B-006

マルチ GPU を用いた CGH 計算の高速化と精度比較
Accelerate CGH calculation using multi-GPU and comparison of its accuracy

片岡 昌彦*　　増田 信之*
Masahiko Kataoka　　Nobuyuki Masuda

1. はじめに

ゲームなどの映像の高画質化に伴い，グラフィックスの描画処理を行うユニットである GPU(Graphics Processing Unit)が発達してきた．CPU との比較として，数百倍もの計算コアを有する点や高度な計算を不得手とする点が挙げられ，これらより GPU は大量かつ単純な数値計算に長けているといえる．また GPU の優れた計算性能を一般的な数値計算にも利用する GPGPU(General-purpose computing on graphics processing units)が誕生した．これにより流体計算や画像分析，気象シミュレーション，石油採掘現場での地震波解析など様々な領域において従来のコンピュータでは膨大な時間がかかってしまうような大規模な演算を，GPU を用いることでより短時間で行うことが可能となった．本研究では 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 GPGPU として用いることで演算の高速化を行う．本研究は 2 つの GPU を用いた場合での CGH(Computer Generated Hologram)計算の検証と高速化を行うことを目的とする．メモリアクセスを変化させた場合について，および単精度浮動小数点演算と倍精度浮動小数点演算におけるシミュレーションを行うことで目的の達成を図る．

2. CUDA(Compute Unified Device Architecture)

GeForce シリーズなどの GPU を開発している NVIDIA 社によって，GPU コンピューティング環境「CUDA」が提供されている．CUDA は他の GPGPU 環境と比較して GPU の内部構造などを理解せずとも使用が可能である点や，無料で用いられる点などにより広く普及している．[1]

2.1 CUDA プログラミング

CUDA を用いることによって，C/C++や Fortran で書かれたコードを比較的容易に並列化し GPU 上で高速に処理することが可能となる．並列計算の手法として Grid，Block，Thread という概念を導入しており，これらが階層的な構造をとることによって一度に多数の Thread に対して実行を命令することができる．また 32 個の Thread をまとめたものを warp と呼び，warp 単位で同時に命令が実行される．CUDA プログラミングの主な流れを図 1 に示す．[1]

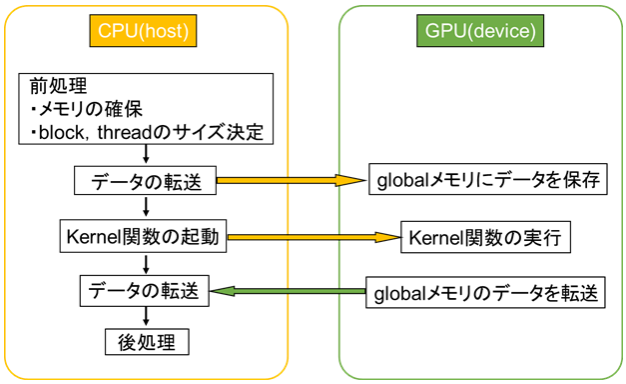

図 1. CUDA プログラミングの流れ

2.2 CUDA のメモリモデル

CUDA で用いられるメモリにはいくつかの種類が存在する．表 1 に CUDA のプログラマブルメモリの一部を示す．

表1. CUDAのプログラマブルメモリ

メモリ	容量	速度	場所	アクセス
Shared	小	高	チップ上	R/W
Global	大	低	チップ外	R/W
Constant	小	高	チップ外	R

表 1 に示したように Global メモリは GPU チップの外部に存在するため，アクセス速度が遅いという特徴を持つ．しかし GPU において容量が最も多く，またすべてのブロック内のすべてのスレッドから読み書きが可能である点から，最も利用されるメモリである．Global メモリへのアクセスは一定のサイズでまとめて行うようになっているため，それより小さいサイズのデータのアクセスを行うと効率の低下が引き起こされる．このため，上記の条件に合わせたアクセスを行うプログラムを作成することが効率の良いメモリアクセスを行うために必要となる．

Constant メモリは名前が示唆するようにカーネルの実行中に変化しないデータを扱うメモリである．チップの外部に存在し容量が 64KB と小さいが，キャッシュが効くため Global メモリよりも高速にアクセスが可能である．しかしアクセス面として，書き込みがホスト側からのみ可能で，デバイス側からは読み込みしか行えないという特徴を持つ．Shared メモリは上記の二つとは異なり，GPU チップの内部に存在する．

*東京理科大学　Tokyo University of Science

Shared メモリはバンクと呼ばれるユニットの集まりになっており，warp 内の Thread がそれぞれ異なるバンクにアクセスすることで一度にすべてのデータにアクセスし，高速な処理を行うことが可能である．しかし複数の Thread が同時に同一のバンクにアクセスを行った場合アクセスが逐次化され，アクセス速度の低下が引き起こされる．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となっている．

これらより数値計算の特徴に合わせてメモリを最適に使用することが出来ればプログラムの高速処理が可能となる．[1][2][3]

3. 実験

本研究では一定の物体点データから CGH を作成する際の数値計算に要する時間を測定する．CGH の作成には点光源モデルを採用する．複数点とホログラムの位置関係を図 2 に示す．

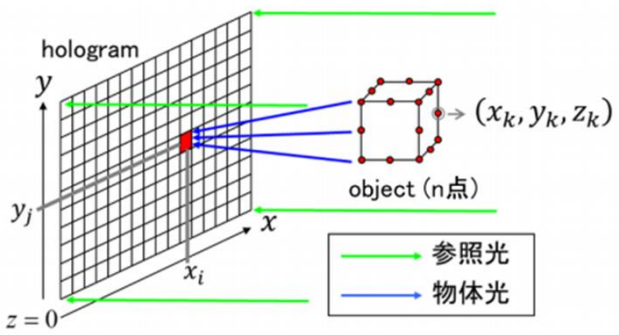

図 2．複数点とホログラムの位置関係[4]

図 2 のように，物体の各点から得られる光強度を重ね合わせることで 1 画素の光強度を求める．ホログラム面 *I(x, y)*における 1 画素の光強度を求める式は次のように表せる．

$$I(x,y) = \sum_{k=1}^{n} \cos\left(\pi p\left(\frac{(x-x_k)^2 + (y-y_k)^2}{\lambda_{z_k}}\right)\right) \quad (1)$$

（1）において *n* は物体点数，*p* は画素ピッチを表しており，この計算をすべての画素に対して行うことで 1 枚のホログラムを作成する．

3.1 実験環境，条件

本研究におけるシミュレーション環境を表 2 に，シミュレーション条件を表 3 に示す．

表2：シミュレーション環境

OS	CentOS Linux 7.1.1503
CPU	Intel(R) Xeon CPU E5-2697 v2 @ 2.70GHz
RAM	64GB(32GB×2)
GPU1	Tesla(R) P100-PCIE-12GB
GPU2	GeForce(R) GTX 1080
CUDA Compiler	nvcc9.1

表3．シミュレーション条件

物体点数	11616点
物体点データ容量	136KB
ホログラム画素数	1024×1024
波長	633nm
画素ピッチ	10.5μm

本研究では GPU を 2 つ同時に用いる手法として OpenMP を使用する．OpenMP で CPU スレッドを GPU の数実行しそれぞれが別の GPU と接続することによって GPU を並列に動作させる．また 2GPU それぞれが受け持つ計算量は半々となるよう分割する．表 3 の条件を元に GPU 上で（1）式を計算し，それにかかる時間を測定する．測定は nvprof を用いて行い，それぞれ 10 回測定しその平均時間を測定結果とする．また Block,Thread の分割は(32, 256),(32,4)とする．

3.2 測定結果

Global メモリを用いて単精度浮動小数点演算と倍精度浮動小数点演算を行った測定結果を表 4 に示す．

表4．Globalメモリの測定結果

精度	計算時間[ms]
単精度	219.89
倍精度	1806.0

表 4 より単精度と倍精度では計算時間に 8.2 倍ほどの差が生じることが分かった．

4. まとめと今後の課題

本研究より CGH 作成において Global メモリを用いた場合，単精度浮動小数点演算は倍精度浮動小数点演算都の 8.2 倍ほど高速になることが分かった．目的達成のため今後検証していく案について述べる．まず現在 Global メモリのみ測定を行っているが，他のメモリを用いて同様の測定を行う．各メモリでアクセスに注目しプログラムを最適化，精度ごとにその測定結果を比較する．またその際生成されたホログラム画像を比較し精度を確認する．これらによって高速化の観点と精度から 2GPU を用いた場合の CGH 作成に適したメモリを考察していく．

参考文献

[1] 伊藤智義, “GPU プログラミング入門-CUDA5 による実装” 講談社, 2013 年, p4, p41, p49-52
[2] Jason Sanders, Edward Kandrot, “CUDA by Example 汎用 GPU プログラミング入門” 株式会社クイープ訳, 株式会社インプレスジャパン, 2011 年, p79-80
[3] 吉田慧吾, “GPU を用いた CGH 計算の高速化と制度比較”, 東京理科大学基礎工学研究科修士論文, 2021 年
[4] 野村英雄, “マルチ GPU を用いた高速計算システムの構築”, 東京理科大学基礎工学研究科修士論文, 2021 年