GPU 計算を用いたイメージベース EFIT の高速化と 超音波計測による解析妥当性の検証

Acceleration of Image-based EFIT using GPU Computing and
Validation by Ultrasonic Measurement

中畑和之*・田中貴之**

Kazuyuki NAKAHATA and Takayuki TANAKA

*正会員 博士(工) 愛媛大学准教授, 大学院理工学研究科生産環境工学専攻 (〒790-8577 愛媛県松山市文京町 3)

** (研究当時) 学生会員 修士(工) 愛媛大学 大学院理工学研究科生産環境工学専攻 (同上)

For nondestructive testing, we developed a simulation tool of elastic wave propagation in a material with complex outer surfaces or various inclusions. The tool was based on the elastodynamic finite integration technique (EFIT) with an image-based modeling. In this study, the image-based EFIT code is accelerated using graphics processing units (GPUs) with the CUDA Fortran programming language. The methodology on the speed-up using the CUDA Fortran is described, and checked by numerical experiments with multiple GPU boards. The calculation speed with a GPU can be dramatically improved in comparison with the speed obtained by running the same simulation on a classical CPU. In addition, the validation of the image-based EFIT is performed by ultrasonic measurements of SH Lamb wave. The results of the simulation show good agreements with the measurement.

Key Words : *GPGPU, CUDA Fortran, image-based EFIT, validation, ultrasonic nondestructive testing*

1. はじめに

著者らは, これまでに動弾性有限積分法^{1, 2)}(Elastodynamic Finite Integration Technique: EFIT) にイメージベース処理³⁾を適用した弾性波動のシミュレーション手法^{4, 5)}について研究を行ってきた. この手法は主として非破壊検査の波動シミュレーションを意図したものであり, モデル作成の簡便さと計算時間の短縮を重点的に工夫したものである. EFIT は, 支配方程式 (偏微分方程式) をセルと呼ばれる微小正方形 (2 次元) or 立方体 (3 次元) で積分した後, その積分方程式を離散化する方法であり, 陽的に解の更新を行うことからマルチ CPU 使用時の並列計算効率がよいことが示されている⁶⁾.

近年では, パーソナルコンピュータに搭載されたグラフィックボード (以下, GPU と略す) の科学技術計算に果たす役割が年々加速している⁷⁾. 汎用計算に GPU を使用する試みは GPGPU (General-purpose computing on GPUs) と呼ばれ, 国内外で盛んに行われている. 最近のデスクトップ CPU の浮動小数点演算の理論ピーク性能はさほど伸びてはおらず, 現時点で Intel Core i7 系で 50GFLOPS 程度にとどまっている. 一方, GPU はコアの数が CPU に比べて多く, ビデオメモリとの通信バンド幅を広く備えるために, CPU と比べて性能の成長率が高い. このように GPU を取り巻く環境が急激に進化したことにより, GPU をグラフィック処理専用の

CPU の補助プロセッサとしてだけでなく, 並列計算ユニット⁸⁾として利用するなど活発化している. また, GPU のハードウェアとしての性能の向上ばかりでなく, GPU の持つ演算機能をプログラムする環境も整備されつつある.

本研究では, NVIDIA 社の統合開発環境 CUDA を用いて, GPU 計算のためのイメージベース EFIT のコーディングを行い, PC レベルで高速化を行うことを目的とする. CUDA は C 言語ベースの開発環境 (以後 CUDA C と呼ぶ) であり, CUDA C の API を使用して明示的にプログラミング⁹⁾する. これまで, 弾性波動問題では, CUDA C を利用した有限差分法の GPU 計算例が数多く報告^{10, 11)}されている. しかし, 本研究では, 著者等の過去の EFIT コードを拡張して用いるために CUDA Fortran¹²⁾を用いた. これまで, Fortran アプリケーションから CUDA を使用するためには, CPU-GPU のデータ移動, 関数呼び出しなど, 詳細なレベルで C 言語を使わざるを得ない状況であったが, CUDA Fortran では, すべて Fortran 言語の記述だけでコードを構築できるのが特徴である. Fortran 言語と C 言語では, 配列のメモリ構造や, サブプログラムへの配列の参照方法が異なるため, CUDA C によるプログラミングとは並列計算の粒度や領域分割等の仕方によって性能が異なるものとする. そこで, 本論文では CUDA Fortran によるイメージベース EFIT の高速化について述べ, GPU 計算の粒度に着目した実効性能について検証する. こ

れまで、イメージベース EFIT は非均質材料や複雑形状を扱うためにコード内で分岐処理が多用されており、上記の報告例^{10, 11)}のように均質媒体が大部分を占める状況とは使用環境が異なる。GPU の各プロセッサは SIMD(Single Instruction Multiple Data) 型のデータ処理をこなすために、分岐が入るとオーバーヘッドがかさみ、効率が低下することが指摘されている。そこで、イメージベース EFIT を高速化する手段としてここでは Lookup table の導入を行った。また、GPU は構造上、単精度の浮動小数点演算を得意としているため、科学技術計算で一般的に用いられている倍精度以上の浮動小数点演算では性能が低下する。そこで、単精度計算でも精度を保証しながらイメージベース EFIT を使用することができるか、単精度と倍精度計算の残差についても検証を行った。

一方、近年はシミュレーションの妥当性を示すために実験結果との比較が要求されており、日本学術会議でもシミュレーションの品質保証に関する規格化に向けた検討¹³⁾がなされている。これまで EFIT の信頼性に関する研究はあまり行われていない。そこで、本論文の後半で、解析と計測実験とを比較し、本手法の妥当性の検証を行うことを本論文のもう一つの目的とする。

2. 計算機の仕様と開発環境

2.1 本研究で用いた GPU 計算機

本研究では、GPU ボードとして NVIDIA 社の Tesla C2075 を用いた。C2075 には 448 個のストリーミング・プロセッサ (SP) と呼ばれる最小単位の演算処理ユニットが内蔵されている。この SP が 32 個集まって 1 つのストリーミング・マルチプロセッサ (SM) を形成する。従って、Tesla C2075 に SM は 14 個存在する。SM 内部には、高速メモリであるレジスタメモリやシェアードメモリ等のオンチップメモリがあるが、いずれも数十 KB 程度しかない。一方、オフチップメモリとして大容量の GDDR5 RAM(以下、グローバルメモリと呼ぶ)6GB を搭載している。グローバルメモリと CPU 側の DDR3 SDRAM(以下、ホストメモリと呼ぶ)をつなぐインターフェースは PCI-Express 2.0 を用いた。表-1

表-1 Tesla C2075 のハードウェア性能 (理論値)。

		Tesla C2075(GPU)
プロセッサ	SP数	448個
	クロック周波数	1147MHz
	ピーク演算性能	1027.7GFLOPS(単精度) 515GFLOPS(倍精度)
メモリ	標準メモリ設定	6GB GDDR5
	メモリバンド幅	144GB/s

に Tesla C2075 のハードウェア性能を示す。例えば Intel Core i7 系の CPU と比較して、理論ピーク演算性能 (倍精度) とメモリバンド幅 (主記憶) が、それぞれ 10 倍と 5 倍程度大きい。

本研究で用いた計算機は、パーソナルワークステーションに GPU ボードを挿入したものである。ここでは 2 台のワークステーション (以下、ノードと呼ぶ) を用意した。各ノードに GPU ボード (すべて Tesla C2075) が 3 つ挿入されている。ノードは 2 台あるので、本研究では最大で 6 つの GPU ボードが使用できる。

2.2 開発環境

(1) CUDA Fortran

本研究では、GPU 計算を行うためのプラットフォームとして CUDA Fortran を用いた。CUDA Fortran を提供するのは、現在のところ PGI 社¹²⁾のみであり、ここでは PGI Accelerator Fortran Ver11.10 for Linux を用いた。CUDA Fortran のプログラムは、CPU を動作させるホスト用コードと GPU を動作させるデバイス用コードの 2 つのコードから成り立つ。(以下では、CPU 側の系をホスト、GPU 側の系をデバイスと呼ぶ。) ホスト用コードには、通常のプログラミングの他に CUDA のランタイム API、Built-in のベクトル型宣言、およびデバイス用コードを呼ぶ call 文等が含まれている。

(2) CUDA の階層構造

図-1 に示すように、CUDA はグリッド、ブロック、スレッドという並列計算の粒度単位を持っている。下向きの波線矢印で示したスレッドが計算を行う最小単位であり、そのスレッドの集まりがブロック、ブロックの集まりがグリッドとなる。ブロックとスレッドは 3 次元的な構造をしており、それぞれ最大数が決められている。図-1 上側に示すように、1 つのグリッド内の 1 方向のブロックの最大数は 65535 個である。また、1 ブロック内で使用できるスレッドの最大数は、 x_1 方向が 1024 個、 x_2 方向が 1024 個、 x_3 方向が 64 個である。ただし、1 ブロック内の合計スレッド数は 1024 個までという制限がある。なお、ブロックやスレッドの ID は構造体で管理され、CUDA C と異なり、CUDA Fortran では 1, 2, ..., 65535 とカウントされることに注意が必要である。

プログラム実行時、グリッド上にデバイスコードの実行形式 (カーネル) を生成し、スレッドそれぞれがそのカーネルを実行する。従って、GPU 上の各プロセッサは SIMD 型のデータ処理を実行することになる*。

* CUDA におけるカーネルは if 文等の分岐命令も実行できるため、厳密な SIMD 処理ではなく、NVIDIA 社は SIMT(Single Instruction Multiple Thread) と呼んで区別して表記している。

図-1 CUDA Fortran における計算の階層 (グリッド, ブロック, スレッド)。

3. イメージベース EFIT への GPU 計算の導入

3.1 イメージベース EFIT

イメージベース EFIT の詳細な定式化については、過去の論文^{4,5)}に記載されているので、ここでは概略について述べる。本論文では、2次元弾性波動場を対象とし、波動の偏向方向は面外とする。この場合の弾性波はSH(Shear Horizontal)波と呼ばれる。x₃軸を面外方向とし、SH波は(x₁, x₂)面を伝搬するものとする。x₃軸方向の粒子速度をv₃、せん断応力をτ₃₁、τ₃₂とおいたとき、SH波の伝搬を支配する波動方程式および構成式は以下ようになる。

$$\rho(\mathbf{x})\dot{v}_3(\mathbf{x}, t) = \frac{\partial \tau_{31}(\mathbf{x}, t)}{\partial x_1} + \frac{\partial \tau_{32}(\mathbf{x}, t)}{\partial x_2} \quad (1)$$

$$\frac{\dot{\tau}_{31}(\mathbf{x}, t)}{\mu(\mathbf{x})} = \frac{\partial v_3(\mathbf{x}, t)}{\partial x_1} \quad (2)$$

$$\frac{\dot{\tau}_{32}(\mathbf{x}, t)}{\mu(\mathbf{x})} = \frac{\partial v_3(\mathbf{x}, t)}{\partial x_2} \quad (3)$$

ここで、 $\dot{(\)}$ は時間 t に関する偏微分 ($\partial/\partial t$) を表す。上式で、 ρ は材料の密度、 μ はせん断弾性係数を表し、等方性材料の場合には横波音速との関係式 $c_T = \sqrt{\mu/\rho}$ が成立する。

離散化後は、陽的に解を更新するアルゴリズムとなる。x₁ 方向の配列要素を i 、x₂ 方向のそれを j とする

と、上式の支配方程式は次のように解くことになる。

$$V_3(i, j) \Leftarrow V_3(i, j) + \delta_1 [T_{31}(i+1, j) - T_{31}(i, j)] + \delta_2 [T_{32}(i, j+1) - T_{32}(i, j)] \quad (4)$$

$$T_{31}(i, j) \Leftarrow T_{31}(i, j) + \varepsilon_1 [V_3(i, j) - V_3(i-1, j)] \quad (5)$$

$$T_{32}(i, j) \Leftarrow T_{32}(i, j) + \varepsilon_2 [V_3(i, j) - V_3(i, j-1)] \quad (6)$$

ここで、

$$\delta_1 = \frac{\Delta t}{\rho_{ij} \Delta x}, \quad \delta_2 = \frac{\Delta t}{\rho_{ij} \Delta x}, \quad \varepsilon_1 = \frac{\bar{\mu} \Delta t}{\Delta x}, \quad \varepsilon_2 = \frac{\bar{\mu} \Delta t}{\Delta x} \quad (7)$$

である。ここで、 $\Leftarrow$ は右辺の計算結果を左辺に代入することを示す。上式で、 Δx と Δt は、それぞれセル長および時間ステップ刻みであり、また、 $\bar{\mu}$ は x₁ 方向に隣り合うセル、 $\bar{\mu}$ は x₂ 方向に隣り合うセルにおいて μ の平均化処理を実行した値である。なお、丸め誤差を少なくするために、式(4)~(6)の V_3 、 T_{31} 、 T_{32} の計算は規格化して行っている。

3.2 スレッドと配列

前述のように、本研究では CUDA Fortran を用いてコーディングを行った。図-2 に示すように、計算領域が大きい場合には配列を副領域に分配する。図-1 では、ブロックとスレッド共に3次元配列で説明をしたが、SH波伝搬問題の EFIT は2次元解析であるので、ここではブロックとスレッドは2次元配列とする (x₃ 方向のブロック、スレッド数は共に1とする)。

図-2 に配列の割り当ての概要を示す。この例では、計算領域を4つの副領域に分割する。副領域を1つのグリッドとし、副領域①では赤線で囲んだ上下のブロックに分かれているとする。グリッド内部のブロックの大きさは構造体を使って表現される、この場合、gridDim%x=1, gridDim%y=2 となる。また、1つのブロックには8つの配列がある。本研究では、1つの配列の計算 (例えば式(4)における $V_3(1, 1)$ の更新) に1つのスレッドを割り当てる。図-2の①における1つのブロックでは、x₁ 方向のスレッド数は4、x₂ 方向のスレッド数は2となる。これは、CUDA Fortran の記述では、blockDim%x=4, blockDim%y=2 となる。従って、1つのグリッドでは、x₁ 方向に (gridDim%x) × (blockDim%x) 個、x₂ 方向に (gridDim%y) × (blockDim%y) 個の粒度でスレッドが作られることになる。

Fortran の特徴として、図-2の右側に示すように、配列は i のインデックスが先に動く順番でメモリ上に並び、従って、ブロック内でメモリが連続するのは、x₁ 方向に並ぶ場合であることに注意する。これは、CUDA Fortran でも同様である。また、実際、スレッド処理は1ワープ(32スレッド)単位で行われるため、スレッド数は32の倍数が推奨されている⁹⁾。

3.3 マルチ GPU 計算

Telsa C2075 の場合、グローバルメモリは 6GB 程度であり、これより大きな配列サイズを確保できない。従って、計算領域が大きい場合には、計算領域を分割して複数のデバイスに副領域を割り当てることが必要になる。本研究では、1 つの副領域の計算を 1 つのデバイスにタイアップする。すなわち、マルチプロセスに分割後、1 つのプロセスが 1 つの副領域を担当する。その 1 つのプロセスに 1 つのデバイスを割当てて、マルチ GPU 計算を行う。本研究では、2 次元 MPI 通信を用いて、プロセス間のデータ通信を行う。2 次元 MPI 通信時には、隣接する副領域の間で袖領域 (図-2 に示す色づけした帯部分) を交換する必要がある。通信のタイミングは次節で示すが、本研究では、上記のように 1 プロセスと 1 デバイスをタイアップさせているため、まずホスト側のメモリにデバイス側からデータを転送し、次にホストメモリを介して交換したデータをデバイスメモリに送るといった段階を踏んでいる。本研究で用いたハードウェア構成では、デバイス間、特にノード間のメモリバンド幅が小さいため、できるだけ最小限の通信となるように、 V_3 成分のみを通信に用いている。なお、通信ライブラリとして MPI-1 と MPI-2 をサポートする MPICH2(Ver.1.4.1p1) を用いた。

3.4 イメージベース EFIT の流れ

イメージベース EFIT の流れを図-3 に示す。まず、CPU 側で、イメージベース処理から得られた解析領域に関する情報や、材料定数やセル長等の条件を read する。また、袖通信用あるいは計算結果の出力のために V_3 の配列を CPU 側に確保する。次に、式 (4), (5), (6) を計算するために、デバイス側のグローバルメモリに

図-2 左: EFIT で用いられる 2 次元配列のインデックス, および副領域と袖通信の概要。右: Fortran における 2 次元配列のメモリ上の並び。

V_3, T_{31}, T_{32} に相当する配列を確保する。ここでは、それらを $d_V_3, d_T_{31}, d_T_{32}$ とする。式 (4), (5), (6) を計算するために、それぞれデバイス用コードを実行する。各デバイスコードの実行時、1 つのブロックに 1SM が割り当てられる。従って、ブロックの中で、高速なメモリ (シェアードメモリ, レジスタメモリ等) が使用できる。式 (4) を見れば、1 セルあたり 1 回の更新で d_T_{31} と d_T_{32} は計 4 回アクセスすることになるから、スレッド毎にグローバルメモリにアクセスするよりも、シェアードメモリにアクセスしたほうが効率的である。シェアードメモリにおける T_{31} の配列を、 s_T_{31} と書く。従って、1 ブロックの計算では、最初に $s_T_{31} \leftarrow d_T_{31}$ としてデバイスメモリからシェアードメモリにコピーしておき、

$$d_V_3(i, j) \leftarrow d_V_3(i, j) + \delta_1 [s_T_{31}(i+1, j) - s_T_{31}(i, j)] + \delta_2 [s_T_{32}(i, j+1) - s_T_{32}(i, j)] \quad (8)$$

のように計算する。ここで、 δ_1 と δ_2 はレジスタメモリに入っている。同様なコーディング手法を式 (5) と (6) にも適用する。また、マルチ GPU の場合、各時間ステップで V_3 の計算が終わったら、他のプロセスと袖領域のデータ通信を行う。なお、各ステップで途中経過を出力したい場合、このタイミングで d_V_3 成分をホストメモリに V_3 にコピーし、ファイル出力する。ここ

図-3 イメージベース EFIT の GPU 計算の流れ。

では、各プロセスから MPI-IO¹⁴⁾ を用いて write しており、高速化に寄与している。これらの動作を、決められたステップ数まで繰り返す。

3.5 Lookup table の導入

イメージベース EFIT は非均質材料や複雑な外部形状を容易にモデル化できることが特徴であり、従来の CPU 用のコード⁴⁾では、それに対応するために if 分岐を多用した複雑なネスト構造になっていた。CPU だけで計算する場合、使用できるキャッシュが GPU よりもずっと大きいので、ネスト構造でも高速な計算が可能であった。しかし、GPU コアは SIMD 型の処理を行うため、命令中に分岐処理が入るとオーバーヘッドがかさみ、効率が落ちることが予想される。ここでは Lookup table を用いて効率よく材料パラメータを参照することにした。図-3 のプロセスに示すように、数値解析の条件ファイルを読み込んだ後に Lookup table を作成する。これまで、非破壊検査のための多様なシミュレーションを実施したが、その場合、セルの多くが弾性領域であり、その隣のセルも弾性領域である場合が多かった。それ以外は、空洞(完全反射境界)、PML(吸収境界層)、ソース発生領域等のセルが占める。従って、時間更新ループに入る前にこれらの判定処理を行っておき、判定結果(1:弾性領域、2:その他)を、メモリの少ない論理型の構造体配列に格納することにした。カーネルでは、Lookup table に格納されている判定結果を参照するだけとし、前のコードよりも判定処理を減らした。

4. 高速化の検証

GPU ボードは、本来はグラフィック処理・表示等の単純な演算を実行するために作られたものであり、表-1 に示すように倍精度演算の理論ピーク性能は単精度計算の半分程度である。そこで、本節では、まず倍精度計算と単精度計算の計算速度、および両者の計算値の差について検討を行う。次に、単精度計算を実行した場合の計算速度の性能評価を行う。

4.1 倍精度計算と単精度計算

(1) 計算値の残差比較

EFIT は式 (4)~(6) を実行して解くものであるから、桁落ちや丸め誤差が生じる。しかも、陽解法であるため、誤差は時間ステップ数の増加と共に蓄積されるものである。誤差を小さくするには倍精度計算が望ましいが、ここでは倍精度計算と単精度計算でどの程度、計算値に差(以後、残差)が生じるのか検証を行った。図-4 に示すように、 $c_T=3200\text{m/s}$ 、 $\rho=7850\text{kg/m}^3$ の弾性体に直径 10mm の円形の空洞がある。空洞の 30mm 上方に

図-4 数値実験のための波動伝搬領域と計算結果の出力点。

置かれた長さ 20mm のソース領域から SH 波が発振されるものとする。SH 波はせん断応力 τ_{32} を変動させることで発振しており、中心周波数 1MHz の 1.5 波の波形で励起した。セル長は、前論文⁵⁾の提案式に従って $\Delta x=0.05\text{mm}$ とした。時間ステップ幅は $\Delta t=10\text{ns}$ とし、1800 ステップまで計算した。図-4 中の出力点における面外変位 $u_3(=\int v_3 dt)$ の値を、時系列にプロットしたものを図-5 に示す。図-5 の上部がイメージベース EFIT による倍精度計算と単精度計算の結果を、下部が倍精

図-5 空洞による散乱波の計算結果(上:EFIT(GPU 計算), 下: BEM(倍精度 CPU 計算))。

度計算の境界要素法 (BEM) で同じ形状 (ただし, 円形境界は滑らかに近似) に対して計算¹⁵⁾を行った場合に得られた波形を参照解として示す. 図-5 に示す第 1 波目は送信波が通過したときの波形であり, 第 2 波目は空洞によって散乱された波動がソース領域に向かって伝搬するときの波形である. この結果, イメージベース EFIT による計算は BEM による解と良好に一致しており, また, 倍精度と単精度 GPU 計算の値に違いはほとんど見られないことがわかる.

ここで, 時間ステップが増加した場合の残差を詳しく検証するために, 200,000 ステップまで計算を続けた. このときの倍精度と単精度計算の残差を図-6 にプロットした. 横軸は伝搬時間を, 縦軸は倍精度と単精度の残差 (得られた u_3 のうち最大値 $|u_3(\text{double})|_{\max}$ で無次元化したもの)

$$\text{Residual} = \frac{u_3(\text{double}) - u_3(\text{single})}{|u_3(\text{double})|_{\max}}$$

をプロットしている. 200,000 ステップの計算 (3.2mm の波長を持つ波動が 6.4m 遠方に伝搬する時間に相当) において, 倍精度と単精度の残差は最大でも $\pm 2.6\text{E-}04$ 程度であり, 単精度計算でも工学的には十分な精度であることがわかる.

(2) 計算速度の比較

ここでは, 単精度と倍精度計算の速度比較に関する検討結果を示す. セル数を $1,000^2$, $2,000^2$, $\dots$, $8,000^2$ と変化させた場合の計算実効速度を計測した. イメージベース EFIT では, 式 (4), (5), (6) を 1 ステップ実行する際に, 1 セルあたり 28 回の浮動小数点演算を行っている. 空洞境界やソース点があるセルでは演算回数が変動するが, 平均として 1 ステップあたり 28 回の浮動小数点演算を行うとして, FLOPS を計算した. なお, EFIT はステンシル計算となりメモリバンド幅が律速となる. 式 (4)~(6) をみると, 倍精度 1 データ (8Byte) のロード/ストアあたり約 1 回の演算が必要であるとすれば, Tesla C2075 の場合, $144[\text{GByte/s}] / 8 [\text{Byte/FLOP}] = 18 \text{ GFLOPS}$ が倍精度計算の理論値と考えることができる.

図-6 200,000 ステップ (2,000 μs) まで計算したときの倍精度と単精度計算の残差.

図-7 に, GPU 計算 (倍精度, 単精度) と CPU 計算 (倍精度, 1 コア) の検証結果を示す. ここでは, CPU として Xeon E5645 (クロック周波数 2.4GHz, 主記憶メモリバンド幅 32GB/s) を用いた. 図-7 より, セル数が増えても CPU 計算は 0.38GFLOPS とほぼ一定である (並列化していないため) が, GPU による単精度および倍精度計算では, セル数の増加と共に FLOPS が上昇している. Tesla C2075 ではデバイスメモリの制限上, 倍精度計算が $8,000^2$ までしか実行できなかった. 単精度においてはセル数が $8,000^2$ のとき, CPU に比べて 42.5 倍の計算効率を得られた. 以上の結果は, 理論値の半分程度のパフォーマンスしか得られていないが, この原因として, カーネルには分岐処理を含むため実際には計算時間が余計かかるためであると考えられる. また, 倍精度演算は単精度演算の 69% 程度の FLOPS であり, 表-1 の理想値の約 50% を上回っているが, このことから, EFIT がメモリ律速型の計算手法であることがわかる.

以上の検討により, イメージベース EFIT は, 単精度計算でも工学的に十分な精度を有しており, 速度もより高速であることがわかったので, 以下の検討では, すべて単精度で GPU 計算を行うこととする.

4.2 Lookup table の効果

ここでは, Lookup table を導入した効果を示す. セル数を $8,000^2$ とし, 解析領域の中心に SH 波を励起させ, 100 ステップまで計算を行った場合の計算時間をプロットした結果を図-8 に示す. ここでは, 均質材料, 非均質材料 (空隙 3%, 介在物 40% を含む) の 2 つを計算モデルとし, さらに PML の有無によって計算時間を比較した. Lookup table が無い場合に比べて, 全体的に約 1.4 倍の高速化率となった. PML があっても実行速度はほとんど低下することなく, 計算が可能であることが示された. Lookup table なしの場合は, コード中で領域, 空洞境界, 界面等の分岐判定を次々と実行しながらブ

図-7 セル数を増やした場合の FLOPS の変化. 単精度および倍精度 GPU 計算, 倍精度 CPU 計算を比較したもの.

図-8 Lookup table の効果 (セル数 $8,000^2$, 100 ステップの更新に要した計算時間と高速化率)。

ログラムを進めていた。一方, Lookup table 有りの場合は予め判定処理を行っているために分岐が少なく, 均質材料・非均質材料に関係なく高速化できることが実証された。この導入は GPU 計算に限らず CPU 計算でも有効であると思われる。

4.3 スレッド数の検討

本研究では 2 次元スレッド分割を用いており, 各方向に設定したスレッド数によって計算速度に大きな違いが生じるため, ここではどのようにスレッド数を割り振れば効率的に計算できるかを調べた。まず, x_1 方向と x_2 方向それぞれ 1 方向のみスレッド数を変化させた場合の計算速度について検証を行った。計算領域は $4,000^2$ のセル集合とし, 領域中心にソース点を設定した。また領域周辺は完全反射境界とした。図-9 の赤丸が $\text{blockDim}\%x$ を変化させた場合 (すなわち, x_1 方向のスレッド数を変化させた場合), 青い四角が $\text{blockDim}\%y$

図-9 スレッド数を最大 512 とした場合のスレッド数の違いによる実効性能 (x_1 方向を 1 に固定したとき (青印), x_2 方向のスレッド数を 1 に固定したとき (赤印))。

図-10 x_1 方向と x_2 方向のスレッド数を変えたときの実効性能 (ただしスレッド数の合計 (= $\text{blockDim}\%x \times \text{blockDim}\%y$) は 128 に固定)。

を変化させた場合 (すなわち, x_2 方向のスレッド数を変化させた場合) の実効性能を表している。この結果より, x_2 方向よりも x_1 方向にスレッド数を増やした方が, FLOPS の値が高いことがわかった。この理由は, Fortran 配列のメモリへの格納方法が挙げられる。図-2 に示すように, 2 次元配列は x_1 方向に増える配列がシーケンスになるような順でメモリ格納されている。よって, ブロック内で $\text{blockDim}\%x$ が増加すれば, メモリ上連続した配列が増えるため, コアレスアクセス⁹⁾が達成されることによって効率がよいからであると考えられる。ただし, $\text{blockDim}\%x$ を増やしすぎると FLOPS が徐々に低下するため, ここでは $\text{blockDim}\%x=128$ のとき最も実効性能が良いようである。

さて, CUDA Fortran ではスレッド数は x_1 方向と x_2 方向のいずれも設定することが可能である。次に, 上記の検討を考慮して, $(\text{blockDim}\%x) \times (\text{blockDim}\%y) = 128$ になるように, $\text{blockDim}\%x$ と $\text{blockDim}\%y$ の組み合わせを変えながら FLOPS を計算してみた。図-10 に示すように, $\text{blockDim}\%x$ が 32 程度以上あれば, 一定の実効性能が出現していることがわかる。以上の結果より, コアレスアクセスが達成される程度に連続した配列と, ワープ単位で処理可能であるスレッド数を設定することで, 計算がより効率化できることを示した。なお, 次節で示すマルチ GPU 計算の性能検証では, $\text{blockDim}\%x=128$, $\text{blockDim}\%y=1$ として計算を行う。

4.4 マルチ GPU 計算の実効性能

Tesla C2075 は 6GB のデバイスメモリを有しているが, それ以上のメモリを利用するような解析では, 複数のデバイスを用いる必要がある。ここでは, セル数を図-7 の検証時よりも, 徐々に大きくした場合の実効性能について検証する。 x_2 方向のセル数を 8,000 に固定し, x_1 方向のセル数を 8,000, 12,000, ..., 60,000 まで増加させたときの FLOPS を計算した。複数のデバイ

スを用いるときには、前節で述べたように MPI 通信を行うが、ここでは x_1 方向にのみセル数を増やしているため、袖通信は x_1 方向のみである。本研究では 1 つのノードに 3 つのデバイスが挿入されており、3GPU まではノード内で並列に実行できる。4GPU 以上はノードを跨いで Ethernet を介したデータ転送となる。

検証の前に、参考までに GPU 間 (デバイス間) の転送速度を測った。50MB のデータを MPI 通信によって、ホストを介してデバイス間で通信したときのメモリバンド幅は、ノード内のデバイス間で 618.7MB/s、ノードを跨いだ場合は 10.8MB/s であった。また、袖領域の要素数は 8,000 個であるから、これに相当するサイズを転送したときのレイテンシはノード内では $905\mu\text{s}$ 、ノードを跨いだ場合 $2859\mu\text{s}$ であり、約 3 倍の時間を要することが計測された。汎用のパーソナルワークステーションに GPU ボードを複数挿しただけであるので通信に工夫を凝らしたわけではないが、以下では、この 2 つのノードを用いてマルチ GPU 計算の性能を検証した。

結果を図-11 に示す。2GPU を使用した時は、1GPU で計算するよりも大幅に実効性能が低下している。これはデバイス間の転送速度が、デバイスの内部でのデータ転送速度よりも大幅に遅いことが原因である。しかし、デバイスを増やすことにより、デバイス間のデータ転送による遅延を補う形で実効性能が向上している。デバイスが 1 つ増えるごとに 5GFLOPS 程度向上するが、3GPU 計算から 4GPU 計算になったときは 2.5GFLOPS 程度の向上である。ノードを跨ぐ場合のレイテンシは、ノード内のそれよりも約 3 倍程度増加するため、3GPU 計算から 4GPU 計算に移行するときの実効性能の向上率が小さい。5GPU と 6GPU 計算では、それぞれ 5GFLOPS ずつ増加している。以上の検討により、デバイスを 1 つ増やせば実効性能は 5GFLOPS 程度向上することがわかった。ただし、ノードを跨ぐ場合には、これより

図-11 最大 6 つの GPU ボードを使用したときの実効性能。

も劣ることがわかった。

5. 超音波計測実験による解析精度の検証

近年、シミュレーションの妥当性を示すために実験結果との比較が要求されている。これまで EFIT と実験結果との比較に関する研究はあまり報告されておらず、イメージベース EFIT の妥当性を検証しておくことは重要である。ここでは、既往の超音波計測実験を模擬するように数値モデルを作成し、シミュレーションを行った。Nurmalia らの論文¹⁶⁾では、減肉部を含む薄板 (材質: アルミニウム, $c_T=3100\text{m/s}$, $\rho=2700\text{kg/m}^3$) 中の SH 波の伝搬を可視化しているため、EFIT との直接比較が可能である。図-12 に示すように、実験では高さ 3mm、長さ 1,000mm の薄板に対して、EMAT (電磁超音波センサ) から 0.727MHz で SH 波の 1 次モード** (以下では、SH₁ 波と呼ぶ) を発生させ、板の上側表面に接触させたピンデューサで位置を変えながら受信する。減肉部は切り欠きであり、切り欠きの形状は直角 (供試体 A) とスロープ状 (供試体 B) の 2 種類である。この実験を模擬するように、イメージベース処理によって数値モデルを作成した。ここで、波動伝搬を比較する領域は長さが 200mm 程度であるので、イメージベース EFIT の数値モデルは高さ 3mm、長さ 500mm のものを作成した。 $\Delta x=0.025\text{mm}$ としたので、セル数は $120 \times 20,000=240$ 万個であり、GPU ボード 1 つで解析可能であった。また、 $\Delta t=4\text{ns}$ とし、総ステップ数は 40,000 とした。入射波形は、10 波のバースト波 (中心周波数 0.727MHz) とした。このときの、実験結果と解析結果の比較を図-13 に示す。

図-13 では、ピンデューサで受信された面外振動の振幅を縦軸に、ピンデューサの移動距離を横軸にプロットしている。SH 波の減肉部におけるモード変換に関する詳細な考察は Nurmalia らの論文¹⁶⁾を参照して頂くこととし、実験と解析の比較という観点で考察を述べ

図-12 EMAT センサから発生する SH 波 (SH₁ モード) が減肉部を含むアルミニウム板を伝搬する。このときピンデューサによって表面変位を計測する実験の概要図。

** 薄板を伝搬する波動はラム波となるため、分散性を有する。ここでは詳細に記述しないが、SH₁ 波は長軸方向に対して非対称なモードである。

図-13 ピンデューサを平行移動することで得られる表面変位をプロットした結果。上図：供試体 A(減肉部の切り欠きが直角のもの) に対する計測実験¹⁶⁾ と数値解析の比較，下図：供試体 B(減肉部の切り欠きがスロープ状のもの) に対する計測実験と数値解析の比較。

る。SH₁ 波は分散性を有するモードであり、バルク波で見られる一定な横波音速 $c_T=3100\text{m/s}$ では伝搬しない。伝搬速度(群速度)は図-13の波群の傾きを読み取ることによって推定可能であり、解析結果は実験結果を精度良く表していることがわかる。図-13の上図に示す供試体 A の場合、減肉部で SH₁ 波の散乱が生じ、一部は反射して逆方向に伝搬し、残りは 0 次モードの SH₀ 波として減肉部を伝搬しており、実験結果と解析結果は良好な一致を示している。図-13の下部に示す供試体 B の場合、減肉部でほぼ反射しており、ほとんどのエネルギーが逆方向に伝搬しており、解析結果は実験結果を良好に模擬できている。

以上の検証により、イメージベース EFIT は実験を模擬できており、解析精度について十分な信頼性を有する結果が示された。なお、この数値モデルの計算時間(単精度計算、1GPU 使用)は約 260s であった。

6. 結論

非破壊検査の波動伝搬シミュレーションのために、動弾性有限積分法(EFIT)とイメージベース処理を組み合わせたイメージベース EFIT が提案されている。本研究では、CUDA Fortran を用いてイメージベース EFIT の高速化を試み、超音波計測実験によって本手法の妥当性を検証した。

高速化に関する結果を以下に示す。

- 単精度計算と倍精度計算の残差について検証を行った結果、非破壊検査のモデル化には十分なほど長い伝搬時間を計算する場合でも、単精度と倍精度の計算値の差は小さく、工学的には十分許容できる範囲であった。
- CPU 計算(倍精度、1 コア)に比べて GPU 計算(単精度)は最大で 42.5 倍の高速化が得られた。しかし、EFIT はメモリ律速型の計算手法であり、Tesla C2075 を用いる場合、倍精度計算では単精度計算の 69% の FLOPS が得られた。
- GPU 計算ではプログラムコードにネスト構造が多用されると性能が落ちる。そこで、非均質材料や複雑な反射境界形状を含むモデルに対して性能を落とさずに計算するために、Lookup table の導入を行った。Lookup table の効果は大きく、導入しない場合に比べて 1.4 倍の高速化が得られた。
- 並列粒度に関する検討では、Fortran 配列のメモリ上の並びを考慮して、 x_1 方向のスレッド数を増やす方が、効果が得られることがわかった。
- 本研究では、デバイス間の通信とノード間の通信レイテンシが大きく、マルチ GPU の効率がリニアに得られなかった。ただ、デバイスを増やせば、通信レイテンシを補う形で実効性能が向上することが示された。

イメージベース EFIT の解析精度を検証するために、薄板中を伝搬する SH 波の超音波計測実験との比較を行った。薄板中では、超音波は分散性を有するラム波となるが、伝搬モード、群速度、減肉部に対する散乱等、実験結果を良好に模擬できていることがわかった。以上の結果から、PC レベルでの高速化が実現でき、さらに解析精度についても十分な信頼性を有する結果が示された。本ツールの実用化に向けて、さらにブラッシュアップを図る予定である。

今後の課題は、3 次元イメージベース EFIT に GPU 計算を導入することである。2 次元の場合 N^2 で要素数が増えるのに対し、3 次元では N^3 に比例して要素数が増加する。また、アロケートする物理節点の数も増えるため、デバイスメモリが不足するのは明らかである。従って、マルチ GPU の導入が不可欠であり、ノード間通信を高速化するための工夫を検討していきたい。ま

た，現状の使い方では，CUDA の 3 次元階層を導入したときにシェアードメモリが不足することが予想される．既に Okamoto ら¹⁰⁾ が有限差分法で 1 つの対処法を示しているが，今後は，イメージベース EFIT に適した効率的な方法について考えていきたい．

謝辞： 本研究は，財団法人 JKA 「平成 23 年度自転車等機械工業振興事業に関する補助金」を受けて行われました．

また，大阪大学 Nurmalia 氏から超音波計測実験の結果を頂きました．ここに，記して謝意を表します．

参考文献

- 1) Fellinger, P., Marklein, R., Langenberg, K.J. and Klaholz, S.: Numerical modeling of elastic wave propagation and scattering with EFIT-elastodynamic finite integration technique, *Wave Motion*, Vol.21, pp.47-66, 1995.
- 2) Schubert, F.: Numerical time-domain modeling of linear and nonlinear ultrasonic wave propagation using finite integration techniques-theory and applications, *Ultrasonics*, Vol.42, pp.221-229, 2004.
- 3) Terada, K., Miura, T. and Kikuchi, N.: Digital image-based modeling applied to the homogenization analysis of composite materials, *Computational Mechanics*, Vol.20, No.4, pp.331-346, 1997.
- 4) Nakahata, K., Tokunaga, J., Kimoto, K. and Hirose, S.: A large scale simulation of ultrasonic wave propagation in concrete using parallelized EFIT, *Journal of Solid Mechanics and Materials Engineering*, Vol.2, No.11, pp.1462-1469, 2008.
- 5) 中畑和之，徳永淳一，廣瀬壮一：イメージベース波動伝搬シミュレーションと超音波探傷法のモデル化への応用，非破壊検査，Vol.59, No.5, pp.231-238, 2010.
- 6) 檜山和男，牛島省，寺田賢二郎，岡澤重信，木村一郎，中畑和之，浅井光輝，松本純一，岩下武史，小山田耕二：マルチフィジックスおよび最適化問題に向けたハイパフォーマンス計算力学，学際大規模情報基盤共同利用・共同研究拠点 平成 23 年度共同研究 中間報告書，11-NA04, 2011.
- 7) 宮田一乗，高橋誠史，黒田篤：GPU コンピューティングの動向と将来像，芸術科学会論文誌，Vol.4, No.1, pp.13-19, 2005.
- 8) 東京工業大学 GSIC, TSUBAME 計算サービス，<<http://tsubame.gsic.titech.ac.jp>> (accessed 2012-3-4).
- 9) 青木尊之，額田彰：はじめての CUDA プログラミング，工学社，ISBN 978-4-7775-1477-9, 2009.
- 10) Okamoto, T., Takenaka, H., Nakamura, T. and Aoki, T.: Accelerating large-scale simulation of seismic wave propagation by multi-GPUs and three-dimensional domain decomposition, *Earth, Planets and Space*, Vol.62, pp.939-942, 2010.
- 11) Michéa, D. and Komatitsch, D.: Accelerating a three-dimensional finite-difference wave propagation code using GPU graphics cards, *Geophysical Journal International*, Vol.182, pp.389-402, 2010.
- 12) The Portland Group, Inc., PGI CUDA Fortran Compiler, <<http://www.pgroup.com/resources/cudafortran.htm>> (accessed 2012-3-4)
- 13) 日本学術会議 総合工学委員会・機械工学委員会合同 計算科学シミュレーションと工学設計分科会：報告-ものづくり支援のための計算力学シミュレーションの品質保証に向けて-，<<http://www.scj.go.jp/ja/info/kohyo/pdf/kohyo-21-h123-2.pdf>> (accessed 2012-3-4), 2011.
- 14) Fineberg, S., Wong, P., Nitzberg, B. and Kuzmaul, C.: PMPIO - A portable implementation of MPI-IO, *Proceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation*, IEEE, pp.188-195, 1996.
- 15) Niwa, Y., Kitahara, M. and Ikeda, H.: The BIE approach to transient wave propagation problems around elastic inclusions, *Theoretical and Applied Mechanics*, Vol.32, pp.183-198, 1984.
- 16) Nurmalia, Nakamura, N., Ogi, H., Hirao, M. and Nakahata, K.: Mode conversion behavior of SH guided wave in a tapered plate, *NDT & International*, Vol.45, pp.156-161, 2012.

(2012 年 3 月 8 日 受付)