8.57
	512	8.12	8.54	8.76	8.79	8.74	8.71	8.70
	∞	8.13	8.54	8.76	8.79	8.74	8.72	8.70