1J-05

行列和計算最適化による Strassen 法の高速化

坂本真貴人† 藤井昭宏‡ 小西克巳‡ 田中輝雄‡
工学院大学大学院情報学専攻† 工学院大学情報学部‡

1 はじめに

さまざまな科学技術計算の基本演算である行列積を高速に計算する手法として Strassen 法[1]がある．これは行列積の計算量を削減する手法であり，再帰的に適用できる．再帰的に適用する回数（以下，Strassen 段数）を増やすことで計算量をさらに削減できる．

Strassen 法は小行列の行列積を行う部分と行列和を行う部分に分かれている．現在主流のメモリが階層構造の計算機では，行列積の性能が演算律速であるのに対し，行列和の性能はメモリ律速である．したがって Strassen 法を並列化した場合，行列積と行列和の性能差が広がり，行列和部分の計算時間が全計算時間の多くを占めるようになる．

本研究では，共有メモリ並列環境において行列和部分を最適化し，Strassen 法を高速化する手法の提案を行う．

2 Strassen 法

N次正方行列A,B,Cについて，行列積$C=A\times B$を計算する際に，Strassen 法ではまず図 1 のように行列A,B,Cを四分割する．次に$N/2\times N/2$行列である小行列T, Sを作り，小行列Qを計算する．最後にQを用いて行列Cを計算する．Strassen 法において，小行列Qの計算が行列積部分，小行列T,S，行列Cの計算が行列和部分となる．行列積部分に線形代数ライブラリ BLAS［2］のルーチンである DGEMM を用いると高速に計算できる．

$$A=\begin{pmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{pmatrix}\quad B,C\text{も同様}$$

$T_1=A_{12}-A_{22}$	$S_1=B_{21}+B_{22}$	$Q_1=T_1\cdot S_1$
$T_2=A_{11}+A_{22}$	$S_2=B_{11}+B_{22}$	$Q_2=T_2\cdot S_2$
$T_3=A_{11}-A_{21}$	$S_3=B_{11}+B_{12}$	$Q_3=T_3\cdot S_3$
$T_4=A_{11}+A_{12}$	$S_4=B_{22}$	$Q_4=T_4\cdot S_4$
$T_5=A_{11}$	$S_5=B_{12}-B_{22}$	$Q_5=T_5\cdot S_5$
$T_6=A_{22}$	$S_6=B_{21}-B_{11}$	$Q_6=T_6\cdot S_6$
$T_7=A_{21}+A_{22}$	$S_7=B_{11}$	$Q_7=T_7\cdot S_7$

$$C_{11}=Q_1+Q_2-Q_4+Q_6\qquad C_{12}=Q_4+Q_5$$
$$C_{21}=Q_6+Q_7\qquad C_{22}=Q_2-Q_3+Q_5-Q_7$$

図１ Strassen 法

3 行列和計算の最適化手法

Strassen 法の行列和部分は，メモリアクセス回数やメモリアクセス時間の削減が重要である．この章では，これらを削減する手法を提案する．

3.1 行列和をまとめて計算（手法１）

Strassen 法では，通常は図２左のように計算する．この場合，小行列T,Sを１つずつ計算するため，行列A,Bの同じ要素に対して複数回アクセスする必要がある．

そこで手法１では，図２右のように小行列T,Sの行列和計算を１つのループ文内でまとめて計算する．それにより，行列A,Bの各要素へのアクセスが１回で済む．

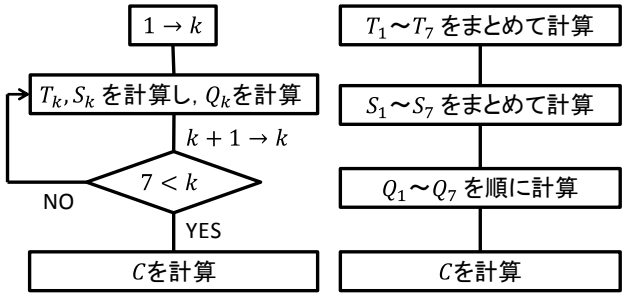

図２ Strassen 法の計算手順
（左:通常の手法，右:手法１）

3.2 式の展開（手法２A）

Strassen 段数が２段以上の場合，図３に示すT_{11}の式のように最下段の式を展開することで，行列A,Bから最下段の小行列T,S（２段の場合T_{11}～T_{77}，S_{11}～S_{77}）を直接計算できる．それによ

Acceleration of Strassen Algorithm from tuning matrix addition
Makito Sakamoto†, Akihiro Fujii‡, Katsumi Konishi‡, and Teruo Tanaka‡
†Major in Informatics, Kogakuin University
‡Faculty of Informatics, Kogakuin University

$$A = \begin{pmatrix} A'_{11} & A'_{12} \\ A'_{21} & A'_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{pmatrix}$$

$$\begin{aligned} T_1 &= A'_{12} - A'_{22} \\ T_{11} &= (T_1)_{12} - (T_1)_{22} \\ &= (A'_{12} - A'_{22})_{12} - (A'_{12} - A'_{22})_{22} \\ &= (A'_{12})_{12} - (A'_{22})_{12} - (A'_{12})_{22} + (A'_{22})_{22} \\ &= A_{14} - A_{34} - A_{24} + A_{44} \end{aligned}$$

図３　Strassen 法の式の展開（２段）

り中間の小行列は不要になる．また，最下段の行列和を手法１のようにまとめて計算することで，行列A, Bへのメモリアクセスを最小化できる．

3.3　式の展開とループ分割（手法２Ｂ）

式を展開することで，理想的にはメモリアクセス回数は削減される．しかし実際は１つのループ文で読み込むデータが多いと，レジスタの退避，回復の発生によるメモリアクセス回数の増加や，キャッシュヒット率低下によるメモリアクセス時間の増加が起こる．

そこで，１つのループ文を複数に分割することでこれらを回避する．今回は，小行列T, Sをそれぞれ１度に７つずつ計算するように分割する．

4　数値実験

数値実験では，通常の計算方法（BLAS の DGEMM）と Strassen 法で行列積$C = A \times B$の性能を比較する．実験環境は CPU が Intel Core i7 3770K 3.5GHz（４コア），BLAS は高速な BLAS である OpenBLAS [3] を用いる．手法１の適用前と適用後の比較を図４に，手法１，２Ａ，２Ｂをそれぞれ適用し，比較した結果を図５に示す．

図４を見ると，手法１を適用すると Strassen 段数 1 段，２段で共に性能が向上している．手法１適用前は１段の性能が DGEMM を上回る行列サイズが$N = 4500$程度であったのに対し，適用後は$N = 3500$程度まで小さくできた．

図５では，手法２Ｂを適用することで２段の性能が手法１より向上している．２段の性能が１段を上回る行列サイズが，手法１では$N = 7500$程度であったのが$N = 4500$程度まで小さくできた．手法２Ａを適用した２段は手法１より性能が低下していた．これは図６に示すように行列和部分の計算時間が増加したためである．式の展開のみではメモリアクセス回数やメモリアクセス時間が手法１より増加していると考えられる．

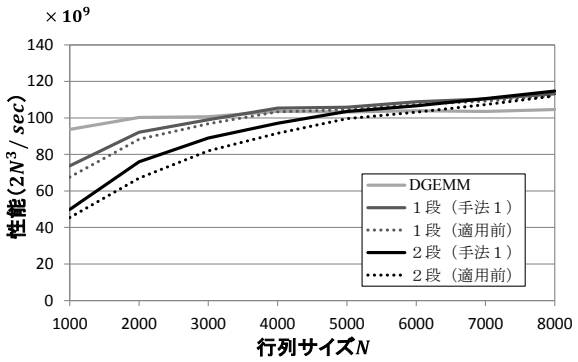

図４　手法１適用前と適用後の性能比較

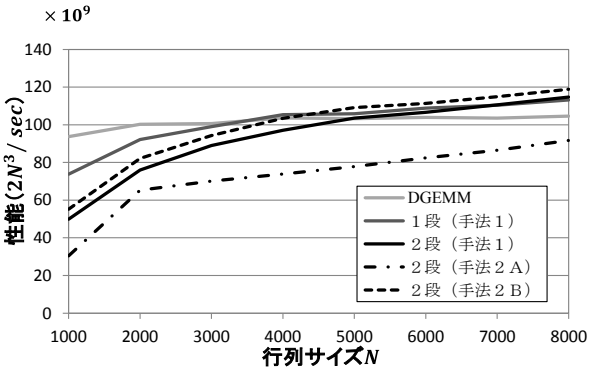

図５　手法１，２Ａ，２Ｂを適用した性能

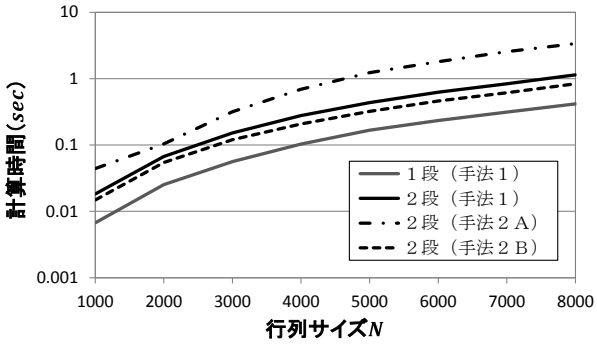

図６　行列和部分の計算時間

5　まとめ

本研究では，Strassen 法における行列和計算を最適化する手法を提案し，全体を高速化した．

実験の結果，行列和計算をまとめることで Strassen 段数１段の性能が通常の計算方法を上回る行列サイズNを約 1000 小さくし，式の展開とループ分割を適用することで２段の性能が１段を上回るNを約 3000 小さくできた．

今後はより最適なループ分割の仕方やキャッシュサイズを考慮した最適化手法を見つけたい．

参考文献

[1] V. Strassen, Gaussian Elimination is not Optimal, Numer. Math., vol. 13, pp.354-356, Aug. 1969.
[2] BLAS (Basic Linear Algebra Subprograms), http://www.netlib.org/blas/.
[3] OpenBLAS Homepage, http://www.openblas.net/.