CB-005

並列処理を用いた対話的多倍長演算環境 MuPAT の高速化

Acceleration of interactive multi-precision arithmetic toolbox MuPAT using parallel processing

八木 武尊† 長谷川 秀彦‡ 石渡 恵美子†
Hotaka Yagi Hidehiko Hasegawa Emiko Ishiwata

1. はじめに

現在，ほとんどのコンピュータが浮動小数点数の規格として，IEEE754-2008 を採用し，ハードウェアで実装された浮動小数点数の演算は非常に高速である．演算精度は，単精度は 10 進 7 桁，倍精度は 16 桁であるが，それより高精度に計算したほうが良い場面が色々ある． たとえば，Krylov 部分空間法では高精度演算を用いると，収束しなかった問題が収束したり，反復回数が減ることがある[1]．また，半正定値計画法[2]や，非対称固有値問題の解析を行う場合など，高精度演算が必要となる場合が多い．

一般のコンピュータで高精度演算を実装する方法として，倍精度数を 2 つ組み合わせて擬似 4 倍精度を実現する Double-double (DD)演算[3]，倍精度数 4 つを組み合わせて擬似 8 倍精度を実現する Quad-double (QD)演算がある[4]．著者らは，これらを Scilab と Matlab 上に実装し，Multiple Precision Arithmetic Toolbox (MuPAT)として提案した[5]．

しかしながら，DD/QD 演算は倍精度演算による四則演算の回数が非常に多い．また，Matlab 上では，インタプリタ形式での実行となるため，実行速度が遅くなる．本研究では，対話的な PC 利用環境を想定した MuPAT の高速化を行う．具体的には，CPU に備わっている Fused-Multiply-and-Add(FMA)[6,7]，Advanced vector extensions (AVX) [6,7]，OpenMP [8]などを用いて高速化し，ルーフラインモデル[9]を用いて性能を評価する．これらの高速化手法によって理論演算性能は最大で 32 倍となる．実際， 4 コアの環境で n=4096 の DD/QD 演算の行列積だと，高速化しない場合と，FMA，AVX2，OpenMP のすべてを用いて高速化した場合の比較では，DD 演算で 17 倍(ピーク性能の 44%)，QD 演算で 16 倍(同 38%)となった．

2 章で DD/QD 演算，MuPAT について紹介し，3 章で FMA，AVX2，OpenMP やその組み合わせによる MuPAT の高速化を検討する．4 章で数値実験を示し，5 章でまとめと今後の課題を述べる．

2. 多倍長演算と MuPAT

2.1 Double-Double(DD)と Quad-Double(QD)

Double-double (DD)演算[3]は，2 つの倍精度数を組み合わせて擬似 4 倍精度数を，Quad-double (QD)演算[4]は，4 つの倍精度数を組み合わせて擬似 8 倍精度数を表現する手法である．例えば，DD 精度の数 A は 2 つの倍精度数a_0 とa_1 を用いて$A = a_0 + a_1$のように表現される．ただし，上位の数a_0と下位の数a_1の間には次の関係が成り立つ．

$$|a_1| \le 1/2\ ulp\ (a_0)$$

ここで，$ulp(x)$ は”units in the last place”の略で，その浮動小数点数で表現可能な最小単位のことである．最上位項のa_0は DD 精度の数 Aの倍精度数への近似であり，その誤差は$ulp\ (a_0)$ の半分以下である．

同様に， QD 精度の数 B は $B = b_0 + b_1 + b_2 + b_3$ と表現され $|b_{i+1}| \le 1/2\ ulp(b_i)$， $(i = 0,1,2)$をみたす．

倍精度数が符号部 1bit，指数部 11bit，仮数部 52bit で構成されていることから，DD 数は仮数部 104bit，QD 数で仮数部 208bit となる．符号部と指数部は倍精度と同じ bit 数である．このことから，有効桁数は DD 精度では 10 進で約 31 桁，QD 精度では 10 進で約 63 桁となっている．

DD 精度の四則演算は，Dekker[10]と Knuth[11]の丸め誤差のない倍精度加算と乗算アルゴリズム，QD 精度の四則演算は Hida[4]による倍精度加算と乗算アルゴリズムに基づき，倍精度の四則演算の組み合わせのみで実現できる．表 1 で DD/QD 精度の四則演算に必要な倍精度演算回数を表す．

表 1 DD/QD 演算に必要な倍精度演算回数

		add & sub	mul	div	Total
DD	add & sub	11	0	0	11
	mul	15	9	0	24
	div	17	8	2	27
QD	add & sub	91	0	0	91
	mul	171	46	0	217
	div	579	66	4	649

DD 精度の乗算では 15 回の倍精度加減算と 9 回の倍精度乗算を必要とする．アルゴリズムを以下に記す．アルゴリズム中の a.hi が上位パートa_0，a.lo が下位パートa_1に対応しており，四則演算記号は倍精度演算を意味する．

```
C = dd_mul_dd(A, B)
[p, e] = twoProd(a.hi, b.hi);
e = e + (a.hi × b.lo);
e = e + (a.lo × b.hi);
c.hi = p + e;
c.lo = e - (c.hi - p)
C = [ c.hi, c.lo ]
```

```
[a0, a1] = split(a)
v = 134217729 × a - a;
a0 = 134217729 × a - v;
a1 = a – a0;
```

```
[p, e] = twoProd(a,b)
p = a × b;
[a0, a1] = split(a);
[b0, b1] = split(b);
e = ((a0 × b0 - p) + a0 × b1
    + a1 × b0) + a1 × b1;
```

図 1 DD 精度の乗算

2. 2 対話的多倍長演算環境 MuPAT

MuPAT は DD 精度と QD 精度を利用した擬似 4 倍精度と擬似 8 倍精度からなる多倍長演算環境であり，Scilab のツールボックスとして実装されている[5]．

Scilab/Matlab は線形代数演算が簡単に記述でき，データ型や関数を独自に定義できる．演算子記号や関数名を複数

† 東京理科大学 Tokyo University of Science
‡ 筑波大学 University of Tsukuba

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

定義できるオーバーロード機能により，MuPAT では，倍精度，DD/QD 精度で共通の演算子や関数を使うことができ，変数の定義以外はプログラムの変更がほとんど必要ない．さらに，すべての精度の演算を同時に扱えるため，部分的な高精度化，混合精度演算も実行可能である．

Matlab では，MEX ファイルと呼ばれるユーザー独自の C, C++または Fortran プログラムを Matlab 上でビルドすることによって，それらの関数を組み込み関数のように呼び出すことができる．本研究では，Matlab 上に実装した MuPAT について，ベクトル，行列などの繰り返しを高速化するため，外部 C 関数で記述した MEX ファイルを用いた．

3. 高速化手法とボトルネック解析

3.1 高速化手法

DD/QD 演算は倍精度演算の組み合わせによって実行されており，表 1 のように約 10〜600 倍の演算回数がかかる．そこで，以下のような高速化手法を用いて，外部 C 関数のコードを高速化する．

3.1.1 FMA

FMA(Fused Multiply Add)演算[6, 7]は，$x = a \times b + c$の形式で表される積和演算を 1 演算で実行する．

これにより，メモリアクセス数は変わらないが，演算回数を半減でき，最大で 2 倍の性能向上が見込める．ただし，FMA を用いて高速化できるのは積と和の組で表される場合のみである．

倍精度の内積，行列ベクトル積，行列積の 1 反復で必要な加算 1 回と乗算 1 回は積和演算であり，FMA 演算 1 回で計算できる．

DD 精度の内積，行列ベクトル積，行列積に必要な倍精度演算の回数は表 1 より，DD 加算で加算が 11 回，DD 乗算で加算が 15 回，乗算が 9 回で合計 35 回の倍精度演算である．このうち FMA で実行可能な積和演算は図 1 より split 関数で 1 箇所，twoProd 関数で 4 箇所のため，dd_mul_dd 関数で 2+2+4 箇所と合計 8 箇所となる．FMA を適用すると，必要な演算回数は加算 18(=26-8)回，乗算 1(=9-8)回，FMA 8 回の合計 27 回となる．35 回の浮動小数点数演算を 27 演算で実行するため，1.3 倍の高速化が期待できる．

QD 精度の内積，行列ベクトル積，行列積に必要な倍精度演算の回数は表 1 より，QD 加算で加算が 91 回，QD 乗算で加算が 171 回，乗算が 46 回で合計 308 回の倍精度演算である．このうち FMA で実行可能な積和演算は 40 箇所（twoProd 関数が 6 回で 6×6=36 箇所，qd_mul_qd 関数中に 4 箇所）あるので，FMA を適用すると，必要な演算回数は加算 222(=262-40)回，乗算 6(=46-40)回，FMA 40 回の合計 268 回となる．308 回の浮動小数点数演算を 268 演算で実行するため，1.15 倍の高速化が期待できる．

3.1.2 AVX2

Intel AVX(Advanced Vector Extensions)[6, 7]は，1 命令で 4 つの倍精度浮動小数点数演算を実行できる．メモリアクセス数は変わらず，4 倍の性能向上が見込める．FMA 命令を併用することで×2×4=8 倍の性能向上が可能である．

AVX2 では常に 4 つの倍精度数を扱う必要がある．入力データの次数 n が 4 の倍数でないとき，ベクトルの次数を n+3 とし，n+1，n+2，n+3 には 0 をセットして用いた．この操作によって安全に AVX2 を利用することができる．

また，内積演算は，並列に計算した 4 つの要素を 1 つのスカラーに足しこむ必要があり，Double 精度，DD 精度，QD 精度の加算がそれぞれ 3 回必要となる．そのため，AVX2 を用いた内積演算では Double で 3 回，DD で 33 (=3 × 11)回，QD で 273 (=3 × 91)回，加算の演算回数が増える．

3.1.3 OpenMP

OpenMP(Open Multi-Processing)[8]は共有メモリ型マシンで並列プログラミングを可能にする API で，コア数の分だけ高速化が可能であり，C/C++ と Fortran に適用可能である．

また，OpenMP の変数に，スケジューリング方式とスレッド数がある．今回は 1 コアに 1 スレッドを割り当て，スケジューリングタイプは guided とした．

3.2 ルーフラインモデルと演算強度

メモリアクセスを考慮したプロセッサの性能モデルとして，ルーフラインモデルが Williams らにより提案されている[9]. このモデルでは，演算性能とメモリバンド幅のどちらかがアプリケーションの性能を律速すると仮定する．

アプリケーションに含まれる浮動小数点演算量と，メインメモリから転送されるデータ量の比を演算強度(Flops/Byte)と定義し，演算強度が低いアプリケーションではメモリバンド幅が実効性能を律速し，演算強度が高いアプリケーションでは演算器性能が実効性能を律速すると考える．演算強度とピーク理論演算性能，メモリバンド幅を用いて，アプリケーションの性能を評価する．ルーフラインモデルにおける理論演算性能は次の式で表される．

達成可能な理論演算性能= min{理論演算性能，メモリバンド幅 × 演算強度}

3.2.1 理論演算性能

使用したプロセッサは Intel Core i7 7820 HQ 2.9 GHz 4 core で，メモリは 16GB LPDDR3-2133 dual channel，メモリバンド幅は 2133(line/sec) × 8(byte/line) × 2(interface) = 34.1 [GB/s]である．

高速化手法を用いない場合には，クロック周波数 2.9[GHz]であることと，使用したプロセッサには FPU (Floating Point-Unit)が 2 つ搭載されていることから，理論演算性能は 2.9×2=5.8 [GFlops/sec]となる．FMA を用いると 5.8×2=11.6 [GFlops/sec]， AVX2 を用いると 5.8 × 4 = 23.2 [GFlops/sec]，OpenMP を用いると 5.8×4=23.2 [GFlops/sec]，FMA，AVX2，OpenMP は同時に適用可能なのでピークは，5.8 × 2 × 4 × 4 = 185.6 [GFlops/sec]となる．

コンパイラは LLVM/Clang 5.0.0，Matlab R2017a で実験を行った．なお，コンパイラオプションは OpenMP を有効にする -fopenmp，AVX を有効にする -mavx，FMA を有効にする –mfma，最適化オプションとして -O2 を用いた．

3.2.2 演算強度

DD 精度では倍精度の 2 倍のメモリ，QD 精度では倍精度の 4 倍のメモリを使用する．ベクトル和，行列和，内積，行列ベクトル積，行列積の演算強度を表 2 に示す．n はベクトル，行列の次数を表し，表の xpy，MpM，dot，MV，MM は順にベクトル和，行列和，内積，行列ベクトル積，行列積を表す．

ベクトルの次数 n に対し，DD ベクトル和では，DD 精度の加算が n 回必要になる．表 1 から DD 精度の加算が倍精度の加算 11 回に相当するので，DD ベクトル和の演算回数は倍精度演算で加算 11n 回となる．

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

メモリアクセスは入出力に 3 本のベクトルが必要になり，データ転送数は 6 n で，データ転送量は 48n(Byte)となる．以上より演算強度は 11n / 48n = 0.23 となる．

表 2 各演算の演算強度(n=4096)

演算	精度	総演算数	データ転送量(Byte)	演算強度
xpy	DD	11n	2×3n×8	0.23
	QD	84n	4×3n×8	0.875
MpM	DD	$11n^2$	$2\times3n^2\times8$	0.23
	QD	$84n^2$	$4\times3n^2\times8$	0.875
dot	DD	35n	2×2n×8	1.1
	QD	308n	4×2n× 8	4.8
MV	DD	$35n^2$	$2\times(n^2+2n)\times8$	2.19
	QD	$308n^2$	$4\times(n^2+2n)\times8$	9.63
MM	DD	$35n^3$	$2\times3n^2\times8$	2986.7
	QD	$308n^3$	$4\times3n^2\times8$	13141.3

3.2.3 実験に用いた PC のルーフラインモデル

表 2 に示した各演算の演算強度を横軸とし，3.2.1 節で述べた高速化手法ごとの達成可能な理論演算性能をルーフラインとして図 2 に示した．

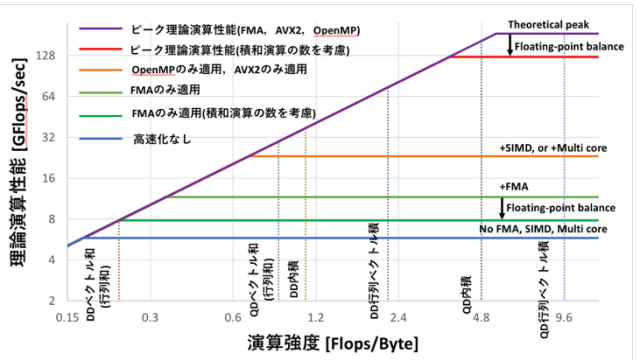

図 2 演算，高速化手法ごとのルーフライン

行列積については演算強度が行列の次数によって変化するが，表 2 は n=4096 のときの演算強度を示している．図 2 では横軸の値を 12.8 までと設定したため，DD/QD 行列積は図のはるか右側となる．

高速化手法を用いない場合は，演算強度が 0.17 以上の演算では演算器性能で律速され，演算器性能がボトルネックである．よってこの場合は実験の対象であるすべての DD/QD 演算の理論演算性能は 5.8[GFlops/sec]となる．

FMA を適用すると演算ごとに理論演算性能が異なる．FMA の適用箇所がない DD/QD ベクトル和(行列和)は 5.8[GFlops/sec]であり，FMA が適用できる DD/QD 内積，DD/QD 行列ベクトル積，DD/QD 行列積は，DD 演算で 7.54 (5.8 × 1.3)[GFlops/sec]，QD 演算で 6.67(5.8×1.15) [GFlops/sec]となり，すべて演算器性能に律速される．

AVX2 または OpenMP を利用した場合では，演算強度が 0.68 以下の DD ベクトル和(行列和)のみメモリバンド幅で律速され，理論演算性能は 7.84[GFlops/sec]となる．他の演算では演算器性能で律速され，理論演算性能は 23.2[GFlops/sec]となる．

FMA と AVX2，OpneMP を組み合わせた場合では，演算強度 2.72 以下の DD/QD ベクトル和(行列和)，DD 内積，DD 行列ベクトル積はメモリバンド幅で律速される．理論演算性能は DD ベクトル和(行列和)で 7.84[GFlops/sec]，QD ベクトル和(行列和)で 29.8[GFlops/sec]，DD 内積で 37.5[GFlops/sec]，DD 行列ベクトル積で 74.7 [GFlops/sec]となる．QD 内積は FMA で性能が 2 倍向上するのならメモリバンド幅で律速されるが，実際には 1.15 倍の性能向上にとどまるため，演算器性能で律速される．その他の QD 行列ベクトル積，DD/QD 行列積も演算器性能で律速され，理論演算性能は DD 行列積で 120.6[GFlops/sec] (92.8 × 1.3)，QD 内積，QD 行列ベクトル積，QD 行列積で 106.7 [GFlops/sec] (92.8×1.15)となる．

4. 性能評価実験

比較する高速化手法は，高速化手法を用いない実装と，FMA のみを利用した実装，AVX2 のみを利用した実装，OpenMP のみを利用した実装と，FMA，AVX2，OpenMP すべてを組み合わせた 実装の 5 通りである．

各高速化手法に対し，5 つの演算（ベクトル和，行列和，内積，行列ベクトル積，行列積）の DD/QD 精度での実行時間を計測し，実測データの単位時間当たりの演算回数を表す“実効性能“と，その向上率で評価する．

実験に用いたベクトルと行列はすべて乱数で発生させ，次数 n は(2^{12}=)4096 とした．

4.1 高速化手法を用いない場合（MEX）

高速化手法を用いない場合の結果を表 3 に示す．DD ベクトル和は表 3 より演算回数が 11n = 45056 [flops]，表 3 から計算時間は 4.2×10^{-5}[sec] より，実効性能は 1.07 (=45056/4.2×10^{-5}/10^9) [GFlops/sec]となる．

表 3 高速化非利用時の計算時間(sec)，実効性能 (GFlops/sec)

MEX	DD		QD	
	計算時間	実効性能	計算時間	実効性能
xpy	4.2×10^{-5}	1.07	1.1×10^{-4}	3.39
MpM	1.6×10^{-1}	1.15	6.0×10^{-1}	2.46
dot	6.4×10^{-5}	2.27	3.1×10^{-4}	4.07
MV	1.3×10^{-1}	4.52	1.1×10^{0}	4.7
MM	4.8×10^{2}	5.01	4.5×10^{3}	4.7

達成可能な理論演算性能と実効性能を比較すると，DD/QD 行列ベクトル積，DD/QD 行列積では上限の 5.8 [GFlops/sec]に近い値で，DD 行列ベクトル積でも 0.78 (4.52 / 5.8)より 8 割近い性能が出ている．

一方，DD ベクトル和は 0.18 (1.07 / 5.8)，DD 行列和は 0.20 (1.15 / 5.8) より，2 割程度の性能しか出ていない．QD ベクトル和は，0.58 (3.39 / 5.8)，QD 行列和は 0.42 (2.46 / 5.8) である．DD 内積は 0.39 (2.27 / 5.8)，QD 内積は 0.70 (4.07 / 5.8) より 7 割程度の性能である．

性能が 4 割以下であった DD ベクトル和(0.18)と DD 内積(0.39)に対して，問題サイズ依存性を調べるため，次数 n を

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

2^{13}=8192 から 2^{25}=33554432 まで 2 倍刻みで変えて追加の実験を行なった．DD ベクトル和の実効性能は，次数 2^{12}=4096 で最小 1.07[GFlops/sec]，2^{18}=262144 で最大 3.4 [GFlops/ sec]，理論演算性能の 0.58 (3.4 / 5.8)であった．DD 内積の場合，次数 2^{25}=33554432 で最大 3.66[GFlops/sec]，理論演算性能の 0.63 (3.66 / 5.8)であった．データが小さいときは理論演算性能の 4 割以下であったが，次数を変えると最大で 6 割程度の性能が出るようになる．

4.2 FMA を利用した場合

FMA を用いた場合の結果を表 4 に示す．DD 内積は，表 2 より演算回数が 35n＝143360 [flops]，表 4 から計算時間は 5.7×10^{-5} [sec]より，実効性能は 2.51 (=143360/5.7×10^{-5}/10^{9}) [GFlops/sec]となる． ベクトル和，行列和は FMA 適用箇所がないのでデータを割愛した．

表 4　FMA 利用時の計算時間(sec)，実効性能 (GFlops/sec)，性能向上率(MEX 比)(ratio)

FMA	DD			QD		
	計算時間	実効性能	MEX 比	計算時間	実効性能	MEX 比
dot	5.7×10^{-5}	2.51	1.1	2.9×10^{-4}	4.42	1.1
MV	1.2×10^{-1}	4.98	1.1	9.7×10^{-1}	5.34	1.1
MM	4.3×10^{2}	5.2	1.04	4.0×10^{3}	5.3	1.1

DD 内積の実効性能は，FMA を利用した場合に 2.51[GFlops/sec]，高速化しない場合（MEX）は表 3 より 2.27[GFlops/sec]である．よって，MEX と比較した性能向上率は 1.1 倍(2.51 / 2.27)である．QD 内積の性能向上率は 1.1 倍(4.42 / 4.07)である．行列ベクトル積の性能向上率は，DD の場合は 1.1 倍(4.98 / 4.52)，QD の場合は 1.1 倍(5.34 / 4.7)である．行列積の性能向上率は，DD の場合は 1.04 倍(5.2 / 5.01)，QD の場合は 1.1 倍(5.3 / 4.7)である．

3.1.1 節で DD では 1.3 倍，QD では 1.15 倍の高速化ができると予想した．結果は DD/QD 内積で 1.1 倍，DD/QD 行列ベクトル積で 1.1 倍，DD 行列積で 1.04 倍，QD 行列積で 1.1 倍と，いずれも想定に近い性能向上率であった．

4.3 AVX2 を利用した場合

AVX2 は同時に処理できるデータの数が 4 倍となるため，理論上 4 倍の性能向上が見込まれ，理論演算性能は 23.2 [GFlops/sec] となる．

AVX2 を用いた場合の結果を表 5 に示す．DD ベクトル和は表 2 より演算回数が 11n＝45056[flops], 表 5 から計算時間は 4.4×10^{-5}[sec]より，実効性能は 1.02 [GFlops/sec]となる．

表 5 で性能向上率を表す MEX 比に注目すると，メモリバンド幅で律速される DD ベクトル和で性能向上率が 0.96 倍，DD 行列和については 1.1 倍となり，3.2.3 節で示した理論演算性能の比 1.35(7.84/5.8)と大きな差はなかった．

表 5　AVX2 利用時の計算時間(sec)，実効性能 (GFlops/sec)，性能向上率(MEX 比)(ratio)

AVX2	DD			QD		
	計算時間	実効性能	MEX 比	計算時間	実効性能	MEX 比
xpy	4.4×10^{-5}	1.02	0.96	8.0×10^{-5}	4.65	1.4
MpM	1.5×10^{-1}	1.23	1.1	1.9×10^{-1}	4.77	1.9
dot	4.3×10^{-5}	3.31	1.5	9.7×10^{-5}	13.0	3.2
MV	3.4×10^{-2}	17.27	3.8	2.6×10^{-1}	19.87	4.2
MM	1.2×10^{2}	20.04	4.0	1.1×10^{3}	19.24	4.1

QD ベクトル和の性能向上率は 1.4 倍，DD 内積の性能向上率は 1.5 倍，QD 行列和の性能向上率は 1.9 倍で，想定された 4 倍に対して低い値となった．これらの演算の演算強度は QD ベクトル和と QD 行列和で 0.875，DD 内積で 1.1 であり，演算強度が低かったためと考えられる．

一方，他の演算の性能向上率は QD 内積が 3.2 倍（演算強度 4.8），DD 行列ベクトル積が 3.8 倍（演算強度 2.19），QD 行列ベクトル積が 4.2 倍（演算強度 9.6）である．DD/QD 行列積の性能向上率は DD が 4 倍，QD が 4.1 倍で，AVX2 使用時の性能向上率 4 倍に近い値となり，演算強度が 2.19 以上の演算では AVX2 の機能が有効に働いた．

性能向上率 2 倍以下であった DD ベクトル和(0.96)，QD ベクトル和(1.4)，DD 内積(1.5)に対して，4.1 節と同様に問題サイズ依存性を調べるため，次数 n を 2^{13} から 2^{25} と 2 倍刻みで変えて実験した．DD/QD ベクトル和の性能は，次数 2^{16} のときが最大で，DD で 4.3 [GFlops/ sec]，性能向上率 1.4 倍，QD で 9.7[Gflops/sec]，性能向上率 2.9 倍だった．DD 内積は，次数 2^{19} のとき最大 14.1 [GFlops/sec]，性能向上率 4.15 倍であった．

DD ベクトル和は次数 2^{15} ~2^{17} で性能向上率 1.3~1.4 倍，次数 2^{17}~2^{25} は効果がなかった．QD ベクトル和は次数 2^{13}~2^{23} で性能向上率が 2.1~2.9 倍となった．特に，次数 2^{15}~2^{17} では 2.7~2.9 倍と高い向上率だった．DD ベクトル和と QD ベクトル和にはピークとなる次数が存在し，そこでは良好な性能向上率が得られた．DD 内積は次数 2^{15} 以上なら性能向上率が 3 倍を超え，最高で 4.15 倍となることから，次数 2^{15}~2^{25} で AVX2 は十分に有効である．

演算強度が低い演算でも，演算器性能で律速される演算はデータ量が小さいと性能向上率は低かったが，大きくすると性能が向上し，理論演算性能に近づくことがある．しかし，ベクトル和演算では単純に問題を大きくすればよいわけでなく，性能にピークがあることがわかる．

4.4 OpenMP を利用した場合

OpenMP は 4 コアを利用できるため，理論上 4 倍の性能向上が見込まれ，理論演算性能は 23.2 [GFlops/sec]となる．

OpenMP を用いた場合の結果を表 6 に示す．DD ベクトル和は表 2 より演算回数が 11n＝45056[flops], 表 6 から計算時間は 4.4×10^{-5}[sec]より，実効性能は 1.02 [GFlops/sec]となる．

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

表 6　OpenMP 利用時の計算時間(sec)，実効性能(GFlops/sec)，性能向上率(MEX 比)(ratio)

OpenMP	DD			QD		
	計算時間	実効性能	MEX 比	計算時間	実効性能	MEX 比
xpy	4.4×10^{-5}	1.02	0.96	9.0×10^{-5}	4.14	1.2
MpM	5.6×10^{-2}	3.3	2.9	3.5×10^{-1}	8.04	3.3
dot	4.3×10^{-5}	3.32	1.5	1.8×10^{-4}	6.97	1.7
MV	4.9×10^{-2}	11.98	2.6	3.5×10^{-1}	14.76	3.1
MM	1.3×10^{2}	18.5	3.7	1.3×10^{3}	16.1	3.4

表 6 で性能向上率を表す MEX 比に注目すると，QD 行列ベクトル積，DD/QD 行列和，DD/QD 行列積では性能向上率が想定の 4 倍に近い．データ量が他の演算よりも大きいこれらの演算に関しては OpenMP の機能による性能向上率が高いと考えられる．

一方，DD/QD ベクトル和の性能向上率は DD の場合が 0.96 倍，QD の場合が 1.2 倍，DD/QD 内積の性能向上率は DD の場合が 1.5 倍，QD の場合が 1.7 倍，DD 行列ベクトル積の場合が 2.6 倍である．DD 行列ベクトル積は次数を 2 倍の 2^{13}=8192 にすると 3.1 倍の性能向上となった．

性能向上率 2 倍以下であった DD ベクトル和(0.96)，QD ベクトル和(1.2)，DD 内積(1.2)，QD 内積(1.7)に対して，問題サイズ依存性を調べるために，次数 n を 2^{13} から 2^{25} と 2 倍刻みで変えて実験した．DD/QD ベクトル和の性能は，DD で次数 2^{16} のとき最大 4.51 [GFlops/ sec]，性能向上率 1.4 倍，QD で次数 2^{17} のとき最大 8.52[GFlops/sec]，性能向上率 2.8 倍だった．DD/QD 内積の性能は，次数 2^{25} のとき，DD で最大 14.4 [GFlops/sec]，性能向上率 3.94 倍，QD で最大 16.6 [GFlops/sec]，性能向上率 3.7 倍だった．

DD ベクトル和の性能向上率は次数 2^{15} ~ 2^{17} で 1.2~1.4 倍，次数 2^{18} ~ 2^{23} では 0.9~1.1 倍，次数 2^{24} ~2^{25} では 2.4~2.5 倍となった．QD ベクトル和は次数 2^{14} 以上で性能向上率が 2.2~3 倍，次数 2^{16} ~2^{17}，2^{24} ~2^{25} で 2.5~3 倍となる．DD/QD 内積は，DD では次数 2^{16} 以上，QD では次数 2^{14} 以上で性能向上率が 3 倍を超える．

DD/QD ベクトル和，DD/QD 内積でデータ量を大きくすると性能が向上し，理論演算性能に近づくことがある．ベクトル和は単純に問題を大きくすればよいわけでなく，性能向上率は問題サイズと関係がある．

4.5 FMA, AVX2, OpenMP を組み合わせた場合

AVX2, OpenMP を組み合わせると×4×4 の性能向上が見込まれ，さらに FMA を適用すると，その理論演算性能は DD で 120.6[GFlops/sec]，QD で 106.7[GFlops/sec]となる．

すべて組み合わせた場合の結果を表 7 に示す．DD ベクトル和は，表 2 より，11n = 45056[flops]，表 7 から計算時間は 5.4×10^{-5}[sec]より，実効性能は 0.834 [GFlops/sec]となる．

これらの演算器性能で律速される演算は想定の 8 割程度が得られ，組み合わせは有効だった．

表 7　FMA, AVX2, OpenMP 利用時の計算時間(sec)，実効性能(GFlops/sec), 実行効率(%), 性能向上率(MEX 比)(ratio)

FMA +AVX2 + OMP	DD			QD		
	計算時間	実効性能 (実行効率)	MEX 比	計算時間	実効性能 (実行効率)	MEX 比
xpy	5.4×10^{-5}	0.83 (0.5)	0.8	7.6×10^{-5}	4.9 (2.6)	1.4
MpM	5.8×10^{-2}	3.18 (1.7)	2.8	1.2×10^{-1}	12.72 (6.9)	5.2
dot	4.4×10^{-5}	2.98 (1.6)	1.4	6.0×10^{-5}	20.87 (11.3)	5.1
MV	1.3×10^{-2}	45.17 (24.3)	9.9	7.3×10^{-2}	70.79 (38.1)	14.9
MM	2.9×10^{1}	84.15 (45.3)	16.7	2.8×10^{2}	75.59 (40.7)	16.1

表 7 で性能向上率を表す MEX 比に注目すると，性能向上率が高いのは，QD 行列ベクトル積 14.9 倍，DD 行列積 16.7 倍，QD 行列積 16.1 倍で，これらの演算では FMA, AVX2, OpenMP のすべてが有効に働いた．性能向上の想定は，DD 演算で 20.8 倍，QD 演算で 18.4 倍であったため，

DD 行列ベクトル積の性能向上率は 9.9 倍だった．すべてを組み合わせるとメモリバンド幅で律速されるようになるため，想定される性能向上率は 12.9 倍の 8 割程度(9.9/12.9)の性能向上が達成され，すべての組み合わせは有効である．

DD ベクトル和は，性能向上率が AVX2 で 0.96 倍，OpenMP で 0.96 倍より，併用すると性能向上率は 0.92 倍(0.96×0.96)と考えられる．結果は，表 7 から 0.8 倍であり．性能は上がらない．QD ベクトル和は，性能向上率が AVX2 で 1.4 倍，OpenMP で 1.2 倍より，併用すると性能向上率は 1.68 倍 (1.4×1.2)と考えられる．結果は 1.4 倍であり，小さな次数でも併用は有効と考えられる． DD 内積は，性能向上率が FMA で 1.1 倍，AVX2 で 1.5 倍，OpenMP で 1.5 倍より， 組み合わせると性能向上率は 2.5 倍 (1.1×1.5×1.5)と考えられるが，結果は 1.4 倍であり，小さな次数では思ったより性能が向上しなかった．QD 内積は，性能向上率が FMA で 1.1 倍，AVX2 で 3.2 倍，OpenMP で 1.7 倍より，組み合わせると性能向上率は 5.9 倍 (1.1×3.2×1.7)と考えられる．結果は 5.1 倍性能であり，小さな次数でも組み合わせは有効である．

演算強度が低く，データ量が小さい DD/QD ベクトル和と DD 内積，演算強度が高いがデータ量が小さい QD 内積で同様に次数 n を 2^{13} から 2^{25} と 2 倍刻みで変えて実験した．DD/QD ベクトル和の性能は，次数 2^{16} のとき，DD で最大 5.34[GFlops/sec]，性能向上率 1.67 倍，QD で最大 12.9 [GFlops/sec]，性能向上率 3.89 倍だった．DD/QD 内積の性能は，次数 2^{25} のとき，DD は最大 26.45 [GFlops/sec]，性能向上率 7.24 倍，QD は最大 71.72[GFlops/sec]，性能向上率 16.0 倍であった．

DD ベクトル和の性能向上率は，次数 2^{16} で AVX2 が 1.4 倍，OpenMP が 1.4 倍であり，併用すると 1.67 倍となるが，次数 2^{17}~2^{23} の場合 AVX2，OpenMP 共に 0.9~1.0 倍の性能向上となる．次数 2^{24}~2^{25} の場合 AVX2 で 1.0 倍，OpenMP で 2.5 倍であり，併用すると 2^{24}~2^{25} で併用の効果が見込める．QD ベクトル和の性能向上率は，次数を上げると

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

2.5~4.4 倍となることから併用はプラスに働き，次数が小さいときよりも併用の効果は高まる．DD 内積の性能向上率で 2^{16} 以上で 6~8 倍，QD で 2^{14} 以上で 9~16 倍となる．

4.6 数値実験のまとめ

Matlab 版 MuPAT に FMA, AVX2, OpenMP を用いて高速化処理を行った．

FMA に関しては，積と和が組で出現しないと効果がない．効果がある場合でも DD 行列積で 1.04 倍，DD/QD 内積，DD/QD 行列ベクトル積，QD 行列積で 1.1 倍と低い．実験では想定の 8 割~9 割の効果があった．データ量によらない．

AVX2 に関しては，QD 内積，DD/QD 行列ベクトル積，DD/QD 行列積は演算器性能で律速されており，次数を変えずとも 3.2~4 倍性能が向上した．QD ベクトル和と DD 内積は演算強度が 0.875 と 1.1 のため演算器性能に律速され，次数が小さいときは 1.5 倍程度の性能向上にとどまる．次数を変えると QD ベクトル和では 3 倍，DD 内積では 4 倍性能の向上となる．DD ベクトル和は演算強度 0.23 でメモリバンド幅に律速され，次数を変えても実効性能が最高で 1.4 倍までしか向上しない．

OpenMP に関しては，DD/QD 行列和，QD 行列ベクトル積，DD/QD 行列積では 2.8~3.7 倍の性能が向上した．QD ベクトル和，DD/QD 内積，DD 行列ベクトル積は演算器性能に律速され,QD ベクトル和で 1.2 倍，DD/QD 内積で 1.5~1.7 倍，DD 行列ベクトル積では 2.6 倍の性能向上にとどまったが，次数を変えると 3~4 倍性能が向上する．DD ベクトル和はメモリバンド幅に律速され，次数 2^{12}~ 2^{23} までは 0.9~1.4 倍の性能向上であったが，次数 2^{24}~2^{25} は 2.5 倍の性能向上となった．

全部組み合わせた場合は，QD ベクトル和，DD 内積，DD 行列ベクトル積は全部組み合わせると，メモリバンド幅で律速されるようになるため，個別の実験の性能向上率を掛け合わせた値よりも性能向上率は低い．たとえば，DD 内積は次数を上げると FMA で 1.1 倍，AVX2 と OpenMP はどちらも 4 倍近く性能が向上していたが，組み合わせた場合は最高でも 8 倍程度の性能向上だった．QD 行列ベクトル積，DD/QD 行列積は演算器性能に律速され，次数を変えずともピークの 4 割~5 割程度の性能で，性能向上率は 15~17 倍である．QD 内積は演算器性能で律速されるが，次数が小さいときには 5.1 倍の性能向上にとどまる．次数を変えればピークの 4 割程度の性能であり，性能向上率は 16 倍である．DD/QD ベクトル和，DD 内積，DD 行列ベクトル積は演算強度 2.72 以下でメモリバンド幅に律速され，次数を変えると達成可能な理論演算性能の 6 割~7 割の性能となる．QD ベクトル和では 4 割程度の性能が出ていた．性能向上率は DD ベクトル和で 1.67 倍，QD ベクトル和で 3.9 倍，DD 内積で 7.2 倍，DD 行列ベクトル積で 9.9 倍であり，これらの性能向上の倍率は演算強度に依存する．

5. おわりに

スカラー演算は外部関数を使ってもオーバーヘッドのため高速化することができない．頻出・多用するベクトル，行列演算は，外部関数にオフロードすることによって高速化することができる．今回は外部関数に FMA, SIMD, マルチプロセッシングを使って高速化を試みた．

演算器性能で律速される（演算強度が 3.1 以上）演算(QD 内積，QD 行列ベクトル積，DD/QD 行列積)の実効性能をピークの 4 割~5 割まで高めることができた．メモリバンド幅で律速される演算(DD/QD ベクトル和，DD 内積，DD 行列ベクトル積)ではピーク性能を出せないため，ピーク時と比べると理論演算性能は低くなり，高速化してもその性能向上率は演算強度に依存する．それでも，高速化によって DD ベクトル和で 1.6 倍性能が向上し，達成可能な理論演算性能の 7 割，QD ベクトル和で 3.9 倍性能が向上し，達成可能な理論演算性能の 4 割，DD 内積で 7.2 倍性能が向上し，達成可能な理論演算性能の 7 割，DD 行列ベクトル積で 9.9 倍性能が向上し，達成可能な理論演算性能の 6 割まで性能を出せるようになり，高速化によっていずれの場合も実効性能は向上した．しかし，小さな次数の DD/QD ベクトル和，DD/QD 内積は問題サイズに依存して性能向上率が期待より小さくなってしまうことがある．さらに，DD ベクトル和ではデータ量が小さく，演算強度も低いため，小さな次数だと高速化されない場合もある．

メモリバンド幅で律速されている DD/QD ベクトル和，DD 内積，DD 行列ベクトル積に対して，キャッシュの挙動を含めたパフォーマンス向上が今後の課題である．

謝辞

本研究の実施には，JSPS 科研費 JP17K00164 の助成を受けた．

参考文献

[1] 小武守恒，藤井昭宏，長谷川秀彦，西田晃，SSE2 を用いた反復解法ライブラリ Lis 4 倍精度版の高速化, 2006-HPC-108, pp7-12 (2006)
[2] H. Waki , M. Nakata and M. Muramatsu, Strange behaviors of interior-point methods for solving semidefinite programming problems in polynomial optimization, Computational Optimization and Applications, Vol.53(3), pp.823--844 (2012).
[3] D. H. Bailey, High-Precision Floating-point arithmetic in scientific computation, Computing in Science and Engineering, Vol.7(3), pp.54-61
[4] Y. Hida, X. S. Li and D. H. Baily, Quad-double arithmetic: algorithms, implementation and application, Technical Report LBNL-46996, Lawrence Berkeley National Laboratory, Berkeley (2000).
[5] S. Kikkawa, T. Saito, E. Ishiwata and H. Hasegawa, Development and acceleration of multiple precision arithmetic toolbox MuPAT for Scilab, JSIAM Letters, Vol.5, pp.9-12 (2013).
[6] Intel: Intrinsics Guide, available from <https://software.intel.com/sites/landingpage/IntrinsicsGuide/>
[7] Intel: 64 and IA-32 Architectures Optimization Reference Manual, <https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual>
[8] OpenMP : < http://www.openmp.org/>
[9] S. Williams, A. Waterman and D. Patterson, Roofline: An insightful visual performance model for multicore architectures, Communications of the ACM, Vol.52(4), pp.65-76 (2009).
[10] T. Dekker, A floating-point technique for extending the available precision, Numerische Mathematik, Vol.18, pp.224-242 (1971).
[11] D.E Knuth, The Art of Computer Programming, Seminumerical Algorithmes, Vol.2, Addison-Wesley(1969).
[12] E.Peise, Performance Modeling and Prediction for Dense Linear Algebra, arXiv:1706.01341(2017).
[13] 佐藤義永，永岡龍一，撫佐昭裕，江川隆輔，滝沢寛之，岡部公起，小林広明，ルーフラインモデルに基づくベクトルプロセッサ向けプログラム最適化戦略，情報処理学会論文誌 コンピューティングシステム Vol. 4 (3), pp.77-87 (2011).
[14] R. Dolbeau, Theoretical Peak FLOPS per instruction set on modern Intel CPUs, <http://www.dolbeau.name/dolbeau/publications/peak>

Copyright © 2018 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.