超高速 GPU 並列計算を利用した医用超音波イメージングのための数値解析法の確立（継続）

代表研究者　大久保　寛　首都大学東京システムデザイン学部准教授

1 はじめに

本研究は，GPU（Graphics Processing Unit）による超高速並列計算と超音波・波動伝搬シミュレーション技術を開発することが目的である．

現在，医工学分野では生体への影響の少ない超音波による計測が多く行われている．特に，生体内超音波の非線形伝搬現象については，高調波を利用した超音波エコー法（Tissue Harmonic Imaging : THI）として，医用領域での工学的な応用を視野に可視化技術(アコースティック・イメージング）の開発が始まっている．しかし，超音波の伝搬シミュレーションの手法は確立しているとは言えない．

そこで，本研究では，そのためのシミュレーション技術として GPU コンピューティングによる超高速計算を適用し，CUDA によるシミュレーション実装方法及び可視化方法の開発を行い，マルチ GPU 並列計算とともに検討する．

2　GPU による音場数値解析

2-1 概要

近年の計算機環境の向上は，多くの分野での数値シミュレーションを可能としている．現在一般的に用いられているシミュレーション手法の 1 つに，時間領域での数値シミュレーションが挙げられる．この時間領域での解析手法はいくつかあるが，その中でも FDTD(finite difference time domain)法は音場シミュレーションの際に最も広く用いられている手法である．FDTD 法においては，時間・空間の中心差分が支配方程式の導関数の近似に用いられる．中心差分近似は精度により様々な差分式とすることができる．

以降ではリアルタイムの高速可視化を検討するため,比較的定式化が簡単で計算負荷の少ない FDTD 法を用いる．本手法は時間領域で解く手法であり，各グリッド上の値の更新は順序性がなく，場の値はそれぞれ独立して求めることができるため並列化に適した手法と言える．GPU の演算性能とメモリバンド幅を考慮すると，FDTD 計算のボトルネックはメモリ転送であり，基本的な解析性能はメモリバンド幅に律則していると考えられる．

2-2 数値解析手法

FDTD(finite difference time domain)法は，数値シミュレーションにおいて現在最も広く用いられている手法である．これは，変数に音圧と粒子速度を用いて音場の支配方程式である運動方程式と連続の式を，時間領域と空間領域において差分化する手法である．線形，無損失の場合，音場の支配方程式(運動方程式と連続の式)は

$$\frac{\partial p}{\partial t} = -\kappa\left(\frac{\partial v_x}{\partial x} + \frac{\partial v_y}{\partial y} + \frac{\partial v_z}{\partial z}\right) \tag{1}$$

$$\frac{\partial v_x}{\partial t} = -\frac{1}{\rho}\frac{\partial p}{\partial x}, \frac{\partial v_y}{\partial t} = -\frac{1}{\rho}\frac{\partial p}{\partial y}, \frac{\partial v_z}{\partial t} = -\frac{1}{\rho}\frac{\partial p}{\partial z} \tag{2}$$

で与えられる．ただし，p は音圧，v_x, v_y, v_z はそれぞれ x, y, z 方向の粒子速度，κ は体積弾性率，ρ は密度である．

このとき FDTD 法において使用されるグリッドは，図 1 の Staggered grid である．このグリッドでは，音圧と粒子速度は時間的・空間的に 1/2 グリッドずらして配置され，支配方程式を中心差分近似により直接離

散化する.

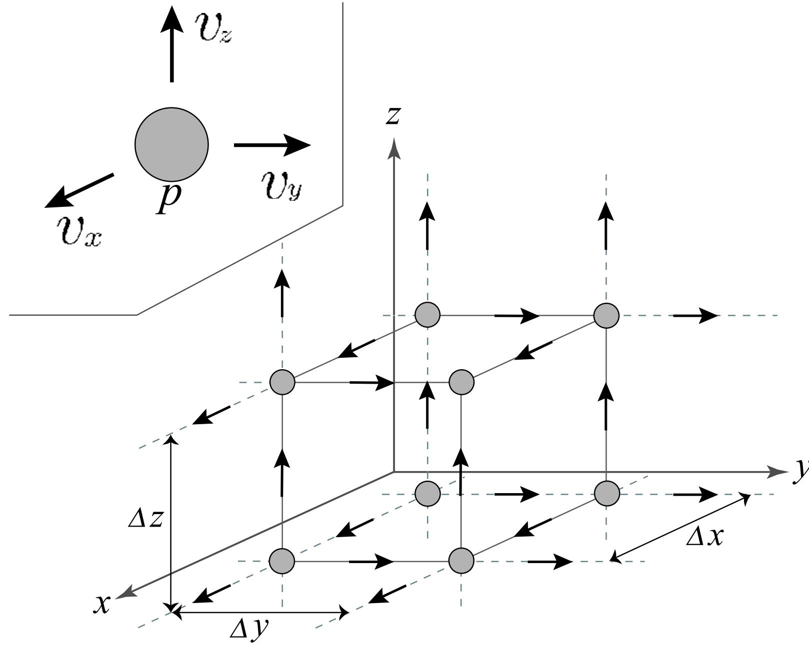

図 1　FDTD 法のグリッド

まず，時間方向の離散化を行う．時間方向にだけ微分を 2 次精度中心差分で離散化した式は，

$$\frac{p^{n+1}(i,j,z)-p^{n}(i,j,z)}{\Delta t}=-\kappa\left(\frac{\partial v_x}{\partial x}\Big|_{i,j,z}^{n+1/2}+\frac{\partial v_y}{\partial y}\Big|_{i,j,z}^{n+1/2}+\frac{\partial v_z}{\partial z}\Big|_{i,j,z}^{n+1/2}\right) \tag{3}$$

$$\frac{v_x^{n+1/2}(i+1/2,j,z)-v_x^{n-1/2}(i+1/2,j,z)}{\Delta t}=-\frac{1}{\rho}\frac{\partial p}{\partial x}\Big|_{i+1/2,j,z}^{n} \tag{4}$$

$$\frac{v_y^{n+1/2}(i,j+1/2,z)-v_y^{n-1/2}(i,j+1/2,z)}{\Delta t}=-\frac{1}{\rho}\frac{\partial p}{\partial y}\Big|_{i,j+1/2,z}^{n} \tag{5}$$

$$\frac{v_z^{n+1/2}(i,j,z+1/2)-v_y^{n-1/2}(i,j,z+1/2)}{\Delta t}=-\frac{1}{\rho}\frac{\partial p}{\partial z}\Big|_{i,j,z+1/2}^{n} \tag{6}$$

となる．(6)式ついては$t=(n+1/2)\Delta t$ において差分化し，(7)式と(8)式については$t=n\Delta t$ において，それぞれ差分化している．さらに，時間更新に着目して(6)～(8)式を変形すると，

$$p^{n+1}(i,j,z)=p^{n}(i,j,z)-\kappa\Delta t\left(\frac{\partial v_x}{\partial x}\Big|_{i,j,z}^{n+1/2}+\frac{\partial v_y}{\partial y}\Big|_{i,j,z}^{n+1/2}+\frac{\partial v_z}{\partial z}\Big|_{i,j,z}^{n+1/2}\right) \tag{7}$$

$$v_x^{n+1/2}(i+1/2,j,z)=v_x^{n-1/2}(i+1/2,j,z)-\frac{\Delta t}{\rho}\frac{\partial p}{\partial x}\Big|_{i+1/2,j,z}^{n} \tag{8}$$

$$v_y^{n+1/2}(i,j+1/2,z)=v_y^{n-1/2}(i,j+1/2,z)-\frac{\Delta t}{\rho}\frac{\partial p}{\partial y}\Big|_{i,j+1/2,z}^{n} \tag{9}$$

$$v_z^{n+1/2}(i,j,z+1/2)=v_y^{n-1/2}(i,j,z+1/2)-\frac{\Delta t}{\rho}\frac{\partial p}{\partial z}\Big|_{i,j,z+1/2}^{n} \tag{10}$$

となる．次に，空間方向に差分化を行う．式(7)～(10)より，

$$p^{n+1}(i,j,z)=p^{n}(i,j,z)-\kappa\Delta t\left\{\begin{aligned}&\frac{v_x^{n+1/2}(i+1/2,j,z)-v_y^{n+1/2}(i-1/2,j,z)}{\Delta x}\\&+\frac{v_y^{n+1/2}(i,j+1/2,z)-v_y^{n+1/2}(i,j-1/2,z)}{\Delta y}\\&+\frac{v_z^{n+1/2}(i,j,z+1/2)-v_z^{n+1/2}(i,j,z-1/2)}{\Delta z}\end{aligned}\right\} \quad (11)$$

$$v_x^{n+1/2}(i+\frac{1}{2},j,z)=v_x^{n-1/2}(i+\frac{1}{2},j,z)-\frac{\Delta t}{\rho}\frac{p^{n}(i+1,j,z)-p^{n}(i,j,z)}{\Delta x} \quad (12)$$

$$v_y^{n+1/2}(i,j+\frac{1}{2},z)=v_y^{n-1/2}(i,j+\frac{1}{2},z)-\frac{\Delta t}{\rho}\frac{p^{n}(i,j+1,z)-p^{n}(i,j,z)}{\Delta y} \quad (13)$$

$$v_z^{n+1/2}(i,j,z+\frac{1}{2})=v_z^{n-1/2}(i,j,z+\frac{1}{2})-\frac{\Delta t}{\rho}\frac{p^{n}(i,j,z+1)-p^{n}(i,j,z)}{\Delta z} \quad (14)$$

となる．ここで，$\Delta x,\Delta y,\Delta z$ は x,y,z 方向の空間刻み幅，Δt は時間刻み幅，i,j,k は x,y,z 座標に対応する空間離散地点，n は離散時刻を表している．

3 GPU による計算

3-1 GPU 化

計算の高速化という点では，従来スーパーコンピュータやクラスタなどの大型の計算機が利用されているが，最近では GPU(Graphics Processing Unit)を用いて汎用的な数値計算を行おうとする GPGPU(General Purpose computation on GPUs)が様々な分野で注目され始めている．現在では GPU の演算性能は数年前より飛躍的に向上しており,最新の GPU は少し前のスーパーコンピュータ並の性能を秘めているとも考えられる．

ここでは GPU による音場数値シミュレーションの実現へ向けて，マルチ GPU を搭載した計算機を用いた高速並列計算によるシミュレーションを評価する．

3-2 CUDA による GPU プログラミング

GPU は本来画像処理に特化したアーキテクチャとして生まれたものであるが，最近では並列計算を行う場合などでは CPU を使った計算より圧倒的な高速化が可能であるといわれる．

GPGPU の初期の段階では，コンピュータグラフィックス向けの専用言語を用いてプログラミングする必要があつたため,変数の型に GPU 特有の型しか使えないなど汎用的なプログラムの記述はかなり困難であつた．しかしながら，GPGPU 向けのプログラミング言語として CUDA(Compute Unified Device Architecture)などの C 言語に近い言語が開発されたことにより，C 言語の知識だけで比較的手軽にプログラミング可能となった[1,2]．

CUDA (NVIDIA GPU) のハードウェアモデルを図 1 に示す．同図は本研究で用いた GeForce GTX 580 (GeForce GTX シリーズに属している) であり，512 基のストリーミングプロセッサ(SP，CUDA Core とも呼ばれる)で構成され，高速アクセス可能な共有メモリ(SM)を持っている．

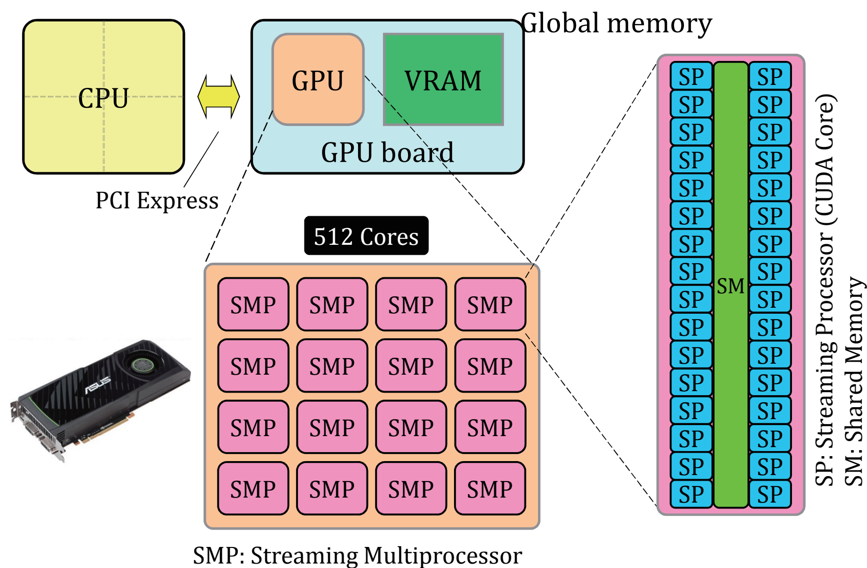

図 2　ハードウェアモデル

GPU 上のグローバルメモリ(GM)から MP へのメモリ転送はかなり高速ではあるが,MP の並列計算に掛かる時間と比べるとデータ転送時間が総計算時間のかなりの部分を占めることになる．したがって，計算アルゴリズムによるが頻繁に GM にアクセスする必要がある場合には，参照するデータを一旦 SM へ転送してから演算を行う方が高速に処理できる．CUDA で GPU プログラミングを行う場合，高速化の重要なポイントはブロック内のスレッドモデルをどのように構成するかと，レジスタや SM を活用することである．すなわち，複数回 GM にアクセスする必要がある場合には，参照するデータを一旦 SM やレジスタへ転送し，メモリアクセスを極力低減させる必要がある．

CUDA における最小実行単位はスレッド（Thread）と呼ばれる．さらに 32 スレッドをワープと呼び，1 ワープの単位で MP の中の多数の SP により並列実行される．また，CUDA では図 18 に示すように，スレッドのまとまりをブロック，ブロックのまとまりをグリッドと呼び管理している．グリッドは PC（ホスト側）から実行を指令する単位で，グリッド内の全スレッドは同じプログラム(カーネルと呼ぶ)を実行する．

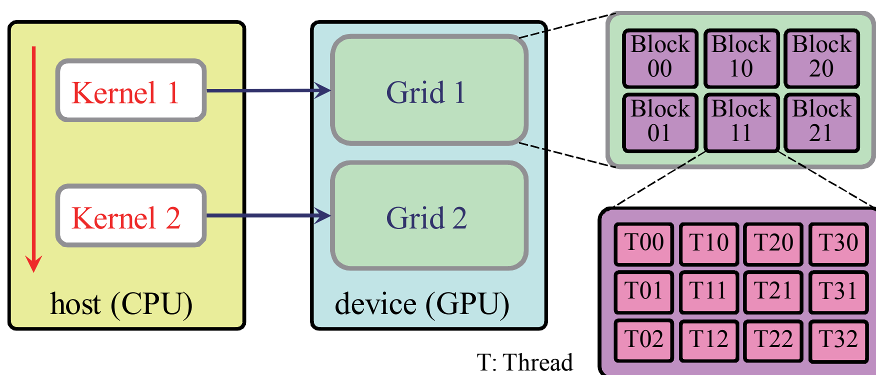

図 3　スレッドモデル

3-3 OpenGL による GPGPU 可視化

続いて計算結果の可視化について述べる．従来の CPU 計算における可視化の方法としては，一般には，数値解析結果をディスプレイ上にリアルタイムで可視化しようとする場合，解析結果を色情報へ変換したのちにビデオカード上の描画用の VRAM 領域に転送する（図 4 参照）．このプロセスは，CPU を用いて音場数値計算をした場合，解析結果が格納されている RAM から VRAM への転送が必要であるため，表示させる結果によっ

てはPCIインターフェイスの転送が問題となる．

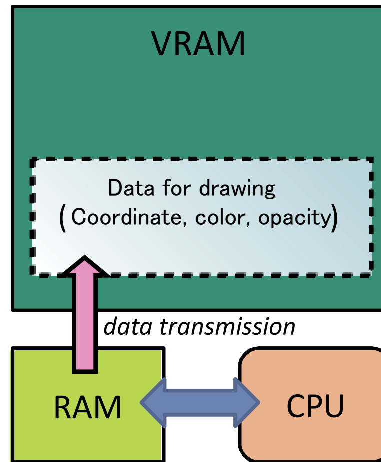

図4　CPU使用時の可視化手順

他方，CUDAでGPU計算を実装する場合，OpenGLの関数をCUDAから直接呼び出すことができ，また，計算領域がVRAM内に確保されているため，PCIインターフェイスの転送をバイパスしてビデオカード上の描画用のVRAM領域へ表示情報を書き込むことができる（図5参照）．これはリアルタイム可視化を目指す上で非常に大きな利点である．CUDAによるGPU計算によって，計算速度自体が高速化させられるが，CUDAとOpenGLの連携によってさらに描画におけるアドバンテージも受けることができるのである．

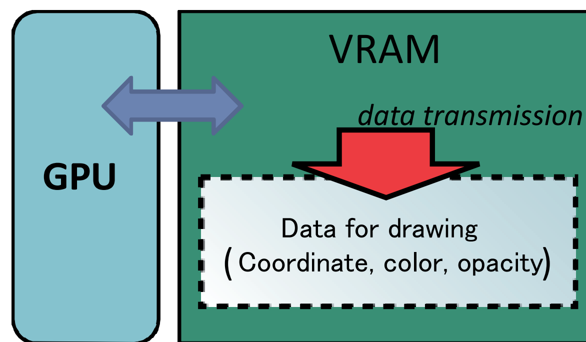

図5　GPU使用時の可視化手順

3-4 GPGPUと高速可視化

従来，可視化を含めた音場解析を行う場合，計算環境への負荷を考え，対称性を仮定した2次元モデルや1次元モデルを用いた解析が行われてきていた．特に，パーソナルコンピュータ（パソコン）で解析する場合には，3次元空間をモデル化するためのメモリ容量の問題や，メモリが確保できたとしてもその計算を行うための計算時間を考慮すると，現実問題として3次元解析は困難な場合が多かった．

さらに，シミュレーションをしながら，同時に解析結果を可視化する，いわゆるリアルタイム可視化を行う場合は，十分な描画スピードを維持するためには，3次元解析はパソコンレベルではほぼ不可能であった．

しかし，近年の計算機技術の発展 ――特に，GPUによる高速並列計算は，描画スピードの問題を一気に解決し，3次元の音場シミュレーションとそのリアルタイム可視化を現実のものとしようとしている．すなわち，GPUを用いたパーソナルスーパーコンピュータが同時可視化シミュレーションを大きく後押しすることになる．今後，このGPUパーソナルスパコンは，3次元音場シミュレーションのリアルタイム可視化において，中心的な役割を担っていくことと考えている．

3-5 提案可視化方法（PMCC）

従来，3次元音場の計算結果を可視化する場合，初歩的な手法として3次元空間の1つの断面を表示する方法やこの断面の見る角度を変え，z軸方向に音圧値などの強度を割り当てる（サーフェスプロット）方法

などが挙げられる．しかし，この方法で解析空間の 3 次元的な広がりを描画することは難しい．
一方，近年では，不透過度（不透明度）を利用するボリュームレンダリング(VR) なる可視化法が提案されている．3 次元音場の可視化にこのボリュームレンダリングを利用する場合，3 次元空間の全領域における各ボクセルに対して，音圧などの強度に合わせた不透過度を設定し，それをある視点からのパス上で積分することで 3 次元の空間を表す．特徴として，3 次元音場の解析空間中の全ての情報を利用しており，3 次元空間が一気に表示できる点でとても優れている．
しかし，この方法は 3 次元音場の各ボクセルのすべての情報を処理するため，描画のために多くの計算負荷がかかるという欠点があり，結果として他の手法に比べて描画スピードが低下する．さらに，マルチ GPU を用いる場合，セカンダリ GPU の VRAM から描画を行うプライマリ GPU へ描画データを転送する場合，3 次元解析空間の割り当て分を全て転送する必要があるため，極端に描画速度が低下する可能性が指摘される．
本研究では従来の可視化手法の問題を解決するための方法をとして PMCC (Permeable Multi Cross-section Contours) [3]を検討する．
図 6 に PMCC の可視化例を示す．図は時間経過となっている．この PMCC は一言でいえば，表示する複数の断面に対して，表示する強度に合わせて不透過度を設定する方法である．
描画の手順をまとめると，以下のようになる．

- 3 次元空間のある断面に対して，音圧値などをカラー表示する．
- さらに，音圧値の強度に合わせて不透過度（α 値）を設定する．(音圧値が大きい場合に不透過度は 1 に近づく)
- 不透過度の与え方は計算対象によって変動させることができるので，表示させる強度の下限や強度と α 値の関係式などを自由に設定できる．
- 同様の手順で表示させたい断面を複数描画する．

特徴としては，

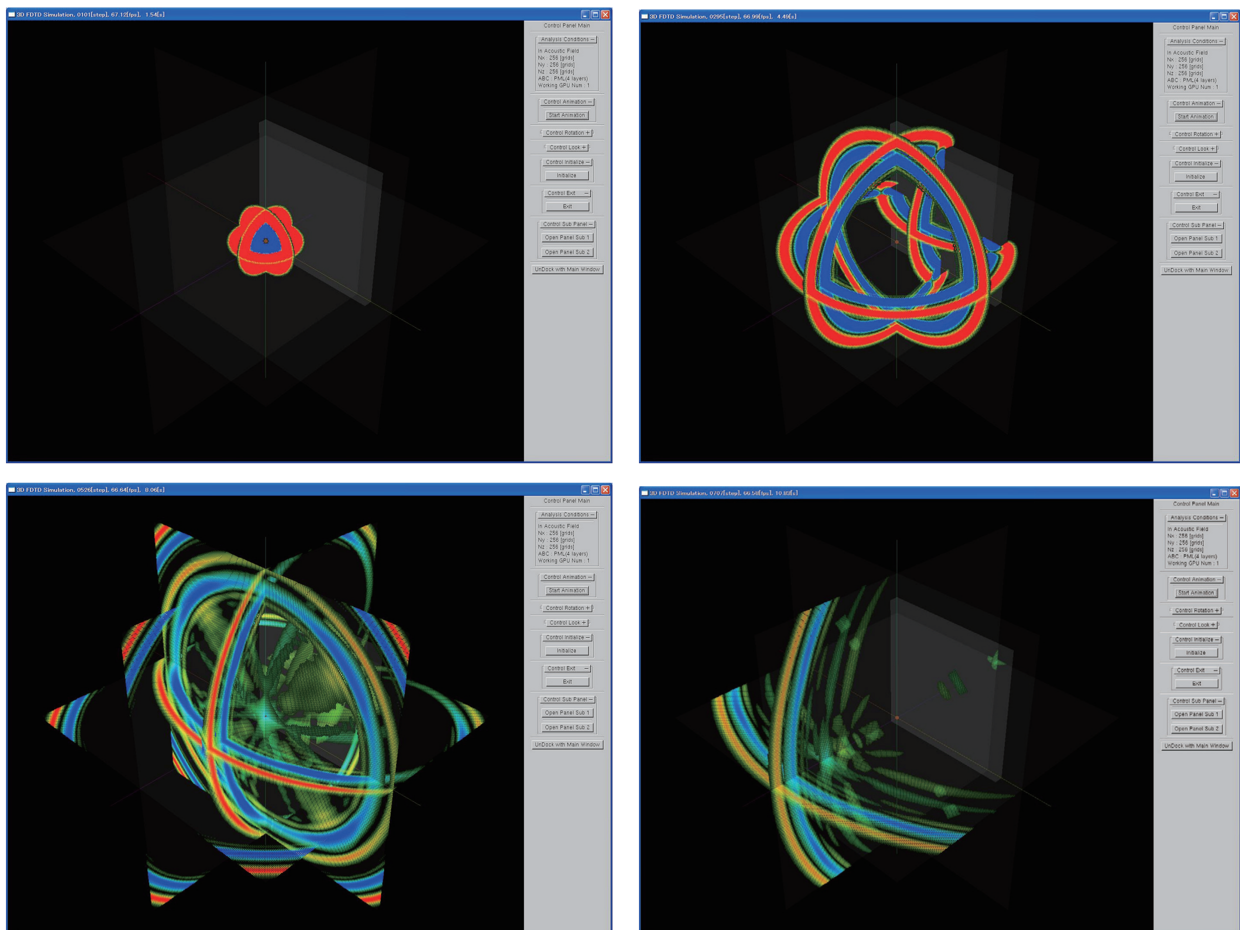

図 5　PMCC (Permeable Multi Cross-section Contours) による可視化

・複数の断面を表示しても，音圧値が小さい点は不透過度が小さくなるため，オクルージョンが発生しにくく，ある断面の裏に隠れてしまっていた情報も見ることができる．
・したがって，平行な複数の断面も同時に表示可能である．
・断面の回転及び移動も可能である．しかも，GPU 実装では高速に動作し，待ち時間無く回転・移動することを確認している．
・断面のみの不透過度を利用しているため，ボリュームレンダリングに比べて，描画のための計算負荷が極端に少ない．
・断面表示をもとにしているため，複雑な波動伝搬現象も把握しやすい．
ということが挙げられる．

3-5 マルチ GPU を使用した GPGPU 可視化

シングル GPU における OpenGL を用いた可視化では単に GPU 上の計算領域から描画領域に描画用のデータを書き込むことでイメージングが可能であった．しかし，PCI バスで接続された 2 つ以上の GPU を用いて計算を行う場合（いわゆる単一ノード上のマルチ GPU 計算），計算したデータを描画するには，描画を担当する GPU に PCI バスを通して，描画用データを転送する必要がある．これは，マルチ GPU 計算での GPU 間の境界データの転送に加えて，描画のために必要な GPU 間のデータ転送作業となる．

このマルチ GPU を用いた可視化の流れを簡単に説明すると，(1)複数の GPU で領域を分割・担当し計算，(2)GPU 間の境界領域のデータを交換，(3)描画用のデータを PCI バスを通して RAM に転送（ただし，描画担当の GPU は除く），(4)描画用のデータを描画担当の GPU の描画データ領域に転送（ただし，描画担当の GPU は PCI バスを介さずに直接 GPU 内で転送），(5)再び複数の GPU で領域を分割・担当し計算，を繰り返す．このようにすることで，プライマリ GPU 以外からの PCI 転送のコストは必要とはなるが，マルチ GPU を用いたリアルタイム可視化が可能となる．

また，境界領域のデータの GPU 間転送については，非同期通信を利用し境界領域以外の計算中に転送を行うことで通信隠ぺいを行うことができる．一方，CUDA4.0 より GPU Direct 2.0 による GPU 間メモリ転送が新たにリリースされた．従来の GPU 間転送では，メインメモリ（RAM）を介する必要があったが，GPU Direct 2.0 では直接 GPU 間を転送するため，GPU 間のデータ転送時間を短くすることができる

4 数値計算結果

3-1 計算速度の評価

本節では，計算速度の比較評価を行う．計算手法としては FDTD 法を用いている．使用した計算環境は OS：Windows 7 Pro x64 edition，CPU ：Intel Core i7 930 2.80GHz，メモリ ：DDR3 6GB，コンパイラ ：マイクロソフト C/C++ コンパイラ Ver.15.00 for x64，OpenMP ：OpenMP 2.0 である．以下では単精度型を用いて計算を行っている．本計算では GPU として GeForce GTX 580 を使用している．表 1 に GPU の主要スペックを示す．

表 1　GeForce GTX 580 のスペック一覧

主要スペック	GTX 580
CUDA Core 数	512
プロセッサクロック	1544 MHz
メモリクロック	2004 MHz
メモリインターフェイス	384 bit
メモリサイズ	1536 MByte
メモリタイプ	GDDR5

また,性能評価の指標として FLOPS（Floating point number Operations Per Second）と FUPS（Fields Update Per Second）を用いる．FLOPS は 1 秒間に浮動小数点演算が何回行われたか、FUPS は 1 秒間に計算領域内の何点の場の値が更新されたかを表す．音場解析の場合，計算速度としては FLOPS よりも FUPS を用いたほうが適正な指標と言える．

表 2 に 3 次元計算領域のグリッド数を 256×256×256，計算回数を 1024 と固定した時の FDTD 解析について，実行した場合の計算時間をそれぞれ示す．ただし計算には単精度型を用いている．また，CPU の結果も

あわせて示している.計算領域は立方体の領域として,吸収境界条件は除いた領域のみの評価を行っている.ここでは，1 thread i7(CPU)，8 thread i7(CPU)，GTX 580 (1GPU)の結果を示し，さらに，GTX 580 を2個利用したマルチ GPU 計算の結果も示している.

同表より，以下のことがわかる．FDTD 法ではシングル GPU（GTX 580）で CPU（i7 8 スレッド並列）の約 14.9 倍の高速化が実現出来ている．一方，2 つの GPU を利用した場合では約 29.0 倍の高速化が実現出来ている．3 次元解析において，4.752 GFUPS はパソコンレベルでは非常に高速といえ，スーパーコンピュータに迫る環境がマルチ GPU 計算によって実現できることがわかる.

表 2　**計算速度の比較**

計算環境	GFLOPS	GFUPS
Core i7 930(8 thread)	2.955	0.164
GTX 580 ×1	44.076	2.448
GTX 580 ×2	85.542	4.752

3-2 描画スピードの比較評価

次に CUDA と OpenGL を組み合わせることで可視化について描画スピードの比較評価を行う.

領域サイズを 256×256×256，可視化手法を PMCC とし，吸収境界として 4 層 PML を適用した場合のフレームレートを表 3 に示す.

フレームレートは 1024 ステップまでの描画にかかった時間より計算している．同表より，CPU で計算して可視化した場合と比較して，GeForce GTX 580 を 1 個用いた場合は約 34 倍，2 個用いた場合は約 55 倍の高速化が達成できたことが分かる.

表 2　**描画速度の比較**

計算環境	フレームレート[fps]
Core i7 930(8 thread)	2.3
GTX 580 ×1	77.8
GTX 580 ×2	130

次に計算領域（グリッド数）に対する描画速度の結果を図 6 に示す．図中は可視化手法として PMCC[3]を用いた場合の 3 次元シミュレーションの描画スピードを示している．同図では横軸がグリッド(セル)数，縦軸が描画速度(fps)としている．また吸収境界として，4 層 PML を適用した場合の結果を表示している.

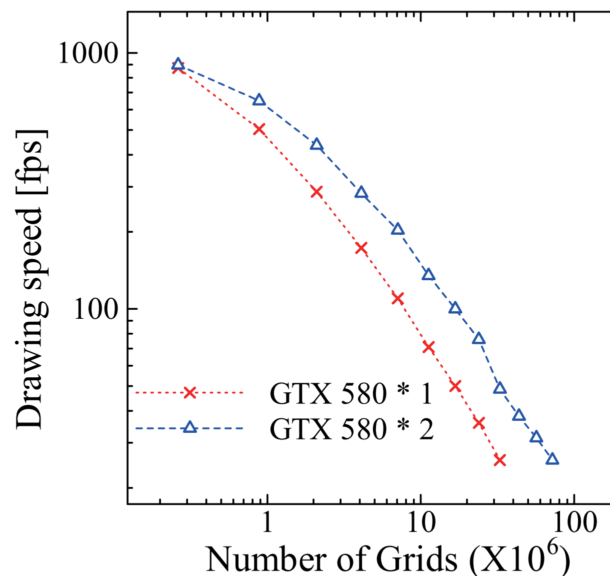

図 5　グリッド数に対する描画速度（FDTD，3 次元音場解析）

同図にはシングル GPU(GTX 580×1)，マルチ GPU(GTX 580×2)の比較結果を表示している．ただし，グラフ

ィック出力はプライマリ GPU からのみ行われている．図より解析領域が大きくなると，マルチ GPU の効果がより顕著になることがわかる．GPU をシングルからマルチにすることで，大きな解析領域において高速化が可能となっており，PMCC による可視化を用いたインタラクティブ音場シミュレーションを行う場合 GPU のマルチ化が有効な手法であることが分かり，従来の方法では考えられないような効果が得られている．

5 おわりに

本研究では,医用超音波イメージングのための数値解析法の確立を目指し,GPU(Graphics Processing Unit)による超高速並列計算と超音波・波動伝搬シミュレーション技術を開発してきた．

Fermi シリーズ（GTX 580）のマルチ GPU を用いて 3 次元音場数値解析のリアルタイムの可視化（計算と同時に可視化する音場シミュレーション）について検討した．3 次元音場の可視化法として，PMCC を示し，その方法を用いた GPGPU 可視化の評価を行った．PMCC は従来の 3 次元空間中の複数断面を表示する（Multi Cross-section Contours）方法に不透過度を組み合わせたシンプルかつ高速な可視化方法である．

本研究の結果より，3 次元音場数値解析の高速可視化において，マルチ GPU 計算と時間領域手法，そして OpenGL と PMCC を組み合わせて利用することは，非常に有用な方法であることが明らかとなった．

【参考文献】

[1]http://www.nvidia.co.jp/object/cuda_home_jp.html

[2]青木尊之, 情報処理学会誌, Vol.50, No.2, pp.107-115, 2009.

[3]河田直樹, 大久保寛, 田川憲男, 土屋隆生, 石塚崇, CUDA と OpenGL を用いた三次元音響数値解析の GPGPUリアルタイム可視化—PMCC(Permeable Multi Cross-section Contours)の提案と評価—, 電子情報通信学会論文誌 A, vol.J94-A, no.11, pp.854-861, 2011.

〈発　表　資　料〉

題　　名	掲載誌・学会名等	発表年月
CUDA と OpenGL を用いた三次元音響数値解析の GPGPU リアルタイム可視化 —PMCC（Permeable Multi Cross-section Contours）の提案と評価—	電子情報通信学会論文誌 A, vol.J94-A, no.11, pp.854-861	2011/11
マルチ GPU コンピューティングによる音響シミュレーションの高速可視化	音響学会発表会予稿集	2011/09
直交格子を用いた音響数値解析の媒質間境界の取り扱いに関する再考察	音響学会発表会予稿集	2011/09
GPU コンピューティングによるインタラクティブ音響シミュレーション	音響学会発表会予稿集	2012/03
FDTD 系音響数値解析のための媒質間境界及びインピーダンス境界の設定に関する考察	音響学会発表会予稿集	2012/03