Vol.2012-HPC-135 No.39
2012/8/3

OpenCLによる行列乗算カーネル実装と性能評価

松本 和也[1,a)]　中里 直人[1]　Stanislav Sedukhin[1]

概要：本稿では，OpenCL により実装した行列乗算 (GEMM: General Matrix Multiply) カーネルの性能評価の結果を報告する．性能測定は二種類の GPU(Tahiti と Cayman) と二種類の CPU(Sandy Bridge と Bulldozer) で行った．我々が開発した GEMM コードジェネレータを用いて多数の GEMM カーネルを生成し，その中から各プロセッサごとに最速なカーネルを選択することで，高速な倍精度行列乗算 (DGEMM) と単精度行列乗算 (DGEMM) カーネルを実装することを可能にした．GPU においては，我々の DGEMM 実装はピーク性能比の 80%を超える高い性能を達成した．

1. はじめに

行列-行列乗算は計算科学や計算機科学において様々な応用があり，高速化が求められている基本演算の一つである．また，行列乗算は Level-3 BLAS (Basic Linear Algebra Subroutines)[1] の一つであり，GEMM (General Matrix Multiply) とも呼ばれる．GEMM は LAPACK (Linear Algebra PACKage)[2] や他の Level-3 BLAS ルーチン [3] の実装においても利用できる．GEMM アルゴリズムの演算密度 (Flops / Byte) は $O(n)$ と高く，そのアルゴリズムの規則性の高さもあり，GEMM は GPU のようなアクセラレータによる高速化に向いている．そのため行列乗算を高速化する試みは多々ある [4], [5], [6], [7].

OpenCL (Open Computing Language) は並列計算のためのクロスプラットフォームなフレームワークであり，OpenCL に対応しているあらゆる GPU や CPU 間で OpenCL によるプログラムの移植性はある．しかし，OpenCL 言語で書かれたカーネルの性能移植性は通常はなく，高速な OpenCL カーネルを実現するためには各々のプロセッサに応じた性能チューニングが必要がある．本研究では，我々が開発した [8]GEMM の OpenCL カーネルコードを生成するコードジェネレータを用いて，各種 GPU や CPU それぞれで高速な行列乗算カーネルの実装を実現する．本稿では，次の二種の GPU と二種の CPU で性能評価を行った結果を報告する．

(1) AMD Tahiti GPU (Radeon HD 7970)
(2) AMD Cayman GPU (Radeon HD 6970)
(3) Intel Sandy Bridge CPU (Core i7 3960X)
(4) AMD Bulldozer CPU (FX-8150)

[1] 会津大学
The University of Aizu, Fukushima 965-8580, Japan
[a)] d8121101@u-aizu.ac.jp

2. 行列乗算カーネル

BLAS[1] において，GEMM は $C \leftarrow \alpha\mathrm{op}(A)\mathrm{op}(B) + \beta C$ と定義される．ここで，α と β はスカラー値であり，A, B と C はそれぞれ $m \times k$ 行列，$k \times n$ 行列，$m \times n$ 行列である．そして，$\mathrm{op}(X)$ は非転置 (non-transposed) 行列 X または転置 (tranpososed) 行列 X^T をとり，実数型の GEMM は四種類の行列乗算 ($C \leftarrow \alpha AB + \beta C$ (以後 NN と呼ぶ), $C \leftarrow \alpha AB^T + \beta C$ (NT), $C \leftarrow \alpha A^T B + \beta C$ (TN), $C \leftarrow \alpha A^T B^T + \beta C$ (TT)) から成る．

実装したコードジェネレータはパラメタの組み合わせを与えられたとき，それに対応した OpenCL で書かれた行列乗算を行うカーネルコードを生成する．パラメタにはブロックサイズに関わるものと，行列データの読み込み方法に関するものに大別される．本稿では高速な GEMM カーネル生成に寄与するパラメタについてのみ記述し，コードジェネレータに設定することができる全てのパラメタについては説明しない．

2.1 行列乗算アルゴリズムのブロック化

単純な行列乗算アルゴリズムは三重ループから成るが，そのアルゴリズムを単純に OpenCL で実装しただけでは高速化は達成できない．行列乗算アルゴリズムをブロック化しプロセッサの階層化されたメモリを効率的に利用することは，高性能な実装を実現するには必須である．本研究におけるコードジェネレータは，そのメモリの階層構造と OpenCL の実行モデルに適した二段階にブロック化した行列乗算アルゴリズムを基にしたカーネルを生成する．

本研究では，OpenCL の仕様に合うようにプロセッサの

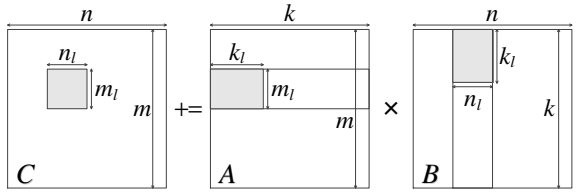


図 1 行列乗算 $(AB+C)$ アルゴリズムをブロック化因数 m_l, n_l, k_l でブロック分割した際のイメージ ([8] の図 1 より転載)

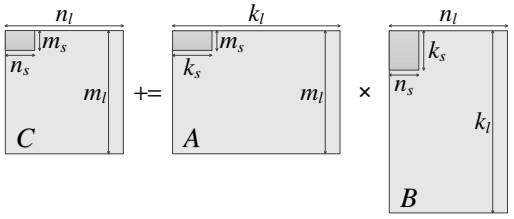


図 2 ブロック同士の乗算をより小さなブロック化因数 m_s, n_s, k_s でさらにブロック分割した際のイメージ ([8] の図 2 より転載)

メモリシステムを三層に抽象化している．その三層とはオフチップメモリ (グローバルメモリ) とオンチップメモリ (キャッシュやローカルメモリ) とプライベートメモリ (レジスタファイル) である．大雑把に各メモリ層の特徴について記述する．オフチップメモリは最大の容量を持つメモリだがメモリバンド幅は最も狭い．プライベートメモリのメモリ容量は最小だがメモリバンド幅は最も広い．オンチップメモリはオフチップメモリとプライベートメモリの中間的な特徴を持つ．

OpenCL の実行モデルはその三層のメモリシステムに沿う．OpenCL のカーネルコードは C 言語を並列計算用に拡張した OpenCL 独自の言語により記述される．OpenCL において最小の処理単位は work-item と呼ばれ，複数の work-item は work-group を構成する．各 work-item のプライベートメモリは他の work-item からは不可視であるが，同一 work-group 内の全ての work-item は並列的に同じ命令を実行する．また各 work-group の work-item はローカルメモリ (AMD の GPU では LDS (local data store) と呼ばれる) 内でデータを共有することができる．グローバルメモリは全ての work-item からデータの読み書きを行うことができるが，同メモリ上で work-group 間のデータの共有はできない．

二段階のブロック化をどのように行うかを指定するために，行列の m, n, k の各次元ごとに大小異なる二種類のブロック化因数をコードジェネレータに指定できるようにした．本稿では，大きい方のブロック化因数を m_l, n_l, k_l と記し，小さい方のブロック化因数を m_s, n_s, k_s と記述する．大きい方のブロック化因数 m_l, n_l, k_l により，図 1 に描かれるように行列 A, B, C はブロック分割される．行列 C の各 $m_l \times n_l$ ブロックに関する処理は，単一の work-group に割り当てられる．処理が割り当てられた work-group は行列 A の $m_l \times k$ ストライプと行列 B の $k \times n_l$ ストライプの積を計算し行列 C の $m_l \times n_l$ ブロックに加算する．

我々の行列乗算アルゴリズムでは，そのストライプとストライプの乗算はその最外部のループを k/k_l 回繰り返す (m, n, k はそれぞれ m_l, n_l, k_l の倍数の場合のみを考える)．毎回の繰り返しにおいておいて，work-group は $m_l \times k_l$ ブロックと $k_l \times n_l$ ブロックを掛けた積を $m_l \times n_l$ ブロックに足し合わせる．そのブロック同士の乗算において，各ブロックは小さい方のブロック化因数 m_s, n_s, k_s によりさらにブロック分割される．図 2 にその乗算のイメージを示す．行列 C に関する $m_l \times n_l$ ブロックの $m_s \times n_s$ サブブロックの処理は，単一の work-item に割り当てられる．

2.2 行列データの読み込み方法の最適化

OpenCL では work-group 内のデータを共有するためにローカルメモリを使用することができる．特に GPU においては，ローカルメモリへのアクセス速度は L1 キャッシュへの速度より二倍速い．しかし，ローカルメモリを利用した場合にはデータを同期させる必要があり，その同期にかかるオーバヘッドが発生するために，必ずしも共有メモリを使用した方が良い訳ではない．そのため，入力行列 A, B それぞれのデータをローカルメモリで共有するかどうかを，コードジェネレータの入力パラメタとして指定できるようにした．

OpenCL ではベクタ変数を用いることができる．例えば，`double2` は 2 つの倍精度浮動小数点数から成るベクタ変数を表す．ベクタ変数の幅（変数当たりの浮動小数点数の数）を変化させることは性能に影響を与えるため，パラメタの一つとした．

行列データを行優先で格納する場合は GEMM の四種類の行列乗算の中では，TN 演算が最も高速となる傾向がある．これは行列データをメモリから読み込む際の効率が最も良いためである．実際，著者らの Tahiti GPU (Radeon HD 7970) における研究 [8] において，TN カーネルの最大性能は 790 GFlop/s であり，NN カーネルの最大性能 689 GFlop/s より高い性能値が得られた．この TN カーネルが最速という傾向は本研究で性能測定を行った全てのプロセッサにおいて同様である．一つの高速な行列乗算カーネルがあれば，四種全ての GEMM 演算はそのカーネルを利用できるように演算前にデータを並び替えることで実現できるので，以後は TN カーネルの最適化に絞って説明する．

メモリ上の行列データの局所性を高め，メモリへのアクセスパターンを最適化する方法として，ブロック優先なレイアウトにデータを格納する方法がある．入力行列 A, B のデータを格納する順序として行優先レイアウトに加えて，二種類のブロック優先なレイアウトもパラメタとして指定することで使えるようにした．それらのデータレイアウトを図 3 に示す (図は $m \times k$ 転置行列をブロック化因数

m_l, k_l でブロック分割した場合)．図 3(a) は行優先レイアウト (ROW: row-major layout) であるが，図 3(b) は列ブロック優先なレイアウトで各 $k \times m_l$ 列ブロック内のデータは行優先順に並んでいる．このレイアウトを列ブロックレイアウト (CBL: column block layout) と呼ぶ．CBL では，$k \times m_l$ ストライプと $k \times n_l$ ストライプを TN カーネルで乗算する際に必要なデータが連続したメモリ領域に格納される．図 3(c) は行ブロック優先なレイアウトで $k_l \times m$ 行ブロックの各 $k_l \times m_l$ サブブロックのデータは行優先で並ぶ．このレイアウトを行ブロックレイアウト (RBL: row block layout) と呼ぶ．RBL でデータを格納することで，$k_l \times m_l$ ブロックと $k_l \times n_l$ ブロックを TN カーネルで乗算する際には連続したメモリ領域にアクセスできるようになる．このようなブロック優先なレイアウトを使うことの有効性については，シングルコア CPU においてではあるが調べられている [9], [10].

3. 性能評価

この節では開発したコードジェネレータを使って生成した DGEMM と SGEMM 実装の性能評価をした結果を記す．性能測定は二種の GPU(Tahiti,Cayman) と二種の CPU(Sandy Bridge,Bulldozer) で行った．それらのプロセッサの仕様と性能測定をした環境を表 1 に示す．オペレーティングシステムは全て Ubuntu 10.04 で，コンパイルは `gcc 4.6.2` で行った．本稿で述べる性能測定の結果は，正方行列 ($n = m = k$) の場合のみである．また，性能値にはホストメモリと OpenCL デバイスのメモリ間のデータ転送にかかる時間は含んでいない．

3.1 TN カーネルの性能

開発したコードジェネレータにどのようなパラメタの組み合わせを与えれば最も高速な TN カーネルを生成することができるのかは明らかではない．本研究では，プロセッサと精度ごとに一万以上の TN カーネルの組み合わせを試し，その中で最高速なカーネルを選択した．それらの多数の組み合わせは測定した性能値をフィードバックとする経験則に基づき選んだ．そのフィードバックを基にしたパラメタの組み合わせの決定法はまだ改良の余地がある．以下は最速なカーネルを選択するための手順である．

(1) 生成した各行列乗算カーネルの二点の問題サイズ (GPU では n=1536 と 4096，CPU では n=768 と 1536) での性能を測定する．
(2) (1) で調べたカーネルのなかで最も速い 50 カーネルの性能を詳細に測定する (8192 までの 256 の倍数の問題サイズ n での性能測定を行う)．
(3) (2) で性能測定を行ったカーネルの中で最も高性能なカーネルを選ぶ．

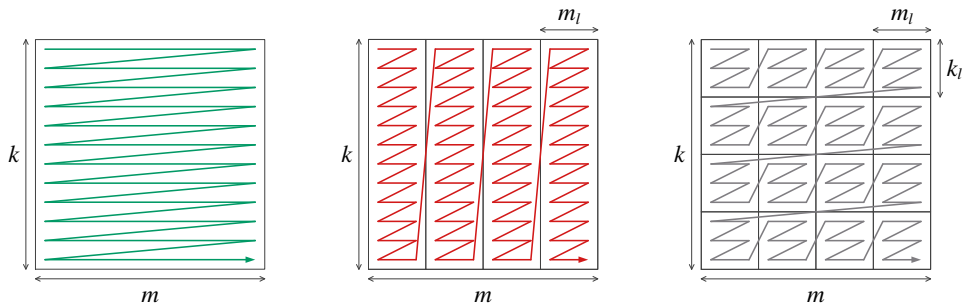
図 4 に上記手順により選択した高速な TN カーネルの演算性能を示す (性能の単位は GFlop/s で，行列サイズは 256 の倍数)．表 2 にそれらの TN カーネルを生成するのに設定したパラメタの組み合わせと，カーネルの計測された最大性能とピーク性能比を示す．Tahiti における DGEMM カーネルの最大性能が 848 GFlop/s と性能測定を行った四種のプロセッサの中では最も高く，そのピーク性能比は 90%に達する．SGEMM カーネルの Tahiti での最大性能は 2646 GFlop/s で，そのピーク性能比は 70%と実効効率は DGEMM の場合より低い．Cayman での TN カーネルのピーク性能比は Tahiti の場合と比べると数パーセント落ちるが，ほぼ同じような性能傾向を示す．ただし，他のプロセッサではデータレイアウトに CBL を使用することが最適であったが，Cayman では ROW や RBL を使った方がより高い性能が得られた．GPU での結果に対して，Sandy Bridge と Bulldozer の両 CPU の行列乗算カーネルのピーク性能比は 40%程度に止まった．この低い処理効率の主な要因としては次の二つが考えられる．

(1) 本研究では，AMD の GPU 向けに最適化した行列乗算カーネルの雛形を CPU に適用したため，その雛形が CPU に適していない．
(2) CPU 向けの OpenCL 言語のコンパイラがまだ未成熟．

3.2 TN カーネルを利用する GEMM 実装の性能

高速な TN カーネルを利用できるようにするためには，TN カーネル実行前に入力行列 A, B のデータを対応するレイアウトにコピーしなければならない．四種類全ての GEMM 演算をサポートするには，行列転置しながらコピーするカーネルと転置なしで単純に対応するレイアウトにコピーするカーネルという二種類の OpenCL によるコピーカーネルが求められる．例えば，NN 演算を TN カーネルを用いて行うためには，行列 A を転置しながらコピーし行列 B を転置なしでコピーする必要がある．正方行列の場合，各コピーカーネルは $n \times n$ 行列をグローバルメモリ上のメモリ領域から読み込み，読み込んだデータをグローバルメモリ上の別のメモリ領域に書き込む．コピーカーネルの時間計算量は $O(n^2)$ であり TN カーネルの時間計算量は $O(n^3)$ なので，行列サイズが大きくなるにつれてコピー処理にかかるオーバーヘッド時間の演算時間全体に占める割合は相対的に小さくなる．

図 5 にそのコピーカーネルと最速な TN カーネルを組み合わせて実現させた我々の GEMM 実装の演算性能を示す．この図では各種プロセッサにおける NN 演算の実装の性能を比較している．Tahiti において，行列サイズ n が 6400 のときに DGEMM 実装は 823 GFlop/s の最大性能を記録した (ピーク性能比は 87%)．また，SGEMM 実装は $n = 7424$ のときに 2541 GFlop/s(67%のピーク性能比) という最大性能が得られた．表 3 に DGEMM と SGEMM 実装の性能を測定した結果をまとめた．表ではプロセッサ



(a) 行優先レイアウト (ROW)　(b) 列ブロックレイアウト (CBL)　(c) 行ブロックレイアウト (RBL)

図 3　行列データの格納レイアウト．これは $m \times k$ 転置行列を m_l, k_l のブロック化因数でブロック分割した場合．

Code name	Tahiti	Cayman	Sandy Bridge	Bulldozer
Product name	HD 7970	HD 6970	Core i7 3960X	FX-8150
Core clock speed [GHz]	0.925	0.88	3.3	3.6
Number of compute units (cores)	32	24	6	8
Max DP operations / clock	1024	768	48	32
Max SP operations / clock	4096	3072	96	64
Peak DP performance [GFlop/s]	947	676	158.4	115.2
Peak SP performance [GFlop/s]	3789	2703	316.8	230.4
Global memory size [GB]	3	2	-	-
L2 cache size [kB]	768[a]	512[a]	256[b]	2048[c]
L1 cache size [kB]	16[b]	8[b]	32[b]	64[c]
Local memory size [kB]	64[b]	32[b]	-	-
OpenCL SDK	AMD APP 2.6	AMD APP 2.6	Intel SDK 2.0	AMD APP 2.7
AMD Catalyst Driver version	12.3	11.11	-	-

SP: Single-precision
DP: Double-precision
SDK: Software Development Kit
[a]: Size per processor
[b]: Size per compute unit (core)
[c]: Size per two cores

表 1　プロセッサ仕様と性能測定環境

ごとに我々の実装の性能とベンダライブラリに含まれる GEMM ルーチンの性能とを比較している．GPU においては，我々の実装は四種の GEMM 演算のタイプにほとんどよらず安定した性能を発揮し，ベンダライブラリのものより高い性能を達成した．それに対して CPU では，我々の OpenCL による実装はベンダライブラリを上回る演算性能を得ることができなかった．

4. 関連研究

Du ら [6] は OpenCL により行列乗算カーネルを実装した．彼らの AMD の Cypress GPU(Radeon HD 5870) に対する実装は，列優先 (column-major) に行列データが並んでいると想定したもので，その場合に高速な NT カーネル (行優先では TN カーネルと同等) を利用できるように行列データを演算前にコピーしている．我々の場合とは異なり CBL や RBL などのブロック優先なレイアウトは用いていない．彼らの倍精度 NN 演算実装の最大性能は 308 GFlop/s(ピーク性能比は 69%) と報告されている．

Nakasato[7] は，Intermediate Language (IL) により行列乗算カーネルを実装した．彼らの Cypress における倍精度 TN カーネルの最大性能は 472 GFlop/s である．本研究では Cypress においては詳細な性能評価は行わなかったが，我々の GEMM コードジェネレータを用いて生成した Cypress で最速な TN カーネルは最大で 513 GFlop/s(94%のピーク性能比) の性能を記録している．性能チューニングの仕方も深さも異なるので単純には比較できないが，OpenCL でも IL というアセンブリレベルの言語による実装と同等以上の性能が Cypress の DGEMM 演算の場合では得られた．

コードジェネレータは自動チューニング機構の主な構成要素の一つであり，本研究でも高速な行列乗算カーネルを生成するために簡易な自動チューニングを行った．自動チューニングは最適に近い GEMM 実装を実現するために有効な技術であり，CPU の BLAS ルーチンの自動チューニングを行う ATLAS (Automatically Tuned Linear Algebra Software) ライブラリはよく知られている [14]．また，Kurzak ら [4], [15] は CUDA(Compute Unified Device Architecture) 対応の GPU 向けに GEMM 実装の自動チューニング機構を開発した．彼らの SGEMM 実装 (NN 演算)

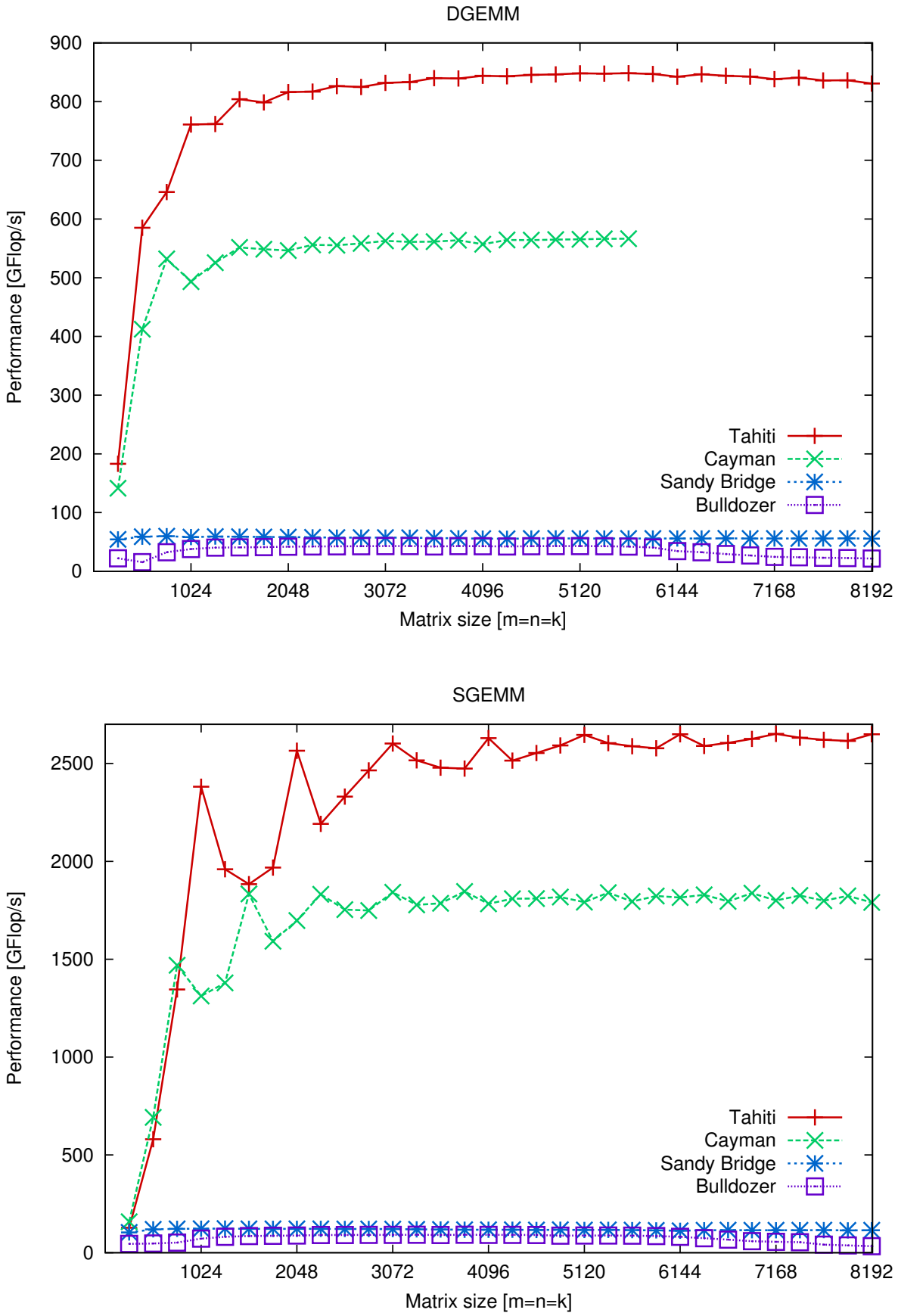


図 4　各種プロセッサにおける最速な TN カーネルの演算性能

Precision	Processor	m_l, n_l, k_l	m_s, n_s, k_s	Vector[a]	Shared[b]	Layout[c]	Perf. [Gflop/s]	Efficiency
DGEMM	Tahiti	64,16,16	4,4,2	2	B	CBL,CBL	848	90%
	Cayman	32,64,256	8,4,2	2	-	CBL,ROW	566	84%
	Sandy Bridge	16,8,4	16,8,4	4	-	CBL,CBL	60	38%
	Bulldozer	32,4,8	4,4,8	2	-	CBL,CBL	43	37%
SGEMM	Tahiti	128,128,256	16,8,4	4	-	CBL,CBL	2646	70%
	Cayman	128,64,32	16,4,4	4	-	RBL,RBL	1845	68%
	Sandy Bridge	16,32,128	8,4,4	4	-	CBL,CBL	124	39%
	Bulldozer	64,8,8	4,8,8	4	-	CBL,CBL	92	40%

a. Width of vector variables.

b. Matrix whose data are shared in local memory.

c. Data layout for matrices A, B, respectively.

表 2 最速な TN カーネルの生成するためにコードジェネレータへ与えたパラメタとそのカーネルの最大性能とピーク性能比

Precision	Operation	Tahiti		Cayman		Sandy Bridge		Bulldozer	
		Ours	Vendor[a]	Ours	Vendor[a]	Ours	Vendor[b]	Ours	Vendor[c]
DGEMM	$\alpha AB + \beta C$	823	612	532	369	53	122	36	50
	$\alpha AB^T + \beta C$	826	413	534	268	52	123	35	50
	$\alpha A^T B + \beta C$	826	623	525	404	52	122	36	50
	$\alpha A^T B^T + \beta C$	831	598	528	366	52	124	36	50
SGEMM	$\alpha AB + \beta C$	2541	1496	1694	1073	108	275	76	100
	$\alpha AB^T + \beta C$	2505	906	1708	646	107	274	77	102
	$\alpha A^T B + \beta C$	2601	1796	1688	1336	108	275	81	103
	$\alpha A^T B^T + \beta C$	2530	1493	1706	1075	107	276	80	102

[a]: AMD Accelerated Parallel Processing Math Libraries (APPML) clBLAS 1.8.269 [11]

[b]: Intel Math Kernel Library (MKL) 10.319 [12]

[c]: AMD Core Math Library (ACML) 5.1.0 [13]

表 3 各種プロセッサにおける我々の GEMM 実装とベンダライブラリの GEMM ルーチンの最大性能 (単位は GFlop/s)

の最大性能は 1125 GFlop/s(36%のピーク性能比) と報告されている [15].

5. おわりに

本研究は，開発した GEMM コードジェネレータを利用することで，倍精度行列乗算 (DGEMM) と単精度行列乗算 (SGEMM) を高速に行う OpenCL によるカーネルを各プロセッサごとに実装することを目指した．GPU においては最大性能のピーク性能比が DGEMM カーネルで 80%，SGEMM カーネルで 65%を超え高い性能が得られた．しかし，CPU の場合のピーク性能比は 40%程度に止まった．コードジェネレータを改良し，GPU だけでなく CPU でもより高性能な行列乗算カーネルを生成できるようにすることは今後の課題である．

参考文献

[1] Basic Linear Algebra Subprograms Technical Forum: Basic Linear Algebra Subprograms Technical (BLAST) Forum Standard, available from http://www.netlib.org/blas/blast-forum/blas-report.pdf (2001).

[2] Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A. and Sorensen, D.: *LAPACK User's Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 3rd edition (1999).

[3] Kågström, B., Ling, P. and van Loan, C.: GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark, *ACM Transactions on Mathematical Software*, Vol. 24, No. 3, pp. 268–302 (1998).

[4] Kurzak, J., Tomov, S. and Dongarra, J.: Autotuning GEMM kernels for the Fermi GPU, *IEEE Transactions on Parallel and Distributed Systems*, available from http://dx.doi.org/10.1109/TPDS.2011.311 (2012 (in press)).

[5] Tan, G., Li, L., Triechle, S., Phillips, E., Bao, Y. and Sun, N.: Fast implementation of DGEMM on Fermi GPU, *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC '11)*, Seattle, WA, USA, ACM (2011).

[6] Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G. and Dongarra, J.: From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming, *Parallel Computing*, Vol. 38, No. 8, pp. 391–407 (2011).

[7] Nakasato, N.: A fast GEMM implementation on the Cypress GPU, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 38, No. 4, pp. 50–55 (2011).

[8] Matsumoto, K., Nakasato, N. and Sedukhin, S. G.: Implementing a code generator for gast matrix multiplica-

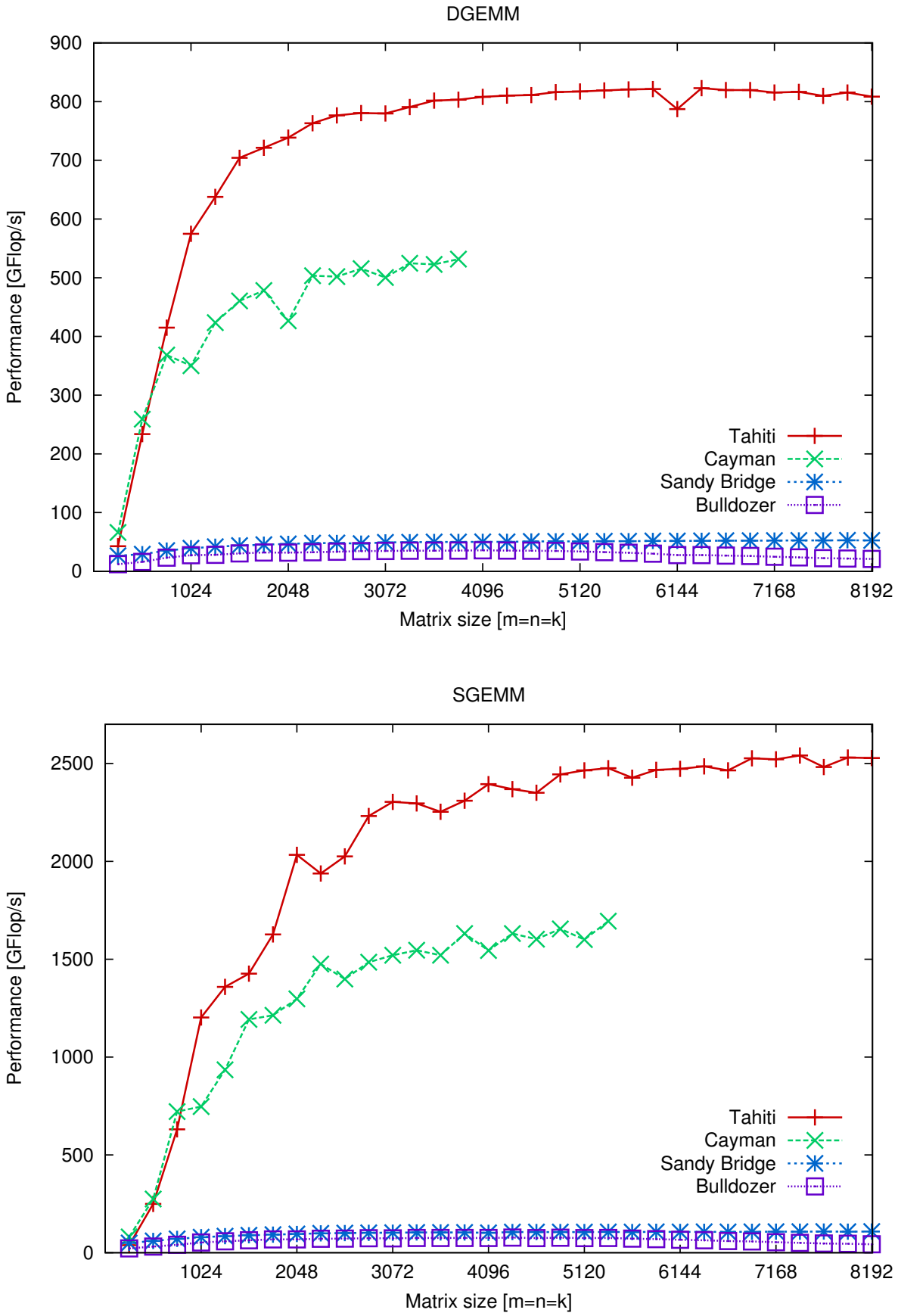


図 5　TN カーネルとコピーカーネルを組み合わせて実現した我々の NN 演算実装の性能

tion in OpenCL on the GPU, *Auto-Tuning for Multicore and GPU (ATMG) Workshop*, Fukushima, Japan (2012 (accepted)).

[9] Prasanna, V. K., Park, N. and Hong, B.: Tiling, block data layout, and memory hierarchy performance, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, No. 7, pp. 640–654 (2003).

[10] Gustavson, F. G.: New generalized data structures for matrices lead to a variety of high performance algorithms, *Proceedings of the 4th International Conference on Parallel Processing and Applied Mathematics (PPAM '01)*, LNCS, Vol. 2328, pp. 418–436 (2002).

[11] AMD Inc.: AMD Accelerated Parallel Processing Math Libraries (APPML) (online), available from http://developer.amd.com/libraries/appmathlibs (accessed 2012.06.27).

[12] Intel Corp.: Math Kernel Library from Intel (online), available from http://software.intel.com/en-us/articles/intel-mkl (accessed 2012.06.27).

[13] AMD Inc.: AMD Core Math Library (ACML) (online), available from http://developer.amd.com/libraries/acml (accessed 2012.06.27).

[14] Whaley, R. C., Petitet, A. and Dongarra, J. J.: Automated empirical optimizations of software and the ATLAS project, *Parallel Computing*, Vol. 27, No. 1-2, pp. 3–35 (2001).

[15] Kurzak, J., Luszczek, P., Tomov, S. and Dongarra, J.: Preliminary results of autotuning GEMM kernels for the NVIDIA Kepler architecture - GeForce GTX 680, LAPACK Working Note 267, available from http://www.netlib.org/lapack/lawnspdf/lawn267.pdf (2012).