Vol.2019-HPC-170 No.36
2019/7/26

高速化手法を取り入れた 多倍長精度行列乗算ライブラリの性能評価

幸谷 智紀[1,a)]

概要：ソフトウェア実装による多倍長精度浮動小数点演算の高速化は限界を迎えており，BLAS レベルでの高速化が求められるようになっている．その中で，行列乗算は高速化の余地が大きく，キャッシュヒット率の向上，並列化といった手法に加え，分割統治法に基づく演算量の低減化の効果が著しい．本講演ではこれらを取り入れた多倍長精度行列乗算ライブラリ BNCmatmul の実装と性能評価の結果について，MBLAS と比較しつつ報告する．

キーワード：多倍長精度演算，行列乗算，並列化

Performance evaluation of multiple precision matrix multiplication library using acceleration techniques

Abstract: Due to the wall of acceleration limit for multiple precision (MP for short) floating-point arithmetic, MP BLAS functions must speed up. In these functions, matrix-matrix multiplication may be easily accelerated by using optimization techniques such as improving cache-hit ratio, parallelization, and application of divide-and-conquare algorithms like Strassen's one. In this paper, we explain BNCmatmul library, our own MP real matrix-matrix multiplication library, and show the result obtained through performance evaluation, comparing with MBLAS.

Keywords: multiple precision floating-point arithmetic, matrix multiplication, parallelization

1. 初めに

コンピュータ上で有限桁の浮動小数点演算が使用されるようになって以来，ユーザーが要求する精度の計算結果を求めることは主要課題の一つとして常に俎上に上っている．

初期誤差や計算途中で発生する丸め誤差の影響に対して鋭敏な悪条件問題の数値解の精度を保つ方法の一つとして，ハードウェアがサポートする IEEE754 倍精度より長い仮数部を持つ多倍長精度浮動小数点 (Multiple Precision Floating-Point, MPFP) 演算ライブラリの使用が挙げられる．長い仮数部の末尾にこれらの誤差を追いやり，桁落ちによる有効桁数の減少の影響を少なくすることで，計算結果の有効桁数を多く確保できるというメリットがある反面，MPFP ライブラリの多くは既存のデータ型を複数組み合わせてソフトウェアによって実装されていることから，標準 FP 演算よりも計算速度が劇的に下がるというデメリットがあることは広く知られている．そのため，基盤的な MPFP 演算ライブラリはアルゴリズムの改良やハードウェアによる高速化技法を用いてパフォーマンスを極力上げることが求められる．現在主流となっている MPFP 演算ライブラリのうち，無料で使用でき，高速性と信頼性に優れたものは次の 2 つの系統に集約される．

- IEEE754 倍精度浮動小数点数を使用するマルチコンポーネント方式の QD[1] とその派生型ライブラリ
- GNU MP(GMP)[3] が提供する様々な CPU アーキテクチャに最適化された任意長自然数演算 (Multiple Precision Natural number arithmetic, MPN) カーネルを土台とする MPFR[2] とその派生型ライブラリ

今から多倍長精度の数値計算を行うのであれば，これら二つの MPFP 演算ライブラリを土台とした計算機環

[1] 静岡理工科大学
Shizuoka Institute of Science and Technology, Fukuroi, Shizuoka 437–8555, Japan
[a)] kouya.tomonori@sist.ac.jp

境を使用することを推奨したい．特に線型計算については，BLAS や LAPACK の Fortran コードを土台とし，QD や MPFR を取り込んで C++クラスライブラリ化した MPACK(MPBLAS + MPLAPACK)[10] の利用をまず考えるべきであろう．ソースからビルドするのに要する多大な時間は度外視しても，マルチコア環境に適応して最大のスレッド数 (≠ 最大コア数) で自動的に並列化された BLAS, LAPACK とほぼ同じインターフェースの関数群を利用できるというメリットは大きい．

但し，後述するように，MPACK は安定的な性能を発揮する現在では最良の多倍長精度線型計算ライブラリであるが，必ずしも最高性能を発揮するものではない．いわば Reference BLAS 的な位置づけであって，キャッシュ最適化や SIMD 命令の使用などの高速化手法を駆使した ATLAS や Intel Math Kernel ライブラリのようなものが望まれる状況である．GPU 等のメニーコアハードウェアを用いた高速化は椋木 [13]，菱沼 [14]，中里 [15] らが QD ベースの高速化を施したものを公開しており，MPFP 演算ライブラリとしては CUMP[11]，マルチコンポーネント型の任意精度ライブラリとしては CPU，GPU 両方に対応した CAMPARY[12] があるが，BLAS レベルの高速化の比較，特に MBLAS との比較は見当たらない．

本稿では，既に開発した BNCmatmul[9] に基づく任意精度実行列乗算の性能評価を行い，8 コアの Core i7 環境における評価結果を提示する．我々のライブラリの最大の特徴は，Strassen や Winograd のアルゴリズムを取り入れたことで，これにより演算量，特に乗算回数を減らし，計算時間の短縮が可能になったことである．成果については，MPFR/GMP 環境における高速性 [4] の提示や，OpenMP を用いた並列計算の性能向上 [5] を既に行っており，それらを総合した簡易チューニングツール [6] の開発も行っている．今回は，MPFR/GMP を用いた任意精度実行列乗算について，MBLAS の Rgemm 関数と比較を行って，比較的短い桁数でも長い桁数でもより高速な計算が可能であることを示す．

2. 多倍長精度行列乗算ライブラリ BNCmatmul

BNCmatmul[9] は C および C++で記述された多倍長精度行列乗算ライブラリで，MPFP 演算ライブラリとしては QD と MPFR/GMP を土台としており，DD(Double-Double) 精度，QD(Quadruple-Double) 精度，MPFR による任意精度をサポートしている．MPFR 用には任意精度実行列 (MPFMatrix 型) を使用し，同様にして DD 精度実行列 (DDMatrix 型)，QD 精度実行列 (QDMatrix 型) を定義して使用する．現状では，C++バージョンの DDMatrix と QDMatrix の行列乗算が低速であるので，今回は高速性が担保された MPFMatrix 型のみ取り上げる．行列要素は

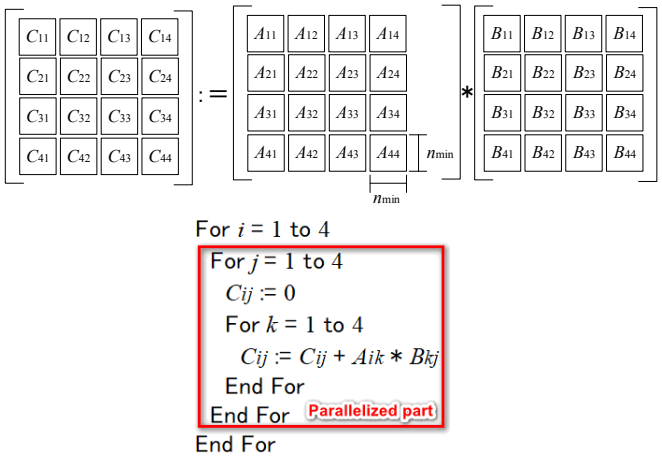

図 1 ブロック化アルゴリズム (上) とその並列化 (下)

行優先 (Row-major) 方式を使用する．

本稿で対象とする任意サイズの実行列乗算を $C := AB = [c_{ij}] \in \mathbb{R}^{m\times n}$ とする．ここで，$A = [a_{ij}] \in \mathbb{R}^{m\times l}$ and $B = [b_{ij}] \in \mathbb{R}^{l\times n}$ である．ここで C の要素 c_{ij} は

$$c_{ij} := \sum_{k=1}^{l} a_{ik} b_{kj}. \qquad (1)$$

である．この定義式をそのまま計算する方法を，本稿では単純行列乗算 (Simple) と呼ぶ．我々のライブラリではその他，ブロック化アルゴリズム (Block) と Strassen アルゴリズム (Strassen)，Winograd アルゴリズム (Winograd) をサポートしており，全てのアルゴリズムで OpenMP による並列化を行っている．

以下，ブロック化アルゴリズムと Strassen アルゴリズムについて述べる．

2.1 ブロック化アルゴリズム

ブロック化アルゴリズムは CPU のキャッシュメモリに格納された行列データの再利用を促す効果が高いことが知られている．例えば正方行列の場合，A と B を $n_{\min}$ 以下の小行列に分割し，

$$C_{ij} := \sum_{k=1}^{L} A_{ik} B_{kj}.$$

として計算を行う．

この際，キャッシュメモリ再利用の最適化を行うことで，計算時間の更なる低減を図ることができる．ATLAS[7] にはそれを自動的に行う機能が備わっている．標準的な IEEE754 精度の浮動小数点演算環境下では必要不可欠な機能であり，MPFP 演算でも，DD 精度，QD 精度だけでなく，MPFR の任意精度でも比較的小さい桁数の場合は最適化の効果があることは既に示した [5]．しかしその程度は IEEE754 精度より低く，特に任意精度では桁数が増えるに従って効果が薄れる．並列化は図 1 の下図に示すように，内側ループの部分でのみ行っている．

2.2 Strassen アルゴリズムとその並列化

ブロック化アルゴリズムは単純行列乗算と演算量は全く同じであるのに対し，Strassen のアルゴリズム，及び Winograd のアルゴリズムは，再帰的呼び出しを使って繰り返しを行うことで演算量そのものを削減することができる．そのため関連研究は多数あるが，任意精度計算に適用して性能向上を図ったものは見当たらないようである．MPFP 演算は，特に乗除算が重いため，演算量の削減効果は IEEE754 演算より高い．

我々が実装した行列乗算プログラムは，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．計算量としては Winograd のアルゴリズムが少ないことが知られているが，実装したものを比較すると，MPFP 演算環境ではあまり差がなく，並列化効率も Strassen に比べて劣ることから，今回は Strassen アルゴリズムの結果のみ示す．

Strassen のアルゴリズムは，図 2 に示すように，並列化可能な部分を `omp section` で分割，セクション単位で並列化を行う．ちなみに，Strassen のアルゴリズムに比べ，Winograd のアルゴリズムは依存関係が多く分割が複雑になるが，再帰的に行列乗算を呼び出す部分はどちらも同じく 7 スレッドに分割可能になる．

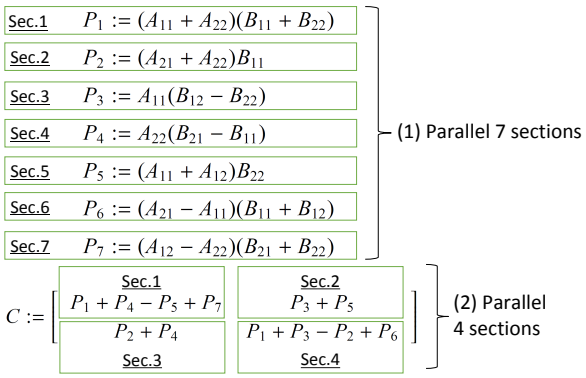

図 2 並列化した Strassen のアルゴリズム

これにより，MPFR/GMP はもとより，(擬似的)4 倍精度 (DD) および 8 倍精度 (QD) でも，ブロック化，Strassen, アルゴリズム対して並列化による性能向上を図ることができるようになったものの，ブロック化アルゴリズムに比べるとその効果は低いことが判明している [5].

3. 性能評価

性能評価のための計算機環境は以下の通りである．MPACK は Github から 2019 年 6 月中旬にダウンロードしたものをインストールし，使用する MPFP 演算ライブラリは MPACK インストール時に使用されるものを BNCmatmul でも共通して使うようにした．

H/W Intel Core i7-9700K (3.6GHz, 8 cores), 16GB RAM

S/W Ubuntu 18.04.2, GCC 7.3.0, MPACK 0.8.0 (2019 年 6 月ダウンロード)，MPFR 4.0.2[2]/GMP 6.1.2[3], QD 2.3.22[1], BNCpack 0.8[8]

使用した実正方行列 A，B は次の通りである．

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

要素全てが正の実数であるため，桁落ちは殆ど起きない．計算結果は，どの計算においても使用する桁数より 10 進 1〜2 桁の精度低下がみられる程度であった．

ベンチマークで使用したメイン関数を含むプログラムは全て C++で実装し，MBLAS の Rgemm 関数を直接呼び出せるようにしている．C++のコンパイルオプションは

```
g++ -O3 -std=c++11 -fopenmp
```

である．

3.1 MBLAS(Rgemm) の性能評価

MBLAS の関数は全て BLAS と同じ引数を取るようにできており，オリジナル同様，列優先 (Column-major) 方式のみサポートしている．今回使用したのは MBLAS の Rgemm 関数で，これは本来 $C := \alpha AB + \beta C$ を実行するためのものであるが，純粋に $C := AB$ を求めるために $\alpha = 1, \beta = 0$ と引数を指定した．また，行列も Rgemm 用に行優先で要素を格納したものを別途使用した．

計算精度 (仮数部長) を 128bits と 1024bits で指定し，1〜8 スレッドで Rgemm を実行して計算に要した時間 (秒) を計測した．全て複数回実行した結果の平均値である．その結果を表 1 に示す．

この MBLAS の Rgemm 関数の計算時間に対し，我々が実装した単純行列乗算 (simple) の計算時間の比のグラフを図 3 に示す．MBLAS の計算時間を単純行列乗算の計算時間で割った比を縦軸に，横軸に正方行列のサイズ (n) を示しており，この比が 1 以上の時には MBLAS より高速であることを表わしている．

単独で見ると分かりづらいが，BNCmatmul の単純行列乗算結果を比較すると，小さいサイズでは，C++のクラスライブラリ (MPFR C++) のオーバーヘッドが影響しているのか，Rgemm の方が低速になる．行列サイズを大きくすると，ブロック化の効果か，単純行列乗算よりも高速になることが分かる．

3.2 MPFR 128bits 計算

MPFR 128bits 計算 (10 進約 38 桁) で，BNCmatmul のブロック化アルゴリズム (Block($n_{\min}$)) と Strassen アルゴリズム (Strassen($n_{\min}$)) の計算時間と並列化効率を調べた．

表 1 Computational time of MBLAS(Rgemm) with MPFR

Unit: seconds, MPFR 128bits

n	1 Thread	2	4	8
63	0.022	0.011	0.008	0.004
64	0.023	0.012	0.008	0.004
65	0.024	0.012	0.009	0.005
127	0.18	0.09	0.06	0.03
128	0.18	0.09	0.06	0.03
129	0.19	0.10	0.06	0.03
511	11.27	5.73	3.87	1.95
512	11.33	5.76	3.89	1.95
513	11.38	5.78	3.91	1.97
1535	304.0	154.1	103.8	52.1
1536	304.4	154.3	104.1	52.4
1537	305.7	153.7	104.2	52.3

Unit: seconds, MPFR 1024bits

n	1 Thread	2	4	8
63	0.060	0.041	0.021	0.010
64	0.062	0.041	0.021	0.011
65	0.065	0.045	0.023	0.013
127	0.480	0.328	0.164	0.083
128	0.488	0.337	0.169	0.084
129	0.515	0.348	0.175	0.090
511	30.57	20.67	10.46	5.27
512	30.83	20.92	10.52	5.27
513	32.05	21.79	10.60	5.34
1535	830.6	416.7	292.4	141.7
1536	831.2	416.9	283.2	142.1
1537	831.6	418.1	283.8	142.7

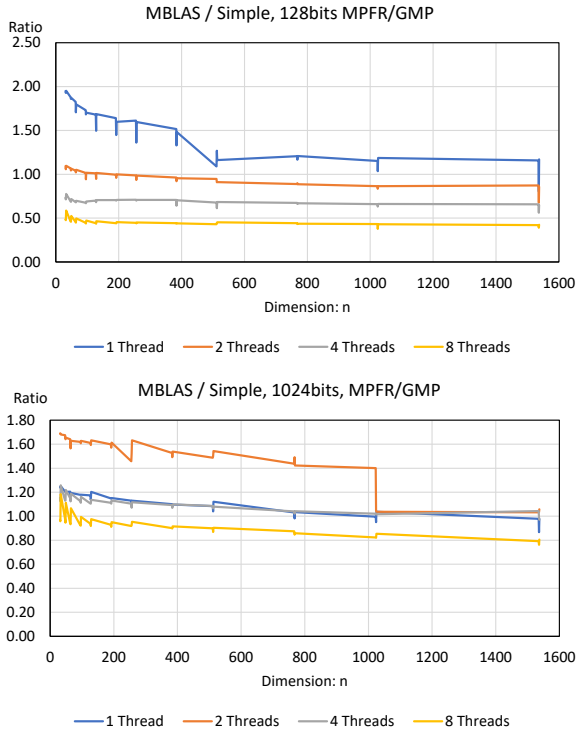

図 3 単純行列乗算 (Simple) の対 MBLAS 性能比: 128bits(上), 1024bits(下)

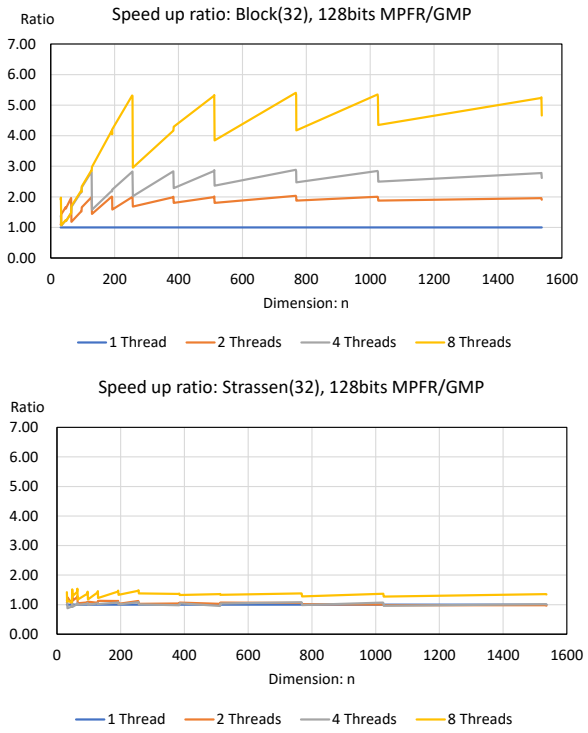

図 4 Block と Strassen の並列化効率 (128bits): Block(32)(上), Strassen(32)(下)

その結果を図 4 に示す．

ブロック化アルゴリズムの方は順当にスレッド数に応じた並列化効率が得られていることが分かる．但し，ブロック化の境界に当たるサイズでは極端に並列化効率が落ちる，つまり計算時間の低減がうまく図れておらず，荒っぽいノコギリのような図形となる．対して Strassen アルゴリズムの方は，並列化効率が殆ど上がっていないことが分かる．

この結果から予想できる通り，128bits 計算では，スレッド数が大きくなると，小さいサイズを除いてブロック化アルゴリズムの方が高速に実行できるようになる．実際，MBLAS の計算時間をブロック化アルゴリズムの計算時間と Strassen のアルゴリズムの計算時間で割った比率で表してみると，図 5 に示すように，1 スレッドでは Strassen の方が高速だが，スレッド数が上がるにつれてブロック化アルゴリズムの方が MBLAS より高速になることが分かる．

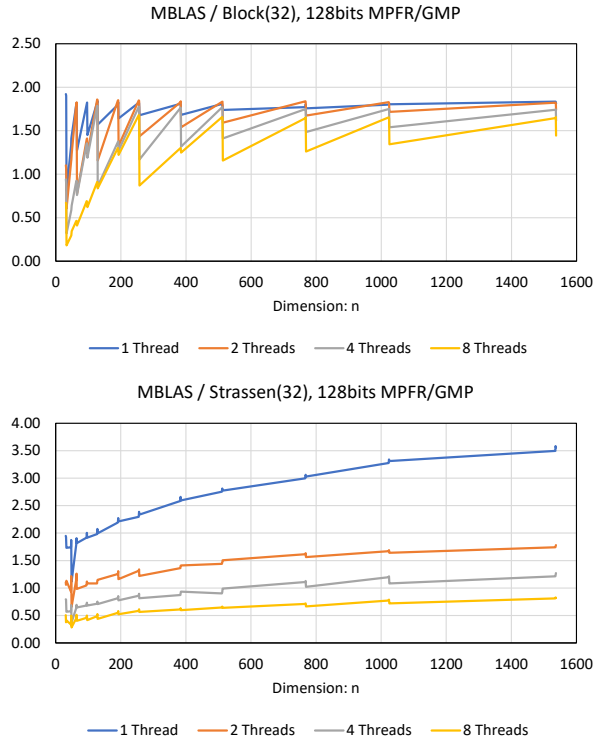

図 5 対 MBLAS 性能比 (128bits): Block(32)(上), Strassen(32)(下)

特にサイズが大きい時の，最小計算時間の比較の図を表 2 に示す．$n = 1023, 1024, 1025$ の時は 1.34〜1.65 倍，$n = 1535, 1536, 1537$ の時は 1.44〜1.66 倍，MBLAS より我々の実装したブロック化アルゴリズムの方が高速になっている．

表 2 最小計算時間の比較:128bits, 8 Threads

	Unit: seconds		MBLAS/
n	MBLAS	Block(32)	Block(32)
1023	15.5	9.4	1.65
1024	15.5	9.4	1.65
1025	15.5	11.5	1.34
1535	52.1	31.7	1.65
1536	52.4	31.6	1.66
1537	52.3	36.2	1.44

3.3 MPFR/GMP 1024bits 計算

上記と同様に，MPFR 1024bits(10 進約 308 桁) で，Block($n_{\min}$) と Strassen($n_{\min}$) の計算時間と並列化効率を調べた．その結果を図 6 に示す．

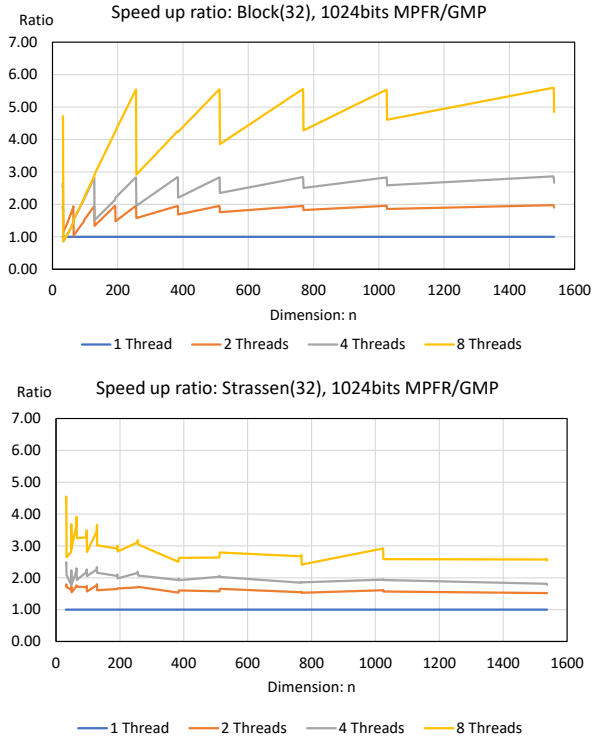

図 6 Block と Strassen の並列化効率 (1024its): Block(32)(上), Strassen(32)(下)

図 6 を見ると，Block(32) の結果は 128bits 計算時とほとんど同じであるが，Strassen(32) では並列化効率が上がっていることが分かる．精度桁数が大きくなると，Strassen アルゴリズム適用による演算量低減の効果がより高まることから，並列化効率が上がっていることも手伝って，全般的に Strassen(32) の方が Block(32) よりも高速になる．

実際，MBLAS との計算時間の比較を行った結果を図 7 に示す．

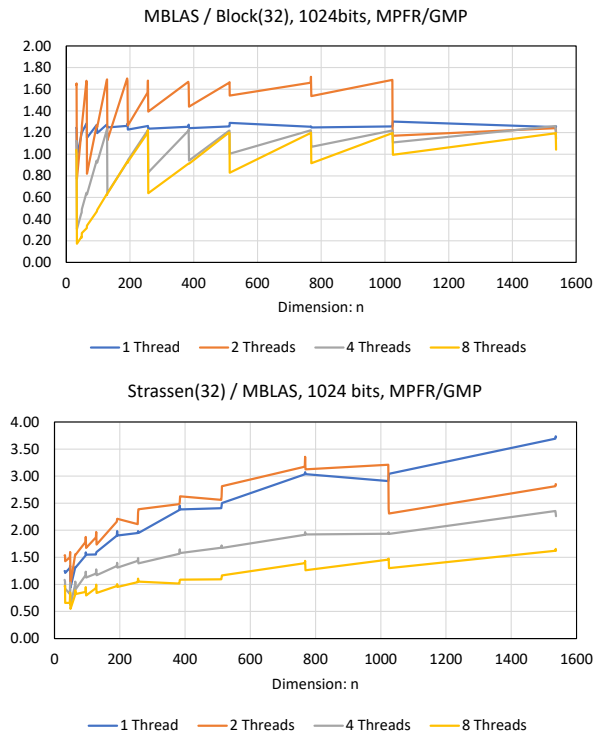

図 7 対 MBLAS 性能比 (1024bits): Block(32)(上), Strassen(32)(下)

この精度桁数では，Block(32) は MBLAS と同程度の性能であることが分かる．それに比べ，Strassen(32) は全てのスレッド数で MBLAS を上回る計算速度となっており，最大 8 スレッドでも表 3 に示す通り，$n = 1023, 1024, 1025$ で 1.30〜1.48 倍，$n = 1535, 1536, 1537$ では 1.62〜1.66 倍の高速性を示している．

表 3 最小計算時間の比較:1024bits, 8 Threads

	Unit: seconds		MBLAS/
n	MBLAS	Strassen(32)	Strassen(32)
1023	42.0	28.8	1.46
1024	42.1	28.5	1.48
1025	42.4	32.6	1.30
1535	141.7	87.4	1.62
1536	142.1	85.8	1.66
1537	142.7	87.8	1.62

4. 結論と今後の課題

以上の結果より，我々の BNCmatmul は，大きいサイズでは，精度桁数が小さい場合はブロック化アルゴリズム，大きい場合は Strassen アルゴリズムが高速に実行でき，MBLAS より計算時間を短縮できることが示された．現状では Strassen のアルゴリズムの並列化効率が悪いため，コア数がもっと多い環境では不利になる可能性があるものの，市販の CPU がサポートする範囲のコア数では有効性を示すことができていると言える．

今後の課題としては，

(1) C++環境下において，DD 精度，QD 精度の高速な実行列乗算の実現
(2) 行列サイズと計算精度に応じた最適なアルゴリズムを選択する多倍長行列乗算関数の実現

といったものが挙げられる．

参考文献

[1] D.H. Bailey, *QD*, http://crd.lbl.gov/~dhbailey/mpdist/.
[2] MPFR Project, *The MPFR library*, http://www.mpfr.org/.
[3] T.Granlaud and GMP development team, *The GNU Multiple Precision arithmetic library*, http://gmplib.org/.
[4] T.Kouya, *Accelerated multiple precision matrix multiplication using Strassen's algorithm and Winograd's variant*, *JSIAM Letters*, Vol. 6, pp. 81–84, 2014.
[5] T.Kouya, *Performance evaluation of multiple precision matrix multiplications using parallelized Strassen and Winograd algorithms*, *JSIAM Letters*, Vol. 8, pp. 21–24, 2016.
[6] T.Kouya, *Efficient Multiple Precision Dense Matrix Multiplication Library with Automatic Turning Tool*, SIAM PP 2018, 2018.
[7] *ATLAS: Automatically Tuned Linear Algebra Software*, http://math-atlas.sourceforge.net/.
[8] T.Kouya. *BNCpack*, http://na-inet.jp/na/bnc/.
[9] T.Kouya. *BNCmatmul*, http://na-inet.jp/na/bnc/bncmatmul-0.2.tar.bz2.
[10] M.Nakata, *MPLAPACK, MBLAS*, https://github.com/nakatamaho/mplapack
[11] T.Nakayama, *CUMP*, https://github.com/skystar0227/CUMP
[12] M. Joldes, J.-M.Muller, V.Popescu, W.Tucker, *CAMPARY: Cuda Multiple Precision Arithmetic Library and Applications*, ICMS 2016, 2016.
[13] D.Mukunoki, T.Daisuke, *Performance Comparison of Double, Triple and Quadruple Precision Real and Complex BLAS Subroutines on GPUs*, ATIP '12, 2012.
[14] 菱沼利彰, 中田真秀, PEZY-SC2 上における倍々精度演算ライブラリ pzqd を用いた倍々精度 Rgemm の高速化, 研究報告ハイパフォーマンスコンピューティング (2018-HPC-167), No.10, 2018.
[15] 松本和也, 中里直人, StanislavSedukhin, OpenCL による行列乗算カーネル実装と性能評価, 研究報告ハイパフォーマンスコンピューティング (2012-HPC-135), No.39, 2012.