C-009

シングル CUDA 互換 GPU を用いた FDTD 法の計算高速化
A Fast FDTD Simulation Using a Single CUDA-Compatible GPU

高田直樹† 下馬場朋禄‡ 増田信之‡ 伊藤智義‡
Naoki Takada Tomoyoshi Shimobaba Nobuyuki Masuda Tomoyoshi Ito

1. まえがき

FDTD（Finite-Difference Time-Domain）法[1]は，アンテナ解析など電磁界問題の解析に幅広く使用されている[2]．計算領域が大規模となる場合，または，解析モデルが複雑である場合には，使用する計算機メモリ量及び計算時間は膨大となる．これらは，FDTD 法において重要な問題となっている．この問題を克服するアプローチとして，PC クラスタを用いた分散並列 FDTD 法[3]～[6] が提案された．しかし，大規模な計算機メモリの確保と計算高速化を実現するためには，計算機システムは大規模となりメンテナンス性においても優れているとはいえない．

一方，GPU（Graphics Processing Unit）は，コンピュータグラフィックス(CG) 処理を行うために PC 内のグラフィックスボード上に搭載されており，一般的に利用されている．最新の GPU は 1 チップあたり約 1 TFLOPS の単精度浮動小数点演算の理論性能を持ち，CPU（Central Processing Unit）に比べ格段に優れている．当初，GPU を用いた一般的な数値計算の高速化に関する研究は，GPGPU（General Purpose Computation on GPU）と呼ばれた．3 次元 CG 処理のプログラム開発環境が使用されており，様々なシェーダ言語を用いて実装された．GPGPU の研究[7], [8] において，GPU を用いた FDTD 法の計算高速化に関する研究も盛んに行われた[9], [10]．FDTD 法の計算に使用する電磁界成分をテクスチャに格納し，様々なシェーダ言語を使用して GPU に実装された．

現在，GPU は CG 処理の専門知識を必要とせず，C に似た言語を用いて GPU 用プログラムを開発することが可能となっている[11][12]．GPU 用プログラム開発環境を用いた一般的な数値計算の高速化に関する研究[13], [14] は，GPU コンピューティングと呼ばれている．現在も GPU のアーキテクチャ及びプログラム開発環境は発展し続けており，GPU を用いた FDTD 法の計算高速化に関する研究は盛んに行われている[15]～[19]．

FDTD 法はスタッガード格子を用いており，電界及び磁界成分の座標位置は，時間及び空間に対してずらして配置される．そのため，著者らは電界及び磁界成分の計算において，それぞれ異なる副領域を用いた GPU への実装法を提案した[18][19]．本手法は副領域間の重複領域を低減させており，GPU 内の共有メモリを効率的に活用することが可能となっている．さらに，FDTD 法のシミュレーション時刻を GPU 内の特定のスレッドでカウントさせることにより，電界または磁界計算における境界条件の処理を低減させ計算速度を向上させている．しかし，FDTD 法の計算領域が大きくなるにつれ計算速度が低下する問題が確認された[18]．本論文では，この問題を改善することを目的とする．

† 湘北短期大学情報メディア学科
‡ 千葉大学大学院工学研究科

表 1 NVIDIA Geforce GTX 280 の仕様

Processor Clock	1.296 GHz
The number of Streaming Processors	240
Memory Size	1GByte
Memory Bandwidth	141.7 GByte/sec

本論文では，著者らの方法を用いて FDTD 法 をシングル GPU（NVIDIA Geforce GTX 280）に実装し，計算高速化について検討する．なお，GPU 用プログラム開発環境として CUDA[11]を使用した．一般的に，1 つのブロックに対して 2 次元スレッド 16×16 を使用して実装する方法[16][17]が利用されている．しかし，著者らの実装法を用いて計算性能を検討した結果，2 次元スレッド 16×16 では計算性能において最速となるサイズではないことが確認された．本手法にのみ起こる問題であるかを調べるため，共有メモリを用いない場合においても同様に検討した．その結果，著者らの方法と同様に 2 次元スレッド 16×16 では計算性能において最速となるサイズではないことが確かめられた．さらに，この場合でも計算領域が大きくなるにつれ計算速度が低下する現象が確認された．また，著者らの方法において計算速度が最も速くなる 2 次元スレッドサイズを割り出した．最速となる 2 次元スレッドサイズを用いた場合，計算領域の増加に対し，計算速度の低下は起こらず一定の速度を保つことができた．最終的に 1 つの GPU チップにおいて約 30 GFLOPS を達成した．CPU（Intel Core2Duo 3.0 GHz）のシングルコアを用いた場合と演算性能について比較した結果，計算領域 8, 192 × 8, 192 において約 113 倍の計算高速化を達成した．さらに，実測した GPU のメモリバンド幅の最大値をもとに導出したピーク性能（38.67 GFLOPS）に対して約 80%の性能を引き出していることが確認され，有効性が示された．

2. で GPU アーキテクチャについて述べる．3. で GPU による 2 次元 FDTD 法差分計算の理論性能について述べる．4. で著者らが提案している GPU への実装について述べ，GPU コードを示す．5. で計算高速化について検討し，有効性を示す．6. でまとめと今後の展開を述べる．

2. CUDA 互換 GPU

本論文では，GPU として NVIDIA 社の Geforce GTX 280 を使用した（表 1）．CUDA 互換 GPU ボードのブロック図を図 1 に示す．CUDA 互換 GPU ボードは，主に GPU チップとデバイスメモリから構成される．

GPU は，複数のマルチプロセッサ（MP）を持つ．1 つの MP 内に 8 個のストリームプロセッサ（SP）と共有メモリ（Shared Memory）が存在する．8 つの SP は同一の MP 内にある共有メモリにだけデータアクセスすることが可能である．MP 毎に，SIMD 処理がなされる．図 1 において，SP はデバイスメモリへアクセスするよりも，共有

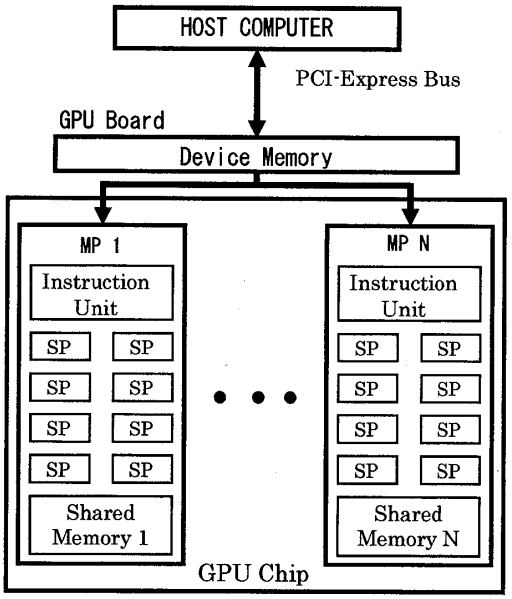

図 1　統合型シェーダを搭載した GPU ボード

メモリへアクセスするほうが格段に速い．また，各 SP はレジスタを持つ．

GPU で処理されるプログラムをカーネルと呼ぶ．カーネルは，PCI-Express バスを経由して，ホスト PC から GPU ボードに転送され，GPU 上でカーネルが動作する．CUDA プログム開発環境を用いた場合，GPU 上で行う処理はブロック，スレッド，グリッドと呼ばれる単位に分けられる．1 つの SP に割り当てられる処理をスレッド，スレッドのまとまりをブロックと呼び，1 つの MP に 1 つのブロックが割り当てられる．ブロックは最大で 3 次元のスレッド配列を含むことができる．同じサイズのブロックをまとめたものをグリッドと呼び，グリッドは最大で 2 次元のブロック配列を含むことができる．また，グリッドはホスト PC から GPU に実行を指令する単位であり，グリッド内の全スレッドは同じカーネルを実行する

3. GPU による 2 次元 FDTD 法差分計算の理論性能

本章では，GPU を用いて2 次元FDTD 法差分計算を行った場合の理論ピーク性能を求める．

GPU（NVIDIA Geforce GTX 280）の理論ピーク性能（単精度浮動小数点演算）を求める．1 つのSPあたり，積和算 2 オペレーションと乗算の1 オペレーションの合計3 オペレーションを 1 クロックで行われるとする場合，3 Operation/SP × 240 SP × 1.296 GHz = 933.12 GFLOPS となる．これは，メモリアクセスを考慮せず理想的なピーク性能である．

次に，メモリアクセスを考慮した2 次元FDTD 法差分計算を行う場合の理論ピーク性能を求める．2 次元FDTD 法（TM 波）の差分式は次式となる．

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

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

$$E_z^{n+1}(i,j) = E_z^n(i,j) - \frac{\Delta t}{\varepsilon_0 \Delta y}\left\{H_x^{n+1/2}(i,j+1/2) - H_x^{n+1/2}(i,j-1/2)\right\} + \frac{\Delta t}{\varepsilon_0 \Delta x}\left\{H_y^{n+1/2}(i+1/2,j) - H_y^{n+1/2}(i-1/2,j)\right\} , (3)$$

ここで，ε_0, μ_0 はそれぞれ真空中の誘電率，透磁率を，Δx, Δy は空間離散間隔を，Δt は時間離散間隔を示す．また，$E_z^n(i, j)$ は時刻 $n\Delta t$ における座標 (i, j)の z 方向電界成分 E_z を表し，他の電磁界成分についても同様に表す．

FDTD 法の式（1）～（3）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（1）～（3）は 12 オペレーションとなる．一方，ロードデータとストアデータは，式（1），（2）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（3）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．1 ワード（4 Byte）当たりのメモリバンド幅の理論性能は，141.7 GByte/sec ÷ 4 Byte/Word = 35.43 GWord/sec となる（表 1）．よって，メモリバンド幅の理論性能から導出された GPU のピーク性能は，35.43 GWord/sec × 12 Operation / 9 Word = 47.24 GFLOPS となる．

メモリアクセスを考慮した GPU のピーク性能（47.24 GFLOPS）は，GPU の理論ピーク性能である 933.12 GFLOPS より小さい．これは，メモリバンド幅がボトルネックとなり，本計算の理論ピーク性能は 47.24 GFLOPS であることを示している．

4. GPU への実装 [18] [19]

4.1 実装に用いる副領域

解析する計算領域（Computational Domain）の電磁界成分 E_z, H_x, H_y のデータは，大容量であるためデバイスメモリ内のグローバルメモリに格納する．グローバルメモリアクセスは共有メモリに比べ格段に遅い．そのため，本手法では共有メモリを CPU におけるキャッシュのような役割として使用する．計算領域を複数に分割し，GPU 内の MP に割り当てる．分割した領域を副領域（Subdomain）と呼ぶ．電磁界の差分計算を行う際，頻繁にアクセスする電磁界成分のデータを MP 内の共有メモリへ移動し，メモリアクセス時間を短縮させる．

著者らが提案している副領域を図 2 に示す．ここでは，計算領域を 9 つの副領域に分割する．図 2 の領域 5 に示した電界成分 $E_z(i, j)$ において，式（3）の計算を行うには領域 6,8 内の 2 つの磁界成分 $H_y(i+1/2, j)$, $H_x(i, j + 1/2)$ の値が必要となる．そのため，領域 5 の副領域に領域 6,8 と重複した領域（Overlapping Area）が必要となる．最終的に，領域 5 は領域 2，4，6，8 と隣接するため 4 つの重複領域が必要となる．しかし，重複領域が多くなるとグローバルメモリから共有メモリへのデータ移動に余分な時間を

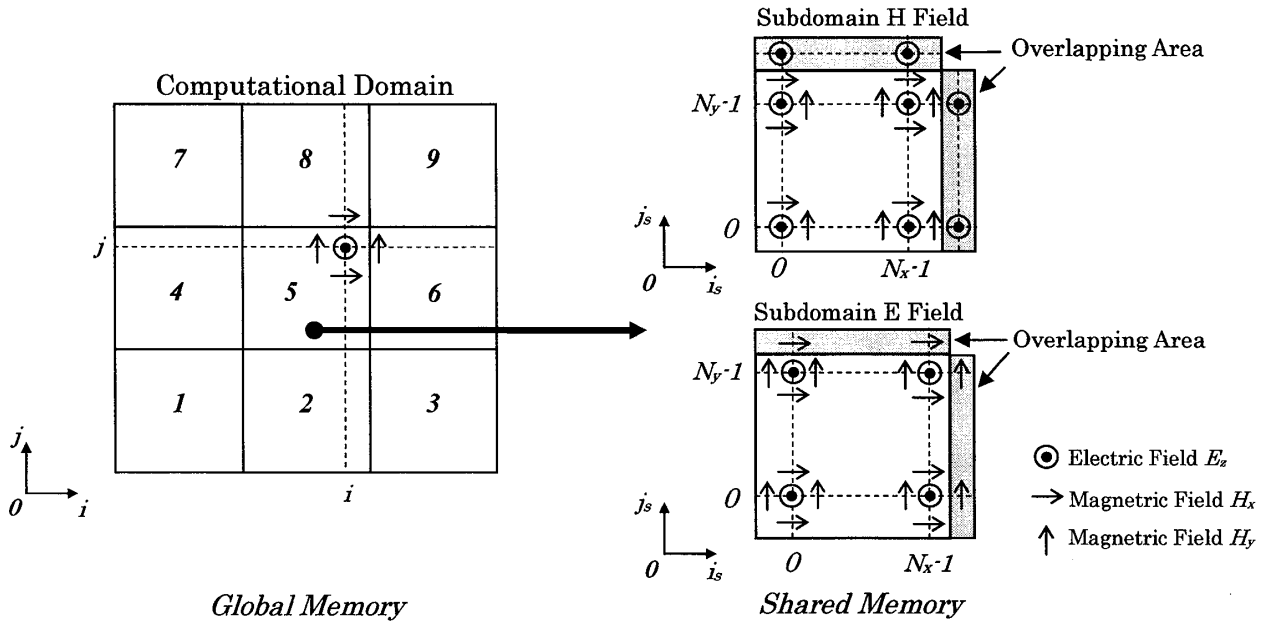

図 2　GPU への実装に用いる副領域

要してしまい計算速度は低下する．そのため，本手法では隣接する 2 つの領域のみ重複させる（図 2）．

CUDA プログラムにおいて，各副領域は GPU の MP に割り当てるためブロックとして扱う．1 つのブロックあたりの 2 次元スレッド配列を $N_x \times N_y$ とする．1 つのスレッドは 1 つの電界または磁界計算を行う．よって，各副領域において，計算される電磁界成分 E_z, H_x, H_y は，それぞれ $N_x \times N_y$ 個となる．

磁界計算において，式（1），(2) より，同じ電界成分 $E_z^n(i, j)$ のデータを 2 回読み込み，さらに 2 つの異なる座標の電界成分 $E_z^n(i, j+1)$, $E_z^n(i+1, j)$ のデータを読み込むことになる．電界成分データを合計 4 回読み込むことになり，電界成分のメモリアクセスは多くなる．本手法では，磁界計算を行う前に各ブロック内に存在する $N_x \times N_y$ 個のスレッドを用いて重複領域を含む磁界計算用副領域（Subdomain H field）内の全ての電界成分データをグローバルメモリから共有メモリに格納する．また，式（1），(2) で使用する電界成分 $E_z^n(i, j)$ をレジスタに格納する [15]．その後，共有メモリ及びレジスタのデータを用いて，磁界計算用副領域内の磁界成分 H_x, H_y の次ステップの値を，式（1），(2) により計算する．このようにして，本手法ではメモリアクセス時間を大幅に短縮させる．なお，磁界成分 $H_x^{n-1/2}(i, j+1/2)$, $H_y^{n-1/2}(i+1/2, j)$ の値は，一度しか計算に使用しないため共有メモリには格納しない．共有メモリに格納した場合，僅かではあるが格納に要する時間だけ計算が遅くなる．また，式（1），(2) の計算により求められた $H_x^{n+1/2}(i, j+1/2)$, $H_y^{n+1/2}(i+1/2, j)$ の値も，同様な理由により直接グローバルメモリに格納する．

電界計算は，磁界計算と同様に行う．本手法では，電界計算用副領域（Subdomain E field）と磁界計算用副領域とは異なる．これは，電磁界成分の位置座標が異なるためである．異なる副領域を用いることにより，効率よく共有メモリを使用することができる．

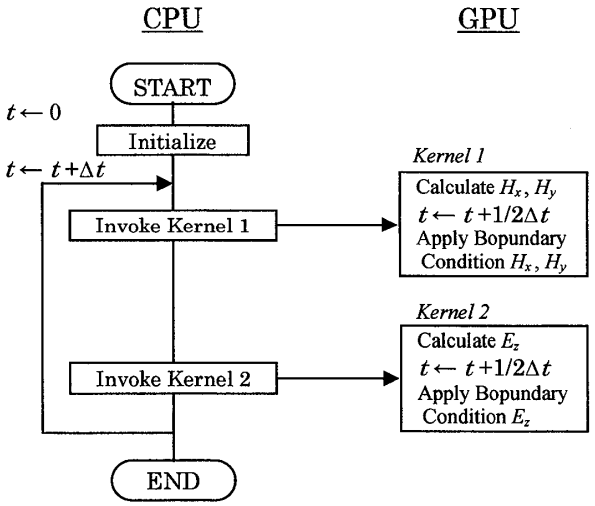

図 3　GPU による FDTD 法差分計算フローチャート

以上のように，2 次元 FDTD 法の電界及び磁界計算は異なる副領域を用いるため，CUDA プログラムにおいて 2 つのカーネルが必要となる．

4.2 GPU による計算の流れ

本手法のフローチャートは図 3 となる．“ Initialize ”において電磁界成分 E_z, H_x, H_y の配列を初期化し，GPU ボード上のデバイスメモリに割り当てられたグローバルメモリに確保する．ここで，スレッドがグローバルメモリにアクセスする際，メモリバンド幅の性能を十分発揮するためには，16 スレッドでメモリアクセスを結合させる必要がある [11][12]．結合させるには，単精度浮動小数点（4 Byte）の場合，配列を 64Byte（16 SP × 4 Byte）境界にアラインし，配列の大きさは 64 Byte の倍数となるようにする．

```
__global__ void maxwH( float *HX, float *HY, float *EZ)
{
  int tx=threadIdx.x;
  int ty=threadIdx.y;
  unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
  unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;
  __shared__ float SH_EZ[NY+1][NX+1];
  float xx;
  if(ty==(NY-1))SH_EZ[ty+1][tx]=EZ[(y+1)*w+x];
  if(tx==(NX-1))SH_EZ[ty][tx+1]=EZ[y*w+x+1];
  SH_EZ[ty][tx]=EZ[y*w+x];
  xx=SH_EZ[ty][tx];
  HX[(y+1)*w+x]=HX[(y+1)*w+x]-dtdy*(SH_EZ[ty+1][tx]-xx);
  HY[y*w+x+1]=HY[y*w+x+1]+dtdx*(SH_EZ[ty][tx+1]-xx);
}
```

図 4　磁界計算の GPU コード (kernel 1)

さらに，配列のメモリ上のアドレスが連続となるように配置する必要がある．グローバルメモリに確保する電磁界成分 E_z, H_x, H_y の配列はこの条件を満たすようにする．その後，CPU は式（1），(2）の磁界計算のカーネル（Kernel 1）を呼び出し，その計算を GPU で行わせる．なお，この磁界計算のカーネルにおいて，磁界 H の境界条件を処理する．FDTD 法の解析領域内の全磁界計算が終了した後，CPU は式（3）の電界計算のカーネル（Kernel 2）を呼び出し，GPU で行う．電界計算のカーネルにおいて，電界 E の境界条件を処理する．その後，CPU は，電磁界解析に必要な時刻になるまで，2 つのカーネルを繰り返し呼び出し，GPU 上で電磁界計算が行われる．電界 E 及び磁界 H の境界条件において，電磁波の波源のようにシミュレーションの時刻を境界条件で用いる場合がある．このときは，“ Initialize ”において，グローバルメモリ上に時刻を格納する変数を用意する．特定のスレッドにより時刻を GPU 内で計算し，それを境界条件に用いる．CPU で時刻の計算を行い GPU にそのデータを転送するよりも，特定のスレッドにより GPU 内でシミュレーションの時刻を管理するほうが計算時間は短縮する．

計算領域 1, 024 × 1, 024 の中心に正弦波の波源を置き，計算領域の終端を完全導体としたときの GPU コードを図 4, 図 5 に示す．図 5 において，境界条件の処理に条件分岐を使用した．ブロック毎に境界条件の有無が判別される．各ブロックは CUDA により自動的に効率よく MP に割り当てられ，MP 内において並列処理がなされることから，条件分岐による計算速度の低下は確認されなかった．

本章で述べた GPU での処理は，図 4，図 5 の GPU コードで全て行われる．本手法の GPU コードは短く，プログラム開発が容易である．

5.　計算高速化の検討

5.1 数値計算モデルと計算環境

本章では差分計算の高速化を評価するため，数値計算モデルとして基本的な 2 次元問題を扱った．計算領域の中心に正弦波の線波源を配置し，計算領域の終端を完全導体（$E_z = 0$）とした．時間ステップを 1,000 ステップとし，その計算時間を性能評価に使用した．GPU の計算と CPU のみの計算において，Intel Core2Duo E8400（3.0 GHz）の CPU と，2 GB のメインメモリ（DDR3-1333）を搭載した同じ PC を用いた．オペレーティングシステムとして，共に Linux（Fedora 9）を使用した．GPU として

```
__global__ void maxwEZ( float *HX, float *HY, float *EZ, float *T)
{
    int tx=threadIdx.x;
    int ty=threadIdx.y;
    unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
    unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;
    __shared__ float SH_HX[NY+1][NX+1];
    __shared__ float SH_HY[NY+1][NX+1];
    if(ty==(NY-1))SH_HX[ty+1][tx]=HX[(y+1)*w+x];
    if(tx==(NX-1))SH_HY[ty][tx+1]=HY[y*w+x+1];
    SH_HX[ty][tx]=HX[y*w+x];
    SH_HY[ty][tx]=HY[y*w+x];
    __syncthreads();
  if((x==511)&&(y==511)){
      T[0]+=dt;
    EZ[y*w+x]=am*sin(omega*T[0]);
  }else{
    EZ[y*w+x]=EZ[(y)*w+x]+dtdx*(SH_HY[ty][tx+1]-SH_HY[ty][tx])
                          -dtdy*(SH_HX[ty+1][tx]-SH_HX[ty][tx]);
  }
    if(x==0)EZ[(y)*w+x]=0;
    if(x==1023)EZ[(y)*w+x]=0;
    if(y==0)EZ[(y)*w+x]=0;
    if(y==1023)EZ[(y)*w+x]=0;
}
```

図 5　電界計算の GPU コード (kernel 2)

NVIDIA GeforceGTX 280 を使用した（表 1）．

本手法による GPU 計算プログラムは C 言語ベースで記述し，NVIDIA 社の提供する CUDA 2.1 を用いてコンパルした．従来の FDTD 法による CPU のみの計算では，プログラムを C 言語で記述し，Gnu C コンパイラ（バージョン 4.3.0）を用いた．なお，コンパイラオプションとして“ -march=core2 -msse-mfpmath=sse -O3 ”を使用した．CPU のみの計算は，CPU のシングルコアにより計算がなされ，SSE 命令が使用されていることを確認した．本手法による計算高速化において，計算処理速度（FLOPS 値）を用いて評価をした．計算領域 $L \times L$ において，時間ステップとして N_{itr} ステップ計算したときの計算時間を T_{GPU} とすると，式（1）～（3）の演算数は 12 オペレーションより，FLOPS 値は 12 Operation× $L \times L \times N_{itr} \div T_{GPU}$ となる．

5.2 計算速度の検討

GPU を用いた FDTD 法の計算では 1 ブロックあたりの総スレッド数を 256 スレッドとし，2 次元スレッド 16×16 で計算されることが一般的となっている[16][17]. そのため，本実装による GPU の計算において同じ条件で性能評価した．本論文で使用した GPU ボードのデバイスメモリの容量は 1 GByte より，性能評価において最大の計算領域の大きさを 8, 192 × 8, 192（メモリ使用量：768 MByte）とした．計算領域（$L \times L$）に対する本手法による GPU の計算時間及び FLOPS 値を表 2 に示す．表 2 より，FLOPS 値は計算領域 2, 048 × 2, 048 において 28.24 GFLOPS となりピーク値を示している．しかし，計算領域が大きくなるにつれて FLOPS 値は低下し，計算領域 8, 192 × 8, 192 において，20.0 GFLOPS となった．共有メモリを使用しない場合についても試したが，同様に，計算領域が大きくなるにつれ FLOPS 値が低下する現象が確認された．

2 次元配列の電磁界成分（E_z, H_x, H_y）の値をグローバルメモリ及び共有メモリに格納する場合，x 軸方向については連続に配置され，逐次的にメモリアクセスがなされ高速である．一方，y 軸方向については連続に配置されず，格納されたデータのメモリアドレス距離は計算領域が大

表 2 GPU による 2 次元 FDTD 法差分計算処理速度
（2 次元スレッド：16×16，計算領域：$L \times L$，時間ステップ： 1,000 step）

L	1,024	2,048	3,072	4,096	5,120	6,144	7,168	8,192
Computation time (ms)	461.14	1, 782.54	4, 085.39	7, 481.69	12, 482.39	19, 381.07	28, 579.79	40, 256.19
Computation speed (GFLOPS)	27.29	28.24	27.72	26.91	25.20	23.37	21.57	20.00

表 3 1 ブロックあたりの 2 次元スレッド $N_x \times N_y$ に対する計算時間
（計算領域：8, 192×8, 192，時間ステップ： 1, 000 step）

Total number of threads per a block							
512		256		128		64	
$N_x \times N_y$	Time (ms)	$N_x \times N_y$	Time (ms)	$N_x \times N_y$	Time (ms)	$N_x \times N_y$	Time (ms)
512 × 1	34,910.91	256 × 1	30,332.19	128 × 1	29,208.40	64 × 1	31,378.07
256 × 2	27,860.34	128 × 2	26,828.33	64 × 2	26,546.84	32 × 2	31,021.64
128 × 4	27,023.15	64 × 4	26,404.10	32 × 4	26,193.54	16 × 4	32,172.59
64 × 8	28,329.08	32 × 8	27,766.32	16 × 8	31,641.55	8 × 8	49,528.76
32 × 16	31,408.21	16 × 16	40,315.71	8 × 16	81,558.07	4 × 16	151,044.05
16 × 32	43,464.29	8 × 32	92,666.19	4 × 32	179,097.02	2 × 32	334,621.00
8 × 64	99,550.48	4 × 64	201,960.70	2 × 64	462,255.13	1 × 64	1,156,390.13
4 × 128	240,642.34	2 ×128	513,940.38	1 ×128	1,135,948.50		
2 × 256	531,257.31	1 × 256	1,137,908.38				
1 × 512	1,142,198.88						

表4 GPU 計算時間(T_{GPU}) とCPU のみの計算時間(T_{CPU}) の比較
（2次元スレッド：32×4，計算領域：$L \times L$，時間ステップ： 1,000 step）

L	1,024	2,048	3,072	4,096	5,120	6,144	7,168	8,192
T_{CPU} (sec)	40.78	143.87	330.71	593.91	990.17	1481.69	2117.63	2955.00
T_{GPU} (sec)	0.44	1.67	3.73	6.57	10.22	14.85	20.05	26.19
Speedup Factor (T_{CPU} / T_{GPU})	91.92	86.19	88.68	90.44	96.85	99.76	105.62	112.81

表5 GPU を用いた2 次元FDTD 法の計算性能
（2次元スレッド：32×4，計算領域：$L \times L$，時間ステップ： 1,000 step）

L	1,024	2,048	3,072	4,096	5,120	6,144	7,168	8,192
Computation Speed (GFLOPS)	28.36	30.15	30.37	30.66	30.77	30.77	30.75	30.74
vs. Actual Peek Performance	0.73	0.78	0.79	0.79	0.80	0.79	0.80	0.80

きくなるほど遠くなりメモリアクセスは遅くなる．x 軸方向の差分計算は y 軸方向に比べ高速に処理されることが考えられる．したがって，1 ブロックあたりの 2 次元スレッド $N_x \times N_y$ において，N_x が N_y より大きいときに計算高速化される可能性がある．また，1 ブロックあたりの総スレッド数は最大で 512 である．総スレッド数により計算速度が異なる可能性がある．以上のことより，計算領域 8, 192 × 8, 192 において，総スレッド数及び 1 ブロックあたりの 2 次元スレッド $N_x \times N_y$ に対する計算時間を測定した（表 3）．表 3 の結果より，1 ブロックあたりの 2 次元スレッド $N_x \times N_y$ において N_x, N_y の組み合わせにより計算時間は大きく異なることが確認された．また，N_x と N_y は計算時間において対称性は成り立たず，N_x が N_y よりも大きいときに計算が速くなることが確認された．さらに，1 ブロックあたりの 2 次元スレッド 16 × 16 ではピーク性能となっておらず，1 ブロックあたりの 2 次元スレッドが 32 × 4（総スレッド数 128）であるとき，本手法において最速となることがわかった．

5. 3 性能評価

5. 2 で割り出した最適な条件（1 ブロックあたりの 2 次元スレッド 32×4（総スレッド数 128））を用いて，本手法による GPU を用いた 2 次元 FDTD 法差分計算の性能評価を行った．本手法による GPU の計算時間（T_{GPU}），従来の FDTD 法による CPU のみの計算時間（T_{CPU}），及び，計算高速化比（T_{CPU} / T_{GPU}）を表 4 に示す．計算領域 8, 192× 8, 192 において，本手法による GPU の計算は CPU のみの計算に比べ約 113 倍計算高速化なされた．本手法による GPU の計算処理速度（FLOPS 値）を表 5 に示す．計算領域の大きさに関わらず約 30 GFLOPS を達成した．CUDA 2.1 のテストプログラムを用いてメモリバンド幅を測定したところ，実測のピーク性能は 116 GByte/sec であった．この実測したメモリバンド幅より求めた実際のピーク性能（Actual Peek Performance）は，116 GByte/sec ÷ 4 Byte × 12 Operation ÷ 9 Word = 38.67 GFLOPS となる．このピーク性能に対する本手法の GPU 計算性能の割合を表 5 に示す．実測したメモリバンド幅をもとに導出したピーク性能に対して，本手法による GPU を用いた 2 次元 FDTD 法の差分

計算は約 80%の性能を引き出していることが確認された．なお，1,000 ステップにおいて，本手法による GPU 計算の結果と CPU のみの計算結果を比較したところ良く一致し単精度を保持した．

さらに，表 4 より，計算領域 8, 192 × 8, 192 において 1 秒あたり約 38 ステップ計算が行われる．リアルタイムで可視化をするには，計算結果を毎秒 30 フレームでディスプレイ表示しなければならない．本来，GPU は CG 処理を行うものであり計算結果を CPU を利用せず，直接ディスプレイ表示することができる．よって，本手法によりシングル GPU で計算領域 8, 192 × 8, 192 の大きさまで，可視化を含むリアルタイムシミュレーションを実現できる可能性があることが示された．

6. むすび

共有メモリを効率よく活用した著者らの方法により FDTD 法を CUDA 互換 GPU へ実装した．さらに，1 ブロックあたりの 2 次元スレッドの最適値を実測により割り出した．その最適値を用いて GPU により計算を行ったところ，計算領域の大きさに関わらず約 30 GFLOPS の計算性能を達成した．実測したメモリバンド幅より導出したピーク性能に対し，本手法は約 80%の性能を引き出していることが確認された．計算領域 8, 192 × 8, 192 において，従来の FDTD 法による CPU のみの計算に比べ最大 113 倍の計算高速化を実現した．

3 次元の場合，演算数とロード・ストアデータ数の比の値は 2 次元の場合よりも大きい．電磁界計算において，同じ電磁界成分の値を使用する頻度は 2 次元よりも多く本手法におけるレジスタの活用は 3 次元においても有効である．よって，本手法を 3 次元に適用した場合，2 次元よりも計算高速化がなされる可能性は高い．今後，本手法を 3 次元 FDTD 法に適用する．

謝辞

本研究の一部は、文部科学省科学研究費補助金若手研究（B）（課題番号 20700053）による．

参考文献

[1] K. S. Yee, “Numerical solution of initial boundaryvalue problems involving Maxwell’s equations inisotropic media,” IEEE Trans. Antennas Propag., vol.14, no.3, pp.302-307, MAY 1996.

[2] A. Taflove and S. C. Hagness, Computational Electrodynamics: The Finite-Difference Time-Domain Method, 3rd ed., Artech House, Norwood, MA, 2005.

[3] D. P. Rodohan, S. R. Sunders, and R. J. Glover,“A distributed implementation of the finite differencetime-domain (FDTD) method,” Int. J. Numerical Modelling Electronic Networks, Devices and Fields, vol.8, pp.283-291,1995.

[4] 高田直樹, 安藤勝則, 本島邦行, 伊藤智義, 上崎省吾, “新たな分散 FDTD 法アルゴリズム,” 信学論（C-I）, vol.J80-C-I, no.2, pp.47-54, Feb. 1997.

[5] 高田直樹, “分散 FDTD 法による計算高速化,” 信学技報, AP98-128, pp.43-50, 1998.

[6] 高田直樹, 分散 FDTD 法と計算高速化に関する研究博士学位論文, 群馬大学, 2000.

[7] N. Masuda, T. Ito, T. Tanaka, A. Shiraki and T. Sugie, “Computer generated holography using a graphics processing unit,” Optics Express, vol.14, No.2, pp.587-592, Jan. 2006.

[8] NVIDIA, GPU Gems 3, Addison-Wesley,2007.

[9] M. J. Inman and A. Z. Elsherbeni, “Programming video cards for computational electromagnettics application-n,”IEEE Antennas and Propag-ation Magazine, vol.47, no.6, pp.71-78, Dec. 2005.

[10] N. Takada, N. Masuda, T. Tanaka, Y. Abe, and T. Ito, “A GPU implementation of the 2-D finite difference time-domain code using high level shader language,” ACES J., vol.23, no.4, Dec. 2008.

[11] NVIDIA, NVIDIA CUDA Computational unified device architecture programming guide version 2.1, NVIDIA, 2008.

[12] 青木尊之, 額田彰, はじめての CUDA プログラミング, 工学社, 2009.

[13] 下馬場朋禄, 伊藤智義, 杉江崇繁, 増田信之, 阿部幸男, 白木厚司, 市橋保之, 高田直樹, “統合型シェーダを搭載した GPU によるフレネル回折積分の高速化,” 信学論（D）, vol.J90-D, no.9, pp.2656-2658, Sept. 2007.

[14] T. Shimobaba, T. Ito, N. Masuda, Y. Abe, Y. Ichihashi, H. Nakayama, N. Takada, A. Shiraki and T. Sugie, “Numerical calculation library for diffraction integrals using the graphic processing unit: the GPUbased wave optics library,” Journal of Optics A: Pure and Applied Optics, Vol.10, 075308 (5pp), 2008.

[15] 高田直樹, 滝沢努, 宮兆喆, 増田信之, 伊藤智義, 下馬場朋禄, “統合型シェーダを搭載した GPU による二次元 FDTD 法差分計算の高速化,” 信学論（D）, vol.J91-D, No.10, pp.2562-2564, Oct. 2008.

[16] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W. W. Hwu, “Optimization principles and application performance evaluation of a multithreaded GPU using CUDA,” In PPoPP ’08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, 2008.

[17] P. Sypek, A. Dziekonski, and M. Morozowski, “How to Render FDTD Computation More Effective Using a Graphics Accelerator,” IEEE Trans. Magn., vol. 45, no.3, pp.1324-1327, March 2009.

[18] N. Takada, T. Shimobaba, N. Masuda, and T. Ito,“High-Speed FDTD simulation algorithm for GPU with compute unified dvice architecture,” Proc. 2009 IEEE AP-S Int. Symposium and USNC/URSI National Radio Science, June 2009 (Invited).

[19] N. Takada, T. Shimobaba, N. Masuda, and T. Ito, “The Performance Evaluation of CUDA Implementation for Fast FDTD Computation Using a Single GPU,” Proc. The 26th International Review of Progress in Applied Computational Electromagnetics, April 2010 (Invited).