GPUの共有メモリアクセスのレイテンシとスループットの計測

岡本 悟史
広島大学大学院工学研究科

概 要

グラフィックス処理を行うためのデバイスであるGPUの演算能力を汎用演算に応用する技術をGPGPUと呼び，近年活発に研究が進められている．NVIDIA社のGPUには複数の演算コアで共有可能で尚且つアクセスレイテンシの非常に小さなオンチップ共有メモリが存在し，GPGPUによる処理の高速化において重要な役割を担っている．しかしこの共有メモリはバンクコンフリクトの発生等の要因によってメモリアクセスのレイテンシ，及びスループットが大きく変動する特徴がある．

本研究では，GPUのクロックカウンタを利用して共有メモリアクセス時間の計測を行い，バンクコンフリクト，メモリアクセス命令の実行回数，実行時のスレッド数が共有メモリのレイテンシ，スループットにどのような影響を与えるかを明らかにする．NVIDIA GeForce GTX780Tiを用いて共有メモリアクセス命令の実行クロックサイクル数を計測することで，最適なアクセスを行った際のレイテンシが16clock cycleであるという結果が得られた．また，バンクコンフリクト，メモリアクセス命令数の実行回数，実行時のスレッド数とメモリアクセスに必要な実行時間の関係を表す回帰式も得ることが出来た．

1. はじめに

複数のプロセッサが同時に処理を行うことで演算スループットを向上させる並列計算は，高い計算性能を実現する方法として非常に有効である．GPU(Graphics Processing Unit) は内部に搭載した多数の演算コア，高いメモリバンド幅から並列計算への応用が盛んに行われ，HPC(High Performance Computing) 分野で注目を集めている．本来グラフィックス処理を行うためのデバイスであるGPUを汎用演算に用いるこの技術はGPGPU(General-Purpose computing on GPUs)と呼ばれている．NVIDIA社のGPUでは同社の提供するGPU向け統合開発環境CUDA(Compute Unified Device Architecture)[1]を用いることで並列処理を実装することが出来る．

NVIDIA社のGPUは並列処理に用いる大量のデータを効率良く扱うために図1のようなメモリ階層構造を取り入れている．CUDAを用いたGPGPUを実装する上で重要となるのがグローバルメモリ，シェアードメモリの二つのメモリである．グローバルメモリはオフチップDRAM上にあり，数GBと容量が非常に多いメモリである．しかしグローバルメモリはオフチップメモリであるためレイテンシが非常に大きいという欠点がある．これに対してシェアードメモリはオンチップ共有メモリであり，容量が小さいがオフチップメモリに比べてアクセスが非常に早く，GPGPUによる処理の高速化において重要な役割を担っている．

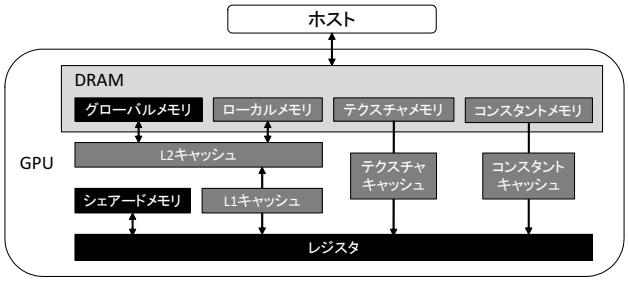


図 1: GPUのメモリ階層構造

しかし，シェアードメモリのレイテンシ及びスループットはアクセスのパターン等によって大きく変動してしまう．NVIDIA社はグローバルメモリやシェアードメモリ等のアクセスの特徴をCUDA Programming Guide[2]に記載しているが具体的な数値による詳細な説明はされていない．そのため，GPUのベンチマークを行うことでメモリスループット等を計測する研究[3, 4]や，GPUのシミュレータを用いたアーキテクチャの分析[5]等が進められている．

本研究では，NVIDIA社のGPUであるGeForce 780Tiを用いたシェアードメモリアクセス時間の測定を行うことによってレイテンシ，及びスループットを明らかにすることを目的とする．

2. GPU アーキテクチャ

2.1. ハードウェアアーキテクチャ

GPUのハードウェアアーキテクチャを図2に示す．GPUは複数のSM(Streaming Multiprocessor)を持ち，また一つのSM内には複数の演算コアが存在する．オンチップ共有メモリであるシェアードメモリは各SM上に存在し，SM内の演算コア間で共有することができる．表1は本研究に用いるGeForce GTX 780TiについてNVIDIA社によって公開されている性能をまとめた表である．グローバルメモリが置かれるDRAMは3072MBであるのに対してシェアードメモリは各SM当たり16から48KBと非常に容量が少ないことが読み取れる．

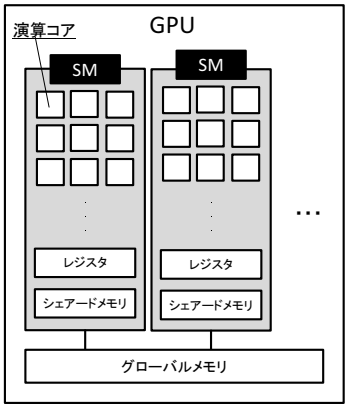


図 2: ハードウェアアーキテクチャ

表 1: GeForce GTX 780Ti の性能一覧

GPU	
コアアーキテクチャ	GK110
ベースクロック	875MHz
ブーストクロック	928MHz
演算コア数	2880
SMX 数	15 基
DRAM サイズ	3GB
SMX	
SMX 当たりの演算コア数	192
レジスタ数	65536
シェアードメモリ	16KB or 32KB or 48KB

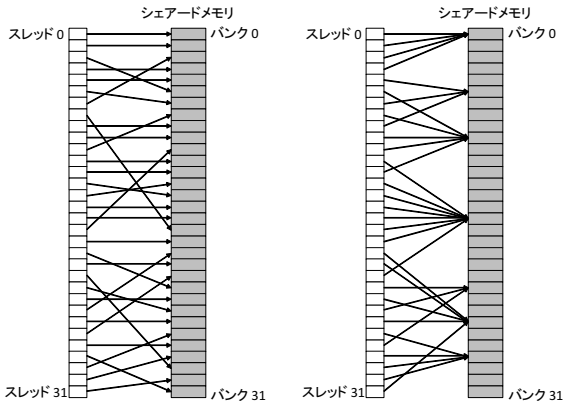
2.2. CUDA

NVIDIA 社の GPU で汎用並列計算を行うための統合開発環境として CUDA が提供されている．CUDA はスレッドレベル並列性によって並列計算を実現する．複数の演算コアで各スレッドが同じ命令を実行し，全体のスループットを向上させている．32 個のスレッドをまとめたものはワープと呼ばれ，ワープ内のスレッドは同時に同じ命令を実行する．

2.3. バンクコンフリクト

シェアードメモリアクセスもワープ単位で実行されるため，ワープ内のシェアードメモリアクセスは同時に実行される．しかし，アクセスのパターンによってはアクセスが逐次化され，シェアードメモリアクセスを完了するまでの時間が長くなってしまう可能性がある．

シェアードメモリはバンクと呼ばれる単位に区切られ，バンクそれぞれに 0〜31 の番号が割り当てられる．ワープ内のスレッドが同時にアクセスを行う際にそれらのバンク番号が全て異なっていれば，一度に全てのデータにアクセスできる (図 3(a))．しかしバンク番号が異なる場合，アクセスが逐次化されることで速度が非常に遅くなってしまう (図 3(b))．このような現象をバンクコンフリクトと呼び，処理の高速化のためにはバンクコンフリクトを出来る限り回避することが重要である．



(a) バンクコンフリクトが発生しないアクセス　(b) バンクコンフリクトが発生するアクセス

図 3: シェアードメモリとバンクコンフリクト

3. 計測方法

シェアードメモリは SM 毎に搭載されているため，計測には一つの SM のみを使用する．ある一つの SM におけるシェアードメモリアクセスの挙動を知ることが出来れば，他の SM も同じ用に振る舞うため GPU 全体での動きも予想することが出来る．

3.1. PTX(Parallel Thread Execution)

シェアードメモリアクセスのみの実行時間を計測するために，CUDA のアセンブリ言語である PTX(Parallel Thread Execution) を使用した．PTX を用いることで，より機械語に近い命令を直接記述することが出来る．double 型 64bit のシェアードメモリ読み込みを行うための命令はソースコード 1 のように記述できるようになる．なおソースコード中の reg は格納先のレジスタ，addr はアクセスするアドレスを格納したレジスタである．

ソースコード 1: シェアードメモリ読み込み命令

```
ld.shared.f64 reg, [addr]
```

3.2. クロックカウンタ

時間計測には GPU のクロックカウンタを利用した．ソースコード 2 のように，PTX の mov 命令を使用し，レジスタ%clock から値を取得することでクロックカウンタの値が得られる．

ソースコード 2: クロックカウンタの値を取得する命令

```
mov.u32 reg, %clock
```

ソースコード 3 は CUDA プログラム中に PTX 命令を記述できるインライン PTX を利用した計測用のソースコードである．処理内容は，まずスレッド間同期を行う bar.sync 命令で全スレッドが同時に計測を始められるようにする．そしてメモリアクセス前のクロックをクロックカウンタから取得し，シェアードメ

モリアクセスを行う．最後に再びクロックカウンタから処理後のクロックを取得する．

ソースコード 3: 実行クロックサイクル数の計測

```
asm volatile {
  "bar.sync 0;\n\t"
  "mov.u32 %0, %%clock;\n\t"
  "ld.volatile.shared.f64 %2, [%%a0];\n\t"
  "mov.u32 %1, %%clock;\n\t"
  :"=r"(begin), "=r"(end), "=l"(reg)
}
dummy[0] = reg;
```

こうして処理前後のクロックを取得した後，それらの差分を取ることでシェアードメモリアクセスに必要な実行クロックサイクル数を求めることができる．

ソースコード 3 からシェアードメモリアクセスを取り除き，クロックの取得が二回並ぶようなコードで計測を行い，差分を取ると 16clock cycle という結果が得られる．これは mov 命令でクロックカウンタの値を取得するのに必要なレイテンシと考えられるため，計測したシェアードメモリアクセスの実行クロックサイクル数からは引いておく必要がある．

なお，ソースコード 3 でメモリアクセス命令に volatile が指定されているのは PTX コードをコンパイルする際に，コンパイラによる最適化でメモリアクセスが消されてしまわないようにするためである．また計測後にシェアードメモリアクセスに用いたレジスタの値をメモリに書き込むことで依存関係を作っておくことも同様の理由で必要である．

3.3. 計測条件

クロックカウンタの取得によるシェアードメモリアクセス実行クロックサイクル数の計測を行う上で，レイテンシ，スループットに影響を与える三つの条件を考慮する必要がある．

一つ目は実行時のワープ数である．ワープ数が増加するとシェアードメモリアクセス命令の発行数が増加し，それだけアクセスを終えるまでのクロックサイクル数が増加する．クロックカウンタの取得はワープ単位で実行されるため，そのワープがメモリアクセスを開始したタイミング，及びメモリアクセスが完了したタイミングで個別にクロックの取得が行われる．アクセスを開始してから全てのワープがアクセスを終えるまでの時間を導出するため，図 4 のように処理前は最小値，処理後は最大値を求めた差を使用する．SM 当たりの最大スレッド数は 1024 スレッドであるため，ワープ数は 32 まで増加させて計測を行うことが出来る．

二つ目はメモリアクセス命令の実行回数である．ワープが実行するシェアードメモリアクセス命令が増えることで同様にクロックサイクル数が増加する．ソースコード 4 は命令実行回数が三回の場合のものである．格納先レジスタに別のレジスタを指定しているのは，コンパイラによる最適化で命令が一度しか実行されなくなるのを防ぐためである．

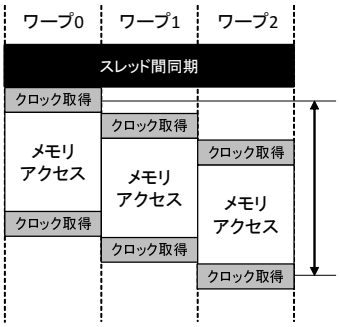


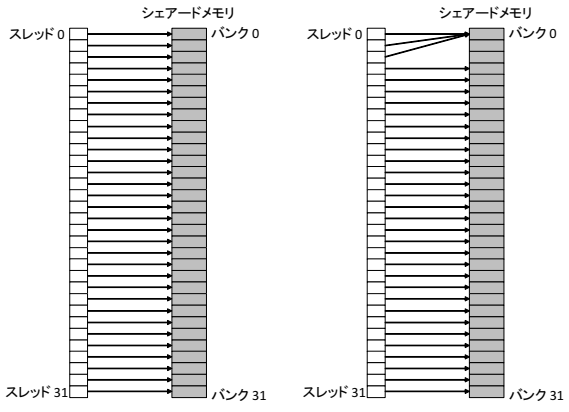
図 4: 複数ワープにおける実行クロックサイクル数

ソースコード 4: 命令実行回数が 3 回のときの測定

```
asm volatile {
  "bar.sync 0;\n\t"
  "mov.u32 %0, %%clock;\n\t"
  "ld.volatile.shared.f64 %2, [%%a0];\n\t"
  "ld.volatile.shared.f64 %3, [%%a0];\n\t"
  "ld.volatile.shared.f64 %4, [%%a0];\n\t"
  "mov.u32 %1, %%clock;\n\t"
  :"=r"(begin), "=r"(end), "=l"(reg0), "=l"(reg1),
       "=l"(reg2)
}
```

三つ目はバンクコンフリクトである．2.3 節で説明したように，バンクコンフリクトが生じるとアクセスが逐次化されクロックサイクル数が増加する．ここで，各バンクに同時にアクセスするスレッド数の最大値を congestion と定義する．バンクコンフリクトとシェアードメモリアクセス時間の関係は congestion の大きさに応じて決まる．図 5(a) はバンクコンフリクトが発生しないアクセスであり，各バンクへのアクセス数は全て 1 であるため，congestion は 1 となる．一方，図 5(b) はバンク 0 に対して 3 つのスレッドのアクセスが集中している．そのため congestion は 3 となり，メモリアクセスの逐次化によってアクセスの完了までにかかる時間が長くなってしまう．

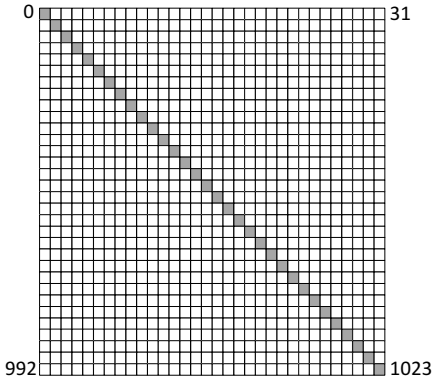
本研究ではバンク 0 に同時にアクセスするスレッド数を変化させることで congestion を操作しつつ計測を行う．しかし，シェアードメモリが同じアドレスにアクセスを行った場合，バンクコンフリクトは発生せずにアクセスがブロードキャストされる．そのため 1024 要素のシェアードメモリを用いて次のように congestion の操作を行う．図 6(a) は congestion が 1 のときのアクセスインデックスを表した図である．シェアードメモリの要素を，左上のインデックスが 0 として 32 要素ずつ行方向に並べ，アクセスするインデックスは灰色で示している．バンク番号は 32 要素ずつ割り当てられるため列方向に並んでいるメモリは全て同じバンクへのアクセスとなり，各列で一度しかアクセスが行われていない場合はバンクコンフリクトが発生しない．左上から対角線を引くようにインデッ



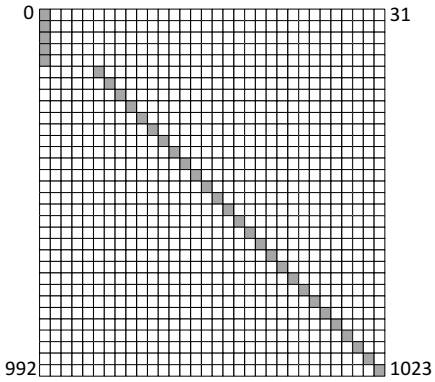
(a) congestion=1　　(b) congestion=3

図 5: シェアードメモリと congestion

クスを決定すると，列方向にアクセスが並ばず，バンクコンフリクトは発生しない．図 6(b) は congestion が 5 のときのアクセスインデックスを表している．5 つのスレッドのインデックスを左端の列，すなわちバンク 0 に集めることで congestion を大きくする．



(a) congestion=1



(b) congestion=5

図 6: アクセスインデックスと congestion

4. 計測結果

メモリアクセス時間の計測に用いた環境を表 2 に示す．得られた実行クロックサイクル数は 100 回計測を繰り返し行った平均を使用している．

表 2: 計測環境

GPU	GeForce GTX 780Ti
CUDA 環境	CUDA 5.5
計測する命令	ld.shared(シェアードメモリ読み込み)
アクセスサイズ	double 型 (64bit)

4.1. レイテンシの計測

命令実行回数，ワープ数，congestion 全てが 1 のとき，計測結果はシェアードメモリアクセスの最小レイテンシとなる．

実際に計測を行ったところ，16clock cycle という結果が得られた．グローバルメモリのレイテンシが 300〜400clock cycle であることと比較すると，バンクコンフリクトを考慮したシェアードメモリアクセスが非常に高速であることが分かる．

4.2. スループットの計測

シェアードメモリアクセスの実行クロックサイクル数に影響を与える命令実行回数，ワープ数，congestion を 3.3 節で説明したように変化させながら計測を行う．

シェアードメモリアクセスの条件は表 3 の範囲で計測を行った．

表 3: 計測条件

命令実行回数	1〜32
ワープ数	1〜32
congestion	1〜32

ここで命令実行回数を c，実行時のワープ数を w，congestion を c とおくことで，実行クロックサイクル数 $T(i,w,c)$ が定数 C_1 と C_2 を用いて式 (1) で表されると仮定する．

$$T(i,w,c) = C_1 \times iwc + C_2 \tag{1}$$

実際に計測を行い取得したデータを iwc と実行クロックサイクル数 T についてプロットを行ったグラフが図 7 である．

iwc と T がほぼ線形関係にあるため，計測結果に対して最小自乗法を用いることで線形回帰を行う．計測で得られた全データに対して最小自乗法による線形回帰を行うと，式 (2) が得られる．図 7 の赤色の直線は，この回帰式をプロットしたものである．線形回帰によって得られたこの式を用いることで，シェアードメモリアクセスの命令実行回数，ワープ数，congestion の値から実際にアクセスに必要な実行クロックサイクル数を予測することが出来る．

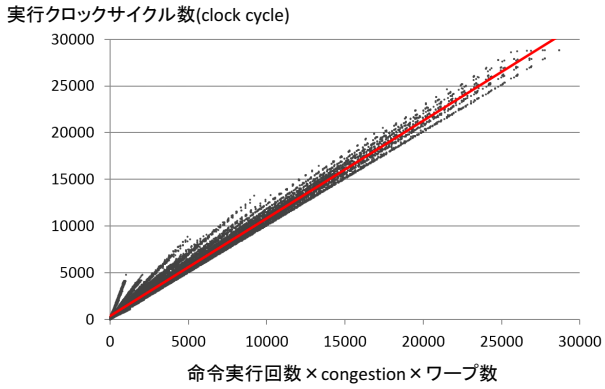


図 7: iwc と実行クロックサイクル数の関係と線形回帰式

$$T(i, w, c) = 1.047 \times iwc + 337.698 \qquad (2)$$

5. まとめ

NVIDIA 社の GPU に搭載されたオンチップ共有メモリであるシェアードメモリの実行クロックサイクル数を計測し，命令の実行回数，ワープ数，バンクコンフリクトとの関係を確認した．

GeForce GTX780Ti 上での計測の結果，バンクコンフリクトの生じない最適なアクセスにおけるレイテンシは 16clock cycle であることが分かった．また，様々な条件で計測したデータを用いて最小自乗法による線形回帰を行うことで，バンクコンフリクト，メモリアクセス命令数の実行回数，実行時のスレッド数とメモリアクセスに必要な実行時間の関係を表す回帰式を得ることが出来た．得られた式を用いることで，実際にアクセスに必要な実行クロックサイクル数を予測することが出来る．

参考文献

[1] NVIDIA. CUDA ZONE. http://www.nvidia.com/page/home.html.

[2] NVIDIA. Programming Guide :: CUDA Toolkit Documentation. http://docs.nvidia.com/cuda/cuda-c-programming-guide

[3] Wong, H. Demystifying GPU microarchitecture through microbenchmarking. Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on, pp.235-246, 28-30 March 2010

[4] Vasily Volkov, James W. Demmel. Benchmarking GPUs to Tune Dense Linear Algebra. SC '08 Proceedings of the 2008 ACM/IEEE conference on Supercomputing Article, No.31

[5] Ali Bakhoda, George L. Yuan, Wilson W.L. Fung, Henry Wong and Tor M. Aamondt: Analyzing CUDA Workloads Using a Detailed GPU Simulator, 2009 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).