3J-07

GPGPUによる電位・電界シミュレーションの並列化

松原　翼†　　長尾　栄作†　　上嶋　明‡　　尾崎　亮‡　　小畑　正貴‡

岡山理科大学大学院工学研究科情報工学専攻†　　岡山理科大学工学部情報工学科‡

１．はじめに

電磁気学で用いられる電位・電界シミュレーションは，落雷の予測や，半導体デバイスの動作の解析などに応用されている。このシミュレーションにはコストが小さく，詳細な解析データを得られることができるという大きな特徴があるものの，膨大な計算量と計算時間を要するという問題が存在する。そこで本研究では，画像処理用の演算装置 GPU を汎用計算に使用する GPGPU(General Purpose Computing on GPUs)を用いて，シミュレーションを並列処理により高速化する。

２．電位・電界シミュレーション

２．１　概要

電位・電界シミュレーションの手法として，ポアソン方程式を用いて電位を計算し，その結果を基に電位と電界を可視化する。GPU への実装には，NVIDIA 社から提供される GPU 上での汎用計算を目的とした C 言語統合開発環境である CUDA を利用する。

２．２　ポアソン方程式

ポアソン方程式で記述される電磁気学における物理現象として，静電ポテンシャルがある。与えられた電荷の分布をρとしたとき，静電ポテンシャルϕは次のポアソン方程式を満たす[1]。

$$\Delta\phi = -\frac{\rho}{\varepsilon_0} \qquad (1)$$

ただし，ε_0は真空中の誘電率で$8.85 \times 10^{-12}[\mathrm{F/m}]$とする。また，本研究では座標系を 2 次元とする。つまり，電位はx, yの関数でz方向には変化がないものとする。すると式(1)は以下のように表すことができる[2]。

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = -\frac{\rho}{\varepsilon_0} \qquad (2)$$

２．３　計算手順

図 1 に電位・電界シミュレーションのプログラムの流れを示す。まず，電位を格納する配列と電荷を格納する配列を定義する。次に，電位と電界を領域全体にわたり 0 で初期化し，電荷密度を設定する。本研究では，電荷の数を 10 個，密度の値を$1.0 \times 10^{-8}[\mathrm{C/m^2}]$とする。そして，領域端を除く範囲で電位の計算を繰り返す。その際，前回計算した電位の値との残差を全領域に対して求め，残差の最大値が収束判定係数以下となるまで処理を繰り返す。最後に電界を計算し，電位とともに出力する。

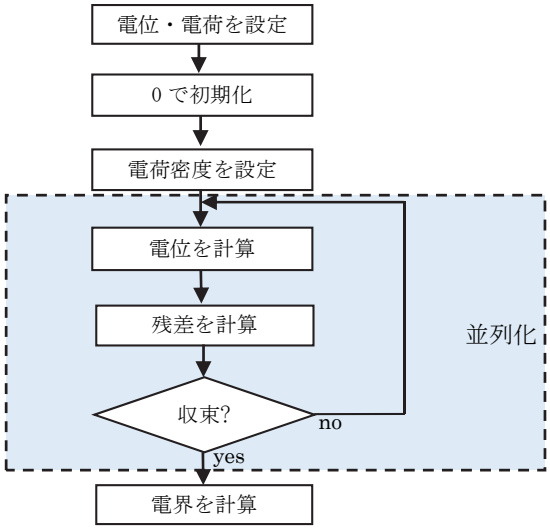


図 1　シミュレーションプログラムの流れ

２．４　並列化方法

電位と残差の計算を繰り返す処理を，GPU 上で計算領域の一つの格子の計算を 1 スレッドに割り当てて並列化する。また，実行時間の比較のため，CPU 上で OpenMP を用いて計算領域を等分割して各コアへ均等に割り当てて並列化する。

GPU 上での並列化の際，当初は全格子の残差の最大値が収束判定係数を超えるかどうかを基に収束判定を行っていた。しかし，GPU の特性により実数値の最大値計算に時間を要し，高速化の妨げとなった。そこで本研究では，収束判定係数を超える格子の数をカウントし，それが 0 になると領域全体が収束したと判定するようにした。これにより，共有メモリを用いた高速な並列リダクションを行うことが可能となった。

ただし，並列リダクションを行う際には条件分岐命令で 32 スレッドの集合であるウォープ内のスレッドの分岐方向が一致せずに性能が低下してしまうウォープダイバージェントが発生する。そこで，スレッド番号が小さいスレッドが優先的に処理を実行するようにした。これにより，分岐方向の不一致が減少するのでウォープダイバージェントの数を減らすことが可能となる。

一般的に，GPU のようなマルチスレッドアーキテクチャでは複数スレッドが同時にメモリアクセスするため，メモリが一度に 1 アクセスしか処理でき

Parallelization of Electric Potential and Field using GPGPU

Tsubasa Matsubara†, Eisaku Nagao†, Akira Uejima‡, Ryo Ozaki‡ and Masaki Kohata‡

†Graduate School of Engineering, Okayama University of Science

‡Faculty of Engineering, Okayama University of Science

ない場合はボトルネックになりがちである。具体的には，複数スレッドが共有メモリの同一バンクにアクセスするバンクコンフリクトが発生するため処理に遅延が生じる。そこで，バンクの異なる連続データに同時にアクセスすることにより，バンクコンフリクトを除去することが可能となる。

また，電位の計算と収束判定で除算を行う部分で，超越関数演算を担当する演算器であるSFUを使用することによりさらなる高速化を実現する。さらに，収束判定の際の判定周期については複数回の試行により，計算時間を短くできる値を得て適用する。

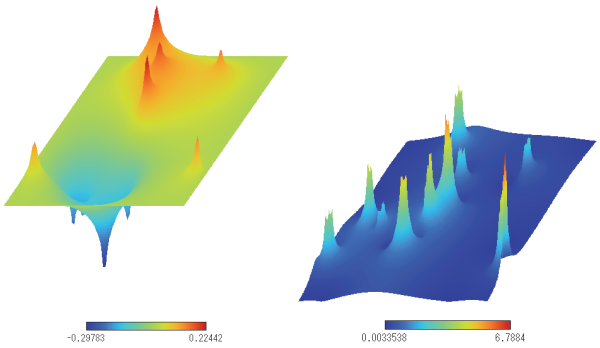
3．実験結果

3．1　シミュレーション結果

実験環境を表1に示す。CPU 1コア上で逐次実行するプログラム，同6コア上で並列実行するプログラム，GPU 上で並列実行するプログラムを用いて実験を行った。計算領域の分割数 128 での電位と電界のシミュレーション結果を図2に示す。電位と電界の値が大きい部分ほど赤色に近づいていることがわかる。

表1　実験環境

	CPU	GPU
プロセッサ	Xeon E5-1650	Tesla K20
コア数	6	2496
コア周波数	3.2 GHz	0.71 GHz
演算性能	0.3 TFlops	3.52 TFlops
メモリバンド幅	51.2 GB/s	208 GB/s
メモリ容量	32 GB	5 GB
OS	Linux 2.6(64bit)	
コンパイラ	gcc 4.4.7 (-O2)	nvcc 6.5



(a) 電位の可視化　　(b) 電界の可視化

図2　シミュレーション結果

3．2　収束判定周期の決定

GPU 上の並列実行では，まず計算格子数960の状態で収束判定周期を変化させて実行時間を測定した。収束判定周期と実行時間の関係を図3に示す。収束判定周期が1のとき実行時間が121.51秒となり，その後，収束判定周期を増やすごとに実行時間が短くなっているが，収束判定周期が 32 の辺りから実行時間の変化が小さくなっている。その後，収束判定周期が 262144 となったときから実行時間が長くなっている。これは収束判定ループを打ち切るタイミングが遅れることが原因である。したがって，GPUでさらなる高速化を実現するためには収束判定周期が 32 から 131072 までの値が望ましいと判断した[3]。

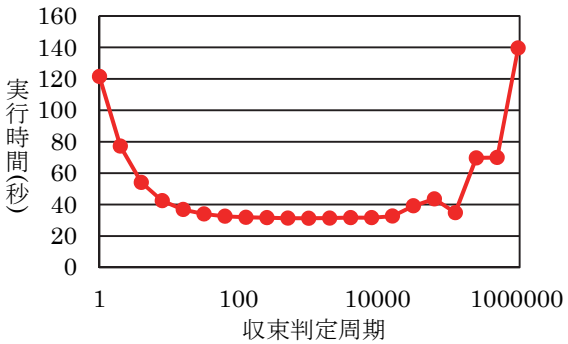


図3　収束判定周期とGPU上での実行時間の変化

3．3　実行時間

CPU 1コアとCPU 6コア，GPU 上でのシミュレーションの実行時間を測定した。CPU 1コアとCPU 6コア，GPU 上でのシミュレーションの実行時間を図4に示す。計算格子数960において，CPU 1コアで1,437.3秒，CPU 6コアで250.5秒，GPUで29.4秒となり，GPU上で並列実行することでCPU 1コアに対して約46倍，CPU 6コアに対しても約9倍の速度を達成することができた。

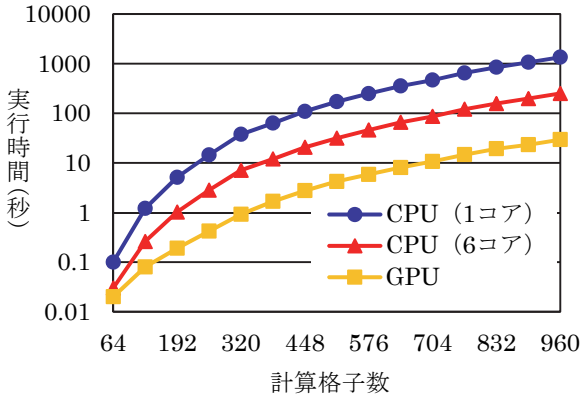


図4　実行時間

4．おわりに

電荷シミュレーションを GPU 上で並列化した。CPU 1コア時と比較して，計算格子数960において約 46 倍の高速化が実現できた。今後の課題として，シミュレーション結果をグラフィックスハードウェアの API である OpenGL を用いて可視化することや，3次元空間におけるシミュレーションを実現することなどがあげられる。

参考文献

[1]　“ポアソン方程式”：<http://jiten.biglobe.ne.jp/j/ee/3d/aa/ec04c693976aa12a40f3b0b6db42099c.htm> (2014 年 12 月 26 日アクセス)

[2]　“世界一やさしい Poisson 方程式シミュレーション”：<http://teamcoil.sp.u-tokai.ac.jp/classes/EM1/Poisson/index.html> (2014 年 12 月 26 日アクセス)

[3]　松原翼 他：“GPGPU での電荷シミュレーションの並列化”，電気・情報関連学会中国支部第65回連合大会講演論文集，pp. 362 - 363 (2014)