B-006

GPGPU における複数カーネルを用いた数値計算の高速化についての検討

Study on the speed of numerical calculation using multiple kernels in GPGPU

南　昇吾 * Shogo Minami　山本 未来呂 * Mikuro Yamamoto　臼倉 圭亮 * Keisuke Usukura　増田 信之 * Nobuyuki Masuda

1. まえがき

近年では，様々な分野において数値シミュレーションや数値計算が行われており，より大規模かつ高精度な計算が必要とされている．しかし，大規模かつ膨大な計算にはより多くの時間がかかるために高速化が求められる．計算を高速化する一つの手段として，本来では画像処理用に使用される GPU(Graphics Processing Unit) を数値計算に用いる GPGPU(General purpose computing on GPU) が一般的に使用されるようになってきた．

GPGPU は，基本的に演算を並列化することで計算の高速化を図る．そのため，演算の中に条件分岐処理が含まれる場合，計算プロセッサ同士の待ち時間が生じるため，パフォーマンスが落ちてしまう [1]．そのため本研究では，条件分岐処理が多く含まれる数値計算における GPU を用いた高速化について検討を行った．計算題材として，FDTD 法 (Finite Difference Time Domain method) を使用した．

具体的には，条件分岐を含む領域と含まない領域とで計算領域を分割し，それぞれを別のカーネル関数として実行させる．結果として，計算量が少ない場合では分割数を少なく，計算量が多い場合では分割数を多くすることで演算が高速となることが分かった．

2. FDTD 法

FDTD 法は，電磁波シミュレーションになどに用いられる電磁界解析法の一つである．マックスウェル方程式を差分化し，時間領域で解くことから，最終的な結果だけでなく，結果に至るまでの電磁界の変化を解析することができる．本稿では，境界条件として Mur の吸収境界条件を用いた二次元 FDTD 法について解析を行った．

2.1 FDTD の計算

今回は電解が 1 方向成分のみの場合である TE 波を考える. マックスウェル方程式を時間, 空間に対して中心差分し, 以下の二次元 FDTD 法の式を得る．

$$H_x^{n+1/2}(i,j+1/2) = H_x^{n-1/2}(i,j+1/2) - \Delta t/\mu\Delta y\{E_z^n(i,j+1) - E_z^n(i,j)\} \quad (1)$$

$$H_y^{n+1/2}(i+1/2,j) = H_y^{n-1/2}(i+1/2,j) - \Delta t/\mu\Delta x\{E_z^n(i+1,j) - E_z^n(i,j)\} \quad (2)$$

$$E_z^{n+1}(i,j) = E_z^n(i,j) - \Delta t/\epsilon\Delta y\{H_x^{n+1/2}(i,j+1/2) - H_x^{n+1/2}(i,j-1/2)\} + \Delta t/\epsilon\Delta x\{H_y^{n+1/2}(i+1/2,j) - H_y^{n+1/2}(i-1/2,j)\} \quad (3)$$

ここで，空間離散間隔を Δx , Δy とし, 時間離散間隔を $\Delta\ \ t$ とする. また, n は時間ステップ，E は電界，H は磁界を表す．また，パラメーター ϵ と μ はそれぞれ誘電率と透磁率を表す．Mur の吸収境界条件については別紙に譲る [2].

2.2 計算手順

計算手順は，ある時間ステップにおける磁界と電界の値を GPU で並列計算し，計算結果をグローバルメモリ上に保存させる．ステップを進めて再び並列計算を行い，求めたい時間ステップ分繰り返し計算を行ったら演算を終了する．本稿では，磁界計算部分に条件分岐が発生しないため，吸収境界条件を含む電界計算部分についてのみカーネルの分割を検討した．

3. カーネル分割方法

本稿では，電界計算部分のカーネルを 7 通りの分割方法について比較した．カーネルの分割方法について次の図 1 に示した [3].

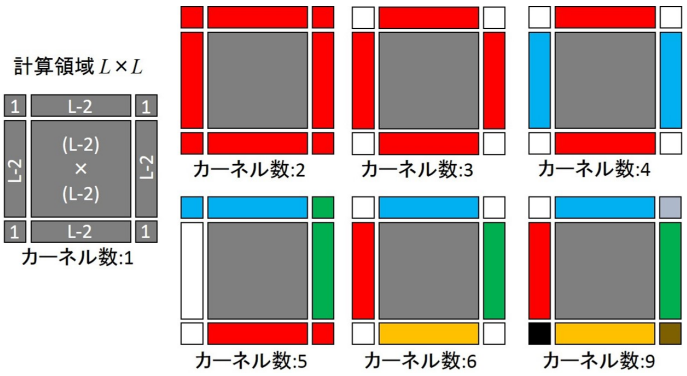

図 1: 分割方法

図 1 は，同じ色で塗られた領域が同一のカーネルとなるように表記してある．また，吸収境界条件に当たる領域は計算量が中央部分に比べて比較的少ない周辺部分が相当する．

4. 実験結果

4.1 実験環境

ここでは，磁界計算と電界計算を GPU によって行い，電界部分についての計算時間の比較を行う．電界部分におけるカーネルの分割方法は，前章で述べたように 7 通りについて比較する．計算時間の計測には nvprof を用い，CPU-GPU 間のメモリアクセス時間はグローバルメモリを用いるためどの分割方法も等しいとする．カーネル起動時間に関しては今回に限り無視し，純粋な計算時間のみの比較を行った．時間ステップ数を 850，計算領域を $L \times L$，thread 数を x 方向 y 方向共に 16 とし，初期条件として磁界は 0，電界は半分の領域を 0 もう半分を 1 とした．計測時間は 850 ステップの平均を 1 ステップ分として算出した．また，ソフトウェア開発環境を表

*東京理科大学基礎工学部

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.

表 3: 単精度演算において cuda8.0 環境で GTX 1080 を使用したときにおける計算時間

計算領域 $[L \times L]$	計算時間 $[\mu s]$						
	kernel1	kernel2	kernel3	kernel4	kernel5	kernel6	kernel9
18×18	**3.288**	5.980	7.255	8.110	9.256	10.187	14.022
34×34	**4.566**	6.057	7.279	8.193	9.321	10.276	14.080
66×66	**5.063**	6.254	7.542	8.412	9.512	10.470	14.154
130×130	12.102	**8.880**	10.040	10.918	11.362	12.406	16.001
258×258	211.303	**19.102**	20.376	21.052	21.624	22.703	26.318
514×514	1,081.473	**76.748**	77.877	78.341	80.186	79.011	84.829
1026×1026	654.281	331.284	332.733	332.960	**330.802**	335.282	338.856
2050×2050	3,849.307	**1,195.214**	1,196.067	1,198.086	1,199.128	1,198.889	1,201.850
4098×4098	15,015.013	**4,174.785**	4,177.175	4,178.622	4,180.687	4,177.059	4,180.711
8194×8194	276,639.678	26,802.339	26,805.982	26,804.586	**26,800.212**	26,801.731	26,813.548
16386×16386	233,637.001	106,012.725	106,015.211	**106,008.519**	106,015.430	106,067.856	106,031.097

1 に示した．また，計算に使用した GPU について概要を表 2 に示した [4].

表 1: ソフトウェア開発環境

CPU	Intel Xeon E5-2697 v2 2.70GHz ×2
メモリ	64 GB (32GB×2)
GPU	NVIDIA GeForce GTX 1080
OS	CentOS Linux 7.1.1503
開発環境	CUDA 8.0

表 2: GTX 1080 概要

CUDA コア	2560
ベースクロック	1,607MHz
メモリ量	8GB

4.2 計測結果と考察

表 3 に各々のカーネル分割時における 1 ステップ分の電界計算にかかる計測時間を示した．表 3 より，カーネル 1 に対する, 分割を行った際の高速化比は計算領域の大きさによってまばらな値となったが, ピーク時の高速化比は 3.5 倍を超える高いものとなった．

また, 表 3 において赤字で示した数字は同一計算領域で最速の時間である。この結果から計算量がある程度少ない場合ではカーネルを分割しないほうが高速となっていることがわかる．この原因として，CUDA には Warp という概念があり，32 個のスレッドを 1 つのグループとして並列化する [5]．そのため計算量が少ない場合，カーネルを分割すると境界条件部分の計算量が少ないため，GPU のリソースを活かすことができずパフォーマンスが低下する．

一方, 計算量が大きい場合, カーネルを多く分割する方が計算が高速となっている事がわかる．ここで, カーネルを分割することによってプログラムが高速となったことについて考える. カーネルを分割すると, 条件分岐 (if 文) がカーネル関数内に存在しなくなる. そのため, ワープダイバージェンスが発生しなくなり高速となったと考えられる. しかし計算量によってはカーネルの数が多すぎてもプログラムが高速にはならないことがわかった.

5. まとめと今後の課題

本論文では, GPGPU が苦手とする条件分岐処理を含む数値計算における高速化の検討を行った．GPU 上で起動するカーネル関数を複数用いることにより，1 つのカーネル関数内における条件分岐処理の影響を減らし，計算時間の低減を行えると仮定した．その結果，計算量が比較的多い場合では分割数を増やすことで計算時間を低減することができた．しかし，計算量が少ない場合ではカーネルを分割しないほうが高速となることがわかった．

今回，計算量に応じて最適なカーネル数が異なることがわかった．そのため今後の展望として，計算量に応じて最適なカーネルの分割を選択するといったアルゴリズムの開発を検討する．

謝辞

本研究は JSPS 科研費 15K00175，の助成を受けたものです.

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”, 株式会社インプレス, 2015

[2] 宇野 亨, ”FDTD 法による電磁界およびアンテナ解析”, コロナ社, 1998

[3] 臼倉圭亮, 山形健太, 山本未来呂, 増田信之 , ”GPGPU における条件分岐処理を含む数値計算の高速化についての検討 ”, 情報科学技術フォーラム講演論文集 ,p201-202, 2016.

[4] ”NVIDIA Home Page ” , http://www.nvidia.co.jp/object/geforce-gtx-titan-x-jp

[5] 伊藤 智義, ”GPU プログラミング入門”, 講談社, 2013

Copyright © 2017 by Information Processing Society of Japan and
The Institute of Electronics, Information and Communication Engineers
All rights reserved.