Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA

著者： Shane Ryooy , Christopher I. Rodriguesy , Sara S. Baghsorkhiy , Sam S. Stoney , David B.Kirk , Wen-mei W. Hwuy

出典： *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, Pages 73-82, 2008*

発表者： 本多研究室 0953009 佐々木 信

1 はじめに

GPU は最近多くのアプリケーション開発者に便利なデータ並行コプロセッサとして注目されている。最新世代の GPU は、従来の GPU より簡単なプログラミング能力と、計算力、かなりの記憶帯域を維持しつつ汎用性を増加している。これにより、近年、GPU で汎用計算を行う GPGPU が注目されている。本研究では、GPU のひとつである GeForce8800 GTX プロセッサの構造、特徴、そして一般的な最適化の戦略について議論する。

2 アーキテクテャの概要

本研究では CUDA プログラミングモデルを用いる。CUDA(Compute Unified Device Architecture) は、NVIDIA が、最新世代の GPU のために作ったプログラミングモデルである [1]。CUDA は C 言語を拡張したもので、これまでのプログラミングモデルより簡単に習得できる。

2.1 マイクロアーキテクチャ

図 1 は GeForce8800 のマイクロアーキテクチャを表している。次のような構成となっている。

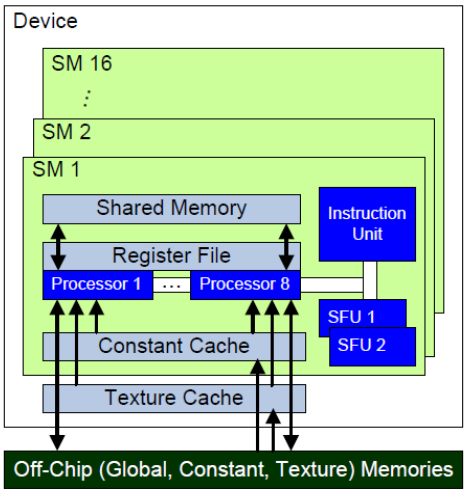

図 1: GeForce 8800 の基本構成

- ストリーミングマルチプロセッサ (SM):16 個
  - ストリーミングプロセッサ (SP):8 個/SM
  - 特別な機能ユニット (SFU):2 個/SM
  - レジスタ:8192 個/SM

各々のコアは SIMD(single-instruction, multiple-data) 方式で単一スレッドの命令を実行する。そして、GPU のピーク理論値は 388.8GLOPS(16SMs*18FLOPS/SM*1.35GHz) である。また、オンチップメモリの帯域は 86.4GB/s と非常に高いが、メモリ帯域は短い期間に多くのスレッドがアクセスを要求する場合、飽和することがある。

各メモリの遅れ時間は、グローバルメモリとローカルメモリは 200〜300 サイクル、共有メモリは数サイクル、コンスタントキャッシュは数サイクル、テクスチャキャッシュは 100 サイクル以上である。

実行中、ブロック内のスレッドは 32 個の並列スレッドのワープにグループ化される。ワープは粗いマルチスレッディングスケジューリングユニットであり、スレッドブロック内のスレッドの連続した領域で構成されている。

SM は、ワープをインターリーブし、グローバルメモリへのアクセスの遅れ時間と演算オペレーションの長い遅れ時間を隠すために、ゼロオーバーヘッドスケジューリングを実行できる。ひとつのワープがもたつくとき、SM はすぐに利用可能状態の他のワープにスイッチできる。もし処理対象の値が準備できていないならば、SM はもたつく。

要約すると、GeForce8800 上でアプリケーションを実行するためには、メモリ、スレッド、そして利用可能なトータルの帯域に対してきつい制限がある。これらの制限を管理することはアプリケーションの最適化において重要である。しかし、一つの制限を回避するための戦略は他の制限を引き起こす可能性がある。また同時に実行できるスレッドブロックの数も減らす可能性がある。

2.2 スレッディングモデル

GPU はデータ並列カーネルコードを実行するコプロセッサとして扱われている。ユーザーは、ホスト (CPU) とカーネル (GPU) コードの両方を含んでいる一つのソースプログラムを作成する。これれは、図 2 に示すように、分離されコンパイルされる。各々の CUDA プログラムは、CPU か GPU 上で実行される複数のフェーズで構成されている。ほとんどデータ並列処理を示していないフェーズはホスト (CPU) コードで実行される。それは、ANSI C

で表現されていて、図 2 で示しているようにホスト C コンパイラでコンパイルされる。

豊富なデータ並列処理を示しているフェーズはカーネル関数としてデバイス (GPU) コードにおいて実行される。カーネル関数は、データ並列フェーズのために呼び出される大量の数のスレッドの各々によって実行されるコードを定義する。これらのカーネル関数は NVIDIA CUDA C コンパイラとカーネル GPU オブジェクトコードジェネレータによってコンパイルさる。

カーネル関数にはいくつかの制限がある：再帰があってはならない、静的変数宣言があってはならない、そして、引数は非可変数でなければならない。ホストコードはデータを API の呼出しを用いて GPU のグローバルメモリへ転送したり、グローバルメモリから転送したりする。カーネルコードは関数呼び出しの実行によって開始される。

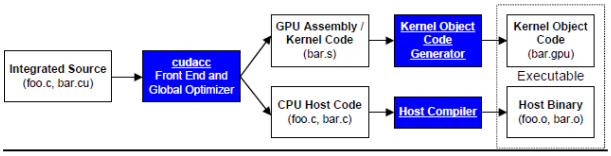

図 2: CUDA のコンパイルの流れ

3 性能と最適化

アプリケーション最適化を行う際、次のような 3 つの基本的な原則がある。

- アプリケーションの浮動小数点スループットは、浮動小数点オペレーションの割合に依存する。
- アプリケーションの最大パフォーマンスを達成するには、グローバルメモリの遅れ時間の管理が重要である。
- グローバルメモリ帯域はスループットを制限する可能性がある。

これらの原則を説明するために、行列乗算カーネルを用いる。図 3 は行列乗算カーネルの異なるタイルサイズによる実験の結果を示している。アンローリング 16 × 16 タイルバージョンは 91.14GFLOPS を達成する。これは、NVIDIA により高最適化された CUDA0.8 ライブラリによるパフォーマンスと同程度である。

4 アプリケーション研究

我々は、実際のアプリケーション上で前章での原則の効果と適用性をテストする目的で、アプリケーションの研究を行う。

図 4 は、最適化されたアプリケーションの実行の特徴を示している。ベースとなるシングルスレッドＣＰＵの

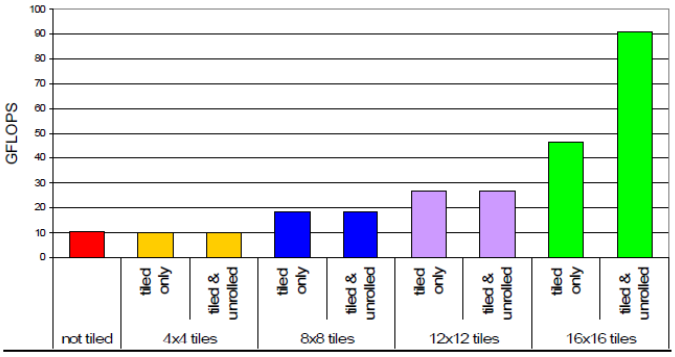

図 3: 行列乗算カーネルのパフォーマンス

パフォーマンスは、1GB のメインメモリを持った 2.2GHz の Opteron248 システムで測定した。

グローバルメモリアクセスが比較的少ない計算集約型カーネルは非常に高いパフォーマンスを達成する。計算集約型でないカーネルでさえ、かなりのパフォーマンス増加を達成する。それは、同時に多数のスレッドを実行する GeForce8800 の能力のためである。

MRI アプリケーションは著しく高いパフォーマンスを達成する。そのパフォーマンスの大きな理由は「相当数のオペレーションが三角法機能である」ということである。SFU はこれらを CPU より非常に速く実行できる。

Application	Max Simultaneously Active Threads	Registers per Thread	Shared Mem per Thread (B)	Global Memory to Computation Cycles Ratio	GPU Exec %	CPU-GPU Transfer %	Architectural Bottleneck(s)	Kernel Speedup on GPU	Application Speedup
Mat Mul	12288	9	8.1	0.276	16.2%	4%	Instruction issue	7.0X	2.0X
H.264	3936	30	55.1	0.006	2.6%	4.5%	Register file capacity and cache latencies	20.2X	1.47X
LBM	3200	32	84.2	0.415	98.3%	0.4%	Shared memory capacity	12.5X	12.3X
RC5-72	3072	42	0.3	~0	64.3%	0.5%	Instruction issue	17.1X	11.0X
FEM	4096	18	61	1.135	91.4%	≪ 1%	Global memory bandwidth	11.0X	10.1X
RPES	4096	23	24.8	0.01	37.5%	1%	Instruction issue	210X	79.4X
PNS	2048	32	9.9	0.241	98%	≪ 1%	Global memory capacity	24.0X	23.7X
SAXPY	12288	7	0.3	0.375	88%	4.5%	Global memory bandwidth	19.4X	11.8X
TPACF	4096	24	52.2	0.0002	34.3%	≪ 1%	Shared memory capacity	60.2X	21.6X
FDTD	12288	11	8.1	0.516	1.8%	0.9%	Global memory bandwidth	10.5X	1.16X
MRI-Q	8192	11	20.1	0.008	> 99%	≪ 1%	Instruction issue	457X	431X
MRI-FHD	8192	12	20.1	0.006	99%	1%	Instruction issue	316X	263X
CP	6144	20	0.4	0.0005	> 99%	≪ 1%	Instruction issue	102X	102X

図 4: 最適化後のアプリケーションのパフォーマンス

5 結論

管理する資源は、レジスタの数と、スレッド毎に使われるオンチップメモリの量、マルチプロセッサ毎のスレッドの数、そしてグローバルメモリ帯域である。

我々は、いろいろなアプリケーションの領域わたってこれらの戦略を適用し、カーネルコードにおいて 10.5 倍から 457 倍のスピードアップを成し遂げた。また、1.16 倍から 431 倍のトータルのスピードアップを成し遂げた。

参考文献

[1] CUDA テクニカルトレーニング Vol.1,NVIDIA
http://www.nvidia.co.jp/docs/IO/59373/VolumeI.pdf