1J-08

NVIDIA A100 GPUにおける電位・電界シミュレーションの性能評価

小西 秀策 †　　上嶋 明 ‡
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

電位・電界シミュレーションは, 半導体の動作解析や落雷の予測などの分野に応用されている. このシミュレーションは膨大な計算量を要することから, GPGPU(General Purpose Computing on GPUs) を用いた実装 [1] が提案されている. 本研究では, 松原 (2015) による Poisson 方程式を用いた電位・電界シミュレーションの Kepler 世代 GPU 用プログラムを NVIDIA A100 GPU に移植し, 旧世代の GPU と比較してどの程度性能が向上するかについての評価を行う.

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, Poisson 方程式による電位の導出を行う. GPU への実装法には, NVIDIA 社による GPGPU 向けの開発環境である CUDA を用いる.

2.2 Poisson 方程式

電磁気学の分野において, Poisson 方程式は静電ポテンシャルの記述に用いられる. 与えられる電荷の分布を ρ, 真空中の誘電率を $\epsilon_0 = 8.85 \times 10^{-12}$[F/m] としたとき, 静電ポテンシャル ϕ は以下の Poisson 方程式を満たす.

$$\Delta\phi = -\frac{\rho}{\epsilon_0} \tag{1}$$

本研究では座標系を 2 次元とし, 電位は x,y の関数で z 方向は考慮しないものとする. この場合, 式 (1) は以下のように表すことが可能である.

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = -\frac{\rho}{\epsilon_0} \tag{2}$$

2.3 計算手順

本シミュレーションのプログラムの流れを図 1 に示す. まず, 電位を格納する配列と電荷を格納する配列を確保し, 領域全体を 0 で初期化する. 次に, 電荷密度の設定を行う. 本プログラムにおいては, 電荷の数を 10, 密度を 1.0×10^{-8}[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, 電界を算出し電位とともに出力する.

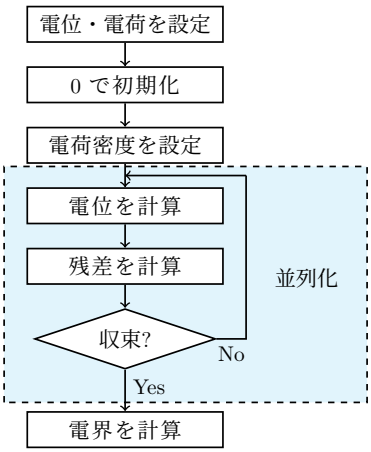

図 1　シミュレーションの流れ

残差の計算および収束判定においては，各格子点に対してリダクション演算を行う. 本シミュレーションプログラムでは, Warp Shuffle 命令を用い, GPU の 32 スレッドの集合である Warp 内のレジスタを用いて並列リダクションを行う. Warp Shuffle 命令では, 同じ Warp 内のスレッドのレジスタに直接アクセスするため, シェアードメモリを経由する場合よりも処理時間が短縮される.

2.4 NVIDIA A100 への移植

Kepler 世代向けのコードを NVIDIA A100 で動作させるにあたって, いくつかの部分を Ampere 世代の仕様に合わせた実装に変更する必要がある. 従来のコードでは Warp Shuffle 命令として組み込み関数の`__shfl_xor()` および`__any()` を使用していたが, Volta 世代以降においてこれらは非推奨となってい

Performance Evaluation of Potential and Electric Field Simulations on NVIDIA A100 GPUs
† Shusaku Konishi, Gradudte School of Engineering, Okayama University of Science
‡ Akira Uejima, Faculty of Engineering, Okayama University of Science

る. Kepler 世代以前では Warp 内の 32 スレッドの演算が同時に実行されることから暗黙の同期が保証されていたが, Volta 世代以降では Warp 内の各スレッドが個別にスケジューリングが可能であり, これらが全て同時に実行される保証はない. よって, Volta 世代より追加された Warp 同期用の新たな組み込み関数を仕様するか, Cooperative Groups によってスレッドを明示的に同期する必要がある [2]. 本研究では, 元のコードからの変更量が少なく済むことから warp 同期用の組み込み関数を使用する.

また, 推奨されてはいないものの, nvcc の`-gencode arch=compute_60,code=sm_80` オプションを用いることにより, Kepler 世代以前の暗黙の同期を用いたコードを Ampere 世代の GPU で実行することが可能である. 計算の内容によっては暗黙同期を用いたほうが高速になる場合があるが, Ampere 世代で導入された機能がいくつか無効化されることが報告されている [3].

3 実験結果

3.1 実験環境

実験環境を表 1 に示す. GPU 上で並列実行するプログラムを用いて実験を行った. A100 GPU 上では, 従来のコードを nvcc の`-gencode arch=compute_60,code=sm_80` オプションを用いてコンパイル・実行した場合 (Kepler モード) と, A100 向けに変更・コンパイルし実行した場合 (Ampere モード) の 2 種類で比較を行った.

表 1 計算機環境

CPU	型式	Xeon E5-1650	EPYC 7413
	コア数	6	24
	ベースクロック	3.2GHz	2.65GHz
GPU	型式	Tesla K20	A100 x2
	アーキテクチャ	Kepler	Ampere
	CUDA コア数	2880	6912 × 2
	ベースクロック	750MHz	1065MHz
	VRAM	12GB	80GB × 2
	ピーク性能 (FP32)	4.29TFLOPS	19.5TFLOPS × 2
OS		Linux 2.6	Linux 5.4
コンパイラ		gcc 4.4.7 nvcc 7.5	gcc 9.4.0 nvcc 11.6

3.2 実行時間

収束判定周期を 100 とし, A100 GPU にて Kepler モードと Ampere モードでそれぞれ実行した. 図 2 に実行時間を示す. 格子数 1920 において, K20 で 485.66 秒, A100 の Kepler モードで 75.90 秒, 同 GPU の Ampere モードで 75.73 秒となり, A100 は K20 比で約 6.4 倍の速度を達成することができた. 一方で, 同じ A100 GPU 同士で従来のコードを Kepler モードで実行した場合と, 明示的な warp 同期を取り入れた今回のコードを比較した場合では 1% 程度の差にとどまった.

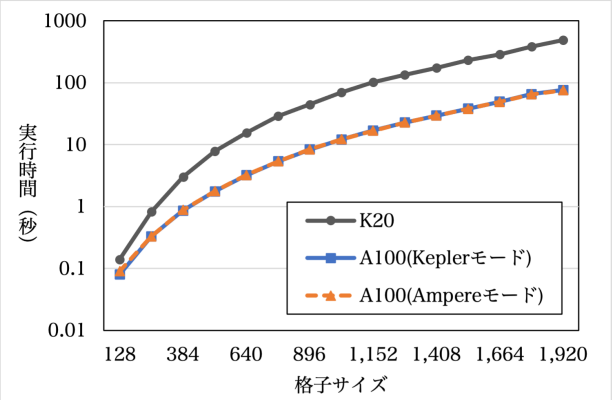

図 2 実行時間

4 おわりに

Kepler 世代 GPU 向け電位・電界シミュレーションを NVIDIA A100 GPU を対象に移植し, 性能評価を行った。K20 GPU と比較して飛躍的に性能が向上した一方で, 暗黙の同期を明示的な同期に置き換えることによる明確な性能向上は見られなかった. 今後の課題として, Ampere 世代で追加されたグローバルメモリからシェアードメモリへの非同期コピー [4] を用いた場合の性能評価や, 複数 GPU 環境への最適化などが挙げられる.

参考文献

[1] 松原翼, 長尾栄作, 上嶋明, 尾崎亮, 小畑正貴 : 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, pp. 1-73–1-74, 2015.

[2] 三木洋平 : Volta 世代の GPU における重力ツリーコードの性能評価, 情報処理学会研究報告, Vol. 2018-HPC-166, No. 6, pp. 1-9, 2018.

[3] 三木洋平 : NVIDIA A100 における重力ツリーコードの性能評価, 情報処理学会研究報告, Vol. 2021-HPC-180, No. 24, pp. 1-7, 2021.

[4] Pramod Ramarao : CUDA 11 Features Revealed (2020), `https://developer.nvidia.com/blog/cuda-11-features-revealed/`, (参照 2023-01-12).

[5] CUDA Toolkit Documentation, `https://docs.nvidia.com/cuda/archive/11.6.2/`, (参照 2023-01-12).