

Introduction to Parallel Programming for Multicore/Manycore Clusters

Part III: Reordering

Kengo Nakajima
Information Technology Center
The University of Tokyo

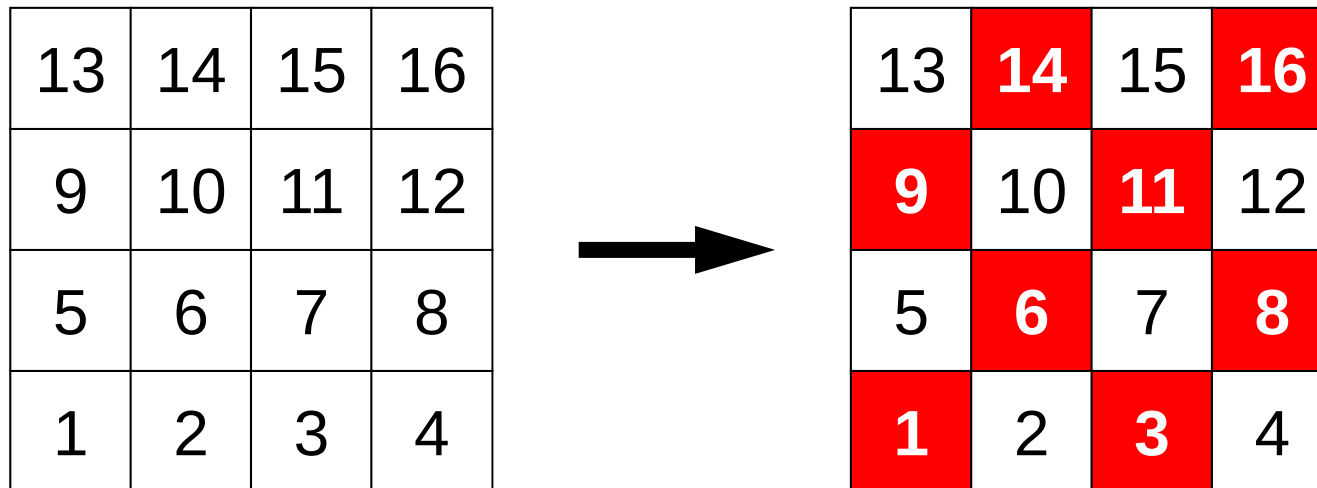
- Remedy for Data Dependency
- Ordering/Reordering
 - Red-Black, Multicoloring (MC)
 - Cuthill-McKee (CM), Reverse-CM (RCM)
 - Reordering and Convergence
- Implementation
- ICCG with Reordering

Parallelize ICCG Method

- Dot Product: OK
- DAXPY: OK
- Matrix-Vector Multiply: OK
- Preconditioning: **Something special needed !**
 - Just inserting OpenMP directive is not enough

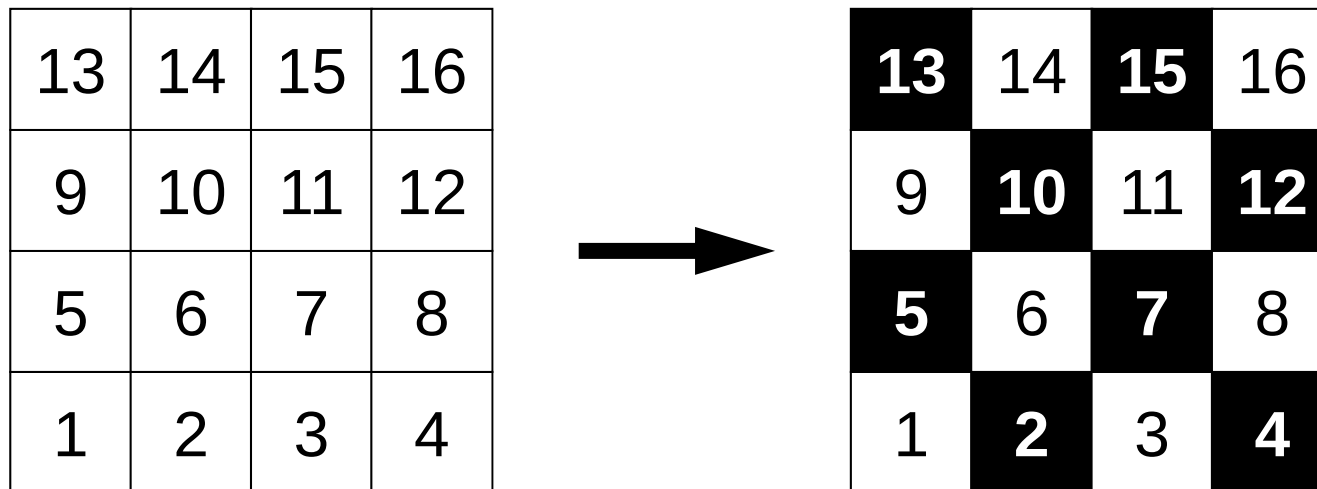
One Remedy for Data Dependency = Coloring

- Parallel (concurrent) processing is possible for independent meshes without dependency



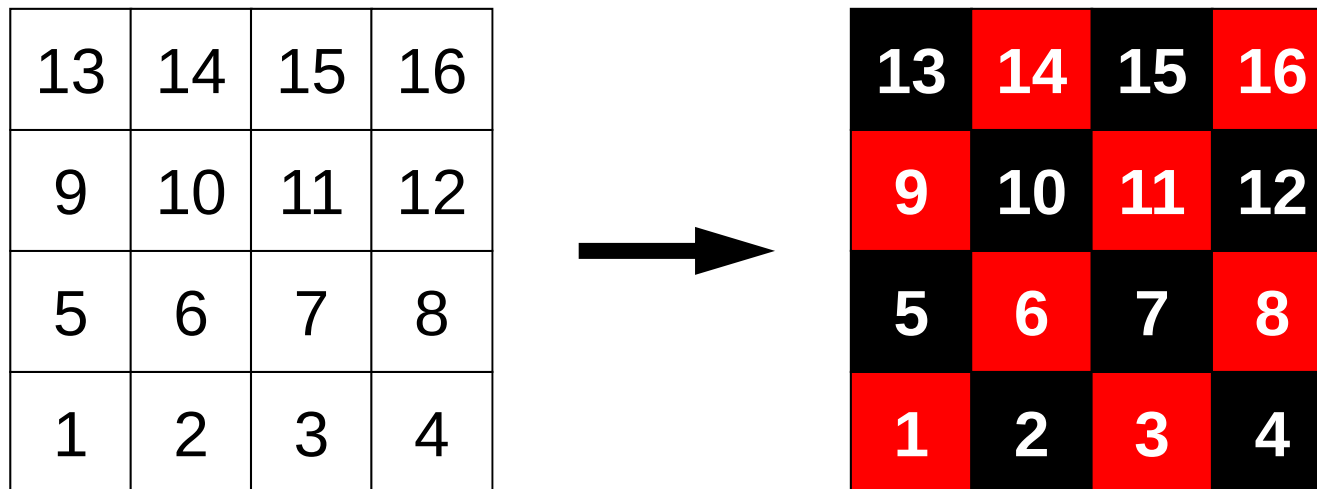
One Remedy for Data Dependency = Coloring

- Parallel (concurrent) processing is possible for independent meshes without dependency



One Remedy for Data Dependency = Coloring

- Applying same “color” to independent meshes without dependency: Coloring
- Most simple case: Red-Black with 2 Colors



Red-Black (1/3)

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

```

!$omp parallel do private (ip, i, k, VAL)
do ip= 1, 4
do i= INDEX(ip-1)+1, INDEX(ip)
  if (COLOR(i).eq. RED) then
    WVAL= W(i, Z)
    do k= indexL(i-1)+1, indexL(i)
      WVAL= WVAL - AL(k) * W(itemL(k), Z)
    enddo
    W(i, Z)= WVAL * W(i, DD)
  endif
enddo
enddo
!$omp parallel enddo

!$omp parallel do private (ip, i, k, VAL)
do ip= 1, 4
do i= INDEX(ip-1)+1, INDEX(ip)
  if (COLOR(i).eq. BLACK) then
    WVAL= W(i, Z)
    do k= indexL(i-1)+1, indexL(i)
      WVAL= WVAL - AL(k) * W(itemL(k), Z)
    enddo
    W(i, Z)= WVAL * W(i, DD)
  endif
enddo
enddo
!$omp parallel enddo

```

Red-Black (2/3)

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

```

!$omp parallel do private (ip, i, k, VAL)
do ip= 1, 4
do i= INDEX(ip-1)+1, INDEX(ip)
  if (COLOR(i).eq. RED) then
    WVAL= W(i, Z)
    do k= indexL(i-1)+1, indexL(i)
      WVAL= WVAL - AL(k) * W(itemL(k), Z)
    enddo
    W(i, Z)= WVAL * W(i, DD)
  endif
enddo
enddo
!$omp parallel enddo

```

- During operations on “red” meshes, only “black” meshes appear in RHS.
 - “red”: writing, “black”: reading
- During operations on “red” meshes, values on “black” meshes do not change.
- Data dependency is avoided.

Red-Black (3/3)

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

- During operations on “black” meshes, only “red” meshes appear in RHS.
– “black”: writing, “red”: reading
- During operations on “black” meshes, values on “red” meshes do not change.
- Data dependency is avoided.

```

!$omp parallel do private (ip, i, k, VAL)
do ip= 1, 4
do i= INDEX(ip-1)+1, INDEX(ip)
if (COLOR(i).eq. BLACK) then
WVAL= W(i, Z)
do k= indexL(i-1)+1, indexL(i)
WVAL= WVAL - AL(k) * W(itemL(k), Z)
enddo
W(i, Z)= WVAL * W(i, DD)
endif
enddo
enddo
!$omp parallel enddo
  
```

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

```

do icol= 1, 2
!$omp parallel do private (ip, I, j, VAL)
  do ip= 1, 4
    do i= INDEX(ip-1, icol)+1, INDEX(ip, icol)
      WVAL= W(i, Z)
      do k= indexL(i-1)+1, indexL(i)
        WVAL= WVAL - AL(k) * W(itemL(k), Z)
      enddo
      W(i, Z)= WVAL * W(i, DD)
    enddo
  enddo
!$omp parallel enddo
enddo

```

INDEX (0, 1)= 0
 INDEX (1, 1)= 2
 INDEX (2, 1)= 4
 INDEX (3, 1)= 6
 INDEX (4, 1)= 8

INDEX (0, 2)= 8
 INDEX (1, 2)=10
 INDEX (2, 2)=12
 INDEX (3, 2)=14
 INDEX (4, 2)=16

- Renumbering/reordering meshes from “red” to “black”
- Simpler, more efficient

- Remedy for Data Dependency
- **Ordering/Reordering**
 - **Red-Black, Multicoloring (MC)**
 - **Cuthill-McKee (CM), Reverse-CM (RCM)**
 - Reordering and Convergence
- Implementation
- ICCG with Reordering

Effect of Reordering

- **Extracting parallelism, removing dependency**
- Reducing
 - fill-in's, “bandwidth of matrix”, “profile”
- Blocking
- Related to “four color problem”, “travelling salesman problem” etc.
 - applied to numerical analysis

Ordering/Reordering Method for Parallel Computing

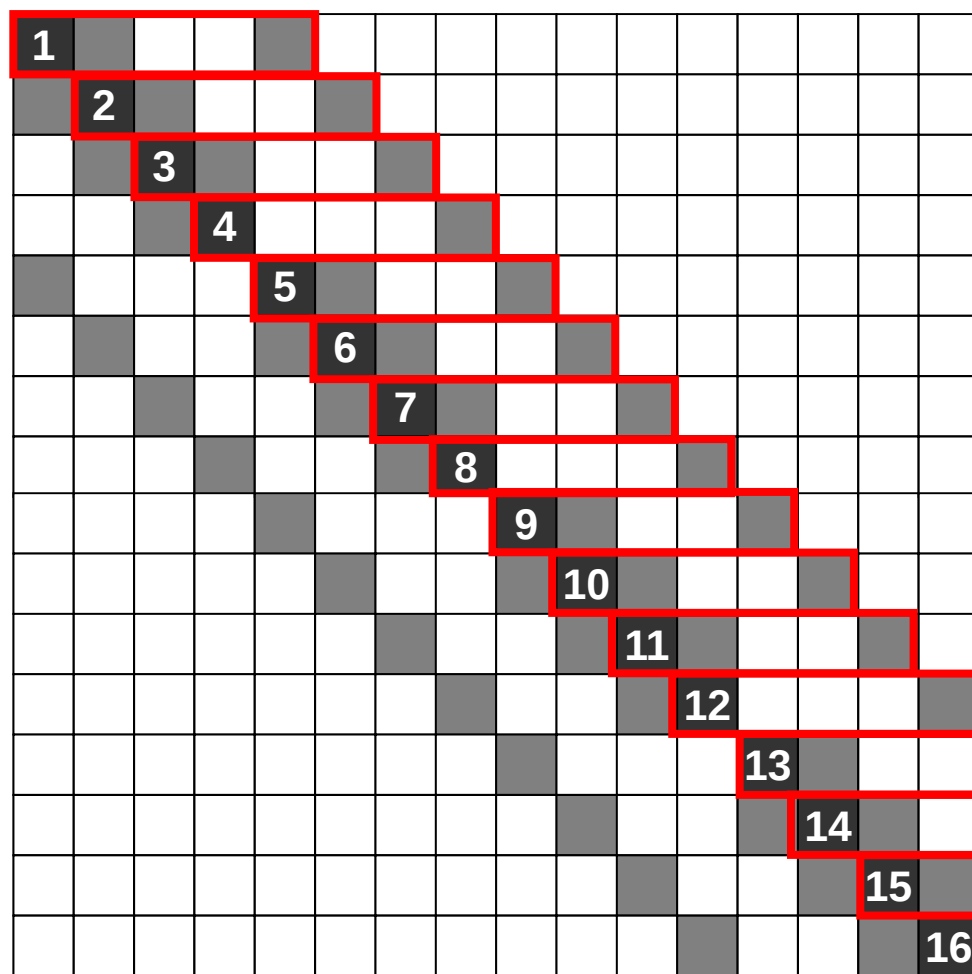
- Multicoloring (MC)
 - Parallelism
 - Red-Black with 2 colors
- CM (Cuthill-McKee), RCM (Reverse Cuthill-McKee)
 - Reducing fill-in's, matrix bandwidth, profiles
 - Parallelism

Technical Terms for Matrix

- β_i : $\beta_i = k - i$ where maximum ID number of non-zero column is k at i -th row of the target matrix
- Bandwidth: Maximum value of β_i
- Profile: Total sum of β_i
- Bandwidth, Profile, Fill-in
 - smaller is better
 - Bandwidth and profile of matrices affects convergence.

$$\beta_i$$

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



■ Non-zero off-diagonals

Multicoloring (MC), Multicolor Ordering

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

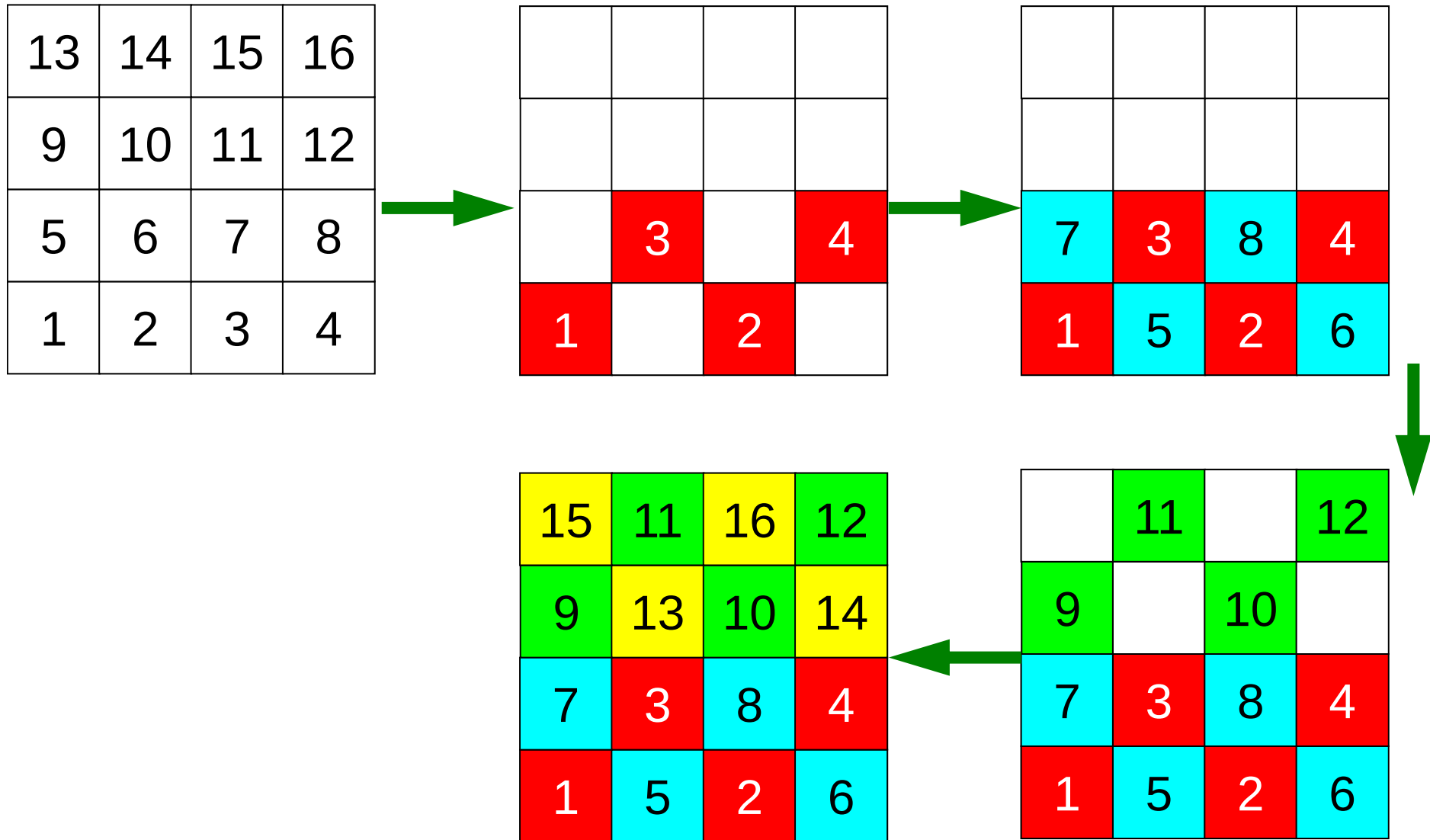
15	7	16	8
5	13	6	14
11	3	12	4
1	9	2	10

- Meshes in same color are independent, and renumbered according to the color ID.
 - Red-Black: MC with 2 colors
 - More colors needed for complicated geometries
- Parallel operations are possible for meshes in same color.
- A mesh and its neighboring meshes belong to different colors.

Fundamental Algorithm of MC Method

- ① $m = \text{mesh\#} / \text{color\#}$
- ② Color “m” independent meshes in ascending orders according to initial mesh ID, then proceed to the next “color”
- ③ Repeat ②, until every mesh is colored
- ④ Renumber meshes in ascending orders according to color ID. In each color, numbering is in ascending orders according to initial mesh ID.

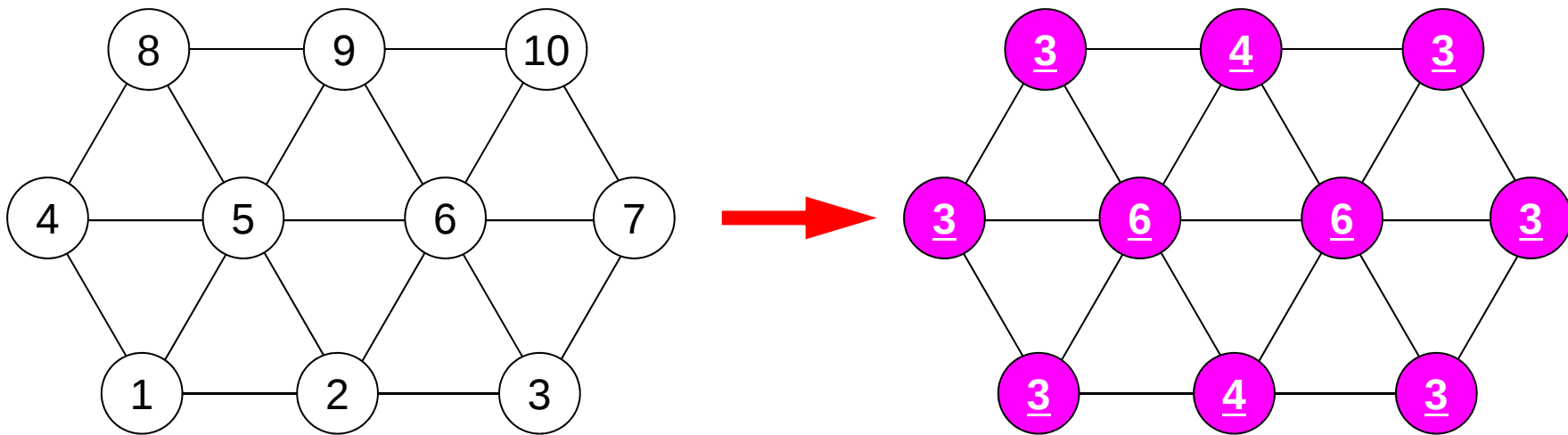
MC with 4 Colors



Modified MC Method

- ① The mesh with minimum value of “degree” is set to “NEW mesh ID= 1”, “Color ID= 1”, and “counter for color number” is 1.
- ② Define “ $ITEMcou = ICELTOT/NCOLORtot$ ”, where $ITEMcou$ is maximum number of meshes in each color.
- ③ Color $ITEMcou$ independent meshes in ascending order according to initial mesh ID.
- ④ If $ITEMcou$ meshes are colored, or no more independent meshes do not exist, add “1” to the “counter for color number”, and proceed to the next color.
- ⑤ Repeat ③ and ④, until all meshes have been colored.
- ⑥ “Final counter for color” is $NCOLORtotF$. Renumber meshes in ascending orders according to color ID. In each color, numbering is in ascending orders according to initial mesh ID. In each color, new numbering of meshes is continuous.

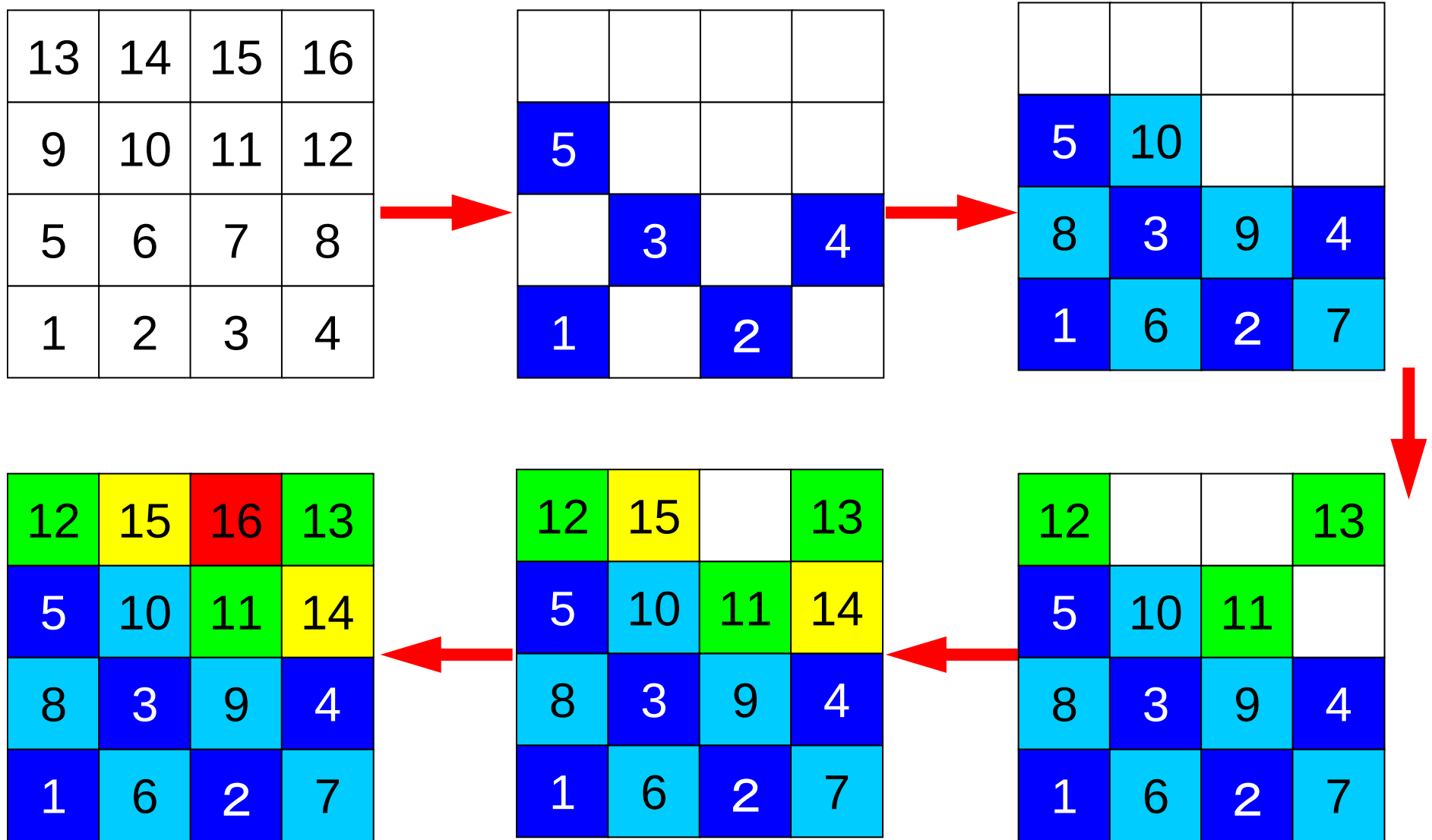
“Degree”: Number of vertices adjacent to each vertex



Modified MC Method

- ① The mesh with minimum value of “degree” is set to “NEW mesh ID= 1”, “Color ID= 1”, and “counter for color number” is 1.
- ② Define “ $ITEMcou = ICELTOT/NCOLORtot$ ”, where $ITEMcou$ is maximum number of meshes in each color.
- ③ Color $ITEMcou$ independent meshes in ascending order according to initial mesh ID.
- ④ If $ITEMcou$ meshes are colored, or no more independent meshes do not exist, add “1” to the “counter for color number”, and proceed to the next color.
- ⑤ Repeat ③ and ④, until all meshes have been colored.
- ⑥ “Final counter for color” is $NCOLORtotF$. Renumber meshes in ascending orders according to color ID. In each color, numbering is in ascending orders according to initial mesh ID.
In each color, new numbering of meshes is continuous.

MC with 3 colors, finally 5 colors



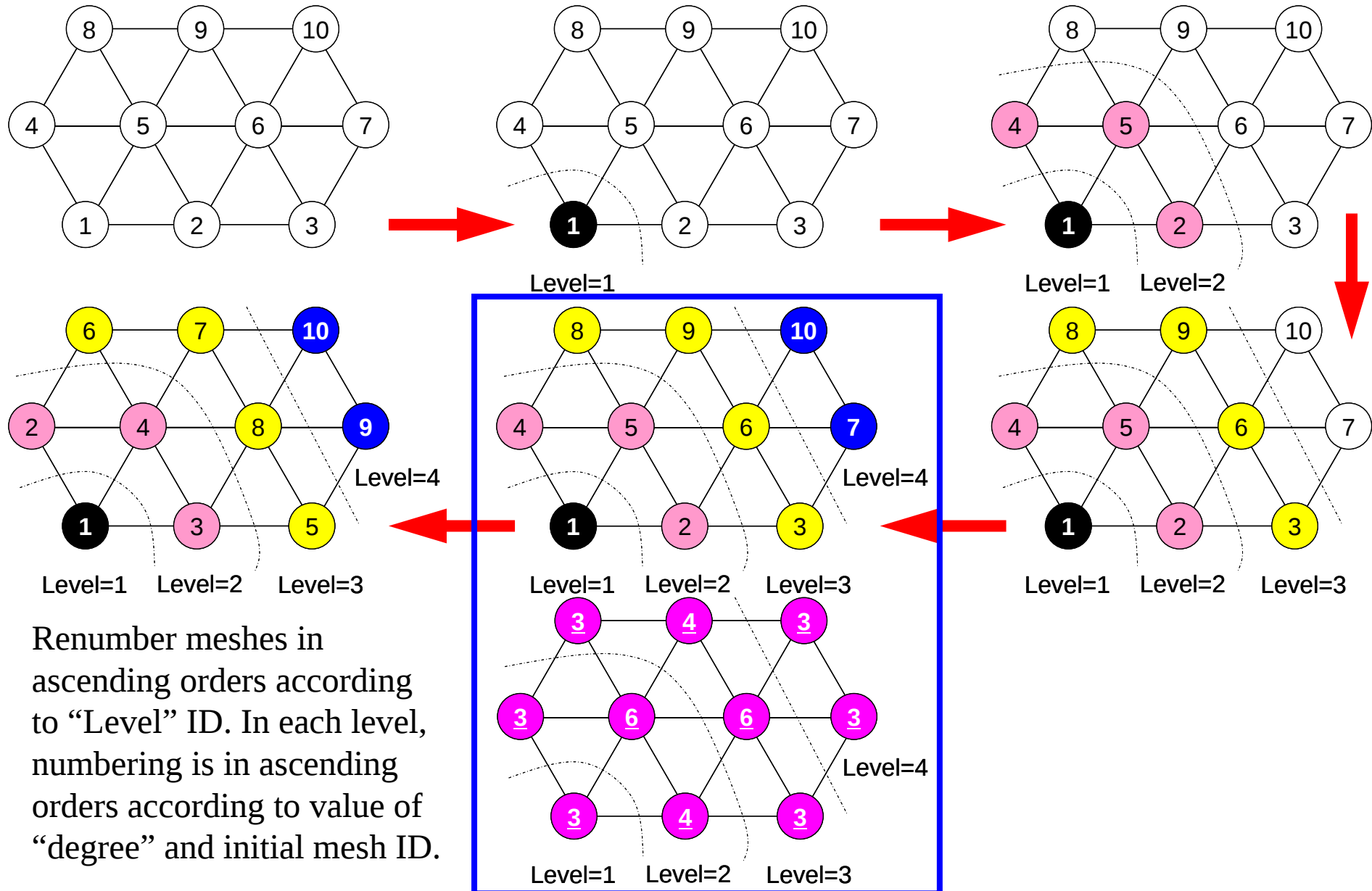
Ordering/Reordering Method for Parallel Computing

- Multicoloring (MC)
 - Parallelism
 - Red-Black with 2 colors
- CM (Cuthill-McKee), RCM (Reverse Cuthill-McKee)
 - Reducing fill-in's, matrix bandwidth, profiles
 - Parallelism

Fundamental Algorithm for CM Method (Cuthill-McKee)

- ① The mesh with minimum value of “degree” is set to “Level=1”.
- ② Meshes adjacent to “Level=k-1” meshes are set to “Level=k”.
- ③ Repeat ②, until all meshes are flagged to “levels”
- ④ Renumber meshes in ascending orders according to “Level” ID. In each level, numbering is in ascending orders according to value of “degree” and initial mesh ID. In each level, new numbering of meshes is continuous.

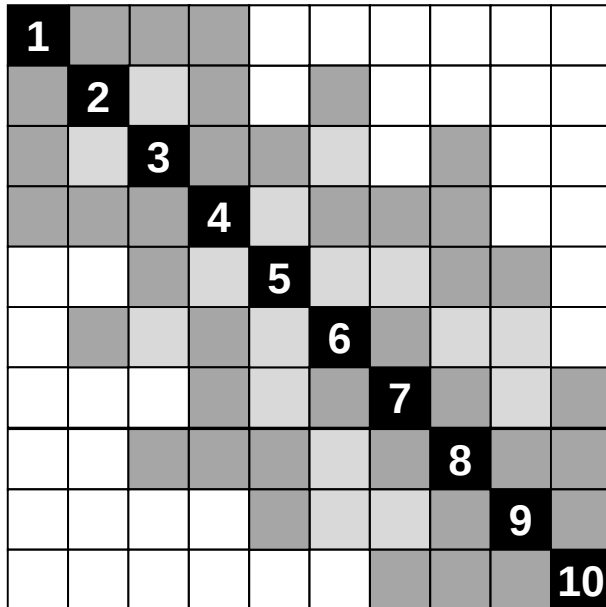
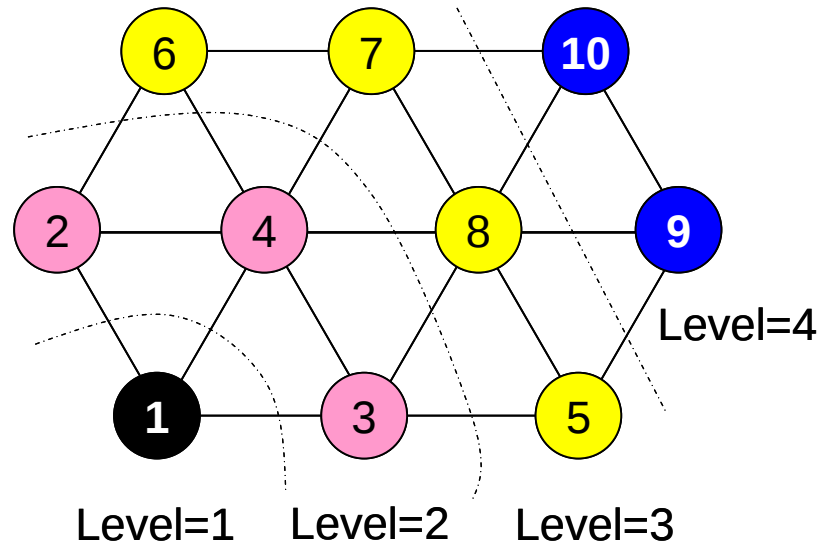
Example of CM Method



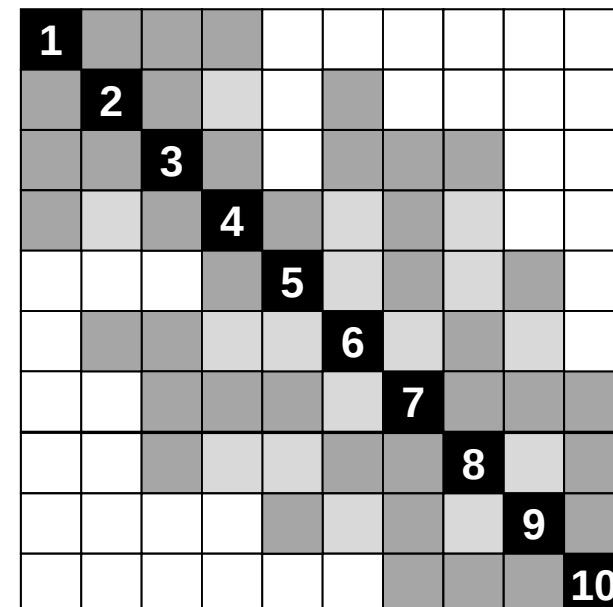
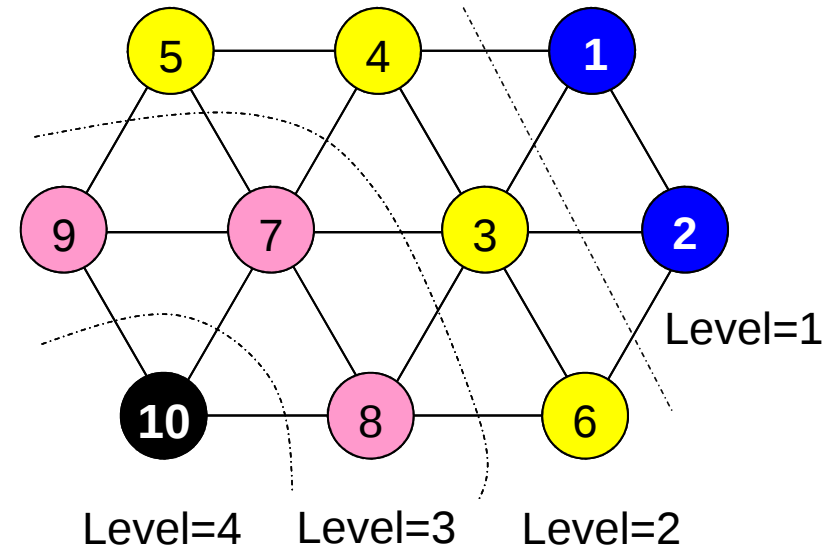
RCM: Reverse Cuthill-McKee

- Do operations for “CM” method
 - Calculate “degree” at each mesh
 - Flag “level k (1,2,...)” to meshes
 - Repeat processes, final renumbering
- Renumbering Again
 - Renumber meshes reordered by CM method in reverse order.
 - Fill-in's (for full LU factorization) are fewer than CM

CM: 16 fill-in's



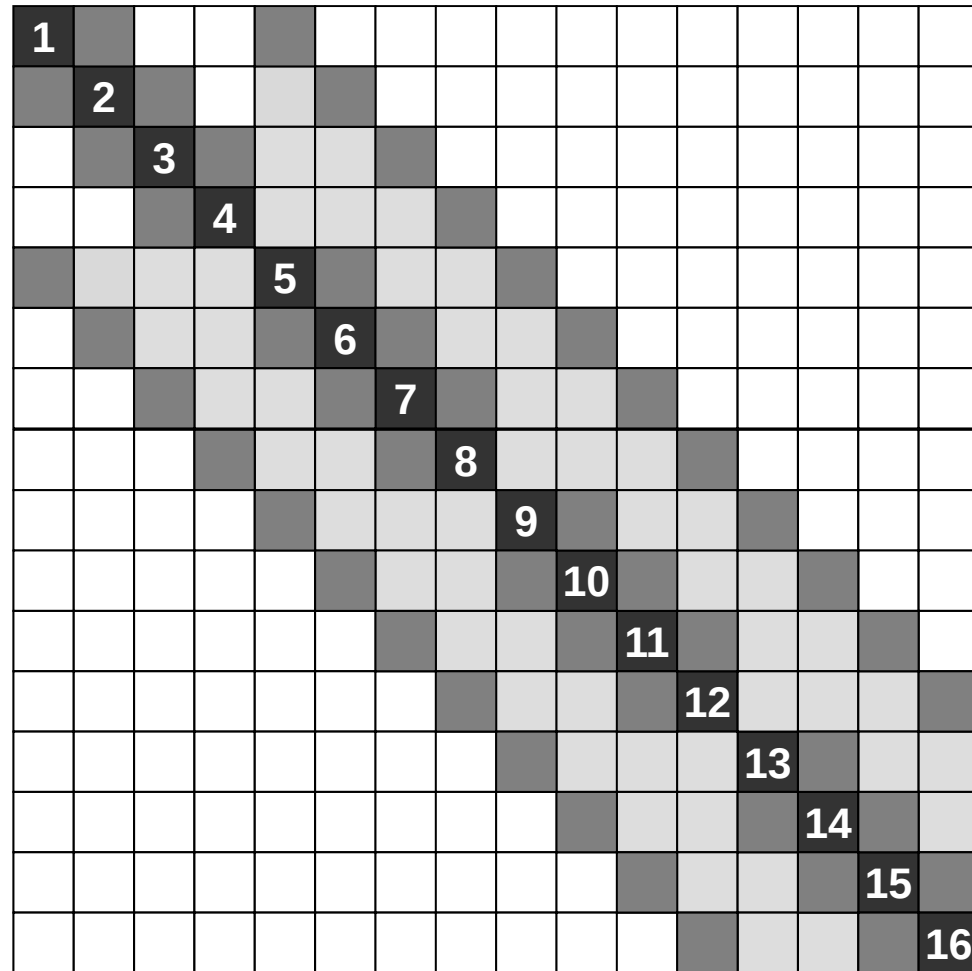
RCM: 14 fill-in's



Initial Matrix

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

Bandwidth 4
Profile 51
Fill-in 54

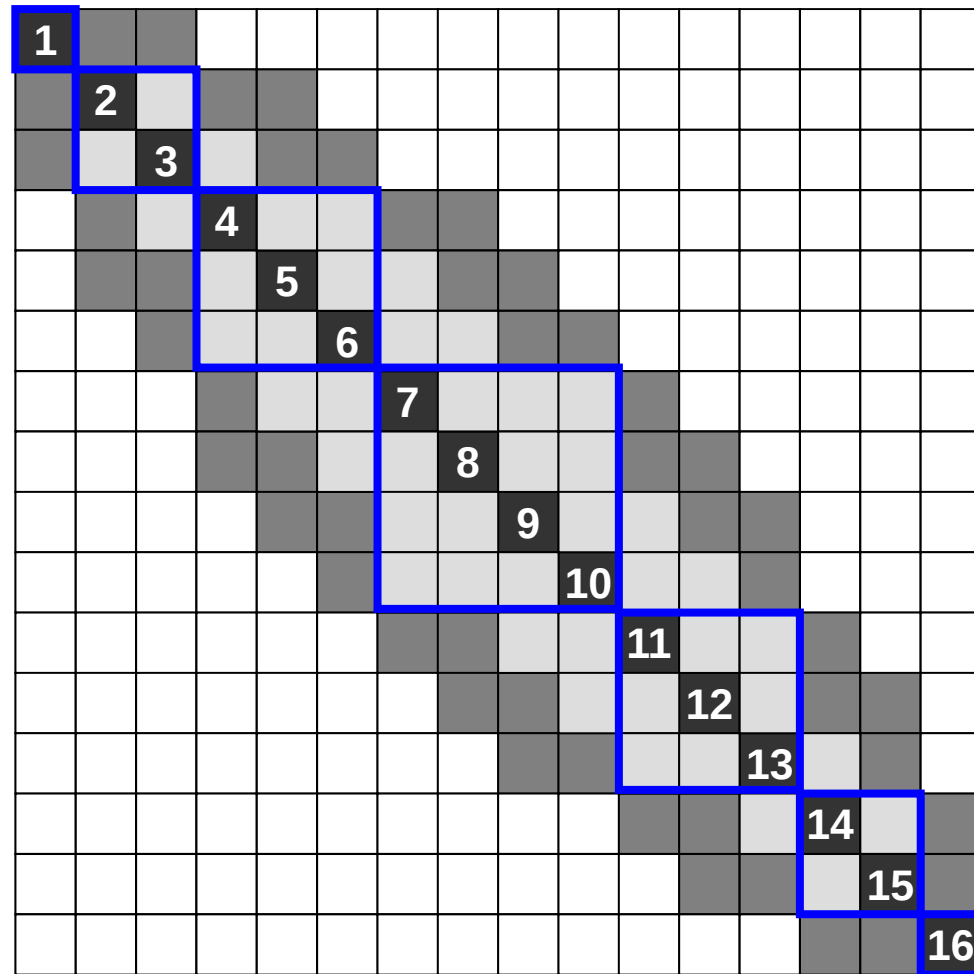


■ Non-zero, ■ Fill-in

CM

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

Bandwidth 4
Profile 46
Fill-in 44

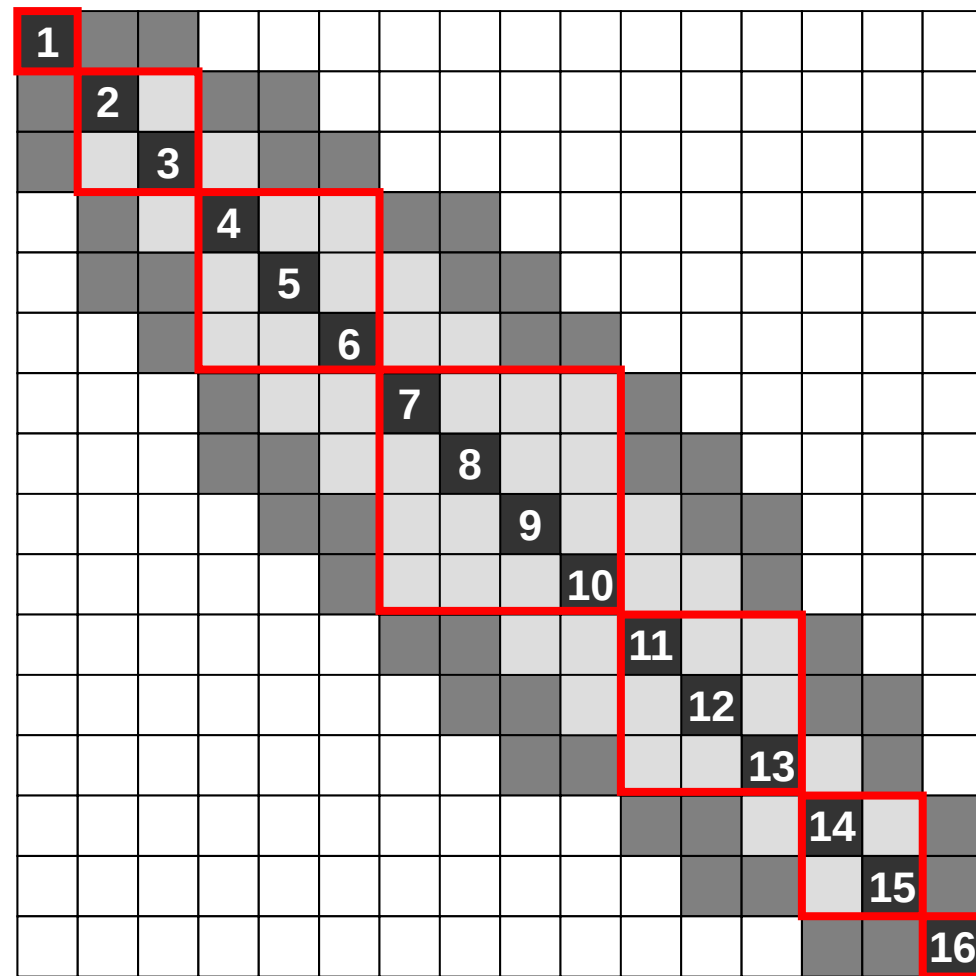


■ Non-zero, ■ Fill-in

RCM

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

Bandwidth 4
Profile 46
Fill-in 44



■ Non-zero, ■ Fill-in

CM, RCM: Hyperline ($i+j=\text{const.}$)

3D: Hyperplane ($i+j+k=\text{const.}$)

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



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

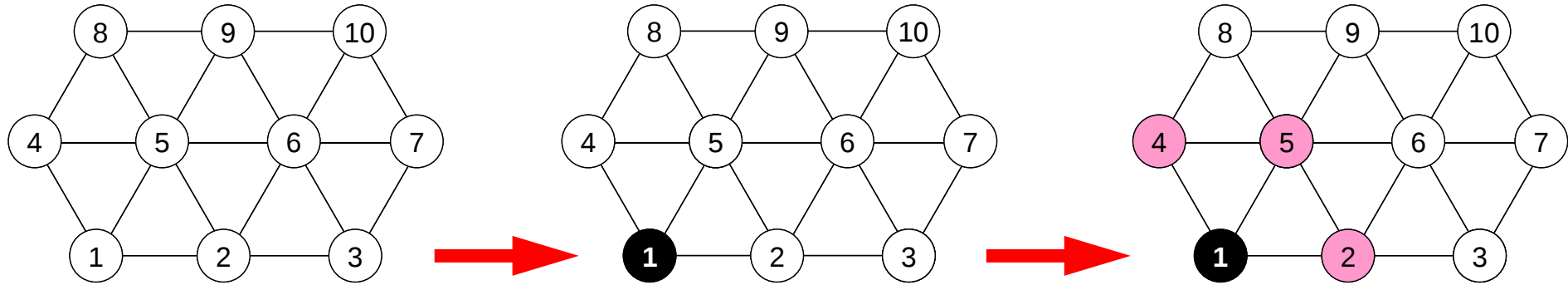
1,4	2,4	3,4	4,4
1,3	2,3	3,3	4,3
1,2	2,2	3,2	4,2
1,1	2,1	3,1	4,1

$i+j=5$

Modified CM Method for Parallel Computing

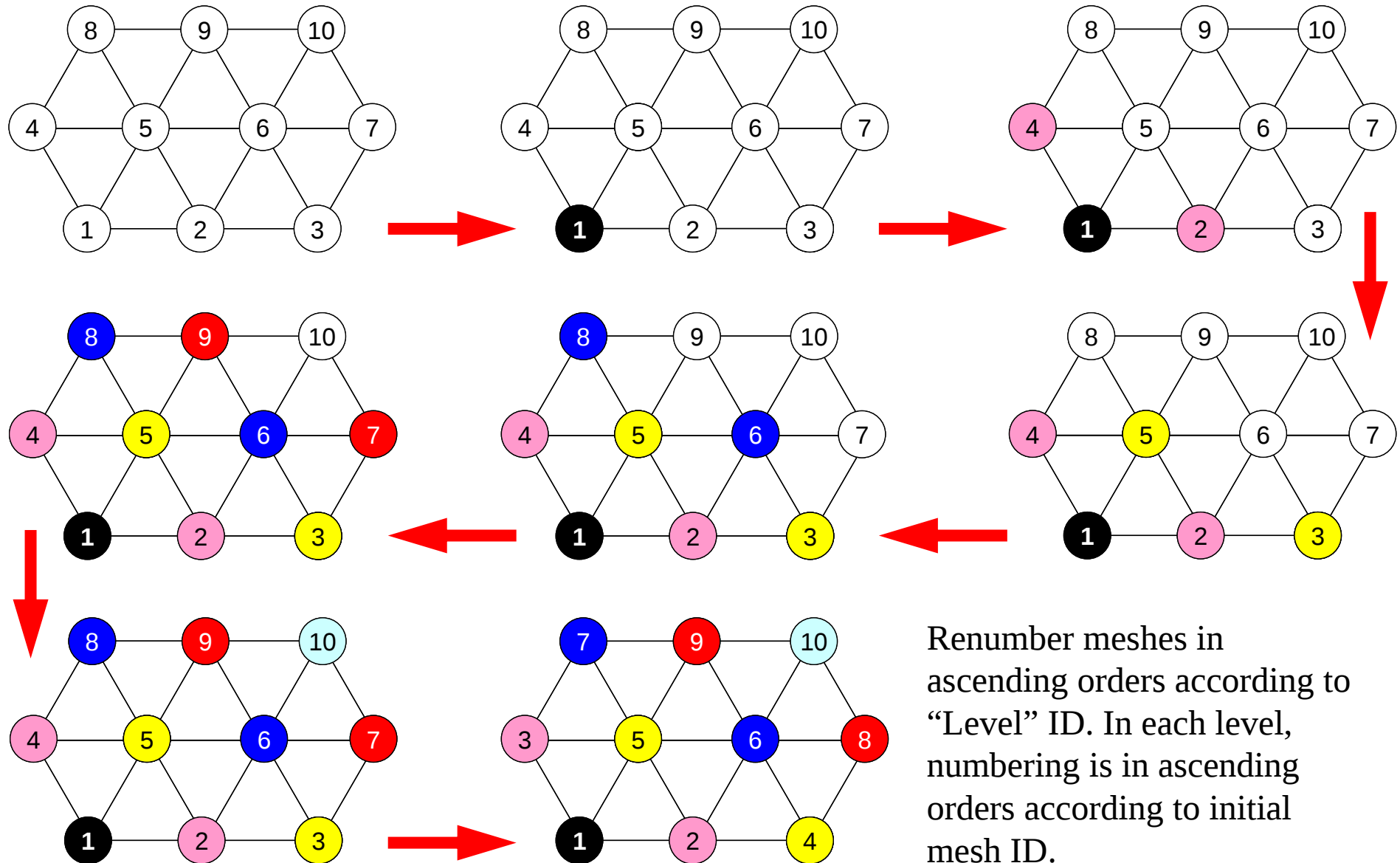
- ① The mesh with minimum value of “degree” is set to “Level=1”.
- ② Meshes adjacent to “Level=k-1” meshes are set to “Level=k”. In each level, meshes must be independent (not directly connected). If a dependent pair is found in same color, one mesh is removed (In current implementation, a mesh found later is removed).
- ③ Repeat ②, until all meshes are flagged to “levels”
- ④ Renumber meshes in ascending orders according to “Level” ID. In each level, numbering is in ascending orders according to initial mesh ID. In each level, new numbering of meshes is continuous.

Modified CM Method

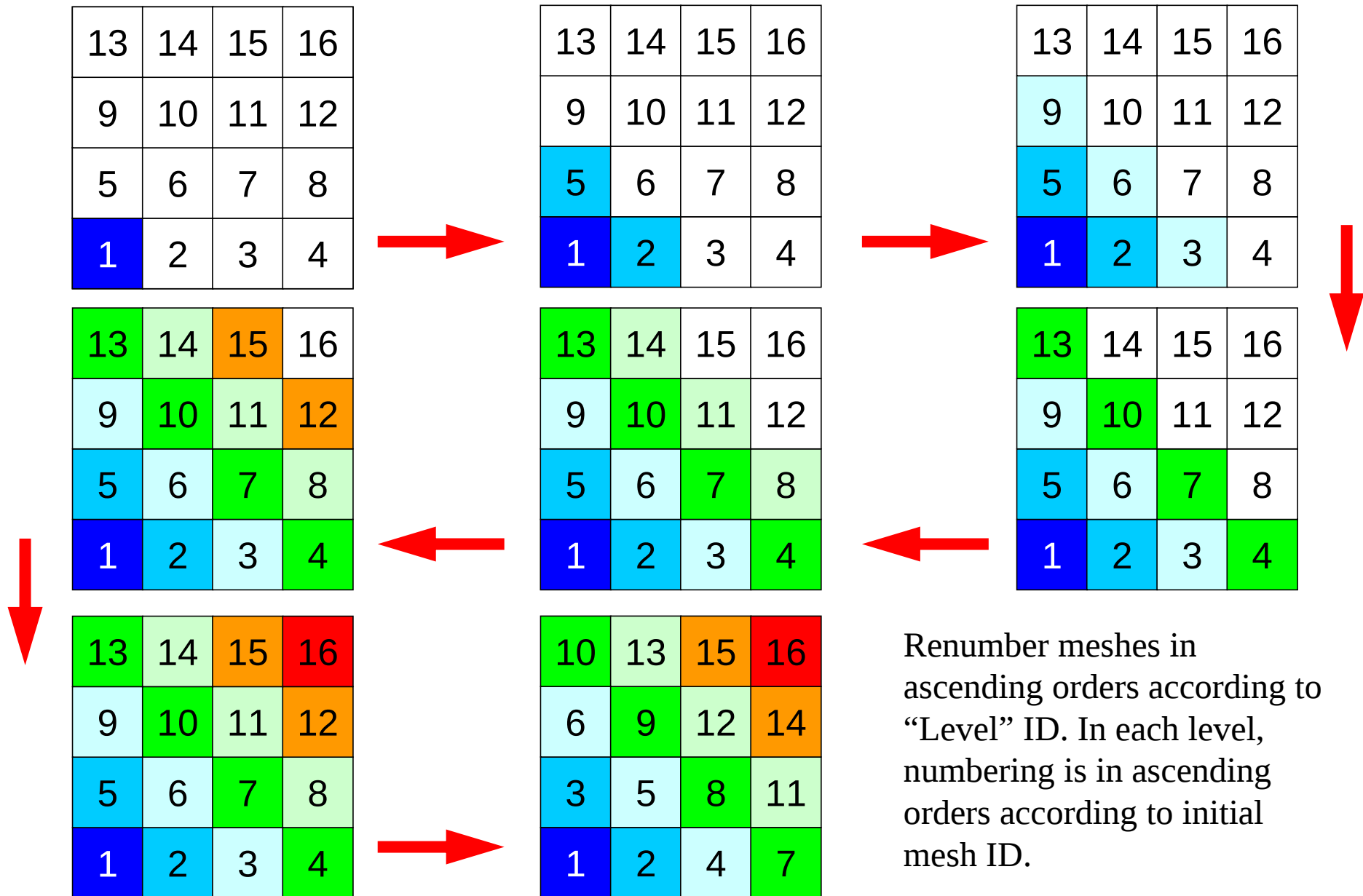


In each level, meshes are independent

Modified CM Method



Modified CM Method



MC and CM/RCM

- In CM/RCM, sequence of computations, and dependency between levels (color) are also considered.

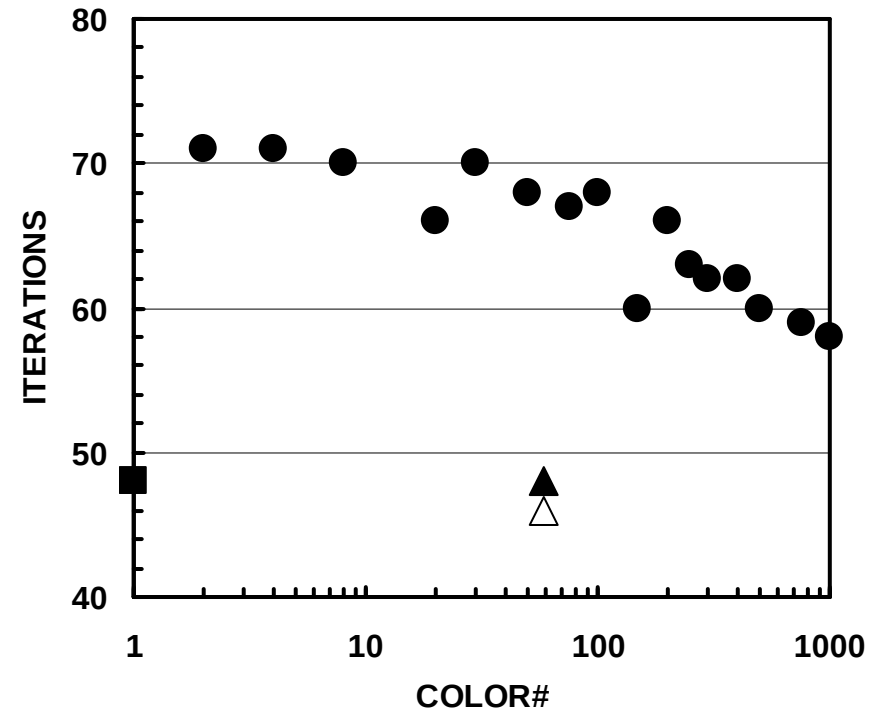
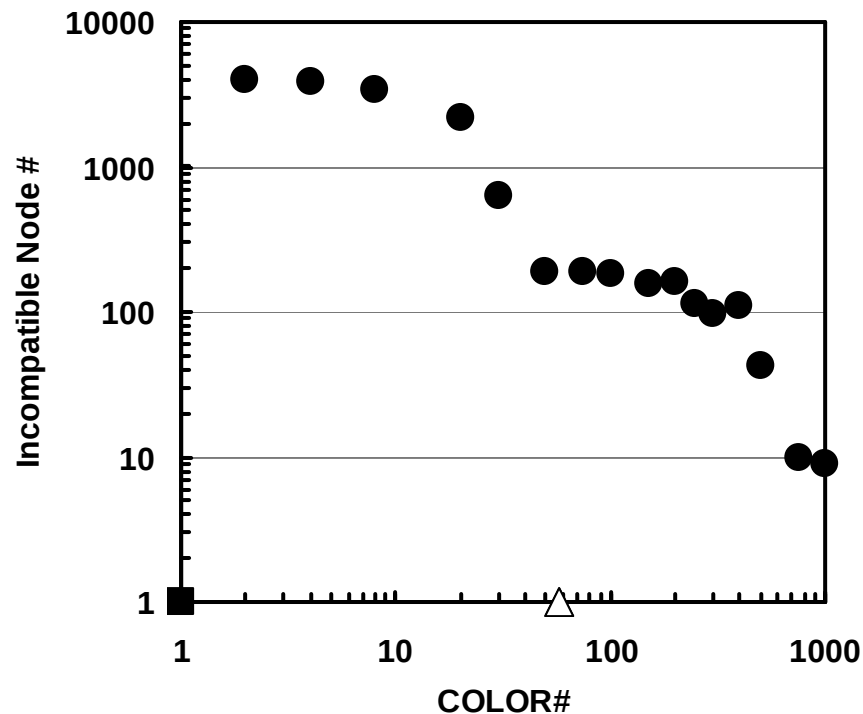
15	7	16	8
5	13	6	14
11	3	12	4
1	9	2	10

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

- Remedy for Data Dependency
- **Ordering/Reordering**
 - Red-Black, Multicoloring (MC)
 - Cuthill-McKee (CM), Reverse-CM (RCM)
 - **Reordering and Convergence**
- Implementation
- ICCG with Reordering

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)

Effect of Color Number on Convergence of ICCG

- Number of Elements: 20^3
- Red-Black ~ 4-Colors < Initial Numbering ~ CM, RCM

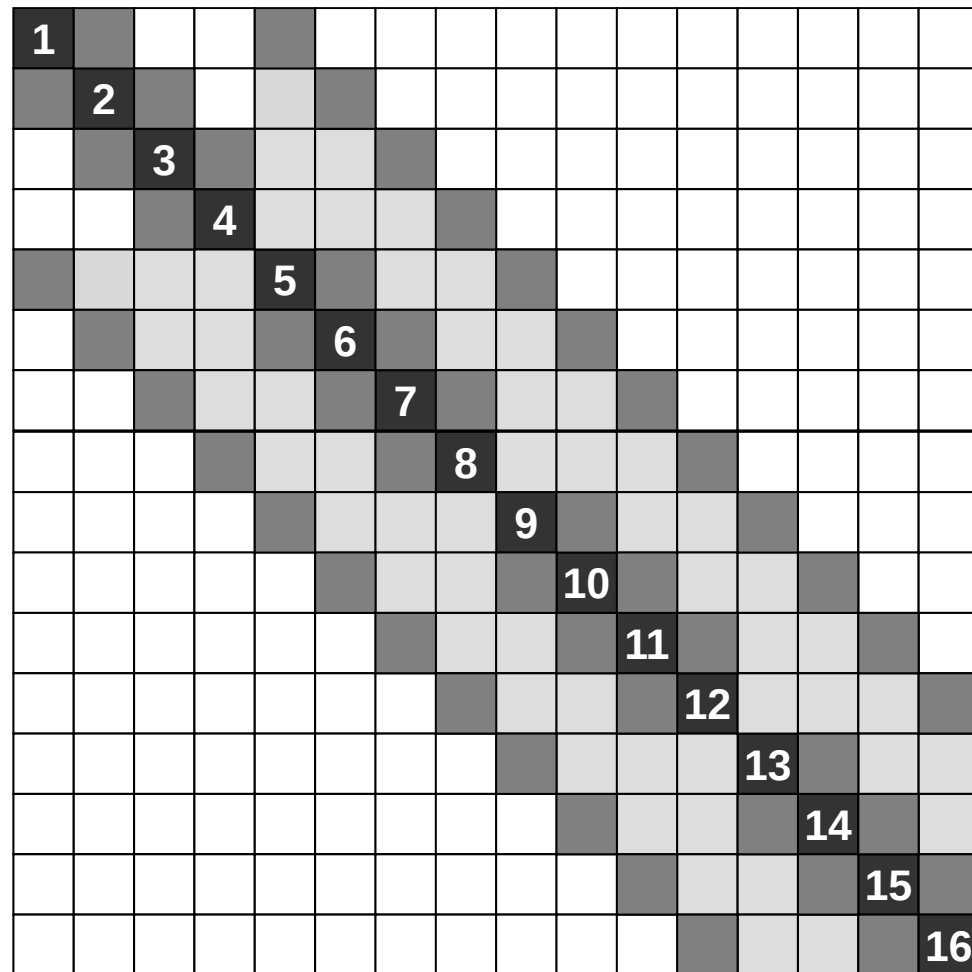
<u>Initial Numbering</u>		<u>Red-Black</u>	
Bandwidth	4	Bandwidth	10
Profile	51	Profile	77
Fill-in	54	Fill-in	44

<u>4-Colors</u>		<u>CM, RCM</u>	
Bandwidth	10	Bandwidth	4
Profile	57	Profile	46
Fill-in	46	Fill-in	44

Initial Matrix

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

Bandwidth 4
Profile 51
Fill-in 54

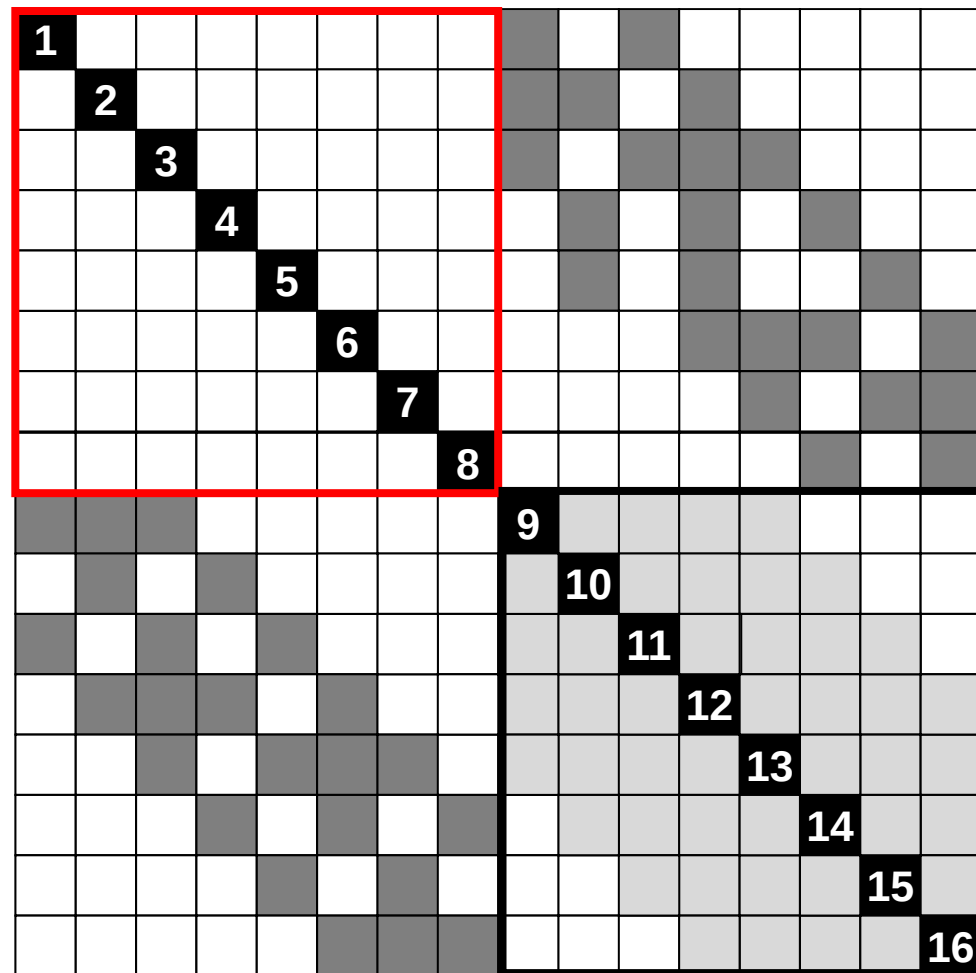


■ Non-zero, ■ Fill-in

Red-Black (2-Colors)

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

Bandwidth 10
 Profile 77
 Fill-in 44

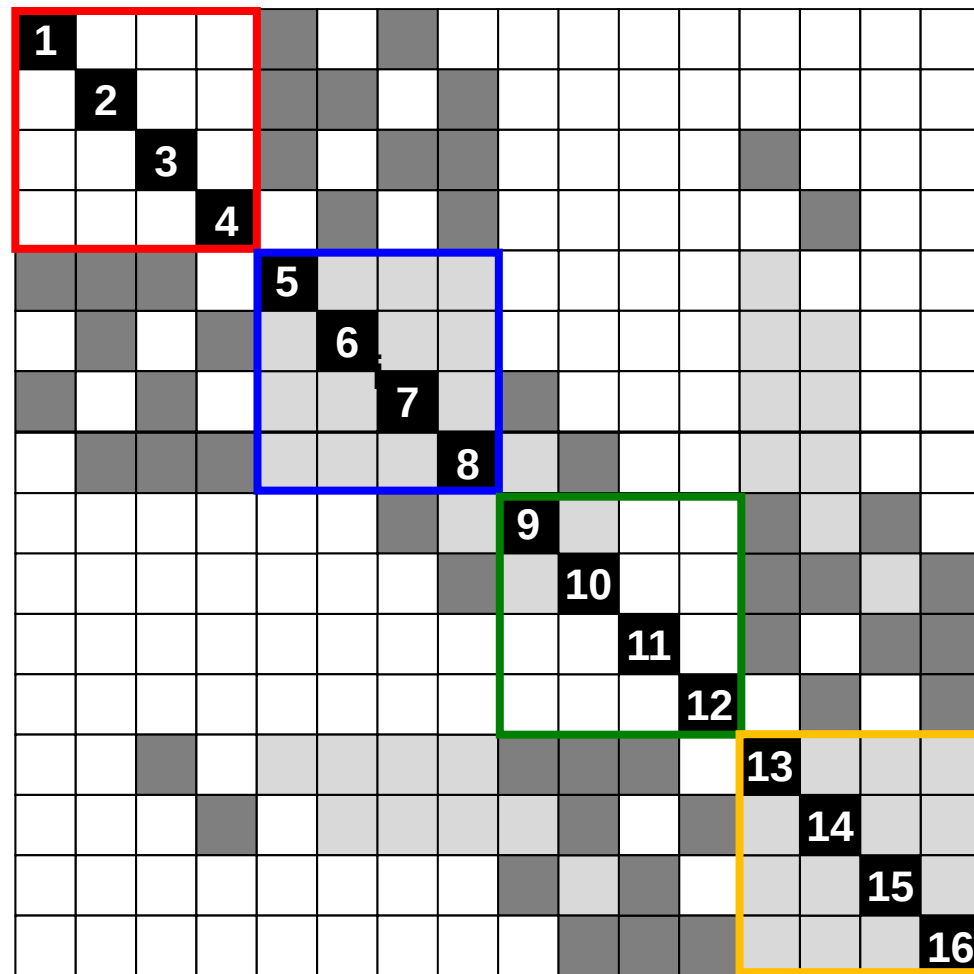


■ Non-zero, ■ Fill-in

MC (4-Colors)

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

Bandwidth 10
 Profile 57
 Fill-in 46

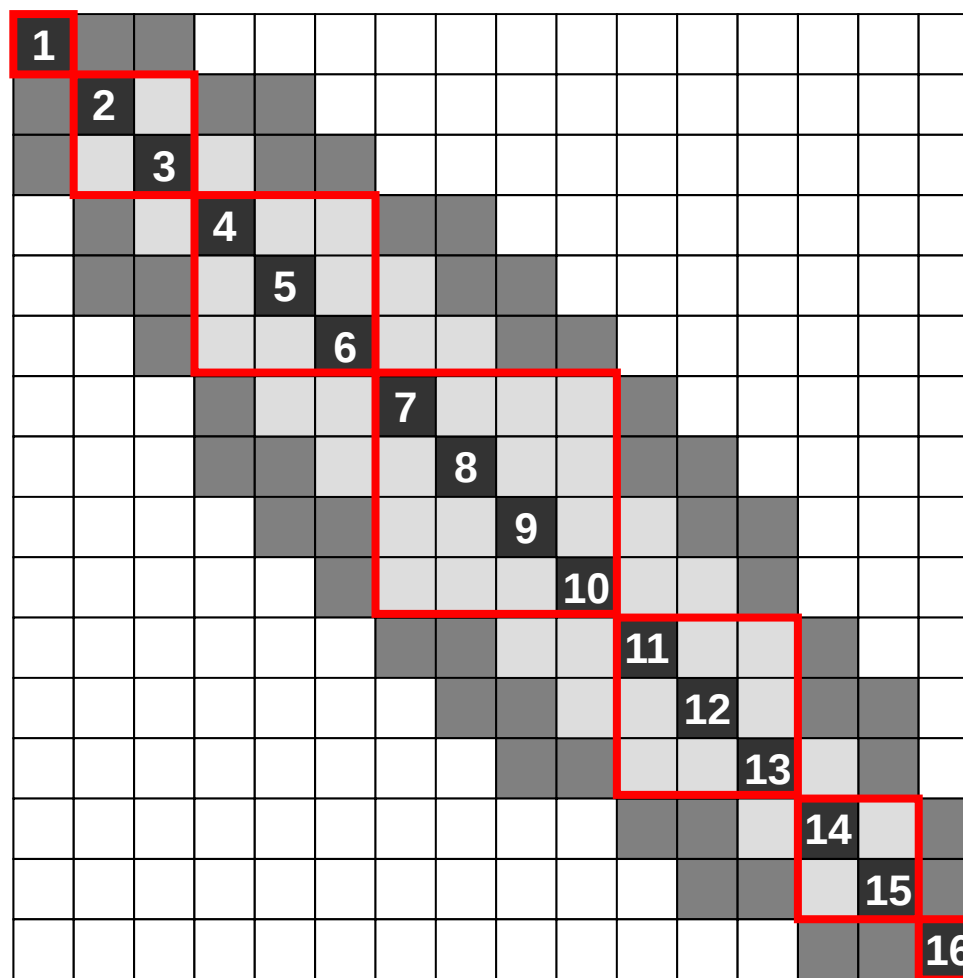


■ Non-zero, ■ Fill-in

RCM

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

Bandwidth 4
Profile 46
Fill-in 44

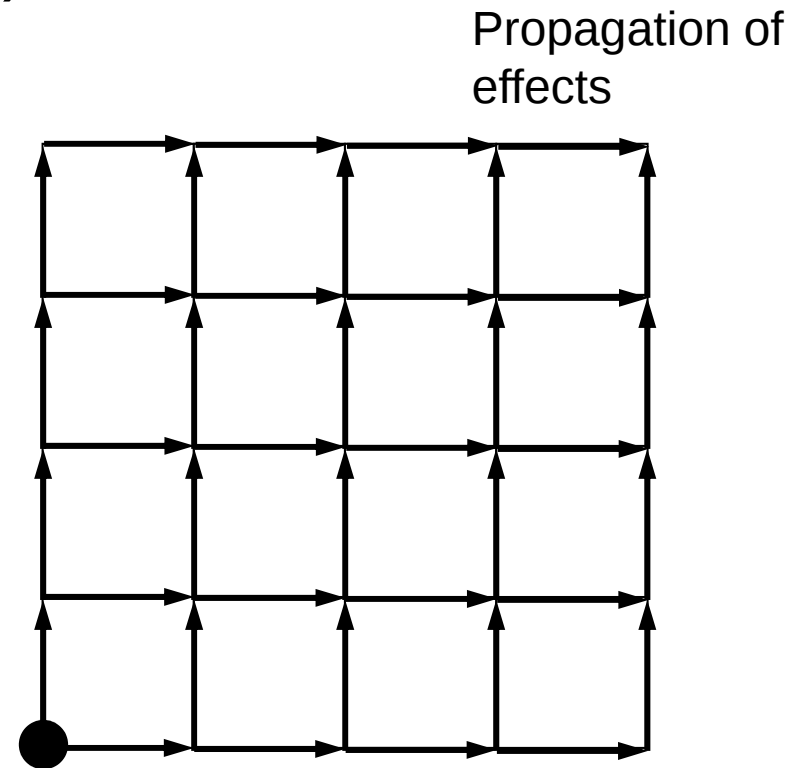
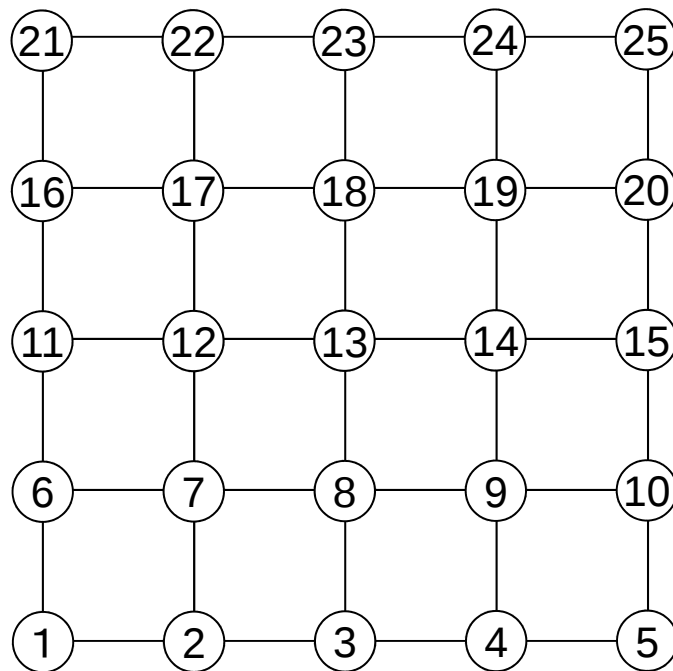


■ Non-zero, ■ Fill-in

Color Number and Convergence

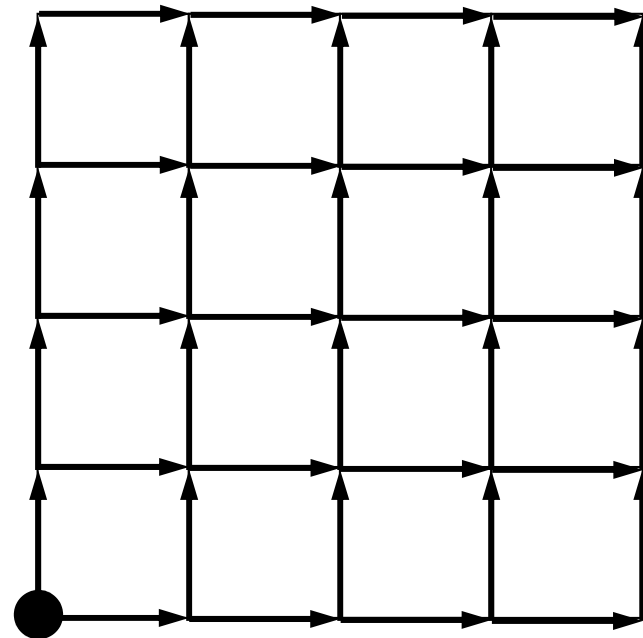
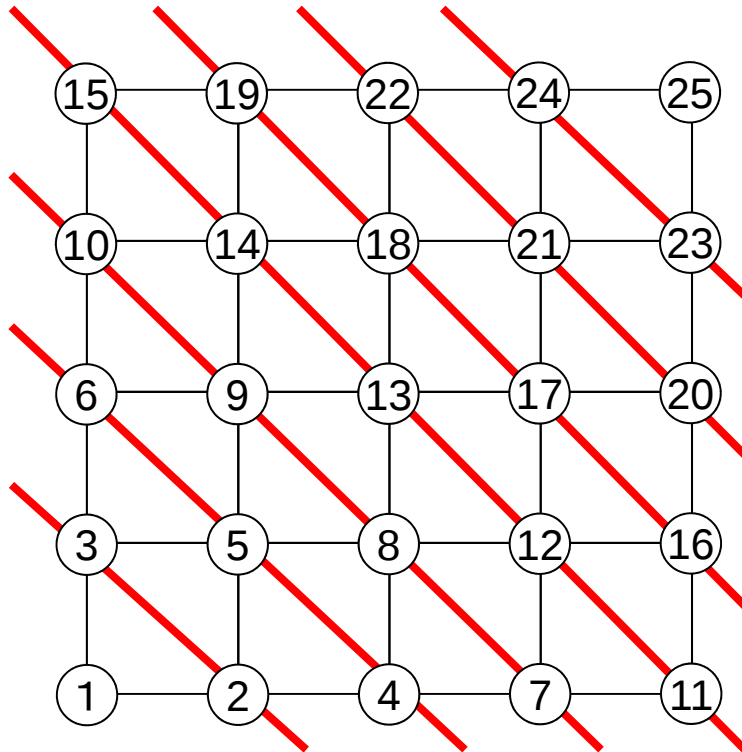
Incompatible Nodes

Doi, S. (NEC) et al.



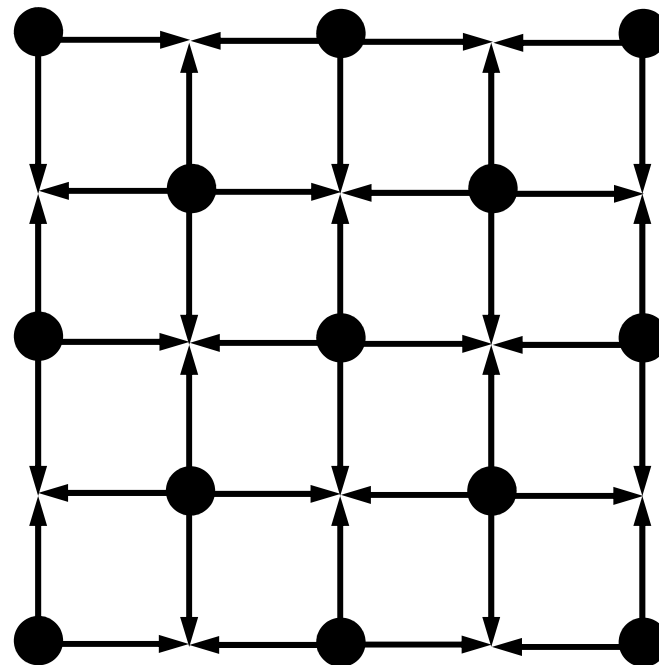
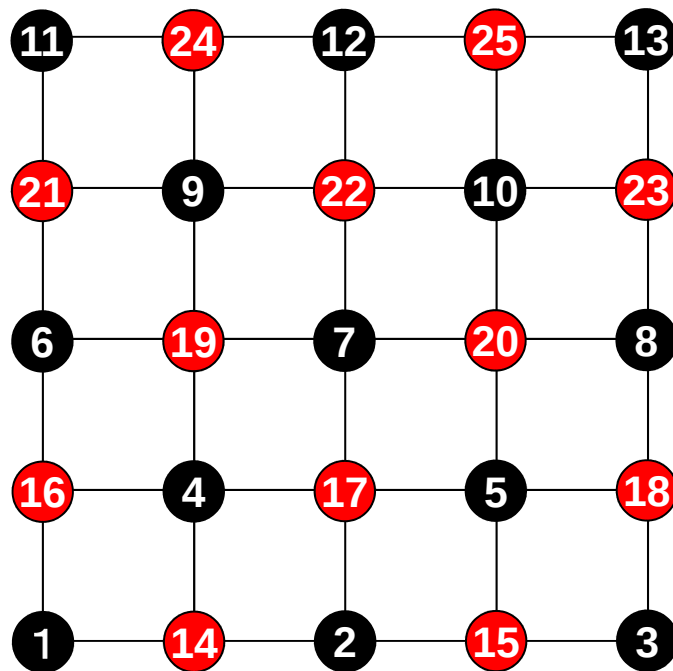
“Incompatible Nodes” not affected by other nodes. ID of such node is smaller than those of adjacent nodes. If we have smaller number of “incompatible nodes”, convergence is faster.

CM (Cuthill-McKee)



Red-Black

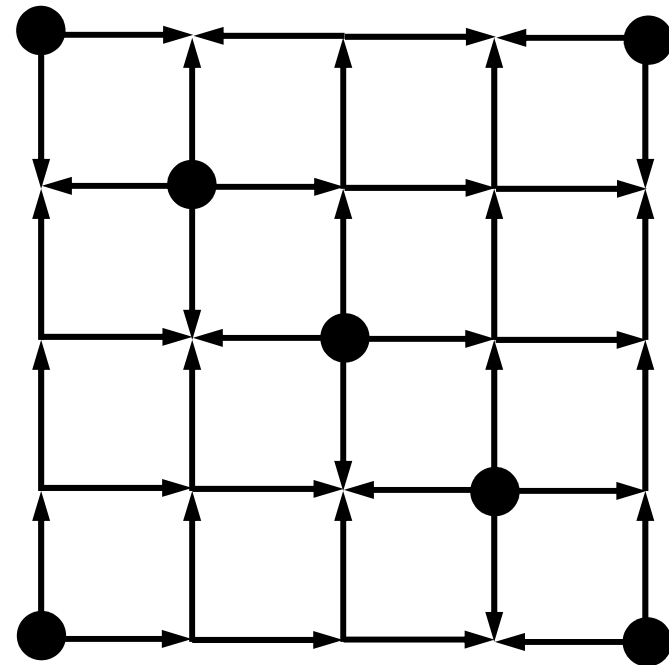
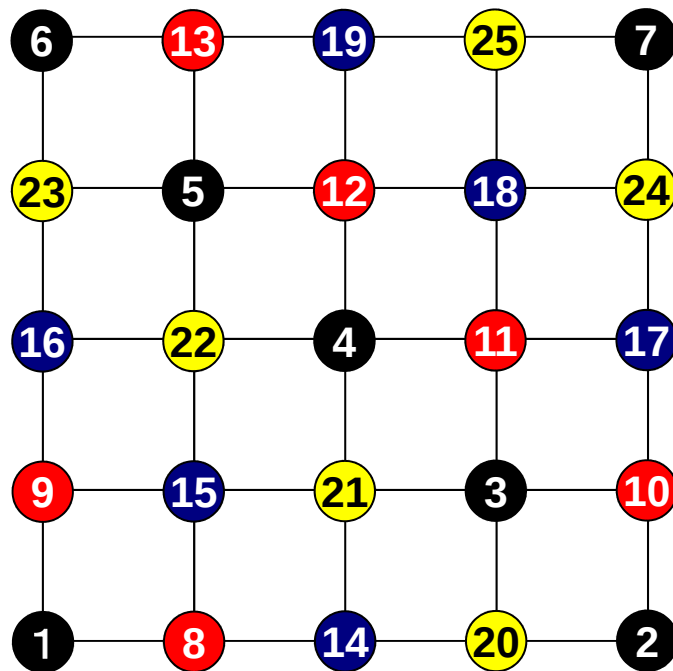
Higher parallelism, but many “incompatible nodes”
Slower convergence in ICCG, Gauss-Seidel etc.



4-Colors

Still many “incompatible nodes”

Slower convergence in ICCG, Gauss-Seidel etc.



**Generally speaking,
convergence of ICCG is better
for configurations with ...**

- **More Colors**
- Smaller Bandwidth
- Smaller Profile
- Fewer Fill-in's
- Fewer Incompatible Nodes

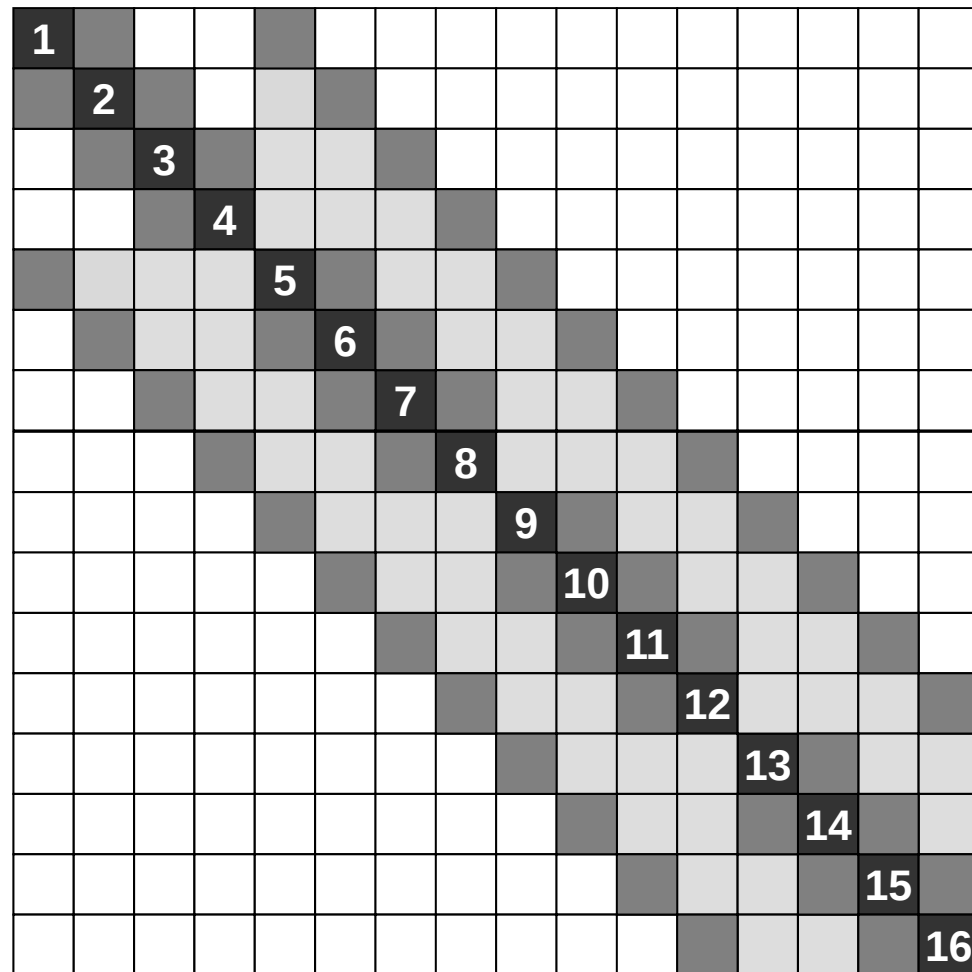
Generally speaking, convergence of ICCG is better for configurations with fewer fill-in's

- ICCG (IC(0)-CG) ignores fill-in's
- **More fill-in's are ignored for configurations (e.g. coloring) with more fill-in's**
 - **IC(0) for configurations with more fill-in's may be weaker than that with fewer fill-in's**
- **Distribution of fill-in's may affect the convergence**
 - **Initial Matrix**
 - **Red-Black (MC with 2-Colors)**

Initial Matrix

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

Bandwidth 4
Profile 51
Fill-in 54

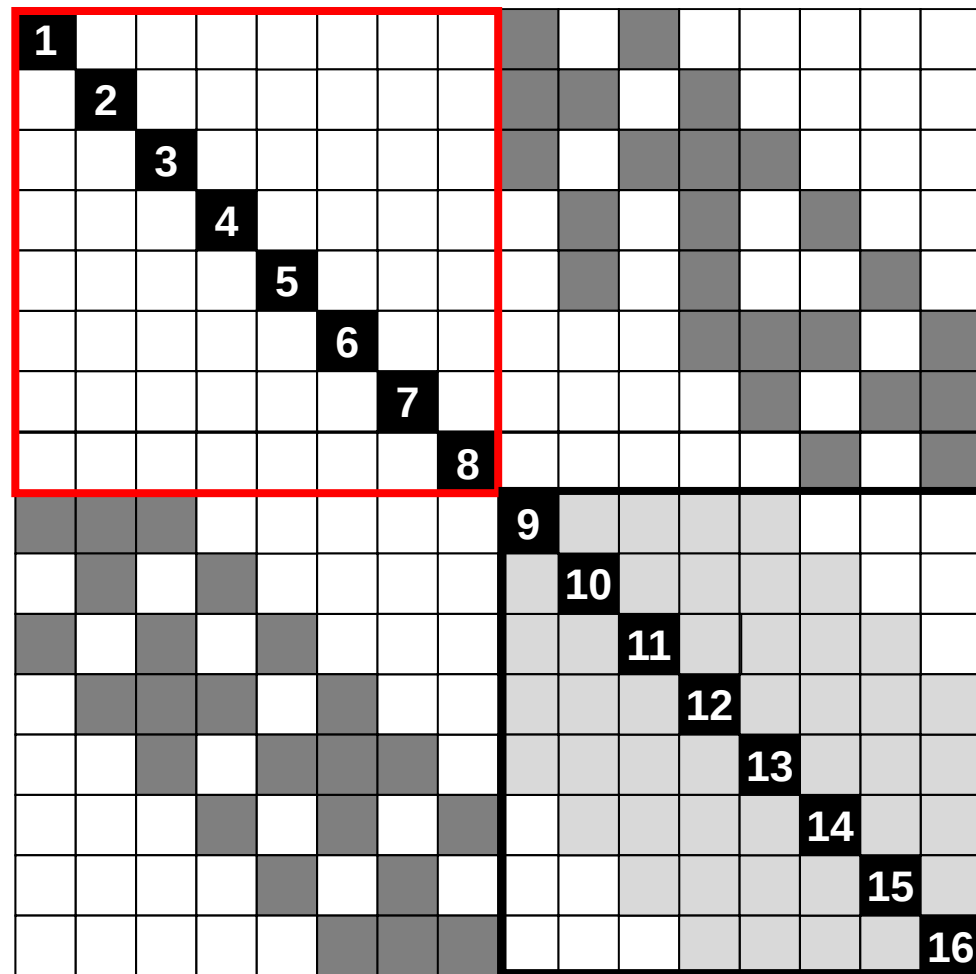


■ Non-zero, ■ Fill-in

Red-Black (2-Colors)

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

Bandwidth 10
 Profile 77
 Fill-in 44



■ Non-zero, ■ Fill-in

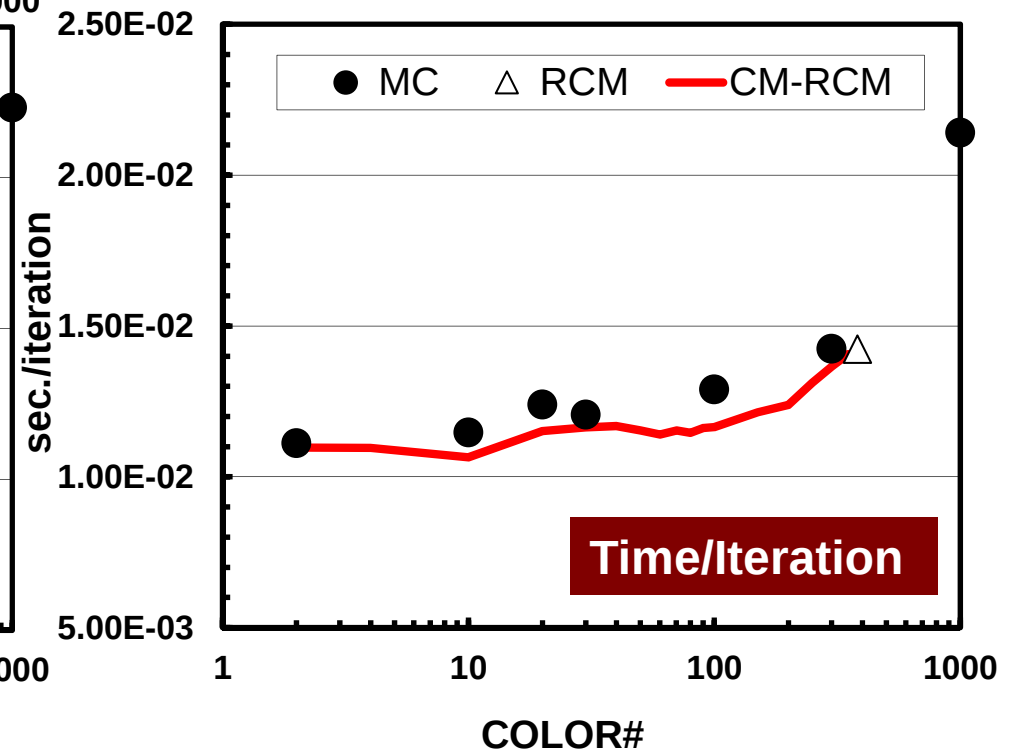
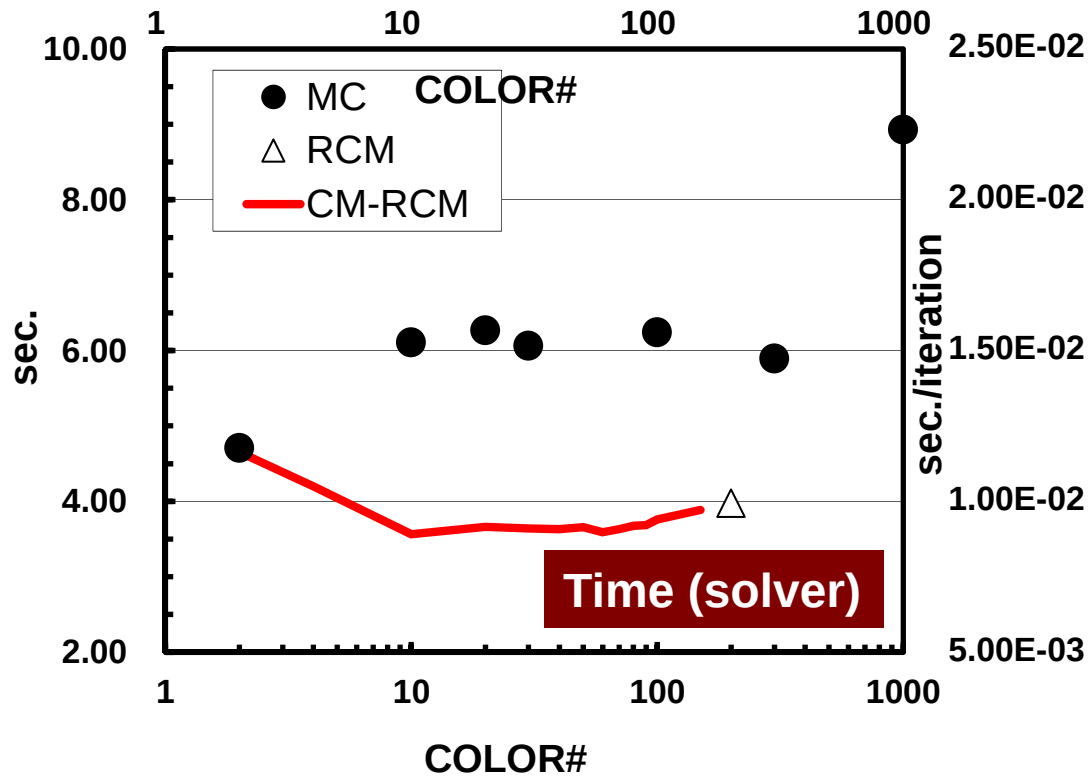
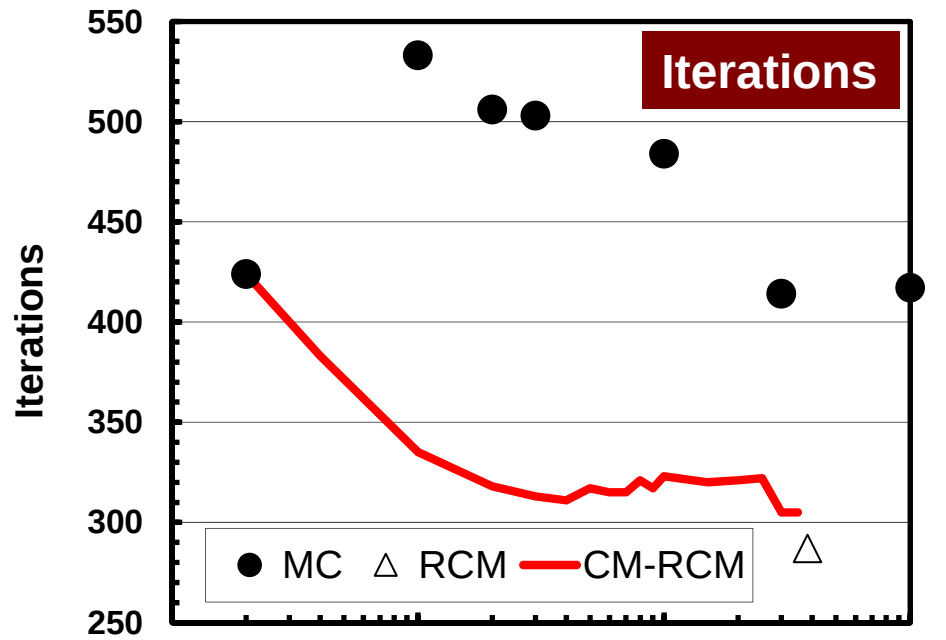
Effect of Reordering/Color # on Convergence

- Other effects (e.g. B.C.) should be considered.
- It is difficult to provide very general remarks.
- e.g. RCM provides slightly faster convergence than CM, although parameters (Bandwidth, profile, fill-in's) are same.

Effect of Reordering

- Reordering changes sequence of matrix operations, and sometimes improves convergence.
 - Parallelism, Faster Convergence
- We need some kind of reordering for parallel ICCG on such simple meshes described in examples.
- Notice
 - Reordering may change results.
 - We need deep insight and understanding on background physics and mathematics.

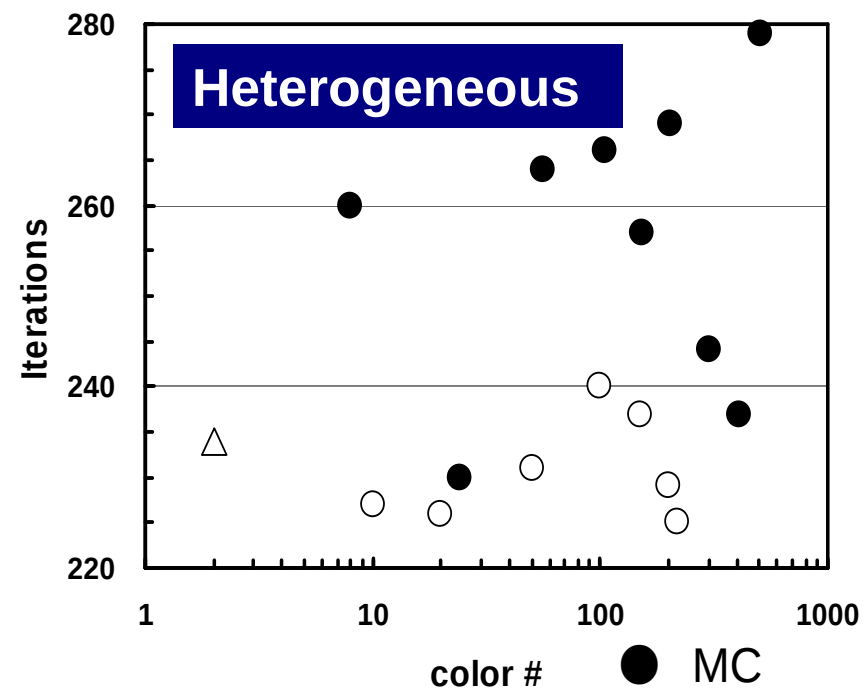
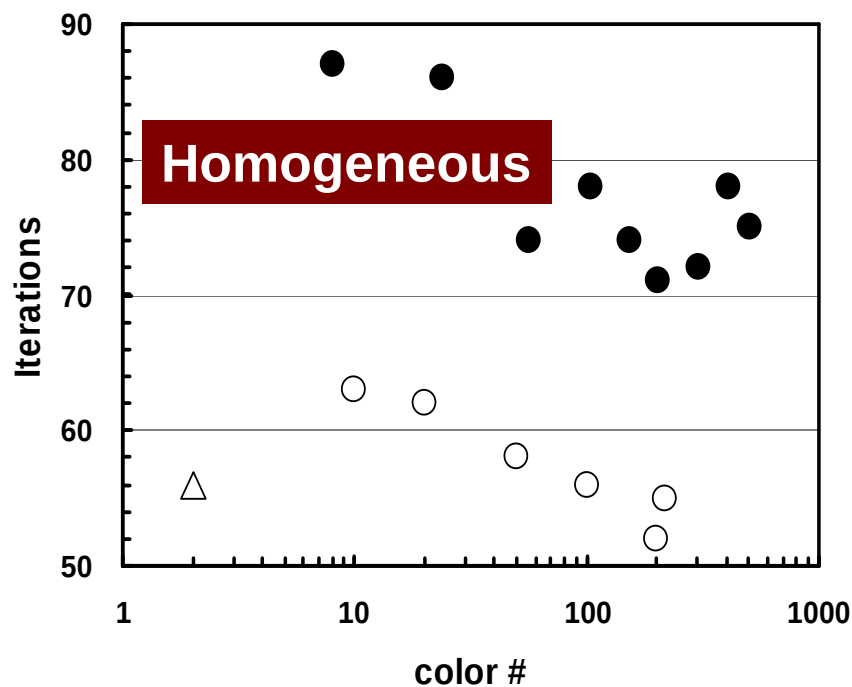
Reedbush-U, 1-
node/18-cores,
 128^3
(●: MC, △: RCM, -: CM-RCM)



Comparison of Reordering Methods

3D Linear Elastic Problems

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



3D Linear-Elastic Problems with 32,768 DOF

- MC
- CM-RCM
- △ No reordering

- Remedy for Data Dependency
- Ordering/Reordering
 - Red-Black, Multicoloring (MC)
 - Cuthill-McKee (CM), Reverse-CM (RCM)
 - Reordering and Convergence
- **Implementation**
- ICCG with Reordering

Implementation: L2-color (1/2)

- Program for Coloring
 - MC, CM, RCM, and CM-RCM

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

4 4 1 **2D geometry with 16 meshes**

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

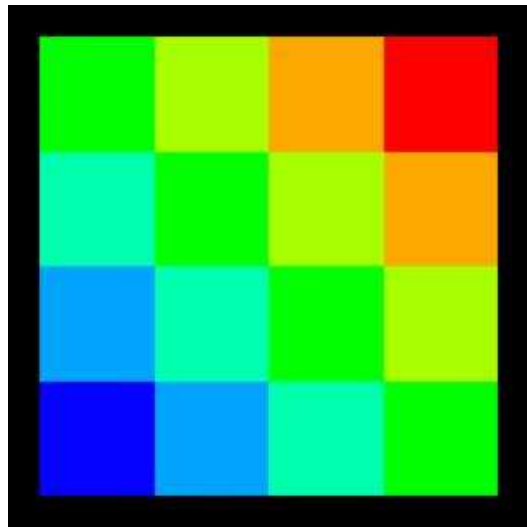
CMRCM if #COLOR .le.-2

=>

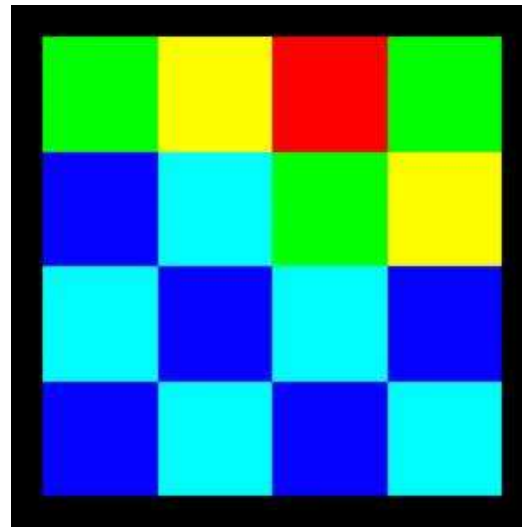
13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Results: L2-color

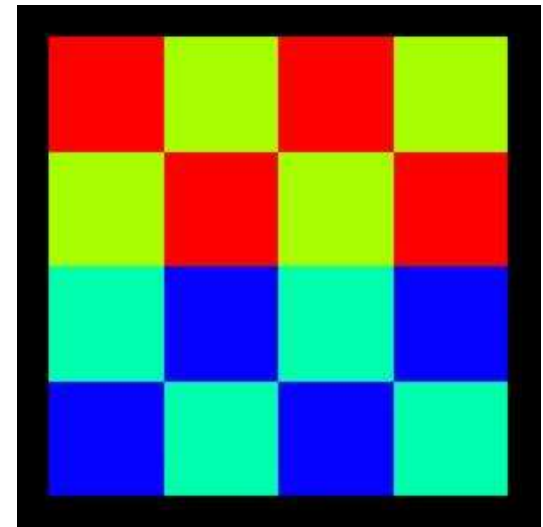
- 2 files are created
 - color.log Table of OLD-to-NEW mesh ID
Information of the matrix
 - color.inp Colors/levels of meshes (ParaView)



INPUT: 0
(CM, 7 levels)



INPUT: 3
(MC, 5 colors)



INPUT: 4
(MC, 4 colors)

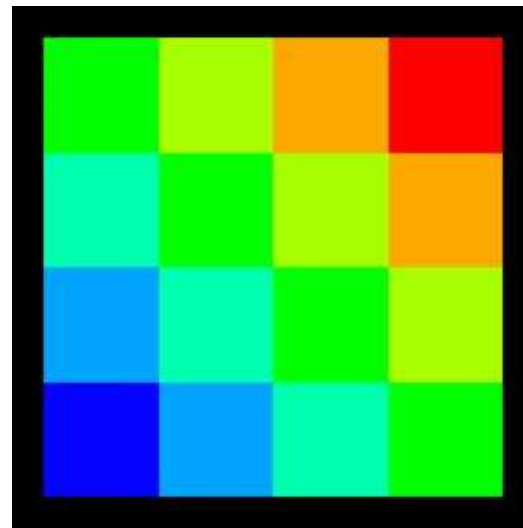
INPUT=0: CM, 7-Levels

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



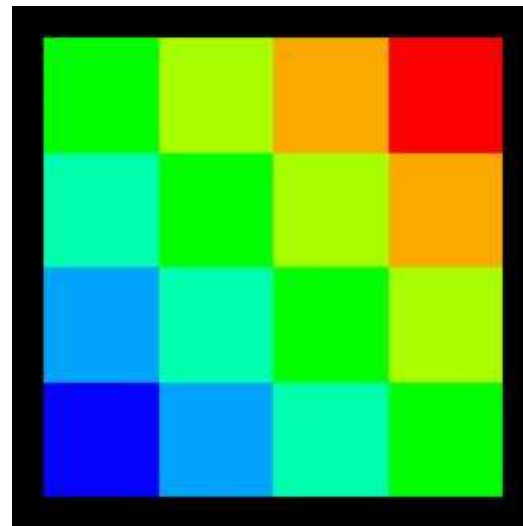
INPUT=0: CM, 7-Levels

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



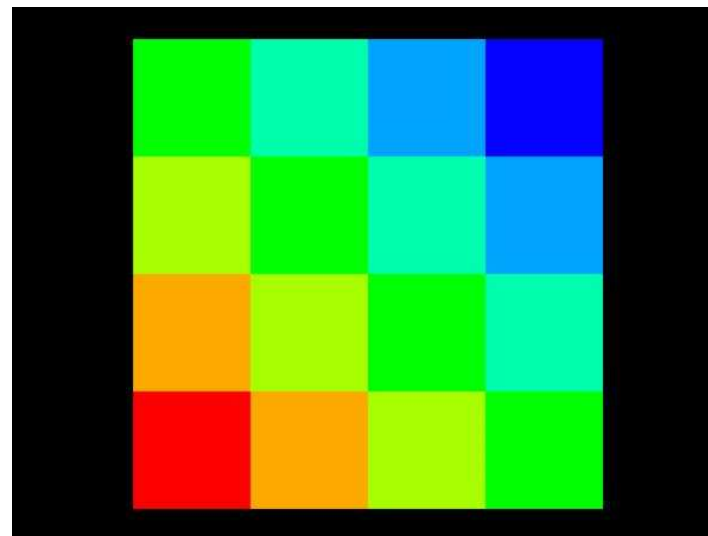
INPUT=-1: RCM, 7-Levels

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



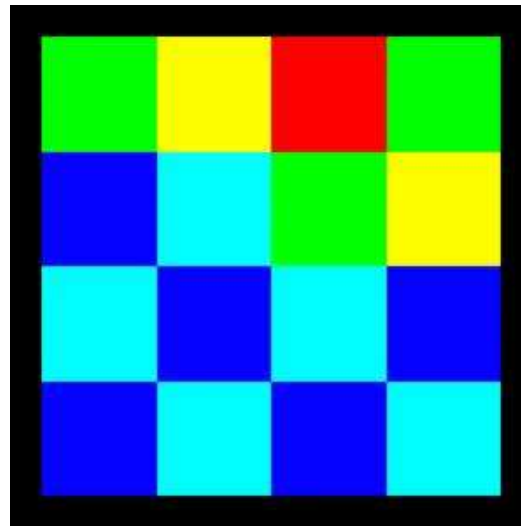
INPUT=3: MC, 5-Colors

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



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

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

5 independent meshes

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

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

5 independent meshes

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

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

Proceed to the next color, if no more independent meshes.

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

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

Proceed to the next color, if no more independent meshes.

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

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

Finally, 5 colors are needed.

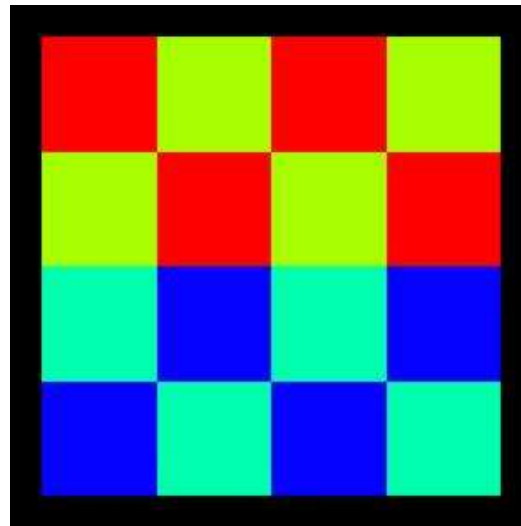
INPUT=4: MC, 4-Colors

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



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:

```

Source Files: L2-color

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

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

Target geometry to be colored

Main Program

```

program MAIN

use STRUCT
use PCG

implicit REAL*8 (A-H, O-Z)

call POINTER_INIT
call POI_GEN

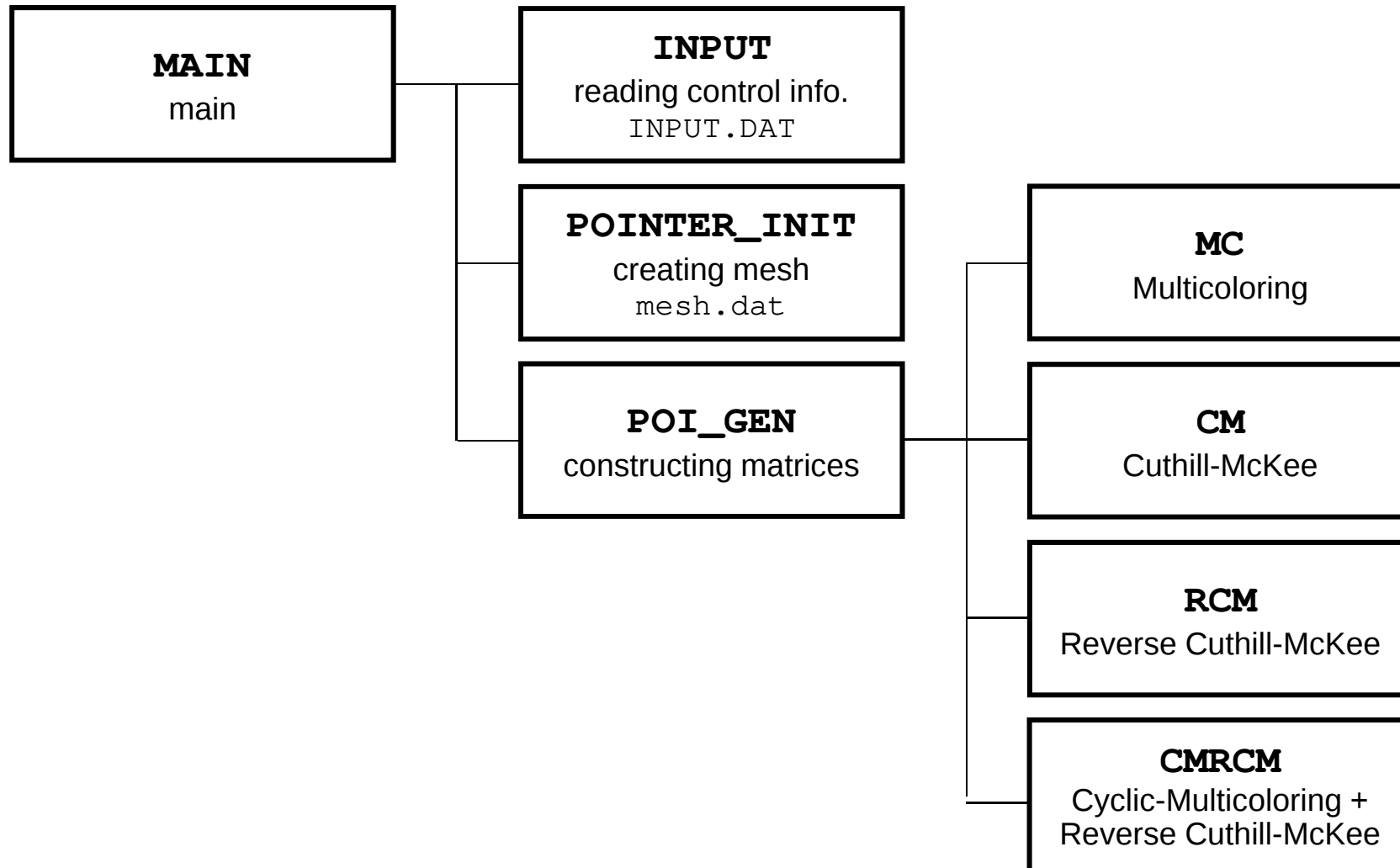
call OUTUCD

open (21, file='color.log', status='unknown')
  write (21, '(//,a,i8,/)' ) 'COLOR number', NCOLORTot
  do ic= 1, NCOLORTot
    do i= COLORindex(ic-1)+1, COLORindex(ic)
      write (21, '(3(a,i8))' ) ,    #new', i, '    #old', NEWtoOLD(i), &
&      color', ic
    enddo
  enddo
close (21)

stop
end

```

Structure of L2-color



Main Program

```

program MAIN

  use STRUCT
  use PCG

  implicit REAL*8 (A-H, O-Z)

  call POINTER_INIT
  call POI_GEN

  call OUTUCD

  open (21, file='color.log', status='unknown')
    write (21, '(//,a,i8,/)' ) 'COLOR number', NCOLORTot
    do ic= 1, NCOLORTot
      do i= COLORindex(ic-1)+1, COLORindex(ic)
        write (21, '(3(a,i8))' ) ,   #new', i, '    #old', NEWtoOLD(i), &
&                                     color', ic
      enddo
    enddo
  close (21)

  stop
end

```

module STRUCT

```

module STRUCT

  include 'precision.inc'

!C
!C--- METRICs & FLUX
  integer (kind=kint) :: ICELTOT, ICELTOTp, N
  integer (kind=kint) :: NX, NY, NZ, NXP1, NYP1, NZP1, IBNODTOT
  integer (kind=kint) :: NXc, NYc, NZc

  real (kind=kreal) ::
&    DX, DY, DZ, XAREA, YAREA, ZAREA, RDX, RDY, RDZ,
&    RDX2, RDY2, RDZ2, R2DX, R2DY, R2DZ

  real (kind=kreal), dimension(:), allocatable ::
&    VOLCEL, VOLNOD, RVC, RVN

  integer (kind=kint), dimension(:,:), allocatable ::
&    XYZ, NEIBcell

!C
!C--- BOUNDARYs
  integer (kind=kint) :: ZmaxCELTot
  integer (kind=kint), dimension(:), allocatable :: BC_INDEX, BC_NOD
  integer (kind=kint), dimension(:), allocatable :: ZmaxCEL

!C
!C--- WORK
  integer (kind=kint), dimension(:,:), allocatable :: IWKX
  real (kind=kreal), dimension(:,:), allocatable :: FCV

end module STRUCT

```

ICELTOT:

Number of meshes (NX x NY x 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 x NYP1

XYZ (ICELTOT, 3):

Location of meshes

NEIBcell (ICELTOT, 6):

Neighboring meshes

module PCG

```

module PCG

integer, parameter :: N2= 256
integer :: NUmax, NLmax, NCOLORTtot, NCOLORK, NU, NL

real(kind=8) :: EPSICCG

real(kind=8), dimension(:), allocatable :: D, PHI, BFORCE
real(kind=8), dimension(:, :), allocatable :: AL, AU

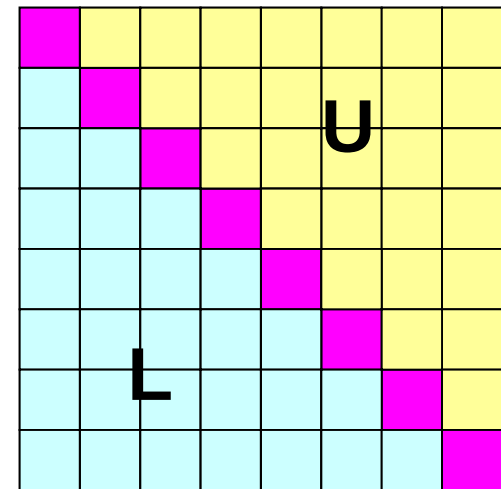
integer, dimension(:), allocatable :: INL, INU, COLORindex
integer, dimension(:), allocatable :: OLDtoNEW, NEWtoOLD

integer, dimension(:, :), allocatable :: IAL, IAU

end module PCG

```

- Sparse Matrix
- Only non-zero off-diagonal components (CRS)
- Diagonal/Lower/Upper components are stored separately



module PCG

```

module PCG

integer, parameter :: N2= 256
integer :: NUmax, NLmax, NCOLORTot, NCOLORk, NU, NL

real(kind=8) :: EPSICCG

real(kind=8), dimension(: ), allocatable :: D, PHI, BFORCE
real(kind=8), dimension(:, :), allocatable :: AL, AU

integer, dimension(:), allocatable :: INL, INU, COLORindex
integer, dimension(:), allocatable :: OLDtoNEW, NEWtoOLD

integer, dimension(:, :), allocatable :: IAL, IAU

end module PCG

```

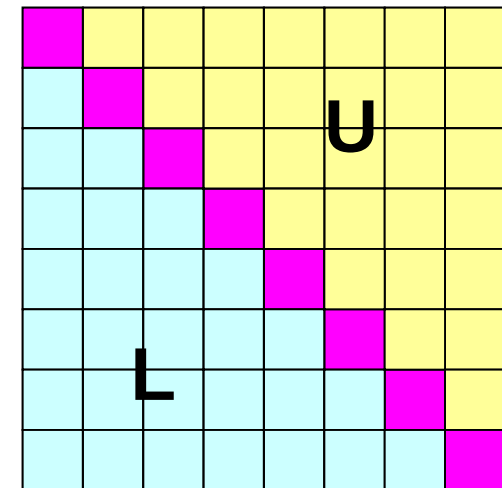
Lower Part (Column ID)

IAL(icou, i) < i

Upper Part (Column ID)

IAU(icou, i) > i

INL (ICELTOT)	# Non-zero off-diag. components (lower)
IAL (NL, ICELTOT)	Col. ID: non-zero off-diag. comp. (lower)
INU (ICELTOT)	# Non-zero off-diag. components (upper)
IAU (NU, ICELTOT)	Col. ID: non-zero off-diag. comp. (upper)
NU, NL	Max # of L/U non-zero off-diag. comp.s (=6)
NCOLORTot	Total number of colors/levels
COLORindex	Index of number of meshes in each color/level
(0:NCOLORTot)	(COLORindex(icol) - COLORindex(icol-1))
OLDtoNEW, NEWtoOLD	Reference table before/after renumbering



module PCG

```

module PCG

integer, parameter :: N2= 256
integer :: NUmax, NLmax, NCOLORTot, NCOLORk, NU, NL

real(kind=8) :: EPSICCG

real(kind=8), dimension(: ), allocatable :: D, PHI, BFORCE
real(kind=8), dimension(:, :), allocatable :: AL, AU

integer, dimension(:), allocatable :: INL, INU, COLORindex
integer, dimension(:), allocatable :: OLDtoNEW, NEWtoOLD

integer, dimension(:, :), allocatable :: IAL, IAU

end module PCG

```

Lower Part (Column ID)

IAL(icou, i) < i

INL(i) : Number@each row

Upper Part (Column ID)

IAU(icou, i) > i

INU(i) : Number@each row

INL(ICELTOT)

IAL(NL, ICELTOT)

INU(ICELTOT)

IAU(NU, ICELTOT)

NU, NL

NCOLORTot

COLORindex

(0:NCOLORTot)

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)

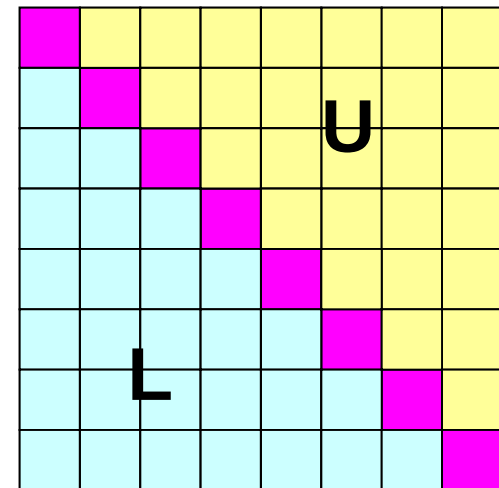
Total number of colors/levels

Index of number of meshes in each color/level

(COLORindex(icol) - COLORindex(icol-1))

OLDtoNEW, NEWtoOLD

Reference table before/after renumbering



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

```

```

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

```

Main Program

```

program MAIN

use STRUCT
use PCG

implicit REAL*8 (A-H, O-Z)

call POINTER_INIT
call POI_GEN

call OUTUCD

open (21, file='color.log', status='unknown')
  write (21, '(//, a, i8, /)') 'COLOR number', NCOLORTot
  do ic= 1, NCOLORTot
    do i= COLORindex(ic-1)+1, COLORindex(ic)
      write (21, '(3(a, i8))') '#new', i, '#old', NEWtoOLD(i), &
&      color', ic
    enddo
  enddo
close (21)

stop
end

```

pointer_init (1/3)

```

!C
!C***
!C*** POINTER_INIT
!C***
!C
      subroutine POINTER_INIT
      use STRUCT
      use PCG
      implicit REAL*8 (A-H,O-Z)

!C
!C +-----+
!C | Generating MESH info. |
!C +-----+
!C==
      write (*,*) 'NX/NY/NZ ?'
      read  (*,*)  NX, NY, NZ

      ICELTOT= NX  * NY  * NZ

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

      allocate (NEIBcell(ICELTOT,6), XYZ(ICELTOT,3))
      NEIBcell= 0

```

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 :

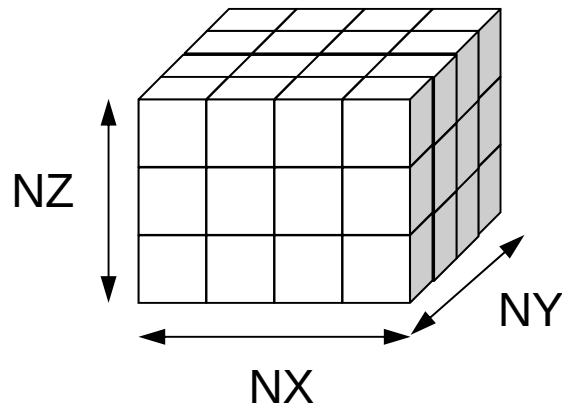
Number of meshes (NX x NY x NZ)

XYZ (ICELTOT, 3) :

Location of meshes

NEIBcell (ICELTOT, 6) :

Neighboring meshesc



pointer_init (2/3)

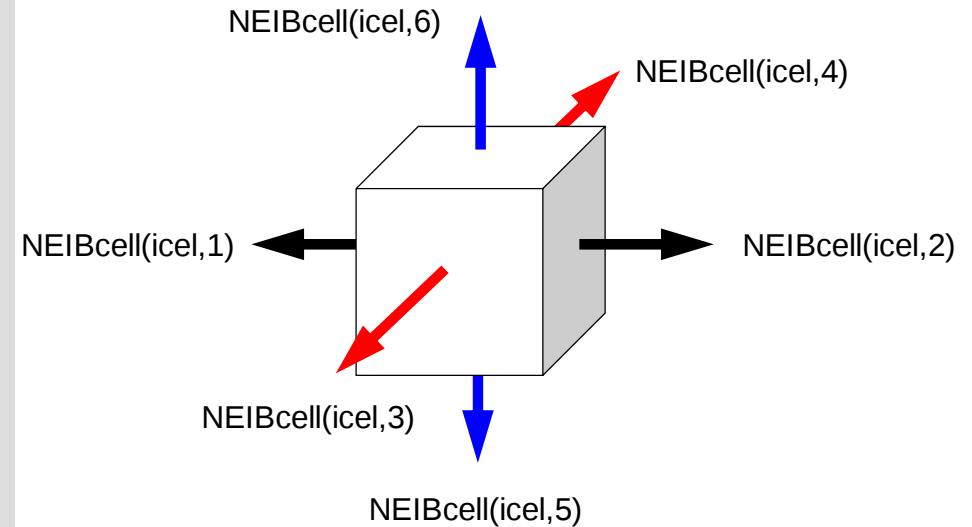
```

do k= 1, NZ
  do j= 1, NY
    do i= 1, NX
      icel= (k-1)*NX*NY + (j-1)*NX + i
      NEIBcell(icel, 1)= icel - 1
      NEIBcell(icel, 2)= icel + 1
      NEIBcell(icel, 3)= icel - NX
      NEIBcell(icel, 4)= icel + NX
      NEIBcell(icel, 5)= icel - NX*NY
      NEIBcell(icel, 6)= icel + NX*NY
      if (i. eq. 1) NEIBcell(icel, 1)= 0
      if (i. eq. NX) NEIBcell(icel, 2)= 0
      if (j. eq. 1) NEIBcell(icel, 3)= 0
      if (j. eq. NY) NEIBcell(icel, 4)= 0
      if (k. eq. 1) NEIBcell(icel, 5)= 0
      if (k. eq. NZ) NEIBcell(icel, 6)= 0

      XYZ(icel, 1)= i
      XYZ(icel, 2)= j
      XYZ(icel, 3)= k

    enddo
  enddo
enddo
!C===

```



i= XYZ(icel,1)
j= XYZ(icel,2), k= XYZ(icel,3)
icel= (k-1)*NX*NY + (j-1)*NX + i

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

pointer_init (3/3)

```
!C
!C +-----+
!C | Parameters |
!C +-----+
!C==
      if (DX.le.0.0e0) then
        DX= 1.d0 / dfloat(NX)
        DY= 1.d0 / dfloat(NY)
        DZ= 1.d0 / dfloat(NZ)
      endif

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

      IBNODTOT= NXP1 * NYP1
      N        = NXP1 * NYP1 * NZP1
!C==
      return
      end
```

Main Program

```

program MAIN

use STRUCT
use PCG

implicit REAL*8 (A-H, O-Z)

call POINTER_INIT
call POI_GEN

call OUTUCD

open (21, file='color.log', status='unknown')
  write (21, '(//, a, i8, /)') 'COLOR number', NCOLORTot
  do ic= 1, NCOLORTot
    do i= COLORindex(ic-1)+1, COLORindex(ic)
      write (21, '(3(a, i8))') '#new', i, '#old', NEWtoOLD(i), &
&      color', ic
    enddo
  enddo
close (21)

stop
end

```

poi_gen (1/4)

```
subroutine POI_GEN

  use STRUCT
  use PCG

  implicit REAL*8 (A-H, O-Z)

!C
!C--- INIT.
  nn = ICELTOT

  NU= 6
  NL= 6

  allocate (INL(nn), INU(nn), IAL(NL, nn), IAU(NU, nn))

  INL= 0
  INU= 0
  IAL= 0
  IAU= 0
```

```

!C
!C +-----+
!C | CONNECTIVITY |
!C +-----+
!C==
do icel= 1, ICELTOT
  icN1= NEIBcell(icel,1)
  icN2= NEIBcell(icel,2)
  icN3= NEIBcell(icel,3)
  icN4= NEIBcell(icel,4)
  icN5= NEIBcell(icel,5)
  icN6= NEIBcell(icel,6)

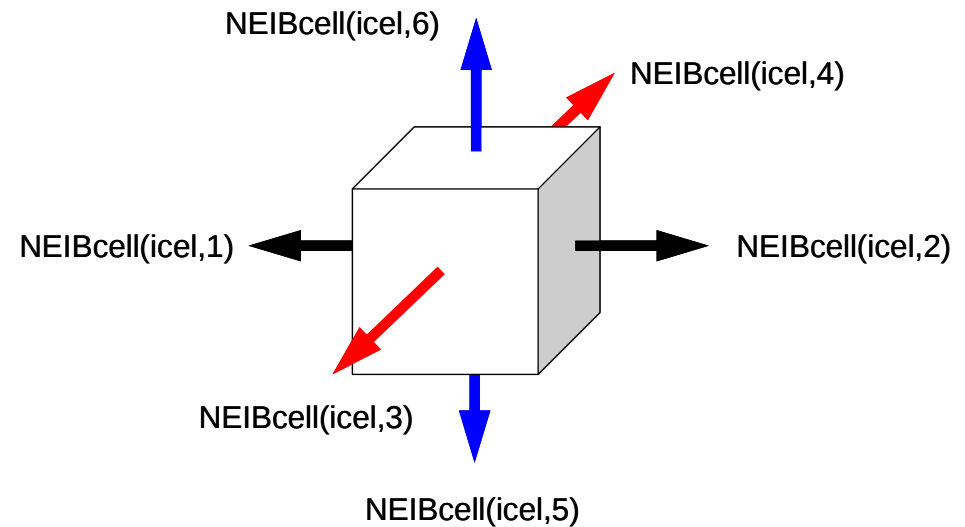
  icouG= 0
  if (icN5.ne.0.and.icN5.le.ICELTOT) then
    icou= INL(icel) + 1
    IAL(icou,icel)= icN5
    INL(      icel)= icou
  endif

  if (icN3.ne.0.and.icN3.le.ICELTOT) then
    icou= INL(icel) + 1
    IAL(icou,icel)= icN3
    INL(      icel)= icou
  endif

  if (icN1.ne.0.and.icN1.le.ICELTOT) then
    icou= INL(icel) + 1
    IAL(icou,icel)= icN1
    INL(      icel)= icou
  endif
endif

```

poi_gen (2/4)



Lower Triangular Part

```

NEIBcell(icel,5)= icel - NX*NY
NEIBcell(icel,3)= icel - NX
NEIBcell(icel,1)= icel - 1

```

```

!C
!C +-----+
!C | CONNECTIVITY |
!C +-----+
!C===
do icel= 1, ICELTOT
  icN1= NEIBcell(icel,1)
  icN2= NEIBcell(icel,2)
  icN3= NEIBcell(icel,3)
  icN4= NEIBcell(icel,4)
  icN5= NEIBcell(icel,5)
  icN6= NEIBcell(icel,6)

  ...

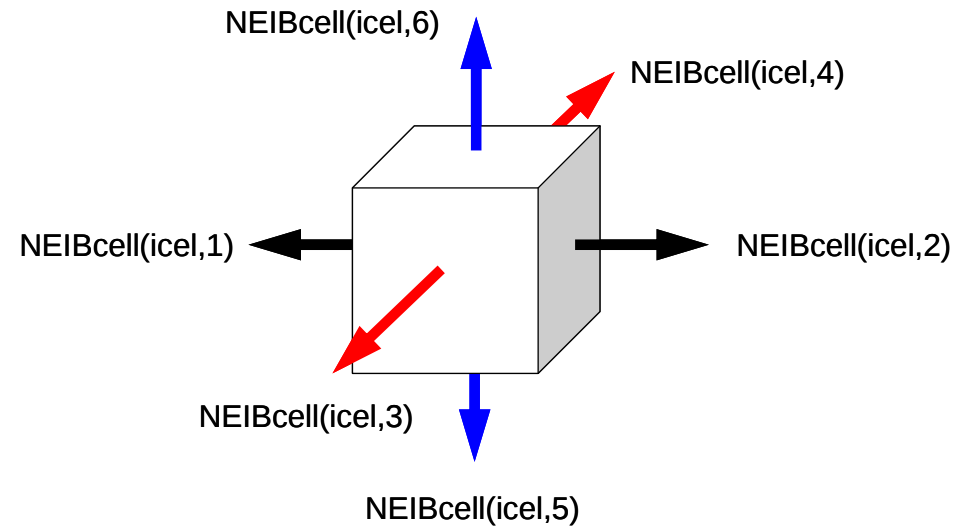
  if (icN2.ne.0.and.icN2.le.ICELTOT) then
    icou= INU(icel) + 1
    IAU(icou,icel)= icN2
    INU(      icel)= icou
  endif

  if (icN4.ne.0.and.icN4.le.ICELTOT) then
    icou= INU(icel) + 1
    IAU(icou,icel)= icN4
    INU(      icel)= icou
  endif

  if (icN6.ne.0.and.icN6.le.ICELTOT) then
    icou= INU(icel) + 1
    IAU(icou,icel)= icN6
    INU(      icel)= icou
  endif
enddo
!C===

```

poi_gen (3/4)



Uppr Triangular Part

```

NEIBcell(icel,2)= icel + 1
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,6)= icel + NX*NY

```

poi_gen (4/4)

```

!C
!C +-----+
!C | MULTICOLORING |
!C +-----+
!C==
111 continue
   write (*, '(//a, i8, a)') 'You have', ICELTOT, ' elements.'
   write (*, '( a      )') 'How many colors do you need ?'
   write (*, '( a      )') ' #COLOR must be more than 2 and'
   write (*, '( a, i8  )') ' #COLOR must not be more than', ICELTOT
   write (*, '( a      )') ' if #COLOR=0 : CM ordering'
   write (*, '( a      )') ' if #COLOR<0 : RCM ordering'
   write (*, '( a      )') '=>'
   read (*, *) NCOLORTot
   if (NCOLORTot.eq.1.or.NCOLORTot.gt.ICELTOT) goto 111

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

   if (NCOLORTot.gt.0) then
      call MC (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&          NCOLORTot, COLORindex, NEWtoOLD, OLDtoNEW)
&
   endif
   if (NCOLORTot.eq.0) then
      call CM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&          NCOLORTot, COLORindex, NEWtoOLD, OLDtoNEW)
&
   endif
   if (NCOLORTot.lt.0) then
      call RCM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&          NCOLORTot, COLORindex, NEWtoOLD, OLDtoNEW)
&
   endif
!C==

   write (*, '(//a, i8)') '# TOTAL COLOR number', NCOLORTot

```

Reading “initial” color
number

poi_gen (4/4)

```

!C
!C +-----+
!C | MULTICOLORING |
!C +-----+
!C===
111 continue
write (*, '(//a, i8, a)') 'You have', ICELTOT, ' elements.'
write (*, '( a      )') 'How many colors do you need ?'
write (*, '( a      )') ' #COLOR must be more than 2 and'
write (*, '( a, i8  )') ' #COLOR must not be more than', ICELTOT
write (*, '( a      )') ' if #COLOR=0 : CM ordering'
write (*, '( a      )') ' if #COLOR<0 : RCM ordering'
write (*, '( a      )') '=>'
read (*, *) NCOLORTot
if (NCOLORTot.eq.1.or.NCOLORTot.gt.ICELTOT) goto 111

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

if (NCOLORTot.gt.0) then
  call MC (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&          NCOLORTot, COLORindex, NEWtoOLD, OLDtoNEW)
endif
if (NCOLORTot.eq.0) then
  call CM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&          NCOLORTot, COLORindex, NEWtoOLD, OLDtoNEW)
endif
if (NCOLORTot.lt.0) then
  call RCM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&           NCOLORTot, COLORindex, NEWtoOLD, OLDtoNEW)
endif
!C===

write (*, '(//a, i8)') '# TOTAL COLOR number', NCOLORTot

```

Allocate matrices

poi_gen (4/4)

```

!C
!C +-----+
!C | MULTICOLORING |
!C +-----+
!C==
...
      if (NCOLORtot.gt.0) then
        call MC (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&              NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW)
      endif
      if (NCOLORtot.eq.0) then
        call CM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&              NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW)
      endif
      if (NCOLORtot.lt.0) then
        call RCM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&               NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW)
      endif
!C==

      write (*,' (/a, i8)') '# TOTAL COLOR number', NCOLORtot

```

INL, INU, IAL, IAU
OLDtoNEW, NEWtoOLD
NCOLORtot
COLORindex (0:NCOLORtot)

Info. after renumbering
 Reference table before/after renumbring
 Final number of colors (g.e. initial number)
 Meshes from **COLORindex(ic-1)+1** to
COLORindex(ic) are in ic-th color.
 Meshes in same color are independent: Parallel
 processing can be applied.

COLORindex

COLORindex (0:NCOLORtot)

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

Meshes in same color are independent: Parallel processing can be applied.

```
do ic= 1, NCOLORtot
  do i= COLORindex(ic-1)+1, COLORindex(ic)
    write (21,*) i, NEWtoOLD(i), ic
  enddo
enddo
```

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

mc (1/8)

```

!C
!C***
!C*** MC
!C***
!C
!C   Multicolor Ordering Method
!C

      subroutine MC (N, NL, NU, INL, IAL, INU, IAU,          &
&                  NCOLortot, COLORindex, NEWtoOLD, OLDtoNEW)

      implicit REAL*8 (A-H, O-Z)

      integer, dimension(N)      :: INL, INU, NEWtoOLD, OLDtoNEW
      integer, dimension(0:N)    :: COLORindex
      integer, dimension(NL, N)  :: IAL
      integer, dimension(NU, N)  :: IAU

      integer, dimension(:)      , allocatable :: IW, INLw, INUw
      integer, dimension(:, :), allocatable :: IALw, IAUw

```

mc (2/8)

```

!C
!C +-----+
!C |  INIT.  |
!C +-----+
!C===
      allocate (IW(N))
      IW= 0

      NCOLORK = NCOLORTOT

      do i= 1, N
        NEWtoOLD (i)= i
        OLDtoNEW (i)= i
      enddo

      INmin= N
      NODmin= 0
      do i= 1, N
        icon= INL(i) + INU(i)
        if (icon.lt.INmin) then
          INmin= icon
          NODmin= i
        endif
      enddo

      OLDtoNEW (NODmin)= 1
      NEWtoOLD ( 1)= NODmin
      IW      = 0
      IW (NODmin)= 1

      ITEMcou= N/NCOLORK
!C===

```

IW:

Work array

“Color ID” of each mesh

IW=0 at initial stage

NODmin:

ID of the mesh with minimum value of “degree”

```

!C
!C +-----+
!C |  INIT.  |
!C +-----+
!C===
      allocate (IW(N))
      IW= 0

      NCOLORK = NCOLORTot

      do i= 1, N
        NEWtoOLD (i)= i
        OLDtoNEW (i)= i
      enddo

      INmin= N
      NODmin= 0
      do i= 1, N
        icon= INL(i) + INU(i)
        if (icon.lt. INmin) then
          INmin= icon
          NODmin= i
        endif
      enddo

      OLDtoNEW (NODmin)= 1
      NEWtoOLD (      1)= NODmin
      IW      = 0
      IW (NODmin)= 1

      ITEMcou= N/NCOLORK
!C===

```

mc (2/8)

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

OLDtoNEW (NODmin) = 1
NEWtoOLD (1) = NODmin

IW (NODmin) =1: Color ID

mc (2/8)

```

!C
!C +-----+
!C |  INIT.  |
!C +-----+
!C===
      allocate (IW(N))
      IW= 0

      NCOLORK = NCOLORTOT

      do i= 1, N
        NEWtoOLD (i)= i
        OLDtoNEW (i)= i
      enddo

      INmin= N
      NODmin= 0
      do i= 1, N
        icon= INL(i) + INU(i)
        if (icon.lt.INmin) then
          INmin= icon
          NODmin= i
        endif
      enddo

      OLDtoNEW (NODmin)= 1
      NEWtoOLD (      1)= NODmin
      IW      = 0
      IW(NODmin)= 1

      ITEMcou= N/NCOLORK
!C===

```

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

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

Finally, 5 colors are needed.
Affected by
reevaluation/devaluation

mc (3/8)

Initialization of Counters

icou : Global Counter

icouK: Intra-Color Counter

```
!C
!C +-----+
!C | MULTicoloring |
!C +-----+
!C===
      icou = 1
      icouK= 1

      do icol= 1, N
        NCOLORK= icol
        do i= 1, N
          if (IW(i).le.0) IW(i)= 0
        enddo

        do i= 1, N
!C
!C-- already COLORED
          if (IW(i).eq.icol) then
            do k= 1, INL(i)
              ik= IAL(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
            do k= 1, INU(i)
              ik= IAU(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
          endif
!C
!C-- not COLORED
          if (IW(i).eq.0) then
            icou = icou + 1
            icouK= icouK + 1
            IW(i)= icol
            do k= 1, INL(i)
              ik= IAL(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
            do k= 1, INU(i)
              ik= IAU(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
          endif
        enddo
      enddo
```

mc (3/8)

Loop on Colors

```

!C
!C +-----+
!C | MULTicoloring |
!C +-----+
!C===
      icou = 1
      icouK= 1

      do icol= 1, N
        NCOLORK= icol
        do i= 1, N
          if (IW(i).le.0) IW(i)= 0
        enddo

        do i= 1, N
!C
!C-- already COLORED
          if (IW(i).eq.icol) then
            do k= 1, INL(i)
              ik= IAL(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
            do k= 1, INU(i)
              ik= IAU(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
          endif
!C
!C-- not COLORED
          if (IW(i).eq.0) then
            icou = icou + 1
            icouK= icouK + 1
            IW(i)= icol
            do k= 1, INL(i)
              ik= IAL(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
            do k= 1, INU(i)
              ik= IAU(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
          endif

```

mc (3/8)

```

!C
!C +-----+
!C | MULTicoloring |
!C +-----+
!C===
      icou = 1
      icouK= 1

      do icol= 1, N
        NCOLORK= icol
        do i= 1, N
          if (IW(i).le.0) IW(i)= 0
        enddo

        do i= 1, N
!C
!C-- already COLORED
          if (IW(i).eq.icol) then
            do k= 1, INL(i)
              ik= IAL(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
            do k= 1, INU(i)
              ik= IAU(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
          endif
!C
!C-- not COLORED
          if (IW(i).eq.0) then
            icou = icou + 1
            icouK= icouK + 1
            IW(i)= icol
            do k= 1, INL(i)
              ik= IAL(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
            do k= 1, INU(i)
              ik= IAU(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
          endif

```

NCOLORk :

Current number of colors

IW(i)=0 :

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

mc (3/8)

```

!C
!C +-----+
!C | MULTicoloring |
!C +-----+
!C===
      icou = 1
      icouK= 1

      do icol= 1, N
        NCOLORK= icol
        do i= 1, N
          if (IW(i).le.0) IW(i)= 0
        enddo

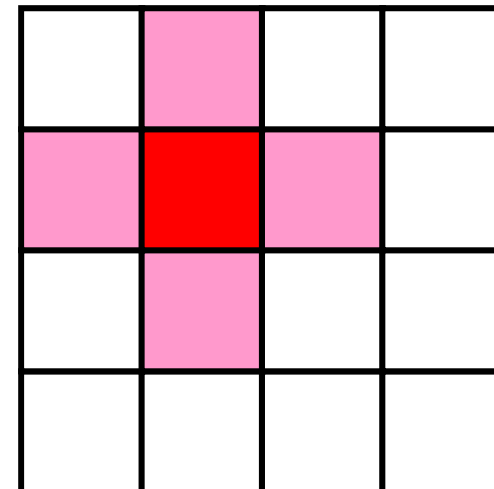
        do i= 1, N
!C
!C--- already COLORED
          if (IW(i).eq.icol) then
            do k= 1, INL(i)
              ik= IAL(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
            do k= 1, INU(i)
              ik= IAU(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
          endif
!C
!C--- not COLORED
          if (IW(i).eq.0) then
            icou = icou + 1
            icouK= icouK + 1
            IW(i)= icol
            do k= 1, INL(i)
              ik= IAL(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
            do k= 1, INU(i)
              ik= IAU(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
          endif

```

Loop on Colors

If mesh is already assigned to the “current color”, components of **IW** array for adjacent meshes are set to “-1”.

Remove meshes connected to meshes assigned to the “current color”, because they cannot be into the “current color”.



mc (3/8)

```

!C
!C +-----+
!C | MULTicoloring |
!C +-----+
!C===
      icou = 1
      icouK= 1

      do icol= 1, N
        NCOLORk= icol
        do i= 1, N
          if (IW(i).le.0) IW(i)= 0
        enddo

        do i= 1, N
!C
!C-- already COLORED
          if (IW(i).eq.icol) then
            do k= 1, INL(i)
              ik= IAL(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
            do k= 1, INU(i)
              ik= IAU(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
          endif
!C
!C-- not COLORED
          if (IW(i).eq.0) then
            icou = icou + 1
            icouK= icouK + 1
            IW(i)= icol
            do k= 1, INL(i)
              ik= IAL(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
            do k= 1, INU(i)
              ik= IAU(k,i)
              if (IW(ik).le.0) IW(ik)= -1
            enddo
          endif

```

if IW(i)=0:

- **icou =icou+1**
- **icouK=icouK+1**
- **IW(i)=icol**: Color ID

• **IW(ik)=-1** where *ik-th* mesh is adjacent to *i-th* mesh

mc (3/8)

```

do icol= 1, N
  NCOLORK= icol
  do i= 1, N
    if (IW(i).le.0) IW(i)= 0
  enddo

```

```

do i= 1, N
!C
!C-- already COLORED
  if (IW(i).eq.icol) then
    do k= 1, INL(i)
      ik= IAL(k,i)
      if (IW(ik).le.0) IW(ik)= -1
    enddo
    do k= 1, INU(i)
      ik= IAU(k,i)
      if (IW(ik).le.0) IW(ik)= -1
    enddo
  endif
!C
!C-- not COLORED
  if (IW(i).eq.0) then
    icou = icou + 1
    icouK= icouK + 1
    IW(i)= icol
    do k= 1, INL(i)
      ik= IAL(k,i)
      if (IW(ik).le.0) IW(ik)= -1
    enddo
    do k= 1, INU(i)
      ik= IAU(k,i)
      if (IW(ik).le.0) IW(ik)= -1
    enddo
  endif
  if (icou .eq. N)      goto 200
  if (icouK. eq. ITEMcou) goto 100
enddo

```

```

100   continue
      icouK= 0

```

```

enddo

```

```

200   continue

```

icou : Global Counter
icouK: Intra-Color Counter

if **icou=N (ICELTOT)**:

- All meshes are colored (completed).

if **icouK=ITEMcou**:

- icouK=0**
- Proceed to the next color.

if **icouK<ITEMcou.and.i=N**:

- No more independent meshes.
- Proceed to the next color.

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

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

5 independent meshes

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

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

5 independent meshes

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

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

Proceed to the next color, if no more independent meshes.

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

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

Proceed to the next color, if no more independent meshes.

mc (5/8)

```

!C
!C +-----+
!C | FINAL COLORING |
!C +-----+
!C===
      NCOLORTot= NCOLORK
      COLORindex= 0
      icoug= 0
      do ic= 1, NCOLORTot
        icou= 0
        do i= 1, N
          if (IW(i).eq.ic) then
            icou = icou + 1
            icoug= icoug + 1
            NEWtoOLD(icoug)= i
            OLDtoNEW(i)= icoug
          endif
        enddo
        COLORindex(ic)= icou
      enddo

      do ic= 1, NCOLORTot
        COLORindex(ic)= COLORindex(ic-1) + COLORindex(ic)
      enddo
!C===

```

NCOLORTot= NCOLORK:

Final number of colors.

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

mc (5/8)

```

!C
!C +-----+
!C | FINAL COLORING |
!C +-----+
!C===
      NCOLORTot= NCOLORK
      COLORindex= 0
      icoug= 0
      do ic= 1, NCOLORTot
        icou= 0
        do i= 1, N
          if (IW(i).eq.ic) then
            icou = icou + 1
            icoug= icoug + 1
            NEWtoOLD(icoug)= i
            OLDtoNEW(i)= icoug
          endif
        enddo
        COLORindex(ic)= icou
      enddo

      do ic= 1, NCOLORTot
        COLORindex(ic)= COLORindex(ic-1) + COLORindex(ic)
      enddo
!C===

```

Renumber meshes in ascending orders according to color ID.

OLDtoNEW (OldID) = NewID

NEWtoOLD (NewID) = OldID

COLODindex (ic) :

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

mc (6/8)

```

!C
!C +-----+
!C | FINAL COLORING |
!C +-----+
!C===
      NCOLORTot= NCOLORK
      COLORindex= 0
      icoug= 0
      do ic= 1, NCOLORTot
        icou= 0
        do i= 1, N
          if (IW(i).eq.ic) then
            icou = icou + 1
            icoug= icoug + 1
            NEWtoOLD(icoug)= i
            OLDtoNEW(i)= icoug
          endif
        enddo
        COLORindex(ic)= icou
      enddo

      do ic= 1, NCOLORTot
        COLORindex(ic)= COLORindex(ic-1) + COLORindex(ic)
      enddo
!C===

```

COLODindex(ic) :

Now it is 1D index.

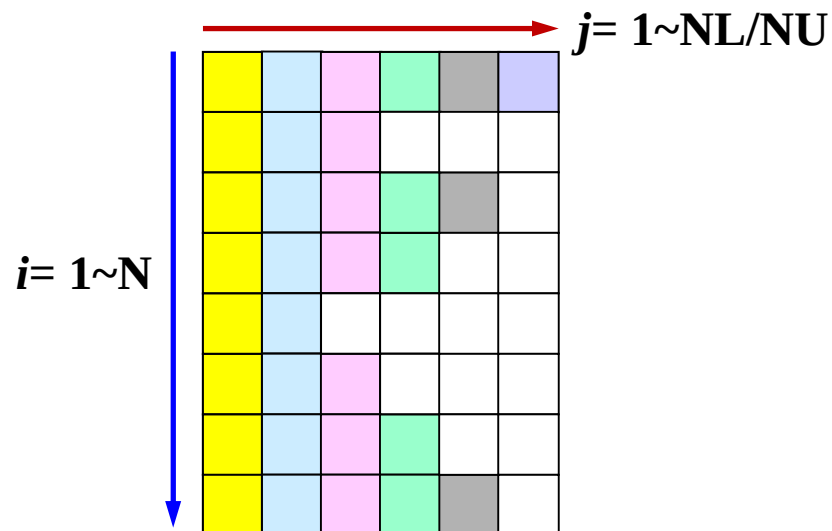
```

!C
!C +-----+
!C | MATRIX transfer |
!C +-----+
!C==
      allocate (INLw(N), INUw(N), IALw(NL, N), IAUw(NU, N))

      do j= 1, NL
        do i= 1, N
          IW(i) = IAL(j, NEWtoOLD(i))
        enddo
        do i= 1, N
          IAL(j, i) = IW(i)
        enddo
      enddo

      do j= 1, NU
        do i= 1, N
          IW(i) = IAU(j, NEWtoOLD(i))
        enddo
        do i= 1, N
          IAU(j, i) = IW(i)
        enddo
      enddo

```



mc (6/8)

Arrays for Work

- INLw (N)
- INUw (N)
- IALw (NL, N)
- IAUw (NU, N)

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

ID's of column ID are by old
numbering.

mc (7/8)

```

do i= 1, N
  IW(i) = INL (NEWtoOLD(i))
enddo

do i= 1, N
  INLw(i) = IW(i)
enddo

do i= 1, N
  IW(i) = INU (NEWtoOLD(i))
enddo

do i= 1, N
  INUw(i) = IW(i)
enddo

do j= 1, NL
  do i= 1, N
    if (IAL(j,i).eq.0) then
      IALw(j,i) = 0
    else
      IALw(j,i) = OLDtoNEW(IAL(j,i))
    endif
  enddo
enddo

do j= 1, NU
  do i= 1, N
    if (IAU(j,i).eq.0) then
      IAUw(j,i) = 0
    else
      IAUw(j,i) = OLDtoNEW(IAU(j,i))
    endif
  enddo
enddo

```

Information for number of lower/upper components based on new numbering is stored into:

- INL → INLw
- INU → INUw

mc (7/8)

```
do i= 1, N
  IW(i) = INL (NEWtoOLD(i))
enddo
```

```
do i= 1, N
  INLw(i) = IW(i)
enddo
```

```
do i= 1, N
  IW(i) = INU (NEWtoOLD(i))
enddo
```

```
do i= 1, N
  INUw(i) = IW(i)
enddo
```

```
do j= 1, NL
  do i= 1, N
    if (IAL(j,i).eq.0) then
      IALw(j,i) = 0
    else
      IALw(j,i) = OLDtoNEW(IAL(j,i))
    endif
  enddo
enddo
```

```
do j= 1, NU
  do i= 1, N
    if (IAU(j,i).eq.0) then
      IAUw(j,i) = 0
    else
      IAUw(j,i) = OLDtoNEW(IAU(j,i))
    endif
  enddo
enddo
```

“Renumbered” lower/upper components (column ID) are stored into:

- IAL -> IALw
- IAU -> IAUw

mc (8/8)

Operation for lower triangular components (column ID) in the original matrix. "Renumbered" components (column ID) are stored into:

• IALw → IAL, IAU

```

INL= 0
INU= 0
IAL= 0
IAU= 0

do i= 1, N
  jL= 0
  jU= 0
  do j= 1, INLw(i)
    if (IALw(j, i).gt. i) then
      jU= jU + 1
      IAU(jU, i)= IALw(j, i)
    else
      jL= jL + 1
      IAL(jL, i)= IALw(j, i)
    endif
  enddo

  do j= 1, INUw(i)
    if (IAUw(j, i).gt. i) then
      jU= jU + 1
      IAU(jU, i)= IAUw(j, i)
    else
      jL= jL + 1
      IAL(jL, i)= IAUw(j, i)
    endif
  enddo

  INL(i)= jL
  INU(i)= jU
enddo

!C===
deallocate (IW, INLw, INUw, IALw, IAUw)

return
end

```

```

do i= 1, N
  jL= 0
  jU= 0
  do j= 1, INLw(i)
    if (IALw(j, i).gt. i) then
      jU= jU + 1
      IAU(jU, i)= IALw(j, i)
    else
      jL= jL + 1
      IAL(jL, i)= IALw(j, i)
    endif
  enddo
enddo

```

Because IALw(j, i) could be larger than i according to new numbering.

Why do we need these operations ?

Original

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

$INL(7) = 2$
 $IAL(1, 7) = 3, \quad IAL(2, 7) = 6$

$INU(7) = 2$
 $IAU(1, 7) = 8, \quad IAU(2, 7) = 11$

5 Color

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

$INL(9) = 3$
 $IAL(1, 9) = 2, \quad IAL(2, 9) = 3$
 $IAL(3, 9) = 4$

$INU(9) = 1$
 $IAU(1, 9) = 11$

Magnitude correlation with adjacent meshes could change after renumbering.

mc (8/8)

```

INL= 0
INU= 0
IAL= 0
IAU= 0

do i= 1, N
  jL= 0
  jU= 0
  do j= 1, INLw(i)
    if (IALw(j, i).gt. i) then
      jU= jU + 1
      IAU(jU, i)= IALw(j, i)
    else
      jL= jL + 1
      IAL(jL, i)= IALw(j, i)
    endif
  enddo

  do j= 1, INUw(i)
    if (IAUw(j, i).gt. i) then
      jU= jU + 1
      IAU(jU, i)= IAUw(j, i)
    else
      jL= jL + 1
      IAL(jL, i)= IAUw(j, i)
    endif
  enddo

  INL(i)= jL
  INU(i)= jU
enddo

!C===
deallocate (IW, INLw, INUw, IALw, IAUw)

return
end

```

Operation for upper triangular components (column ID) in the original matrix. “Renumbered” components (column ID) are stored into:

• IAUw -> IAL, IAU

mc (8/8)

```

INL= 0
INU= 0
IAL= 0
IAU= 0

do i= 1, N
  jL= 0
  jU= 0
  do j= 1, INLw(i)
    if (IALw(j, i).gt. i) then
      jU= jU + 1
      IAU(jU, i)= IALw(j, i)
    else
      jL= jL + 1
      IAL(jL, i)= IALw(j, i)
    endif
  enddo

  do j= 1, INUw(i)
    if (IAUw(j, i).gt. i) then
      jU= jU + 1
      IAU(jU, i)= IAUw(j, i)
    else
      jL= jL + 1
      IAL(jL, i)= IAUw(j, i)
    endif
  enddo

  INL(i)= jL
  INU(i)= jU
enddo

!C===
deallocate (IW, INLw, INUw, IALw, IAUw)

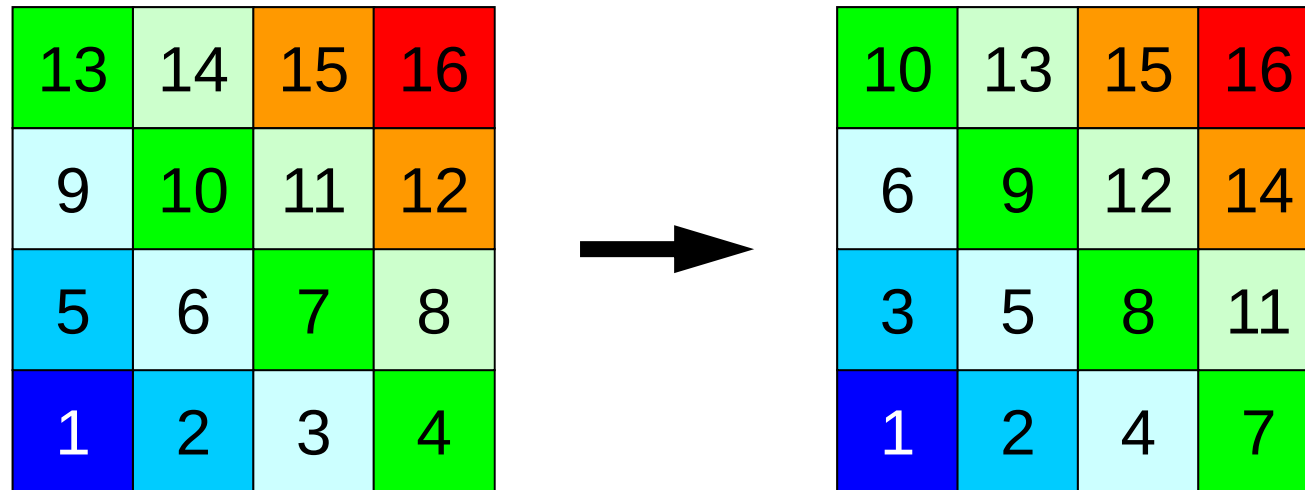
return
end

```

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

- INL
- INU

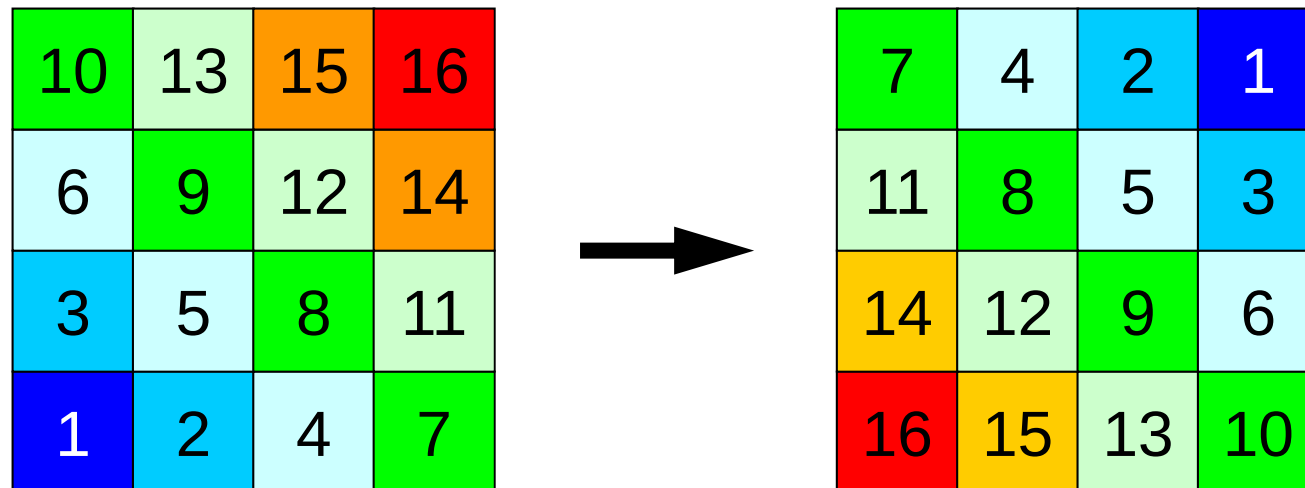
Procedure of CM Method



- Meshes adjacent to “Level=k-1” meshes are set to “Level=k” (Repeat until all meshes are flagged into “levels”)
 - In each level, meshes must be independent (not directly connected). If a dependent pair is found in same color, one mesh is removed (In current implementation, a mesh found later is removed).
- Renumber meshes in ascending orders according to “Level” ID.

RCM: Reverse Cuthill-McKee

- Do operations for “CM” method
 - Calculate “degree” at each mesh
 - Flag “level k (1,2,...)” to meshes
 - Repeat processes, final renumbering
- Renumbering Again
 - Renumber meshes reordered by CM method in reverse order.
 - Fill-in’s are fewer than CM



cm (1/5)

```
!C
!C***
!C*** CM
!C***
!C
      subroutine CM (N, NL, NU, INL, IAL, INU, IAU,
&                  NCOLortot, COLORindex, NEWtoOLD, OLDtoNEW)

      implicit REAL*8 (A-H, O-Z)
      integer, dimension(N) :: INL, INU, NEWtoOLD, OLDtoNEW
      integer, dimension(0:N) :: COLORindex
      integer, dimension(NL, N) :: IAL
      integer, dimension(NU, N) :: IAU

      integer, dimension(:, :), allocatable :: IW
      integer, dimension(:) , allocatable :: INLw, INUw
      integer, dimension(:, :), allocatable :: IALw, IAUw
```

cm (2/5)

```

!C
!C +-----+
!C | INIT. |
!C +-----+
!C==
      allocate (IW(N,2))

      IW = 0

      INmin= N
      NODmin= 0

      do i= 1, N
        icon= 0
        do k= 1, INL(i)
          icon= icon + 1
        enddo
        do k= 1, INU(i)
          icon= icon + 1
        enddo

        if (icon.lt.INmin) then
          INmin = icon
          NODmin= i
        endif
      enddo
200  continue

      if (NODmin.eq.0) NODmin= 1

      IW(NODmin,2)= 1

      NEWtoOLD(1)= NODmin
      OLDtoNEW(NODmin)= 1

      icol= 1
!C==

```

IW(i,1):

Work array

IW(i,2):

“Level ID” of each mesh

cm (2/5)

```

!C
!C +-----+
!C | INIT. |
!C +-----+
!C===
      allocate (IW(N,2))

      IW = 0

      INmin= N
      NODmin= 0

      do i= 1, N
        icon= 0
        do k= 1, INL(i)
          icon= icon + 1
        enddo
        do k= 1, INU(i)
          icon= icon + 1
        enddo

        if (icon.lt.INmin) then
          INmin = icon
          NODmin= i
        endif
      enddo
200  continue

      if (NODmin.eq.0) NODmin= 1

      IW(NODmin,2)= 1

      NEWtoOLD(1)= NODmin
      OLDtoNEW(NODmin)= 1

      icol= 1
!C===

```

NODmin :

ID of the mesh with minimum value of “degree”

cm (2/5)

```

!C
!C +-----+
!C | INIT. |
!C +-----+
!C===
      allocate (IW(N,2))

      IW = 0

      INmin= N
      NODmin= 0

      do i= 1, N
        icon= 0
        do k= 1, INL(i)
          icon= icon + 1
        enddo
        do k= 1, INU(i)
          icon= icon + 1
        enddo

        if (icon.lt.INmin) then
          INmin = icon
          NODmin= i
        endif
      enddo
200  continue

      if (NODmin.eq.0) NODmin= 1

      IW(NODmin,2)= 1

      NEWtoOLD(1)= NODmin
      OLDtoNEW(NODmin)= 1

      icol= 1
!C===

```

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

OLDtoNEW(NODmin) = 1
NEWtoOLD(1) = NODmin

IW(NODmin,2)=1: Level ID

cm (3/5)

```

!C
!C +-----+
!C | COLORING |
!C +-----+
!C===
      icouG= 1
      do icol= 2, N
        icou = 0
        do i= 1, N
          if (IW(i,2).eq.icol-1) then
            do k= 1, INL(i)
              in= IAL(k,i)
              if (IW(in,2).eq.0) then
                IW(in,2)= -icol
                icou      = icou + 1
                IW(icou,1)= in
              endif
            enddo
          do k= 1, INU(i)
            in= IAU(k,i)
            if (IW(in,2).eq.0) then
              IW(in,2)= -icol
              icou      = icou + 1
              IW(icou,1)= in
            endif
          enddo
        endif
      enddo

      if (icou.eq.0) then
        do i= 1, N
          if (IW(i,2).eq.0) then
            icou= icou + 1
            IW(i,2)= -icol
            IW(icou,1)= i
            goto 850
          endif
        enddo
      endif
      continue
850

```

icouG: Global Counter

icou : Intra-Level Counter

Loop on Levels

cm (3/5)

```

!C
!C +-----+
!C | COLORING |
!C +-----+
!C===
      icouG= 1
      do icol= 2, N
        icou = 0
        do i= 1, N
          if (IW(i,2).eq.icol-1) then
            do k= 1, INL(i)
              in= IAL(k,i)
              if (IW(in,2).eq.0) then
                IW(in,2)= -icol
                icou      = icou + 1
                IW(icou,1)= in
              endif
            enddo
          do k= 1, INU(i)
            in= IAU(k,i)
            if (IW(in,2).eq.0) then
              IW(in,2)= -icol
              icou      = icou + 1
              IW(icou,1)= in
            endif
          enddo
        endif
      enddo

      if (icou.eq.0) then
        do i= 1, N
          if (IW(i,2).eq.0) then
            icou= icou + 1
            IW(i,2)= -icol
            IW(icou,1)= i
            goto 850
          endif
        enddo
      endif
      continue
850

```

icouG: Global Counter

icou : Intra-Level Counter

Loops for Each Element

If i_n th mesh is adjacent to i th mesh where $IW(i, 2) = icol - 1$, and level of i_n th mesh is not finalized, i_n th mesh could be a candidate for meshes in $icol$ th level.

- $IW(in, 2) = -icol$
- $icou = icou + 1$
- $IW(icou, 1) = in$: This array indicates “when” this mesh was found, and nominated as a candidate.

What does it mean ?

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

`icol=4`

`IW(i,2) = icol-1=3: i=3,6,9`

icouG: Global Counter

icou : Intra-Level Counter

Loops for Each Element

If **in**th mesh is adjacent to **i**th mesh where `IW(i,2)=icol-1`, and level of **in**th mesh is not finalized, **in**th mesh could be a candidate for meshes in **icol**th level.

- `IW(in,2) = -icol`
- `icou = icou + 1`
- `IW(icou,1) = in`: This array indicates “when” this mesh was found, and nominated as a candidate.

What does it mean ?

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

`icol=4`

`IW(i,2) = icol-1=3: i=3, 6, 9`

`IW( 4,2) = -4`

`IW( 7,2) = -4`

`IW(10,2) = -4`

`IW(13,2) = -4`

`IW(1,1) = 4`

`IW(2,1) = 7`

`IW(3,1) = 10`

`IW(4,1) = 13`

icouG: Global Counter

icou : Intra-Level Counter

Loops for Each Element

If **in**th mesh is adjacent to **i**th mesh where `IW(i,2)=icol-1`, and level of **in**th mesh is not finalized, **in**th mesh could be a candidate for meshes in **icol**th level.

- `IW(in,2) = -icol`

- `icou = icou + 1`

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

cm (3/5)

```

!C
!C +-----+
!C | COLORING |
!C +-----+
!C===
      icouG= 1
      do icol= 2, N
        icou = 0
        do i= 1, N
          if (IW(i,2).eq.icol-1) then
            do k= 1, INL(i)
              in= IAL(k,i)
              if (IW(in,2).eq.0) then
                IW(in,2)= -icol
                icou      = icou + 1
                IW(icou,1)= in
              endif
            enddo
          do k= 1, INU(i)
            in= IAU(k,i)
            if (IW(in,2).eq.0) then
              IW(in,2)= -icol
              icou      = icou + 1
              IW(icou,1)= in
            endif
          enddo
        endif
      enddo

      if (icou.eq.0) then
        do i= 1, N
          if (IW(i,2).eq.0) then
            icou= icou + 1
            IW(i,2)= -icol
            IW(icou,1)= i
            goto 850
          endif
        enddo
      endif
      850 continue

```

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.

cm (4/5)

```

!C
!C +-----+
!C | COLORING |
!C +-----+
!C===

...

do ic= 1, icou
  inC= IW(ic,1)
  if (IW(inC,2).ne.0) then
    do k= 1, INL(inC)
      in= IAL(k,inC)
      if (IW(in,2).le.0) IW(in,2)= 0
    enddo
    do k= 1, INU(inC)
      in= IAU(k,inC)
      if (IW(in,2).le.0) IW(in,2)= 0
    enddo
  endif
enddo

do ic= 1, icou
  inC= IW(ic,1)
  if (IW(inC,2).ne.0) then
    icouG = icouG + 1
    IW(inC,2)= icol
  endif
enddo

if (icouG.eq.N) exit
enddo

!C===

```

Candidates for icolth level are stored in:

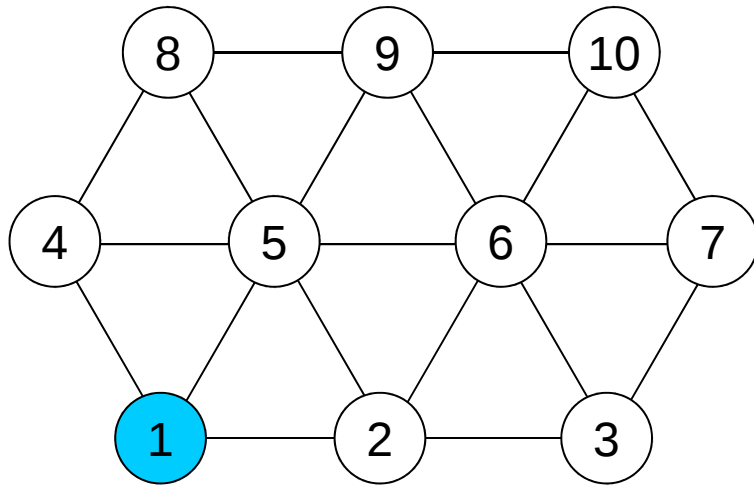
- **$IW(ic,1)$, $ic= 1, icou$**

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

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

- **$IW(in,2) = 0$**

What does is mean ?



e.g.
Mesh (1) belongs to $(ic_{ol}-1)^{th}$ level

Candidates for ic_{ol}^{th} level are stored in:

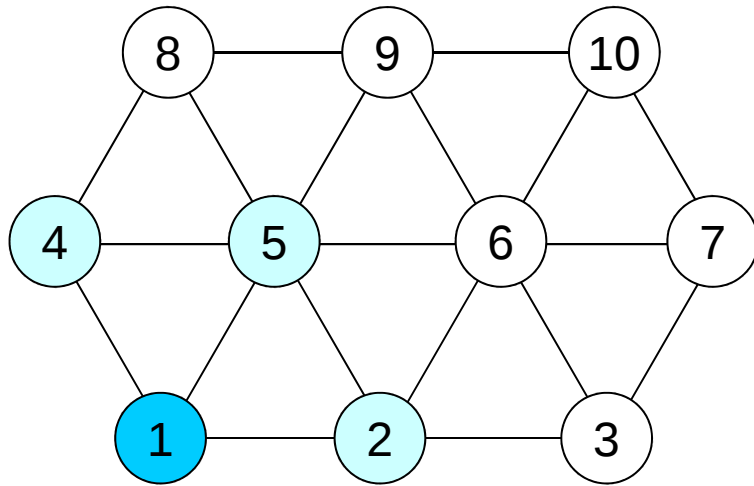
- $IW(ic, 1), ic = 1, icou$

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

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

- $IW(in, 2) = 0$

What does it mean ?



(2),(4) and (5) are candidates for (ic_{ol})th level

$$IW(2, 2) = -ic_{ol}$$

$$IW(4, 2) = -ic_{ol}$$

$$IW(5, 2) = -ic_{ol}$$

$$IW(1, 1) = 2$$

$$IW(2, 1) = 4$$

$$IW(3, 1) = 5$$

Candidates for ic_{ol}th level are stored in:

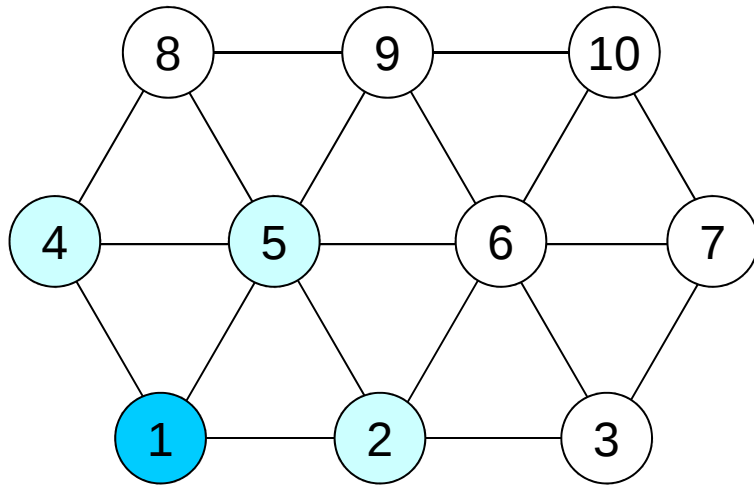
- $IW(ic, 1), \quad ic = 1, \quad ic_{ou}$

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

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

- $IW(in, 2) = 0$

What does it mean ?



Considering dependency:

$$IW(2, 2) = -icol$$

$$IW(4, 2) = -icol$$

$$IW(5, 2) = 0$$

$$IW(1, 1) = 2$$

$$IW(2, 1) = 4$$

$$IW(3, 1) = 5$$

Candidates for icolth level are stored in:

$$\bullet IW(ic, 1), \quad ic = 1, \quad icou$$

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

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

$$\bullet IW(in, 2) = 0$$

```

!C
!C +-----+
!C | COLORING |
!C +-----+
!C===

...

do ic= 1, icou
  inC= IW(ic, 1)
  if (IW(inC, 2).ne.0) then
    do k= 1, INL(inC)
      in= IAL(k, inC)
      if (IW(in, 2).le.0) IW(in, 2)= 0
    enddo
    do k= 1, INU(inC)
      in= IAU(k, inC)
      if (IW(in, 2).le.0) IW(in, 2)= 0
    enddo
  endif
enddo

do ic= 1, icou
  inC= IW(ic, 1)
  if (IW(inC, 2).ne.0) then
    icouG = icouG + 1
    IW(inC, 2)= icol
  endif
enddo

if (icouG.eq.N) exit
enddo
!C===

```

cm (4/5)

icouG: Global Counter

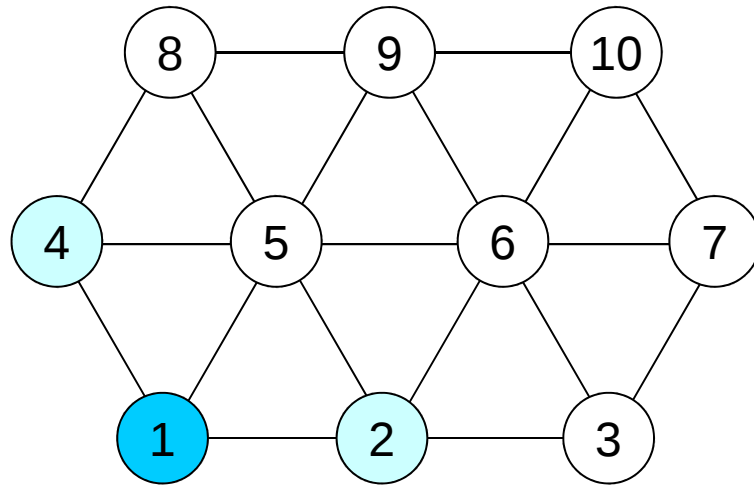
icou : Intra-Level Counter

Finally, meshes which satisfies $IW(inC, 2) = -icol$, belong to icolth level.

For such meshes, apply $IW(inC, 2) = icol$.

Finally, $icouG = icouG + 1$

What does it mean ?



Considering dependency:

$$IW(2, 2) = -icol$$

$$IW(4, 2) = -icol$$

$$IW(5, 2) = 0$$

$$IW(1, 1) = 2$$

$$IW(2, 1) = 4$$

$$IW(3, 1) = 5$$

Finally:

$$IW(2, 2) = icol$$

$$IW(4, 2) = icol$$

Finally, meshes which satisfies $IW(inC, 2) = -icol$, belong to $icol^{th}$ level.

For such meshes, apply $IW(inC, 2) = icol$.

Finally, $icouG = icouG + 1$

```

!C
!C +-----+
!C | COLORING |
!C +-----+
!C===

...

do ic= 1, icou
  inC= IW(ic,1)
  if (IW(inC,2).ne.0) then
    do k= 1, INL(inC)
      in= IAL(k,inC)
      if (IW(in,2).le.0) IW(in,2)= 0
    enddo
    do k= 1, INU(inC)
      in= IAU(k,inC)
      if (IW(in,2).le.0) IW(in,2)= 0
    enddo
  endif
enddo

do ic= 1, icou
  inC= IW(ic,1)
  if (IW(inC,2).ne.0) then
    icouG = icouG + 1
    IW(inC,2)= icol
  endif
enddo

  if (icouG.eq.N) exit
enddo
!C===

```

cm (4/5)

icouG: Global Counter

icou : Intra-Level Counter

if icouG=N (ICELTOT):

- All meshes are colored (completed).

Otherwise, proceed to the next level.

```

!C
!C +-----+
!C | FINAL COLORING |
!C +-----+
!C===
3000 continue
  NCOLORTot= icol
  icoug= 0
  do ic= 1, NCOLORTot
    icou= 0
    do i= 1, N
      if (IW(i,2).eq.ic) then
        icou = icou + 1
        icoug= icoug + 1
        NEWtoOLD(icoug)= i
        OLDtoNEW(i)= icoug
      endif
    enddo
    COLORindex(ic)= icou
  enddo

  COLORindex(0)= 0
  do ic= 1, NCOLORTot
    COLORindex(ic)= COLORindex(ic-1) + COLORindex(ic)
  enddo
!C===

!C
!C +-----+
!C | MATRIX transfer |
!C +-----+
!C===
...
!C===
  return
end

```

cm (5/5)

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

NEWtoOLD(NewID) = OldID

COLODindex(ic):

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

cm (5/5)

```

!C
!C +-----+
!C | FINAL COLORING |
!C +-----+
!C===
3000 continue
    NCOLORTot= icol
    icoug= 0
    do ic= 1, NCOLORTot
        icou= 0
        do i= 1, N
            if (IW(i,2).eq.ic) then
                icou = icou + 1
                icoug= icoug + 1
                NEWtoOLD(icoug)= i
                OLDtoNEW(i)= icoug
            endif
        enddo
        COLORindex(ic)= icou
    enddo

    COLORindex(0)= 0
    do ic= 1, NCOLORTot
        COLORindex(ic)= COLORindex(ic-1) + COLORindex(ic)
    enddo
!C===

!C
!C +-----+
!C | MATRIX transfer |
!C +-----+
!C===
...
!C===
    return
end

```

COLODindex(ic) :

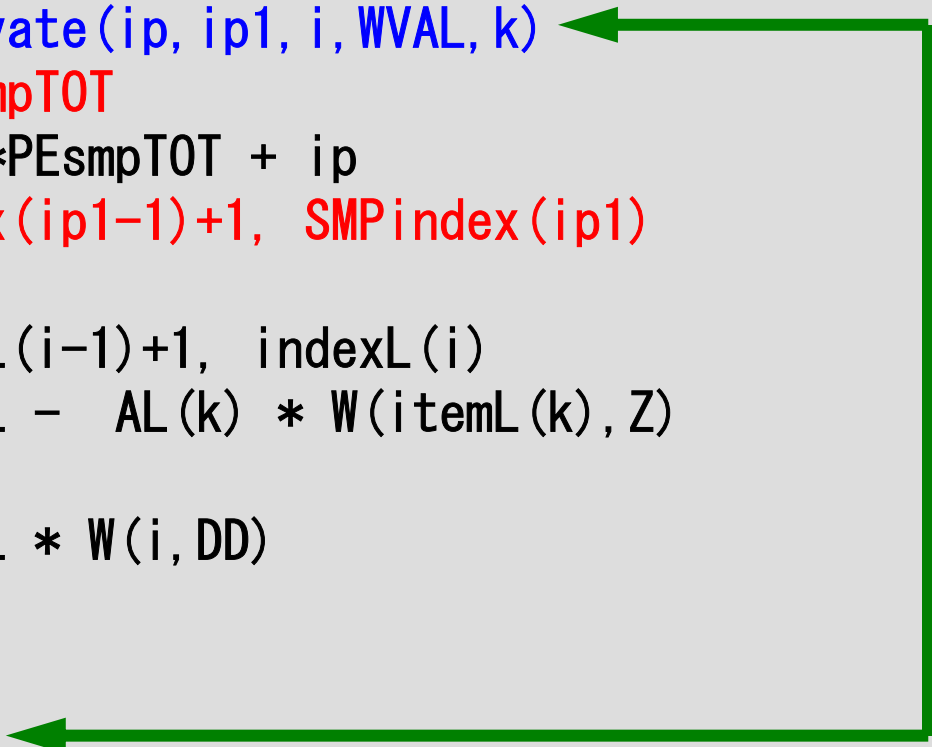
Now it is 1D index.

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

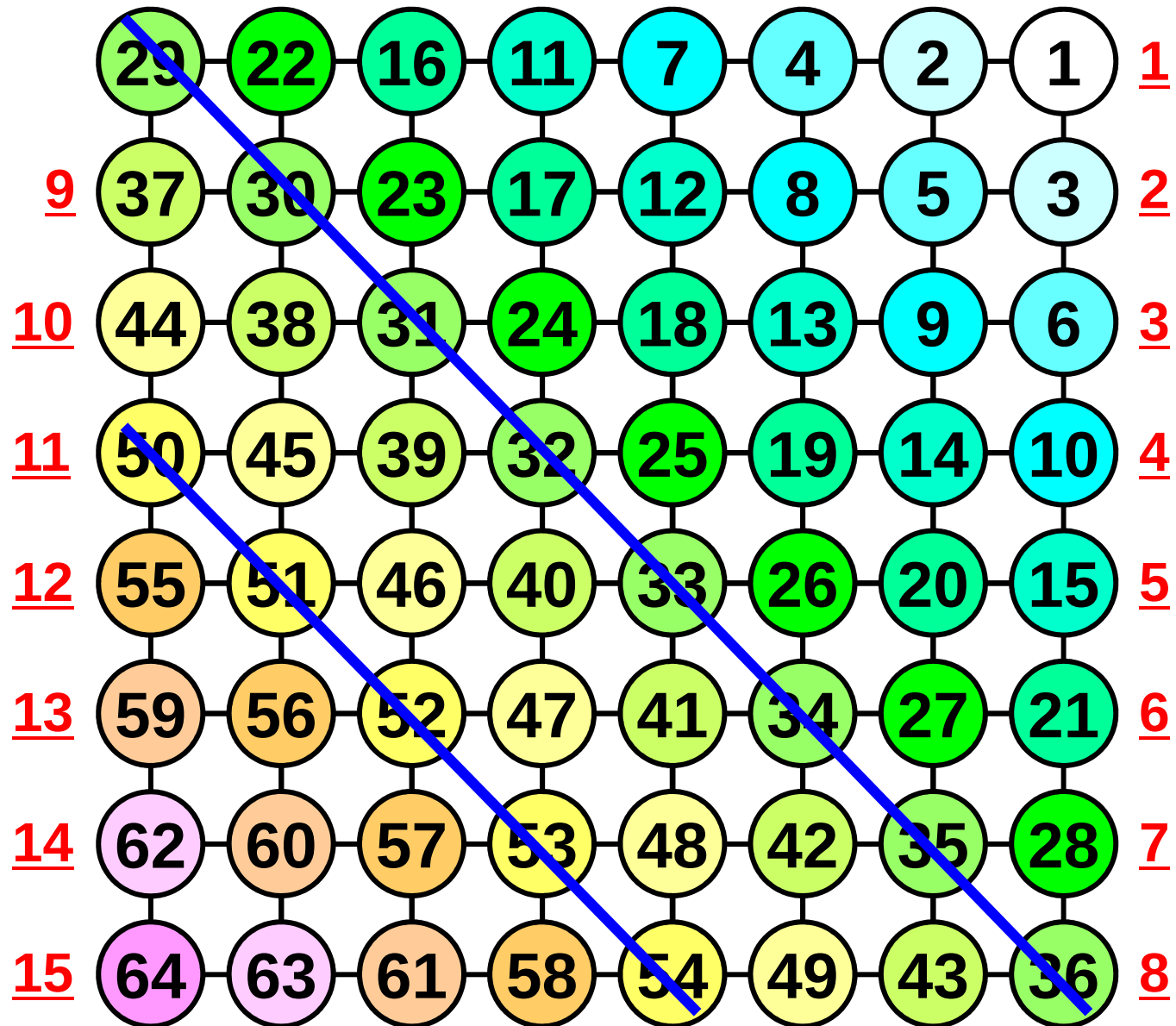
```
do ic= 1, NCOLORTot
!$omp parallel do private(ip, ip1, i, WVAL, k)
  do ip= 1, PEsmptOT
    ip1= (ic-1)*PEsmptOT + ip
    do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      WVAL= W(i, Z)
      do k= indexL(i-1)+1, indexL(i)
        WVAL= WVAL - AL(k) * W(itemL(k), Z)
      enddo
      W(i, Z)= WVAL * W(i,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

A diagram consisting of two green arrows. The first arrow starts at the right side of the line 'do ip= 1, PEsmptOT' and points left to the line '!\$omp parallel do private(ip, ip1, i, WVAL, k)'. The second arrow starts at the right side of the line '!\$omp end parallel do' and points left to the same line '!\$omp parallel do private(ip, ip1, i, WVAL, k)'. These arrows indicate that the parallel region must wait for all threads to finish before proceeding.

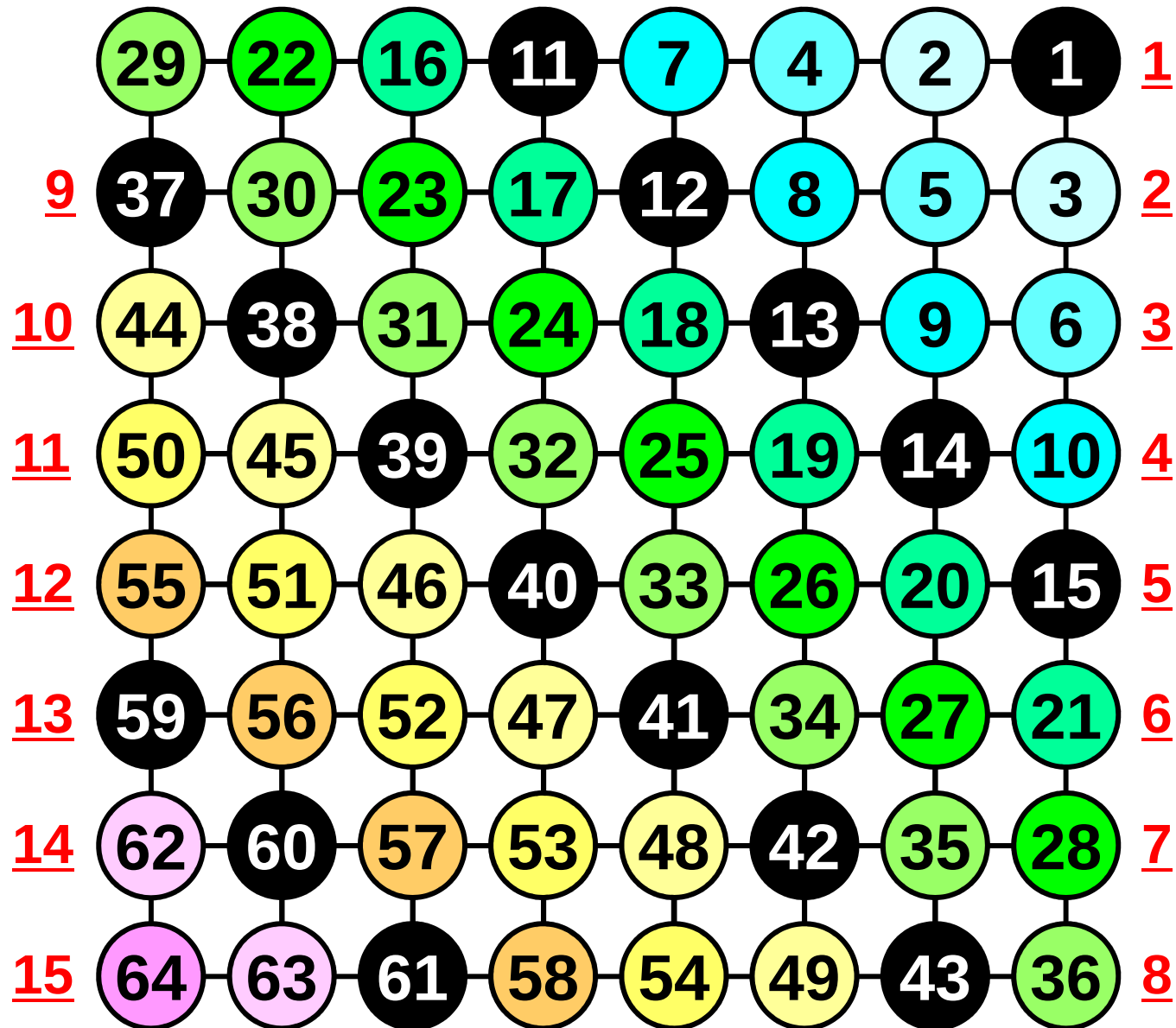
Remedy: CM-RCM

- RCM + Cyclic Multicoloring [Doi, Osoda, Washio]
- Procedures
 - Apply RCM
 - Define “Nc” (Color number of Cyclic Multicoloring (CM))
 - 1st-Color in CM-RCM: 1st, (Nc+1)th, (2Nc+1)th ... levels in RCM
 - 2nd-Color in CM-RCM: 2nd, (Nc+2)th, (2Nc+2)th ... levels in RCM
 - kth-Color in CM-RCM: kth, (Nc+k)th, (2Nc+k)th ... 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 $Nc=Nc+1$.

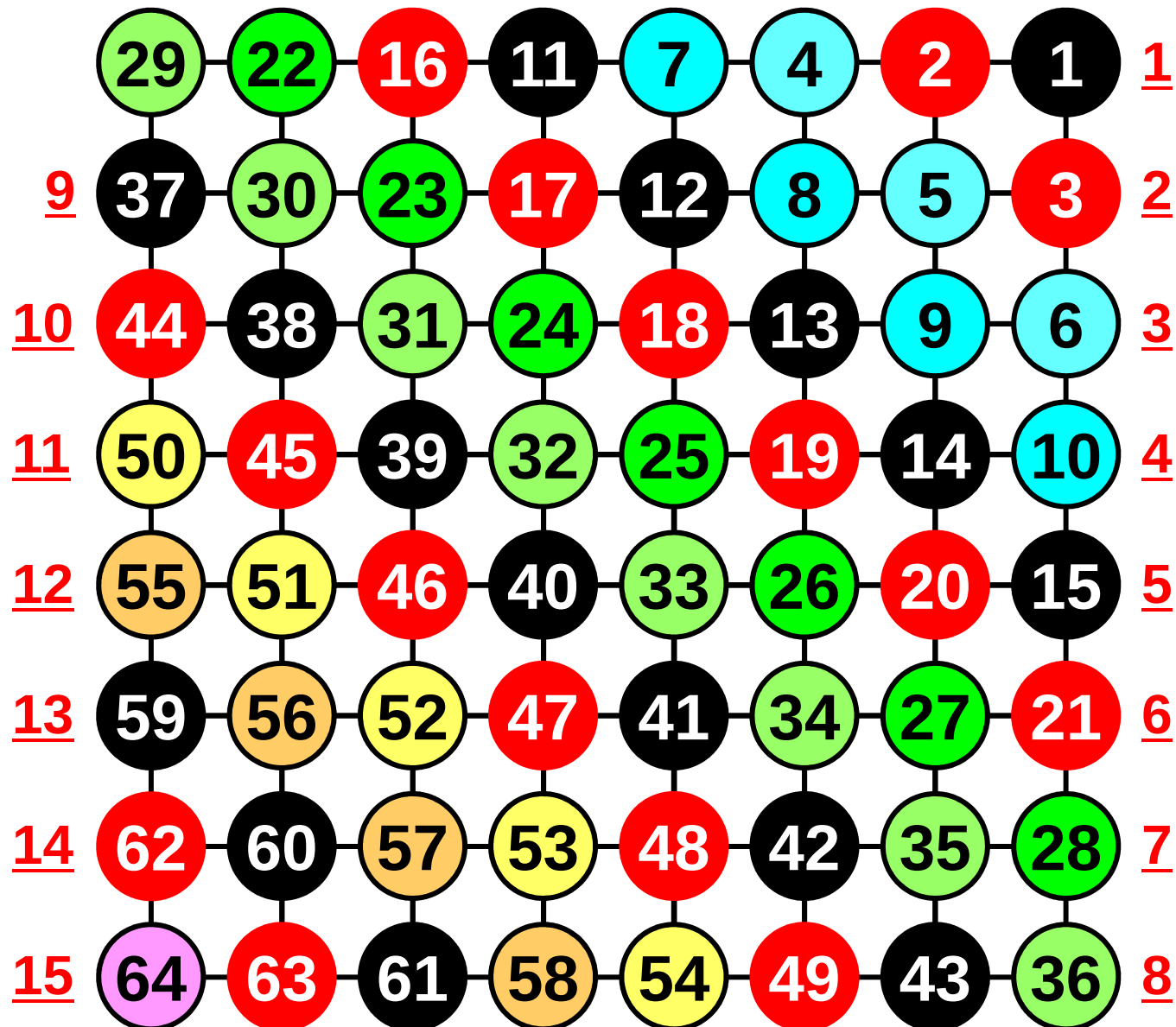
RCM



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



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



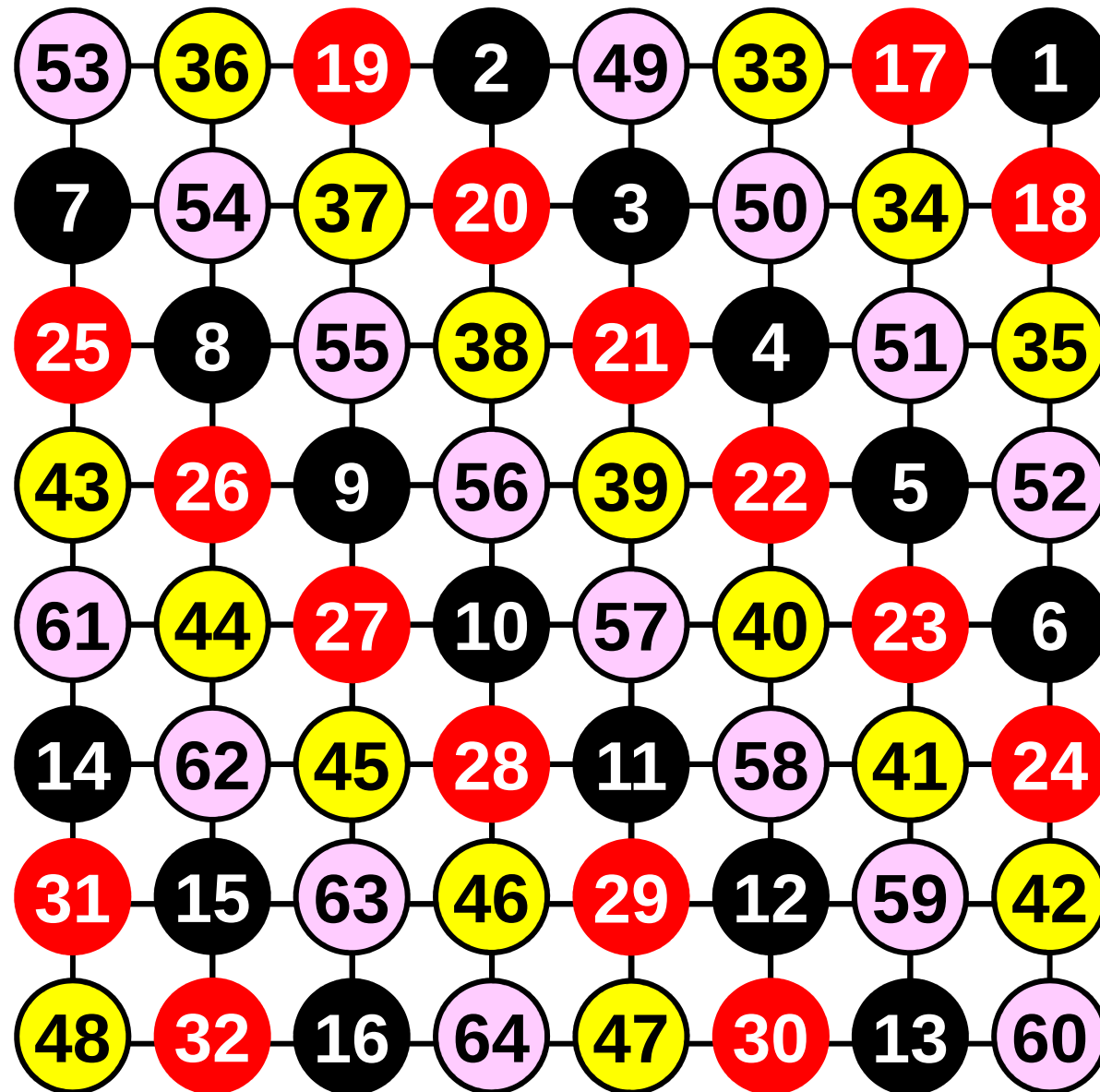
CM-RCM($N_c=4$): Renumbering

k=1: 16

k=2: 16

k=3: 16

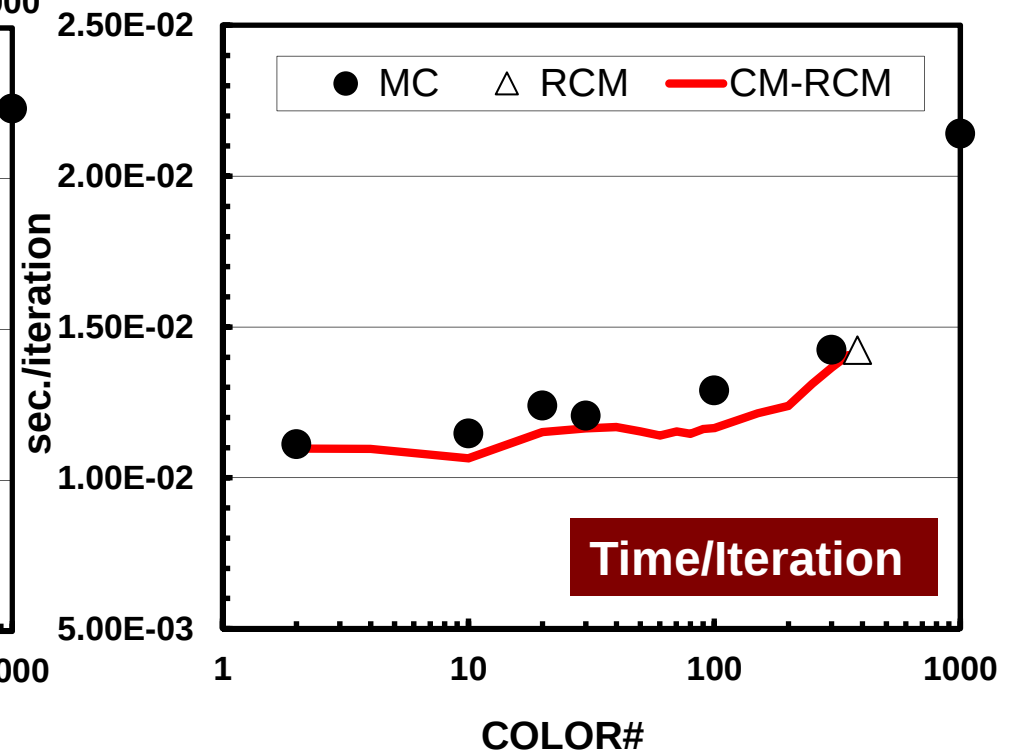
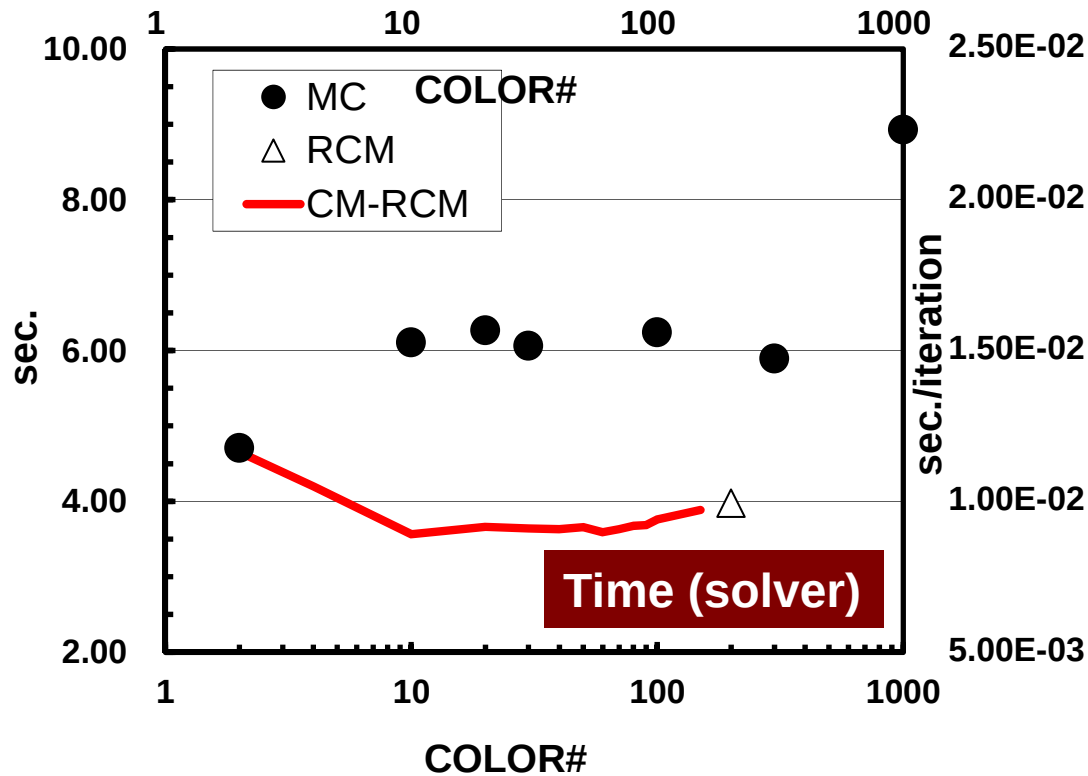
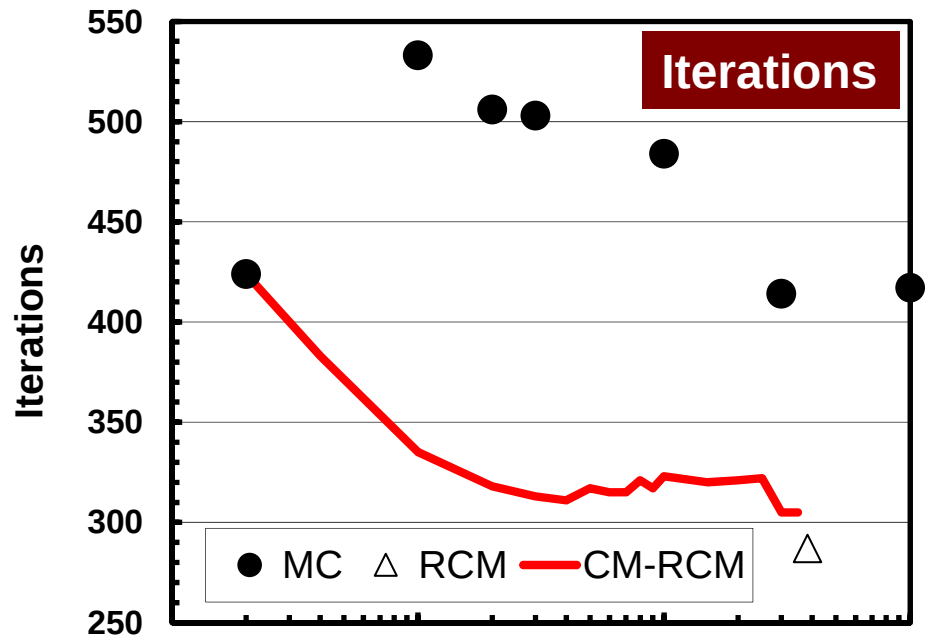
k=4: 16



CM-RCM

- How to run
 - "**NCOLOR_{tot} = -N_c**" in INPUT.DAT
 - Already implemented in L2
- cmrcm.f, cmrcm.c

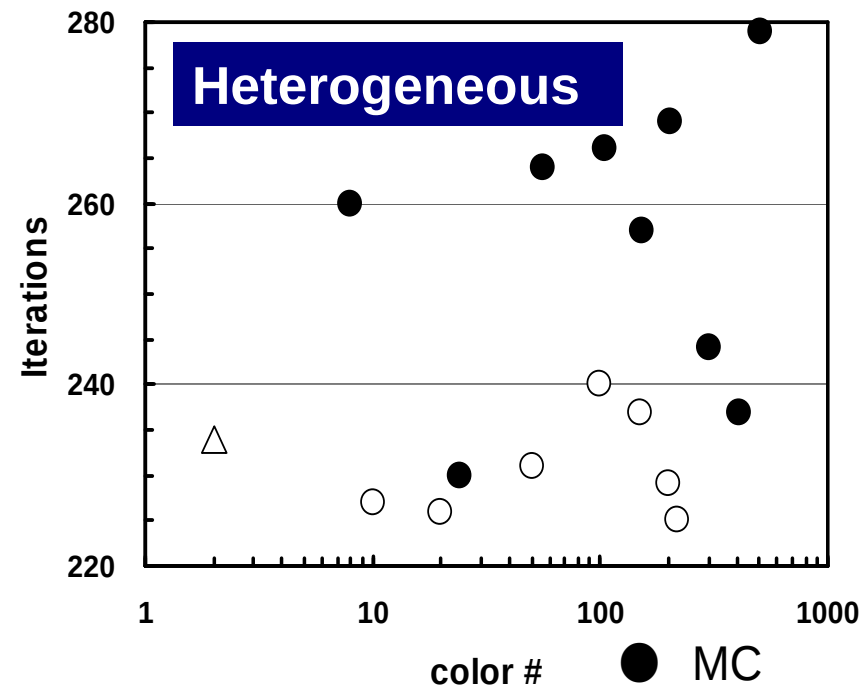
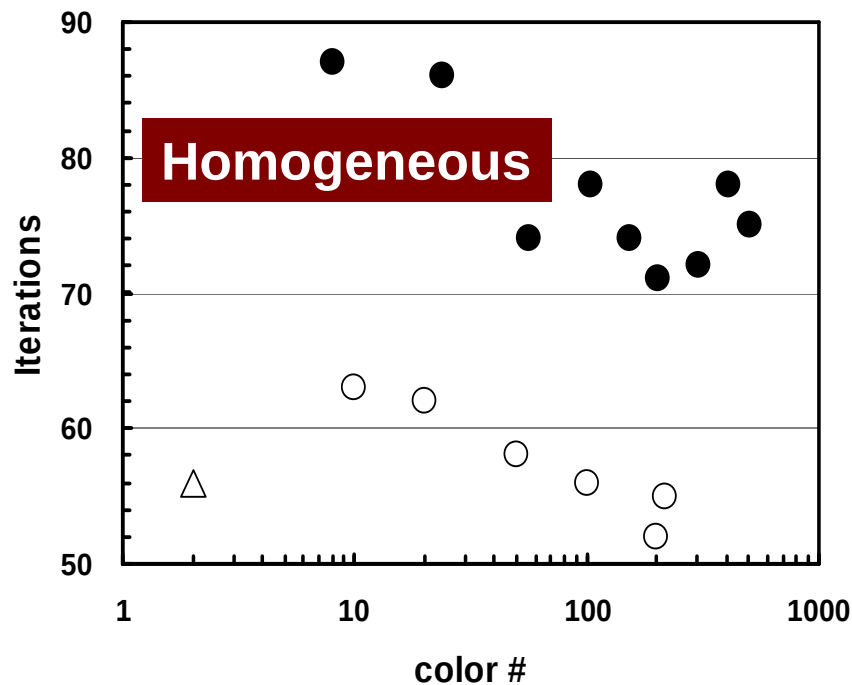
Reedbush-U, 1-
node/18-cores,
 128^3
(●:MC, △:RCM, -:
CM-RCM)



Comparison of Reordering Methods

3D Linear Elastic Problems

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



3D Linear-Elastic Problems with 32,768 DOF

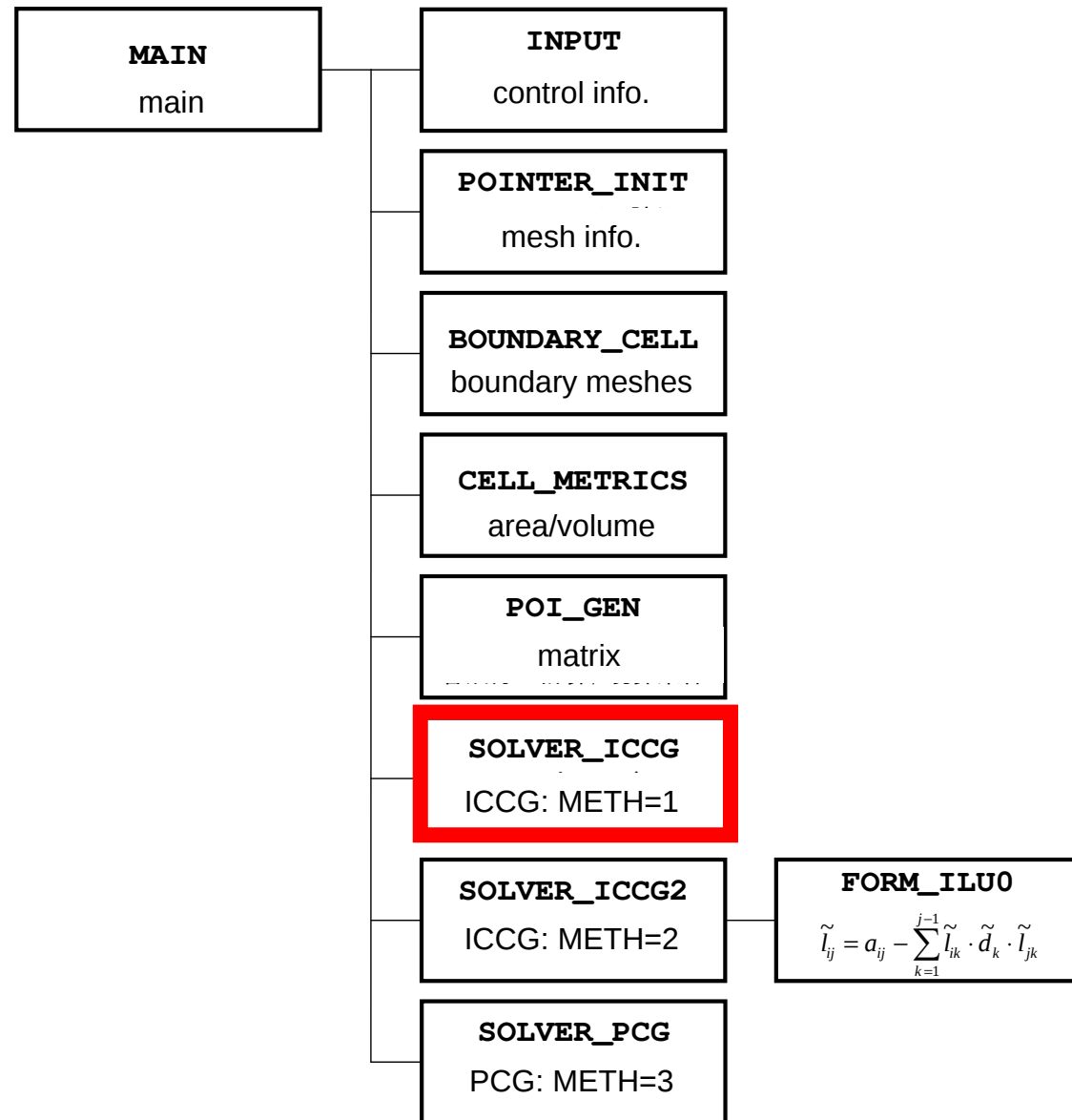
- MC
- CM-RCM
- △ No reordering

- Remedy for Data Dependency
- Ordering/Reordering
 - Red-Black, Multicoloring (MC)
 - Cuthill-McKee (CM), Reverse-CM (RCM)
 - Reordering and Convergence
- Implementation
- **ICCG with Reordering**

Implementation of Reordering to ICCG

- Apply “L2-color” to “L1-sol”
- Calling “mc”, “cm”, “rcm” and “cmrcm” after computation of “INU, INL, IAL, IAU” in “poi_gen”.
- Computing “AL,AU” by new numbering.
- B.C., and RHS are applied by new numbering.
- Calling “ICCG”
- Renumbering components of “PHI (results)” into initial numbering.
- OUTPUT_UCD (UCD file)

L1-sol



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

Forward Substitution

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

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

```
do i= 1, N
  WVAL= R(i)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL - MAL(k) * Z(itemL(k))
  enddo
  Z(i)= WVAL / D(i)
enddo
```

Backward Substitution

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

```
do i= N, 1, -1
  SW = 0.0d0
  do k= indexU(i-1)+1, indexU(i)
    SW= SW + MAU(k) * Z(itemU(k))
  enddo
  Z(i)= Z(i) - SW / MD(i)
enddo
```

Data Dependency
Z appears in both
of LHS and RHS.

Reordering may
eliminate this data
dependency.

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

Forward Substitution

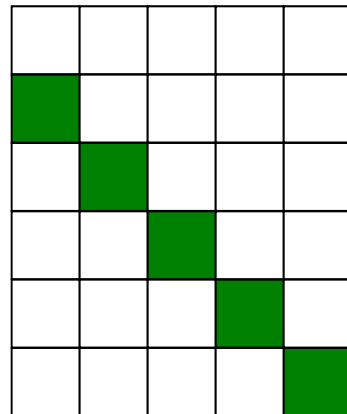
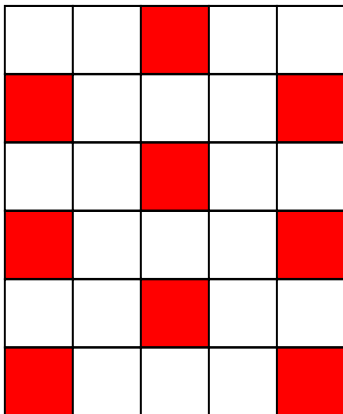
$$(L)\{z\}=\{r\}$$

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

```

do icol= 1, NCOLORTot
  do i= COLORindex(icol-1)+1, COLORindex(icol)
    WVAL= R(i)
    do k= indexL(i-1)+1, indexL(i)
      WVAL= WVAL - MAL(k) * Z(itemL(k))
    enddo
    Z(i)= WVAL / D(i)
  enddo
enddo

```



“Z” components in RHS do not belong to “icol-th” color.

Meshes in same color are independent.
(No Data Dependency)

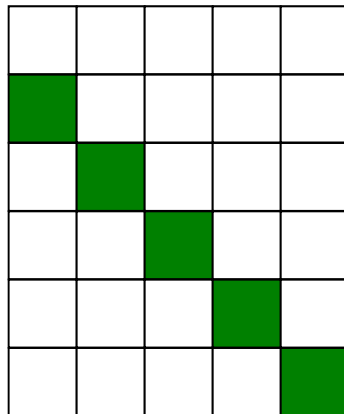
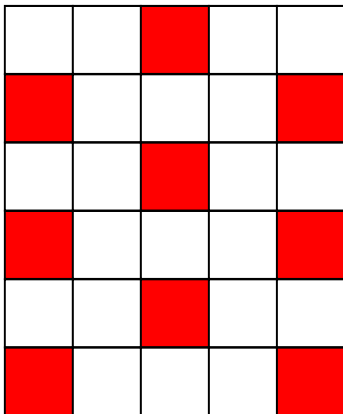
Minv{r}={z} (2/2)

Forward Substitution

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

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

```
do icol= 1, NCOLortot
  do i= COLORindex(icol-1)+1, COLORindex(icol)
    WVAL= R(i)
    do k= indexL(i-1)+1, indexL(i)
      WVAL= WVAL - MAL(k) * Z(itemL(k))
    enddo
    Z(i)= WVAL / D(i)
  enddo
enddo
```



Parallel processing can be applied to these loops.

Files

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

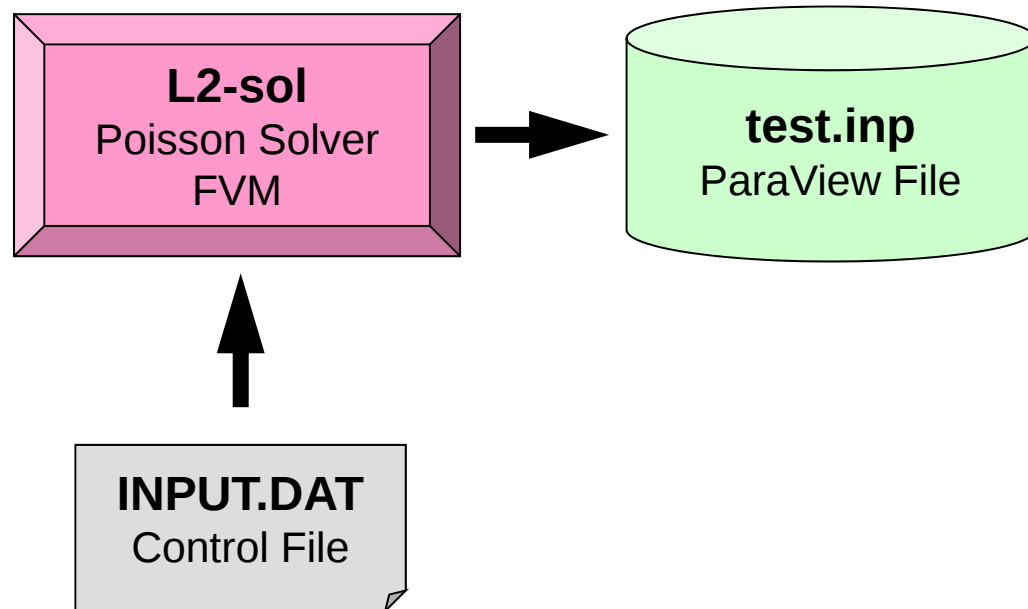
```
$> cd ../src
```

```
$> make
```

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

Running the Program

<\$P-L2>/solver/run

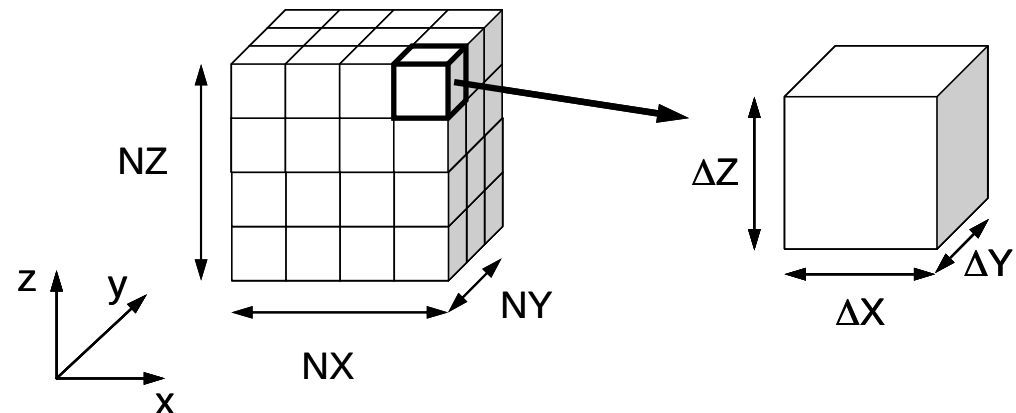


Running the Program

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

32 32 32	NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00	DX/DY/DZ
1.0e-08	EPSICCG

- NX, NY, NZ
 - Number of meshes in X/Y/Z dir.
- DX, DY, DZ
 - Size of meshes
- EPSICCG
 - Convergence Criteria for ICCG



Running the Program

<\$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
```

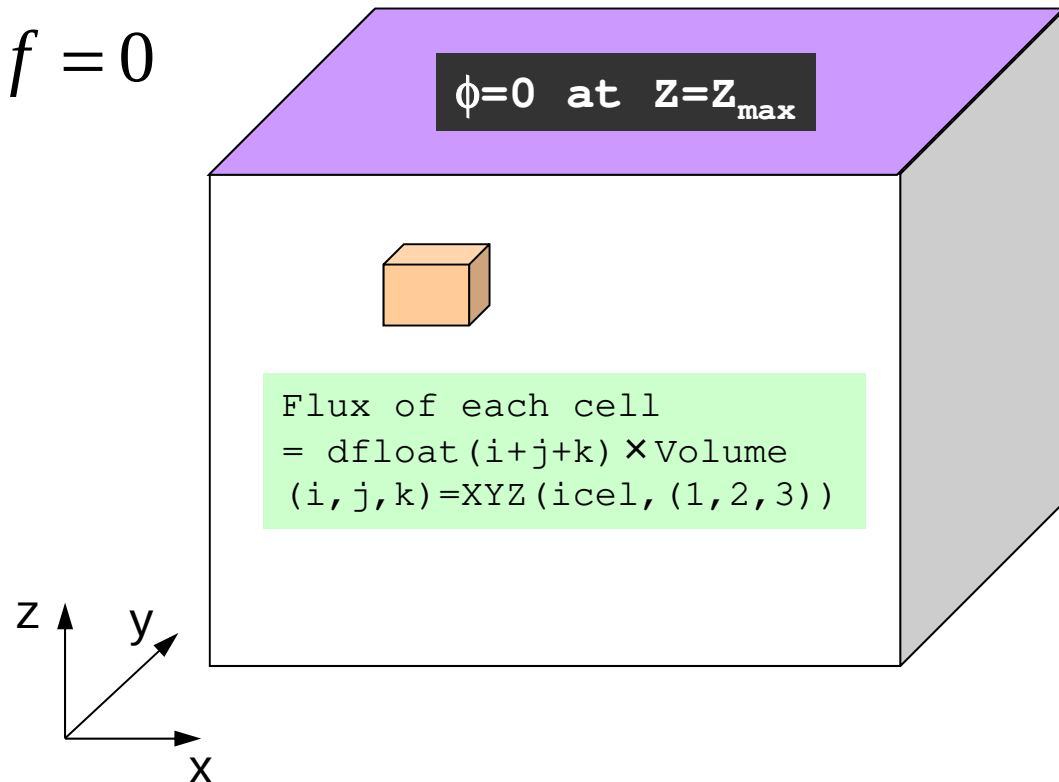
Target Problem: Variables are defined at cell-center'

Poisson Equation

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

B.C.

- Volume Flux
- $\phi = 0 @ Z = Z_{\max}$



Main Program

```

program MAIN

use STRUCT
use PCG
use solver_ICCG_mc

implicit REAL*8 (A-H,O-Z)
real(kind=8), dimension(:), allocatable :: WK

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN

PHI= 0.d0
call solve_ICCG_mc
&      ( ICELTOT, NPL, NPU, indexL, itemL, indexU, itemU, D,      &
&      BFORCE, PHI, AL, AU, NCOLORTot, COLORindex,      &
&      EPSICCG, ITR, IER)

allocate (WK(ICELTOT))

do ic0= 1, ICELTOT
  icel= NEWtoOLD(ic0)
  WK(icel)= PHI(ic0)
enddo

do icel= 1, ICELTOT
  PHI(icel)= WK(icel)
enddo

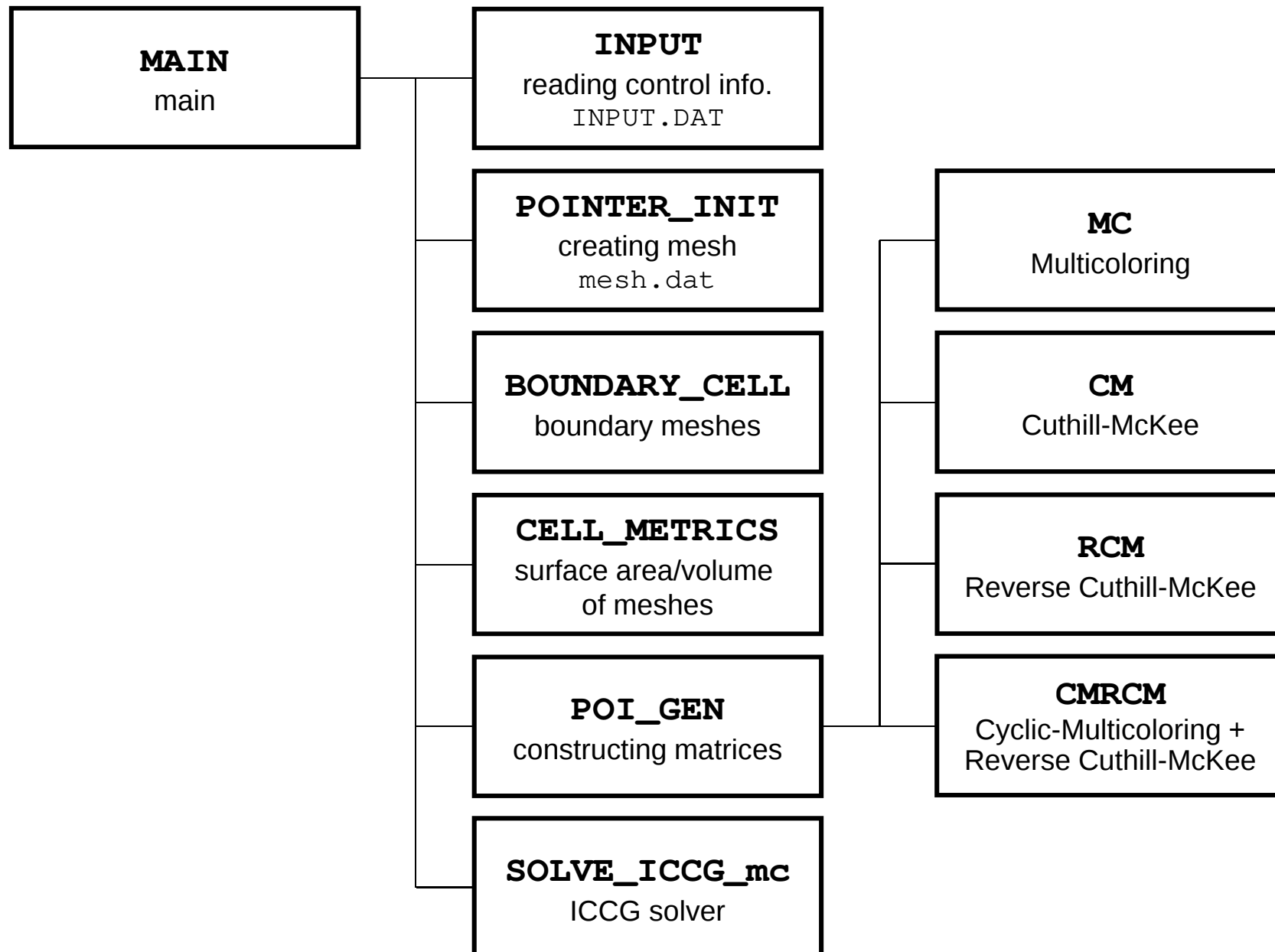
call OUTUCD

stop
end

```

Renumbering of “PHI”
to original numbering

Structure of L2-sol



Variables/Arrays for Matrix (1/2)

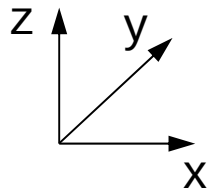
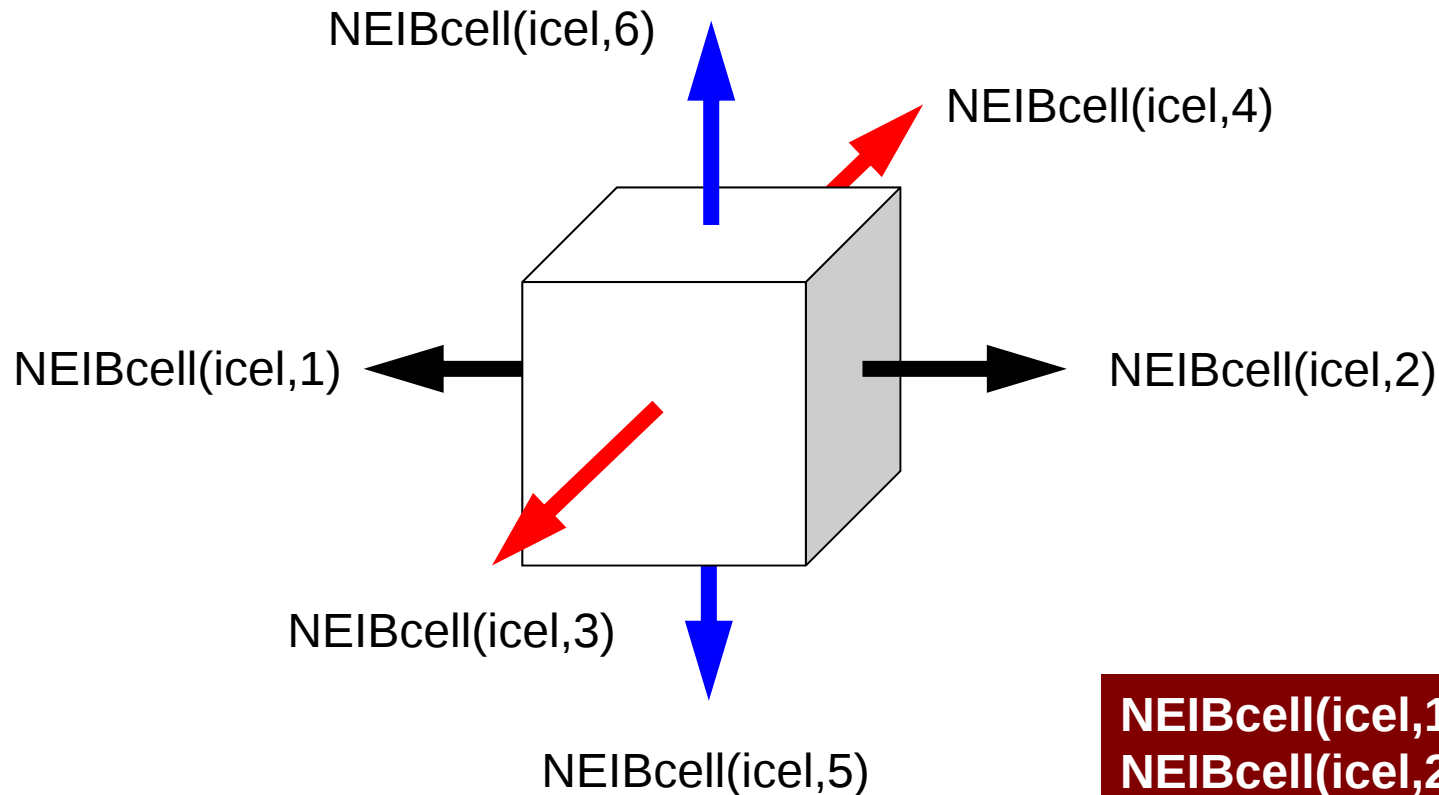
Name	Type	Content
D (N)	R	Diagonal components of the matrix (N= ICELTOT)
BFORCE (N)	R	RHS vector
PHI (N)	R	Unknown vector
indexL (0:N) , indexU (0:N)	I	# of L/U non-zero off-diag. comp. (CRS)
NPL, NPU	I	Total # of L/U non-zero off-diag. comp. (CRS)
itemL (NPL) , itemU (NPU)	I	Column ID of L/U non-zero off-diag. comp. (CRS)
AL (NPL) , AU (NPU)	R	L/U non-zero off-diag. comp. (CRS)

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

Variables/Arrays for Matrix (2/2)

Name	Type	Content
NCOLORtot	I	<u>Input:</u> reordering method + initial number of colors/levels ≥ 2 : MC, =0: CM, =-1: RCM, $-2 \leq$: CMRCM <u>Output:</u> Final number of colors/levels
COLORindex (0:NCOLORtot)	I	Number of meshes at each color/level 1D compressed array Meshes in icolth color/level are stored in this array from COLORindex(icol-1)+1 to COLORindex(icol)
NEWtoOLD(N)	I	Reference array from New to Old numbering
OLDtoNEW(N)	I	Reference array from Old to New numbering

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



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

Main Program

```

program MAIN

use STRUCT
use PCG
use solver_ICCG_mc

implicit REAL*8 (A-H,O-Z)
real(kind=8), dimension(:), allocatable :: WK

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN

PHI= 0.d0
call solve_ICCG_mc
&      ( ICELTOT, NPL, NPU, indexL, itemL, indexU, itemU, D,      &
&      BFORCE, PHI, AL, AU, NCOLORTot, COLORindex,      &
&      EPSICCG, ITR, IER)

allocate (WK(ICELTOT))
do ic0= 1, ICELTOT
  icel= NEWtoOLD(ic0)
  WK(icel)= PHI(ic0)
enddo

do icel= 1, ICELTOT
  PHI(icel)= WK(icel)
enddo

call OUTUCD

stop
end

```

poi_gen (1/8)

```
subroutine POI_GEN

  use STRUCT
  use PCG

  implicit REAL*8 (A-H, O-Z)

!C
!C--- INIT.
  nn = ICELTOT

  NU= 6
  NL= 6

  allocate (BFORCE(nn), D(nn), PHI(nn))
  allocate (INL(nn), INU(nn), IAL(NL, nn), IAU(NU, nn))

  PHI= 0. d0
  D= 0. d0
  BFORCE= 0. d0

  INL= 0
  INU= 0
  IAL= 0
  IAU= 0
```

```

!C
!C +-----+
!C | CONNECTIVITY |
!C +-----+
!C===
      do icel= 1, ICELTOT
        icN1= NEIBcell (icel, 1)
        icN2= NEIBcell (icel, 2)
        icN3= NEIBcell (icel, 3)
        icN4= NEIBcell (icel, 4)
        icN5= NEIBcell (icel, 5)
        icN6= NEIBcell (icel, 6)

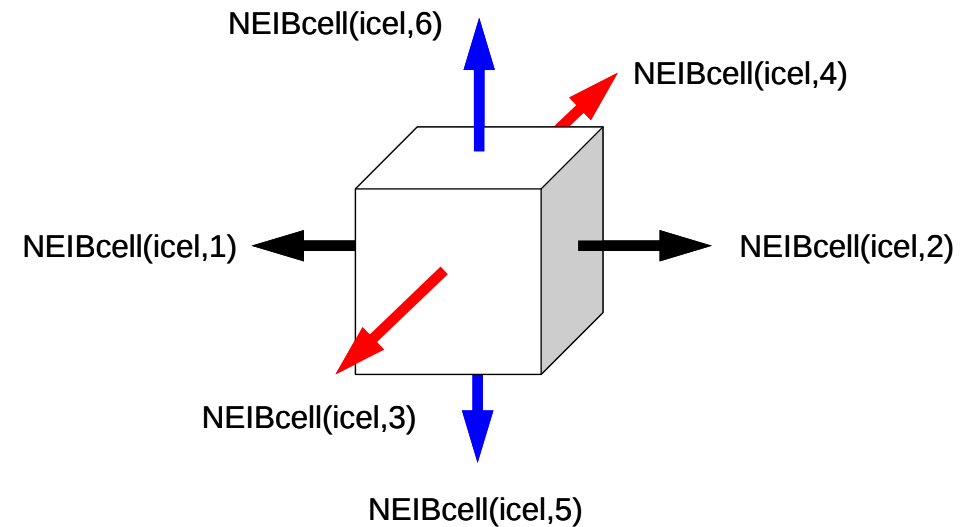
        icouG= 0
        if (icN5.ne. 0. and. icN5.le. ICELTOT) then
          icou= INL(icel) + 1
          IAL(icou, icel)= icN5
          INL(      icel)= icou
        endif

        if (icN3.ne. 0. and. icN3.le. ICELTOT) then
          icou= INL(icel) + 1
          IAL(icou, icel)= icN3
          INL(      icel)= icou
        endif

        if (icN1.ne. 0. and. icN1.le. ICELTOT) then
          icou= INL(icel) + 1
          IAL(icou, icel)= icN1
          INL(      icel)= icou
        endif
      enddo

```

poi_gen (2/8)



Lower Triangular Part

```

NEIBcell(icel,5)= icel - NX*NY
NEIBcell(icel,3)= icel - NX
NEIBcell(icel,1)= icel - 1

```

```

!C
!C +-----+
!C | CONNECTIVITY |
!C +-----+
!C===
      do icel= 1, ICELTOT
        icN1= NEIBcell (icel, 1)
        icN2= NEIBcell (icel, 2)
        icN3= NEIBcell (icel, 3)
        icN4= NEIBcell (icel, 4)
        icN5= NEIBcell (icel, 5)
        icN6= NEIBcell (icel, 6)

        icouG= 0

...

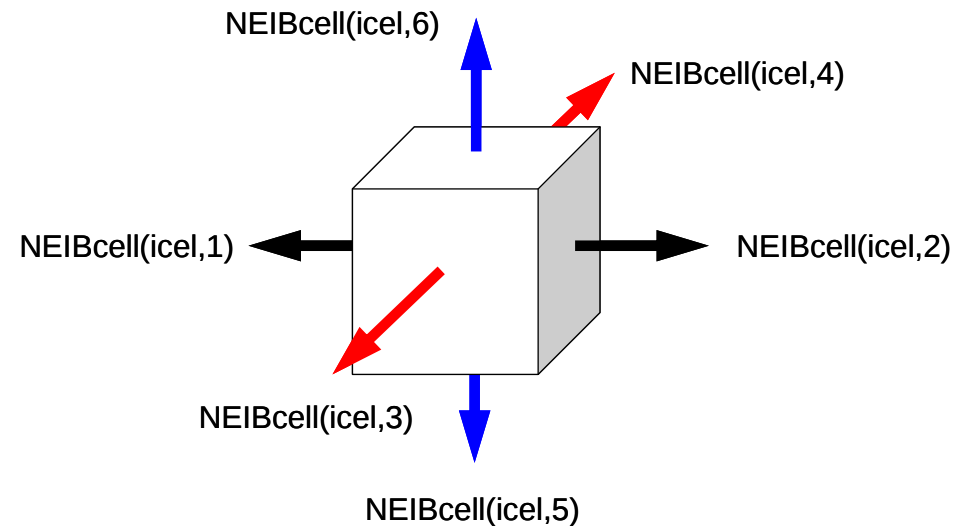
      if (icN2.ne. 0.and. icN2.le. ICELTOT) then
        icou= INU(icel) + 1
        IAU(icou, icel)= icN2
        INU(      icel)= icou
      endif

      if (icN4.ne. 0.and. icN4.le. ICELTOT) then
        icou= INU(icel) + 1
        IAU(icou, icel)= icN4
        INU(      icel)= icou
      endif

      if (icN6.ne. 0.and. icN6.le. ICELTOT) then
        icou= INU(icel) + 1
        IAU(icou, icel)= icN6
        INU(      icel)= icou
      endif
    enddo
!C===

```

poi_gen (3/8)



Upper Triangular Part

```

NEIBcell(icel,2)= icel + 1
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,6)= icel + NX*NY

```

poi_gen (4/8)

```

!C
!C +-----+
!C | MULTICOLORING |
!C +-----+
!C===
      allocate (OLDtoNEW(ICELTOT), NEWtoOLD(ICELTOT))
      allocate (COLORindex(0:ICELTOT))

111   continue
      write (*, '(//a,i8,a)') 'You have', ICELTOT, ' elements.'
      write (*, '( a )') 'How many colors do you need ?'
      write (*, '( a )') ' #COLOR must be more than 2 and'
      write (*, '( a,i8 )') ' #COLOR must not be more than', ICELTOT
      write (*, '( a )') ' CM if #COLOR .eq. 0'
      write (*, '( a )') ' RCM if #COLOR .eq. -1'
      write (*, '( a )') ' CMRCM if #COLOR .le. -2'
      write (*, '( a )') '=>'
      read (*,*) NCOLORTot
      if (NCOLORTot.eq.1.or.NCOLORTot.gt.ICELTOT) goto 111

      if (NCOLORTot.gt.0) then
        call MC (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&              NCOLORTot, COLORindex, NEWtoOLD, OLDtoNEW)
&
      endif
      if (NCOLORTot.eq.0) then
        call CM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&              NCOLORTot, COLORindex, NEWtoOLD, OLDtoNEW)
&
      endif
      if (NCOLORTot.eq.-1) then
        call RCM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&              NCOLORTot, COLORindex, NEWtoOLD, OLDtoNEW)
&
      endif
      if (NCOLORTot.lt.-1) then
        call CMRCM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&              NCOLORTot, COLORindex, NEWtoOLD, OLDtoNEW)
&
      endif

      write (*, '(//a,i8,//)') '### FINAL COLOR NUMBER', NCOLORTot
!C===

```

!C
!C--- 1D array

```
allocate (indexL(0:nn), indexU(0:nn))
indexL= 0
indexU= 0

do icel= 1, ICELTOT
  indexL(icel)= INL(icel)
  indexU(icel)= INU(icel)
enddo

do icel= 1, ICELTOT
  indexL(icel)= indexL(icel) + indexL(icel-1)
  indexU(icel)= indexU(icel) + indexU(icel-1)
enddo
```

```
NPL= indexL(ICELTOT)
NPU= indexU(ICELTOT)
```

```
allocate (itemL(NPL), AL(NPL))
allocate (itemU(NPU), AU(NPU))
```

```
itemL= 0
itemU= 0
```

```
AL= 0. d0
AU= 0. d0
```

!C===

```
do i= 1, N
  VAL= D(i)*p(i)

  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL + AL(k)*p(itemL(k))
  enddo

  do k= indexU(i-1)+1, indexU(i)
    VAL= VAL + AU(k)*p(itemU(k))
  enddo

  q(i)= VAL
enddo
```

poi_gen (5/8)

New numbering is applied after this point

Name	Type	Content
D (N)	R	Diagonal components of the matrix (N= ICELTOT)
BFORCE (N)	R	RHS vector
PHI (N)	R	Unknown vector
indexL (0:N) , indexU (0:N)	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)

```

!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C==
      icouG= 0
      do icol= 1, NCOLortot
      do icel= COLORindex(icol-1)+1, COLORindex(icol)
      ic0 = NEWtoOLD(icel)
      icN1= NEIBcell(ic0, 1)
      icN2= NEIBcell(ic0, 2)
      icN3= NEIBcell(ic0, 3)
      icN4= NEIBcell(ic0, 4)
      icN5= NEIBcell(ic0, 5)
      icN6= NEIBcell(ic0, 6)
      VOL0= VOLCEL (ic0)

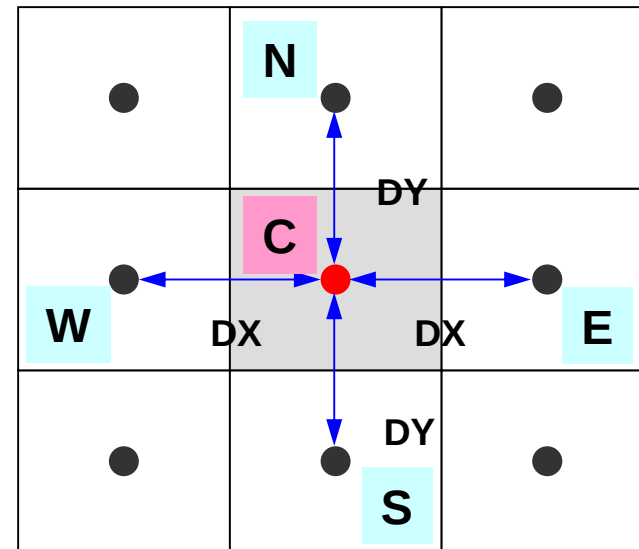
      if (icN5.ne.0) then
      icN5= OLDtoNEW(icN5)
      coef= RDZ * ZAREA
      D(icel)= D(icel) - coef

      if (icN5.lt.icel) then
      do j= 1, INL(icel)
      if (IAL(j, icel).eq.icN5) then
      itemL(j+indexL(icel-1))= icN5
      AL(j+indexL(icel-1))= coef
      exit
      endif
      enddo
      else
      do j= 1, INU(icel)
      if (IAU(j, icel).eq.icN5) then
      itemU(j+indexU(icel-1))= icN5
      AU(j+indexU(icel-1))= coef
      exit
      endif
      enddo
      endif
      endif
      endif

```

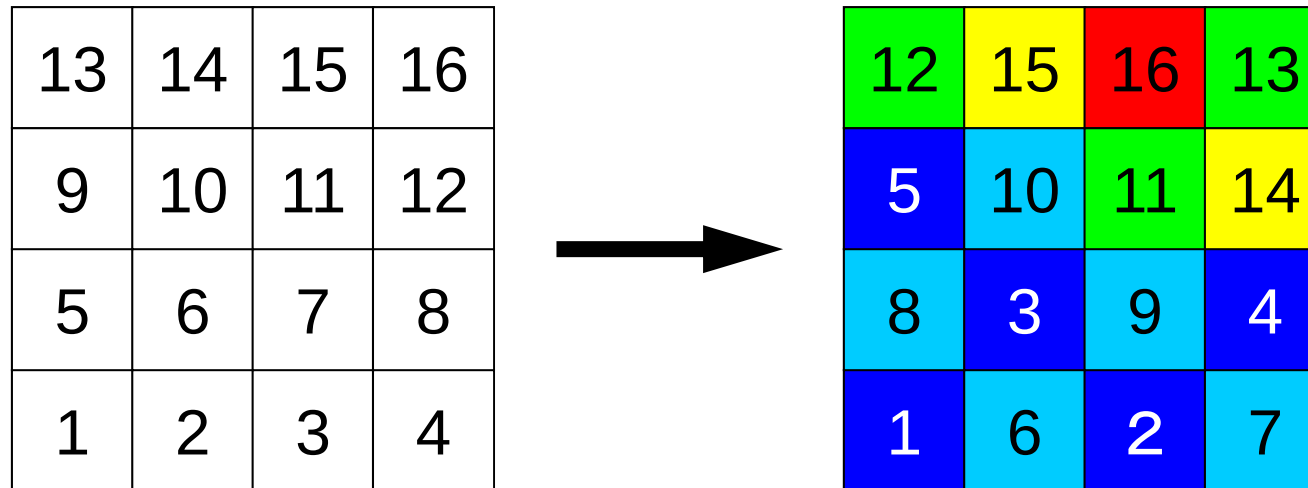
poi_gen (6/8)

Calculation of Coefficients



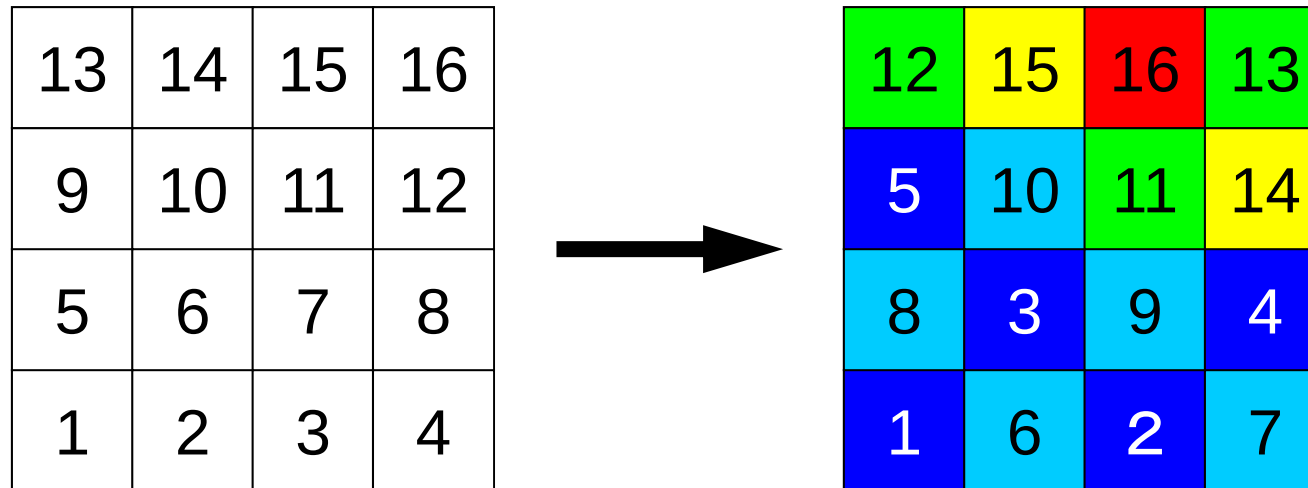
$$\begin{aligned}
 & \frac{\phi_E - \phi_i}{\Delta x} \Delta y + \frac{\phi_W - \phi_i}{\Delta x} \Delta y + \\
 & \frac{\phi_N - \phi_i}{\Delta y} \Delta x + \frac{\phi_S - \phi_i}{\Delta y} \Delta x = f_c \Delta x \Delta y
 \end{aligned}$$

New Numbering



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

New Numbering (cont.)



NCOLORtot= 5

COLORindex(0)= 0, COLORindex(1)= 5, COLORindex(2)= 10

COLORindex(3)= 13, COLORindex(4)= 15, COLORindex(5)= 16

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

```

!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C==
      icouG= 0
      do icol= 1, NCOLortot
      do icel= COLORindex(icol-1)+1, COLORindex(icol)
        ic0 = NEWtoOLD(icel)
        icN1= NEIBcell(ic0, 1)
        icN2= NEIBcell(ic0, 2)
        icN3= NEIBcell(ic0, 3)
        icN4= NEIBcell(ic0, 4)
        icN5= NEIBcell(ic0, 5)
        icN6= NEIBcell(ic0, 6)
        VOLO= VOLCEL (ic0)

        if (icN5.ne.0) then
          icN5= OLDtoNEW(icN5)
          coef= RDZ * ZAREA
          D(icel)= D(icel) - coef

          if (icN5.lt.icel) then
            do j= 1, INL(icel)
              if (IAL(j, icel).eq.icN5) then
                itemL(j+indexL(icel-1))= icN5
                AL(j+indexL(icel-1))= coef
              exit
            endif
          enddo
        else
          do j= 1, INU(icel)
            if (IAU(j, icel).eq.icN5) then
              itemU(j+indexU(icel-1))= icN5
              AU(j+indexU(icel-1))= coef
            exit
          endif
        enddo
      endif
    enddo
  enddo
endif

```

poi_gen (6/8)

$$\frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z$$

```

!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C==
      icouG= 0
      do icol= 1, NCOLortot
      do icel= COLORindex(icol-1)+1, COLORindex(icol)
        ic0 = NEWtoOLD(icel)
        icN1= NEIBcell (ic0, 1)
        icN2= NEIBcell (ic0, 2)
        icN3= NEIBcell (ic0, 3)
        icN4= NEIBcell (ic0, 4)
        icN5= NEIBcell (ic0, 5)
        icN6= NEIBcell (ic0, 6)
        VOL0= VOLCEL (ic0)

        if (icN5.ne.0) then
          icN5= OLDtoNEW(icN5)
          coef= RDZ * ZAREA
          D(icel)= D(icel) - coef
          if (icN5.lt.icel) then
            do j= 1, INL(icel)
              if (IAL(j,icel).eq.icN5) then
                itemL(j+indexL(icel-1))= icN5
                AL(j+indexL(icel-1))= coef
              exit
            endif
          enddo
        else
          do j= 1, INU(icel)
            if (IAU(j,icel).eq.icN5) then
              itemU(j+indexU(icel-1))= icN5
              AU(j+indexU(icel-1))= coef
            exit
          endif
        enddo
      endif
    enddo
  enddo
enddo

```

**icN5 < icel
Lower Part**

poi_gen (6/8)

$$\frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z$$

```

!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C==
      icouG= 0
      do icol= 1, NCOLortot
      do icel= COLORindex(icol-1)+1, COLORindex(icol)
        ic0 = NEWtoOLD(icel)
        icN1= NEIBcell (ic0, 1)
        icN2= NEIBcell (ic0, 2)
        icN3= NEIBcell (ic0, 3)
        icN4= NEIBcell (ic0, 4)
        icN5= NEIBcell (ic0, 5)
        icN6= NEIBcell (ic0, 6)
        VOL0= VOLCEL  (ic0)

        if (icN5.ne.0) then
          icN5= OLDtoNEW(icN5)
          coef= RDZ * ZAREA
          D(icel)= D(icel) - coef
          if (icN5.lt.icel) then
            do j= 1, INL(icel)
              if (IAL(j,icel).eq.icN5) then
                itemL(j+indexL(icel-1))= icN5
                AL(j+indexL(icel-1))= coef
              exit
            endif
          enddo
        else
          do j= 1, INU(icel)
            if (IAU(j,icel).eq.icN5) then
              itemU(j+indexU(icel-1))= icN5
              AU(j+indexU(icel-1))= coef
            exit
          endif
        enddo
      endif
    enddo
  enddo
endif

```

**icN5 > icel
Upper Part**

poi_gen (6/8)

$$\frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z$$

```

if (icN6.ne.0) then
  icN6= OLDtoNEW(icN6)
  coef= RDZ * ZAREA
  D(icel)= D(icel) - coef

  if (icN6.lt.icel) then
    do j= 1, INL(icel)
      if (IAL(j,icel).eq.icN6) then
        itemL(j+indexL(icel-1))= icN6
        AL(j+indexL(icel-1))= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j,icel).eq.icN6) then
        itemU(j+indexU(icel-1))= icN6
        AU(j+indexU(icel-1))= coef
        exit
      endif
    enddo
  endif
endif
endif

```

```

ii= XYZ(ic0,1)
jj= XYZ(ic0,2)
kk= XYZ(ic0,3)

```

```

BFORCE(icel)= -dfloat(ii+jj+kk) * VOL0

```

```

enddo
enddo

```

```

!C===

```

BFORCE
using original
mesh ID

poi_gen (7/8)

$$\frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z$$

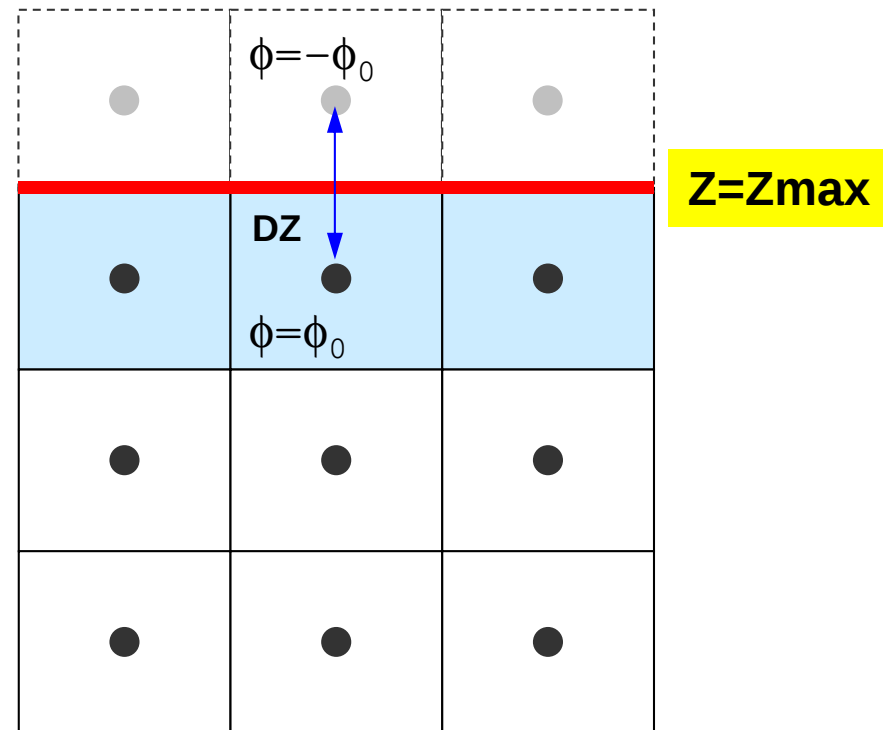
```

!C
!C +-----+
!C | DIRICHLET BOUNDARY CELLS |
!C +-----+
!C TOP SURFACE
!C==
  do ib= 1, ZmaxCELtot
    ic0= ZmaxCEL(ib)
    coef= 2.d0 * RDZ * ZAREA
    icel= OLDtoNEW(ic0)
    D(icel)= D(icel) - coef
  enddo
!C==
  return
end

```

poi_gen (8/8)

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



1st 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

```

program MAIN

use STRUCT
use PCG
use solver_ICCG_mc

implicit REAL*8 (A-H,O-Z)
real(kind=8), dimension(:), allocatable :: WK

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN

PHI= 0.d0
call solve_ICCG_mc
&      ( ICELTOT, NPL, NPU, indexL, itemL, indexU, itemU, D, &
&      BFORCE, PHI, AL, AU, NCOLORTot, COLORindex, &
&      EPSICCG, ITR, IER)
allocate (WK(ICELTOT))

do ic0= 1, ICELTOT
  icel= NEWtoOLD(ic0)
  WK(icel)= PHI(ic0)
enddo

do icel= 1, ICELTOT
  PHI(icel)= WK(icel)
enddo

call OUTUCD

stop
end

```

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

solve_ICCG_mc (1/7)

```

!C***
!C*** module solver_ICCG_mc
!C***
!
      module solver_ICCG_mc
      contains
!C
!C*** solve_ICCG
!C
      subroutine solve_ICCG_mc                                &
      &      ( N, NPL, NPU, indexL, itemL, indexU, itemU, D, B, X, &
      &      AL, AU, NCOLORtot, COLORindex, EPS, ITR, IER)
!C
      implicit REAL*8 (A-H, O-Z)

      integer :: N, NL, NU, NCOLOR

      real(kind=8), dimension(N) :: D
      real(kind=8), dimension(N) :: B
      real(kind=8), dimension(N) :: X
      real(kind=8), dimension(NPL) :: AL
      real(kind=8), dimension(NPU) :: AU

      integer, dimension(0:N) :: indexL, indexU
      integer, dimension(NPL) :: itemL
      integer, dimension(NPU) :: itemU

      integer, dimension(0:NCOLORtot) :: COLORindex

      real(kind=8), dimension(:, :), allocatable :: W

      integer, parameter :: R= 1
      integer, parameter :: Z= 2
      integer, parameter :: Q= 2
      integer, parameter :: P= 3
      integer, parameter :: DD= 4

```

solve_ICCG_mc (2/7)

```

!C
!C +-----+
!C |  INIT  |
!C +-----+
!C==
      allocate (W(N,4))
      do i= 1, N
        X(i) = 0. d0
        W(i, 2)= 0. 0D0
        W(i, 3)= 0. 0D0
        W(i, 4)= 0. 0D0
      enddo

      do ic= 1, NCOLRtot
        do i= COLORindex(ic-1)+1, COLORindex(ic)
          VAL= D(i)
          do k= indexL(i-1)+1, indexL(i)
            VAL= VAL - (AL(k)**2) * W(itemL(k), DD)
          enddo
          W(i, DD)= 1. d0/VAL
        enddo
      enddo
!C==

```

**Incomplete
"Modified" Cholesky
Factorization**

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

```

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) + αi p(i)
  r(i) = r(i-1) - αi q(i)
  check convergence |r|
end

```

solve_ICCG_mc (2/7)

```

!C
!C +-----+
!C |  INIT  |
!C +-----+
!C==
      allocate (W(N,4))
      do i= 1, N
        X(i) = 0. d0
        W(i, 2)= 0. 0D0
        W(i, 3)= 0. 0D0
        W(i, 4)= 0. 0D0
      enddo

      do ic= 1, NCOLRtot
        do i= COLORindex(ic-1)+1, COLORindex(ic)
          VAL= D(i)
          do k= indexL(i-1)+1, indexL(i)
            VAL= VAL - (AL(k)**2) * W(itemL(k),DD)
          enddo
          W(i,DD)= 1. d0/VAL
        enddo
      enddo
!C==

```

**Incomplete
"Modified" Cholesky
Factorization**

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



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

$W(i,DD):$	d_i
$D(i):$	a_{ii}
$itemL(j):$	k
$AL(j):$	a_{ik}

Incomplete “Modified” Cholesky Factorization

```
do i= 1, N
  VAL= D(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL - (AL(k)**2) * W(itemL(k), DD)
  enddo
  W(i, DD)= 1. d0/VAL
enddo
```



```
do ic= 1, NCOLORtot
  do i= COLORindex(ic-1)+1, COLORindex(ic)
    VAL= D(i)
    do k= indexL(i-1)+1, indexL(i)
      VAL= VAL - (AL(k)**2) * W(itemL(k), DD)
    enddo
    W(i, DD)= 1. d0/VAL
  enddo
enddo
```

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

NO data dependency.

solve_ICCG_mc (3/7)

```

!C
!C +-----+
!C | {r0} = {b} - [A] {xini} |
!C +-----+
!C==
do i= 1, N
  VAL= D(i)*X(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL + AL(k)*X(itemL(k))
  enddo
  do k= indexU(i-1)+1, indexU(i)
    VAL= VAL + AU(k)*X(itemU(k))
  enddo
  W(i, R)= B(i) - VAL
enddo

BNRM2= 0.0D0
do i= 1, N
  BNRM2 = BNRM2 + B(i) **2
enddo
!C==

```

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) + αi p(i)
  r(i) = r(i-1) - αi q(i)
  check convergence |r|
end

```

solve_ICCG_mc (4/7)

```

!C
!C***** ITERATION
      ITR= N

      do L= 1, ITR

!C
!C +-----+
!C | {z}= [Minv] {r} |
!C +-----+
!C==
        do i= 1, N
          W(i,Z)= W(i,R)
        enddo

        do ic= 1, NCOLRtot
          do i= COLORindex(ic-1)+1, COLORindex(ic)
            WVAL= W(i,Z)
            do k= indexL(i-1)+1, indexL(i)
              WVAL= WVAL - AL(k) * W(itemL(k),Z)
            enddo
            W(i,Z)= WVAL * W(i,DD)
          enddo
        enddo

        do ic= NCOLRtot, 1, -1
          do i= COLORindex(ic-1)+1, COLORindex(ic)
            SW = 0.0d0
            do k= indexU(i-1)+1, indexU(i)
              SW= SW + AU(k) * W(itemU(k),Z)
            enddo
            W(i,Z)= W(i,Z) - W(i,DD) * SW
          enddo
        enddo
!C==

```

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

for $i = 1, 2, \dots$

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

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

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)

```

!C
!C***** ITERATION
      ITR= N

      do L= 1, ITR

!C
!C +-----+
!C | {z}= [Minv] {r} |
!C +-----+
!C==
      do i= 1, N
        W(i, Z)= W(i, R)
      enddo

      do ic= 1, NCOLRtot
        do i= COLORindex(ic-1)+1, COLORindex(ic)
          WVAL= W(i, Z)
          do k= indexL(i-1)+1, indexL(i)
            WVAL= WVAL - AL(k) * W(itemL(k), Z)
          enddo
          W(i, Z)= WVAL * W(i, DD)
        enddo
      enddo

      do ic= NCOLRtot, 1, -1
        do i= COLORindex(ic-1)+1, COLORindex(ic)
          SW = 0.0d0
          do k= indexU(i-1)+1, indexU(i)
            SW= SW + AU(k) * W(itemU(k), Z)
          enddo
          W(i, Z)= W(i, Z) - W(i, DD) * SW
        enddo
      enddo
!C==

```

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

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

Forward Substitution

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

solve_ICCG_mc (4/7)

```

!C
!C***** ITERATION
      ITR= N

      do L= 1, ITR

!C
!C +-----+
!C | {z}= [Minv] {r} |
!C +-----+
!C==
      do i= 1, N
        W(i, Z)= W(i, R)
      enddo

      do ic= 1, NCOLRtot
        do i= COLORindex(ic-1)+1, COLORindex(ic)
          WVAL= W(i, Z)
          do k= indexL(i-1)+1, indexL(i)
            WVAL= WVAL - AL(k) * W(itemL(k), Z)
          enddo
          W(i, Z)= WVAL * W(i, DD)
        enddo
      enddo

      do ic= NCOLRtot, 1, -1
        do i= COLORindex(ic-1)+1, COLORindex(ic)
          SW = 0.0d0
          do k= indexU(i-1)+1, indexU(i)
            SW= SW + AU(k) * W(itemU(k), Z)
          enddo
          W(i, Z)= W(i, Z) - W(i, DD) * SW
        enddo
      enddo
!C==

```

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

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

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)

```

!C
!C***** ITERATION
      ITR= N

      do L= 1, ITR

!C
!C +-----+
!C | {z}= [Minv] {r} |
!C +-----+
!C==
        do i= 1, N
          W(i, Z)= W(i, R)
        enddo

!C
!C (M){z} = (LDL^T){z} = {r}
!C
        do ic= 1, NCOLRtot
          do i= COLORindex(ic-1)+1, COLORindex(ic)
            WVAL= W(i, Z)
            do k= indexL(i-1)+1, indexL(i)
              WVAL= WVAL - AL(k) * W(itemL(k), Z)
            enddo
            W(i, Z)= WVAL * W(i, DD)
          enddo
        enddo

!C
!C (L){z} = {r}
!C
        do ic= NCOLRtot, 1, -1
          do i= COLORindex(ic-1)+1, COLORindex(ic)
            SW = 0.0d0
            do k= indexU(i-1)+1, indexU(i)
              SW= SW + AU(k) * W(itemU(k), Z)
            enddo
            W(i, Z)= W(i, Z) - W(i, DD) * SW
          enddo
        enddo
!C==

```

Forward Substitution

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

solve_ICCG_mc (4/7)

```

!C
!C***** ITERATION
      ITR= N

      do L= 1, ITR

!C
!C +-----+
!C | {z}= [Minv] {r} |
!C +-----+
!C==
      do i= 1, N
        W(i, Z)= W(i, R)
      enddo

      do ic= 1, NCOLRtot
        do i= COLORindex(ic-1)+1, COLORindex(ic)
          WVAL= W(i, Z)
          do k= indexL(i-1)+1, indexL(i)
            WVAL= WVAL - AL(k) * W(itemL(k), Z)
          enddo
          W(i, Z)= WVAL * W(i, DD)
        enddo
      enddo

      do ic= NCOLRtot, 1, -1
        do i= COLORindex(ic-1)+1, COLORindex(ic)
          SW = 0.0d0
          do k= indexU(i-1)+1, indexU(i)
            SW= SW + AU(k) * W(itemU(k), Z)
          enddo
          W(i, Z)= W(i, Z) - W(i, DD) * SW
        enddo
      enddo
!C==

```

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

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

Forward Substitution

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

ic=4

solve_ICCG_mc (4/7)

```

!C
!C***** ITERATION
      ITR= N

      do L= 1, ITR

!C
!C +-----+
!C | {z}= [Minv] {r} |
!C +-----+
!C==
      do i= 1, N
        W(i, Z)= W(i, R)
      enddo

      do ic= 1, NCOLRtot
        do i= COLORindex(ic-1)+1, COLORindex(ic)
          WVAL= W(i, Z)
          do k= indexL(i-1)+1, indexL(i)
            WVAL= WVAL - AL(k) * W(itemL(k), Z)
          enddo
          W(i, Z)= WVAL * W(i, DD)
        enddo
      enddo

      do ic= NCOLRtot, 1, -1
        do i= COLORindex(ic-1)+1, COLORindex(ic)
          SW = 0.0d0
          do k= indexU(i-1)+1, indexU(i)
            SW= SW + AU(k) * W(itemU(k), Z)
          enddo
          W(i, Z)= W(i, Z) - W(i, DD) * SW
        enddo
      enddo
!C==

```

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

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

Forward Substitution

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

Backward Substitution

Forward/Backward Substitution

前進後退代入

```

do i= 1, N
  WVAL= W(i, Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL - AL(k) * W(itemL(k), Z)
  enddo
  W(i, Z)= WVAL * W(i, DD)
enddo

do i= N, 1, -1
  SW= 0.0d0
  do k= indexU(i-1)+1, indexLU(i)
    SW= SW + AU(k) * W(itemU(k), Z)
  enddo
  W(i, Z)= W(i, Z) - W(i, DD) * SW
enddo

```



```

do ic= 1, NCOLORTot
  do i= COLORindex(ic-1)+1, COLORindex(ic)
    WVAL= W(i, Z)
    do k= indexL(i-1)+1, indexL(i)
      WVAL= WVAL - AL(k) * W(itemL(k), Z)
    enddo
    W(i, Z)= WVAL * W(i, DD)
  enddo
enddo

do ic= NCOLORTot, 1, -1
  do i= COLORindex(ic-1)+1, COLORindex(ic)
    SW= 0.0d0
    do k= indexL(i-1)+1, indexL(i)
      SW= SW + AU(k) * W(itemU(k), Z)
    enddo
    W(i, Z)= W(i, Z) - W(i, DD) * SW
  enddo
enddo

```

solve_ICCG_mc (5/7)

```

!C
!C +-----+
!C | RHO= {r} {z} |
!C +-----+
!C==
      RHO= 0. d0
      do i= 1, N
        RHO= RHO + W(i, R)*W(i, Z)
      enddo
!C==

!C
!C +-----+
!C | {p} = {z} if      ITER=1
!C | BETA= RHO / RHO1 otherwise
!C +-----+
!C==
      if ( L. eq. 1 ) then
        do i= 1, N
          W(i, P)= W(i, Z)
        enddo
      else
        BETA= RHO / RHO1
        do i= 1, N
          W(i, P)= W(i, Z) + BETA*W(i, P)
        enddo
      endif
!C==

```

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)

```

!C
!C +-----+
!C | {q} = [A] {p} |
!C +-----+
!C==
do i= 1, N
  VAL= D(i)*W(i,P)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL + AL(k)*W(itemL(k),P)
  enddo
  do k= indexU(i-1)+1, indexU(i)
    VAL= VAL + AU(k)*W(itemU(k),P)
  enddo
  W(i,Q)= VAL
enddo
!C==

```

```

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

```

solve_ICCG_mc (7/7)

```

!C
!C +-----+
!C | ALPHA= RHO / {p} {q} |
!C +-----+
!C==
      C1= 0. d0
      do i= 1, N
        C1= C1 + W(i, P)*W(i, Q)
      enddo
      ALPHA= RHO / C1
!C==
!C
!C +-----+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +-----+
!C==
      do i= 1, N
        X(i) = X(i) + ALPHA * W(i, P)
        W(i, R)= W(i, R) - ALPHA * W(i, Q)
      enddo
      DNRM2= 0. d0
      do i= 1, N
        DNRM2= DNRM2 + W(i, R)**2
      enddo
!C==
      ERR = dsqrt(DNRM2/BNRM2)
      if (ERR .lt. EPS) then
        IER = 0
        goto 900
      else
        RH01 = RHO
      endif
      enddo
      IER = 1

900 continue

```

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

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

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

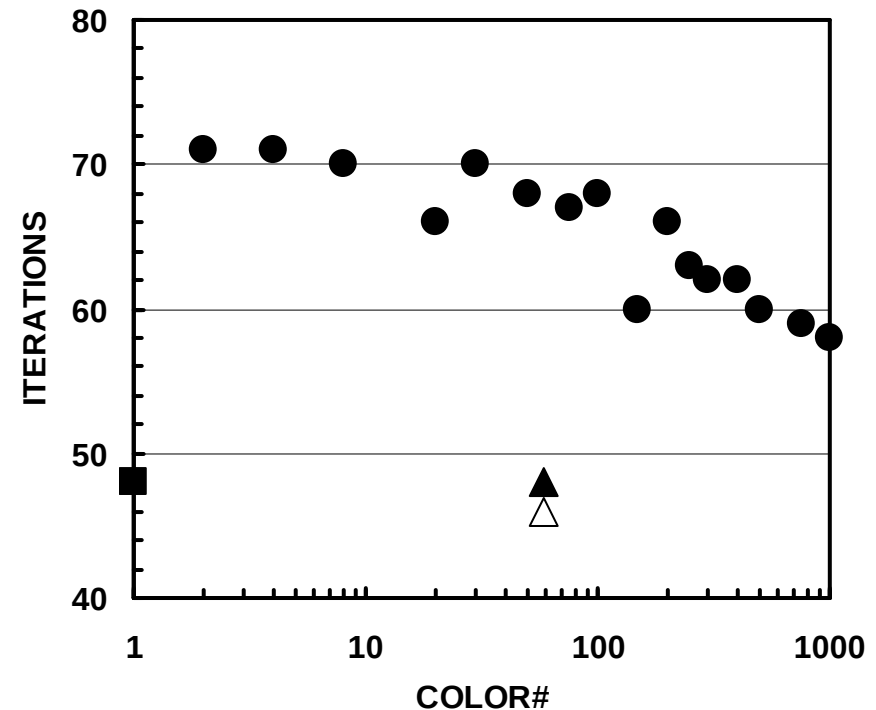
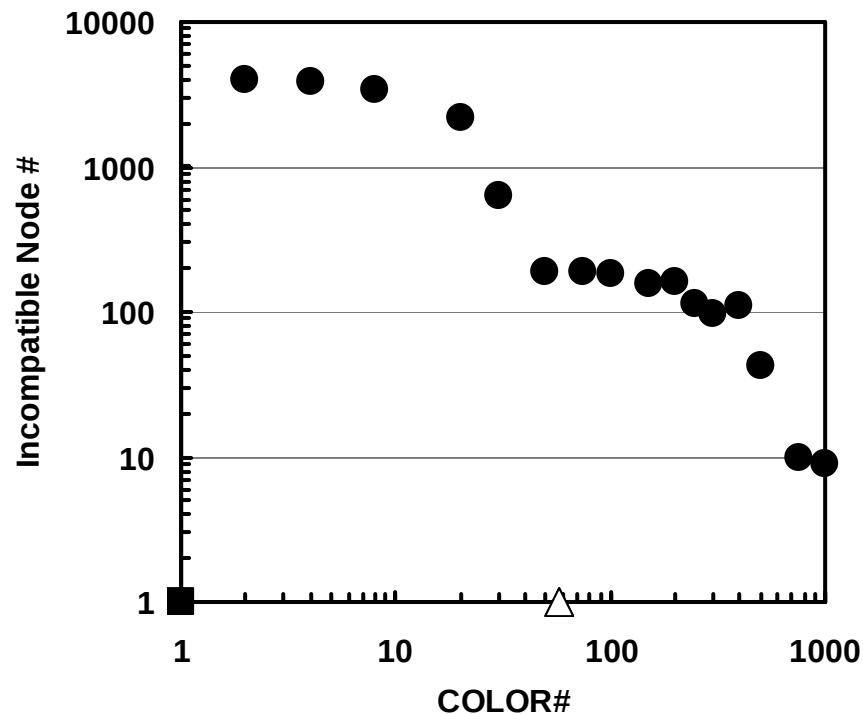
$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

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

check convergence $|r|$

end

Effect of Color Number on Convergence of ICCG



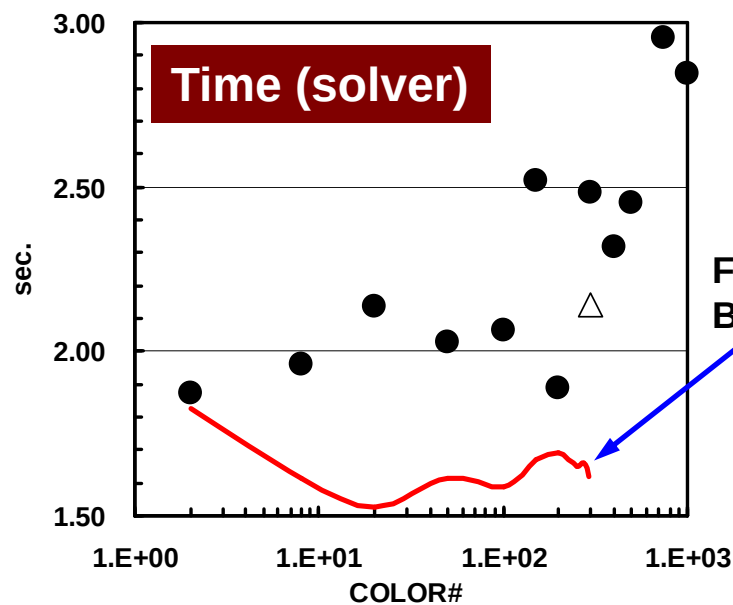
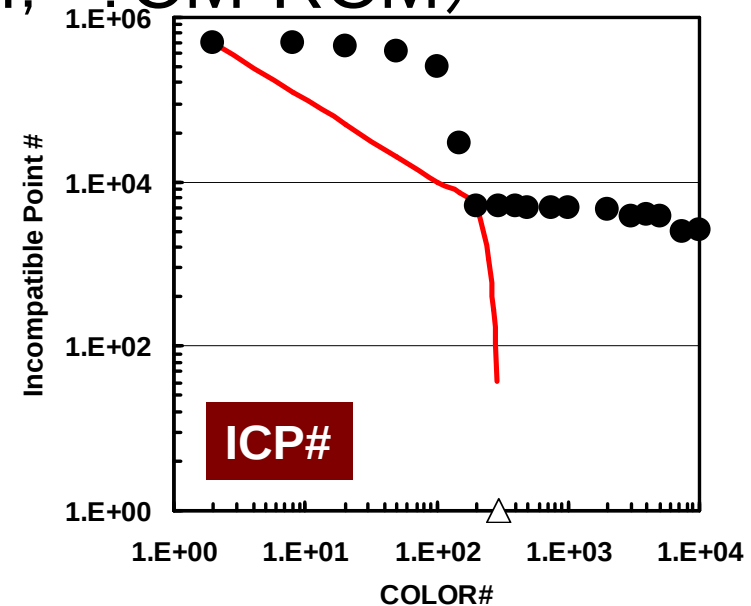
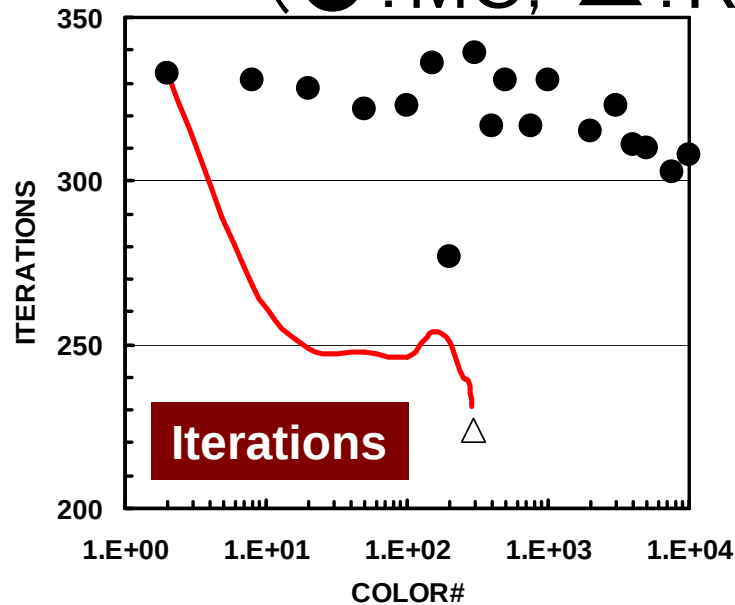
($20^3=8,000$ meshe, $\text{EPSICCG}=10^{-8}$)

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

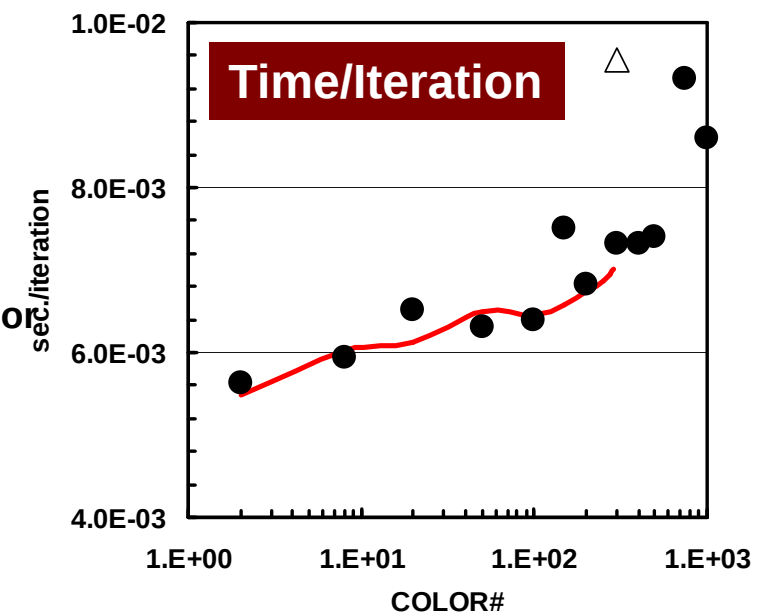
- Remedy for Data Dependency
- Ordering/Reordering
 - Red-Black, Multicoloring (MC)
 - Cuthill-McKee (CM), Reverse-CM (RCM)
 - Reordering and Convergence
- Implementation
- ICCG with Reordering
- **ICCG with Reordering on Multicores**
 - **Just apply OpenMP to L2-sol**

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

(●:MC, △:RCM, - :CM-RCM)

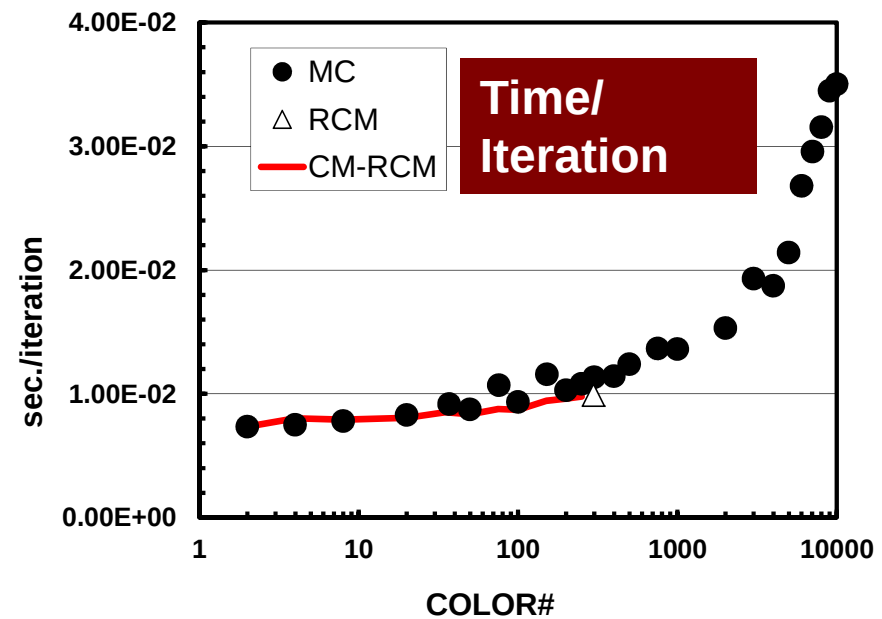
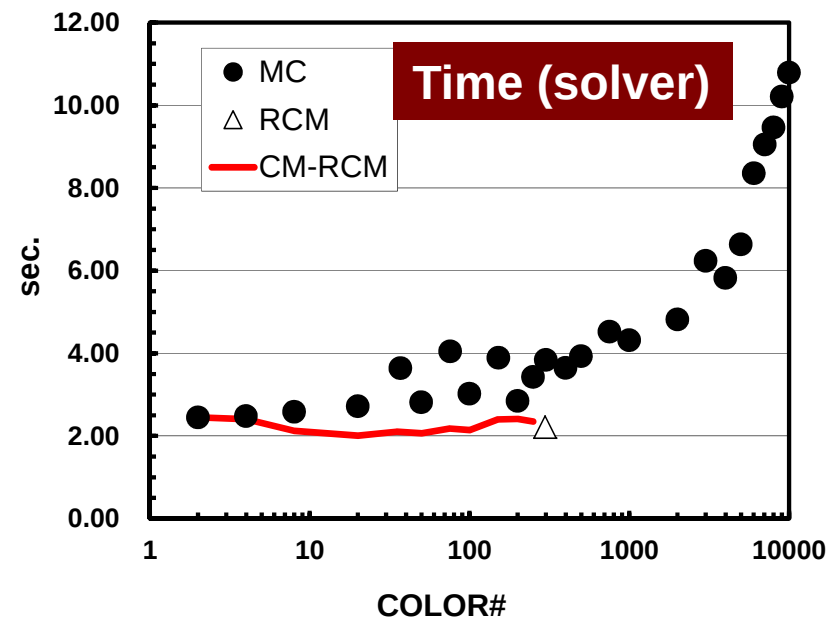
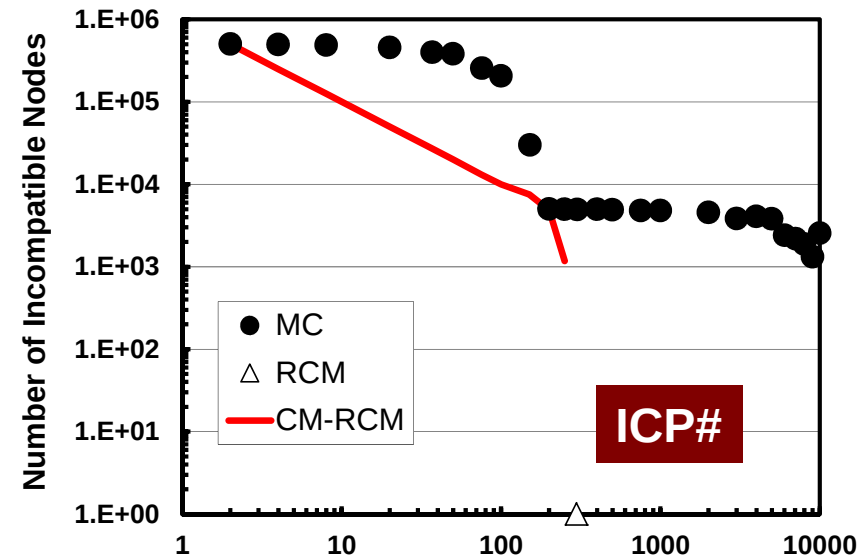
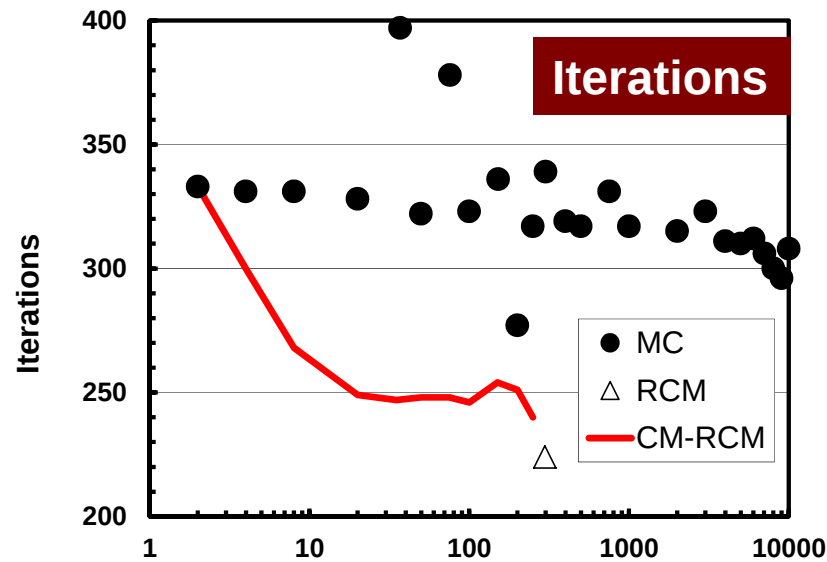


Funny Behavior



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

(●:MC, △:RCM, -:CM-RCM)



Reedbush-U, 1-
node/18-cores,
 128^3
(●: MC, △: RCM, -: CM-RCM)

