5H-07

GPGPUによる電位・電界シミュレーションの高効率化

松原　翼†　　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。シミュレーションには詳細な解析データを得られるという大きな特徴があるが，大きな計算量を要する。そこで我々は，画像処理用の演算装置GPUを汎用計算に使用するGPGPUを用いたシミュレーションの高速化方法を検討してきた。本研究では，並列処理による計算の高効率化を図るため，利用するGPUの特性に合わせた並列リダクションの実装方法の改良とメモリアクセス方法の変更を行い，その効果を評価する。

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。

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

ここで，ε_0は真空中の誘電率で$8.85 \times 10^{-12}[\mathrm{F/m}]$とする。本研究では座標系を2次元とし，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)を次のように表すことができる[1]。

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

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。そして，領域端を除く範囲で電位の反復計算を行う。その際，前回計算した電位の値との正規化した残差を全領域に対して求め，その残差の最大値が収束判定係数以下となれば反復を終了する。最後に電界を計算し，電位とともに出力する。

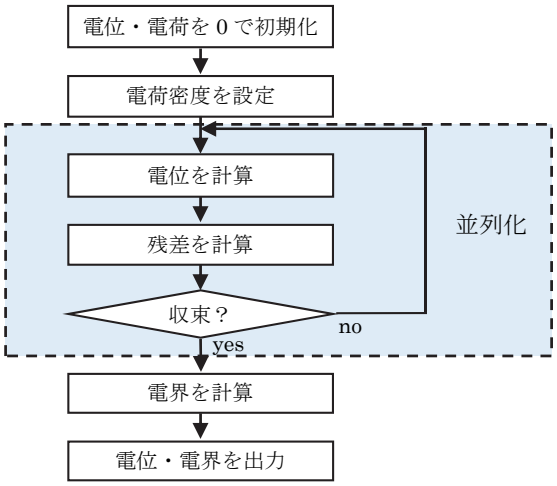


図1　シミュレーションプログラムの流れ

３．並列化

電位の計算は計算領域の格子点ごとに独立して行えるため，GPU上で一つの格子点の計算を1スレッドに割り当てることにより並列化する。残差の計算と収束判定をGPU上で高速に処理するため，並列リダクションを用いる[2]。

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ることにより並列化する。

３．１　並列リダクション

残差の計算では，現在の電位と反復計算の前回の電位との差を格子点ごとに求め，その各々を電位の最大値で除算して正規化する。この処理では，最大値を求める際にリダクション演算を使用する。収束判定では，正規化した残差が全格子点において指定した係数以下となるかを判定する。この処理では，全格子点での収束の確認にリダクション演算を使用する。いずれのリダクション演算でも，高速化のためにGPU上での並列リダクションを行う。

一般的にGPU上でのリダクション演算には時間を要するが，各ブロック内で共有メモリを利用し，木構造的にリダクションを行うことで高速化が可能である。また，残差の計算と収束判定を複数回おきに周期的に行うことで高速化が可能である。本研究では，複数回の試行により，計算時間を短くできる収束判定周期を100に決定して適用している[2]。

３．２　ウォープシャッフル命令

Kepler世代のGPUでは，同じウォープ（32スレッド）内に限り，他のスレッドのレジスタを直接読み取るウォープシャッフル命令を使用できる[3]。その一つとして，ウォープ内のスレッドをバタフラ

Improvement of Parallel Efficiency for Electric Potential and Field Simulation using GPGPU

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

†Graduate School of Engineering, Okayama University of Science

‡Faculty of Engineering, Okayama University of Science

イ形式で入れ替える XOR シャッフルがある。ウォープシャッフル命令を使用することにより，共有メモリやグローバルメモリを経由せずにウォープ内のスレッド間でデータを直接交換できるようになる。

32 スレッドでのリダクションの場合，5 ステップの XOR シャッフルと演算でリダクションを完了する。ブロック内の複数ウォープでのリダクションの際には，各ウォープでの結果を共有メモリ経由で一つのウォープに集めることで，再度ウォープシャッフルによるリダクションを行う。これにより，並列リダクションを高速かつ効率よく実行できる。

３．３　ウォープ Vote 関数

ウォープ内の各スレッドが持つ真理値に対して AND/OR 演算などを行う Vote 関数が利用できる[3]。これにより，収束判定においてウォープ内の全スレッドの判定結果を 1 回の関数呼び出しでリダクションできる。ブロック内の複数ウォープでのリダクションの際には，各ウォープでの判定結果を一つのウォープに集めることで，再度ウォープ Vote 関数によるリダクションを行う。

３．４　Read-Only データキャッシュ

読み取り専用のキャッシュであり，Kepler 世代の GPU ではキャッシュからレジスタにデータを直接転送できる。反復計算中には値の更新を行わない配列のアクセスに Read-Only データキャッシュを利用することで高速化を試みる。

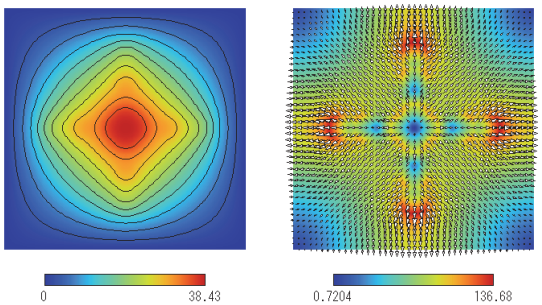
４．実験結果

４．１　シミュレーション結果

表 1 に示す環境で CPU 上と GPU 上での実験を行った。格子サイズ 128 での電位・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。

表 1　実験環境

	CPU	GPU
プロセッサ	Xeon E5-1650	Tesla K40
コア数	6	2880
コア周波数	3.2 GHz	0.75 GHz
演算性能(単精度)	0.31 TFLOPS	4.29 TFLOPS
メモリバンド幅	51.2 GB/s	288 GB/s
メモリ容量	32 GB	12 GB
OS	Linux 2.6(64bit)	
コンパイラ	gcc 4.4.7 (-O2)	nvcc 7.5



(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

４．２　実行時間

まず，リダクション方法による比較として，共有メモリのみの場合に対して，ウォープシャッフル命令，および，それに加えてウォープ Vote 関数を用いた場合の結果を表 2 に示す。元の並列リダクションに対し，最大 3.00 倍の速度を実現できた。

表 2　リダクション方法による比較

格子サイズ	共有メモリ	シャッフル	シャッフル＋Vote
128	9.256 (1.00)	6.167 (1.50)	5.977 (1.55)
1,024	203.202 (1.00)	76.998 (2.64)	69.898 (2.91)
1,920	699.387 (1.00)	258.838 (2.70)	233.477 (3.00)

（μsec，括弧内は共有メモリに対する速度向上）

次に，Read-Only データキャッシュを利用した場合の効果を評価した。格子サイズによって効果が変化し，サイズ 512 のとき速度が 1.15 倍となったものの，効果が見られない場合（サイズ 768）もあった。

最後に，CPU 1 コアと CPU 6 コア，実装方法改良前後の GPU でのシミュレーションの実行時間を測定した（図 3）。改良によりサイズが 128 のときに最小の 1.17 倍，サイズが 1,792 のときに最大の 1.62 倍の高速化を実現できた。サイズ 1,920 において，改良後（GPU2）で 369.24 秒となり，改良前（GPU1）との比較で約 1.31 倍の速度となった。これにより，CPU 1 コアに対して約 69.62 倍，CPU 6 コアに対して約 12.05 倍の速度を達成できた。

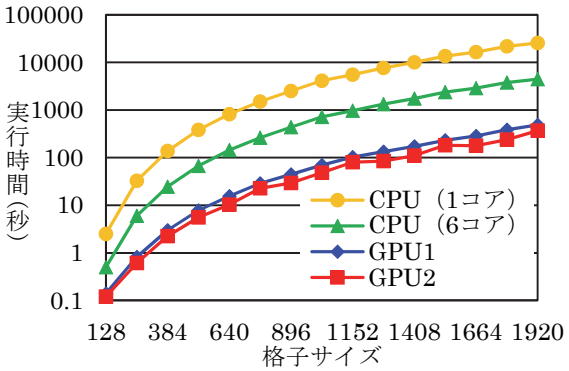


図 3　実行時間

５．おわりに

電位・電界シミュレーションにおいて並列リダクションの実装方法とメモリアクセス方法の変更を行うことで高効率化を図り，最大 1.62 倍の速度向上を達成した。今後の課題として，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] “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> (2016 年 1 月 5 日アクセス)

[2] 松原翼 他：“GPGPU による電位・電界シミュレーションの並列化，”情報処理学会第 77 回全国大会講演論文集，p. 1-73 – 1-74 (2015)

[3] “CUDA C Programming Guide”: <http://docs.nvidia.com/cuda/cuda-c-programming-guide/> (2016 年 1 月 5 日アクセス)