Vol.2010-HPC-124 No.11
2010/2/23

OpenCLによるGPUコンピューティングの性能評価

荒井勇亮[†1] 佐藤功人[†2]
滝沢寛之[†2,†4] 小林広明[†3,†4]

近年，従来のCUDAに加えて，GPGPUプログラミングのための新たな標準プログラミング環境としてOpenCLが利用可能となった．本論文では，CUDAとOpenCLのプログラムの実行性能差を定量的に評価する．まず，ほぼ同等の処理を行うCUDAとOpenCLのプログラムを実装し，性能を比較する．次に，その性能差の主要因を調査し，CUDAコンパイラではサポートされているいくつかのコンパイラ最適化手法が，現在のOpenCLコンパイラではサポートされていないことを明らかにする．最後に，OpenCLコンパイラで生成されるコードを手動で最適化することによってCUDAと同等の性能を達成できた結果から，今後のOpenCLコンパイラの最適化機能が強化されることにより，CUDAコードをOpenCLに単純変換するだけでも，CUDAと同等の性能を達成できる可能性が示された．

Performance Evaluation of GPU Computing with OpenCL

Yusuke Arai ,[†1] Katsuto Sato ,[†2]
Hiroyuki Takizawa [†2,†4] and Hiroaki Kobayashi [†3,†4]

Recently, a new open programming standard for GPGPU programming, OpenCL, has become available in addition to CUDA. In this paper, we quantitatively evaluate the performance of CUDA and OpenCL program. First, we develop some CUDA and OpenCL programs of almost the same computations and compare their performances. Then, we investigate the main factor causing their performance differences. As a result, it is shown that the current OpenCL compiler does not support several compiler optimizations that are used in the CUDA compiler. Our evaluation results also shows that OpenCL programs can achieve comparable performances with CUDA programs if the codes generated by the OpenCL compiler are manually optimized in the same way as the CUDA compiler. Therefore, these results suggest a possibility that OpenCL codes simply translated from CUDA codes can achieve the same performance with the original CUDA codes if the OpenCL compiler supports those optimizations.

1. はじめに

近年，描画処理専用プロセッサ (Graphics Processing Unit : GPU) の発展により，GPUを描画処理以外の汎用計算に応用する General Purpose computation on GPU (GPGPU) という概念が提唱されている．GPUを利用するプログラムであるGPGPUプログラムは，GPUの持つ高い演算能力とメモリバンド幅を利用することで，汎用プロセッサ (CPU) のみを利用する場合に比べて高い演算性能を達成することが可能である．

2006年に，GPUを用いたプログラムを開発する環境として，NVIDIAからCompute Unified Device Architecture(CUDA)[6] が発表された．AMDも同時期にBrook+[3] というソフトウェア開発環境を提供し，これまで特殊な知識や技能が要求されたGPUプログラミングが比較的容易になった．これらの開発環境を用いることで，各社のGPU上で動作するプログラムを開発できるようになったが，CUDAとBrook+は互いにC言語を拡張した言語であるにもかかわらず，追加された予約語や文法が異なるために，ソースコードを相互に流用することができない．結果として，GPGPUプログラムは開発環境とハードウェア構成に依存するプログラムとなる．

これに対して，ベンダに依存しないGPGPU用ソフトウェア開発環境としてOpenCL[1] が提案されている．OpenCLはGPUのみならず，並列処理を目的として幅広いハードウェアへの対応を目的としている．そのため，マルチコアCPUやCELL Broadband Engine等の他のハードウェアに対しても再実装を行うことなく対応可能である．OpenCLを用いてプログラムを実装することで，ベンダに依存することの無いGPGPUプログラムを記述することが可能となり，プログラムの再利用性と，開発効率を改善することが可能である．しかし，CUDAやBrook+等の自社製品に特化した開発環境と比較し，ベンダ非依存の抽象化が行われているOpenCLで実現可能な性能について定量的に評価した結果は報告されていない．

†1 東北大学工学部 機械知能・航空工学科
Department of Mechanical and Aerospace Engineering, School of Engineering, Tohoku University.
†2 東北大学大学院 情報科学研究科
Graduate School of Information Sciences, Tohoku University.
†3 東北大学 サイバーサイエンスセンター
Cyberscience Center, Tohoku University.
†4 科学技術振興機構 戦略的創造研究推進事業
Japan Science and Technology Agency, Core Research for Evolutional Science and Technology

本論文では，CUDA と OpenCL のプログラムの実行性能差を定量的に評価する．まず，ほぼ同等の処理を行う CUDA と OpenCL のプログラムを実装し，性能を比較する．さらに，両者の性能差を生じる要因を解析する．

2. OpenCL の概要

2.1 CUDA と OpenCL の比較

CUDA は，2007 年に NVIDIA によって提供が開始された GPU 向けソフトウェア開発環境である．CUDA を用いることで，GPU に対して描画用の Application Programming Interface（API）を用いることなく GPU 上で動作するプログラムを記述することが可能となり，GPU の持つ高い演算能力を容易に利用することが可能となる．CUDA は GPGPU プログラミングの発展に大きく貢献し，CUDA によって様々なプログラムが記述されるようになったが，CUDA は NVIDIA 社製 GPU のみに提供される環境であるために，OS の標準機能として組み込まれるには難があった．

2009 年に Apple 社から自社製 OS 向けに GPGPU を行うための環境を標準搭載することが発表され，それに伴いベンダに依存しない GPU 向けソフトウェア開発環境として OpenCL が提案された[1)]．OpenCL が提供する言語で記述されたプログラムは NVIDIA のみならず，他社の GPU や並列処理システム上でもサポートが表明されており，移植性や再利用性に優れた開発環境である．OpenCL は言語仕様において CUDA と対応する部分を持ち，CUDA プログラミングで獲得した知識や経験を応用することが可能である．

CUDA と OpenCL には共通する部分が多い一方で，ハードウェアアーキテクチャの抽象化の方法は CUDA と OpenCL では若干異なっている．CUDA と OpenCL におけるハードウェアアーキテクチャの概要を **図 1** と**図 2** に示す．これらの図からわかるように，OpenCL には CUDA のアーキテクチャに見られる GPU 特有のテクスチャ用のキャッシュメモリは規定されておらず，より汎用的なハードウェアアーキテクチャを想定している．

図 3 に CUDA と OpenCL の thread 階層に関する概略図を示す．CUDA では，thread をいくつか集めたものを thread block と呼び，さらに thread block を集めたものを grid と呼ぶ．また，CUDA では thread を実行する際に，1 つの thread block 内においてハードウェア的に同時に処理される thread set が定義されており，この thread の集合を warp と呼ぶ．

OpenCL では，CUDA における thread を work-item と呼び，work-item をいくつか集めたものを work-group と呼ぶ．この work-group が CUDA における thread block に対応

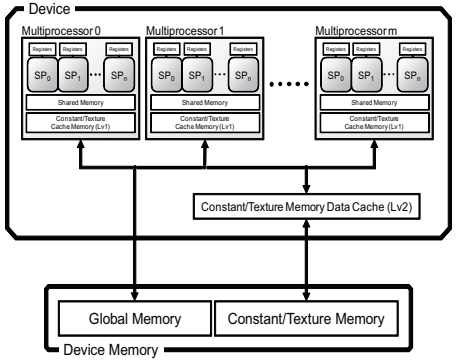

図 1 CUDA における GPU アーキテクチャ概要
Fig. 1 GPU Architecture on CUDA.

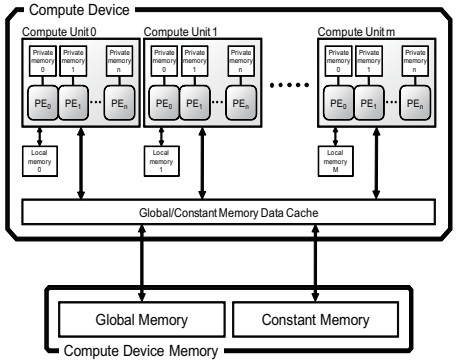

図 2 OpenCL における GPU アーキテクチャ概要
Fig. 2 GPU Architecture on OpenCL.

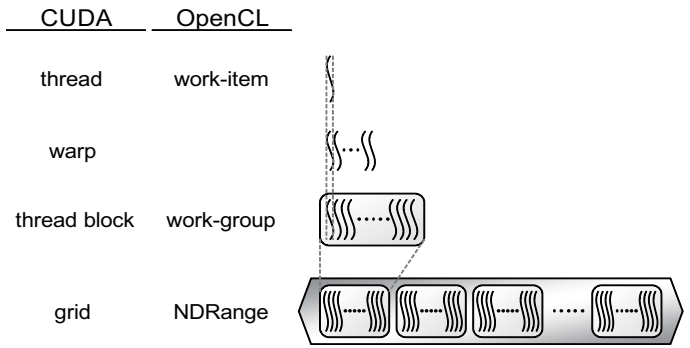

図 3 thread 階層
Fig. 3 Thread Hierarchy.

表 1 メモリ階層
Table 1 Memory hierarchy.

アクセス可能単位	CUDA	OpenCL
per thread / work-item	private local memory	private memory
per thread block / work-group	shared memory	local memory
per grid / NDRange	global memory	global memory
	constant memory	constant memory
	texture memory	—

する．さらに，work-group をいくつか集めたものを NDRange と呼ぶ．この NDRange は CUDA における grid に対応する．このように CUDA と OpenCL では，thread 階層における各階層の名称が異なるだけで，ほぼ同じ thread 階層を定義していることがわかる．一方で，OpenCL には CUDA における warp のような単位は存在せず，より汎用的なシステムに対応できる形をとっている．

表 1 に OpenCL と CUDA のメモリ階層の対応関係を示す．CUDA では，各 thread に対して，private local memory が割り当てられている．このメモリ領域は割り当てられた thread のみアクセスが可能である．各 thread block に対しては，shared memory が割り当てられている．shared memory にアクセスできるのは，このメモリ領域に割り当てられたブロック内の thread からのみである．global memory はすべての thread からアクセスできる．さらに，CUDA ではキャッシュを備えた constant memory と，texture memory が存在し，どちらもすべての thread からアクセス可能である．

CUDA における private local memory は，OpenCL では，private memory に対応する．また，CUDA における shared memory は，OpenCL における local memory に対応する．global memory は，CUDA と OpenCL の両方で同じ名称で定義されている．同様に，CUDA における constant memory に対応するメモリ領域として，OpenCL においても constant memory と呼ばれるメモリ領域が存在する．このように CUDA におけるメモリ階層と OpenCL におけるメモリ階層には共通点が多く見られる．

CUDA と OpenCL における，カーネルのコンパイル方法の違いについて述べる．CUDA では，カーネルのコードは静的にコンパイルされる．一方で，OpenCL では，カーネルのコードはプログラムの実行時に Just-In-Time コンパイラ (JIT コンパイラ) を用いてコンパイルされる．このため，OpenCL ではカーネルのコンパイル時間がプログラムの実行時間に含まれるため，コンパイル時間に制約が存在する．

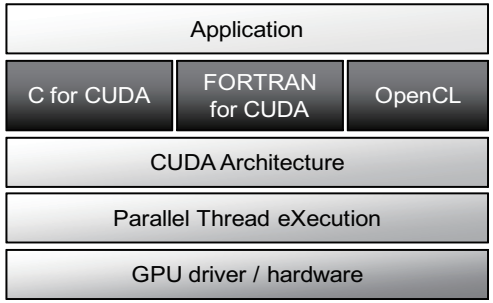

図 4 ソフトウェア階層
Fig. 4 Software hierarchy.

2.2 仮想命令セットアーキテクチャ Parallel Thread eXecution (PTX)

NVIDIA 社製の GPU において，GPU 上で実行されるプログラムは Parallel Thread eXecution (PTX)[5] という中間コードを用いてプログラム中に格納されている．PTX は RISC に似た仮想命令セットであり，実行時にビデオドライバによって GPU の実命令セットに変換される．NVIDIA が提供する CUDA および OpenCL では，どちらを用いた場合でも GPU 上で実行されるコードは PTX に変換されるため[⋆1]，カーネル部分での性能の違いは，コンパイラから生成される PTX を解析することで調べることが可能である．**図 4** に PTX の位置づけを表したソフトウェア階層を示す．また，OpenCL の JIT コンパイラは，PTX から GPU の実命令へ変換することも可能であるため，OpenCL では PTX を用いてカーネルを記述することも可能である．

3. 性 能 評 価

本節では，OpenCL および CUDA で実装されたベンチマークを用いて，それぞれのソフトウェア開発環境について評価を行う．

3.1 性能評価環境

評価には 2 種類のベンチマークセットを用いる．ベンチマークセットの一つ目は NVIDIA が提供している OpenCL および CUDA の SDK に付属するサンプルプログラムである．こ

⋆1 NVIDIA が提供する OpenCL では，カーネルコードはコンパイラによって PTX に変換されるが，他社が提供する OpenCL ではたとえ同じ GPU であるとしても PTX に変換されることは保証されていない．

のサンプルプログラムには，基本的な演算性能を計測するためのベンチマークが含まれており，本研究では，このサンプルプログラムの中から CUDA と OpenCL において同条件で性能評価可能なプログラムを選び，それらを用いることで性能を定量的に比較する．本論文では，行列積の演算性能を計測する `matrixMul` と，実効メモリバンド幅を計測する `bandwidthTest` を用いる．`bandwidthTest` に関しては，変更を加えずとも同条件で実行可能であったので，何も手を加えずそのまま性能評価を行った．`matrixMul` においては，性能評価条件を同一にするために 2 点変更を加えている．1 点は，オンチップメモリへのアクセス方法が CUDA と OpenCL で異なっていた点である．この点においては，CUDA におけるアクセス方法に統一した．もう一点は，演算に用いる行列のサイズを，より大きなサイズを用いている OpenCL に合わせたことである．

もう一つのベンチマークセットは，Illinois Microarchitecture Project Utilizing Advanced Compiler Technology(IMPACT) プロジェクトが公開している `Parboil Benchmark Suite`[4] である．Parboil Benchmark は 6 種類のプログラムから構成されており，中でも Coulombic Potential (`CP`)，Magnetic Resonance Imaging Q (`MRI-Q`)，Magnetic Resonance Imaging FHD (`MRI-FHD`) は，GPU が高い演算性能を発揮できるアプリケーションである[7]．このベンチマークセットでは CUDA で実装されたプログラムは提供されているが，OpenCL で実装されたものは提供されていない．本研究では前述した `CP`，`MRI-Q`，`MRI-FHD` の 3 つのプログラムを直訳する形で OpenCL に移植し，性能評価に用いる．

本節の性能評価は以下のシステム環境で行う．

- CPU : Intel Core i7 920 2.66GHz
- GPU : Tesla C1060
- Main Memory : 12 Gbytes
- Video Memory : 4 Gbytes
- Video Driver Version: 190.29
- OS : Linux 2.6.18-164.el5 (CentOS 5.3) x86_64
- Compiler for CPU : gcc 4.1.2
- Compiler for GPU : nvcc release 2.3

使用した SDK は NVIDIA GPU Computing SDK 2.3 である．

3.2 性能評価結果

CUDA および OpenCL の SDK に含まれる，`bandwidthTest` と `matrixMul`，Parboil Benchmark の `CP`，`MRI-FHD`，`MRI-Q` の実効演算性能を**図 5** に示す．図中の棒グラフは，

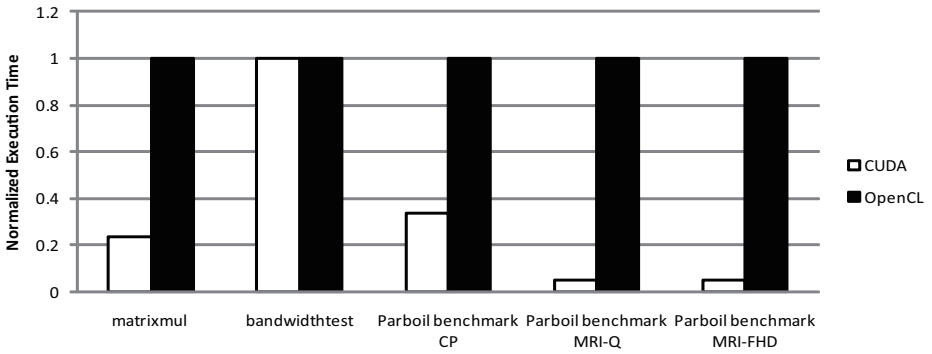

図 5 性能評価結果
Fig. 5 Performance Differences between CUDA and OpenCL.

OpenCL で実装された各ベンチマークの実行時間を 1 として，CUDA で実装したベンチマークの実行時間を正規化して示している．図 5 からわかるように，NVIDIA の SDK で提供されている CUDA と OpenCL のプログラムでは，ライブラリが提供する API のみを使う `bandwidthTest` と，カーネルをユーザ定義コードとして記述している `matrixMul` で異なる傾向が見られた．`bandwidthTest` では，CUDA と OpenCL の間に有意な差は見られない．一方，カーネルを用いる `matrixMul` では，CUDA と OpenCL で約 4.2 倍の性能差が見られる．以上の結果から，カーネル以外の部分における API 呼び出しでは CUDA と OpenCL の間には性能差がない一方で，ユーザ定義カーネルでは大きな性能差があることがわかる．実用的なプログラムを記述する場合，ライブラリが提供する基本的な API のみで構成することはないため，現在の OpenCL では CUDA と同等のコードを記述しても CUDA に匹敵する性能を達成することはできないことが分かる．

次に， Parboil Benchmark の `CP`，`MRI-FHD`，`MRI-Q` の性能比較結果について述べる．どのベンチマークでも，CUDA に対して OpenCL によって実装されたプログラムの性能は低く，`CP` では，CUDA に比べて約 3 倍の実行時間を要し，`MRI-Q` および `MRI-FHD` では約 19 倍もの実行時間を要している．これらのプログラムも，OpenCL への移植時に特別な変換を行っているわけではなく，予約語と API 関数の変換程度しか行っていないにもかかわらず，性能に大きな差が見られる結果となった．

前述の `matrixMul` による評価結果と同様に，Parboil benchmark の 3 つのアプリケー

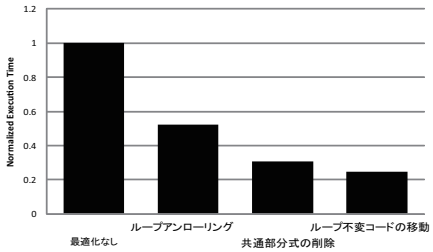

図 6 PTX 修正による実行時間の減少率
Fig. 6 Reduction Rate of Execution Time by Modifying the PTX Code.

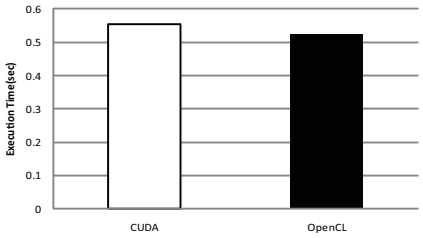

図 7 PTX 修正後の OpenCL と CUDA の性能比較
Fig. 7 Performance Difference between CUDA and Optimized OpenCL.

ションにおいても，OpenCL で実装されたプログラムの実効性能は CUDA に比べ低いことが明らかになった．CUDA と OpenCL は類似するプログラミングモデルを持っており，これらの性能差はプログラミングモデルによるものではない．高位言語の段階では同じ処理内容であっても，両者に性能差が生じている原因として，より機械語に近いレベルで処理内容を比較した時に，両者の処理内容が異なっている可能性が考えられる．そこで次節では，OpenCL の性能を低下させている原因を明らかにするために，本節で用いた `matrixMul` を用いて，さらに詳細な性能の解析を行う．

4. OpenCL における性能に対する考察

前節では，OpenCL は CUDA に比べて低い実効演算性能となることを述べた．CUDA および OpenCL において，API レベルでは性能に差がないことから，性能の異なる原因はカーネル部分のコードにおける性能差であると推測される．本節では，カーネル部分のコードをコンパイルした結果として出力される PTX コードを解析し，前節の結果にあるような大きな性能差を生じる原因について考察を行う．

本節では，考察の対象として `matrixMul` ベンチマークを用いる．行列積は Operation Intencity[8] が高く，GPU が高い性能を達成できるプログラムである．このため `matrixMul` カーネルの性能について調査をすることで，現状の OpenCL において高い性能が得られていない理由について議論する．

まず，CUDA 言語で記述されたカーネルと OpenCL 言語で記述されたカーネルをそれぞれの環境でコンパイルし，得られた PTX コードを解析した．その結果，ほぼ同じ内容を記述しているカーネルコードをコンパイルしているにもかかわらず，全く異なる PTX コードが生成されていることが明らかとなった．これより，OpenCL プログラムが CUDA プログラムと同等の性能を達成できない原因は，PTX レベルにおいて，CUDA では行われている自動最適化が OpenCL では行われていないためであると考えられる．以降では OpenCL において生成された PTX コードを，CUDA において生成された PTX コードに近づくように手作業で修正する．段階的に修正した際の性能の変化を **図 6** に示す．この図では，手動最適化を施していない `matrixMul` の実行時間を 1 として，各最適化を施したプログラムの実行時間を正規化して示している．

ループ内の繰り返し処理を展開する最適化であるループアンローリング[2] は，最も一般的なループ最適化手法の一つである．特に定数回のループ処理においては，ループを展開することでループの終了判定回数やインデックス計算回数を削減することができるとともに，前後の繰り返し処理で重複するメモリアクセスを統合してメモリアクセス回数を削減する効果も得られる．CUDA では，すでに 16 段のループアンローリングが適用された状態となって PTX コードが生成されるが，OpenCL ではループアンローリングが適用されていない PTX コードが生成される．この OpenCL のコードを手動でループアンローリングすることにより，CDUA と同様にループアンローリングされた PTX コードが生成された．このループアンローリングにより，約 48%の実行時間短縮が達成された．

ループアンローリングを適用した後の PTX コードでは，配列アクセス時のアドレス計算方法が CDUA と OpenCL で異なっている．CUDA における配列アクセスでは, 一度計算した配列の先頭アドレスを共通部分式[2] として扱い，配列の先頭アドレスの再計算を避けている．一方 OpenCL では，すべてのアドレス計算を個別に行い，実効アドレスを算出してからメモリアクセスを行っている．CUDA で生成された PTX コードと同様に共通部分式の削除を手動で行ったところ，OpenCL 版 `matrixMul` の実行時間をさらに約 40%短縮できた．

PTX には積和演算命令（mad 命令）も用意されている．CUDA で生成された `matrixMul` カーネルの PTX コードではこの mad 命令が使用されているのに対し，OpenCL の PTX コードでは積和演算を，乗算と加算の 2 命令で行っていた．乗算と加算の代わりに，mad 命令を用いれば命令数が減り，性能向上が見込まれる．この最適化手法を，ループアンローリングと共通部分式の削除を適用した後の OpenCL のコードに適用したところ有意な性能向上はみられなかった．本評価で扱ったコードでは演算がボトルネックになっていなかったと考えられる．

CUDA ではループ不変式をループの前に移動する，ループ不変式の移動と呼ばれる最適化[2] が適用された PTX コードが生成される．一方 OpenCL ではこのような移動は行われておらず，ループ内にループ不変式が残されたままであった．ループ不変式の移動をこれまで行った最適化と合わせて OpenCL における PTX コードに適用したところ，さらに約 20 %実行時間を短縮できた．

以上，OpenCL のコンパイラが生成した PTX コードに対して手動で基本的な最適化を適用したところ，実行時間を大幅に短縮することができた．OpenCL の PTX コードにすべての最適化を適用した場合の実行時間を，CUDA の PTX コードの実行時間と比較した結果を**図 7** に示す．図 7 が示すとおり，以上に挙げた最適化手法をすべて適用することで，OpenCL 版の `matrixMul` の実効性能は CUDA 版の実効性能とほぼ同等となった．

以上の結果から，CUDA 版の `matrixMul` カーネルの PTX コードには，自動的に適用されているいくつかの基本的な自動最適化手法が，OpenCL 版の PTX コードには適用されていないことが明らかになった．CUDA コンパイラが静的なコンパイルによって PTX コードを生成しているために，コンパイル時間には制約がない．これに対して，OpenCL では実行時にコンパイルを行うために，変換に長い時間を要するような高度な解析および最適化が意図的に抑えられている可能性が考えられる．本評価結果より，OpenCL においても PTX コードに基本的な最適化手法を施すことによって，CUDA と同等の性能が達成できることが明らかになった．今後，OpenCL の PTX コンパイラが改良されることにより，自動生成される PTX コードの性能的品質が改善される可能性がある．したがって，現在 CUDA で開発されているコードを OpenCL に単純変換するだけで同程度の性能を達成できる可能性があり，CUDA から OpenCL への円滑な移行の可能性が示唆されたといえる．

これらの結果から，Parboil benchmark の 3 つのプログラムにおける性能差の要因を考察する．本評価により，OpenCL の matrixMul カーネルには基本的なコンパイラ最適化手法すら適用されていないことが示された．このため，Parboil benchmark の 3 つのプログラムに関しても，同様の理由で大きな性能差となっている可能性が考えられる．`CP` に関しては，実際にそれぞれの PTX コードを確認したところ，CUDA では mad 命令が使用されているのに対し，OpenCL では積和演算を乗算 1 回，加算 1 回の 2 つの命令で行っていることが分かった．また，CDUA では行われていた共通部分式の削除が，OpenCL で行われていなかった．これらは，行列積演算において見られた差異と同様である．したがって，`CP` においても OpenCL の PTX コードを修正すれば，OpenCL が CUDA と同程度の性能を達成できる可能性がある．`CP` における OpenCL の PTX コードの修正および，`MRI-Q`, `MRI-FHD` の詳細な解析は今後の課題である．

5. おわりに

本論文では，現在 GPGPU プログラミングで事実上の標準として利用されている CUDA と，ベンダ非依存の新しいプログラミング環境である OpenCL の性能を定量的に比較した．SDK 付属のサンプルコードと Parboil benchmark を用いた性能評価結果から，OpenCL と CUDA で同じ処理を実装した場合に前者の方が実効性能が低くなる傾向があることが明らかになった．SDK に含まれている `matrixMul` カーネルについて PTX コードを解析した結果，OpenCL では PTX コード生成時に基本的なコンパイラ最適化手法が適用されていないことが明らかになった．CUDA で適用されている最適化手法を OpenCL で生成される PTX コードにも手作業で適用したところ，OpenCL と CDUA でほぼ同等の性能が達成できることが明らかになった．

OpenCL はまだ発展途上であり，Just-In-Time 方式でコンパイルを行うことも最適化が不十分な原因になっていると考えられる．今後コンパイラが改善され．OpenCL で生成される PTX コードにも CUDA と同等の自動最適化が期待できる場合には，既存の CUDA コードを OpenCL に単純変換するだけでも，CUDA と同等の性能を実現できる可能性が示された．

謝　辞

本研究の一部は, 科研費若手研究（B）(21700049)，中山隼雄科学技術文化財団，および東北大学グローバル COE プログラム 流動ダイナミクス知の融合教育研究世界拠点の助成による．また，本研究の一部は科学技術振興機構戦略的創造研究推進事業「自己修復機能を有する 3 次元 VLSI システムの創製」によるものである．

参 考 文 献

1) Khronos OpenCL Working Group . The OpenCL Specification version 1.0 . `http://www.khronos.org/opencl/`.
2) RaviSethi Alfred V.Aho, Monica S.Lam and JeffreyD. Ullman. *Compilers: Principles, Techniques, and Tools, 2nd Edition.* Addision-Wesley, 2 edition, 2007.
3) AMD Corporation. ATI STREAM ATI stream computing user guide version 1.4 beta, April 2009.
4) The IMPACTResearch Group. Perboil Benchmark suite . `http://impact.crhc.`

illinois.edu/parboil.php.
5) NVIDIA Corporation. NVIDIA Compute PTX : Parallel Tread Execution ISA Version 1.2, 2008.
6) NVIDIA Corporation. NVIDIA CUDA Compute Unified Device Architecture programming guide version 2.0, 2008.
7) Shane Ryoo, ChristopherI. Rodrigues, SaraS. Baghsorkhi, SamS. Stone, DavidB. Kirk, and Wen-meiW. Hwu. Optimization principles and application performance evaluation of a multithreaded GPU using CUDA. In *PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 73–82, New York, NY, USA, 2008. ACM.
8) David A. Patterson Samuel Webb Williams, Andrew Wateman. Roofline: An insightful visual performance model for floating-point programs and multicore architectures. *Electrical Engineering and Computer Sciences University of California at Berkley*, 2008.