Strassenのアルゴリズムを用いた4倍精度，8倍精度実行列積の高速化
～多倍長行列積ライブラリBNCmatmulの構築と応用～

幸谷 智紀
静岡理工科大学　総合情報学部

http://na-inet.jp/na/bnc/bncmatmul-0.12.tar.bz2

1. 目的

●多倍長浮動小数点数を用いた計算＝多倍長計算
　[DD/QD]　　IEEE754倍精度実数演算をベースにしたBaileyによる4倍精度(DD)・8倍精度浮動小数点数(QD)の実装
　[MPFR/GMP] GMP(GNU MP)のmpn(多倍長自然数演算)カーネルをベースとした任意精度浮動小数点演算ライブラリ

⇩

●多倍長計算はIEEE754単精度・倍精度より多大な計算時間を要する
●Strassenのアルゴリズムは行列積の演算量を減らすことができる

⇩

☆MPFR/GMPを用いたStrassenのアルゴリズムは効果大

⇩ (cf. JSIAM Letters Vol.6, pp.81-84)

DD/QDでも効果が見込めるのでは？

2. 行列積のアルゴリズム

$$C := AB$$

(A, B は偶数列数・偶数行数の行列)

行・列を2等分割してブロック化

⇨ $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, $B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$

●Strassenのアルゴリズム

$$P_1 := (A_{11} + A_{22})(B_{11} + B_{22})$$
$$P_2 := (A_{21} + A_{22})B_{11}$$
$$P_3 := A_{11}(B_{12} - B_{22})$$
$$P_4 := A_{22}(B_{21} - B_{11})$$
$$P_5 := (A_{11} + A_{12})B_{22}$$
$$P_6 := (A_{21} - A_{11})(B_{11} + B_{12})$$
$$P_7 := (A_{12} - A_{22})(B_{21} + B_{22})$$

⇩

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

↑ 再帰的適用で演算量が減らせる→ $C := \begin{bmatrix} M_2 + M_3 & T_1 + M_5 + M_6 \\ T_2 - M_7 & T_2 + M_5 \end{bmatrix}$

●Winogradの改良アルゴリズム

$$S_1 := A_{21} + A_{22} \quad M_1 := S_2 S_6$$
$$S_2 := S_1 - A_{11} \quad M_2 := A_{11} B_{11}$$
$$S_3 := A_{11} - A_{21} \quad M_3 := A_{12} B_{21}$$
$$S_4 := A_{12} - S_2 \quad M_4 := S_3 S_7$$
$$S_5 := B_{12} - B_{11} \quad M_5 := S_1 S_5$$
$$S_6 := B_{22} - S_5 \quad M_6 := S_4 B_{22}$$
$$S_7 := B_{22} - B_{12} \quad M_7 := A_{22} S_8$$
$$S_8 := S_6 - B_{21} \quad T_1 := M_1 + M_2$$
$$T_2 := T_1 + M_4$$

●奇数行数・奇数列数の場合の処理
　・パディング・・・0で埋める

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

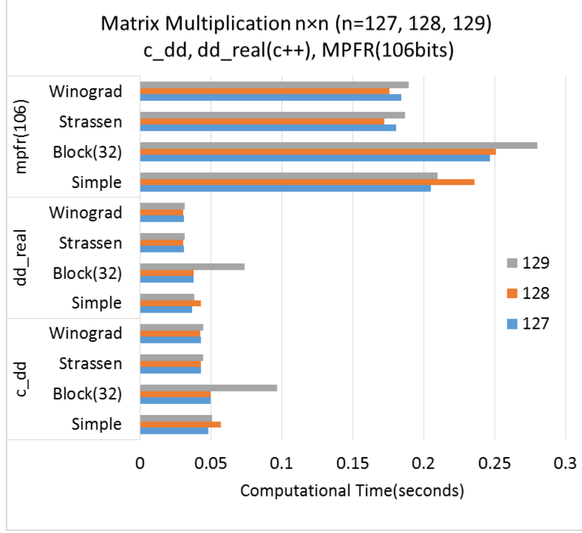
　・ピーリング・・・余剰部分を別途計算

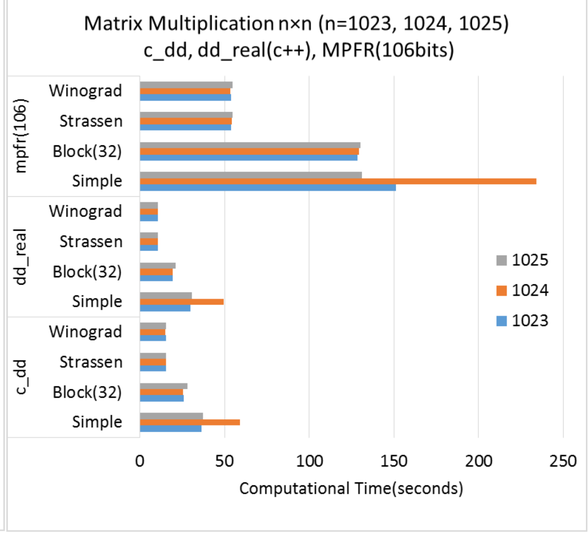
$$A := \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right], B := \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right] \Longrightarrow C := AB = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]$$

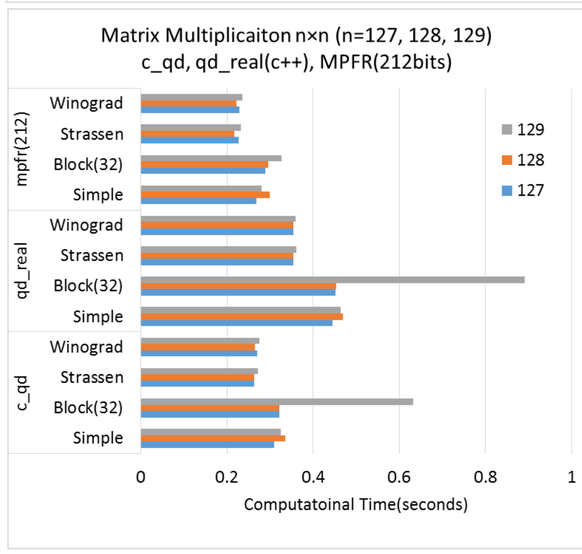
⇒混合して動的にメモリ領域を確保しつつ使用
⇒任意の行数・列数の行列積が可能に

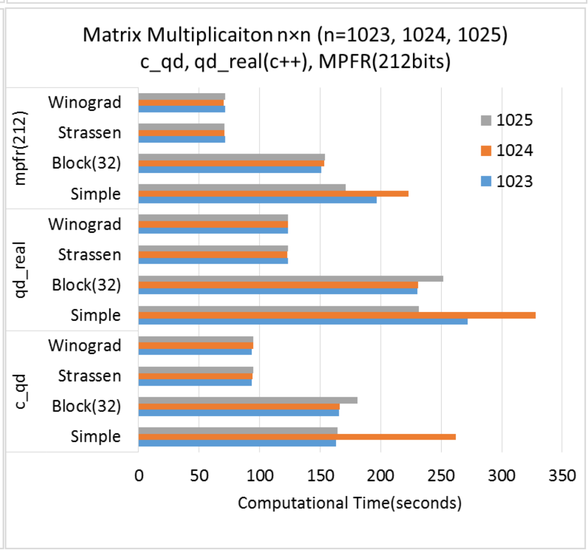
3. 実正方行列積のベンチマークテスト

●計算環境
[H/W] Intel Xeon E5-2620 v2(2.10GHz)×2 = 12 cores
[S/W] CentOS 6.5 x86_64, Intel C/C++ compiler 13.1.3,
MPFR 3.1.2/GMP 6.0.0a, http://www.mpfr.org/
QD 2.3.15 (C++ & C), http://crd-legacy.lbl.gov/~dhbailey/mpdist/









・4倍精度ではDD(C++)が最高速
・8倍精度ではMPFR(212bits)が最高速
・Strassen, Winogradアルゴリズムは4倍精度，8倍精度でも有効

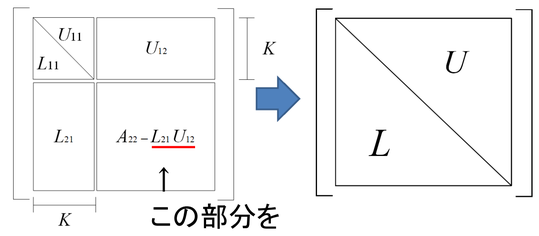
4. LU分解への応用

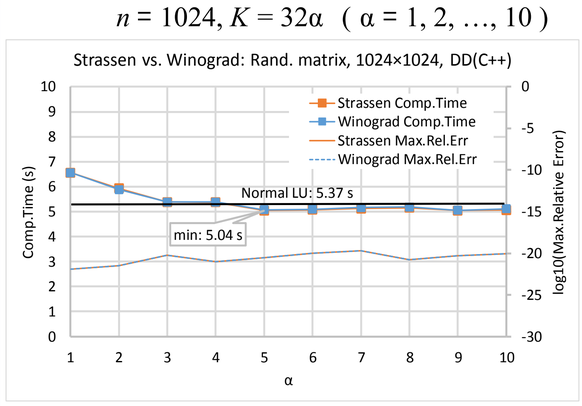
連立一次方程式を直接法で解く

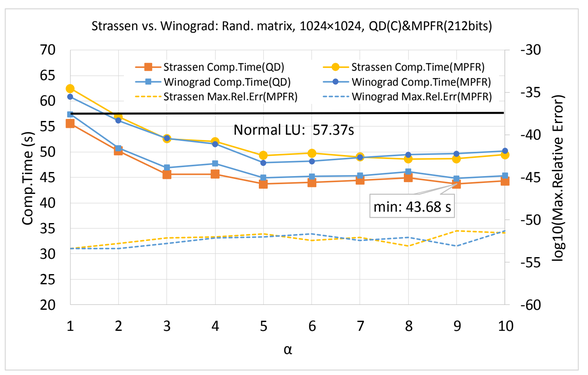
$$A\mathbf{x} = \mathbf{b}$$

$A \in \mathbb{R}^{n \times n}$　$\mathbf{b} \in \mathbb{R}^n$

行列成分は[-1,1]の一様乱数
$n = 1024, K = 32\alpha$　($\alpha = 1, 2, \ldots, 10$)







・高速な4倍精度(DD(C++))での計算時間削減は6.1%
・8倍精度ではQD(C)がMPFR(212bits)より高速，23.9%の計算時間減
・低速な行列積計算に対しての効果が大≒高精度になるほど効果大

5. 今後の課題

①効率的な並列化の追求→OpenMP化したが，並列化効率が頭打ち
②GPU化
　→GQDライブラリを使って単純行列積(Simple)は高速化可能
　→①を達成してからStrassen, Winogradアルゴリズムの実装へ
　→さらなる応用に展開していきたい