大規模疎行列演算と3次元有限要素並列計算

鈴木, 厚
九州大学大学院数理学研究院

https://doi.org/10.15017/1470385

出版情報：九州大学情報基盤センター広報 : 全国共同利用版. 1 (3), pp.135-143, 2001-10. 九州大学情報基盤センター
バージョン：
権利関係：

大規模疎行列演算と3次元有限要素並列計算

鈴木 厚 *

1 はじめに

有限要素法 [2, 9] は偏微分方程式の離散化手法の一つであり, 領域の形状に対する汎用性があること, 数学理論が整備され数値解に対する誤差評価を持つスキームを用いることができることなどの長所がある. 筆者らは地球マントル対流問題を記述する無限プラントル数を持つブシネスク流体の熱対流の有限要素スキームと3次元計算コードの開発を行ってきた [5, 6, 7, 8].

3次元問題に対する有限要素法では, 領域を四面体に分割し離散化を行うことで, 節点での未知量に関する連立方程式を得る. 六面体を用いた要素分割も可能であるが, 要素分割が容易であること, 演算量が少ないことを考慮して四面体分割を採用する. この連立方程式を構成する行列は疎行列である. 有限要素法では, 行列データに規則性の無い非構造データを扱うことになる.

クリロフ部分空間法 [1, 3] などの反復解法を用いて連立方程式を解く. この解法では行列とベクトルの積計算と, ベクトルの内積計算を繰り返し行う. 本稿では, ワークステーション型のRISCプロセッサーの大規模疎行列とベクトルの積演算の性能を調べることと, キャッシュメモリーの利用と並列化を行い演算の高速化を図ることを目的とする.

2 流れ問題に現れる有限要素行列

有限要素法では支配方程式にテスト関数を乗じ, 対象とする領域で積分を行い, 弱形式を得る. さらにこの弱形式を離散化することで, 剛性行列からなる有限要素方程式を得る. 本稿では, 3次元の流れ問題を考える際に必要な次の剛性行列を取り扱う. $\{\varphi_i\}_{0\le i<N}$ を流速の有限要素基底 [2] とする. 流速の自由度 N は節点自由度の3倍である. $N\times N$ 行列 A を次のものする.

$$[A]_{ij} := 2\int_\Omega \sum_{1\le k,l\le 3} D_{kl}(\varphi_j)D_{kl}(\varphi_i)dx \quad (0\le i,j<N). \tag{1}$$

ここに $D(u)$ は変形速度テンソル $D_{kl}(u)=(\partial u_k/\partial x_l+\partial u_l/\partial x_k)/2$ $(1\le k,l\le 3)$ である. この行列は, 基底関数が局所的であることから疎行列となり, 要素分割に自由度があることから一般に零成分の分布に規則性のない非構造データとなる. また, A は対称行列であることに注意する.

以下, 行列と数ベクトルのインデックスはC言語の慣例に従い0から数えるものとする. ベクトル $\vec{v}$ のインデックス i で示される値を $[\vec{v}]_i$, 行列 A の i 行 j 列の成分を $[A]_{ij}$ と書く.

3 疎行列の記憶

CRS (Compressed Row Storage) フォーマット [1] を用いて, 疎行列を記憶する. このフォーマットは, 各行の零成分を取り除いて圧縮し, 行列の非零成分のみを記憶するため, 省メモリー

*九州大学 大学院数理学研究院 E-mail: asuzuki@math.kyushu-u.ac.jp

である. n_X を行列 A の行の数, n_A を行列 A の非零要素の数とする. CRS フォーマットでは行列 A の値を格納する長さ n_A の浮動小数点ベクトル $\vec{v}_A \in \mathbb{R}^{n_A}$, 列のインデックスを記憶する長さ n_X の整数ベクトル $\vec{c}_A \in \mathbb{Z}^{nx}$ と, 各行での列の開始位置を記憶する長さ n_X+1 の整数ベクトル $\vec{r}_A \in \mathbb{Z}^{nx+1}$ を用いる. A の k 行 l 列にある非零要素 $[A]_{kl}$ を配列 $\vec{v}_A$ の インデックス β の場所 $[\vec{v}_A]_\beta$ に格納する. この時 $\vec{c}_A$ のインデックス β での値 $[\vec{c}_A]_\beta$ は l を記憶している. $\vec{r}_A$ は各行で列を開始するインデックスを記憶するため, $[\vec{r}_A]_k \leq \beta < [\vec{r}_A]_{k+1}$ である. ただし, $\vec{r}_A$ の n_X+1 番目の値 $[\vec{r}_A]_{n_X}$ は n_A とする.

行列 A が対称行列の場合, 下三角部分と対称部分のみを記憶すれば良いことに注意する.

4 行列ベクトル積演算と正射影演算

図 1 (左) に CRS フォーマットによるデータ構造 (C 言語の構造体による記述) を示す. SIZE はベクトル長さ, すなわち行列の行数 n_X を表し, ROWS1 は $\vec{r}_A$ の長さを表す. また, NONZEORS は行列の非零要素の数 n_A を表す. 説明の簡単化のため, これらのサイズを静的に決定するが, 実際の有限要素プログラムコードでは, プログラム中で動的に決定する. これは次の理由による. ベクトル長さは自由度数, 非零要素数は自由度数と要素分割に依存する. 要素分割に関するデータを入力データとして与えることで, 領域形状に対するプログラムの汎用性を実現することができる.

```
#define SIZE       /****/
#define ROWS1      (SIZE + 1)
#define NONZEROS   /****/

typedef struct {
  int row[ROWS1];
  int index[NONZEROS];
  double value[NONZEROS];
} sparse_matrix;

sparse_matrix aa;
```

```
for (i = 0; i < SIZE; i++) {
  v_tmp = 0.0;
  j_begin = aa.row[i];
  j_end = aa.row[i + 1] - 1;
  for (j = j_begin; j < j_end; j++) {
    j_tmp = aa.index[j];
    v_tmp += aa.value[j] * u[j_tmp];
    v[j_tmp] += aa.value[j] * u[i];
  }
  j_tmp = aa.index[j_end];
  v_tmp += aa.value[j_end] * u[j_tmp];
  v[i] += v_tmp;
}
```

図 1: 疎行列データ構造 (左) と行列ベクトル積 (右)

図 1. (右) に対称行列 A とベクトル $\vec{u}$ の積演算を行い, ベクトル $\vec{v}$ を得る手続き

$$\vec{v} := A\vec{u}$$

を示す. この演算ではベクトル $\vec{u}$ (u[]), $\vec{v}$ (v[]) の要素のアクセスに, 間接参照が必要である. これは, 各行の零成分を取り除いて圧縮して成分を記憶することから生じている. 行列が対称であることを利用し, 間接参照のインデックス j に関する for ループで行列の下三角と上三角部

```
s_tmp = 0.0;
for (i = 0; i < SIZE; i++) {
  s_tmp += x[i] * y[i];
}
for (i = 0; i < SIZE; i++) {
  x[i] -= s_tmp * y[i];
}
```

図 2: 正射影演算

分の積を計算した後, 対角部分の積の計算を行っている. ここで, 配列 v[] は 0 に初期化されているものとする.

図 2. にベクトルの正射影演算を示す. ベクトル $x \in \mathbb{R}^{n_X}$ から長さが 1 に規格化されたベクトル $\vec{y} \in \mathbb{R}^{n_X}$ $((\vec{y},\vec{y}) = 1)$ の成分を除去する演算

$$\vec{x} := \vec{x} - (\vec{x},\vec{y})\,\vec{y}$$

である. ここで, $(\vec{x},\vec{y})$ はベクトル $\vec{x}$ と $\vec{y}$ の内積である

$$(\vec{x},\vec{y}) := \sum_{0 \le i < n_X} [\vec{x}]_i [\vec{y}]_i .$$

正射影演算は, 非圧縮流れ問題において, 圧力の自由度を除去する時などに必要な演算である.

表 1, 2 に数値実験に使用した計算機と C コンパイラオプションをそれぞれ示す.

表 3 に 図 1 (右) に示される行列ベクトル積の実行結果, 表 4 に 図 2 に示される正射影演算の実行結果を示す. 両者とも 1 CPU により実行した. 表 3 の結果は, 図 7 に示される球殻領域の有限要素分割と離散化によって得られた流速の剛性行列 (1) に対するものである. ベクトルの長さ n_X (SIZE) は 973,956 であり, 行列 A の下三角部分と対角部分の非零要素の数 n_A (NONZEROS) は 21,909,498 である. 行列 A の一行あたりの平均的な非零要素の数は 23×2 程度であり, 行列 A の非零成分は要素全体 ($n_X \times n_X$ 個) に対して, (0.0023×2)% 程度である.

表 3, 表 4 とも演算量 (Flop) は SR8000/MPP のハードウェア機能により計測したものであり, MFLOPS 値は (演算量)/(実行時間) により算出した.

SR8000/MPP の 1 CPU での理論性能は 1.8GFLOPS であるが, 行列ベクトル積の演算では 10% 程度, 正射影演算は 22% 程度の性能になっている. SR8000/MPP のベクトル化機能では図 1 (右) のコードは十分にベクトル化されないため, スカラー型の CPU として利用している. ワークステーション型の計算機 GP7000F, GS320 では主記憶へのアクセス速度が演算の制約になっていると考えられる.

CRS フォーマットはデータ構造が単純であり, 省メモリーの観点から優れた記憶方式であるが, 計算効率は高くない. ベクトル化に向いた JDS フォーマット [1] があるが, 本稿ではスカラー型の RISC プロセッサーでの速度向上を目指す.

表 1: 数値実験で使用した計算機

計算機名	CPU 仕様	設置場所
GP7000F	SPARC 64	九州大学 情報基盤センター
GS320	Alpha 21264 (731MHz)	九州大学 情報基盤センター
SR8000/MPP	RISC プロセッサー (1 CPU あたりの 理論性能 1.8GFLOPS)	東京大学 情報基盤センター

表 2: コンパイラオプション

GP7000F	`fcc -Kfast -Kfast_GP=2 -Kprefetch -KV9FMADD`
GS320	`cc -fast -O4 -arch host -tune host -inline speed -non_shared`
SR8000/MPP	`cc -Os -64 -restrict -noparallel -prefetch -nopreload`

表 3: 行列ベクトル積 ($n_X = 973,596$, $n_A = 21,909,498$, 演算量 88.612M Flop)

計算機名	実行時間 (秒)	MFLOPS
GP7000F	2.03	43.7
GS320	0.446	198.7
SR8000/MPP	0.490	180.8

表 4: 正射影 ($n_X = 973,596$, 演算量 3.894M Flop)

計算機名	実行時間 (秒)	MFLOPS
GP7000F	0.0936	41.6
GS320	0.0330	118.0
SR8000/MPP	0.00962	404.8

5 高速化

5.1 合同な部分領域への分割

計算領域の特徴を活用し, 合同な部分領域への分割を行う. 領域は M 個の重りなのない部分領域 $\Omega^{(m)}$ $(0 \leq m < M)$ に分割されているとし, それぞれ, ある直交変換 $R^{(m)}$ で参照領域 $\Omega^{(0)}$ を写したものであるとする. また, 部分領域での要素分割も参照領域での要素分割を $R^{(m)}$ で写したものであるとする.

成分が $-1, 0, 1$ のみからなる直交変換 $R^{(m)}$ に対して, 次の性質が成り立つ.

定理 1 $A^{(m)}$ を $\Omega^{(m)}$ での流速の剛性行列とする．直交変換の符号より定まる，対角要素が -1 と 1 からなる対角行列 $S^{(m)}$ が存在して，

$$A^{(m)} = S^{(m)} A^{(0)} S^{(m)} \tag{2}$$

が成り立つ (証明は [7] を参照)．

この定理より，流速の剛性行列は領域全体で作成すること無く，参照領域 $\Omega^{(0)}$ でのみ作成し記憶すれば良いことになり，大幅なメモリー節約が可能になる (詳細は [7] を参照)．領域分割を並列計算に利用することにより計算の高速化を実現することができる．

図 7 に球殻領域の 48 部分領域への領域分割を示す．ここで，領域分割と要素分割が見易いように，参照領域を取り除き，各部分領域をシフトしている．

5.2 キャッシュメモリーの活用と OpenMP による並列化

式 (2) を利用して行列ベクトル積の演算を行う．領域の分割によって生じた内部境界上のデータを足し合わせる操作が必要であるが，部分領域毎に独立して次の計算を行うことができる．

$$\vec{u}^{(m)} := S^{(m)} \vec{x}^{(m)} \quad (0 \le m < M), \tag{3}$$

$$\vec{v}^{(m)} := A^{(0)} \vec{u}^{(m)} \quad (0 \le m < M), \tag{4}$$

$$\vec{y}^{(m)} := S^{(m)} \vec{v}^{(m)} \quad (0 \le m < M). \tag{5}$$

M (SUBDOMAINS) 個の領域に対して P (PROCS) 個の CPU を使用して計算を行う．P を M の約数になるように選ぶと，各 CPU は M/P (SIZE2) 個の部分領域を扱うことなり，負荷分散の良い並列計算が可能である．図 3, 4 に (4) の疎行列ベクトル積に対応する演算を示す．図 3 は，領域分割のループを行列ベクトル積の外側に置いたもの，図 4 は，再内側に領域分割のループを置くようにループの入れ替えと配列の記憶方法の変更を行ったものである．id は CPU のインデックスを表す (0 ≤id<PROCS). また，SIZE1 は部分領域での自由度を表す．

並列化は OpenMP[4] を用いて行った．loop 1, loop 2 ともに OpenMP の並列実行領域の中に記述され，インデックス id に対する並列化が行われる．OpenMP では 変数と配列に対して，スレッド間で共有するかしないかの指示が必要であるが，ここでは OpenMP による並列化コードの詳細は省略する．

図 5 と図 6 に部分領域数を $M = 48$ としたときの，全体領域の行列とベクトルの積の演算 (内部境界での足し合わせ処理を含む) に対する GP7000F と GS320 での並列化効率を示す．

loop 2 では，行列の値 aa.value[j] が インデックス q に関するループで再利用されるため，キャッシュメモリーを有効に利用していると推測される．1 CPU での実行時，loop2 は loop1 に比較し，GP7000F では 1.73 倍，GS320 では 1.42 倍の速度向上が得られている．

また，図 4 から GP7000F では，単体 CPU あたりの演算能力は GS320 に比べ低いが，OpenMP による並列化が効率良く行われることがわかる．loop2 で 6 CPU までは，ほぼ CPU 台数分の速度向上が得られている．24 CPU 使用時の速度向上は 13.2 倍である．

6 おわりに

ワークステーション型の RISC 計算機による疎行列ベクトル演算は，主記憶へのランダムな間接アドレス参照を含むことなどから，ピーク性能を実現できない．しかし，行列データにある

```
SIZE2 = SUBDOMAINS / PROCS;
for (q = 0; q < SIZE2; q++) {
  p =  id * SIZE2 + q;
  for (i = 0; i < SIZE1; i++) {
    v_tmp = 0.0;
    j_begin = aa.row[i];
    j_end = aa.row[i + 1] - 1;
    for (j = j_begin; j < j_end; j++) {
      j_tmp = aa.index[j];
      v_tmp += aa.value[j] * u[p][j_tmp];
      v[p][j_tmp] += aa.value[j] * u[p][i];
    }
    j_tmp = aa.index[j_end];
    v_tmp += aa.value[j_end] * u[p][j_tmp];
    v[p][i] += v_tmp;
  }
}
```

図 3: 部分領域ループを外側に持つ loop 1

```
SIZE2 = SUBDOMAINS / PROCS;
for (i = 0; i < SIZE1; i++) {
  for (q = 0; q < SIZE2; q++) {
    p = id * SIZE2 + q;
    v_tmp[p] = 0.0;
  }
  j_begin = aa.row[i];
  j_end = aa.row[i + 1] - 1;
  for (j = j_begin; j < j_end; j++) {
    j_tmp = aa.index[j];
    for (q = 0; q < SIZE2; q++) {
      p = id * SIZE2 + q;
      v_tmp[p] += aa.value[j] * u[j_tmp][p];
      v[j_tmp][p] += aa.value[j] * u[i][p];
    }
  }
  j_tmp = aa.index[j_end];
  for (q = 0; q < SIZE2; q++) {
    p = id * SIZE2 + q;
    v_tmp[p] += aa.value[j_end] * u[j_tmp][p];
    v[i][p] += v_tmp[p];
  }
}
```

図 4: 部分領域ループを再内側に持つ loop 2

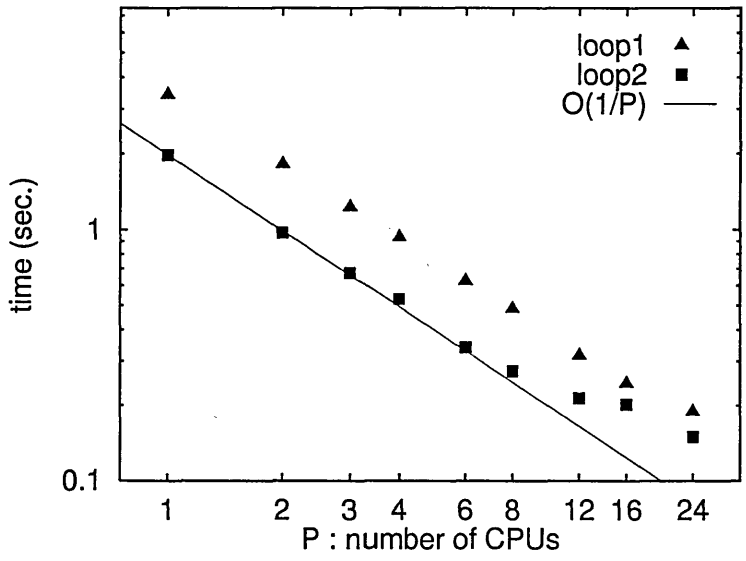

図 5: GP7000F での領域分割計算の効率

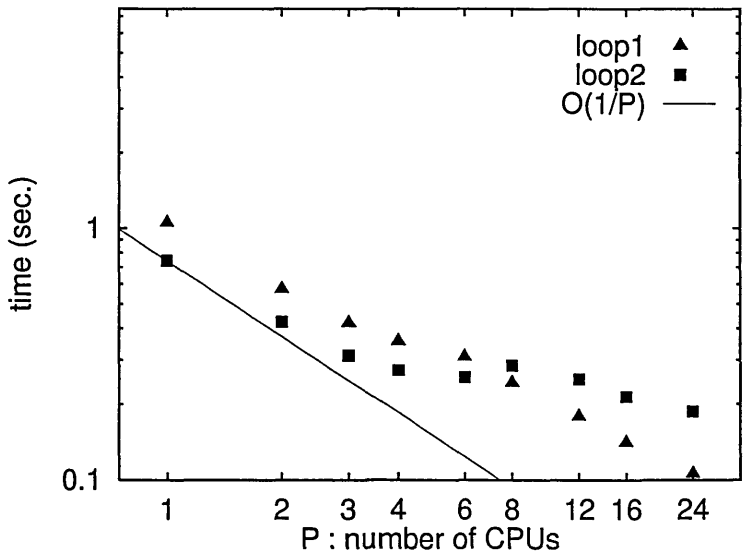

図 6: GS320 での領域分割計算の効率

種の構造があり, データの再利用を用いてキャッシュメモリーを利用できる場合には, 演算速度向上が得られることを確認した. また, OpenMP を用いた並列計算によっても高速化が実現できた.

共有メモリー型計算機と OpenMP を利用する並列計算は分散メモリー型の計算機を用いる場合と比較し, 並列コードへの書き換えとデバッグが比較的容易である. OpenMP による並列コードの実行性能の高い計算機が望まれる.

本研究は, 九州大学大学院数理学研究院 田端 正久教授との共同研究の成果の一部であり, 文部科学省 科学技術振興調整費 「全地球ダイナミクス」の援助を受けた.

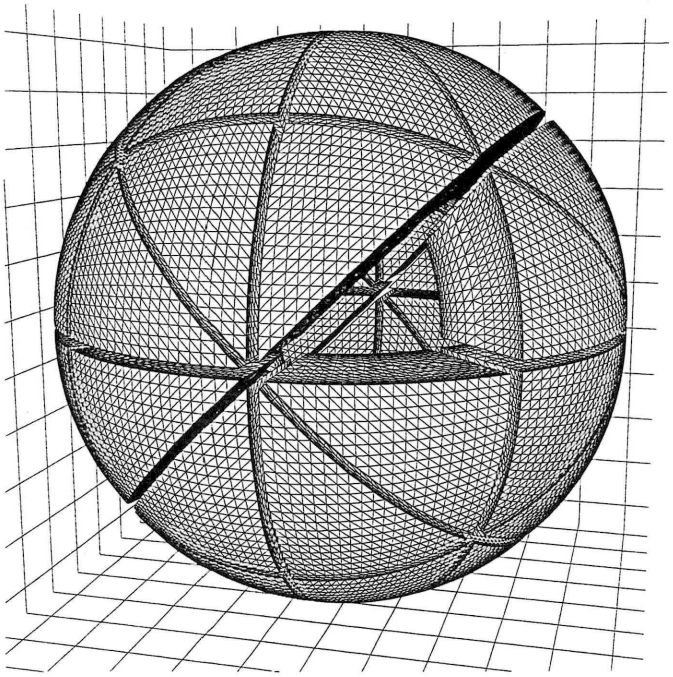

図 7: 球殻領域の合同な部分領域への分割

参考文献

[1] R. Barett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H .Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, 1994. 長谷川 里美, 長谷川 秀彦, 藤野 清次 訳, 応用数値計算ライブラリ 反復法 Templates, 朝倉 書店, 1996.

[2] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations, 3rd ed.* The John Hopkins University Press, Baltimore, 1996.

[4] *OpenMP C and C++ Application Program Interface Version 1.0 - Octorber 1998*, `http://www.openmp.org`.

[5] A. Suzuki, M. Tabata, and S. Honda, Numerical solution of an unsteady Earth's mantle convection by a finite element method, *Theoretical and Applied Mechanics*, **48** (1999), 371–378.

[6] M. Tabata and A. Suzuki, A stabilized finite element method for the Rayleigh-Bénard equations with infinite Prandtl number in a spherical shell, *Comp. Meth. Appl. Mech. Engrg.*, **190** (2000), 387–402.

[7] 鈴木 厚, 田端 正久, 合同な部分領域への分割による3次元球殻内熱対流問題の並列計算, 計算工学講演会論文集, **5** (2000年5月), 165–168.

[8] M. Tabata and A. Suzuki, Mathematical modeling and numerical simulation of Earth's mantle convection. to appear in *Proceedings of the International Symposium on Mathematical Modeling and Numerical Simulation in Continuum Mechanics, Lecture Notes in Computational Science and Engineering*, P. G. Ciarlet I. Babuska and T. Miyoshi (eds), Springer.

[9] O. C. Zienkiewicz and F. C. Scott, On the principle of repeatability and its application in analysis of turbine and pump impellers, *Int. J. Num. Meth. Eng.*, 4 (1972) 445-452.