B-026

GPU を用いた計算コストの小さい数値計算の高速化

Acceleration of numerical calculation with low calculation cost using GPU

吉田　慧吾 *　　魚谷　智志 *　　増田　信之 *
Keigo Yoshida　　Satoshi Uotani　　Nobuyuki Masuda

1. まえがき

近年，グラフィックス処理の分野で GPU(Graphics Processing Unit) が多く用いられている．GPU とは 3D グラフィックスなどの画像描写を行う際に必要な計算処理を行うプロセッサで単純な演算かつ大量の計算処理に長けている。また，グラフィックス処理以外の汎用計算技術である GPGPU(General-purpose computing on graphics processing units) も盛んとなっており，気象シミュレーション，石油採掘現場での地震波解析，信号処理，金融工学など，広い分野で用いることができる．本研究では半導体チップメーカーの NVIDIA 社が提供するプログラミングモデルである CUDA(Compute Unified Device Architecture) を用いている．CUDA は GPU の性能をプログラミング手法によって最大限に活用することができる利点がある．本研究では，GPU が得意としない計算コストの小さい数値計算においての検証と高速化を図った．

2. CUDA(Compute Unified Device Architecture)

GPGPU の分野では，汎用的な並列処理言語を用いてプログラムを実装することが一般的となっている．

2.1 CUDA

GPU のチップメーカーである NVIDIA 社が自社製品向けに C 言語に並列処理用の拡張を加えた CUDA というものがある．CUDA はさまざまな GPU カーネルを高速化するために最も普及しているアプリケーションプログラミングインターフェイスの 1 つである．CUDA を利用することによって，C/C++で書かれたコードをほんのわずかなプログラミング作業で，GPU 上で効率よく実行できる．CUDA プログラミングの主な流れを図 1 に示す．

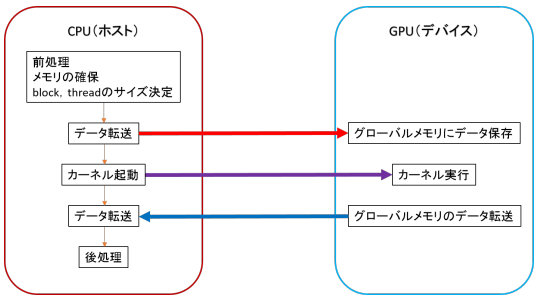

図 1: CUDA プログラミングの流れ

2.2 CUDA ストリーム

CUDA ストリームとは CUDA の一連の非同期操作である．この操作は CPU ホスト側で命令された順番に従って GPU デバイス側で実行される．ストリームは発行された操作をカプセル化し，順序を保ったまま，ストリームのキューに追加して前の操作の後に実行できるようにする．そして，キューに追加した操作の状態を取得できるようにする．CUDA ストリームのキューに追加された操作はすべて非同期である．そのため，ホスト/デバイスシステムではそれらの実行とほかの実行をオーバーラップさせることが可能である．

CUDA プログラミングの典型的なパターンとして，

1. 入力データをホストからデバイスに移動する．
2. デバイスでカーネルを実行する．
3. 結果をデバイス側からホスト側に移動する．

がある [1]．この構図のタイムラインを図 2 に示す．

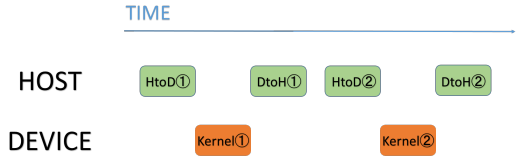

図 2: CUDA プログラミングの典型的なタイムライン

図 2 にある「HtoD] の「H」と「D」はそれぞれホストとデバイスのことを指している.「HtoD] はホスト側からデバイス側へのデータ転送,「DtoH」はデバイス側からホスト側へのデータ転送を指している.「Kernel」はデバイスでの実行を指している．数字はデータの順番を表している．この操作を繰り返している．また，横軸は経過時間を示している．このように，データ転送している間やカーネルを実行している間はどちらかが何もしていない状態となってしまう．そこで，CUDA ストリームを用いることによりホストとデバイス間のデータ転送とデバイス側でのコンピューティングを並列処理することでき，オーバーラップすることができる．この構図をタイムラインで図 3 に示す．

*東京理科大学基礎工学部

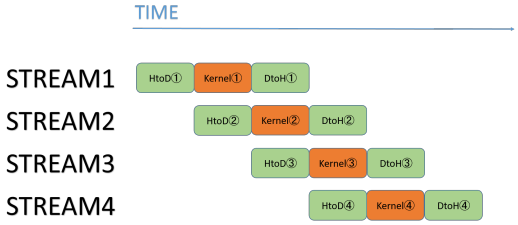

図 3: CUDA ストリームを用いた時のタイムライン

3. 実験結果

本研究の実験方法を示す．解像度が 128 × 128 の画像 1 枚分のすべての画素値をファイルから読み込み，用意した配列に代入していく．1 枚分計算させたら次のデータをファイルから読み込み，同じように計算させた．この過程を 1 回と 10 回繰り返して行った時の計算時間を CPU と GPU それぞれで測定した．また，CUDA ストリームを用いた場合 10 枚分の計算時間を測定値とした

3.1 実験環境

本実験で用いたシミュレーション環境を表 1 に示す．計算時間の測定に CPU では getrusage()，GPU では nvprof を用いた．また，CUDA ストリームを用いた場合には NVIDIA Visual Profiler で，最初にホスト側からデバイス側にデータを転送し始めてから最後にデバイス側からホスト側に結果データを転送し終わるまでの時間を計算時間とした．そして，ブロックを (8,8,1)，スレッドを (16,16,1) にして測定した．

表 1: シミュレーション環境

OS	Windows 10 Pro
CPU	Intel(R) Core(TM) i5-8400 2.80GHz
RAM	32GB (8GB × 4)
GPU	NVIDIA GeForce® GTX 1060 3GB
開発環境	CUDA10.0 V10.0.130

3.2 測定結果

CPU と GPU それぞれの計算時間の測定結果を表 2 に示す．

表 2: GPU と CPU の比較

画像数	GPU[μs]				CPU[μs]
	HtoD	DtoH	演算	合計	
1 枚	6.752	5.76	5.532	18.044	133
10 枚	66.688	52.096	47.488	166.272	1302

表 2 より CPU よりも GPU の方が計算速度が速いことが分かった．また NVIDIA Visual Profiler でストリームを用いたタイムラインを図 4 に示す．

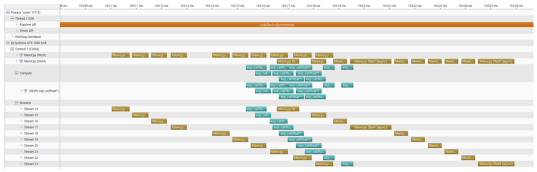

図 4: CUDA ストリームを用いた時のタイムライン

図 4 において茶色がデータ転送，青が GPU での計算を表している．このように，データ転送と GPU での計算がオーバーラップされていることがわかる．また，測定結果を表 2 の結果と比較して表 3 に示す．

表 3: CUDA ストリームの有無の比較

	計算時間 [μs]
ストリームなし	166.27
ストリームあり	157.92

ストリームを用いた方が計算時間が短くなることが分かった．

4. まとめと今後の課題

本研究で演算量の少ない画像処理を CPU と GPU で比較すると GPU の方が速く処理できた．そこに CUDA ストリームを用いることでさらに高速化することができた．今後さらに計算時間の短縮をするために検証したいことや案について述べる．CUDA ストリームを用いたがうまくオーバーラップしきれていないところが多くあった．ここを今後修正していきたい．また，CUDA ストリームを用いて画像出力まで行った際の単位時間当たり処理させるフレーム数を算出し，検証していきたい．そして，計算時間を短縮するための手法として GPU と CPU の処理を並列にするプログラムを考えたい．本研究ではデバイス側からデータを受け取ってから OpenGL を用いて画像を出力することを考えている．OpenGL のバッファオブジェクトという機能を利用すると，GPU によって生成された画像データをそのまま CPU を介することなくテクスチャに移動することができる [2]．本研究においては PBO(Pixel Buffer Object) を用いることで，さらに計算時間を短くすることができると期待できる．この応用として CUDA ストリームと併用することができるかの検証もしていきたい．

参考文献

[1] John Cheng,Max Grossman,Ty McKercher『CUDA C プロフェッショナル プログラミング』森野慎也訳，株式会社インプレス，2015 年

[2] 乾正知『GPU 並列図形処理入門ー CUDA･OpenGL の導入と活用』技術評論社，2014 年