Vol.2010-ARC-192 No.23
Vol.2010-HPC-128 No.23
2010/12/17

OpenCLを用いた異種GPUにおける性能特性に応じた最適化

島田大地[†1,†2] 遠藤敏夫[†1,†2]
丸山直也[†1,†2] 松岡聡[†1,†2,†3]

近年、GPUは科学技術計算などに広く用いられおり、現在多くの種類のGPUが存在している。また、CUDAやOpenCLを使用することによりプログラムに可搬性を持たせることが出来る。しかし、各GPUでアーキテクチャが異なるため性能可搬性は保証されない。したがって、性能向上をさせたい場合にはそれぞれのGPUに対して最適化を行う必要があるが、最適化による効果が必ずしも労力に見合ったものとは限らない。そのため、事前に最適化効果を定量的に把握できると人的リソースを用いるかどうかの指針を得ることができる。本論文では最適化効果を定量的に予測するためにパフォーマンスカウンタから実行時間予測を行いモデルを作成、評価する。

Towards Characteristic-aware Optimization of OpenCL programs on Heterogeneous GPUs

Daichi Shimada,[†1,†2] Toshio Endo,[†1,†2]
Naoya Maruyama[†1,†2] and Satoshi Matsuoka[†1,†2,†3]

Recently various graphic processing units (GPU) are widely accepted to accelerate scientific applications. With the OpenCL language, programmers can develop programs that can be executed on those heterogeneous GPUs. While portability has been realized, performance portability is not, since there exist gaps in characteristics among different GPU architectures. Thus programmers need to optimize their programs depending on GPU architecture. However, it is often hard to say whether performance improvement by such tasks are worthy to commit human resource beforehand. In order to give quantitative hints for such situations, this paper presents a performance model to estimate effects of optimizations on different GPU architectures, by using performance counters.

1. はじめに

近年、GPUの発展に伴いGPUを画像処理以外への応用として科学技術計算などに用いるGPGPUが注目されている[8)]。CPUとGPUのピーク性能の差は広がっており、効率よくGPUを用いることでCPUよりも高い性能を得ることが出来ると期待されている。また、GPUの年間性能向上率も高く比較的安価に入手できるため、大規模な計算環境では複数のヘテロなGPUを搭載したマシンが用意されていくと考えられる。

一方、GPU以外にもCell/B.EやCore i7など多数のメニーコアプロセッサがあり、これらのプロセッサも高性能演算に用いられている。しかし、プロセッサ毎の異なる言語を用いなくてはこれらのプロセッサを用いた演算を行うことが出来ず、一方のプロセッサに対応する言語で書かれたプログラムを別のプロセッサで実行したいときなどはプログラムを書き直す必要があった。そこで、メニーコア並列計算のための共通フレームワークとしてOpenCL(Open Computing Language)が作成された。OpenCLを用いることで異種メニーコアプロセッサ上でもCPU上でも単一プログラムが動作可能となるよう可搬性が実現された。

しかし、それぞれのプロセッサ毎にアーキテクチャの差異があるため、異種プロセッサ上で同一プログラムを動作させたとしても性能可搬性は保証されない。そのため、異なるプロセッサで高い性能を得るためにはそれぞれのプロセッサ毎に合わせた最適化を実装する必要があるが、ヘテロなメニーコアプロセッサがある環境では最適化を行わずに別のプロセッサを用いたほうが良いかもしれない。最適化が行われていないプログラムに対して現在のプロセッサに合わせた最適化を行った場合の実行時間や、別のプロセッサを用いた場合の実行時間を予測することができれば人的リソースを投入するかどうかの判断をすることが出来る。

そこで、それぞれの実行時間を予測するためにパフォーマンスカウンタからモデルを作成する。パフォーマンスカウンタからのモデルを作成することが出来れば、どのパフォーマンスカウンタが実行時間にどの程度影響しているかを知ることが出来る。そのパフォーマンスカウンタを元のプログラムから予測して得ることが出来れば、最適化されていないプログラムに対する指針を得ることが出来ると考えられる。

本論文では、異種GPUにおける最適化効果を定量的に予測し、パフォーマンスカウンタ

†1 東京工業大学
†2 科学技術振興機構
†3 国立情報学研究所

© 2010 Information Processing Society of Japan

から実行時間を予測するモデルを作成する。そのために、2 種類の GPU 上で 10 種類のプログラムを実行することによりパフォーマンスカウンタを取得する。そのパフォーマンスカウンタを回帰分析にかけることにより、パフォーマンスカウンタと実行時間の予測モデルを作成した。また、3 種類の GPU 上で 10 種類のプログラムの実行時間を取得して回帰分析を行うことで、別 GPU 上での実行時間予測モデルを作成した。そして、作成した予測モデルからパフォーマンスカウンタと実行時間の関係を調査した。

2. 背　　景

2.1 GPU

GPU(Graphics Processing Unit) は画像処理を担当するプロセッサである。画像処理計算を行うために GPU は高い並列処理能力を持っており、大量のデータを一度に処理することが出来る。GPU を開発している企業は様々あるが、NVIDIA 社[6] 製の GPU と AMD 社[1] 製の GPU について説明をする。

2.1.1 NVIDIA 社製 GPU

NVIDIA 社の提供している GPU は Compute Capability と呼ばれるグループに分かれており、Compute Capability 毎にアーキテクチャが異なっている。しかし、基本的なアーキテクチャの形は似ており、Compute Capability 1.x ではストリーミングプロセッサ (SP)、2.0 では CUDA コアと呼ばれるものが複数集まりストリーミングマルチプロセッサ (SM) を構成している。そして、SM が複数集まることで 1.x ではテクスチャプロセッサクラスタ (TPC)、2.0 ではグラフィックプロセッシングクラスタ (GPC) が構成され、TPC や GPC が集まることによって GPU が構成されている。

メモリに関しては、チップ内とチップ外にそれぞれ複数のメモリを持っている。チップ外のメモリは大容量だがアクセス遅延が大きくなってしまい、チップ内のメモリは容量は小さいが高速にアクセスできる特徴がある。Compute Capavility 2.0 ではこれらのメモリに加えてキャッシュが存在する。L1 キャッシュではチップ内の共有メモリと呼ばれるメモリとあわせて容量を変更することが出来るため、共有メモリ 16KB ＋ L1 キャッシュ48KB、共有メモリ 48KB ＋ L1 キャッシュ16KB の 2 通りの使い方が出来る。また、L2 キャッシュも存在しライトバックキャッシュになっている。

2.1.2 AMD 社製 GPU

AMD 製の GPU は SIMD エンジンと呼ばれるハードウェアブロックを単位として構成されている。例として、Radeon5870 では SIMD エンジンは 20 基搭載されており、SIMD エンジンには 16 基のスレッドプロセッサがある。スレッドプロセッサは単精度浮動小数点演算などを実行する 4 つの 32 ビット演算ユニットと超越関数演算を行う 1 つの演算ユニット、分岐制御用のユニットから構成されている。

また、メモリについては NVIDIA 社製の GPU と同様に GPU チップの外に大容量メモリと SIMD エンジン内にスレッドプロセッサ間で共有できるメモリがそれぞれ複数存在する。遅延時間も NVIDIA 社製の GPU と同じで、GPU チップの外側にあるメモリにアクセスしようとするとアクセス遅延時間が大きくなってしまい、SIMD エンジン内のメモリを用いると高速にアクセスできる。

AMD の GPU では同じ命令ストリームを wavefront と呼ばれるグループ単位で管理している。wavefront は 1 つの SIMD エンジンに割り当てられ、wavefront 内のスレッドがスレッドプロセッサで実行される。そのため、wavefront 内のスレッドの処理が分岐を含んでいると、分岐する場合と分岐しない場合の 2 つの処理を行う必要があり、プログラムの効率が低下してしまう。この概念は NVIDIA 社の CUDA で用いられる warp とほぼ同様である。

2.2 OpenCL

OpenCL はクロノス・グループ[4] によって策定されたヘテロジニアス並列計算環境のための共通フレームワークである。CPU と GPU、Cell/B.E などのヘテロジニアスな環境を考えて策定されている。したがって、OpenCL を用いてプログラムを作成することにより単一プログラムを異種 GPU 上でも CPU 上でも動作させることが可能になるため、プログラムに可搬性を持たせることが出来る。

また、OpenCL では、カーネル関数からアクセスできるメモリの種類を次の 4 種類に分割している。

- グローバルメモリ：全てのワークアイテムから読み書きできるメモリ領域
- コンスタントメモリ：全てのワークアイテムから読み込みだけ出来るメモリ領域
- ローカルメモリ：同一ワークグループ内のワークアイテム間で共有されるメモリ領域
- プライベートメモリ：ワークアイテムで使用するメモリ領域

このうち、グローバルメモリとコンスタントメモリはチップの外側、ローカルメモリとプライベートメモリはチップの内側に存在する。

2.3 最適化の必要性

OpenCL を用いることでプログラムに可搬性を持たせることが出来るが、各 GPU でアーキテクチャが異なることによる同一プログラムの性能がどのように変化するかを確かめる。

ⓒ 2010 Information Processing Society of Japan

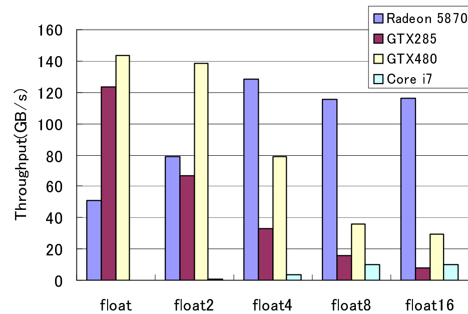

図 1 配列コピー

簡単な例として OpenCL C 言語で書かれた配列コピープログラムを 3 種類の GPU と 1 種類の CPU で実行する。

この配列プログラムは配列 A から配列 B に要素をコピーするプログラムで、ベクタ型のサイズを変更しながらコピーを行っている。ベクタ型を用いると演算や読み込み、書き出しをいくつかの要素をひとまとめにして行うことができる。OpenCL ではひとまとめに出来る要素数が 2,4,8,16 に限られている。このベクタ型を用いて、配列 A から n 要素読み込んで配列 B に n 要素書き込むことを行う。

実行結果を図 1 に示す。この実行結果を見ると、NVIDIA 社製の GTX280 と GTX485 では float を用いたときに最大スループットとなっているが、AMD 社製の Radeon5870 では float4 を用いたとき、CPU の Core i7 では float16 を用いたときにそれぞれ最大スループットとなっており、それぞれのデバイスで性能を最大にする手法が異なっている。また、GTX285 と GTX480 を比較しても性能変化が異なっている。したがって、この結果より配列コピーの方法を変更するだけで各 GPU で異なる性能変化が起こることが確認できた。そのため、OpenCL によって書かれたプログラムでは性能可搬性が保証されないことを確認することができた。したがって、性能向上を求める場合には最適化が必要となる。また、小松ら[5] は複数 GPU 上でブロックサイズを変更させながらプログラムの実行時間を測定しており、この論文からも OpenCL を用いたプログラムでは各プロセッサで異なる性能変化が起きていることが分かる。

表 1 パフォーマンスカウンタの種類

プログラム	Compute Capability	特徴
gld32b	1.x	グローバルメモリから 32 バイト読み込む回数 (他に gld64b,gld128b がある)
gst32b	1.x	グローバルメモリに 32 バイト書き込む回数 (他に gst64b,gst128b がある)
gstrequest	2.0	グローバルメモリ読み込み要求回数
gldrequest	2.0	グローバルメモリ書き込み回数
l1globalloadhit	2.0	グローバルメモリ読み込み時の L1 キャッシュヒット回数
l1globalloadmiss	2.0	グローバルメモリ読み込み時の L1 キャッシュミス回数
sharedload	2.0	マルチプロセッサ上での warp ごとの共有メモリ読み込み実行回数
sharedstore	2.0	マルチプロセッサ上での warp ごとの共有メモリ書き込み実行回数
branch	1.x,2.0	カーネルを実行しているスレッドによる分岐の数
instruction	1.x	実行された命令数
instruction executed	2.0	実行された命令数
active cycle	2.0	少なくとも一つアクティブな warp を持つマルチプロセッサのサイクル数
work groups per compute unit	1.x	TPC0 内の計算ユニットで実行したワークグループ数

3. 解析手法

GPU プログラムのモデル構築法には様々な種類があるが、本研究ではパフォーマンスカウンタを用いて回帰分析を行った。

3.1 パフォーマンスカウンタ

今回の研究では NVIDIA 社が提供している NVIDIA Visual Profiler を用いてパフォーマンスカウンタを用いて測定をした。NVIDIA Visual Profiler を用いることで測定できるパフォーマンスカウンタの一部を表 1 に記した。注意として、NVIDIA Visual Profiler を用いてカウンタを取得する場合、Compute Capability が異なると取得できるカウンタの種類が異なってしまう。そのため、GTX285 と GTX480 では取得できるカウンタが違うので、回帰分析に使用できるカウンタが異なってしまう。

3.2 回帰分析

実行時間予測モデル作成のために線形回帰分析を行う。カーネル関数を 1 回実行したときの実行時間を目的関数、そのときのパフォーマンスカウンタの値を説明変数として線形回帰分析にかける。すなわち、実行時間を T、カウンタの種類を n、パフォーマンスカウンタ値

© 2010 Information Processing Society of Japan

表 2 GPU の特徴

	GTX 285	GTX 480	Radeon 5870
CUDA Core	240	240	
Stream Core			1600
Warp Size	32	32	
Wavefront Size			64
BandWidth(GB/sec)	159.0	177.4	153.6
Cache(L1)	None	16KB〜48KB	texure cache(8KB)
Cache(L2)	None	768KB	512KB

表 3 マシンの特徴

	GTX 285	GTX 480	Radeon 5870
CPU	Intel Core i7 920		
OS	Ubuntu 9.10		
Platform	CUDA 3.1.1 OpenCL 1.0		ATI Stream v2.1 OpenCL 1.0
Driver	NVIDIA Driver 256.40		AMD Driver 10.6
GCC	4.4.1		

を p として次の式 (1) で表したときに、T を最も高い精度で予測できる a_i を求める。

$$T = a_0 + \sum_{i=1}^{n} a_i p_i \tag{1}$$

線形回帰分析を行った結果の精度を解析するために leave-one-out 手法をとる。この手法では、まず i 番目のサンプル以外のサンプルで線形回帰分析を行い、その結果を用いて i 番目のサンプルの実行時間を予測する。そして、予測値と実際のデータを比較し精度を確認する。この操作を全サンプルに対して行う。カウンタの値はサンプルによって桁が異なる場合があるため、平均が 0、分散が 1 になるよう正規化をした後に回帰分析を行う。

4. 実　　装

4.1 実験環境

今回の実験では、NVIDIA 社製 GeForce GTX285 と Geforce GTX480、AMD 社製 Radeon HD 5870 の 3 基の GPU を用いた。それぞれの GPU の特徴を表 2 に示す。また、使用したそれぞれのマシンの特徴を表 3 に示す。

4.2 実験対象

実験に使用したプログラムは NVIDIA 社が提供している GPU Computing SDK と ATI 社が提供している ATI Stream SDK から 10 個のサンプルプログラムを選び実行し、GTX285 と GTX480 でパフォーマンスカウンタを取得した。表 4 に一覧を示す。また、今回選んだサンプルプログラムはローカルメモリを用いるようにプログラムされているため、ローカルメモリを用いずに計算を行うようにソースコードを書き換え、ローカルメモリ有り、ローカルメモリ無しの 2 通りのパフォーマンスカウンタを取得し、回帰分析を行った。

表 4 実行プログラム

プログラム	SDK	略称	特徴
MatrixMul	NVIDIA	Matmul(n)	タイリングを用いた行列積
MatrixMultiplication	AMD	Matmul(a)	float4 を用いた行列積
DCT	AMD	DCT	離散コサイン変換
DwtHaar1D	AMD	Dwt	離散ウェーブレット変換
NBody	AMD	NBody	N 体問題
FDTD3d	NVIDIA	FDTD	電磁場解析
MedianFilter	NVIDIA	Median	メディアンフィルター
SobelFilter	NVIDIA	Sobel	そーベルフィルター
BinomialOption	AMD	Binomial	オプション価格
AESEncryptDecrypt	AMD	AES	AES 暗号生成

4.3 使用カウンタ

2 章で述べた GPU 毎に異なる点としてメモリアーキテクチャがある。そこで、回帰分析に用いるカウンタではグローバルメモリの読み込みと書き込みを用いる。また、命令数が多いほど実行時間が長くなると考えられるため、命令の発行数も用いる。さらに、分岐の数が多くなると実行時間も多くなってしまうと考えられるため分岐数も回帰分析に用いる。

GTX480 ではローカルメモリの読み込み数を参照できるため、ローカルメモリの読み込み数が多いほど元のプログラムからグローバルメモリを減らすことが出来ていると考えられるため回帰分析に用いる。

他に使用できるパフォーマンスカウンタがある場合はそのときに適したパフォーマンスカウンタを使用する。

5. 評　　価

5.1 GTX285

5.1.1 最適化無し

図 2 に leave-one-out による回帰分析結果から得られた実行時間予測と実際に測定された実行時間の比を示す。説明変数として用いたカウンタは gld32b、gld64b、gld128b、gst(gst32b、gst64b、gst128b)、branch、instruction である。gst は gst32b、gst64b、gst128b をまと

© 2010 Information Processing Society of Japan

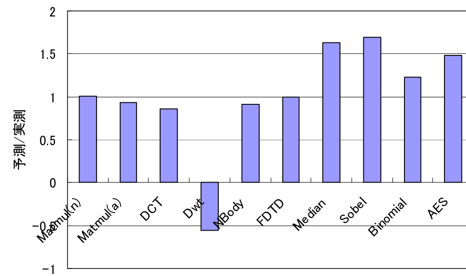

図 2　GTX285 最適化無し

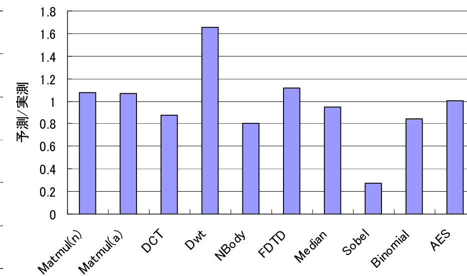

図 3　GTX285 最適化あり

めた値である。まとめずに回帰分析を行うと、ほとんどのプログラムでは gst32b、gst64b、gst128b のどれか一つにしか値が入っていないため予測結果が大きく外れてしまう。そのため、似た命令である gst を一つにまとめて回帰分析を行った。

平均誤差率は 38.9 %となっており、Dwt ではマイナスの予測時間となってしまった。この原因として gld32b のカウンタが考えられる。Dwt の gld32b は他のプログラムの平均に比べて約 1/25000 となっており、他のプログラムの gld32b と比較して小さすぎるため予測時間が小さくなりすぎていると考えられる。また、他の実行時間予測が大きくなっているプログラムを見ると、共通している点として実行時間が短く、さらに gld64b、gld128b のカウンタ値が 0 となっている。そのため、カウンタの種類が少なく正しい予測がしにくいため、このような結果になったと考えられる。

5.1.2　最適化あり

図 3 に leave-one-out による回帰分析結果から得られた実行時間予測と実際に測定された実行時間の比を示す。説明変数として用いたカウンタは gld(gld32b、gld64b、gld128b)、gst(gst32b、gst64b、gst128b)、branch、instruction、warp serialize、threadNum である。gld と gst をまとめた理由は最適化無しの場合と同じである。

平均誤差率は 21.8 %となっていた。この原因として、Sobel の予測実行時間が短いことと Dwt が大きいことが考えられる。Dwt、Sobel の理由として共に実行時間も短かったことが挙げられる。さらに、Sobel では instruction の値が他のプログラムに比べて小さいことの影響が出ていると思われる。

5.2　GTX480

5.2.1　最適化無し

図 4 に leave-one-out による回帰分析結果から得られた実行時間予測と実際に測定された実行時間の比を示す。説明変数として用いたカウンタは gld request、gst request、branch、instruction issued、l1 global load hit、l1 global load miss、active cycle である。

平均誤差率は 4.6 %となっており、GTX285 と異なって精度の良い予測となった。全体の回帰係数を見ると gld request や gst request、branch により予測実行時間が増していた。実行時間が遅くなる原因としてメモリアクセスによる遅延や分岐の存在があるため、それらが反映されていると考えられる。逆に l1 global load hit や active cycle では符号がマイナスだったため予測実行時間が減少していた。これらは、キャッシュヒットやワープごとの実行が行われていることを示すカウンタなので、それらの効果により予測実行時間を減少させていると考えられる。

5.2.2　最適化あり

図 5 に leave-one-out による回帰分析結果から得られた実行時間予測と実際に測定された実行時間の比を示す。説明変数として用いたカウンタは gld request、gst request、shared load、branch、instruction issued、l1 global load miss、active cycle である。

平均誤差率は 5.0 %となっていた。最適化無しと使用したカウンタを比較すると、最適化無しで用いていた l1 global load hit を shared load に変更しただけである。全体の回帰係数を見ると gld request や gst request は正負両方の係数があり、平均するとほぼ 0 になっていた。今回の回帰分析で最も実行時間を増やしているのは l1 global load miss であった。l1 global load miss が大きいと l2 キャッシュやグローバルメモリから読み込みを行わなくてはならないため、そのことが反映されたと考えられる。逆に、実行時間を減らしているカウンタは shared load と active cycle であった。これは、最適化無しの場合と似ており、l1 キャッシュを用いていた部分がローカルメモリに置き換わったものと考えられる。

5.3　他　GPU

5.3.1　GTX285 から GTX480 を予測

図 6 に leave-one-out による GTX285 のパフォーマンスカウンタから GTX480 の実行時間予測と実際に測定された実行時間の比を示す。説明変数として用いたカウンタは gld32b、gld64b、gld128b、branch、instruction、work groups per compute unit である。

平均誤差率は 22.0 %となっていた。理由として DCT の実行予測時間が大きく、Sobel の実行予測時間が短いことが原因と考えられる。DCT の実行時間が大きくなった理由として、

© 2010 Information Processing Society of Japan

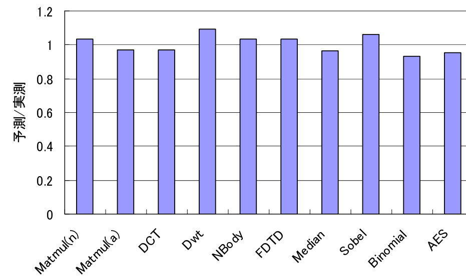

図 4 GTX480 最適化無し

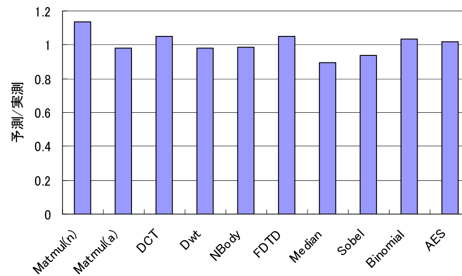

図 5 GTX480 最適化あり

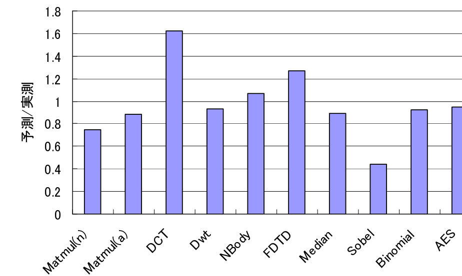

図 6 GTX480 予測

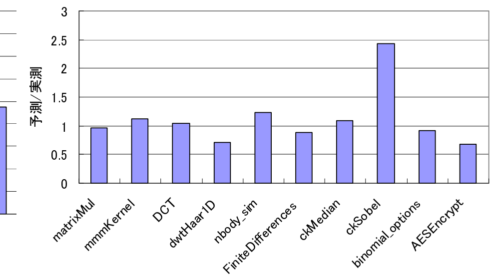

図 7 Radeon5870 予測

work groups per compute unit のカウンタが考えられる。DCT ではこのカウンタの値が他のプログラムの平均に比べて約 1/16 になっており、work groups per computeunit の回帰係数は正なので DCT の予測実行時間が大きくなったと考えられる。逆に、Sobel が短くなった理由として考えられるのは instruction が考えられる。Sobel の instruction のカウンタは他のプログラムの平均に比べて約 1/564 になっている。また、instruction の回帰係数は正なので予測実行時間が短くなったと考えられる。

5.3.2 GTX285 から Radeon5870 を予測

図 7 に leave-one-out による GTX285 のパフォーマンスカウンタから GTX480 の実行時間予測と実際に測定された実行時間の比を示す。説明変数として用いたカウンタは gld(gld32b、gld64b、gld128b)、gst(gst32b、gst64b、gst128b)、branch、instruction、divergent branch、GTX285time である。

平均誤差率は 27.5 %となっている。この理由として、Sobel の予測実行時間が大きいことが考えられる。Sobel の予測実行時間が大きくなった理由としては、実行時間が最も短いことが挙げられる。他のプログラムを用いて実行時間を予測しているため、実行時間が短い Sobel を予想するときには実行時間が Sobel よりも大きいプログラムを用いる。そのため、回帰分析の結果を用いても短い実行時間を予測することが難しいと考えられる。

6. 関 連 研 究

GPU の実行時間をモデル化により予測する研究が行われている。例えば、伊藤ら[7] はアルゴリズム開発において有用となる性能モデルを提案している。この性能モデルは主記憶、GPU メモリ、GPU の演算器間の転送バンド幅と転送遅延の組で作成されている。この性能モデルにより、プログラム作成前から実行時間を予測することができる。また、Hong ら[3] はワープ単位のメモリアクセスや実行に注目して性能モデルを構築している。この性能モデルは主にワープ単位実行時のメモリアクセスを考慮することによりモデル式を構築している。メモリ要求のコストを見積もることにより全体の実行時間を予測することが出来る。さらに、Baghsorkhi ら[2] はカーネル関数実行時のワークフローグラフを作成し、どのように実行されるかを考えながらモデル式の作成している。

7. 終 わ り に

7.1 ま と め

本研究ではパフォーマンスカウンタを用いてプログラムの実行時間予測のモデルを作成し、どのカウンタが実行時間に影響しているかを調査した。GTX285 では、最適化の有無に関わらず使用できるカウンタの値が少ないため、実行時間が短いプログラムの予想が上手くいかなかったが、それ以外のプログラムではある程度精度の良い予測が出来た。

一方、GTX480 では高精度な予測をすることが出来た。予測に用いたカウンタからメモリアクセス、l1 キャッシュミスなどが実行時間を増加させており、ワープ実行や l1 キャッシュヒット、ローカルメモリの使用により実行時間を減少させていることがわかった。

別 GPU の予測では、誤差が大きすぎるプログラムがあった。特に、Sobel は 2 つの分析で両方とも精度が良くなかった。これは実行時間やカウンタの値が小さすぎたためと考えられる。そのため、実行時間を長くしてカウンタの値を大きくすれば実行時間予測の精度が上がると考えられる。

© 2010 Information Processing Society of Japan

7.2 今後の課題

今回の実験ではパフォーマンスカウンタから実行時間の予測を行ったが、サンプル数が少ないため他の種類のプログラムについても回帰分析を行いたい。特に、GTX285 では使用できるカウンタの数が少ないためサンプル数を増やすことにより実行時間とカウンタの関係を把握したい。また、元のプログラムからパフォーマンスカウンタを予測する解析器を作成する。解析器を作成することにより、共有メモリを用いた最適化を行わずにパフォーマンスカウンタを予測することが出来れば始めに書いた人的リソースを投入して最適化を行うかどうかの指針を得ることができる。

謝　　辞

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「Ultra-Low-Powr HPC: 次世代テクノロジのモデル化・最適化による超低消費電力ハイパフォーマンスコンピューティング」, および NVIDIA CUDA Center of Excellence による。

参 考 文 献

1) AMD. http://www.amd.com/jp/Pages/AMDHomePage.aspx.
2) Sara S. baghsorkhi, Matthieu Delahaye, Sanjay J.Patel, William D. Gropp, and Wen mei W.Hwu. An Adaptive Performance Modeling Tool for GPU Architecture. In *PPoPP*, pp. 105–114, 2010.
3) Sunpyo Hong and Hyesoon Kim. An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness. In *ISCA*, pp. 152–163, 2009.
4) KHRONOS. http://www.khronos.jp.
5) Kazuhiko Komatsu, Katsuto Sato, Yusuke Arai, Kentaro Koyama, Hiroyuki Takizawa, and Hitoaki Kobayashi. Evaluating Performance and Portability of OpenCL Programs. In *The Fifth International Workshop on Automatic Performace Tuning*, 2010.
6) NVIDIA. http://www.nvidia.co.jp/page/home.html.
7) 伊藤信悟, 伊野文彦, 萩原兼一. GPGPU アプリケーションの開発を支援するための性能モデル. 情報処理学会論文誌コンピューティングシステム, 第 48 巻, pp. 235–246, 2007.
8) 遠藤敏夫. 東京工業大学 TSUBAME におけるアクセラレータ活用事例. 情報処理, 第 50 巻, pp. 100–106, 2009.

© 2010 Information Processing Society of Japan