OpenMPによるマルチコア・メニィコア並列プログラミング入門
C言語編
Part-B2：オーダリング

中島研吾
東京大学情報基盤センター

- データ依存性の解決策は?
- オーダリング（Ordering）について
  - Red-Black，Multicolor（MC）
  - Cuthill-McKee（CM），Reverse-CM（RCM）
  - オーダリングと収束の関係
- オーダリングの実装
- オーダリング付ICCG法の実装

ICCG法の並列化

- 内積：OK
- DAXPY：OK
- 行列ベクトル積：OK
- 前処理：なんとかしなければならない
  - 単純にOpenMPなどの指示行（directive）を挿入しただけでは「並列化」できない。

データ依存性の解決策＝色分け，色づけ（coloring）

- 依存性を持たない（互いに独立な）要素を同時に処理するようにすれば良い

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

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

データ依存性の解決策＝色分け，色づけ（coloring）

- 依存性を持たない（互いに独立な）要素を同時に処理するようにすれば良い

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

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

データ依存性の解決策＝色分け，色づけ（coloring）

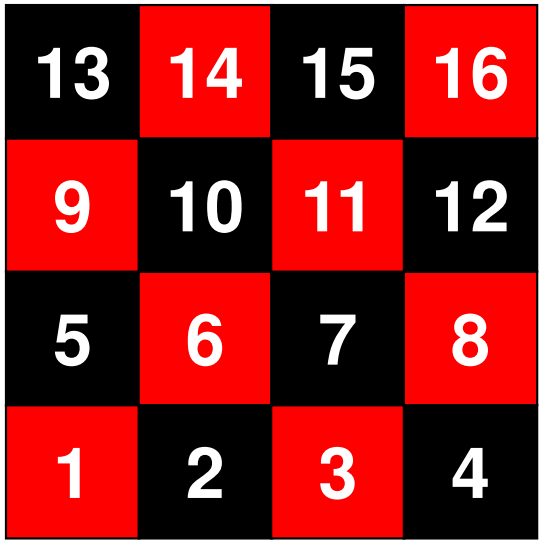
- 依存性を持たない要素群⇒同じ「色」に色づけ（coloring）する
- 最も単純な色づけ：Red-Black Coloring（2色）

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

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

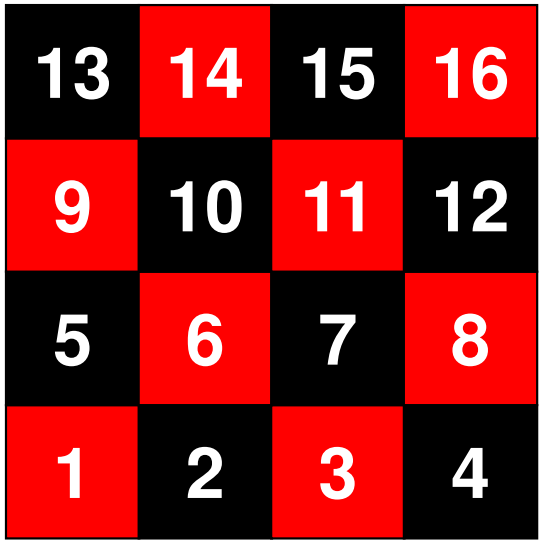
Numbering starts at 0 in the program, but I would like to use this one starting at 1. Please do not confuse !!

Red-Black (1/3)




```
#prgma omp parallel for private (ip,i,k,VAL)
for(ip=0; ip<4; ip++){
  for(i=INDEX[ip]; i<INDEX[ip+1]; i++){
    if (COLOR[i]== “RED” ){
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
  }
}

#prgma omp parallel for private (ip,i,k,VAL)
for(ip=0; ip<4; ip++){
  for(i=INDEX[ip]; i<INDEX[ip+1]; i++){
    if (COLOR[i]== “BLACK” ){
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
  }
}
```

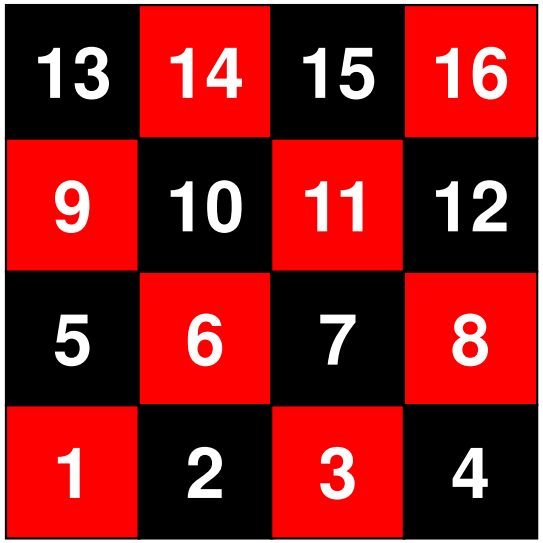
Red-Black (2/3)




```
#prgma omp parallel for private (ip,i,k,VAL)
for(ip=0; ip<4; ip++){
  for(i=INDEX[ip]; i<INDEX[ip+1]; i++){
    if (COLOR[i]== “RED” ){
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
  }
}
```

- 「Red」要素を処理する間，右辺に来るのは必ず「Black」要素
  - RED：書き出し，BLACK：読み込み
- 「Red」要素の処理をする間，「Black」要素の内容が変わることは無い
- データ依存性が回避される

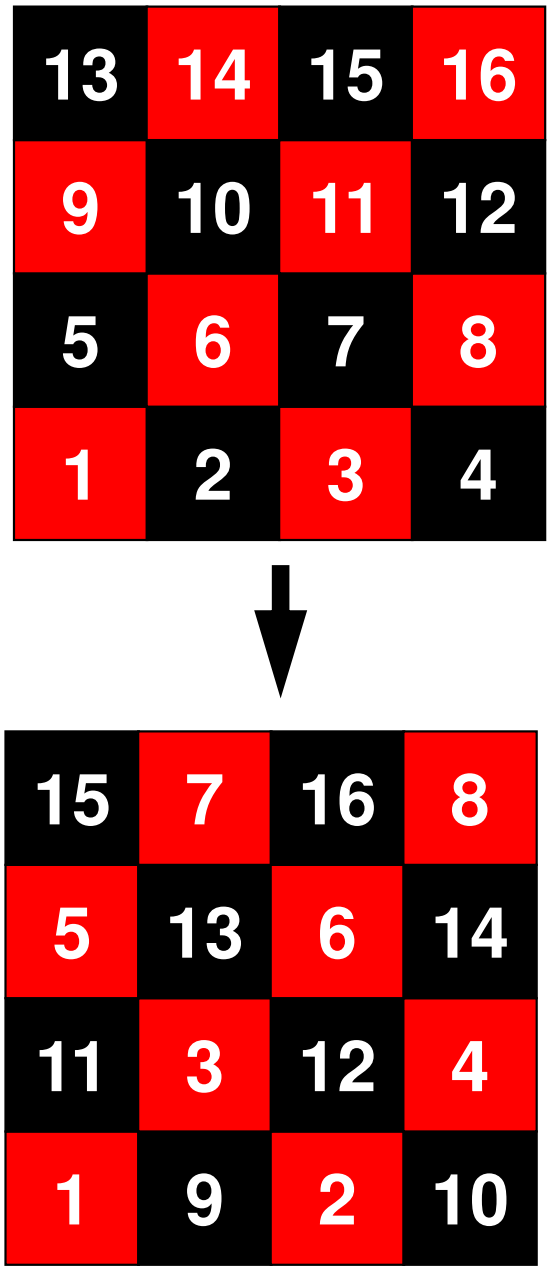
Red-Black (3/3)



- 「Black」要素を処理する間，右辺に来るのは必ず「Red」要素
  - RED：読み込み，BLACK：書き出し
- 「Black」要素の処理をする間，「Red」要素の内容が変わることは無い
- データ依存性が回避される

```
#prgma omp parallel for private (ip,i,k,VAL)
for(ip=0; ip<4; ip++){
  for(i=INDEX[ip]; i<INDEX[ip+1]; i++){
    if (COLOR[i]== “BLACK” ){
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
  }
}
```

Red-Black Ordering/Reordering




```
for(icol=0; icol<2; icol++){
#pragma omp parallel for private (ip,i,j,WVAL)
  for(ip=0; ip<4; ip++){
      for(i=INDEX[ip][icol]; i<INDEX[ip+1][icol]; i++){
          WVAL = W[Z][i];
          for(j=indexL[i]; j<indexL[i+1]; j++) {
              WVAL -= AL[j] * W[Z][itemL[j]-1];
          }
          W[Z][i] = WVAL * W[DD][i];
      }
  }
}
```

```
INDEX[0][0]= 0
INDEX[1][0]= 2
INDEX[2][0]= 4
INDEX[3][0]= 6
INDEX[4][0]= 8
```

```
INDEX[0][1]=  8
INDEX[1][1]= 10
INDEX[2][1]= 12
INDEX[3][1]= 14
INDEX[4][1]= 16
```

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

- 要素番号を「Red」⇒「Black」の順にふり直す(reordering, ordering)とプログラムが簡単になる(計算効率も高い)

- データ依存性の解決策は?
- **オーダリング（Ordering）について**
  - **Red-Black，Multicolor（MC）**
  - **Cuthill-McKee（CM），Reverse-CM（RCM）**
  - オーダリングと収束の関係
- オーダリングの実装
- オーダリング付ICCG法の実装
- マルチコアへの実装（OpenMP）へ向けて

オーダリング（ordering）の効用

- 並列性を得る：依存性の排除
- Fill-inを減らす
- バンド幅を減らす，プロファイルを減らす
- ブロック化

- もともとは，4色問題，一筆書き（巡回セールスマン問題）等と関連
  - 数値計算への適用

- 小国他「行列計算ソフトウェア」，丸善（1991）

並列計算のためのオーダリング法

- マルチカラーオーダリング
  - 並列性
  - Red-Black オーダリング（2色）等
- CM法（Cuthill-McKee），RCM法（Reverse Cuthill-McKee）
  - fill-inを減らす
  - マトリクスのバンド幅を減らす，プロファイルを減らす
  - 並列性

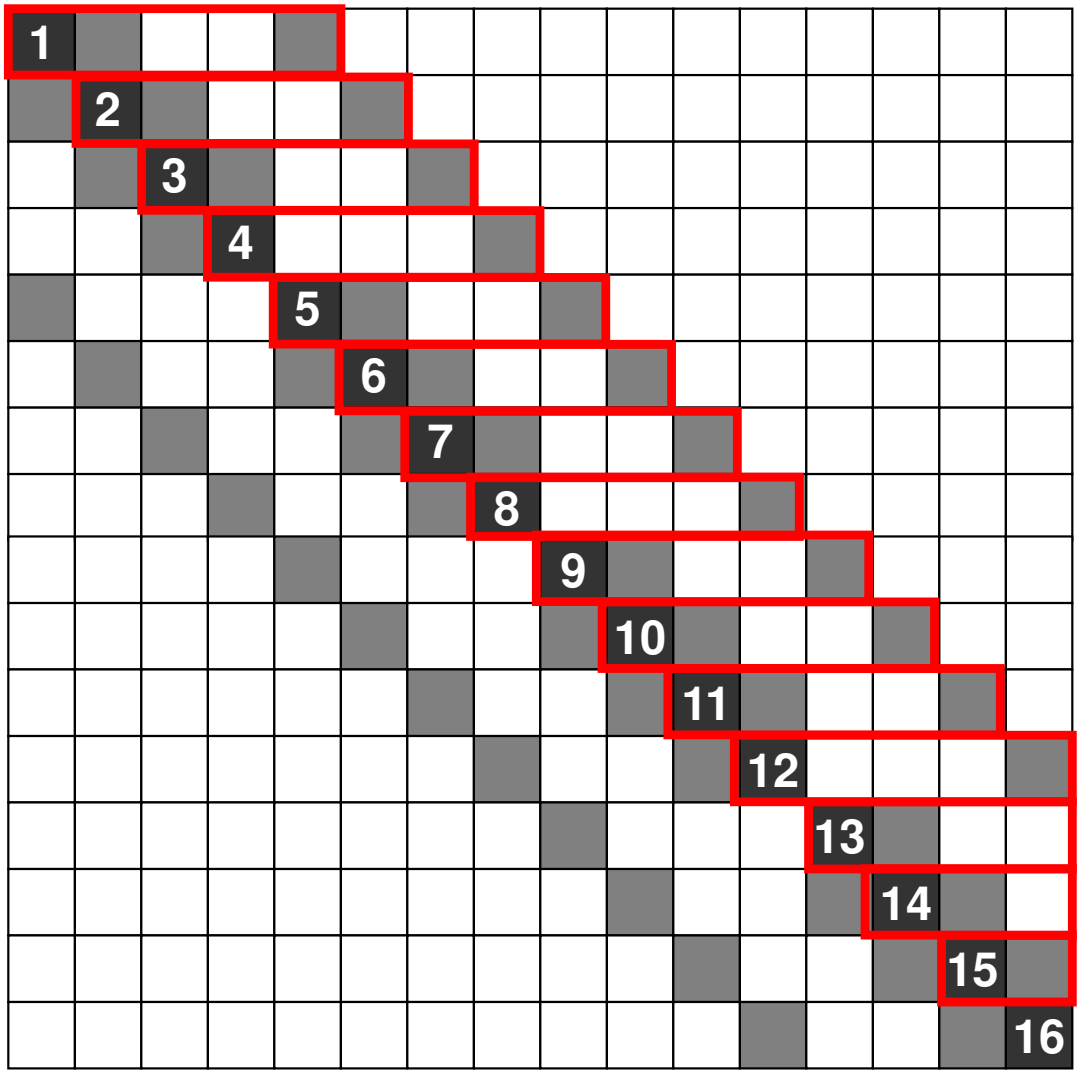
用語の定義

- β_i: i 行における非ゼロ成分の列番号の最大値を k とするとき，$\beta_i = k - i$
- バンド幅: β_iの最大値
- プロファイル: β_iの和

- バンド幅，プロファイル，Fill-inともに少ない方が都合が良い
- 特にバンド幅，プロファイルは（前処理付き反復法の）収束に影響

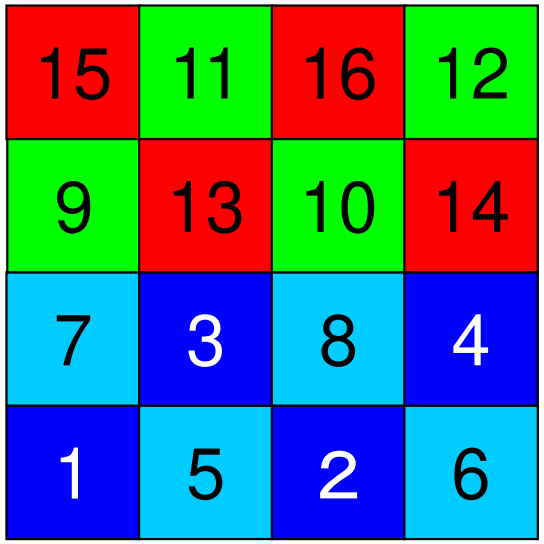
β_i の定義

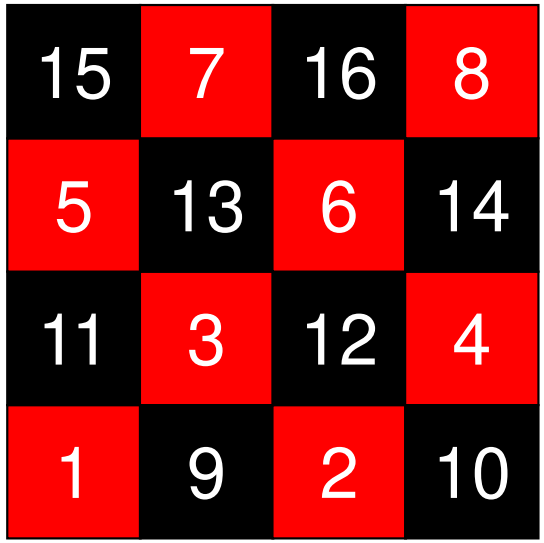
13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4



非ゼロ成分

マルチカラーオーダリング





- Multicolor Ordering
  - 「MC法」と呼ぶ
- マルチカラーオーダリングは，互いに独立で依存性のない要素同士を同じ「色」に分類し，その分類に従って要素や節点の番号を振りなおす手法である。
  - Red-Blackは2色の場合
  - 複雑形状の場合，より多い「色」が必要
- 同じ「色」に分類された要素に関する計算は並列に実施できる。
- ある要素と，その要素に接続する周囲の要素は違う「色」に属している。

MC法の基本的なアルゴリズム

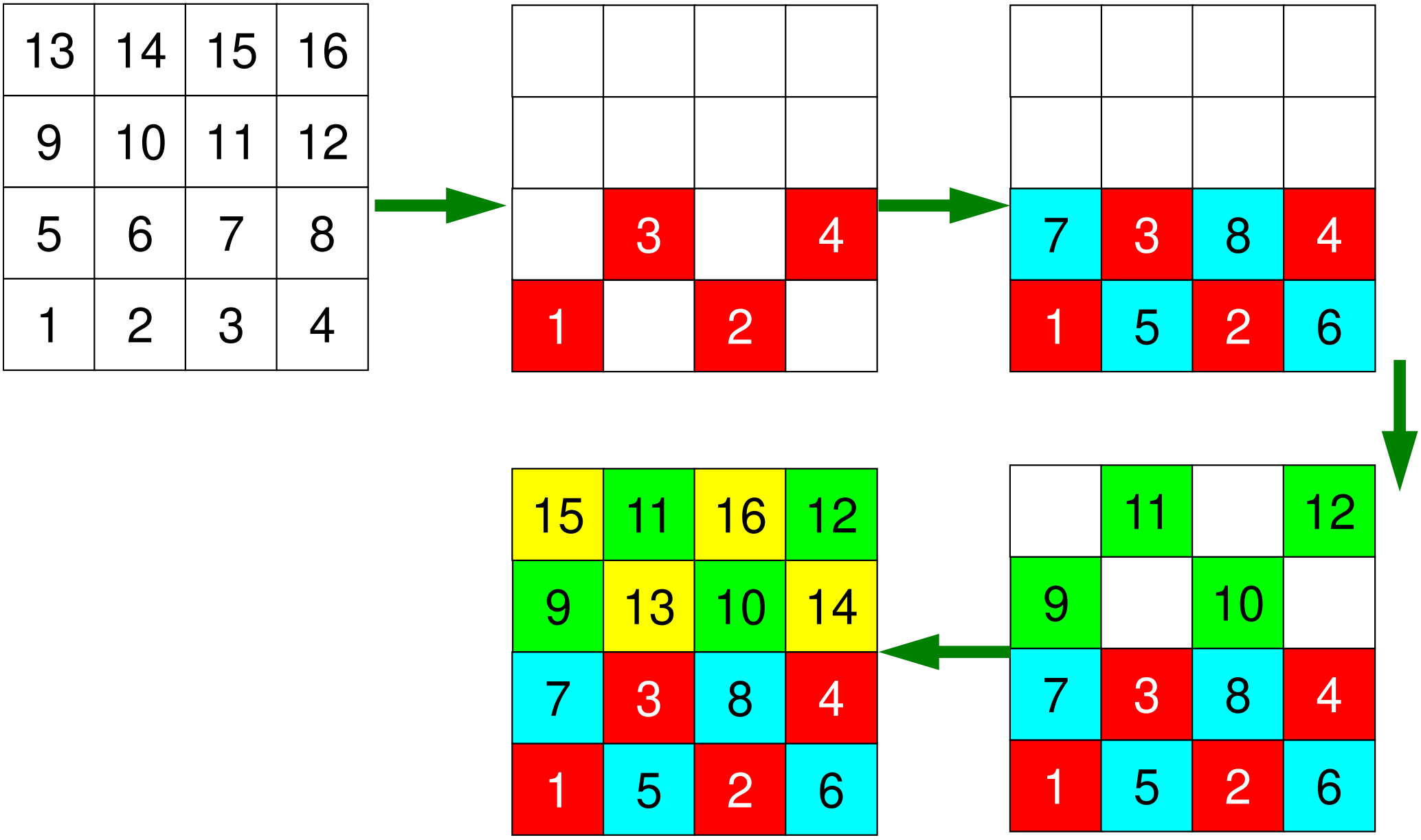
① 「要素数÷色数」を「m」とする。

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）。

MC法:4色


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

修正されたMC法

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。

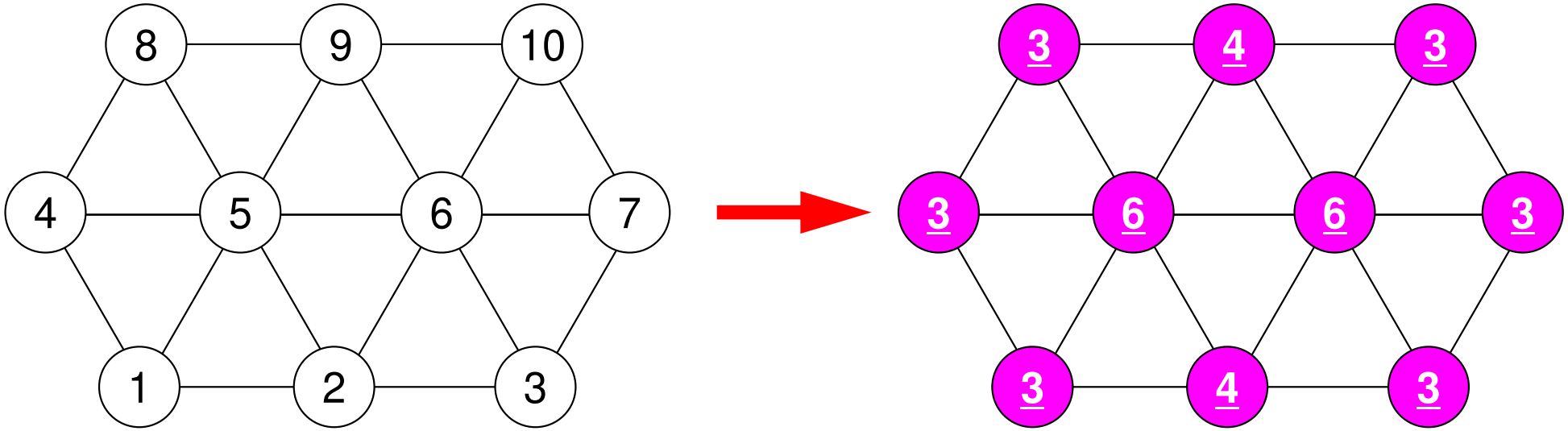
③ ITEMcou個の独立な要素を初期要素番号が若い順に選び出す。

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）。

次数(degree):各点に隣接する点の数



修正されたMC法

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。

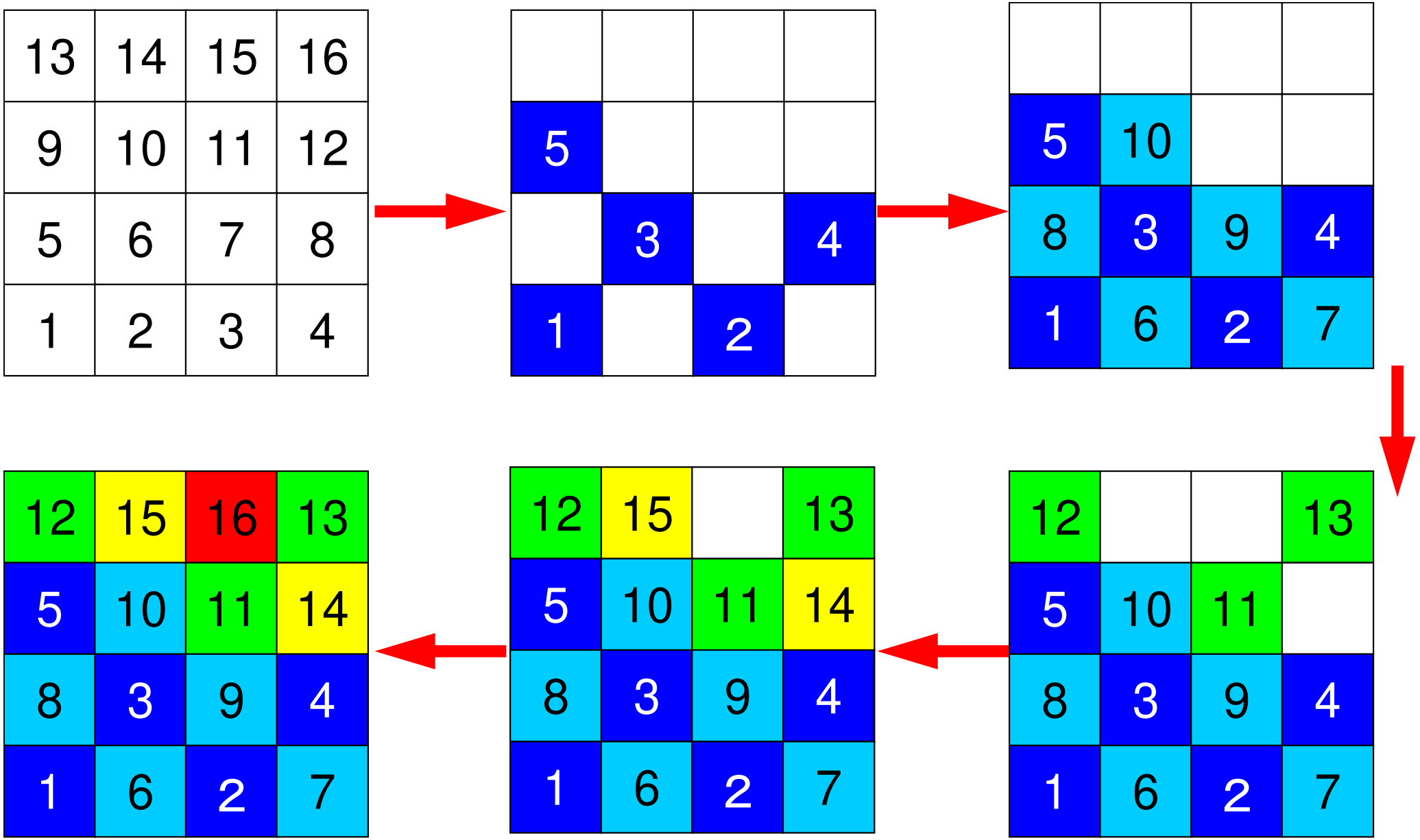
③ ITEMcou個の独立な要素を初期要素番号が若い順に選び出す。

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）。同じ色：連続した「新」要素番号

MC法：3色に設定，実は5色



並列計算のためのオーダリング法

- マルチカラーオーダリング
  - 並列性
  - Red-Black オーダリング（2色）等

- CM法（Cuthill-McKee），RCM法（Reverse Cuthill-McKee）
  - fill-inを減らす
  - マトリクスのバンド幅を減らす，プロファイルを減らす
  - 並列性

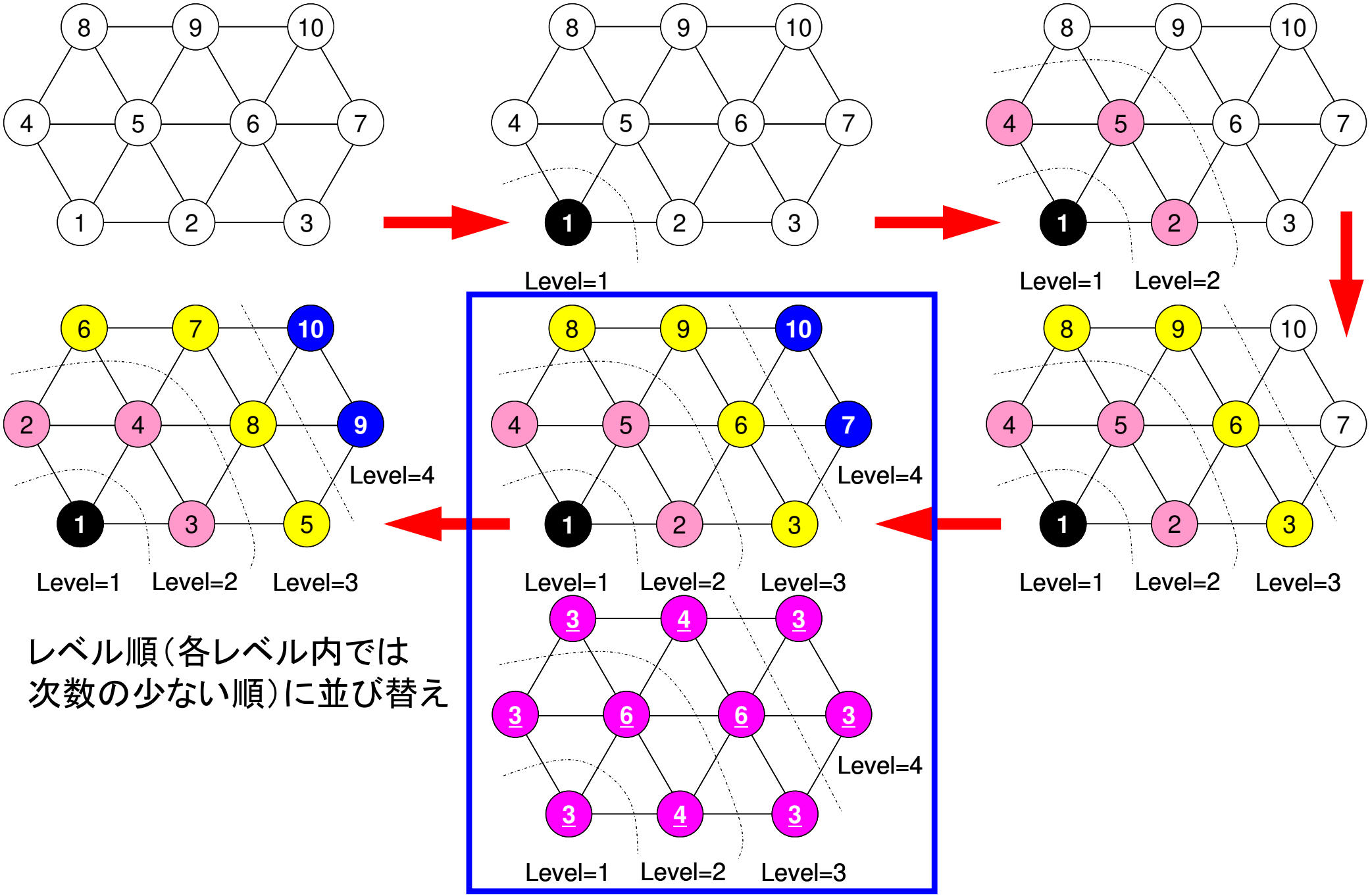
CM法の基本的なアルゴリズム

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。

③ 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）。同じ色：連続した「新」要素番号

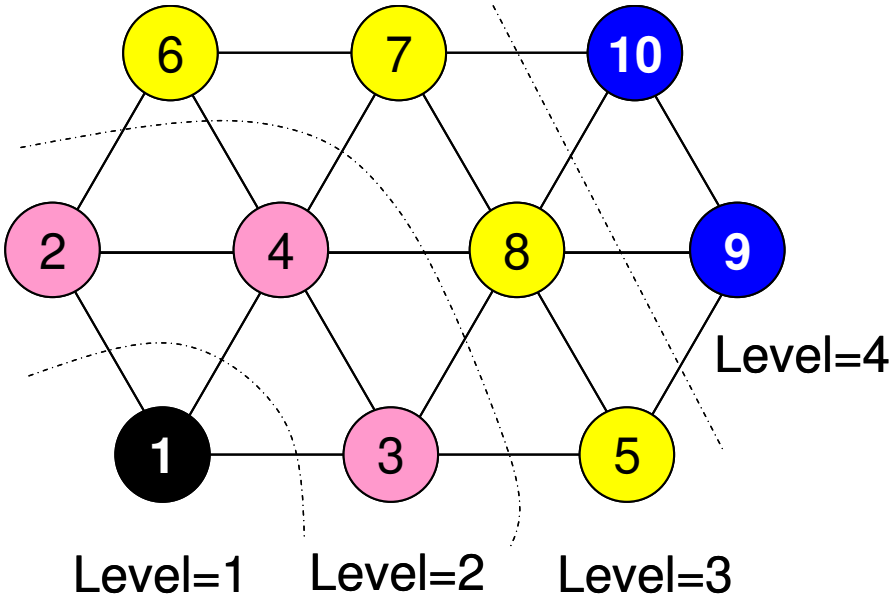
Cuthill-Mckee Ordering（CM法）の例

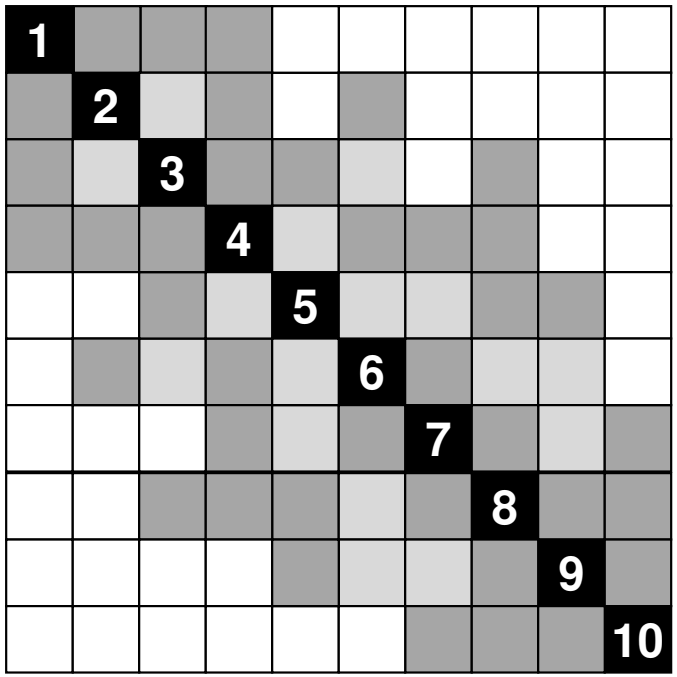


RCM（Reverse CM）

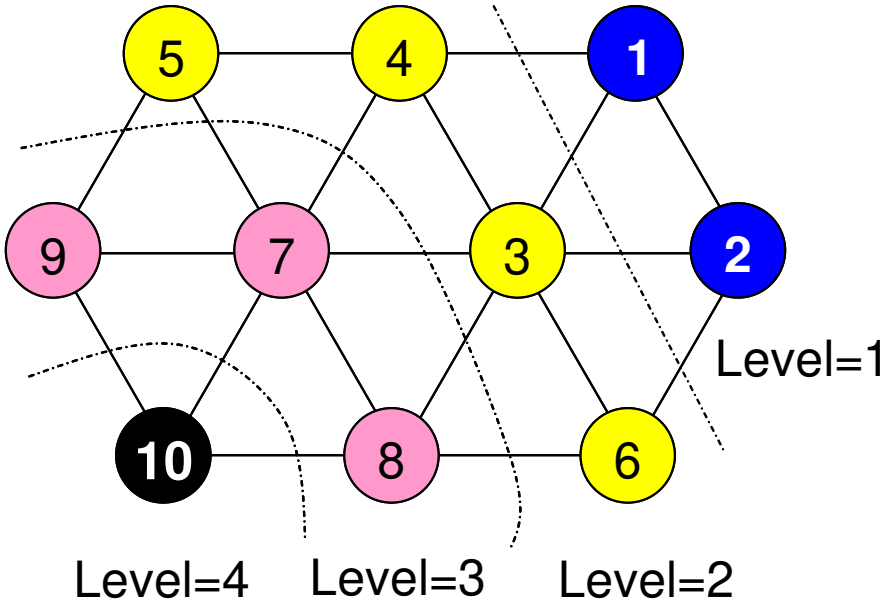
- まずCMの手順を実行
  - 次数（degree）の計算
  - 「レベル（k（k≧2））」の要素の選択
  - 繰り返し，再番号付け
- 再々番号付け
  - CMの番号付けを更に逆順にふり直す
  - Fill-inがCMの場合より少なくなる

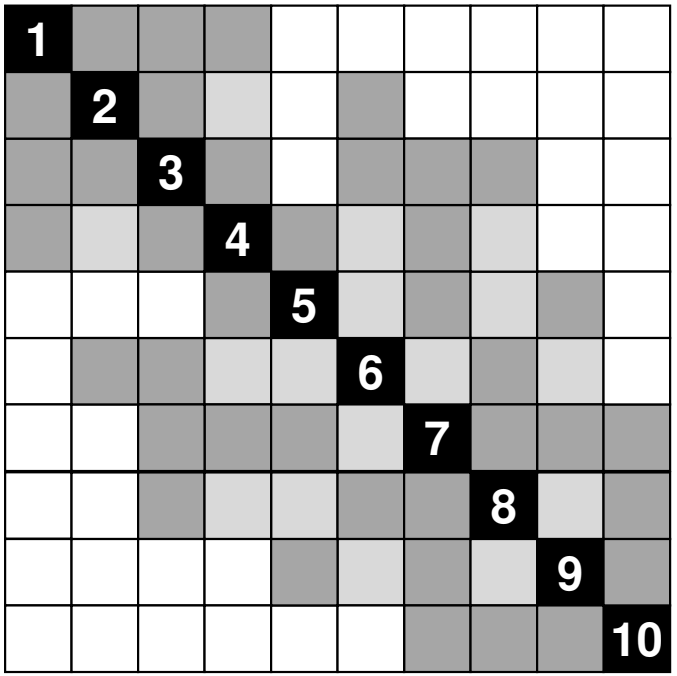
CM: 16 fill-in's





RCM: 14 fill-in's

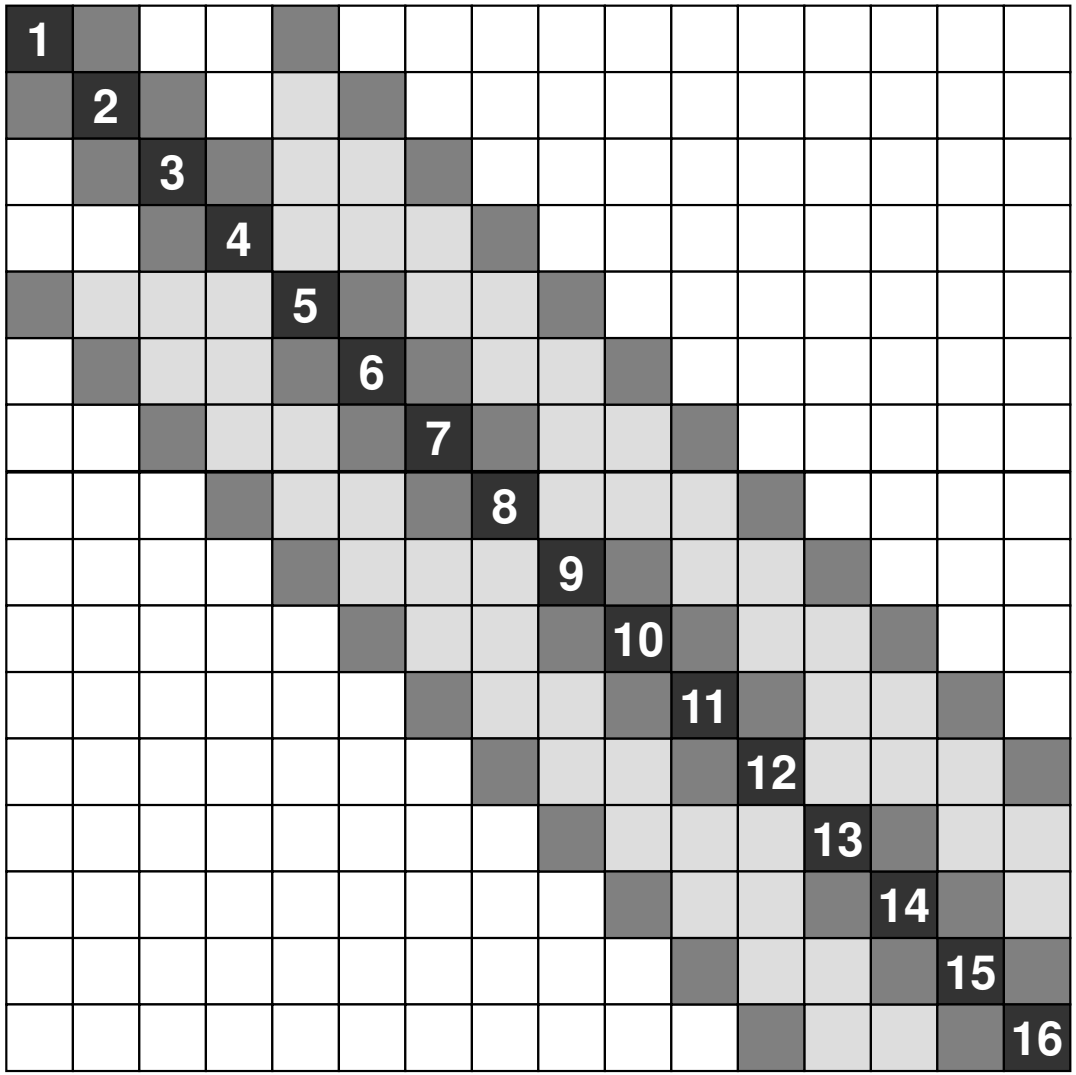




初期状態

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

バンド幅　4
プロファイル　51
Fill-in　54

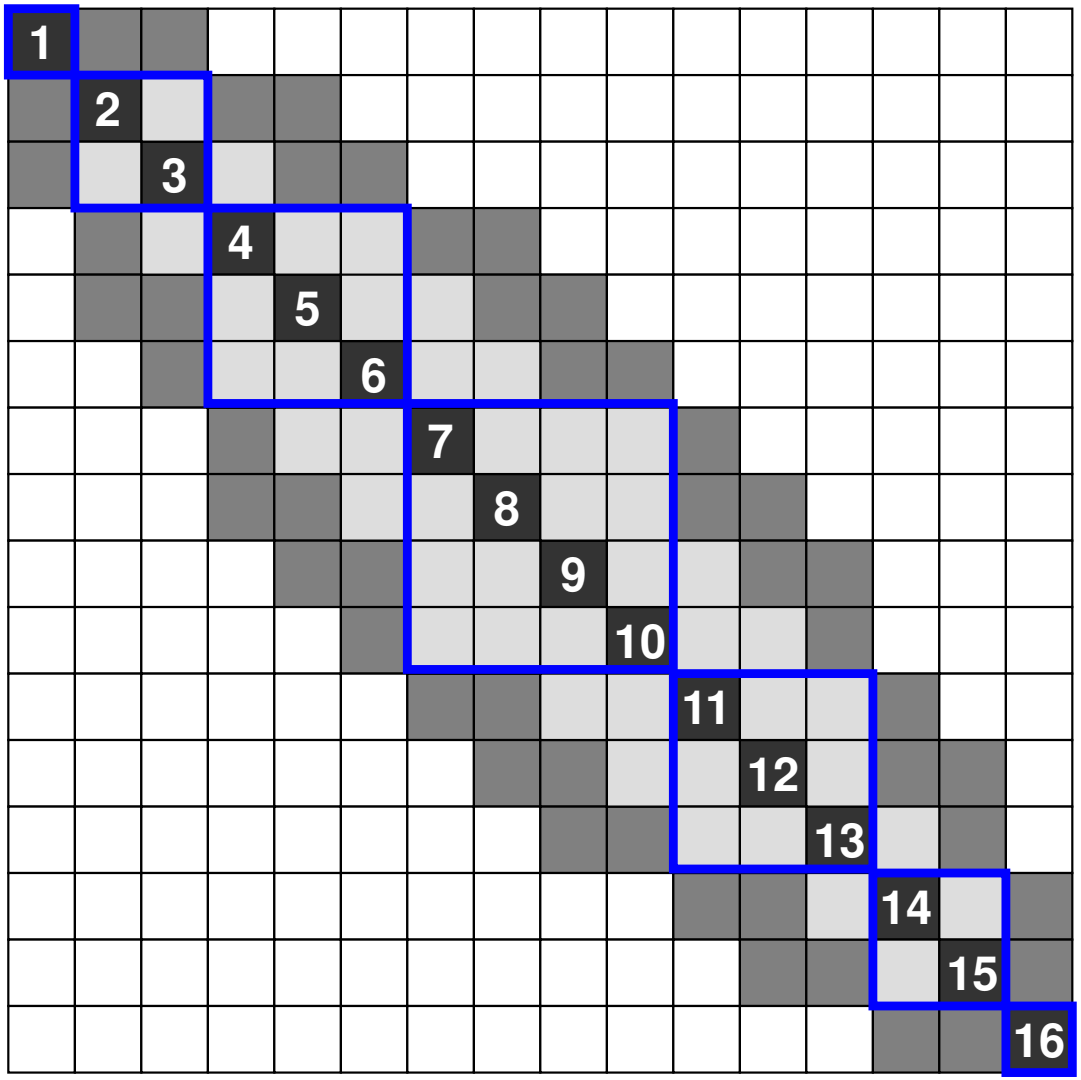


■非ゼロ成分，■**Fill-in**

CM

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

バンド幅　4
プロファイル　46
Fill-in　44

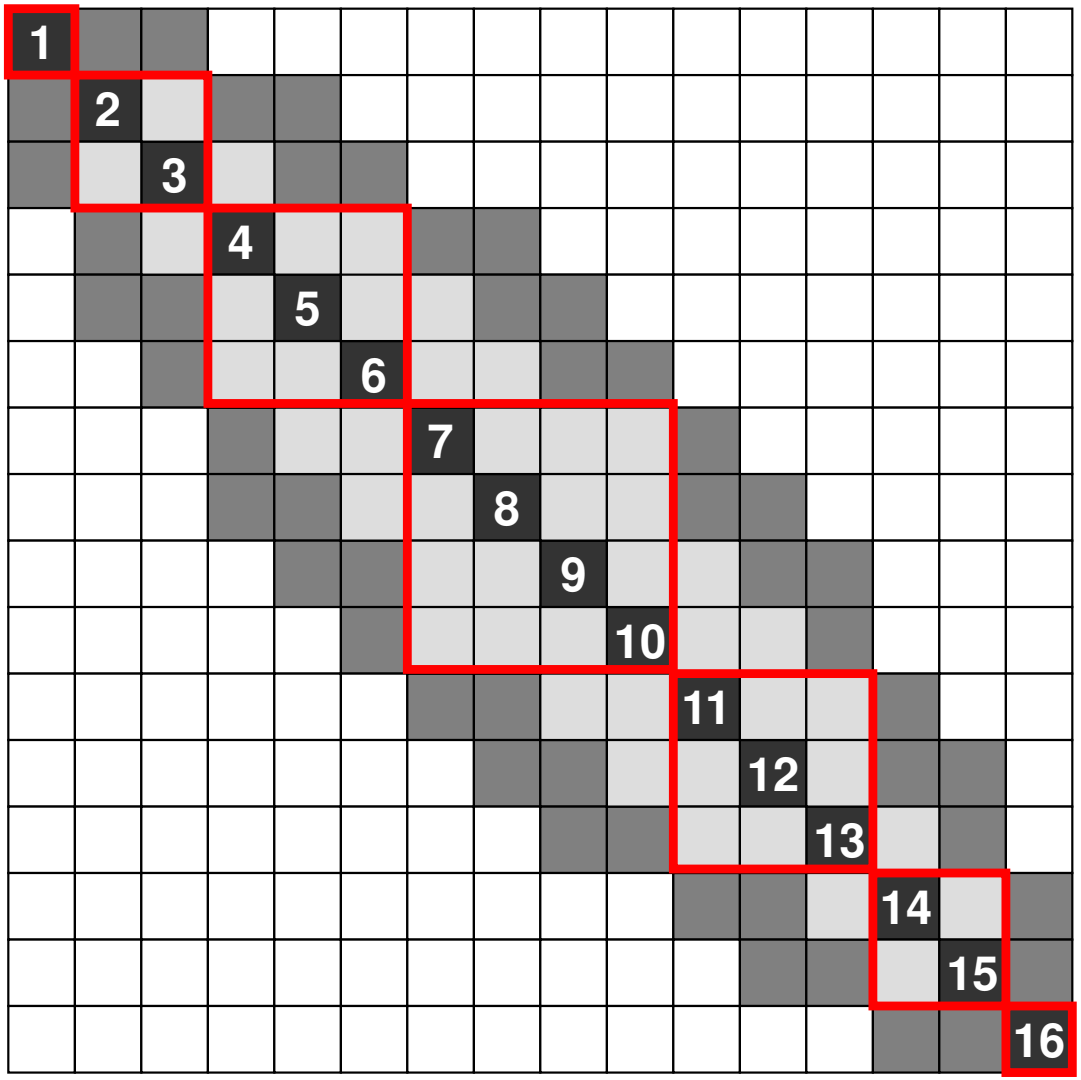


■Non-zero, ■Fill-in

RCM

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

バンド幅　4
プロファイル　46
Fill-in　44



■Non-zero, ■Fill-in

CM, RCM: Hyperline (i+j=const.)
3D: Hyperplane (i+j+k=cons.)

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

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

1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
1,1	2,1	3,1	4,1

i+j=5

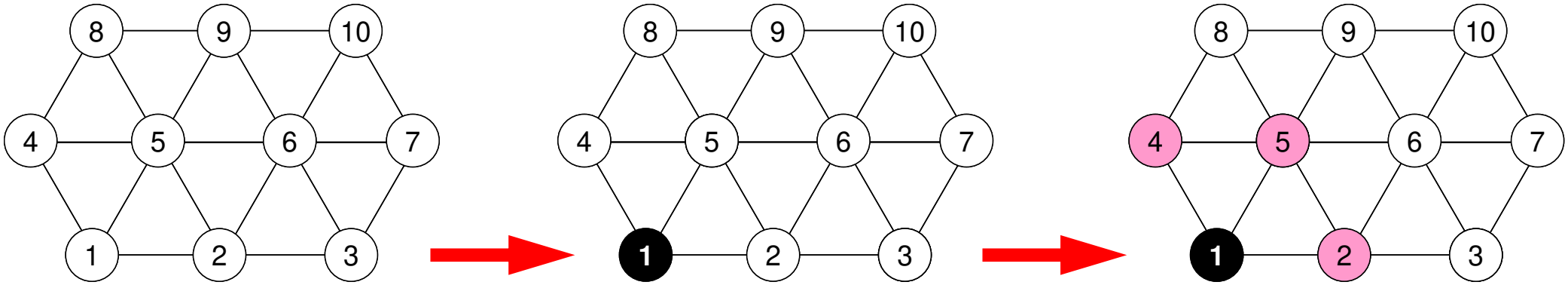
修正されたCM法
並列計算向け

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。

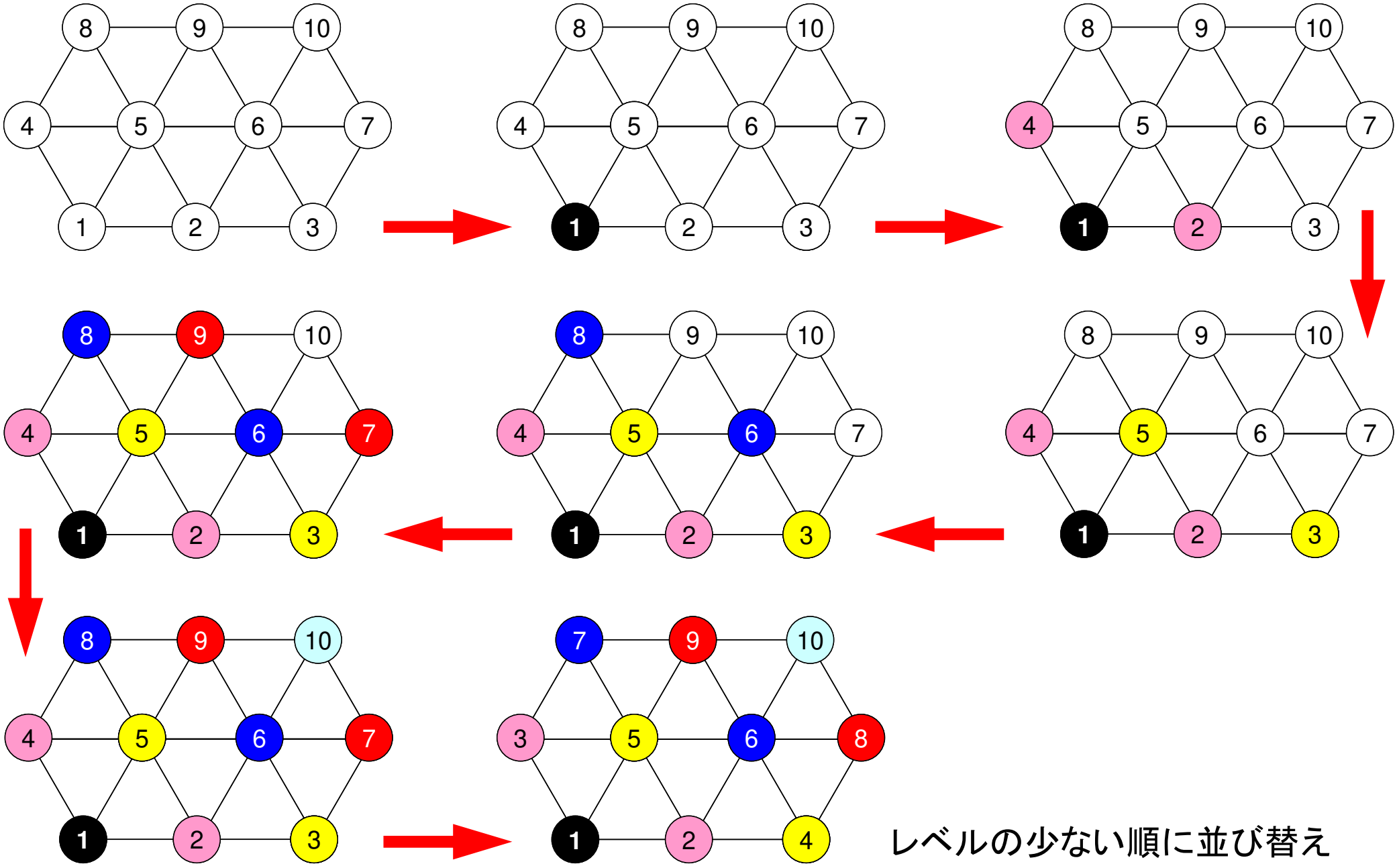
③ 全ての要素がレベルづけされたら，レベルの順番に再番号づけする。同じレベル：連続した「新」要素番号

修正されたCM法

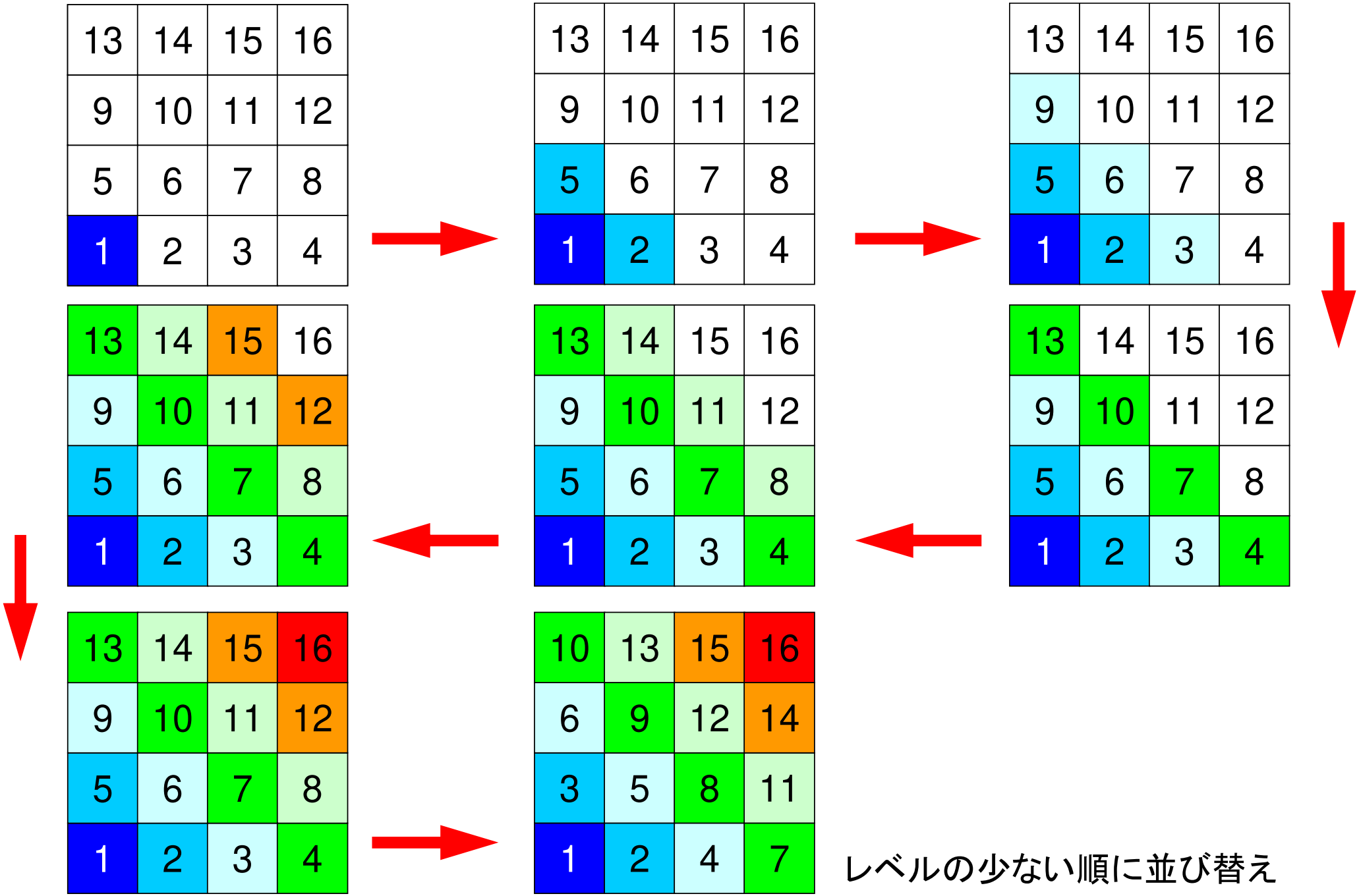


同じレベルに属する要素は
データ依存性が発生しない

修正されたCM法

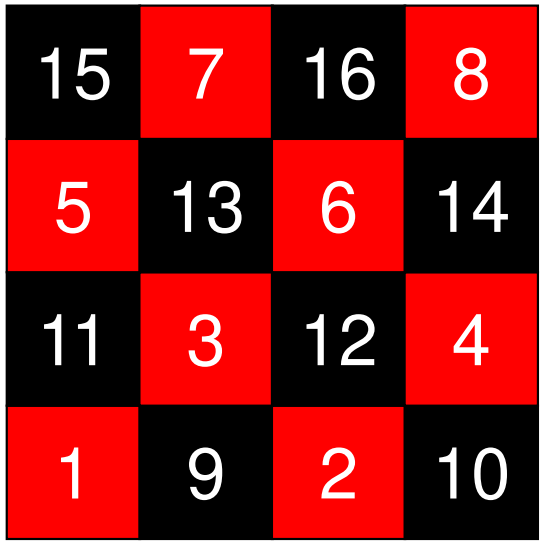

8 9 10
4 5 6 7
1 2 3
1
4
1
8 9 10
4 5 6 7
1 2 3
1 2 3
4 5 6 7
1 2 3
8 9 10
4 5 6 7
1 2 3
8 9 10
4 5 6 7
1 2 3
8 9 10
4 5 6 7
1 2 3
7 9 10
3 5 6 8
1 2 4
レベルの少ない順に並び替え

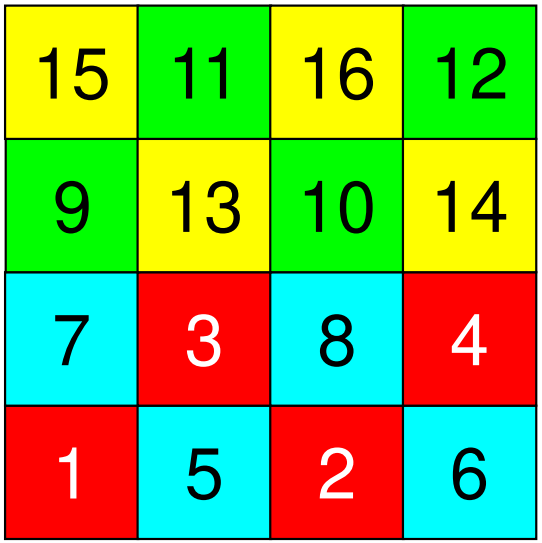
修正されたCM法

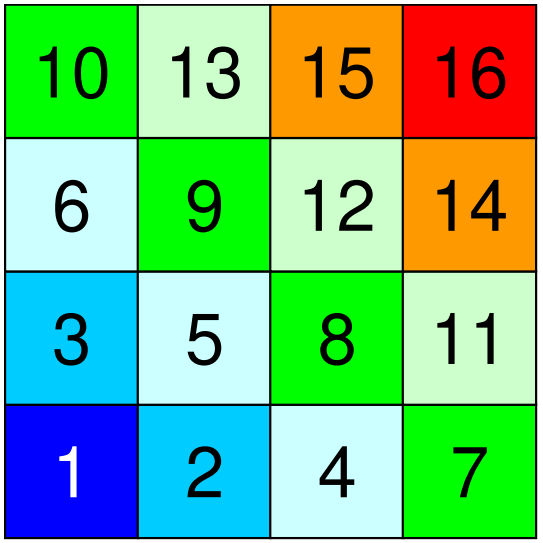

13 14 15 16
9 10 11 12
5 6 7 8
1 2 3 4
13 14 15 16
9 10 11 12
5 6 7 8
1 2 3 4
13 14 15 16
9 10 11 12
5 6 7 8
1 2 3 4
13 14 15 16
9 10 11 12
5 6 7 8
1 2 3 4
13 14 15 16
9 10 11 12
5 6 7 8
1 2 3 4
13 14 15 16
9 10 11 12
5 6 7 8
1 2 3 4
10 13 15 16
6 9 12 14
3 5 8 11
1 2 4 7
レベルの少ない順に並び替え

MC法，CM/RCM法の違い

- 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

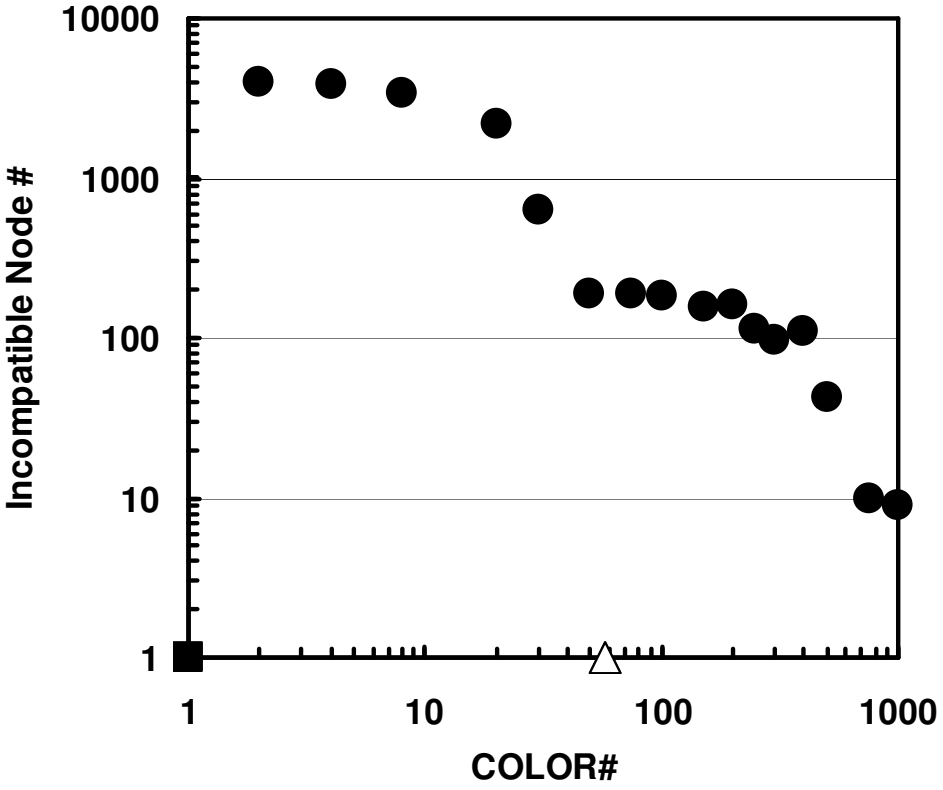


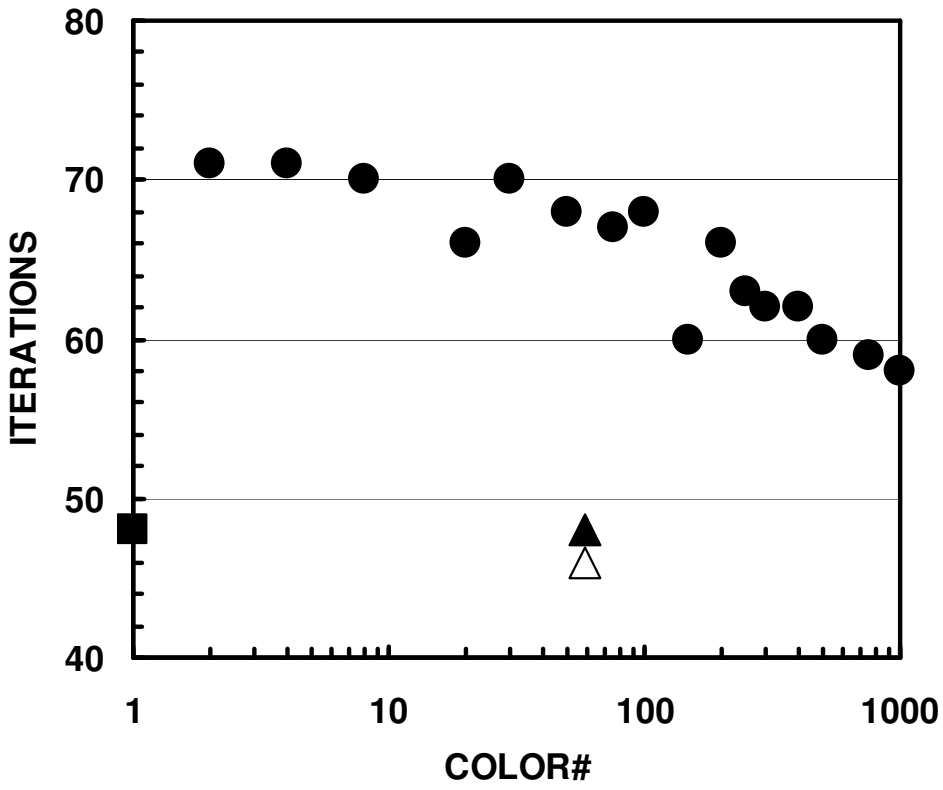




- データ依存性の解決策は?
- **オーダリング（Ordering）について**
  - Red-Black，Multicolor（MC）
  - Cuthill-McKee（CM），Reverse-CM（RCM）
  - **オーダリングと収束の関係**
- オーダリングの実装
- オーダリング付ICCG法の実装
- マルチコアへの実装（OpenMP）へ向けて

ICCG法の収束（後述）





（20^3=8,000要素，EPSICCG=10^{-8}）

（■：ICCG(L1)，●：ICCG-MC，▲：ICCG-CM，△：ICCG-RCM）

収束とオーダリングの関係(後述)

- 要素数＝20^3
- Red-Black ～ 4色 ＜ 初期状態 ～ CM, RCM

初期状態	
バンド幅	4
プロファイル	51
Fill-in	54

Red-Black	
バンド幅	10
プロファイル	77
Fill-in	44

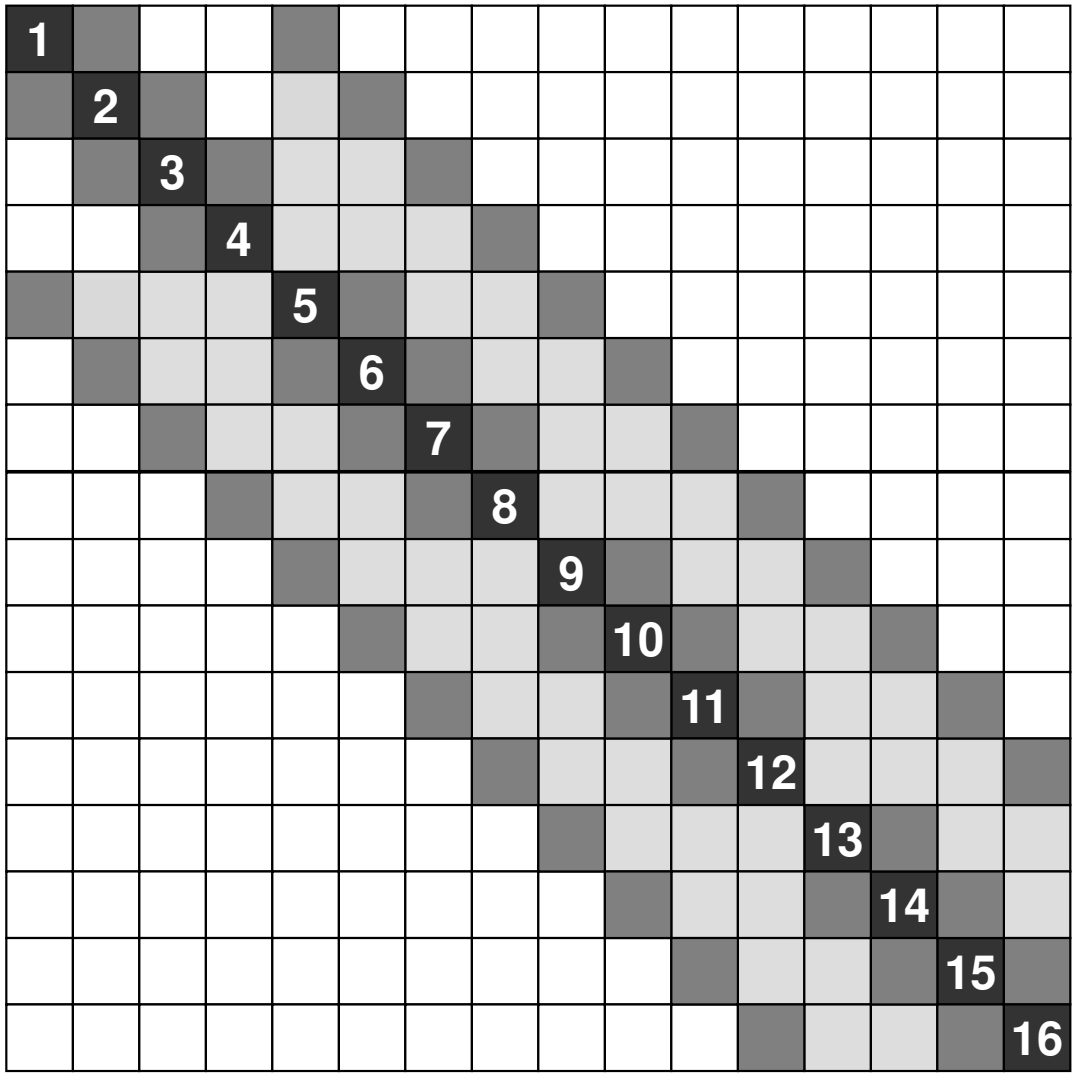
4色	
バンド幅	10
プロファイル	57
Fill-in	46

CM, RCM	
バンド幅	4
プロファイル	46
Fill-in	44

Initial Matrix

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

バンド幅　　4
プロファイル　51
Fill-in　　54

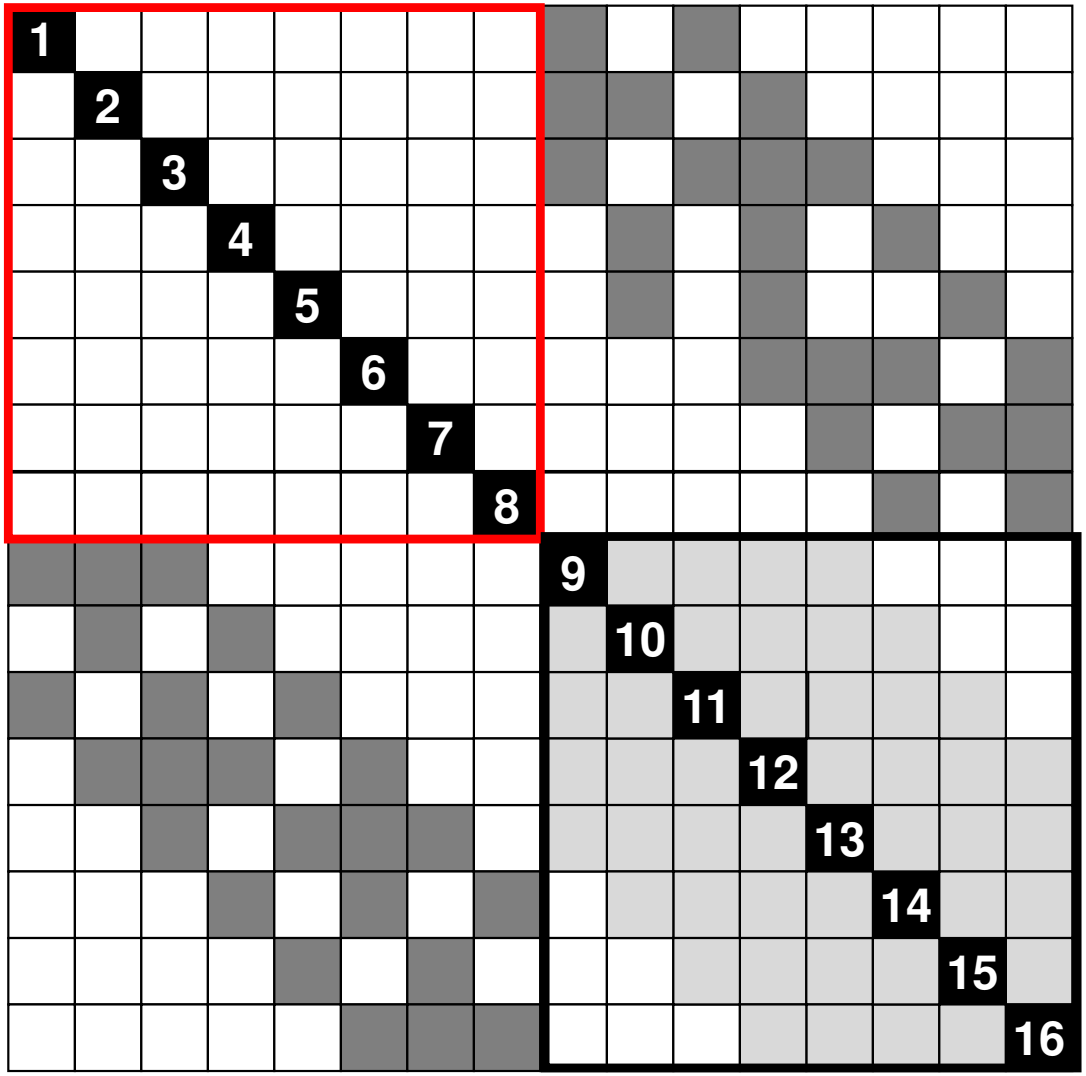


■**Non-zero,** ■**Fill-in**

Red-Black (2-Colors)

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

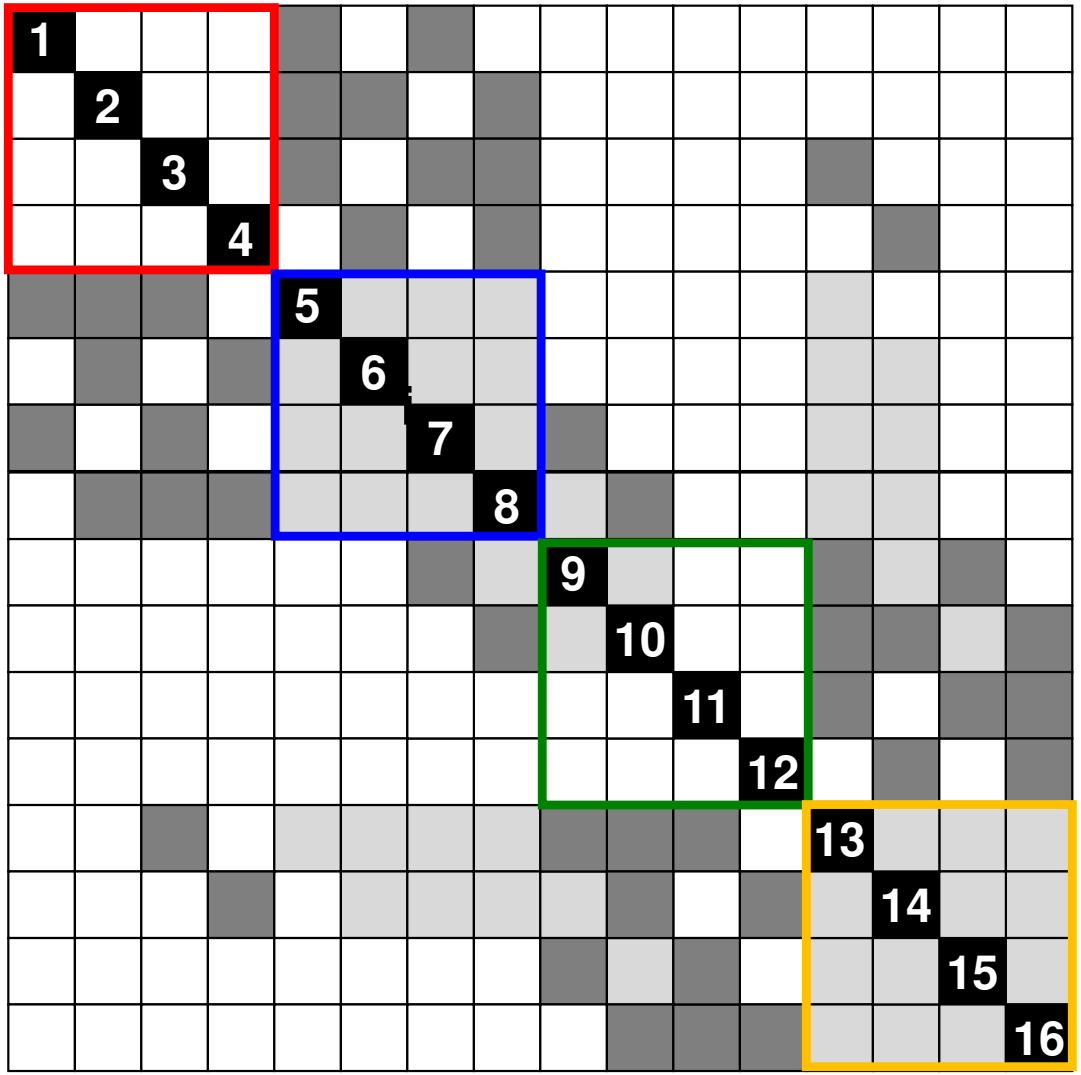
バンド幅　10
プロファイル　77
Fill-in　44



Non-zero, Fill-in

MC (4-Colors)

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



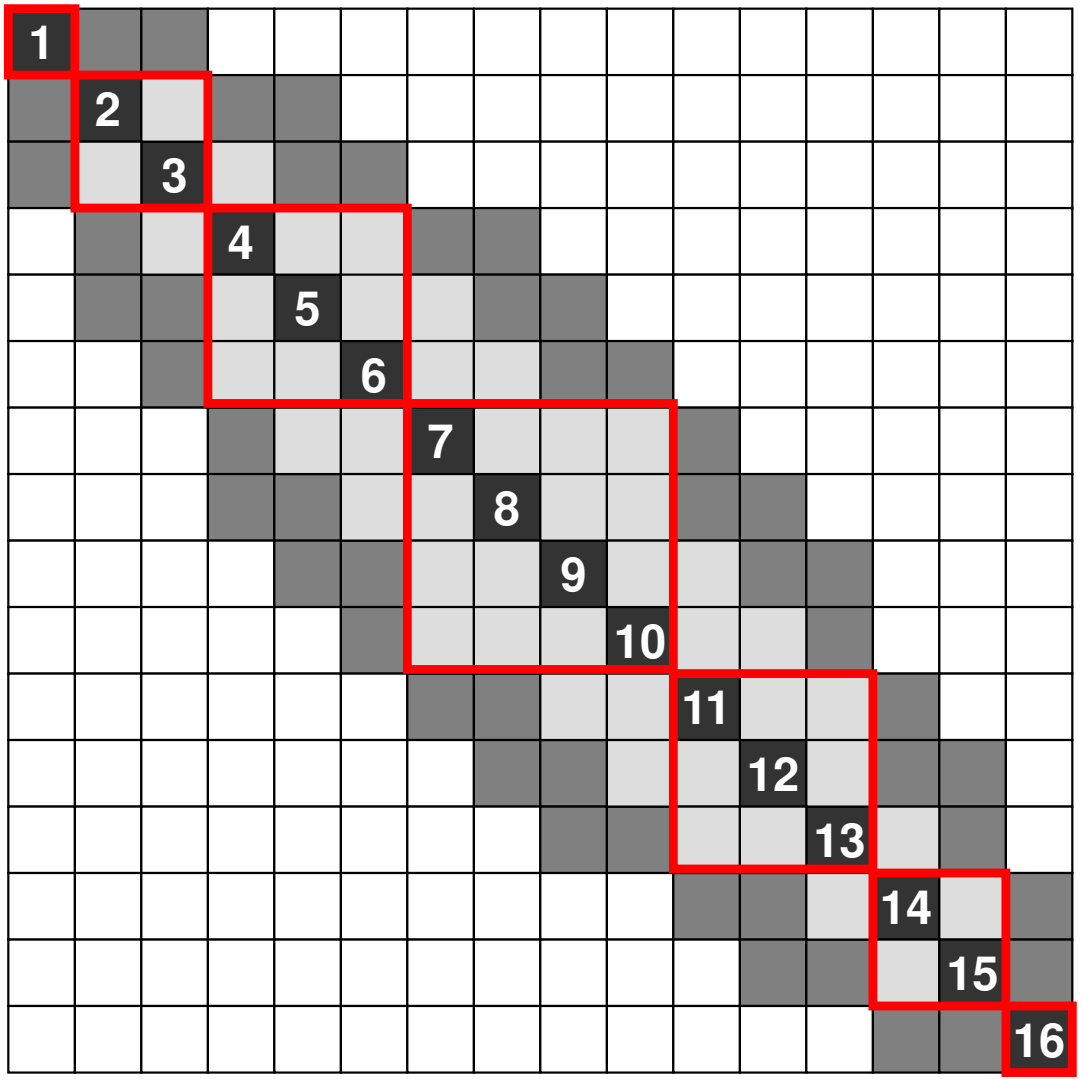
バンド幅	10
プロファイル	57
Fill-in	46

Non-zero, Fill-in

RCM

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

バンド幅　4
プロファイル　46
Fill-in　44

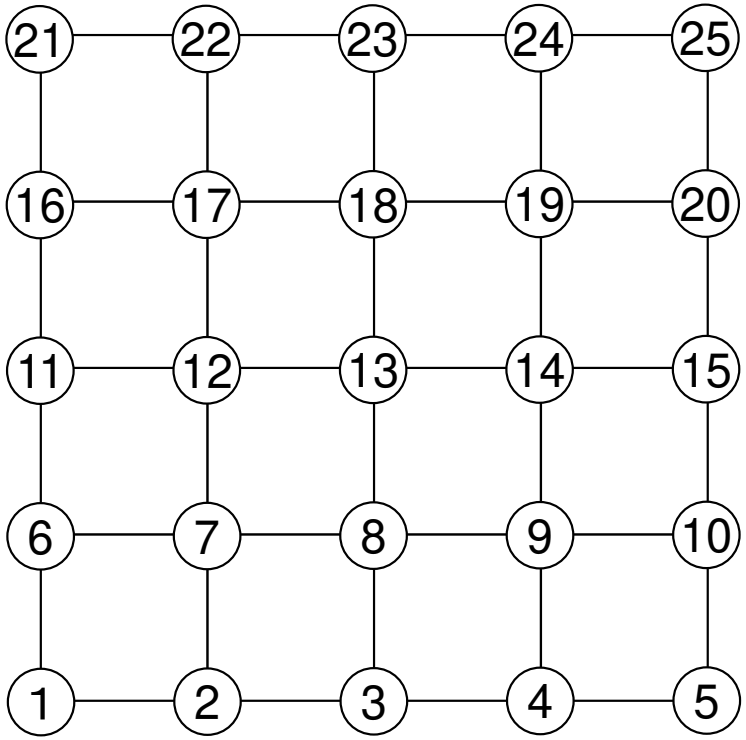


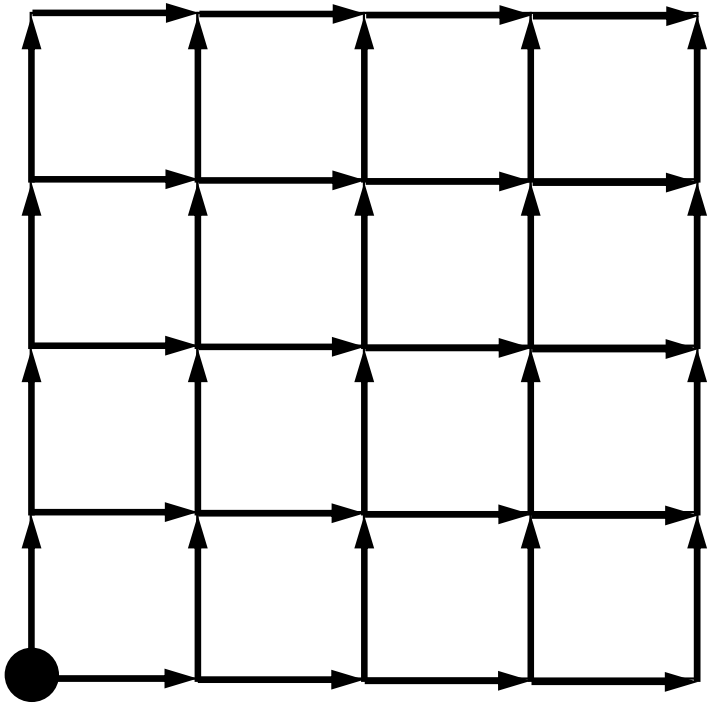
■Non-zero, ■Fill-in

反復回数と色数の関係
Incompatible Nodesとは?

Doi, S. (NEC) et al.

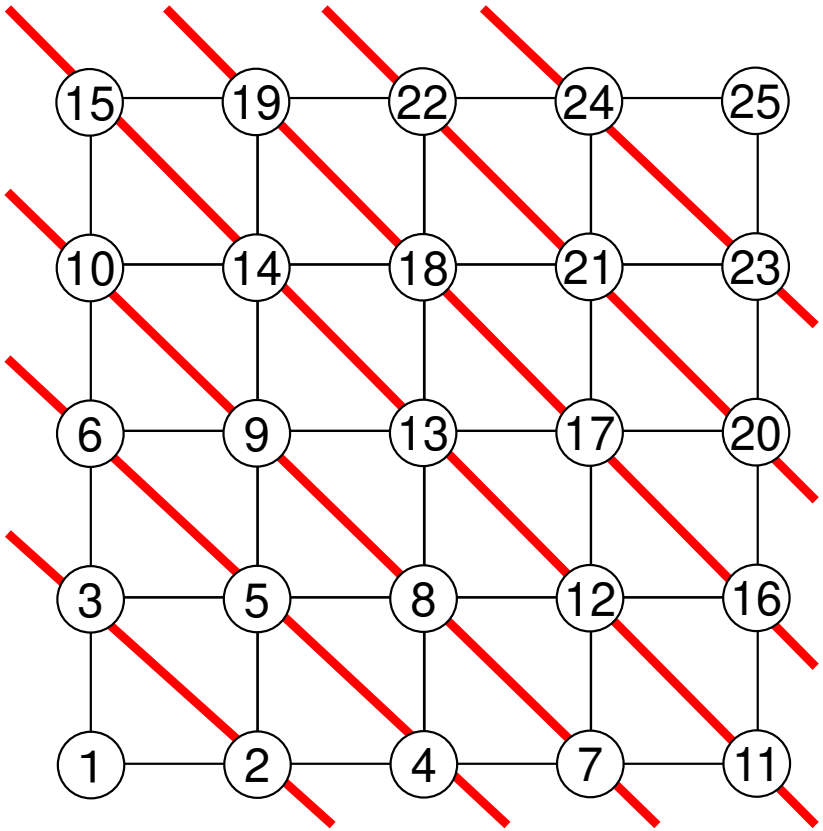
前進代入における
影響の伝わり方

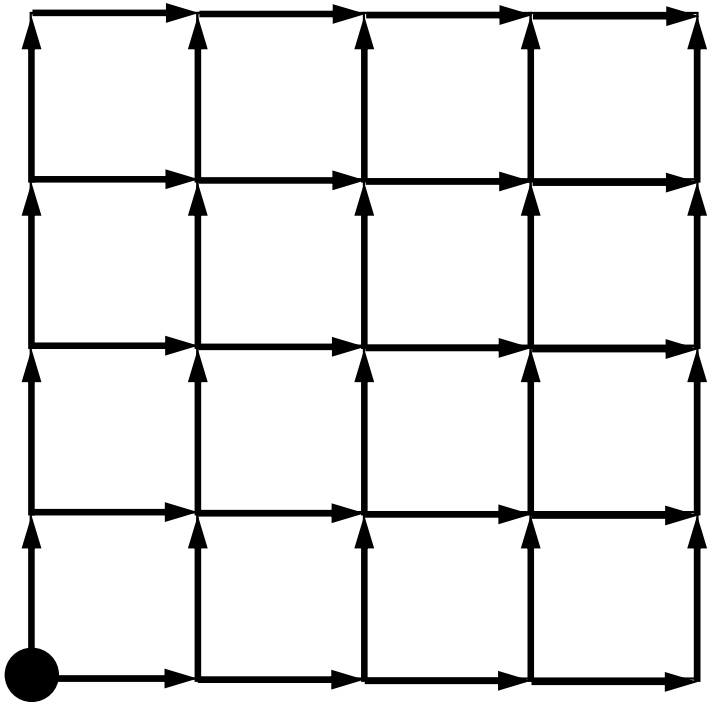




他から影響を受けない点が「Incompatible Node」
周囲の全ての点よりも番号が若い，ということ
少ない方が収束が早い

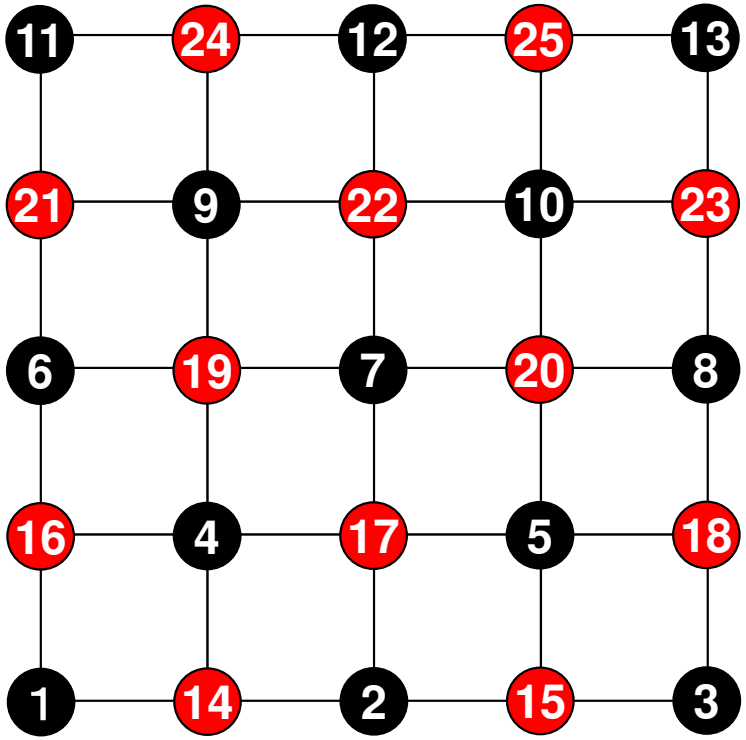
CM（Cuthill-McKee）の場合

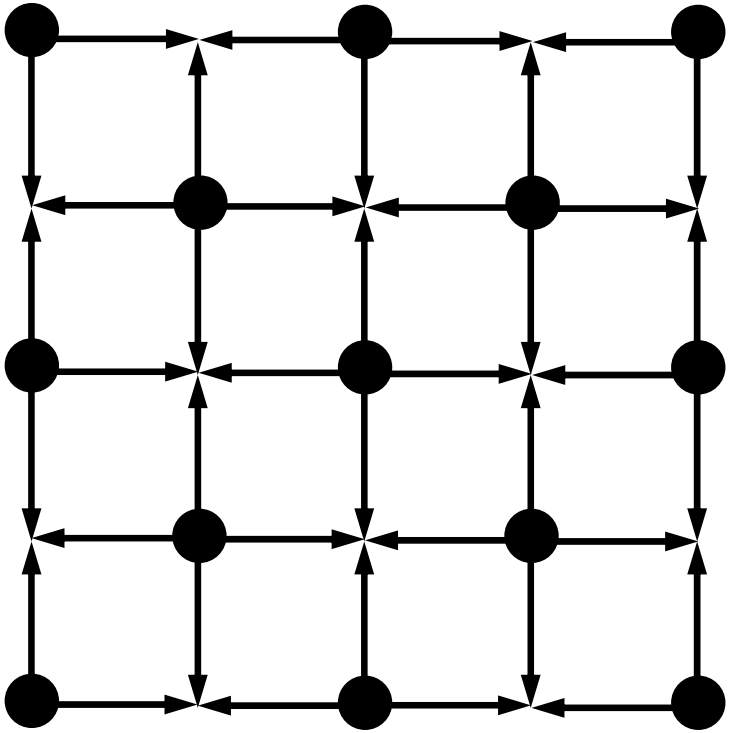




Red-Blackの場合

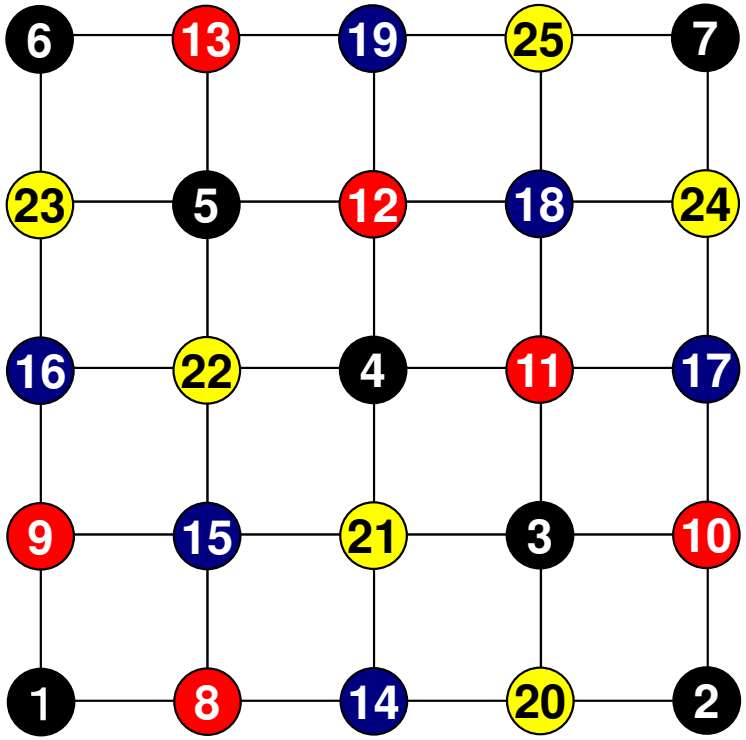
並列性は高いがincompatible node多い
ILU系前処理，Gauss-Seidelで反復回数増加

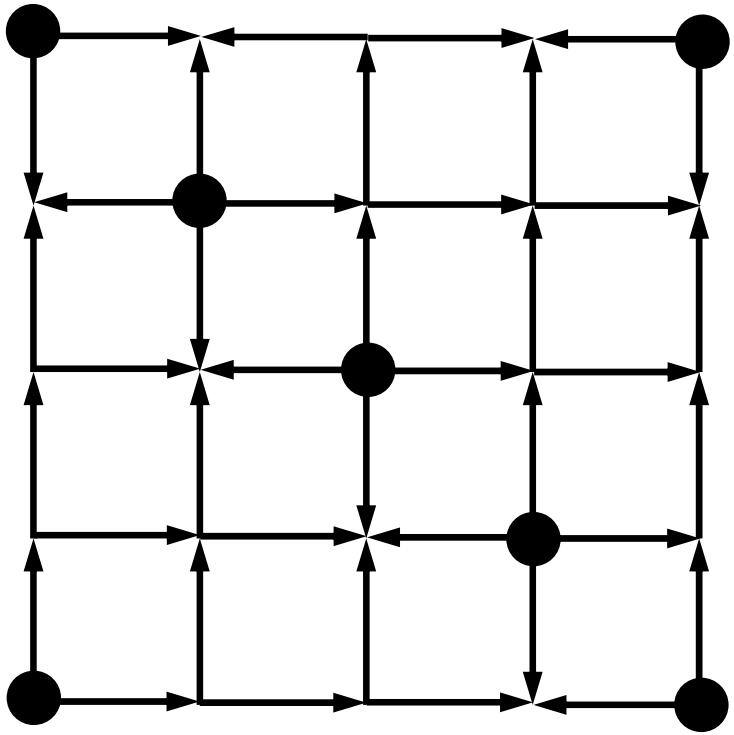




4色の場合

並列性は弱まるがincompatible nodeは減少
ILU系前処理，Gauss-Seidelで反復回数減少





一般に，下記の条件が満たされるとICCG法の収束は改善される

- 色数多い

- バンド幅小
- Profile小
- Fill-in少ない
- Incompatible Node少ない

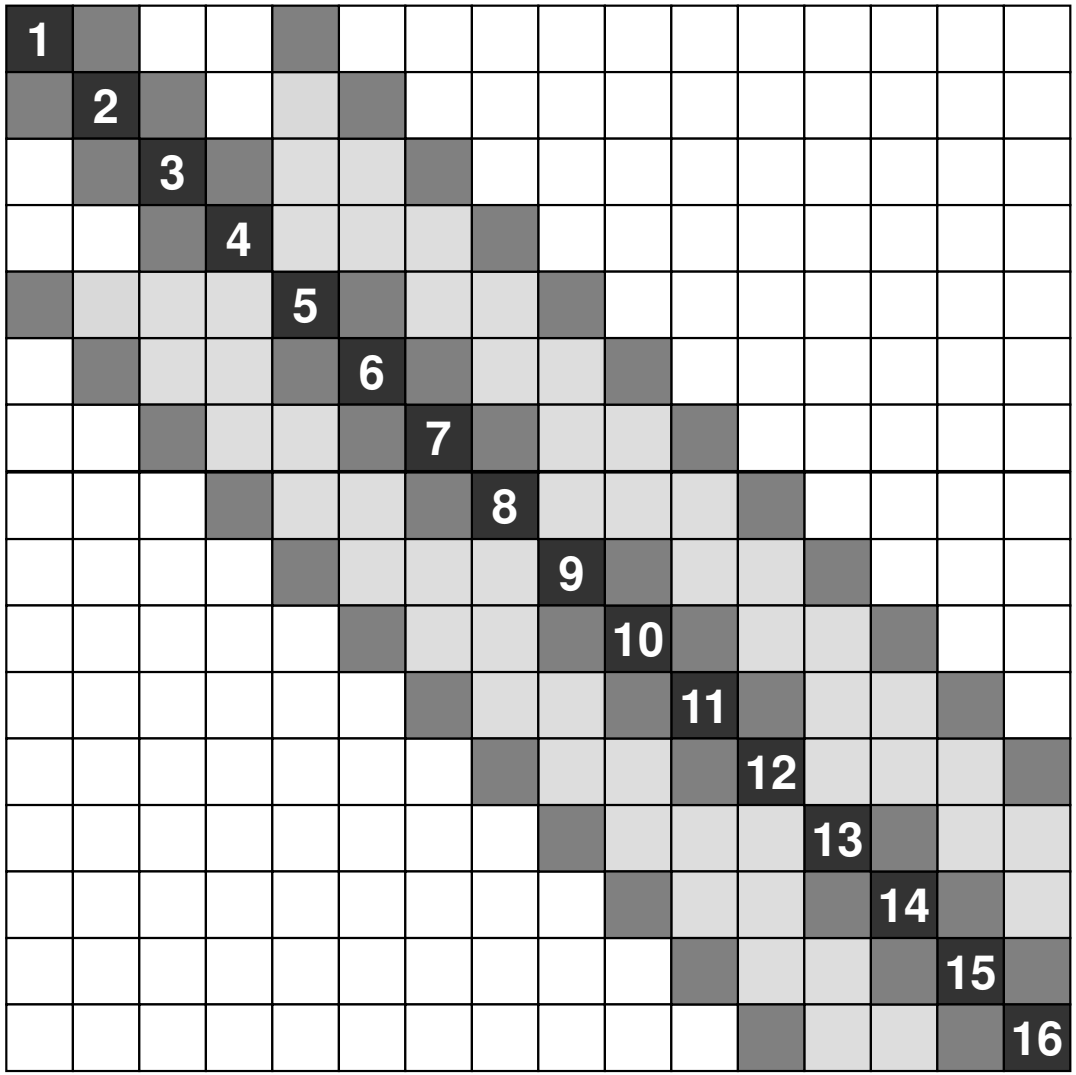
一般にICCG法の収束性は，fill-inが少ない色づけの場合ほど良くなる（はず）

- ICCG (IC(0)-CG) はそもそもFill-inを無視している
- **Fill-inが多くなる色づけでは，より多くのFill-inが無視される**
  - **Fill-inが多くなる色づけにおけるIC(0)は，Fill-inが少ない色づけと比較して「弱い」**
- **Fill-inの分布も収束に影響を与える**
  - **初期行列**
  - **Red-Black（2色）**

Initial Matrix

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

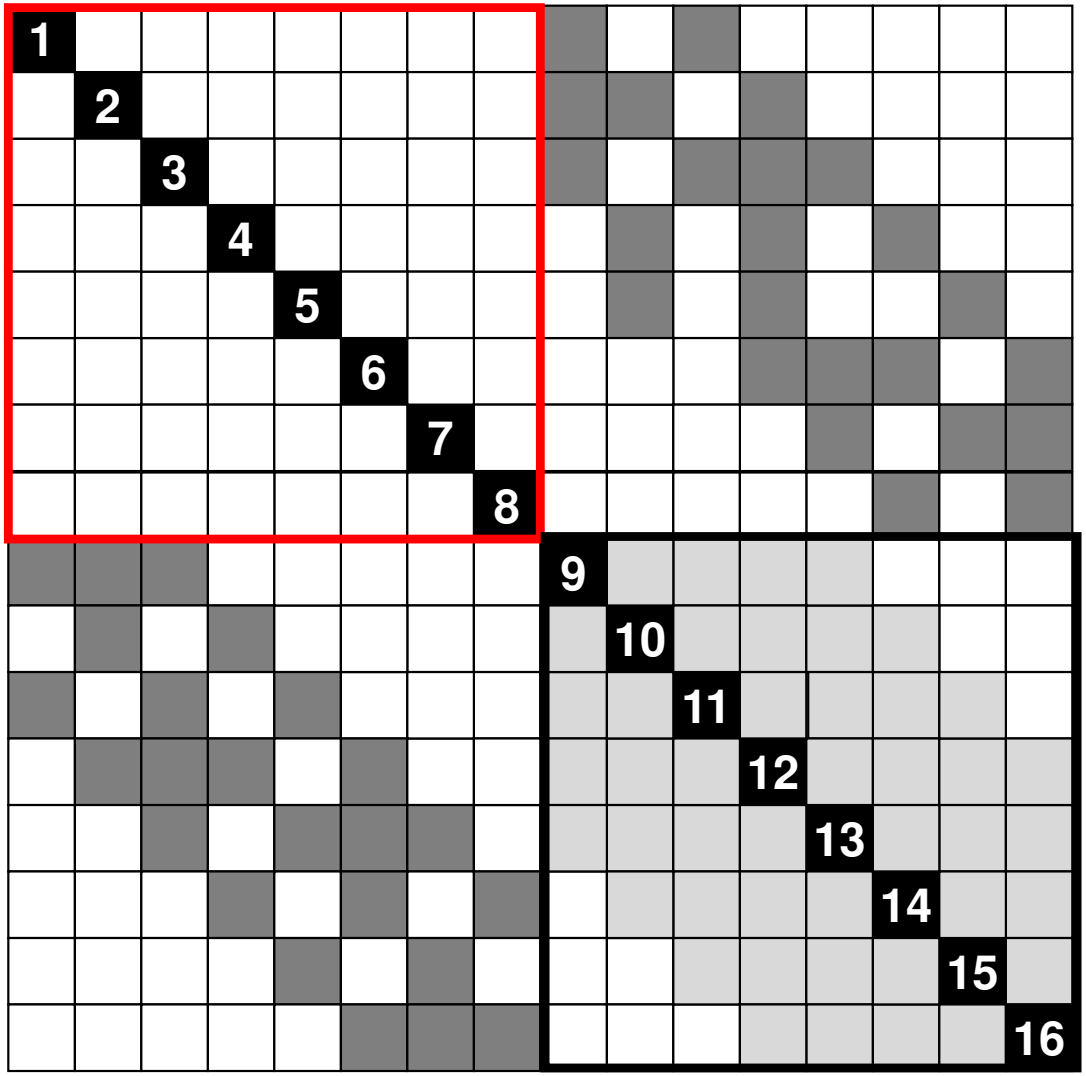
バンド幅　　4
プロファイル　51
Fill-in　　54



■**Non-zero,** ■**Fill-in**

Red-Black (2-Colors)

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



バンド幅	10
プロファイル	77
Fill-in	44

Non-zero, Fill-in

色数の収束に対する影響

- 色数だけで決まるわけではない：境界条件等，他の点についても考慮する必要がある
- 一般的な結論を導くことは困難
- たとえば今回扱っている例では，RCMとCMはフィルイン，プロファイル，バンド幅等全く同じパラメータ値となるにも関わらずRCMの収束が若干速い

オーダリングの効果

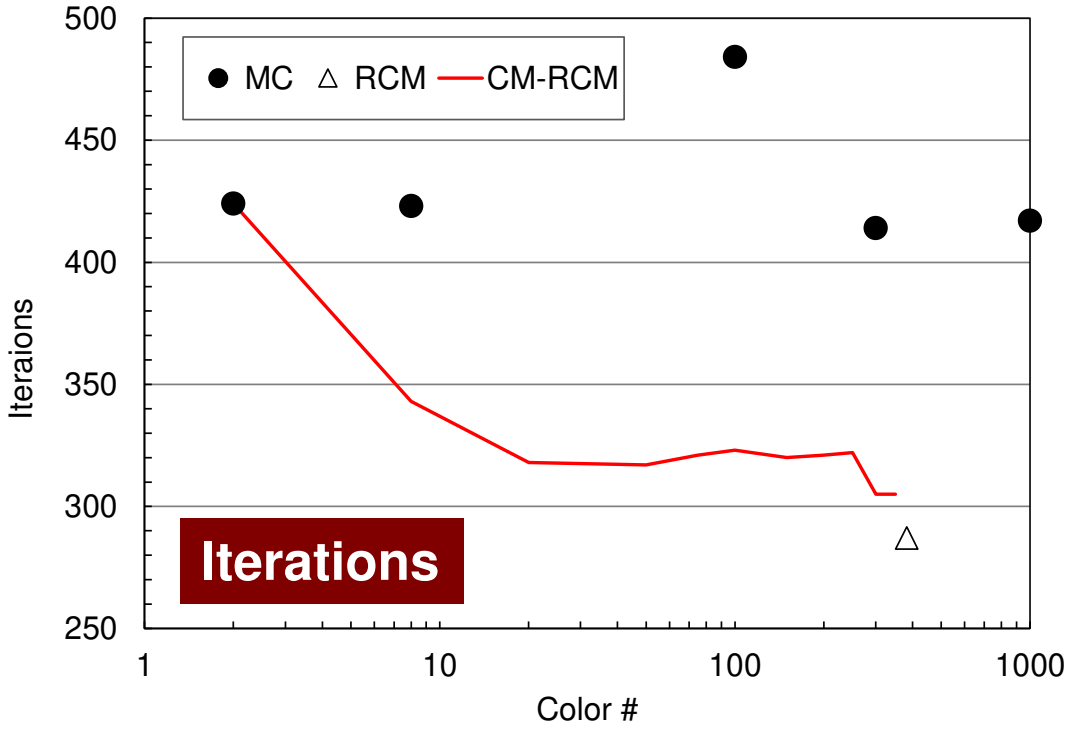
- オーダリングによって，行列の処理の順番が変わり，何らかの「改善」が得られる。
  - 並列性の付与：並列計算，ベクトル計算への適合性
  - 収束が早くなる場合もある。
- 例に示したような単純な形状ですら，オーダリングをしなければ，並列化，ベクトル化できない。

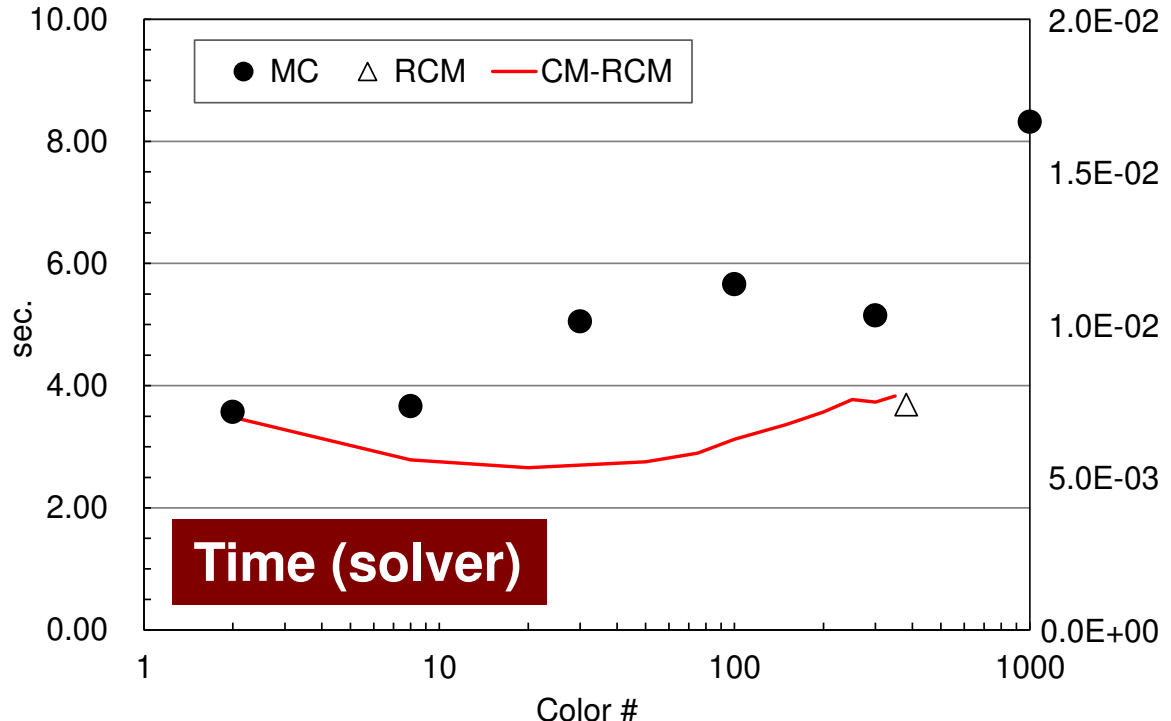
- 注意点
  - オーダリングによって答えが変わることもある。
  - 対象としている物理，数学に関する深い知識と洞察を要する。

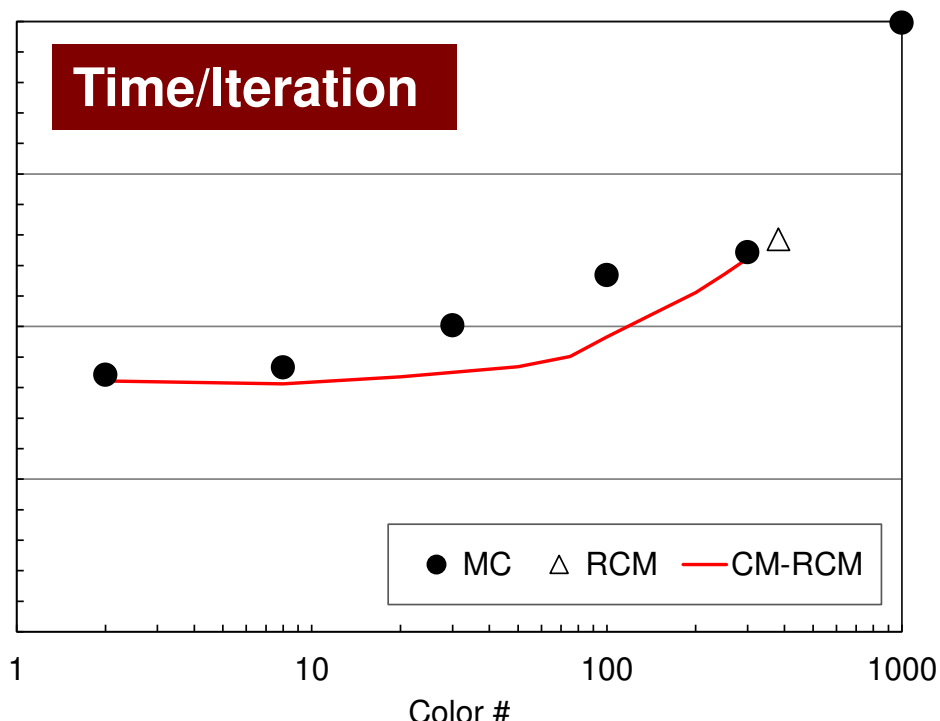
Odyssey

1-CMG/12-cores, 128^3

(●:MC, △:RCM, -:CM-RCM)

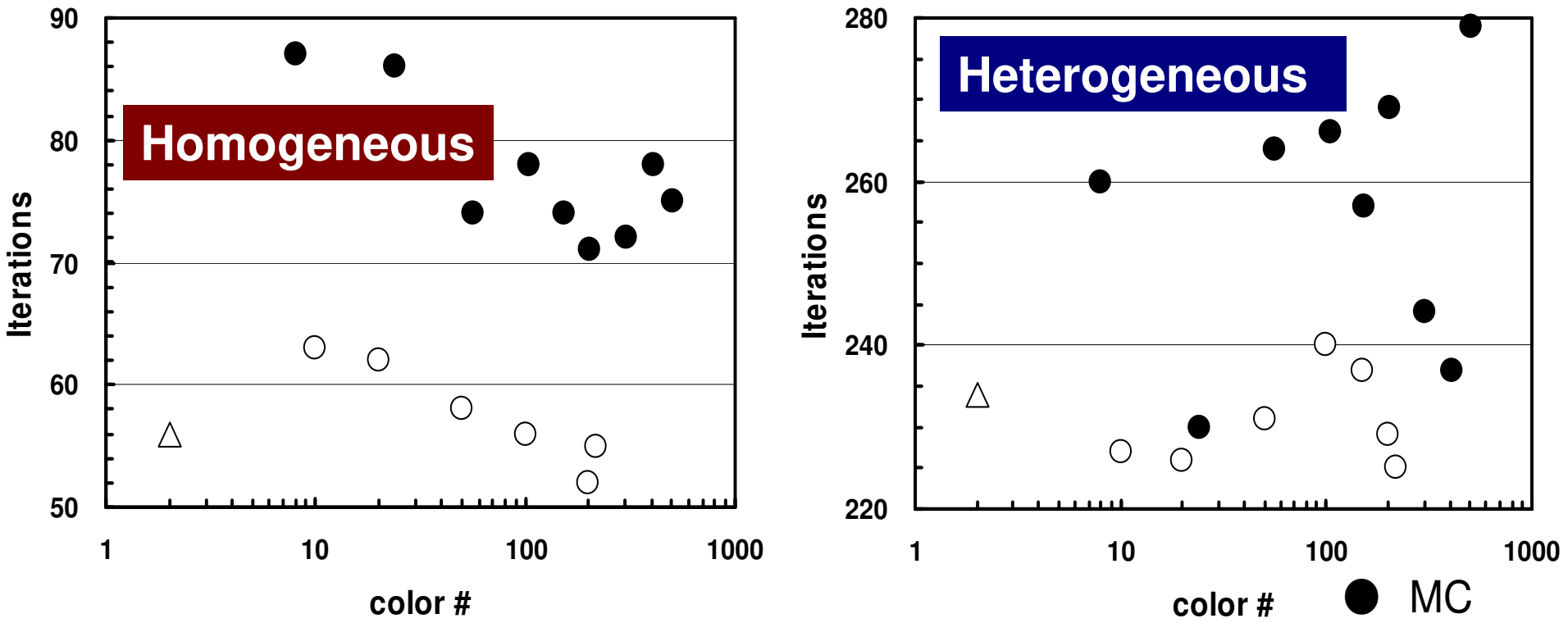






オーダリング手法の比較
三次元弾性問題

- MCは収束遅い，不安定（特に不均質（悪条件）問題）
- Cyclic-Mulricoloring + RCM（CM-RCM）が有効（後述）
  〔Washio et al. 2000〕



3D Linear-Elastic Problems with 32,768 DOF

- データ依存性の解決策は?
- オーダリング(Ordering)について
  - Red-Black, Multicolor(MC)
  - Cuthill-McKee(CM), Reverse-CM(RCM)
  - オーダリングと収束の関係
- **オーダリングの実装**
- オーダリング付ICCG法の実装
- マルチコアへの実装(OpenMP)へ向けて

オーダリング実装：L2-color

実習

- 三次元形状（ここでは二次元）の色づけのプログラム
  - マルチカラーオーダリング，CM法，RCM法（CMRCMについてはあとで）

```
$ cd <$P-L2>/coloring/src
$ make
$ cd ../run
$ ./L2-color
   NX/NY/NZ ?

4    4    1      ← このように入力する
You have        16 elements.
How many colors do you need ?
   #COLOR must be more than 2 and
   #COLOR must not be more than       16
    CM if #COLOR .eq. 0
   RCM if #COLOR .eq.-1
CMRCM if #COLOR .le.-2
=>
```

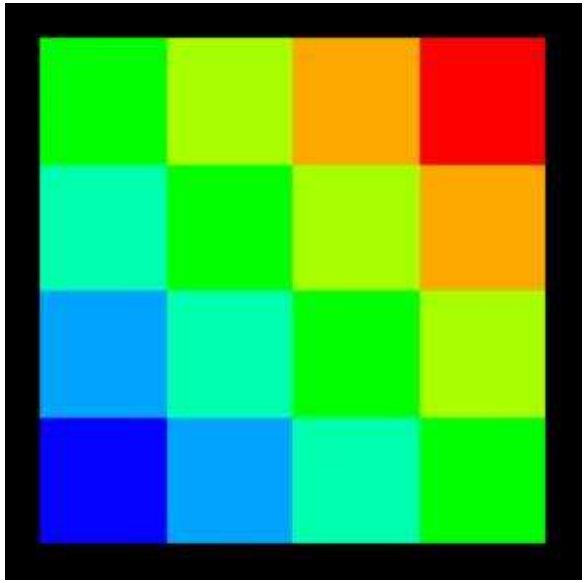
この2次元形状を接続関係に基づき色づけする。

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

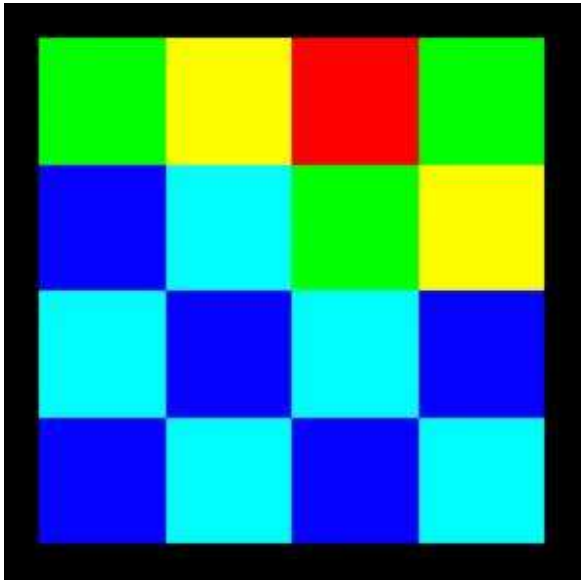
実習

実施内容(2/2)

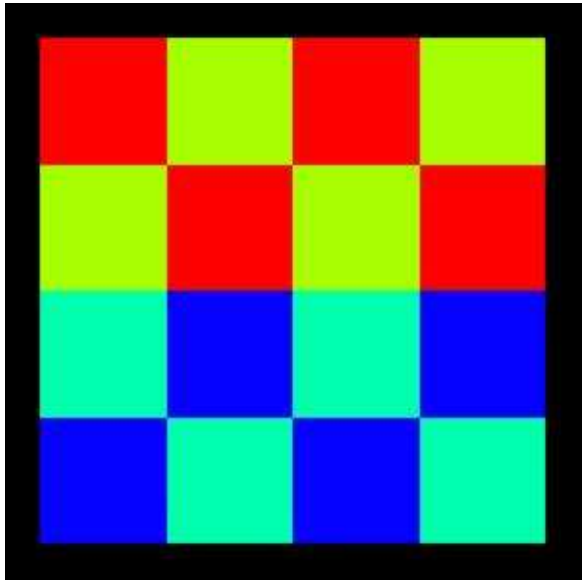
- 2つのファイルが生成される
  - color.log　　新旧メッシュ番号の対応表
    行列関連情報
  - color.inp　　メッシュの色分けファイル(ParaView用)



入力: 0
(CM, 7 colors)



入力: 3
(5 colors)



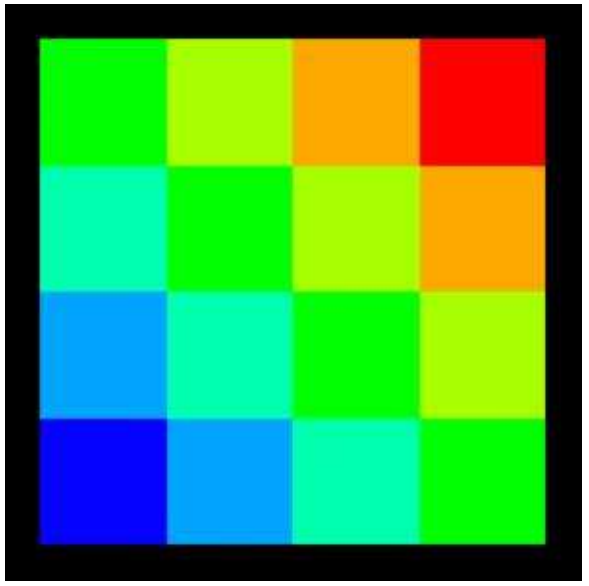
入力: 4
(4 colors)

入力=0：CM(7色)

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

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

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

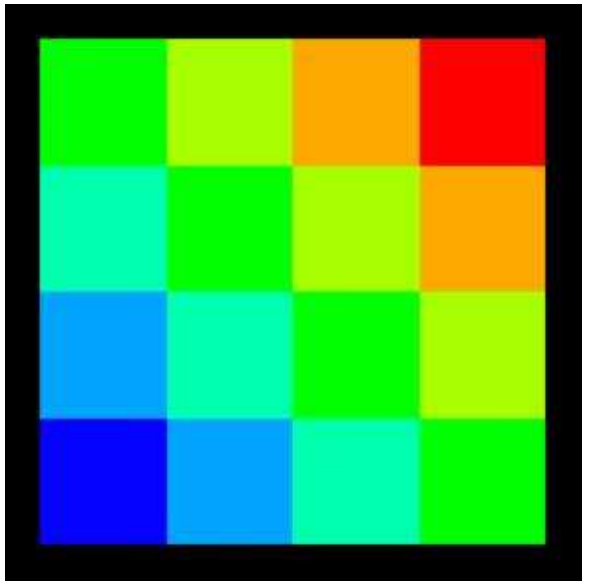


入力=0：CM(7色)

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

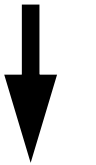
10	13	15	16		
6	9	12	14		
3	5	8	11		
1	2	4	7		

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



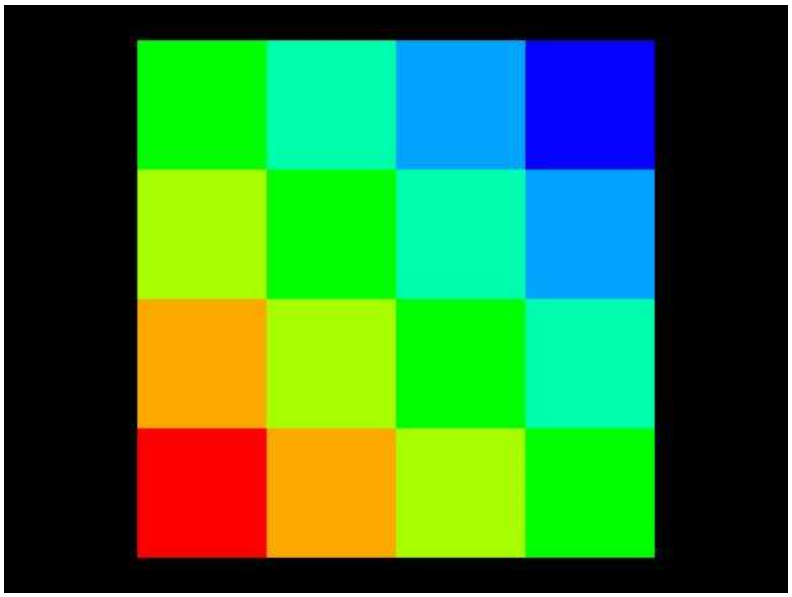
入力=-1：RCM(7色)

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



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

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

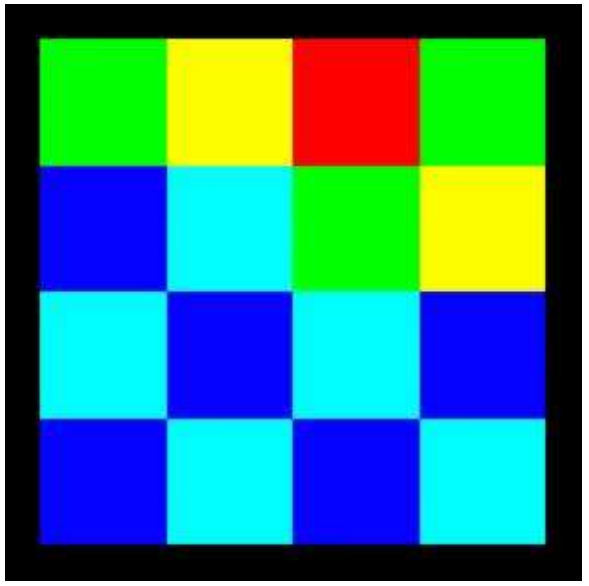


入力=3：実際は5色（マルチカラー）

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

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

```
#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
```



入力=3：実際は5色（マルチカラー）

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

5					
	3		4		
1		2			

```
#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
```

16/3=5

「5」個ずつ独立な要素を元の番号順に選択

入力=3：実際は5色（multicolor）

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

5	10				
8	3	9	4		
1	6	2	7		

```
#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
```

16/3=5

「5」個ずつ独立な要素を元の番号順に選択

入力=3：実際は5色（マルチカラー）

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

12			13		
5	10	11			
8	3	9	4		
1	6	2	7		

```
#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
```

16/3=5

独立な要素が無くなったら次の色へ

入力=3：実際は5色（マルチカラー）

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

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

```
#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
```

16/3=5

独立な要素が無くなったら次の色へ

入力=3：実際は5色（マルチカラー）

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

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

```
#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
```

16/3=5

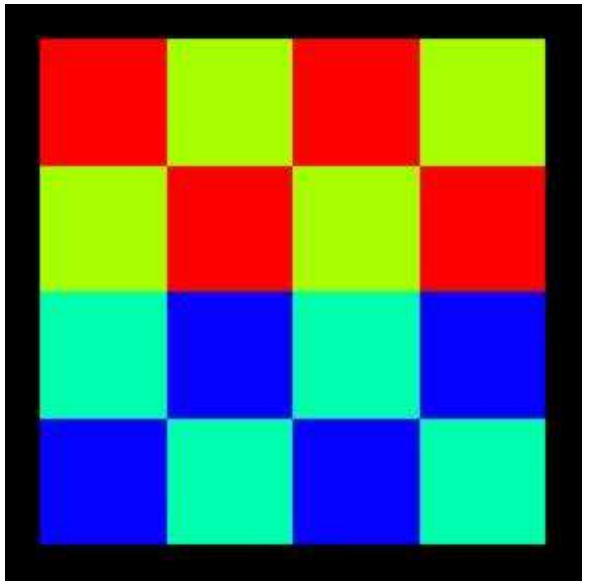
最終的に5色必要であった

入力=4：4色（マルチカラー）

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

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

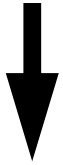
```
#new     1   #old     1   color     1
#new     2   #old     3   color     1
#new     3   #old     6   color     1
#new     4   #old     8   color     1
#new     5   #old     2   color     2
#new     6   #old     4   color     2
#new     7   #old     5   color     2
#new     8   #old     7   color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



入力=3：実際は5色（マルチカラー）

color.log：行列関連情報出力

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



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

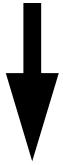
```
### INITIAL connectivity
I=    1      INL(i)=    0      INU(i)=    2
  IAL:
  IAU:    2    5
I=    2      INL(i)=    1      INU(i)=    2
  IAL:    1
  IAU:    3    6
I=    3      INL(i)=    1      INU(i)=    2
  IAL:    2
  IAU:    4    7
I=    4      INL(i)=    1      INU(i)=    1
  IAL:    3
  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:


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
```

入力=3：実際は5色（マルチカラー）

color.log：行列関連情報出力

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



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

```
### 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:
```

「L2-color」のソースファイル

```
$ cd <$P-L2>/coloring/src
$ ls
```

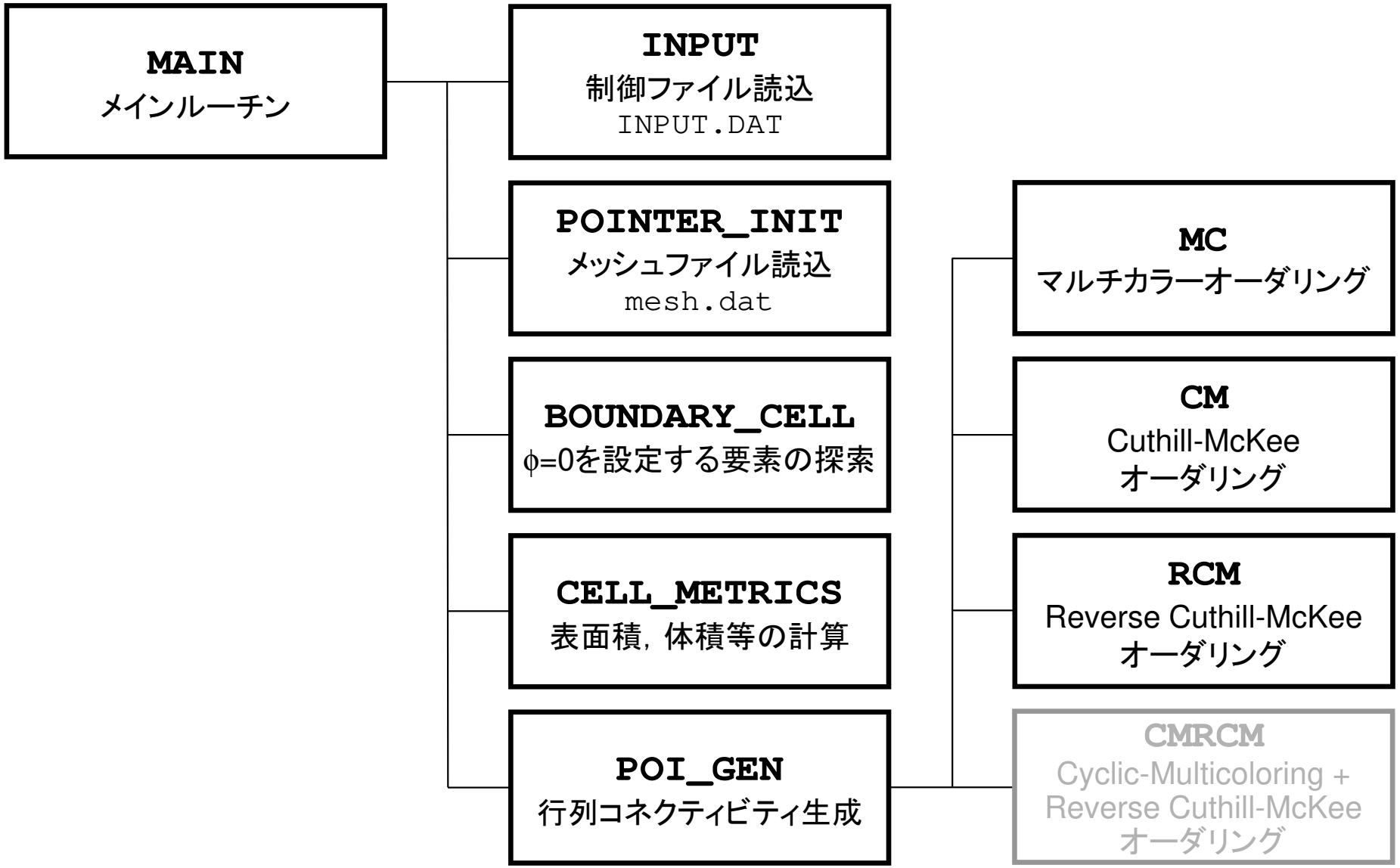
13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

この2次元形状を色づけする。

Main Program

```
#include <stdio.h> ...

int
main(int argc, char **argv)
{
        FILE *fp21;
        int i, ic, k;

        if(POINTER_INIT()) goto error;
        if((fp21 = fopen("color.log", "w")) == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
        if(POI_GEN(fp21)) goto error;
        if(OUTUCD()) goto error;

        fprintf(fp21, "¥n¥nCOLOR number%8d¥n¥n", NCOLORtot);
        for(ic=1; ic<=NCOLORtot; ic++) {
                for(i=COLORindex[ic-1]+1; i<=COLORindex[ic]; i++) {
                        fprintf(fp21, "  #new%8d   #old%8d  color%8d¥n", i, NEWtoOLD[i-1]+1, ic);
                }
        }

        (...)

        fclose(fp21);
        return 0;

error:
        return -1;
}
```

Structure of L2-color



Main Program

```
#include <stdio.h> ...

int
main(int argc, char **argv)
{
        FILE *fp21;
        int i, ic, k;

        if(POINTER_INIT()) goto error;
        if((fp21 = fopen("color.log", "w")) == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
        if(POI_GEN(fp21)) goto error;
        if(OUTUCD()) goto error;

        fprintf(fp21, "¥n¥nCOLOR number%8d¥n¥n", NCOLORtot);
        for(ic=1; ic<=NCOLORtot; ic++) {
                for(i=COLORindex[ic-1]+1; i<=COLORindex[ic]; i++) {
                        fprintf(fp21, "  #new%8d   #old%8d  color%8d¥n", i, NEWtoOLD[i-1]+1, ic);
                }
        }

        (...)

        fclose(fp21);
        return 0;

error:
        return -1;
}
```

struct.h

```
#ifndef __H_STRUCT
#define __H_STRUCT

#include <omp.h>

int ICELTOT, ICELTOTp, N;
int NX, NY, NZ, NXP1, NYP1, NZP1, IBNODTOT;
int NXc, NYc, NZc;

double DX, DY, DZ, XAREA, YAREA, ZAREA;
double RDX, RDY, RDZ, RDX2, RDY2, RDZ2, R2DX, R2DY, R2DZ;
double *VOLCEL, *VOLNOD, *RVC, *RVN;

int **XYZ, **NEIBcell;

int ZmaxCELtot;
int *BC_INDEX, *BC_NOD;
int *ZmaxCEL;

int **IWKX;
double **FCV;

int my_rank, PETOT, PEsmpTOT;

#endif /* __H_STRUCT */
```

ICELTOT
要素数（NX×NY×NZ）

N
節点数

NX,NY,NZ
x,y,z方向要素数

NXP1,NYP1,NZP1
x,y,z方向節点数

IBNODTOT
NXP1×NYP1

XYZ[ICELTOT][3]
要素座標

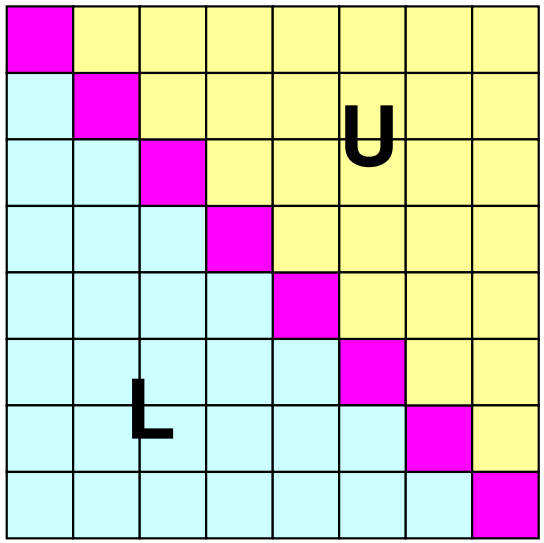
NEIBcell[ICELTOT][6]
隣接要素

pcg.h

```
#ifndef __H_PCG
#define __H_PCG
        static int N2 = 256;
        int NUmax, NLmax, NCOLORtot, NCOLORk, NU,
NL;
        int METHOD, ORDER_METHOD;
        double EPSICCG;

        double *D, *PHI, *BFORCE;
        double *AL, *AU;

        int *INL, *INU, *COLORindex;
        int *indexL, *indexU;
        int *OLDtoNEW, *NEWtoOLD;
        int **IAL, **IAU;
        int *itemL, *itemU;
        int NPL, NPU;
#endif /* __H_PCG */
```

扱う行列：疎行列
（自分の周辺のみと接続）
⇒ 非ゼロ成分のみを記憶する

上下三角成分を別々に記憶



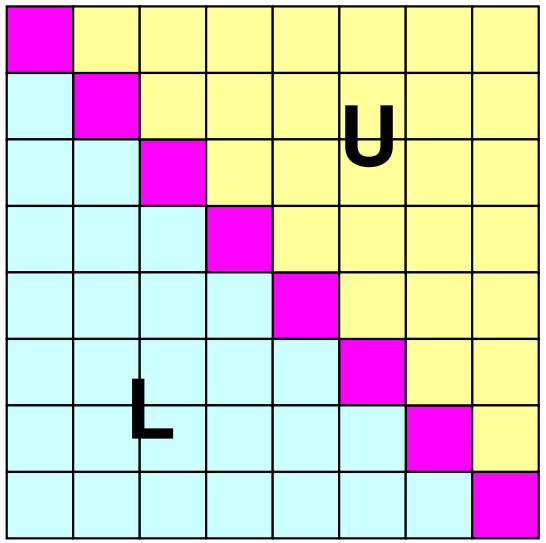
```
#ifndef __H_PCG
#define __H_PCG
        static int N2 = 256;
        int NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL;
        int METHOD, ORDER_METHOD;
        double EPSICCG;

        double *D, *PHI, *BFORCE;
        double *AL, *AU;

        int *INL, *INU, *COLORindex;
        int *indexL, *indexU;
        int *OLDtoNEW, *NEWtoOLD;
        int **IAL, **IAU;
        int *itemL, *itemU;
        int NPL, NPU;
#endif /* __H_PCG */
```

pcg.h

補助配列

下三角成分(列番号):
非対角成分で自分より要素番号が小さい。
IAL[i][icou] < i

上三角成分(列番号):
非対角成分で自分より要素番号が大きい。
IAU[i[icou] > i



INL{ICELTOT}	非零下三角成分の数
IAL[ICELTOT][NL]	非零下三角成分（列番号）
INU[ICELTOT]	非零上三角成分の数
IAU[ICELTOT][NU]	非零上三角成分（列番号）
NU,NL	非零上下三角成分の最大数（ここでは6）
indexL[ICELTOT+1]	各行の非零下三角成分数（CRS）
indexU[ICELTOT+1]	各行の非零上三角成分数（CRS）
NPL, NPU	非零上下三角成分数の合計（CRS）
itemL[NPL],itemU[NPU]	比零上下三角成分（列番号）（CRS）

```
#ifndef __H_PCG
#define __H_PCG
        static int N2 = 256;
        int NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL;
        int METHOD, ORDER_METHOD;
        double EPSICCG;

        double *D, *PHI, *BFORCE;
        double *AL, *AU;

        int *INL, *INU, *COLORindex;
        int *indexL, *indexU;
        int *OLDtoNEW, *NEWtoOLD;
        int **IAL, **IAU;
        int *itemL, *itemU;
        int NPL, NPU;
#endif /* __H_PCG */
```

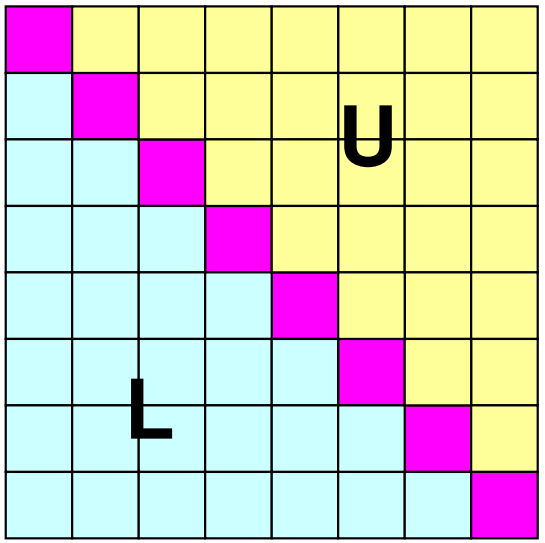
pcg.h

補助配列

下三角成分(列番号):
IAL[i][icou] < i
その個数が INL[i]

上三角成分(列番号):
IAU[i][icou] > i
その個数がINU[i]

INL{ICELTOT}	非零下三角成分の数
IAL[ICELTOT][NL]	非零下三角成分（列番号）
INU[ICELTOT]	非零上三角成分の数
IAU[ICELTOT][NU]	非零上三角成分（列番号）
NU,NL	非零上下三角成分の最大数（ここでは6）
indexL[ICELTOT+1]	各行の非零下三角成分数（CRS）
indexU[ICELTOT+1]	各行の非零上三角成分数（CRS）
NPL, NPU	非零上下三角成分数の合計（CRS）
itemL[NPL],itemU[NPU]	比零上下三角成分（列番号）（CRS）



入力=3：実際は5色（multicolor）

color.logに行列関連情報出力

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

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

```
### 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:


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
```

入力=3：実際は5色（multicolor）

color.logに行列関連情報出力

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

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

```
### 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
```

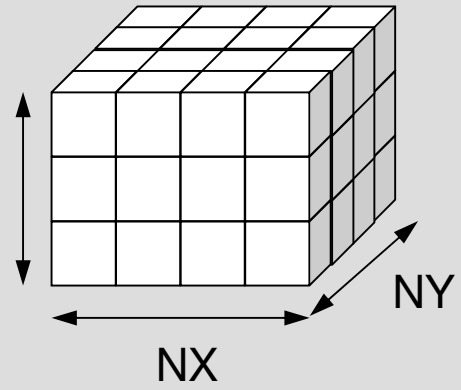
```
I=   14     INL(i)=  3     INU(i)=  0
  IAL:    4   11   13
  IAU:
I=   15     INL(i)=  2     INU(i)=  1
  IAL:   10   12
  IAU:   16
I=   16     INL(i)=  3     INU(i)=  0
  IAL:   11   15   13
  IAU:
```

プログラムの構成

```
#include <stdio.h> ...

int
main(int argc, char **argv)
{
        FILE *fp21;
        int i, ic, k;

        if(POINTER_INIT()) goto error;
        if((fp21 = fopen("color.log", "w")) == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
        if(POI_GEN(fp21)) goto error;
        if(OUTUCD()) goto error;

        fprintf(fp21, "¥n¥nCOLOR number%8d¥n¥n", NCOLORtot);
        for(ic=1; ic<=NCOLORtot; ic++) {
                for(i=COLORindex[ic-1]+1; i<=COLORindex[ic]; i++) {
                        fprintf(fp21, "  #new%8d   #old%8d  color%8d¥n", i, NEWtoOLD[i-1]+1, ic);
                }
        }

        (...)

        fclose(fp21);
        return 0;

error:
        return -1;
}
```

pointer_init (1/3)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "struct_ext.h"
#include "pcg_ext.h"
#include "pointer_init.h"
#include "allocate.h"

extern int
POINTER_INIT(void)
{

        int icel, ipe, i, j, k;

 fprintf(stderr, "input NX NY NZ=>¥n");
 fscanf(stdin, "%d%d%d", &NX, &NY, &NZ);
 fprintf(stderr, "NX=%d NY=%d NZ=%d¥n", NX, NY, NZ);

/*
 * INIT.
 */

  ICELTOT = NX * NY * NZ;

  NXP1 = NX + 1;
  NYP1 = NY + 1;
  NZP1 = NZ + 1;


  NEIBcell =
    (int **)allocate_matrix(sizeof(int),ICELTOT,6);

  XYZ      =
    (int **)allocate_matrix(sizeof(int),ICELTOT,3);
```



NX,NY,NZ
x,y,z方向要素数

NXP1,NYP1,NZP1
x,y,z方向節点数

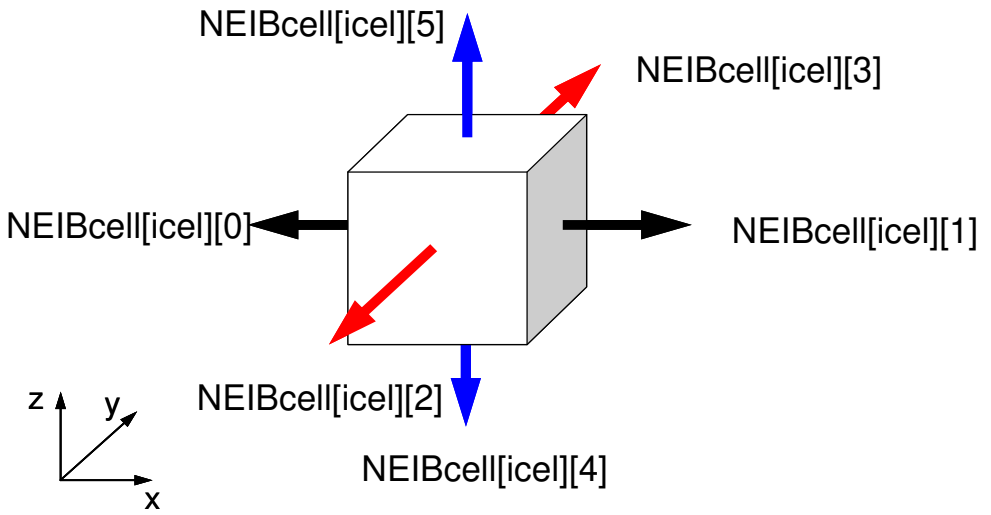
ICELTOT
要素数（NX×NY×NZ）

XYZ[ICELTOT][3]
要素座標

NEIBcell[ICELTOT][6]
隣接要素

pointer_init (2/3)

```
for(k=0; k<NZ; k++) {
    for(j=0; j<NY; j++) {
        for(i=0; i<NX; i++) {
            icel = k * NX * NY + j * NX + i;
            NEIBcell[icel][0] = icel - 1       + 1;
            NEIBcell[icel][1] = icel + 1       + 1;
            NEIBcell[icel][2] = icel - NX      + 1;
            NEIBcell[icel][3] = icel + NX      + 1;
            NEIBcell[icel][4] = icel - NX * NY + 1;
            NEIBcell[icel][5] = icel + NX * NY + 1;
            if(i == 0)    NEIBcell[icel][0] = 0;
            if(i == NX-1) NEIBcell[icel][1] = 0;
            if(j == 0)    NEIBcell[icel][2] = 0;
            if(j == NY-1) NEIBcell[icel][3] = 0;
            if(k == 0)    NEIBcell[icel][4] = 0;
            if(k == NZ-1) NEIBcell[icel][5] = 0;

            XYZ[icel][0] = i + 1;
            XYZ[icel][1] = j + 1;
            XYZ[icel][2] = k + 1;
        }
    }
}
```



i= XYZ[icel][0]
j= XYZ[icel][1], k= XYZ[icel][2]
icel= k*NX*NY + j*NX + i

“icel” starts at 0

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

“NEIBcell” starts at 1

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

NEIBcell[icel][0]= icel – 1 + 1
NEIBcell[icel][1]= icel + 1 + 1
NEIBcell[icel][2]= icel – NX + 1
NEIBcell[icel][3]= icel + NX + 1
NEIBcell[icel][4]= icel – NX*NY + 1
NEIBcell[icel][5]= icel + NX*NY + 1

pointer_init (3/3)

```
        if(DX <= 0.0) {
                DX = 1.0 / (double)NX;
                DY = 1.0 / (double)NY;
                DZ = 1.0 / (double)NZ;
        }

        NXP1 = NX + 1;
        NYP1 = NY + 1;
        NZP1 = NZ + 1;

        IBNODTOT = NXP1 * NYP1;
        N        = NXP1 * NYP1 * NZP1;

        return 0;
}
```

NXP1, NYP1, NZP1などはUCDファイル出力に必要

Main Program

```
#include <stdio.h> ...

int
main(int argc, char **argv)
{
        FILE *fp21;
        int i, ic, k;

        if(POINTER_INIT()) goto error;
        if((fp21 = fopen("color.log", "w")) == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
        if(POI_GEN(fp21)) goto error;
        if(OUTUCD()) goto error;

        fprintf(fp21, "¥n¥nCOLOR number%8d¥n¥n", NCOLORtot);
        for(ic=1; ic<=NCOLORtot; ic++) {
                for(i=COLORindex[ic-1]+1; i<=COLORindex[ic]; i++) {
                        fprintf(fp21, "  #new%8d   #old%8d  color%8d¥n", i, NEWtoOLD[i-1]+1, ic);
                }
        }

        (...)

        fclose(fp21);
        return 0;

error:
        return -1;
}
```

poi_gen (1/4)

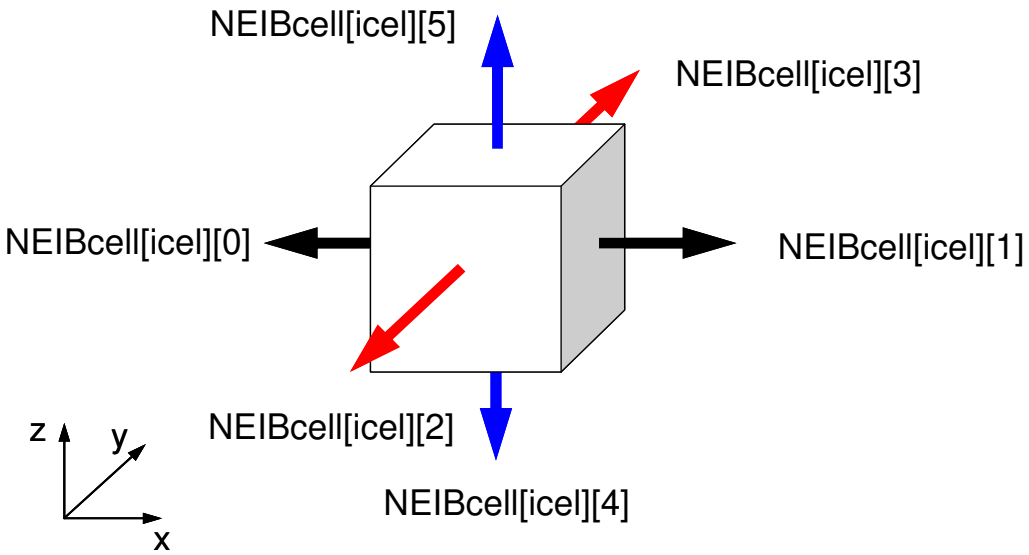
```
#include "allocate.h"
extern int
POI_GEN(void)
{
  int nn;
  int ic0, icN1, icN2, icN3, icN4, icN5, icN6;
  int i, j, k, ib, ic, ip, icel, icou, icol, icouG;
  int ii, jj, kk, nn1, num, nr, j0, j1;
  double coef, VOL0, S1t, E1t;
  int isL, ieL, isU, ieU;
  NL=6; NU= 6;

  IAL = (int **)allocate_matrix(sizeof(int),ICELTOT,NL);
  IAU = (int **)allocate_matrix(sizeof(int),ICELTOT,NU);
  INL = (int *)allocate_vector(sizeof(int),ICELTOT);
  INU = (int *)allocate_vector(sizeof(int),ICELTOT);

  for (i = 0; i <ICELTOT ; i++) {
       BFORCE[i]=0.0;
       D[i]  =0.0; PHI[i]=0.0;
       INL[i] = 0; INU[i] = 0;
       for(j=0;j<6;j++){
           IAL[i][j]=0; IAU[i][j]=0;
       }
  }
  for (i = 0; i <=ICELTOT ; i++) {
       indexL[i] = 0; indexU[i] = 0;
  }
```

allocate.c

```
/*****************************
  allocate matrix
*****************************/
void** allocate_matrix(int size,int m,int n)
{
  void **aa;
  int i;
  if ( ( aa=(void ** )malloc( m * sizeof(void*) ) ) == NULL ) {
     fprintf(stdout,"Error:Memory does not enough! aa in matrix ¥n");
     exit(1);
  }
  if ( ( aa[0]=(void * )malloc( m * n * size ) ) == NULL ) {
     fprintf(stdout,"Error:Memory does not enough! in matrix ¥n");
     exit(1);
  }
        for(i=1;i<m;i++) aa[i]=(char*)aa[i-1]+size*n;
  return aa;
}
```

```
for(icel=0; icel<ICELTOT; icel++) {

     icN1 = NEIBcell[icel][0];
     icN2 = NEIBcell[icel][1];
     icN3 = NEIBcell[icel][2];
     icN4 = NEIBcell[icel][3];
     icN5 = NEIBcell[icel][4];
     icN6 = NEIBcell[icel][5];

     if(icN5 != 0) {
        icou = INL[icel] + 1;
        IAL[icel][icou-1] = icN5;
        INL[icel]         = icou;
      }

      if(icN3 != 0) {
        icou = INL[icel] + 1;
        IAL[icel][icou-1] = icN3;
        INL[icel]         = icou;
      }

      if(icN1 != 0) {
        icou = INL[icel] + 1;
        IAL[icel][icou-1] = icN3;
        INL[icel]         = icou;
      }
```

poi_gen (2/4)



下三角成分

NEIBcell[icel][4]= icel – NX*NY + 1
NEIBcell[icel][2]= icel – NX + 1
NEIBcell[icel][0]= icel – 1 + 1

“icel” starts at 0

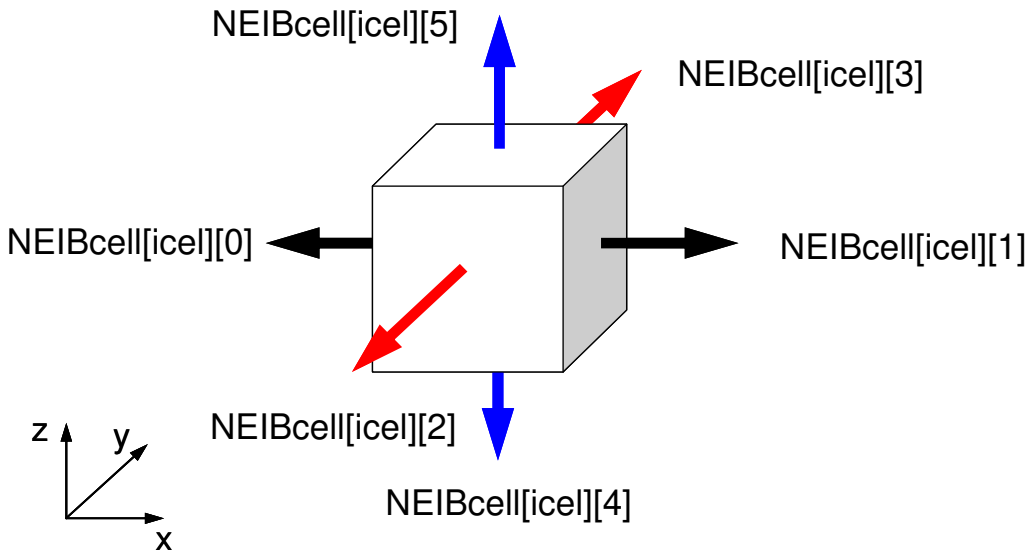
12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

“IAL” starts at 1

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

poi_gen (3/4)

```
 for(icel=0; icel<ICELTOT; icel++) {

      icN1 = NEIBcell[icel][0];
      icN2 = NEIBcell[icel][1];
      icN3 = NEIBcell[icel][2];
      icN4 = NEIBcell[icel][3];
      icN5 = NEIBcell[icel][4];
      icN6 = NEIBcell[icel][5];
....
      if(icN2 != 0) {
        icou = INU[icel] + 1;
        IAU[icel][icou-1] = icN2;
        INU[icel]         = icou;
      }

      if(icN4 != 0) {
        icou = INU[icel] + 1;
        IAU[icel][icou-1] = icN4;
        INU[icel]         = icou;
      }

      if(icN6 != 0) {
        icou = INU[icel] + 1;
        IAU[icel][icou-1] = icN6;
        INU[icel]         = icou;
      }
}
```



上三角成分

NEIBcell[icel][1]= icel + 1 + 1
NEIBcell[icel][3]= icel + NX + 1
NEIBcell[icel][5]= icel + NX*NY + 1

“icel” starts at 0

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

“IAU” starts at 1

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

poi_gen (4/4)

初期色数を読み込む

```
N111:
        fprintf(stderr, "¥n¥nYou have%8d elements¥n", ICELTOT);
        fprintf(stderr, "How many colors do you need ?¥n");
        fprintf(stderr, "  #COLOR must be more than 2 and¥n");
        fprintf(stderr, "  #COLOR must not be more than%8d¥n", ICELTOT);
        fprintf(stderr, "  if #COLOR= 0 then CM  ordering¥n");
        fprintf(stderr, "  if #COLOR=-1 then RCM ordering¥n");
        fprintf(stderr, "  if #COLOR<-1 then CMRCM ordering¥n");
        fprintf(stderr, "=>¥n");
        fscanf(stdin, "%d", &NCOLORtot);
        if(NCOLORtot == 1 && NCOLORtot > ICELTOT) goto N111;

        OLDtoNEW = (int *)calloc(ICELTOT, sizeof(int));
        if(OLDtoNEW == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
        NEWtoOLD = (int *)calloc(ICELTOT, sizeof(int));
        if(NEWtoOLD == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
        COLORindex = (int *)calloc(ICELTOT+1, sizeof(int));
        if(COLORindex == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }

        if(NCOLORtot > 0) {
                MC(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot == 0) {
                CM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot ==-1) {
                RCM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot <-1) {
                CMRCM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        }

        fprintf(stderr, "¥n# TOTAL COLOR number%8d¥n", NCOLORtot);
        return 0;
}
```

poi_gen (4/4)

配列宣言を実施する。

```
N111:
        fprintf(stderr, "¥n¥nYou have%8d elements¥n", ICELTOT);
        fprintf(stderr, "How many colors do you need ?¥n");
        fprintf(stderr, "  #COLOR must be more than 2 and¥n");
        fprintf(stderr, "  #COLOR must not be more than%8d¥n", ICELTOT);
        fprintf(stderr, "  if #COLOR= 0 then CM  ordering¥n");
        fprintf(stderr, "  if #COLOR=-1 then RCM ordering¥n");
        fprintf(stderr, "  if #COLOR<-1 then CMRCM ordering¥n");
        fprintf(stderr, "=>¥n");
        fscanf(stdin, "%d", &NCOLORtot);
        if(NCOLORtot == 1 && NCOLORtot > ICELTOT) goto N111;

        OLDtoNEW = (int *)calloc(ICELTOT, sizeof(int));
        if(OLDtoNEW == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
        NEWtoOLD = (int *)calloc(ICELTOT, sizeof(int));
        if(NEWtoOLD == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
        COLORindex = (int *)calloc(ICELTOT+1, sizeof(int));
        if(COLORindex == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }

        if(NCOLORtot > 0) {
                MC(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot == 0) {
                CM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot ==-1) {
                RCM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot <-1) {
                CMRCM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        }

        fprintf(stderr, "¥n# TOTAL COLOR number%8d¥n", NCOLORtot);
        return 0;
}
```

poi_gen (4/4)

```
        if(NCOLORtot > 0) {
                MC(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot == 0) {
                CM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot ==-1) {
                RCM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot <-1) {
                CMRCM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        }

        fprintf(stderr, "¥n# TOTAL COLOR number%8d¥n", NCOLORtot);
        return 0;
```

INL, INU, IAL, IAU	再番号付け後の情報が入る
OLDtoNEW, NEWtoOLD	新旧要素番号対象表
NCOLORtot	色数（入力値と同じかそれより大きくなる）
COLORindex[NCOLORtot+1]	**COLORindex[ic]から COLORindex[ic+1]-1までの要素（再番号付後）が「ic+1」番目の色に所属する** 同じ色の要素は互いに独立：並列計算可能 本プログラム内では色番号は「1」から開始

COLORindex[NCOLORtot+1] **COLORindex[ic]から COLORindex[ic+1]-1までの要素(再番号付後)が「ic+1」番目の色に所属する**

同じ色の要素は互いに独立:並列計算可能

本プログラム内では色番号は「1」から開始

```
for(ic=1; ic<=NCOLORtot; ic++) {
    for(i=COLORindex[ic-1]+1; i<=COLORindex[ic]; i++) {
        fprintf(fp21, "#new%8d #old%8d  color%8d¥n",
                      i, NEWtoOLD[i-1]+1, ic);
    }
}
```

COLORindex

```
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
```

修正されたMC法

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）。同じ色：連続した「新」要素番号

mc (1/8)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "mc.h"

int
MC(int N, int NL, int NU, int *INL, int **IAL, int *INU, int **IAU,
        int *NCOLORtot, int *COLORindex, int *NEWtoOLD, int *OLDtoNEW)
{
        int NCOLORk;
        int *IW, *INLw, *INUw;
        int **IALw, **IAUw;
        int INmin, NODmin, ITEMcou;
        int i, j, k, icon, icou, icouK, icoug, icol, ic, ik, jL, jU;
```

mc (2/8)

```
IW = (int *)calloc(N, sizeof(int));
if(IW == NULL) {
        fprintf(stderr, "Error: %s¥n",
        strerror(errno));
              return -1;
}

NCOLORk = *NCOLORtot;

for(i=0; i<N; i++) {
    NEWtoOLD[i] = i;
    OLDtoNEW[i] = i;
}

INmin = N;
NODmin = -1;

for(i=0; i<N; i++) {
    icon = INL[i] + INU[i];
    if (icon < INmin) {
        INmin = icon;
        NODmin = i;
     }
}

OLDtoNEW[NODmin] = 1;
NEWtoOLD[0     ] = NODmin+1;

memset(IW, 0, sizeof(int)*N);
IW[NODmin] = 1;

ITEMcou = N / NCOLORk;
```

作業配列「IW」に「0」を入れる
IWには各要素の色番号が入る

接続要素数(次数)が最小の要素を探索
NODmin

mc (2/8)

```
IW = (int *)calloc(N, sizeof(int));
if(IW == NULL) {
         fprintf(stderr, "Error: %s¥n",
         strerror(errno));
                return -1;
}

NCOLORk = *NCOLORtot;

for(i=0; i<N; i++) {
    NEWtoOLD[i] = i;
    OLDtoNEW[i] = i;
}

INmin = N;
NODmin = -1;

for(i=0; i<N; i++) {
    icon = INL[i] + INU[i];
    if (icon < INmin) {
        INmin = icon;
        NODmin = i;
     }
}

OLDtoNEW[NODmin] = 1;
NEWtoOLD[0     ] = NODmin+1;

memset(IW, 0, sizeof(int)*N);
IW[NODmin] = 1;

ITEMcou = N / NCOLORk;
```

New mesh ID of **NODmin** is set to 0
Color ID of **NODmin** is set to 1

OLDtoNEW[NODmin]= 1
NEWtoOLD[0]= NODmin+1

IW[NODmin]=1: Color ID

mc (2/8)

```
IW = (int *)calloc(N, sizeof(int));
if(IW == NULL) {
         fprintf(stderr, "Error: %s¥n",
         strerror(errno));
                return -1;
}

NCOLORk = *NCOLORtot;

for(i=0; i<N; i++) {
    NEWtoOLD[i] = i;
    OLDtoNEW[i] = i;
}

INmin = N;
NODmin = -1;

for(i=0; i<N; i++) {
    icon = INL[i] + INU[i];
    if (icon < INmin) {
        INmin = icon;
        NODmin = i;
     }
}

OLDtoNEW[NODmin] = 1;
NEWtoOLD[0     ] = NODmin+1;

memset(IW, 0, sizeof(int)*N);
IW[NODmin] = 1;

ITEMcou = N / NCOLORk;
```

ITEMcou= N/NCOLORk:
各色に含まれる要素数の目安

INPUT=3: MC, 5-Colors

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

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

```
#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
```

16/3=5 = **ITEMcou**

最終的に5色必要であった

切り上げ・切り下げで変わる

mc (3/8)

カウンタ初期化

icou　: 全体のカウンタ
icouK : 色内のカウンタ

```
icou  = 1;
icouK = 1;
for(icol=0; icol<N; icol++) {
    NCOLORk = icol + 1;
    for(i=0; i<N; i++) {
        if(IW[i] <= 0) IW[i] = 0;}
           for(i=0; i<N; i++) {
/* already COLORED*/
               if(IW[i] == icol+1) {
                  for(k=0; k<INL[i]; k++) {
                      ik = IAL[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
                  for(k=0; k<INU[i]; k++) {
                      ik = IAU[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
               }

/* not COLORED */
               if(IW[i] == 0) {
                  icou++; icouK++;
                  IW[i] = icol + 1;
                  for(k=0; k<INL[i]; k++) {
                      ik = IAL[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
                  for(k=0; k<INU[i]; k++) {
                      ik = IAU[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
               }
               if(icou == N) goto N200;
               if(icouK == ITEMcou) goto N100;
    }
N100:
           icouK = 0;
}
N200:
```

mc (3/8)

Loop on Colors

```
icou  = 1;
icouK = 1;
for(icol=0; icol<N; icol++) {
    NCOLORk = icol + 1;
    for(i=0; i<N; i++) {
        if(IW[i] <= 0) IW[i] = 0;}
           for(i=0; i<N; i++) {
/* already COLORED*/
               if(IW[i] == icol+1) {
                  for(k=0; k<INL[i]; k++) {
                      ik = IAL[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
                  for(k=0; k<INU[i]; k++) {
                      ik = IAU[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
               }

/* not COLORED */
               if(IW[i] == 0) {
                  icou++; icouK++;
                  IW[i] = icol + 1;
                  for(k=0; k<INL[i]; k++) {
                      ik = IAL[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
                  for(k=0; k<INU[i]; k++) {
                      ik = IAU[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
               }
               if(icou == N) goto N200;
               if(icouK == ITEMcou) goto N100;
           }
N100:
           icouK = 0;
}
N200:
```

mc (3/8)

```
icou  = 1;
icouK = 1;
for(icol=0; icol<N; icol++) {
    NCOLORk = icol + 1;
    for(i=0; i<N; i++) {
        if(IW[i] <= 0) IW[i] = 0;}
           for(i=0; i<N; i++) {
/* already COLORED*/
               if(IW[i] == icol+1) {
                  for(k=0; k<INL[i]; k++) {
                      ik = IAL[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
                  for(k=0; k<INU[i]; k++) {
                      ik = IAU[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
               }

/* not COLORED */
               if(IW[i] == 0) {
                  icou++; icouK++;
                  IW[i] = icol + 1;
                  for(k=0; k<INL[i]; k++) {
                      ik = IAL[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
                  for(k=0; k<INU[i]; k++) {
                      ik = IAU[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
               }
               if(icou == N) goto N200;
               if(icouK == ITEMcou) goto N100;
           }
N100:
           icouK = 0;
}
N200:
```

現在の色数を「NCOLORk」とする。

色づけされていない要素の「IW」の値を0とする。

mc (3/8)

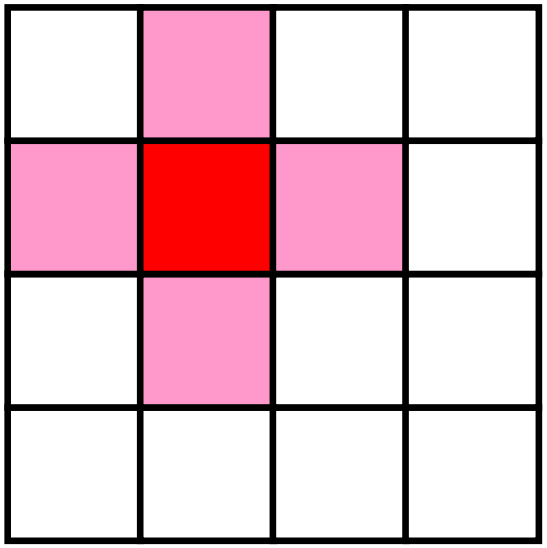
```
icou  = 1;
icouK = 1;
for(icol=0; icol<N; icol++) {
    NCOLORk = icol + 1;
    for(i=0; i<N; i++) {
        if(IW[i] <= 0) IW[i] = 0;}
           for(i=0; i<N; i++) {
/* already COLORED*/
               if(IW[i] == icol+1) {
                  for(k=0; k<INL[i]; k++) {
                      ik = IAL[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
                  for(k=0; k<INU[i]; k++) {
                      ik = IAU[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
               }

/* not COLORED */
               if(IW[i] == 0) {
                  icou++; icouK++;
                  IW[i] = icol + 1;
                  for(k=0; k<INL[i]; k++) {
                      ik = IAL[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
                  for(k=0; k<INU[i]; k++) {
                      ik = IAU[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
               }
               if(icou == N) goto N200;
               if(icouK == ITEMcou) goto N100;
           }
N100:
           icouK = 0;
}
N200:
```

Loop on Colors

全要素に関するループ。

すでに現在の色に色づけされている場合は，隣接する要素のIWの値を「-1」とする（実際にこの部分を通る可能性は最初の一回のみであるが，念のため）。

すでに「現在の色」に色づけされている要素の隣接要素は「現在の色」に入る可能性が無いため除外。



mc (3/8)

```
icou  = 1;
icouK = 1;
for(icol=0; icol<N; icol++) {
    NCOLORk = icol + 1;
    for(i=0; i<N; i++) {
        if(IW[i] <= 0) IW[i] = 0;}
           for(i=0; i<N; i++) {
/* already COLORED*/
               if(IW[i] == icol+1) {
                  for(k=0; k<INL[i]; k++) {
                      ik = IAL[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
                  for(k=0; k<INU[i]; k++) {
                      ik = IAU[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
               }

/* not COLORED */
               if(IW[i] == 0) {
                  icou++; icouK++;
                  IW[i] = icol + 1;
                  for(k=0; k<INL[i]; k++) {
                      ik = IAL[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
                  for(k=0; k<INU[i]; k++) {
                      ik = IAU[i][k];
                      if(IW[ik-1] <= 0) IW[ik-1] = -1;
                  }
               }
               if(icou == N) goto N200;
               if(icouK == ITEMcou) goto N100;
           }
N100:
           icouK = 0;
}
N200:
```

if **IW[i]=0:**

- **icou =icou+1**
- **icouK=icouK+1**
- **IW[i]=icol+1:** Color ID

- **IW[ik]=-1** where ***ik-th*** mesh is adjacent to ***i-th*** mesh

mc (3/8)

```
icou  = 1;
icouK = 1;
for(icol=0; icol<N; icol++) {
    NCOLORk = icol + 1;
    for(i=0; i<N; i++) {
        if(IW[i] <= 0) IW[i] = 0;}
        for(i=0; i<N; i++) {
/* already COLORED*/
            if(IW[i] == icol+1) {
               for(k=0; k<INL[i]; k++) {
                   ik = IAL[i][k];
                   if(IW[ik-1] <= 0) IW[ik-1] = -1;
               }
               for(k=0; k<INU[i]; k++) {
                   ik = IAU[i][k];
                   if(IW[ik-1] <= 0) IW[ik-1] = -1;
               }
            }

/* not COLORED */
            if(IW[i] == 0) {
               icou++; icouK++;
               IW[i] = icol + 1;
               for(k=0; k<INL[i]; k++) {
                   ik = IAL[i][k];
                   if(IW[ik-1] <= 0) IW[ik-1] = -1;
               }
               for(k=0; k<INU[i]; k++) {
                   ik = IAU[i][k];
                   if(IW[ik-1] <= 0) IW[ik-1] = -1;
               }
            }
            if(icou == N) goto N200;
            if(icouK == ITEMcou) goto N100;
        }
N100:
        icouK = 0;
}
N200:
```

icou : Global Counter
icouK: Intra-Color Counter

要素数のカウンタ(icou)が「N(ICELTOT)」を超えたらループを抜ける(全要素の色付け完了)。

色内のカウンタ(icouK)が「ITEMcou」を超えたら「icouK=0」として次の色へ移る。

「icouK < ITEMcou」でも「i」が「N」に到達したら(独立な要素がもうないので)次の色へ移る。

INPUT=3: MC, 5-Colors

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

5					
	3		4		
1		2			

```
#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
```

16/3=5

「5」個ずつ独立な要素を元の番号順に選択

INPUT=3: MC, 5-Colors

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

5	10				
8	3	9	4		
1	6	2	7		

```
#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
```

16/3=5

「5」個ずつ独立な要素を元の番号順に選択

INPUT=3: MC, 5-Colors

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

12			13		
5	10	11			
8	3	9	4		
1	6	2	7		

```
#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
```

16/3=5

独立な要素が無くなったら次の色へ

INPUT=3: MC, 5-Colors

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

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

```
#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
```

16/3=5

独立な要素が無くなったら次の色へ

mc (5/8)

```
/******************
 * Final Coloring *
 ******************/

*NCOLORtot = NCOLORk;
memset(COLORindex, 0, sizeof(int)*(N+1));

icoug = 0;
for(ic=1; ic<=(*NCOLORtot); ic++) {
    icou = 0;
    for(i=0; i<N; i++) {
        if(IW[i] == ic) {
           NEWtoOLD[icoug] = i+1;
           OLDtoNEW[i]     = icoug+1;
           icou++;
           icoug++;
         }
    }
    COLORindex[ic] = icou;
}

for(ic=1; ic<=(*NCOLORtot); ic++) {
    COLORindex[ic] += COLORindex[ic-1];
}
```

NCOLORtot= NCOLORk:
Final number of colors.

NCOLORtot g.e. (Initial number of colors provided by user)

色づけを終了した時点での色数「NCOLORk」を最終的な色数とする。
ユーザーが最初に設定した色数と同じかそれより多くなっている。

mc (5/8)

```
/*******************
 * Final Coloring *
 *******************/

*NCOLORtot = NCOLORk;
memset(COLORindex, 0, sizeof(int)*(N+1));

icoug = 0;
for(ic=1; ic<=(*NCOLORtot); ic++) {
    icou = 0;
    for(i=0; i<N; i++) {
        if(IW[i] == ic) {
           NEWtoOLD[icoug] = i+1;
           OLDtoNEW[i]     = icoug+1;
           icou++;
           icoug++;
         }
    }
    COLORindex[ic] = icou;
}

for(ic=1; ic<=(*NCOLORtot); ic++) {
    COLORindex[ic] += COLORindex[ic-1];
}
```

各要素の色に従って，色番号の少ない方から，要素の再番号付けを行なう。

OLDtoNEW[OldID]= NewID+1
NEWtoOLD[NewID]= OldID+1
starting@”1”

この時点では「COLORindex」には各色の要素数が入っている。

mc (6/8)

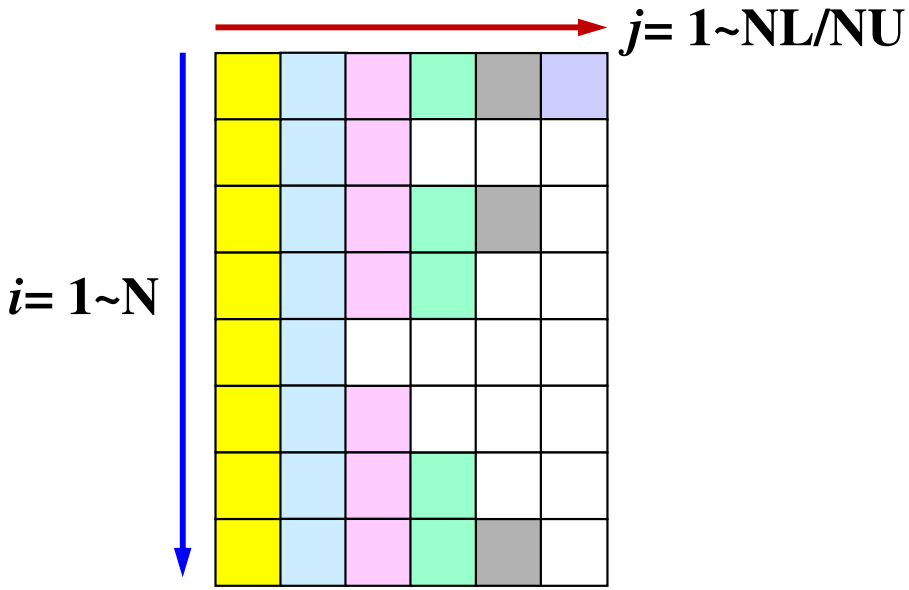
```
/******************
 * Final Coloring *
 ******************/

*NCOLORtot = NCOLORk;
memset(COLORindex, 0, sizeof(int)*(N+1));

icoug = 0;
for(ic=1; ic<=(*NCOLORtot); ic++) {
    icou = 0;
    for(i=0; i<N; i++) {
        if(IW[i] == ic) {
           NEWtoOLD[icoug] = i+1;
           OLDtoNEW[i]     = icoug+1;
           icou++;
           icoug++;
         }
    }
    COLORindex[ic] = icou;
}

for(ic=1; ic<=(*NCOLORtot); ic++) {
    COLORindex[ic] += COLORindex[ic-1];
}
```

COLODindex[ic]:

Now it is 1D index.

mc (6/8)

```
INLw = (int *)calloc(N, sizeof(int));
if(INLw == NULL) {
            fprintf(stderr, "Error: %s¥n", strerror(errno));
            return -1;
}...

for(j=0; j<NL; j++) {
    for(i=0; i<N; i++) {
        IW[i] = IAL[NEWtoOLD[i]-1][j];
    }
    for(i=0; i<N; i++) {
        IAL[i][j] = IW[i];
    }
}

for(j=0; j<NU; j++) {
    for(i=0; i<N; i++) {
        IW[i] = IAU[NEWtoOLD[i]-1][j];
    }
    for(i=0; i<N; i++) {
        IAU[i][j] = IW[i];
    }
}
```



ワーク配列の定義

- **INLw[N]**
- **INUw[N]**
- **IALw[N][NL]**
- **IAUw[N][NU]**

Lower/upper components (column ID) are reordered according to new numbering.

ID's of column ID are by old numbering.

上下三角成分を，新しい番号付けに従って入れ替える。上下三角成分そのものの番号はそのまま。

mc (7/8)

```
for(i=0; i<N; i++) {
    IW[i] = INL[NEWtoOLD[i]-1];
}
for(i=0; i<N; i++) {
    INLw[i] = IW[i];
}
for(i=0; i<N; i++) {
    IW[i] = INU[NEWtoOLD[i]-1];
}
for(i=0; i<N; i++) {
    INUw[i] = IW[i];
}

for(j=0; j<NL; j++) {
    for(i=0; i<N; i++) {
        if(IAL[i][j] == 0) {
            IALw[i][j] = 0;
        } else {
            IALw[i][j] = OLDtoNEW[IAL[i][j]-1];
        }
    }
}

for(j=0; j<NU; j++) {
    for(i=0; i<N; i++) {
        if(IAU[i][j] == 0) {
            IAUw[i][j] = 0;
        } else {
            IAUw[i][j] = OLDtoNEW[IAU[i][j]-1];
        }
    }
}
```

上下三角成分の数を，新しい番号付けに従って入れ替える。

INLを並び替えてINLw に格納する。
INUを並び替えてINUw に格納する。

mc (7/8)

```
for(i=0; i<N; i++) {
    IW[i] = INL[NEWtoOLD[i]-1];
}
for(i=0; i<N; i++) {
    INLw[i] = IW[i];
}
for(i=0; i<N; i++) {
    IW[i] = INU[NEWtoOLD[i]-1];
}
for(i=0; i<N; i++) {
    INUw[i] = IW[i];
}

for(j=0; j<NL; j++) {
    for(i=0; i<N; i++) {
        if(IAL[i][j] == 0) {
            IALw[i][j] = 0;
        } else {
            IALw[i][j] = OLDtoNEW[IAL[i][j]-1];
        }
    }
}

for(j=0; j<NU; j++) {
    for(i=0; i<N; i++) {
        if(IAU[i][j] == 0) {
            IAUw[i][j] = 0;
        } else {
            IAUw[i][j] = OLDtoNEW[IAU[i][j]-1];
        }
    }
}
```

上下三角成分を，新しい番号付けに従って新しい番号に付け替える。

IALw, IAUw に格納する

mc (8/8)

```
memset(INL, 0, sizeof(int)*N);
memset(INU, 0, sizeof(int)*N);
for(i=0; i<N; i++) {
    memset(IAL[i], 0, sizeof(int)*NL);}
for(i=0; i<N; i++) {
    memset(IAU[i], 0, sizeof(int)*NU);}

for(i=0; i<N; i++) {
    jL = 0;
    jU = 0;
    for(j=0; j<INLw[i]; j++) {
        if(IALw[i][j] > i+1) {
           IAU[i][jU] = IALw[i][j];
           jU++;
        } else {
           IAL[i][jL] = IALw[i][j];
           jL++;
        }
     }

     for(j=0; j<INUw[i]; j++) {
         if(IAUw[i][j] > i+1) {
            IAU[i][jU] = IAUw[i][j];
            jU++;
         } else {
            IAL[i][jL] = IAUw[i][j];
            jL++;
         }
      }
      INL[i] = jL;
      INU[i] = jU;
}
        free(IW);
        free(INLw);          free(INUw);
        free(IALw);          free(IAUw);

        return 0;
}
```

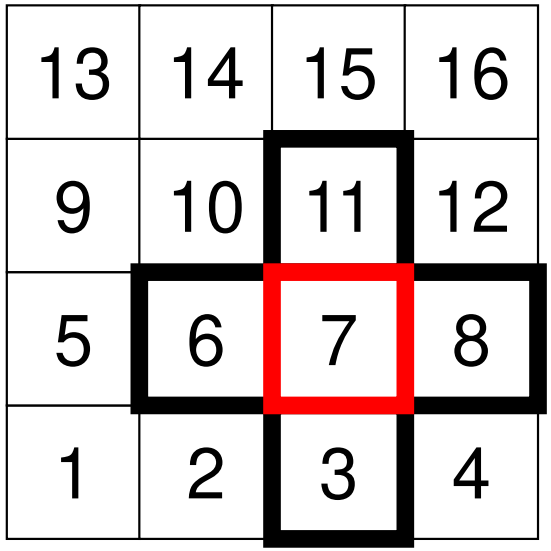
もともとの下三角成分にたいする処理。

```
for(i=0; i<N; i++) {
    jL = 0;
    jU = 0;
    for(j=0; j<INLw[i]; j++) {
        if(IALw[i][j] > i+1) {
           IAU[i][jU] =
IALw[i][j];
           jU++;
        } else {
           IAL[i][jL] =
IALw[i][j];
           jL++;
        }
     }
}
```

Because **`IALw[i][j]`** could be larger than **`i`** according to new numbering.

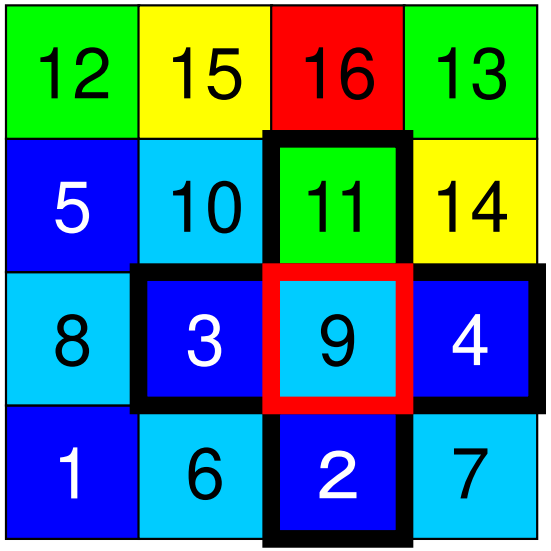
この操作が必要な理由

Original




```
INL[7-1]= 2
IAL[6][0]= 3,  IAL[6][1]= 6

INU[7-1]= 2
IAU[6][0]= 8,  IAU[6][1]=11
```

5 Color




```
INL[9-1]= 3
IAL[8][0]= 2,  IAL[8][1]= 3
IAL[8][2]= 4

INU[9-1]= 1
IAU[8][0]=11
```

再番号付けによって隣接要素との大小関係も変わってしまうため

mc (8/8)

```
memset(INL, 0, sizeof(int)*N);
memset(INU, 0, sizeof(int)*N);
for(i=0; i<N; i++) {
    memset(IAL[i], 0, sizeof(int)*NL);}
for(i=0; i<N; i++) {
    memset(IAU[i], 0, sizeof(int)*NU);}

for(i=0; i<N; i++) {
    jL = 0;
    jU = 0;
    for(j=0; j<INLw[i]; j++) {
        if(IALw[i][j] > i+1) {
           IAU[i][jU] = IALw[i][j];
           jU++;
        } else {
           IAL[i][jL] = IALw[i][j];
           jL++;
        }
     }

     for(j=0; j<INUw[i]; j++) {
         if(IAUw[i][j] > i+1) {
            IAU[i][jU] = IAUw[i][j];
            jU++;
         } else {
            IAL[i][jL] = IAUw[i][j];
            jL++;
         }
      }
      INL[i] = jL;
      INU[i] = jU;
}
        free(IW);
        free(INLw);          free(INUw);
        free(IALw);          free(IAUw);

        return 0;
}
```

もともとの上三角成分にたいする処理。

mc (8/8)

```
memset(INL, 0, sizeof(int)*N);
memset(INU, 0, sizeof(int)*N);
for(i=0; i<N; i++) {
    memset(IAL[i], 0, sizeof(int)*NL);}
for(i=0; i<N; i++) {
    memset(IAU[i], 0, sizeof(int)*NU);}

for(i=0; i<N; i++) {
    jL = 0;
    jU = 0;
    for(j=0; j<INLw[i]; j++) {
        if(IALw[i][j] > i+1) {
           IAU[i][jU] = IALw[i][j];
           jU++;
        } else {
           IAL[i][jL] = IALw[i][j];
           jL++;
        }
     }

     for(j=0; j<INUw[i]; j++) {
         if(IAUw[i][j] > i+1) {
            IAU[i][jU] = IAUw[i][j];
            jU++;
         } else {
            IAL[i][jL] = IAUw[i][j];
            jL++;
         }
      }
      INL[i] = jL;
      INU[i] = jU;
}
        free(IW);
        free(INLw);          free(INUw);
        free(IALw);          free(IAUw);

        return 0;
}
```

"Final" number of upper/lower components (column ID) in the renumbered new matrix.

- **INL**
- **INU**

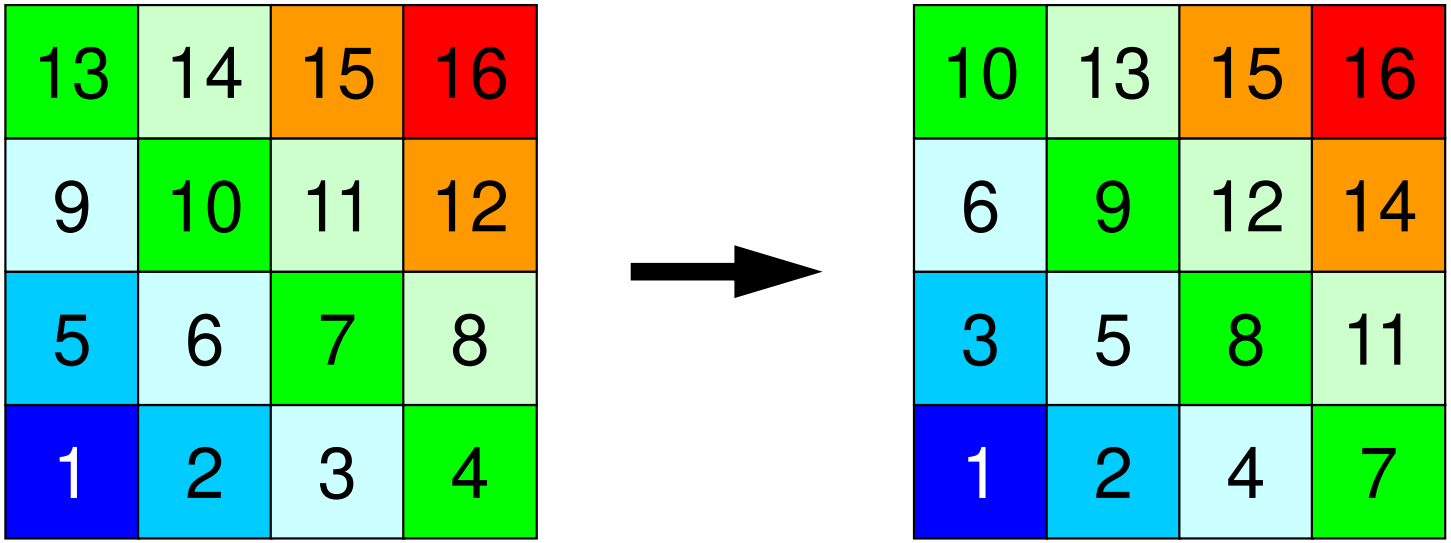
修正されたCM法
並列計算向け

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。同じレベル：連続した「新」要素番号

CM法の手順

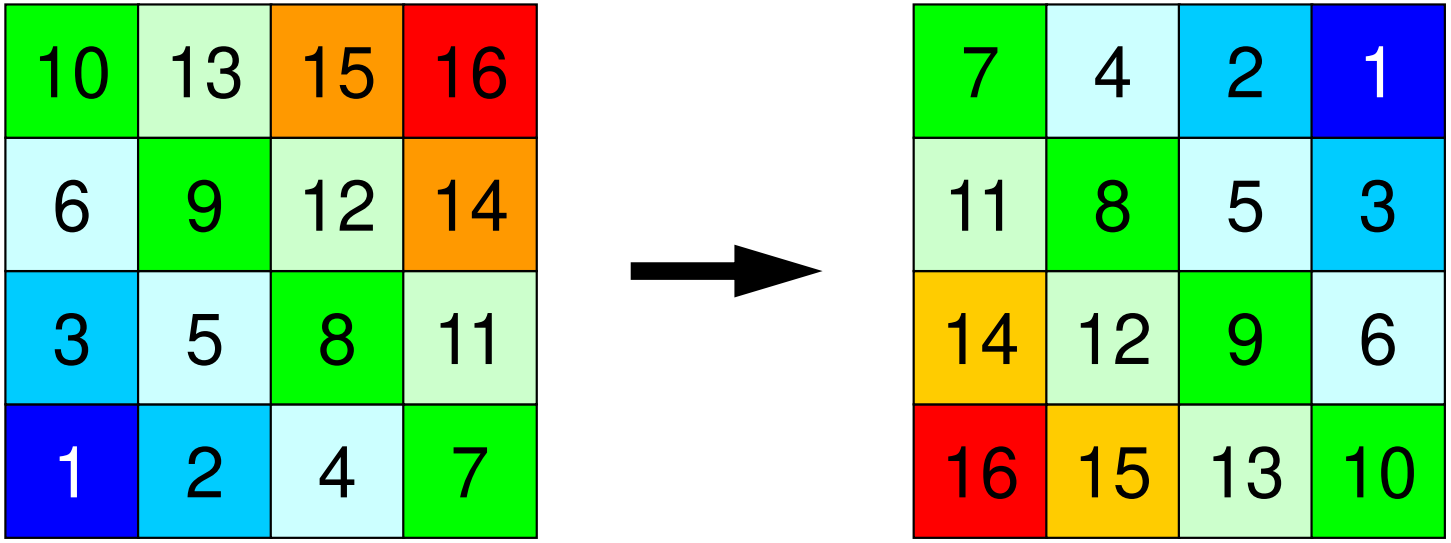


- 手順3：繰り返し，再番号付け
  - 「レベル(k)」に属する条件を満たす要素が無くなったら，k=k+1として，手順2を繰り返し，全要素の「レベル」が決定したら終了
  - 「レベル」の若い順に要素番号をふり直す

RCM（Reverse CM）

- まずCMの手順を実行
  - 手順1：次数（degree）の計算
  - 手順2：「レベル（k（k≧2））」の要素の選択
  - 手順3：繰り返し，再番号付け

- 手順4：再々番号付け
  - CMの番号付けを更に逆順にふり直す
  - Fill-inがCMの場合より少なくなる



cm (1/5)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "rcm.h"

extern int
CM(int N, int NL, int NU, int *INL, int **IAL, int *INU, int **IAU,
   int *NCOLORtot, int *COLORindex, int *NEWtoOLD, int *OLDtoNEW)
{
        int **IW;
        int *INLw, *INUw;
        int **IALw, **IAUw;
        int KC, KCT, KCTO, KMIN, KMAX;
        int JC, JN;
        int II;
        int i, j, k, ic, jL, jU;
        int INmin, NODmin;
        int icon, icol, icouG, icou, icou0, in, inC;
```

cm (2/5)

```
IW = (int **)calloc(2, sizeof(int *));
if(IW == NULL) {
   fprintf(stderr, "Error: %s¥n", strerror(errno));
   return -1;
}

for(i=0; i<2; i++) {
    IW[i] = (int *)calloc(N, sizeof(int));
}

INmin  = N;
NODmin = -1;

for(i=0; i<N; i++) {
    icon = 0;
    for(k=0; k<INL[i]; k++) {
        icon++;
    }
    for(k=0; k<INU[i]; k++) {
        icon++;
    }

    if(icon < INmin) {
       INmin  = icon;
       NODmin = i;
    }
}

if(NODmin == -1) NODmin = 0;

IW[1][NODmin] = 1;

NEWtoOLD[0]      = NODmin+1;
OLDtoNEW[NODmin] = 1;

icol = 0;
```

IW[0][i]:
Work array

IW[1][i]:
“Level ID” of each mesh starting at “1”

cm (2/5)

```
IW = (int **)calloc(2, sizeof(int *));
if(IW == NULL) {
   fprintf(stderr, "Error: %s¥n", strerror(errno));
   return -1;
}

for(i=0; i<2; i++) {
    IW[i] = (int *)calloc(N, sizeof(int));
}

INmin  = N;
NODmin = -1;

for(i=0; i<N; i++) {
    icon = 0;
    for(k=0; k<INL[i]; k++) {
        icon++;
    }
    for(k=0; k<INU[i]; k++) {
        icon++;
    }

    if(icon < INmin) {
       INmin  = icon;
       NODmin = i;
    }
}

if(NODmin == -1) NODmin = 0;

IW[1][NODmin] = 1;

NEWtoOLD[0]      = NODmin+1;
OLDtoNEW[NODmin] = 1;

icol = 0;
```

NODmin:
接続要素数(次数)が最小の要素を探索

cm (2/5)

```
IW = (int **)calloc(2, sizeof(int *));
if(IW == NULL) {
   fprintf(stderr, "Error: %s¥n", strerror(errno));
   return -1;
}

for(i=0; i<2; i++) {
    IW[i] = (int *)calloc(N, sizeof(int));
}

INmin  = N;
NODmin = -1;

for(i=0; i<N; i++) {
    icon = 0;
    for(k=0; k<INL[i]; k++) {
        icon++;
    }
    for(k=0; k<INU[i]; k++) {
        icon++;
    }

    if(icon < INmin) {
       INmin  = icon;
       NODmin = i;
    }
}

if(NODmin == -1) NODmin = 0;

IW[1][NODmin] = 1;

NEWtoOLD[0]      = NODmin+1;
OLDtoNEW[NODmin] = 1;

icol = 0;
```

New mesh ID of **NODmin**
is set to 0.
Level ID of **NODmin** is set to 1

OLDtoNEW[NODmin]= 1
NEWtoOLD[1]= NODmin+1

IW[1][NODmin]=1: Level ID

cm (3/5)

```
icouG = 1;
for(icol=1; icol<N; icol++) {
    icou  = 0;
    icou0 = 0;
    for(i=0; i<N; i++) {
        if(IW[1][i] == icol) {
           for(k=0; k<INL[i]; k++) {
               in = IAL[i][k];
               if(IW[1][in-1] == 0) {
                  IW[1][in-1] = -(icol + 1);
                  icou++;
                  IW[0][icou-1] = in;
               }
           }
           for(k=0; k<INU[i]; k++) {
               in = IAU[i][k];
               if(IW[1][in-1] == 0) {
                  IW[1][in-1] = -(icol + 1);
                  icou++;
                  IW[0][icou-1] = in;
               }
           }
        }
    }

    if(icou == 0) {
       for(i=0; i<N; i++) {
           if(IW[1][i] == 0) {
              icou++;
              IW[1][i     ] = -(icol + 1);
              IW[0][icou-1] = i + 1;
             break;
           }
        }
     }
```

`icouG` ： 全体のカウンタ
`icou` ： レベル内のカウンタ

「レベル」に関するループ

cm (3/5)

```
icouG = 1;
for(icol=1; icol<N; icol++) {
    icou  = 0;
    icou0 = 0;
    for(i=0; i<N; i++) {
        if(IW[1][i] == icol) {
           for(k=0; k<INL[i]; k++) {
               in = IAL[i][k];
               if(IW[1][in-1] == 0) {
                  IW[1][in-1] = -(icol + 1);
                  icou++;
                  IW[0][icou-1] = in;
               }
           }
           for(k=0; k<INU[i]; k++) {
               in = IAU[i][k];
               if(IW[1][in-1] == 0) {
                  IW[1][in-1] = -(icol + 1);
                  icou++;
                  IW[0][icou-1] = in;
               }
           }
        }
    }

    if(icou == 0) {
       for(i=0; i<N; i++) {
           if(IW[1][i] == 0) {
              icou++;
              IW[1][i     ] = -(icol + 1);
              IW[0][icou-1] = i + 1;
              break;
           }
        }
     }
```

`icouG` ： 全体のカウンタ
`icou` ： レベル内のカウンタ

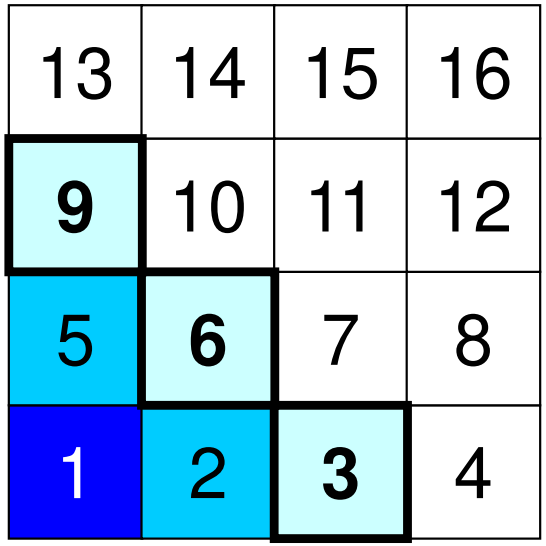
「レベル」内での各要素に関するループ

「`in-1`」番目のメッシュが「`i`」番目のメッシュに隣接し，`IW[1][i]=icol`（所属レベル番号：`icol`）であり，「`in-1`」番目のメッシュのレベルが決まっていなければ，「`in-1`」番目のメッシュは「`icol+1`」番目のレベルに属する可能性がある。

- **`IW[1][in-1]= -(icol+1)`**
- **`icou= icou + 1`**
- **`IW[0][icou-1]= in`: この配列には「見つかった」順番に格納される**

mesh, color: 0から開始
IAL[:][:]:　　1から開始

What does it mean ?



icol=4-1
IW[1][i]= icol-1=2: i=3,6,9

icouG ： 全体のカウンタ
icou ： レベル内のカウンタ

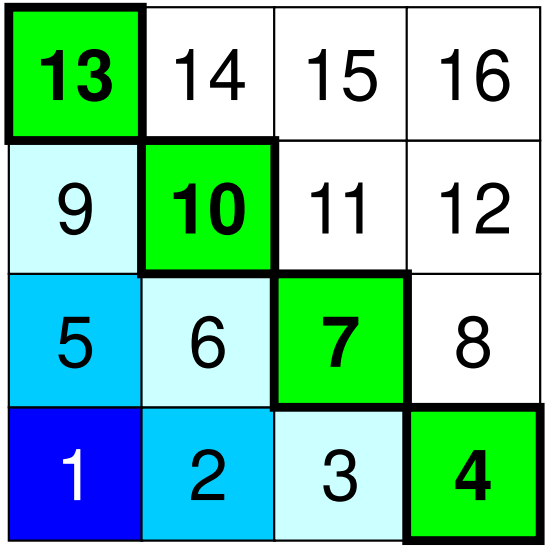
「レベル」内での各要素に関するループ

「in-1」番目のメッシュが「i」番目のメッシュに隣接し，IW[1][i]=icol（所属レベル番号：icol）であり，「in-1」番目のメッシュのレベルが決まっていなければ，「in-1」番目のメッシュは「icol+1」番目のレベルに属する可能性がある。

- **IW[1][in-1]= -(icol+1)**
- **icou= icou + 1**
- **IW[0][icou-1]= in**: この配列には「見つかった」順番に格納される

mesh, color: 0から開始
IAL[:][:]: 1から開始

What does it mean ?




```
icol=4-1
 IW[1][i]= icol-1=2: i=3,6,9
```

```
IW[1][ 3(= 4-1)]= -4
IW[1][ 6(= 7-1)]= -4
IW[1][ 9(=10-1)]= -4
IW[1][12(=13-1)]= -4

IW[0][0]=  4
IW[0][1]=  7
IW[0][2]= 10
IW[0][3]= 13
```

icouG ： 全体のカウンタ
icou ： レベル内のカウンタ

「レベル」内での各要素に関するループ

「in-1」番目のメッシュが「i」番目のメッシュに隣接し，IW[1][i]=icol（所属レベル番号：icol）であり，「in-1」番目のメッシュのレベルが決まっていなければ，「in-1」番目のメッシュは「icol+1」番目のレベルに属する可能性がある。

- **IW[1][in-1]= -(icol+1)**
- **icou= icou + 1**
- **IW[0][icou-1]= in**: この配列には「見つかった」順番に格納される

mesh, color: 0から開始
IAL[:][:]: 1から開始

cm (3/5)

```
icouG = 1;
for(icol=1; icol<N; icol++) {
    icou  = 0;
    icou0 = 0;
    for(i=0; i<N; i++) {
        if(IW[1][i] == icol) {
           for(k=0; k<INL[i]; k++) {
               in = IAL[i][k];
               if(IW[1][in-1] == 0) {
                  IW[1][in-1] = -(icol + 1);
                  icou++;
                  IW[0][icou-1] = in;
               }
           }
           for(k=0; k<INU[i]; k++) {
               in = IAU[i][k];
               if(IW[1][in-1] == 0) {
                  IW[1][in-1] = -(icol + 1);
                  icou++;
                  IW[0][icou-1] = in;
               }
           }
        }
    }

    if(icou == 0) {
       for(i=0; i<N; i++) {
           if(IW[1][i] == 0) {
              icou++;
              IW[1][i     ] = -(icol + 1);
              IW[0][icou-1] = i + 1;
              break;
           }
        }
     }
```

`icouG` : 全体のカウンタ
`icou` : レベル内のカウンタ

もし `icou=0` となったら，まだ所属レベルが決定しない要素の中で最も要素番号が小さいものを選び出す(通常はこのループは通らない)

cm (4/5)

```
for(icol=1; icol<N; icol++) {
....

for(ic=0; ic<icou; ic++) {
    inC = IW[0][ic];
    if(IW[1][inC-1] != 0) {
       for(k=0; k<INL[inC-1]; k++) {
           in = IAL[inC-1][k];
           if(IW[1][in-1] <= 0) {
              IW[1][in-1] = 0;
           }
       }
       for(k=0; k<INU[inC-1]; k++) {
           in = IAU[inC-1][k];
           if(IW[1][in-1] <= 0) {
              IW[1][in-1] = 0;
           }
       }
    }
}

for(ic=0; ic<icou; ic++) {
    inC = IW[0][ic];
    if(IW[1][inC-1] != 0) {
       icouG++;
       IW[1][inC-1] = icol + 1;
    }
}

if(icouG == N) break;
}
```

所属レベル「icol」の候補となっている要素の番号が下記に格納されている:

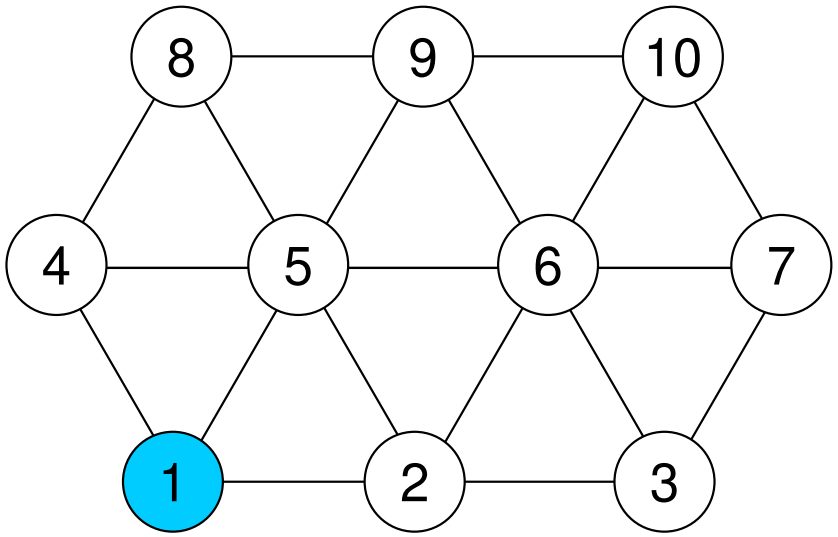
- **IW[0][ic], ic= 0~icou**

所属レベルicolの候補となっている要素に隣接している要素は候補からはずす(隣接する要素同士は同じレベルには属することができない)

そのような要素inについては, 下記のよな操作を実行する:

- **IW[1][in-1]= 0**

どういうことかというと



e.g.
Mesh (1) belongs to **(icol-1)th** level

所属レベル「icol」の候補となっている要素の番号が下記に格納されている：

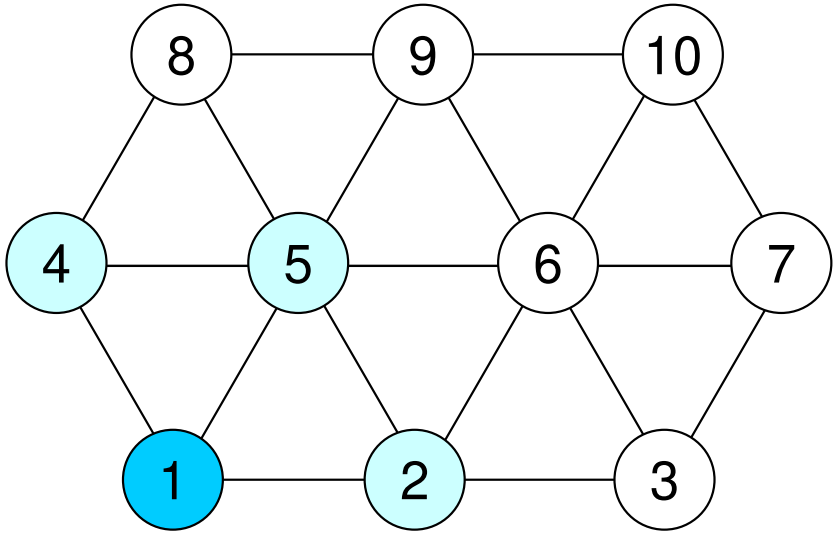
- **IW[0][ic], ic= 0~icou**

所属レベルicolの候補となっている要素に隣接している要素は候補からはずす（隣接する要素同士は同じレベルには属することができない）

そのような要素inについては，下記のよな操作を実行する：

- **IW[1][in-1]= 0**

どういうことかというと



(2),(4) and (5) are candidates for **(icol)th** level

```
IW[1][2-1]= -(icol+1)
IW[1][4-1]= -(icol+1)
IW[1][5-1]= -(icol+1)

IW[1][0]= 2
IW[1][1]= 4
IW[1][2]= 5
```

所属レベル「icol」の候補となっている要素の番号が下記に格納されている:

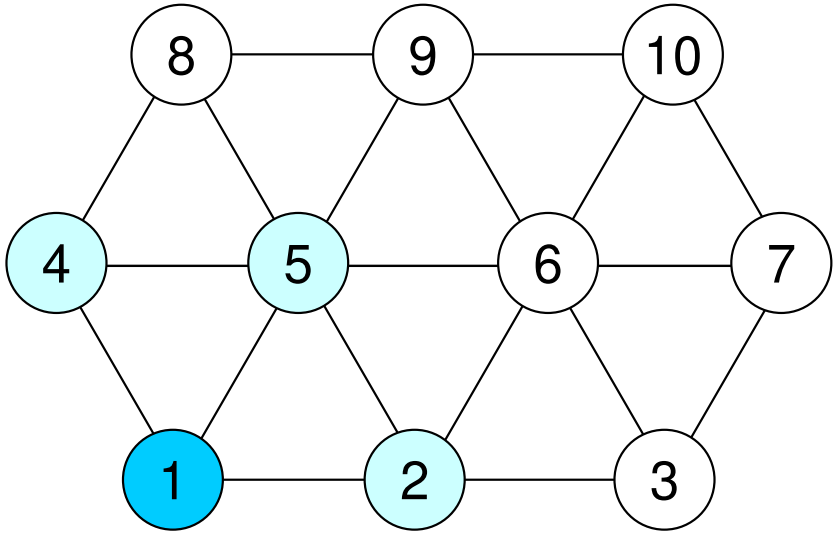
- **IW[0][ic], ic= 0~icou**

所属レベルicolの候補となっている要素に隣接している要素は候補からはずす（隣接する要素同士は同じレベルには属することができない）

そのような要素inについては，下記のよな操作を実行する:

- **IW[1][in-1]= 0**

どういうことかというと



Considering dependency:

```
IW[1][2-1]= -(icol+1)
IW[1][4-1]= -(icol+1)
IW[1][5-1]= 0

IW[1][0]= 2
IW[1][1]= 4
IW[1][2]= 5
```

所属レベル「icol」の候補となっている要素の番号が下記に格納されている:

- **IW[0][ic], ic= 0~icou**

所属レベルicolの候補となっている要素に隣接している要素は候補からはずす（隣接する要素同士は同じレベルには属することができない）

そのような要素inについては，下記のよな操作を実行する:

- **IW[1][in-1]= 0**

cm (4/5)

```
for(icol=1; icol<N; icol++) {
....

for(ic=0; ic<icou; ic++) {
    inC = IW[0][ic];
    if(IW[1][inC-1] != 0) {
       for(k=0; k<INL[inC-1]; k++) {
            in = IAL[inC-1][k];
            if(IW[1][in-1] <= 0) {
               IW[1][in-1] = 0;
            }
       }
       for(k=0; k<INU[inC-1]; k++) {
            in = IAU[inC-1][k];
            if(IW[1][in-1] <= 0) {
               IW[1][in-1] = 0;
            }
       }
    }
}

for(ic=0; ic<icou; ic++) {
    inC = IW[0][ic];
    if(IW[1][inC-1] != 0) {
       icouG++;
       IW[1][inC-1] = icol + 1;
    }
}

if(icouG == N) break;
}
```

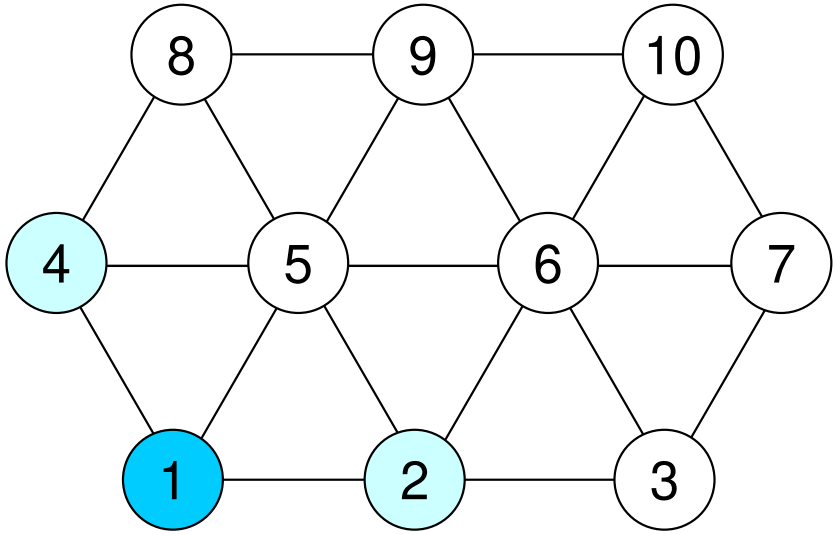
icouG ： 全体のカウンタ
icou ： レベル内のカウンタ

IW[1][inC-1]=-(icol+1),を満たす要素inCが最終的にレベル番号「icol+1」に所属する。

そのようなメッシュinCに対して下記のようにする:

IW[1][inC-1]= icol + 1.
icouG= icouG + 1

どういうことかというと



Considering dependency:

```
IW[1][2-1]= -(icol+1)
IW[1][4-1]= -(icol+1)
IW[1][5-1]= 0
IW[1][0]= 2
IW[1][1]= 4
IW[1][2]= 5
```

Finally:

```
IW[1][2-1]= icol
IW[1][4-1]= icol
```

IW[1][inC-1]=-(icol+1),を満たす要素inCが最終的にレベル番号「icol+1」に所属する。

そのようなメッシュinCに対して下記のようにする:

IW[1][inC-1]= icol + 1.
icouG= icouG + 1

cm (4/5)

```
for(icol=1; icol<N; icol++) {
....

for(ic=0; ic<icou; ic++) {
    inC = IW[0][ic];
    if(IW[1][inC-1] != 0) {
       for(k=0; k<INL[inC-1]; k++) {
            in = IAL[inC-1][k];
            if(IW[1][in-1] <= 0) {
               IW[1][in-1] = 0;
            }
       }
       for(k=0; k<INU[inC-1]; k++) {
            in = IAU[inC-1][k];
            if(IW[1][in-1] <= 0) {
               IW[1][in-1] = 0;
            }
       }
    }
}

for(ic=0; ic<icou; ic++) {
    inC = IW[0][ic];
    if(IW[1][inC-1] != 0) {
       icouG++;
       IW[1][inC-1] = icol + 1;
    }
}

if(icouG == N) break;
}
```

icouG : 全体のカウンタ
icou : レベル内のカウンタ

icouG=Nとなっていたら，全要素の所属レベルが決まったことになるので，終了。

そうでない場合は，レベル数を一つ増やして，探索を継続する。

cm (5/5)

```
/*******************/
/* FINAL COLORING */
/*******************/

*NCOLORtot = icol + 1;
icouG = 0;

for(ic=1; ic<=(*NCOLORtot); ic++) {
    icou = 0;
    for(i=0; i<N; i++) {
        if(IW[1][i] == ic) {
           NEWtoOLD[icouG] = i+1;
           OLDtoNEW[i    ] = icouG+1;
           icou++;
           icouG++;
        }
    }
    COLORindex[ic] = icou;
}

COLORindex[0] = 0;

for(ic=1; ic<=(*NCOLORtot); ic++) {
    COLORindex[ic] = COLORindex[ic-1] +
                     COLORindex[ic];
}

/*******************
 * MATRIX transfer *
 *******************/
```

icouG=Nとなった時点でのレベル数icolを総レベル数 NCOLORtot とする。

各要素所属レベルに従って，レベル番号の少ない方から，要素の再番号付けを行なう。

```
OLDtoNEW[old_ID]= new_ID + 1
NEWtoOLD[new_ID]= old_ID + 1
```

この時点では「COLORindex」には各色の要素数が入っている。

cm (5/5)

```
/******************/
/* FINAL COLORING */
/******************/

*NCOLORtot = icol + 1;
icouG = 0;

for(ic=1; ic<=(*NCOLORtot); ic++) {
    icou = 0;
    for(i=0; i<N; i++) {
        if(IW[1][i] == ic) {
           NEWtoOLD[icouG] = i+1;
           OLDtoNEW[i    ] = icouG+1;
           icou++;
           icouG++;
        }
    }
    COLORindex[ic] = icou;
}

COLORindex[0] = 0;

for(ic=1; ic<=(*NCOLORtot); ic++) {
    COLORindex[ic] = COLORindex[ic-1] +
                     COLORindex[ic];
}

/********************
 * MATRIX transfer *
 ********************/
```

COLODindex[ic]:

Now it is 1D index.

MCとRCMの比較

- MC
  - 並列性高い，負荷分散も良い（そのように設定されている）
  - 特に色数少ないと反復回数多い
  - 色数を増やす，反復回数減るが同期オーバーヘッドの影響で性能低下（反復回数あたりの計算時間増加）の可能性あり
- RCM
  - 収束は早いが，レベル数が多く，同期オーバーヘッドの影響受けやすく，コア数が増えると不利
  - 負荷分散もいまーつ
- 反復回数少なくて同期オーバーヘッドの影響が少ない方法が無いものか？
  - 色数が少なくて，かつ反復回数が少ないという都合の良い方法

色数増加⇒同期オーバーヘッド増加

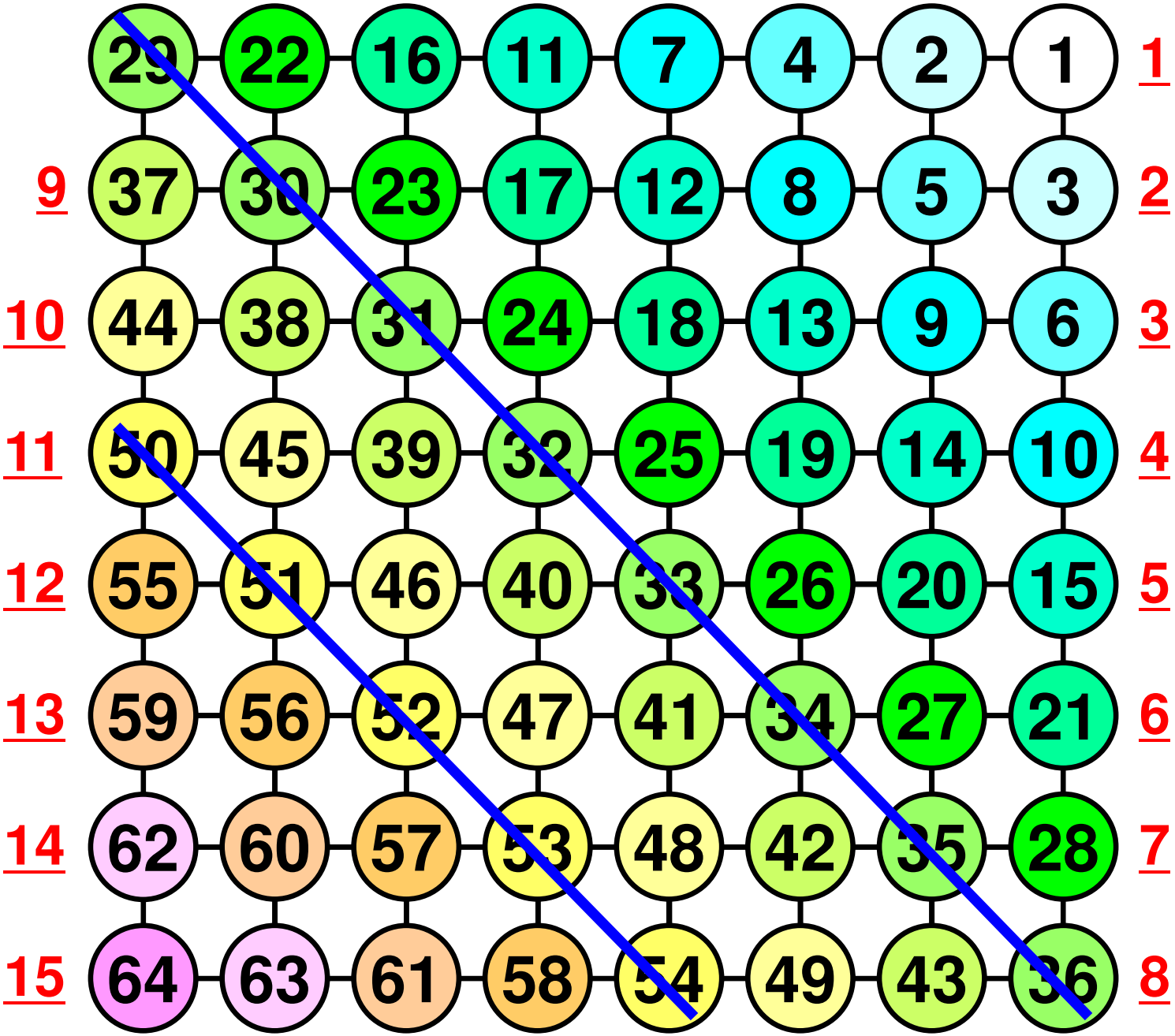
必ずある「色」の計算が終わってから次の「色」に行く

```
      do ic= 1, NCOLORtot
!$omp parallel do private(ip,ip1,i,WVAL,k)
        do ip= 1, PEsmpTOT
          ip1= (ic-1)*PEsmpTOT + ip
        do i= SMPindex(ip1-1)+1, SMPindex(ip1)
          WVAL= W(i,Z)
          do k= indexL(i-1)+1, indexL(i)
            WVAL= WVAL -  AL(k) * W(itemL(k),Z)
          enddo
          W(i,Z)= WVAL * W(i,DD)
        enddo
        enddo
!$omp end parallel do
      enddo
```

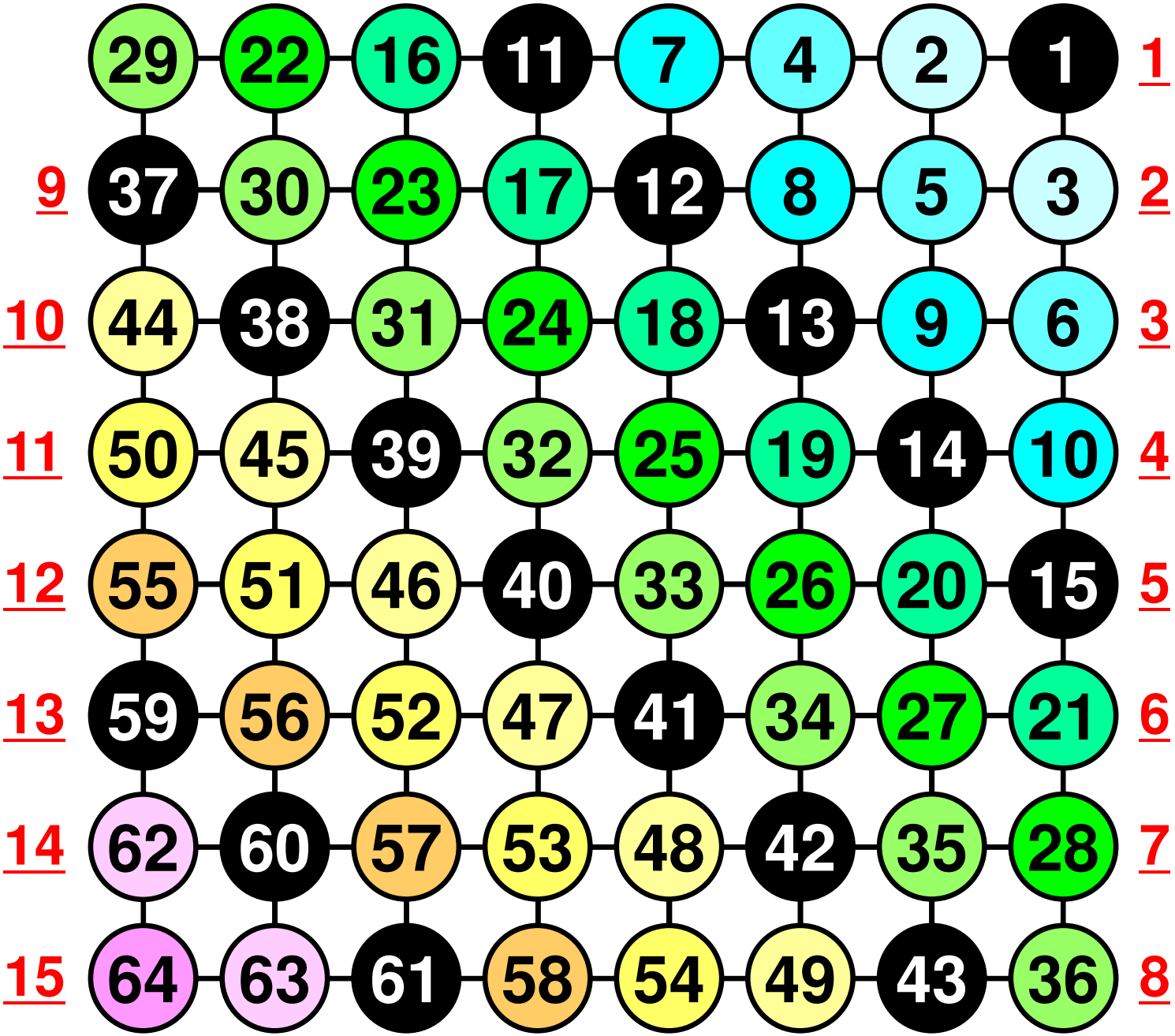
解決策：CM-RCM

- RCM＋Cyclic Multicoloring 〔土肥，襲田，鷲尾他〕
- 手順
  - まずRCMを施す
  - Cyclic Multicoloring（CM）の色数を決める（Nc）
  - RCMの1番目，（Nc+1）番目，（2Nc+1）番目・・・のレベルに属する要素を「1」色に分類する
  - RCMのk番目，（Nc+k）番目，（2Nc+k）番目・・・のレベルに属する要素を「k」色に分類する
  - 「k」が「Nc」に達して，要素が「1~Nc」で色付けされたら完了
    - あとはMCのときと同じように，色の順番に再番号付
    - RCMの各レベルに対して「Nc」のサイクルで再色付けを実施している
  - もし同じ色の要素の中に依存性が見つかったら，Nc=Nc+1として最初からやり直し（ここは少し原始的）

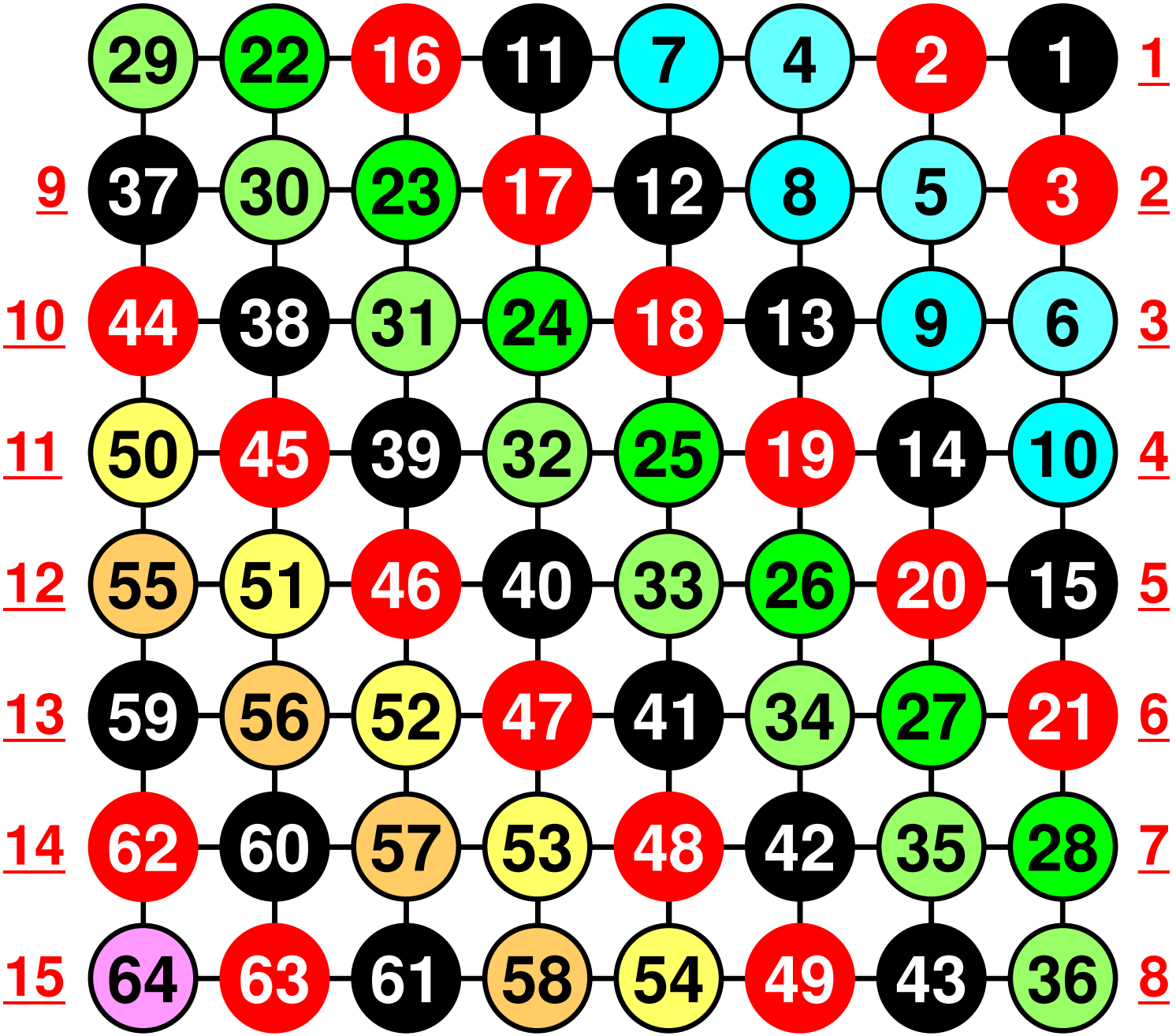
RCM


29 22 16 11 7 4 2 1
1
9 37 30 23 17 12 8 5 3
2
10 44 38 31 24 18 13 9 6
3
11 50 45 39 32 25 19 14 10
4
12 55 51 46 40 33 26 20 15
5
13 59 56 52 47 41 34 27 21
6
14 62 60 57 53 48 42 35 28
7
15 64 63 61 58 54 49 43 36
8

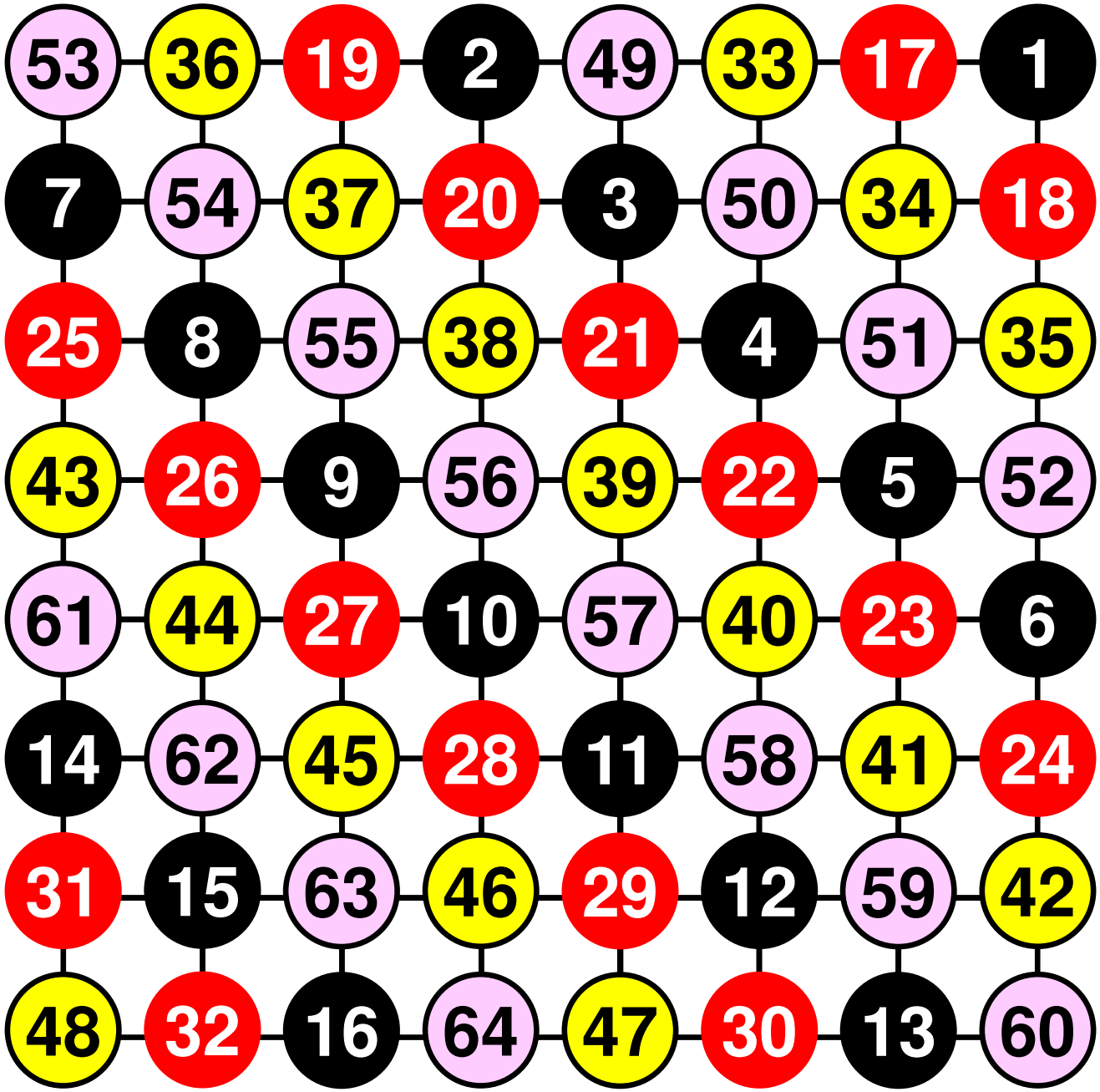
Nc=4, k=1 : 1,5,9,13レベルを選択



Nc=4, k=2:2,6,10,14レベルを選択



CM-RCM(Nc=4):「色」の順番に再並替



k=1: 16
k=2: 16
k=3: 16
k=4: 16

CM-RCM

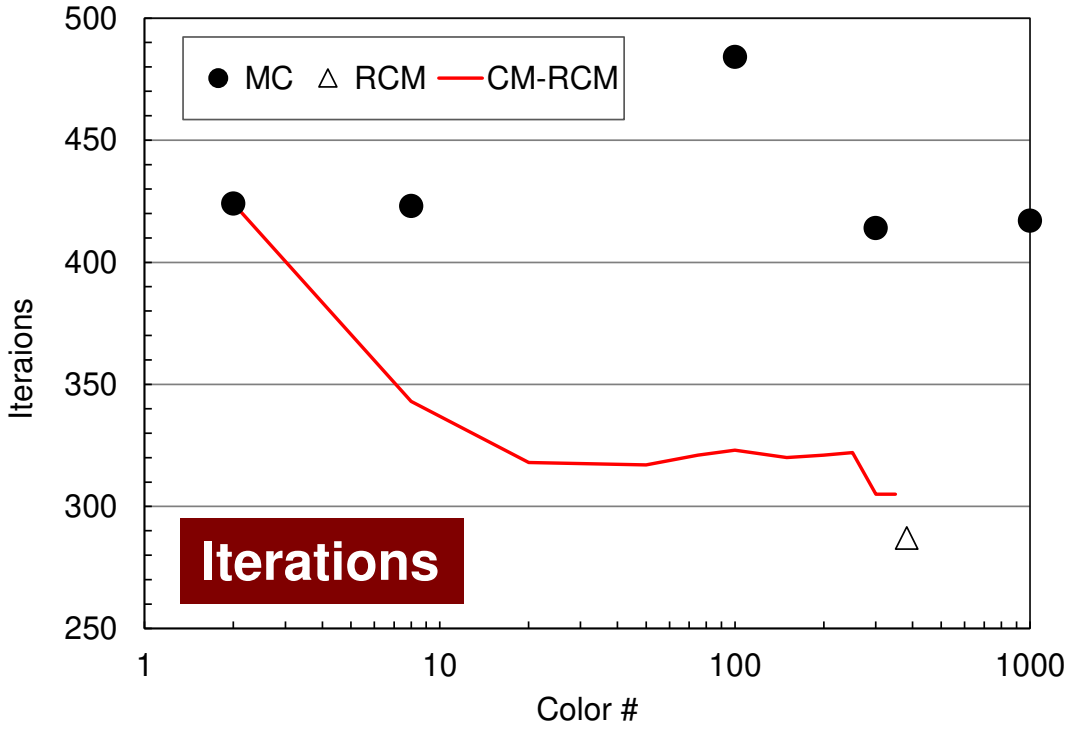
- 実行方法
  - INPUT.DATで「**NCOLORtot=-Nc**」とする
  - L2に有効なオプションとして導入済み

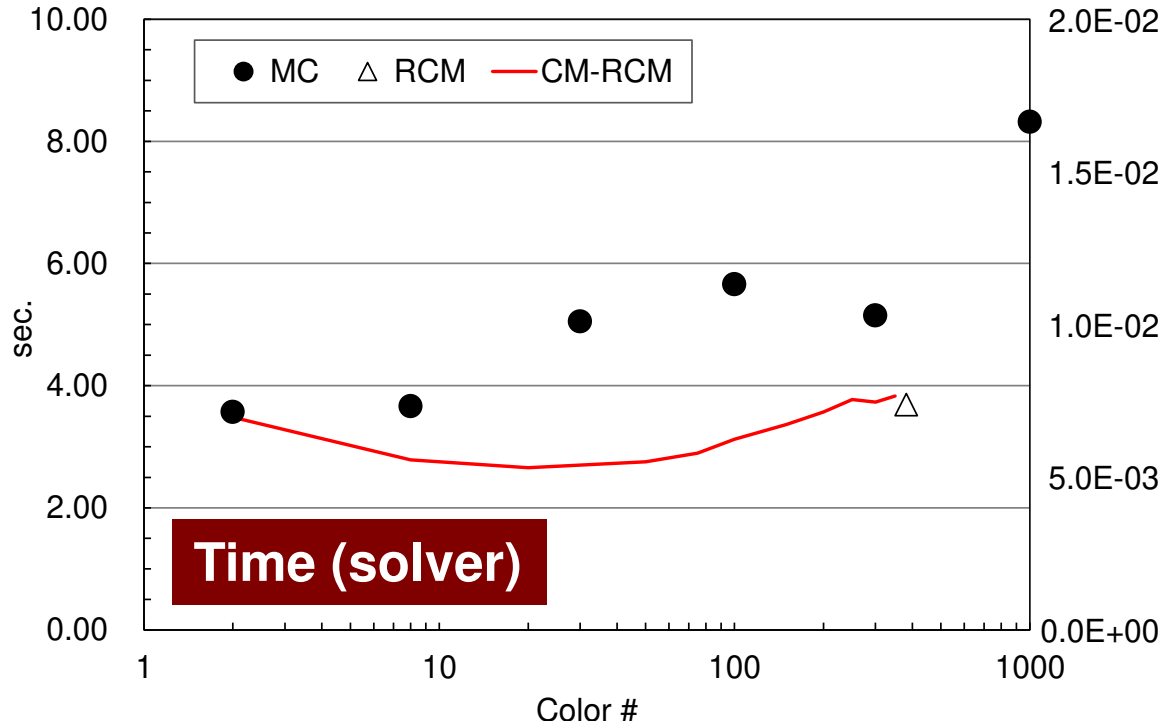
- 実装は「cmrcm.f」を参照ください

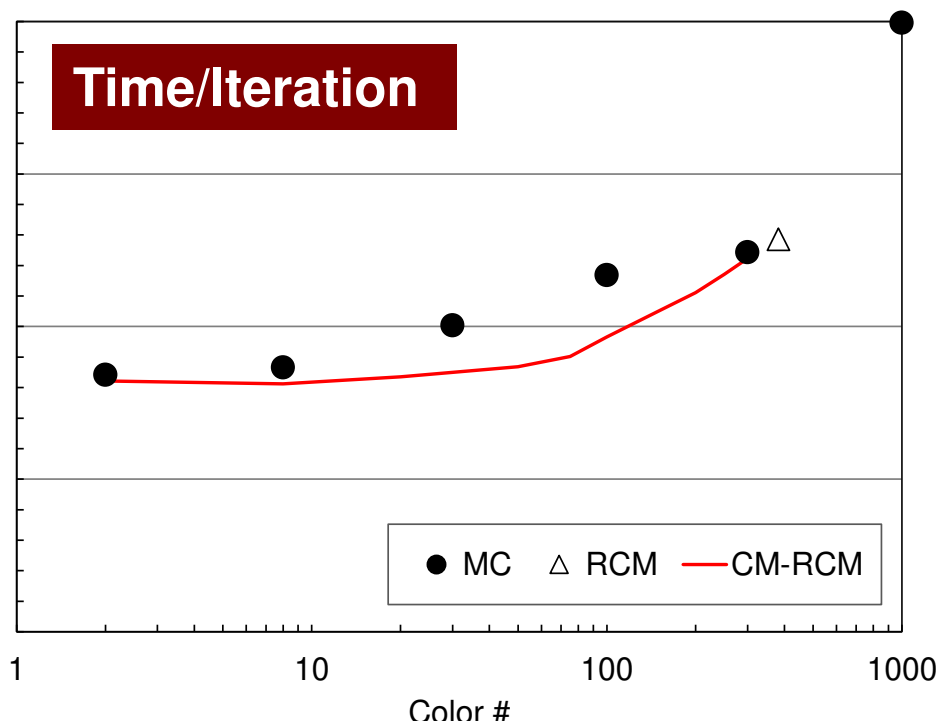
Odyssey

1-CMG/12-cores, 128^3

(●:MC, △:RCM, -:CM-RCM)

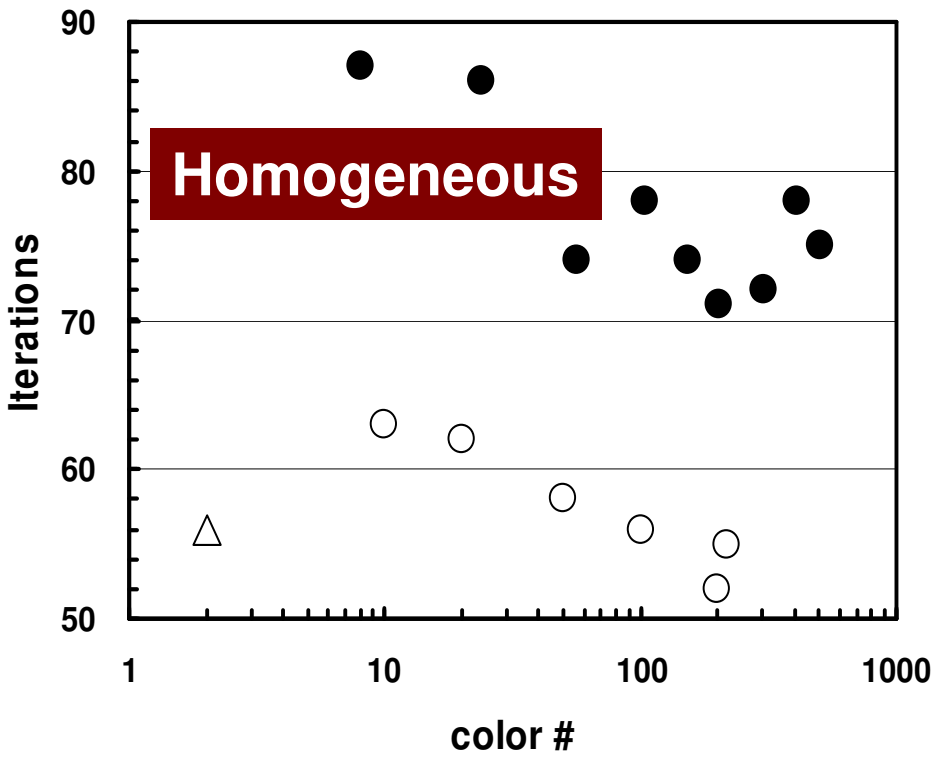


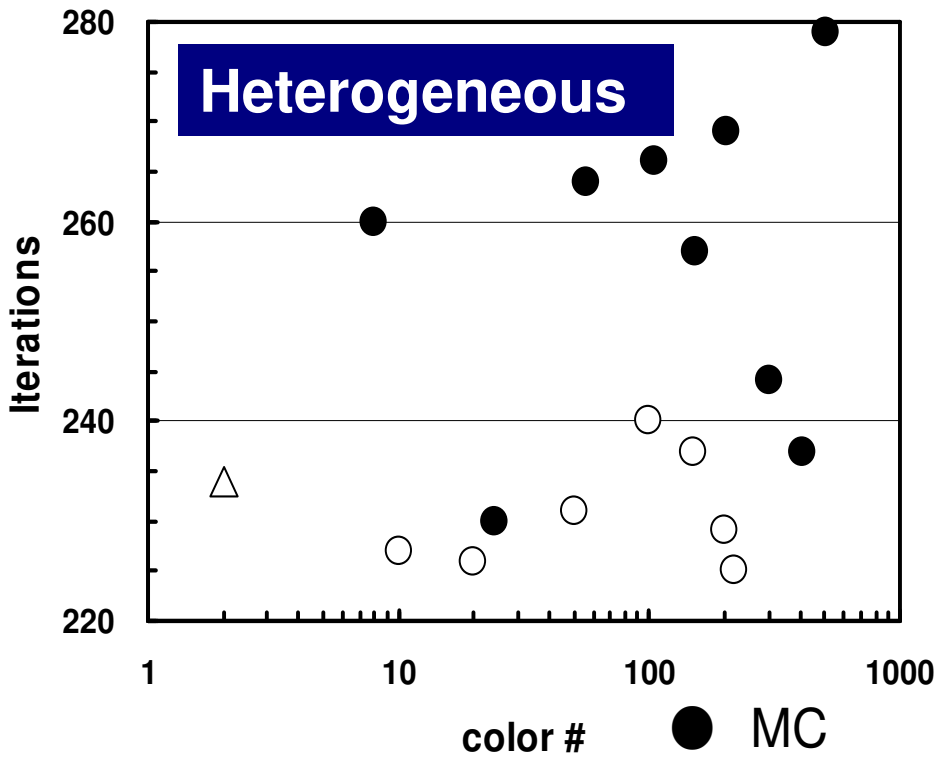




Comparison of Reordering Methods 3D Linear Elastic Problems

- MC: Slow convergence, unstable for heterogeneous cases (ill-conditioned problems).
- Cyclic-Mulricoloring + RCM (CM-RCM) is effective





MC
CM-RCM
No reordering

3D Linear-Elastic Problems with 32,768 DOF

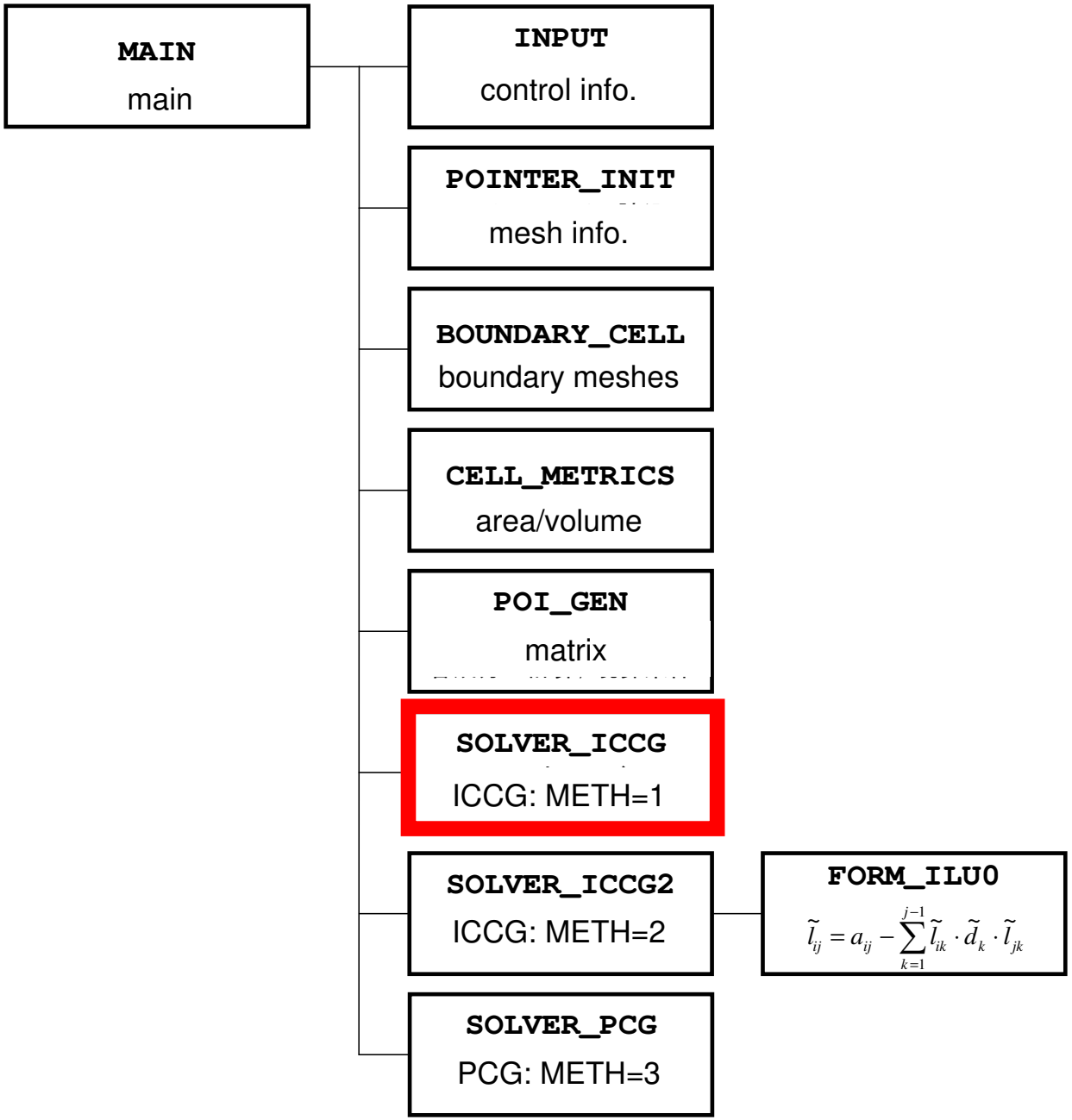
- データ依存性の解決策は?
- オーダリング(Ordering)について
  - Red-Black, Multicolor(MC)
  - Cuthill-McKee(CM), Reverse-CM(RCM)
  - オーダリングと収束の関係
- オーダリングの実装
- **オーダリング付ICCG法の実装**
- マルチコアへの実装(OpenMP)へ向けて

オーダリング付きICCG法の実装

- 「L2-color」の機能を「L1-sol」へ組み込む

- 「POI_GEN」において，INU,INL,IAL,IAUを求めた時点で，「mc」，「cm」，「rcm」を呼ぶ。
- AL，AUを新しいオーダリングに準じて計算する。
- 境界条件，右辺（体積フラックス項）を新しいオーダリングに準じて計算する。
- ソルバーを呼ぶ。
- 結果（PHI）を古いオーダリングに戻す。
- UCDファイルを書き出す（OUPUT_UCDを呼ぶ）

L1-sol


MAIN
main
INPUT
control info.
POINTER_INIT
mesh info.
BOUNDARY_CELL
boundary meshes
CELL_METRICS
area/volume
POI_GEN
matrix
SOLVER_ICCG
ICCG: METH=1
SOLVER_ICCG2
ICCG: METH=2
FORM_ILU0
$\tilde{l}_{ij} = a_{ij} - \sum_{k=1}^{j-1} \tilde{l}_{ik} \cdot \tilde{d}_k \cdot \tilde{l}_{jk}$
SOLVER_PCG
PCG: METH=3

Minv{r}={z} (1/2)

Forward Substitution $(L)\{z\}=\{r\}$ $(M)\{z\}=(LDL^T)\{z\}=\{r\}$

```
for(i=0; i<N; i++) {
     WVAL = W[Z][i];
     for(j=indexL[i]; j<indexL[i+1]; j++) {
          WVAL -= AL[j] * W[Z][itemL[j]-1];
     }
     W[Z][i] = WVAL * W[DD][i];
}
```

Backward Substitution $(DL^T)\{z\}=\{z\}$

```
for(i=N-1; i>=0; i--) {
     SW = 0.0;
     for(j=indexU[i]; j<indexU[i+1]; j++) {
     SW += AU[j] * W[Z][itemU[j]-1];
     }
     W[Z][i]= W[Z][i]  - W[DD][i] * SW;
}
```

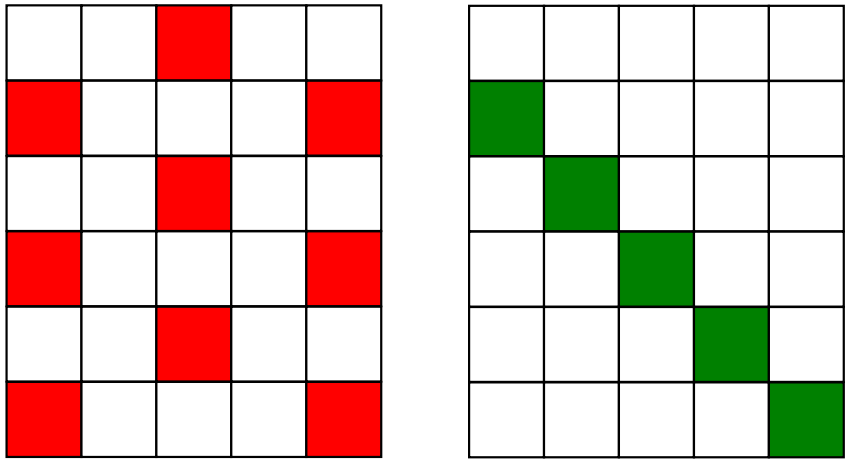
データ依存性あり。
ある点におけるZを求めるために，他の点におけるZが右辺に現れる:
⇒ 並列化困難

右辺に自身に依存しないZが現れるようにすればよい
⇒ オーダリング

Minv{r}={z} (2/2)

Forward Substitution $(L)\{z\}=\{r\}$ $(M)\{z\}=(LDL^T)\{z\}=\{r\}$

```
for(icol=0; icol<NCOLORtot; icol++){
    for(i=COLORindex[icol]; i<COLORindex[icol+1]; i++) {
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
}
```



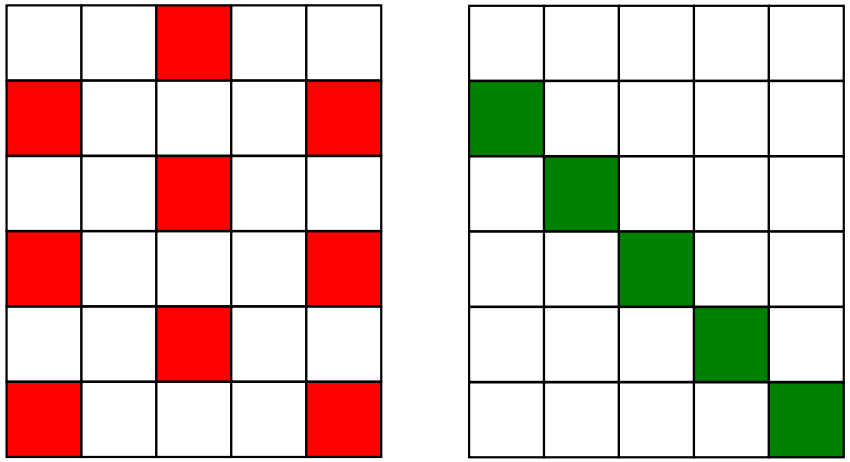
右辺に現れる「Z」は，必ず，現在の「色」（icol）以外の色に属している。
同じ色に属している要素はお互いに依存性，関連性を持たない。

⇒データ依存性回避可能

Minv{r}={z} (2/2)

Forward Substitution $(L)\{z\}=\{r\}$ $(M)\{z\}=\left(LDL^T\right)\{z\}=\{r\}$

```
for(icol=0; icol<NCOLORtot; icol++){
    for(i=COLORindex[icol]; i<COLORindex[icol+1]; i++) {
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
}
```

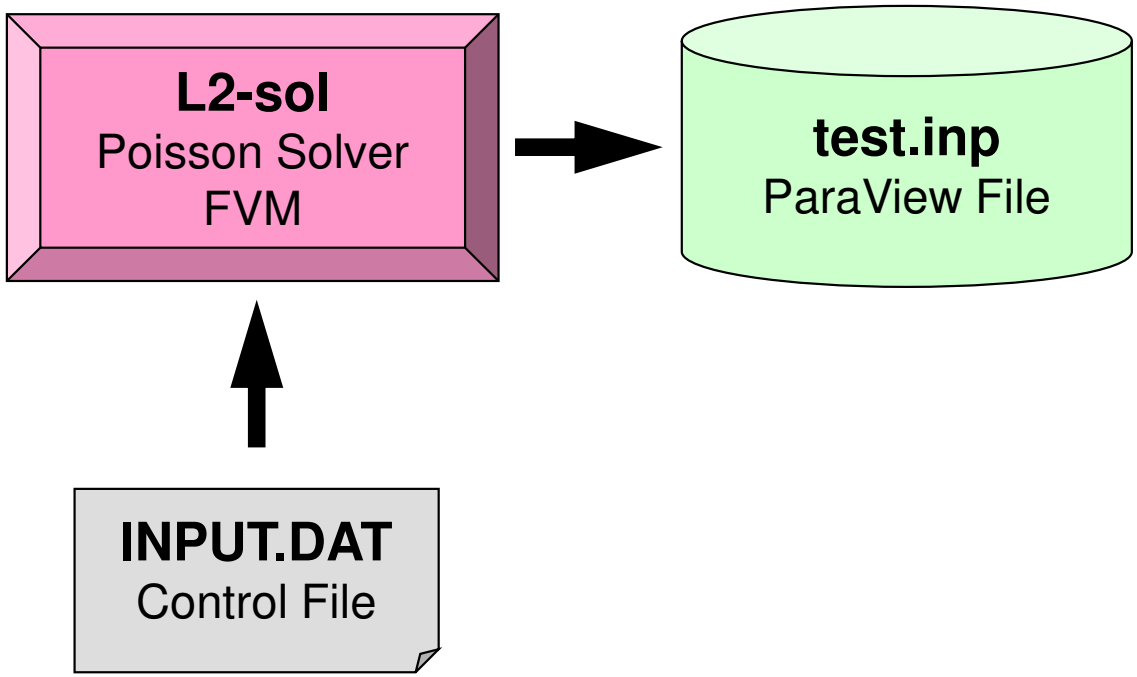


このループ（各色内のループ）は並列，独立に計算可能である。

Files

```
$> cd <$P-L2>/solver/run

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

Running the Program <$P-L2>/solver/run

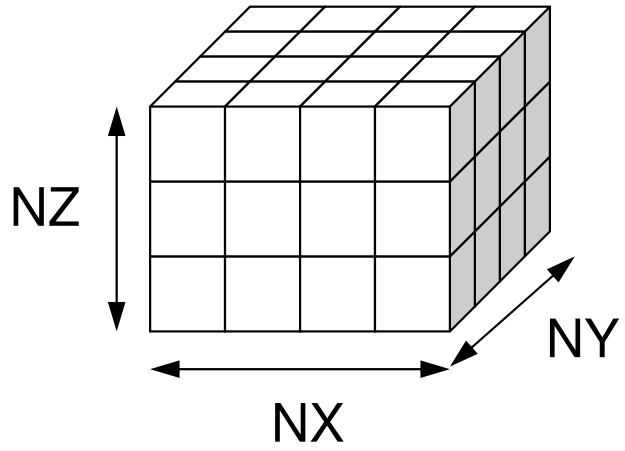


Running the Program

Control Data: <$P-L2>/solver/run/INPUT.DAT

```
20 20 20                                NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00              DX/DY/DZ
1.0e-08                                 EPSICCG
```

- NX, NY, NZ
  - 各方向のメッシュ数
- DX, DY, DZ
  - 各要素のX,Y,Z方向辺長さ
- EPSICCG
  - ICCG法の収束判定値



プログラムの実行
<$P-L2>/solver/run/

```
$ cd <$P-L2>/solver/run
$ ./L2-sol

You have    8000 elements.
How many colors do you need ?
  #COLOR must be more than 2 and
  #COLOR must not be more than    8000
   CM if #COLOR .eq. 0
  RCM if #COLOR .eq.-1
CMRCM if #COLOR .le.-2

=> XXX

$ ls test.inp
```

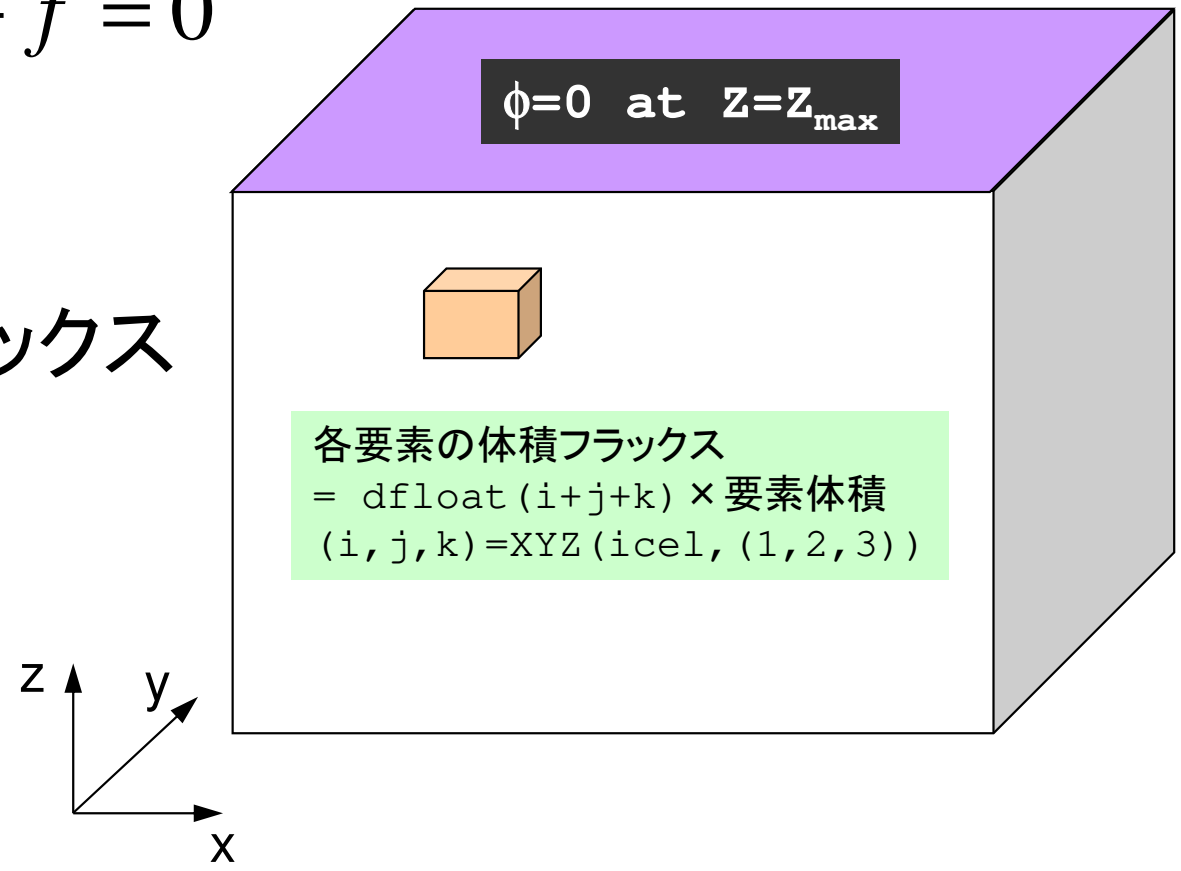
解いている問題：三次元ポアソン方程式

ポアソン方程式

$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}+f=0$$

<u>境界条件等</u>

- 各要素で体積フラックス
- Z=Z_{max}面でϕ=0

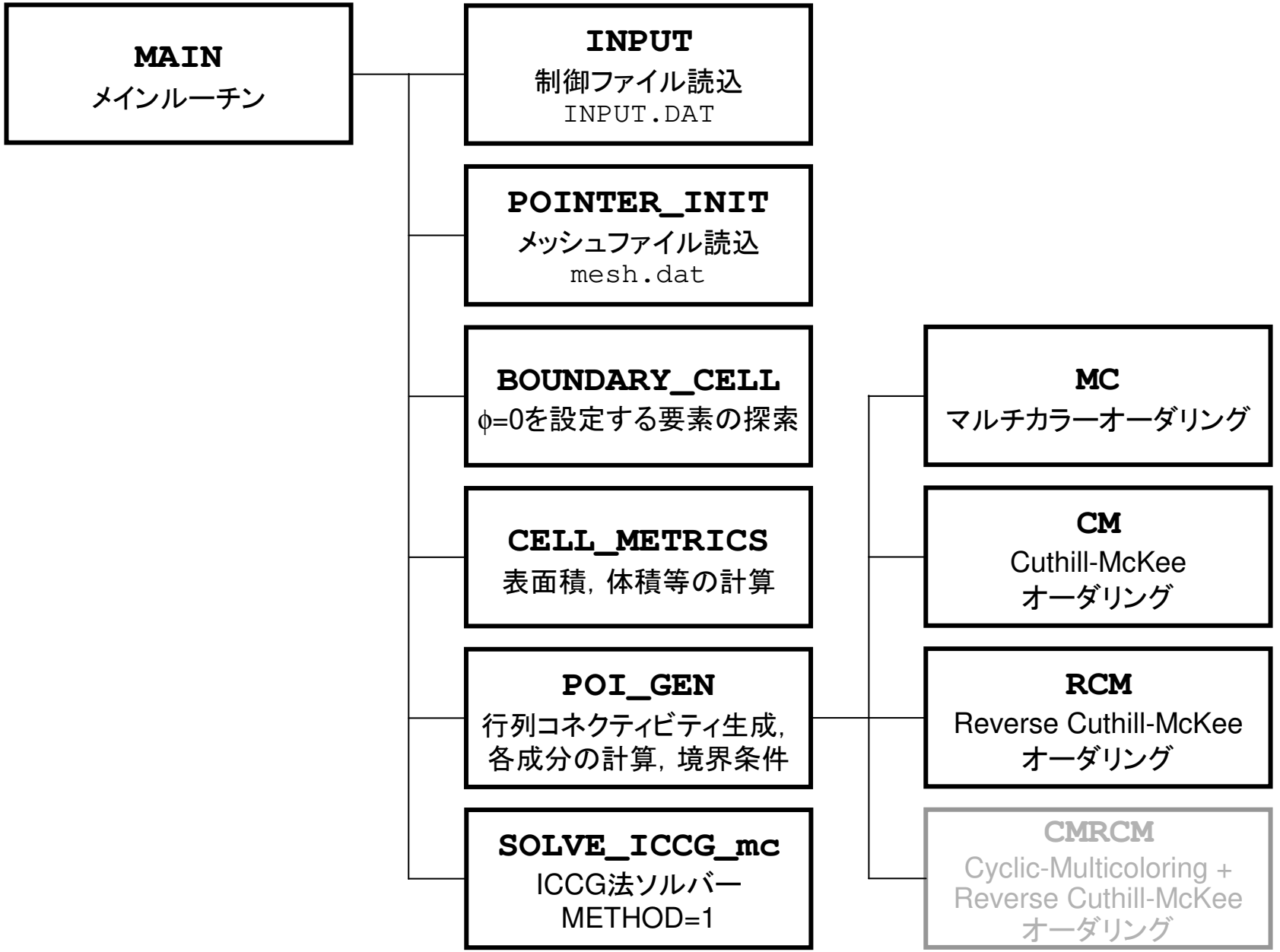


Main Program

```
#include <stdio.h> ...

int
main()
{
        double *WK;
        int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
        double xN, xL, xU; Stime, Etime;

        if(INPUT()) goto error;
        if(POINTER_INIT()) goto error;
        if(BOUNDARY_CELL()) goto error;
        if(CELL_METRICS()) goto error;
        if(POI_GEN()) goto error;

        ISET = 0;
        WK = (double *)malloc(sizeof(double)*ICELTOT);
        if(WK == NULL) {
           fprintf(stderr, "Error: %s¥n", strerror(errno));
           goto error;}
        if(solve_ICCG_mc(ICELTOT, NL, NU, indexL, itemL, indexU, itemU,
                         D, BFORCE, PHI, AL, AU, NCOLORtot, COLORindex,
                         EPSICCG, &ITR, &IER)) goto error;

        for(ic0=0; ic0<ICELTOT; ic0++) {
            icel = NEWtoOLD[ic0];
            WK[icel-1] = PHI[ic0];
        }
        for(icel=0; icel<ICELTOT; icel++) {
            PHI[icel] = WK[icel];
        }
        if(OUTUCD()) goto error;
        return 0;
error:
        return -1;
}
```

Renumbering of “PHI” to original numbering

プログラムの構成図


MAIN
メインルーチン
INPUT
制御ファイル読込
INPUT.DAT
POINTER_INIT
メッシュファイル読込
mesh.dat
BOUNDARY_CELL
ϕ=0を設定する要素の探索
CELL_METRICS
表面積，体積等の計算
POI_GEN
行列コネクティビティ生成，
各成分の計算，境界条件
SOLVE_ICCG_mc
ICCG法ソルバー
METHOD=1
MC
マルチカラーオーダリング
CM
Cuthill-McKee
オーダリング
RCM
Reverse Cuthill-McKee
オーダリング
CMRCM
Cyclic-Multicoloring +
Reverse Cuthill-McKee
オーダリング

変数表(1/2)

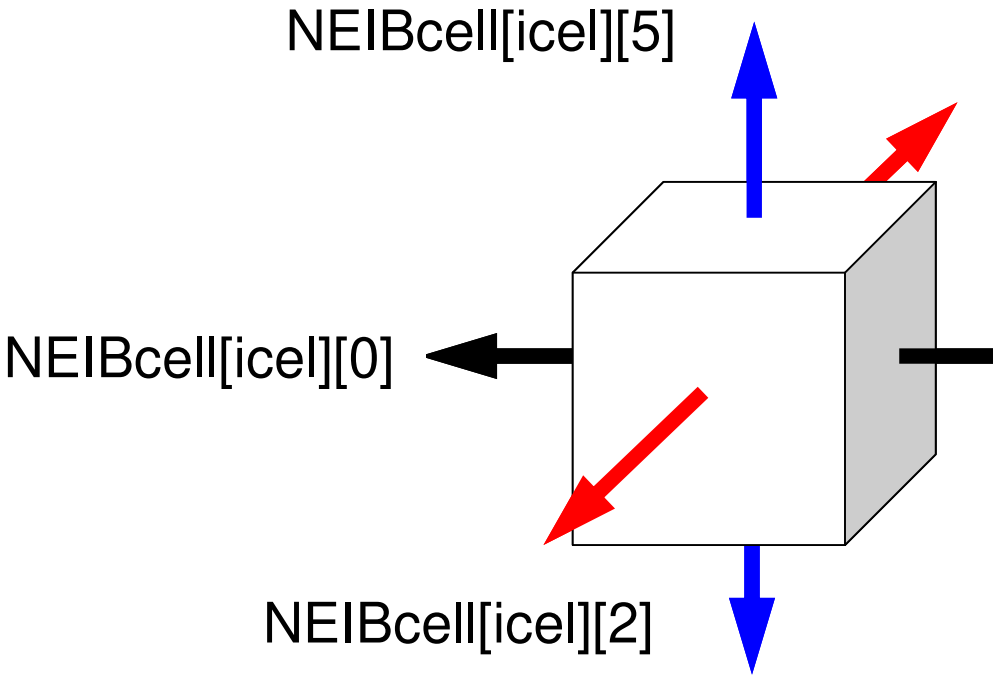
配列・変数名	型	内　容
D[N]	**R**	対角成分，(N:全メッシュ数=ICELTOT)
BFORCE[N]	**R**	右辺ベクトル
PHI[N]	**R**	未知数ベクトル
indexL[N+1] **indexU[N+1]**	**I**	各行の非零上下三角成分数(CRS)
NPL, NPU	**I**	非零上下三角成分総数(CRS)
itemL[NPL] **itemU[NPU]**	**I**	非零上下三角成分(列番号)(CRS)
AL[NPL] **AU[NPU]**	**R**	非零上下三角成分(係数)(CRS)

NL,NU	**I**	各行の非零上下三角成分の最大数（ここでは6）
INL[N] **INU[N]**	**I**	各行の非ゼロ上下三角成分数
IAL[N][NL] **IAU[N][NU]**	**I**	各行の非ゼロ上下三角成分数に対応する列番号

変数表(2/2)

配列・変数名	型	内　容
NCOLORtot	**I**	入力時にはOrdering手法（≧2：MC, =0：CM, =-1：RCM,-2≧：CMRCM），最終的には色数，レベル数が入る
COLORindex [NCOLORtot+1]	**I**	各色，レベルに含まれる要素数の一次元圧縮配列，COLORindex[icol]からCOLORindex[icol+1]-1までの要素がicol番目の色（レベル）に含まれる。
NEWtoOLD[N]	**I**	新番号⇒旧番号への参照配列
OLDtoNEW[N]	**I**	旧番号⇒新番号への参照配列

NEIBcell：隣接している要素番号
境界面の場合は0



i= XYZ[icel][0]
j= XYZ[icel][1], k= XYZ[icel][2]
icel= k*NX*NY + j*NX + i

NEIBcell[icel][0]= icel – 1 + 1
NEIBcell[icel][1]= icel + 1 + 1
NEIBcell[icel][2]= icel – NX + 1
NEIBcell[icel][3]= icel + NX + 1
NEIBcell[icel][4]= icel – NX*NY + 1
NEIBcell[icel][5]= icel + NX*NY + 1

プログラムの構成

```
#include <stdio.h> ...

int
main()
{
        double *WK;
        int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
        double xN, xL, xU; Stime, Etime;

        if(INPUT()) goto error;
        if(POINTER_INIT()) goto error;
        if(BOUNDARY_CELL()) goto error;
        if(CELL_METRICS()) goto error;
        if(POI_GEN()) goto error;

        ISET = 0;
        WK = (double *)malloc(sizeof(double)*ICELTOT);
        if(WK == NULL) {
           fprintf(stderr, "Error: %s¥n", strerror(errno));
           goto error;}
        if(solve_ICCG_mc(ICELTOT, NL, NU, indexL, itemL, indexU, itemU,
                         D, BFORCE, PHI, AL, AU, NCOLORtot, COLORindex,
                         EPSICCG, &ITR, &IER)) goto error;

        for(ic0=0; ic0<ICELTOT; ic0++) {
            icel = NEWtoOLD[ic0];
            WK[icel-1] = PHI[ic0];
        }
        for(icel=0; icel<ICELTOT; icel++) {
            PHI[icel] = WK[icel];
        }
        if(OUTUCD()) goto error;
        return 0;
error:
        return -1;
}
```

poi_gen (1/8)

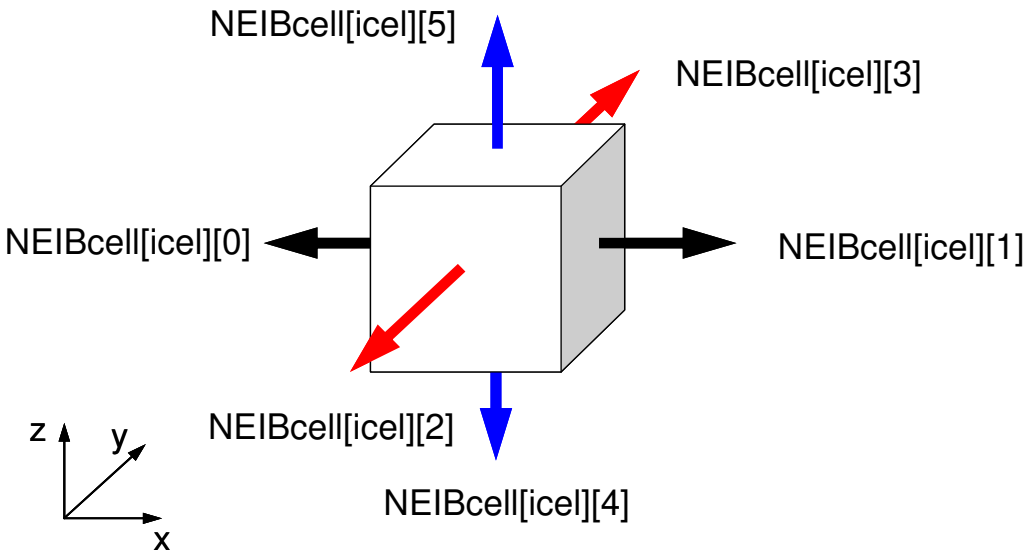
```
#include "allocate.h"
extern int
POI_GEN(void)
{ int nn;
  int icO, icN1, icN2, icN3, icN4, icN5, icN6;
  int i, j, k, ib, ic, ip, icel, icou, icol, icouG;
  int ii, jj, kk, nn1, num, nr, j0, j1;
  double coef, VOLO, S1t, E1t;
  int isL, ieL, isU, ieU;
  NL=6; NU= 6;
  IAL = (int **)allocate_matrix(sizeof(int),ICELTOT,NL);
  IAU = (int **)allocate_matrix(sizeof(int),ICELTOT,NU);
  BFORCE = (double *)allocate_vector(sizeof(double),ICELTOT);
  D      = (double *)allocate_vector(sizeof(double),ICELTOT);
  PHI    = (double *)allocate_vector(sizeof(double),ICELTOT);
  INL = (int *)allocate_vector(sizeof(int),ICELTOT);
  INU = (int *)allocate_vector(sizeof(int),ICELTOT);

  for (i = 0; i <ICELTOT ; i++) {
       BFORCE[i]=0.0;
       D[i]  =0.0; PHI[i]=0.0;
       INL[i] = 0; INU[i] = 0;
       for(j=0;j<6;j++){
           IAL[i][j]=0; IAU[i][j]=0;
       }
  }
  for (i = 0; i <=ICELTOT ; i++) {
       indexL[i] = 0; indexU[i] = 0;
  }
```

allocate.c

```
/*************************
  allocate matrix
*************************/
void** allocate_matrix(int size,int m,int n)
{
  void **aa;
  int i;
  if ( ( aa=(void ** )malloc( m * sizeof(void*) ) ) == NULL ) {
     fprintf(stdout,"Error:Memory does not enough! aa in matrix ¥n");
     exit(1);
  }
  if ( ( aa[0]=(void * )malloc( m * n * size ) ) == NULL ) {
     fprintf(stdout,"Error:Memory does not enough! in matrix ¥n");
     exit(1);
  }
        for(i=1;i<m;i++) aa[i]=(char*)aa[i-1]+size*n;
  return aa;
}
```

```
for(icel=0; icel<ICELTOT; icel++) {

      icN1 = NEIBcell[icel][0];
      icN2 = NEIBcell[icel][1];
      icN3 = NEIBcell[icel][2];
      icN4 = NEIBcell[icel][3];
      icN5 = NEIBcell[icel][4];
      icN6 = NEIBcell[icel][5];

      if(icN5 != 0) {
        icou = INL[icel] + 1;
        IAL[icel][icou-1] = icN5;
        INL[icel]         = icou;
      }

      if(icN3 != 0) {
        icou = INL[icel] + 1;
        IAL[icel][icou-1] = icN3;
        INL[icel]         = icou;
      }

      if(icN1 != 0) {
        icou = INL[icel] + 1;
        IAL[icel][icou-1] = icN3;
        INL[icel]         = icou;
      }
```

poi_gen (2/8)



下三角成分

NEIBcell[icel][4]= icel – NX*NY + 1
NEIBcell[icel][2]= icel – NX + 1
NEIBcell[icel][0]= icel – 1 + 1

“icel” starts at 0

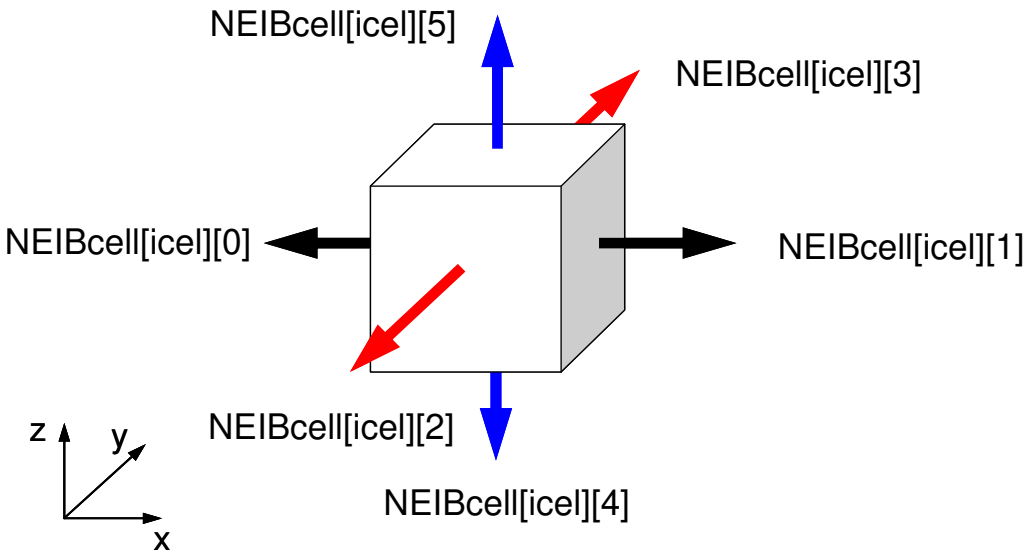
12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

“IAL” starts at 1

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

poi_gen (3/8)

```
 for(icel=0; icel<ICELTOT; icel++) {

      icN1 = NEIBcell[icel][0];
      icN2 = NEIBcell[icel][1];
      icN3 = NEIBcell[icel][2];
      icN4 = NEIBcell[icel][3];
      icN5 = NEIBcell[icel][4];
      icN6 = NEIBcell[icel][5];
....
      if(icN2 != 0) {
        icou = INU[icel] + 1;
        IAU[icel][icou-1] = icN2;
        INU[icel]         = icou;
      }

      if(icN4 != 0) {
        icou = INU[icel] + 1;
        IAU[icel][icou-1] = icN4;
        INU[icel]         = icou;
      }

      if(icN6 != 0) {
        icou = INU[icel] + 1;
        IAU[icel][icou-1] = icN6;
        INU[icel]         = icou;
      }
}
```



上三角成分

NEIBcell[icel][1]= icel + 1 + 1
NEIBcell[icel][3]= icel + NX + 1
NEIBcell[icel][5]= icel + NX*NY + 1

“icel” starts at 0

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

“IAU” starts at 1

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

poi_gen (4/8)

```
N111:
        fprintf(stderr, "¥n¥nYou have%8d elements¥n", ICELTOT);
        fprintf(stderr, "How many colors do you need ?¥n");
        fprintf(stderr, "  #COLOR must be more than 2 and¥n");
        fprintf(stderr, "  #COLOR must not be more than%8d¥n", ICELTOT);
        fprintf(stderr, "  if #COLOR= 0 then CM  ordering¥n");
        fprintf(stderr, "  if #COLOR=-1 then RCM ordering¥n");
        fprintf(stderr, "  if #COLOR<-1 then CMRCM ordering¥n");
        fprintf(stderr, "=>¥n");
        fscanf(stdin, "%d", &NCOLORtot);
        if(NCOLORtot == 1 && NCOLORtot > ICELTOT) goto N111;

        OLDtoNEW = (int *)calloc(ICELTOT, sizeof(int));
        if(OLDtoNEW == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
        NEWtoOLD = (int *)calloc(ICELTOT, sizeof(int));
        if(NEWtoOLD == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }
        COLORindex = (int *)calloc(ICELTOT+1, sizeof(int));
        if(COLORindex == NULL) {
                fprintf(stderr, "Error: %s¥n", strerror(errno));
                return -1;
        }

        if(NCOLORtot > 0) {
                MC(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot == 0) {
                CM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot ==-1) {
                RCM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        } else if(NCOLORtot <-1) {
                CMRCM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
                                &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
        }

        fprintf(stderr, "¥n# TOTAL COLOR number%8d¥n", NCOLORtot);
        return 0;
}
```

poi_gen (5/8)

これ以降は新しい番号付けを使用

```
indexL =
   (int *)allocate_vector(sizeof(int),ICELTOT+1);
indexU =
   (int *)allocate_vector(sizeof(int),ICELTOT+1);

for(i=0; i<ICELTOT; i++){
    indexL[i+1]=indexL[i]+INL[i];
    indexU[i+1]=indexU[i]+INU[i];
}
NPL = indexL[ICELTOT];
NPU = indexU[ICELTOT];

itemL = (int *)allocate_vector(sizeof(int),NPL);
itemU = (int *)allocate_vector(sizeof(int),NPU);
AL = (double *)allocate_vector(sizeof(double),NPL);
AU = (double *)allocate_vector(sizeof(double),NPU);

memset(itemL, 0, sizeof(int)*NPL);
memset(itemU, 0, sizeof(int)*NPU);
memset(AL, 0.0, sizeof(double)*NPL);
memset(AU, 0.0, sizeof(double)*NPU);

for(i=0; i<ICELTOT; i++){
    for(k=0;k<INL[i];k++){
        kk= k + indexL[i];
        itemL[kk]= IAL[i][k];
    }
    for(k=0;k<INU[i];k++){
        kk= k + indexU[i];
        itemU[kk]= IAU[i][k];
    }
}

free(INL); free(INU);
free(IAL); free(IAU);
```

配列・変数名	型	内　容
D[N]	R	対角成分，(N:全メッシュ数=ICELTOT)
BFORCE[N]	R	右辺ベクトル
PHI[N]	R	未知数ベクトル
indexL[N+1] indexU[N+1]	I	各行の非零上下三角成分数(CRS)
NPL, NPU	I	非零上下三角成分総数(CRS)
itemL[NPL] itemU[NPU]	I	非零上下三角成分(列番号)(CRS)
AL[NPL] AU[NPU]	R	非零上下三角成分(係数)(CRS)

“itemL” /”itemU” start at 1

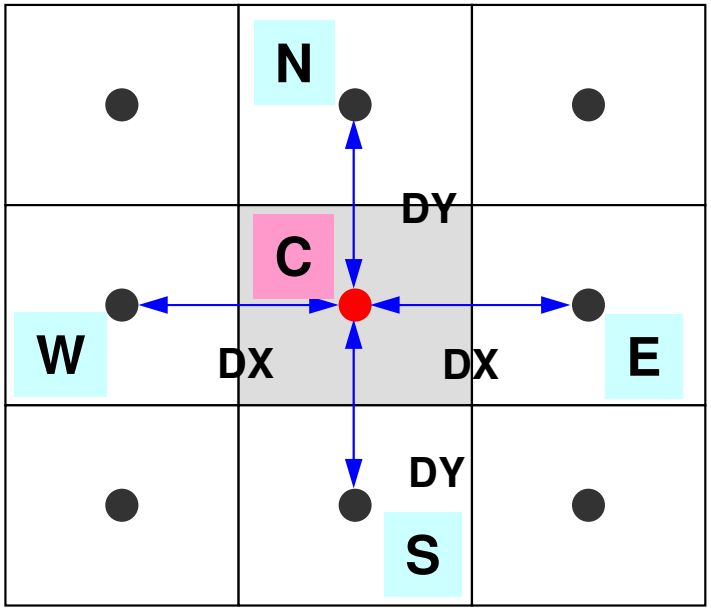
13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

```
for (i=0; i<N; i++) {
    q[i]= D[i] * p[i];
    for (j=indexL[i]; j<indexL[i+1]; j++) {
        q[i] += AL[j] * p[itemL[j]-1];
    }
    for (j=indexU[i]; j<indexU[i+1]; j++) {
        q[i] += AU[j] * p[itemU[j]-1];
    }
}
```

```
for(icel=0;  icel<N; icel++) {
    ic0  = NEWtoOLD[icel];
    icN1 = NEIBcell[ic0-1][0];
    icN2 = NEIBcell[ic0-1][1];
    icN3 = NEIBcell[ic0-1][2];
    icN4 = NEIBcell[ic0-1][3];
    icN5 = NEIBcell[ic0-1][4];
    icN6 = NEIBcell[ic0-1][5];
    VOL0 = VOLCEL[ic0];

    isL =indexL[icel  ];    ieL =indexL[icel+1];
    isU =indexU[icel  ];    ieU =indexU[icel+1];

    if(icN5 != 0) {
       icN5 = OLDtoNEW[icN5-1];
       coef = RDZ * ZAREA;
       D[icel] -= coef;

       if(icN5-1 < icel) {
          for(j=isL; j<ieL; j++) {
              if(itemL[j] == icN5) {
                 AL[j] = coef;
                 break;
              }
          }
       } else {
          for(j=isU; j<ieU; j++) {
              if(itemU[j] == icN5) {
                 AU[j] = coef;
                 break;
              }
           }
        }
      }
...
```

poi_gen (6/8)

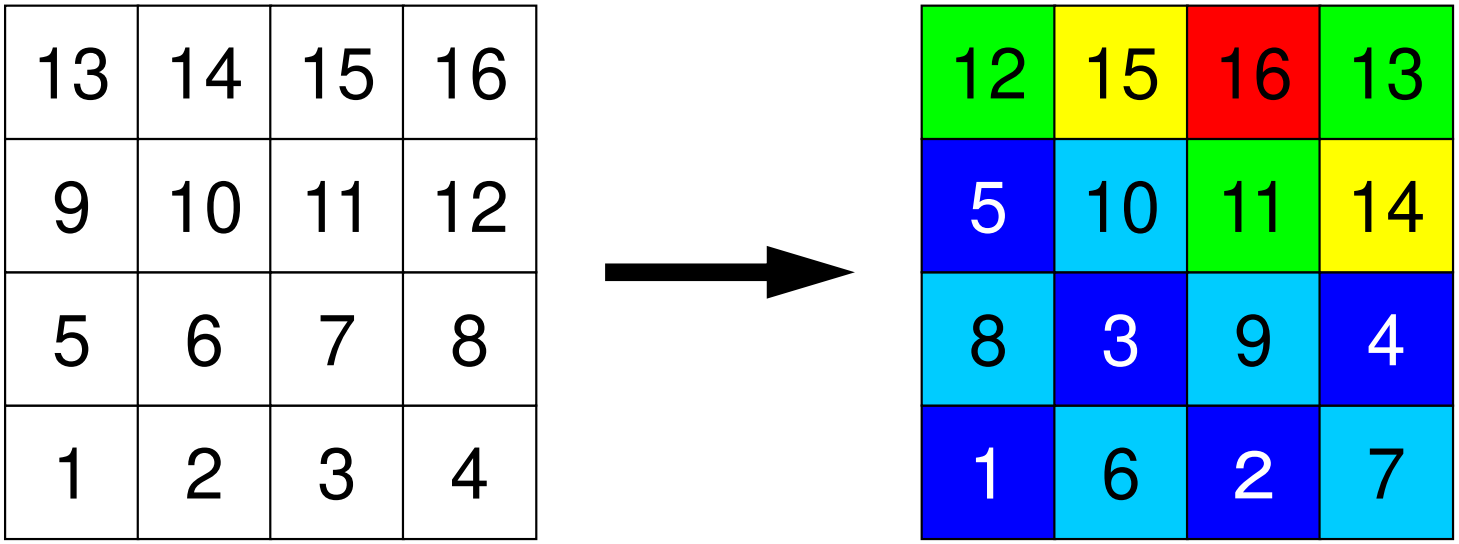
係数の計算(境界面以外)



$$\frac{\phi_E-\phi_i}{\Delta x}\Delta y+\frac{\phi_W-\phi_i}{\Delta x}\Delta y+$$

$$\frac{\phi_N-\phi_i}{\Delta y}\Delta x+\frac{\phi_S-\phi_i}{\Delta y}\Delta x=f_c\Delta x\Delta y$$

「新しい番号付け」とは?



- RCMまたはマルチカラーによるオーダリングを施す。
- 色番号順に要素番号が並んでいる。上記の例では:
  - 第1色(濃い青):1,2,3,4,5(旧:1,3,6,8,9)
  - 第2色(薄い青):6,7,8,9,10(旧:2,4,5,7,10)
  - 第3色(黄緑) :11,12,13(旧:11,13,16)
  - 第4色(黄) :14,15(旧:12,14), 第5色(赤):16(旧:15)

「新しい番号付け」とは?(続き)

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

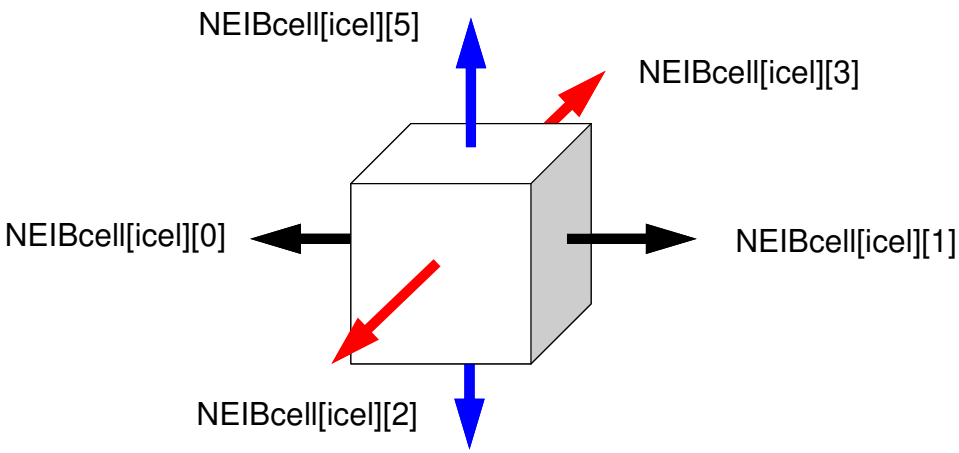
12	15	16	13
5	10	11	14
8	3	9	4
1	6	2	7

```
NCOLORtot= 5
COLORindex[0]=  0, COLORindex[1]=  5, COLORindex[2]= 10
COLORindex[3]= 13, COLORindex[5]= 15, COLORindex[6]= 16
```

- NEWtoOLD, OLDtoNEW
  - **OLDtoNEW[6-1]=3, NEWtoOLD[3-1]=6**

poi_gen (6/8)

```
for(icel=0;  icel<N; icel++) {
    ic0  = NEWtoOLD[icel];
    icN1 = NEIBcell[ic0-1][0];
    icN2 = NEIBcell[ic0-1][1];
    icN3 = NEIBcell[ic0-1][2];
    icN4 = NEIBcell[ic0-1][3];
    icN5 = NEIBcell[ic0-1][4];
    icN6 = NEIBcell[ic0-1][5];
    VOL0 = VOLCEL[ic0];

    isL =indexL[icel  ];    ieL =indexL[icel+1];
    isU =indexU[icel  ];    ieU =indexU[icel+1];

    if(icN5 != 0) {
       icN5 = OLDtoNEW[icN5-1];
       coef = RDZ * ZAREA;
       D[icel] -= coef;

       if(icN5-1 < icel) {
          for(j=isL; j<ieL; j++) {
              if(itemL[j] == icN5) {
                 AL[j] = coef;
                 break;
              }
          }
       } else {
          for(j=isU; j<ieU; j++) {
              if(itemU[j] == icN5) {
                 AU[j] = coef;
                 break;
              }
           }
        }
      }
...
```



$$\frac{\phi_{neib[icel][0]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib[icel][1]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib[icel][2]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib[icel][3]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

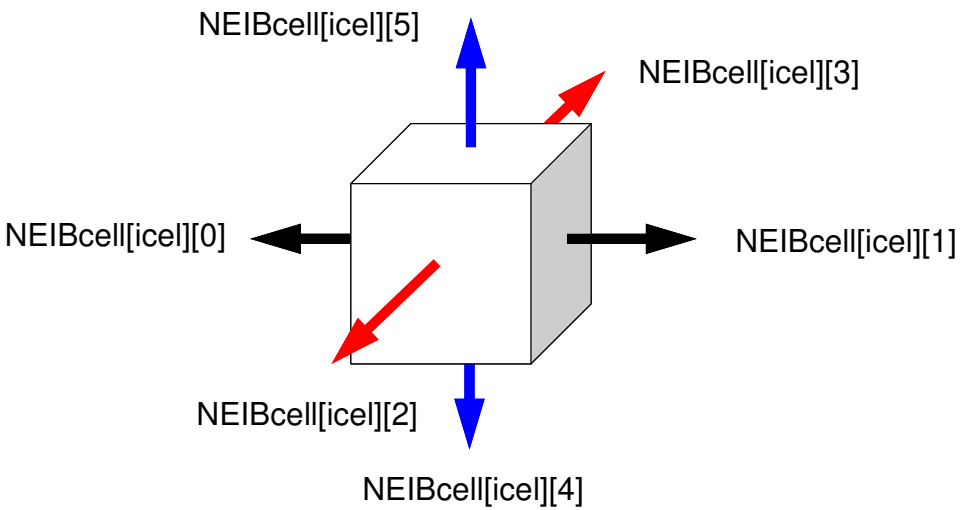
$$\frac{\phi_{neib[icel][4]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+$$

$$\frac{\phi_{neib[icel][5]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

poi_gen (6/8)

```
for(icel=0;  icel<N; icel++) {
    ic0  = NEWtoOLD[icel];
    icN1 = NEIBcell[ic0-1][0];
    icN2 = NEIBcell[ic0-1][1];
    icN3 = NEIBcell[ic0-1][2];
    icN4 = NEIBcell[ic0-1][3];
    icN5 = NEIBcell[ic0-1][4];
    icN6 = NEIBcell[ic0-1][5];
    VOL0 = VOLCEL[ic0];

    isL =indexL[icel  ];    ieL =indexL[icel+1];
    isU =indexU[icel  ];    ieU =indexU[icel+1];

    if(icN5 != 0) {
       icN5 = OLDtoNEW[icN5-1];
       coef = RDZ * ZAREA;
       D[icel] -= coef;

       if(icN5-1 < icel) {
          for(j=isL; j<ieL; j++) {
              if(itemL[j] == icN5) {
                 AL[j] = coef;
                 break;
              }
          }
       } else {
          for(j=isU; j<ieU; j++) {
              if(itemU[j] == icN5) {
                 AU[j] = coef;
                 break;
              }
           }
        }
      }
...
```

icN5がicelより小さければ下三角成分



$$\frac{\phi_{neib[icel][0]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib[icel][1]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib[icel][2]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib[icel][3]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

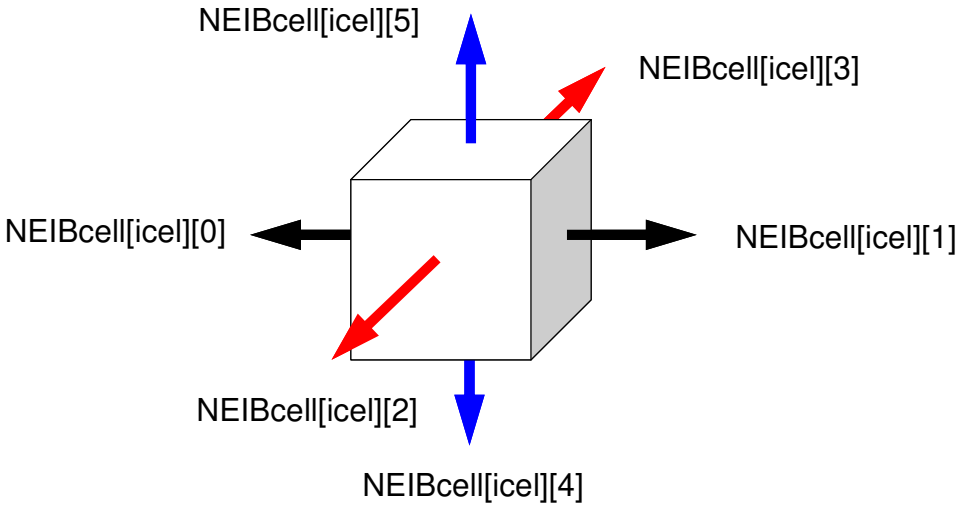
$$\frac{\phi_{neib[icel][4]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+$$

$$\frac{\phi_{neib[icel][5]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

poi_gen (6/8)

```
for(icel=0;  icel<N; icel++) {
    ic0  = NEWtoOLD[icel];
    icN1 = NEIBcell[ic0-1][0];
    icN2 = NEIBcell[ic0-1][1];
    icN3 = NEIBcell[ic0-1][2];
    icN4 = NEIBcell[ic0-1][3];
    icN5 = NEIBcell[ic0-1][4];
    icN6 = NEIBcell[ic0-1][5];
    VOL0 = VOLCEL[ic0];

    isL =indexL[icel  ];    ieL =indexL[icel+1];
    isU =indexU[icel  ];    ieU =indexU[icel+1];

    if(icN5 != 0) {
       icN5 = OLDtoNEW[icN5-1];
       coef = RDZ * ZAREA;
       D[icel] -= coef;

       if(icN5-1 < icel) {
          for(j=isL; j<ieL; j++) {
              if(itemL[j] == icN5) {
                 AL[j] = coef;
                 break;
              }
          }
       } else {
          for(j=isU; j<ieU; j++) {
              if(itemU[j] == icN5) {
                 AU[j] = coef;
                 break;
              }
          }
       }
    }
...
```

icN5がicelより大きければ上三角成分



$$\frac{\phi_{neib[icel][0]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib[icel][1]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib[icel][2]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib[icel][3]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib[icel][4]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+$$

$$\frac{\phi_{neib[icel][5]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

```
    if(icN6 != 0) {
       icN6 = OLDtoNEW[icN6-1];
       coef = RDZ * ZAREA;
       D[icel] -= coef;

       if(icN6-1 < icel) {
          for(j=isL; j<ieL; j++) {
              if(itemL[j] == icN6) {
                 AL[j] = coef;
                 break;
              }
          }
       } else {
          for(j=isU; j<ieU; j++) {
              if(itemU[j] == icN6) {
                 AU[j] = coef;
                 break;
              }
           }
        }
      }

      ii = XYZ[ic0-1][0];
      jj = XYZ[ic0-1][1];
      kk = XYZ[ic0-1][2];

      BFORCE[icel]= -(double)(ii+jj+kk) * VOL0;

}
```

もとの座標に従って
BFORCE計算

poi_gen (7/8)

$$\frac{\phi_{neib[icel][0]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib[icel][1]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib[icel][2]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib[icel][3]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

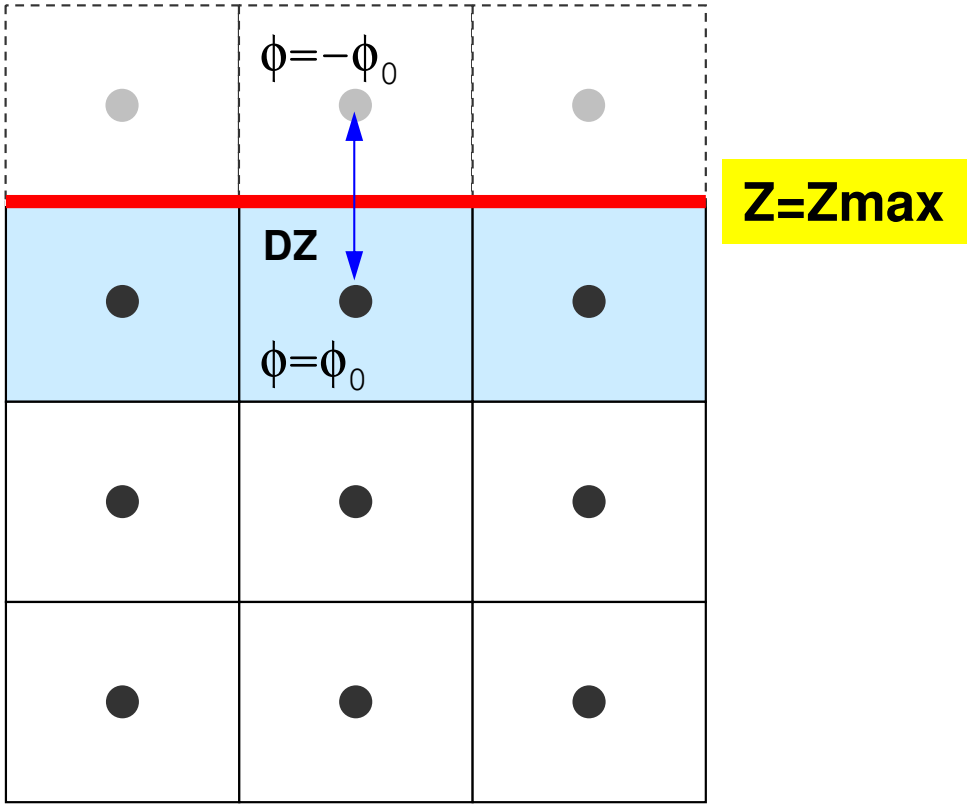
$$\frac{\phi_{neib[icel][4]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+$$

$$\frac{\phi_{neib[icel][5]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

```
        for(ib=0; ib<ZmaxCELtot; ib++) {
            ic0  = ZmaxCEL[ib] - 1;
            coef = 2.0 * RDZ * ZAREA;
            icel = OLDtoNEW[ic0];
            D[icel-1] -= coef;
        }


        return 0;
}
```

poi_gen (8/8)

係数の計算(境界面以外)
係数の計算(境界面)



境界面の外側に，大きさが同じで，値が$\phi=-\phi_0$となるような要素があると仮定(境界面で丁度$\phi=0$となる)。

プログラムの構成

```
#include <stdio.h> ...

int
main()
{
        double *WK;
        int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
        double xN, xL, xU; Stime, Etime;

        if(INPUT()) goto error;
        if(POINTER_INIT()) goto error;
        if(BOUNDARY_CELL()) goto error;
        if(CELL_METRICS()) goto error;
        if(POI_GEN()) goto error;

        ISET = 0;
        WK = (double *)malloc(sizeof(double)*ICELTOT);
        if(WK == NULL) {
           fprintf(stderr, "Error: %s¥n", strerror(errno));
           goto error;}
        if(solve_ICCG_mc(ICELTOT, NL, NU, indexL, itemL, indexU, itemU,
                         D, BFORCE, PHI, AL, AU, NCOLORtot, COLORindex,
                         EPSICCG, &ITR, &IER)) goto error;

        for(ic0=0; ic0<ICELTOT; ic0++) {
            icel = NEWtoOLD[ic0];
            WK[icel-1] = PHI[ic0];
        }
        for(icel=0; icel<ICELTOT; icel++) {
            PHI[icel] = WK[icel];
        }
        if(OUTUCD()) goto error;
        return 0;
error:
        return -1;
}
```

この時点で，係数，右辺ベクトルともに，「新しい」番号にしたがって計算，記憶されている。

solve_ICCG_mc (1/7)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <math.h> etc.

#include "solver_ICCG.h"

extern int
solve_ICCG_mc(int N, int NL, int NU, int *indexL, int *itemL, int *indexU,
              int *itemU,
              double *D, double *B, double *X, double *AL, double *AU,
              int NCOLORtot, int *COLORindex, double EPS, int *ITR, int *IER)
{

        double **W;
        double VAL, BNRM2, WVAL, SW, RHO, BETA, RHO1, C1, DNRM2, ALPHA, ERR;
        int i, j, ic, ip, L, ip1;
        int R = 0;
        int Z = 1;
        int Q = 1;
        int P = 2;
        int DD = 3;
```

solve_ICCG_mc (2/7)

```
W = (double **)malloc(sizeof(double *)*4);
  if(W == NULL) {
  fprintf(stderr, "Error: %s¥n", strerror(errno));
  return -1;
  }

  for(i=0; i<4; i++) {
      W[i] = (double *)malloc(sizeof(double)*N);
      if(W[i] == NULL) {
         fprintf(stderr, "Error: %s¥n",
         strerror(errno)); return -1;
      }
  }

  for(i=0; i<N; i++) {
      X[i] = 0.0;
      W[1][i] = 0.0;
      W[2][i] = 0.0;
      W[3][i] = 0.0;
  }

  for(ic=0; ic<NCOLORtot; ic++) {
      for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++){
          VAL = D[i];
          for(j=indexL[i]; j<indexL[i+1]; j++) {
              VAL-= AL[j]*AL[j]*W[DD][itemL[j] - 1];
          }
          W[DD][i] = 1.0 / VAL;
       }
  }
```

不完全修正
コレスキー分解

Compute $r^{(0)}= b-[A]x^{(0)}$

for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence $|r|$
end

solve_ICCG_mc (2/7)

```
W = (double **)malloc(sizeof(double *)*4);
  if(W == NULL) {
  fprintf(stderr, "Error: %s¥n", strerror(errno));
  return -1;
  }

  for(i=0; i<4; i++) {
      W[i] = (double *)malloc(sizeof(double)*N);
      if(W[i] == NULL) {
         fprintf(stderr, "Error: %s¥n",
         strerror(errno)); return -1;
      }
  }

  for(i=0; i<N; i++) {
      X[i] = 0.0;
      W[1][i] = 0.0;
      W[2][i] = 0.0;
      W[3][i] = 0.0;
  }


  for(ic=0; ic<NCOLORtot; ic++) {
      for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++){
          VAL = D[i];
          for(j=indexL[i]; j<indexL[i+1]; j++) {
              VAL-= AL[j]*AL[j]*W[DD][itemL[j] - 1];
          }
          W[DD][i] = 1.0 / VAL;
       }
  }
```

不完全修正
コレスキー分解

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^{\ 2} \cdot d_k \right)^{-1} = l_{ii}^{\ -1}$$

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} a_{ik}^{\ 2} \cdot d_k \right)^{-1} = l_{ii}^{\ -1}$$

W[DD][i]:	d_i
D[i]:	a_{ii}
itemL[j]:	k
AL[j]:	a_{ik}

不完全修正コレスキー分解

```
for(i=0; i<N; i++) {
    VAL = D[i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        VAL-= AL[j]*AL[j]*W[DD][itemL[j] - 1];
    }
    W[DD][i] = 1.0 / VAL;
}
```

```
for(ic=0; ic<NCOLORtot; ic++) {
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++){
        VAL = D[i];
          for(j=indexL[i]; j<indexL[i+1]; j++) {
              VAL-= AL[j]*AL[j]*W[DD][itemL[j] - 1];
          }
          W[DD][i] = 1.0 / VAL;
     }
}
```

右辺に現れる要素「itemL[k]」は
要素「i」とは異なる「色」に
属している　⇒　データ依存性排除

同じ「色」に属する要素はお互いに
依存性を持たない（接続しない）

solve_ICCG_mc (3/7)

```
for(i=0; i<N; i++) {
    VAL = D[i] * X[i];

    for(j=indexL[i]; j<indexL[i+1]; j++) {
        VAL += AL[j] * X[itemL[j]-1];
    }

    for(j=indexU[i]; j<indexU[i+1]; j++) {
        VAL += AU[j] * X[itemU[j]-1];
    }
    W[R][i] = B[i] - VAL;
}

BNRM2 = 0.0;
for(i=0; i<N; i++) {
    BNRM2 += B[i]*B[i];
}
```

Compute $r^{(0)}= b-[A]x^{(0)}$

for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence $|r|$
end

solve_ICCG_mc (4/7)

```
*ITR = N;
for(L=0; L<(*ITR); L++) {


for(i=0; i<N; i++) {
    W[Z][i] = W[R][i];
}

for(ic=0; ic<NCOLORtot; ic++){
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
}

for (ic=NCOLORtot-1; ic>=0; ic--){
     for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
         SW = 0.0;
         for(j=indexU[i]; j<indexU[i+1]; j++) {
             SW += AU[j] * W[Z][itemU[j]-1];
         }
         W[Z][i]= W[Z][i]  - W[DD][i] * SW;
     }
}
```

Compute $r^{(0)}= b-[A]x^{(0)}$

for i= 1, 2, …

  solve $[M]z^{(i-1)}= r^{(i-1)}$

  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$

  if i=1

    $p^{(1)}= z^{(0)}$

  else

    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$

    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$

  endif

  $q^{(i)}= [A]p^{(i)}$

  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$

  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$

  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$

  check convergence $|r|$

end

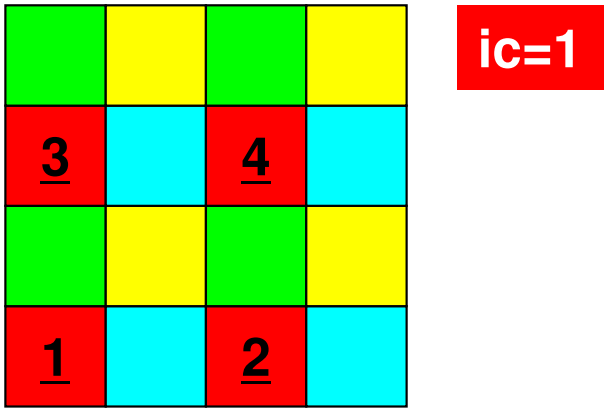
solve_ICCG_mc (4/7)

```
*ITR = N;
for(L=0; L<(*ITR); L++) {


for(i=0; i<N; i++) {
    W[Z][i] = W[R][i];
}

for(ic=0; ic<NCOLORtot; ic++){
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
}

for (ic=NCOLORtot-1; ic>=0; ic--){
     for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
         SW = 0.0;
         for(j=indexU[i]; j<indexU[i+1]; j++) {
             SW += AU[j] * W[Z][itemU[j]-1];
         }
         W[Z][i]= W[Z][i]  - W[DD][i] * SW;
     }
}
```

$$(M)\{z\}=\left(LDL^T\right)\{z\}=\{r\}$$

$$(L)\{z\}=\{r\}$$

前進代入
Forward Substitution



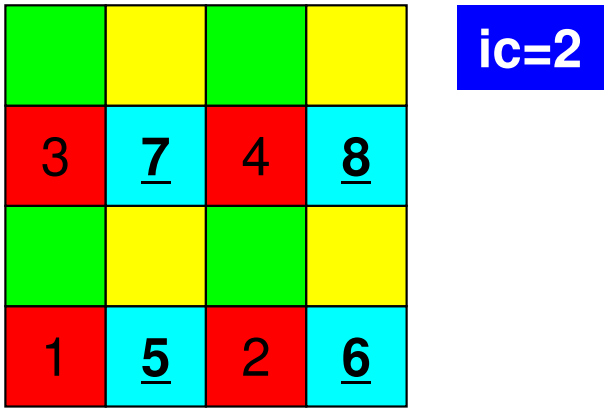
solve_ICCG_mc (4/7)

```
*ITR = N;
for(L=0; L<(*ITR); L++) {


for(i=0; i<N; i++) {
    W[Z][i] = W[R][i];
}

for(ic=0; ic<NCOLORtot; ic++){
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
}

for (ic=NCOLORtot-1; ic>=0; ic--){
     for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
         SW = 0.0;
         for(j=indexU[i]; j<indexU[i+1]; j++) {
             SW += AU[j] * W[Z][itemU[j]-1];
         }
         W[Z][i]= W[Z][i]  - W[DD][i] * SW;
     }
}
```

$$(M)\{z\}=\left(LDL^T\right)\{z\}=\{r\}$$

$$(L)\{z\}=\{r\}$$

前進代入
Forward Substitution



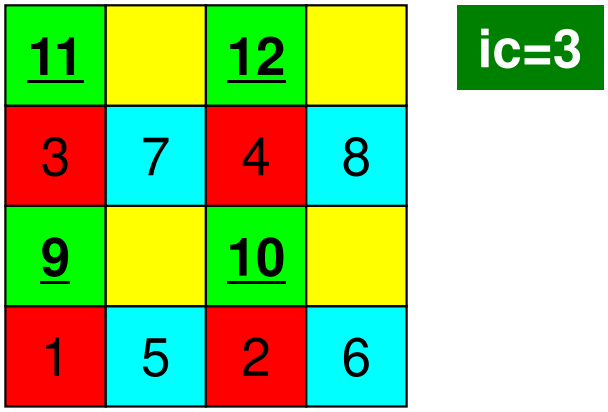
solve_ICCG_mc (4/7)

```
*ITR = N;
for(L=0; L<(*ITR); L++) {


for(i=0; i<N; i++) {
    W[Z][i] = W[R][i];
}

for(ic=0; ic<NCOLORtot; ic++){
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
}

for (ic=NCOLORtot-1; ic>=0; ic--){
     for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
         SW = 0.0;
         for(j=indexU[i]; j<indexU[i+1]; j++) {
             SW += AU[j] * W[Z][itemU[j]-1];
         }
         W[Z][i]= W[Z][i]  - W[DD][i] * SW;
     }
}
```

$$(M)\{z\}=\left(LDL^T\right)\{z\}=\{r\}$$

$$(L)\{z\}=\{r\}$$

前進代入
Forward Substitution



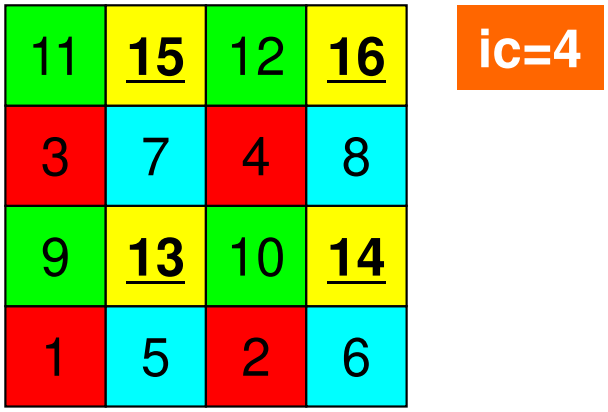
solve_ICCG_mc (4/7)

```
*ITR = N;
for(L=0; L<(*ITR); L++) {


for(i=0; i<N; i++) {
    W[Z][i] = W[R][i];
}

for(ic=0; ic<NCOLORtot; ic++){
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
}

for (ic=NCOLORtot-1; ic>=0; ic--){
     for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
         SW = 0.0;
         for(j=indexU[i]; j<indexU[i+1]; j++) {
             SW += AU[j] * W[Z][itemU[j]-1];
         }
         W[Z][i]= W[Z][i]  - W[DD][i] * SW;
     }
}
```

$$(M)\{z\}=\left(LDL^T\right)\{z\}=\{r\}$$

$$(L)\{z\}=\{r\}$$

前進代入
Forward Substitution



solve_ICCG_mc (4/7)

```
*ITR = N;
for(L=0; L<(*ITR); L++) {


for(i=0; i<N; i++) {
    W[Z][i] = W[R][i];
}

for(ic=0; ic<NCOLORtot; ic++){
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
}

for (ic=NCOLORtot-1; ic>=0; ic--){
     for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
         SW = 0.0;
         for(j=indexU[i]; j<indexU[i+1]; j++) {
             SW += AU[j] * W[Z][itemU[j]-1];
         }
         W[Z][i]= W[Z][i]  - W[DD][i] * SW;
     }
}
```

$$(M)\{z\}=(LDL^T)\{z\}=\{r\}$$

$$(L)\{z\}=\{r\}$$

$$(DL^T)\{z\}=\{z\}$$

前進代入
Forward Substitution

後退代入
Backward Substitution

solve_ICCG_mc (4/7)

```
*ITR = N;
for(L=0; L<(*ITR); L++) {


for(i=0; i<N; i++) {
    W[Z][i] = W[R][i];
}

for(ic=0; ic<NCOLORtot; ic++){
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
                                                ]; j++) {
                                              nL[j]-1];

for (ic=NCOLORtot-1; ic>=0; ic--){
     for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {
          SW = 0.0;
          for(j=indexU[i]; j<indexU[i+1]; j++) {
              SW += AU[j] * W[Z][itemU[j]-1];
          }
          W[Z][i]= W[Z][i]  - W[DD][i] * SW;
     }
}
```

$$(M)\{z\}=(LDL^T)\{z\}=\{r\}$$

おなじ色の中で計算順序を変えても
結果は同じ:

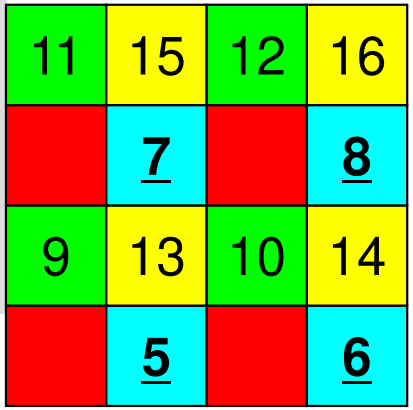
i= COLOR[ic], COLOR(ic+1)
i= COLOR[ic+1], COLOR[ic], -1

$$(L)\{z\}=\{r\}$$

$$(DL^T)\{z\}=\{z\}$$

前進代入
Forward Substitution

後退代入
Backward Substitution



ic=2

Forward/Backward Substitution
前進後退代入

```
for(i=0; i<N; i++) {
    WVAL = W[Z][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        WVAL -= AL[j] * W[Z][itemL[j]-1];
    }
    W[Z][i] = WVAL * W[DD][i];
}

for(i=N-1; i>=0; i--) {
    SW = 0.0;
    for(j=indexU[i]; j<indexU[i+1]; j++) {
        SW += AU[j] * W[Z][itemU[j]-1];
    }
    W[Z][i]= W[Z][i]  - W[DD][i] * SW;
}
```

```
for(ic=0; ic<NCOLORtot; ic++){
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++){
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }
}

for (ic=NCOLORtot-1; ic>=0; ic--){
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++){
        SW = 0.0;
        for(j=indexU[i]; j<indexU[i+1]; j++) {
            SW += AU[j] * W[Z][itemU[j]-1];
        }
        W[Z][i]= W[Z][i]  - W[DD][i] * SW;
    }
}
```

solve_ICCG_mc (5/7)

```
/****************
 * RHO = {r}{z} *
 ****************/

RHO = 0.0;
for(i=0; i<N; i++) {
    RHO += W[R][i] * W[Z][i];
}

/*********************************
 * {p}  = {z} if      ITER=0    *
 * BETA = RHO / RHO1  otherwise *
 *********************************/

if(L == 0) {
   for(i=0; i<N; i++) {
       W[P][i] = W[Z][i];
   }
   } else {
   BETA = RHO / RHO1;
   for(i=0; i<N; i++) {
       W[P][i] = W[Z][i] + BETA * W[P][i];
   }
}
```

Compute $r^{(0)}= b-[A]x^{(0)}$

for i= 1, 2, …

  solve $[M]z^{(i-1)}= r^{(i-1)}$

  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$

  if i=1

    $p^{(1)}= z^{(0)}$

  else

    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$

    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$

  endif

  $q^{(i)}= [A]p^{(i)}$

  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$

  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$

  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$

  check convergence $|r|$

end

solve_ICCG_mc (6/7)

```
/****************
 * {q} = [A]{p} *
 ****************/

for(i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        VAL += AL[j] * W[P][itemL[j]-1];
    }
    for(j=indexU[i]; j<indexU[i+1]; j++) {
        VAL += AU[j] * W[P][itemU[j]-1];
    }
    W[Q][i] = VAL;
}
```

Compute $r^{(0)}= b-[A]x^{(0)}$

for i= 1, 2, …

  solve $[M]z^{(i-1)}= r^{(i-1)}$

  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$

  if i=1

    $p^{(1)}= z^{(0)}$

  else

    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$

    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$

  endif

  $q^{(i)}= [A]p^{(i)}$

  $\alpha_i\ = \rho_{i-1}/p^{(i)}q^{(i)}$

  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$

  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$

  check convergence $|r|$

end

solve_ICCG_mc (7/7)

```
/*************************
 * ALPHA = RHO / {p}{q} *
 *************************/
C1 = 0.0;
for(i=0; i<N; i++) {
    C1 += W[P][i] * W[Q][i];
}
ALPHA = RHO / C1;

/*****************************
 * {x} = {x} + ALPHA * {p} *
 * {r} = {r} - ALPHA * {q} *
 *****************************/
for(i=0; i<N; i++) {
    X[i]    += ALPHA * W[P][i];
    W[R][i] -= ALPHA * W[Q][i];
}

DNRM2 = 0.0;
for(i=0; i<N; i++) {
    DNRM2 += W[R][i]*W[R][i];
}

ERR = sqrt(DNRM2/BNRM2);
if((L+1)%100 ==1) {
   fprintf(stderr, "%5d%16.6e¥n", L+1, ERR);
}
if(ERR < EPS) {
   *IER = 0; goto N900;
  } else {
   RHO1 = RHO;
  }
}
*IER = 1;
```

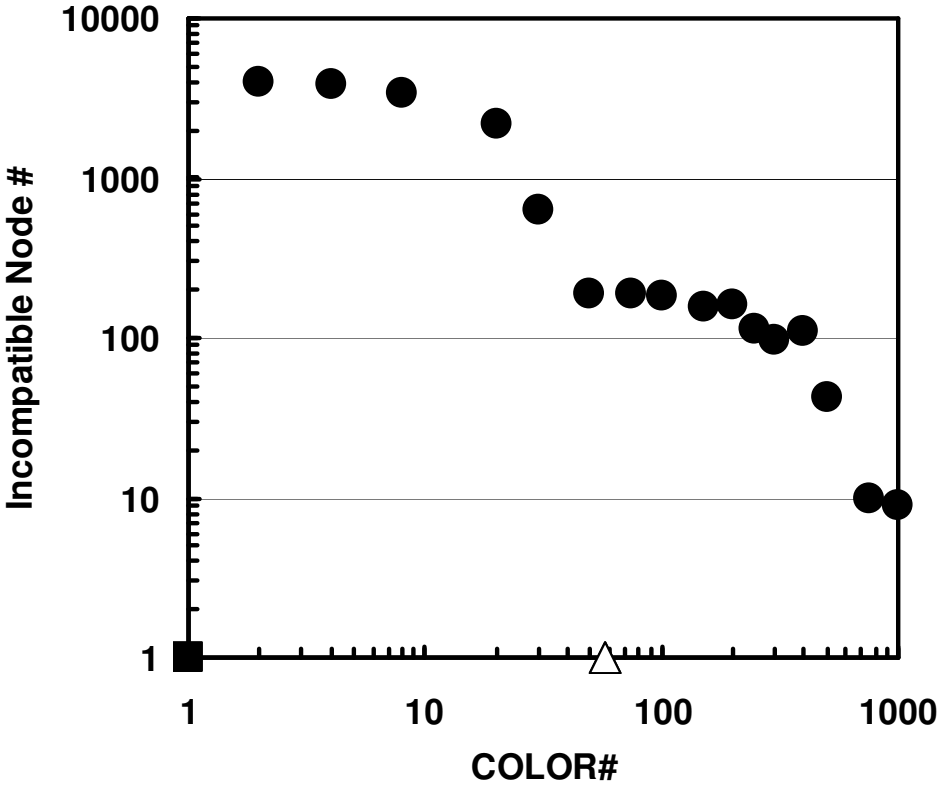
Compute $r^{(0)}= b-[A]x^{(0)}$

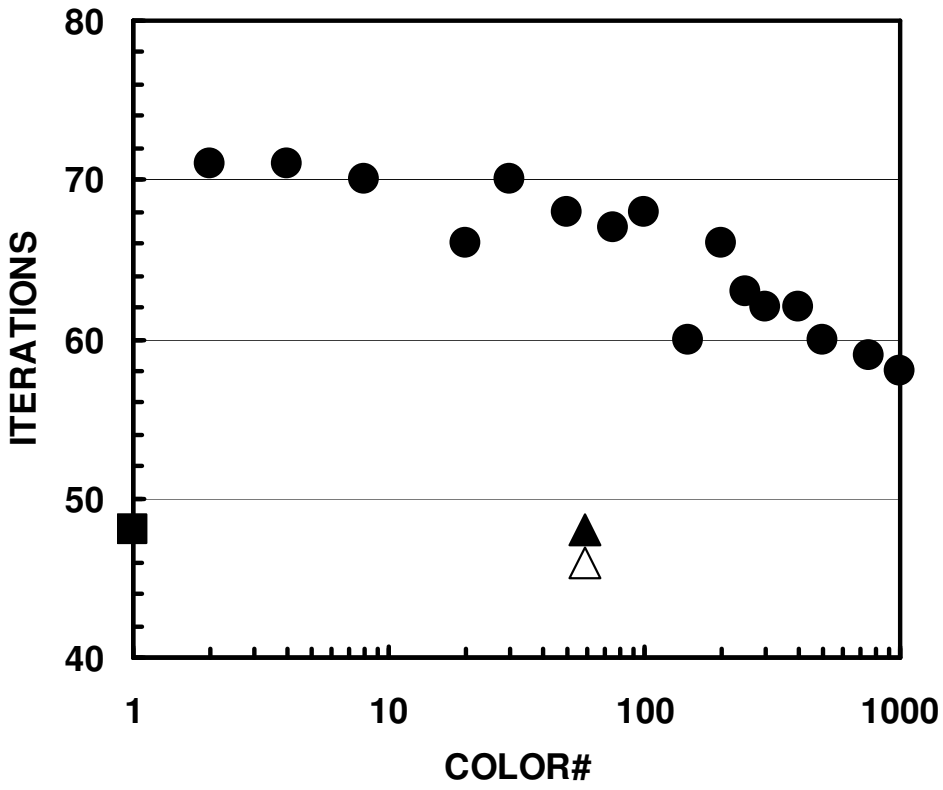
for i= 1, 2, …

- solve $[M]z^{(i-1)}= r^{(i-1)}$
- $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
- if i=1
  - $p^{(1)}= z^{(0)}$
- else
  - $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
  - $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
- endif
- $q^{(i)}= [A]p^{(i)}$
- **$\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$**
- **$x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$**
- **$r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$**
- **check convergence |r|**

end

ICCG法の収束





(20^3=8,000要素, EPSICCG=10^{-8})

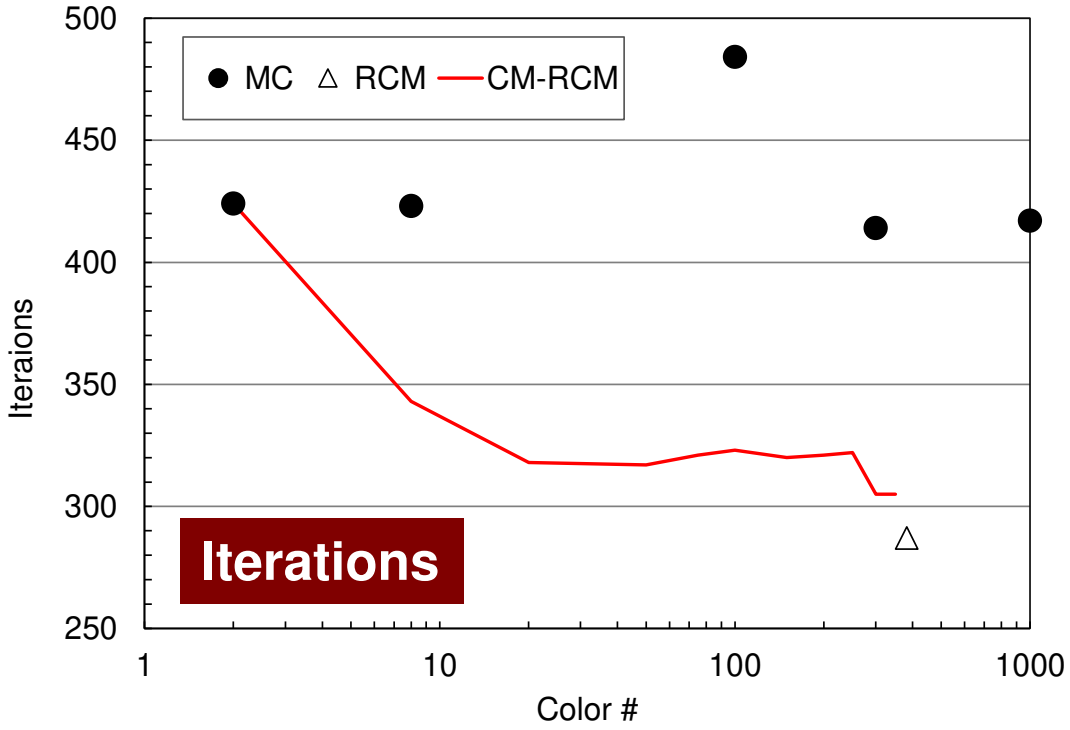
(■:ICCG(L1), ●:ICCG-MC, ▲:ICCG-CM, △:ICCG-RCM)

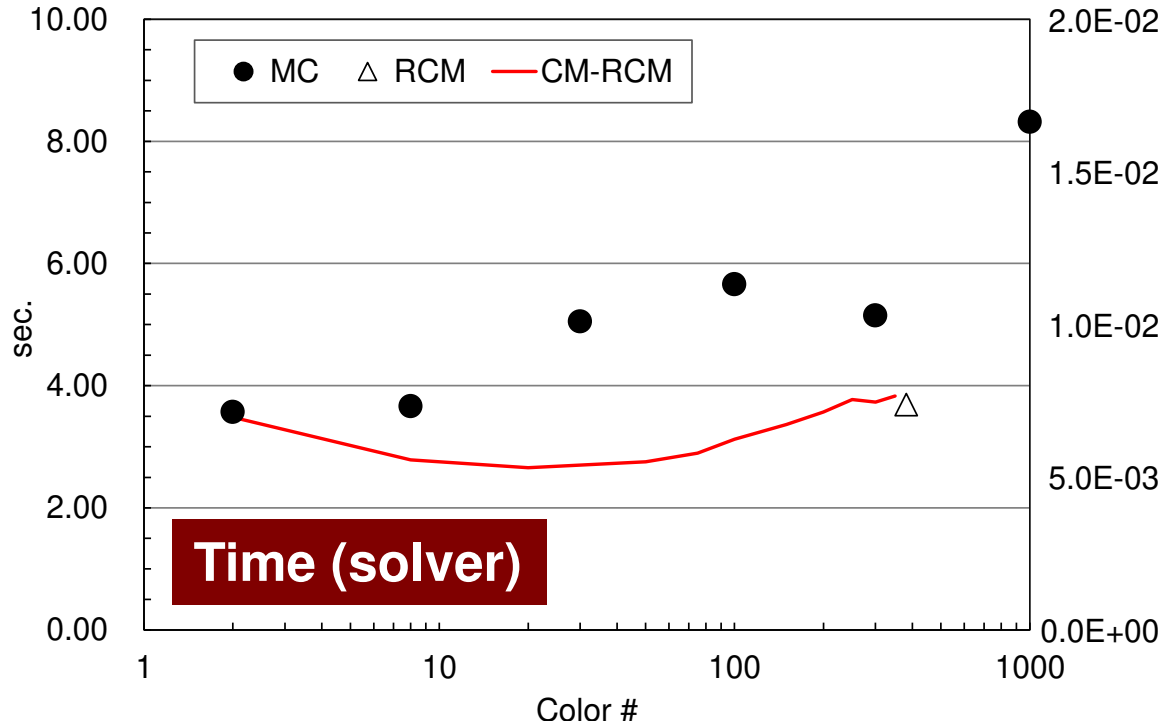
- データ依存性の解決策は?
- オーダリング(Ordering)について
  - Red-Black, Multicolor(MC)
  - Cuthill-McKee(CM), Reverse-CM(RCM)
  - オーダリングと収束の関係
- オーダリングの実装
- オーダリング付ICCG法の実装
- **マルチコアへの実装(OpenMP)へ向けて**
  - **L2-solにOpenMPを適用するだけ・・・**

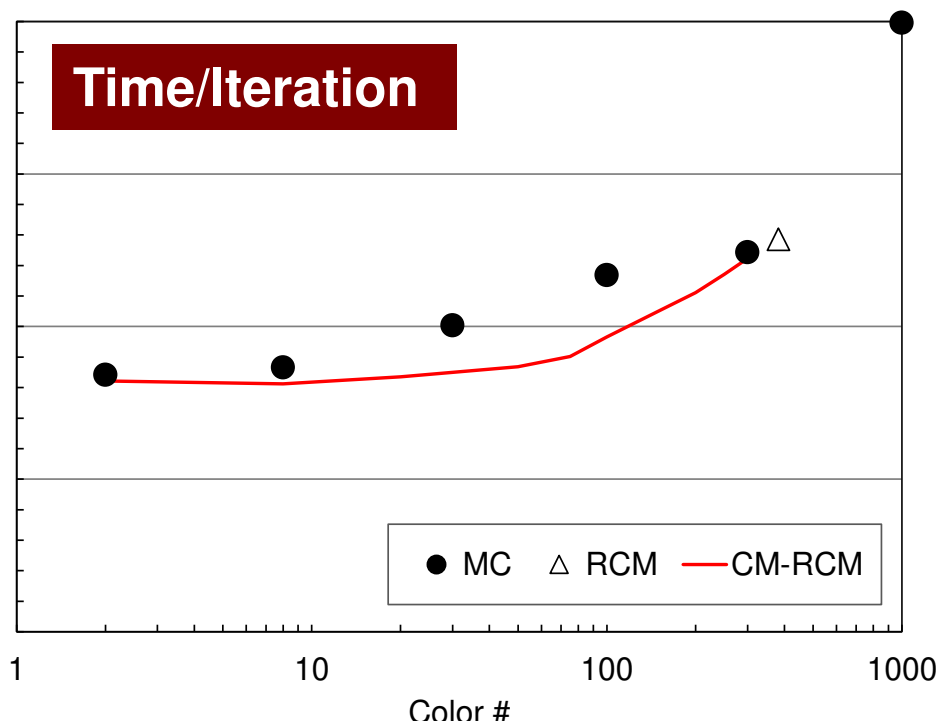
Odyssey

1-CMG/12-cores, 128^3

(●:MC, △:RCM, -:CM-RCM)







OBCX, 1-soket/24-cores, 128^3

(●:MC, △:RCM, -:CM-RCM)