Vol.2011-HPC-129 No.20
2011/3/16

三次元有限要素法アプリケーションの CUDA 向け実装と性能評価

大島聡史, 林雅江, 片桐孝洋, 中島研吾[†1]

本稿では三次元弾性静力学を対象とした有限要素法 (Finite Element Method, FEM) の GPU(CUDA) 向け実装と性能評価について述べる．高い演算性能・メモリ転送性能を持つ GPU は様々な科学技術計算アプリケーションに利用されており，FEM についても多くの研究がなされている．本稿では特に前処理付き共役勾配法 (Conjugate Gradient Method, CG 法) による疎行列ソルバーと係数行列生成部分に注目し，CUDA 向けの実装と性能評価を行った結果を報告する．

Implementation and Evaluation of 3D Finite Element Method for CUDA

Satoshi OHSHIMA, Masae HAYASHI, Takahiro KATAGIRI, Kengo NAKAJIMA[†1]

In this paper, we describe the implementation and evaluation of Finite Element Method(FEM) on GPU(CUDA). Because GPU has high calculation performance and memory transfer performance, GPU is now utilizing for several scientific applications include FEM. We show the result of implementation and performance evaluation especially about sparse matrix solver using Conjugate Gradient Method and matrix creation.

1. はじめに

高い並列演算性能とメモリ性能を持つ GPU(Graphics Processing Unit) の活用が進んでいる．GPU を用いた汎用計算は GPGPU や GPU コンピューティングと呼ばれており，特に数値シミュレーションなど科学技術計算アプリケーションへの応用についての研究が盛んに行われている．日本で初めて 1PFlops を越える性能に到達した東京工業大学の TSUBAME2.0 のように，従来は多数の CPU によって構成されていたスーパーコンピュータにも GPU が使われ始めており，GPU への期待は大きい．

GPU は並列度が高く連続メモリアクセスを主とする計算に適しているため，行列積などの計算においては高い性能を発揮しやすい．しかし，GPU に適したアプリケーションばかりが利用されているわけではない．今後さらに GPU を適材適所で活用していくためには，様々な計算を GPU 向けに実装して性能の分析を行い，実装手法やアルゴリズムの開発を進めていく必要がある．

現在我々は三次元弾性静力学問題を対象として，GPU(CUDA[1)]) を用いた有限要素法 (Finite Element Method, FEM) アプリケーションの高速化に取り組んでいる．FEM は数値シミュレーションにおいて多く用いられている解法である．FEM の実行時間の多くを占める疎行列ソルバーがベクトルや行列に関する演算を多く含むため，疎行列ソルバー，特に共役勾配法 (Conjugate Gradient 法，CG 法) の GPU 実装などについては GPGPU 初期の頃から研究開発が行われている．

我々は CPU と GPU の単純な性能比較のみならず，CPU と GPU それぞれの特性に合わせたアルゴリズム選択や，また FEM における疎行列ソルバー以外の処理についての GPU 実装についても大きな興味を持っている．そこで本稿では，

(1) CG 法における CPU 向け実装と GPU 向け実装について
(2) FEM における係数行列生成処理について

の 2 点について実装し性能評価を行った結果を報告する．

本稿の構成は以下の通りである．2 章では FEM と GPU について，本稿において注目している特徴を中心に述べる．3 章では疎行列ソルバー (CG 法) の実装と性能評価について述べる．4 章と係数行列生成の実装と性能評価について述べる．5 章はまとめの章とする．

2. 有限要素法と GPU

2.1 有限要素法

本稿の対象アプリケーションである有限要素法 (Finite Element Mehtod, FEM) は偏微分方程式の数値解法の一つであり，連続体力学が対象とする熱伝導解析，構造解析，流体解析，及びそれらの連成問題など，幅広い分野へ適用されている．FEM を用いた計算の手順

†1 東京大学 情報基盤センター
Information Technology Center, The University of Tokyo

は以下の通りである．

(1) プリプロセス処理
    1. メッシュ生成
(2) FEM 本体
    1. データ入力
    2. 行列コネクティビティ生成
    3. 係数行列生成
    4. 境界条件処理
    5. 線形方程式ソルバー (疎行列ソルバー)
(3) ポストプロセス
    1. データ処理
    2. 可視化処理

FEM では，解析領域全体を節点 (離散点) で構成される要素とよばれる微小領域に分割して扱う．支配方程式に対して要素単位で重み付き残差法を適用することで要素剛性マトリクスを計算し，それを全節点数と自由度の積の次元を持つ全体行列へ足し込むことで全体剛性行列を計算する．したがって，全体剛性行列は差分法と同様に疎である．要素剛性マトリクスの全体化には各要素を構成する節点の情報 (コネクティビティ) が用いられ，全体剛性行列生成後，境界条件処理を施すことで，最終的な係数行列が求められる．最後に疎な全体剛性行列を係数行列とする連立一次方程式を解くことで，離散方程式の変数であった速度，温度，変位等が解として求められる．

FEM 本体の計算は，係数行列の生成と線形方程式ソルバによる連立一次方程式の求解に大部分が費やされ，線形方程式ソルバ部分が約 9 割を占める．線形方程式ソルバとしては，係数行列が疎であることから反復解法が用いられ，また行列の定値対称性から共役勾配法 (Conjugate Gradient Method, CG 法) の適用が一般的であり，前処理と合わせて適用されることが多い．

2.2 有限要素法の並列化と GPU

CG 法の計算にはベクトルや行列に関する計算 (ベクトル同士の積や和および行列とベクトルの積) が用いられるため，比較的容易に並列化を行うことができる．ただし間接参照が多いためメモリに対する負荷が大きく，並列化による性能向上を得るためには高いメモリ性能が要求される．また前処理については，ブロック Jacobi 法など単純な手法であれば容易に並列化できるものの，係数行列が悪条件な問題となった場合にも効果の高い不完全 Cholesky 分解等の手法を用いる場合には，グローバルな演算や Fill-in について考慮することが必要となるため，並列高速化が困難となる．

一方で GPU はベクトルや行列に関する演算に適しており，CG 法で用いられるベクトルや行列に関する計算については GPU に適した典型的な計算として GPGPU 研究の初期から高速な実装が行われてきた．GPU を用いた CG 法の実装も GPGPU 研究の初期から多く行われてきたテーマである．CG 法は FEM に限らず様々なアプリケーションで用いることができるため，GPU を用いた CG 法の研究は現在も様々なアプリケーションについて進められている[2)3)]．

GPU(CUDA) を用いて CG 法を解く機能を持つライブラリも開発・公開が行われている．例えば，行列計算やベクトル計算のためのクラスライブラリ cusp[4)] には単純な前処理にも対応した CG 法が実装されている．

本稿では Fermi アーキテクチャの GPU である Tesla C2050 を対象として実装と性能評価を行う．Fermi アーキテクチャでは従来の GPU と比べてキャッシュや倍精度浮動小数点演算性能が強化されており，ECC にも対応している．本稿の性能測定では CPU と GPU 両方における全ての浮動小数点演算を倍精度で行っている．ただし倍精度演算を用いても並列化による演算順序の変更による計算誤差の発生を防ぐのは困難である．そのため，本稿の各実験において CG 法の反復回数が数回変わるケースがあったことを付記しておく．また，ECC については ON の状態で測定を行っている．

3. 疎行列ソルバーの実装と性能評価

2.2 節で示したように，GPU を用いた疎行列ソルバー，特に CG 法については既にいくつかの研究において実装や性能評価が行われている．本研究では GeoFEM プロジェクト[5)] で開発された並列有限要素法アプリケーションを元に整備した性能評価のためのベンチマークプログラム群の 1 つである三次元弾性静解析プログラム[6)] を CUDA 向けに書き換えて用いている．このプログラムは CPU 上での性能とメモリ効率を上げるために，係数行列に対して 3x3 のブロック化を行いさらに対角ブロック・上三角ブロック・下三角ブロックに分けて保持している．CPU において性能改善に寄与しているブロック化が GPU ではどのように影響するかを調査することは本性能評価の主題である．行列格納形式は CRS(Compressed Row Storage) 形式である．

本章および次章において用いる実験環境は**表 1** の通りである．

Vol.2011-HPC-129 No.20
2011/3/16

表 1　実験環境

	環境 1: T2K 東大版 日立 HA8000 (1 ノード)	環境 2: 汎用 PC
CPU	Opteron 8356 2.3GHz (4core × 4CPU)	Xeon W3520 2.67GHz (4core × 1CPU)
メインメモリ	32GB	12GB
GPU	-	Tesla C2050 448 core
GPU メモリ	-	3GB GDDR5
コンパイラ	Intel コンパイラ Version 11.0	gcc version 4.4.0
CUDA 開発/実行環境	-	CUDA toolkit 3.2 GPU ドライバ 260.19.26

3.1　CPU を用いた実装

はじめに，各実験環境の CPU のみを用いた場合の FEM アプリケーション全体および CG 法の実行時間とその内訳を確認する．以下，環境 1(T2K) では C/C++を用いたプログラムを Intel コンパイラでコンパイルしたもの，環境 2(汎用 PC) では C/C++を用いたプログラムを gcc(g++) でコンパイルしたものを用いている．問題サイズ (弾性体の分割数) は xyz 各方向に 80，節点数と要素数はそれぞれ 531441 節点および 512000 要素である．計算打ち切り誤差は 1.0e-8 に設定した．

測定結果を**図 1** に示す．FEM 全体の実行時間は環境 1(T2K) で 360.24 秒，環境 2(汎用 PC) で 107.76 秒かかっており，それぞれ 95.57%および 90.79%が CG 法の実行時間である．さらに CG 法の実行の時間のうち，環境 1(T2K) では 89.6%，環境 2(汎用 PC) では 88.4%を疎行列ベクトル積が占めており，疎行列ベクトル積が CG 法および FEM の性能を左右していることがわかる．

2.1 節で述べたように，CG 法内部の計算はベクトルや行列に関する計算が多いため，OpenMP による並列化に適している．OpenMP を用いて並列化した CG 法の実行時間は**図 2**(a) の通りである．いずれの環境においてもリニアな性能向上は得られていないものの，環境 1(T2K)・環境 2(汎用 PC) ともに並列化による速度向上が得られていることが確認できる．実行時間は環境 1(T2K)16 スレッドでは 151.47 秒，環境 2(汎用 PC)4 スレッドでは 52.31 秒となり，逐次実行と比べてそれぞれ 55.34%および 46.40%の実行時間が削減された．

環境による性能の差はメモリ性能の差に寄るところが大きい．CG 法は実行時間の多くを疎行列ベクトル積が占めており，疎行列ベクトル積の性能にはメモリアクセス性能の影響が大きいため，メモリ性能の良い環境 2(汎用 PC) の方が高い性能が得られたと考えられる．

一方，性能向上のために行われていたブロック化を解除した場合の実行時間は図 2(b) の

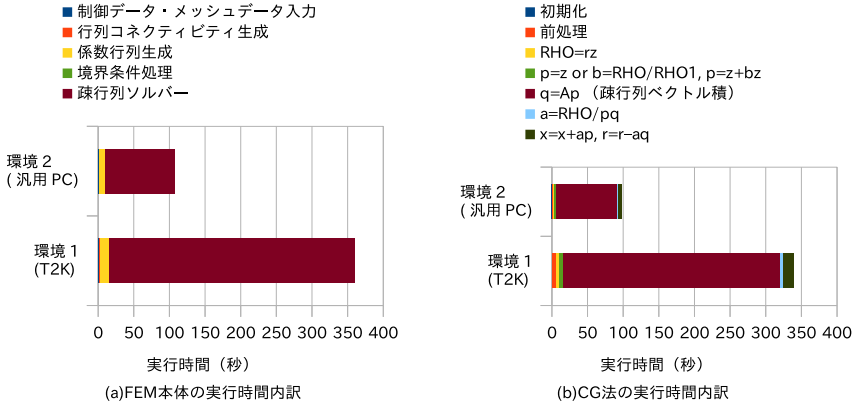

図 1　CPU を用いた FEM および CG 法の実行時間と内訳

通りである．ブロック化の有無による性能を比較すると，ブロック化を解除したことによりブロック化した状態と比べて環境 1(T2K) 16 スレッドで 37.7%，環境 2(汎用 PC) 4 スレッドで 30.8%の実行時間が増加している．

ブロック化により速度向上が行える理由は以下の通りである．ブロック化を行わずに疎行列ベクトル積を行った場合，ベクトルに対するメモリアクセスの連続性が行列内の非零要素の配置に依存し，ランダムなメモリアクセスが多数行われる．一方で行列をブロック化して保持した場合，各ブロック単位の疎行列ベクトル積におけるベクトルへのメモリアクセスがブロックサイズ分の行に制限される (局所化される) ため，ベクトルに対するメモリアクセスにおいてキャッシュの効果により速度が向上する．

3.2　GPU を用いた実装

前節で用いたプログラムを CUDA 向けに実装した．CG 法の高速化にはベクトル同士の計算や疎行列ベクトル積の高速な実装が必要であるが，これらの GPU による実装は既知である．ベクトル同士の積や和については，GPU 上で多数のスレッドを動かし，それらのスレッドがコアレスなメモリアクセス (同一スレッドブロック上の複数スレッドによる連続した，もしくは近接したメモリアドレスへのメモリアクセス) を行えば良好な性能が得られる．疎行列ベクトル積については，行単位で WARP を割り当て，WARP 単位で同期の不要なリダクション演算を行えば良好な性能が得られる．

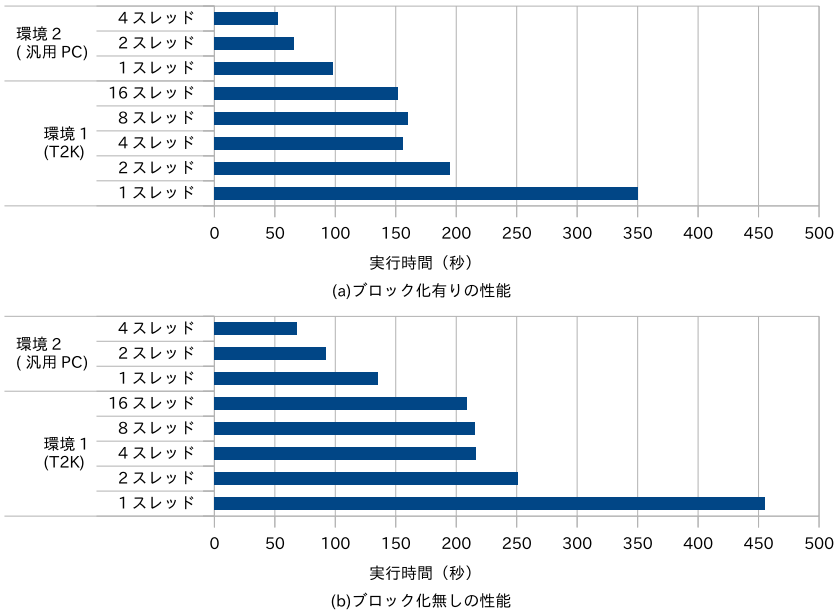

図 2 CPU を用いた CG 法の実行時間

そこで，既知の実装を用いてナイーブな実装を行い，実行時間を測定した．また，ブロック化を解除した場合についても実装して実行時間を測定した．測定の結果を**図 3** に示す．CPU-GPU 間のデータ転送時間については，主なデータを GPU に転送した状態で測定を開始しているため CPU から GPU へのデータ転送は実行時間に含まれておらず，GPU から CPU への書き戻しについても 1 反復毎の残差確認に必要な 1 変数分以外のデータ転送は実行時間に含まれていない．

ブロック化した状態での実行時間は 45.96 秒であり，環境 1(T2K) と比べて 69.66%，環境 2(汎用 PC) と比べて 12.13%短い時間で処理を行うことができた．一方でブロック化を解除した場合については，CPU と異なりブロック化したままの状態とほぼ同一の 45.98 秒という性能が得られた．いずれの実装についてもさらに最適化を行うことで性能の差が生じる可能性はあるが，現時点ではブロック化の有無による性能の差はないため，むしろ前処理との親和性などとあわせてアルゴリズム選択等を行うべきであると考えられる．

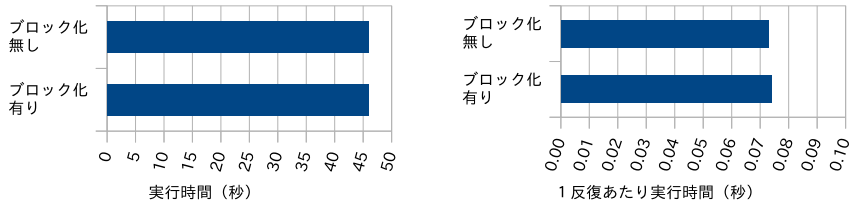

図 3 GPU を用いた CG 法の性能

一方，GPU に要求されるメモリ容量については，現在の実装ではブロック化した状態では約 1.0GByte を使用しているのに対してブロック化を解除した状態では約 1.5GByte を使用しており，対象問題によってはメモリ使用量による制限についても考慮する必要がある．

既存の CUDA 向け数値計算ライブラリ cusp にも共役勾配法の実装があるものの，前処理の内容などが本稿の実装とは異なるため性能の比較ができていない．既存の実装との性能比較も今後行っていく予定である．

4. 係数行列生成の実装と性能評価

3 章で示したように，我々が対象としている FEM アプリケーションの実行時間は疎行列ソルバー (CG 法) が多くを占めている．一方で CG 法の並列高速化を行った場合，CG 法が依然として FEM 本体の中で最も実行時間が長いのは変わらないものの，CG 法の次に実行時間が長い係数行列生成の実行時間全体に占める割合が顕在化する．そこで本章では係数行列生成の GPU 向け実装について述べる．

4.1 CPU を用いた実装

まずは係数行列生成処理の手順と CPU 向けの並列化手法について確認する．係数行列生成処理の手順 (プログラムの概形) を**図 4** に示す．係数行列生成処理は大きく 2 つの多重ループにわかれているが，前半は計算量も実行時間も小さいため，後半のループのみを測定や GPU 化の対象とする．

係数行列生成の後半ループ部は要素や接点に対して次々に処理を行うために多重ループ構造となっている．特に，最外ループである全ての要素に対して処理を行うためのループ (icel ループ) のループ回数が多いため，これを並列化することができれば性能向上が期待できる．しかし，要素毎に計算した結果を全体行列に足し込む処理が要素毎に完全には独立し

```
do k = 1,2
  do j = 1,2
    do i = 1,2
      ガウス積分点(8点)における形状関数とその自然座標系における微分の算出
    enddo
  enddo
enddo

do icel = 1, 要素数
  8節点の座標から, ガウス積分点における形状関数の全体座標系における微分とヤコビアンを算出
  do ie = 1,8
    do je= 1,8
      全体接点番号ip, jpを元に係数行列における位置を算出
      do k = 1,2
        do j = 1,2
          do i = 1,2
            要素積分, 要素行列成分計算, 全体行列への足し込み
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```

図 4 係数行列生成の手順 (プログラムの概形)

ていないため，単純に最外ループを並列化することはできない．

これを解決する方法として，カラーリング (色の塗り分け) が知られている．すなわち，事前に全要素をチェックし，同時に処理して問題のない要素群を同じ色を持つ要素としてマークしておき，後に各色毎に並列に要素の計算を行う手法である．代表的なカラーリング手法としては MC 法 (MultiColor 法) などが挙げられる．本稿でも MC 法を用いる．

問題設定としては，3 章では GPU メモリ容量の都合で xyz 各方向に 80 分割した問題を用いたが，係数行列生成についてはより大きな問題設定である xyz 各方向に 100 分割，節点数と要素数がそれぞれ 1030301 節点および 1000000 要素の問題を用いることにする．

CPU を用いた場合の MC 法による係数行列生成の実行時間を**図 5** に示す．今回対象としている問題は最低 8 色での塗り分けが可能であり，図 5 も 8 色で塗り分けた際の性能を表している．環境 1(T2K) の 16 スレッドでは 1.38 秒，環境 2(汎用 PC)4 スレッドでは 4.73 秒の時間を要した．環境 1(T2K) の 8 スレッドから 16 スレッドにかけてあまり性能向上が得られていないものの，並列化により実行時間が大きく削減されていることが確認できる．

4.2 GPU を用いた実装

GPU を用いた実装についても，CPU と同様に MC 法の適用を行った．GPU への適用においては段階的に並列化を行ったため，並列化の過程と性能を示す．なお，実行時間の測定には CPU-GPU 間のデータ転送を含めていない．

はじめに，MC 法にて色分けされたプログラムを 1 スレッドブロック・1 スレッドで逐

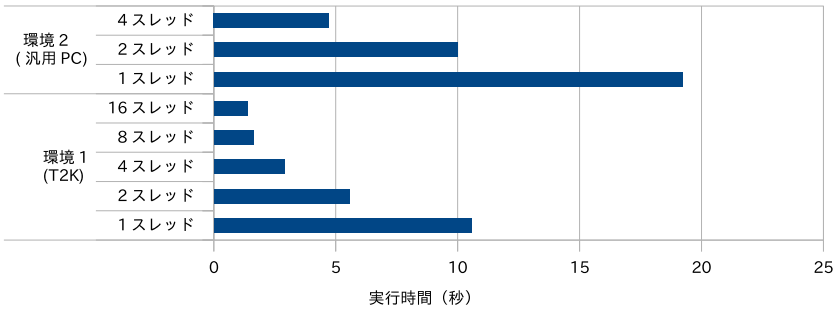

図 5 CPU を用いた係数行列生成処理の実行時間

次実行し実行時間を測定したところ，1204.19 秒を要した．つづいて，単純に最外ループを GPU 上のスレッドブロックへと割り当て，スレッドブロックあたりのスレッド数を 1 にして実行時間を測定したところ，41.02 秒に短縮された．スレッドブロックレベルの単純な並列化だけでも性能は向上したが，この時点では CPU と比べて非常に低速である．この際のスレッドブロック数については Tesla C2050 に搭載されたマルチプロセッサ数 14 の 4 倍の数である 56 の場合に良い性能が得られた．以下の測定についても 56 を採用している．

つづいて GPU 上のスレッドを用いた並列化に取り組んだ．MC 法における最内の 2x2x2 ループは，リダクション演算が必要ではあるが並列に実行可能であるため，このループを完全に展開して 8 スレッドで計算することにした．この実装は SharedMemory によってスレッド間のデータ共有を行うにも都合が良く，実行時間を 9.28 秒まで短縮することができた．またこの際，実際には 8 スレッドしか使わないが GPU のスケジューリングサイズである WARP サイズに合わせて 32 スレッドで実行 (9 番目以降のスレッドは GPU カーネルの冒頭で if 文を用いて排除) したところ，わずかに実行時間が短くなり，9.07 秒となった．

GPU がさらに高い性能を発揮するにはより高い並列度が望ましい．そこで，1 つのスレッドブロックが一度により多くのループを計算できるようにした．具体的には生成するスレッド数を 16 に増やし，9 番目から 16 番目の 8 スレッドには 1 番目から 8 番目のスレッドの次の要素を計算を計算させた．これにより，実行時間は 5.28 秒に短縮された．同様にしてスレッド数を 32，48 と増やせば並列度を向上させることが可能であるが，最も実行時間が短くなったのは 4 並列 (32 スレッド) にした場合で実行時間は 2.57 秒に短縮された．

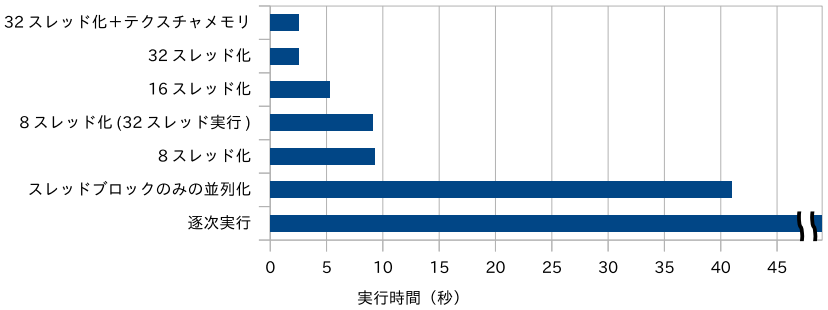

図 6 GPU を用いた係数行列生成処理の実行時間

これらの利用するスレッド数を増やす実装は不連続なメモリアクセスの増加や WARP 内での分岐回数の増加を引き起こすため性能が低下する可能性があったが，今回の問題では性能低下よりも並列度向上による性能向上の効果が大きかった．

また，メモリへのランダムアクセス時の性能を改善する方法として GPU からは読み取り専用として扱えるデータをテクスチャメモリに配置する方法が挙げられる．今回読み取り専用であるデータの一部をテクスチャメモリに配置してみたところ，極めてわずかながら性能が向上し，実行時間は 2.53 秒に短縮された．

以上の実装により，実行時間は 2.53 秒まで短縮された．**図 6** に各最適化手法の適用と実行時間の変化を示す．現在の実装では環境 2(汎用 PC) の性能には勝っているものの，環境 1(T2K) の性能には追いついていない．さらに性能を向上させる方法としては，メモリの連続性や分岐方向の同一性をより高めるために要素毎の計算を途中で分断して再構築することなどが考えられる．また，CPU-GPU 間のデータ転送に 0.8 秒程度要するため，係数行列生成の前後の処理までを考慮してこれを隠蔽できるような実装の工夫も重要である．

5. おわりに

本稿では三次元弾性静力学を対象として GPU(CUDA) を用いた有限要素法の実装を行った．特に疎行列ソルバーと係数行列生成に注目して実装と性能評価を行った．

疎行列ソルバーとしては CG 法を用いた．531441 節点 512000 要素の問題において GPU を用いたナイーブな実装で T2K 1 ノード 16 スレッドに比べて 69.66%，汎用 PC(Xeon W3520) に比べて 12.13%短い実行時間で処理を行うことができた．

係数行列生成については MC 法の GPU 化を行った．1030301 節点 1000000 要素の問題において T2K 1 ノード 4 スレッドの性能を上回ったが，16 スレッドの性能には現状では追いついていない．

GPU を用いることで疎行列ソルバー，係数行列生成それぞれにおいて一定の成果は得られたが，以下に示すように性能改善やさらなる性能評価を行う余地は大きい．

疎行列ソルバーについては，既に他の研究でも行われているように行列格納形式を CRS 以外の形式に変更するとより高い性能が得られる可能性がある．cusp など既存の実装と性能比較をする必要もある．係数行列生成については，要素毎の計算を分断して再構築することや，MC 法以外のカラーリングアルゴリズムの適用，CPU では有効であるリオーダリングについての性能評価などが挙げられる．また FEM アプリケーション全体としては，1GPU のメモリに収まらない大きなサイズの問題を解くために複数 GPU での実装を行うこと，CPU と GPU を併用してさらに高い性能を得ることを検討している．今後はこれらの実装と評価を進めていく予定である．

謝辞 本研究は，科学技術振興機構戦略的国際科学技術協力推進事業（共同研究型）「日本－フランス共同研究」「ポストペタスケールコンピューティングのためのフレームワークとプログラミング」の補助を受けている．

参 考 文 献

1) NVIDIA: NVIDIA GPU Computing Developer Home Page, `http://developer.nvidia.com/object/gpucomputing.html`.
2) Cevahir, A., Nukada, A. and Matsuoka, S.: An Efficient Conjugate Gradient Solver on Double Precision Multi-GPU Systems, 先進的計算基盤システムシンポジウム SACSIS2009，pp.353–360 (2009).
3) Bolz, J., Farmer, I., Grinspun, E. and Schröder, P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid, *ACM SIGGRAPH 2003*, pp.917–924 (2003).
4) cusp library: Generic Parallel Algorithms for Sparse Matrix and Graph Computations, http://code.google.com/p/cusp-library/.
5) GeoFEM: `http://geofem.tokyo.rist.or.jp/`.
6) Nakajima, K.: Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator, *ACM/IEEE Proceedings of SC2003* (2003).