

論文**並列計算による音響 FDTD 法的高速化**福庭隆弘^a, 北村達也^b^a 甲南大学大学院 自然科学研究科 情報システム工学専攻

神戸市東灘区岡本 8-9-1, 658-8501

^b 甲南大学 知能情報学部 知能情報学科

神戸市東灘区岡本 8-9-1, 658-8501

(受理日 2010 年 11 月 1 日)

概要

時間領域差分法 (Finite Difference Time Domain, FDTD) 法は近年音響分野を含む幅広い分野で用いられるようになった。本研究では、声道の音響特性を求めるために FDTD 法を用いて 3 次元声道音響シミュレータを開発した。FDTD 法は計算量の多い手法であるため、並列演算を用いることにより計算速度を向上させるメリットがある。そこで並列計算化手法である GPGPU やマルチコア CPU を利用してシミュレータの高速化を行った。逐次計算との計算速度や計算結果の比較を行い、並列計算の有用性を示した。

キーワード: FDTD 法, 並列演算, GPGPU, GPU, CUDA, OpenMP, 音響解析

1 はじめに

近年、差分法の一つである時間領域差分 (Finite Difference Time Domain, FDTD) 法は様々な分野のシミュレーションに利用されるようになってきている。音響分野でも注目を集めており、建築や音声の分野での論文や学会発表が数多く見られるようになった。FDTD 法は比較的計算量が多い数値計算であり、かつては非常に計算時間のかかる手法であった。しかし、昨今の計算機の高速化に伴って計算に要する時間も年々短縮され、現在市販されている一般的な計算機でも十分計算できるようになった。

この背景には計算機のハードウェア面における性能向上とソフトウェア面における並列計算開発環境の普及の 2 つが大きく寄与している。ハードウェアの面では、メモリの大容量化、CPU コアの複数化、General Purpose computing on Graphics Processing Units (GPGPU) の登場などがある。CPU や Graphics Processing Units (GPU) のコアが複数個搭載された計算機を用いることによって各プロセスに処理を分割することができるようになり並列演算が可能になった。また、それぞれに並列演算のための開発環境が用意され、以前より開発が比較的容易になってきている。GPU は昨今のスーパーコンピュータにも採用されており東京工業大学の TSUBAME には GPU が 680 基、長崎大学のスーパーコンピュータには 760 基搭載されており、それぞれ性能ランキングや賞の受賞を通して評価されてい

る. GPU は一般にも入手しやすいため, 大規模なシステムだけでなく小規模な開発においても注目されている.

本論文では, FDTD 法で計算する 3 次元声道音響シミュレータを 2 つの並列演算方法を用いて高速化し, それぞれの計算速度や計算結果を逐次計算と比較した. 2 章では, 音響 FDTD 法や吸収境界条件についての式を示し, 3 章では GPU とマルチコア CPU の並列計算の概要を述べる. 4 章では, 3 次元声道音響シミュレータの実装方法を示し, 並列計算による計算時間の短縮について述べる.

2 3 次元音場シミュレーション

2.1 音響 FDTD 法

音響 FDTD 法とは音響の波動方程式を時間的, 空間的に中心差分した差分法の一つである. FDTD 法は電磁場解析などによく利用されるが, 音響 FDTD 法では各パラメータを音波用のものに置き換えて利用できる. FDTD 法の長所としてアルゴリズムが比較的簡単でプログラムがしやすいという点が挙げられる. また, 解析空間が複雑な形状であっても簡単に解析できる. 短所として計算機のメモリを大量に使用し, 計算時間もかかるという点があり, 3 次元空間を扱う場合には大きな課題となる.

はじめに, 離散化する前の音波の 1 次元波動方程式 (文献 [1]) を示す.

$$\frac{\partial p}{\partial t} = -\kappa \frac{\partial u}{\partial x} \quad (1)$$

$$\frac{\partial u}{\partial t} = -\frac{1}{\rho} \frac{\partial p}{\partial x} \quad (2)$$

ここで p は音圧, u は粒子速度, κ は弾性体積率, t は時間, ρ は媒質の密度である. これらの式を以下のように時間的, 空間的に差分化したものが FDTD 法である.

$$p^{n+\frac{1}{2}}(i) = p^{n-\frac{1}{2}}(i) - \kappa \frac{\Delta t}{\Delta x} \left\{ u^n \left(i + \frac{1}{2} \right) - u^n \left(i - \frac{1}{2} \right) \right\} \quad (3)$$

$$u^{n+1} \left(i + \frac{1}{2} \right) = u^n \left(i + \frac{1}{2} \right) - \frac{1}{\rho} \frac{\Delta t}{\Delta x} \left\{ p^{n+\frac{1}{2}}(i+1) - p^{n+\frac{1}{2}}(i) \right\} \quad (4)$$

ここで Δt は離散時間間隔, Δx は離散空間間隔, n は時間ステップ, i は空間離散点である. この 2 式を時間配置によって図示すると図 1 のように表すことができる. p の点と u の点が時間的にも空間的にも半ステップずれた差分となっており, p と u を交互に計算するアルゴリズムとなっている. これを蛙跳び (Leap-frog) アルゴリズムと呼ぶ.

次に次元を拡張して 3 次元空間における波動方程式を示す.

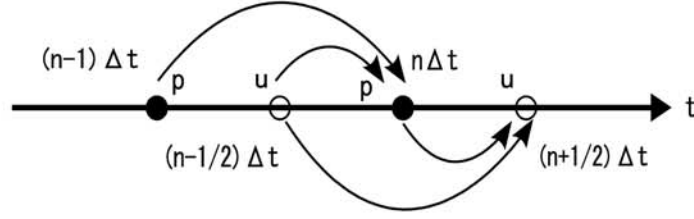


図 1: FDTD 法における時間配置 (矢印終点の計算には始点の 2 つの値を利用)[1]

$$\frac{\partial p}{\partial t} = -\kappa \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} \right) \quad (5)$$

$$\frac{\partial u_x}{\partial t} = -\frac{1}{\rho} \frac{\partial p}{\partial x} \quad (6)$$

$$\frac{\partial u_y}{\partial t} = -\frac{1}{\rho} \frac{\partial p}{\partial y} \quad (7)$$

$$\frac{\partial u_z}{\partial t} = -\frac{1}{\rho} \frac{\partial p}{\partial z} \quad (8)$$

ここで u_x , u_y , u_z は、各軸方向の粒子速度である。これらの式を離散化し、FDTD 法にすると以下の式になる。

$$\begin{aligned} p^{n+\frac{1}{2}}(i, j, k) &= p^{n-\frac{1}{2}}(i, j, k) - \kappa \frac{\Delta t}{\Delta x} \left\{ u_x^n \left(i + \frac{1}{2}, j, k \right) - u_x^n \left(i - \frac{1}{2}, j, k \right) \right\} \\ &\quad - \kappa \frac{\Delta t}{\Delta y} \left\{ u_y^n \left(i, j + \frac{1}{2}, k \right) - u_y^n \left(i, j - \frac{1}{2}, k \right) \right\} \\ &\quad - \kappa \frac{\Delta t}{\Delta z} \left\{ u_z^n \left(i, j, k + \frac{1}{2} \right) - u_z^n \left(i, j, k - \frac{1}{2} \right) \right\} \end{aligned} \quad (9)$$

$$u_x^{n+1} \left(i + \frac{1}{2}, j, k \right) = u_x^n \left(i + \frac{1}{2}, j, k \right) - \frac{1}{\rho} \frac{\Delta t}{\Delta x} \left\{ p^{n+\frac{1}{2}}(i+1, j, k) - p^{n+\frac{1}{2}}(i, j, k) \right\} \quad (10)$$

$$u_y^{n+1} \left(i, j + \frac{1}{2}, k \right) = u_y^n \left(i, j + \frac{1}{2}, k \right) - \frac{1}{\rho} \frac{\Delta t}{\Delta y} \left\{ p^{n+\frac{1}{2}}(i, j+1, k) - p^{n+\frac{1}{2}}(i, j, k) \right\} \quad (11)$$

$$u_z^{n+1} \left(i, j, k + \frac{1}{2} \right) = u_z^n \left(i, j, k + \frac{1}{2} \right) - \frac{1}{\rho} \frac{\Delta t}{\Delta z} \left\{ p^{n+\frac{1}{2}}(i, j, k+1) - p^{n+\frac{1}{2}}(i, j, k) \right\} \quad (12)$$

また, 離散空間間隔 ($\Delta x, \Delta y, \Delta z$) と離散時間間隔 (Δt) は式 (13) の安定条件を満たすように設定する必要がある. これを満たさない場合, 値が発散するなど安定性が保証されない.

$$\Delta t \leq \frac{1}{c \sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}} \quad (13)$$

ここで c は音速である.

FDTD 法のアルゴリズムを図 2 に示す. はじめに初期値設定で各パラメータの設定をし, 波源入力から音圧の記録まで順番通りに処理する. シミュレーション時間 ($t_{\text{Sim_max}}$) が終了するまで繰り返し計算を行い, 結果を出力する. 開口端から x 軸方向に 5 cm 離れた点でステップ毎に音圧の値を記録した.

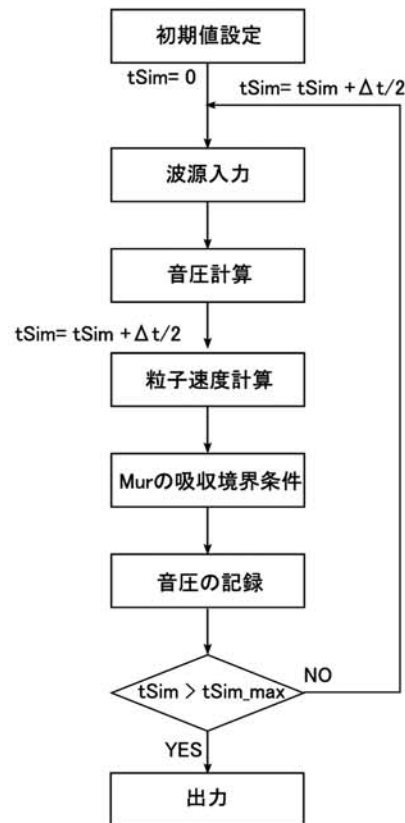


図 2: FDTD 法のアルゴリズム

2.2 吸収境界条件

解析領域が剛壁に囲まれている場合は, 境界の粒子速度を 0 とすればよい. しかし, 自由空間を含んだモデルを解析する場合は, 計算量の問題で莫大な領域を解析するわけにはいかない. そこで擬似

的に無反射の境界を設定する必要がある。この条件を吸収境界条件という。領域の境界上では解析領域とは異なる処理が必要になる。吸収境界条件にはいくつかの手法があり、Mur の吸収境界条件 [2] と PML (Perfect Matched Layer) [3] の 2 つがポピュラーである。Mur の吸収境界条件は精度は中程度で簡単に導入しやすく、PML は精度の面では最も良いが計算量が Mur より増える。また、PML は Mur の吸収境界条件に比べ精度は良いものの使用メモリが多く、アルゴリズムも複雑であるため並列計算、高速計算において多少不利である。以下に 1 次元における Mur の 1 次の吸収境界条件を示す。

$$u^{n+1}(NX) = u^n(NX - 1) + \frac{c\Delta t - \Delta x}{c\Delta t + \Delta x} \{u^{n+1}(NX - 1) - u^n(NX)\} \quad (14)$$

ここで NX は吸収境界条件を設定する点である。Mur の 1 次吸収境界条件では、境界面に対して垂直な波しか吸収できない。1 次元の解析をする場合は音波が吸収点において垂直に入射するため 1 次の吸収境界で吸収させることができる。しかし、3 次元空間の解析では、斜め入射が発生するため 1 次の吸収境界では十分に吸収しなくなる。そこで 3 次元空間の解析の場合には 2 次吸収境界を設定する。 x 軸方向における Mur の 2 次吸収境界条件を式 (15) に示す。

$$\begin{aligned} u_x^{n+1}(NX, j, k) = & -u_x^{n-1}(NX - 1, j, k) + \frac{c\Delta t - \Delta x}{c\Delta t + \Delta x} \{u_x^{n+1}(NX - 1, j, k) - u_x^{n-1}(NX, j, k)\} \\ & + \frac{2\Delta x}{c\Delta t + \Delta x} \{u_x^n(NX, j, k) - u_x^n(NX + 1, j, k)\} \\ & + \frac{\Delta x(c\Delta t)^2}{2(\Delta y)^2(c\Delta t + \Delta x)} \{u_x^n(NX, j + 1, k) - 2u_x^n(NX, j, k) + u_x^n(NX, j - 1, k) \\ & \quad + u_x^n(NX - 1, j + 1, k) - 2u_x^n(NX - 1, j, k) + u_x^n(NX + 1, j - 1, k)\} \\ & + \frac{\Delta x(c\Delta t)^2}{2(\Delta z)^2(c\Delta t + \Delta x)} \{u_x^n(NX, j, k + 1) - 2u_x^n(NX, j, k) + u_x^n(NX, j, k - 1) \\ & \quad + u_x^n(NX - 1, j, k + 1) - 2u_x^n(NX - 1, j, k) + u_x^n(NX + 1, j, k - 1)\} \quad (15) \end{aligned}$$

これらの式を自由空間のそれぞれの終端面に設定した。

3 並列演算

3.1 GPU と CUDA

GPU は、CPU よりも並列性が非常に高いデバイスである。この理由はアーキテクチャの違いにあり、メインメモリとの接続を比較してみると CPU では 64 bit、GPU では 512 bit 程度で、GPU ではデータ転送が高速にできる。NVIDIA 製の GPU のアーキテクチャを図 3 に示す。

NVIDIA 製の 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ている。GPU チップの内部には Streaming Multiprocessor (SM) というマルチプロセッサが多数搭載されており、この SM には Streaming Processor (SP) という演算処理ユニット 8 個と共有メモリ 1 個が入っている。この共有メモリと SP は非常にデータアクセスが速く、デバイスメモリへのアクセスより 100 倍以上速い。代表

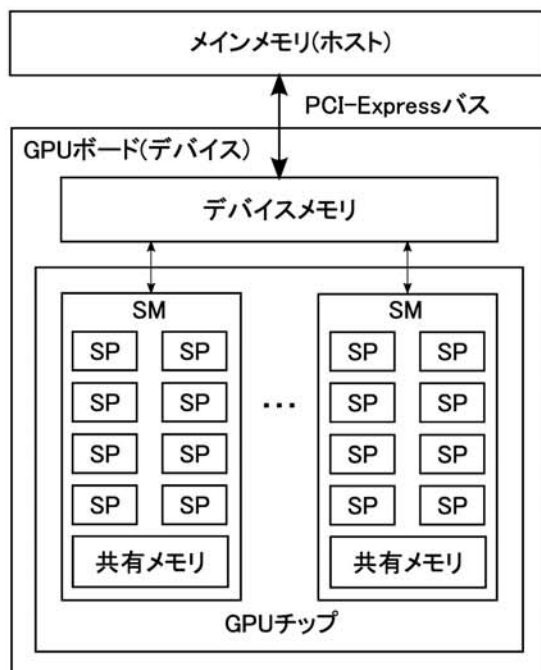


図 3: NVIDIA 製 GPU のアーキテクチャ

的なGPUではSMが30個、SPが240個入っている。これは、1.4 GHz程度の演算処理ユニットが240個あることになる。SPはCPUのコアと比べると演算性能が低く、単純には比較できないがSPの数とCPUのコア数を比較すると大きな違いであるといえる。さらにGPUにおける性能の特徴としてデバイスメモリのメモリバンド幅が大きいことが挙げられる。GPUのメモリバンド幅は100 GB/secを超えるものが多く、CPU(例えばIntelのCore i7)ではせいぜい36 GB/sであるため3倍以上の差がある。

従来、GPUを利用してプログラムを作成するには専門的な知識が必要であった。専門的な知識とは、グラフィックスAPIとシェーダ言語を用いたプログラミングの知識などである。この問題を解決するためにNVIDIA社はCompute Unified Device Architecture (CUDA)という開発環境を提供している。CUDAによって専門的な知識を持たない人でもGPUを利用しやすくなったため近年盛んに研究が行われている。

CUDAの基本的なプログラム(図4)について説明する。はじめにCPU(ホスト)から実行が開始され、ホストメモリ用とデバイスメモリ用のメモリポインタを宣言する。次にCUDAランタイムAPIである`cudaMalloc`を使ってデバイスにメモリ(配列)を確保する。ホストメモリをデバイスメモリにコピーし、カーネル関数からデバイスコードを呼び出す。こうすることによりデバイスメモリを使用することができ、高速な演算をさせることができる。ここで、デバイスに確保したメモリをマルチスレッドで処理するためにカーネル関数で引数を設定している。CUDAではグリッドという最上位の階層があり、グリッドの中にブロック、ブロックの中にはスレッドという階層が設けられている。これらのグリッドやスレッドの数を適切に設定し、カーネル関数で指定するとデバイスメモリが並列に処理される[4]。

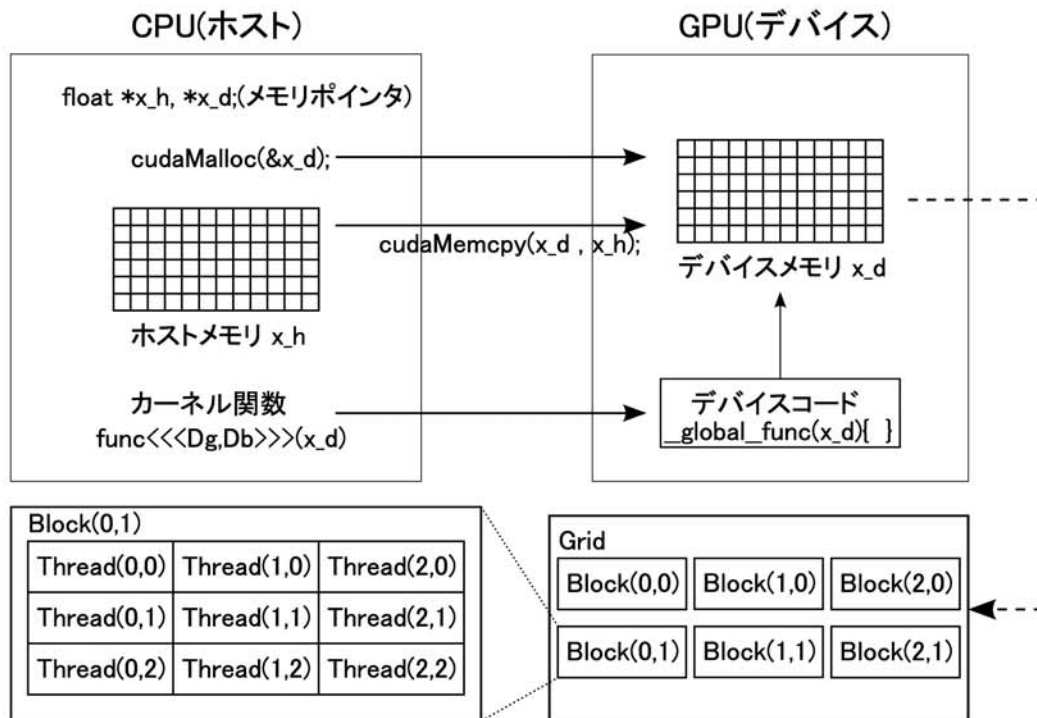


図 4: CUDA の基本的なプログラム [4]

3.2 OpenMP

OpenMP は、共有メモリマルチプロセッサ上のマルチスレッドプログラミングのための標準 API である。これまでシングルコア CPU で逐次的に実行されていた処理を各スレッドが並列に実行し、より高速な処理を可能にする。これは、新しい言語でなく既存のソースにプラグマ (並列化指示文) を追加することで並列演算を実現する。特徴としては容易に並列化ができることが挙げられ、ベースとなる言語 (Fortran, C/C++) で書かれたプログラムが用意できていれば並列化できる。並列化の前後でほとんど同じソースが利用でき、並列化と非並列化のスイッチングが簡単である。並列化をするための指示文は難しくなく、並列化用に最適化された逐次プログラムを作成することが重要になる。並列演算特有のスレッドの生成や各スレッドの同期コントロールのような制御についてユーザが意識する必要がない。

OpenMP の実行モデルとして Fork - join (図 5) と呼ばれる並列実行処理が用いられている。OpenMP で記述されたプログラムは、マスタースレッドが起動され、実行が開始される。マスタースレッドがプラグマの記述されている部分に到達すると複数のスレーブスレッドを生成 (fork) し、タスクを並列に処理する。全スレッドの処理が終了するまで、先に終了したスレッドは待機状態となる。全スレッドの処理が終了した地点 (join) で再びマスタースレッドに処理が渡される。このような処理を繰り返すことから Fork-join モデルと呼ばれる。また、1つのプログラム内においてプラグマをいくつでも使用することができる [5]。

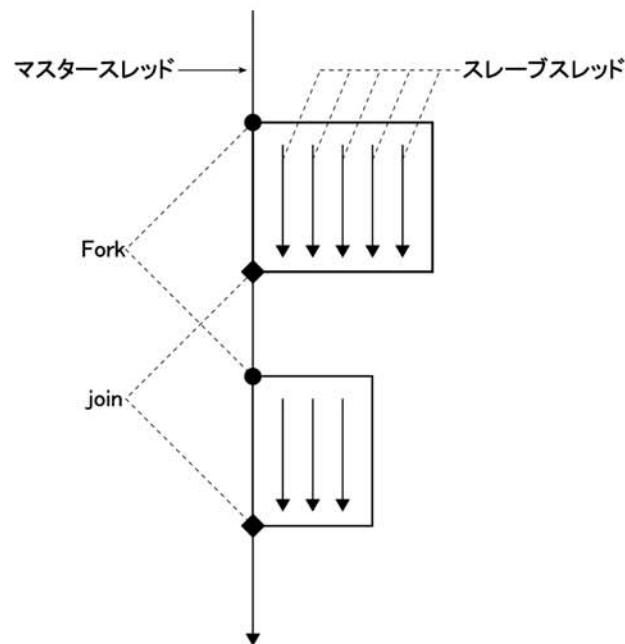


図 5: Fork-join モデル [5]

4 並列化の効果の評価実験

4.1 条件

4.1.1 解析空間

FDTD 法を用いて計算する 3 次元声道音響シミュレータについて説明する. 声道とはヒトの声帯から口唇までの間の管状の空間で, 断面積変化のある複雑な形状となっている. 本シミュレータは, 声道の形状とその音響特性の関係など様々な知見を得ることを目的とするものである. 例えば, ガウシアンパルスを入力することによってインパルス応答を求めることができる. また, 声帯音源波を入力することによって母音を合成することもできる.

FDTD 法で解析を行う声道の形状は, 磁気共鳴画像法 (Magnetic Resonance Imaging, MRI) を用いて得られた声道断面積関数を用いた. 声道断面積関数とは, 声帯側から順に一定の間隔で断面積が記述されているデータである. 使用した声道断面積関数は, ATR 母音発話 MRI データベース [6] から抽出したものである. 母音/a/の声道断面積関数をもとに声道の解析領域の 3 次元のシミュレーションモデル (図 6) を作成した. なお, 実際の声道形状はこのように直線的でなく複雑に曲がった形をしている.

この図の左端の面が声門面, 声道部分の右端を口唇面であり, 口唇面より右の大きな空間は自由 (解放) 空間である. 自由空間のサイズは, 口唇から放射された音波を吸収境界条件で十分に吸収できるように設定した. 音波を十分に吸収するには開口端 (口唇面) と吸収境界面との距離を長く取れば取るほどよい. これは, 距離を長くすることで各吸収境界面に音波が垂直に入射しやすくなり, 音波の吸収がしやすくなるからである. これにより解析時のノイズを減らすことができる. ただし, 自由空間を

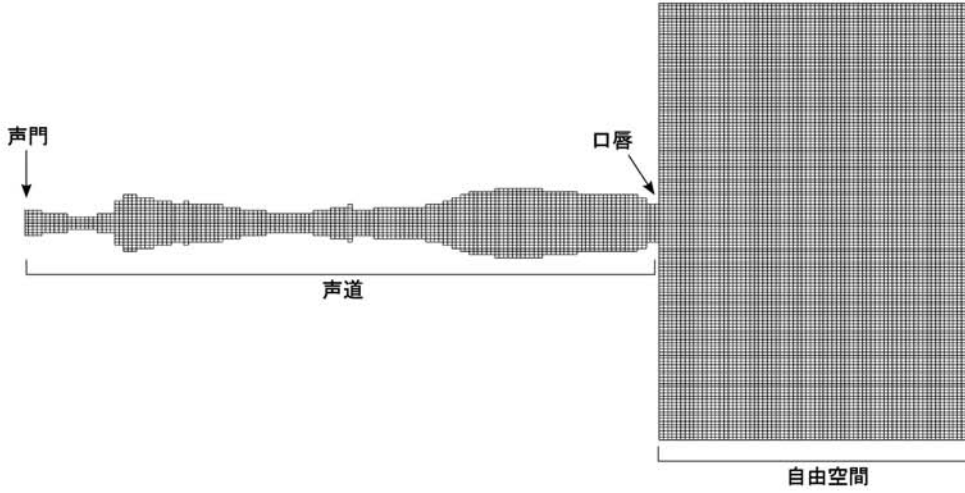


図 6: 母音/a/のシミュレーションモデル

広くしすぎると計算量の増大につながる. 声門面から口唇面へ向かう方向を x 軸方向とすると, x 軸方向については, 口唇面から 10 cm 程度離れた距離をとり, 他の境界面は口唇面の断面積の 10 倍の面積になるように設定した. このような領域をマップとして作成し, マップを参照することで計算する領域か否かを判定し, 音圧や粒子速度を計算した. また, 入力波を与える位置は声門面の全てのセルとし, ガウシアンパルスを与えた. 音波の記録点は, 開口端から x 軸方向に 5 cm 離れた点とした.

シミュレーション条件を表 1 に示す. シミュレーション条件は, 式 (13) の安定条件にあてはめると左辺は 1.0×10^{-6} , 右辺は 2.1×10^{-6} となりこの式を満たす.

表 1: シミュレーション条件

離散空間間隔 $\Delta x = \Delta y = \Delta z$	1.25 mm
離散時間間隔 Δt	1.0×10^{-6} s
解析空間	$256 \times 160 \times 160$ セル
音速 c	352.9 m/s
媒質の密度 ρ	1.12 kg/m^3
吸収境界条件	Mur の 2 次吸収境界条件

4.1.2 プログラム

上記の方法, 条件を用いて 3次元声道音響シミュレータを(1) CUDA を用いた GPU 計算と(2) OpenMP を用いた並列計算と(3) 逐次計算 (CPU1 コアによる計算) で実装した. シミュレータはまず CPU1 コア用に C 言語で開発し, CUDA と OpenMP 用にそれぞれ書き換えた. CUDA と OpenMP による実装では, 音圧と粒子速度の計算を並列化した. 音圧と粒子速度の繰り返し計算は FDTD 法で最も時間の

かかる処理であるため、音圧と粒子速度の解析領域を分割してスレッドに割り当て、並列計算を行った。いずれの手法でも 3 次元空間を 1 次元配列で表現し、単精度 (float) で計算を行った。

CUDA を利用したプログラムでは、図 2 の波源入力から Mur の吸収境界条件までの処理をデバイス (GPU) で処理した。デバイスからホスト (CPU)、ホストからデバイスへのメモリのコピーは、時間のかかる処理であるため極力行わないようにした。各処理ごとにカーネル関数を呼び出して処理をし、同期を取った。音圧計算と粒子速度計算ではブロックを 12,800 個、スレッドを 512 個に設定してマルチスレッド処理を行った。音圧の記録をする処理はデバイスから記録点の値をホストにコピーした。一方、OpenMP を利用したプログラムは音圧、粒子速度の解析領域を CPU のコア数で割ってマルチスレッド処理を行った。

4.1.3 実験環境

GPU の計算および CPU のみの計算の実行環境は、CPU に Intel Core i7 920 (2.66 GHz)、メインメモリに 3 GB を搭載した同じ Windows XP の PC を用いた。使用した GPU は NVIDIA 社の Tesla C1060 で CUDA のバージョンは 3.1 である。そのスペックを表 2 に示す。

表 2: Tesla C1060 のスペック

シェーダクロック	1300 MHz
メモリクロック	16,000 MHz
メモリバンド幅	102 GB/s
デバイスメモリ	4 GB
SP 数	240
SM 数	30

OpenMP はコンパイラに gcc のバージョン 4.3 を使用し、最適化に関するコンパイラオプションとして -O2 を指定した。

4.2 結果

はじめに計算時間について述べる。3 次元声道音響シミュレータを CUDA, OpenMP, 逐次計算の 3 つの方法によって 200 ステップ計算し、それぞれの計算に要した時間を図 7 に示す。CUDA は 1.39 秒、OpenMP は 17.46 秒、逐次計算は 42.79 秒という結果が得られた。CUDA と逐次計算の計算時間を比較すると約 30 倍の計算時間短縮が実現され、OpenMP と逐次計算では約 2.5 倍の計算時間の短縮ができた。また、CUDA の実行時のメモリバンド幅は、理論ピークメモリバンド幅 102 GB/s のうち 90 GB/s 程度出ており高いメモリバンド幅を引き出すことができた。OpenMP は、実行環境の CPU にコアが 8 個入っているため 4 倍程度まで時間を短縮できると推測したがそうではなかった。この原因は、解析領域のマップを参照しながら音圧や粒子速度を計算しているためメモリアクセスに無駄が発生するためであると考えられる。

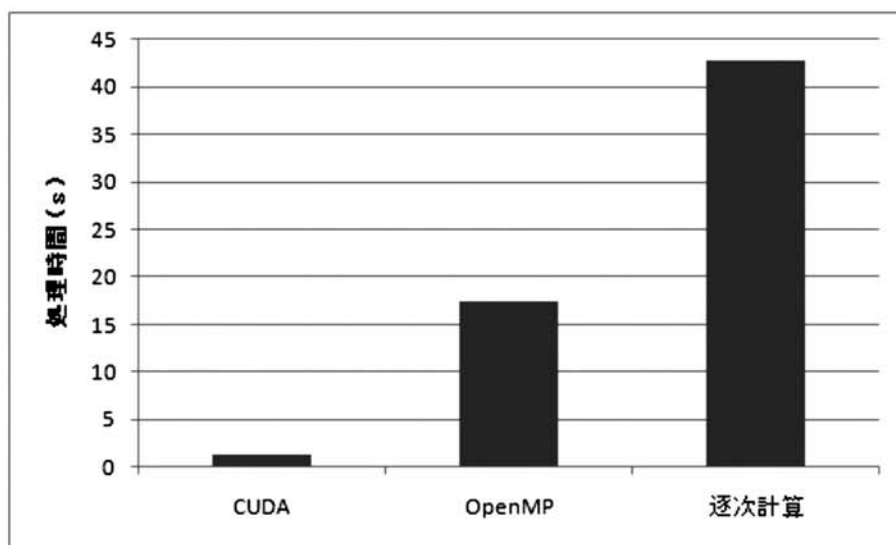


図 7: 3 手法による計算時間の比較

次に CUDA で計算した伝達特性 (出力結果) を図 8 に示す. CUDA プログラムのカーネル関数内の計算は, 計算順序が逐次計算とは異なり計算結果も異なることがある. CUDA で計算した伝達特性と C で計算したものを比較したところわずかに差異は確認されたが, 伝達特性を出力するシミュレータとしては全く問題のない程度であった.

5 まとめ

本研究は, FDTD 法を用いて 3 次元声道音響シミュレータを開発し, 並列計算によって計算時間の短縮を試みた. GPU による並列計算の効果は大きく, 逐次計算と比較すると 30 倍程度の速度で得られた. また, CUDA と C の計算結果を比較し差異も検討した. CUDA で開発したシミュレータで伝達特性を求めた結果は逐次計算のものと差異はほとんどなくシミュレータとして問題なく使用できるレベルであった.

今後の課題は, CUDA プログラムのメモリアクセス方法やアルゴリズムを見直し, 無駄をさらに減らすことである.

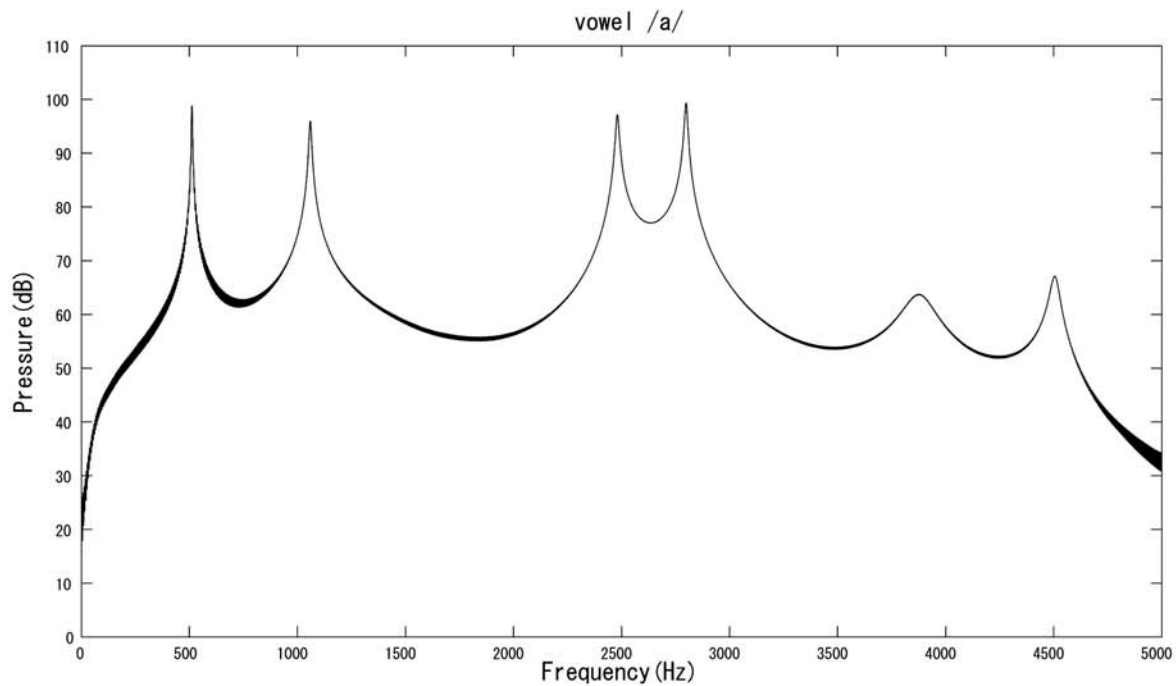


図 8: 母音/a/ の伝達特性

謝辞

本研究を進めるにあたり, ご助言を頂きました甲南大学知能情報学部若谷彰良教授に感謝いたします。2010 年度科研費 (21300071) の援助に感謝いたします。

参考文献

- [1] 橋本 修, 実践 FDTD 時間領域差分法. 森北出版, 2006.
- [2] G. Mur, “Absorbing boundary condition for the finite-difference approximation of the time-domain electromagnetic-field equation,” *IEEE Trans. Electromagnetic Compat.*, vol. EMC-23, no. 4, pp. 377–382, 1981.
- [3] J.-P. Berenge, “A Perfectly Matched Layer for the Absorption of Electromagnetic Waves,” *Journal of Computational Physics*, vol. 114, no. 1, pp. 185–200, 1993.
- [4] 青木尊之, 額田彰, はじめての CUDA プログラミング. 工学社, 2009.
- [5] 戸室隆彦, インテルコンパイラー OpenMP 入門. インテル株式会社, 2005.
- [6] ATR-P BAIC, ATR 母音発話 MRI データ CD-ROM 説明書第 4 版. http://www.baic.jp/product/product_artic.html, 2007.