OpenMP によるプログラミング入門（ I ）

中 島 研 吾

東京大学大学院理学系研究科地球惑星科学専攻

１．はじめに，本連載の概要

東京大学大学院理学系研究科地球惑星科学専攻では，2003 年度から 21 世紀 COE プログラム「多圏地球システムの進化と変動の予測可能性（観測地球科学と計算地球科学の融合拠点の形成）」（以下「多圏地球 COE」）を実施している。筆者が担当している「並列計算プログラミング」，「先端計算機演習 I・II」[1]は，2004 年度から多圏地球 COE の一環として開講されているもので，計算機科学を専門としない学生に対して科学技術シミュレーションのための並列プログラミング技術を体系的に教える講義・演習としては日本で初めての試みである。

毎年後期に開講される「先端計算機演習 II」では，OpenMP[2] に関する講義・演習を中心に実施している。OpenMP は対称共有メモリ型並列計算機（Symmetric Multiple Processors, SMP）におけるスレッド並列化のためにソースコードに挿入するディレクティヴ（directive，指示行）の共通規格である。Hitachi SR11000 の各ノードは SMP 型のアーキテクチュアであり，OpenMP はノード内の並列化に使用される。「先端計算機演習 II」では，東京大学情報基盤センター（以下「情報基盤センター」）の「教育利用」の制度を活用して，Hitachi SR11000　1 ノード（8CPU）を全受講者に対して利用させて頂いており，大きな教育効果をあげている（2004 年度は Hitachi SR8000 を使用）〔1〜3〕。

本連載は，**情報基盤センターの Hitachi SR11000 の利用者を対象として**，「先端計算機演習 II」で扱っている「ICCG 法による有限体積法向けポアソン方程式ソルバーの OpenMP による並列化」の内容を解説するものである。本連載の目的は，OpenMP の個々のディレクティヴや文法的な事項について解説することではなく，OpenMP を適用するために必要なプログラミングやデータ構造の考え方を伝えることである。

もともと，SMP におけるスレッド並列化のためのディレクティヴは各計算機会社によってばらばらであった。アメリカ合衆国エネルギー省で 1996 年より実施されている ASC プロジェクト[3]（Advanced Simulation and Computing，当初は「ASCI」という名称だった）で開発された様々なハードウェアでは，主として SMP クラスタ型のアーキテクチュアが採用された。それによって，プログラムの移植性のためにディレクティヴの共通化の必要性が生じ，OpenMP という規格が生まれる契機となった。近年は，マルチコアプロセッサの出現により，OpenMP を利用した並列プログラミングが更に普及し，文献〔4,5〕等の解説書，日本計算工学会の連載記事[4] などから様々な情報を得ることができる。

OpenMP については「MPI（Message Passing Interface）[5]と比べてプログラミングが簡単」，「並

---

[1] http://www-solid.eps.s.u-tokyo.ac.jp/~nakajima/class/
[2] http://www.openmp.org/
[3] http://www.llnl.gov/asc/
[4] 南里豪志「OpenMP 入門－マルチコア CPU 時代のプログラミング－」，日本計算工学会誌（2006〜2007）
[5] http://www.mpi-forum.org/

列を意識する必要が無い」とよく言われる。実際，FORTRAN や C などで記述されたソースコードにディレクティヴを挿入すればよいのだが，一筋縄ではいかないこともある。ディレクティヴの挿入による単純な並列化では，非常に計算時間を要したり，正しい答えを得られない場合もある。

本連載で取り扱う「有限体積法から導かれる疎行列を対象とした ICCG 法」はその一例で，メモリへの書き込みと参照が同時に起こるような「データ依存性（data dependency）」を回避するため，適切なデータの並べ替えを施す必要がある。このような場合の対策は OpenMP 向けの解説書でも詳しく取り上げられることは余り無い。本連載は，そのような OpenMP の言わば「暗黒面（dark side）」について解説するものである。

本連載は今回を含めて 3 回で以下のような内容である：

- 第 1 回：対象とするアプリケーションの概要と並列化における問題点
- 第 2 回：並列化のための並び替え（reordering）について
- 第 3 回：OpenMP による並列化

本連載で紹介するプログラム（FORTRAN および C）については下記のホームページから公開する。プログラムをダウンロードして，実際に動かしながら学習することによってより理解を深めて頂きたい：

http://www-solid.eps.s.u-tokyo.ac.jp/~nakajima/tutorial/sr11k/

株式会社ケイ・ジー・ティー殿のご厚意により，Windows 上で稼動する可視化ソフトウェア「MicroAVS」[6] を無料で利用できる。計算結果の可視化に利用して頂きたい。利用方法等は上記ホームページに掲載する。

既に述べたように，本連載の目的は，OpenMP の個々のディレクティヴについて解説することではないので，読者は前述の解説書等で情報を得るとともに，OpenMP による簡単なプログラミングを経験しておいて頂きたい。と言っても，本連載で登場するディレクティヴは以下の 2 種類（正確には 1 種類）であるので，これらについて知っていれば充分である：

```
!$omp parallel do private (ip,i)
#pragma omp parallel for private(ip,i)

!$omp parallel do private(ip,i) reduction(+:RHO)
#pragma omp parallel for private(ip,i) reduction(+:RHO)
```

本連載は，線形代数，行列計算に関する初歩的な知識を前提としている。紙面の都合もあり，詳細な説明は省いているので，文献〔6〕等を参照して補足して頂きたい。

[6] http://www.kgt.co.jp/feature/microavs/

２．プログラムのダウンロード，利用環境

連載の各回で解説するプログラム，関連ファイルは以下のような TAR ファイルとして，前述の本連載のホームページよりダウンロード可能である：

第 1 回：Lesson-1-C.tar（C，以下同様），Lesson-1-F.tar（FORTRAN，以下同様）
第 2 回：Lesson-2-C.tar, Lesson-2-F.tar
第 3 回：Lesson-3-C.tar, Lesson-3-F.tar

読者は，まず本連載用のディレクトリ（今後<$tutorial-top>とする）を作成し，そこへ上記の TAR ファイルをダウンロードし，解凍すると，以下のようなディレクトリが生成される：

<$tutorial-top>/Lesson-1 （今後<$L1>とする）
<$tutorial-top>/Lesson-2 （<$L2>）
<$tutorial-top>/Lesson-3 （<$L3>）

C と FORTRAN では同じ名前のディレクトリが生成されるので，両方を使用したい場合には 2 種類の<$tutorial_top>を準備し，それぞれに C と FORTRAN 用のファイルをダウンロードする必要がある。

プログラム類は，Hitachi SR11000 上でそのままコンパイル，リンク，実行ができるようになっているが，makefile を書き換えれば，他の環境でも実行は可能である。

注意点，問題点等は，本連載のホームページに随時掲載していく予定である。また，質問がある場合は中島（nakajima@eps.s.u-tokyo.ac.jp）まで遠慮なくご連絡頂きたい。

３．三次元ポアソン方程式ソルバーの概要

本連載で対象とするアプリケーション（以下「対象アプリケーション」）は図 1 に示す差分格子によってメッシュ分割された三次元領域において，以下のポアソン方程式を解くものである：

$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = f \qquad (1)$$

$$\phi = 0 @ z = z_{\max} \qquad (2)$$

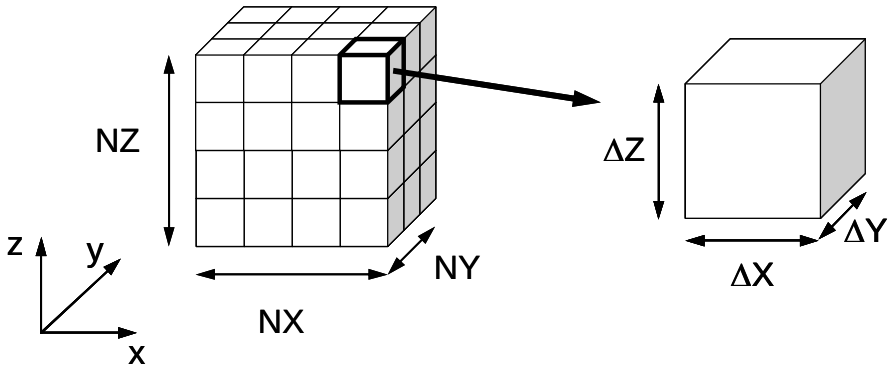


図 1: 三次元ポアソン方程式ソルバーの解析対象

差分格子の各メッシュは直方体（辺長さはΔX，ΔY，ΔZ），X，Y，Z 各方向のメッシュ数は NX，NY，NZ

ここで，式（1）の右辺の f は体積あたりのフラックス（flux，流束）項で，以下に示すような空間分布を有すると仮定している：

$$f = dfloat(i_0 + j_0 + k_0) \tag{3}$$

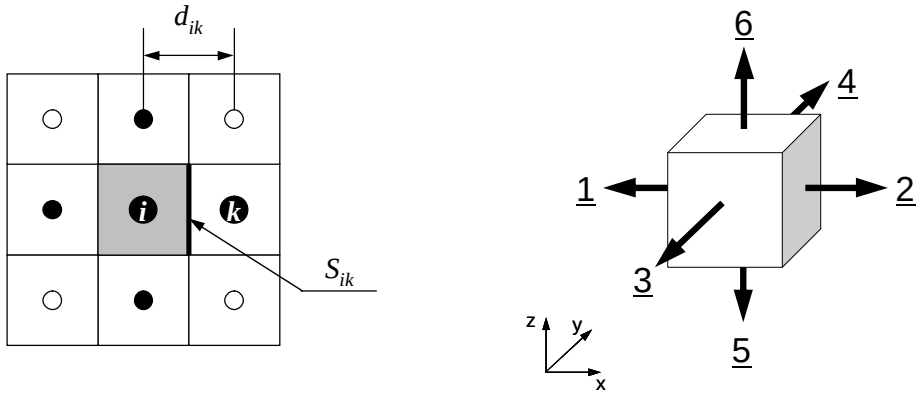
$$i_0 = XYZ(icel,1), \quad j_0 = XYZ(icel,2), \quad k_0 = XYZ(icel,3) \tag{4}$$

式（4）における $XYZ(icel, k)$（k=1,2,3）は X，Y，Z 方向の差分格子のインデックスで各メッシュが X，Y，Z 方向の何番目にあるかを示している。

形状は規則正しい差分格子であるが，プログラムの中では，一般性を持たせるために，有限体積法に基づき，非構造格子型のデータとして考慮する。図 2 における任意のメッシュ i の各面を通過するフラックスについて，式（1）により以下に示す式（5）のような釣り合い式が成立する：

$$\sum_{k=1}^{6}\left[\frac{S_{ik}}{d_{ik}}\left(\phi_k - \phi_i\right)\right] + V_i f_i = 0 \tag{5}$$

ここで，S_{ik}：メッシュ i と隣接メッシュ k 間の表面積，d_{ik}：メッシュ i-k 重心間の距離，V_i：メッシュ i の体積，f_i：メッシュ i の体積あたりフラックスである（図 2（a）参照）。三次元問題の場合，各直方体メッシュは 6 個の面を持っているため，隣接メッシュ数は（最大で）6 であり，式（5）左辺第一項は k=1~6 の和となっている。図 2（b）は，三次元問題における各メッシュの局所面番号（すなわち隣接メッシュの番号付けのルール）である（5. で詳しく説明する）。



（a）メッシュ i と隣接メッシュ k の関係　　（b）局所面番号，隣接メッシュナンバリング

図 2　隣接メッシュとの関係

式（5）を図 1 のような三次元領域に適用すると，メッシュ i について式（6）が得られる：

$$\frac{\phi_{k=1}-\phi_i}{\Delta x}\Delta y\Delta z + \frac{\phi_{k=2}-\phi_i}{\Delta x}\Delta y\Delta z + \frac{\phi_{k=3}-\phi_i}{\Delta y}\Delta z\Delta x + \frac{\phi_{k=4}-\phi_i}{\Delta y}\Delta z\Delta x + \frac{\phi_{k=5}-\phi_i}{\Delta z}\Delta x\Delta y + \frac{\phi_{k=6}-\phi_i}{\Delta z}\Delta x\Delta y = f_i\Delta x\Delta y\Delta z \tag{6}$$

これを整理すると，式（7）のようになり，三次元差分法の場合と同様の式が得られる：

$$\frac{\phi_{k=1}-2\phi_i+\phi_{k=2}}{\Delta x^2}+\frac{\phi_{k=3}-2\phi_i+\phi_{k=4}}{\Delta y^2}+\frac{\phi_{k=5}-2\phi_i+\phi_{k=6}}{\Delta z^2}=f_i \tag{7}$$

式（5）にもどって，これを整理すると，式（8）が得られる：

$$\left[\sum_{k=1}^{6}\frac{S_{ik}}{d_{ik}}\right]\phi_i-\left[\sum_{k=1}^{6}\frac{S_{ik}}{d_{ik}}\phi_k\right]=+V_i f_i \tag{8}$$

これは各メッシュ i について成立する式であるので，全メッシュ数を N とすると，N 個の方程式を連立させて，境界条件を適用し，連立一次方程式 $[A]\{\phi\}=\{b\}$ を解くことに帰着される。式（8）の左辺第一項は[A]の対角項，第二項は非対角項，式（8）の右辺は{b}に対応している。式（8）からもわかるように，各メッシュ i に対応する非対角成分の数は最大 6 個であるので，係数行列[A]は疎（sparse）な行列となる。

４．ICCG 法について

式（8）を連立させて得られる連立一次方程式$[A]\{\phi\}=\{b\}$を解く方法として，本稿では反復法（iterative method），特に Krylov 部分空間法といわれる，非定常型の反復法を使用する〔6〕。係数行列[A]は，式（7）からも類推されるように，対称かつ正定（symmetric positive definite）であり，このような行列に対しては，通常は共役勾配法（conjugate gradient method，CG 法）が使用される〔6〕。Krylov 部分空間法の収束は係数行列の性質（固有値分布）に強く依存するため，前処理を適用して固有値が 1 の周辺に集まるように行列の性質を改善する。前処理行列を[M]とすると，$[\tilde{A}]=[M]^{-1}[A]$，$\{\tilde{b}\}=[M]^{-1}\{b\}$として（$[M]^{-1}$ を右から乗ずる場合もある），すなわち $[\tilde{A}]\{\phi\}=\{\tilde{b}\}$ という方程式を代わりに解くことになる。$[M]^{-1}$ が$[A]^{-1}$ をよく近似した行列であれば，$[\tilde{A}]=[M]^{-1}[A]$は単位行列に近くなり，それだけ解きやすくなる。前処理付き CG 法のアルゴリズムの概要は以下のようになる：

```
compute {r}(0)={b}-[A]{x}(0)
for i= 1, 2, …
    solve [M]{z}(i-1)={r}(i-1)
    ρi-1= {r}(i-1) {z}(i-1)
    if i=1
      {p}(1)={z}(0)
     else
      βi-1= ρi-1/ρi-2
      {p}(i)={z}(i-1) + βi-1{z}(i)
    endif
    {q}(i)=[A]{p}(i)
    αi = ρi-1/{p}(i){q}(i)
    {x}(i)={x}(i-1) + αi{p}(i)
    {r}(i)={r}(i-1) – αi{q}(i)
    check convergence |r|
end
```

前処理手法として広く使用されているのは，不完全 LU 分解（Incomplete LU factorization，ILU）である。「完全」LU 分解は[*A*]=[*L*][*U*]のように係数行列を上下三角行列の積に分解し，前進後退代入によって連立一次方程式の解を求める直接法（direct method）の一種である。前項の最後で述べたように，本対象アプリケーションの場合は[*A*]は疎な行列であるが，[*L*]，[*U*]は必ずしもそうではなく，もともと 0 であったところに非ゼロ成分（fill-in）が生じる場合もある。不完全 LU 分解ではこの fill-in のレベルや数を制御して，前処理行列 $[M]=[\tilde{L}][\tilde{U}](\approx[A])$ を生成する。実用的には，ILU(0)（カッコ内は fill-in のレベル），すなわち fill-in を全く考慮しない場合でも，広範囲の問題に対応できる。ILU(0)では，元の行列[*A*]と前処理行列[*M*]の非ゼロ成分の位置が同じとなる。

対称行列における LU 分解に相当するものとして，コレスキー分解（Cholesky factorization）がある。本稿では，コレスキー分解の一種である，修正コレスキー分解（modified Cholesky factorization）を適用する。修正コレスキー分解は，通常のコレスキー分解[*A*]=[*L*][*L*]^T の代わりに以下に示すような[*A*]=[*L*][*D*][*L*]^T を用いる手法である。

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix} \begin{bmatrix} d_1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 \\ 0 & 0 & 0 & 0 & d_5 \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} & l_{41} & l_{51} \\ 0 & l_{22} & l_{32} & l_{42} & l_{52} \\ 0 & 0 & l_{33} & l_{43} & l_{53} \\ 0 & 0 & 0 & l_{44} & l_{54} \\ 0 & 0 & 0 & 0 & l_{55} \end{bmatrix}$$

$$\sum_{k=1}^{i} l_{ik} \cdot d_k \cdot l_{jk} = a_{ij} \quad (j = 1,2,\cdots,i-1) \tag{9}$$

$$\sum_{k=1}^{i} l_{ik} \cdot d_k \cdot l_{ik} = a_{ii} \tag{10}$$

ここで $l_{ii} \cdot d_i = 1$ とすると以下が成立する：

$$\left\{ \begin{array}{l} i = 1,2,\cdots,n \\ \quad \left\{ \begin{array}{l} j = 1,2,\cdots,i-1 \\ \quad l_{ij} = a_{ij} - \displaystyle\sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} \\ \quad d_i = \left(a_{ii} - \displaystyle\sum_{k=1}^{i-1} {l_{ik}}^2 \cdot d_k \right)^{-1} = {l_{ii}}^{-1} \end{array} \right. \end{array} \right. \tag{11}$$

本稿では，fill-in を考慮しない以下のような不完全（修正）コレスキー分解（Incomplete (modified) Cholesky factorization，IC）を適用し，前処理行列の成分 $\tilde{d}_i, \tilde{l}_{ij}$ を計算する（fill-in を考慮しないことから正確には IC(0)とするべきである）。すなわち，$a_{ij} \neq 0$ の場合にのみ $\tilde{l}_{ij} = a_{ij} - \sum_{k=1}^{j-1} \tilde{l}_{ik} \cdot \tilde{d}_k \cdot \tilde{l}_{jk}$ を計算している：

$$
\begin{cases}
i = 1,2,\cdots,n \\
\quad \begin{cases}
\quad j = 1,2,\cdots,i-1 \\
\quad \tilde{l}_{ij} = a_{ij} - \sum_{k=1}^{j-1} \tilde{l}_{ik} \cdot \tilde{d}_k \cdot \tilde{l}_{jk} \quad if\ a_{ij} \neq 0 \\
\quad \tilde{d}_i = \left(a_{ii} - \sum_{k=1}^{i-1} \tilde{l}_{ik}^{\,2} \cdot \tilde{d}_k \right)^{-1} = \tilde{l}_{ii}^{\,-1}
\end{cases}
\end{cases}
\tag{12}
$$

$\tilde{l}_{ij} = a_{ij} - \sum_{k=1}^{j-1} \tilde{l}_{ik} \cdot \tilde{d}_k \cdot \tilde{l}_{jk}$ **の部分は非常に計算時間を要するプロセスであるため** $\tilde{l}_{ij} = a_{ij}$ **と更に省略した形で計算が実施される場合もある。実は，対象アプリケーションの場合，図** 3 **に示すように，** $\tilde{l}_{ij} = a_{ij} - \sum_{k=1}^{j-1} \tilde{l}_{ik} \cdot \tilde{d}_k \cdot \tilde{l}_{jk}$ **において** $\tilde{l}_{ik}$ **か** $\tilde{l}_{ik}$ **のいずれかが必ず** 0 **となるため，** $\tilde{l}_{ij} = a_{ij}$ **となる。すなわち，前処理行列において非対角成分は元の行列** [*A*] **と同じであるため，** $\tilde{d}_i = \left(a_{ii} - \sum_{k=1}^{i-1} \tilde{l}_{ik}^{\,2} \cdot \tilde{d}_k \right)^{-1} = \tilde{l}_{ii}^{\,-1}$ **を計算するだけでよい。**

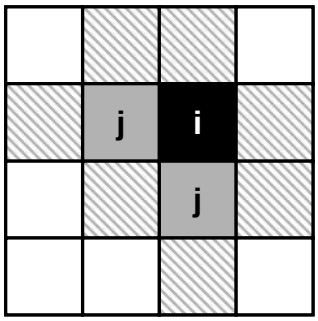


図 3　**二次元格子における不完全（修正）コレスキー分解**

斜線を施したメッシュが *i* **または** *j* **に接続する** *k* **の候補であるが，** *i* **と** *j* **に同時に接続するメッシュは存在しない。三次元の場合も同様である。**

対象アプリケーションでは，このような不完全（修正）コレスキー分解（IC**）による前処理手法を適用した共役勾配法（**CG **法），すなわち** ICCG **法を使用する。**

前処理付き CG **法の** $[M]\{z\} = [\tilde{L}\ \tilde{D}\ \tilde{L}^T]\{z\} = \{r\}$ **を解く（** $\{z\} = [LDL^T]^{-1}\{r\}$ **）部分では：**

- **前進代入**　$[\tilde{L}]\{y\} = \{r\}$
- **後退代入**　$[\tilde{D}\ \tilde{L}^T]\{z\} = \{y\}$

のようにして，{*z*}**を計算する。**

５．プログラムのコンパイル，実行方法

Lesson-1-C.tar，Lesson-1-F.tar を<$tutorial-top>において解凍すると，以下のディレクトリが得られる：

```
<$tutorial-top>/Lesson-1/run = <$L1>/run
<$tutorial-top>/Lesson-1/src = <$L1>/src
```

続いて，以下のコマンドを実行し，<$L1>/run/L1-sol という実行形式が生成されていることを確認する：

```
$> cd <$L1>/src
$> make
$> ls ../run/L1-sol
```

計算を実施するためには，形状データ「mesh.dat」と制御データ「INPUT.DAT」が必要である。まず，形状データを作成するためのメッシュジェネレータ mg を生成する：

```
$> cd <$L1>/run
$> f90 –O mg.f –o mg  または cc –O mg.c –o mg
```

続いて：

```
$> ./mg
```

とすると入力待ちの状態になるので，ここで図 1 の NX，NY，NZ に相当する数を入力すると同じディレクトリ内に「mesh.dat」というファイルが生成される。例えば NX=4, NX=3, NX=2 と入力した場合の「mesh.dat」の中身は以下のようになる：

```
 4   3   2
24
 1   0   2   0   5   0  13   1   1   1
 2   1   3   0   6   0  14   2   1   1
 3   2   4   0   7   0  15   3   1   1
 4   3   0   0   8   0  16   4   1   1
 5   0   6   1   9   0  17   1   2   1
 6   5   7   2  10   0  18   2   2   1
 7   6   8   3  11   0  19   3   2   1
 8   7   0   4  12   0  20   4   2   1
 9   0  10   5   0   0  21   1   3   1
10   9  11   6   0   0  22   2   3   1
11  10  12   7   0   0  23   3   3   1
12  11   0   8   0   0  24   4   3   1
13   0  14   0  17   1   0   1   1   2
14  13  15   0  18   2   0   2   1   2
15  14  16   0  19   3   0   3   1   2
16  15   0   0  20   4   0   4   1   2
17   0  18  13  21   5   0   1   2   2
18  17  19  14  22   6   0   2   2   2
19  18  20  15  23   7   0   3   2   2
20  19   0  16  24   8   0   4   2   2
21   0  22  17   0   9   0   1   3   2
22  21  23  18   0  10   0   2   3   2
23  22  24  19   0  11   0   3   3   2
24  23   0  20   0  12   0   4   3   2
```

1 行目の「4 3 2」は「NX，NY，NZ」に相当する。2 行目の「24」は「4×3×2」すなわち全メッシュ数である。3 行目以降は各メッシュとその隣接メッシュの情報である。各行ごとに 10 列がありそれぞれ以下のような内容となっている：

- **第 1 列：メッシュ番号（*i*，通し番号）**
- **第 2 列～7 列：隣接メッシュ番号（*NEIBcell(i,k)*, *k*=1~6）**
- **第 8 列～10 列：各方向の差分格子のインデックス（*XYZ(ik)*, *k*=1~3，式（4）参照）**

隣接メッシュ番号は，*NEIBcell(i,k)*（*k*=1~6）という配列に格納されている。6 つの隣接メッシュは，図 2（b）のように定義されている。境界面を持つため隣接しているメッシュが無い場合は 0 が入っている。各隣接メッシュ番号は *i* に対して以下のような関係になる。

- ***NEIBcell(i,1)= i-1*，*NEIBcell(i,2)= i+1***
- ***NEIBcell(i,3)= i-NX*，*NEIBcell(i,4)= i+NX***
- ***NEIBcell(i,5)= i-NX*×*NY*，*NEIBcell(i,6)= i+ NX*×*NY***

図 4 は 15 番のメッシュの隣接メッシュ番号について示したものである。前掲の「mesh.dat」の中身と比較されると良いであろう（*NEIBcell(15,3)*, *NEIBcell(15,6)*は 3 番目と 6 番目の面が境界面であるため値が 0 となっている）。

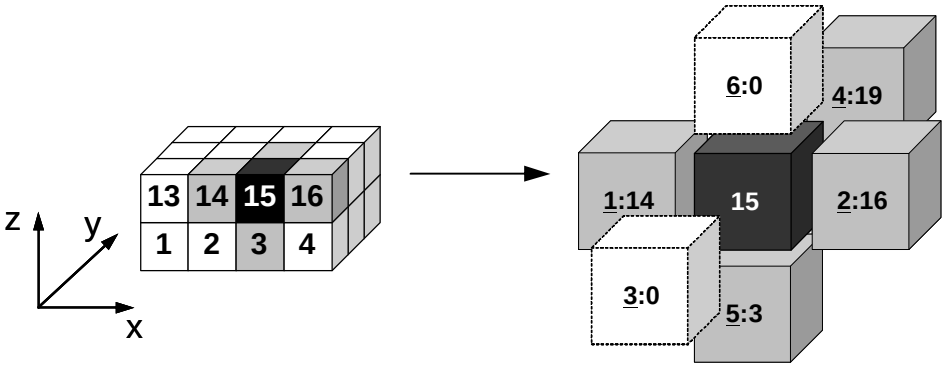


図 4　15 番メッシュの隣接メッシュ

対象アプリケーションでは規則的な差分格子を扱っているため，*XYZ(i,k)*の情報があれば，必然的に隣接メッシュは決まるのであるが，本稿では，非構造格子への拡張を考慮して，敢えて陽的に隣接メッシュの情報を生成する。

計算を実施する前に制御データ「INPUT.DAT」を編集する。「INPUT.DAT」の内容は自由形式で以下のようになっている：

変数名	型	内　容
METHOD	整数	前処理手法（=1~3）（後述）
DX，DY，DZ	倍精度実数	各メッシュの 3 辺の長さ（図 1 のΔX，ΔY，ΔZ）
OMEGA	倍精度実数	不使用（適当な値を入れておく）
EPSICCG	倍精度実数	収束判定値

前処理手法は以下の 3 種類がある：

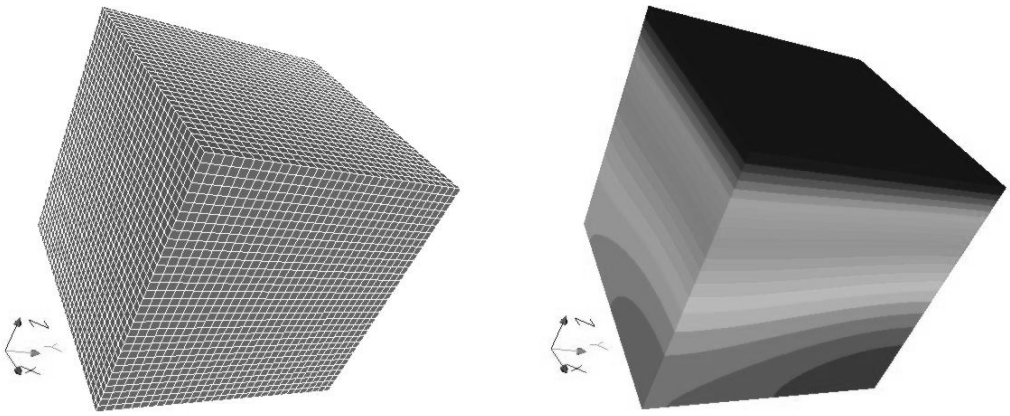
前処理手法	内　容
METHOD=1	式（12）で $\tilde{l}_{ij} = a_{ij}$ とした場合
METHOD=2	式（12）で $\tilde{l}_{ij} = a_{ij} - \sum_{k=1}^{j-1} \tilde{l}_{ik} \cdot \tilde{d}_k \cdot \tilde{l}_{jk}$ をまじめに計算する場合
METHOD=3	前処理行列として係数行列[*A*]の対角成分を使用する場合（対角スケーリング，点ヤコビと呼ばれる）

基本的には METHOD=1 の場合が中心であるが，比較のため他の手法についてもテストできるようになっている。以下では，主に METHOD=1 の場合を中心として説明する。

続いて，ディレクトリ<$L1>/run に形状データ「mesh.dat」と制御データ「INPUT.DAT」が存在することを確認して，計算を実行する：

```
$> cd <$L1>/run
$> ./L1-sol
```

画面には収束履歴が 100 回の反復ごとに表示され，計算終了後，<$L1>/run に「test.inp」というファイルが出力される。これを前述の「MicroAVS」で読み込み結果を表示することができる（図 5 参照）。



メッシュ分割図　　計算結果分布図

図 5　計算結果の MicroAVS による表示（32^3=32,768 メッシュ）

６．プログラムの処理内容

続いてプログラムの処理内容を簡単に説明する。図 6 は本プログラムのサブルーチン構成である。

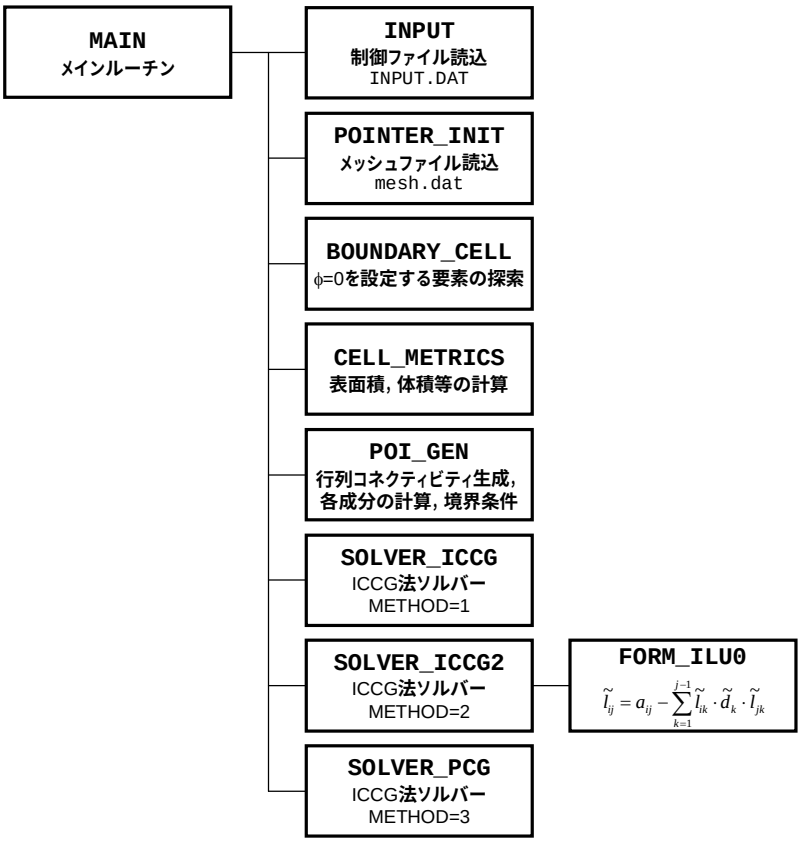


図 6　サブルーチン構成

行列は対称であるが，対角成分，上三角成分，下三角成分に分けて記憶している。また，非ゼロ成分のみを CRS（compressed row storage）〔6〕法によって記憶している。各成分は以下のように記憶されている：

配列名	型	内 容
D(i)	倍精度実数	対角成分，i=1~N（N：全メッシュ数）
B(i)	倍精度実数	右辺ベクトル，i=1~N
INL(i)	整数	*i* の下三角成分数，i=1~N
INU(i)	整数	*i* の上三角成分数，i=1~N
IAL(j,i)	整数	*i* の *j* 番目の下三角成分に対応する列番号，i=1~N, j=1~INL(i)
IAU(j,i)	整数	*i* の *j* 番目の上三角成分に対応する列番号，i=1~N, j=1~INU(i)
AL(j,i)	倍精度実数	*i* の *j* 番目の下三角成分，i=1~N，j=1~INL(i)
AU(j,i)	倍精度実数	*i* の *j* 番目の上三角成分，i=1~N，j=1~INU(i)

例えば，CG 法に現れる[*A*]{*p*}={*q*}の計算は上記配列により，以下のように実施されている。

```
do i= 1, N
  VAL= D(i)*p(i)
  do j= 1, INL(i)
    VAL= VAL + AL(j,i)*p(IAL(j,i))
  enddo
  do j= 1, INU(i)
    VAL= VAL + AU(j,i)*p(IAU(j,i))
  enddo
  q(i)= VAL
enddo
```

ここでは FORTRAN の場合を示したが，C の場合は二重配列のメモリ上でのアクセスの順番が異なるため，添え字の参照の順番が逆になっている。

ICCG 法の処理内容は，大きく分けて以下の 4 種類である：

① ベクトルの内積
② ベクトルの実数倍の加減
③ 行列ベクトル積
④ 前処理（前処理行列生成，前進後退代入）

このうち①～③については OpenMP のディレクティヴを挿入するだけで容易に並列化可能である：

① ベクトル内積

```
RHO= 0.d0
do i= 1, N
  RHO= RHO + r(i)*z(i)
enddo
```

```
  RHO= 0.d0
!$omp  parallel do private(i)
!$omp & reduction(+:RHO)
  do i= 1, N
    RHO= RHO + r(i)*z(i)
  enddo
```

② ベクトルの実数倍の加減

```
do i= 1, N
  p(i)= z(i) + BETA*p(i)
enddo
```

```
!$omp  parallel do private(i)
  do i= 1, N
    p(i)= z(i) + BETA*p(i)
  enddo
```

③ 行列ベクトル積

```
do i= 1, N
  VAL= D(i)*p(i)
  do j= 1, INL(i)
    VAL= VAL + AL(j,i)*p(IAL(j,i))
  enddo
  do j= 1, INU(i)
    VAL= VAL + AU(j,i)*p(IAU(j,i))
  enddo
  q(i)= VAL
enddo
```

```
!$omp  parallel do
!$omp & private(i,j,VAL)
  do i= 1, N
    VAL= D(i)*p(i)
    do j= 1, INL(i)
      VAL= VAL + AL(j,i)*p(IAL(j,i))
    enddo
    do j= 1, INU(i)
      VAL= VAL + AU(j,i)*p(IAU(j,i))
    enddo
    q(i)= VAL
  enddo
```

しかし，前処理行列生成部，前進後退代入の部分はこのように単純ではない。例えば，$\tilde{d}_i = \left(a_{ii} - \sum_{k=1}^{i-1} \tilde{l}_{ik}^{\,2} \cdot \tilde{d}_k \right)^{-1} = \tilde{l}_{ii}^{\,-1}$を計算している部分は以下のようになる（$\tilde{d}_i \Rightarrow$DD(i)）。

```
do i= 1, N
  VAL= D(i)
  do j= 1, INL(i)
    VAL= VAL - (AL(j,i)**2) * DD(IAL(j,i))
  enddo
  DD(i)= 1.d0/VAL
enddo
```

また，前進後退代入（$[M]\{z\} = [\tilde{L}\ \tilde{D}\ \tilde{L}^T]\{z\} = \{r\}$を解く部分）は以下のようになる：

```
do i= 1, N
  z(i)= r(i)
enddo
```

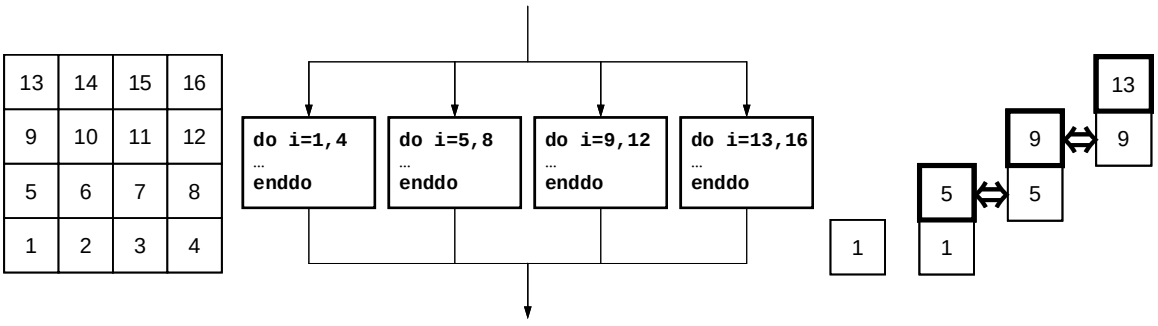
前進代入：$[\tilde{L}]\{y\} = \{r\}$

```
do i= 1, N
  WVAL= z(i)
  do j= 1, INL(i)
    WVAL= WVAL -  AL(j,i) * z(IAL(j,i))
  enddo
  z(i)= WVAL * DD(i)
enddo
```

後退代入：$[\tilde{D}\ \tilde{L}^T]\{z\} = \{y\}$

```
do i= N, 1, -1
  SW  = 0.0d0
  do j= 1, INU(i)
    SW= SW + AU(j,i) * z(IAU(j,i))
  enddo
  z(i)= z(i) - DD(i)*SW
enddo
```

これらの処理では，DD，z を計算するためのループで，右辺に他のメッシュにおける DD，z が参照されている。図 7（a）のような 16 メッシュから成る領域において，図 7（b）のように 4 スレッドを使用した並列計算を実施しようとしても，例えば前進代入部分では，図 7（c）において⇔で示されたメッシュに関して，メモリへの書き込みと参照が同時に生じ，データ依存性が発生する可能がある（後退代入，前処理行列生成部も同様）。



（a）計算対象　　（b）4 スレッドによる並列計算　　（c）前進代入におけるデータ依存性

図 7　前進代入において並列計算ができない例

このようなデータ依存性を持つループを OpenMP で強制的に並列化すると，非常に長い計算時間を要したり，正しい計算結果を得られない場合がある。並列計算を実現するためには，こうしたデータ依存性の除去，すなわち右辺で参照される DD(k)や z(k)の内容が，DD(i)や z(i)を計算している間に「絶対に変わらない」ことを保証する必要がある。

メッシュ間の接続に関するグラフ情報をもとに，メッシュを並び替える（reordering）ことによって，データ依存性を除去することが可能である〔7〕。次号では，reordering の手法と実装について説明する。

（次号に続く）

参 考 文 献

[1] 中島研吾（2005），Hitachi SR8000 を利用した並列プログラミング教育：東京大学 21 世紀 COE プログラム 多圏地球システムの進化と変動の予測可能性，スーパーコンピューティングニュース（東京大学情報基盤センター） 7-5, 21-28

[2] 中島研吾（2006），Hitachi SR11000 を利用した並列プログラミング教育：東京大学 21 世紀 COE プログラム 多圏地球システムの進化と変動の予測可能性，スーパーコンピューティングニュース（東京大学情報基盤センター） 8-3, 21-28

[3] 中島研吾（2007），Hitachi SR11000 を利用した並列プログラミング教育：東京大学 21 世紀 COE プログラム 多圏地球システムの進化と変動の予測可能性，スーパーコンピューティングニュース（東京大学情報基盤センター） 9-3, 21-30

[4] Chandra, R.他（2001），Parallel Programming in OpenMP，Morgan Kaufmann Publishers

[5] 牛島省（2006），OpenMP による並列プログラミングと数値計算法，丸善

[6] Dongarra, J.J. et al.（1994）, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM（邦訳：長谷川里美, 長谷川秀彦, 藤野清次（1996），反復法 Templates，朝倉書店）

[7] Nakajima, K. (2005), Three-Level Hybrid vs. Flat MPI on the Earth Simulator: Parallel Iterative Solvers for Finite-Element Method, Applied Numerical Mathematics 54, 237-255