GPUによる4倍・8倍精度BLASの実装と評価

椋木大地† 高橋大介†

本研究では4倍・8倍精度演算に対応したBLAS（Basic Linear Algebra Subprograms）関数をGPU（Graphics Processing Unit）向けに実装し評価を行った．4倍・8倍精度演算にはdouble型倍精度数を2つ連結して4倍精度数を表すdouble-double（DD）型4倍精度演算，および4つ連結して8倍精度数を表現するquad-double（QD）型8倍精度演算を用いた．NVIDIA Tesla C2050による性能評価では，Intel Core i7 920での同一処理と比べ，4倍精度AXPYが約9.5倍，8倍精度AXPYが約19倍高速化された．また4倍精度GEMMはCPUに比べて約29倍，8倍精度GEMMは約24倍の高速化を達成した．さらにTesla C2050では4倍精度AXPYが倍精度演算の高々2.1倍の演算時間となり，GEMV，GEMMでも倍精度演算に対する計算時間の増大がCPUの場合と比べ大幅に削減された．一方でPCI-Express（PCIe）によるデータ転送時間を考慮した場合，倍精度GEMMはPCIeデータ転送性能に律速される傾向が見られたが，4倍・8倍精度GEMMではこれがほぼ解消されることが示された．本論文では4倍・8倍精度BLAS演算がGPUに適しており，CPUに比べ実用的な性能が得られることを示す．

Implementation and Evaluation of Quadruple and Octuple Precision BLAS on GPUs

Daichi Mukunoki† and Daisuke Takahashi†

We implemented quadruple and octuple precision Basic Linear Algebra Subprograms (BLAS) functions on graphics processing units (GPUs), and evaluated their performances. We used DD-type quadruple precision operation, which combines two double precision values to represent a quadruple precision value, and QD-type octuple precision operation, which combines four double precision value, to represent a octuple precision value. On NVIDIA Tesla C2050, quadruple precision AXPY is approximately 9.5 times faster, and octuple precision AXPY is approximately 19 times faster than that on Intel Core i7 920. Additionally, quadruple precision GEMM is approximately 29 times faster, and octuple precision GEMM is approximately 24 times faster than that on the CPU. Moreover, the execution time of quadruple precision AXPY takes only approximately 2.1 times longer than that of double precision AXPY on the GPU. Also on quadruple and octuple precision GEMV and GEMM on the GPU, the increase of the execution time relative to double precision operation is decreased compared to the CPU. On the other hand, taking the PCI-Express (PCIe) data transfer time into consideration, the performance of double precision GEMM is limited by PCIe data transfer time, but that of quadruple and octuple precision GEMM is almost not limited by them. In this research, we show that quadruple and octuple precision BLAS operations are suitable for GPUs.

1. はじめに

浮動小数点演算にはその原理上丸め誤差が存在し，主に科学技術計算などでは倍精度でも精度が不足する計算が存在する．例えばCG法などの反復法では，丸め誤差の影響で倍精度であっても収束が停滞するケースが存在する．また特に近年では計算の大規模化により，丸め誤差の蓄積が問題となることが予想される．このような背景から，倍精度を超える高精度演算が必要とされており，古くから様々な試みがなされてきた．

一方，現在の一般的なプロセッサがハードウェアレベルでサポートするのは64bit倍精度浮動小数点演算までであり，それ以上の高精度演算にはソフトウェアエミュレーションが必要となる．ソフトウェアによる高精度演算は，四則演算などの単純な二項演算であっても多数の演算を組み合わせて計算するため，その演算量は膨大なものとなる．そのため従来のプロセッサの性能では実用的な性能が得られず，高精度計算は敬遠される傾向にあった．しかし近年，プロセッサの演算性能が著しく向上し，高精度計算の活用が注目され

† 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba

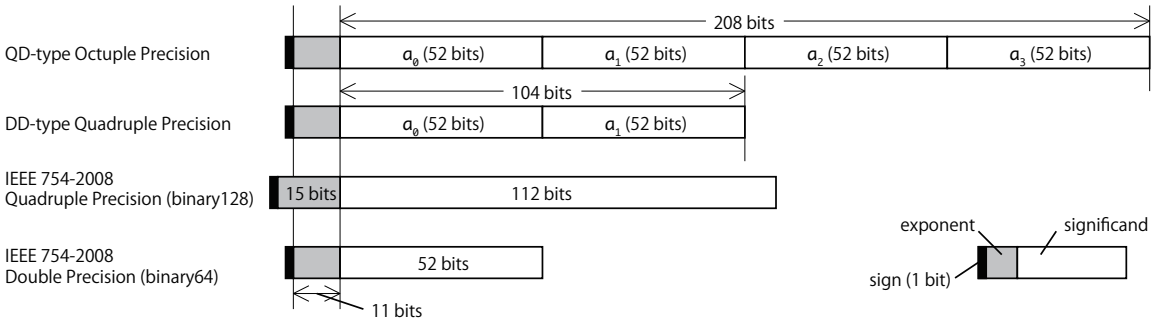

図 1 4 倍・8 倍精度データフォーマット

るようになった．

近年，高性能なプロセッサとして特に注目されているのが GPU（Graphics Processing Unit）である．GPU は CPU を上回る演算性能とコストパフォーマンスが得られることから，GPU を汎用計算に応用する GPGPU（General Purpose computing on GPU）が広く普及するようになった．GPU は数百個のコアを搭載するメニーコアプロセッサである．また NVIDIA Tesla C2050 では理論ピーク演算性能が単精度で 1030GFlops，倍精度で 515GFlops に達するなど，CPU と比べて高い演算性能を有する．一方で GPU は CPU のアクセラレータとして動作し，CPU–GPU 間の接続には PCI-Express（PCIe）が用いられている．しかし PCIe 2.0 x16 の場合でも理論ピークバンド幅は 8GB/s と，GPU の演算性能に対して明らかに不足している．そのため GPU の優れた演算性能を活用できるのは並列性が高く，1 演算あたりのメモリ転送量である Byte/Flop 値の小さいアプリケーションに限られる．

本研究では GPU による 4 倍・8 倍精度演算に対応した BLAS（Basic Linear Algebra Subprograms）関数を NVIDIA 製 GPU 上に実装し，性能評価を行った．BLAS は行列・ベクトルの演算であるため，メニーコア・マルチスレッドの超並列アーキテクチャを持つ GPU に適した処理である．また 4 倍・8 倍精度演算は演算量が多く Byte/Flop 値が小さい処理である．そのため，4 倍・8 倍精度 BLAS 演算は GPU による高速化が期待できる．

以下，2 章では本論文の関連研究と 4 倍・8 倍精度演算手法ついて述べ，3 章で 4 倍・8 倍精度 BLAS 関数の GPU 実装について述べる．また性能評価の結果を 4 章で述べる．最後に 5 章で本論文のまとめとする．

2. 4 倍・8 倍精度演算

2.1 関 連 研 究

ソフトウェアによる高精度演算手法は，高精度数の格納方法によって 2 種類に大別できる．高精度数を整数配列によって定義された独自の型に格納する方式と，既存の浮動小数点数型を複数連結して格納する方式である．前者の手法を用いた高精度演算ライブラリとして，GMP[1]，MPFR[2]，ARPREC[3] などが知られており，これらは任意精度の実現に適している．一方，後者の方式を採用した高精度演算ライブラリとして，4 倍・8 倍精度演算ライブラリの QD[4] が知られている．QD ライブラリでは倍精度浮動小数点数（double 型）を 2 個連結して 4 倍精度浮動小数点数を表現する double-double（DD）型 4 倍精度演算と，double 型を 4 個連結して 8 倍精度浮動小数点数を表現する quad-double（QD）型 8 倍精度演算を採用している．この方式では既存の浮動小数点数型が持つ指数部，仮数部をそのまま流用するため，整数配列を用いた方式と比べ比較的単純に実装できるとともに，高速に計算できる．しかし指数部を拡張できないため，表現できる数の範囲が限られ，8 倍精度を超える精度の実現には適さない．本研究では 4 倍・8 倍精度演算に，この QD ライブラリの DD 型 4 倍精度演算および QD 型 8 倍精度演算を用いる．詳細は次節で説明する．

高精度演算に対応した BLAS の実装として，DD 型演算を使用した XBLAS[5] が知られている．XBLAS は入出力データは倍精度であるが，内部の演算に DD 型演算を用いることで高精度化を実現している．一方，MBLAS[6] は既存の高精度演算ライブラリを用いた高精度 BLAS 実装である．GMP や MPFR による任意精度演算と QD ライブラリによる 4 倍・8 倍精度演算に対応し，入出力データも各ライブラリが採用する高精度データ形式のまま扱うことができる．

これらの研究はすべて CPU 上に実装されたものであるが，GPU 上で高精度演算を行った事例もいくつか存在する．まず Göddeke ら[7] は倍精度演算に対応しない GPU に対し，FEM ソルバに double-float 型による倍精度演算を適用した事例を示している．また Thall[8] は double-float 型，quad-float 型の高精度演算を GPU 上に実装しているほか，中里ら[9] は DD 型 4 倍精度演算を AMD 社製 GPU，GRAPE-DR 向けに実装している．一方，Zhao ら[10] は GPU に対応した GMP 相当のライブラリ GPUMP，Lu ら[11] は QD，ARPREC の GPU 版である GQD，GARPREC を

実装している．

GPU 向けの BLAS 実装で GPU による BLAS の加速が有効なことは知られており，GPU 上での 4 倍・8 倍精度演算に関する研究例から，4 倍・8 倍精度 BLAS が GPU 上で加速されることが期待できる．しかし GPU において BLAS の関数単位で 4 倍・8 倍精度演算を実装・評価した事例は存在しておらず，従来の倍精度関数と比較してどれほどの計算コストがかかるのかといった，定量的な評価はなされていなかった．

2.2 DD 型 4 倍精度演算と QD 型 8 倍精度演算

ここでは本研究で用いる QD ライブラリにおける DD 型 4 倍精度演算および QD 型 8 倍精度演算について説明する．まずデータフォーマットを**図 1** に示す．DD 型 4 倍精度では 4 倍精度浮動小数点数 a を 2 つの倍精度浮動小数点数 a_0 と a_1 によって $a = a_0 + a_1$, $|a_0| > |a_1|$ と表す．同様に QD 型 8 倍精度では 4 つの倍精度浮動小数点数を用いて $a = a_0 + a_1 + a_2 + a_3$, $|a_0| > |a_1| > |a_2| > |a_3|$ と表す．IEEE 754-2008 の倍精度型（binary64）は仮数部が 52 ビット（ケチ表現により実際には 53 ビット相当）であり，DD 型 4 倍精度の仮数部は 104 ビット（同様にケチ表現で 106 ビット相当），十進で約 32 桁の精度となる．これは IEEE754-2008 の 4 倍精度型（binary128）より仮数部，指数部ともに小さい．また QD 型 8 倍精度は仮数部が 208 ビット（同様にケチ表現で 212 ビット相当）で，十進約 64 桁の精度となる．なお 8 倍精度型は IEEE 754-2008 では定義されていない．

DD 型 4 倍精度演算は 4 倍精度数 $a = a_0 + a_1$, $b = b_0 + b_1$ の a_0，a_1 と b_0，b_1 同士を筆算の原理で計算する．QD 型 8 倍精度演算も同様の原理である．アルゴリズムの詳細は Hida[12] らの論文で説明されている．なお QD ライブラリでは 106 ビットの精度を保証するアルゴリズムと，106 ビットの精度を保証しない代わりに演算量を削減したアルゴリズム（sloppy アルゴリズム）の 2 種類が実装されている．後者のアルゴリズムは，4 倍精度数の下位データ（a_1 と b_1）同士の加算時にキャリーが発生した場合，その分だけ仮数部の下位桁が失われてしまうが，これを考慮しないというものである．本論文では前者の 106 ビットの精度を保証するアルゴリズムを想定する．また DD 型 4 倍精度および QD 型 8 倍精度の乗算では，積和演算 $a \times b + c$ の中間の演算結果を丸め誤差なしの 106 ビットで保持し一命令で実行可能な倍精度 Fused-Multiply Add（FMA）命令を用いることで，演算量を削減することができる．DD 型 4 倍精度および QD 型 8 倍精度の加算，乗算に要する演算量を**表 1** に示す．倍精度演算に比べ演算量が大きくなる一方で，メモリ参照量は DD 型演算が倍精度演算の高々2 倍，QD 型演算が 4 倍であるため，メモリ参照量に対して演算量の多い，GPU に適した処理である．

表 1 DD 型 4 倍精度演算および QD 型 8 倍精度演算の演算量

	DD 型 4 倍精度	QD 型 8 倍精度
加算	20 Flop	90 Flop
乗算（FMA あり）	10 Flop	193 Flop
乗算（FMA なし）	24 Flop	333 Flop

3. 4 倍・8 倍精度 BLAS の実装

本研究では QD ライブラリで採用されているものと同じ DD 型 4 倍精度演算および QD 型 8 倍精度演算を採用した BLAS を GPU 上に実現する．入出力データも DD 型 4 倍精度および QD 型 8 倍精度で取り扱う．実装には NVIDIA 社の GPGPU 開発環境である CUDA（Compute Unified Device Architecture）を使用し，ハードウェアとして倍精度演算が可能な GT200 アーキテクチャ以降の GPU を想定する．

3.1 BLAS 関数の実装

今回，Level 1–3 BLAS の代表的な関数として Level 1 BLAS の AXPY ($y = \alpha x + y$)，Level 2 BLAS の GEMV ($y = \alpha Ax + \beta y$)，Level 3 BLAS の GEMM ($C = \alpha AB + \beta C$) を実装した．行列データの格納は列優先順で行う．なお現段階の実装では計算できる次元数の制約があるほか，転置行列のサポートが省略されており，BLAS の仕様に完全には準拠していない．

CUDA では CPU 側のメモリ空間と GPU 側のメモリ空間は独立しており，GPU で処理を行うデータは明示的に CPU から GPU へ PCIe 経由でデータ転送を行う必要がある．今回の実装では BLAS の演算は演算に必要なデータの転送完了後に行い，演算とデータ転送のオーバーラップは行っていない．

BLAS 演算はベクトル，行列の各要素に対し，CUDA の 1 スレッドを割り当てて計算する．今回 AXPY，GEMV ではスレッドブロックあたりのスレッド数を 128 とし，65535×128 要素までのベクトルに対し，各スレッドが持つ ID 番号をインデクスとして演算を行うようにした．GEMM ではキャッシュに相当する高速なオンチップメモリである共有メモリを利用し，キャッシュブロッキングを行った．共有メモリに入りきる大きさでブロッキングサイズを検討した結果，4 倍精度では 8×8，8 倍精度では 16×16 の場合に最速であったためこれを採用した．スレッドブロックあたりのスレッド数もそれぞれ $8 \times 8 = 64$，$16 \times 16 = 256$ スレッドとした．なおこの共有メモリは Tesla C1060 で 16KB，Tesla C2050 では 64KB のうち共有メモリ 16KB ＋ L1 キャッシュ48KB，または共有メモリ 48KB ＋ L1 キャッシュ16KB の 2 通りの構成で利用できるが，今回はそれぞれの GPU 向けに個別のチューニングを行わないこととしたため，前者の共有メモリが 16KB の場合を想定して実装を行った．

3.2 4倍・8倍精度演算の実装

4倍・8倍精度演算部分はQDライブラリをCUDAに移植する形で実装した．QDをCUDA環境に移植したものとしてLuらによるGQDが存在するが，GQDでは106ビットの精度を保証しないsloppyアルゴリズムしか実装されていないことや，GPUがサポートするFMA命令を利用した実装がなされていないこと，また我々もQDライブラリのDD型演算をCUDAに移植[13]していたことから，これにQD型演算を追加する形で実装した．

4倍・8倍精度演算関数群は基本的にCPU向けのQDライブラリと同一の処理を行っており，BLASの演算であるベクトル・行列の各要素を計算するスレッド内で，それぞれ逐次的に処理される．関数はGPU用のデバイス関数として実装しているが，コンパイル時にインライン展開されるため，関数呼び出しのオーバーヘッドは存在しない．

GT200アーキテクチャ以降のGPUでは，積和演算の中間の演算結果を丸め誤差なしの106ビットで保持する倍精度FMA命令を利用できるため，乗算アルゴリズムにおいてFMA命令を使用している．CUDAではコンパイラがFMA命令に置き換え可能な積和演算を自動でFMA命令に置き換えるため，FMA命令を使用するアルゴリズムをそのまま記述すればFMA化されるが，明示的にするためFMA命令部分は組み込み関数`__fma_rn`を用いて実装した．一方，FMA命令を使用してはならない箇所ではコンパイラによる意図しないFMA命令への置き換えを防止するため，乗算・加算のみを行う組み込み関数`__dmul_rn`，`__dadd_rn`を使用して実装を行った．

4. 性能評価

4.1 評価方法

本研究で実装したDD型4倍精度およびQD型8倍精度AXPY，GEMV，GEMMの性能を，NVIDIA Tesla C2050（Fermiアーキテクチャ）および一世代前のNVIDIA Tesla C1060（GT200アーキテクチャ）の2つのGPU上で測定した．倍精度理論ピーク演算性能はTesla C1060が78GFlops，Tesla C2050が515GFlopsである．Tesla C1060は4GBのGDDR3メモリ（理論ピークメモリ帯域102GB/s），Tesla C2050は3GBのGDDR5メモリ（理論ピークメモリ帯域144GB/s）が搭載されている．なおTesla C2050にはメモリにECC機能が搭載されているが，ECC機能を有効にするとメモリ性能が低下するため，今回は速度を優先してECCを無効として測定した．

GPUにおける我々の4倍・8倍精度BLAS実装をここではCUDDBLAS（DD型4倍精度），CUQDBLAS（QD型8倍精度）と呼ぶ．性能比較のためGotoBLAS 2-1.13（倍精度，CPU），CUBLAS 3.1（倍精度，GPU）の性能も測定した．またCPUとの比較のため，CUDDBLAS，CUQDBLASと同じ演算を行うDD型4倍精度演算およびQD型8倍精度演算に対応したBLASをCPU向けに実装し，性能を測定した．これらのBLASをそれぞれDDBLAS，QDBLASと呼ぶ．CPU向けのDD型4倍精度演算およびQD型8倍精度演算に対応したBLASとしてMBLASが存在するが，MBLASの現バージョンでは一部の関数を除いてマルチスレッド実行に対応していない．DDBLAS，QDBLASはQD 2.3.11[4]を用いて実装し，OpenMPによる並列化，キャッシュブロッキングなどの最適化を行った．

CPUにはIntel Core i7 920（2.67GHz，Quad-Core）を使用し，Hyper-Threadingは無効として，GotoBLAS，DDBLAS，QDBLASはCPUのコア数と同じ4スレッドで実行した．OSはCentOS 5.5（x86-64，kernel 2.6.18-194.11.4.el5），CUDAはVersion 3.1，コンパイラはCPUコードにg++ 4.1.2（–O3），GPUコードにnvcc 3.1（–O3）を使用したが，DDBLAS，QDBLASおよびQD 2.3.11のみIntel C++コンパイラicpc 11.1（–fast）を用いた．

演算性能は1秒間に計算した倍精度演算回数，DD型4倍精度演算回数，QD型8倍精度演算回数をそれぞれFlops，DDFlops，QDFlopsで示した．実行時間の測定は実行時間が1秒以上となるように繰り返し回数を調節して測定したものの平均値とした．またGPUで動作するBLASの性能にはCPU–GPU間のPCIeによるデータ転送時間は含めていない．行列およびベクトルは一様乱数で初期化された$N \times N$の正方行列，および大きさNのベクトルである．4倍・8倍精度乱数の生成にはQDライブラリの乱数生成関数dd_randおよびqd_randを使用した．なおAXPYのαは乱数で初期化し，GEMV，GEMMのαは1.0，βは0.0に固定した．

4.2 AXPY

倍精度演算，4倍精度演算，8倍精度演算によるAXPYの評価結果を**図2**–**図4**にそれぞれ示す．AXPYは$2N$回の演算と$3N$回のメモリのロード・ストアが発生し，演算回数とロード・ストア回数の比が2:3となる．Byte/Flop値が大きく，演算性能がメモリ性能に依存する．$N = 8,192,000$の場合，倍精度演算はTesla C2050で約10.6GFlopsとなり，理論ピーク演算性能の約2.1%の性能であった．一方，4倍精度演算ではTesla C2050で約5.0GDDFlopsとなった．表1よりFMA命令を使用した4倍精度積和演算（2DDFlop）は合計30Flopの倍精度演算で構成されることから，約5.0GDDFlopsは倍精度演算では約75.5GFlopsに相当し，これは理論ピーク演算性能の約14.7%の性能である．同様にTesla C2050の8倍精度演算では約0.76GQDFlopsで，倍精度演算では約107.8GFlopsに相当し，理論ピーク演算性能の約

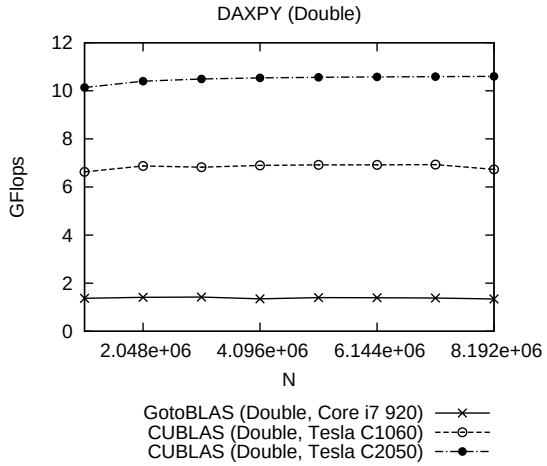

図 2 倍精度 AXPY の性能

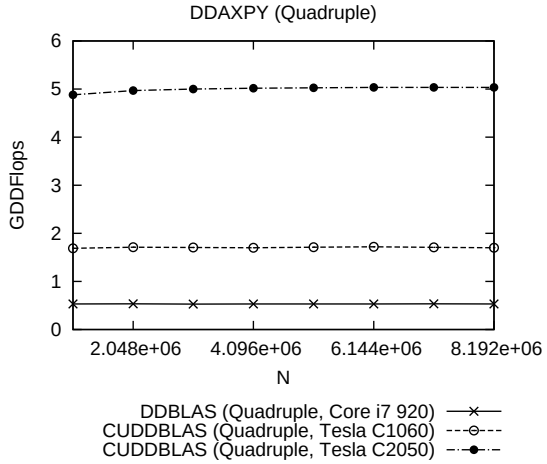

図 3 4 倍精度 AXPY の性能

20.9%の性能となった．4 倍・8 倍精度演算では倍精度演算に比べ演算量が増加した分 Byte/Flop 値が小さくなり，GPU の演算性能を引き出せていることが分かる．

一方で CPU に対し Tesla C2050 の 4 倍精度演算は約 9.5 倍，8 倍精度演算は約 19 倍高速となった．また 4 倍精度 AXPY は倍精度 AXPY の 15 倍の演算量であるが，CPU では倍精度 AXPY の約 2.5 倍の演算時間，Tesla C2050 では倍精度 AXPY の約 2.1 倍の演算時間であった．AXPY の性能はメモリ性能に依存するため倍精度演算とのコスト差は小さくなるが，DD 型演算でメモリアクセス量が 2 倍となるため，性能は倍精度の約半分に近づいたと考えられる．また 8 倍精度 AXPY は倍精度演算に対し，CPU では約 34 倍の演算時間を要したのに対し，Tesla C2050 では約 14 倍の演算時間に抑えられた．

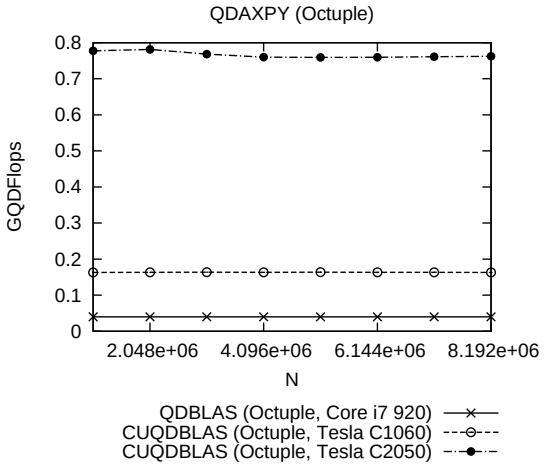

図 4 8 倍精度 AXPY の性能

4.3 GEMV

倍精度演算，4 倍精度演算，8 倍精度演算による GEMV の評価結果を**図 5**–**図 7** に示す．$N = 8,192$ の場合，CPU に対して Tesla C2050 の 4 倍精度演算が約 18 倍，8 倍精度演算が約 19 倍高速となった．CPU では倍精度 GEMV に対して 4 倍精度 GEMV が約 6.4 倍の演算時間，8 倍精度 GEMV が約 89 倍の演算時間を要していたのに対し，Tesla C2050 ではそれぞれ倍精度 GEMV に対し約 3.1 倍，約 41 倍の演算時間に抑えられた．

GEMV では $2N^2$ 回の演算とグローバルメモリに対する N^2+2N 回のロード・ストアが発生し，演算回数とロード・ストア回数の比は約 2:1 と，AXPY の 2:3 に比べ Byte/Flop 値は小さくなる．さらにデータの再利用が可能となるため，実装次第では AXPY よりも高い演算性能が得られるはずである．しかし CPU の倍精度 GEMV は倍精度 AXPY の約 2.7 倍の性能である一方，4 倍・8 倍精度 GEMV は 4 倍・8 倍精度 AXPY とほぼ同じ性能であった．また Tesla C1060 においても倍精度 GEMV が倍精度 AXPY の約 3.2 倍の性能である一方で，4 倍・8 倍精度演算では CPU と同様に GEMV と AXPY の性能がほとんど変わらない結果となった．これに対して Tesla C2050 では倍精度 GEMV が倍精度 AXPY の約 3.0 倍，4 倍精度 GEMV は 4 倍精度 AXPY の約 2.0 倍の性能が得られ，8 倍精度 GEMV は 8 倍精度 AXPY とほぼ同じ性能となった．CPU と Tesla C1060 では 4 倍・8 倍精度演算によって演算量が増加したことで，倍精度演算では通常，メモリ性能に律速されるはずの AXPY が演算性能に律速されるようになったためと考えられる．また Tesla C2050 では演算性能が高い分，4 倍精度 AXPY ではメモリ性能に律速され，8 倍精度 AXPY で演算性能に律速されたと考えられる．これらと同様の傾向は後述する GEMM においても見られる．

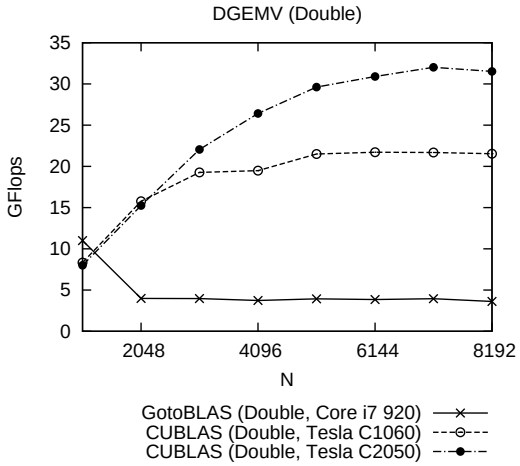

図 5 倍精度 GEMV の性能

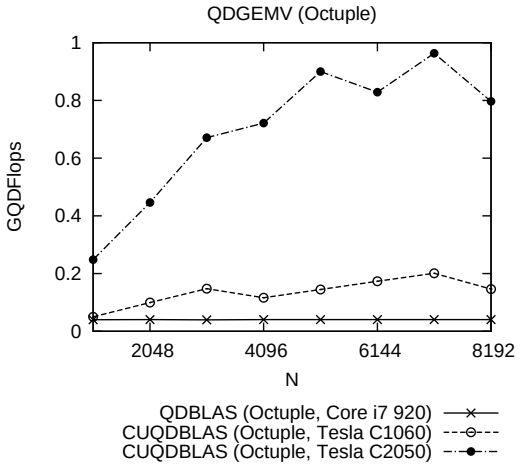

図 7 8 倍精度 GEMV の性能

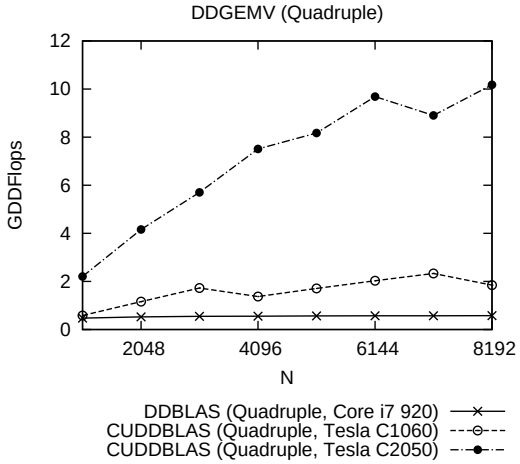

図 6 4 倍精度 GEMV の性能

4.4 GEMM

倍精度演算，4 倍精度演算，8 倍精度演算による GEMV の評価結果を**図 8**–**図 10** に示す．$N = 4,096$ の場合，CPU に対して Tesla C2050 による 4 倍精度演算が約 29 倍，8 倍精度演算が約 24 倍高速化された．また CPU では倍精度演算に対して 4 倍精度演算に約 84 倍，8 倍精度演算に約 1016 倍もの時間を要していたのに対し，Tesla C2050 ではそれぞれ約 12 倍，約 179 倍の計算時間となり，倍精度演算に対する演算時間の増大が大幅に削減された．

理論ピーク演算性能に対する実効性能では，倍精度 GEMM が Tesla C1060 では約 75.2GFlops で，理論ピーク演算性能の約 96.4%の性能，Tesla C2050 では約 173.8GFlops で約 33.8%の性能となった．GEMM は $2N^3$ 回の演算とグローバルメモリに対する $3N^2$ 回のロード・ストアが発生し，演算回数とロード・ストア回数の比は $2N : 3$ である．GEMM は AXPY，GEMV と比較して Byte/Flop 値が小さいため，通常はプロセッサの理論ピーク演算性能に近い性能が得られる処理であるが，Tesla C2050 ではメモリ性能に対して理論ピーク演算性能が高く，GEMM でもメモリバンド幅に律速されていることが分かる．一方，**図 9** の 4 倍精度演算では Tesla C1060 が約 2.6GDDFlops，Tesla C2050 が約 14.2GDDFlops となった．これは倍精度演算ではそれぞれ約 39GFlops，約 212.8GFlops に相当し，理論ピーク演算性能に対してはそれぞれ約 50.1%，約 41.3%の性能である．また図 10 の 8 倍精度演算では Tesla C1060 が約 0.18GQDFlops，Tesla C2050 が約 0.97GQDFlops で，それぞれ倍精度演算では約 25.8GFlops，約 137.4GFlops に相当し，理論ピーク演算性能に対してそれぞれ約 25.8%，約 33.1%の性能である．DD 型・QD 型演算では FMA 命令の活用で演算量を削減しているが，それでも全浮動小数点演算に占める FMA 命令の使用率は DD 型演算で約 3.4%，QD 型演算で約 3.7%であり，最大でも演算性能は理論ピーク演算性能の約半分しか得られないことになる．

表 2 FMA 命令の効果（AXPY: N=8,192,000, GEMV: N=8,192, GEMM: N=4,096）

	FMA 使用	FMA 未使用
4 倍精度 AXPY	5.03 GDDFlops	4.96 GDDFlops
8 倍精度 AXPY	0.76 GQDFlops	0.70 GQDFlops
4 倍精度 GEMV	10.18 GDDFlops	7.10 GDDFlops
8 倍精度 GEMV	0.80 GQDFlops	0.62 GQDFlops
4 倍精度 GEMM	14.19 GDDFlops	10.06 GDDFlops
8 倍精度 GEMM	0.97 GQDFlops	0.74 GQDFlops

4.5 FMA 命令の効果

これまでに示した CUDDBLAS および CUQDBLAS のデータは FMA 命令を使用した場合のデータである．FMA 命令の効果を調べるため，FMA 命令を使用しなかった場合の各関数の性能を測定した．

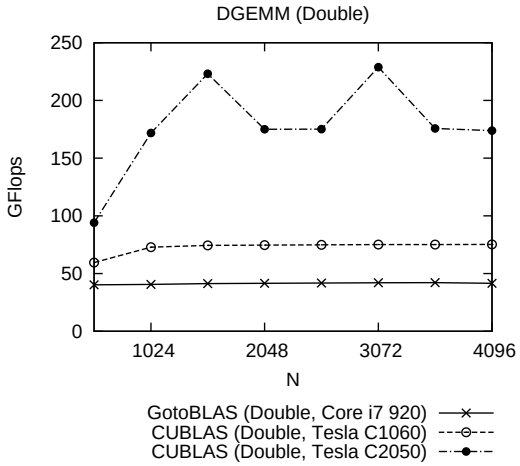

図 8 倍精度 GEMM の性能

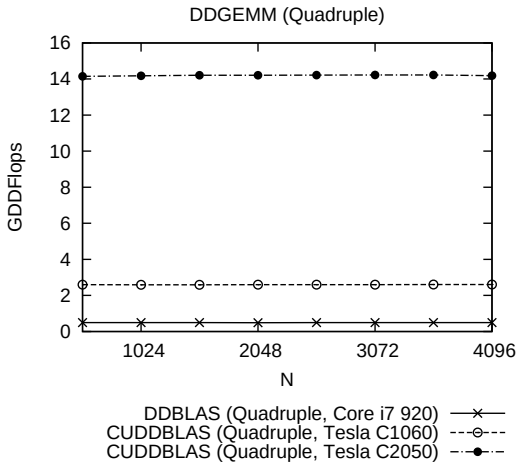

図 9 4 倍精度 GEMM の性能

結果を**表 2** に示す．データ転送量に対して演算量の比に応じて FMA 命令の効果が得られていることが分かる．FMA 命令を使用した 4 倍精度 GEMM では FMA 命令を使用しなかった場合の約 1.4 倍，8 倍精度 GEMM では約 1.3 倍の性能となった．理論値として FMA 命令を使用しない場合の演算量は FMA を使用した場合に比べ，4 倍・8 倍精度ともに積和演算で約 1.5 倍であることから，FMA 命令の効果はほぼ理論通りであるといえる．

4.6 PCIe データ転送時間を考慮した場合

GPU の BLAS 関数が CPU 側のメモリ上に配置されたデータに対して演算を行う場合，すなわち GPU を BLAS のアクセラレータとして用いる場合には，CPU–GPU 間の PCIe データ転送が必要となる．しかし PCIe の理論ピークバンド幅は 8GB/s しかなく，GPU 側のメモリバンド幅（Tesla C1060 で 102GB/s，Tesla C2050 で 144GB/s）と比べて大きく劣り，GPU の高い演算性能に対して PCIe バンド幅が性能上の大きなボトルネックとなることが知られている．

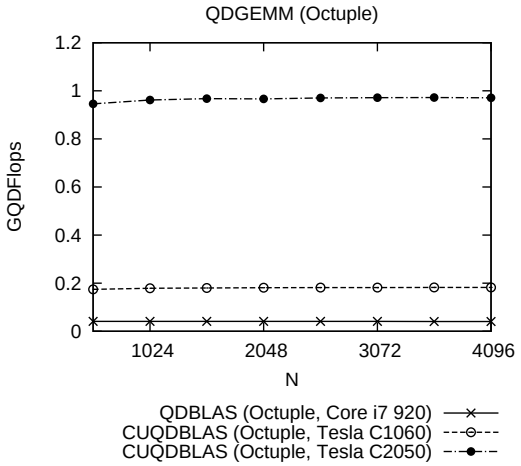

図 10 8 倍精度 GEMM の性能

図 11–**図 13** に各関数の PCIe データ転送を含めた全体の実行時間に占める PCIe データ転送時間の割合を示す．ホストメモリはページロックされており，いずれの BLAS も演算とデータ転送のオーバーラップは行われていない．まず図 11 の AXPY では Tesla C1060 で倍精度演算の場合，約 93.1%が PCIe データ転送に費やされているが，4 倍精度演算で約 87.2%，8 倍精度演算で約 56.8%となり，演算量が増えたことにより相対的に転送時間の割合は低下した．Tesla C2050 では演算性能が Tesla C1060 の約 6.6 倍に向上している一方で PCIe の帯域は変わらないため，PCIe データ転送に要する時間の割合が増える傾向にある．図 12 の GEMV でも同様の傾向である．一方，図 13 の GEMM はデータ転送に対して演算量が多く，演算の割合がほとんどを占めるが，倍精度演算では Tesla C1060 で約 3.5%，Tesla C2050 で約 7.6%が PCIe データ転送に費やされており，PCIe による律速が存在する．一方，4 倍・8 倍精度ではこれが共に 1.5%以下となり，ほとんど PCIe データ転送の影響を受けなくなることが分かる．これらの結果から，4 倍・8 倍精度演算は PCIe による律速の影響を改善し，Byte/Flop 値の小さい処理として GPU に適したアプリケーションであるといえる．

5. ま と め

本論文では DD 型 4 倍精度演算および QD 型 8 倍精度演算に対応した BLAS 関数を GPU 上に実装し，性能評価を行った．NVIDIA Tesla C2050 での性能評価では，Intel Core i7 920 での同一処理と比べ，4 倍精度 GEMM で約 29 倍，8 倍精度 GEMM で約 24 倍の高速化を達成した．また倍精度演算に対する計算

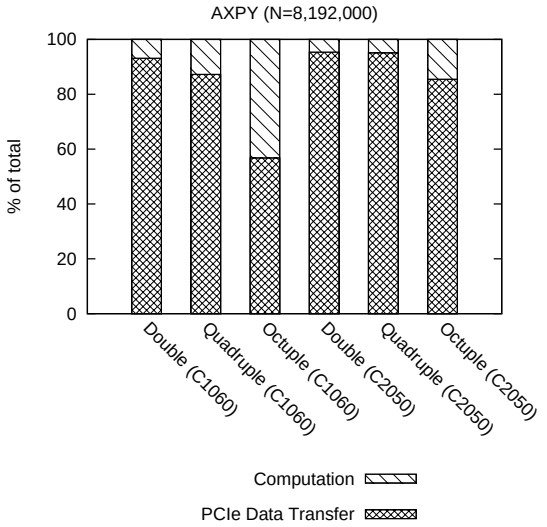

図 11 全体の実行時間に占める PCIe データ転送時間の割合（AXPY）

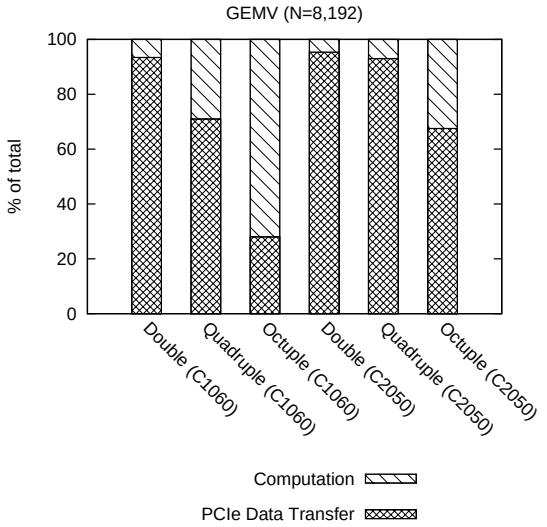

図 12 全体の実行時間に占める PCIe データ転送時間の割合（GEMV）

コストの増加も，CPU では 4 倍精度 GEMM が約 84 倍，8 倍精度 GEMM では約 1016 倍の時間を要していたのに対し，Tesla C2050 ではそれぞれ約 12 倍，約 179 倍と大幅に演算時間の増大を削減した．これは GPU が高い演算性能を有していることに加え，GPU では DD 型・QD 型乗算で FMA 命令を利用することにより，演算量が大幅に削減されたからである．また PCI-Express（PCIe）によるデータ転送時間を考慮した場合，倍精度演算では GEMM であっても PCIe データ転送性能に律速される傾向が見られたが，これが 4 倍・8 倍精度演算を行うことでほぼ解消されることが示された．これらの結果から，4 倍・8 倍精度演算は GPU に適した処理であるといえる．

今後は BLAS の全関数の実装，また実アプリケーションでの評価を行う予定である．一例として，精度が収束に影響しやすい連立一次方程式の反復解法に GPU による 4 倍・8 倍精度 BLAS を適用することで，CPU に比べ求解の高速化が期待できる．連立一次方程式の反復解法は主に AXPY，DOT，SpMV といった Byte/Flop 値の比較的大きい，メモリ性能に影響する処理で構成されている．本研究で Tesla C2050 では 4 倍精度 AXPY が倍精度演算の高々2.1 倍の演算時間であったことから，これらの演算も倍精度演算の 2 倍程度の時間で処理できると期待される．倍精度演算に対して反復回数が半減すれば，求解にかかる時間の高速化も可能であると考えられる．

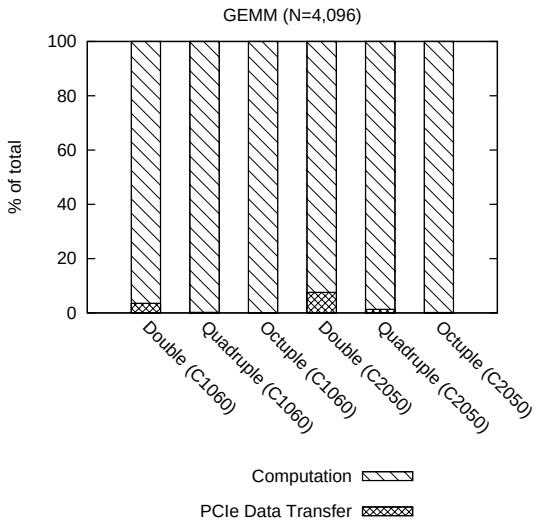

図 13 全体の実行時間に占める PCIe データ転送時間の割合（GEMM）

参 考 文 献

1) Granlund, T.: GMP: GNU Multiple Precision Arithmetic Library, http://gmplib.org/.
2) Fousse, L., Hanrot, G., Lefevre, V., Pelissier, P. and Zimmermann, P.: MPFR : GNU MPFR Library, http://www.mpfr.org/.
3) Bailey, D. H.: ARPREC (C++/Fortran-90 arbitrary precision package), http://crd.lbl.gov/˜dhbailey/mpdist/.
4) Bailey, D. H.: QD (C++ / Fortran-90 double–double and quad-double package), http://crd.lbl.gov/˜dhbailey/mpdist/.
5) Li, X. S., Demmel, J. W., Bailey, D. H., Hida, Y., Iskandar, J., Kapur, A., Martin, M. C., Thompson, B., Tung, T. and Yoo, D. J.:

XBLAS – Extra Precise Basic Linear Algebra Subroutines.
6) 中田真秀: The MPACK; Multiple precision arithmetic BLAS (MBLAS) and LAPACK (MLAPACK), http://mplapack.sourceforge.net/.
7) Göddeke, D., Strzodka, R. and Turek, S.: Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations, *International Journal of Parallel, Emergent and Distributed Systems 22* (2007).
8) Thall, A.: Extended-Precision Floating-Point Numbers for GPU Computation, *ACM SIGGRAPH 2006 Research Posters* (2006).
9) 中里直人, 石川正, 牧野淳一郎, 湯浅富久子: アクセラレータによる四倍精度演算, 情報処理学会研究報告, Vol. 2009–HPC–121, No. 39 (2009).
10) Zhao, K. and Chu, X.: GPUMP: a Multiple-Precision Integer Library for GPUs, *Proc. IEEE International Conference on Computer and Information Technology (CIT 2010)* (2010).
11) Lu, M., He, B. and Luo, Q.: Supporting Extended Precision on Graphics Processors, *Proc. Sixth International Workshop on Data Management on New Hardware (DaMoN 2010)* (2010).
12) Hida, Y., Li, X. S. and Bailey, D. H.: Algorithms for Quad-Double Precision Floating Point Arithmetic, *Proc. 15th Symposium on Computer Arithmetic*, pp. 155–162 (2001).
13) 椋木大地, 高橋大介: GPU による 4 倍精度 BLAS の実装と評価, 情報処理学会研究報告, Vol. 2009–HPC–122, No. 13 (2009).