情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-ARC-186 No.13
Vol.2009-HPC-123 No.13
2009/12/1

GPUによる4倍精度BLASの実装と評価

椋木大地[†1]　高橋大介[†1]

本稿ではGPU（Graphics Processing Unit）で動作する4倍精度BLAS（Basic Linear Algebra Subprograms）の実装を行った．GPUはメモリ性能に対して演算性能が高く，性能を引き出すには演算密度の高いアプリケーションが求められる．我々は演算密度の高い処理として倍精度演算を組み合わせたDouble–Double型（DD型）の4倍精度演算と行列積に着目し，GPU向けの汎用計算開発環境であるCUDA（Compute Unified Device Architecture）を用いて，DD型4倍精度BLASを実装した．NVIDIA Tesla C1060における性能評価では，Intel Core i7 920上で実行したCPU向けのDD型4倍精度BLASであるMBLASと比較し，DD型4倍精度行列積（DDGEMM）で最大約30倍の性能を得た．

Implementation and Evaluation of Quadruple Precision BLAS on GPU

Daichi Mukunoki[†1] and Daisuke Takahashi[†1]

We implemented a quadruple precision BLAS (Basic Linear Algebra Subprograms) on GPU (Graphics Processing Unit). Since GPU computing performance is much higher than memory bandwidth, it needs the computation-intensive applications to give its best performance. As a computation-intensive operation, we focused on Double–Double (DD) quadruple precision operations combined double precision operations and matrix multiplication. We implemented a quadruple precision BLAS using CUDA (Compute Unified Device Architecture) which is general purpose computing environment for GPUs. The experimental results on NVIDIA Tesla C1060 show that DD precision matrix multiplication (DDGEMM) runs maximum 30 times faster than MBLAS on Intel Core i7 920.

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

1. はじめに

近年，GPU（Graphics Processing Unit）を汎用計算のためのアクセラレータとして用いるGPGPU（General Purpose computing on GPU）が注目されている．GPUはCPUと比較して高い演算性能がある一方，特殊なアーキテクチャであるが故に高い実性能を得られるアプリケーションは限られている．

メニーコアプロセッサであるGPUは各コアがスレッドを並列に実行し，単一命令を複数のスレッドで処理するSIMT（Single Instruction Multiple Thread）アーキテクチャである．このため複数のデータに対して同一処理を行うベクトル演算に性能を発揮する．またGPUは高速な専用メモリであるVRAM（Video RAM）を持っているが，CPUからVRAMに，あるいはGPUからCPU側のメインメモリに直接アクセスを行うことはできず，PCI Express等の拡張バスを経由してデータを移動させる必要がある．これらのバスは最も高速なものでもPCI Express x16の8GB/secにとどまり，100GB/sec程度のメモリバンド幅を持つGPUと比較すると極めて低速である．そのためGPUをアクセラレータとしてCPU側のメインメモリ上にあるデータを処理する場合，演算回数とメモリのロード・ストア回数の比率である演算密度が高い処理でなければ，GPUの高い演算性能を生かすことができない．

そこで本研究ではGPUに適する演算密度の高いアプリケーションとして，倍精度演算の組み合わせによるDouble–Double型（DD型）の4倍精度演算と行列積に着目し，ベクトルと行列の基本演算を行うサブルーチン群であるBLAS（Basic Linear Algebra Subprograms）の4倍精度実装を行った．4倍精度演算は高精度が要求される科学技術計算や反復法の収束改善といった目的での需要があるほか，近年はコンピュータの演算性能が向上し，演算規模の拡大とともに蓄積誤差の増大が問題となり，倍精度に代わって4倍精度の必要性が議論されている．しかし一般的なCPUがハードウェアで処理できるのは倍精度演算までであり，4倍精度演算の実現には多数の倍精度演算を組み合わせるなどしたソフトウェアエミュレーションを用いることになる．一方BLASは演算量に応じてLevel–1から3までの3段階に分類され，行列積などの最も演算量が大きいLevel–3 BLASでは次元数$n \times n$の行列の場合，演算量が$O(n^3)$となる．したがってLevel-3 BLASの4倍精度演算は演算密度の高い処理となりGPUでの高速化が期待できる．

以下，第2章で関連研究について，第3章でDD型による4倍精度演算アルゴリズムについて述べる．そして第4章でCUDA[1)]によるGPU向け4倍精度BLAS（CUDDBLAS）

© 2009 Information Processing Society of Japan

Vol.2009-ARC-186 No.13
Vol.2009-HPC-123 No.13
2009/12/1

の実装，第 5 章では Tesla C1060 による性能測定の結果と考察を行い，最後に第 6 章でまとめと今後の課題について述べる．

2. 関 連 研 究

CPU で動作する 4 倍精度 BLAS として MBLAS[4] が開発されている．MBLAS は整数演算を用いた多倍長演算ライブラリである GMP（GNU Multi Precision Library）[3] による任意精度演算のほか，Bailey が開発した DD 型，Quad–Double 型（QD 型）による多倍長演算ライブラリである QD[7] を用いた 4 倍，8 倍精度演算に対応している．同じく DD 型演算を用いる XBLAS[2] は，内部で DD 型演算を行っているものの入出力データは倍精度となっており，入出力データを含めた完全な DD 型演算と比べ精度は低くなっている．また GPU 上で動作する BLAS として GPGPU 開発環境 CUDA（Compute Unified Device Architecture）向けに実装された CUBLAS[5] が存在するが，単精度および倍精度のみの対応となっている．一方で中里ら[6] は DD 型で GPU 上に 4 倍精度演算環境の実装を行い GPU による多倍長演算の有効性を示しているが，BLAS 単位での実装評価はこれまでのところされていない．

現時点で GPU 上で動作する 4 倍精度 BLAS は存在しておらず，本稿では CUDA による DD 型精度の BLAS を CUDDBLAS と名付け実装を行い，CPU 向けの 4 倍精度 BLAS である MBLAS と比較検討を行う．

3. 4 倍精度演算

4 倍精度演算のソフトウェアエミュレーションには整数演算による方法と浮動小数点演算による方法の 2 種類があり，整数演算による方法には FORTRAN の REAL*16 型や，任意精度の GMP が存在する．一方，浮動小数点演算を用いた方法として，2 つの倍精度浮動小数点数を用いて 4 倍精度浮動小数点数を表現し，倍精度の四則演算を組み合わせて 4 倍精度演算を実現する DD 型のアルゴリズムが考案されており，Bailey らによる実装が知られている．DD 型では 2 つの倍精度浮動小数点数を用いるため，仮数部が 52 ビット × 2 の 104 ビットとなり IEEE754 の 4 倍精度データ形式の 112 ビットより 8 ビット少なくなっている．また指数部のビット数は拡張されないため，倍精度に比べて保持できる絶対値が大きくなることはない．

本稿では 4 倍精度演算手法として Bailey の多倍長演算ライブラリ QD における DD 型のアルゴリズムを適用した．その理由として，GPU が整数演算を得意としないこと，4 倍精度においては DD 型アルゴリズムの方が整数演算による方法に比べて演算量が少ない[8] ことが挙げられる．また本稿執筆時点では単精度演算性能が倍精度と比べ 10 倍以上という GPU が多く，DD 型と同様の手法で単精度演算を組み合わせた Quad–Float 型（QF 型）の実装も考えられるが，演算量が DD 型よりも大幅に増加するうえ，次世代 GPU では倍精度演算性能が単精度の半分程度に改善されることなどから検討しないこととする．以下本章では Bailey の DD 型アルゴリズムについて，BLAS の実装に必要な積和演算に関するものを取り上げて説明する．

Quick-TwoSum(a, b, s, e)\{
  $s = a + b$
  $e = b - (s - a)$
\}

図 1 Dekker による加算

TwoSum(a, b, s, e)\{
  $s = a + b$
  $v = s - a$
  $e = (a - (s - v)) + (b - v)$
\}

図 2 Knuth による加算

SPLIT(a, h, l)\{
  $t = 134217729.0 * a$
  $h = t - (t - a)$
  $l = a - h$
\}

図 3 SPLIT

3.1 丸め誤差を考慮した演算

Bailey の DD 型による 4 倍精度演算アルゴリズムは，Dekker[9]，Knuth[10] らによる丸め誤差を考慮した浮動小数点演算のアルゴリズムを基にしている．まず全ての演算が IEEE 754 形式の倍精度で round–to–even 丸めモードであると仮定する．倍精度浮動小数点数 a と b の加算 $a + b$ は $a + b = fl(a + b) + err(a + b)$ と表し，$fl(a + b)$ を $a + b$ の浮動小数点演算結果，$err(a + b)$ を $a + b$ の演算で生じた誤差と定義する．

図 1 に Dekker による加算アルゴリズム Quick-TwoSum(a, b, s, e) を示す．このアルゴリズムは $s = fl(a + b)$，$e = err(a + b)$ を計算し，引数に $|a| \geq |b|$ が仮定される場合のみに適用される．一方で**図 2** に示す Knuth の加算アルゴリズム TwoSum(a, b, s, e) は同様の計算を行うが，引数の大小関係は問わない．その代わりに演算回数は 6 回となり Quick-TwoSum

© 2009 Information Processing Society of Japan

Vol.2009-ARC-186 No.13
Vol.2009-HPC-123 No.13
2009/12/1

```
TwoProd(a, b, p, e){
    p = a * b
    SPLIT(a, a_H, a_L)
    SPLIT(b, b_H, b_L)
    e = ((a_H * b_H − p) + a_H * b_L + a_L * b_H) + a_L * b_L
}
```

図 4 Dekker による乗算

```
TwoProd-FMA(a, b, p, e){
    p = a * b
    e = fl(a * b − p)
}
```

図 5 積和演算を用いた乗算

の 3 回と比べ多くなっている．

図 3 に示す SPLIT(a, h, l) は，倍精度浮動小数点数 a を $a = h + l$ として 2 つの倍精度浮動小数点数に分離するものである．ここで h は a の仮数部上位 26 ビット分，l は下位 26 ビット分を持つ．

図 4 に Dekker による乗算アルゴリズム TwoProd(a, b, p, e) を示す．このアルゴリズムは $p = fl(a \times b)$，$e = err(a \times b)$ を計算する．なお，$a \times b + c$ の演算を 1 命令実行可能な積和演算命令 FMA（Fused Multiply Add）が倍精度演算を丸め誤差なしで処理可能な場合，TwoProd に代わって**図 5** の TwoProd-FMA(a, b, p, e)[11] を用いることができる．ここで $fl(a*b-p)$ は $a \times b - p$ を計算する FMA 命令を表している．TwoProd では 17 回の浮動小数点演算が必要であるのに対し，TwoProd-FMA では 3 回となり，大幅に演算回数を削減することが可能となる．計算結果を検証した結果，本稿で使用する GPU ハードウェアには倍精度演算を丸め誤差なしで計算可能な FMA 命令が搭載されており，実装にはこのアルゴリズムを採用している．

3.2 DD 型の 4 倍精度演算

前述した Dekker，Knuth による丸め誤差を考慮した演算を用いて，Bailey の DD 型による 4 倍精度演算アルゴリズムが可能となる．4 倍精度浮動小数点数 a が 2 つの倍精度浮動小数点数

```
QuadAdd(a_H, a_L, b_H, b_L, c_H, c_L){
    TwoSum(a_H, b_H, sh, eh)
    TwoSum(a_L, b_L, sl, el)
    eh = eh + sl
    Quick-TwoSum(sh, eh, sh, eh)
    eh = eh + el
    Quick-TwoSum(sh, eh, c_H, c_L)
}
```

図 6 4 倍精度加算

```
QuadMul(a_H, a_L, b_H, b_L, c_H, c_L){
    TwoProd(a_H, b_H, p1, p2)
     (TwoProd-FMA(a, b, p1, p2))
    p2 = p2 + (a_H * b_L + a_L * b_H)
    Quick-TwoSum(p1, p2, c_H, c_L)
}
```

図 7 4 倍精度乗算

a_H と a_L によって $a = a_H + a_L$，$|a_H| > |a_L|$，同様に $b = b_H + b_L$，$c = c_H + c_L$ と表されるとき，4 倍精度加算 $c = a + b$ を行うアルゴリズム QuadAdd$(a_H, a_L, b_H, b_L, c_H, c_L)$ を**図 6** に示す．また同様に 4 倍精度乗算 $c = a \times b$ を行うアルゴリズム QuadMul$(a_H, a_L, b_H, b_L, c_H, c_L)$ を**図 7** に示す．

4. CUDA による 4 倍精度 BLAS の実装

CUDA は NVIDIA 社が提供する GPGPU 開発環境である．対応製品は同社製 GPU に限られるが，GPU を並列計算機として C 言語を拡張した言語で容易に開発することができる．CUDA に対応する製品の中で倍精度演算が可能なものは Compute Capability 1.3 を備える GT200 アーキテクチャ以降を採用した製品に限定される．

4.1 GT200 アーキテクチャ

GT200 アーキテクチャの概略を**図 8** に示す．GPU は同一処理を行う複数のスレッドを多数のコアで処理する SIMT アーキテクチャであり，スレッドは単精度演算を行う SP（Streaming Processor），倍精度演算を行う DP（Double Precision Unit）で処理される．これらは 8 個の SP と 1 個の DP で SM（Streaming Multiprocessor）としてまとめられ，この中で内容が共有されるキャッシュ相当の Shared Memory（16KB）を持つ．スレッドは SM に対して複数スレッドを束ねたスレッドブロックという単位で処理される．Global Memory はいわゆる VRAM に相当し，全ての SP と DP からアクセス可能で大容量だが Shared Memory と比べ低速である．またホスト PC 側のメインメモリには直接アクセスできず，一旦メインメモリのデータを Global Memory へ転送し処理する必要がある．

スレッドブロックは最大 65535×65535 の 2 次元，また 1 スレッドブロックあたりのス

© 2009 Information Processing Society of Japan

Vol.2009-ARC-186 No.13
Vol.2009-HPC-123 No.13
2009/12/1

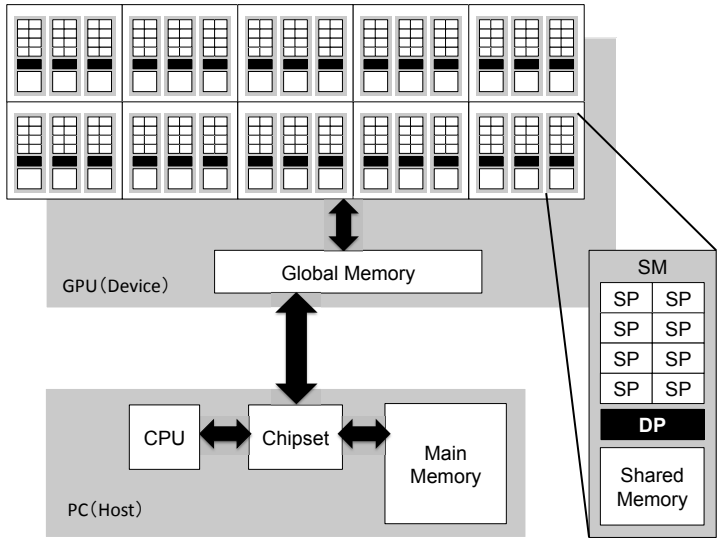

図 8 GT200 アーキテクチャ

レッドは最大 512 × 512 × 64 の 3 次元（ただし合計で最大 512）の ID が与えられており，プログラムからはこの ID を利用することで並列処理が可能となる．行列演算などでは ID を用いることで for 文を置き換えることが可能となる．

4.2 CUDDBLAS の実装

本稿では Level–1 BLAS のベクトル加算（AXPY），Level–2 BLAS の行列ベクトル積（GEMV），Level–3 BLAS の行列積（GEMM）の実装を行った．AXPY ではベクトルを 64 要素単位でブロックに分割し，各スレッドが 1 ブロックで処理を行っている．GEMV では行方向にスレッドを割り当て，各スレッドが行列の 1 行とベクトルの内積を行うようにしている．GEMM では 16 × 16 でブロック化を行い，各スレッドがブロックを Shared Memory に格納することでアクセス速度の向上を図っている．また GEMV と同様に各スレッドが内積を行う．いずれもブロック単位の処理やスレッド，スレッドブロックの生成数に依存した実装となっており，行列サイズが端数となる場合の処理は実装していないため，現状では計算できる行列サイズが限定されている．

表 1 評価環境

CPU	Intel Core i7 920（2.67GHz）
メインメモリ	12GB（DDR3）
GPU	NVIDIA Tesla C1060
ビデオメモリ	4GB（GDDR3）
GPU 接続バス	PCI-Express x16
OS	CentOS 5.3（x86-64），kernel 2.6.18
CUDA 環境	CUDA SDK 2.30, CUDA Driver 2.30

表 2 測定対象

	ハードウェア	演算精度	備考
GotoBLAS	CPU	Double	GotoBLAS2-1.00
CUBLAS	GPU	Double	CUBLAS 2.3
CUBLAS-kernel	GPU	Double	カーネル実行時間のみ測定
MBLAS	CPU	Quad（DD）	Version: 0.5.2 + QD 2.3.8
CUDDBLAS	GPU	Quad（DD）	
CUDDBLAS-kernel	GPU	Quad（DD）	カーネル実行時間のみ測定

5. 評価実験

5.1 評価方法

Level–1 BLAS のベクトル加算（AXPY），Level–2 BLAS の行列ベクトル積（GEMV），Level–3 BLAS の行列積（GEMM）の評価を行う．計算対象は一様乱数で初期化したベクトルおよび正方行列である．性能評価には GPU として GT200 アーキテクチャの GPGPU 専用ハードウェアである NVIDIA Tesla C1060，CPU には Intel Core i7 920 を用いた．両者の倍精度浮動小数点演算理論ピーク性能はそれぞれ 78GFLOPS，42.72GFLOPS である．測定環境を**表 1** に示す．また性能比較を行う BLAS を**表 2** に示す．各 BLAS は本稿執筆時点で最新のものを用い，CPU 側のコンパイラは gcc 4.1.2，GPU 側のコンパイラは CUDA SDK 2.3 に含まれる nvcc 2.3 で，最適化オプションはすべて–O3 を指定している．MBLAS での DD 型演算に必要な QD は最新の 2.3.8 を導入したが，デフォルトでは Cray タイプの倍精度環境を想定したアルゴリズムが有効になっており，IEEE 互換の倍精度環境を想定した DD 型演算を行うようにオプションを指定してコンパイルしている．

性能は 1 秒間に行った浮動小数点演算の回数である FLOPS（Floating point number

© 2009 Information Processing Society of Japan

Vol.2009-ARC-186 No.13
Vol.2009-HPC-123 No.13
2009/12/1

表 3 演算性能と理論ピーク性能比（倍精度）[GFLOPS]

BLAS	AXPY N=26,214,400		GEMV N=8128		GEMM N=4096	
	性能	ピーク比	性能	ピーク比	性能	ピーク比
GotoBLAS	1.21	2.83%	3.78	8.86%	39.4	92.3%
CUBLAS	0.33	0.42%	0.90	1.15%	52.4	67.2%
CUBLAS-kernel	6.94	8.90%	21.3	27.3%	75.3	96.6%

表 4 演算性能と理論ピーク性能比（DD 型 4 倍精度）[GDDFLOPS]

BLAS	AXPY N=26,214,400		GEMV N=8128		GEMM N=4096	
	性能	ピーク比	性能	ピーク比	性能	ピーク比
MBLAS	0.089	3.12%	0.084	2.95%	0.089	3.13%
CUDDBLAS	0.129	2.48%	0.276	5.31%	2.610	50.2%
CUDDBLAS-kernel	0.334	6.42%	0.505	9.70%	2.626	50.5%

※ 4 倍精度 BLAS のピーク比は実際の倍精度演算回数に対して算出

Operations Per Second），また 1 秒間に行った DD 型演算の回数を DDFLOPS として実行時間から算出した．実行時間は GPU で実行する CUBLAS，CUDDBLAS については GPU のカーネル実行時間のみを測定したもの（–kernel と表記）と，BLAS を CPU 側のプログラムから呼び出すことを想定し，GPU 側のメモリアロケーション，往復のデータ転送，カーネル実行，メモリ解放の全時間を測定したものの 2 種類を測定した．なお GPU の初期化には 1 秒程度を要するが，アプリケーション中で BLAS を何度か呼び出すことを想定し，この時間は含めていない．

5.2 評価結果

これらの測定結果を**図 9**–**図 11** に示す．左図が倍精度演算，右図が DD 型 4 倍精度演算である．また倍精度版 BLAS の演算性能を**表 3**，4 倍精度版 BLAS の演算性能を**表 4** に示す．

AXPY は GotoBLAS でも理論性能の 2.83%程度の性能となり演算密度が低くメモリ性能に律速される処理である．倍精度では GotoBLAS の性能に対し CUBLAS を CPU から呼び出した場合の性能は 3 割程度であるが，4 倍精度では CPU の MBLAS に対して CUDDBLAS がメモリの転送時間を考慮しても 1.4 倍程度高速となった．また CUBLAS と CUBLAS-kernel の性能差はメモリアロケーション，メモリ転送によるものであるが，CUBLAS での性能差が約 21 倍であるのに対し CUDDBLAS では 2.6 倍程度となっており，

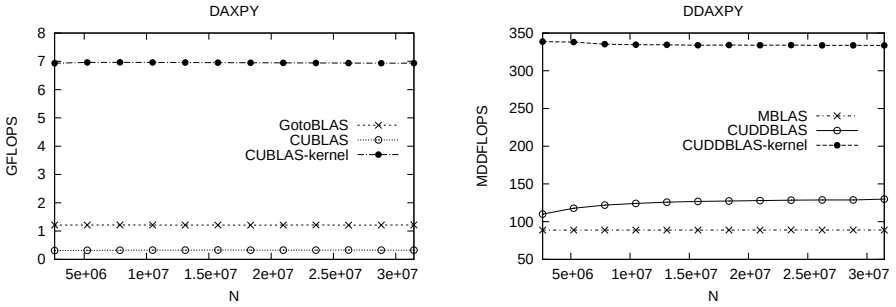

図 9 DAXPY，DDAXPY の性能

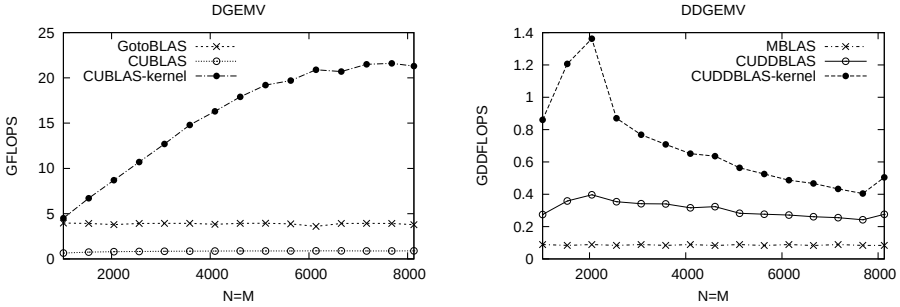

図 10 DGEMV，DDGEMV の性能

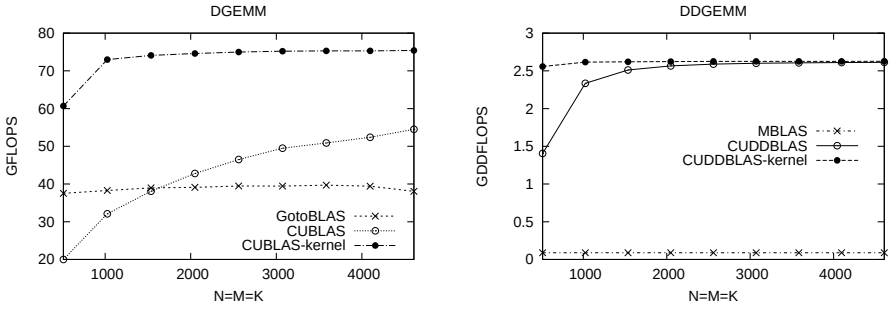

図 11 DGEMM，DDGEMM の性能

© 2009 Information Processing Society of Japan

Vol.2009-ARC-186 No.13
Vol.2009-HPC-123 No.13
2009/12/1

DD 演算によってメモリ律速の傾向が弱まっていることが確認できる．

GEMV では AXPY よりも CPU と GPU の速度差が開き，4 倍精度では MBLAS に対して CUDDBLAS が約 3.3 倍の性能となった．しかし CUBLAS と GotoBLAS の性能差は AXPY より 1 割ほど少なくなっている．GEMV では AXPY よりもメモリのアクセスパタンが複雑となることから，演算密度が高まる一方でメモリ性能がボトルネックとなりやすいのではないかと考えられる．また CUDDBLAS では演算次元数が大きくなるほど性能が低下しているが原因が明らかではなく今後の課題となった．

GEMM では CUDDBLAS の性能は約 2.6GDDFLOPS となり最大で MBLAS の約 30 倍の性能を記録した CUDDBLAS の 4 倍精度積和演算が 23 回の倍精度演算で構成されていることから，実際の倍精度演算回数から算出した性能は約 39GFLOPS であり，GPU の倍精度理論ピーク性能の約 50%である．一方で GotoBLAS が理論ピーク性能比約 92%であるのに対しより演算律速なはずの MBLAS は約 3%であり高速化の余地があると言える．

以上の結果から 4 倍精度演算はアプリケーションの演算律速傾向を高めることが可能であり，GPU のように演算性能に対してメモリ性能が十分ではないハードウェアに対して有効に働くことを示すことができた．またホストメモリからグローバルメモリへの転送速度は実測の結果約 4.5GB/sec であるのに対してグローバルメモリ内の転送速度は約 72GB/sec と高速であり，仮に CPU–GPU 間の転送時間がグローバルメモリと同等になれば，CUDDBLAS-kernel と同等の速度が得られることになる．このことからも GPU–CPU 間の転送速度改善が望まれる．

6. ま と め

本稿では GPU で動作する DD 型の 4 倍精度 BLAS，CUDDBLAS を CUDA により実装し評価を行った．GPU（Tesla C1060）上での DD 型 4 倍精度行列積 DDGEMM では CPU（Core i7 920）上で動作する DD 型 4 倍精度 BLAS の MBLAS と比較し最大で約 30 倍高速になり，GPU の倍精度理論演算性能の約 50%の性能が得られた．一方で演算密度が低い DDGEMV でも MBLAS に対して約 3.3 倍，DDAXPY でも約 1.4 倍高速な性能を示した．演算密度が低い処理ほど GPU カーネル時間のみの性能とメモリ転送時間などを含む CPU 側から呼び出した場合の性能に開きが生まれ，メモリ性能に律速される傾向を示した．これらの結果はデータ転送時間の隠蔽や，高速なテクスチャメモリの活用，アルゴリズムの見直しによりプログラムを高速化できる余地が残るものの，演算量に対してメモリアクセス時間が大きければ，メモリ性能が改善しない限りは根本的な性能改善には繋がらない．

今後の課題として上記のようなプログラムの高速化のほか，精度拡張として Quad-Double 型の 8 倍精度演算[12] の実装や，倍精度演算性能が大幅に向上した次世代 GPU での評価が挙げられる．しかし GPU に限らず多くのハードウェアで演算性能の向上に対してメモリ性能は不足しており，このようなハードウェアに対して多倍長精度演算は有効なアプリケーションであると考えられる．

参 考 文 献

1) NVIDIA : NVIDIA CUDA Programing Guide Version 2.3.1, http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf, (2009).
2) XBLAS - Extra Precise Basic Linear algebra Subroutines, http://www.netlib.org/xblas/.
3) The GNU MP Bignum Library, http://gmplib.org/.
4) The MPACK ; Multiple precision arithmetic BLAS (MBLAS) and LAPACK (MLAPACK), http://mplapack.sourceforge.net/.
5) NVIDIA : CUDA CUBLAS Library, http://developer.download.nvidia.com/compute/cuda/2_1/toolkit/docs/CUBLAS Library 2.1.pdf, (2008).
6) 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).
7) D. H. Bailey. : QD, http://crd.lbl.gov/~dhbailey/mpdist/.
8) 石川正，濱口信行：Bailey アルゴリズムによる多倍長演算の性能評価，情報処理学会研究報告，Vol. 2008–HPC–114, pp. 25–30 (2008).
9) Dekker, T. J. : A Floating–Point Technique for Extending the Available Precision, Numerische Mathematik, Vol.10, pp.224–242 (1971).
10) Knuth, D. E. : The Art of Computer Programming Vol:2 Seminumerical Algorithms, 3rd Edition, Addison-Wesley (1997).
11) 永井貴博，吉田仁，黒田久泰，金田康正：SR11000 モデル J2 における 4 倍精度積和演算の高速化，情報処理学会論文誌：コンピューティングシステム，Vol. 48, pp. 214–222 (2007).
12) Y. Hida, X. S. Li and D. H. Bailey. : Algorithms for Quad-Double Precision Floating Point Arithmetic, Proceedings of the 15th Symposium on Computer Arithmetic, pp. 155–162 (2001).

© 2009 Information Processing Society of Japan