Vol.2010-HPC-127 No.3
2010/10/13

行列乗算カーネルの性能評価

中里　直人[†1]

本研究では, Cypress GPU に最適化された行列乗算 (General Matrix Multiply; GEMM) カーネルの性能評価について報告する. 我々は, Cypress アーキテクチャに最適化された単精度 (SGEMM), 倍精度 (DGEMM) だけでなく, 四倍精度での GEMM(DDGEMM) のカーネルを実装した. 我々の SGEMM と DGEMM カーネルは, それぞれ, 理論性能に対して最大 73%と 87%の演算効率をしめした. 現時点において, 我々の GEMM カーネルは 1 GPU のシステムにおいて世界でも最も高速である. さらに, DDGEMM カーネルの性能は 31 Gflop/s である. DDGEMM カーネルの性能は CPU での四倍精度 GEMM 実装である mpack(0.6.5) と比べて 200 倍以上高速である. 本論文では, GEMM カーネルの実装の詳細を SGEMM カーネルに注目して説明する. これまで GPU でのプログラミングで必須の最適化手法は, 共有メモリの効率のよい利用法が大部分であった. Cypress アーキテクチャではテクスチャキャッシュが有効であり, それにより我々は共有メモリを使わずに高性能な GEMM カーネルを実現している.

On evaluation of matrix-multiply kernels on a GPU

Naohito Nakasato [†1]

We present benchmark results of optimized dense matrix multiplication kernels for Cypress GPU. We write general matrix multiply (GEMM) kernels for single (SP), double (DP) and double-double (DDP) precision. Our SGEMM and DGEMM kernels show 73% and 87% of the theoretical performance of the GPU, respectively. Currently, our SGEMM and DGEMM kernels are fastest with one GPU chip to our knowledge. Furthermore, the performance of our matrix multiply kernel in DDP is 31 Gflop/s. It is more than 200 times faster than the performance results on single core of a recent CPU (with mpack version 0.6.5). We describe our GEMM kernels with main focus on the SGEMM implementation since all GEMM kernels share common programming and optimization techniques. While a conventional wisdom of GPU programming recommends us to heavily use shared memory on GPUs, we show that texture cache is very effective on the Cypress architecture.

1. はじめに

行列乗算は計算科学, 計算機科学や計算工学において, 様々な応用があり, その高速化が最も求められている基本処理の一つである. BLAS Level-3 ルーチンの一つである General Matrix Multiply(GEMM) は, 大規模連立一次方程式の直接解法で必要なだけでなく, BLAS Level-3 の他のルーチンの実装にも利用されている[4),7)]. GEMM は, 行列を $N \times N$ の正方行列としたとき, $O(N^3)$ の演算量を必要とし, そのメモリーのアクセスパターンが一様であるため, GPU のようなメニーコアアクセラレータによる高速化に向いている処理である. 実際, これまで GPU により行列乗算を高速化する試みは多くあった[3),11),12)].

本論文では, もう一つの最新の GPU である Cypress アーキテクチャGPU における GEMM の実装と性能評価について報告する. 我々は通常よく利用されている SGEMM と DGEMM だけでなく, 四倍精度演算 (double-double(DD) エミュレーション法による[2),6)]) での GEMM(DDGEMM) の実装と性能評価についても報告する. 現在のますます問題が大規模化していく傾向を考えると, これから高精度な行列乗算の手法はその重要性が増していくと考えられる.

2. GPU アーキテクチャの概要

本章では, 本研究で利用した Cypress GPU のアーキテクチャの概要を説明し, 他の GPU との比較をおこなう.

2.1 Cypress アーキテクチャ

AMD 社の GPU である Cypress は以下のような特徴を持っている:

- 1 チップに 320 個の演算コア (Thread Processor; TP) が搭載されている.
- TP は 5 個の単精度演算器 (Stream Core; SC) を内蔵する.
- TP は 5-way の VLIW 演算器であり, 最大 5 個の単精度積和演算などが実行可能である.
- 倍精度加算は 2 個の SC, 倍精度乗算は 4 個の SC により実行される.
- TP は 128bit 幅で 1024 語のレジスタファイルを持つ.
- 16 個の TP が一組となり SIMD Engine と呼ばれるユニットを構成する.
- SIMD Engine 内の TP は全て同一の命令流を実行する (SIMT アーキテクチャ).

†1 会津大学
University of Aizu

- 1 チップには 20 個の SIMD Engine が搭載されている.
- SIMD Engine 内の TP が共有するメモリ領域 (Local Data Store; LDS) をもつ.
- SIMD Engine は複数階層のキャッシュ機構を内蔵する.
- 特に, テクスチャ用 1 次キャッシュの帯域幅は 54.5 GB/s である.
- 外部メモリの帯域幅は 153.6 GB/s である.

2.2 Cypress vs. Fermi

表 1 では, どちらも最新の GPU アーキテクチャである, Cypress と Fermi(NVIDIA 社) の両 GPU を利用した演算ボードを比較をした. どちらも 2009 年後半に発表され, 倍精度演算ではほぼ同等の演算性能である. 一方で, 単精度演算性能では Cypress GPU が高速であり, 外部メモリの帯域幅も大きい. 演算器の詳細な構成は, 両者でかなり異なっており, カーネル実行の単位である vector core は, Cypress GPU では 16 個の 5-way VLIW 演算器 (SIMD Engine) である. SIMD Engine の実効的なベクトル長 320 となる[⋆1]. Fermi GPU での vector core は 16+16 個のスカラー演算器 (Streaming Multiprocessor) からなる. Streaming Multiprocessor の実効的なベクトル長は 128 になる. また, 両者とも階層構造のメモリシステムを持つが, その構成は異なるため OpenCL のような統一的なプログラミングシステムを使っても, アーキテクチャ別の最適化が必要となる.

Architecture	Cypress	Fermi
Board Name	Radeon 5870	Tesla C2050
# of SP cores	1600	448
# of DP cores	320	224
# of vector cores	20	14
registers/core	256 KB	128 KB
tex. cache/core	8 KB	12 KB
shared mem./core	32 KB	64 KB
2nd cache	512 KB	768 KB
core clock(GHz)	0.85	1.15
SP peak(Tflop/s)	2.72	1.03
DP peak(Gflop/s)	544	515
memory clock(GHz)	1.2	0.75
memory bus	256	384
memory size(GB)	1	3
memory BW (GB/s)	153.6	144
tex. cache BW(GB/s)	54.5	

表 1 Cypress と Fermi アーキテクチャの比較

3. Cypress GPU での GEMM カーネルの実装

本章では, Cypress GPU での GEMM カーネルの実装の詳細について述べる.

3.1 ブロック化された行列乗算

現代のプロセッサは CPU であっても GPU であっても, 複数階層のメモリシステムを持つため, 行列演算を高速に行うためにはブロック化されたアルゴリズムが必須である. 以下では, ブロック化された $N \times N$ の正方行列の乗算 ($C = AB$) を考える.

乗算する行列を $b \times b$ の小行列にブロック化し, 小行列ごとに行列乗算を計算する場合, 行列 A と行列 B から読み込みが必要なワード数は $2bN$ であり, 必要な浮動小数点演算数は $2Nb^2$ である. よって, 1 演算あたりに必要なワード数は $W = (2bN)/(2Nb^2) = 1/b$ ワード/flop となる. つまり, ブロック化されていない行列乗算 ($b = 1$ の場合) は, 1 演算あたり 1 ワードのデータの読み込みが必要である.

W と演算速度 F(flop/s) と 1 ワードあたりの大きさ S(byte) を組み合わせて, ブロック化された行列乗算に必要なメモリ帯域 (B_{GEMM}) は以下の式で表される.

$$B_{\mathrm{GEMM}} = W \times F \times S \ \ \mathrm{byte/s} \tag{1}$$

この式から, 倍精度 ($S = 8$) 演算性能 $F = 544$ Gflop/s の Cypress の場合に必要なメモリ帯域は $B_{\mathrm{DGEMM}} = 0.544(\mathrm{Tflop/s}) \times 8/b = 4.352/b$ TB/s となる. つまり, ブロック化されていない行列乗算 ($b = 1$) には 4.4 TB/s 以上のメモリ帯域が必要になる. 一方で, Cypres GPU ボードのメモリ帯域はたかだが ~ 150 GB/s でしかない.

例えば $b = 4$ とした場合 $B_{\mathrm{DGEMM}} = 1.088$ TB/s となり, 必要なメモリ帯域が大幅に減少する. 表 1 にあるように, Cypress GPU のテクスチャキャッシュのメモリ帯域は SIMD Engine あたり 54.5 GB/s あるため, チップ全体での 1 次キャッシュの帯域はちょうど 54.4 (GB/s) $\times 20 = 1.088$ TB/s となり釣り合う. 我々は DGEMM の実装においては $b = 4$ を採用した. SGEMM と DDGEMM においても同様の考察から, 最適な b の大きさが決定できる. SGEMM では $b = 8$, DDGEMM では $b = 2$ を採用した.

3.2 GEMM のためメモリ割り当て

ブロック化された GEMM カーネルを GPU に実装する場合, ブロック化された行列 A, B, C のデータをどのメモリ領域に保存するかによって性能が大きく異なる. 例えば Volkov ら[12] は, Geforce アーキテクチャの GPU では, 行列 A, C をレジスタに割り当て, 行列 B を共

⋆1 4 clock 同一の命令を発行するため. Fermi も同様

有メモリに割り当てるのが最適であると示した. 結果として, 彼らは SGEMM カーネルで GPU の理論性能の約 60%の性能を得た. 最新の Fermmi アーキテクチャでも似た手法がとられており[11], Nath らは行列 C をレジスタに割り当て, 行列 A, B を共有メモリに割り当てることで, DGEMM カーネルで GPU の理論性能の約 60%の性能を得た.

本研究では, 過去の研究では詳しく調べられていない Cypress GPU を使っているため, 既存の最適な割り当て方法がそのまま当てはまるとは限らない. 実際に, 既に述べたように, Cypress GPU では読み込み専用のテクスチャキャッシュが高速なため, 我々はその有効利用を検討した. それは, テクスチャキャッシュには以下のような利点があるためである. 共有メモリ (Cypress アーキテクチャでは LDS) を利用するためには, LDS でのメモリを割り当てを自前で管理する必要があり, さらに共有メモリ内のアドレスの計算も明示的に行う必要があるのに対して, キャッシュメモリを利用する場合にはそのような手間が必要ない. また, LDS を効率よく利用するためには, バンク衝突の回避や, coalesced なメモリアクセスとなるよう配慮が必要である. つまり, LDS をソフトウエアキャッシュとして利用するためには, 様々な明示的な処理が増えるのに対して, テクスチャキャッシュはハードウエアにより実現されているためそのような手間がいらない. よって, 本研究では行列 C はレジスタに割り当て, 行列 A, B はストリームメモリとし, テクスチャキャッシュを介して読み込むこととした (図 1 参照). 実際, 以下に示すように, この割り当て方法で高性能な結果を得ている.

3.3 SGEMM の実装の詳細

ここでは GEMM カーネルの実装の例として SGEMM の場合の実装の詳細について説明する. 本研究では, 行列は行優先形式 (C 言語形式) で保存されているとする. この時に, 行列の各要素のインデックスを j, i によって示す. $A(j, i)$ は行列 A の 1 要素 (ワード) を示す. またブロック化された小行列はインデックス J, I によって示すことする. $A[J, I]$ は行列 A のブロック化された小行列を示す. I, J とも 0 から $N/b-1$ の範囲をとる. また, 以下では $a : b$ という表記によって, a から b までの範囲のインデックスを示すこととする. つまり, $A(j, 0 : N-1)$ は行列 A の j 行目 (N ワード) を表す.

Cypress GPU にて行列乗算 ($C = AB$) を実行する際には, 複数のスレッドが異なる小行列 $C[I, J]$ の計算を担当する. $C[I, J]$ を計算するためには, 行列 A から $A(bJ : bJ+7, 0 : N-1)$ の帯 (単精度で $8N$ ワード) を, 行列 B から $A(0 : N-1, bI : bI+7)(8N$ ワード) の帯を読みだす必要がある. 図 1 に, 疑似コードで記述されたカーネルをしめす. カーネルは以下の三つの部分からなる：(1) 初期化, (2) Rank-1 更新のループ, (3) 結果の書き込み.

初期化の部分では $C[I, J]$ に対応するレジスタを 0 で初期化する. ループでは, 各ステップで (k 番目のステップとする), 行列 A から $(bJ : bJ+7, k)$ の列 (8 ワード) を読み込み, 行列 B から $(k, bI : bI+7)$ の行 (8 ワード) を読み込み, それらの外積を計算し $C[I, J]$ を Rank-1 更新する. 最後に $C[I, J]$ の結果に α, β のスカラー変数をかけて結果を GPU メモリに書き込む. 図 2 に, この場合のメモリの読み込みパターンを示す. これは, 入力行列 A, B ともに転置の指定がない場合に対応する. 以下, この場合を NN カーネルと呼ぶ[⋆1]. 入力行列 A に転置の指定があった場合[⋆2]は, 行列 A に対するメモリアクセスが図 3 のようになる. 以下, この場合を TN カーネルと呼ぶ. Cypress GPU ではメモリアクセスの単位が 128 bit(単精度で 4 ワード) であるため, 常に行優先のアクセスとなる TN カーネルのほうが効率が良くなると予想される.

3.4 DGEMM および DDGEMM の実装

DGEMM および DDGEMM は, SGEMM カーネルと同様の手法により実装した. ただし, 既に述べたようにブロック行列のサイズ b は, SGEMM では $b = 8$ であったのに対し, DGEMM では $b = 4$, DDGEMM では $b = 2$ とした. いずれの場合にも, Cypress GPU ではメモリアクセスの単位が 128 bit であり, これは単精度で 4 ワード, 倍精度で 2 ワード, 四倍精度で 1 ワードに相当するため, b がそれぞれの場合のアクセス単位ワード数の倍数となるようにしている.

DDGEMM については, 中里ら[9),15)] による GPU 用の四倍精度ライブラリを利用している. Cypress GPU では倍精度における fused-multiply-add (FMA) 命令が利用できるため, 永井ら[13)] の提案に従って, FMA 命令を使うことで四倍精度の乗算を高速化した. 具体的には, Cypress GPU において四倍精度加算は 21 命令, FMA 命令なしの乗算は 25 命令必要であった. FMA 命令により, 乗算は 8 命令で実行できるようになる. 行列乗算では, 加算と乗算が同数必要であるので, 四倍精度の行列乗算における平均命令数 (倍精度換算) は, FMA 命令なしの場合は $(21+25)/2 = 23$ 命令であるのに対して, FMA 命令ありの場合は $(21+8)/2 = 14.5$ 命令である. 平均命令数から, 四倍精度行列乗算の理論性能を換算すると, FMA 命令なしの場合は 544 (Gflop/s) /23 = 23.7 Gflop/s であり, FMA 命令ありの場合は 544 (Gflop/s) /14.5 = 37.5 Gflop/s となる.

⋆1 GEMM のパラメータ TRANSA と TRANSB がどちらも N
⋆2 GEMM のパラメータ TRANSA が T で, TRANSB が N

```
// 行列 A と B はストリームメモリとして宣言
// 行列 C はグローバルメモリとして宣言
float c[8][8], a[8], b[8]; // レジスタ変数
// 小行列を 0 に初期化
c[0:7][0:7] = 0.0
for k = 0 to N-1
   // 行列 A から 1 列 (8 ワード) 読み込み
  load a[0:3] <- A(J:J+3, k)
  laod a[4:7] <- A(J+4:J+7, k)
   // 行列 B から 1 行 (8 ワード) 読み込み
  load b[0:3] <- B(k, I:I+3)
  laod b[4:7] <- B(k, I+4:I+7)

  // 8x8 の Rank-1 更新
  c[0][0:3] += a[0]*b[0:3]; // 8 flops
  c[0][4:7] += a[0]*b[4:7];
  c[1][0:3] += a[1]*b[0:3];
  c[1][4:7] += a[1]*b[4:7];
  ...
  c[7][0:3] += a[7]*b[0:3];
  c[7][4:7] += a[7]*b[4:7];
end
Merge c[][] with C[J,I]
```

図 1 SGEMM カーネルの疑似コード

4. GEMM カーネルの性能評価

表 2 に, SGEMM(TN のみ), DGEMM (TN と NN), DDGEMM(TN のみ. ただし, FMA ありとなしの場合) の性能をまとめた. ここでの性能はカーネルの実行時間のみを計測したものである. 表の 4,5 行目には, カーネルのメインループにおいて, TP の VLIW 命令のスロットがどれくらい利用されているかを示した. SGEMM カーネルの場合, メインループの

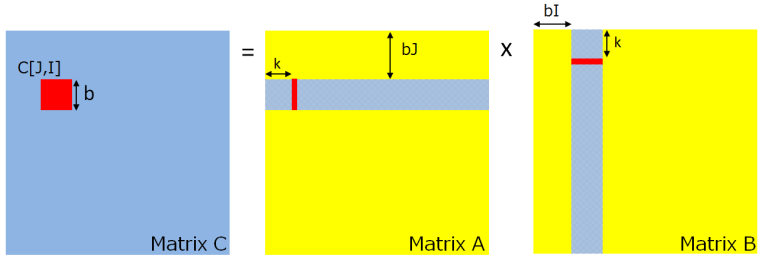

図 2 入力行列 A, B ともに転置の指定がない場合の GEMM カーネルのメモリアクセスパターン. 個々のスレッドは $C[J, I]$ の計算を担当する. k 番目のステップにおいて, 行列 A, B から赤色で示された部分を読み出し Rank-1 更新を計算する.

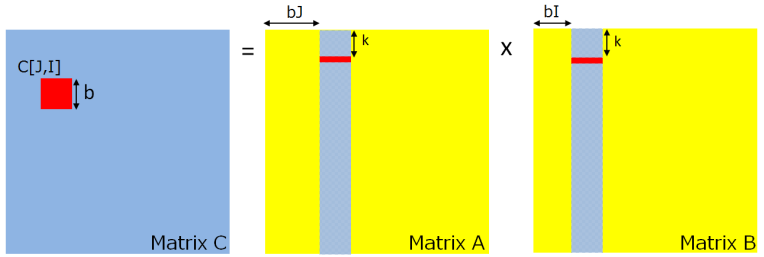

図 3 入力行列 A が転置, B は転置の指定がない場合の GEMM カーネルのメモリアクセスパターン.

全命令のうち 26%では 4 スロットが埋まっており, さらに 58%では 5 スロットが埋まっている. 残りの部分 (16%) では, 1, 2, または 3 スロットしか利用されていない. この部分は, アドレスの計算や, ループの条件分岐, そしてループ変数の更新などに必要である. 同様に DGEMM と DDGEMM カーネルでは, 最大 4 スロットまでしか利用されないため, 90%以上の命令で GPU は最大効率で演算をおこなっていることになる. 以下, DGEMM の場合のより詳細な性能評価と他のライブラリとの比較をおこなう.

4.1 DGEMM カーネルの演算性能

図 4 に, 行列の大きさ N の関数として DGEMM カーネルの演算性能のグラフを示す. この図では DGEMM NN カーネルと DGEMM TN カーネル, そして MAGMA BLAS 0.3[11] の演算性能を比較している. 我々の DGEMM TN カーネルの演算速度 (400 - 470 Gflop/s) は, 現時点で GPU を 1 チップ使った場合で最も高速である. 一方で, NN カーネルと TN カー

	SGEMM TN	DGEMM TN	DGEMM NN	DDGEMM TN (FMA)	DDGEMM TN (no FMA)
Pmax	2014	472	359	31	23
Nmax	4352	1664	3712	1408	768
# of reg.	25	25	25	18	29
4 slots(%)	25.8	94.1	94.1	90.9	90.9
5 slots(%)	58.1	0	0	0	0

表 2 様々な GEMM カーネルの性能. Pmax と Nmax は, それぞれ性能が最大になる場合の演算性能 (Gflop/s) とその時の N の大きさを示す. 3 番目の行は, それぞれのカーネルで必要なレジスタの数を示す. また, 4,5 番目の行は, カーネルのメインループにおいて, VLIW 命令のスロットが 4 スロット利用された割合, 5 スロット利用された割合を示す. なお, 倍精度演算では最大 4 スロットまでしか利用されない.

ネルの間には, 大きな演算性能の差がある. TN カーネルは理論演算性能の 74% - 87%の演算効率であった. NN カーネルは理論演算性能の 58% - 66%の演算効率であった. NN カーネルで理論演算性能が落ちる理由は, Cypress GPU では行優先のメモリアクセスの方が効率が良いためである. また, テクスチャキャッシュを介したメモリアクセスは倍精度の場合 2 ワード単位のため, NN カーネルのメインループでは, 行列 A から $A(bJ, k : k+1), A(bJ+1, k : k+1), A(bJ+2, k : k+1), A(bJ+3, k : k+1)$ の 8 ワードを読み込み, 行列 B から $B(k, bI : BI+3)$(4 ワード) を読み込み, $A(bJ, k), A(bJ+1, k), A(bJ+2, k), A(bJ+3, k)$ と $B(k, bI : BI+3)$ により Rank-1 更新をする. 次に, 行列 B から $B(k+1, bI : BI+3)$(4 ワード) を読み込み, $A(bJ, k+1), A(bJ+1, k+1), A(bJ+2, k+1), A(bJ+3, k+1)$ と $B(k+1, bI : BI+3)$ により Rank-1 更新をする. よって, NN カーネルと比べると余分なデータの読み込みが必要であり, 結果としてキャッシュヒット率が下がるため性能が低下すると推測される.

Fermi GPU における MAGMA BLAS 0.3 の演算性能は約 300 Gflop/s であった[11]. これは理論性能の約 60%の効率に相当する. 我々の DGEMM カーネルは, 多くの場合により高効率 (58% - 87%) で動作している. これは, Cypress GPU のテクスチャキャッシュが GEMM カーネルの実装に最適であることを示す. ただし, MAGMA BLAS 0.3 の結果と比べると, 我々の DGEMM カーネルは, N 対する依存性があり, また TN カーネルでは $N \geq 2048$ の場合に若干性能が低下している. この原因として考えられるのは, メモリ読み込み時のバンク衝突であると推測される. この点を明らかにしバンク衝突を回避する方法の研究は, 今後の課題のひとつである.

4.2 メモリ転送を含んだ性能評価

図 4 で, "I/O" の付されている DGEMM TN と NN カーネルの結果と, ACML-GPU 1.1 の結果は, ホストと GPU のメモリ間のデータ転送時間を含んだ演算性能をしめす. この演算性能が, 実効的な DGEMM の演算性能になる. GEMM の計算には, $O(N^2)$ のデータをホストから GPU に送り, GPU で $O(N^3)$ の計算を行う必要があるため, N が小さい場合 ACML-GPU 1.1 の結果に示されるようにデータ転送時間がボトルネックとなる. 以下, デー

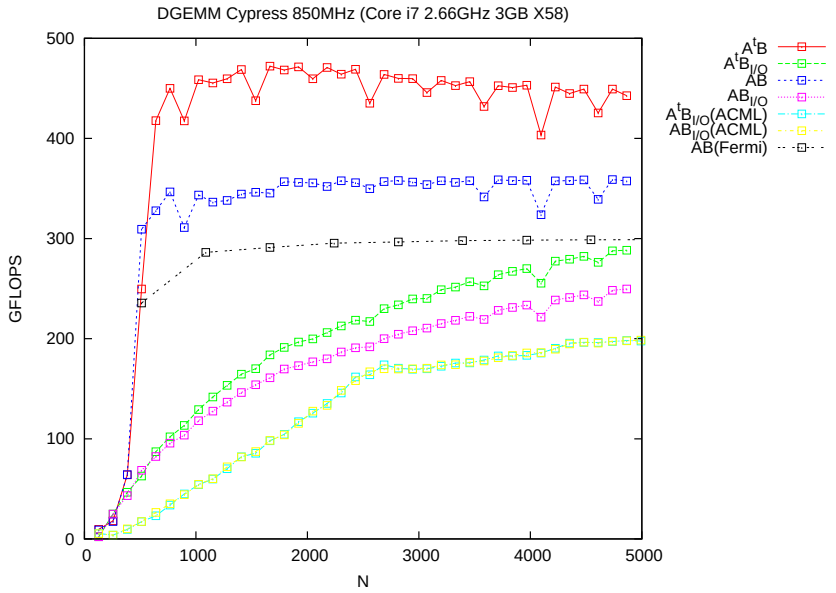

図 4 我々が実装した DGEMM カーネル (TN と NN) と, 他のカーネル (Cypress GPU での ACML-GPU 1.1 と Fermi での MAGMA BLAS 0.3[11]) との演算速度の比較. 我々が実装した DGEMM の結果と ACML-GPU 1.1 の結果のうち, "I/O" の付されている結果は, ホストと GPU の間のデータ転送時間を含んだ性能を示す.

タ転送時間を含めた演算性能の予測モデルを構築する.

データ転送の最適化しない場合, DGEMM カーネルの実行時間 (T_{DGEMM}) は以下のように表される.

$$T_{\mathrm{DGEMM}} = T_{\mathrm{comm}} + T_{\mathrm{kernel}}, \tag{2}$$

ここで, T_{comm} はデータ転送に必要な時間をしめし, T_{kernel} はカーネルの実行時間をしめす. DGEMM($C = \alpha AB + \beta C$) の計算では, 3 個の入力行列を GPU メモリに転送し, 1 個の結果行列を得るため, データの総転送量は $4 \times 8 \times N^2 = 32N^2$ バイトであり, T_{comm} は,

$$T_{\mathrm{comm}} = \frac{32N^2}{B_{\mathrm{PCIe}}}, \tag{3}$$

のように表せられる. ここで, B_{PCIe} はホストメモリと GPU メモリ間のデータ転送速度 (byte/s) を表す. T_{kernel} は,

$$T_{\mathrm{kernel}} = \frac{2N^3}{F}, \tag{4}$$

となる. ここで F は浮動小数点の演算速度 (flop/s) を表す. 以上の定義から, データ転送を考慮した DGEMM の実効性能は,

$$F_{\mathrm{DGEMM}} = 2N^3 / T_{\mathrm{DGEMM}} \tag{5}$$

と評価できる.

図 5 に, 我々の性能モデル (式 (5); 緑の点線) と, DGEMM TN カーネルで得られた演算性能 (赤い四角) の比較をしめす. ここで性能の測定結果から, $\mathrm{B_{PCIe}} = 3$ GB/s and $F = 450$ Gflop/s とした. 我々のモデルと実際の演算性能はよく一致している. 一方で, データ転送時間を含まない演算性能は > 400 Gflop/s(図 4 参照) であるのに, データ転送時間を含んだ実効性能が半分程度の低下している. これは, ホストメモリと GPU メモリ間のデータ転送速度 ($\mathrm{B_{PCIe}}$) が, PCI Express x16 gen.2 の転送速度である 8 GB/s にはるかに及ばず, たかだか 3 GB/s であるためである. 実際には, DMA 転送によるデータ転送を繰り返すベンチマークでは約 6 GB/s の性能が得られているので, 何らかのソフトウエア的手法によりデータ転送速度を高速化する余地があると考えられる. 図 5 の青い点線は, 仮に $\mathrm{B_{PCIe}} = 10$ GB/s が達成できた場合の演算性能である. この場合には, $N \geq 2048$ の場合におおよそ 300 Gflop/s 以上の性能を得ることができる.

現在の我々の実装では, データ転送と GPU でのカーネル実行の非同期実行をおこなっていないが, 我々のカーネルを実用的に利用する際には, このような最適化が必須と考えられる. この場合, 重要なのは $T_{\mathrm{comm}} = T_{\mathrm{kernel}}$ となる N_{balance} の大きさである. 我々のモデル

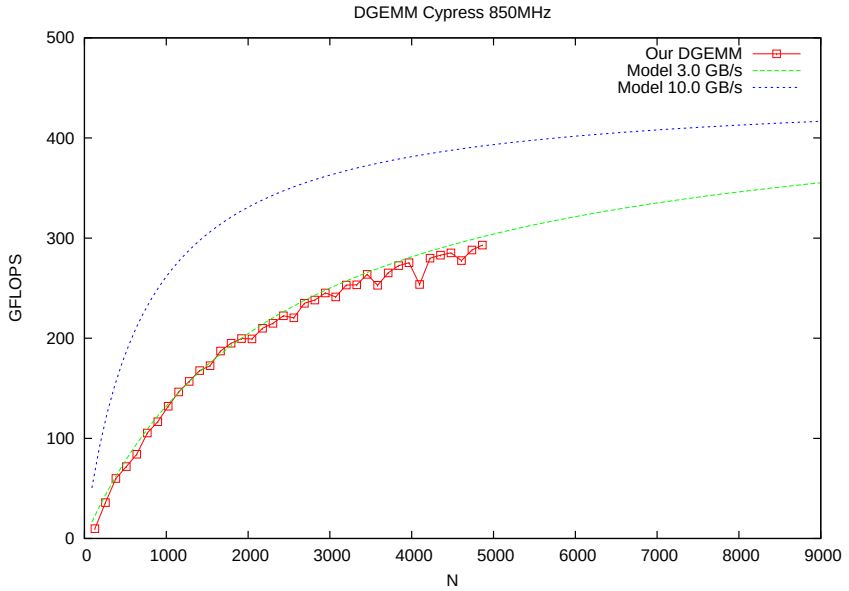

図 5 ホストと GPU メモリ間のデータ転送時間を含んだ場合の DGEMM TN kernel の演算性能 (赤い四角) と, 我々の性能モデル (式 (5); 緑の点線) の比較. 青い点線は, データ転送速度が高速 ($\mathrm{B_{PCIe}} = 10$ GB/s) な場合の予測性能をしめす.

から, DGEMM TN カーネルの場合,

$$N_{\mathrm{balance}} = \frac{16F}{B} \tag{6}$$

であり, $\mathrm{B_{PCIe}} = 3$ GB/s and $F = 450$ Gflop/s の場合, $N_{\mathrm{balance}} = 2400$ となる. よって, データ転送とカーネル実行の overlap による通信の隠蔽を実装する際には, 一回の DGEMM カーネル呼び出しの単位として $N > 2400$ となるブロックサイズを採用する必要があるだろう. SGEMM と DDGEMM についても同様の性能モデルにより, 最適化のためのブロックサイズが決定できる.

5. 関 連 研 究

Volkov ら[12] は, GeForce GPU に最適化された GEMM の実装とそれを使った LU 分解

(単精度) の性能の報告をおこなっている．彼らの最適化された SGEMM カーネルは, 理論性能の約 60%の性能であった．一方で, そのころに利用可能であった CUBLAS バージョン 1.1(NVIDIA 社) の性能は, 同じ GPU で理論性能の約 40%であった．Li ら[8] は, Volkov らの GEMM 実装に基づき, GeForce GPU 用の自動最適化の手法について報告している．彼らの自動最適化で得られた SGEMM と DGEMM カーネルは, その時に利用可能であった CUBLAS バージョン 2.0 より高速であった (なお, CUBLAS バージョン 2.0 は Volkov らの結果を取り込んでいる)．さらに, Jang[5] は, Li らとは独立に OpenCL による GEMM のための自動最適化フレームワークを作成した．また, Nath ら[11] は, 最新の Fermi GPU に GEMM を実装し, Volkov らと同様に, DGEMM カーネルで理論性能の約 60%の性能を得ている．

我々の GEMM カーネルは, 以上の関連研究と比べて, より高速であり, 理論性能に対する演算性能比も高い．具体的には, 我々の SGEMM TN カーネルは, N の大きさに依存して理論性能の 55 - 73%の演算性能比をしめす．また, DGEMM TN カーネルは 74 - 87%, DGEMM NN カーネルは 58 - 66%の演算性能比をしめす．同一の GPU における, ACML GPU 1.1 の演算性能と比べても, 我々の DGEMM TN, TN カーネルは大幅に高速である (図 4 の下部参照)．我々の実装と ACML GPU 1.1 の実装の大きな違いは, 採用したブロックサイズ b の違いである．

高精度な行列計算ライブラリである XBLAS[1] および mpack[10),14)] は, CPU 用の高精度 BLAS/LAPACK ルーチンを提供している．また, 椋木ら[16] は, GeForce GPU による四倍精度 BLAS ルーチンの実装と性能について報告している．本研究で, 我々は DDGEMM カーネルが非常に高速であり, 効率も良いことを示した．実際, FMA 命令を利用した時の DDGEMM カーネルの演算性能は 31 Gflop/s であり, mpack の四倍精度 GEMM ルーチンを single core で実行したときの演算性能 144 Mflop/s[14] と比べると, 200 倍以上高速である．椋木ら[16] の得た性能 2.6 Gflop/s と比べても 10 倍以上高速である．ただし, mpack の演算性能にはまだ高速化の余地があり, 椋木らの利用した GPU の倍精度演算性能は, Cypress GPU の約 7 分の 1 である．我々の結果と椋木らの結果から, 四倍精度演算は非常に GPU に向いている処理であることがわかる．

6. ま と め

本研究では, Cypress GPU 向けの GEMM カーネルの実装と性能評価について報告した．Cypress アーキテクチャ向けに特有の最適化手法として, 我々は読み出し専用のテクスチャキャッシュを活用した．これにより, SGEMM ではブロックサイズ $b = 8$, DGEMM では $b = 4$, DDGEMM では $b = 2$ で非常に高性能な結果を得た．我々の GEMM カーネルは, 現時点で最も高速なだけでなく, 既存の研究結果と比べて理論性能に対する演算効率も高い．これまで, GPU で高速なカーネルの実装するには, 共有メモリをソフトウエアキャッシュとして利用することが必須と考えられていたが, ソフトウエアキャッシュとしての取り扱いのためには様々な手間が増え, GPU プログラミングの困難点となっていた．我々はハードウエアで制御されるテクスチャキャッシュが, GEMM カーネルの実装では非常に有効であることをしめした．テクスチャキャッシュを使うことで, 共有メモリを利用する際に必要な明示的なアドレス演算などがいらず, 我々の GEMM カーネルは非常に簡潔に記述することができている．

今後の研究としては, GEMM カーネルではテクスチャキャッシュが有効であるのは確かであるが, 共有メモリを有効に使うことでより高速な GEMM カーネルを実現できるかの検討が考えられる．また, DGEMM カーネルの演算性能が $N \geq 2048$ の時に徐々に低下しているため, これを回避するような特殊な最適化手法を調べることも検討している．さらに, ホストメモリと GPU との間のデータ転送を考慮した実効的な演算性能は, PCI-Express バスの性能がネックとなり, DGEMM においてたかだか 200 Gflop/s となる．我々の DGEMM カーネルの実際の問題に適用する際には, データ転送とカーネル実行のオーバーラップが必要となるため, この部分の最適化手法の研究をおこなう予定である．

参 考 文 献

1) : XBLAS - Extra Precise Basic Linear Algebra Subroutines: http://www.netlib.org/xblas/.
2) Dekker, T.: A Floating-Point Technique for Extending the Available Precision, *Numerische Mathematik*, Vol.18, pp.224–242 (1971).
3) Fatahalian, K., Sugerman, J. and Hanrahan, P.: Understanding the efficiency of GPU algorithms for matrix-matrix multiplication, *HWWS '04: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics Hardware*, Newyork, NY, USA, pp.133–137 (2004).
4) Igual, F., Quintana-Ortí, G. and van de Geijn, R.: Level-3 BLAS on a GPU: Picking the Low Hanging Fruit, *FLAME Working Note #37. Universidad Jaume I, Depto. de Ingenieria y Ciencia de Computadores. Technical Report*, Vol.DICC 2009-04-01 (2009).
5) Jang, C.: GATLAS GPU Automatically Tuned Linear Algebra Software: http://golem5.org/gatlas/.

6) Knuth, D.: *The Art of Computer Programming vol.2 Seminumerical Algorithms*, Addison Wesley, Reading, Massachusetts, first edition (1998).
7) Kågström, B. and Van Loan, C.: GEMM-Based Level-3 BLAS, *Technical Report, Department of Computer Science, Cornell University*, Vol.CTC91TR47 (1989).
8) Li, Y., Dongarra, J. and Tomov, S.: A Note on Auto-tuning GEMM for GPUs, *Proceedings of ICCS'09*, Baton Rouge, LA, USA (2009).
9) Nakasato, N. and Makino, J.: A Compiler for High Performance Computing With Many-Core Accelerators, *IEEE International Conference on Cluster Computing and Workshops*, pp.1–9 (2009).
10) Nakata, M.: The MPACK (MBLAS/MLAPACK); a multiple precision arithmetic version of BLAS and LAPACK: http://mplapack.sourceforge.net/ (2010).
11) Nath, R., Tomov, S. and Dongarra, J.: An Improved MAGMA GEMM for Fermi GPUs, *Universiyt of Tennessee Computer Science Technical Report*, Vol. UT-CS-10-655 (also LAPACK working note 227) (2010).
12) Volkov, V. and Demmel, J.: Benchmarking GPUs to Tune Dense Linear Algebra, *SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, Piscataway, NJ, USA, pp.1–11 (2008).
13) 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, 情報処理学会誌:コンピューティングシステム, Vol.48, pp.214–222 (2007).
14) 中田真秀：MPACK(MBLAS/MLAPACK) 高精度 BLAS/LAPCK ライブラリの作成, 計算工学講演会論文集 Vol. 15 (2010).
15) 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 (2009-HPC-121) (2009).
16) 椋木大地, 高橋大介：GPU による四倍精度 BLAS の実装と評価, 情報処理学会研究報告 (2009-HPC-123/2009-ARC-186) (2009).