Vol.2015-HPC-152 No.12
2015/12/17

有限要素法係数行列生成プロセスの メニィコア環境における最適化

中島研吾[†1†2] 成瀬 彰[†3] 大島聡史[†1†2] 塙 敏博[†1] 片桐孝洋[†1†2] 田浦健次朗[†1]

有限要素法は偏微分方程式の数値解法として広く計算科学・工学分野で使用されている．係数行列生成プロセスは連立一次方程式求解と並んで時間を要するプロセスである．本研究では，Intel Xeon Phi および NVIDIA Tesla K40 を対象としてそれぞれの特性を生かした最適化を実施した．本稿では最適化の詳細と性能評価結果について述べる．

Optimization of matrix assembly process in FEM applications on manycore architectures

Kengo Nakajima[†1†2] Akira Naruse[†3] Satoshi Ohshima[†1†2] Toshihiro Hanawa[†1]
Takahiro Katagiri[†1†2] Kenjiro Taura[†1]

Finite Element Method (FEM) is widely used for solving Partial Differential Equations (PDE) in various types of applications of computational science and engineering. Matrix assembly and sparse matrix solver are the most expensive processes in FEM procedures. In the present work, the matrix assembly process is optimized on Intel Xeon Phi and NVIDIA Tesla K40 based on features of each architecture. The paper describes details of optimization and results of performance evaluation.

1. はじめに

有限要素法に代表される偏微分方程式の数値解法において，最も計算時間を要するプロセスは大規模な疎行列を係数行列とする連立一次方程式の求解であり，その最適化に向けて様々な試みがなされてきた（例えば〔1,2〕）．有限要素法では，各要素における積分方程式から密な要素行列を計算し，これを重ね合わせることによって疎な全体係数行列を導出する．このような係数行列生成部（Matrix Assembly）は連立一次方程式求解部と比較してアプリケーションに依存する部分も多く，計算プロセスの最適化に関する研究は，これまであまり行われて来なかった．一般に，係数行列生成のコストは連立一次方程式求解よりは少ないものの，例えば非線形計算の場合には係数行列を反復のたびに計算し直す必要があり，できるだけ効率を高める工夫が必要である．

近年，係数行列生成部の重要性が注目されつつある．例えば 2015 年 3 月に開催された SIAM Conference on Computational Science and Engineering（SIAM CSE15）では Minisymposterium（Organized Poster Session）の一つとして「Scalable Finite Element Assembly」が企画され，7 件のポスター発表があった〔3〕．また，著者等も GPU，マルチコアプロセッサにおける有限要素法の行列生成プロセスの最適化に関する研究を実施している〔4,5〕．著者等は科学技術振興機構戦略的創造研究推進事業（CREST）「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」の 1 プロジェクトとして実施されている「ppOpen-HPC：自動チューニング機構を有するアプリケーション開発・実行環境」〔7,8〕において有限要素法に代表される様々な科学技術計算手法の各計算プロセスのマルチコア，メニィコアアーキテクチャ向け最適化，ライブラリ化と自動チューニング手法の適用に関する研究開発を実施している．有限要素法の係数行列生成部もその対象の一つであり，最適化と自動チューニング手法の検討が進められている．本研究及び先行研究〔5,9〕は，ppOpen-HPC における有限要素法アプリケーション開発用フレームワークである ppOpen-APPL/FVM のフィージビリティスタディとして実施したものである．

本論文では，以下，係数行列生成部の処理の概要とその最適化，計算環境の概要，計算結果とその分析について紹介する．ppOpen-HPC はメッセージパッシング（MPI）とプロセス内スレッド並列（OpenMP）を組み合わせたハイブリッド並列プログラミングモデルを基本としているが，本研究では特に各計算ノード上でのスレッド並列化に着目し，MPI プロセス数を 1 として計算を実施した．

計算機環境としてはメニィコアアーキテクチャとして Intel Xeon Phi（MIC）〔10〕と NVIDIA Tesla K40（K40）〔11〕を使用した．

2. 係数行列生成部の概要

2.1 対象アプリケーション

本研究で対象としているのは，GeoFEM プロジェクト

†1 東京大学情報基盤センター
Information Technology Center, The University of Tokyo
†2 科学技術振興機構 CREST
CREST, Japan Science and Technology Agency
†3 エヌビディア
NVIDIA Corporation

〔12,13,14〕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「GeoFEM／Cube」である．本ベンチマークは，三次元弾性静解析問題（Cube 型モデル（図 1））に関する並列前処理付き反復法による疎行列ソルバーの実行時性能（GFLOPS 値）を様々な条件下で計測するものである．要素タイプは三次元一次六面体要素（tri-linear）であり，各要素 8 つの節点を有している．本研究では各六面体要素を 6 個の三次元一次四面体要素に分割した場合の計算も実施した．プログラムは全て OpenMP ディレクティヴを含む FORTRAN90 および MPI で記述されている． GeoFEM で採用されている局所分散データ構造〔12〕を使用しており，マルチカラー法等に基づくリオーダリング手法によりマルチコアプロセッサにおいて高い性能が発揮できるように最適化されている．また，MPI，OpenMP，Hybrid（OpenMP＋MPI）の全ての環境で稼動する．

三次元弾性静解析問題では係数行列が対称正定な疎行列となることから，前処理を施した共役勾配法（Conjugate Gradient，CG）法によって連立一次方程式を解いている．

本来の GeoFEM／Cube ベンチマークでは前処理手法として Symmetric Gauss Seidel（SGS）を使用しているが，本研究では Block Diagonal Scaling 法〔12,13〕を使用しており，OpenMP 並列化した場合の前処理プロセスにおけるデータ依存性を考慮する必要がないため，節点のリオーダリングは実施していない．また，三次元弾性問題では 1 節点あたり 3 つの自由度があるため，これらを 1 つのブロックとして取り扱っている．係数行列はこのブロック型の特性を利用したブロック CRS 形式（Compressed Row Storage）によって格納されている．

本研究では，N_x=N_y=N_z=128 とした場合について検討した．したがって，節点数＝2,097,152（=128^3）であり，要素数は，六面体：2,048,383（=127^3），四面体：12,290,298（=4×127^3）である．

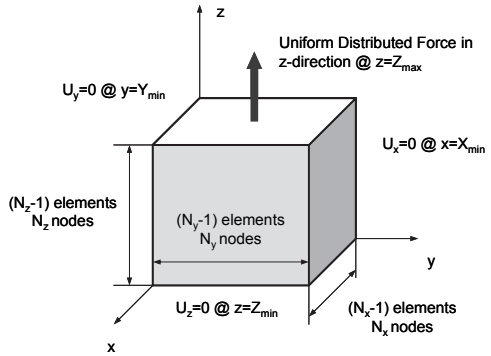

図 1　GeoFEM／Cube の解析対象（Cube モデル）

2.2 係数行列生成部

有限要素法では，要素毎に得られる積分方程式から導かれる密な要素行列を重ね合わせて疎な全体行列を生成する．図 2 に示すような二次元一次四角形要素（bi-linear, 双一次）では各要素の節点数が 4 であるので各節点の自由度数が 1 であれば，要素行列は 4×4 の密行列となる．

図 2 の 7 番の節点は周囲の 4 要素（2，3，5，6 番）からの寄与がある．したがって，係数行列生成のプロセスを OpenMP 等でスレッド並列化した場合，ある節点に複数の要素から同時にデータの書き込みが発生する場合がある．要素行列の重ね合わせを実施する際にはマルチカラーオーダリング等を使用してこのような同時書き込みの発生を回避する方法が広く使用されている〔3〕．図 3 は本研究における行列生成部の処理の概要を示すものである．三次元一次六面体要素を使用している場合は，要素あたりの節点数は 8 であり，8×8 の密な要素行列が生成される（実際は各節点に 3 つの自由度があるため，要素行列は 24×24 となる）．ループの構成としては一番外側が各要素に関するループ（do icel= 1, ICELTOT, ICELTOT：全要素数）である．その内側の二重ループ（do ie=1, 8, do je= 1, 8）は要素行列を生成するためのループであり，各要素の節点が 8 個あることに対応している．更にその内側にはガウスの積分公式に対応する三重ループ（do ipn/jpn/kpn =1, 8）がある．四面体要素の場合は，節点数は 4 であるため，図 3 に示す二重ループは（do ie=1, 4, do je= 1, 4）となる．また，要素内の積分は解析的に実施できるため，要素当りの計算量は六面体要素と比較して少ない．

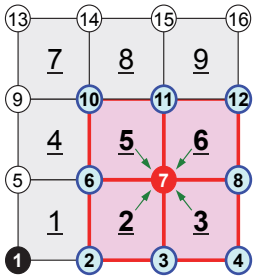

図 2　要素行列の重ね合わせによる全体行列の生成

図 3　GeoFEM／Cube における係数行列生成の処理（六面体要素）

2.3 係数行列生成部の実装手法

図 3 に示す処理をまとめると以下の 4 つとなる：

① 各積分点におけるヤコビアン，形状関数導関数計算
② 要素行列成分の全体行列（疎行列）におけるアドレス探索
③ ガウス数値積分，要素行列成分計算
④ 要素行列成分の全体行列への加算

```
!$omp parallel (…)
    do color= 1, COLORtot
!$omp do
    do icel= col_index(icol-1)+1, col_index(icol)
       icel: calculated by (col_index, ib, blk)
      <①各積分点におけるヤコビアン，形状関数導関数計算>
!$omp simd
      do ie= 1, 8; do je= 1, 8
         <②要素行列成分の全体行列（疎行列）におけるアドレス探索＋格納>
         <③ガウス数値積分，要素行列成分計算＋格納>
         <④要素行列成分の全体行列への加算>
      enddo; enddo
    enddo
    enddo
!$omp end parallel
```

図 4　GeoFEM／Cube オリジナル実装（Original）の概要（六面体要素）（COLORtot：要素色数（=8），col_index(color)：各色に含まれる要素数）

図 4 は，図 3 に示した処理内容を，上記①～④を考慮して簡略化し，OpenMP によるスレッド並列化が適用されていると仮定した場合の六面体要素の場合の実装例の概略である．COLORtot はマルチカラーオーダリングの色数であり，六面体要素の場合には 8，四面体要素を適用した場合には 33～34 程度となる．1 つの節点を共有する要素数は六面体要素の場合，ほとんどの節点で 8 となるが，四面体要素の場合は 24 となる．配列 col_index(icol)は各色に含まれる要素数である．図 4 に示すようにオリジナル実装では，これらの処理を要素毎に実施しており，特に②～④については要素行列の各成分について個別に実施している．

各ループの中で，探索，ガウス積分，全体行列への加算などの複雑な処理が繰り返し実施されるため計算効率が低くなっている可能性がある．

係数行列生成部は連立一次方程式を解く線形ソルバー部と同様に memory bound なプロセスであるが，要素単位のローカルな計算が中心であり計算性能がでやすい一方で，グローバルな係数行列計算部分でのメモリスループットが出にくい傾向があり，Fujitsu PRIMEHPC FX10〔15〕1 ノード上では，対ピーク性能比，メモリスループットはそれぞれ以下のようになっている〔5〕：

- 行列生成部：約 15%，約 20GB/sec
- 線形ソルバー：約 5%，約 60GB/sec

3. 計算機環境

本研究では以下の 2 種類の計算機環境を使用した：

- MIC：Intel Xeon Phi（Knights Corner）
- K40：NVIDIA Tesla K40

プログラムは Fortran90 で記述してあり，MIC では Intel Comliler（Ver.16）／Intel Parallel Studio XE 2016 を使用し，ノード内スレッド並列化には OpenMP を使用している．K40 では，PGI OpenACC Compiler（Ver.15.9），及び CUDA（Ver 7.5）を使用している．表 1 に計算機環境の概要を示す．本研究では，各環境において表 1 に示す 1 ソケットを用いて計算を実施した．1.でも述べたように，MPI プロセス数を 1 とし，ソケット内を OpenMP（または OpenACC，CUDA）によりスレッド並列化したプログラムを実行している．表 1 に示すように MIC では最大 240 スレッドを使用している．

表 1　各計算環境（1 ソケット）の概要

略　称	MIC	K40
名　称	Intel Xeon Phi 5110P (Knights Corner)	NVIDIA Tesla K40
動作周波数（GHz）	1.053	0.745
コア数	60	2880
使用スレッド数	240	-
メモリ種別	GDDR5	GDDR5
理論演算性能（GFLOPS）	1,011	1,430
主記憶容量（GB）	8	12
理論メモリ性能（GB/sec.）	320	288
STREAM Triad 性能（GB/sec.）〔16〕	159	218
キャッシュ構成	L1:32KB/core L2:512KB/core	L1: 16-48KB/SM L2: 1.5MB/socket
コンパイルオプション	-O3 -openmp -mmic -align array64byte	-O3 –acc –ta=tesla,c35,loadcache:L1

4. Intel Xeon Phi における最適化

4.1 最適化の概要

有限要素法における疎な係数行列は要素毎に得られる積分方程式から導出される要素行列に基づくものであり，アプリケーションへの依存性が強い．ppOpen-HPC 開発の見地からはアプリケーション開発者の負担をできるだけ軽減することが重要であり，疎行列計算に関わる上記②，④の処理に関わる機能はできるだけ ppOpen-HPC で提供することが望ましい．①もライブラリとして提供が可能な機能であるが，③は最もアプリケーションに最も依存する部分であり，アプリケーション開発者自ら記述する必要がある．

③の部分を他と切り離す場合には，要素行列用配列（1 要素あたり 4.61KByte（=24×24×8÷1,000））のため記憶容量が必要である．また，②の部分を分離する場合には要素行列各成分の疎行列におけるアドレスを記憶するための配列に要素あたり 256Byte が必要である．これらの配列はスレッド並列化を実施する場合には，各スレッドにおいて別途必要となる．したがって，これらの配列を全要素につい

て記憶することは非現実的であり，100 個以下の要素によるブロックを形成し，ブロック毎に計算を実施するのが適切である．

```
!$omp parallel (…)
    do color= 1, COLORtot
!$omp do
    do ip= 1, THREAD_num
      NBLK: calculated by (col_index, color, thread#)
    do ib= 1, NBLK
      do blk= 1, BLKSIZ
        icel: calculated by (col_index, ib, blk)
!$omp simd
        do je= 1, 8; do je= 1, 8
          <②要素行列成分の全体行列（疎行列）におけるアドレス探索＋格納>
        enddo; enddo
      enddo
      do blk= 1, BLKSIZ
        icel: calculated by (col_index, ib, blk)
        <①各積分点におけるヤコビアン，形状関数導関数計算>
!$omp simd
        do je= 1, 8; do je= 1, 8
          <③ガウス数値積分，要素行列成分計算＋格納>
        enddo; enddo
      enddo
      do blk= 1, BLKSIZ
        icel: calculated by (col_index, ib, blk)
!$omp simd
        do je= 1, 8; do je= 1, 8
          <④要素行列成分の全体行列への加算>
        enddo; enddo
      enddo
    enddo
    enddo
    enddo
!$omp end parallel
```

図 5　Type-A 実装の概要（六面体要素）（COLORtot：要素色数（=8），THREAD_NUM：スレッド数（=240），要素色数(=8)，col_index(color)：各色に含まれる要素数，NBLK：要素ブロック総数，BLKSIZ：要素ブロックサイズ，icel：要素番号）

```
!$omp parallel (…)
    do color= 1, COLORtot
!$omp do
    do ip= 1, THREAD_num
      NBLK: calculated by (col_index, color, thread#)
    do ib= 1, NBLK
      do blk= 1, BLKSIZ
        icel: calculated by (col_index, ib, blk)
!$omp simd
        do je= 1, 8; do je= 1, 8
          <②要素行列成分の全体行列（疎行列）におけるアドレス探索＋格納>
        enddo; enddo
      enddo
      do blk= 1, BLKSIZ
        <①各積分点におけるヤコビアン，形状関数導関数計算>
        icel: calculated by (col_index, ib, blk)
!$omp simd
        do je= 1, 8; do je= 1, 8
          <③ガウス数値積分，要素行列成分計算>
          <④要素行列成分の全体行列への加算>
        enddo; enddo
      enddo
    enddo
    enddo
    enddo
!$omp end parallel
```

図 6　Type-B 実装の概要（六面体要素）（COLORtot：要素色数（=8），col_index(color)：各色に含まれる要素数，THREAD_NUM：スレッド数（=240），NBLK：要素ブロック総数，BLKSIZ：要素ブロックサイズ，icel：要素番号）

以上を考慮して，図 5，図 6 に示すような実装（Type-A，Type-B）を試みる．ここで BLKSIZ は各ブロックに含まれる要素数，NBLK は各色，各スレッド内の要素ブロックの総数である．

Type-A（図 5）

- 図 3 に示した①～④の処理のうち，②，①＋③，④を分離して，3 つのループとする．
- 疎行列アドレス記憶用配列，要素行列用配列のための追加の記憶容量が必要である．

Type-B（図 6）

- 図 3 に示した①～④の処理のうち，②，①＋③＋④を分離して，2 つのループとする．
- 疎行列アドレス記憶用配列のための追加の記憶容量が必要である．要素行列用配列の記憶は不要である．

両者のうち，アプリケーション開発者の負担の少ないのは Type-A である．図 5 に示す②と④の計算を実施しているループは分離してライブラリ化することが可能である．先行研究〔5〕では，do ip= 1, THREAD_num の前に!omp parallel do ディレクティヴを挿入していたが，OpenMP スレッド生成・消滅のオーバーヘッドを回避するために，図 5，図 6 のように do icol= 1, NCOLORtot の前に!omp parallel ディレクティヴを挿入する実装に変更した．

OpenMP 4.0 から，プログラム中で!$omp simd ディレクティヴを使って明示的にベクトル化，SIMD 化を有効にすることが可能となった〔17〕．本研究では，図 4～6 に示すように，要素行列に関連する二重ループ（do ie=1, m_e, do je= 1, m_e, m_e は各要素の節点数）の前に!$omp simd を挿入している．MIC では SIMD 幅が 512bit であるため 64bit の倍精度実数におけるベクトル長は 8 となる．

4.2 計算ケース

表 2 に計算ケースを示す．ここでは：

- 要素（六面体，四面体）
- 実装タイプ（オリジナル，Type-A，Type-B）
- !$omp simd の有無

を考慮して計算ケースを設定している．Hex/Tet-A1, A2, B1, B2 では図 5～6 に示す要素ブロックサイズをパラメータとして計算を実施した．

表 2　計算ケース（MIC）

ケース名	要素タイプ	実装タイプ	SIMD 指示行
Hex-O1	六面体	オリジナル（図 4）	無し
Hex-O2			有り
Hex-A1		Type-A（図 5）	無し
Hex-A2			有り
Hex-B1		Type-B（図 6）	無し
Hex-B2			有り
Tet-O1	四面体	オリジナル（図 4）	無し
Tet-O2			有り
Tet-A1		Type-A（図 5）	無し
Tet-A2			有り
Tet-B1		Type-B（図 6）	無し
Tet-B2			有り

4.3 計算結果

図 7 及び図 8 は六面体要素，四面体要素について，要素ブロックサイズをパラメータとして計算を実施した場合の計算結果（計算時間）である．

六面体要素については,先行研究でも示した通り,!$omp simd を挿入しない場合は，オリジナル実装（Hex-O1）と Type-B （Hex-B1） の性能はほぼ同じであり， Type-A （Hex-A1）の性能はそれより 40%程度高い．!$omp simd の挿入によって更に性能は向上し，Hex-A2 の最大性能は Hex-O1 よりも 77%高くなっている．要素ブロックサイズの影響は小さいが,!$omp simd を挿入した場合(Hex-A2, Hex-B2）は，要素ブロックサイズが 16 より大きい場合に性能が低下している．

全てのケース（Hex-O，Hex-A，Hex-B）において!$omp simd 挿入により性能が向上している．先行研究〔5〕においては，Intel コンパイラ特有のベクトル化／SIMD 化ディレクティヴである!dir$ simd の適用を試みたが却って性能が低下し，計算時間が 1.3 倍～1.7 倍となっていた．Hex-A2 と Hex-B2 は要素ブロックサイズが 16 より小さい場合はほぼ同じ性能であり，!$omp simd 挿入により Type-A と Type-B の差異は解消されている．

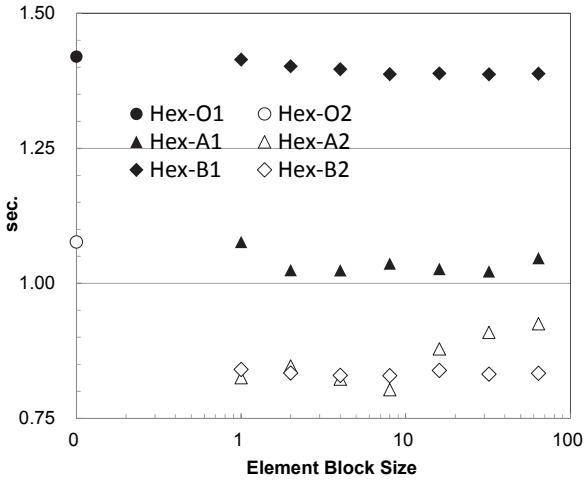

図 7 GeoFEM／Cube 計算結果，係数行列生成部計算時間（六面体要素）

四面体要素については，傾向が六面体要素の場合と非常に異なっている．まず，全般的に Type-B の方が Type-A と比較して性能が高い．四面体要素では，2.3 で述べた②（疎行列アドレス探索)のコストの比率が六面体要素より高い．また，!$omp simd 挿入の効果も六面体の場合と比較して少ない．特に，Tet-O1⇒Tet-O2 ではディレクティヴ挿入によって性能が大幅に低下している．Tet-B1⇒Tet-B2 では一定の効果はあるものの，性能向上は 6%程度に留まっている．四面体の場合，!$omp simd を挿入するループの長さは 4 であるためベクトル化の効果が十分に得られていない可能性がある．また六面体と比較して色数が増加しているため，同期のオーバーヘッドが増大している可能性がある〔2〕．

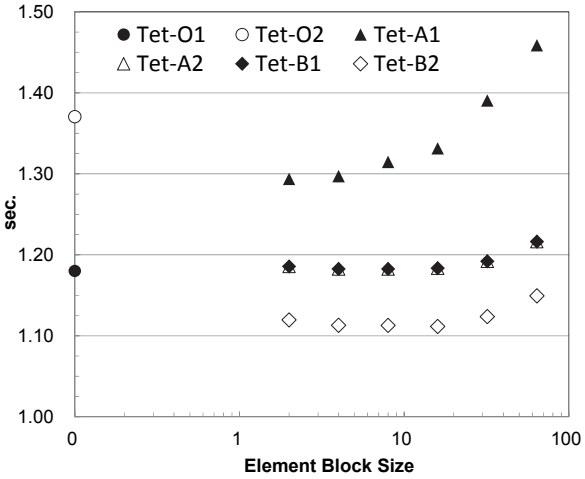

図 8 GeoFEM／Cube 計算結果，係数行列生成部計算時間（四面体要素）

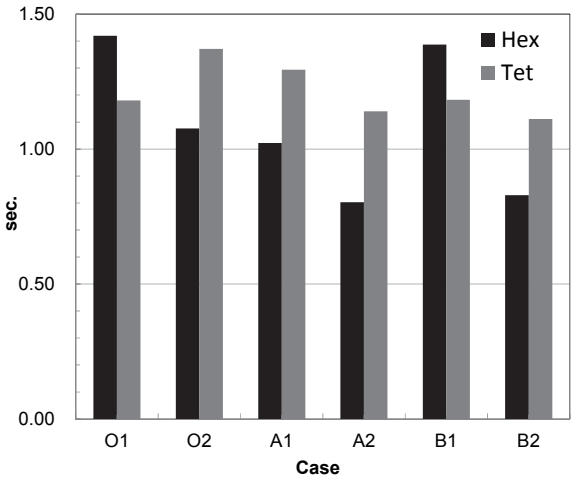

図 9 GeoFEM／Cube 計算結果，係数行列生成部計算時間，最適な要素ブロックサイズを選択した場合

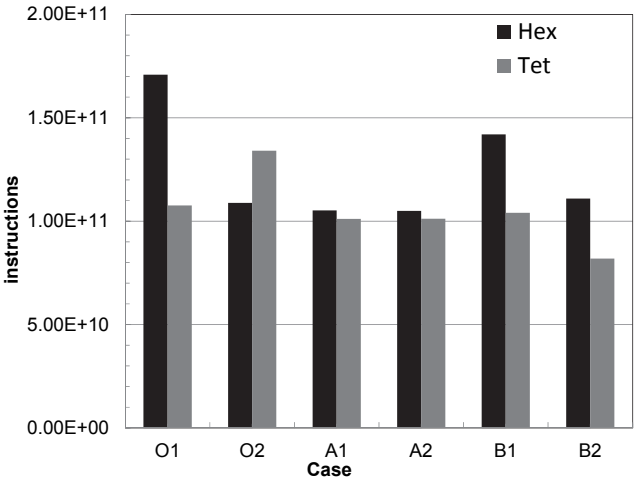

図 10 GeoFEM／Cube 計算結果，係数行列生成部実行命令数（Intel Vtune〔18〕により測定），最適な要素ブロックサイズを選択した場合

図 9 は各ケースにおいて最良の性能を与える要素ブロックサイズにおける計算時間, 図 10 は図 9 の各ケースにおける係数行列生成部の実行命令総数であり，Intel VTune〔18〕を使用して測定した．図 10 と図 9 は概ね相関しているが，図 9 における Hex-A1⇒Hex-A2 の`!$omp simd` 挿入による性能向上に対応した実行命令数の減少が見られない．一方で Tet-B2 では大幅な実行命令数の減少が見られる．Vtune の測定毎の結果の変動もあり，今後の検討が必要である．

5. NVIDIA Tesla K40 における最適化

5.1 最適化の概要

GPU では，分割損が生じない（≒分割することで計算量が増加しない）条件で，全体の処理を細かく分解したときの最小の処理単位を, 各 GPU スレッドに担当させる手法が一般的である．例えば，配列の各成分の計算を，分割損なく独立に実行できる場合は，各 GPU スレッドに配列の 1 成分の計算処理を担当させればよい．この場合は，隣接スレッドは配列の隣接成分をアクセルすることが多く，GPU で高いメモリ性能を実現する条件であるコアレスアクセス条件が自然と満たされる．一方，計算量を増加させないため, 各 GPU スレッドに複数の配列成分を担当させるのが良い場合は，データレイアウトに注意する必要がある．複数の配列成分がメモリ上に連続的に配置されていると，例えば AoS（Array of Structure）と呼ばれるデータレイアウトのときはコアレスアクセス条件を満たせないことが多い．この場合，高い性能を得るにはデータレイアウトを SoA（Structure of Array）に変更する必要がある．

上記観点に基づき，処理①～④（図 3）を，それぞれ別個の GPU カーネルとして実行することを前提とし, 各処理における最適な並列化方法を思考する．説明の簡略化のための，ここでは六面体要素のみを対象とする．また，4. でも説明した通り, 六面体要素の全てを同時に計算するのは，処理間のデータ受け渡しに巨大な一時配列が必要となり現実的でない．一方，六面体要素を 1 つずつ計算したのでは GPU リソースを有効に使えない．マルチコア CPU 向けに採用したブロッキングとは意味合いが異なるが，GPU でも `BLKSIZ` 個の六面体要素を同時に処理することを前提とし，それに伴い，処理間のデータ受け渡し配列のデータレイアウトに関しても思考する．なお，思考結果の概要コードは図 11 に示す通りである．

処理①では，六面体要素あたり 192 個の成分が計算される．この 192 個の成分は，24 個の成分を最小単位として，更に分割損ゼロで 8 組に分解可能であるが，ここでは 192 個の成分の計算をまとめて 1 つの GPU スレッドに担当させる方針をとる（コード変更量を少なくするため）．計算結果が格納される一時配列は `PNX,PNY,PNZ` の 3 種で，この 3 配列の六面体要素あたりの次元構成は `(2,2,2,8)` である．配列への計算結果の書き込みがコアレスアクセス条件を満たすよう，配列の次元構成は `(BLKSIZ,2,2,2,8)` とする．

処理②では，六面体要素あたり 64 個のアドレス（全体行列のインデックス）が探索される．この 64 個のアドレスはそれぞれ分割損なく探索可能なので，各アドレスの探索を 1 つの GPU スレッドに担当させる．探索結果を格納する一時配列を `IDX` とすると，この配列の六面体要素あたりの次元構成は `(8,8)` である．処理①の場合と同様に，配列への計算結果の書き込みがコアレス条件を満たすことを考えると, 処理①と②では GPU スレッドへの処理の割り当て方法が異なるので，配列 `IDX` にとって最適な次元構成は `(8,8,BLKSIZ)` となる．

処理③では，六面体要素あたり 576 個の成分が計算される．この 576 個の成分は，9 個の成分を最小単位として，分割損ゼロで 64 組に分解可能である．従って，9 個の成分の計算を 1 つの GPU スレッドに担当させる. 計算結果を格納する一時配列を `AXX` とすると，この配列の六面体要素あたりの次元構成は `(8,8,9)` である．コアレス条件を満たすことを考えると，一時配列 `AXX` の次元構成は `(8,8,9,BLSIZ)` が望ましい．

```
   do ib= 1, NBLK
!$acc parallel
!$acc loop gang vector
      do blk= 1, BLKSIZ
         icel = blk + BLKSIZ * (ib-1)
         <①各積分点におけるヤコビアン，形状関数導関数計算>
      enddo
!$acc end parallel
!$acc parallel
!$acc loop gang
      do blk= 1, BLKSIZ
         icel = blk + BLKSIZ * (ib-1)
!$acc loop collapse(2) vector
         do ie= 1, 8; do je= 1, 8
            <②要素行列成分の全体行列（疎行列）におけるアドレス探索＋格納>
         enddo; enddo
      enddo
!$acc end parallel
!$acc parallel
!$acc loop gang
      do blk= 1, BLKSIZ
         icel = blk + BLKSIZ * (ib-1)
!$acc loop collapse(2) vector
         do ie= 1, 8; do je= 1, 8
            <③ガウス数値積分，要素行列成分計算>
         enddo; enddo
      enddo
!$acc end parallel
!$acc parallel
!$acc loop gang
      do blk= 1, BLKSIZ
         icel = blk + BLKSIZ * (ib-1)
!$acc loop collapse(3) vector
         do ie= 1, 8; do je= 1, 8 ; do k= 1, 9
            <④要素行列成分の全体行列への加算>
         enddo; enddo; enddo
      enddo
!$acc end parallel
   enddo
```

図 11　Type-G 実装の概要（六面体要素, OpenACC）（`NBLK`：要素ブロック総数，`BLKSIZ`：要素ブロックサイズ）

処理④では，処理③で計算した成分を，処理②で探索したアドレスに足し込む処理が行われる．より正確には，処理②で探索した 64 個のアドレスのそれぞれを先頭アドレスとして，処理③で計算された成分の内，それぞれ対応する 9 個の成分がアドレス連続で加算される．全体行列を `A` とすると，以下の擬似コードに示す通り，六面体要素あたり全体行列 `A` の 576 箇所に対して加算が行われる．

```
do ie=1,8;  do je=1,8;  do k=1,9
    A(k,IDX(je, ie)) += AXX(je,ie,k,icel)
enddo;  enddo;  enddo
```

処理①～③と異なり，書き込み先は一時配列ではなく，プログラム全体で使用される配列である．そのデータレイアウト変更はプログラム全体に影響するため，安易には行えない．そのため，全体行列のデータレイアウトはこのままとし，全体行列への書き込みがコアレスアクセス条件を満たすスレッド配置を選択する．具体的には，576 個の成分のそれぞれに対して 1 GPU スレッドを割り当て，k 軸方向でスレッド番号が連続するスレッド配置である．この配置方法でも，完全なコアレスアクセス状態を作り出すことは出来ないが，連続 9 スレッドのメモリアクセスはコアレスとなる．

5.2 OpenACC 実装

上記の考察結果に基づいて，OpenACC で GPU 対応した擬似コードを図 11 に示す（Type-G 実装）．この実装は，図 4 のオリジナル実装をベースとしている．開発の手順を以下に示す．

1. （オリジナル実装から）カラーリングの取り外し，
2. 処理①～④の分離と一時配列の追加，
3. ブロッキングの適用
4. OpenACC ディレクティブの追加

なお，単純にカラーリングを外してしまうと正しい結果が得られないため，全体行列の加算部分には Atomic 操作を使って同時書き込みを回避している．OpenACC では atomic クローズを使って Atomic 操作が必要な箇所を簡単に指定することができる．擬似コードを以下に示す．

```
!$acc atomic update
     A(k,IDX(je, ie)) += AXX(je,ie,k,icel)
!$acc end atomic
```

もちろん，GPU でもカラーリングを使って全体行列への同時書き込みを回避する方法も有効である．Atomic 実装とカラーリング実装の性能差は後ほど考察する．

5.3 CUDA 実装

CUDA 実装も 5.1 の考察結果に基づいて行った．以下，OpenACC 実装との違いを述べる．

- 処理①：OpenACC 版では，1GPU スレッドあたり 192 個の成分の計算を担当させていたが（コード変更量を少なくするため），CUDA 版では 24 個に変更．それに伴いコアレスアクセス条件を満たすデータ配置が変わるので，一時配列 PNX,PNY,PNZ の次元構成を (2,2,2,8,BLKSIZ)に設定した．
- 処理②と③：OpenACC 版では，スレッドブロックサイズ（スレッド数）を 64 とし，1 つのスレッドブロックに 1 つの六面体要素を担当させていたため（コード変更量を少なくするため），GPU リソース稼働率が若干低下していた．CUDA 版では，スレッドブロックサイズを 128 とし，1 つのスレッドブロックに 2 つの六面体要素を担当させることで，稼働率の低下を抑止する．
- 処理④：OpenACC 版では，全体行列への書き込みがコアレスになるようにスレッドを配置したため，処理③の計算結果の読み込みがコアレスとならず，ロードの効率が悪かった．CUDA 版では，読み込み時と書き込み時でスレッド配置を変更することで，読み込みと書き込み，どちらもコアレスに近い条件で行えるようにした．カーネル実行途中でスレッド配置を切り替えると，スレッド間でデータ交換が必要となるが，それには shared memory を使用する．

5.4 四面体要素の実装

四面体要素は OpenACC+Atomic 版を実装した．処理②と③においては，六面体要素の場合，要素あたり 64 個の探索・計算を独立に行うことができたが，四面体要素の場合は，これが 16 個に減少する．六面体要素のときは，スレッドブロックサイズを 64 とし，1 つのスレッドブロックに 1 つの六面体要素を担当させていた．四面体要素でこの方針を採ると，スレッドブロックサイズは 16 となり，GPU のリソース稼働率が著しく低下する．著しい性能低下を回避するため，スレッドブロックを 128 として，1 つのスレッドブロックに 8 個の四面体要素を担当させるように，OpenACC ディレクティブで簡単に指示できる構造にソースコードを変更した．OpenACC 版には，GPU 内部構造にフィットさせるためのコード変更は入れないことを原則としていたが，ここだけは例外である．

5.5 性能測定と考察

以下の 4 タイプに関して性能測定を実施した：

- タイプ G： 六面体要素，OpenACC，Atomic 操作
- タイプ G_color： 六面体要素，OpenACC，カラーリング
- タイプ G_cuda： 六面体要素，CUDA，Atomic 操作
- タイプ G_tetra： 四面体要素，OpenACC，Atomic 操作

4 つのタイプのそれぞれを 3 種ブロックサイズ(8K, 16K, 32K)で測定した結果を図 12 に示す．図 13 は，ブロックサイズ 32K のときの時間内訳である．表 3 には，NVIDIA Visual Profiler で測定した，総メモリアクセス量とメモリバンド幅を示す（処理③と④）．

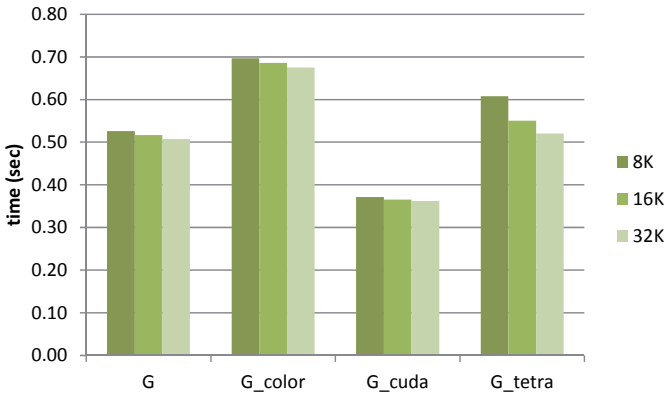

図 12　GPU 上での GeoFEM／Cube 計算結果，係数行列生成部計算時間（ブロックサイズと性能）

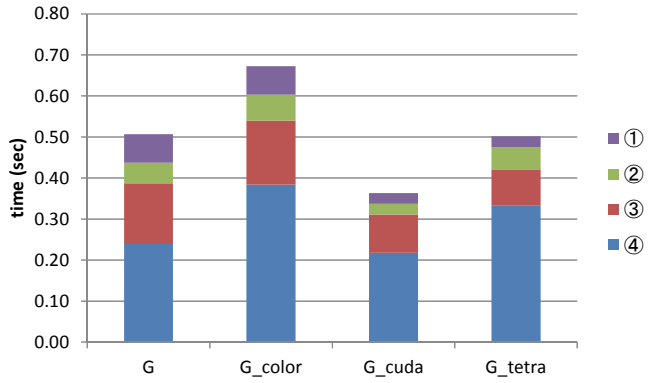

図 13　GPU 上での GeoFEM／Cube 計算結果，係数行列生成部計算時間（ブロックサイズ 32K 時の処理①～④の時間）

表 3　処理③と④の総メモリアクセス量とメモリバンド幅（NVIDIA Visual Profiler 測定値）

		G	G_color	G_cuda	G_tetra
処理③	総アクセス量（GB）	12.8	12.8	12.8	15.5
	バンド幅（GB/sec）	87.9	81.7	138.2	172.5
処理④	総アクセス量（GB）	26.8	37.9	26.3	38.3
	バンド幅（GB/sec）	111.7	98.7	120.1	114.9

以下に各パラメータの影響について考察を示す：

ブロックサイズ

図 12 より，どのタイプでもブロックサイズを大きくするほど性能が向上する傾向が見てとれる．GPU 実装では，ブロックサイズとは各 GPU カーネルの処理量を意味する．一般的には，処理量が多くなると並列性が増え，GPU 稼働率を上げやすくなるので，妥当な結果と言える．

Atomic 版とカラーリング版

Atomic 版（タイプ G）とカラーリング版（タイプ G_color）を比較すると，Atomic 版の方が速いという結果となった．定性的には処理④で顕著な違いが生じるが，その処理④の性能が Atomic 版の方が速いという結果である．やや意外な結果であるが，原因は総メモリアクセス量の違いと推測される．表 3 を見ると，カラーリング版の総メモリアクセス量は，Atomic 版の 1.4 倍と多い．カラーリング版では，全体行列への加算が同じ箇所に行われることがないため，キャッシュは効かない．Atomic 版では全体行列への加算が同じ箇所に行われることがあり，キャッシュが効いてメモリアクセス量を削減できる可能性がある．実際，測定に使った Tesla K40（Kepler アーキテクチャ）は Atomic 操作をハードウェアサポートしており，L2 キャッシュ上で Atomic 操作が行われる．Atomic 操作には処理オーバーヘッドがあるが，そのマイナス分より，メモリアクセス量削減によるプラス分が上回った結果と考えられる．

OpenACC 版と CUDA 版

OpenACC 版（タイプ G）とカラーリング版（タイプ G_cuda）を比較すると，処理①～④の全て，CUDA 版の性能が上回っている（図 13）．前述の通り，処理①～③には OpenACC 実装と比べて GPU 稼働率を高める工夫が入っており，実際にプロファイラで稼働率が上がっていることが確認できている．処理④では shared memory を用いてコアレスアクセス比率を高めている．表 3 を見ると，処理④の実効メモリバンド幅は向上しているが，向上率は 7%とわずかである．おそらく shared memory 使用のオーバーヘッドと相殺されたと考えられる．

六面体版と四面体版

六面体版（タイプ G）と四面体版（タイプ G_tetra）は，処理全体の時間はほぼ同等となったが，時間の内訳はかなり異なる結果となった（図 13）．六面体版と比べ，四面体版では，処理①と③の時間は減少，処理②と④の時間は増加している．これは，六面体版と四面体版の特性の違いを考えると概ね妥当な結果である．

前述の通り，四面体の場合は要素内積分を解析的に実施できるため，その計算量が少ない．これが処理①の時間減の理由である．要素内の節点対数は 64 個から 16 個と 1/4 倍と減るが，要素数は 6 倍に増えるため，全体行列への総加算回数は 1.5 倍に増える．実際，表 3 を見ると，処理④の総メモリアクセス量は，六面体から四面体で 1.43 倍増加しており，これが処理④の時間増の原因と考えられる．処理②のアドレス計算の回数は，全体行列への加算回数に比例するので，これで処理②の時間増も説明がつく．処理③は，加算回数の増加にともない総メモリ書き込み量は 1.5 倍増えるが，積分の減少により総メモリ読み込み量は 4 割弱に減る．四面体の処理③は，総メモリアクセス量は増加するものの，メモリの READ:WRITE 比率が 1:12 とほぼ WRITE 主体となる．一般的に，メモリアクセスパターンが単純なほど実効メモリバンド幅は上げやすいため，これによって高い実効メモリバンド幅が実現でき，処理時間の短

縮に繋がっていると考えられる．

6. まとめ

本研究では，並列有限要素法による三次元弾性静解析アプリケーションに基づく性能評価用ベンチマーク GeoFEM／Cube において，OpenMP によってマルチスレッド並列化された係数行列生成部を元に，Intel Xeon Phi および NVIDIA Tesla K40 を対象としてそれぞれの特性を生かした最適化を実施した．

表 4 は各最適化のベストケース（行列生成部計算時間）である．Intel Xeon Phi（MIC）に対しては，先行研究で提案されたブロック化と OpenMP 4.0 からサポートされている機能である`!$omp simd`ディレクティヴを使って明示的にベクトル化，SIMD 化を適用することによって，オリジナル実装と比較して六面体において約 80%，四面体において約 6%の性能向上を得られた．

NVIDIA Kepler K40（K40）については，まずオリジナル実装に対して GPU に適した変数配置を適用し，OpenACC を使用したスレッド並列化を実施した．並列計算からカラーリングを取り除き，Atomic 操作を適用した実装では，Tesla K40（Kepler アーキテクチャ）によるハードウェアサポートの効果もあり，六面体において 33%の速度向上が得られている．更に CUDA 化によってカラーリング適用の場合と比較して 87%改善している．

表 4　行列生成部計算時間の比較（sec.）

		MIC	K40	
六面体	カラーリング	0.803	0.675	OpenACC
	Atomic	-	0.507	OpenACC
		-	0.362	CUDA
四面体	カラーリング	1.111	-	
	Atomic	-	0.520	

MIC と K40 を比較すると，六面体のカラーリングを適用した場合では，MIC : 0.803 秒（80.2GFLOPS（先行研究〔5〕の結果より推定)），K40 : 0.675 秒（95.4GFLOPS（同））と拮抗している．表 1 に示すハードウェアの性能を比較すると，ピーク演算性能，メモリバンド幅（STREAM Triad 性能）で K40 がそれぞれ約 1.40 倍上回っていることを考慮すると，MIC は健闘しているとも言えるが，MIC においては更に高い演算性能を引き出すために，ループ処理を単純化し，ベクトル化，SIMD 化を適用しやすくする必要がある．現在の実装では図 5，図 6 に示すように，確実に各スレッドにおける負荷を均等にするために，スレッドに対するループが陽に存在しており，そのための特別な配列も準備されている．図 11 に示す K40 用の実装ではそのようなループが存在せず，よりシンプルな構造となっている．

また四面体では，MIC においては十分なベクトル長が得られないため計算時間が六面体より長く，ベクトル化，SIMD 化の効果も少ない．K40 における Type-G 実装ではループ構造を変化させることによって六面体と比較して計算時間を短縮していることから，同様の最適化が効果的であると考えられる．

本研究においては Tesla K40（Kepler アーキテクチャ）における Atomic 操作のハードウェアサポートの効果が顕著であった．実はカラーリングのプロセスは並列化が困難であるため，MIC では係数行列生成部の 10 倍以上の計算時間を要している．K40 ではホスト CPU でカラーリングを実行しているため，この部分の負担は極めて小さい．MIC でも同様にホスト CPU に受け持たせることは考えるが，次世代の Xeon Phi プロセッサである Knights Landing（KNL）への適用を考慮すると，カラーリング部の並列化も必須である．

参考文献

1) 中島研吾，前処理付きマルチスレッド並列疎行列ソルバー，情報処理学会研究報告（HPC-139-6）（2013）
2) 中島研吾，拡張型 Sliced-ELL 行列格納手法に基づくメニィコア向け疎行列ソルバー，情報処理学会研究報告（HPC-147-3）（2014）
3) SIAM Conference on Computational Science & Engineering 2015 (CSE 15), Salt Lake City, UT, USA, March 2015, https://www.siam.org/meetings/cse15/
4) 大島聡史，林雅江，片桐孝洋，中島研吾，三次元有限要素法アプリケーションにおける行列生成処理の CUDA 向け実装，情報処理学会 研究報告（HPC-130-11）（2011）
5) 中島研吾，大島聡史，塙敏博，有限要素法係数行列生成プロセスのマルチコア・メニィコア環境における最適化，情報処理学会研究報告（HPC-146-22）（2014）
6) ppOpen-HPC：科学技術振興機構戦略的創造研究推進事業（CREST)「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出：自動チューニング機構を有するアプリケーション開発・実行環境」，http://ppopenhpc.cc.u-tokyo.ac.jp/
7) 中島研吾，佐藤正樹，古村孝志，奥田洋司，岩下武史，阪口秀，自動チューニング機構を有するアプリケーション開発・実行環境 ppOpen-HPC，情報処理学会研究報告（HPC-130-44）（2011）
8) Nakajima, K., Satoh, M., Furumura, T., Okuda, H., Iwashita, T., Sakaguchi, H., Katagiri, T., Matsumoto, M., Ohshima, M., Jitsumoto, H., Arakawa, T., Mori, F., Kitayama, T., Ida, A., and Matsuo, M.Y., ppOpen-HPC: Open Source Infrastructure for Development and Execution of Large-Scale Scientific Applications on Post-Peta-Scale Supercomputers with Automatic Tuning (AT), Optimization in the Real World –Towards Solving Real-Worlds Optimization Problems, Mathematics for Industry 13, 15-35, Springer (2015)
9) 中島研吾，片桐孝洋，大島聡史，塙敏博，ppOpen-APPL/FVM を使用した並列有限要素法アプリケーションの開発，情報処理学会研究報告（HPC-151-24）（2015）
10) Intel, http://www.intel.com/
11) NVIDIA, http://www.nvidia.com/
12) GeoFEM：並列有限要素法による固体地球シミュレーションプラットフォーム，http://geofem.tokyo.rist.or.jp
13) 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)
14) 中島研吾，片桐孝洋，マルチコアプロセッサにおけるリオーダリング付き非構造格子向け前処理付反復法の性能，情報処理学

会研究報告（HPC-120-6）（2009）
15) 富士通，http://www.fujitsu.com/
16) STREAM, https://www.cs.virginia.edu/stream/
17) OpenMP, http://openmp.org/wp/
18) Intel VTune,
https://software.intel.com/en-us/intel-vtune-amplifier-xe