Technical Reports on Information and
Computer Science from Kochi
Vol. 9 (2017), No. 3

GPU を用いた FDTD 法計算高速化
— 二次元 FDTD 法の実装と評価 —

中村 哲也　　髙田直樹
高知大学理学部応用理学科情報科学コース

1　まえがき

電磁界シミュレーション手法である FDTD 法の差分計算はメモリアクセスがボトルネックとなる．FDTD 法の CUDA プログラムをパフォーマンスチューニングにより計算高速化を実現することを目的とする．また，提案手法の有効性を示すため，Kepler アーキテクチャの GPU を数種類用いて性能評価を行った．最終的に，Kepler アーキテクチャの GPU において，提案手法が有効であることが確認された．

2　GPU による二次元 FDTD 法差分計算の理論性能

2.1　GPU の最大理論性能

GPU の最大理論性能は，次式 (1) で表わされる．

$$\begin{aligned}[\text{最大理論性能}] =&[1\text{ コアが }1\text{ クロックで行う演算数}]\\ &\times[\text{コア数}]\times[\text{動作周波数}],\end{aligned} \tag{1}$$

NVIDIA GeForce GTX 680 の場合，その仕様 (表 1) より，式 (1) を用いて最大理論性能を求める．つまり，その最大理論性能は，2 演算 $\times 1,536$ コア $\times 1.006$ GHz $= 3.090$ TFLOPS となる．

2.2　メモリアクセスを考慮した FDTD 法の理論性能

二次元 FDTD 法 (TM 波) の差分式は式 (2)〜(4) となる．

$$\begin{aligned}H_x^{n+1/2}(i,j+1/2) = H_x^{n-1/2}(i,j+1/2)\\ -\frac{\Delta t}{\mu_0\Delta y}\left\{E_z^n(i,j+1)-E_z^n(i,j)\right\},\end{aligned} \tag{2}$$

$$\begin{aligned}H_y^{n+1/2}(i+1/2,j) = H_y^{n-1/2}(i+1/2,j)\\ +\frac{\Delta t}{\mu_0\Delta x}\left\{E_z^n(i+1,j)-E_z^n(i,j)\right\},\end{aligned} \tag{3}$$

$$\begin{aligned}E_z^{n+1}(i,j) = E_z^n(i,j)\\ -\frac{\Delta t}{\varepsilon_0\Delta y}\left\{H_x^{n+1/2}(i,j+1/2)-H_x^{n+1/2}(i,j-1/2)\right\}\\ +\frac{\Delta t}{\varepsilon_0\Delta x}\left\{H_y^{n+1/2}(i+1/2,j)-H_y^{n+1/2}(i-1/2,j)\right\},\end{aligned} \tag{4}$$

式 (2)〜(4) において $\Delta t/(\mu_0\Delta x)$, $\Delta t/(\mu_0\Delta y)$, $\Delta t/(\varepsilon_0\Delta x)$, $\Delta t/(\varepsilon_0\Delta y)$ をあらかじめ計算しておき定数とする．このとき，式 (2)〜(4) の演算数は 12 となる．一方，読み込むデータと書き込みデータは，式 (2), (3) において 5 個 ($H_x^{n-1/2}, H_x^{n+1/2}, H_y^{n-1/2}, H_y^{n+1/2}, E_z^n$)，式 (4) において 4 個 ($E_z^n$, E_z^{n+1}, $H_x^{n+1/2}$, $H_y^{n+1/2}$) となる．よって，ロード及びストアデータの総数は合計 9 個となる．

このことから，メモリアクセスを考慮した二次元 FDTD 法の最大理論性能は次式 (5) で表される．

表 1　NVIDIA GeForce GTX 680 の仕様

コアの動作周波数	1.006 GHz
コア数	1,536
メモリ量	2,048 MB
メモリバンド幅	192.2 GB/s
最大浮動小数点演算性能	3.09 TFLOPS

表 2　GPU のメモリバンド幅と FDTD 法の最大理論性能

型番	FDTD 法の最大理論性能	メモリバンド幅
GTX TITAN	96.13 GFLOPS	288.4 GB/s
GTX 680	64.07 GFLOPS	192.2 GB/s
GTX 670	64.07 GFLOPS	192.2 GB/s
GTX 660 Ti	48.07 GFLOPS	144.2 GB/s
GTX 650 Ti	28.80 GFLOPS	86.4 GB/s

$$\begin{aligned}&[\text{メモリアクセスを考慮した最大理論性能}]\\ &\quad=[\text{メモリバンド幅}]\div[\text{読み書きデータ数}]\\ &\qquad\times[\text{コアが行う演算数}],\end{aligned} \tag{5}$$

GeForce GTX 680 の場合，メモリアクセスを考慮した FDTD 法の理論性能 (単精度) は，式 (5) より，192.2 GB/s÷(9 × 4 B) × 12 = 64.07 GFLOPS となる．同様に，GeForce GTX 600 シリーズおよび GeForce GTX TITAN の FDTD 法の最大理論性能を表 2 に示す．GPU の最大理論性能に対し，メモリアクセスを考慮した FDTD 法の理論性能は低くなっている．これは，FDTD 法の差分計算が演算に比べ計算に使用するデータの読み書き処理が多いことを意味し，メモリバンド幅がボトルネックとなることを示している．

3　提案手法

3.1　GPU への FDTD 法の実装

FDTD 法で計算される電磁界データ (E_z, H_x, H_y) は, GPU ボードのオフチップメモリであるグローバルメモリに格納される．Fermi アーキテクチャ以降，グローバルメモリのデータを読み書きする際，キャッシュメモリを利用することが可能となった．なお，GPU 上で動作するプログラム (カーネル) において，キャッシュメモリを利用するための特別なプログラムの記述は必要ない．式 (4) の電界計算のカーネルをリスト 1 に示す．式 (2), (3) の磁界計算のカーネルにおいても特別な記述をせず，リスト 1 と同様な記述をしている．

リスト 1　電界計算のカーネル

```
__global__ void maxwEZ( float *HX, float *HY, float *EZ)
{
    unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
    unsigned int j = blockIdx.y*blockDim.y + threadIdx.y;

   EZ[j*NX+i]=EZ[j*NX+i]+dtdx*(HY[j*NX+i+1]-HY[j*NX+i])-
       dtdy*(HX[(j+1)*NX+i]-HX[j*NX+i]);
}
```

表 3　1 ブロックあたりの二次元スレッド $N_x \times N_y$ に対する計算時間
(計算領域: 12,288×12,288，時間ステップ: 1,000 step，使用 GPU: GeForce GTX 680)

512		256		128		64	
$N_x \times N_y$	計測時間 [秒]	$N_x \times N_y$	計測時間 [秒]	$N_x \times N_y$	計測時間 [秒]	$N_x \times N_y$	計測時間 [秒]
512 × 1	36.87	256 × 1	35.01	128 × 1	34.95	64 × 1	44.77
256 × 2	36.72	128 × 2	35.59	64 × 2	35.78	32 × 2	46.83
128 × 4	38.45	64 × 4	38.23	32 × 4	38.35	16 × 4	51.91
64 × 8	37.64	32 × 8	37.21	16 × 8	47.34	8 × 8	72.51
32 × 16	38.67	16 × 16	50.35	8 × 16	94.03	4 × 16	128.57
16 × 32	57.29	8 × 32	98.99	4 × 32	143.25	2 × 32	239.50
8 × 64	108.59	4 × 64	148.07	2 × 64	243.78	1 × 64	465.97
4 × 128	160.13	2 × 128	251.42	1 × 128	465.16		
2 × 256	263.79	1 × 256	473.07				
1 × 512	492.49						

3.2　提案手法

リスト 1 のプログラムを用いて，GPU の性能を十分に引き出す方法を提案する．本手法は次の 3 つの手順から成る．

STEP 1. GPU に二次元 FDTD 法を実装する．
STEP 2. パフォーマンスチューニングを行い，二次元 FDTD 法の計算を高速化させる．
STEP 3. パフォーマンスチューニングにより，十分な性能が引き出されているかを確認する．

提案手法の STEP2 に示すパフォーマンスチューニングの詳細を次に示す．ここで，FDTD 法の計算領域の大きさを $L \times L$ とした．

STEP 1. 電磁界データの配列を，$L \times L$ よりも大きな $(L + Padding) \times (L + Padding)$ とする．*Padding*[1] に対する計算時間を調べ，最適な値を得る．
STEP 2. 最適な *Padding* の値を用いて GPU ボードに搭載されたメモリで計算できる最大領域において，スレッドの値を変動させながら計算時間を測定し，最適なスレッドの値を割り出す．
STEP 3. STEP 2 で割り出した最適なスレッド値での計算が十分な計算速度を実現しているか確認する．十分な速度が出ていない場合は，少し小さめな領域で STEP 2 を再度繰り返す．

パフォーマンスチューニングの STEP 1 で行う *Padding* の結果を図 1 に示す．最適な *Padding* の値は 32 の倍数であることがわかった．STEP 2 で行ったスレッドの値に対する計算時間の結果を表 3 に示す．その結果，128×1 が最適であることがわかった．

4　性能評価

パフォーマンスチューニングを行い，最適なパラメータを使用した場合の計算速度 (GFLOPS 値) とメモリアクセスを考慮した FDTD 法の最大理論性能に対する実行効率を，それぞれ，図 2, 3 に示す．図 2 より，FDTD 法の最大理論性能の約 8 割の性能を示しており，本手法が有効であることが示された．図 2, 3 において，GeForce GTX 660 Ti については最大計算領域において計算速度が低下している．これは，GeForce

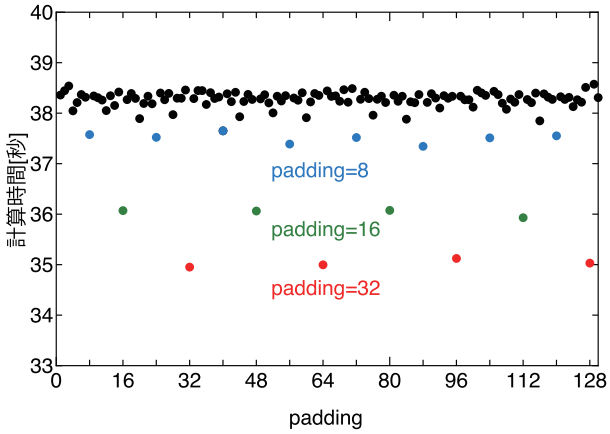

図 1　GeForce GTX 680 の *Padding* 調査結果

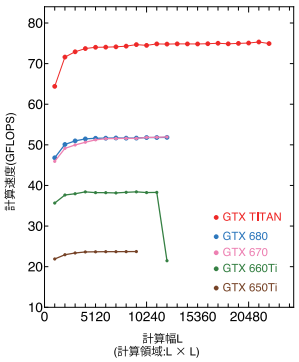

図 2　計算速度

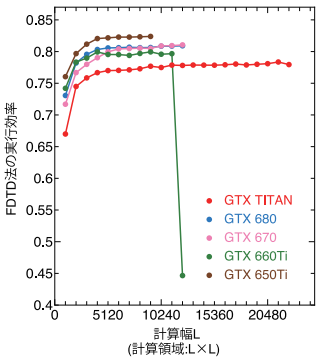

図 3　実行効率

GTX 660 Ti ではメモリコントローラが 3 基と少なくなっていることが原因であると考える．

5　まとめ

Kepler アーキテクチャの各 GPU において，メモリアクセスを考慮した FDTD 法の最大理論性能に対する実行効率を評価した．その結果，提案手法により GPU がもつ性能を十分に引き出すことが可能であることがわかった．

参考文献

[1] 高田直樹，“第 7 章 メモリアクセスがボトルネックとなる問題 -FDTD 法による電磁界シミュレーション”，GPU プログラミング入門-CUDA5 による実装 (伊藤智義編)，講談社，April, 2013.