OpenMP によるプログラミング入門（Ⅱ）

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：データ依存性について

前回は，有限体積法によって離散化された三次元領域において，ポアソン方程式を ICCG 法によって解くプログラムについて紹介した。リスト 1 に示すような前処理付き CG 法（共役勾配法，Conjugate Gradient Method）について，不完全修正コレスキー分解（Incomplete modified Cholesky factorization）を適用したとき，前進後退代入（$[M]\{z\}=[\tilde{L}\,\tilde{D}\,\tilde{L}^T]\{z\}=\{r\}$ を解く部分）はリスト 2 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
```

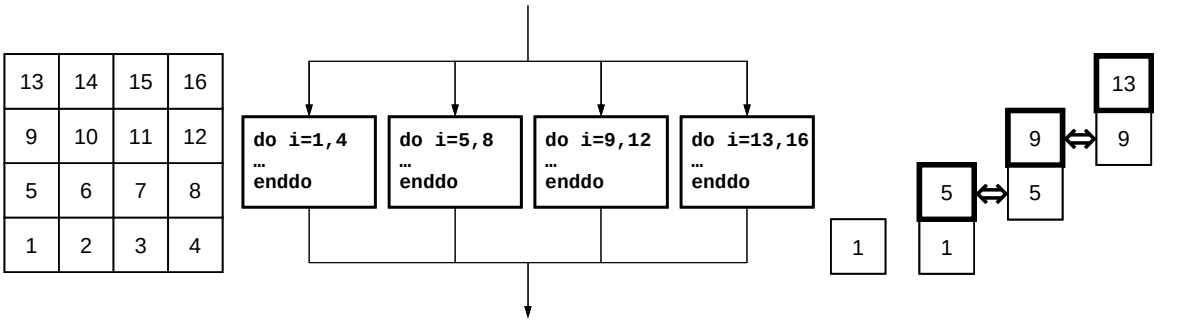
リスト 1　前処理付き CG 法（共役勾配法，Conjugate Gradient Method）のアルゴリズム

```
    do i= 1, N
      z(i)= r(i)
    enddo
前進代入：[L̃]{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：[D̃ L̃T]{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
```

リスト 2　前進後退代入

これらの処理では z(i)を計算するためのループで，右辺に他の要素における z(IAL(j,i))，z(IAU(j,i))が参照されている。図 1（a）のような 16 要素（メッシュ）から成る領域におい

て，図 1（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 1（c）において⇔で示された要素に関して，メモリへの書き込みと参照が同時に生じ，「データ依存性」が発生する可能性がある（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

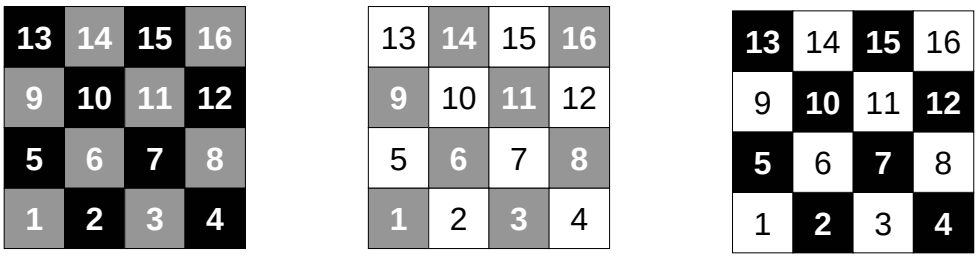
図 1　前進代入において並列計算ができない例

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 `z(IAL(j,i))`（あるいは `z(IAU(j,i))`）の内容が，左辺で `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。このような場合，要素間の接続に関する情報をもとに，要素を並び替える（reordering，oredering）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〔1〕。今回は，「並び替え」の手法と実装について説明する。

2. Ordering について

(1) Red-Black Ordering, Multicolor Ordering

並列計算のためには，データ依存性の除去が必要であり，そのためには，図 1 に示した前進代入の場合を例にとると，右辺で参照される `z(IAL(j,i))`の内容が，`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。対策として広く用いられているのは，「グラフ理論」における「彩色，色づけ（coloring)」の手法を適用して，各要素を，依存性を持たない互いに独立な要素ごとに分類する手法である〔2〕。図 1 に示した 16 要素の領域は図 2（a）に示すように「2 色」に「彩色」できる。このような彩色手法を，チェッカーボードにちなんで，「red-black coloring」と呼ぶ。「red」に彩色された要素｛`1,3,6,8,9,11,14,16`｝(図 2 (b)，白黒印刷なので「gray」）は，お互いに隣接しておらず，依存関係を持たない独立な要素群である。「black」に彩色された残りの要素（図 2（c））についても同様である。



（a）red-black coloring　　（b）red に彩色された要素　　（b）black に彩色された要素

図 2　Red-Black Coloring の例

ここで,

```
character*5 COLOR(N)
  COLOR(i)= ‘RED   ’ (if i= 1,3,6,8, 9,11,14,16)
  COLOR(i)= ‘BLACK’ (if i= 2,4,5,7,10,12,13,15)
```

なる配列 COLOR(:)を導入すると，リスト 2 に示した前進代入の部分はリスト 3 のように書き換えることによって，データ依存性を持たないループとすることができる：

```
      do i= 1, N
        if (COLOR(i).eq.’RED  ’) then
          WVAL= z(i)
          do j= 1, INL(i)
            WVAL= WVAL -  AL(j,i) * z(IAL(j,i))
          enddo
          z(i)= WVAL * DD(i)
        endif
      enddo

      do i= 1, N
        if (COLOR(i).eq.’BLACK’) then
          WVAL= z(i)
          do j= 1, INL(i)
            WVAL= WVAL -  AL(j,i) * z(IAL(j,i))
          enddo
          z(i)= WVAL * DD(i)
        endif
      enddo
```

リスト 3　前進代入（red-black coloring によりデータ依存性を除去）

図 2 からもわかるように，前進後退代入では「red」の計算をしている間は，右辺には「black」の要素のみが現れ，逆に「black」の計算をしている間は，右辺には「red」の要素のみが現れる。ある「色（red，black)」に属する要素の計算中は，同じ「色」に属する要素は右辺では参照されないことになり，データ依存性が回避される。**従って，同じ「色」に属する要素に関する計算は並列に実施することができる。**

実際の計算では，図 3 に示すように要素を「色」の順に並び替える（ordernig，reordering）とプログラムも簡便になり，計算効率も高くなる（リスト 4)。リスト 4 の COLORindex(:)は各「色」に含まれている要素数を表すための一次元圧縮配列である。図 3 の例の場合は,

```
COLORindex(0)= 0
COLORindex(1)= 8
COLORindex(2)=16
```

であり, 1～8 番目の要素が第 1 色(red)に属し(COLORindex(0)+1=1, COLORindex(1)=8)，9 番目～16 番目の要素が第 2 色（black）に属している（COLORindex(1)+1=9，COLORindex(2)=16)。このように，データ依存性を排除するには,「彩色（coloring)」と「並び替え（ordering)」を併用した手法が有効である。一般の複雑な形状に対しては，互いに独立な要素群を抽出するためには，2 色より多い色数が必要となるため，多数の色を使用した multicolor ordering（MC 法）が使用される。ここで示した red-black ordering は色数＝2 で，MC 法のうち，最も単純な形状に対して適用される特別なケースである。

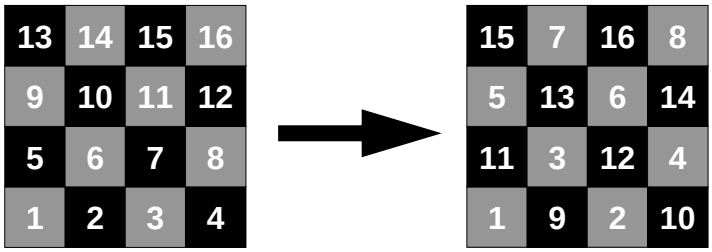


図 3　Red-Black Ordering の例

```
      do icol= 1, 2
        do i= COLORindex(icol-1)+1, COLORindex(icol)
          WVAL= z(i)
          do j= 1, INL(i)
            WVAL= WVAL -  AL(j,i) * z(IAL(j,i))
          enddo
          z(i)= WVAL * DD(i)
        enddo
      enddo
```

リスト 4　前進代入（red-black ordering の導入）

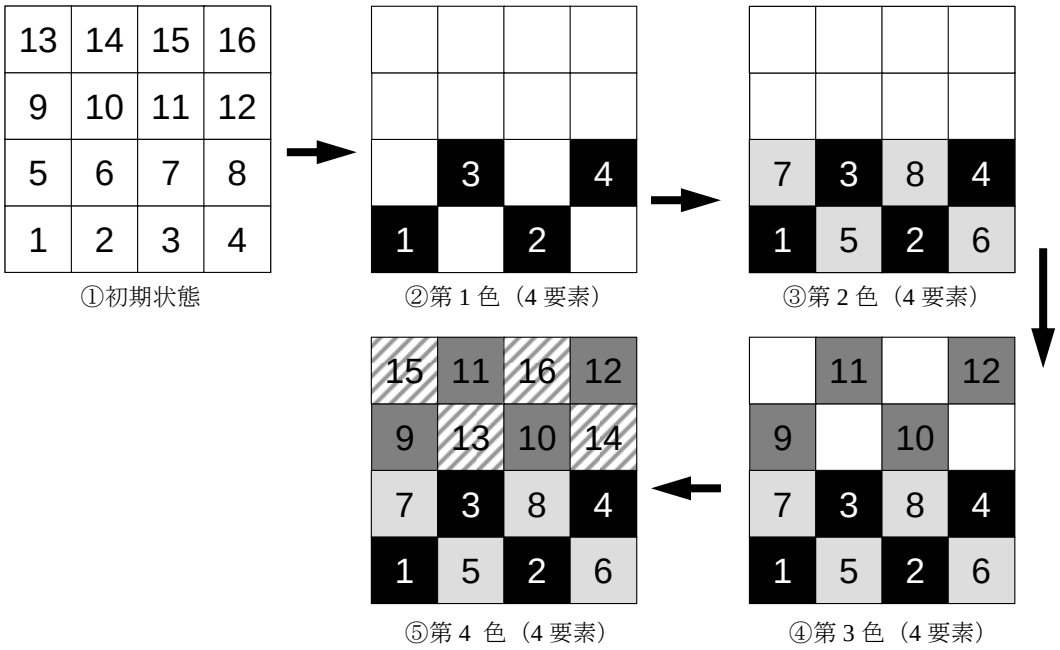


図 4　Multicolor Ordering の例（4 色）

図 4 は 4 色による MC 法の適用例である。16/4=4 であるので，各色に 4 要素が割り当てられるものとして，互いに独立な要素を 4 つずつ，初期状態における若い番号の要素から順番に選び出している。MC 法の基本的なアルゴリズムを以下に示す：

アルゴリズム 1：MC 法の基本的なアルゴリズム

① 「要素数÷色数」を「m」とする。
② 初期要素番号が若い順に互いに独立な要素を「m」個選び出して彩色し，次の色へ進む。
③ 指定した色数に達して，全ての要素が彩色されるまで②を繰り返す。
④ 色番号の若い順番に要素を再番号づけする（各色内では初期要素番号が若い順）。

実際は，要素数が色数で割り切れない場合など，様々な例外処理が必要となる。そのような場合の実装については，4. で解説する。

(2) Cuthill-McKee Ordering, Reverse Cuthill-McKee Ordering

Cuthill-McKee Ordering（以下 CM 法）, Reverse Cuthill-McKee Ordering（RCM 法）〔2,3〕は元々，疎行列のバンド幅やプロフィルを減少させることで，ガウスの消去法や LU 分解における fill-in を減少させ，効率的な計算を実施するための手法である〔2,3〕。

「並び替え（ordering)」は，本稿のように独立な要素を抽出するだけでなく，様々な用途に使用されている。文献〔3〕には様々な ordering 手法とその用途について説明されている。

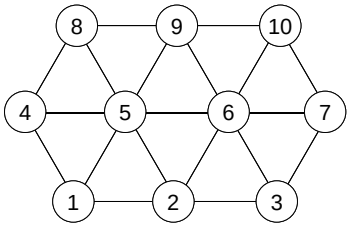


図 5 グラフの例

CM 法は図 5 に示すようなグラフ（点群とそれらを結ぶ辺の集合）において，点を分類する手法である。点群をレベル（level）という，coloring における「色」に相当する考え方に基づき分類している。以下に CM 法の概要を示す（図 6）：

アルゴリズム 2：CM 法の基本的なアルゴリズム

① 各点に隣接する点の数を「次数（degree)」，最小次数の点を「レベル=1」の点とする。
② 「レベル=k-1」に属する点に隣接する点を「レベル=k」とする。全ての点にレベルづけがされるまで「k」を 1 つずつ増やして繰り返す。
① 全ての点がレベルづけされたら，レベルの順番に再番号づけする（各レベル内では「次数」の番号が少ない順)。

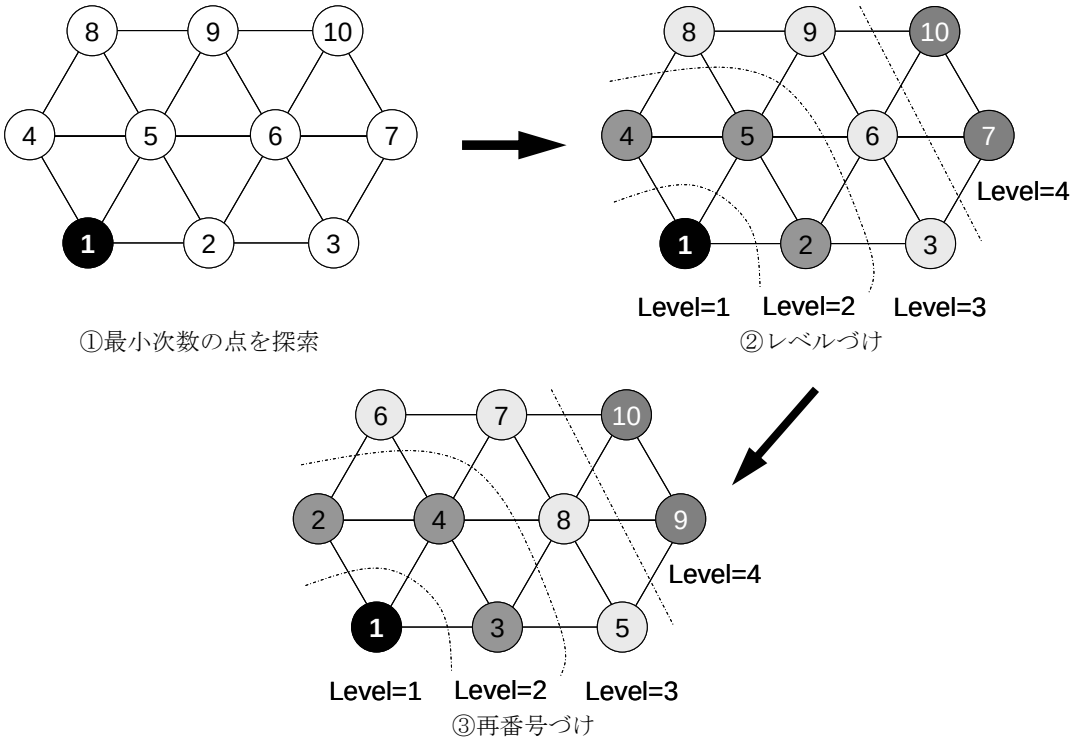
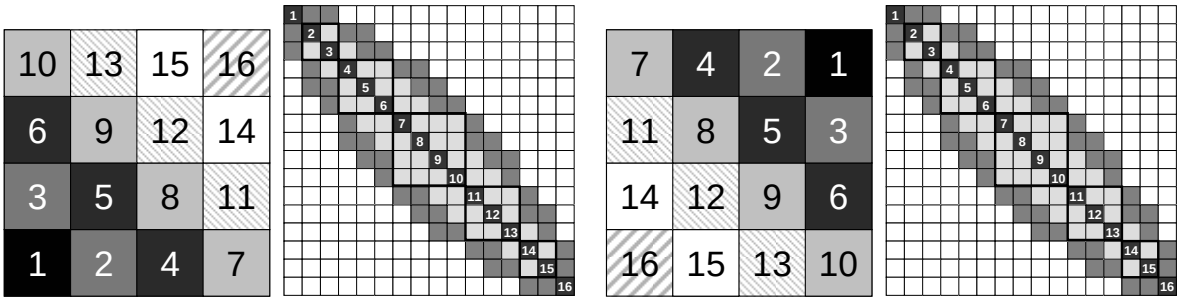


図 6　Cuthill-McKee Ordering（CM 法）の例

RCM 法は図 6 に示す CM 法で得られた番号を逆順にふり直すもので，LU 分解時の fill-in が減

少するため，不完全コレスキー分解時にも有効と考えられる〔2〕。しかし，図 1 に示した 16 要素の形状の場合は，fill-in の数はともに 44 である（図 7（a），（b）参照）。レベル数はともに 7 であり，図 7（a），（b）に示すように対角線上に並んだ要素群が同じレベルに分類される。これは差分格子における Hyberline（三次元では Hyperplane）と同じである〔4〕。



（a）CM 法　　（b）RCM 法

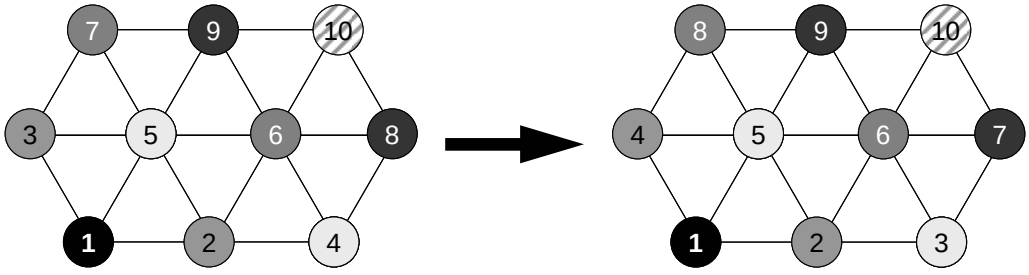
図 7　CM 法，RCM 法の適用例（■：非ゼロ成分，■：fill-in（ともに 44））

本稿では，前述の CM 法のアルゴリズムを，同じレベルに属する点，要素同士が独立となるように，以下のように少し修正している：

アルゴリズム 3：修正された CM 法のアルゴリズム

① 各要素に隣接する要素数を「次数」とし，最小次数の要素を「レベル=1」の要素とする。
② 「レベル=k-1」の要素に隣接する要素を「レベル=k」とする。同じレベルに属する要素はデータ依存性が発生しないように，隣接している要素同士が同じレベルに入る場合は一方を除外する（現状では先に見つかった要素を優先している）。全ての点要素にレベルづけがされるまで「k」を 1 つずつ増やして繰り返す。
③ 全ての要素がレベルづけされたら，レベルの順番に再番号づけする。

図 6 に示した例に，この修正された CM 法（以下単に「CM 法」と呼ぶ）を適用すると図 8 のようになる。レベルの総数が 4 から 6 に増加している。図 6 では，{2,4,5}（旧番号）は同じレベルに属していたが，図 8 では{5}は違うレベルに属している。図 7 の例に関しては，CM 法を適用しても結果は変わらないが，参考のため，図 9 にレベル番号づけの手順を示す。MC 法と CM 法の大きな違いは，CM 法では，同一レベル（色）における各要素の独立性だけでなく，計算順序を考慮して，レベル間の依存性を考慮している点にある。



（a）各点のレベル設定　　（b）CM ordering

図 8　修正された CM 法による ordering

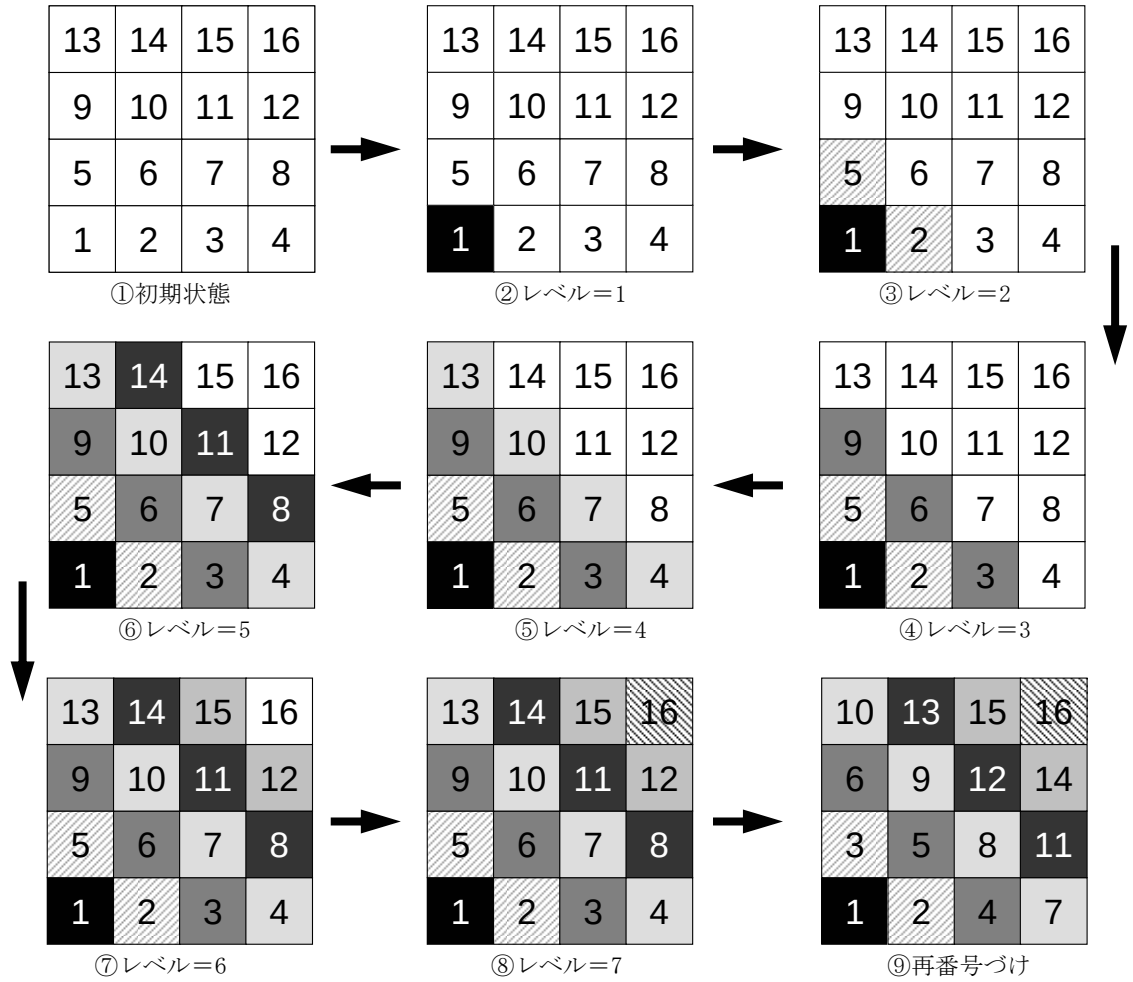


図 9　修正された CM 法による ordering

3. プログラムのダウンロード，コンパイル

今回使用するプログラム，関連ファイルは下記ホームページよりダウンロードできる：

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

読者は，前回作成した，本連載用のディレクトリ（<$tutorial-top>）へ移動し，そこへ上記の TAR ファイルをインストールし，解凍すると，以下のディレクトリが得られる：

```
<$tutorial-top>/Lesson-2/coloring
<$tutorial-top>/Lesson-2/solver
```

それぞれのディレクトリの下には更に以下のディレクトリが生成されている（それぞれ以下のように略称する）：

```
<$tutorial-top>/Lesson-2/coloring/src ⇒ <$L2-color>/src
<$tutorial-top>/Lesson-2/coloring/run ⇒ <$L2-color>/run
<$tutorial-top>/Lesson-2/solver/src   ⇒ <$L2-solve>/src
<$tutorial-top>/Lesson-2/solver/run   ⇒ <$L2-solve>/run
```

<$L2-color>は MC 法, CM 法，RCM 法の実習のためのプログラムであり，<$L2-solve>は

前回紹介した ICCG 法によるプログラム（以下「ICCG-L1」と呼ぶ）に MC 法, CM 法，RCM 法を組み込んだものである。以下のコマンドを実行し，<$L2-color>/run/L2-color，<$L2-solve>/run/L2-sol，という実行形式が生成されていることを確認する：

```
$> cd <$L2-color>/src
$> make
$> ls ../run/L2-color

$> cd <$L2-solve>/src
$> make
$> ls ../run/L2-sol
```

L2-sol を実行する場合には，要素生成の必要がある。要素生成プログラム等は前回使用したものと同様にして，以下のようにコンパイル，リンクを実施しておく。

```
$> cd <$L2-solve>/run
$> f90 -O mg.f -o mg  または cc -O mg.c -o mg
```

4. Ordering の実習 <$L2-color>

<$L2-color>では，図 1 に示した 16 要素からなる二次元形状に対して，様々な ordering 手法を適用することができる。図 1 に示したような形状は，要素生成プログラムを起動させた場合に，NX=4，NY=4，NZ=1 を入力すると生成される（前回連載参照）。ここで，

```
$> cd <$L2-color>/run/
$> ls
   L2-color    mesh.dat
```

において，「mesh.dat」の中身を確認すると，以下のように，「NX=4，NY=4，NZ=1」となっていることがわかる。

```
 4   4   1
16
 1   0   2   0   5   0   0   1   1   1
 2   1   3   0   6   0   0   2   1   1
 3   2   4   0   7   0   0   3   1   1
 4   3   0   0   8   0   0   4   1   1
 5   0   6   1   9   0   0   1   2   1
 6   5   7   2  10   0   0   2   2   1
 7   6   8   3  11   0   0   3   2   1
 8   7   0   4  12   0   0   4   2   1
 9   0  10   5  13   0   0   1   3   1
10   9  11   6  14   0   0   2   3   1
11  10  12   7  15   0   0   3   3   1
12  11   0   8  16   0   0   4   3   1
13   0  14   9   0   0   0   1   4   1
14  13  15  10   0   0   0   2   4   1
15  14  16  11   0   0   0   3   4   1
16  15   0  12   0   0   0   4   4   1
```

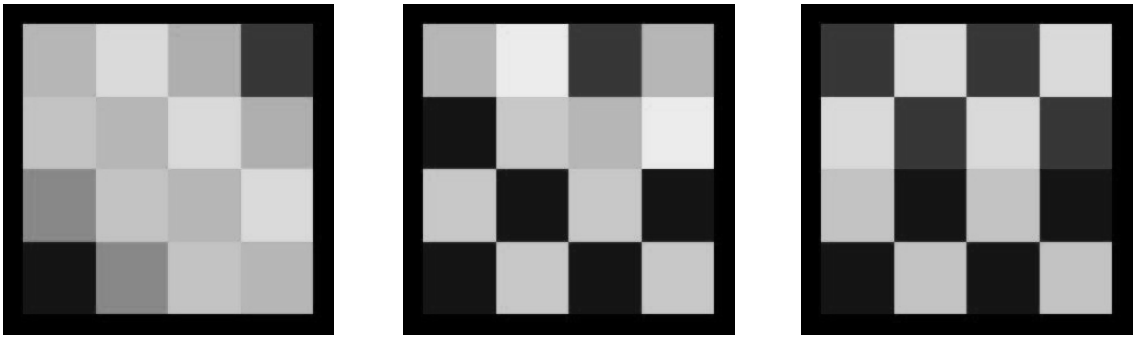
続いて，

```
$> ./L2-color
```

とタイプすると，以下のようなメッセージ画面が現れ，入力待ちの状態になる：

```
You have     16 elements.
How many colors do you need ?
  #COLOR must be more than 2 and
  #COLOR must not be more than     16
  if #COLOR=0 :  CM ordering
  if #COLOR<0 : RCM ordering
=>
```

Ordering 手法，色数を尋ねてくるが，MC 法の場合は色数（2 以上 16 以下），CM 法の場合は 0，RCM 法の場合は負の数を入力すればよい。例えば，「2」を入力すると red-black ordering になる。計算が終了すると，画面には「**# TOTAL COLOR number**」として色数，レベル数が表示され，実行ディレクトリ**<$L2-color>/run/**の中に，「color.inp」，「color.log」という 2 種類のファイルが生成される。「color.inp」は彩色の結果を記述したファイルで，UCD ファイル形式となっており，「MicroAVS」[1]を使って表示することができる。「color.log」は ordering の詳細な情報が記述されているファイルである（後述）。図 10 は「color.inp」を「MicroAVS」で表示した例である。



（a）入力＝0（CM，7 レベル）　（b）入力＝3（実際は 5 色に彩色）　（c）入力＝4（4 色）

図 10　MicroAVS による color.inp の表示

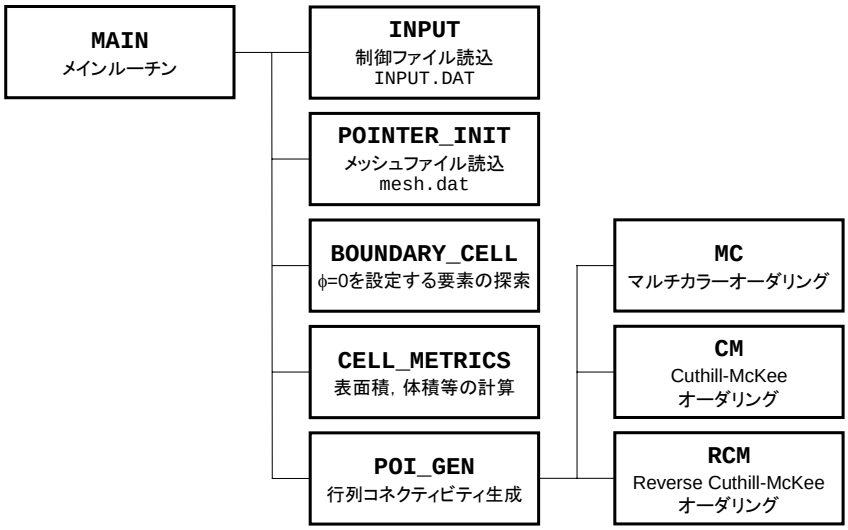


図 11　サブルーチン構成図（<$L2-color>）

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

プログラムのサブルーチン構成は図 11 のようになっており, 行列のコネクティビティを生成するところまでは，前回紹介した ICCG 法（ICCG-L1）の場合とほぼ同じである。ここでは，そのあと ordering を施した状態で計算を終了している。

図 11 に示した，MC（multicolor ordering，MC 法），CM（Cuthill-McKee 法），RCM（Reverse Cuthill-McKee 法）を呼び出す部分はリスト 5 に示すようになっている。

```
111  continue
     write (*,'(//a,i8,a)') 'You have', ICELTOT, ' elements.'
     write (*,'(  a     )') 'How many colors do you need ?'
     write (*,'(  a     )') '  #COLOR must be more than 2 and'
     write (*,'(  a,i8  )') '  #COLOR must not be more than', ICELTOT
     write (*,'(  a     )') '  if #COLOR=0 :  CM ordering'
     write (*,'(  a     )') '  if #COLOR<0 : RCM ordering'
     write (*,'(  a     )') '=>'

     read  (*,*)           NCOLORtot
     if (NCOLORtot.eq.1.or.NCOLORtot.gt.ICELTOT) goto 111

     allocate (OLDtoNEW(ICELTOT), NEWtoOLD(ICELTOT))
     allocate (COLORindex(0:ICELTOT))

     if (NCOLORtot.gt.0) then
       call MC (ICELTOT, NL, NU, INL, IAL, INU, IAU,                &
    &           NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW)
     endif
     if (NCOLORtot.eq.0) then
       call  CM (ICELTOT, NL, NU, INL, IAL, INU, IAU,               &
    &            NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW)
     endif

     if (NCOLORtot.lt.0) then
       call RCM (ICELTOT, NL, NU, INL, IAL, INU, IAU,               &
    &            NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW)
     endif
```

リスト 5　Ordering のためのサブルーチン群の呼び出し

変数名と内容は表 1 に示す通りである：

表 1　　変数表

変数，配列名	型	内 容
NCOLORtot	整数	入力時には Ordering 手法（>2：MC，=0：CM，<0：RCM） 最終的には色数，レベル数が入る
ICELTOT	整数	要素数（＝16）
NL	整数	各要素の下三角成分の最大数（=6 に設定してある）
NU	整数	各要素の上三角成分の最大数（=6 に設定してある）
INL(i)	整数	要素 *i* の下三角成分数（<NL），i=1~ICELTOT
INU(i)	整数	要素 *i* の上三角成分数（<NU），i=1~ICELTOT
IAL(j,i)	整数	要素 *i* の *j* 番目の下三角成分に対応する列番号，i=1~ICELTOT，j=1~INL(i)
IAU(j,i)	整数	要素 *i* の *j* 番目の上三角成分に対応する列番号，i=1~ICELTOT，j=1~INU(i)
COLORindex	整数	各色，レベルに含まれる要素数の一次元圧縮配列，i=0, NCOLORtot COLORindex(icol-1)+1 から COLORindex(icol)までの要素が icol 番目の色（レベル）に含まれる。
NEWtoOLD(i)	整数	新番号⇒旧番号への参照配列，i=1~ICELTOT
OLDtoNEW(i)	整数	旧番号⇒新番号への参照配列，i=1~ICELTOT

NCOLORtot は，入力画面から入力する際には：

≧2	MC 法の色数
= 0	CM 法
＜0	RCM 法

であるが，MC，CM，RCM の各サブルーチンからの戻り値は最終的な色数，レベル数となる。INL，INU，IAL，IAU については前回連載でも紹介したように行列のコネクティビティに関する配列である。これらの配列に関しては，MC，CM，RCM に対しては図 1 に示した初期状態の番号づけに基づいたコネクティビティが送られ，新しい番号づけに基づいた値が戻り値となる。

MC 法を選択した場合，NCOLORtot の値は，最終的には入力と異なった値となる場合がある。例えば，下記のように色数として「3」を入力すると，最終的な色数は「5」となる：

```
You have     16 elements.
How many colors do you need ?
  #COLOR must be more than 2 and
  #COLOR must not be more than     16
  if #COLOR=0 :  CM ordering
  if #COLOR<0 : RCM ordering
=>
3

# TOTAL COLOR number      5
```

CM 法，RCM 法の場合は，NCOLORtot の値は，プログラム側で計算されたレベル数の値が入り，例えば CM 法の場合以下のように「レベル数＝7」となる：

```
You have     16 elements.
How many colors do you need ?
  #COLOR must be more than 2 and
  #COLOR must not be more than     16
  if #COLOR=0 :  CM ordering
  if #COLOR<0 : RCM ordering
=>
0

# TOTAL COLOR number      7
```

MC 法を選択した場合，本プログラム内部では以下のようなアルゴリズムによって，「彩色（coloring)」が行なわれている：

アルゴリズム 4：修正された MC 法のアルゴリズム

① 「次数」最小の要素を「新要素番号＝1」,「第 1 色」とし,「色数のカウンタ＝1」とする。
② ITEMcou=ICELTOT/NCOLORtot に相当する値を「各色に含まれる要素数」とする。
③ ITEMcou 個の独立な要素を初期要素番号が若い順に選び出す。
④ ITEMcou 個の要素が選ばれるか，あるいは独立な要素が無くなったら,「色数のカウンタ＝2」として第 2 色へ進む。
⑤ 全ての要素が彩色されるまで③，④を繰り返す。
⑥ 最終的な色数カウンタの値を NCOLORtot とし，色番号の若い順番に要素を再番号づけする（各色内では初期要素番号の順番)。

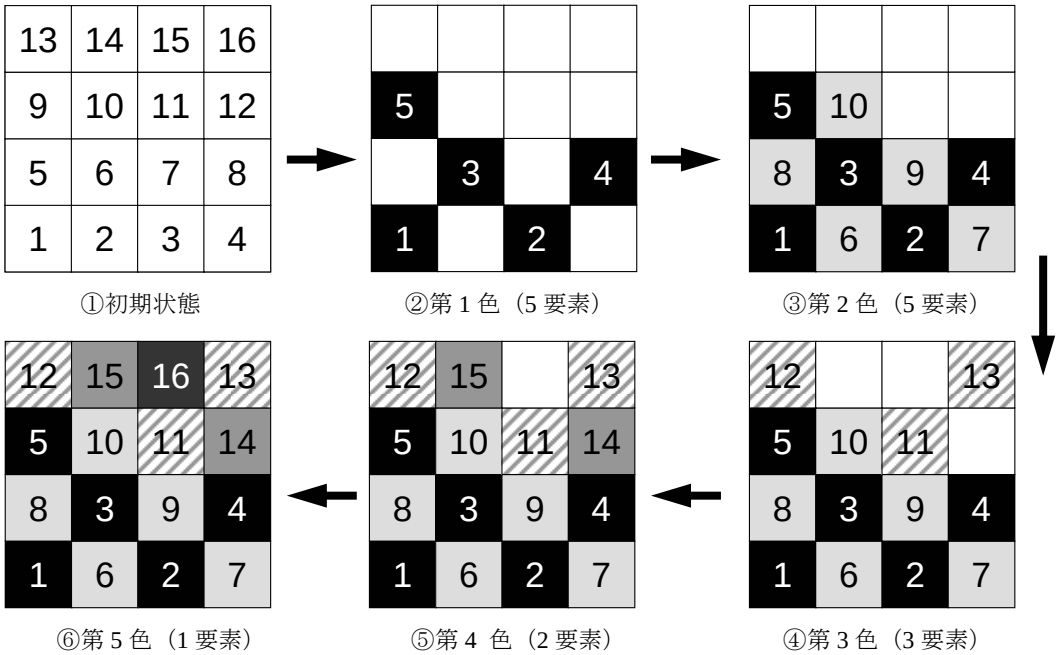


図 12　MC 法：NCOLORtot=3（初期入力） とした場合（実際は 5 色に彩色される）

従って，②で ICELTOT が NCOLORtot で割り切れない場合，④で独立な要素が無くなる場合などは，当初の設定よりも色数 NCOLORtot が多くなることがありうる。図 12 は NCOLORtot の初期入力を「=3」とした場合の彩色（coloring）のプロセスである。この場合 ITEMcou=16/3=5 である。第 1 色，第 2 色では，それぞれ 5 個ずつの独立な要素が見つかっているが，第 3 色では 3 要素，第 4 色では 2 要素しか無く，最終的に NCOLORtot=5 となっている。

以下に出力ファイル「color.log」の内容について説明する。「color.log」は 3 つの部分に分かれているが，そのうち最初の部分は，初期状態（図 12 の①）における，行列のコネクティビティである。表 1 に示した，INL，INU，IAL，IAU の内容が出力されている。例えば「6 番」の要素に注目すると，INL(6)=2，INU(6)=2 で合計 4 つの要素に隣接しており，それらは，「IAL=2,5」，「IAU=7,10」である（リスト 6 参照）。

```
### INITIAL connectivity
I=    1    INL(i)=    0    INU(i)=    2
  IAL:
  IAU:    2    5
I=    2    INL(i)=    1    INU(i)=    2
  IAL:    1
  IAU:    3    6
I=    3    INL(i)=    1    INU(i)=    2
  IAL:    2
  IAU:    4    7
I=    4    INL(i)=    1    INU(i)=    1
  IAL:    3
  IAU:    8
I=    5    INL(i)=    1    INU(i)=    2
  IAL:    1
  IAU:    6    9
I=    6    INL(i)=    2    INU(i)=    2
  IAL:    2    5
  IAU:    7   10
I=    7    INL(i)=    2    INU(i)=    2
  IAL:    3    6
  IAU:    8   11
I=    8    INL(i)=    2    INU(i)=    1
  IAL:    4    7
  IAU:   12
I=    9    INL(i)=    1    INU(i)=    2
  IAL:    5
  IAU:   10   13
I=   10    INL(i)=    2    INU(i)=    2
  IAL:    6    9
  IAU:   11   14
I=   11    INL(i)=    2    INU(i)=    2
  IAL:    7   10
  IAU:   12   15
I=   12    INL(i)=    2    INU(i)=    1
  IAL:    8   11
  IAU:   16
I=   13    INL(i)=    1    INU(i)=    1
  IAL:    9
  IAU:   14
I=   14    INL(i)=    2    INU(i)=    1
  IAL:   10   13
  IAU:   15
I=   15    INL(i)=    2    INU(i)=    1
  IAL:   11   14
  IAU:   16
I=   16    INL(i)=    2    INU(i)=    0
  IAL:   12   15
  IAU:
```

リスト 6　「color.log」（初期コネクティビティ）

「color.log」の第 2 の部分は，新旧番号の対応表と所属する色番号に関する情報が出力されている（リスト 7）。まず，最終的な色数（NCOLORtot=5）が出力されており，以下，「新要

素番号，対応する初期要素番号，所属する色番号」の順番で出力されている。例えば，初期状態で「6 番」であった要素は，新しい番号づけでは「3 番」となっている。このことは図 12①，⑥からも確認できる。また，所属の色番号順に再番号づけされていることも確認できる。また，「COLORindex」については，この色番号から，以下のようになることがわかる：

- COLORindex(0)= 0
- COLORindex(1)= 5
- COLORindex(2)=10
- COLORindex(3)=13
- COLORindex(4)=15
- COLORindex(5)=16

```
COLOR number      5

 #new      1   #old      1  color      1
 #new      2   #old      3  color      1
 #new      3   #old      6  color      1
 #new      4   #old      8  color      1
 #new      5   #old      9  color      1
 #new      6   #old      2  color      2
 #new      7   #old      4  color      2
 #new      8   #old      5  color      2
 #new      9   #old      7  color      2
 #new     10   #old     10  color      2
 #new     11   #old     11  color      3
 #new     12   #old     13  color      3
 #new     13   #old     16  color      3
 #new     14   #old     12  color      4
 #new     15   #old     14  color      4
 #new     16   #old     15  color      5
```

リスト 7　「color.log」（新旧番号対応表，所属色番号）

リスト 8 は，新しい番号づけにおけるコネクティビティである。初期状態で「6 番」だった要素は新しい番号づけでは「3 番」になっている。隣接要素は，「6,8,9,10」となっており，図 12⑥に示した通りである。隣接要素の番号は全て「3 番」より大きいため，「INL=0，INU=4」となっている。リスト 7 より，「6,8,9,10」の色番号は全て「2」である。同様に，各要素とその隣接要素とは異なった番号の「色」に属している。このことは，図 12，リスト 7，リスト 8 からも確認できる。

```
### FINAL connectivity
I=   1    INL(i)=   0    INU(i)=   2
  IAL:
  IAU:   6   8
I=   2    INL(i)=   0    INU(i)=   3
  IAL:
  IAU:   6   7   9
I=   3    INL(i)=   0    INU(i)=   4
  IAL:
  IAU:   6   8   9  10
I=   4    INL(i)=   0    INU(i)=   3
  IAL:
  IAU:   7   9  14
I=   5    INL(i)=   0    INU(i)=   3
  IAL:
  IAU:   8  10  12
I=   6    INL(i)=   3    INU(i)=   0
  IAL:   1   2   3
  IAU:
I=   7    INL(i)=   2    INU(i)=   0
  IAL:   2   4
  IAU:
I=   8    INL(i)=   3    INU(i)=   0
  IAL:   1   3   5
  IAU:
I=   9    INL(i)=   3    INU(i)=   1
  IAL:   2   3   4
  IAU:  11
I=  10    INL(i)=   2    INU(i)=   2
  IAL:   3   5
  IAU:  11  15
I=  11    INL(i)=   2    INU(i)=   2
  IAL:   9  10
  IAU:  14  16
I=  12    INL(i)=   1    INU(i)=   1
  IAL:   5
  IAU:  15
I=  13    INL(i)=   0    INU(i)=   2
  IAL:
  IAU:  14  16
I=  14    INL(i)=   3    INU(i)=   0
  IAL:   4  11  13
  IAU:
I=  15    INL(i)=   2    INU(i)=   1
  IAL:  10  12
  IAU:  16
I=  16    INL(i)=   3    INU(i)=   0
  IAL:  11  15  13
  IAU:
```

リスト 8 「color.log」（新しい番号づけにおけるコネクティビティ）

5. Ordering つき ICCG 法 <$L2-solve>

<$L2-solve>には，前回連載で紹介した ICCG 法のプログラム（ICCG-L1）に 4. で示した ordering 手法（MC 法，CM 法，RCM 法）を組み込んだ ICCG 法のプログラムが含まれている。以後，このプログラムを「ICCG-L2」と呼ぶこととする。「ICCG-L2」では前処理手法としては，「ICCG-L1」における「METHOD=1」の場合，すなわち，式（1）に示す，不完全修正コレスキー分解において，$\tilde{l}_{ij} = a_{ij}$ とした場合のみを考慮している。図 13 にサブルーチン構成図を示す。

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

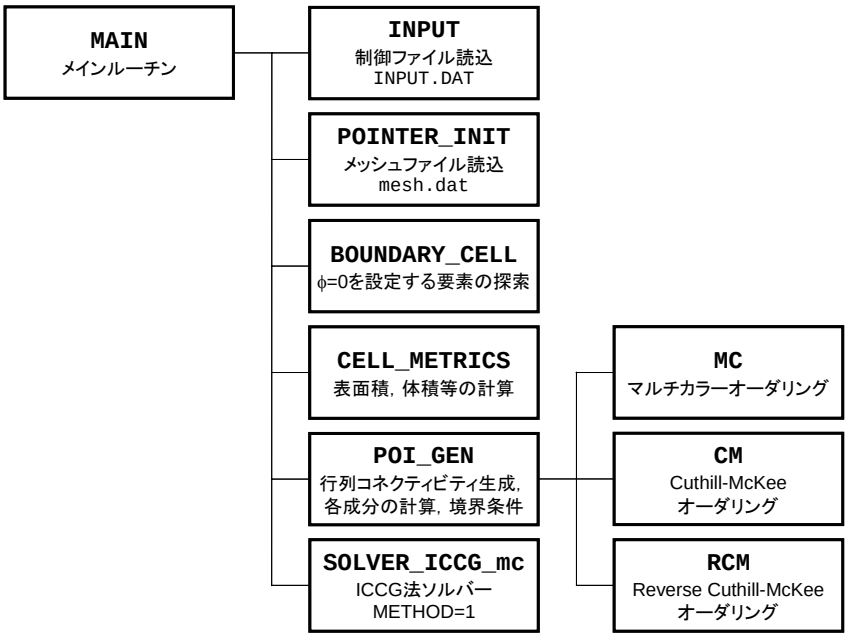


図 13　サブルーチン構成図（ICCG-L2）

「ICCG-L2」は「ICCG-L1」とほとんど同じ処理であるが，「POI_GEN」で ordering のための「MC,CM,RCM」を呼び出して，再番号づけする。再番号づけした状態で各行列の成分の計算，境界条件の適用を実施した後，ICCG 法のサブルーチン（SOLVER_ICCG_mc）を呼び出している。「SOLVER_ICCG_mc」は「ICCG-L1」の「solver_ICCG1」と比較すると，修正不完全コレスキー分解を実施する部分（前処理行列の対角成分の逆数 DD(:)の計算），前進後退代入の部分が以下のように異なっている：

```
      do icol= 1, NCOLORtot
        do i= COLORindex(icol-1)+1, COLORindex(icol)
          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
      enddo
```

リスト 9　修正不完全コレスキー分解

```
      do icol= 1, NCOLORtot
        do i= COLORindex(icol-1)+1, COLORindex(icol)
          WVAL= z(i)
          do j= 1, INL(i)
            WVAL= WVAL - AL(j,i) * z(IAL(j,i))
          enddo
          z(i)= WVAL * DD(i)
        enddo
      enddo
```

リスト 10　前進代入

```
      do icol= NCOLORtot, 1, -1
        do i= COLORindex(icol-1)+1, COLORindex(icol)
          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
      enddo
```

リスト 11　後退代入

それぞれの計算は各色（レベル）ごとに順番に実施されている。同じ色に属する要素は互いに独立であり，右辺に現れる要素は異なった色に属する要素のため，各色の要素が計算される間に右辺の値が更新される可能性は無い。従って，DD(i)，z(i)の計算に当たって，データ依存性を生ずることは無い。

計算を実施するにあたっては，制御データ「INPUT.DAT」と要素データ「mesh.dat」が必要である。<$L2-solve>/run にある「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
DX，DY，DZ	倍精度実数	各要素の 3 辺の長さ（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

初期状態では DX=DY=DZ=1.0，EPSICCG=10^{-8} となっているが，以下この値をそのまま使用する。要素生成に当たっては，3. で述べたように<$L2-solve>/run において予めコンパイル，リンクして要素生成プログラム「mg」を作成しておく。続いて：

```
$> cd <$L2-solve>/run
$> ./mg
```

とすると入力待ちの状態になるので，例えば NX=20，NX=20，NX=20 を入力して，要素数 8,000 の「mesh.dat」が生成される。ここで：

```
$> ./L2-sol
```

とタイプすると，「ICCG-L2」が実行される。以下に示すように入力待ちとなるので：

```
You have    8000 elements.
How many colors do you need ?
  #COLOR must be more than 2 and
  #COLOR must not be more than    8000
  if #COLOR= 0 : CM  ordering
  if #COLOR< 0 : RCM ordering
=>
```

例えばここで「0」と入力すると，CM 法が選択され以下のように出力される（ICCG-L2/CM）：

```
### FINAL COLOR NUMBER     58

    1    3.457810E+00
   48    5.614658E-09

N=       8000
```

これは，色数（レベル数）が 58 で，48 回で収束（収束判定値はこの場合 10^{-8}）していることを示す。

「-1」とすると RCM 法が選択され以下のように出力される（ICCG-L2/RCM）：

```
### FINAL COLOR NUMBER     58

    1    3.523560E+00
   46    9.145094E-09

N=       8000
```

CM 法と RCM 法では，ここで対象としているような形状に対しては，完全 LU 分解時の fill-in 等は変わらないが，境界条件の適用の順番が異なっているため，RCM 法が若干早く収束している可能性がある。これに対して，red-black ordering として「2」を入力すると（ICCG-L2/MC）：

```
### FINAL COLOR NUMBER      2

    1    4.889199E+00
   71    7.443228E-09

N=       8000
```

となり，CM 法，RCM 法と比較して，収束は遅い。

また「53」を入力すると，色数が 53 の MC 法となるが，8,000 が 53 で割り切れないため，以下に示すように最終的な色数は 54 となる：

```
### FINAL COLOR NUMBER     54

    1    3.457810E+00
   65    6.544098E-09

N=       8000
```

反復回数は，red-black（2 色）の場合と比較すると改善されている。

6. 色数とオーダリングの関係

MC法の色数のICCG法の収束性への効果については，これまで様々な研究によって説明が試みられているが，〔5〕においては土肥らによって「Incompatible Nodes（以下ICN)」の概念に基づいて説明されている。図1などに示した16要素の二次元体系において，節点番号順に前進代入あるいはGauss-Seidelのような操作を施した場合，節点1以外の節点は全て，番号の若い要素の影響を受ける。ICNとは，この図における要素1，すなわち，他の要素から影響を受けない要素のことである（図14)。文献〔5〕によれば，一般にICNの数が少ないほど，他の要素の計算結果の効果を考慮しながら計算が実施されていることから，収束が良い。

2色に塗り分けるred-black ordering の場合，多くのICNを持つ（図15)。また，色数を増やして，4色にすると，図16に示すように，ICNの数は減少する。基本的に色数を増加させるとICNの数は減少する（非常に複雑な形状の場合などで例外はあるが)。

また，CM法の場合は，図17に示すようにICNの数は1である。アルゴリズム4，図4，図12に示すように，MC法では，各色における要素の独立性のみが考慮されているのに対して，アルゴリズム3，図9に示すように，CM法，RCM法では，各レベル（色）における要素の独立性とともに，各レベル（色）間の依存性についても考慮されており，不完全修正コレスキー分解，前進後退代入における計算順序に適合した並び替えとなっている。

図18は，8,000要素（NX=NY=NZ=20）における，色数と収束までの反復回数，ICNの数の関係を示したものである。ICCG-L1とICCG-L2/CM，ICCG-L2/RCMの反復回数が，ほぼ同じでICCG-L2/MCと比較して早く収束しているのは，図14～図17で示したICNの数と反復回数の関係に対応している。また，色数の増加とともにICNの数が減少していることもわかる。

ICCG-L2/MCでは若干の例外はあるものの，全体的に色数の増加に従って，収束は改善されている。

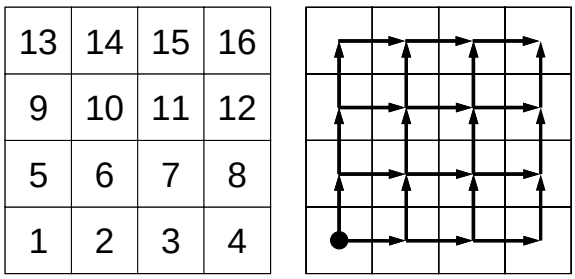


図14　初期状態（●：Incompatible Nodes）

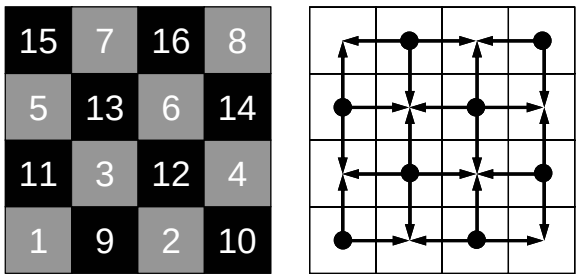


図15　MC（色数:2）（●：Incompatible Nodes）

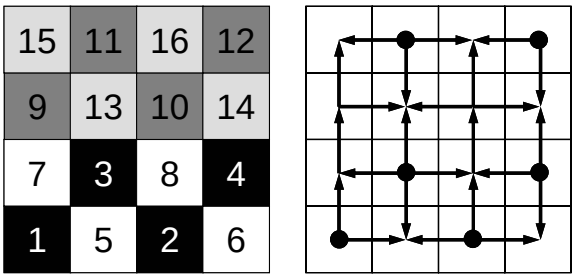


図16　MC（色数:4）（●：Incompatible Nodes）

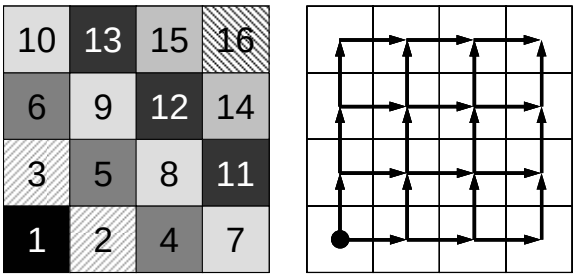


図17　CM（レベル数:7）（●：Incompatible Nodes）

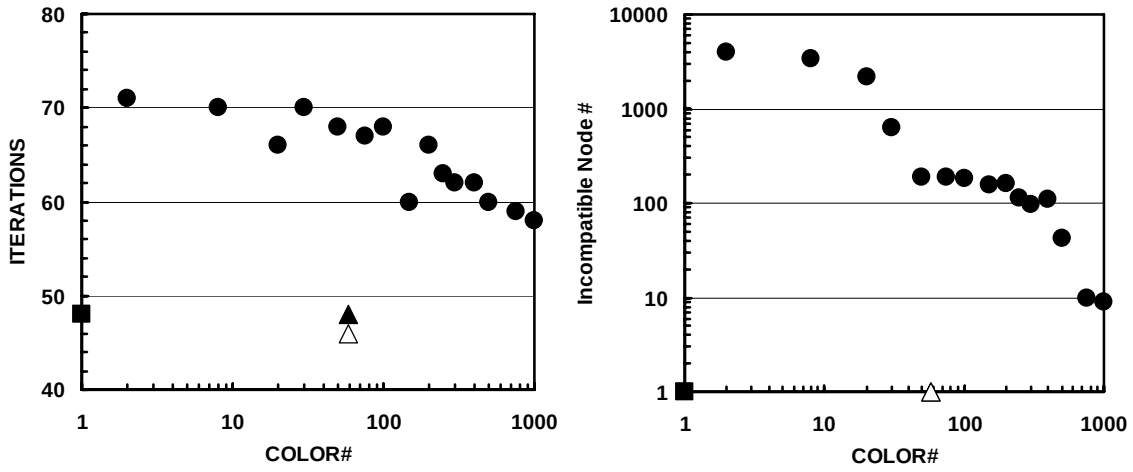


図18　ICCG法の収束（反復回数），Incompatible Nodeの数と色数の関係
（20^3=8,000要素，EPSICCG=10^{-8}）
（■：ICCG-L1，●：ICCG-L2/MC，▲：ICCG-L2/CM，△：ICCG-L2/RCM）

7. まとめ

有限体積法によって離散化された三次元領域で，ポアソン方程式をICCG法によって解くプログラム「ICCG-L1」に，multicolor ordering（MC），Cuthill-McKee ordering（CM），Reverse Cuthill-McKee ordering（RCM）を施したプログラム「ICCG-L2」を紹介した。互いに独立な要素を抽出して，彩色，再番号づけすることによって，ICCG法の修正不完全コレスキー分解，前進後退代入などの処理において，データ依存性を除去することが可能となった。

CM法，RCM法はMC法と比較して，概して収束が早いことがわかった。また，MC法では，色数を増加させることによって収束が改善することを示した。これらの，ordering手法，色数と収束性の関係は「Incompatible Nodes（ICN）」の考え方によって説明が可能である。

次号では，「ICCG-L2」にOpenMPを導入し，Hitachi SR11000/J2における並列計算例について示す。

（次号に続く）

（問い合わせ先）中島 研吾（nakajima@eps.s.u-tokyo.ac.jp）

参　考　文　献

[1] 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

[2] Saad Y (2003) Iterative Methods for Sparse Linear Systems (2nd Edition), SIAM

[3] 小国 力 編（1991）「行列計算ソフトウェア　WS，スーパーコン，並列計算機」，丸善

[4] Dongarra, J.J., Duff, I.S., Sorensen, D.C., and van der Vorst, H.A. (1991) Solving Linear Systems on Vector and Shared Memory Computers, SIAM

[5] Doi, S. and Washio, T. (1999) Using Multicolor Ordering with Many Colors to Strike a Better Balance between Parallelism and Convergence, RIKEN Symposium on Linear Algebra and its Applications, 19-26