Strassenのアルゴリズムによる倍精度・多倍長正方行列積の高速化

Acceleration of Double and Multiple Precision Square Matrix Multiplications by using Strassen's Algorithm

幸谷智紀*
Tomonori KOUYA*

Abstract: Strassen's algorithm is to accelerate matrix multiplication with the less numbers of arithmetics than usual one. Although we can find many related studies to accelerate sigle or double precision matrix multiplicaiton by using Strassen's algorithm, papers to accelerate multiple precision matrix multiplication are not found. In this paper, we estimate the performance of our implemented matrix multiplication program for any sizes of square matrixces, and then unveil the special properties in case of multiple precision as compared with double precision.

1. 初めに

科学技術計算が大規模化する昨今，高精度な数値解を得るために倍精度計算を超える演算精度が要求されるケースが増えている．しかし，一般にはソフトウェアで実装される高精度な多倍長浮動小数点演算（多倍長計算）は，ハードウェアが直接処理する単精度・倍精度に比べて多大な計算時間を要する．そのため，多倍長計算にはアルゴリズムやコンピューターアーキテクチャによる高速化支援が不可欠である．本稿ではStrassenのアルゴリズムを用いた，特に多倍長行列積の高速化について，倍精度計算の結果と比較しながら議論する．

Strassenのアルゴリズムについては Higham のサーベイ[3]が詳しい．それによると，1968年の論文[10]においてWinogradが次のような提案を行ったことが発端のようである，

偶数次 n の正方行列 $A, B \in \mathbb{R}^{n\times n}$ の積 $C := AB$ を求めるとする．この時，$C = [c_{ij}] \in \mathbb{R}^{n\times n}$ の一要素を

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} = \sum_{k=1}^{n/2}(a_{i,2k-1} + b_{2k,j})(a_{i,2k} + b_{2k-1,j}) - \underbrace{\sum_{k=1}^{n/2} a_{i,2k-1}a_{i,2k}}_{(*)} - \underbrace{\sum_{k=1}^{n/2} b_{2k,j}b_{2k-1,j}}_{(**)} \quad (1)$$

と計算することにより，2項目 (*) は C の i 行目の要素の計算，3項目 (**) は j 列目の要素の計算においてはそれぞれ共通しているため，結果として C の計算全体としては乗算の回数を減らすことが出来る．このアルゴリズムの延長上に，Strassenのアルゴリズム[9]が提案されることになる．

このようなアルゴリズムをWinogradが考案した理由をHighamは次のように解説する．

「Winogradの論文は即座に実用化できるアルゴリズムを提案している．というのも，1960年代のコンピュータは浮動小数点演算の乗算が加算に比べて2〜3倍低速なのが普通だったからである」．続けて，「現代のマシンでは両者の計算時間の差はなくなっている」としている．

ハードウェアで実行される倍精度演算においては確かにHighamが指摘する通り乗算だけ減らすことはあまり意味がなく，後にStrassenのアルゴリズムを倍精度行列積計算の高速化に適用する際には，乗算と加算の両方あわせたトータルの演算量が減っていることに着目するようになる．現在はGPU上における倍精度行列積の高速化も行われるようになっており[6]，LU分解[11]，QR分解への応用[4]も提案されている．

しかし，ソフトウェアによる実装がなされる多倍長計算では，計算桁数が伸びれば伸びるほど乗算が加算より低速になっていく．従って，Winogradによる最初の式(1)，つまり乗算の削減効果を期待しての提案が，多倍長行列積計算の高速化に生きてくる．倍精度計算の結果と比較することによって更にその効果が分かりやすく提示できる．

本稿ではまず，Strassenのアルゴリズムと，その変種であるWinogradの改良アルゴリズムを述べ，任意次数の行列積計算に適応させるための主要である動的パディング (dynamic padding) と動的ピーリング (dynamic peeling) について説明する．次に，ベンチマークテストによって倍精度・多倍長精度の正方行列積の高速化が達成できることを示し，両者の差を分析する．最後にまとめと今後の課題を述べる．

2. Strassenのアルゴリズムについて

我々が実装した倍精度・多倍長精度の行列積計算プログラムは，偶数次数の際には以下で述べるStrassenのアルゴリズム，もしくはWinogradの改良アルゴリズムをベースにした計算を行い，奇数次の際には動的パディングと動的ピーリングを適用して計算を行うようになっている．それぞれのアルゴリズムをここで解説する．

2.1 Strassenのアルゴリズム

偶数次数 n の正方行列 A, B に対して，行列積 $C := AB$ を計算するに際し，Strassenのアルゴリズムは次のように行列を4等分割してブロック化して計算を行う．

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \quad (2)$$

この時，$A_{ij}, B_{ij} \in \mathbb{R}^{n/2\times n/2}$ である．このようにブロック化した A,B を用いて，まず次の P_i $(i = 1, 2, ..., 7)$ の計算を行う．

2014年2月28日受理
*総合情報学部コンピュータシステム学科

$$
\begin{aligned}
P_1 &:= (A_{11} + A_{22})(B_{11} + B_{22}) \\
P_2 &:= (A_{21} + A_{22})B_{11} \\
P_3 &:= A_{11}(B_{12} - B_{22}) \\
P_4 &:= A_{22}(B_{21} - B_{11}) \\
P_5 &:= (A_{11} + A_{12})B_{22} \\
P_6 &:= (A_{21} - A_{11})(B_{11} + B_{12}) \\
P_7 &:= (A_{12} - A_{22})(B_{21} + B_{22})
\end{aligned} \tag{3}
$$

これらを用いて，C の計算を次のように行う．

$$
C := \begin{bmatrix} P_1 + P_4 - P_5 + P_7 & P_3 + P_5 \\ P_2 + P_4 & P_1 + P_3 - P_2 + P_6 \end{bmatrix} \tag{4}
$$

このアルゴリズムを適用することにより，n 次正方行列の行列積に必要となる乗算量 $M(n)$ と加減算 $A(n)$ とすると，

$$
M(n) = 7M(n/2),\ A(n) = 18A(n/2)
$$

となる．これを再帰的に適用していくことにより，結果として演算量が通常の行列積アルゴリズムより減らすことができる．

2.2 Winograd の改良法

Winograd の改良アルゴリズムは，偶数次数の行列 A, B を (2) のようにブロック化した後，次のように 3 段階で計算を行う．

$$
\begin{aligned}
S_1 &:= A_{21} + A_{22} \\
S_2 &:= S_1 - A_{11} \\
S_3 &:= A_{11} - A_{21} \\
S_4 &:= A_{12} - S_2 \\
S_5 &:= B_{12} - B_{11} \\
S_6 &:= B_{22} - S_5 \\
S_7 &:= B_{22} - B_{12} \\
S_8 &:= S_6 - B_{21}
\end{aligned} \tag{5}
$$

$$
\begin{aligned}
M_1 &:= S_2 S_6 \\
M_2 &:= A_{11} B_{11} \\
M_3 &:= A_{12} B_{21} \\
M_4 &:= S_3 S_7 \\
M_5 &:= S_1 S_5 \\
M_6 &:= S_4 B_{22} \\
M_7 &:= A_{22} S_8
\end{aligned} \tag{6}
$$

$$
\begin{aligned}
T_1 &:= M_1 + M_2 \\
T_2 &:= T_1 + M_4
\end{aligned} \tag{7}
$$

(5) → (6) → (7) の順に計算した後，C の計算を次のように行う．

$$
C := \begin{bmatrix} M_2 + M_3 & T_1 + M_5 + M_6 \\ T_2 - M_7 & T_2 + M_5 \end{bmatrix} \tag{8}
$$

Winograd の改良アルゴリズムの演算量は

$$
M(n) = 7M(n/2),\ A(n) = 15A(n/2)
$$

となり，Strassen のアルゴリズムに比べ，3 回分，行列の加減算が少なくなっていることが分かる．

2.3 動的パディングと動的ピーリング

次元数 n が奇数時の時には，行列サイズを増やして偶数次にする (パディング) か，行列サイズを減らして偶数部分と奇数部分のブロック化を行うか (ピーリング) するかのどちらかの方法を使う必要がある．

パディングのする際には，例えば適当な奇数次 d を用いて $n + d$ 次の行列 $\widetilde{A}, \widetilde{B}$ を次のように作って $\widetilde{C} := \widetilde{A}\widetilde{B}$ の計算を行えばよい．

$$
\widetilde{A} := \left[\begin{array}{c|c} A & 0 \\ \hline 0 & 0 \end{array}\right],\ \widetilde{B} := \left[\begin{array}{c|c} B & 0 \\ \hline 0 & 0 \end{array}\right]
$$

$$
\Longrightarrow \widetilde{C} := \widetilde{A}\widetilde{B} = \left[\begin{array}{c|c} C & 0 \\ \hline 0 & 0 \end{array}\right]
$$

$C := AB$ であるから，この部分に Strassen のアルゴリズム，もしくは Winograd の改良アルゴリズムを使用することになる．

ピーリングは行列をそれぞれ偶数次数になるように奇数次 $d(<< n)$ を決め，$n - d$ 次と d 次に不均等にブロック化して計算を行う．

$$
A := \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right],\ B := \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right]
$$

この結果，行列 C は

$$
\Longrightarrow C := AB = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]
$$

となる．ここで各ブロック $C_{11}, C_{12}, C_{21}, C_{22}$ は

$$
\begin{aligned}
C_{11} &:= \underline{A_{11}B_{11}} + A_{12}B_{21} \\
C_{12} &:= A_{11}B_{12} + A_{12}B_{22} \\
C_{21} &:= A_{21}B_{11} + A_{22}B_{21} \\
C_{22} &:= A_{21}B_{12} + A_{22}B_{22}
\end{aligned}
$$

となる．基本的には下線部の行列積の計算のみに Strassen のアルゴリズム，もしくは Winograd の改良アルゴリズムを用いる．

3. ベンチマークテスト

以上のアルゴリズムを用いて倍精度・多倍長精度の正方行列積のベンチマークテストを行う．計算は全て下記の環境を用いている．4 コアマシンではあるが，今回は並列化を行っておらず，全てシングルスレッド実行である．

H/W Intel Core i7 3850 (3.6GHz), 64GB RAM

S/W Scientific Linux 6.3 x86_64, Intel C Compiler Ver.13.0.1, BNCpack 0.8[5]/MPFR 3.1.2[8]/GMP 5.1.3[1]

倍精度計算，多倍長計算の行列積 $C := AB$ において使用した正方行列 $A, B \in \mathbb{R}^{n\times n}$ は下記の通りである．

$$A = \left[\sqrt{5}\,(i+j-1)\right]_{i,j=1}^{n},\ B = \left[\sqrt{3}\,(n-i)\right]_{i,j=1}^{n}$$

ベンチマークテストは複数回実施し，計算時間及び計算誤差の再現性があることを確認してある．また，2 秒以内の計算時間の場合は 2 秒以上になるまで計算回数を増やして平均値を算出している．計算時間計測は times 関数を利用し，システム時間を算出し，それを計算時間としている．

3.1 倍精度正方行列積

近年は CPU の性能向上のスピードにメモリアクセスの性能が追い付いていないため，演算コアに近い所に高性能なキャッシュメモリが 2 階層以上設けるアーキテクチャが主流となっている．メインメモリから転送されたデータはキャッシュメモリに一時的に置かれるため，一度キャッシュメモリに入ったデータを再利用する頻度が高ければ高いほど，メモリアクセスに要する時間は軽減されることになる．ことに演算に要する時間が短い低精度計算の場合はこの影響が非常に大きくなる．

加えて倍精度演算は並列性能を上げるための SIMD(Single Instruction Multiple Data) 命令を利用することによって，高速な計算が可能になることが多い．ことに行列積計算のような単純なアルゴリズムに対してはその効果は極めて高い．

メモリアクセスの最適化と，SIMD 命令の両方を用いて，徹底的に高速化を図った高性能数値計算ライブラリが Intel Math Kernel Library(IMKL)[7] である．この中には BLAS[2] ベースの API を備えており，基本線型計算機能，特に行列演算 (BLAS Level 3) の計算の高速性には定評がある．

我々は倍精度の行列積計算には全てこの IMKL を使用して実装を行い，Strassen や Winograd のアルゴリズムを用いてどの程度の高速性を達成できるかを追求することにした．その結果を Table 1 に示す．ブロックサイズは 1024 次とし，これを下回るサイズの行列積は Intel Math Kernel の cblas_dgemm 関数を使用して計算を行っている．

メインメモリに収まる行列サイズの計算に留めるため，16385 次までの計算まで実行し，次のことが判明した．

1. 4097 次以下の行列積計算では，IMKL の方がおおむね高速である．
2. 8192, 16383, 16384 次の行列積計算では，Strassen, Winograd のアルゴリズムが高速になっている．但し，動的ピーリングを行う $2^n + 1$ 次の計算時間が悪化している．
3. 要素ごとの最大相対差異 (IMKL の結果を真値とした相対誤差) は，Strassen のアルゴリズムの方が若干大きくなっている．

動的ピーリングの性能悪化の原因は，余剰部分の計算に時間を要しているためと思われる．

3.2 多倍長正方行列積

多倍長計算の場合，前述したように，加算と乗算の計算時間の差が，計算桁数=仮数部の桁数が長くなるにつれて広くなる．ベンチマークテストを行った環境では，Fig.1 に示す通り，約 1.7 倍から約 8.6 倍まで差が開く．

従って，乗算の計算回数が少なくなる Winograd の改良アルゴリズムの方が，Strassen のアルゴリズムに比べて計算時間が短くなることが期待される．加えて，前述したように階層化

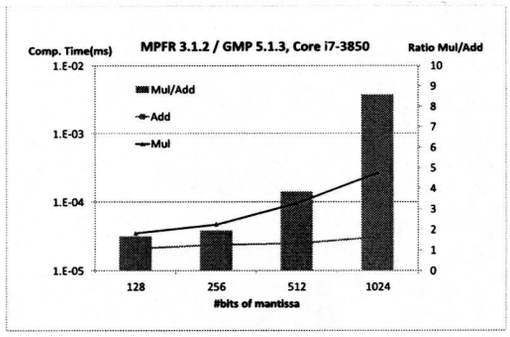

Fig. 1: 多倍長浮動小数点乗算 (Mul) と加算 (Add) の計算時間と比

されたメモリアクセスのコストが多倍長計算においても問題になってくる．

上記 2 点のことを考慮し，我々のベンチマークテストでは次の 4 タイプの正方行列積計算を行った．

Simple 単純な 3 重ループによる実装

Block 正方行列を均等なサイズ (16, 32, 64 次) にブロック化 (+パディング) して行列積計算を行う実装

Strassen Strassen のアルゴリズムを使用した実装

Winograd Winograd の改良アルゴリズムを使用した実装

これらのベンチマーク結果を，2 進 128 桁 (10 進約 38 桁) 計算の場合 (Table 2) と，2 進 1024 桁 (10 進約 308 桁) 計算の場合 (Table 3) に示す．

Table 2, 3 を比較することにより，次のことが判明した．

1. 単純 3 重ループ (Simple) の場合，2 進 128 桁計算における計算時間の振幅が激しい．これはキャッシュメモリアクセスのミスが頻発していることが関係していると思われる．2 進 1024 桁計算になるとその差は少なくなっており，キャッシュミスより演算量の増大の影響が大きいためと思われる．
2. ブロック化 (Block) したアルゴリズムでは，安定した計算時間を得られている．また，行列サイズが大きくなった時には安定的に単純 3 重ループ計算より高精度な結果が得られている．
3. Strassen のアルゴリズムより，Winograd の改良アルゴリズムの方が全般的に高速であり，ことに行列サイズと計算桁数が大きいほどのその傾向が強い．行列の加減算の低減効果が大きいものと思われる．

それぞれの計算桁数における成分の相対誤差の最大値を Table 4, 5 に示す．

128bit 計算，1024bits 計算どちらも次元数が高くなるにつれて Strassen, Winograd のどちらも他の計算法に比べて相対誤差が大きくなることが分かる．この例では 10 進約 1〜3 桁程度悪くなることがあり，特に Winograd にワーストケースが多く見受けられる．

Table 1: 倍精度正方行列積の計算時間 (秒) と最大相対差異 (対 IMKL)

	Computational Time (Unit: seconds)			Max. Relative Differences	
$n \times n$	IMKL	Strassen	Winograd	Strassen	Winograd
1023 x 1023	0.08438	0.08313	0.08531	0.00E+00	0.00E+00
1024 x 1024	0.08281	0.08219	0.08437	0.00E+00	0.00E+00
1025 x 1025	0.08719	0.13875	0.14	6.42E-15	2.68E-15
2047 x 2047	0.6575	0.785	0.785	2.99E-15	1.49E-15
2048 x 2048	0.6525	0.71	0.7125	2.88E-15	1.53E-15
2049 x 2049	0.675	1.135	1.185	1.29E-14	4.66E-15
4095 x 4095	5.2	5.77	5.69	8.55E-15	1.57E-15
4096 x 4096	5.22	5.51	5.46	8.83E-15	1.57E-15
4097 x 4097	5.32	9.1	9.02	2.50E-14	2.39E-15
8191 x 8191	41.63	41.73	41.32	2.06E-14	1.76E-15
8192 x 8192	41.54	40.73	40.03	2.03E-14	1.69E-15
8193 x 8193	42.4	66.68	66.42	8.36E-14	4.86E-15
16383 x 16383	354.89	297.31	307.04	5.30E-14	2.18E-15
16384 x 16384	329.74	291.66	280.72	4.08E-14	1.79E-15
16385 x 16385	344.41	479.86	480.21	1.68E-13	5.12E-15

Table 2: 多倍長正方行列積のベンチマークテスト (2 進 128 桁, 単位 : 秒)

$n \times n$	Simple	Block(16)	Block(32)	Block(64)	min(Simple, Block)	Strassen	Winograd
31 x 31	0.00161	0.00227	0.00155		0.00155	0.00157	0.00161
32 x 32	0.00175	0.00244	0.00173		0.00173	0.00175	0.00175
33 x 33	0.00189	0.00328	0.00443		0.00189	0.00217	0.00219
63 x 63	0.01414	0.01828	0.01844	0.0134	0.0134	0.01473	0.01484
64 x 64	0.01633	0.01898	0.01922	0.01527	0.01527	0.01395	0.01277
65 x 65	0.01523	0.02187	0.02438	0.03359	0.01523	0.01602	0.01477
127 x 127	0.12094	0.14938	0.14875	0.1625	0.12094	0.10531	0.09813
128 x 128	0.1425	0.15187	0.1525	0.17125	0.1425	0.10094	0.08625
129 x 129	0.12406	0.16188	0.16813	0.20312	0.12406	0.11	0.095
255 x 255	1.06	1.2	1.215	1.235	1.06	0.7175	0.63
256 x 256	1.245	1.215	1.22	1.25	1.215	0.7	0.57
257 x 257	1.04	1.25	1.28	1.365	1.04	0.735	0.6025
511 x 511	9.6	9.71	9.61	10.02	9.6	4.84	4.06
512 x 512	10.83	9.7	9.68	9.96	9.68	4.77	3.73
513 x 513	10.02	9.89	9.97	10.44	9.89	4.92	3.88
1023 x 1023	107.78	77.63	77.8	79.36	77.63	32.02	25.57
1024 x 1024	213.09	77.77	77.72	79.51	77.72	31.53	24.1
1025 x 1025	94.62	78.92	78.41	81.48	78.41	32.21	24.77
2047 x 2047	756.81	627.75	619.21	648.31	619.21	211.8	163.87
2048 x 2048	1679.04	624.86	618.87	639.71	618.87	211.19	155.67
2049 x 2049	632.74	623.24	625.69	640.84	623.24	212.79	157.52

Table 3: 多倍長正方行列積のベンチマークテスト (2 進 1024 桁)

$n \times n$	Simple	Block(16)	Block(32)	Block(64)	min(Simple, Block)	Strassen	Winograd
31 x 31	0.00777	0.00992	0.00777		0.00777	0.00766	0.00773
32 x 32	0.00852	0.01086	0.00875		0.00852	0.00844	0.00852
33 x 33	0.00934	0.01258	0.01375		0.00934	0.00965	0.00969
63 x 63	0.06906	0.08313	0.08281	0.06594	0.06594	0.06125	0.06219
64 x 64	0.0725	0.08688	0.08688	0.06969	0.06969	0.06156	0.05453
65 x 65	0.07656	0.09156	0.09625	0.105	0.07656	0.0675	0.06078
127 x 127	0.615	0.6725	0.6775	0.6875	0.615	0.39875	0.365
128 x 128	0.665	0.6875	0.69	0.7025	0.665	0.39375	0.31125
129 x 129	0.6275	0.72	0.7275	0.7525	0.6275	0.4175	0.33375
255 x 255	5.5	5.46	5.47	5.54	5.46	2.33	1.95
256 x 256	5.77	5.53	5.53	5.64	5.53	2.31	1.72
257 x 257	5.68	5.61	5.72	5.8	5.61	2.41	1.81
511 x 511	45.73	43.81	43.96	44.51	43.81	13.4	10.57
512 x 512	49.71	44.37	44.31	44.2	44.2	13.02	9.44
513 x 513	46.58	44.37	44.77	45.12	44.37	13.38	9.81
1023 x 1023	372.1	352.79	354.37	353.15	352.79	76.93	57.98
1024 x 1024	463.79	356.1	356.85	355.99	355.99	74.58	52.47
1025 x 1025	385.24	356.58	357.17	361.24	356.58	76.36	54.22
2047 x 2047	3122.48	2829.43	2820.16	2833.66	2820.16	454.02	329.41
2048 x 2048	3754.89	2845.7	2824.34	2859.24	2824.34	446.87	302.56
2049 x 2049	2933.26	2829.95	2835.54	2859.66	2829.95	456.08	307.05

Table 4: 多倍長正方行列積の成分ごとの最大相対誤差 (2 進 128 桁)

$n \times n$	Simple	Block(16)	Block(32)	Block(64)	Strassen	Winograd
31 x 31	1.02E-38	5.75E-39	1.02E-38		1.02E-38	1.02E-38
32 x 32	1.29E-38	7.00E-39	1.29E-38		1.29E-38	1.29E-38
33 x 33	1.31E-38	6.48E-39	1.35E-38		1.35E-38	1.35E-38
63 x 63	1.88E-38	5.23E-39	1.04E-38	1.88E-38	3.97E-38	3.97E-38
64 x 64	1.81E-38	7.58E-39	9.59E-39	1.81E-38	4.55E-38	1.59E-38
65 x 65	1.93E-38	8.69E-39	1.34E-38	1.93E-38	3.24E-38	1.93E-38
127 x 127	2.75E-38	7.47E-39	8.46E-39	1.16E-38	1.21E-37	3.58E-37
128 x 128	2.13E-38	7.57E-39	8.36E-39	1.04E-38	1.13E-37	1.70E-38
129 x 129	2.67E-38	7.80E-39	1.02E-38	1.16E-38	9.11E-38	2.45E-38
255 x 255	4.56E-38	1.09E-38	1.03E-38	1.18E-38	3.32E-37	2.35E-37
256 x 256	4.28E-38	1.06E-38	7.39E-39	1.23E-38	2.55E-37	1.95E-38
257 x 257	3.68E-38	1.25E-38	9.48E-39	1.31E-38	2.73E-37	3.68E-38
511 x 511	6.12E-38	2.02E-38	1.43E-38	1.19E-38	5.83E-37	7.48E-37
512 x 512	5.46E-38	1.74E-38	1.44E-38	1.26E-38	6.18E-37	2.19E-38
513 x 513	5.54E-38	1.77E-38	1.52E-38	1.16E-38	5.56E-37	5.81E-38
1023 x 1023	8.42E-38	2.12E-38	1.98E-38	1.55E-38	1.46E-36	4.58E-36
1024 x 1024	8.05E-38	2.12E-38	1.70E-38	1.22E-38	1.44E-36	1.85E-38
1025 x 1025	7.91E-38	1.94E-38	1.77E-38	1.63E-38	1.16E-36	7.91E-38
2047 x 2047	1.34E-37	4.33E-38	2.38E-38	1.85E-38	3.20E-36	2.25E-35
2048 x 2048	1.33E-37	3.59E-38	2.32E-38	1.83E-38	2.99E-36	2.12E-38
2049 x 2049	1.31E-37	3.31E-38	2.13E-38	2.16E-38	2.86E-36	1.31E-37

Table 5: 多倍長正方行列積の成分ごとの最大相対誤差 (2 進 1024 桁)

$n \times n$	Simple	Block(16)	Block(32)	Block(64)	Strassen	Winograd
31 x 31	2.39E-308	7.80E-309	2.39E-308		2.39E-308	2.39E-308
32 x 32	2.42E-308	1.30E-308	2.42E-308		2.42E-308	2.42E-308
33 x 33	2.28E-308	1.62E-308	2.28E-308		2.28E-308	2.28E-308
63 x 63	3.30E-308	1.34E-308	1.55E-308	3.30E-308	8.06E-308	8.06E-308
64 x 64	3.53E-308	1.32E-308	1.57E-308	3.53E-308	9.52E-308	3.26E-308
65 x 65	3.24E-308	1.34E-308	1.28E-308	4.19E-308	7.48E-308	4.19E-308
127 x 127	3.64E-308	1.29E-308	1.61E-308	3.04E-308	2.12E-307	1.75E-307
128 x 128	4.20E-308	1.35E-308	1.55E-308	3.14E-308	1.83E-307	2.84E-308
129 x 129	4.69E-308	1.28E-308	1.65E-308	2.71E-308	2.03E-307	3.93E-308
255 x 255	5.66E-308	1.62E-308	2.59E-308	1.77E-308	6.88E-307	6.77E-307
256 x 256	5.49E-308	1.56E-308	1.81E-308	1.96E-308	7.60E-307	3.42E-308
257 x 257	5.50E-308	1.63E-308	1.97E-308	2.03E-308	5.98E-307	5.50E-308
511 x 511	9.73E-308	2.06E-308	2.04E-308	1.77E-308	1.26E-306	1.44E-306
512 x 512	9.65E-308	1.70E-308	1.86E-308	1.72E-308	1.28E-306	3.14E-308
513 x 513	1.01E-307	2.08E-308	2.03E-308	2.32E-308	1.53E-306	1.01E-307
1023 x 1023	1.61E-307	2.87E-308	3.22E-308	2.46E-308	2.65E-306	1.96E-305
1024 x 1024	1.61E-307	3.55E-308	3.83E-308	2.98E-308	2.67E-306	3.72E-308
1025 x 1025	1.65E-307	3.40E-308	3.47E-308	2.32E-308	2.55E-306	1.73E-307
2047 x 2047	2.06E-307	4.05E-308	4.39E-308	3.09E-308	6.56E-306	7.46E-305
2048 x 2048	2.17E-307	4.25E-308	4.54E-308	4.19E-308	6.30E-306	3.64E-308
2049 x 2049	2.15E-307	4.39E-308	4.39E-308	3.27E-308	5.79E-306	2.05E-307

4. 結論と今後の課題

以上のベンチマークテストの結果，次のことが分かった．

- 倍精度計算においては動的パディングより動的ピーリングの非効率性が明らかとなった．また，かなり大規模な行列サイズでなければ Strassen のアルゴリズムの効果が発揮できないことも判明した．
- 多倍長計算においては，下記のことが明らかとなった．
  - 計算桁数が比較的小さい時は，ブロック化アルゴリズムが効果的である．
  - Strassen のアルゴリズムより Winograd の改良アルゴリズムの方が計算時間の点で有利である．
  - 演算誤差についてはどちらも通常の方法より 1 〜 3 桁程度悪化する．

今後の課題としては，次のことが挙げられる．

- 動的パディングと動的ピーリングの混在アルゴリズムの最適化．
- 誤差と計算時間の最適化を自動的に行うチューニング機構の検討．
- GPU，Intel MIC 等のハードウェアアクセラレータへの適用．

参考文献

1) Swox AB. The GNU Multiple Precision arithmetic library. `http://gmplib.org/`.
2) BLAS. `http://www.netlib.org/blas/`.
3) Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.
4) Philip A. Knight. Fast rectangular matrix multiplication and QR decomposition. *Linear Algebra and its Applications*, Vol. 221, No. 0, pp. 69 – 81, 1995.
5) Tomonori Kouya. BNCpack. `http://na-inet.jp/na/bnc/`.
6) Junjie Li, Sanjay Ranka, and Sartaj Sahni. Strassen's matrix multiplication on gpus. In *Proceedings of the 2011 IEEE 17th International Conference on Parallel and Distributed Systems*, ICPADS '11, pp. 157–164, Washington, DC, USA, 2011. IEEE Computer Society.
7) Intel Math Kernel Library. `http://www.intel.com/software/products/mkl/`.
8) MPFR Project. The MPFR library. `http://www.mpfr.org/`.
9) Volker Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, Vol. 13, No. 4, pp. 354–356, 1969.
10) S. Winograd. A new algorithm for inner product. *IEEE Trans. Comput.*, Vol. 17, No. 7, pp. 693–694, July 1968.
11) 古井充, 鈴木健二, 後保範. 拡張ストラッセン法の連立一次方程式への応用. 京大数理研講究録, Vol. 1362, pp. 38–46, 2004.