

科学技術計算のための  
マルチコアプログラミング入門  
C言語編  
第Ⅱ部: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	<b>14</b>	15	<b>16</b>
<b>9</b>	10	<b>11</b>	12
5	<b>6</b>	7	<b>8</b>
<b>1</b>	2	<b>3</b>	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

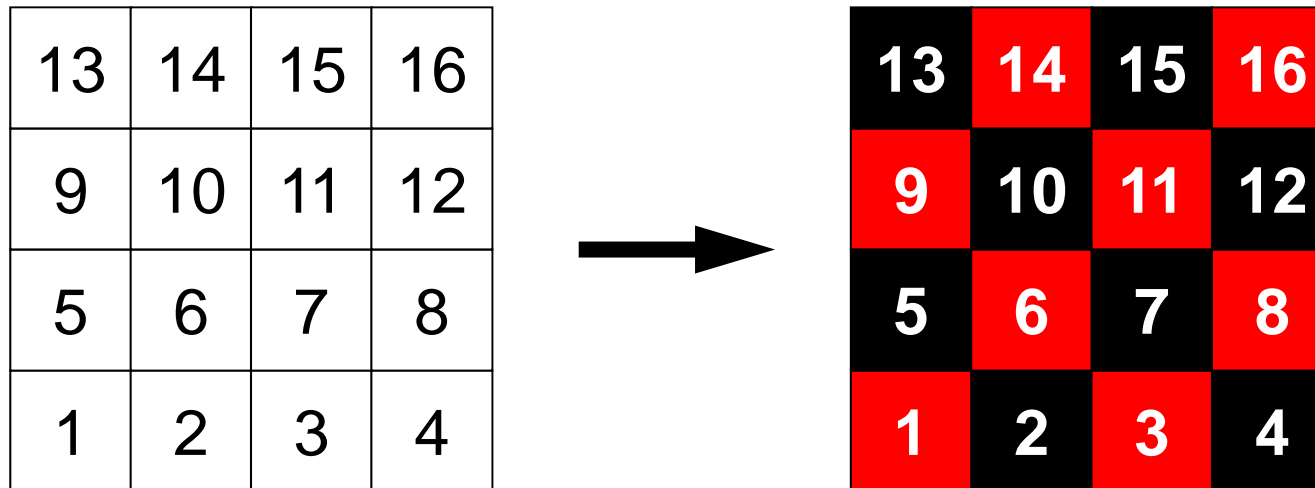
13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4



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

# 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)



**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)

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

```
#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" ){
      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];
      }
    }
  }
}
```

```
#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" ){
      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];
      }
    }
  }
}
```

# Red-Black (2/3)

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

```
#pragma 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" ){
      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];
      }
    }
  }
}
```

- 「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)

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

- 「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

```
#pragma 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" ) {
      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];
      }
    }
  }
}
```

# Red-Black Ordering/Reordering

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



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

```

for(icol=0; icol<2; icol++){
#pragma omp parallel for private (ip, i, j, VAL)
  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

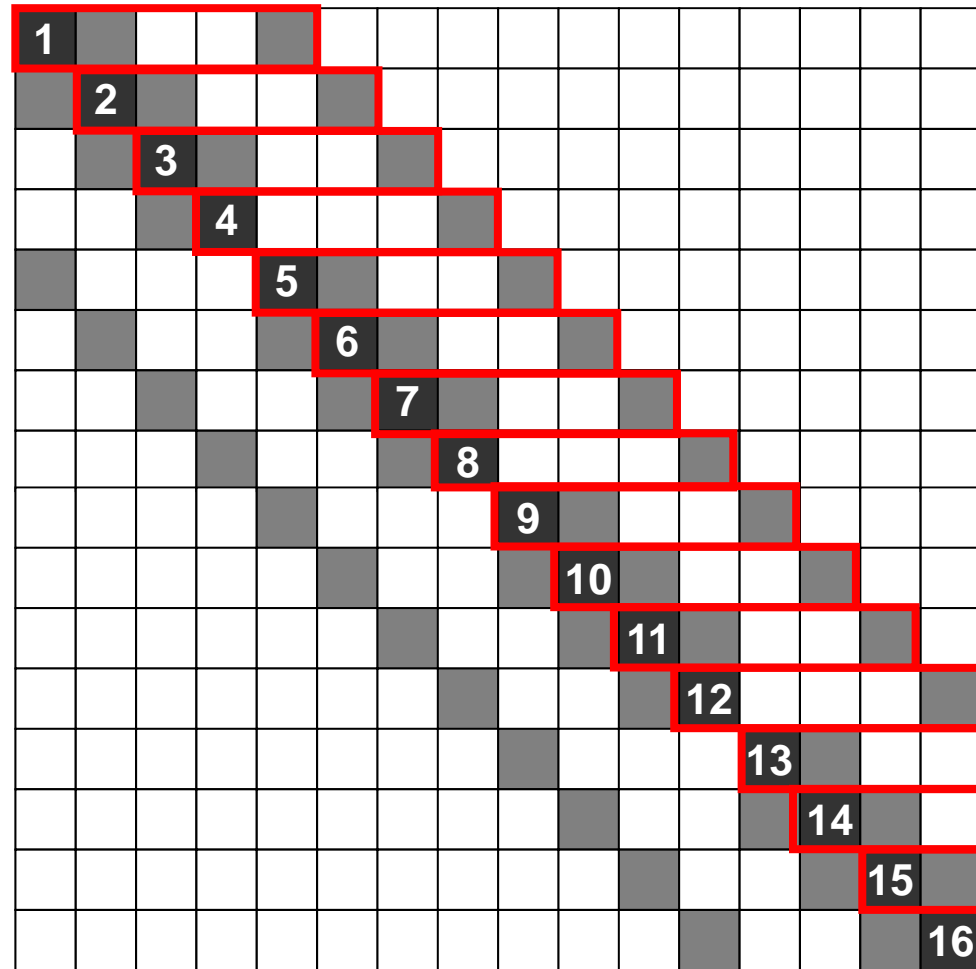
- マルチカラーオーダリング
  - 並列性
  - Red-Black オーダリング(2色)等
- CM法(Cuthill-McKee), RCM法(Reverse Cuthill-McKee)
  - fill-inを減らす
  - マトリクスのバンド幅を減らす, プロファイルを減らす
  - 並列性

# 用語の定義

- $\beta_i$ :  $i$  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:  $\beta_i$  の最大値
- プロフィール:  $\beta_i$  の和
- バンド幅, プロフィール, Fill-in とともに少ない方が都合が良い
- 特にバンド幅, プロフィールは収束に影響

# $\beta_i$ 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

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

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

- 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



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

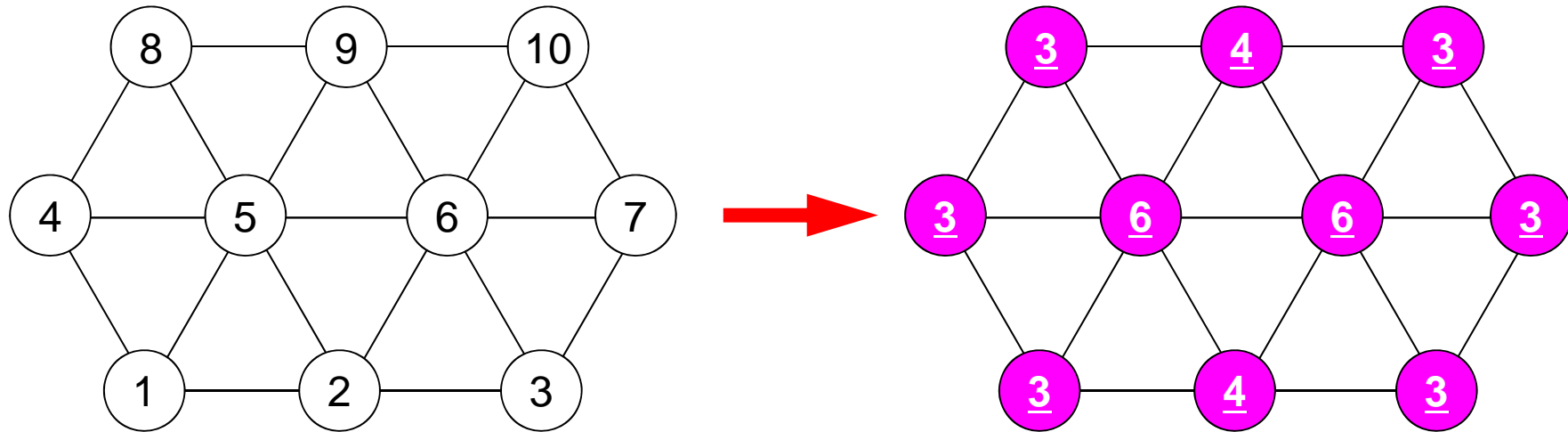


	11		12
9		10	
7	3	8	4
1	5	2	6

# 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  $ITEMcou = ICELTOT / NCOLOR_{tot}$ 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 $NCOLOR_{tot}$ 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)。

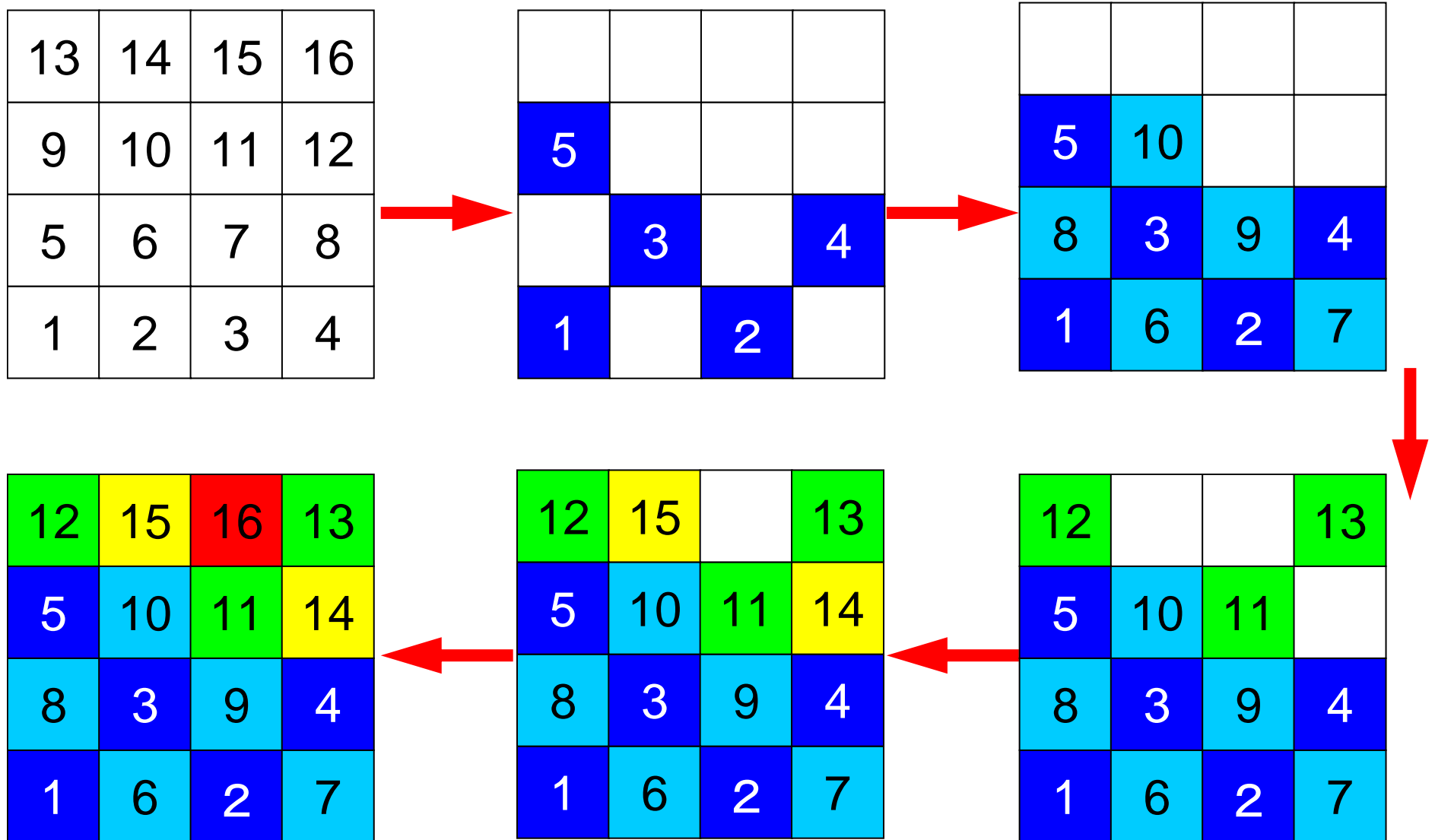
# 次数 (degree) : 各点隣接する点の数



# 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  $ITEM_{cou} = ICELTOT / N_{COLOR_{tot}}$ に相当する値を「各色に含まれる要素数」とする。
- ③  $ITEM_{cou}$ 個の独立な要素を初期要素番号が若い順に選び出す。
- ④  $ITEM_{cou}$ 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 $N_{COLOR_{tot}}$ 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

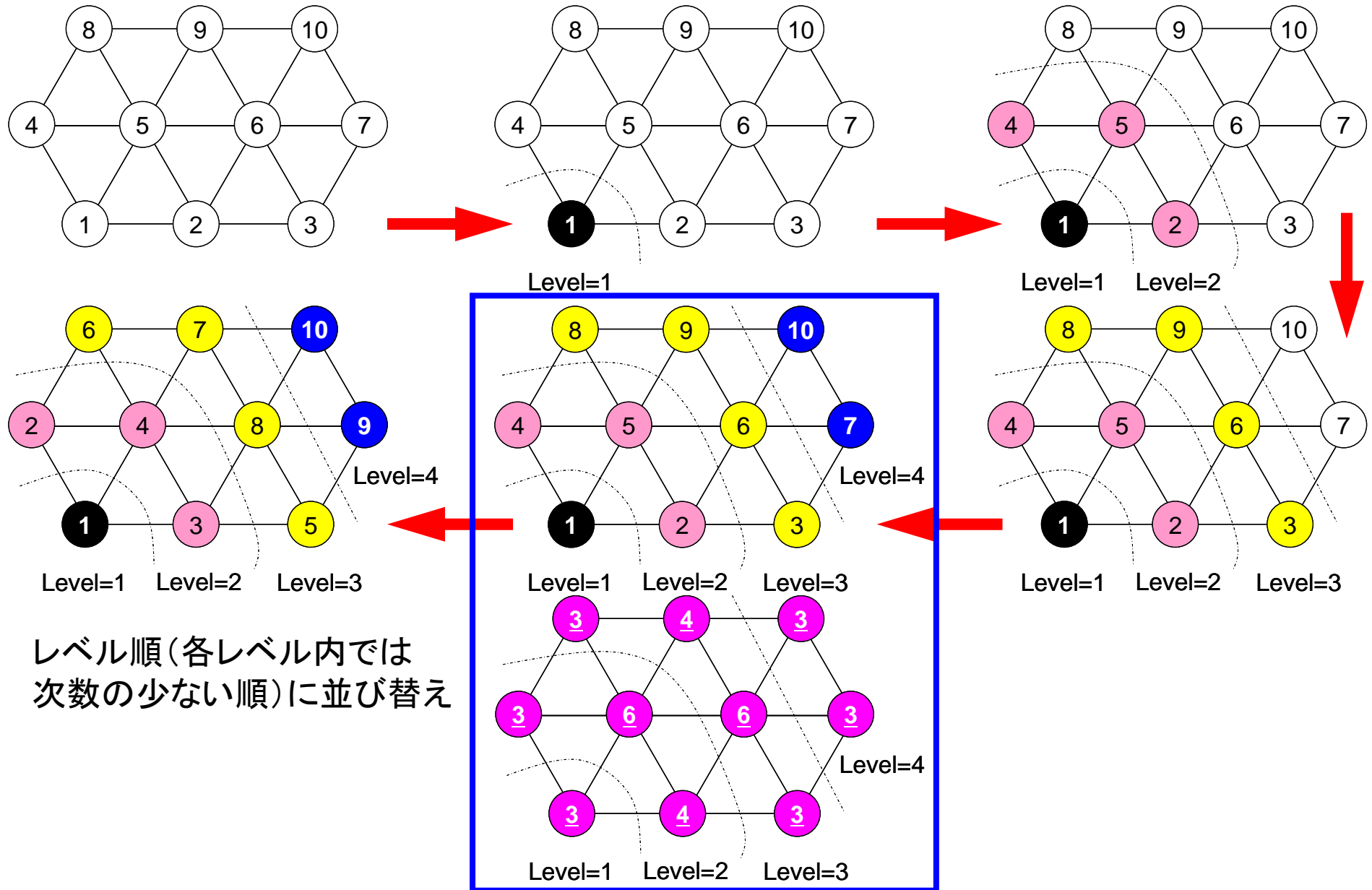
- マルチカラーオーダリング
  - 並列性
  - Red-Black オーダリング(2色)等
- CM法(Cuthill-McKee), RCM法(Reverse Cuthill-McKee)
  - fill-inを減らす
  - マトリクスのバンド幅を減らす, プロファイルを減らす
  - 並列性

# 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)。



# Cuthill-McKee Ordering (CM法) の例



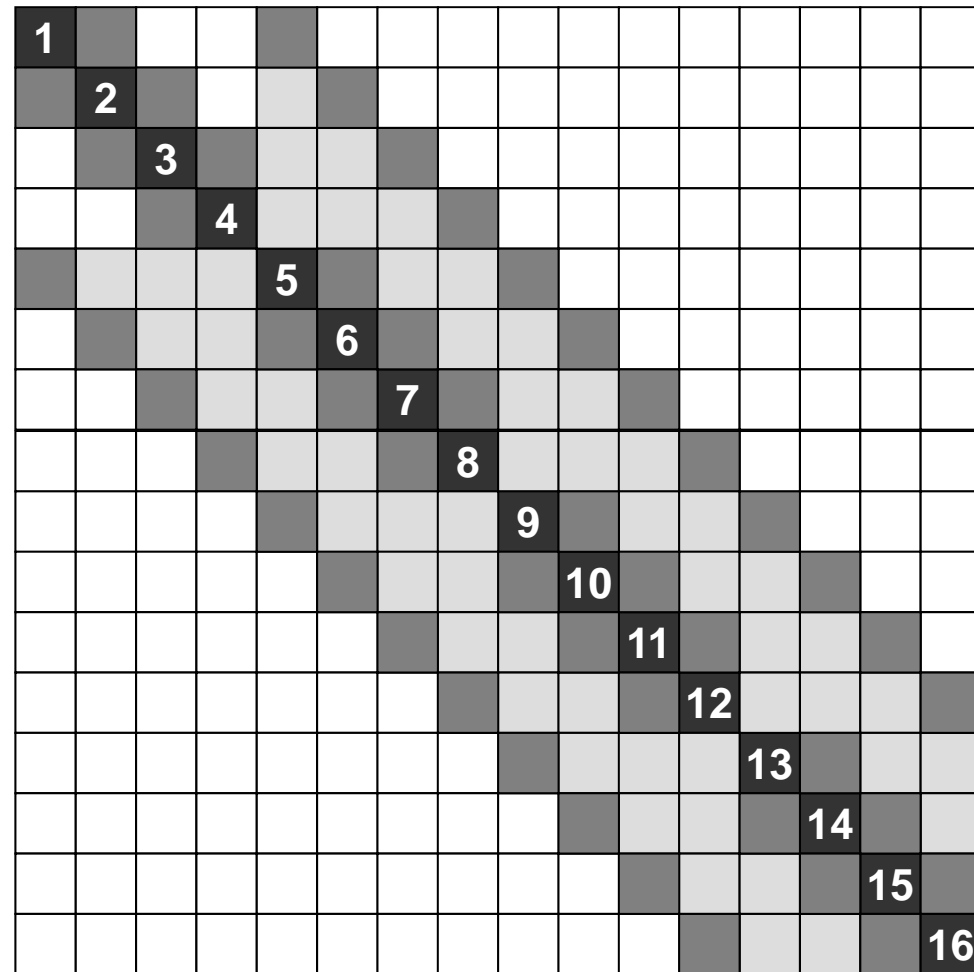
# RCM (Reverse CM)

- まずCMの手順を実行
  - 次数 (degree) の計算
  - 「レベル( $k(k \geq 2)$ )」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

# 初期状態

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

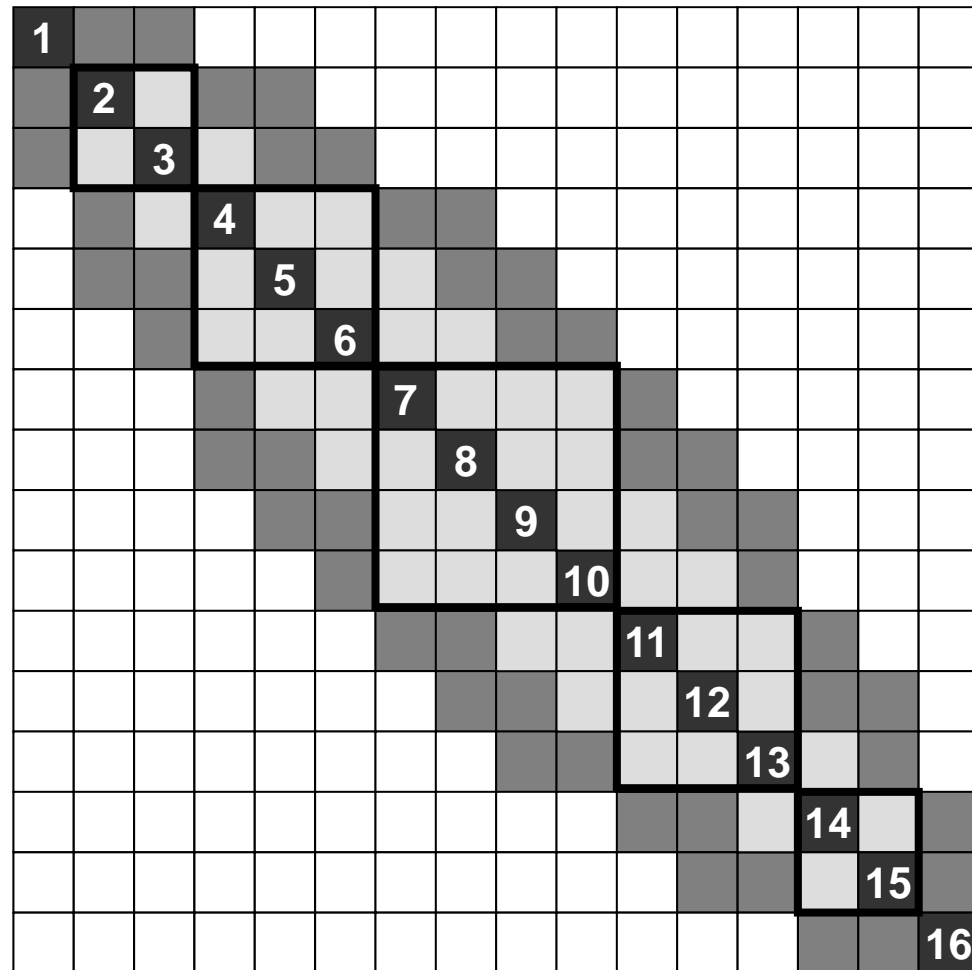
バンド幅 4  
 プロフィール 51  
 Fill-in 54



# CMオーダリング

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

バンド幅        4  
 プロフィール    46  
 Fill-in         44

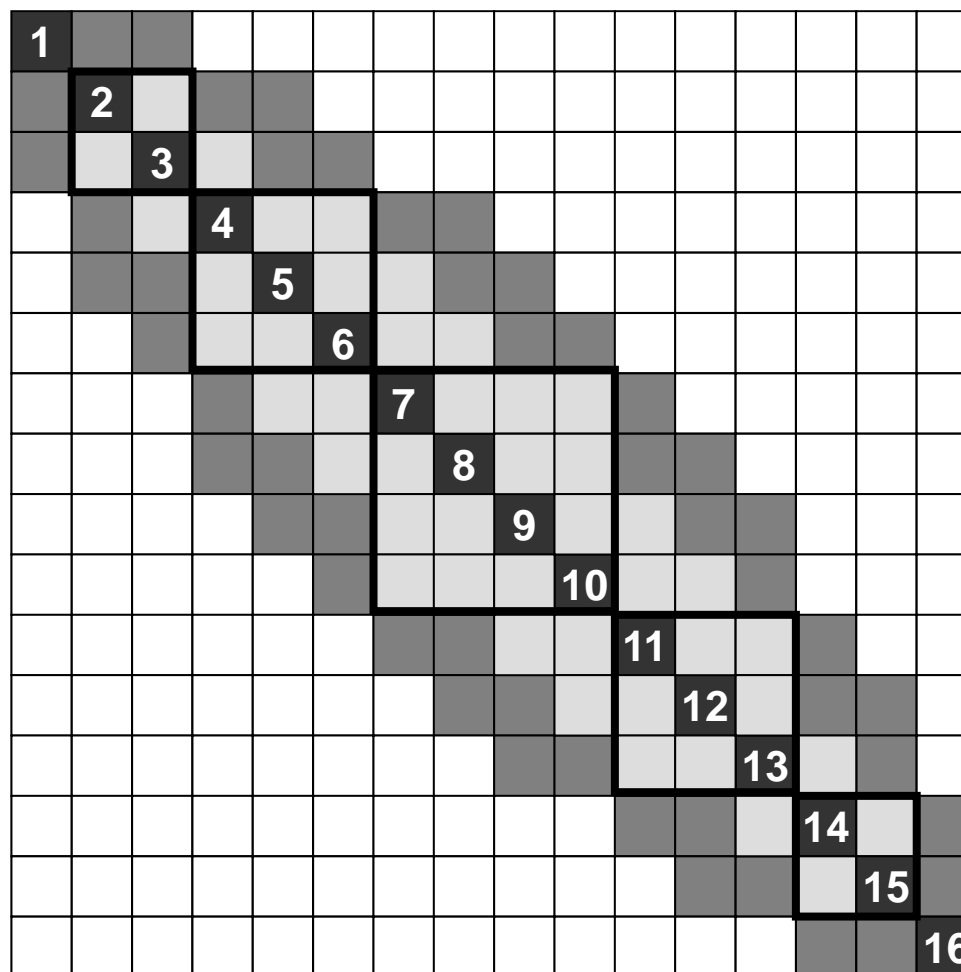


■ 非ゼロ成分, ■ Fill-in

# RCMオーダリング

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

バンド幅 4  
 プロフィール 46  
 Fill-in 44

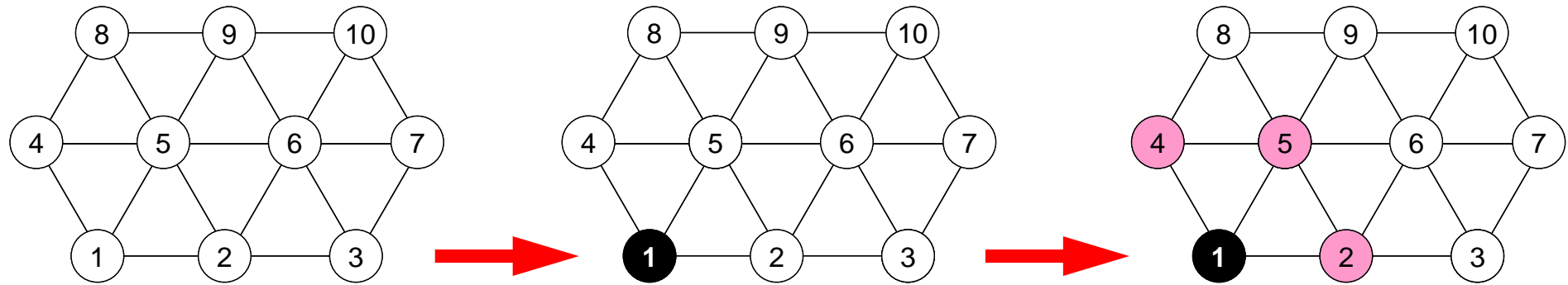


■ 非ゼロ成分, ■ Fill-in

# 修正されたCM法 並列計算向け

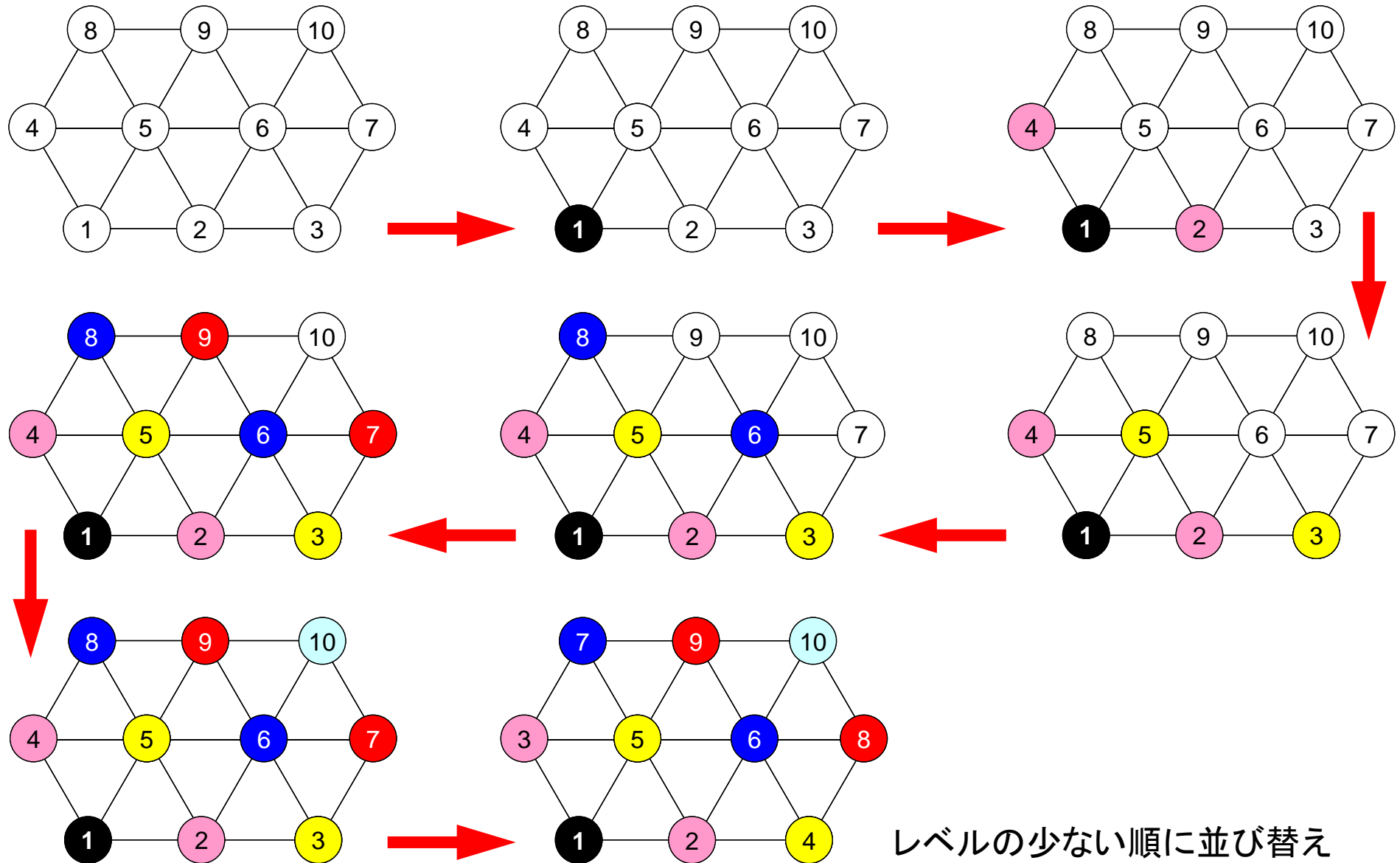
- ① 各要素に隣接する要素数を「次数」とし、最小次数の要素を「レベル=1」の要素とする。
- ② 「レベル=k-1」の要素に隣接する要素を「レベル=k」とする。同じレベルに属する要素はデータ依存性が発生しないように、隣接している要素同士が同じレベルに入る場合は一方を除外する(現状では先に見つかった要素を優先している)。全ての点要素にレベルづけがされるまで「k」を1つずつ増やして繰り返す。
- ③ 全ての要素がレベルづけされたら、レベルの順番に再番号づけする。同じレベル:連続した「新」要素番号

# 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



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



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

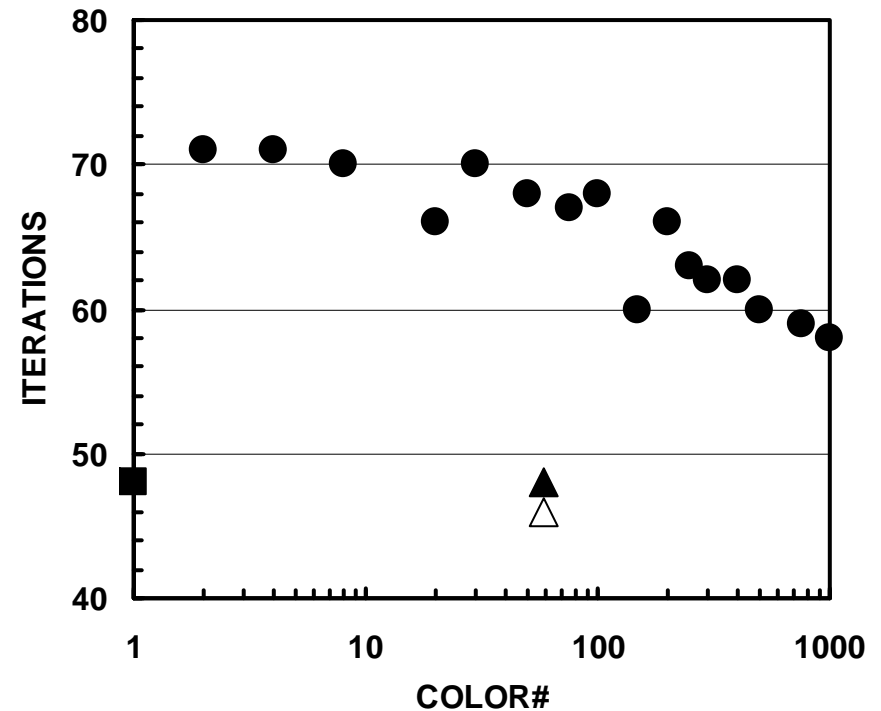
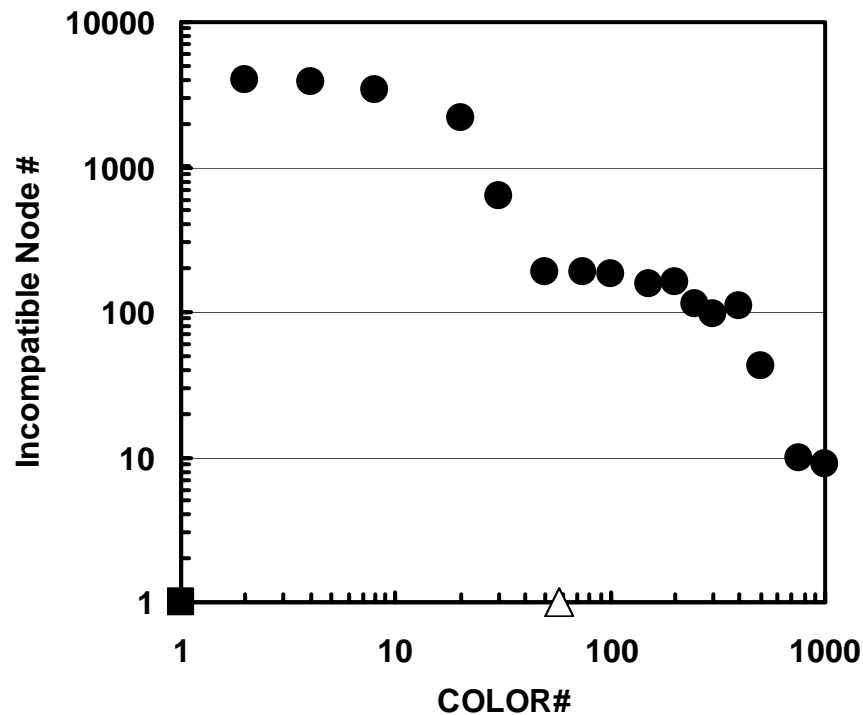
15	7	16	8
5	13	6	14
11	3	12	4
1	9	2	10

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

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

- 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  $\sim$  4色  $<$  初期状態  $\sim$  CM, RCM

## 初期状態

バンド幅	4
プロフィール	51
Fill-in	54

## Red-Black

バンド幅	10
プロフィール	77
Fill-in	44

## 4色

バンド幅	10
プロフィール	57
Fill-in	46

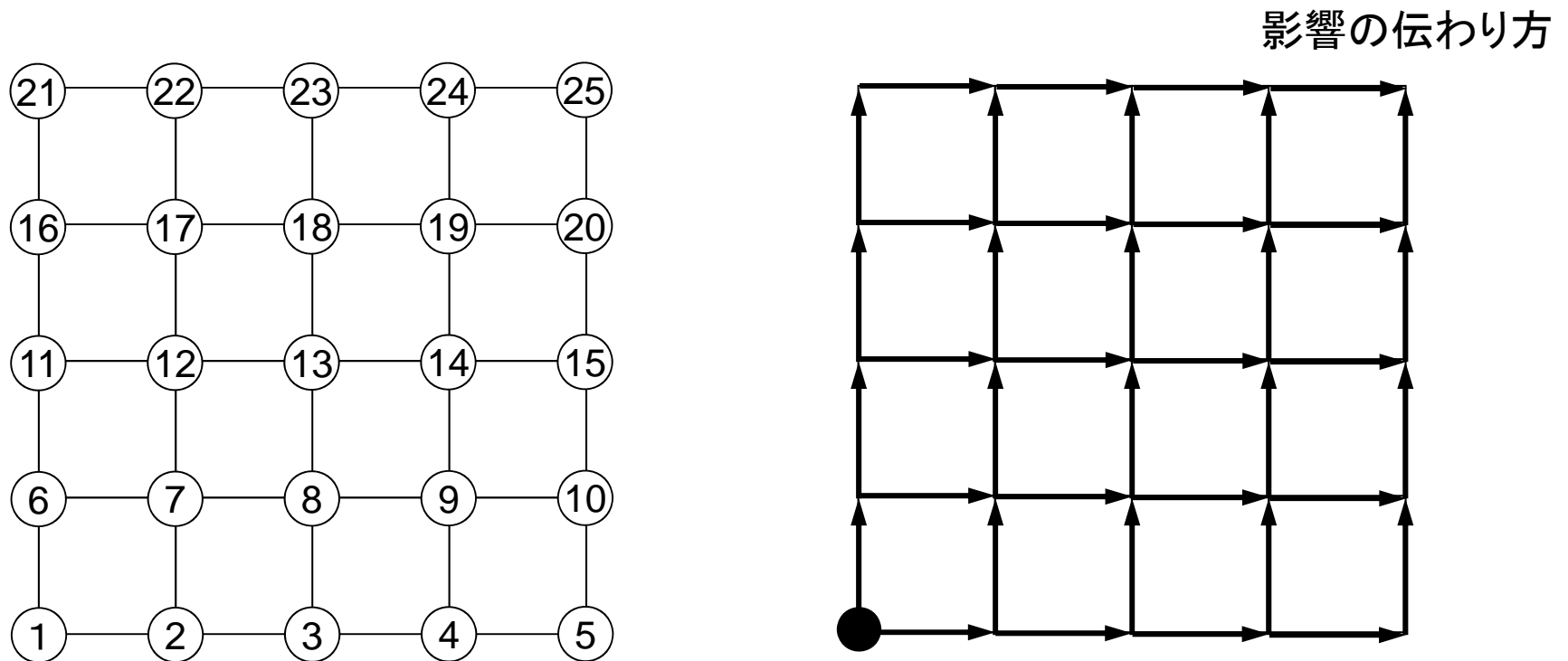
## CM, RCM

バンド幅	4
プロフィール	46
Fill-in	44

# 反復回数と色数の関係

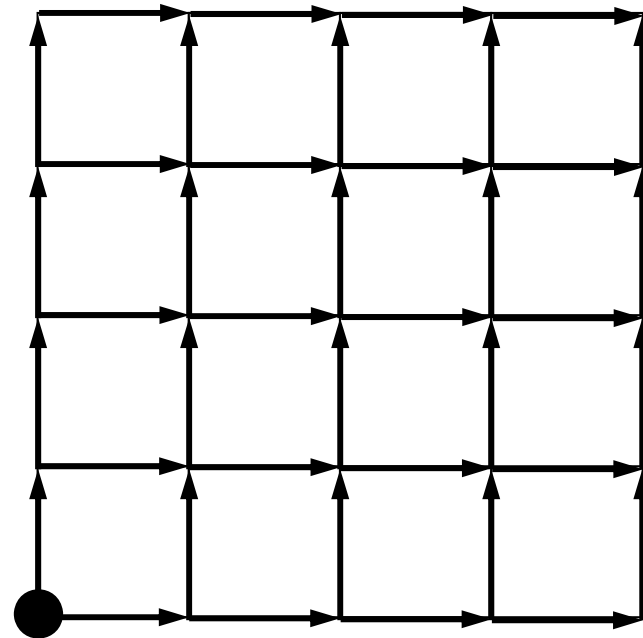
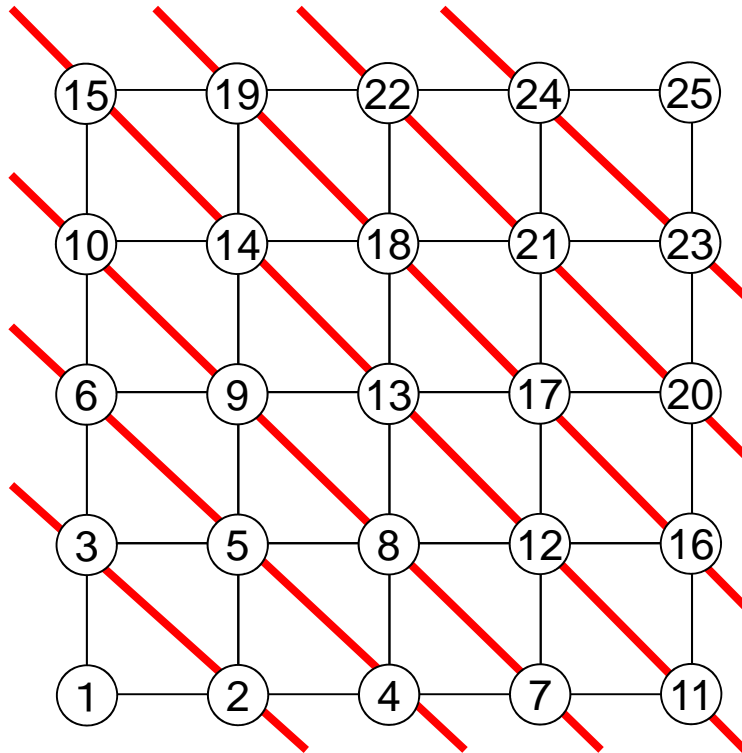
## Incompatible Nodesとは?

Doi, S. (NEC) et al.



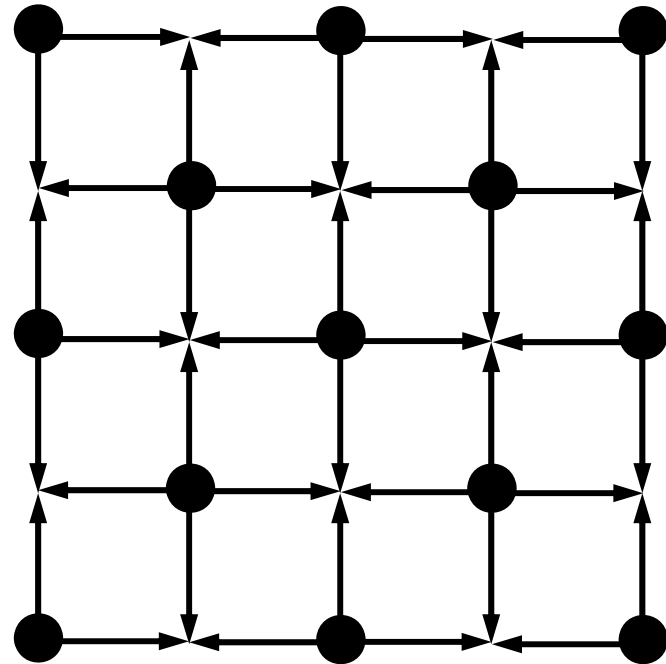
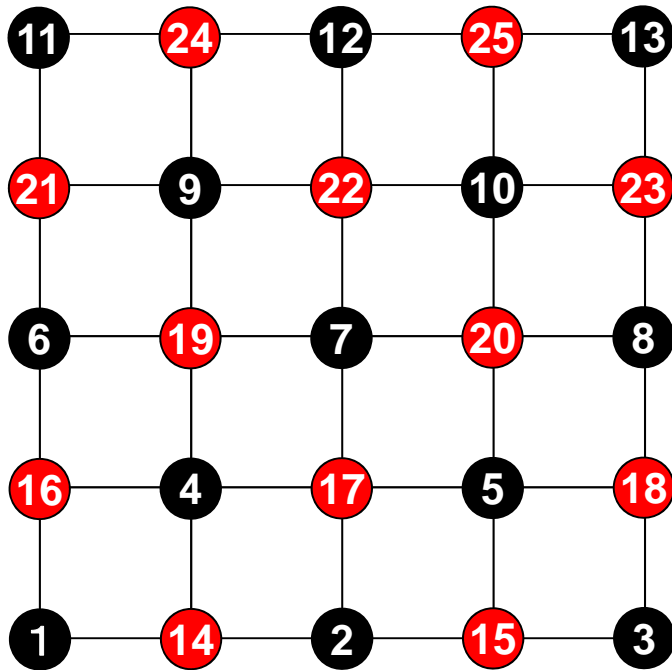
他から影響を受けない点が「Incompatible Node」  
周囲の全ての点よりも番号が若い, ということ  
少ない方が収束が早い

# CM (Cuthill-McKee) の場合



# Red-Blackの場合

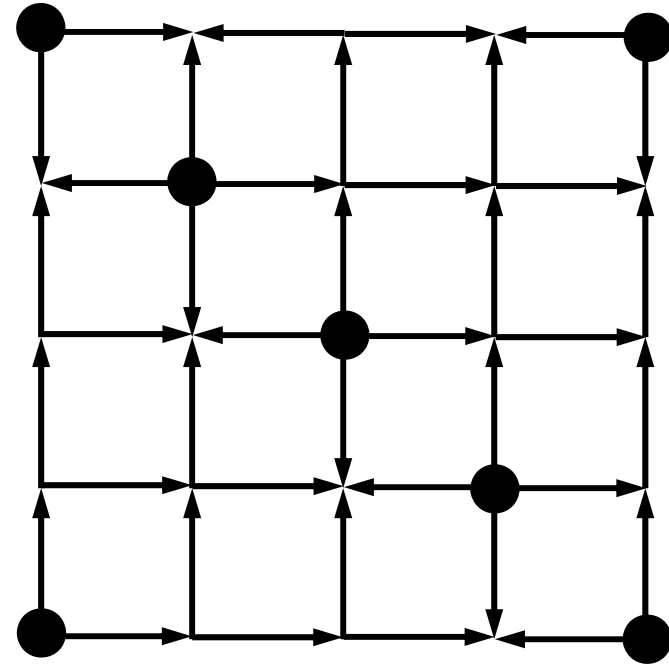
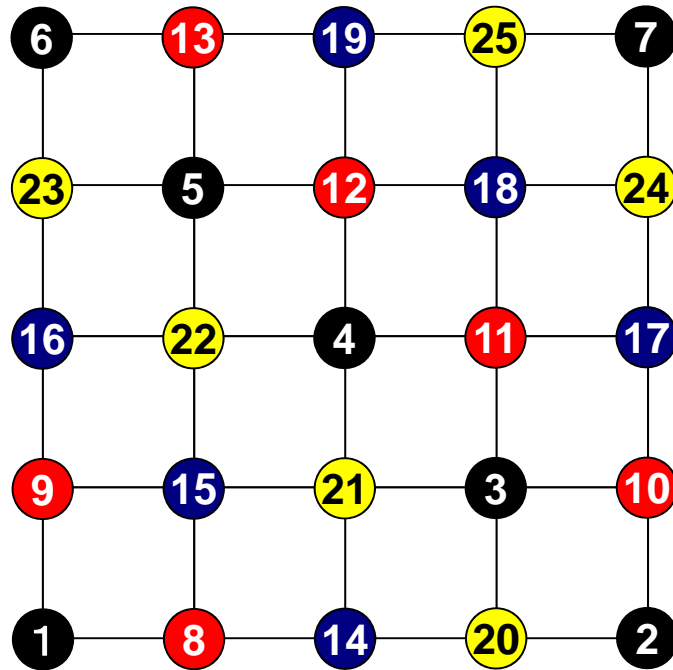
並列性は高いがincompatible node多い  
ILU系前処理, Gauss-Seidelで反復回数増加





# 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



# 収束とオーダーリング

- これ以外にも境界条件の影響などもあり, 一概には言えない
- たとえばCMとRCMは本問題の場合全く同じになるはずであるが, RCMの方が若干収束が良い

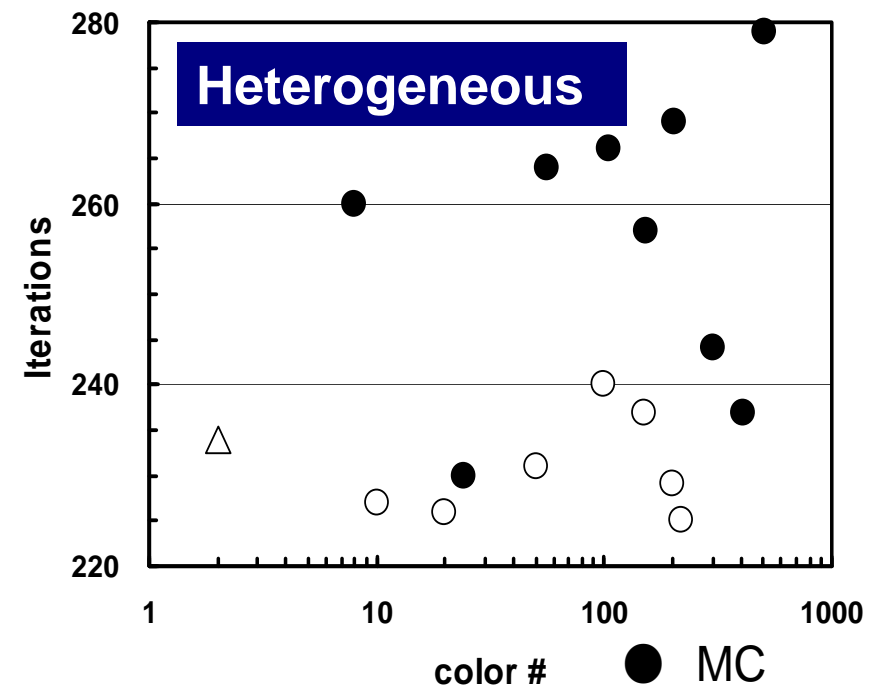
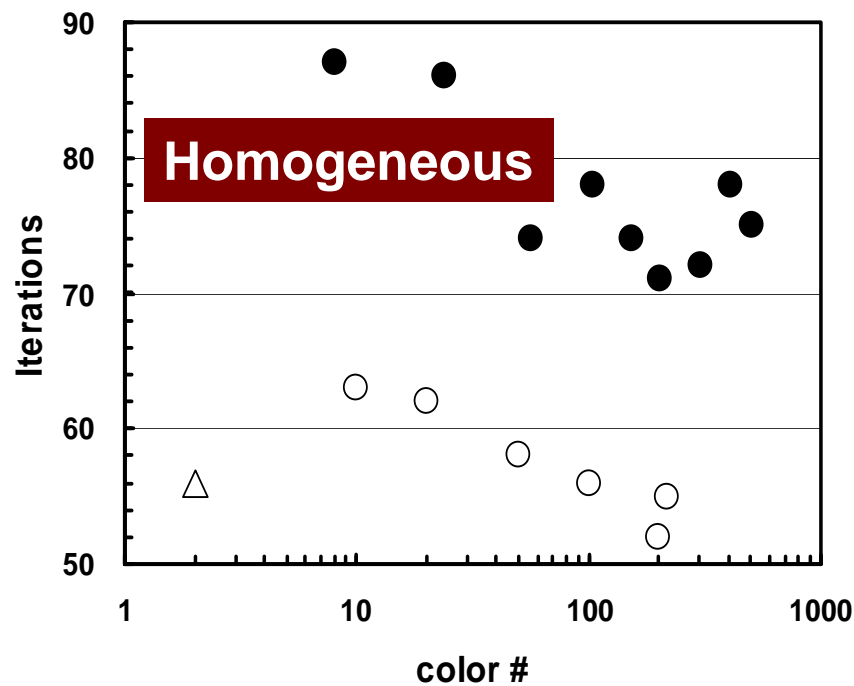
# オーダリングの効果

- 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。

# 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

- MC
- CM-RCM
- △ No reordering

- 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



- 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 ?
```

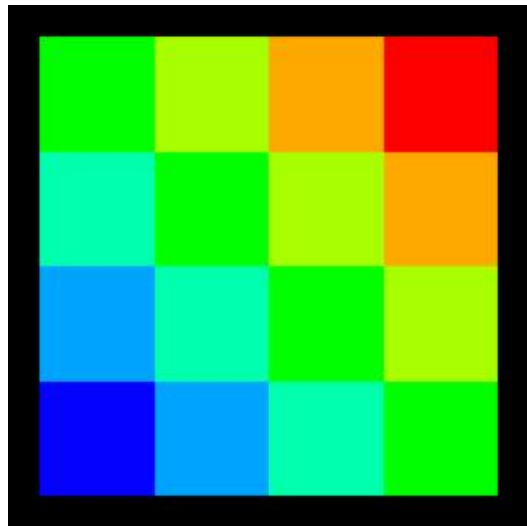
```
4 4 1 ← このように入力する
You have 16 elements.
How many colors do you need ?
#COLOR must be more than 2 and
#COLOR must not be more than 16
CM if #COLOR .eq. 0
RCM if #COLOR .eq.-1
CMRCM if #COLOR .le.-2
=>
```

この2次元形状を接続関係に基づき色づけする。

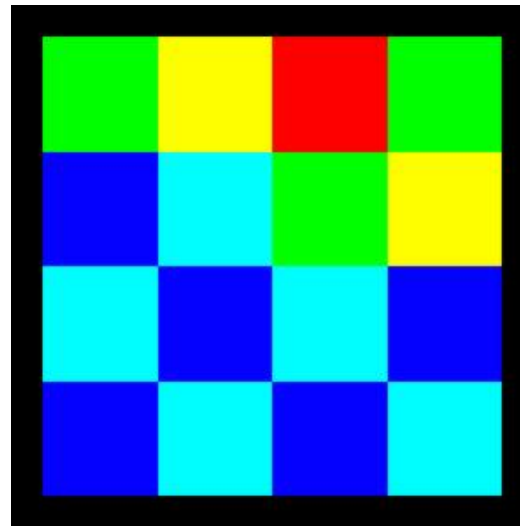
13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

# 実施内容 (2/2)

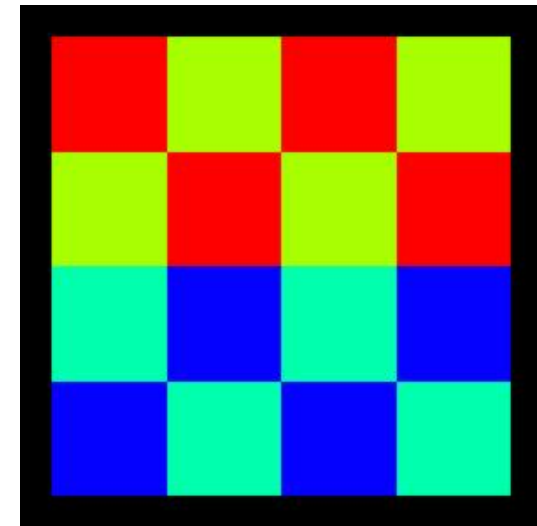
- 2つのファイルが生成される
  - color.log      新旧メッシュ番号の対応表  
                  行列関連情報
  - color.inp      メッシュの色分けファイル (ParaView用)



入力: 0  
(CM, 7 colors)



入力: 3  
(5 colors)



入力: 4  
(4 colors)

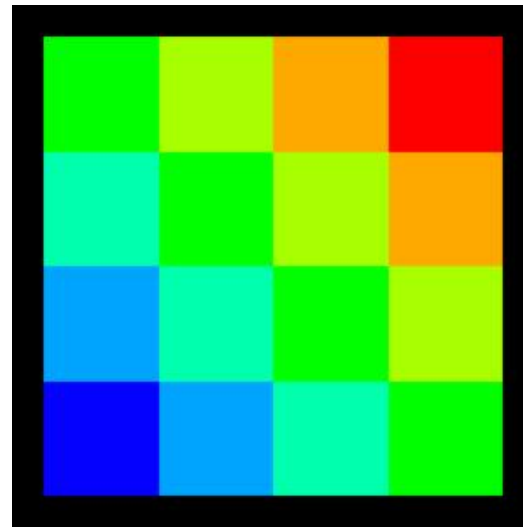
# 入力=0: CM(7色)

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





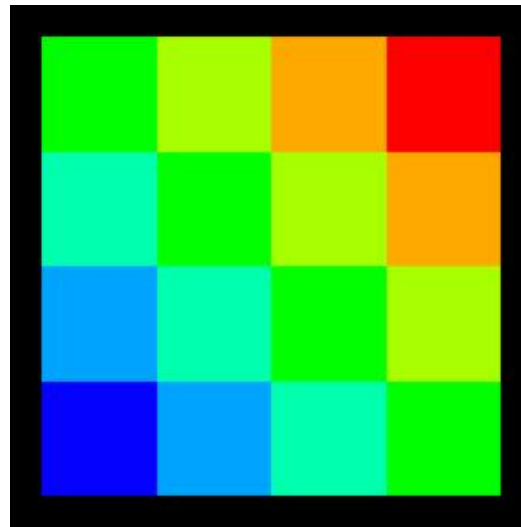
# 入力=0: CM(7色)

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



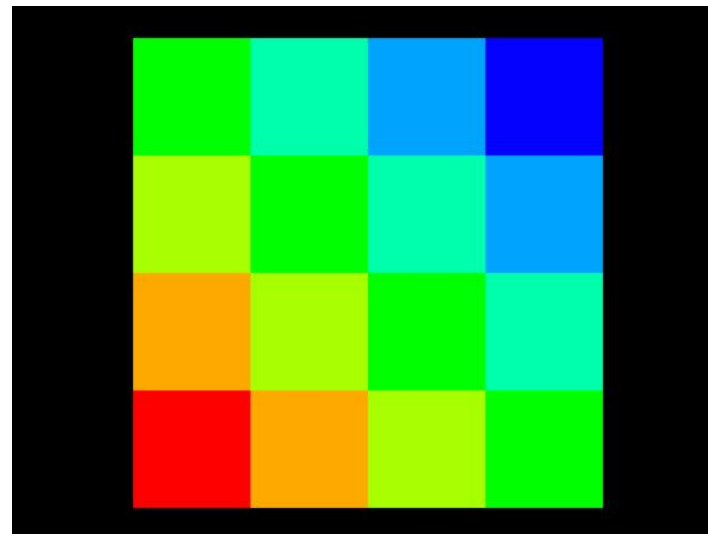
# 入力=-1 : RCM(7色)

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



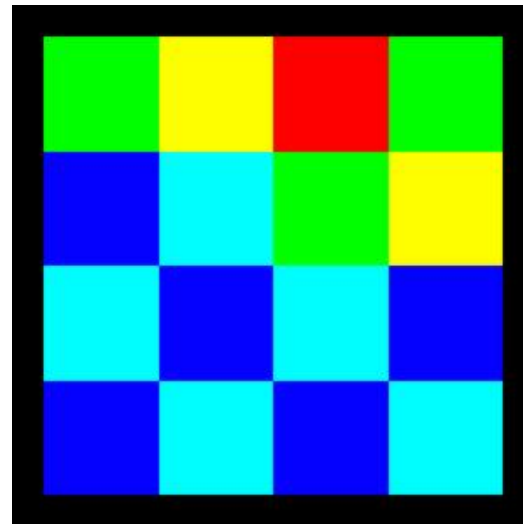
# 入力=3: 実際は5色(マルチカラー)

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



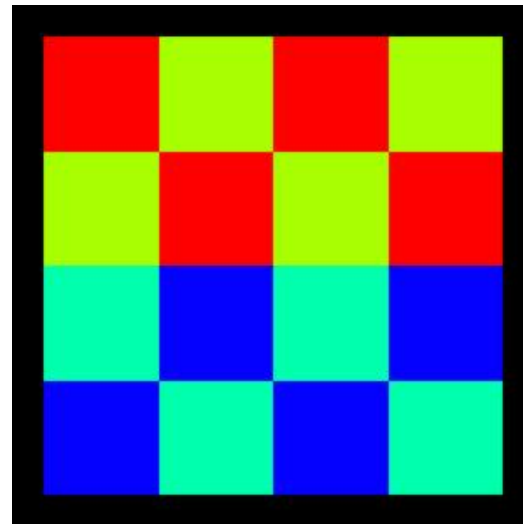
# 入力=4: 4色(マルチカラー)

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	<u>6</u>	<u>7</u>	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: 1
  IAU: 2
I= 2 INL(i)= 1 INU(i)= 2
  IAL: 1
  IAU: 3
I= 3 INL(i)= 1 INU(i)= 2
  IAL: 2
  IAU: 4
I= 4 INL(i)= 1 INU(i)= 1
  IAL: 3
  IAU: 8
I= 5 INL(i)= 1 INU(i)= 2
  IAL: 1
  IAU: 6
I= 6 INL(i)= 2 INU(i)= 2
  IAL: 2
  IAU: 7
I= 7 INL(i)= 2 INU(i)= 2
  IAL: 3
  IAU: 8
I= 8 INL(i)= 2 INU(i)= 1
  IAL: 4
  IAU: 12
I= 9 INL(i)= 1 INU(i)= 2
  IAL: 5
  IAU: 10
I= 10 INL(i)= 2 INU(i)= 2
  IAL: 6
  IAU: 11
I= 11 INL(i)= 2 INU(i)= 2
  IAL: 7
  IAU: 12
I= 12 INL(i)= 2 INU(i)= 1
  IAL: 8
  IAU: 16
I= 13 INL(i)= 1 INU(i)= 1
  IAL: 9
  IAU: 14
    
```

```

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

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	<u>6</u>	<u>7</u>	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:
I= 2 INL(i)= 0 INU(i)= 3
IAL:
IAU:
I= 3 INL(i)= 0 INU(i)= 4
IAL:
IAU:
I= 4 INL(i)= 0 INU(i)= 3
IAL:
IAU:
I= 5 INL(i)= 0 INU(i)= 3
IAL:
IAU:
I= 6 INL(i)= 3 INU(i)= 0
IAL:
IAU:
I= 7 INL(i)= 2 INU(i)= 0
IAL:
IAU:
I= 8 INL(i)= 3 INU(i)= 0
IAL:
IAU:
I= 9 INL(i)= 3 INU(i)= 1
IAL:
IAU:
I= 10 INL(i)= 2 INU(i)= 2
IAL:
IAU:
I= 11 INL(i)= 2 INU(i)= 2
IAL:
IAU:
I= 12 INL(i)= 1 INU(i)= 1
IAL:
IAU:
I= 13 INL(i)= 0 INU(i)= 2
IAL:
IAU:

```

```

I= 14 INL(i)= 3 INU(i)= 0
IAL:
IAU:
I= 15 INL(i)= 2 INU(i)= 1
IAL:
IAU:
I= 16 INL(i)= 3 INU(i)= 0
IAL:
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);
        }
    }

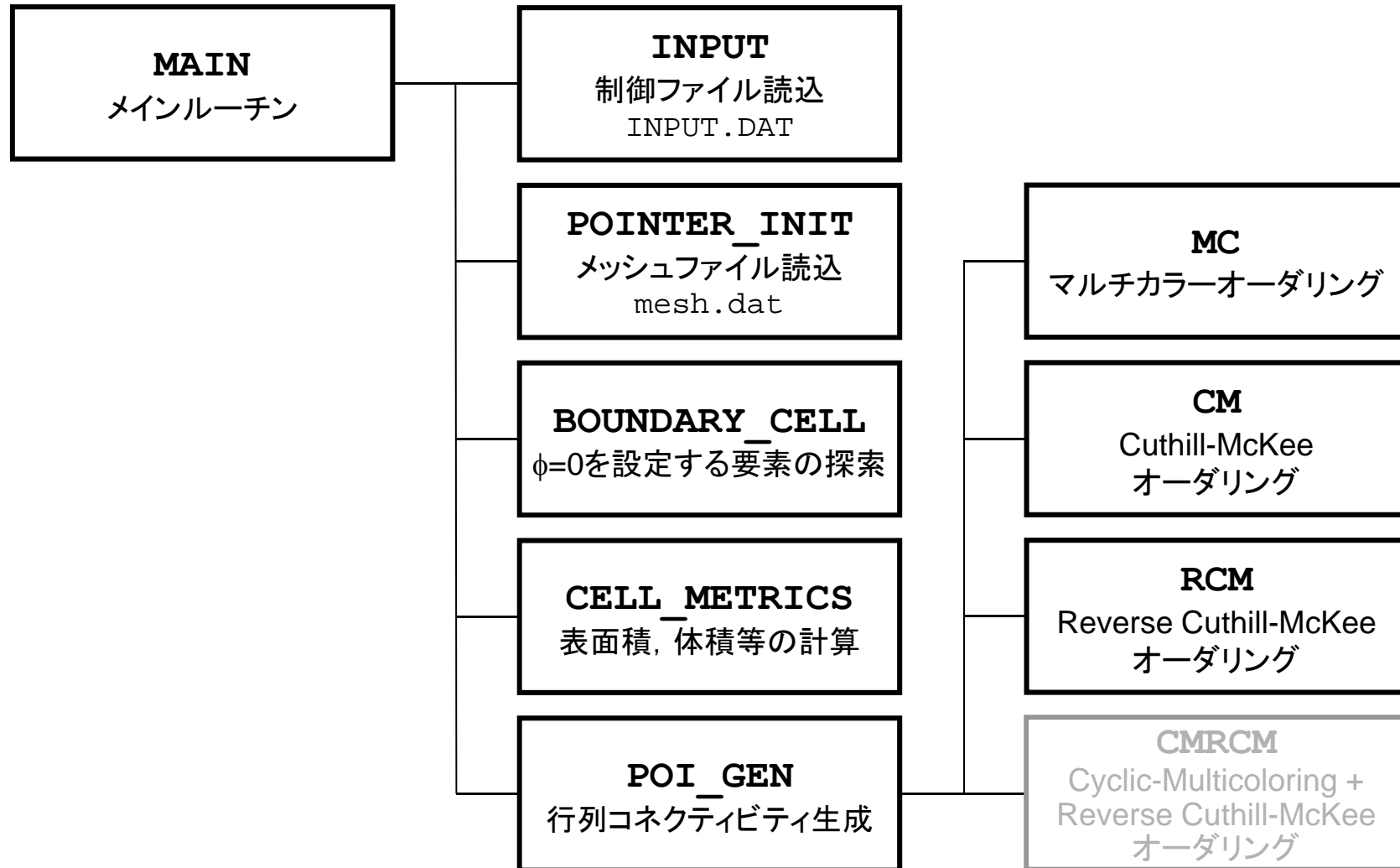
    (...)

    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:**

Number of meshes ( $NX \times NY \times NZ$ )

## **N:**

Number of modes

## **NX, NY, NZ:**

Number of meshes in x/y/z directions

## **NXP1, NYP1, NZP1:**

Number of nodes in x/y/z directions

## **IBNODTOT:**

=  $NXP1 \times NYP1$

## **XYZ [ICELTOT] [3]:**

Location of meshes

## **NEIBcell [ICELTOT] [6]:**

Neighboring meshes



```

#ifndef __H_PCG
#define __H_PCG
    static int N2 = 256;
    int NUm, NLm, NCOLortot, NCOLORk, NU,
    NL;

    int METHOD, ORDER_METHOD;
    double EPSICCG;

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

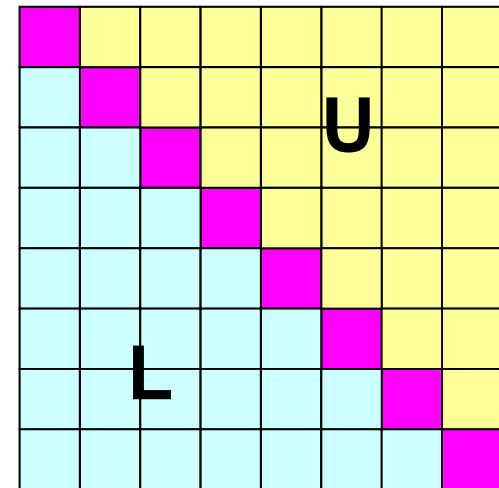
    int *INL, *INU, *COLORindex;
    int *indexL, *indexU;
    int *OLDtoNEW, *NEWtoOLD;
    int **IAL, **IAU;
    int *itemL, *itemU;
    int NPL, NPU;
#endif /* __H_PCG */

```

# pcg.h

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 */

```

```

INL[ ICELTOT ]
IAL[ ICELTOT ] [NL]
INU[ ICELTOT ]
IAU[ ICELTOT ] [NU]
NU, NL

```

# Non-zero off-diag. components (lower)  
**Col. ID: non-zero off-diag. comp. (lower)**  
 # Non-zero off-diag. components (upper)  
**Col. ID: non-zero off-diag. comp. (upper)**  
 Max # of L/U non-zero off-diag. comp.s (=6)

```

indexL[ ICELTOT+1 ]
indexU[ ICELTOT+1 ]
NPL, NPU
itemL[ NPL ], itemU[ NPU ]

```

# Non-zero off-diag. comp. (lower, CRS)  
 # Non-zero off-diag. comp. (upper, CRS)  
 Total # of L/U non-zero off-diag. comp.  
 Col. ID: non-zero off-diag. comp. (L/U, CRS)

# pcg.h

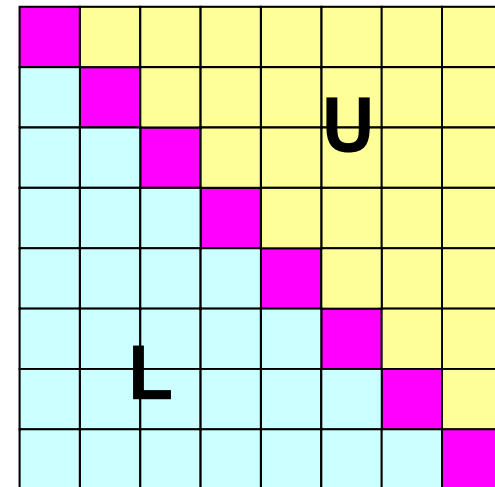
## Auxiliary Arrays

### Lower Part (Column ID)

$IAL[i][icou] < i$

### Upper Part (Column ID)

$IAU[i][icou] > i$



```

#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 */

```

**INL[ICELTOT]**

IAL[ICELTOT][NL]

**INU[ICELTOT]**

IAU[ICELTOT][NU]

NU, NL

indexL[ICELTOT+1]

indexU[ICELTOT+1]

NPL, NPU

itemL[NPL], itemU[NPU]

**# Non-zero off-diag. components (lower)**

Col. ID: non-zero off-diag. comp. (lower)

**# Non-zero off-diag. components (upper)**

Col. ID: non-zero off-diag. comp. (upper)

Max # of L/U non-zero off-diag. comp.s (=6)

# Non-zero off-diag. comp. (lower, CRS)

# Non-zero off-diag. comp. (upper, CRS)

Total # of L/U non-zero off-diag. comp.

Col. ID: non-zero off-diag. comp. (L/U, CRS)

# pcg.h

## Auxiliary Arrays

### Lower Part (Column ID)

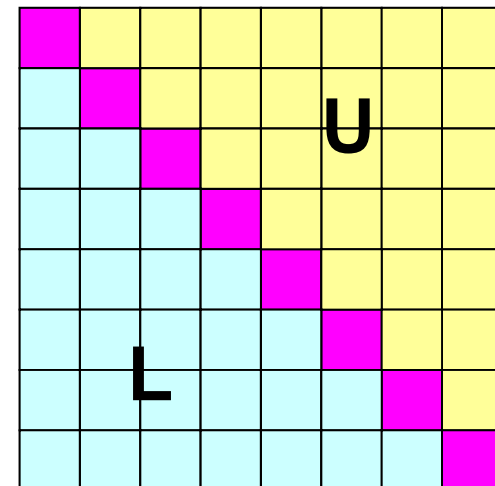
IAL[i][icou] < i

**INL[i] : Number@each row**

### Upper Part (Column ID)

IAU[i][icou] > i

**INU[i] : Number@each row**



# INPUT=3: MC, 5-Colors

color.log: matrix info.

13	14	15	16
9	10	11	12
5	<u>6</u>	<u>7</u>	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: 1
  IAU: 2
I= 2 INL(i)= 1 INU(i)= 2
  IAL: 1
  IAU: 3
I= 3 INL(i)= 1 INU(i)= 2
  IAL: 2
  IAU: 4
I= 4 INL(i)= 1 INU(i)= 1
  IAL: 3
  IAU: 8
I= 5 INL(i)= 1 INU(i)= 2
  IAL: 1
  IAU: 6
I= 6 INL(i)= 2 INU(i)= 2
  IAL: 2
  IAU: 7
I= 7 INL(i)= 2 INU(i)= 2
  IAL: 3
  IAU: 8
I= 8 INL(i)= 2 INU(i)= 1
  IAL: 4
  IAU: 12
I= 9 INL(i)= 1 INU(i)= 2
  IAL: 5
  IAU: 10
I= 10 INL(i)= 2 INU(i)= 2
  IAL: 6
  IAU: 11
I= 11 INL(i)= 2 INU(i)= 2
  IAL: 7
  IAU: 12
I= 12 INL(i)= 2 INU(i)= 1
  IAL: 8
  IAU: 16
I= 13 INL(i)= 1 INU(i)= 1
  IAL: 9
  IAU: 14
  
```

```

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

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
  
```

# INPUT=3: MC, 5-Colors

color.log: matrix info.

13	14	15	16
9	10	11	12
5	<u>6</u>	<u>7</u>	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:
I= 2      INL(i)= 0      INU(i)= 3
  IAL:
  IAU:
I= 3      INL(i)= 0      INU(i)= 4
  IAL:
  IAU:
I= 4      INL(i)= 0      INU(i)= 3
  IAL:
  IAU:
I= 5      INL(i)= 0      INU(i)= 3
  IAL:
  IAU:
I= 6      INL(i)= 3      INU(i)= 0
  IAL:
  IAU:
I= 7      INL(i)= 2      INU(i)= 0
  IAL:
  IAU:
I= 8      INL(i)= 3      INU(i)= 0
  IAL:
  IAU:
I= 9      INL(i)= 3      INU(i)= 1
  IAL:
  IAU:
I= 10     INL(i)= 2      INU(i)= 2
  IAL:
  IAU:
I= 11     INL(i)= 2      INU(i)= 2
  IAL:
  IAU:
I= 12     INL(i)= 1      INU(i)= 1
  IAL:
  IAU:
I= 13     INL(i)= 0      INU(i)= 2
  IAL:
  IAU:

```

```

I= 14     INL(i)= 3      INU(i)= 0
  IAL:
  IAU:
I= 15     INL(i)= 2      INU(i)= 1
  IAL:
  IAU:
I= 16     INL(i)= 3      INU(i)= 0
  IAL:
  IAU:

```

# 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;
}

```

# 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.
     */

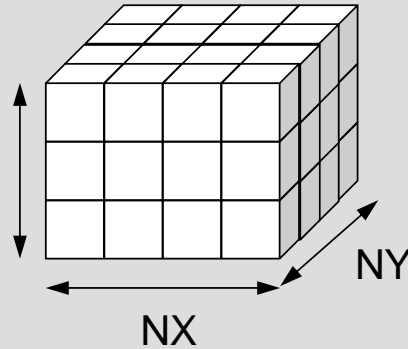
    ICELTOT = NX * NY * NZ;

    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:**

Number of meshes in x/y/z directions

**NXP1, NYP1, NZP1:**

Number of nodes in x/y/z directions  
(for visualization)

**ICELTOT:**

Number of meshes (NX x NY x NZ)

**XYZ [ICELTOT] [3]:**

Location of meshes

**NEIBcell [ICELTOT] [6]:**

Neighboring meshes

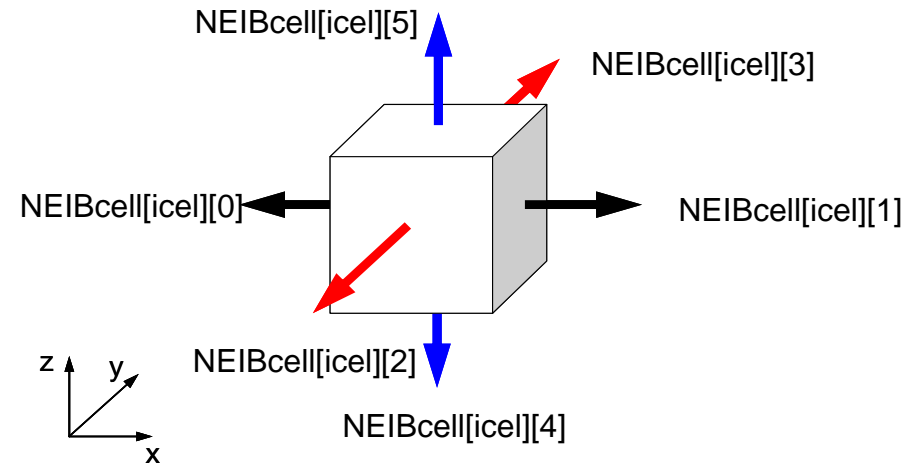
# pointer\_init (2/3)

```

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;
    }
  }
}

```



**$i = \text{XYZ}[\text{icel}][0]$**   
 **$j = \text{XYZ}[\text{icel}][1], k = \text{XYZ}[\text{icel}][2]$**   
 **$\text{icel} = k * \text{NX} * \text{NY} + j * \text{NX} + i$**

**“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

**$\text{NEIBcell}[\text{icel}][0] = \text{icel} - 1 \quad + 1$**   
 **$\text{NEIBcell}[\text{icel}][1] = \text{icel} + 1 \quad + 1$**   
 **$\text{NEIBcell}[\text{icel}][2] = \text{icel} - \text{NX} \quad + 1$**   
 **$\text{NEIBcell}[\text{icel}][3] = \text{icel} + \text{NX} \quad + 1$**   
 **$\text{NEIBcell}[\text{icel}][4] = \text{icel} - \text{NX} * \text{NY} + 1$**   
 **$\text{NEIBcell}[\text{icel}][5] = \text{icel} + \text{NX} * \text{NY} + 1$**



# 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 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];

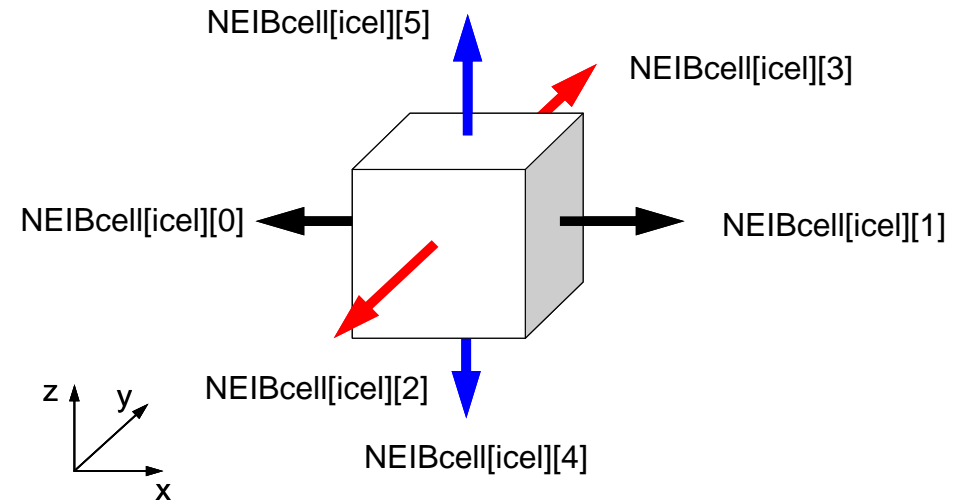
    if(icN5 != 0) {
        icou = INL[icel] + 1;
        IAL[icel][icou-1] = icN5;
        INL[icel] = icou;
    }

    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] = icN1;
        INL[icel] = icou;
    }
}

```

## poi\_gen (2/4)



### Lower Triangular Part

```

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

```
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];
```

```
.....
```

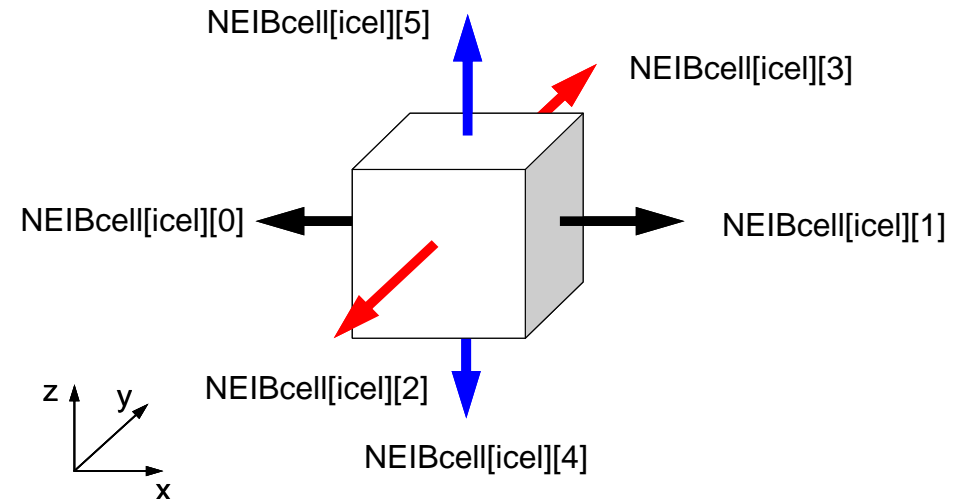
```
if(icN2 != 0) {
    icou = INU[icel] + 1;
    IAU[icel][icou-1] = icN2;
    INU[icel]          = icou;
}
```

```
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;
}
```

```
}
```

## poi\_gen (3/4)



### Upper Triangular Part

```
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)

Reading “initial” color  
number

```

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)

Allocate matrices

```

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;
}

```

```

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)

**INL, INU, IAL, IAU**

**OLDtoNEW, NEWtoOLD**

**NCOLORTot**

**COLORindex[NCOLORTot+1]**

Info. after renumbering

Reference table before/after renumbring

Final number of colors (g.e. initial number)

**Meshes from `COLORindex[ic]` to `COLORindex[ic+1]-1` are in (ic+1)-th color.**

Meshes in same color are independent:

Parallel processing can be applied.

**(ic)-th color: if color ID starts at 0**

**(ic+1)-th color: if color ID starts at 1**

**In the program, color ID starts at 1 !!**



`COLORindex[NCOLORtot+1]` Meshes from `COLORindex[ic]` to `COLORindex[ic+1]-1` are in (ic+1)-th color.  
 Meshes in same color are independent:  
 Parallel processing can be applied.

**(ic)-th color: if color ID starts at 0**  
**(ic+1)-th color: if color ID starts at 1**  
**In the program, color ID starts at 1 !!**

```
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);
  }
}
```

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

# COLORindex

# 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] = 0;
NEWtoOLD[0] = NODmin;

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

ITEMcou = N / NCOLORk;

```

## IW:

Work array

“Color ID” of each mesh

**IW=0** at initial stage

## NODmin:

ID of the mesh with minimum  
value of “degree”

```

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[    ] = NODmin+1;

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

ITEMcou = N / NCOLORk;

```

## mc (2/8)

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

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

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

```

**ITEMcou = N / NCOLORk;**

**ITEMcou = N / NCOLORk :**

(Maximum) number of meshes in  
each color

# 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
```

$$16/3=5 = \underline{\text{ITEMcou}}$$

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)

## Initialization of Counters

```


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

## mc (3/8)

```

icou = 1;
icouK = 1;
for(icol=0; icol<N; icol++) { Loop on Colors
    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:

```

NCOLORk :

Current number of colors

IW[i]=0 :

If i-th mesh (original numbering)  
is not colored.

```

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:

```

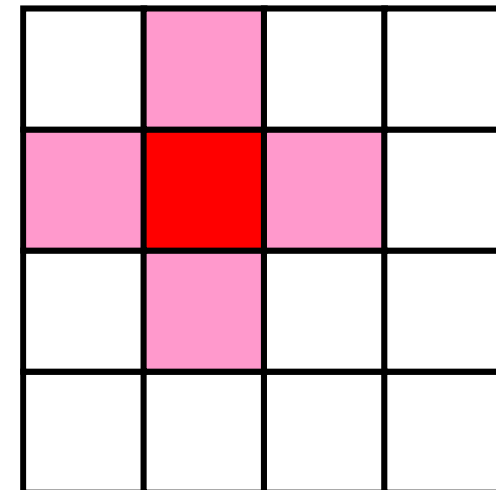
## Loop on Colors

# mc (3/8)

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;
            icoug++;
            icoug++;
        }
    }
    COLORindex[ic] = icou;
}

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

```

Renumber meshes in ascending orders according to color ID.

**OLDtoNEW[OldID] = NewID+1**

**NEWtoOLD[NewID] = OldID+1**

**starting@”1”**

**COLODindex[ic]:**

At this stage, number of meshes in each color (ic+1) is stored.

# 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.

```

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];
    }
}

```

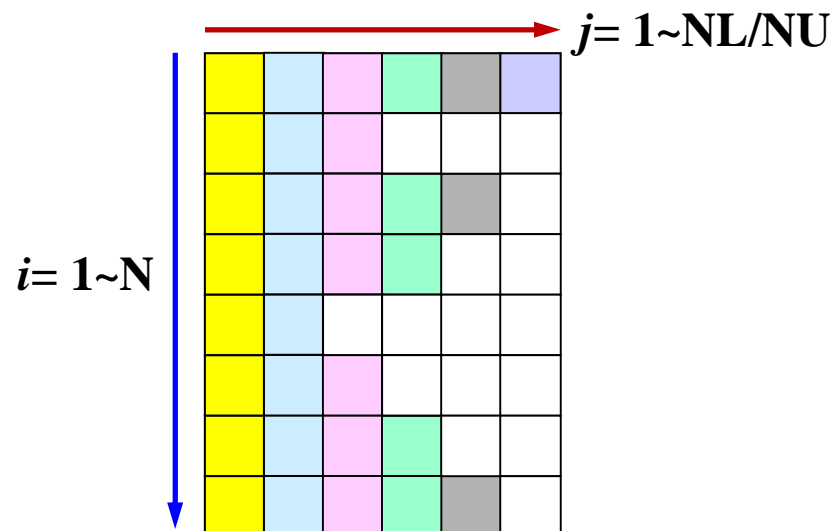
# mc (6/8)

## Arrays for Work

- 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)

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。

```

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];
        }
    }
}

```

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

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

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

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

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);

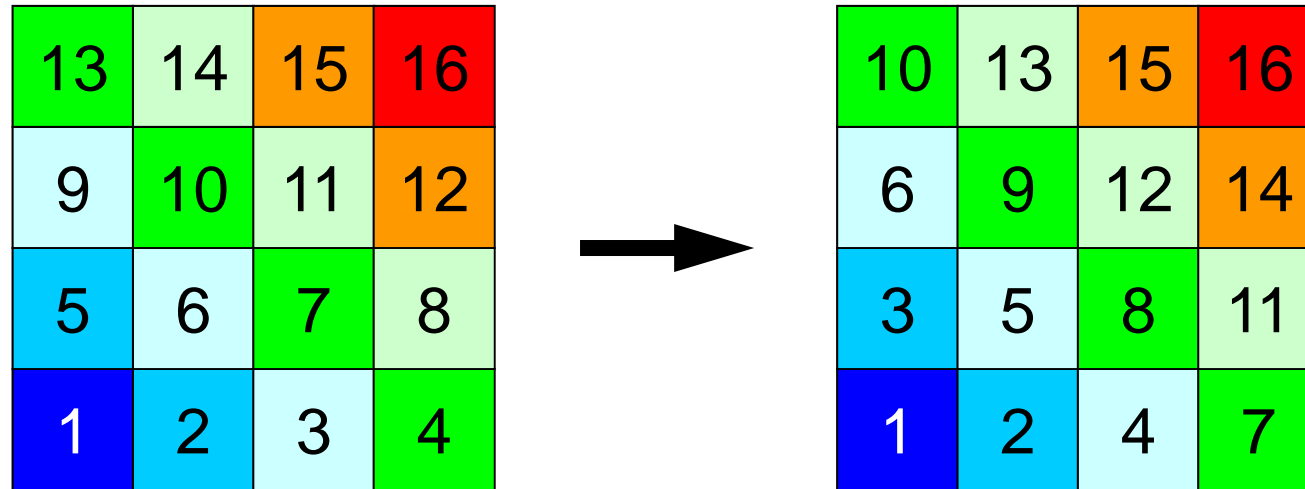
return 0;
}

```

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

- INL
- INU

# 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 ( $k (k \geq 2)$ )」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

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



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

# 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:

ID of the mesh with minimum value of "degree"

# 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: Global Counter

icou : Intra-Level Counter

**Loop on Levels**

# 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**: Global Counter  
**icou** : Intra-Level Counter

## Loops for Each Element

If ( $i_{n-1}$ )<sup>th</sup> mesh is adjacent to  $i$ <sup>th</sup> mesh where  $IW[1][i] = icol - 1$ , and level of ( $i_{n-1}$ )<sup>th</sup> mesh is not finalized, ( $i_{n-1}$ )<sup>th</sup> mesh could be a candidate for meshes in  $icol$ <sup>th</sup> level.

- $IW[1][in-1] = -(icol+1)$
- $icou = icou + 1$
- $IW[0][icou-1] = in$ : This array indicates “when” this mesh was found, and nominated as a candidate.

mesh, color: starting from 0

$IAL[:,:]$ : starting from 1

# What does it mean ?

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

`icol=4-1`

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

**icouG**: Global Counter

**icou** : Intra-Level Counter

## Loops for Each Element

If ( $i_{n-1}$ )<sup>th</sup> mesh is adjacent to  $i$ <sup>th</sup> mesh where `IW[1][i]=icol-1`, and level of ( $i_{n-1}$ )<sup>th</sup> mesh is not finalized, ( $i_{n-1}$ )<sup>th</sup> mesh could be a candidate for meshes in  $icol$ <sup>th</sup> level.

- `IW[1][in-1]= -(icol+1)`
- `icou= icou + 1`
- `IW[0][icou-1]= in`: This array indicates “when” this mesh was found, and nominated as a candidate.

mesh, color: starting from 0

`IAL[:,:]`: starting from 1

# What does it mean ?

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

`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: Global Counter

icou : Intra-Level Counter

## Loops for Each Element

If ( $i_{n-1}$ )<sup>th</sup> mesh is adjacent to  $i$ <sup>th</sup> mesh where `IW[1][i]=icol-1`, and level of ( $i_{n-1}$ )<sup>th</sup> mesh is not finalized, ( $i_{n-1}$ )<sup>th</sup> mesh could be a candidate for meshes in  $icol$ <sup>th</sup> level.

- `IW[1][ $i_{n-1}$ ]= -(icol+1)`
- `icou= icou + 1`
- `IW[0][icou-1]=  $i_{n-1}$` : This array indicates “when” this mesh was found, and nominated as a candidate.

mesh, color: starting from 0

`IAL[:,:]`: starting from 1

# 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**: Global Counter  
**icou** : Intra-Level Counter

## Loops for Each Element

If **icou=0**, a mesh, which satisfies the following conditions, is the candidate (usually, this does not happen):

- Level of this mesh is not finalized.
- Mesh ID according to the initial numbering is the smallest.

```

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;
}

```

## cm (4/5)

Candidates for icol<sup>th</sup> level are stored in:

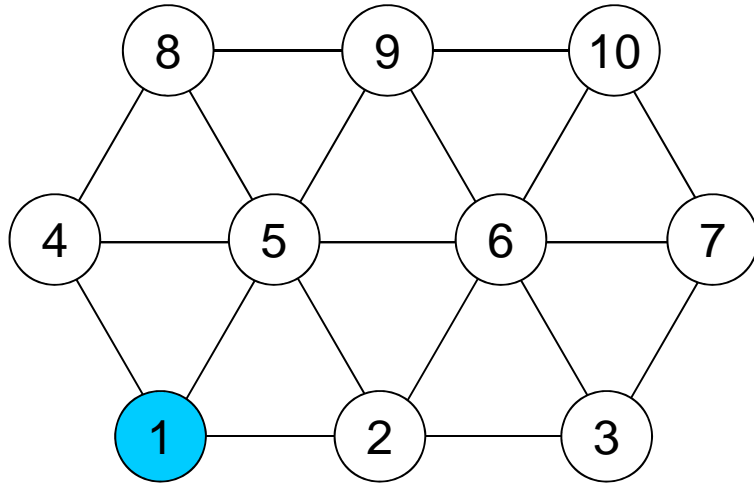
- $IW[0][ic]$ ,  $ic = 0 \sim icou$

Remove such meshes that are adjacent to other candidates, **because neighboring meshes cannot belong to same level.**

If we have such removed mesh in-1, apply the following operations:

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

# What does it mean ?



e.g.  
Mesh (1) belongs to  $(ico1-1)$ <sup>th</sup>  
level

Candidates for  $ico1$ <sup>th</sup> level are stored in:

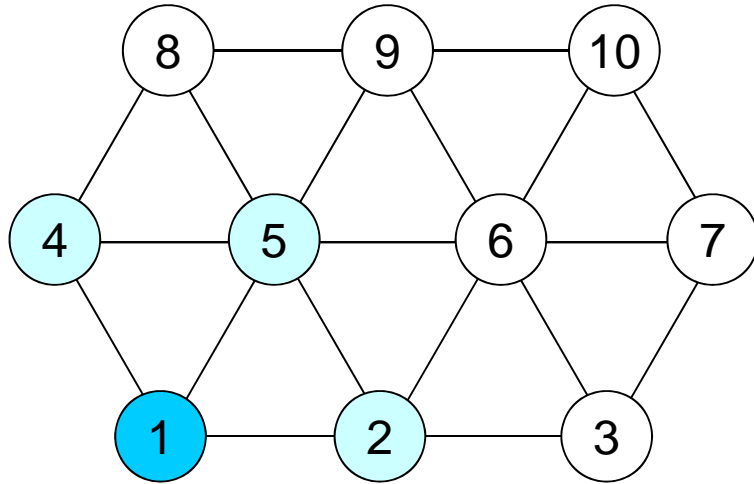
- $IW[0][ic]$ ,  $ic = 0 \sim icou$

Remove such meshes that are adjacent to other candidates, **because neighboring meshes cannot belong to same level.**

If we have such removed mesh  $in-1$ , apply the following operations:

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

# What does it mean ?



(2),(4) and (5) are candidates for (icol)<sup>th</sup> level

$$IW[1][2-1] = -(icol+1)$$

$$IW[1][4-1] = -(icol+1)$$

$$IW[1][5-1] = -(icol+1)$$

$$IW[1][0] = 2$$

$$IW[1][1] = 4$$

$$IW[1][2] = 5$$

Candidates for icol<sup>th</sup> level are stored in:

- $IW[0][ic], ic = 0 \sim icou$

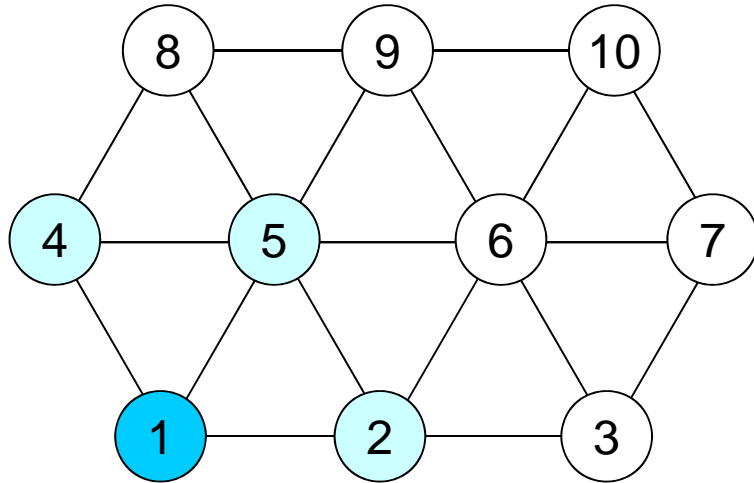
Remove such meshes that are adjacent to other candidates, **because neighboring meshes cannot belong to same level.**

If we have such removed mesh in-1, apply the following operations:

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



# What does it mean ?



Considering dependency:

$$IW[1][2-1] = -(icol+1)$$

$$IW[1][4-1] = -(icol+1)$$

$$IW[1][5-1] = 0$$

$$IW[1][0] = 2$$

$$IW[1][1] = 4$$

$$IW[1][2] = 5$$

Candidates for icol<sup>th</sup> level are stored in:

- $IW[0][ic], ic = 0 \sim icou$

Remove such meshes that are adjacent to other candidates,

**because neighboring meshes cannot belong to same level.**

If we have such removed mesh in-1, apply the following operations:

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

# 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: Global Counter

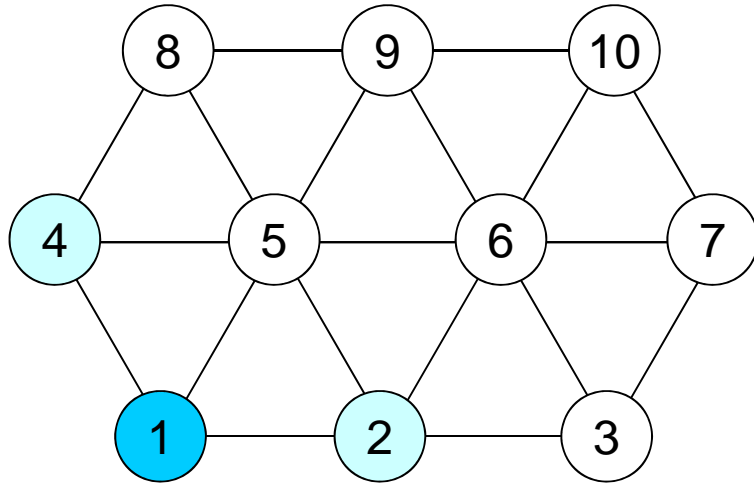
icou : Intra-Level Counter

Finally, meshes which satisfies  $IW[1][inC] = -(icol+1)$ , belong to icol<sup>th</sup> level.

For such meshes, apply  $IW[1][inC] = icol + 1$ .

Finally,  $icouG = icouG + 1$

# What does it mean ?



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:

$$IW[1][2-1] = icol$$

$$IW[1][4-1] = icol$$

Finally, meshes which satisfies  $IW[1][inC] = -(icol+1)$ , belong to  $icol^{th}$  level.

For such meshes, apply  $IW[1][inC] = icol + 1$ .

Finally,  $icouG = icouG + 1$

# 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: Global Counter  
icou : Intra-Level Counter

if icouG=N (ICELTOT):

- All meshes are colored (completed).

Otherwise, proceed to the next level.

# 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 *
*****/

```

**NCOLORTot = NCOLORk :**

Final number of colors.

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

Renumber meshes in ascending orders according to level ID.

**OLDtoNEW[OldID] = NewID+1**  
**NEWtoOLD[NewID] = OldID+1**

**COLODindex[ic] :**

At this stage, number of meshes in each level (ic+1) is stored.

# 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 *
*****/

```

COLORindex[ic]:

Now it is 1D index.

# Comparison between MC & RCM

- MC
  - Good parallel performance & load balancing
  - Many iterations with smaller number of colors
  - Although larger number of colors provides faster convergence, performance is not good due to synchronization overhead.
- RCM
  - Faster convergence than MC.
  - Effect of synch. may be significant. Not good for many threads.
  - Problems in load balancing.
- **Convenient method needed**
  - **Fast convergence, Low overhead**
  - **Smaller number of colors, Fast convergence**

# More Colors: Synch. Overhead

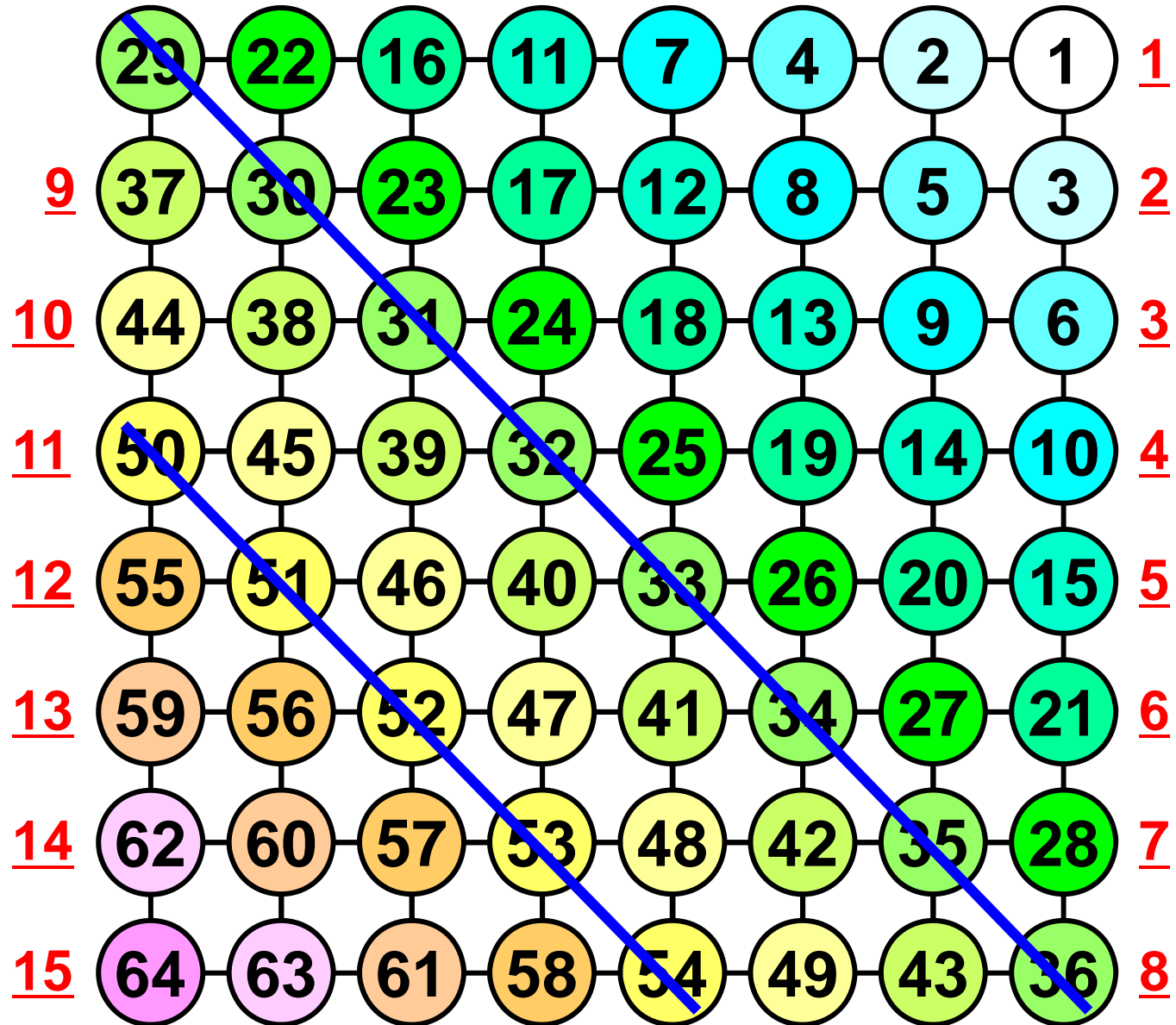
```
for(ic=0; ic<NCOLORtot; ic++) {  
#pragma omp parallel for private (i,WVAL) ←  
    for(i=COLORindex[ic]; i<COLORindex[ic+1]; i++) {  
        WVAL = W[Z][i];  
        for(j=indexL[i]; j<indexL[i+1]; j++) {  
            WVAL -= AL[j] * W[Z][itemL[j]-1];  
        }  
        W[Z][i] = WVAL * W[DD][i];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



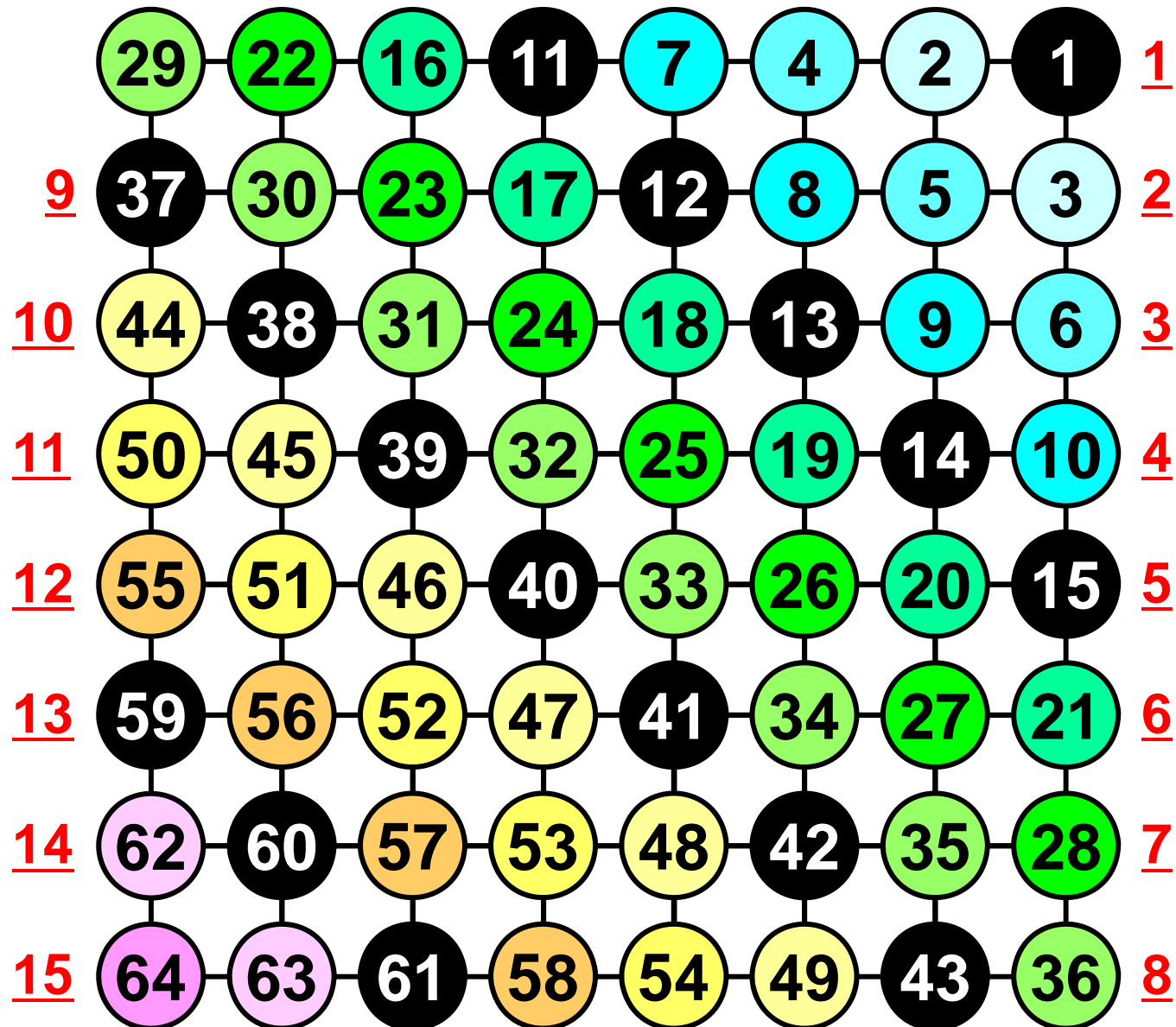
# Remedy: CM-RCM

- RCM + Cyclic Multicoloring [Doi, Osoda, Washio]
- Procedures
  - Apply RCM
  - Define “Nc” (Color number of Cyclic Multicoloring (CM))
    - 1<sup>st</sup>-Color in CM-RCM: 1<sup>st</sup>, (Nc+1)<sup>th</sup>, (2Nc+1)<sup>th</sup> ... levels in RCM
    - 2<sup>nd</sup>-Color in CM-RCM: 2<sup>nd</sup>, (Nc+2)<sup>th</sup>, (2Nc+2)<sup>th</sup> ... levels in RCM
    - k<sup>th</sup>-Color in CM-RCM: k<sup>th</sup>, (Nc+k)<sup>th</sup>, (2Nc+k)<sup>th</sup> ... levels in RCM
    - Each level of RCM is colored in cyclic manner (cycle=Nc).
  - If “k” reaches “Nc”, and all levels of RCM are colored, it’s completed.
    - Renumber meshes in ascending orders according to “Color” ID.
  - If dependency between levels in same color, start from the beginning of the cyclic multicoloring with  $Nc=Nc+1$ .

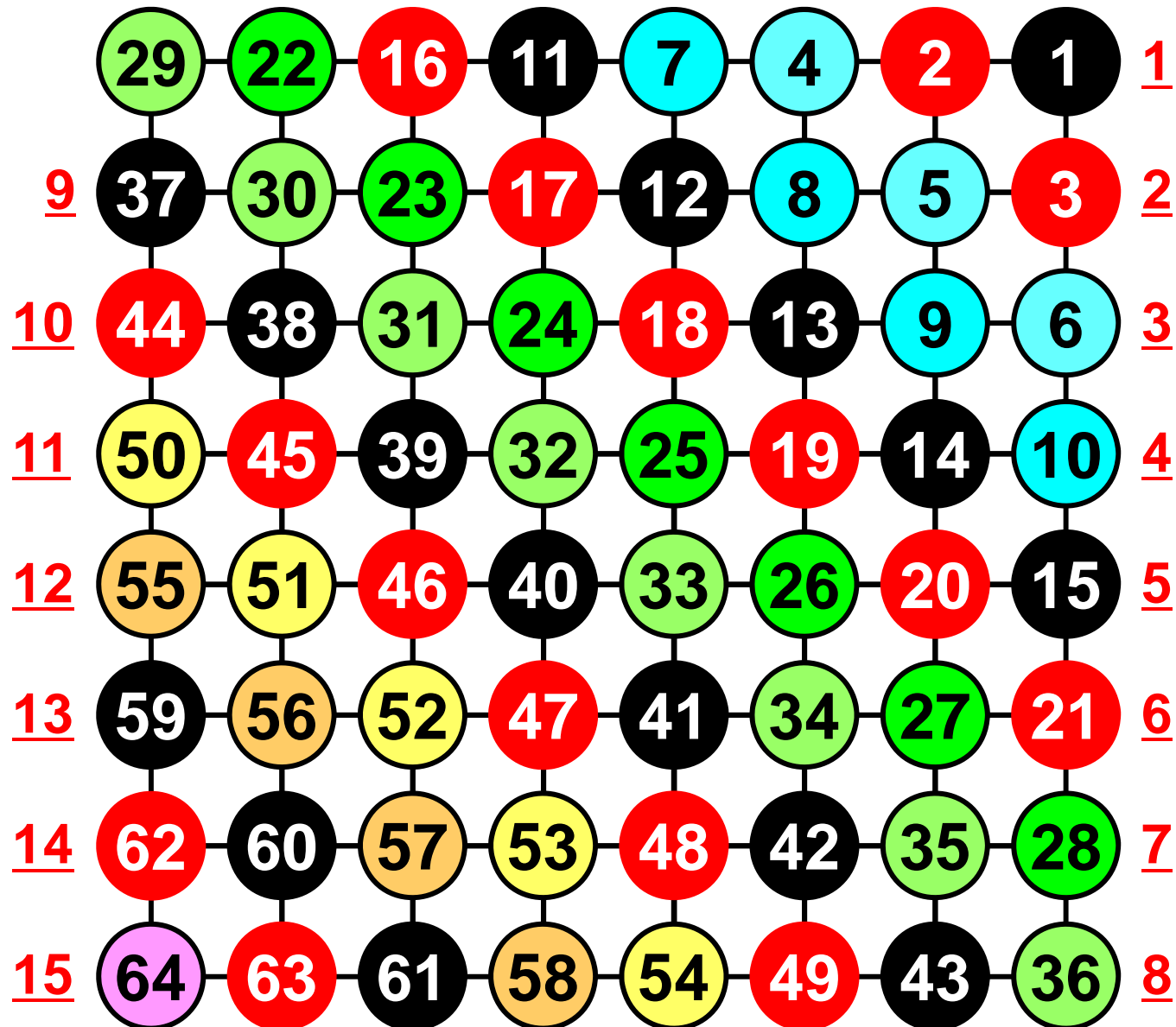
# RCM



$N_c=4$ ,  $k=1$ , Level: 1,5,9,13

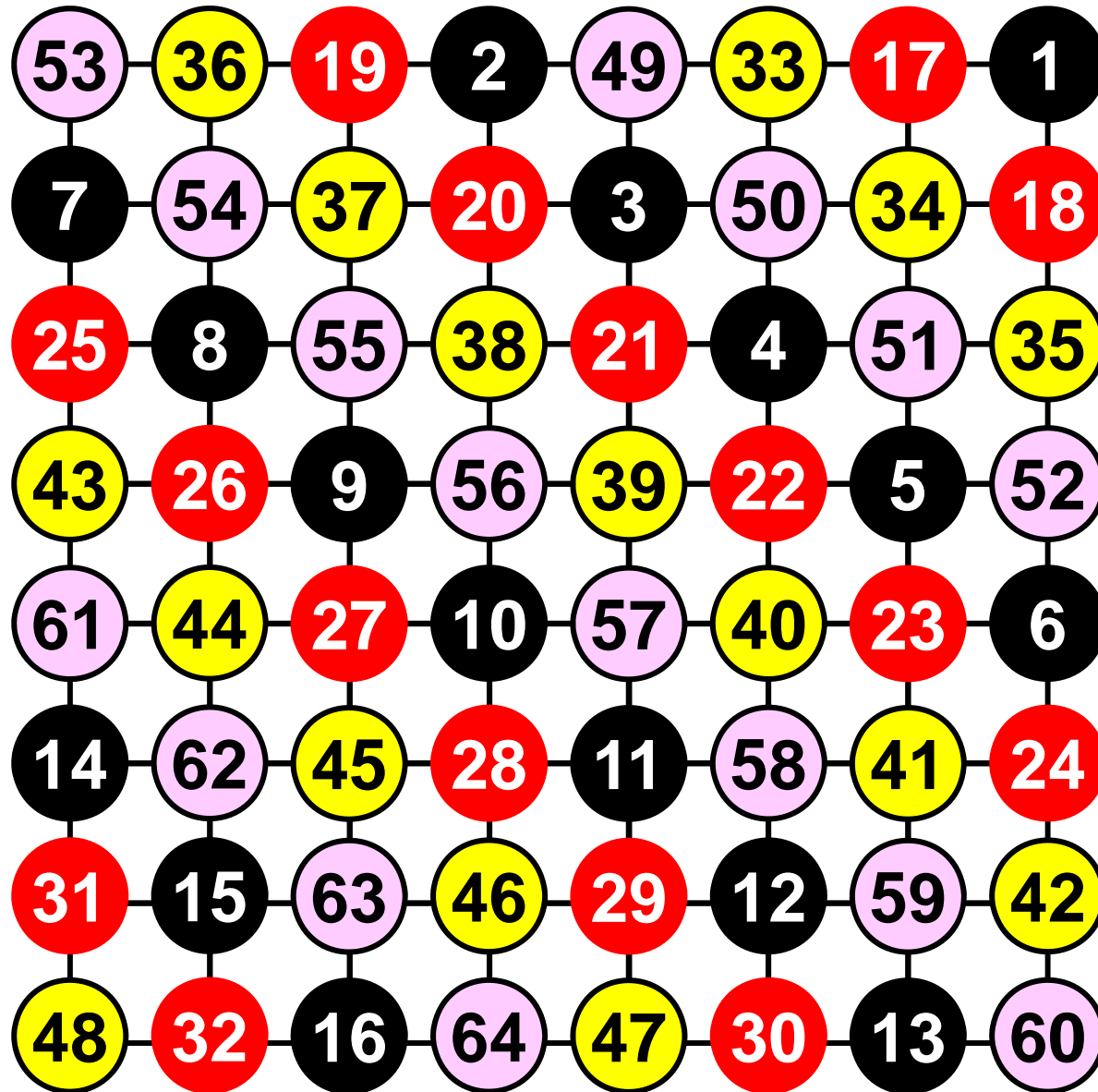


$N_c=4$ ,  $k=2$ , Level: 2,6,10,14



# CM-RCM( $N_c=4$ ): Renumbering

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

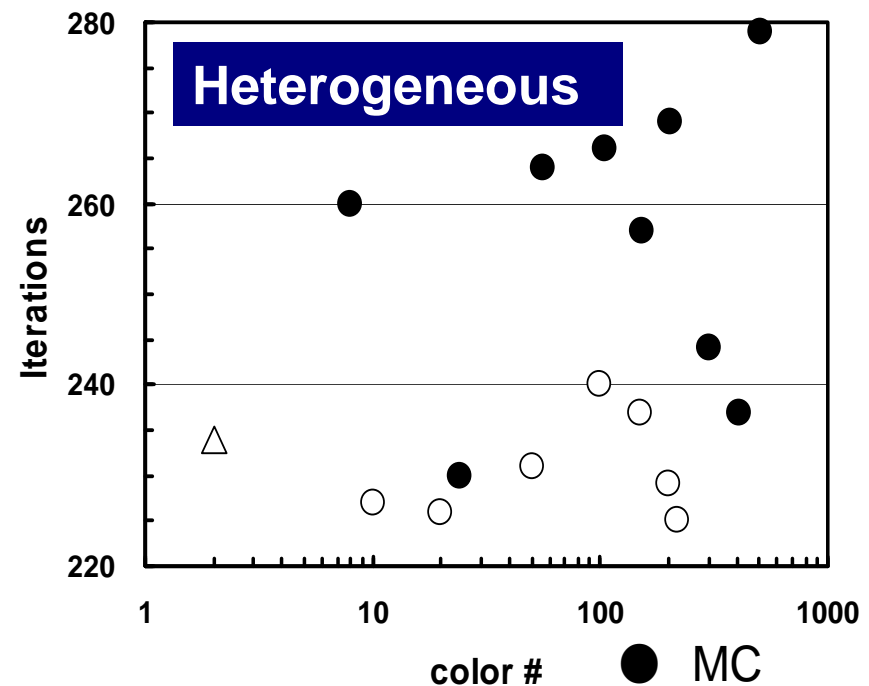
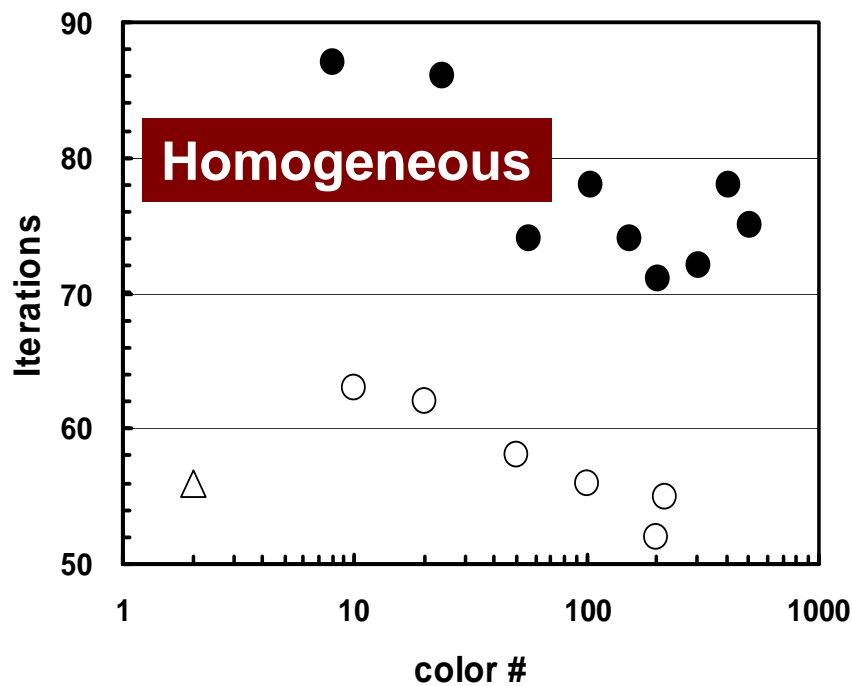


# CM-RCM

- How to run
  - "NCOLOrtot=-Nc" in INPUT.DAT
  - Already implemented in L2
- cmrcm.f, cmrcm.c

# 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



3D Linear-Elastic Problems with 32,768 DOF

- MC
- CM-RCM
- △ No reordering

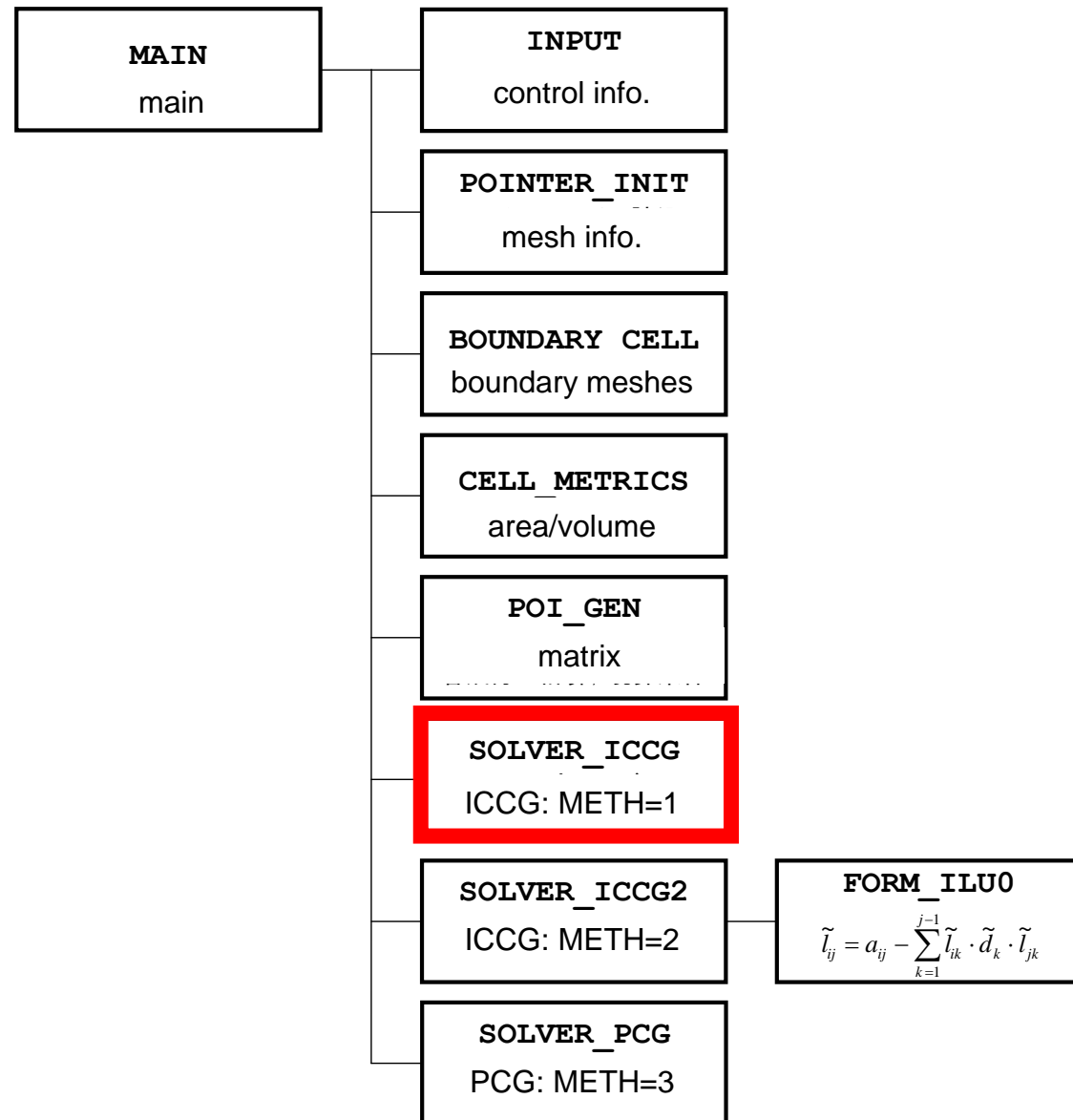
- データ依存性の解決策は?
- オーダリング (Ordering) について
  - Red-Black, Multicolor (MC)
  - Cuthill-McKee (CM), Reverse-CM (RCM)
  - オーダリングと収束の関係
- オーダリングの実装
- **オーダリング付ICCG法の実装**
- マルチコアへの実装 (OpenMP) へ向けて



# 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



# 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)

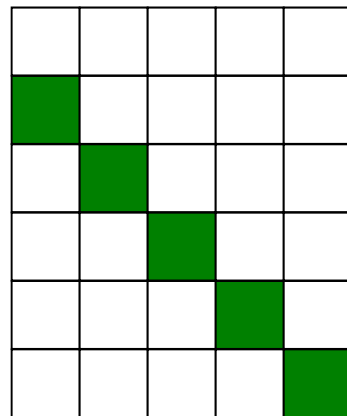
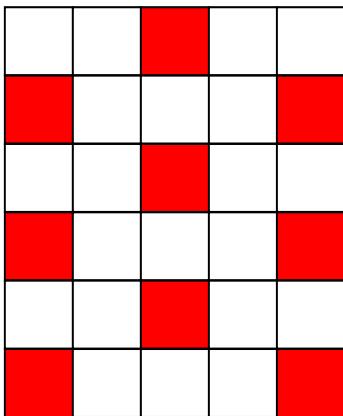
Forward Substitution

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

```

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。

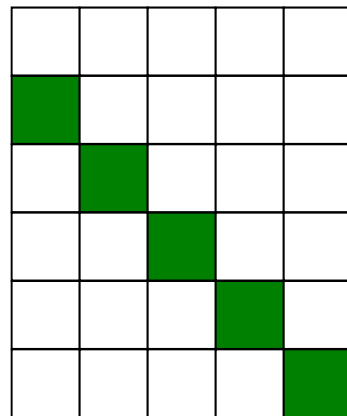
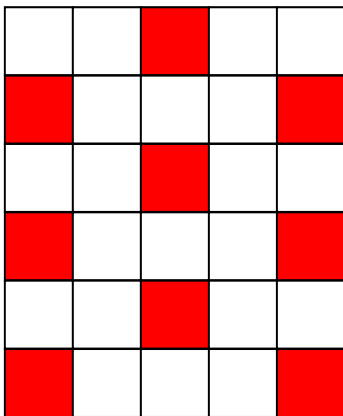
⇒データ依存性回避可能

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

Forward Substitution

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

```
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];
  }
}
```



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
```

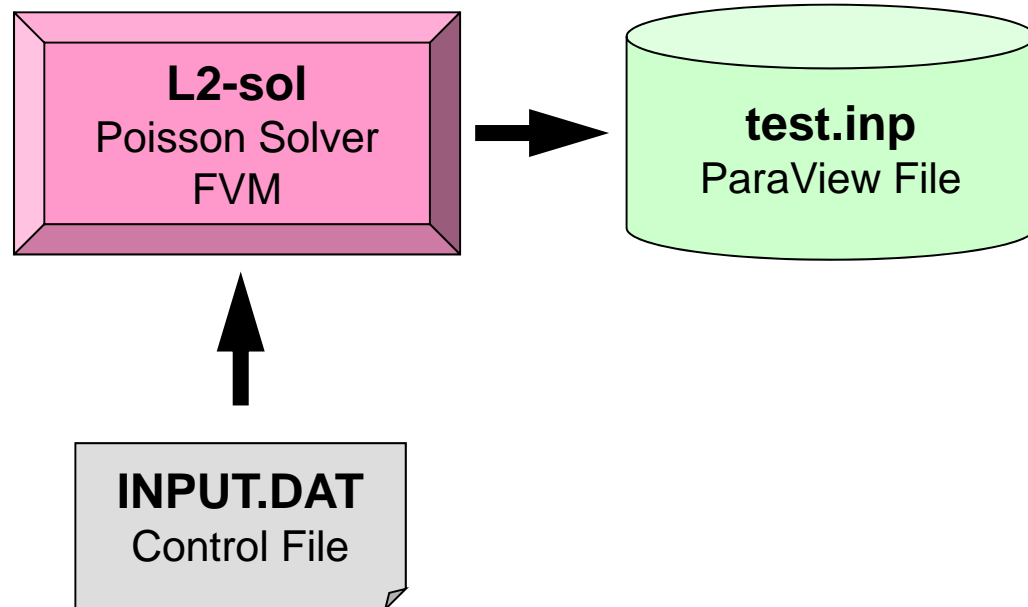
```
$> make
```

```
$> 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

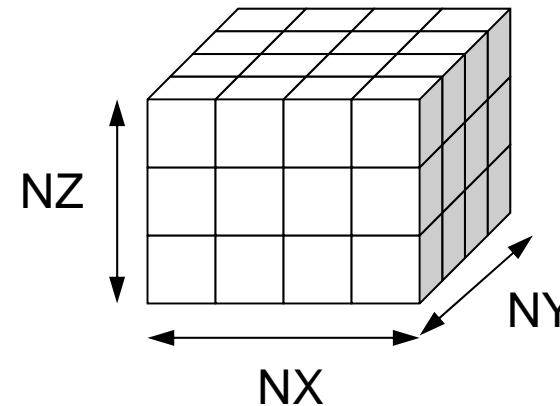
```
1.00e-00 1.00e-00 1.00e-00
```

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
```

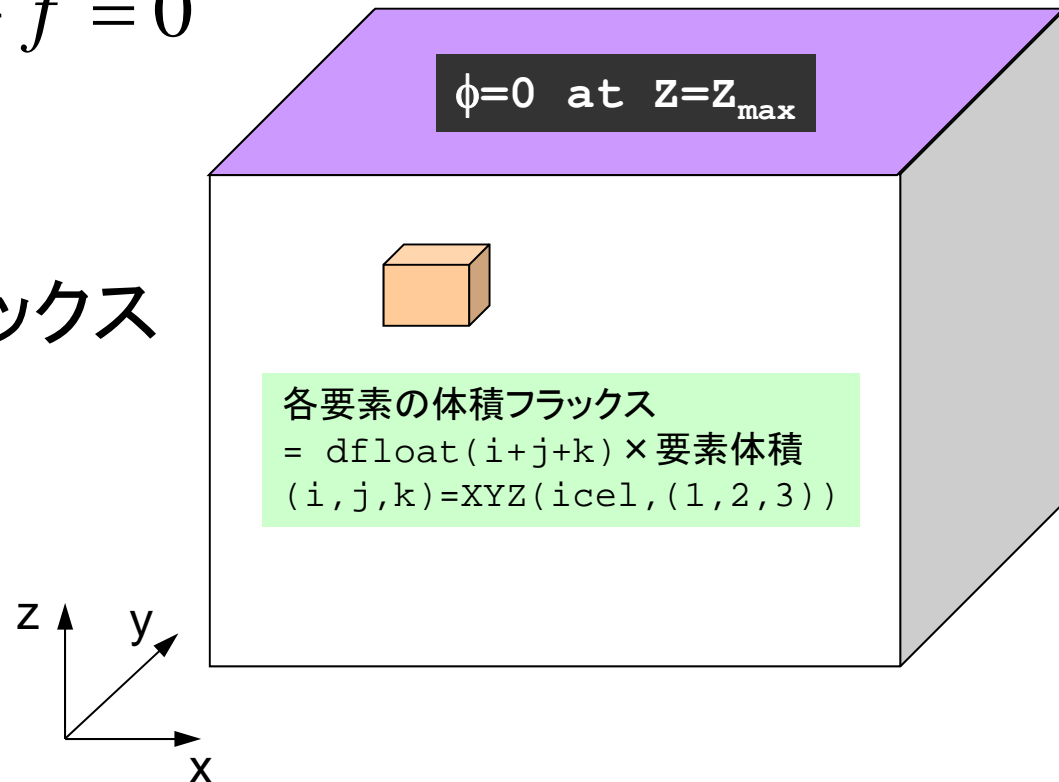
# 解いている問題：三次元ポアソン方程式

ポアソン方程式

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

境界条件

- 各要素で体積フラックス
- $Z = Z_{\max}$  面で  $\phi = 0$



# 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;

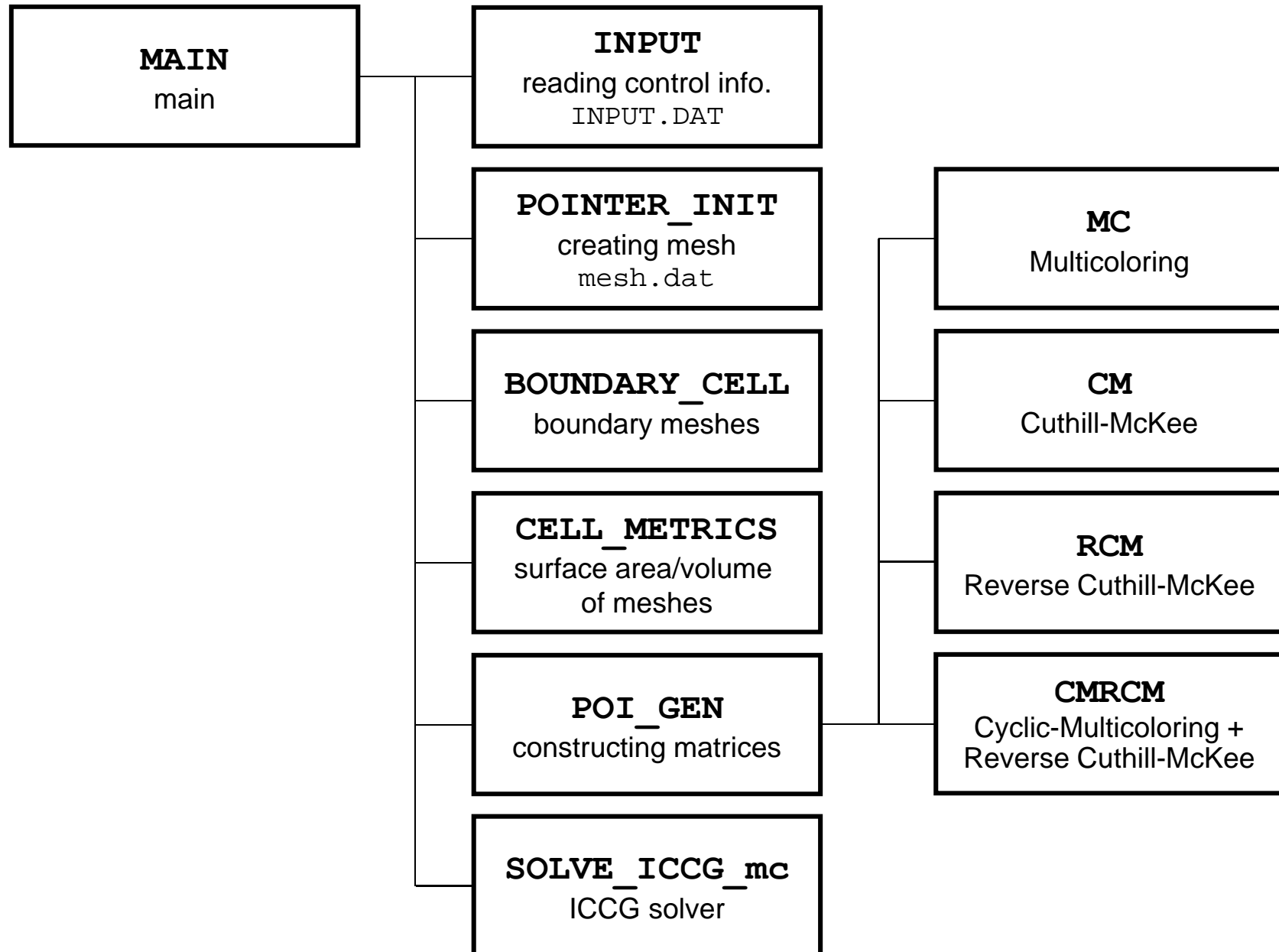
    ISET = 0;
    WK = (double *)malloc(sizeof(double)*ICELTOT);
    if(WK == NULL) {
        fprintf(stderr, "Error: %s\n", strerror(errno));
        goto error;}
    if(solve_ICCG_mc(ICELTOT, NL, NU, indexL, itemL, indexU, itemU,
                    D, BFORCE, PHI, AL, AU, NCOLORTot, COLORindex,
     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

# Structure of L2-sol



# Variables/Arrays for Matrix (1/2)

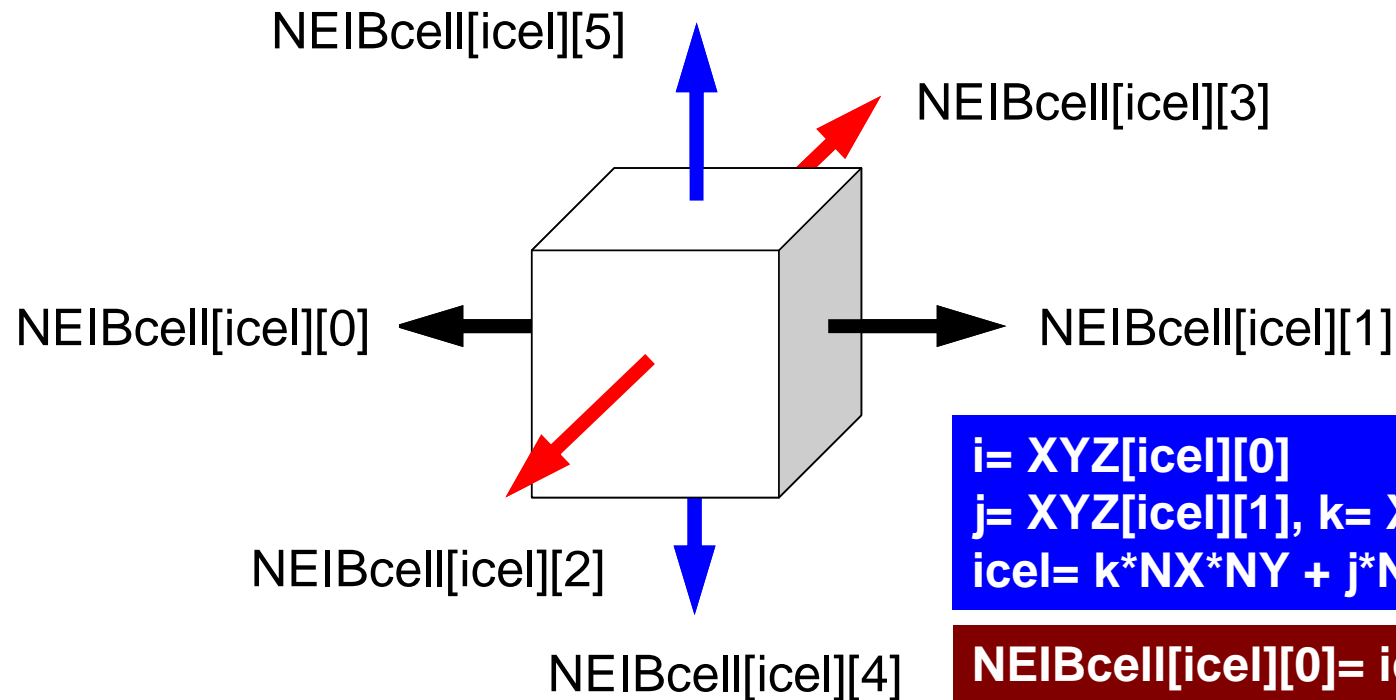
Name	Type	Content
<b>D [N]</b>	<b>R</b>	Diagonal components of the matrix (N= ICELTOT)
<b>BFORCE [N]</b>	<b>R</b>	RHS vector
<b>PHI [N]</b>	<b>R</b>	Unknown vector
<b>indexL [N+2]</b> <b>indexU [N+2]</b>	<b>I</b>	# of L/U non-zero off-diag. comp. (CRS)
<b>NPL, NPU</b>	<b>I</b>	Total # of L/U non-zero off-diag. comp. (CRS)
<b>itemL [NPL]</b> <b>itemU [NPU]</b>	<b>I</b>	Column ID of L/U non-zero off-diag. comp. (CRS)
<b>AL [NPL]</b> <b>AU [NPU]</b>	<b>R</b>	L/U non-zero off-diag. comp. (CRS)

Name	Type	Content
<b>NL, NU</b>	<b>I</b>	MAX. # of L/U non-zero off-diag. comp. for each mesh (=6)
<b>INL [N]</b> <b>INU [N]</b>	<b>I</b>	# of L/U non-zero off-diag. comp.
<b>IAL [N] [NL]</b> <b>IAU [N] [NU]</b>	<b>I</b>	Column ID of L/U non-zero off-diag. comp.

# Variables/Arrays for Matrix (2/2)

Name	Type	Content
<b>NCOLORtot</b>	<b>I</b>	<p><b>Input:</b> reordering method + initial number of colors/levels <math>\geq 2</math>: MC, =0: CM, =-1: RCM, <math>-2 \leq</math>: CMRCM</p> <p><b>Output:</b> Final number of colors/levels</p>
<b>COLORindex</b> <b>[NCOLORtot+1]</b>	<b>I</b>	<p>Number of meshes at each color/level 1D compressed array Meshes in <math>icol^{th}</math> color/level are stored in this array from <b>COLORindex[icol]</b> to <b>COLORindex[icol+1]-1</b></p>
<b>NEWtoOLD [N]</b>	<b>I</b>	Reference array from New to Old numbering
<b>OLDtoNEW [N]</b>	<b>I</b>	Reference array from Old to New numbering

# NEIBcell: ID of Neighboring Mesh/Cell =0: for Boundary Surface



$i = \text{XYZ}[\text{icel}][0]$   
 $j = \text{XYZ}[\text{icel}][1], k = \text{XYZ}[\text{icel}][2]$   
 $\text{icel} = k * \text{NX} * \text{NY} + j * \text{NX} + i$

$\text{NEIBcell}[\text{icel}][0] = \text{icel} - 1 \quad + 1$   
 $\text{NEIBcell}[\text{icel}][1] = \text{icel} + 1 \quad + 1$   
 $\text{NEIBcell}[\text{icel}][2] = \text{icel} - \text{NX} \quad + 1$   
 $\text{NEIBcell}[\text{icel}][3] = \text{icel} + \text{NX} \quad + 1$   
 $\text{NEIBcell}[\text{icel}][4] = \text{icel} - \text{NX} * \text{NY} + 1$   
 $\text{NEIBcell}[\text{icel}][5] = \text{icel} + \text{NX} * \text{NY} + 1$

# 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;
}

```



## poi\_gen (1/8)

```

#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);
  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 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;
}

```

allocate.c

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

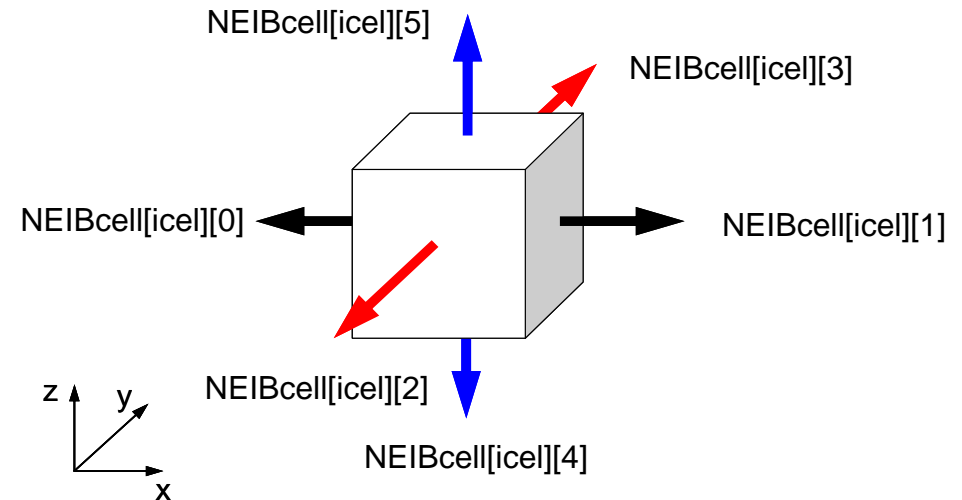
```
  icN1 = NEIBcell[icel][0];
  icN2 = NEIBcell[icel][1];
  icN3 = NEIBcell[icel][2];
  icN4 = NEIBcell[icel][3];
  icN5 = NEIBcell[icel][4];
  icN6 = NEIBcell[icel][5];
```

```
  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] = icN1;
    INL[icel] = icou;
  }
```

## poi\_gen (2/8)



### Lower Triangular Part

```
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

```
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];
```

```
.....
```

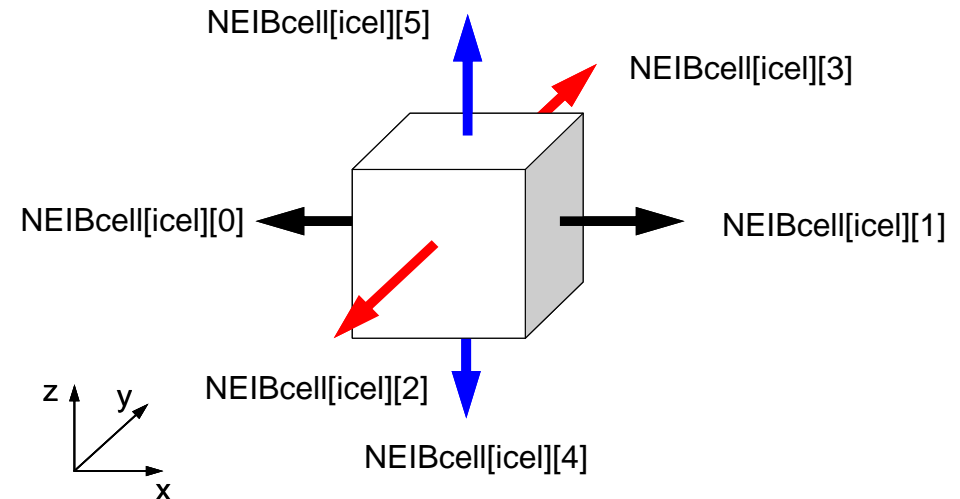
```
if(icN2 != 0) {
    icou = INU[icel] + 1;
    IAU[icel][icou-1] = icN2;
    INU[icel]          = icou;
}
```

```
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;
}
```

```
}
```

# poi\_gen (3/8)



## Upper Triangular Part

```
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)

## New numbering is applied after this point

```

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);

```

**“itemL” / “itemU”  
start at 1**

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

Name	Type	Content
D [N]	R	Diagonal components of the matrix (N= ICELTOT)
BFORCE [N]	R	RHS vector
PHI [N]	R	Unknown vector
indexL [N+1] indexU [N+1]	I	# of L/U non-zero off-diag. comp. (CRS)
NPL, NPU	I	Total # of L/U non-zero off-diag. comp. (CRS)
itemL [NPL] itemU [NPU]	I	Column ID of L/U non-zero off-diag. comp. (CRS)
AL [NPL] AU [NPU]	R	L/U non-zero off-diag. comp. (CRS)

```

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];

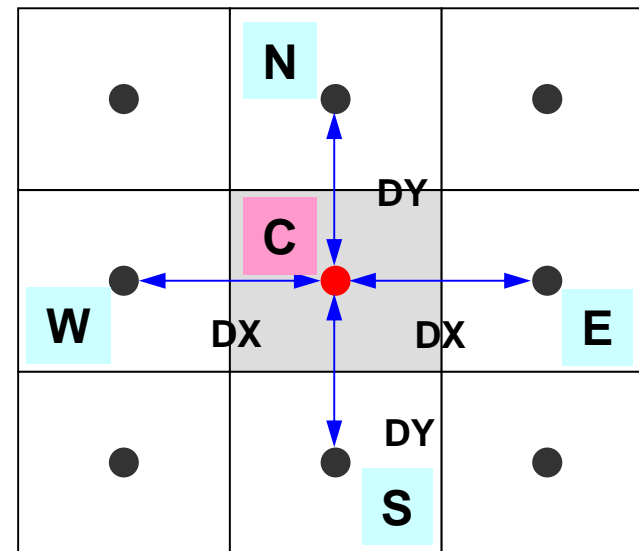
  if(icN5 != 0) {
    icN5 = OLDtoNEW[icN5-1];
    coef = RDZ * ZAREA;
    D[icel] -= coef;

    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;
        }
      }
    }
  }
}
...

```

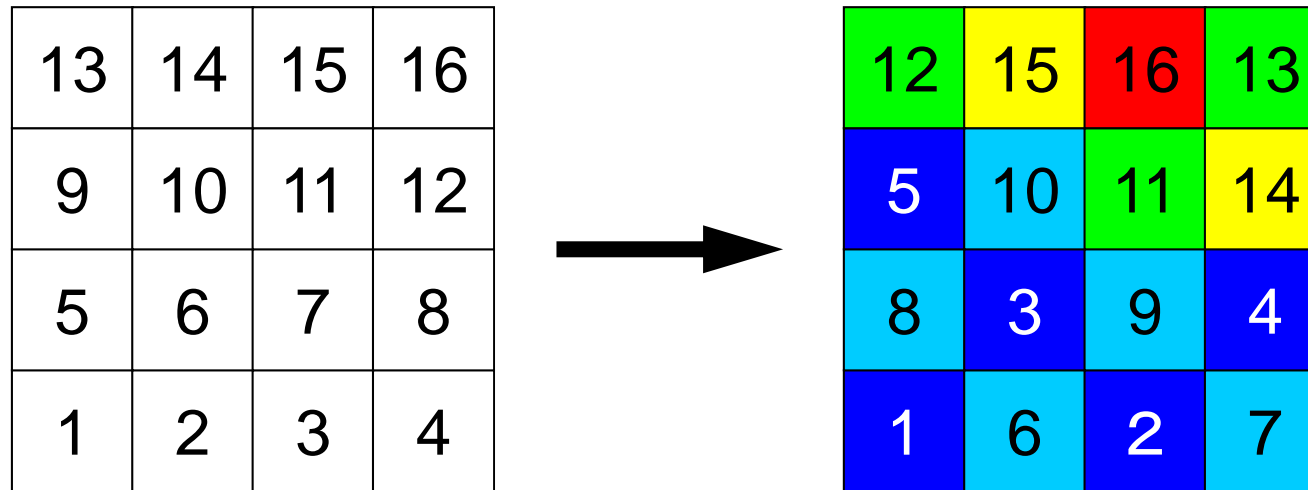
## poi\_gen (6/8)

### Calculation of Coefficients



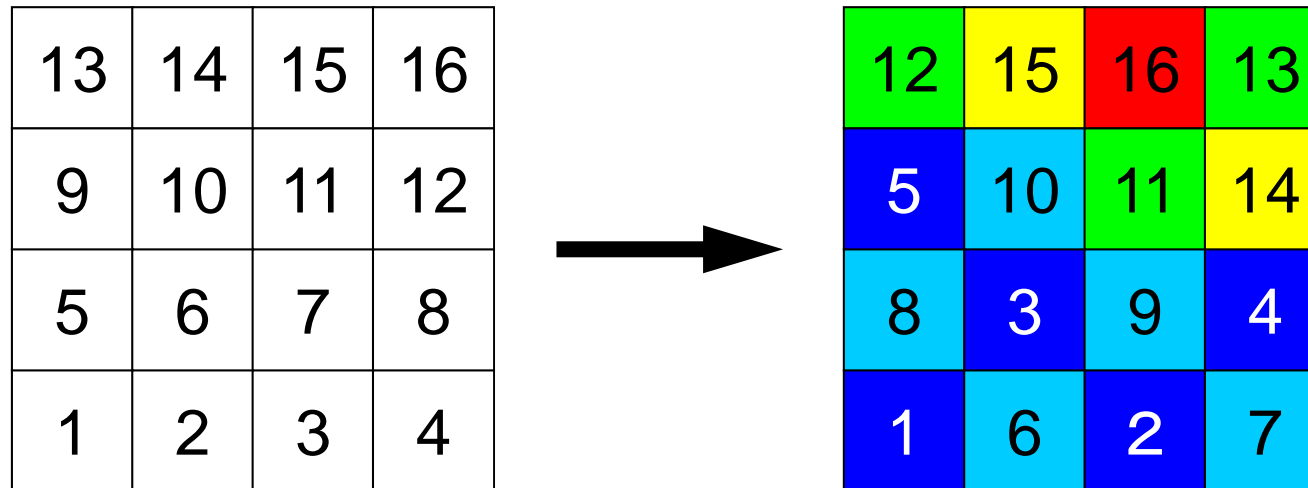
$$\begin{aligned}
 & \frac{\phi_E - \phi_i}{\Delta x} \Delta y + \frac{\phi_W - \phi_i}{\Delta x} \Delta y + \\
 & \frac{\phi_N - \phi_i}{\Delta y} \Delta x + \frac{\phi_S - \phi_i}{\Delta y} \Delta x = f_c \Delta x \Delta y
 \end{aligned}$$

# New Numbering



- Coloring by MC/CM/RCM/CM-RCM
- Renumber meshes in ascending orders according to “Level/Color” ID.
  - 1<sup>st</sup>-Color: 1,2,3,4,5 (Original: 1,3,6,8,9)
  - 2<sup>nd</sup>-Color: 6,7,8,9,10 (2,4,5,7,10)
  - 3<sup>rd</sup>-Color: 11,12,13 (11,13,16)
  - 4<sup>th</sup>-Color: 14,15 (12,14), 5<sup>th</sup>-Color: 16 (15)

# New Numbering (cont.)



`NCOLORTot= 5`

`COLORindex[0]= 0, COLORindex[1]= 5, COLORindex[2]= 10`

`COLORindex[3]= 13, COLORindex[5]= 15, COLORindex[6]= 16`

- **NEWtoOLD, OLDtoNEW**
  - **OLDtoNEW(6)=3, NEWtoOLD(3)=6**



```

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];

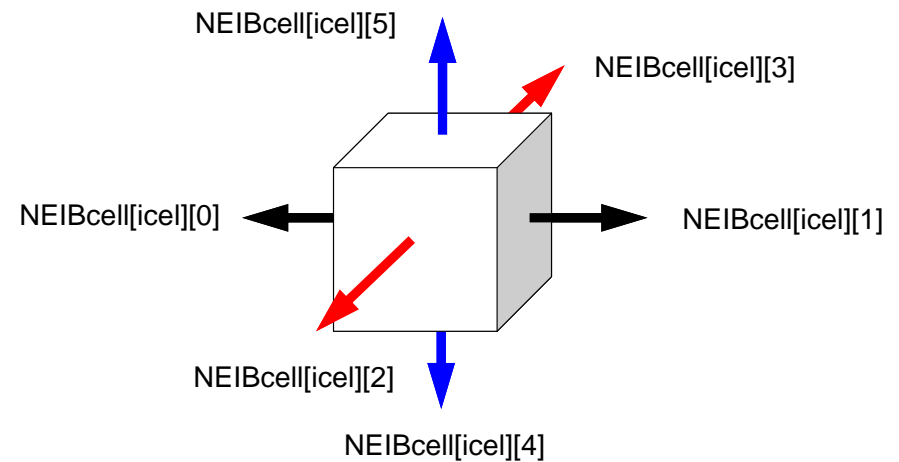
  isL = indexL[icel  ];   ieL = indexL[icel+1];
  isU = indexU[icel  ];   ieU = indexU[icel+1];

  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)



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

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

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

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

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

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

```

for(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];

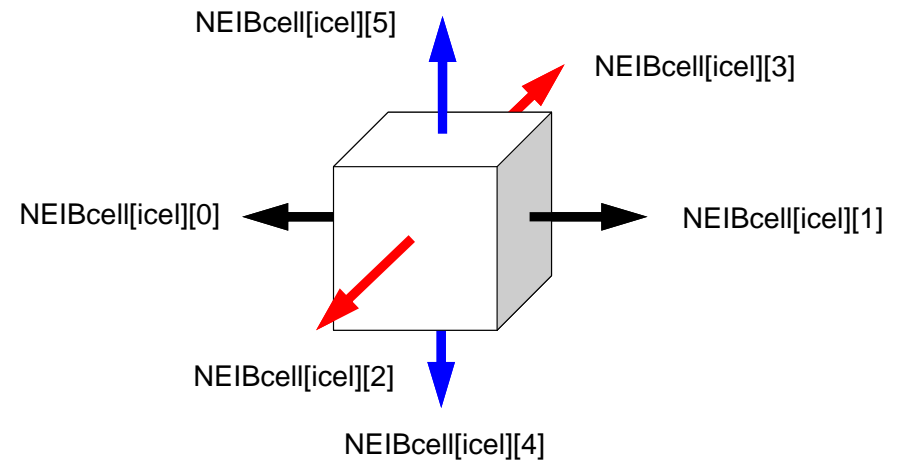
  if(icN5 != 0) {
    icN5 = OLDtoNEW[icN5-1];
    coef = RDZ * ZAREA;
    D[icel] -= coef;

    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 < icel  
Lower Part**

## poi\_gen (6/8)



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

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

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

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

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

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

```

for(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];

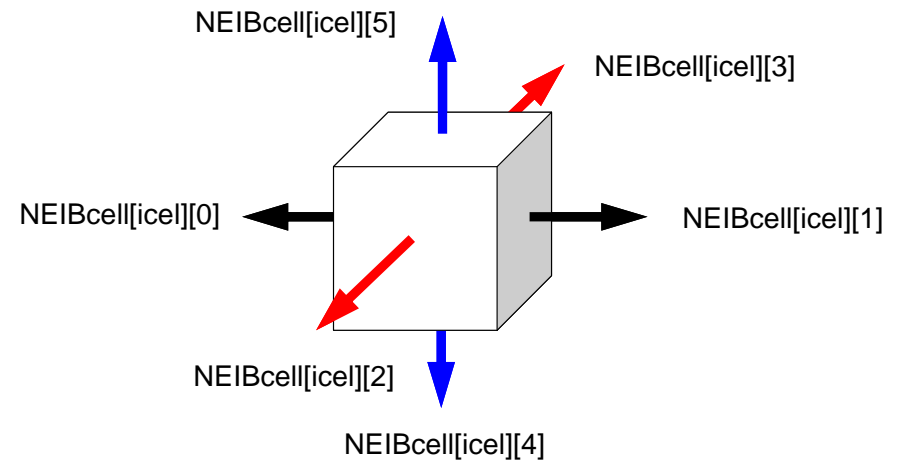
  if(icN5 != 0) {
    icN5 = OLDtoNEW[icN5-1];
    coef = RDZ * ZAREA;
    D[icel] -= coef;

    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 > icel  
Upper Part**

## poi\_gen (6/8)



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

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

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

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

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

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

```

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

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

```

**BFORCE**  
using original  
mesh ID

## poi\_gen (7/8)

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

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

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

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

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

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

```

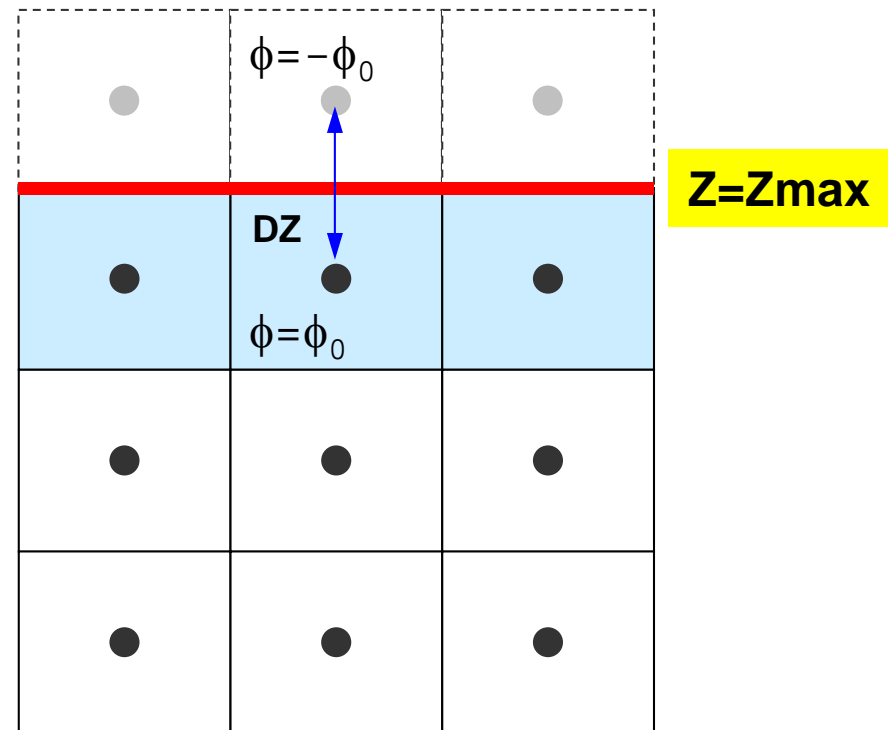
for(ib=0; ib<ZmaxCELtot; ib++) {
    ic0 = ZmaxCEL[ib] - 1;
    coef = 2.0 * RDZ * ZAREA;
    icel = OLDtoNEW[ic0];
    D[icel-1] -= coef;
}

return 0;
}

```

# poi\_gen (8/8)

Calculation of Coefficients  
on Boundary Surface @  $Z=Z_{\max}$



1<sup>st</sup> Order Approximation:

Mirror Image according to  $Z=Z_{\max}$  surface.

$\phi = -\phi_0$  at the center of the (virtual) mesh

$\phi = 0$  @  $Z=Z_{\max}$  surface

# 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;
}

```

**Matrix, RHS are calculated according to new numbering**

# 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;
  }
}

```

**Incomplete  
"Modified" Cholesky  
Factorization**

```

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;
}

```

**Incomplete  
"Modified" Cholesky  
Factorization**

```

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;
    }
}

```

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



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

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

# Incomplete “Modified” Cholesky Factorization

```

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;
}

```



Mesh “i” and “itemL(k)” in RHS belong to different “colors”.

NO data dependency.

```

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;
  }
}

```

# 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)
    ρi-1 = r(i-1) z(i-1)
    if i=1
        p(1) = z(0)
    else
        βi-1 = ρi-1/ρi-2
        p(i) = z(i-1) + βi-1 p(i-1)
    endif
    q(i) = [A]p(i)
    αi = ρi-1/p(i)q(i)
    x(i) = x(i-1) + αip(i)
    r(i) = r(i-1) - αiq(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  $i = 1, 2, \dots$   
**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;
    }
}

```

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

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

Forward Substitution

				<b>ic=1</b>
<u>3</u>		<u>4</u>		
<u>1</u>		<u>2</u>		

# 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;
    }
}

```

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

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

Forward Substitution

				ic=2
3	<u>7</u>	4	<u>8</u>	
1	<u>5</u>	2	<u>6</u>	

# 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;
    }
}

```

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

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

Forward Substitution

<u>11</u>		<u>12</u>	
3	7	4	8
<u>9</u>		<u>10</u>	
1	5	2	6

ic=3

# 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;
    }
}

```

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

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

Forward Substitution

11	<u>15</u>	12	<u>16</u>	ic=4
3	7	4	8	
9	<u>13</u>	10	<u>14</u>	
1	5	2	6	



# 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;
    }
}

```

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

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

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

Forward Substitution

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++) {
            [j]-1];

If order of computations in same color is changed: NO effect

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

(M){z} = (LDLT){z} = {r}

(L){z} = {r}

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;
    }
}

(DLT){z} = {z}

```

Forward Substitution

Backward Substitution

11	15	12	16
	<u>7</u>		<u>8</u>
9	13	10	14
	<u>5</u>		<u>6</u>

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 / RH01 otherwise *
 *****/

if(L == 0) {
    for(i=0; i<N; i++) {
        W[P][i] = W[Z][i];
    }
} else {
    BETA = RHO / RH01;
    for(i=0; i<N; i++) {
        W[P][i] = W[Z][i] + BETA * W[P][i];
    }
}

```

Compute  $r^{(0)} = b - [A]x^{(0)}$   
for  $i = 1, 2, \dots$   
 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  $i = 1, 2, \dots$ 
  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;

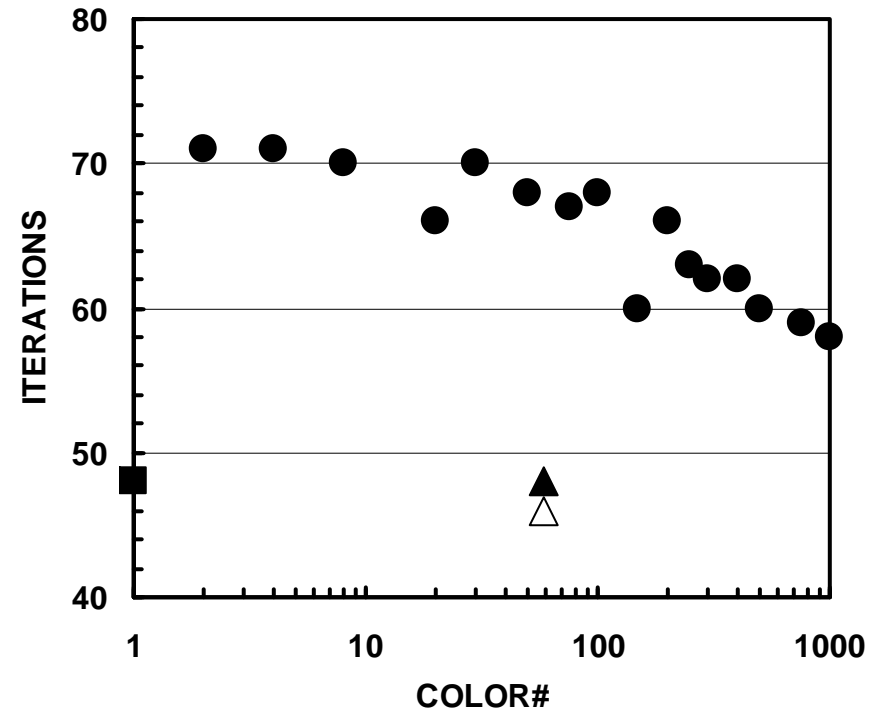
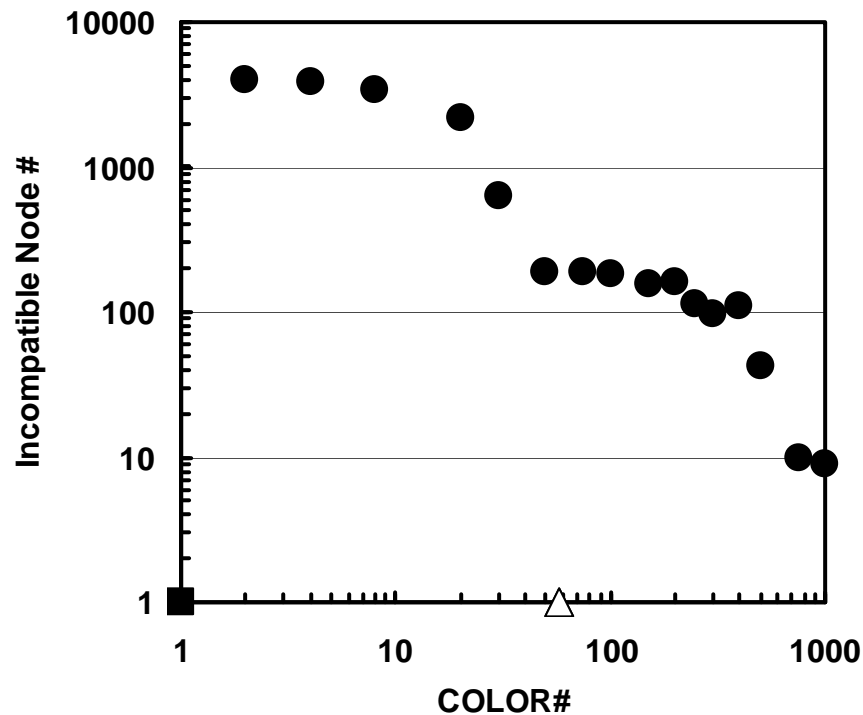
```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
    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

```

# Effect of Color Number on Convergence of ICCG



( $20^3=8,000$  meshe,  $\text{EPSICCG}=10^{-8}$ )

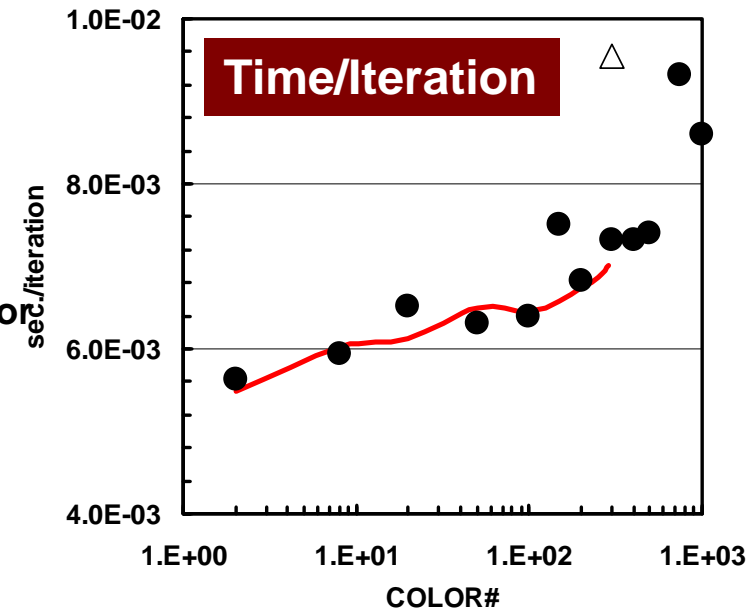
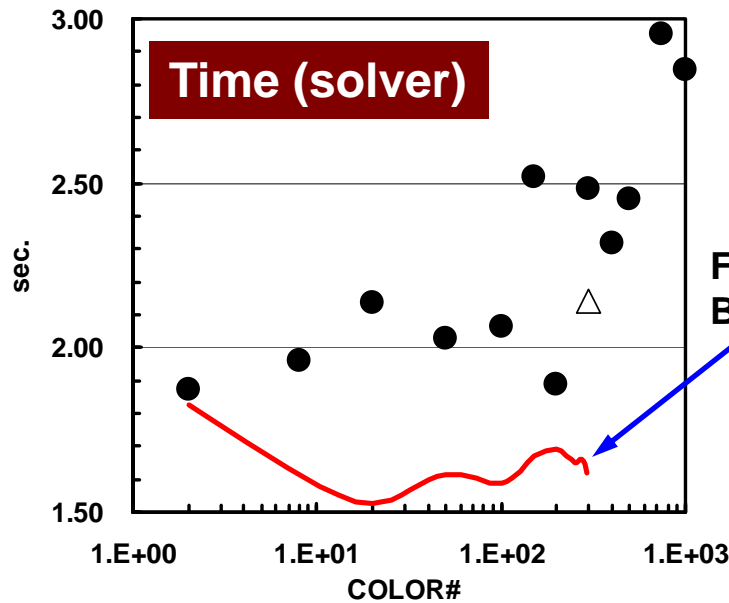
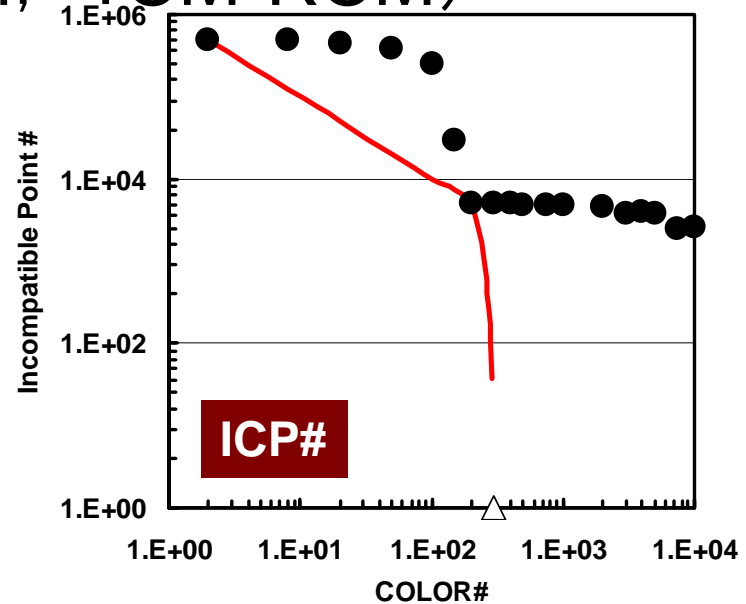
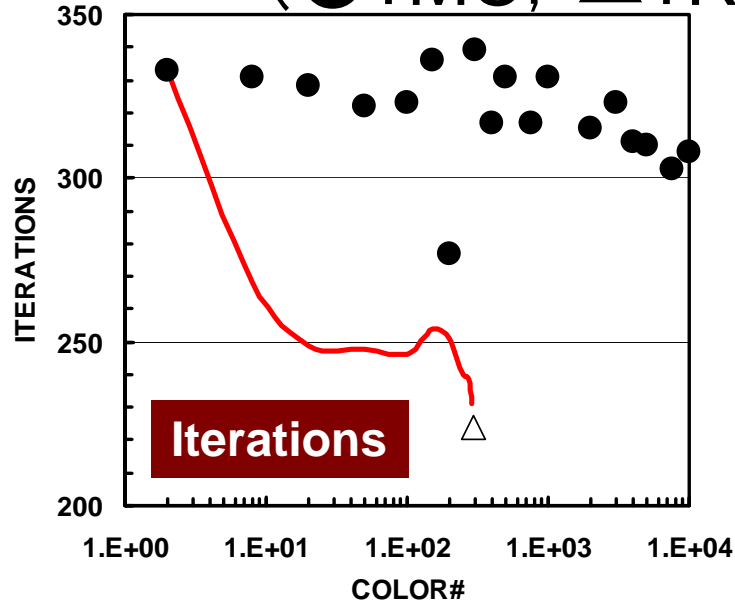
(■ : ICCG(L1), ● : ICCG-MC, ▲ : ICCG-CM, △ : ICCG-RCM)

- 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...**



# SR11000, 1-node/16-cores, $100^3$

(●:MC, △:RCM, -:CM-RCM)



# FX10, 1-node/16-cores, $100^3$

(●:MC, △:RCM, -:CM-RCM)

