Vol.2020-HPC-173 No.13
2020/3/17

3倍精度行列乗算の性能評価

幸谷 智紀[1,a)]

概要：Strassen 及び Winograd アルゴリズムは分割統治法の一種で，計算量が最も少なくなる方法として知られている．我々は既に MPFR や QD といった多倍長精度浮動小数点ライブラリを用いてこれらのアルゴリズムを実装し，その有効性を示すと共にオープンソースコードとして公開している．将来のリリースに向け，我々は今回最適化された，Fabiano らが提唱した 3 倍精度演算を導入し，その有効性を行列乗算で示すことができた．本稿ではその結果を，QD や MPFR ベースの実装と比較しつつ示す．

キーワード：3 倍精度浮動小数点演算，行列乗算，分割統治法，並列化

Performance evaluation of triple-double precision matrix multiplication

Abstract: Strassen and Winograd matrix multiplication can be categorized in divide-and-conquer algorithms, and are well known as the most efficient ones. We have already implemented them supporting multiple precision floating-point arithmetic using MPFR and Bailey's QD library, and have shown their effectiveness in our papers and open-source codes. In preparation for a future release, we have now introduced optimized triple word floating-point arithmetic proposed by Fabiano et.al. and then found its utility in our implementation of multiple precision matrix multiplication. In this paper, we demonstrate the effectiveness of Strassen and Winograd triple-double precision matrix multiplication through performance evaluation compared to those based on QD and MPFR.

Keywords: triple-double precision floating-point arithmetic, matrix multiplication, divide-and-conquare algorithm, parallelization

1. 初めに

科学技術計算において，ユーザの要求する精度を持つ数値演算結果をできるだけ高速に求めるニーズは常に存在する．対象となる問題が丸め誤差に鋭敏な悪条件問題であれば，IEEE754 倍精度浮動小数点数 (binary64) より仮数部の長い多倍長浮動小数点数を用いた演算が必要となる．そのような場合，現在では QD や MPFR といった歴史の長い標準的な多倍長浮動小数点演算ライブラリを使用することが普通である．任意精度浮動小数点演算をサポートする多数桁タイプの MPFR に対し，Bailey らの QD ライブラリは Double-double(DD, 106bits) と Quadruple-double(QD, 212bits) の二つの浮動小数点演算クラスのみサポートするマルチコンポーネントタイプのライブラリであるが，DD や QD と同じ仮数部長の浮動小数点演算は MPFR より高速に実行できるという強みがある．特に DD 演算は MPFR 106bits 計算より高速であり，QD でも除算以外は高速である．Fabiano らが提唱する 3 倍精度 (Triple word) 演算は，binary64 を 3 つ使用する Triple-double(TD) 演算が実現でき，DD と QD の中間精度をサポートできるもので，MPFR 159bits 演算よりも高速に実行できることが示されている．同じマルチコンポーネントタイプの任意精度浮動小数点演算ライブラリである CAMPARY[2] よりも最適化された四則演算と平方根のアルゴリズムが提唱されており，パフォーマンスも DD と QD の中間に位置づけられることが期待できる．

我々は Strassen および Winograd アルゴリズムを使用した多倍長精度行列乗算ライブラリ BNCmatmul を実装しており，OpenMP を使用した並列化も行って DD, QD, MPFR ベースの高速な行列乗算を実現している．同様の機能を持つ多倍長精度線型計算ライブラリ MPLAPACK(MBLAS)

[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

ⓒ 2020 Information Processing Society of Japan

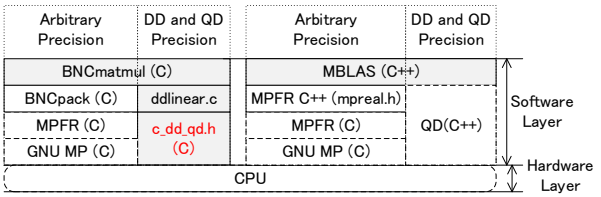


図 1 Software layers of BNCmatmul and MPL PACK(MBLAS)

の Rgemm ルーチンよりも，大きいサイズでは高速であ
今回我々はこの BNCmatmul に TD 演算を取り入れ，DD より約 5 倍，QD より 0.5 倍の計算時間で大きいサイズの行列乗算が実行できることを C インライン関数を用いた実装で確認した．

本稿ではまず我々の BNCmatmul のソフトウェア構成，アルゴリズムと並列化手法，既存の DD, QD，MPFR ベースの行列乗算のパフォーマンスを示す．次に TD 演算の概要を述べ，計算精度も計算時間もちょうど DD と QD の中間に位置づけられることを Intel と AMD CPU 環境下におけるベンチマークテストで示す．最後に結論と今後の課題を述べる．

2. BNCmatmul のソフトウェア構造と性能比較

我々は既に QD と MPFR をベースとした多倍長精度行列乗算ライブラリ BNCmatmul[11] を公開している．このライブラリの一番の特徴は OpenMP を用いて並列化された Strassen および Winograd アルゴリズムという分割統治法を採用していることである．これにより，サイズの大きい行列乗算は，MPLAPACK[6] より高速に実行できることが期待できる．

我々の BNCmatmul と MPLAPACK のソフトウェア階層を図 1 に示す．どちらも QD の提供する 106bits 精度桁の DD(double-double) 演算と 212bits 精度桁の QD(Quad-double) 精度の演算，MPFR の提供する任意精度浮動小数点演算を土台に構築されている．但し，MPLAPACK がどの精度演算も C++クラスを利用しているのに対し，我々の BNCmatmul は QD ベースの演算は独自の C インライン関数 (`c_dd_qd.h`) で，MPFR はネイティブ C 関数をそのまま利用しているという違いがある．

ここでは我々の BNCmatmul の提供する行列乗算性能を，MPLAPACK が提供する Rgemm 関数と比較しながら示すことにする．

2.1 Strassen および Winograd アルゴリズムとその並列化

Strassen および Winograd アルゴリズムは，再帰呼び出しを行うことで，行列サイズが大きくなればなるほど，通常の行列乗算より演算回数を減らせることが知られている．

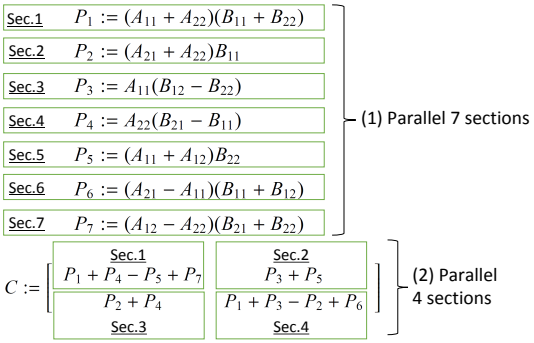


図 2 並列化 Strassen アルゴリズム

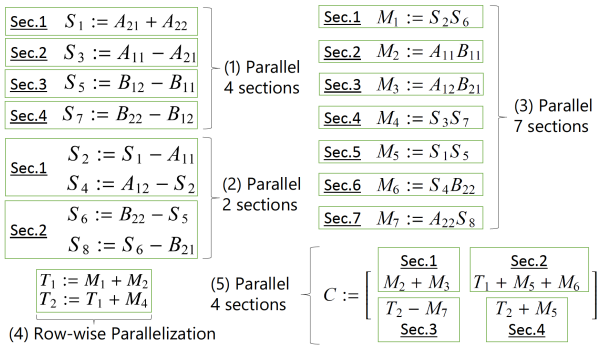


図 3 並列化 Winograd アルゴリズム

特に多倍長演算を使用する行列乗算に対してはその効果が高く，我々は既に QD および MPFR ベースの実装を行い，その効果を確認し，LU 分解の高速化が可能であることを示した [7][8]．使用したアルゴリズムとその並列化方法を図 2 と図 3 に示す．ここで行列 A，B，C は次のように行と列で 2 分割され，$C := AB$ が求められるものとする．

$$A := \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \; B := \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$
$$\Longrightarrow \; C := AB = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}.$$

我々の並列化 Strassen と Winograd アルゴリズムは，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．

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，QD ベースでも MPFR ベースでも計算時間及び並列化効率に大きな違いはない．

これにより，MPFR はもとより，(擬似的)4 倍精度 (DD)

および 8 倍精度 (QD) でも，Strassen および Winograd 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．しかしながら，QD ベースの並列化 Strassen および Winograd アルゴリズムに比較して，MPFR ベースのそれは並列化効率が劣ることが判明している．

2.2 行列乗算アルゴリズムと MPLAPACK との比較

我々の並列化 Strassen と Winograd アルゴリズムを実装した BNCmatmul と比較するため，MPLAPACK との性能評価を行う．計算機環境は Ryzen と Corei7 の二つを用意した．MPLAPACK は Github から 2019 年 12 月にダウンロードしたものをインストールし，使用する QD や MPFR ライブラリは MPLAPACK 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．

Ryzen AMD Ryzen 1700 (2.7 GHz), 32 GB RAM, Ubuntu 16.04.5 x86_64, GCC 5.4.0, MPACK 0.8.0, MPFR 4.0.2[4]/GMP 6.1.2[5], QD 2.3.22[3], BNCpack 0.8[10]

Corei7 Intel Core i7-9700K (3.6GHz), 16 GB RAM, Ubuntu 18.04.2 x86_64, GCC 7.3.0, MPACK 0.8.0, MPFR 4.0.2[4]/GMP 6.1.2[5], QD 2.3.22[3], BNCpack 0.8[10]

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

MPLAPACK の提供する BLAS 相当の多倍長精度基本線型計算は，BLAS と同じ引数を取るように作られており，行列要素は列優先 (Column-major) 方式で格納されるようになっている．今回使用したのは 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．

我々の提供する BNCmatmul の行列乗算関数は前述したように全て C 関数として実装されているが，MPLAPACK と比較するために，同じ C++ベンチマークプログラムから呼び出すようにしている．

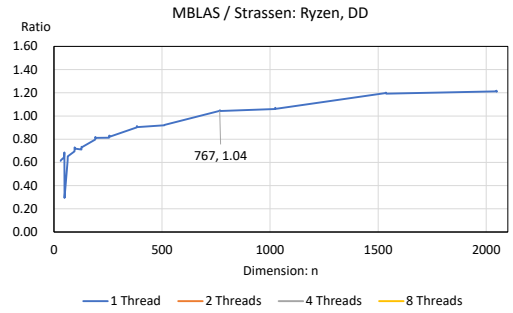
使用した実正方行列 A，B は次の通りである．

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

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計算桁数より 10 進 1〜2 桁の精度低下がみられる程度であった．

2.2.1 QD 行列乗算の性能比較

QD ベースの行列乗算の計算時間を比較したグラフを図 4，5 に示す．残念ながら，現在の MPLAPACK では QD ベースの Rgemm 関数は並列化がなされていないため，逐次実行した時の計算時間の比 (Rgemm/Strassen) のみをプロットした．この比が 1 以上になった時は我々の行列乗算の方が高速であることを意味している．DD 精度，QD 精度



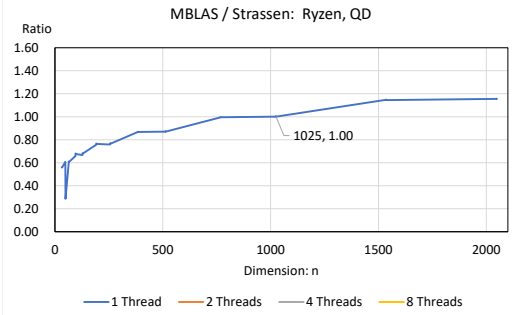
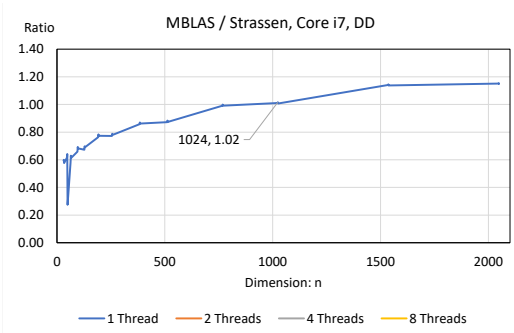


図 4 QD ベースの行列乗算性能比較: Rgemm / Strassen, Ryzen



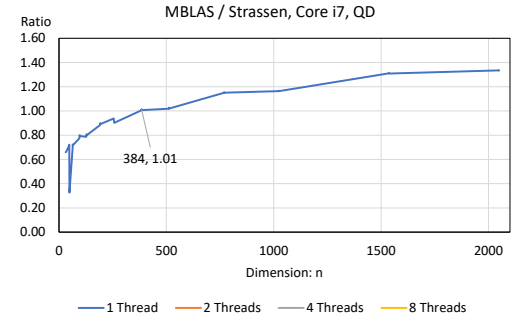


図 5 QD ベースの行列乗算性能比較: Rgemm / Strassen, Corei7

どちらも一定以上のサイズになった時に，我々の Strassen アルゴリズム実装が高速になっていることが分かる．

我々の並列化 Strassen アルゴリズムの並列化効率を求めたものを図 6，7 に示す．これは逐次計算時間を並列実行時の計算時間で割った比をプロットしたものである．DD 精度，QD 精度，どちらの結果も良好な並列化効率を示していることが分かる．

これらの結果から，DD および QD 精度演算と同じ無誤差変換技法を用いた 3 倍精度 (TD, Triple-double) の実装も同様の並列化効率となることが期待できる．

ⓒ 2020 Information Processing Society of Japan

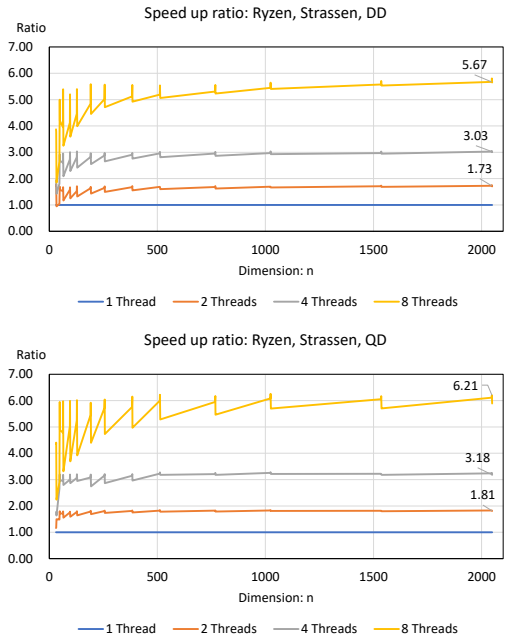


図 6　QD ベースの並列化 Strassen アルゴリズムの並列化効率

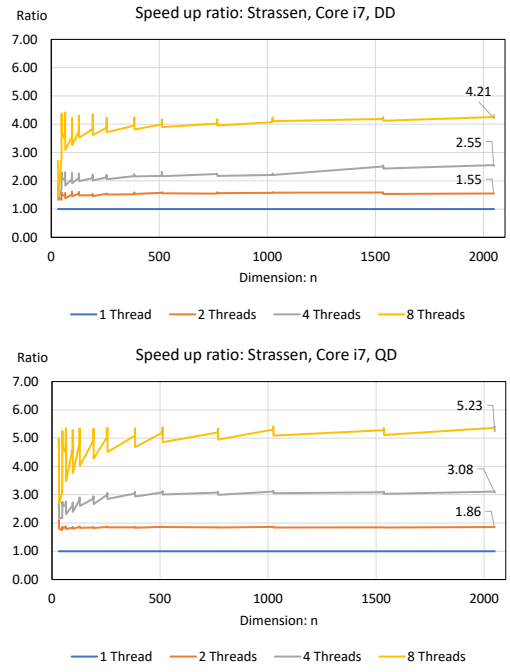


図 7　QD ベースの並列化 Strassen アルゴリズムの並列化効率

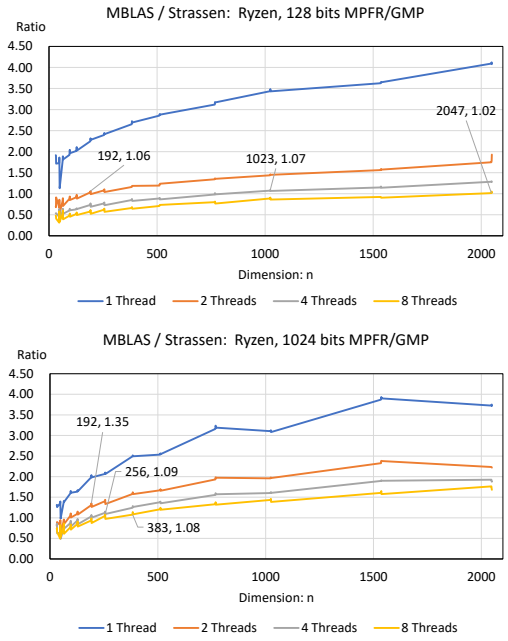


図 8　MPFR ベースの行列乗算性能比: Rgemm / Strassen

2.2.2　MPFR 行列乗算の性能比較

MPFR ベースの行列乗算の結果も，QD との比較のために図 8 に示す．MPFR ベースの Rgemm は OpenMP によって並列化されており，良好な並列化効果を発揮する．実際，並列化した Strassen のアルゴリズムと比較すると，128bits 計算でも 1024bits 計算でも，スレッド数が多くなるにつれて Rgemm との性能比が小さくなっていることが分かる．

QD ライブラリベースの演算に比べ，MPFR による行列乗算は，DD 精度相当 (MPFR 106bits) で約 18 倍，QD 精度相当 (MPFR 212bits) で約 2 倍低速になる．後述する TD 精度でも，最も低速な単純行列積で比較して MPFR 159bits では約 2 倍低速になることが判明している．

3.　3 倍精度演算のアルゴリズム

最適化された 3 倍精度演算は Fabiano ら [1] によって提唱されたもので，QD ライブラリが提供する，無誤差変換技法を用いたマルチコンポーネント方式の多倍長浮動小数点演算を実現するものである．我々の 3 倍精度行列乗算実装も彼らの提唱した加算 (TDadd) と Fast 乗算 (TDmul) を用いている．以下，使用した 3 倍精度演算のアルゴリズムを示す．

ベースとなる無誤差変換技法は QuickTwoSum, TwoSum, TwoProd[9] である．今回の実装では倍精度浮動小数点数を個々の変数に使用している．なお，TwoProd には FMA 演算 (Fused Multiply-Add) を用いたものを使用している．

3 倍精度浮動小数点演算は，既存の浮動小数点数を 3 つ使用して表現した浮動小数点数を用いて実行する．ここでは

ⓒ 2020 Information Processing Society of Japan

3 倍精度浮動小数点数を $x := (x_0, x_1, x_2)$ として表現する．

3 倍精度浮動小数点演算のためには演算結果を正規化する必要がある．Fabiano らは VecSum と VSEB(k) (VecSum with Blanch) を組み合わせて，演算結果を正規化するようにしている．

加算 (TDadd) は，$x := (x_0, x_1, x_2)$, $y := (y_0, y_1, y_2)$ の和，即ち $r = (r_0, r_1, r_2) := x + y$ を求めるものである．まず最初に x と y をマージしてから VecSum で正規化し，しかる後に VSEB(3) で 3 倍精度浮動小数点数として正規化して z を返す．

Algorithm 1 $r :=$ TDsum(x, y)

$(z_0, ..., z_5) :=$ Merge$(x_0, x_1, x_2, y_0, y_1, y_2)$
$(e_0, ..., e_5) :=$ VecSum$(z_0, ..., z_5)$
$(r_0, r_1, r_2) :=$ VSEB(3)$(e_0, ..., e_5)$
return (r_0, r_1, r_2)

Fabiano らは Accurate 乗算と，演算数の少ない Fast 乗算の二つを提唱している．我々は後者の乗算を TDmul として実装した．

Algorithm 2 $r :=$ TDmul(x, y)

$(z_{00}^{\mathrm{up}}, z_{00}^{\mathrm{lo}}) :=$ TwoProd(x_0, y_0); $(z_{01}^{\mathrm{up}}, z_{01}^{\mathrm{lo}}) :=$ TwoProd(x_0, y_1)
$(z_{10}^{\mathrm{up}}, z_{10}^{\mathrm{lo}}) :=$ TwoProd(x_1, y_0)
$(b_0, b_1, b_2) :=$ VecSum$(z_{00}^{\mathrm{lo}}, z_{01}^{\mathrm{up}}, z_{10}^{\mathrm{up}})$
$c :=$ FMA(x_1, y_1, b_2); $z_{31} :=$ FMA$(x_0, y_2, z_{10}^{\mathrm{lo}})$
$z_{32} :=$ FMA$(x_2, y_0, z_{01}^{\mathrm{lo}})$
$z_3 := z_{31} \oplus z_{32}$; $s_3 := c \oplus z_3$
$(e_0, e_1, e_2, e_3) :=$ VecSum$(z_{00}^{\mathrm{up}}, b_0, b_1, s_3)$
$r_0 := e_0$; $(r_1, r_2) :=$ VSEB(2)(e_1, e_2, e_3)
return (r_0, r_1, r_2)

行列乗算のためには TDadd と TDmul があれば実装ができる．我々の実装にあたっては，既に存在する `c_dd_qd.h` に定義された無誤差変換技法をインライン関数として呼び出し，この 2 つの 3 倍精度演算を実装した．

DD 演算，TD 演算，QD 演算の加算と乗算の演算量を比較すると，TD 演算は DD 演算の 3.8 倍，QD 演算の 0.3 倍の演算量となる．また，演算ごとの誤差もそれぞれ $O(2^{-106}) = O(10^{-32})$, $O(2^{-159}) = O(10^{-48})$, $O(2^{-212}) = O(10^{-64})$ となることから，DD 演算では精度が足りず，QD 演算までは必要のない，ちょうど中間の精度で十分な場合に有効な演算と言える．

4. TD 精度行列乗算のベンチマークテスト

実装した TD 行列乗算の結果を図 9, 10 に示す．比較のため，DD, QD 演算の Strassen アルゴリズムの計算時間も並べておく．Ryzen および Corei7 どちらも 8 コアあることから，逐次計算結果と，8 スレッド並列計算の結果もまとめてある．

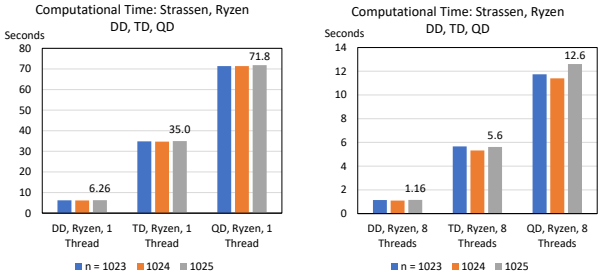


図 9 3 倍精度 Strassen 行列乗算の計算時間: Ryzen

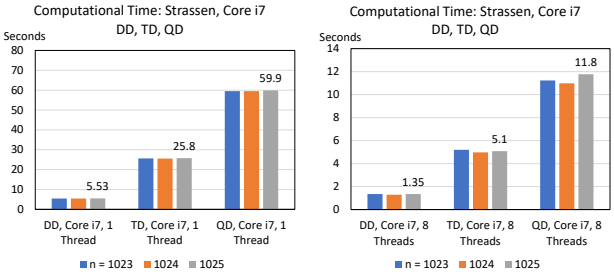


図 10 3 倍精度 Strassen 行列乗算の計算時間: Corei7

ⓒ 2020 Information Processing Society of Japan

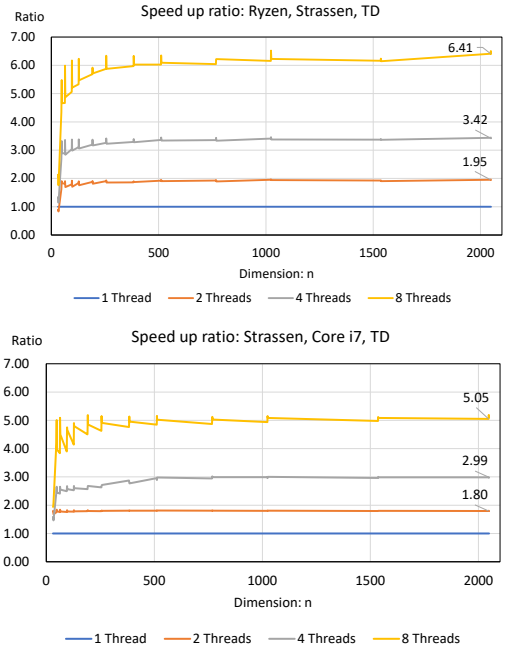


図 11　3 倍精度 Strassen 行列乗算の並列化性能向上比

この結果，スレッド数に依らず，TD 行列乗算は，DD の約 5 倍，QD の半分程度の計算時間で済むことが分かる．

TD 精度の並列化 Strassen アルゴリズムの並列化性能向上比を図 11 に示す．

概ね，DD 精度および QD 精度の並列化性能向上比と同じく，スレッド数に応じた性能向上を見ることができる．

4.1　Winograd アルゴリズムの性能評価

Winograd アルゴリズムは前述したように Strassen アルゴリズムより加減算量が少なくなる．しかしアルゴリズムが複雑になること，多倍長精度浮動小数点演算では乗算が加減算よりコストを要するため，実際にベンチマークテストを行ってみると，低精度で行列サイズの小さい場合は Strassen アルゴリズムよりも劣ることが多い．実際に Strassen アルゴリズムと DD, TD, QD 精度で行列乗算を行い，計算時間短縮化の割合を求めた結果を図 12 と図 13 に示す．

どちらの環境でも DD → TD → QD と精度を増やすにつれ，行列サイズが大きくなると Winograd アルゴリズムの方が 1〜2 割程度は高速になる率が高い．しかし Corei7 では並列度によっては大きく劣るところもあり，我々の現状の実装では必ずしも高速になるとは言い切れない．

5.　結論と今後の課題

我々の並列化 Strassen および Winograd アルゴリズムに基づく 3 倍精度行列乗算ライブラリは，期待通り，DD 精度の 5 倍，QD 精度の 0.5 倍程度の計算時間で実行でき，並列化効率も良好なものを Ryzen および Corei7 両環境で示すことが確認できた．QD ライブラリには今のところ TD

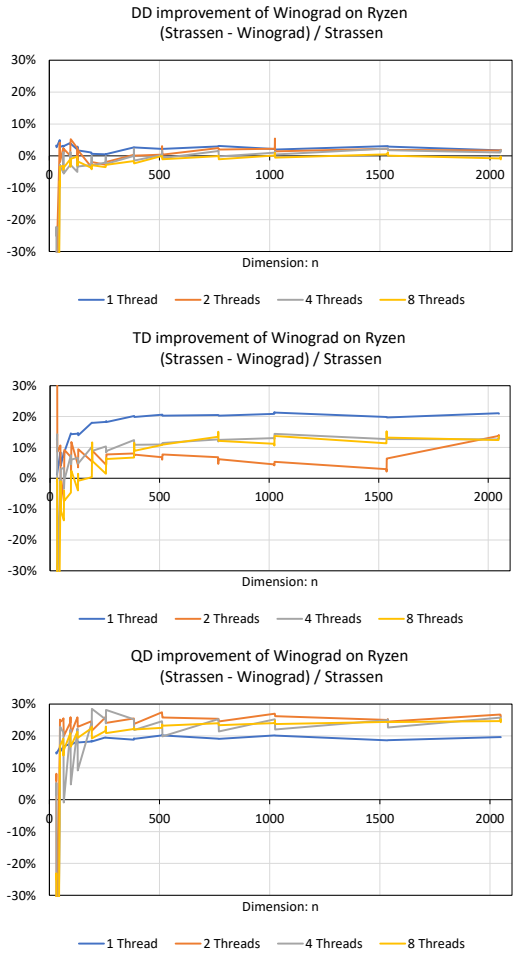


図 12　Winograd アルゴリズムの計算性能向上比: Ryzen

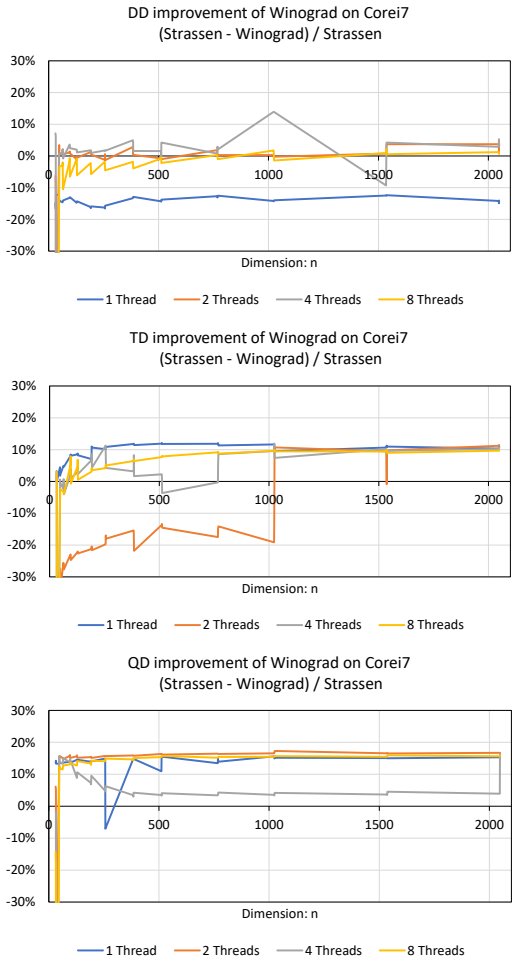


図 13 Winograd アルゴリズムの計算性能向上比: Corei7

実装がないことから，MPLAPACK との計算時間の比較はできないが，仮に存在したとしてもより高速であることが期待できる．

今後の課題としては，TD 演算機能を組み込んだ新しい BNCmatmul を作成し，LU 分解にも適用して有用性を確認することが挙げられる．DD, TD, QD 精度の 3 つを自在に利用できるようになることで，ユーザの要求する精度に応じた適切な計算精度桁数の設定ができ，212bits 以下でより高速かつ高精度な数値計算が必要なニーズには適切に答えることができるようになる．

参考文献

[1] N. Fabiano and J. Muller and J. Picot, Algorithms for Triple-Word Arithmetic, IEEE Trans. on Computers, Vol.68, No.11, pp.1573–1583, 2019.
[2] M. Jolders and J-. M. Muller and V. Popescu and W.Tucker, CAMPARY: Cuda mutiple precision arithmetic library and applications, 5th ICMS,2016.
[3] D.H. Bailey, QD, `http://crd.lbl.gov/~dhbailey/mpdist/`.
[4] MPFR Project, The MPFR library, `http://www.mpfr.org/`.
[5] T.Granlaud and GMP development team, The GNU Multiple Precision arithmetic library. `http://gmplib.org/`.
[6] M.Nakata, MPLAPACK(MBLAS), `https://github.com/nakatamaho/mplapack`.
[7] T.Kouya, Accelerated multiple precision matrix multiplication using Strassen's algorithm and Winograd's variant, *JSIAM Letters*, Vol. 6, pp. 81–84, 2014.
[8] T.Kouya, Performance evaluation of multiple precision matrix multiplications using parallelized Strassen and Winograd algorithms, *JSIAM Letters*, Vol. 8, pp. 21–24, 2016.
[9] 幸谷智紀, 多倍長精度数値計算，森北出版， 2019.
[10] T.Kouya. BNCpack, `http://na-inet.jp/na/bnc/`.
[11] T.Kouya. BNCmatmul, `http://na-inet.jp/na/bnc/bncmatmul-0.2.tar.bz2`.

ⓒ 2020 Information Processing Society of Japan