C-017

GPUを用いた画像処理の並列プログラミング

Parallel Programming for GPU-Based Image Processing

田胡 匡基†
Masaki Tago

ハシタ ムトゥマラ ウィシディスーリヤ†
Hasitha Muthumala Waidyasooriya

張山 昌論†
Masanori Hariyama

亀山 充隆†
Michitaka Kameyama

1. はじめに

3D グラフィックを扱うゲームや 3D CAD などの処理を高速に行うためのプロセッサとして GPU(Graphic Processing Unit)と呼ばれるものがある．GPU はグラフィックに特化し，進化をしてきたため，高い並列性を持っているのが特徴である．この GPU に 3D グラフィックス以外の汎用的な計算を行わせる GPGPU (General Purpose Computation on Graphics Pocessing Unit)，GPU コンピューティングと呼ばれる手法がここ数年，注目を集めている．

NVIDIA 社より提供されている CUDA と呼ばれる開発環境は C 言語を拡張したものとなっており，従来よりも GPU へユーザプログラムをマッピングしやすいのが特徴である．しかしながら GPU は並列性を重視したアーキテクチャを採用しているため，ハードウェアを意識したプログラミングが必要となる．また，GPU では数種類のメモリを有するため，メモリの組み合わせ最適化が重要となる．本稿では，テンプレートマッチングの例を通して，メモリの組み合わせによる処理速度の影響を評価する．

2．GPU アーキテクチャ

図 1 に GPU のアーキテクチャを示す．SP はストリームプロセッサ，Reg はレジスタである．モデルによって異なるが，デバイス内には 2～30 個のマルチプロセッサが入っており，マルチプロセッサはさらに 8 個のストリームプロセッサで構成されている．8 個のストリームプロセッサは SIMD 型プロセッサとなっており，異なるデータに対して同一命令を実行する．デバイス外部に大容量・低速のデバイスメモリを有し，デバイスメモリはグローバルメモリ，コンスタントメモリ，テクスチャメモリに分類される．デバイスとホスト間でデータの転送を行う際は，グローバルメモリを介する必要がある．デバイス内部には高速・小容量のレジスタ，シェアードメモリ，コンスタントキャッシュ，テクスチャキャッシュを有する．これらのメモリの比較を表 1 に示す．グローバルメモリは容量が大きく，画像データなどの保存先として利用される．しかしながら，キャッシュ機構を持たないレイテンシの大きいメモリである．そこで，グローバルメモリと比べて数百倍高速なシェアードメモリをキャッシュのように利用することでグローバルメモリへのアクセス回数を削減し，処理全体を高速化できる．テクスチャメモリはキャッシュ機構を持つレイテンシの小さいメモリであり，フィルタリングを行う専用ハードウェアを有し，線形フィルタや正規化などが実行可能である．コンスタントメモリもキャッシュ機構を持ち，高速なアクセスが可能である．これら 2 つのメモリは GPU 側からはリードのみとなっており，参照用のメモリとなる．

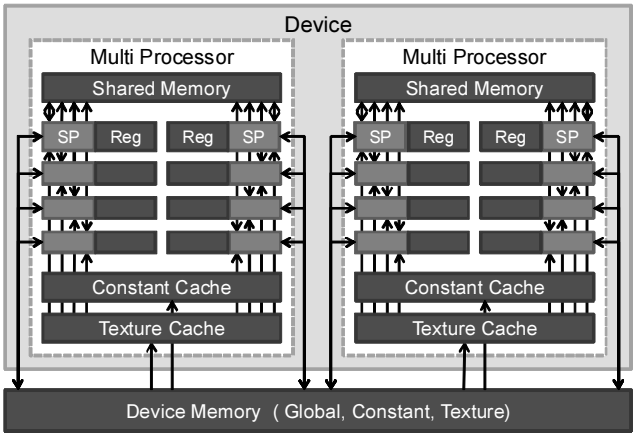

図 1　GPU アーキテクチャ

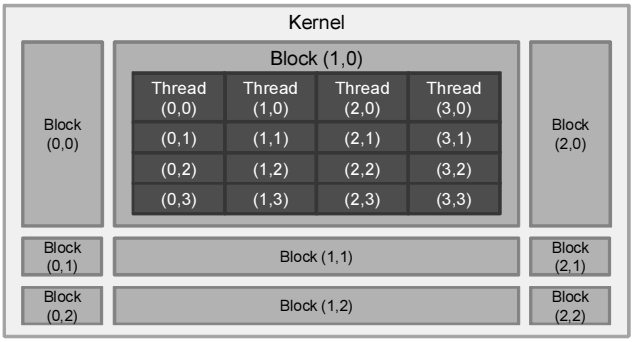

図 2　カーネル，ブロック，スレッドの分割

プログラムの分割について述べる．CUDA を用いたプログラムでは，全体の処理はカーネルと呼ばれる単位に分割され，実行される．そのカーネルはブロックという単位に分割され，それぞれがマルチプロセッサに割り当られる．ブロックはさらにワープと呼ばれる 32 スレッド単位に分割され，8 個のストリームプロセッサがワープを 4 サイクルで実行する．図 2 に分割の例として，カーネルをブロック 3×3 へ，1 つのブロックをスレッド 4×4 へ分割する場合の例を示す．また，シェアードメモリはブロック単位での利用となり，異なるブロック間ではデータの共有ができないため，一度グローバルメモリを通してやりとりを行うことになる．

† 東北大学大学院情報科学研究科　Graduate School of Information Sciences, Tohoku University

表 1　メモリ比較

	Global Memory	Texture Memory	Constant Memory	Shared Memory
容量	512MB ~ 1.8GB	CUDA array 1D : 2^{13} 2D : $2^{16} \times 2^{15}$ 3D: $2^{11} \times 2^{11} \times 2^{11}$	64KB/マルチプロセッサ	16KB/マルチプロセッサ
レイテンシ	初回 400~600 cycle	最大でレジスタと同等	最大でレジスタと同等	1 バンク　32bit/2cycle
GPU 側からのアクセス	Read/Write	Read only	Read only	Read/Write
キャッシュ	なし	Texture Cache あり	Constant Cache あり	－

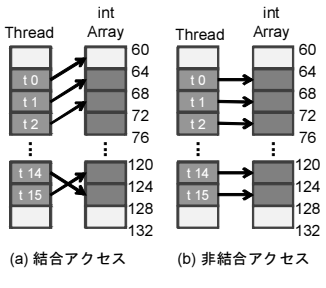

図 3　グローバルメモリ結合アクセスの例

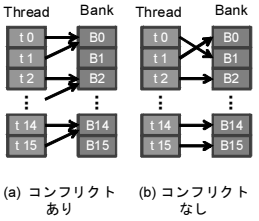

図 4　シェアードメモリバンクコンフリクトの例

表 2　グローバルメモリ結合／非結合アクセスの評価

結合	非結合
7.0 [GB/s]	0.53 [GB/s]

表 3　シェアードメモリバンクコンフリクト有無の評価

バンクコンフリクトなし	バンクコンフリクトあり
1.3 [GB/s]	0.83 [GB/s]

表 4　テンプレートマッチングの処理時間

グローバルメモリ利用	テクスチャメモリ利用
994 [ms]	294 [ms]

3. メモリの選択とアクセスの最適化

CUDA では，表 1 に示した 4 種類のメモリが利用可能である．これらのメモリは性質が異なるため，処理ごとに最適なメモリを選択する必要がある．主として利用されるグローバルメモリ，シェアードメモリには，データ配列への最適なアクセス条件が存在し，それを満たさない場合は一桁以上スループットが低下してしまう．そこでこの 2 つメモリについて最適なアクセス方法を述べる．

まず，グローバルメモリへのアクセス最適化として重要になるのが結合アクセスである．結合アクセスが行われると，グローバルメモリへ並列にアクセスすることが可能となり，ロード命令を削減することができる．結合アクセスの条件は以下の 2 点である．1 つ目は，グローバルメモリ内のデータが 4, 8, 16 バイトでアライメントされている場合，1 命令でレジスタへロードすることができる．2 つ目は，先頭スレッドのアクセス先アドレスが 16×sizeof(type) になっており，半ワープ単位にアドレスがスレッド番号順になっていれば複数のデータに対して並列にロードすることができる．例えば int 型なら，先頭から 64, 68, 72, … , 120, 124 となる．この例を図 3 に示す．

次にシェアードメモリについて述べる．シェアードメモリは 32 ビットの 16 バンク構成になっており，各スレッドがバンクへ並列にアクセスすることができる．しかしながら，1 つのバンクにアクセスが集中するバンクコンフリクトが発生すると，スループットが低下する．例えば複数のスレッドが 1 つのバンクに集中してアクセスをした場合，これらはシリアル化され，複数回ロード命令が発生することとなる．2 つのスレッドが衝突した例を図 4 (a)に示す．これらを回避するために，1 スレッドが 1 バンクになるようにデータ配列に対してアクセスを行う事が重要である．

4. 評価・考察

評価環境は CPU : Intel Core2Duo E8500 (3.16GHz)，GPU : Nvidia GeForce 9500GT，OS : Windows Vista である．9500GT はマルチプロセッサ数 4（ストリームプロセッサ数 32）を有する． まず，グローバルメモリからグローバルメモリの転送において，結合／非結合アクセスについて測定した結果を表 2 示す．結合アクセスによって約 13 倍の高速化が得られた．次に，グローバルメモリからシェアードメモリへ，シェアードメモリからグローバルメモリへの転送において，バンクコンフリクトあり／なしの場合について測定を行った結果を表 3 に示す．バンクコンフリクトを無くなことによって約 1.6 倍の高速化が得られた．表 4 には，画像サイズ 640×480，テンプレートサイズ 16×16 のテンプレートマッチングにおいて，グローバルメモリ，テクスチャメモリを利用した際の計算時間を示す．テクスチャキャッシュの効果により，3.4 倍の高速化が得られた．

5. まとめ

本稿ではグローバルメモリ，シェアードメモリの最適なアクセス条件の影響を評価した．また，テクスチャメモリとグローバルメモリを用いたテンプレートマッチングの処理時間比較によりキャッシュの効果を評価した．今後は，与えられたユーザプログラムに対して最適なメモリアロケーションを求める手法の開拓が重要となる．

参考文献

[1] NVIDIA Corporation, “NVIDIA CUDA Programming Guide” Ver2.2.1, 2009.
[2] 出口大輔, “GPU よる高速画像処理　～実例で分かる GPGPU～”, 第 15 回画像センシングシンポジウム, 2009.