C-013

ＧＰＵによる共有メモリを効率的に用いた ＦＤＴＤ法差分計算の高速化

A Fast FDTD Simulation Using Shared Memory Efficiently with 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，電磁界問題の解析に最も利用されている電磁界シミュレーション手法である．本手法は，解析領域の電磁界成分を時間及び空間に対して離散化し，Maxwell 方程式に直接中心差分を適用して導出された差分方程式をもとに，時間に対して空間の電磁界成分を求める陽解法である．境界条件の取り扱いが容易であり，実験モデルを簡単に計算モデル化できることから，FDTD 法の計算結果と実験値はよく一致する．さらに，パルス応答を FDTD 法により計算し，各時刻の計算結果にフーリエ変換を適用することにより周波数特性を求めることが可能である．このことから，FDTD 法は電磁界散乱問題，アンテナ，マイクロ波回路など，幅広い電磁界問題の解析に使用される[2][3]．しかしながら，FDTD 法は解析領域を空間について離散化する手法であることから，使用する計算機メモリ量は膨大となる．さらに，多重散乱が起こるような電磁界問題や周波数特性を計算する場合には計算時間が膨大となってしまう．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 法が提案されている[4][5]． しかし，ＰＣクラスタは高価であり，大規模なシステムとなってしまう．

一方，近年，３次元コンピュータグラフィックス(3-D CG)の急速な発展に伴い，3-D CG 処理を高速化させるために開発された GPU （Graphics Processing Unit）の演算性能は著しく向上している．さらに，実際の物理現象を再現したゲームのニーズが高まっていることから，GPU は CG 処理以外に汎用的な数値計算を行うことが可能になっている．ＧＰＵ は，ＰＣ 用のグラフィックスボード上に搭載されている．現在，GPU は単精度浮動小数点演算が可能であり，その演算性能は CPU（Central Processing Unit）に比べ格段に優れている．GPU を用いて一般的な数値計算を高速化させる研究は GPGPU（General Purpose on GPU）と呼ばれ，その研究は盛んに行われている[6]．当初，GPU の研究には，CG 処理のプログラム開発環境が使用されていた．数値計算の実装には，様々なシェーダ言語が使用され実装された．現在，GPU のアーキテクチャは汎用的な数値計算において計算高速化なされるように改善されており，その開発環境も 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できる[7].

GPU はコストパフォーマンスに優れていることから，実用化に向け 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[8][9][10][11]．当初，FDTD 法で計算に使用する電磁界成分をテクスチャに格納し，シェーダ言語を使用して GPU に実装された[8][9][10]．Baron らは，シェーダ言語に NVIDIA 社の Cg[12]を使用している．しかし、従来の FDTD 法を CPU で計算した結果と比べ，時間ステップ数に対して誤差が増大している[8]．一方，Inman らはシェーダ言語として Brook[13]を使用しているが，CPU の計算結果と比べ大きな誤差が存在する[9]．高田らが提案した手法[10]はシェーダ言語に Microsoft 社の HLSL を使用した．CPU の計算結果と一致し，GPU を用いた FDTD 法の差分計算において単精度を保持することを示した．また，最新の GPGPU 用のプログラム開発環境として，NVIDIA 社から CUDA （Compute Unified Device Architecture）開発環境[7]が提供されている．CUDA を用いた FDTD 法の計算高速化について報告されている[11]．しかし，磁界計算にのみ共有メモリを使用しており、さらなる計算高速化が期待できる．

本論文では，効率良く共有メモリを利用することにより FDTD 法の差分計算を高速化させるアルゴリズム[14]を CUDA 互換 GPU に実装し，その性能を評価した．２次元計算モデルを使用し，計算高速化及び誤差について検討する．最終的に，本手法により従来の CPU を用いた FDTD 法に比べ、最大約 20 倍の計算高速化を実現した．

２．２次元 FDTD 法

本章では，Yee により考案された２次元 FDTD 法について簡単に述べる[1]．真空中の３次元 Maxwell 方程式において，電界 $\boldsymbol{E}$ の x，y 方向成分，磁界 $\boldsymbol{H}$ の z 方向成分を 0 とし，TM 波の２次元 Maxwell 方程式を得る．これらの式において時間と空間に対して離散化し，中心差分を用いて２次元 Maxwell 方程式の差分式を導出する．これらは，TM 波の２次元 FDTD 法の差分式であり，次式となる．

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

† 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

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

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

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 を表しており，他の電磁界成分についても同様に表す．

FDTD 法は，式(1)～(3)の差分式を用いて，初期値から磁界と電界を交互に計算し，電磁界の時間的変化を計算するシミュレーション手法である．また，この計算が時間に対して解が発散せず，安定性を保つためには，ノイマンの安定性条件より，時間離散間隔は次式を満たさなければならない．

$$\Delta t = \frac{1}{C_0\sqrt{\Delta x^{-2} + \Delta y^{-2}}} \quad , \qquad (4)$$

ここで，C_0 は真空中の光速を示す．

FDTD 法を用いて一般的な電磁界問題を解析する場合，境界条件を取り扱わなければならない．主な，境界条件として散乱物体の境界条件，媒質の誘電率及び透磁率がある．媒質の誘電率及び透磁率が空間に対して変化する場合は，式(1)～(3)において，誘電率，透磁率の値を空間に対する配列に格納し計算する．アンテナの問題などでは，電磁波の波源を境界条件として扱う．開放系の解析領域を扱う場合，解析領域の終端に吸収境界条件が必要である．FDTD 法では，これらの境界条件をまとめて，磁界 $\boldsymbol{H}$ の境界条件，電界 $\boldsymbol{E}$ の境界条件として取り扱う．TM 波の２次元 FDTD 法のフローチャートを図 1 に示す．図 1 において，次ステップの磁界成分 H_x, H_y を計算したあと，時刻を $1/2\Delta t$ 進めて，磁界 $\boldsymbol{H}$ の境界条件を適用する．その後，次ステップの電界成分 E_z を計算し，時刻を $1/2\Delta t$ 進めて電界 $\boldsymbol{E}$ の境界条件を適用する[3].

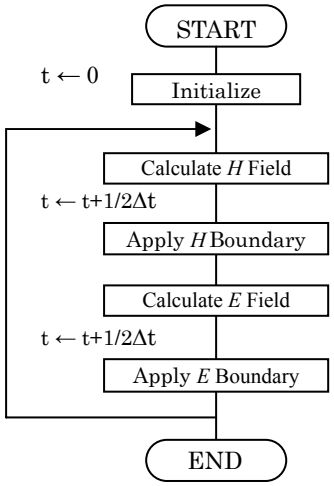

図 1　FDTD 法フローチャート

表 1　使用した GPU の仕様

GPU	NVIDIA Geforce GTX 280
Processor Clock	1.296 GHz
Number of Streaming Processor	240
Memory Bandwidth	141.7 GB/sec

3. 統合型シェーダを搭載した GPU

GPU は，３次元 CG を高速化することを目的として多くのプロセッサが搭載されている．従来の GPU は，頂点シェーダ（Vertex Shader），ラスタライザ，ピクセルシェーダ（Pixel Shader）から構成されていた．最初に，頂点シェーダにより描画するポリゴンの頂点座標を計算し，ラスタライザにより頂点座標をもとに描画するピクセルを生成する．その後，ピクセルシェーダにより各ピクセルの色を計算する．３次元 CG 処理は、このようなパイプライン処理により行われていた．頂点シェーダ及びピクセルシェーダは，プログラムにより単精度浮動小数点演算を行うことが可能であり，その演算は SIMD（Single Instruction Multiple Data）処理がなされた．GPU を用いて一般的な数値計算を行う場合，ピクセルシェーダのプロセッサ数は，頂点シェーダに比べて多いことから主にピクセルシェーダを用いて行われていた．

GPGPU の研究が急速に発展したこともあり，最新の GPU は頂点シェーダとピクセルシェーダが統合され，単に，多数の浮動小数点演算プロセッサから構成されている．頂点及びピクセルの計算量に合わせ，使用するプロセッサ数が可変する．これにより，一般的な数値計算を行う場合，GPU 内の浮動小数点演算プロセッサを全て用いて並列計算することが可能となった．このような，GPU のアーキテクチャを統合型シェーダと呼ぶ．

本論文では，統合型シェーダを搭載した GPU として，NVIDIA 社の Geforce GTX 280 を使用した（表 1）．NVIDIA 社の統合型シェーダを搭載した GPU ボードのブロック図を，図 2 に示す．NVIDIA 社の統合型シェーダを搭載した 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 2 において，SP はデバイスメモリへアクセスするよりも，共有メモリへアクセスするほうが格段に速い．

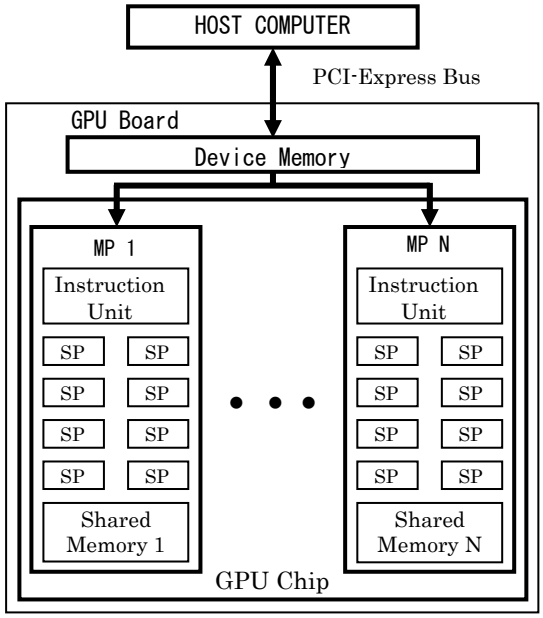

図 2　統合型シェーダを搭載した GPU ボード

統合型シェーダを搭載した GPU のプログラミング環境として NVIDIA 社より CUDA プログラミング環境が提供されている．これは，GPGPU 向けのプログラミング環境であり CG 処理に依存せず，C に似た言語で GPU 用プログラムを記述することができる．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 プログラミング環境を用いた場合，GPU 上で行う処理はブロック，スレッド，グリッドといった単位に分けられる．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．

4. 提案手法

統合型シェーダを搭載した GPU を用いて 2 次元 FDTD 法の差分計算を高速化する方法を提案する．GPU において，デバイスメモリの容量は共有メモリに比べて大きい．例えば，NVIDIA 社の Geforce GTX 280 では，デバイスメモリの容量は 1GB であり，共有メモリは各 MP において 16KB である．FDTD 法の計算領域（Computational Domain）は，実空間を離散化しているため大きくなる．そのため，計算領域の電磁界成分を GPU ボードのデバイスメモリ内のグローバルメモリ（Global Memory）に保存する．

FDTD 法での計算領域内における電磁界成分 E_z，H_x，H_y の計算は，スレッドによって行われる．計算領域をブロック毎に分割した図を，図 3 に示す．ここでは，計算領域を 9 つに分割する．各ブロックは，2 次元のスレッド配列 $N_{th} \times N_{th}$ とする．すなわち，1 つのブロックは $N_{th} \times N_{th}$ 個のスレッドからなる．1 つのブロックにつき最大 512 スレッドまで利用できる．ここでは，1 つのブロックのスレッド配列を 16 ×16 とする．よって，分割された各領域（領域 1～9）で計算される電磁界成分 E_z，H_x，H_y は，16 ×16 個となる．各ブロック内に存在するスレッドは，デバイスメモリ上に割り当てられたグローバルメモリにアクセスすることができるが低速である．一方，MP には共有メモリが存在し，同一ブロック内におけるスレッド間で共有メモリ上のデータを共有することができる．ブロック内のスレッドは，グローバルメモリ上のデータにアクセスするよりも，共有メモリ上のデータにアクセスするほうが格段に速い．よって，計算領域内の領域 1～9 において，式(1),(2)の計算を行う際，頻繁に使用する電磁界成分 E_z，H_x，H_y の値を，式(1),(2)の計算を行う前にグローバルメモリから共有メモリに移し，その後，式(1),(2)の計算を行えば，データアクセス時間を大幅に短縮することができ，結果的に計算時間を大幅に短縮することができる．しかし，図 3 において，計算領域内の領域 5 に存在する時刻 (n+1)Δt の電界成分 $E_z(i, j)$の値を式(3)により計算する場合，領域 8 に存在する時刻 (n+1/2)Δt の磁界成分 $H_x(i, j+1/2)$の値と，領域 6 に存在する時刻(n+1/2)Δt の磁界成分 $H_y(i+1/2, j)$の値が必要となる．領域 6,8 は異なるブロックであるため領域 5 のブロック内のスレッドがアクセスできる共有メモリには存在しない．したがって，領域 5 内の電磁界成分 E_z，H_x，H_yの次ステップの値を式(1)，(2)を用いて計算する場合，必要となる領域 6,8 の電磁界成分 E_z，H_x，H_y を領域 5 のブロック内で使用できる共有メモリに格納する必要がある．ここで，領域 5 のブロック内で使用できる共有メモリに 格納する電磁界成分 E_z，H_x，H_y の領域を副領域（Subdomain）と呼ぶ．領域 5 に対応する副領域に領域 6,8 と重複する領域（Overlapping Area）を持たせなければならない．このように，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．

さらに，本手法では，図 3 に示すように式(1),(2)の磁界計算で使用する領域 5 に対応する磁界計算用副領域(Subdomain H Field)と，式(3)の電界計算で使用する領域 5 に対応する電界計算用副領域(Subdomain E Field）は異なる．これは，電磁界成分 E_z，H_x，H_y の座標位置が異なることによる．磁界計算において，式(1),(2)は，同じ電界成分 $E_z^n(i, j)$と 2 つの異なる座標の電界成分 $E_z^n(i, j+1)$，$E_z^n(i+1, j)$ のデータを必要とする．つまり，磁界計算において，3 回メモリアクセスすることになる．よって，各時刻において磁界計算を行う前に，一度だけ各ブロック内に存在する 16 ×16 のスレッドを用いて重複領域を含む磁界計算用副領域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

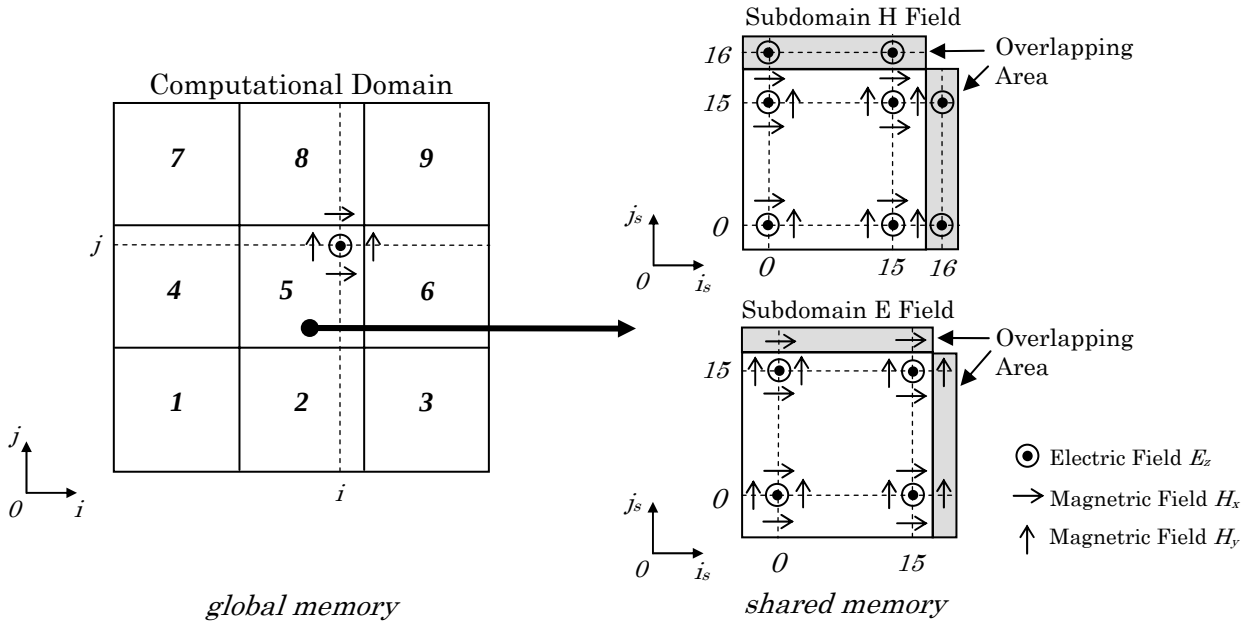

図 3　提案手法で用いる副領域

レジスタに格納する．この後，式(1),(2)により，磁界計算用副領域内に存在する 16 ×16 個の磁界成分 H_x, H_y の次ステップの値を計算する．ここで，磁界計算用副領域内に存在する $H_x^{n-1/2}(i, j+1/2)$，$H_y^{n-1/2}(i+1/2, j+1)$の値は共有メモリに格納しない．これは，各時刻において一度しか計算に使用しないためである．実際，共有メモリに格納すると少しではあるが計算が遅くなる．また，式(1),(2)の計算により求められた $H_x^{n+1/2}(i, j+1/2)$，$H_y^{n+1/2}(i+1/2, j+1)$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．電界計算用副領域についても，磁界計算用副領域の処理と同様に行う．以上のことから，２次元 FDTD 法の電界計算及び磁界計算は異なる処理を GPU で行うため，2 つのカーネルが必要となる．

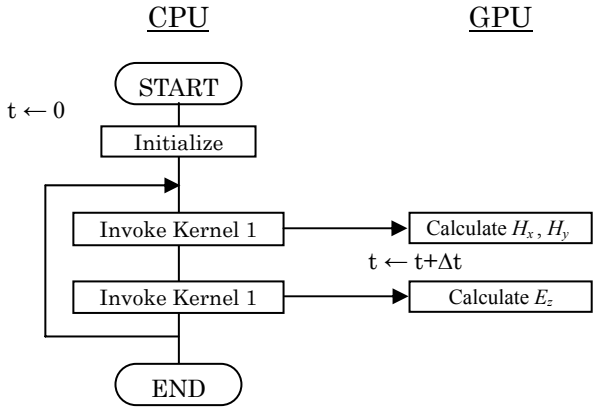

図 4　提案手法のフローチャート

本手法のフローチャートは図 4 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 ボードのグローバルメモリ上に確保する．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（Coalescing）させる必要がある．メモリアクセスを結合するためには，単精度浮動小数点の場合，配列を 64Bytes（16 スレッド×4Bytes）境界にアラインし，配列の大きさは 64Bytes の倍数となるようにする．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． GPU ボード上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を GPU に転送し，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 $\boldsymbol{H}$ 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を GPU に転送し，その計算を GPU で行う．磁界計算のカーネルと同様に，電界計算のカーネルにおいて，電界 $\boldsymbol{E}$ の境界条件を処理し FDTD 法の解析領域内の全電界計算を行う．その後，CPU は，電磁界解析に必要な時刻になるまで，磁界計算のカーネルと電界計算のカーネルの起動を繰り返し，電磁界計算を繰り返し GPU で行う．電界 $\boldsymbol{E}$ 及び磁界 $\boldsymbol{H}$ 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，時刻を CPU から GPU に転送するのではなく，“Initialize”において，グローバルメモリ上に時刻の値を格納する変数を用意する．特定の 1 つのスレッドで時刻の計算を行いその時刻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，このように GPU 内で特定のスレッドでシミュレーションの時刻管理を行うほうが性能は向上する．真空中の電界計算のカーネルは次のようになる．

```
__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 + tx;
  unsigned int y = blockIdx.y*blockDim.y + ty;
  __shared__ float SH_HX[BLOCK+1][BLOCK+1];
  __shared__ float SH_HY[BLOCK+1][BLOCK+1];
  if(ty==(BLOCK-1))SH_HX[ty+1][tx]=HX[(y+1)*w+x];
  if(tx==(BLOCK-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();
   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]);
}
```

5. 計算高速化の検討

本手法による２次元 FDTD 法の差分計算高速化を検討する．数値計算モデルを図 5 に示す．解析領域の中心に，次式で表わされる電磁波の線源（Line Source）を配置する．

$$E_z^n(i_c, j_c) = \sin(\omega n \Delta t) \,, \qquad (5)$$

ここで，$\omega = 2\pi/(40\Delta t)$，$\Delta t = 0.5\Delta h/C_0$，$\Delta h = \Delta x = \Delta y = \lambda/20$ とする．(i_c, j_c)は解析領域の中心座標を示す．解析領域の終端の境界条件をディレクレ条件（E_z=0）とした．

GPU を用いた本手法による計算時間と従来の FDTD 法による CPU のみの計算時間を比較し，計算高速化を検討する．GPU による計算及び CPU のみの計算において，CPU に Intel Core2Duo E8400 (3.0GHz)，メインメモリに 2GB（DDR3-1333）を搭載している同じ PC を用いた．また，GPU として NVIDIA Geforce GTX 280 を使用した．オペレーティングシステムとして，Linux（Fedora 8）を使用した．

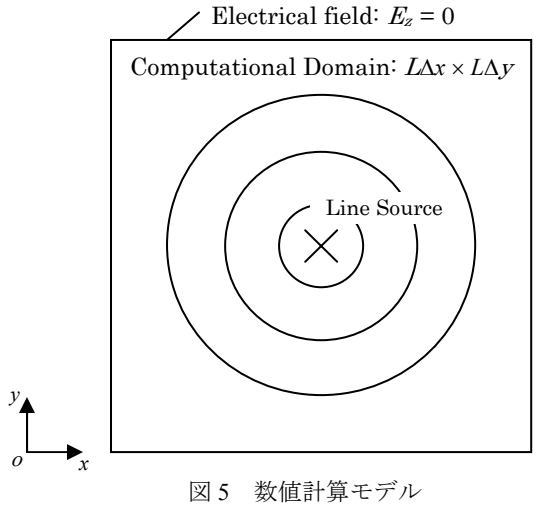

図 5　数値計算モデル

表 2　本手法による GPU 計算時間（T_{GPU}）と CPU のみの計算時間（T_{CPU}）の比較（1,000step）

Comutational domain $L\Delta x \times L\Delta y$	T_{CPU} (sec)	T_{GPU} (sec)	Speedup factor (T_{CPU} / T_{GPU})
1,024×1,024	7.792	0.401	19.43
2,048×2,048	31.308	1.597	19.60
3,072×3,072	70.302	3.687	19.07
4,096×4,096	124.560	6.775	18.39
5,120×5,120	194.532	11.244	17.30
6,144×6,144	280.036	17.594	15.92
7,168×7,168	381.140	26.207	14.54
8,192×8,192	498.076	37.368	13.33

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，1 つのブロックのスレッド配列を 16×16 とし，NVIDIA 社の提供する CUDA 2.0 を用いてコンパイルした．従来の FDTD 法による CPU のみの計算は，C 言語でプログラムを記述し，C 言語のコンパイラとして，Linux 用の Intel C コンパイラ（バージョン 10.1）を用いた．なお、コンパイラオプションとして“-msse –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 GPU の計算時間（T_{GPU}）と従来の FDTD 法による CPU のみの計算時間（T_{CPU}）を表 2 に示す． CPU のみの計算時間は，FDTD 法の解析領域の大きさ，すなわち，FDTD 法の演算量に比例している．解析領域の大きさが 2,048×2,048 のとき，CPU の計算に対して GPU の計算が 19.60 倍高速化しており，ピーク性能を示している．解析領域の大きさが 2,048×2,048 より大きくなると，GPU の計算速度は低下するが，解析領域の大きさが 8,192×8,192 の場合でも CPU のみの計算に比べ，本手法による GPU の計算は 13.33 倍高速化している． 共有メモリを用いずに FDTD 法で計算を行った場合でも同様の現象が起こることから，各ブロックの計算時間のずれが原因となっている可能性がある．

次に，本手法による GPU 計算の性能を評価する．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, ロードデータとストアデータの総数は 9 個となる．解析領域の大きさ $L\Delta x \times L\Delta y$ において，FDTD 法で計算する時間ステップ数を N_{itr} とする場合，本手法による GPU 計算の実測性能（FLOPS 値）は，(12 operation × $L \times L \times N_{itr}$) / T_{GPU} により求まる．よって，解析領域の大きさに対する本手法による GPU 計算の実測性能（FLOPS 値）は表 3 のようになる．

表 1 より，NVIDIA Geforce GTX 280 の理論ピーク性能（Theoretical peek performance）は，SP で積和算の 2 オペ

表 3　本手法による GPU を用いた２次元 FDTD 法差分計算の性能

L ($L\Delta x \times L\Delta y$)	1,024	2,048	3,072	4,096	5,120	6,144	7,168	8,192
GFLOPS	31.40	31.52	30.71	29.71	27.98	25.75	23.53	21.55
vs. Actual Peek Performance	0.84	0.84	0.82	0.79	0.74	0.68	0.63	0.57
vs. Theoretical Peek Performance	0.66	0.67	0.65	0.63	0.59	0.55	0.50	0.46

レーションを 1 クロックで行うとした場合，2 operation/SP × 240 SP ×1.296GHz = 622.08 GFLOPS となる．また，NVIDIA Geforce GTX 280 のメモリバンド幅の理論性能は，141.7 GB/sec である．1 ワード（4 Bytes）当たりのメモリバンド幅の理論値は，141.7 GB/sec ÷ 4 Bytes/Word = 35.43 GWord/sec となる．よって，メモリバンド幅の理論値から導出される GPU の理論性能は，35.43 GWord/sec × 12 operation / 9 Word = 47.24 GFlops となる．このメモリバンド幅の理論値より導出した性能は，GPU の理論ピーク性能である 622.08 GFLOPS よりも小さい．最終的に，GPU として NVIDIA Geforce GTX 280 を用いて２次元 FDTD 法の計算を行った場合，メモリバンド幅がボトルネックとなり，GPU を用いた本計算の理論ピーク性能は，47.24 GFlops となる．また，NVIDIA CUDA SDK 2.0 のテストプログラムを用いて実際のメモリバンド幅を計測したところ 112.80GB/sec であった．1 ワード（4 Bytes）当たりのメモリバンド幅の実測値は， 112.80 GB/sec ÷ 4 Bytes/Word = 28.20 GWord/sec となる．よって，メモリバンド幅の実測値から導出される GPU の実測ピーク性能（Actual peek performance）は，28.20 GWord/sec × 12 operation / 9 Word = 37.60 GFlops となる．以上のことより，GPU の理論ピーク性能及び実測ピーク性能に対する本手法における GPU 計算の実測性能の割合を表 3 に示す。

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，その絶対誤差は 4.0×10^{-7}以下であった．本手法による GPU 計算は，２次元 FDTD 法の解析として十分な精度を得ることができた．

むすび

共有メモリを効率よく活用する FDTD 法アルゴリズムを CUDA 互換 GPU に実装した．従来の FDTD 法による CPU のみの計算に比べ、最大 20 倍の計算高速化を実現した．本手法による計算結果は，十分な精度を得ることができた．今後は本手法に吸収境界条件を適用する．さらに，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 boundary value problems involving Maxwell´s equations in isotropic media,” IEEE Trans. Antennas & Propagat., vol. AP-14, no. 3, pp.302-307, 1966.

[2] A. Taflove, “Computational Electrodynamics: The Finite Difference Time Domain Method,” Artech House, Inc., 1995.

[3] K. S. Kunz and R. J. Luebbers, “The Finite Difference Time Domain Method for Electro-magnetics,” CRC Press, Inc., 1993.

[4] D. P. Rodohan, S. R. Saunders, and R. J. Glover, “A Distributed Implementation of the Finite Difference Time Domain (FDTD) Method,” Int. J. Numerical Modeling: Electronic Networks, Devices and Fields, vol. 8, no.3, pp.283-292, 1995.

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

[6] NVIDIA corporation, GPU Gems 3, Addison-Wisley, 2008.

[7] NVIDIA, NVIDIA CUDA Computational unified device architecture programming guide version 2.0, 2008.

[8] G. S. Baron, C. D. Sarris, and E. Fiume, “Fast and Accurate Time-Domain Simulations with Commodity Graphics Hardware,” Proceedings of Antennas and Propagation Society International Symposium, July 2005.

[9] M. J. Inman and A. Z. Elsherbeni, “Programming Video Cards for Computational Electromagnetics Application,” IEEE Antennas and Propagation Magazine, vol. 47, no. 6, pp.71-78, Dec. 2005.

[10] N. Takada, N. Masuda, T. Tanaka, Y. Abe, T. Ito, “A GPU implementation of 2-D finite-difference time-domain code using high level shader language,” The Appl. Comput. Electromagn. Soc J., vol. 23, no. 4, pp.309-316, 2008.

[11] 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, 2008.

[12] R. Fernando, M. J. Kilgard, The Cg Tutorial, Addison Wisley, 2003.

[13] I. Buck, “Brook Spec v0.2,” Stanford University, 2003.

[14] N. Takada, T. Shimobaba, N. Masuda, T. Ito, “High-Speed FDTD simulation Algorithm for GPU with Cpmpute Unified Device Architecture,” The 2009 IEEE International Symposium on Antennas and Propagation and USNC/URSI National Radio Science Meeting, June, 2009 (invited).