# 3D Parallel FEM (IV)
# (OpenMP + MPI) Hybrid Parallel Programming Model

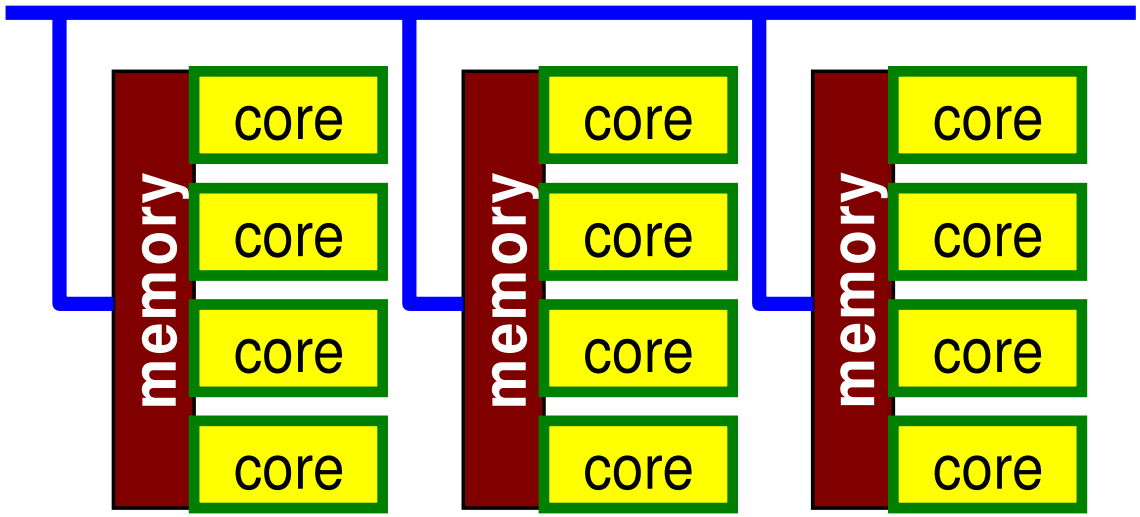
Kengo Nakajima
RIKEN R-CCS

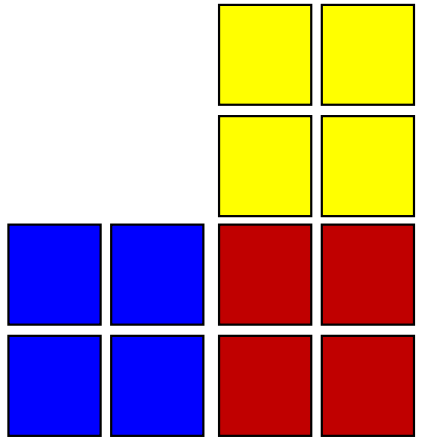
# Hybrid Parallel Programming Model

- Message Passing (e.g. MPI) + Multi Threading (e.g. OpenMP, CUDA, OpenCL, OpenACC etc.)
- Expectations for Hybrid
  - Number of MPI processes (and sub-domains) to be reduced
  - O($10^8$-$10^9$)-way MPI might not scale in Exascale Systems
  - Easily extended to Heterogeneous Architectures
    - CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    - MPI+X: OpenMP, OpenACC, CUDA, OpenCL

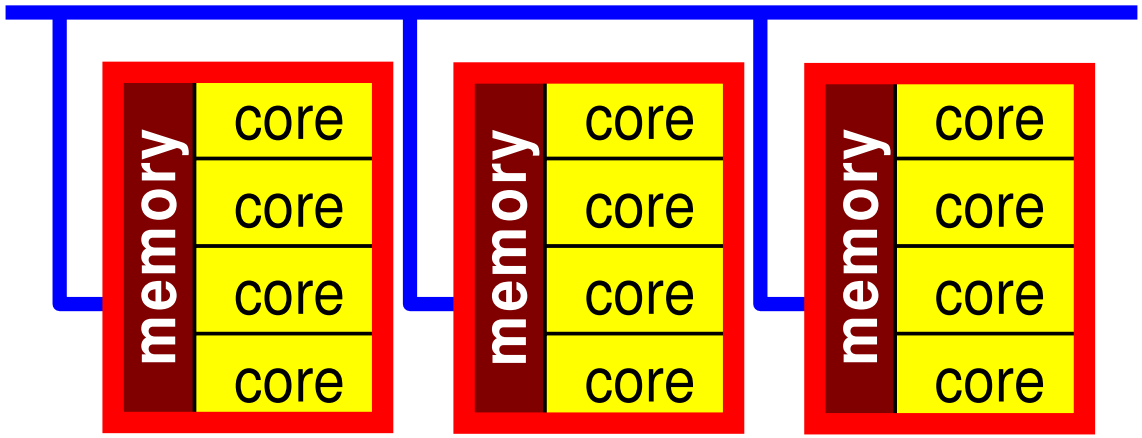
# Flat MPI vs. Hybrid

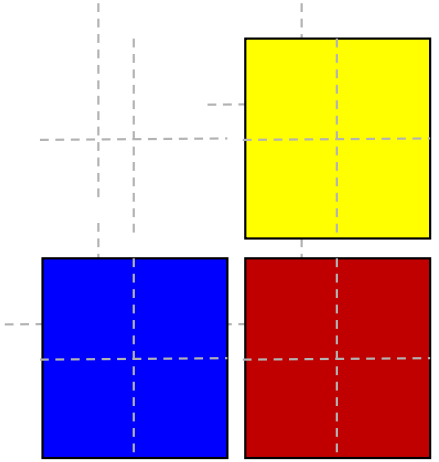
## Flat-MPI：Each Core -> Independent



## Hybrid：Hierarchal Structure

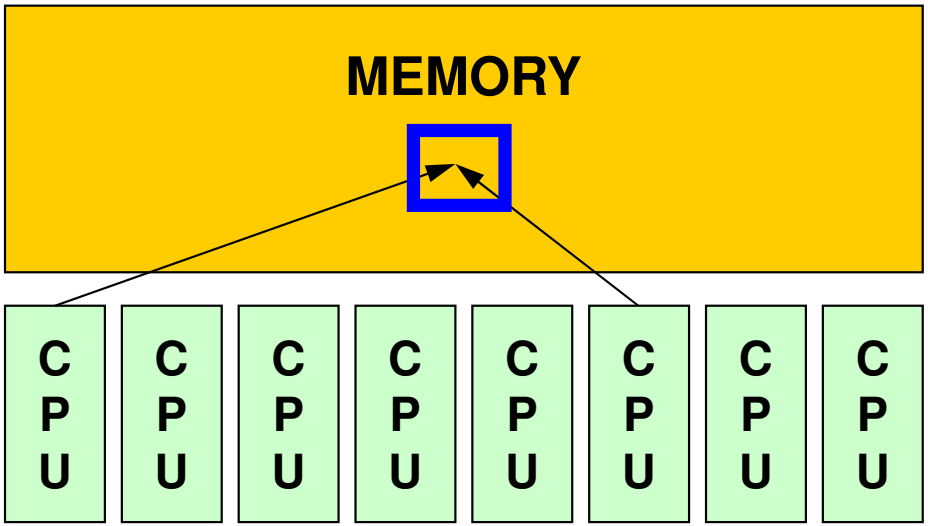


# Background

- Multicore/Manycore Processors
  - Low power consumption, Various types of programming models
- OpenMP
  - Directive based, (seems to be) easy
  - Many books
- Data Dependency
  - Conflict of reading from/writing to memory
  - Appropriate reordering of data is needed for “consistent” parallel computing
  - NO detailed information in OpenMP books: very complicated
    - http://nkl.cc.u-tokyo.ac.jp/21s/
- OpenMP/MPI Hybrid Parallel Programming Model for Multicore/Manycore Clusters

# SMP




- SMP
  - Symmetric Multi Processors
  - Multiple CPU's (cores) share a single memory space

# What is OpenMP ? (1/2)

http://www.openmp.org

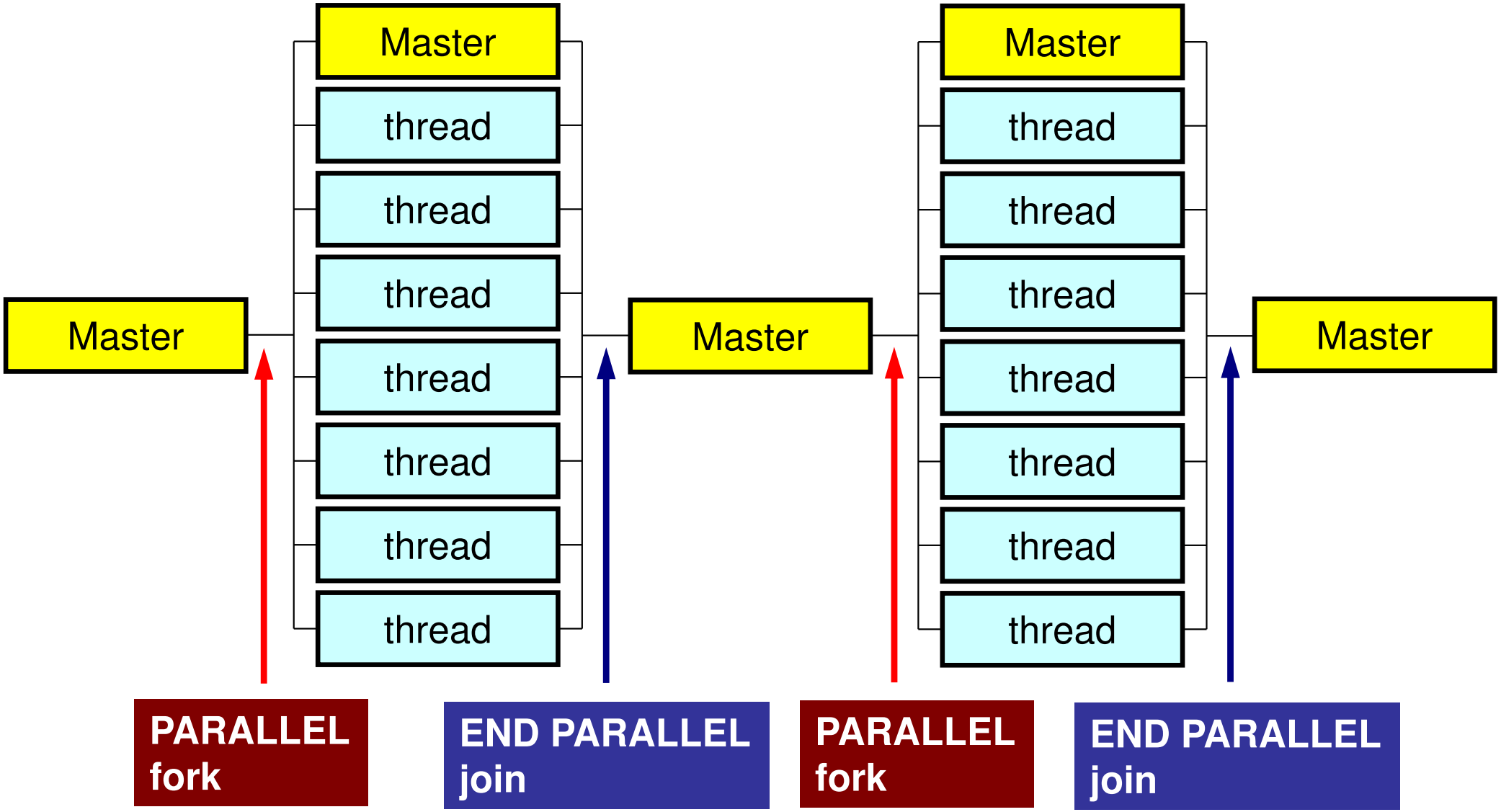
- An API (Application Programming Interface) for multi-platform shared-memory parallel programming in C/C++ and Fortran
  - Current version: 4.X (5.0 is already announced)
    - GPU, Accelerators: close to OpenACC
- Background
  - Merger of Cray and SGI in 1996
    - Separated later, … but both are now merged into HPE
  - ASCI project (US-DOE (Dept. of Energy)) started in 1995
    - Accelerated Strategic Computing Initiative (ASCI) -> Advanced Simulation and Computing Program (ASC)
      - The goal of ASCI is to simulate the results of new weapons designs as well as the effects of aging on existing and new designs, all in the absence of additional data from underground nuclear tests.
    - Development of Supercomputers & Software/Applications
      - SMP Clusters: Intel ASCI Red, IBM Power (Blue, White, Purple)/Blue Gene, SGI
      - Common API for SMP Clusters needed

# What is OpenMP ? (2/2)

http://www.openmp.org

- C/C++ version and Fortran version have been separately developed until ver.2.5.
- Fork-Join Parallel Execution Model (Next Page)
  - Directives: Parallel, End Parallel
  - Serial Execution: Master Thread
  - Parallel Execution: Master Thread/Thread Team
- Users have to specify everything by directives.
  - Nothing happen, if there are no directives

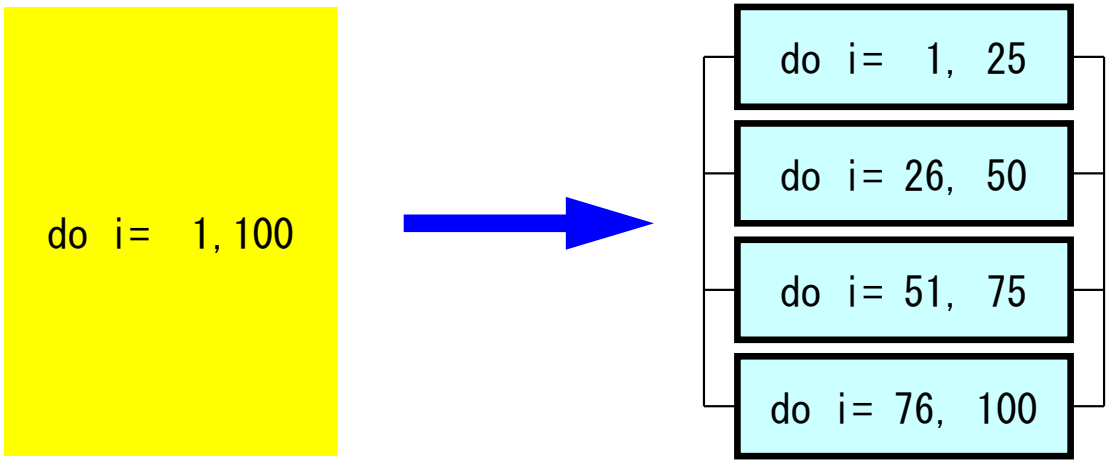
# Fork-Join Parallel Execution Model



# Number of Threads

- **OMP_NUM_THREADS**
  - How to change ?
    - bash(.bashrc) **export OMP_NUM_THREADS=8**
    - csh(.cshrc) **setenv OMP_NUM_THREADS 8**

- **OMP_NUM_THREADS=4**



# Information about OpenMP

- OpenMP Architecture Review Board (ARB)
  - http://www.openmp.org
- References
  - Chandra, R. et al.「Parallel Programming in OpenMP」（Morgan Kaufmann）
  - Quinn, M.J.「Parallel Programming in C with MPI and OpenMP」（McGrawHill）
  - Mattson, T.G. et al.「Patterns for Parallel Programming」（Addison Wesley）
  - 牛島「OpenMPによる並列プログラミングと数値計算法」（丸善）
  - Chapman, B. et al.「Using OpenMP」（MIT Press）
- Japanese Version of OpenMP 3.0 Spec. (Fujitsu etc.)
  - http://www.openmp.org/mp-documents/OpenMP30spec-ja.pdf

# Features of OpenMP

- Directives
  - Loops right after the directives are parallelized.
  - If the compiler does not support OpenMP, directives are considered as just comments.

# OpenMP/Directives Array Operations

## Simple Substitution

```
!$omp parallel do
      do i= 1, N
        W(i,1)= 0.d0
        W(i,2)= 0.d0
      enddo
!$omp end parallel do
```

## Dot Products

```
!$omp parallel do private(i)
!$omp&             reduction(+:RHO)
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo
!$omp end parallel do
```

## DAXPY

```
!$omp parallel do
      do i= 1, N
        Y(i)= ALPHA*X(i) + Y(i)
      enddo
!$omp end parallel do
```

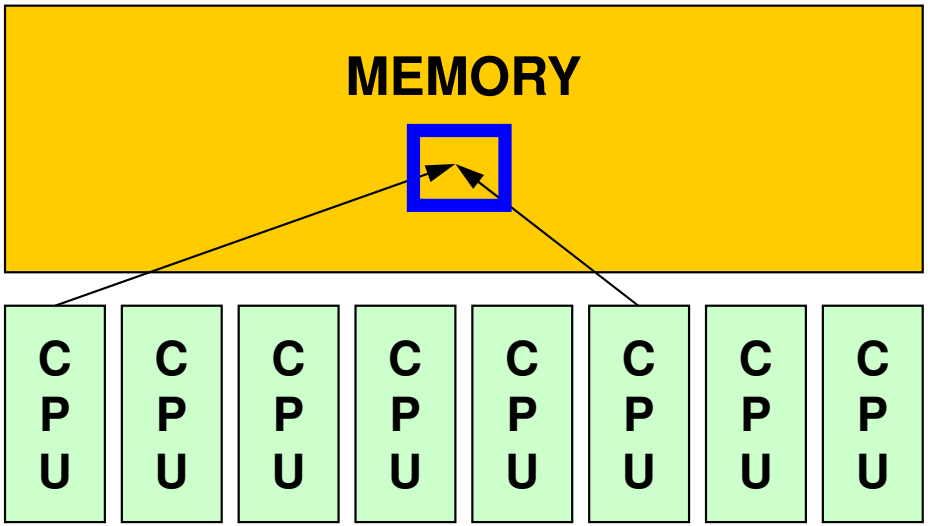
# OpenMP/Direceives Matrix/Vector Products

```
!$omp parallel do private(i,VAL,k)
  do i = 1, N
    VAL= D(i)*W(i,P)
    do k= indexLU(i-1)+1, indexLU(i)
      VAL= VAL + AMAT(k)*W(itemLU(k),P)
    enddo
    W(i,Q)= VAL
  enddo
!$omp end parallel do
```

# Features of OpenMP

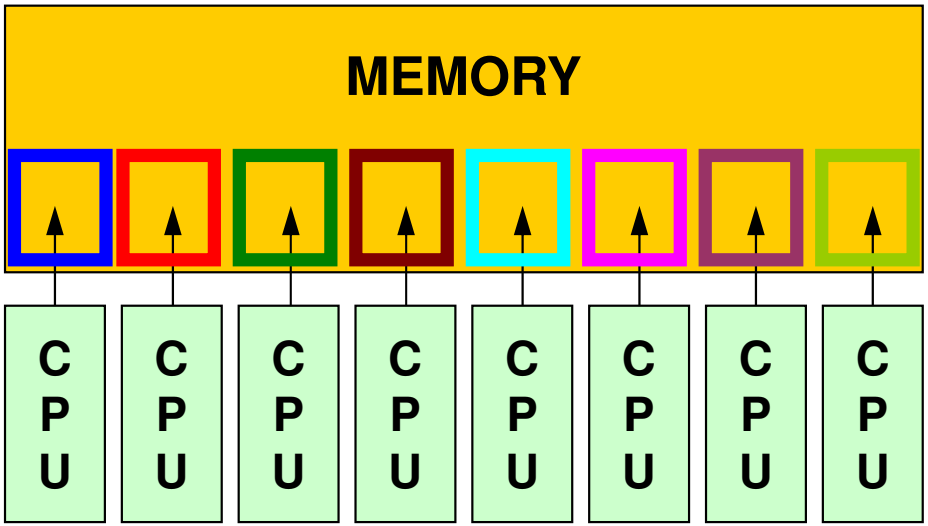
- Directives
  - Loops right after the directives are parallelized.
  - If the compiler does not support OpenMP, directives are considered as just comments.
- Nothing happen without explicit directives
  - Different from “automatic parallelization/vectorization”
  - Something wrong may happen by un-proper way of usage
  - Data configuration, ordering etc. are done under users’ responsibility
- “Threads” are created according to the number of cores on the node
  - Thread: “Process” in MPI
  - Generally, “# threads = # cores”: Xeon Phi supports 4 threads per core (Hyper Multithreading)

# Memory Contention: メモリ競合




- During a complicated process, multiple threads may simultaneously try to update the data in same address on the memory.
  - e.g.: Multiple cores update a single component of an array.
  - This situation is possible.
  - Answers may change compared to serial cases with a single core (thread).

# Memory Contention (cont.)




- In this lecture, any such case does not happen by reordering etc.
  - In OpenMP, users are responsible for such issues (e.g. proper data configuration, reordering etc.)
- Data Dependency
- Performance per core reduces as number of used cores (thread #) increases (Memory Saturation)

# Features of OpenMP (cont.)

- “do” loops with “!$omp parallel do”
- Global (Shared) Variables, Private Variables
  - Default: Global (Shared)
  - Dot Products: reduction

```
!$omp parallel do private(i)
!$omp&                reduction(+:RHO)
      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo
!$omp end parallel do
```

W(:,:), R, Z
global (shared)

# FORTRAN & C

```
use omp_lib
...
!$omp parallel do default(none) shared(n,x,y) private(i)
      do i= 1, n
        x(i)= x(i) + y(i)
      enddo
!$ omp end parallel do (not needed)
```

```
#include <omp.h>
...
#pragma omp parallel for default(none) shared(n,x,y) private(i)
for (i=0; i<n; i++){
        x[i] += y[i];
}
```

# In this class ...

- There are many capabilities of OpenMP.
- In this class, only several functions are shown for parallelization of parallel FEM.

# First things to be done (after OpenMP 3.0)

- use omp_lib Fortran
- #include <omp.h> C

# OpenMP Directives (Fortran)

```
sentinel directive_name [clause[[,] clause]…]
```

- NO distinctions between upper and lower cases.
- sentinel
  - Fortran: !$OMP, C$OMP, *$OMP
    - !$OMP only for free format
  - Continuation Lines (Same rule as that of Fortran compiler is applied)
    - Example for **`!$OMP PARALLEL DO SHARED(A,B,C)`**

```
!$OMP PARALLEL DO
!$OMP+SHARED (A,B,C)
```

```
!$OMP PARALLEL DO &
!$OMP SHARED (A,B,C)
```

# OpenMP Directives (C)

```
#pragma omp directive_name [clause[[,] clause]…]
```

- “＼” for continuation lines
- Only lower case (except names of variables)

```
#pragma omp parallel for shared (a,b,c)
```

# PARALLEL DO/for

```
!$OMP PARALLEL DO[clause[[,] clause] … ]
   (do_loop)
!$OMP END PARALLEL DO
```

```
#pragma omp parallel for [clause[[,] clause] … ]
   (for_loop)
```

- Parallerize DO/for Loops
- Examples of “clause”
  - PRIVATE(list)
  - SHARED(list)
  - DEFAULT(PRIVATE|SHARED|NONE)
  - REDUCTION({operation|intrinsic}: list)

# REDUCTION

```
REDUCTION ({operator|instinsic}: list)
```

```
reduction ({operator|instinsic}: list)
```

- Similar to “MPI_Reduce”
- Operator
  - +, *, -, .AND., .OR., .EQV., .NEQV.
- Intrinsic
  - MAX, MIN, IAND, IOR, IEQR

# Example-1: A Simple Loop

```
!$OMP PARALLEL DO PRIVATE (i)
      do i= 1, N
        B(i)= (A(i) + B(i)) * 0.50
      enddo
!$OMP END PARALLEL DO
```

- Default status of loop variables (“i” in this case) is private. Therefore, explicit declaration is not needed.
  - “PRIVATE (i)” can be optional, but it is recommended to put it explicitly
- “END PARALLEL DO” is not required
  - In C, there are no definitions of “end parallel do”

# Example-2: REDUCTION

```
!$OMP PARALLEL DO DEFAULT(PRIVATE) REDUCTION(+:A,B)
      do i= 1, N
        call WORK (Alocal, Blocal)
        A= A + Alocal
        B= B + Blocal
      enddo
!$OMP END PARALLEL DO
```

- “END PARALLEL DO” is not required

# Functions which can be used with OpenMP

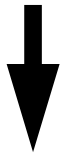
| Name | Functions |
|---|---|
| int omp_get_num_threads (void) | Total Thread # |
| int omp_get_thread_num (void) | Thread ID |
| double omp_get_wtime (void) | = MPI_Wtime |
| void omp_set_num_threads (int num_threads)<br>call omp_set_num_threads (num_threads) | Setting Thread # |

# OpenMP for Dot Products

```
      VAL= 0.d0
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
```

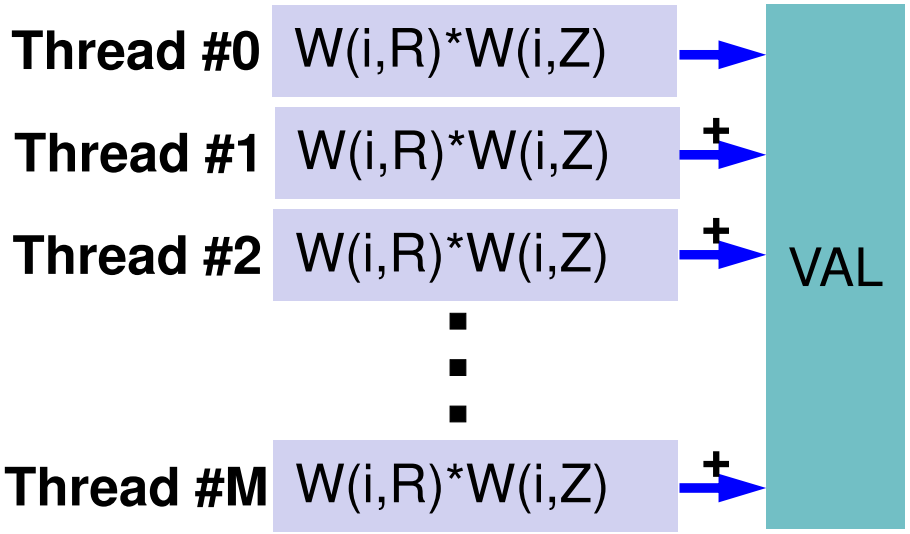
# OpenMP for Dot Products

```
      VAL= 0.d0
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
```



```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(i) REDUCTION(+:VAL)
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
!$OMP END PARALLEL DO
```

Directives are just inserted.



# OpenMP for Dot Products

```
      VAL= 0.d0
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
```

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(i) REDUCTION(+:VAL)
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
!$OMP END PARALLEL DO
```

Directives are just inserted.

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(ip,i) REDUCTION(+:VAL)
      do ip= 1, PEsmpTOT
        do i= index(ip-1)+1, index(ip)
          VAL= VAL + W(i,R) * W(i,Z)
        enddo
      enddo
!$OMP END PARALLEL DO
```

Multiple Loop
**`PEsmpTOT`**: Number of threads

Additional array **`INDEX(:)`** is needed.
Efficiency is not necessarily good, but users can specify thread for each component of data.

# OpenMP for Dot Products

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(ip,i) REDUCTION(+:VAL)
      do ip= 1, PEsmpTOT
        do i= index(ip-1)+1, index(ip)
          VAL= VAL + W(i,R) * W(i,Z)
        enddo
      enddo
!$OMP END PARALLEL DO
```

Multiple Loop
**PEsmpTOT**: Number of threads

Additional array **INDEX(:)** is needed.
Efficiency is not necessarily good, but users can specify thread for each component of data.

e.g.: N=100, PEsmpTOT=4

INDEX(0)= 0
INDEX(1)= 25
INDEX(2)= 50
INDEX(3)= 75
INDEX(4)= 100

NOT good for GPU's

# Matrix-Vector Multiply

```
do i = 1, N
  VAL= D(i)*W(i,P)
  do k= indexLU(i-1)+1, indexLU(i)
    VAL= VAL + AMAT(k)*W(itemLU(k),P)
  enddo
  W(i,Q)= VAL
enddo
```

# Matrix-Vector Multiply

## "!$omp end parallel do" is not essential

```
!$omp parallel do private(ip,i,VAL,k)
      do ip= 1, PEsmpTOT
        do i = INDEX(ip-1)+1, INDEX(ip)
          VAL= D(i)*W(i,P)
          do k= indexLU(i-1)+1, indexLU(i)
            VAL= VAL + AMAT(k)*W(itemLU(k),P)
          enddo
          W(i,Q)= VAL          enddo
      enddo
!$omp end parallel do
```

# Matrix-Vector Multiply: Other Approach

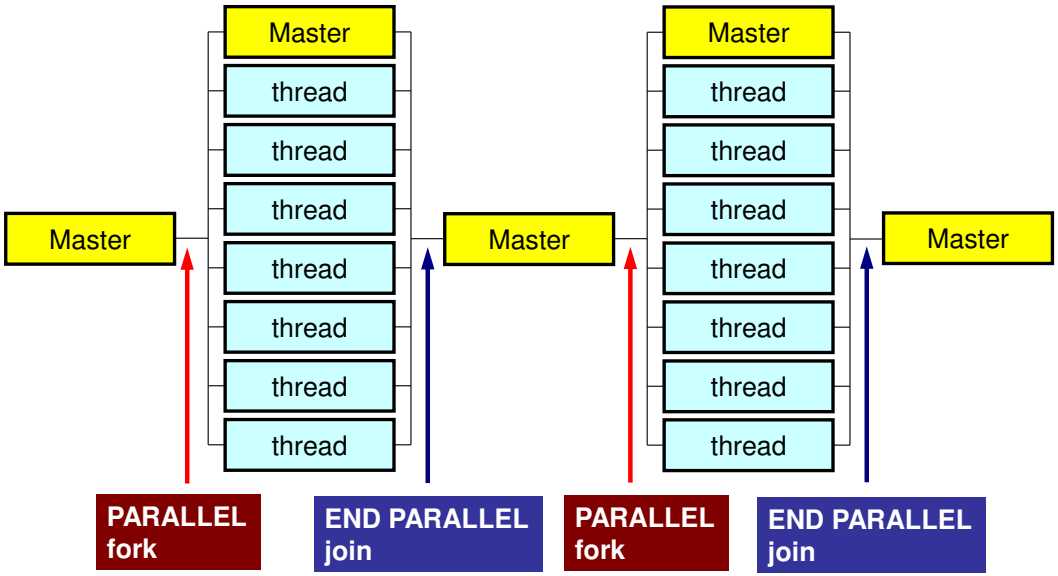
This is rather better for GPU and (very) many-core architectures: simpler structure of loops

"!$omp end parallel do" is not essential

```
!$omp parallel do private(i,VAL,k)
  do i = 1, N
    VAL= D(i)*W(i,P)
    do k= indexLU(i-1)+1, indexLU(i)
      VAL= VAL + AMAT(k)*W(itemLU(k),P)
    enddo
    W(i,Q)= VAL
  enddo
!$omp end parallel do
```

# omp parallel (do)

- “omp parallel-omp end parallel” = “fork-join”
- If you have many loops, these “fork-join’s” cause operations
- omp parallel + omp do/omp for




```
#pragma omp parallel ...

#pragma omp for {

...
#pragma omp for {
```

```
!$omp parallel ...

!$omp do
   do i= 1, N
...
!$omp do
   do i= 1, N
...
!$omp end parallel   required
```

# Exercise !!

- Apply multi-threading by OpenMP on parallel FEM code using MPI
  - CG Solver (solver_CG, solver_SR)
  - Matrix Assembling (mat_ass_main, mat_ass_bc)
- Hybrid parallel programming model

- Evaluate the effects of
  - Problem size, parallel programming model, thread #

# OpenMP（Only Solver）（F・C）

```
>$ cd /home/ra020019/<Your-UID>/pFEM/pfem3d/src1
>$ make
>$ cd ../run
>$ ls sol1
      sol1

>$ cd ../pmesh

<Parallel Mesh Generation>

>$ cd ../run

<modify bXX.sh>

>$ pjsub bXX.sh
```

# Makefile(Fortran)

```
F90       =  mpifrtpx
F90LINKER = $(F90)
LIB_DIR   =
INC_DIR   =

OPTFLAGS  = -Kfast,openmp

FFLAGS = $(OPTFLAGS)
FLIBS =
F90LFLAGS=
#
TARGET = ../run/sol1
default: $(TARGET)
OBJS =¥
pfem_util.o ¥
solver_SR.o solver_CG.o ¥
solver11.o test1.o util.o pfem_init.o input_cntl.o input_grid.o
define_file_name.o¥
mat_con0.o mat_con1.o mat_ass_main.o mat_ass_bc.o pfem_finalize.o
output_ucd.o
```

# How to apply multi-threading

- CG Solver
  - Just insert OpenMP directives
  - ILU/IC preconditioning is much more difficult
- MAT_ASS (mat_ass_main, mat_ass_bc)
  - Data Dependency
  - Avoid to accumulate contributions of multiple elements to a single node simultaneously (in parallel)
    - results may be changed
    - deadlock may occur
  - Coloring
    - Elements in a same color do not share a node
    - Parallel operations are possible for elements in each color
    - In this case, we need only 8 colors for 3D problems (4 colors for 2D problems)
    - Coloring part is very expensive: parallelization is difficult

# FORTRAN(solver_CG)

```
!$omp parallel do private(i)
      do i= 1, N
         X(i)  = X (i)    + ALPHA * WW(i,P)
        WW(i,R)= WW(i,R) - ALPHA * WW(i,Q)
      enddo

      DNRM20= 0.d0
!$omp parallel do private(i) reduction (+:DNRM20)
      do i= 1, N
        DNRM20= DNRM20 + WW(i,R)**2
      enddo

!$omp parallel do private(j,k,i,WVAL)
      do j= 1, N
        WVAL= D(j)*WW(j,P)
        do k= index(j-1)+1, index(j)
           i= item(k)
          WVAL= WVAL + AMAT(k)*WW(i,P)
        enddo
        WW(j,Q)= WVAL
      enddo
```

# solver_SR (send)

```
      do neib= 1, NEIBPETOT
        istart= EXPORT_INDEX(neib-1)
        inum  = EXPORT_INDEX(neib  ) - istart
!$omp parallel do private(k,ii)
        do k= istart+1, istart+inum
           ii   = EXPORT_ITEM(k)
          WS(k)= X(ii)
        enddo

        call MPI_Isend (WS(istart+1), inum, MPI_DOUBLE_PRECISION,        &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib),    &
     &                  ierr)
      enddo
```

# pmesh: 8-nodes, 384-cores

## Flat MPI: 384 processes

**mesh.inp**

```
256 256 192
  8   8   6
pcube
```

**mg.sh**

```
#!/bin/sh
#PJM -N "pmg"

#PJM -L "rscgrp=small"
#PJM -L "node=8:torus"
#PJM --mpi "max-proc-per-node=32"
#PJM -L "elapse=00:15:00"
#PJM -g ra020019
#PJM -s
#PJM -e err
#PJM -o pmg.lst

mpiexec ./pmesh

rm wk.*
```

## HB 12x4: 32 processes

**mesh.inp**

```
256 256 192
  4   4   2
pcube
```

**mg.sh**

```
#!/bin/sh
#PJM -N "pmg"

#PJM -L "rscgrp=small"
#PJM -L "node=8:torus"
#PJM --mpi "max-proc-per-node=4"
#PJM -L "elapse=00:15:00"
#PJM -g ra020019
#PJM -s
#PJM -e err
#PJM -o pmg.lst

mpiexec ./pmesh

rm wk.*
```

# pFEM: 8-nodes, 384-cores

## Flat MPI: 384 processes

**a08.sh**

```
#!/bin/sh
#PJM -N "pmg"

#PJM -L "rscgrp=small"
#PJM -L "node=8:torus"
#PJM --mpi "max-proc-per-node=32"
#PJM -L "elapse=00:15:00"
#PJM -g ra020019
#PJM -s
#PJM -e err
#PJM -o a08.lst

mpiexec ./pmesh

mpiexec ./sol
mpiexec numactl -l ./sol
```

# pFEM: 8-nodes, 384-cores

## HB 12x4: 32 processes

**b08.sh**

```
#!/bin/sh
#PJM -N "hb-12"
#PJM -L “rscgrp=small”
#PJM -L “node=8:torus”
#PJM --mpi “max-proc-per-node=4
#PJM -L elapse=00:15:00
#PJM -g ra020019
#PJM -j
#PJM -e err
#PJM -o b08.lst

export OMP_NUM_THREADS=12
mpiexec ./sol1
mpiexec numactl -l ./sol1

export XOS_MMM_L_PAGING_POLICY=demand:demand:demand
mpiexec ./sol1
mpiexec numactl -l ./sol1
```

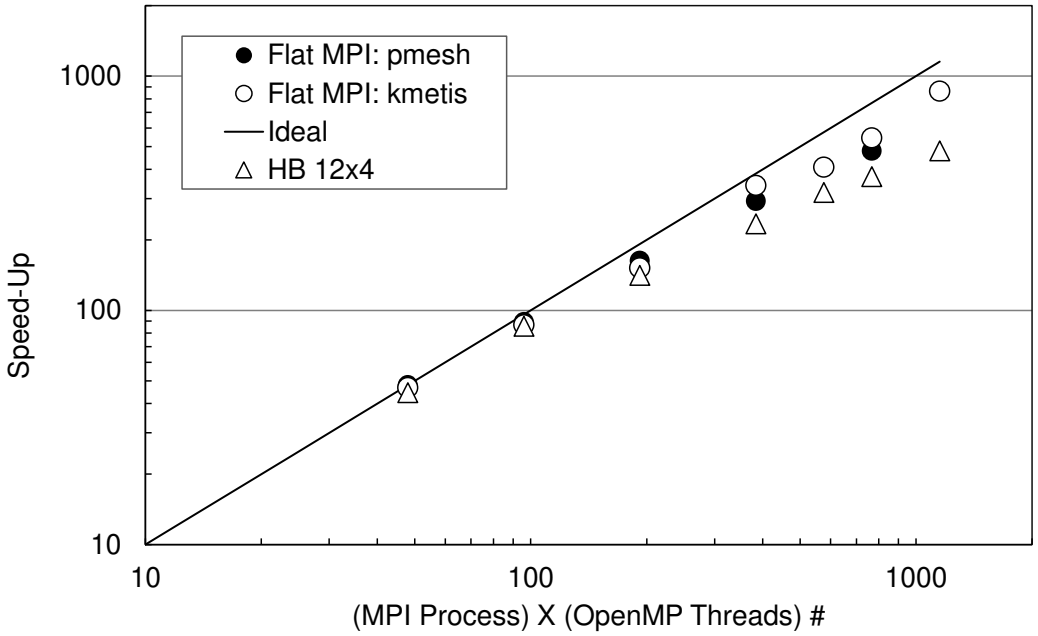
# b08.sh

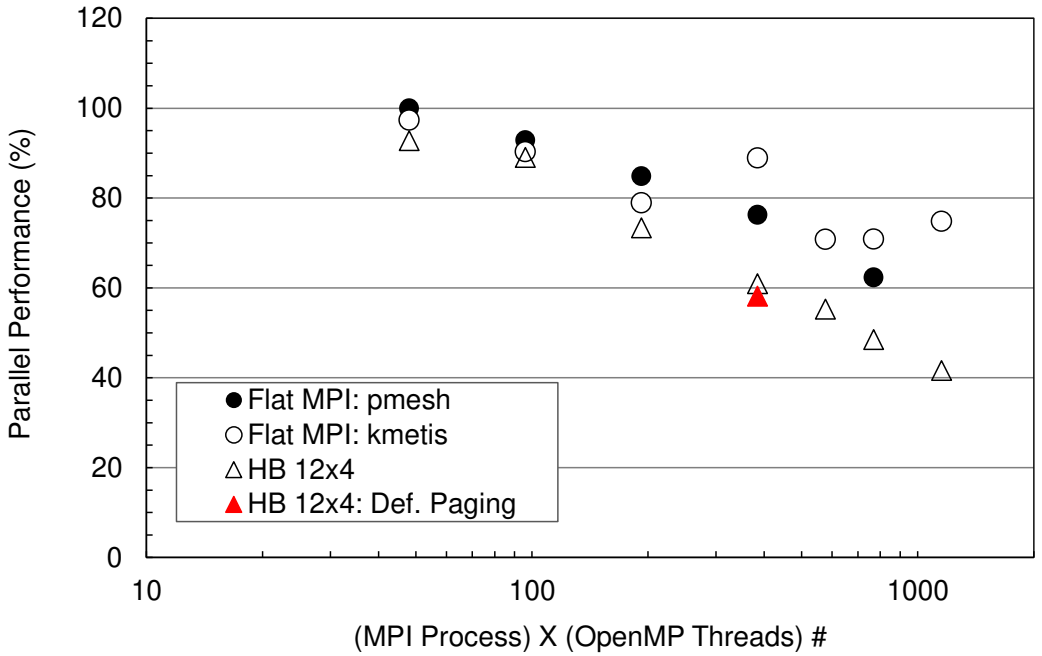
```
export XOS_MMM_L_PAGING_POLICY=
demand:demand:demand
```

| Parameters | Values (Underline: Default) | Description |
|---|---|---|
| XOS_MMM_L_PAGING_POLICY | [demand \| prepage]<br>[demand \| prepage]<br>[demand \| prepage] | Paging policy (page allocation trigger) of each memory unit<br>✓ demand: Demand Paging Method<br>✓ prepage: Prepaging Method<br><br>3 Items are defined<br>✓ 1st Item: .bss area of static data (.data area of static data is always "prepage")<br>✓ 2nd Item: Stack Area, Thread Stack Area<br>✓ 3rd Item: Area for Dynamic Memory Allocation<br><br>If a value other than the specified value (demand/prepage), the configuration is considered as "prepage:demand:prepage"<br><br>**"demand:demand:demand" is recommended for using multiple CMG's** |

# Example: Strong Scaling: C

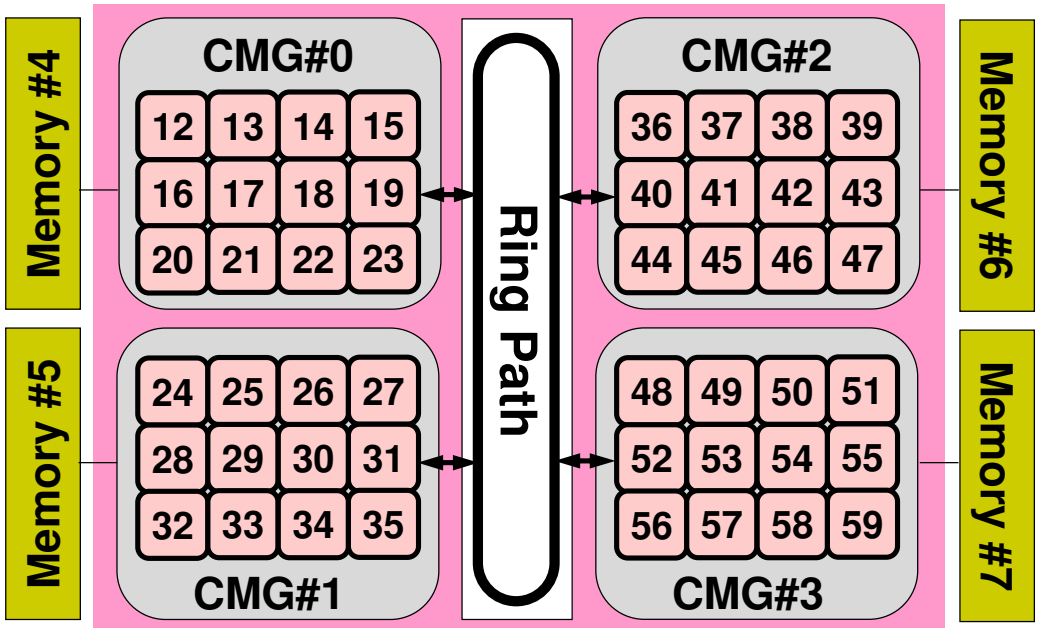
- 256×256×192 nodes, 12,582,912 DOF
- 48～1,152 cores (1~24 nodes)
- Linear Solver



# Flat MPI vs. Hybrid

- Depends on applications, problem size, HW  etc.
- Flat MPI is generally better for sparse linear solvers, if number of computing nodes is not so large.
  - Memory contention
- Hybrid becomes better, if node.# is larger.
  - Fewer number of MPI processes.
- 1 MPI Process/Node is possible
  - NUMA-aware, First-Touch



# mesh.inp

## Flat MPI

**1-node**
```
256 256 192
  4   4   3
pcube
```

**12-nodes**
```
MeTiS
```

**2-nodes**
```
256 256 192
  8   4   3
pcube
```

**16-nodes**
```
256 256 192
  8   8  12
pcube
```

**4-nodes**
```
256 256 192
  8   8   3
pcube
```

**24-nodes**
```
MeTiS
```

**8-nodes**
```
256 256 192
  8   8   6
pcube
```

## HB 12x4

**1-node**
```
256 256 192
  2   2   1
pcube
```

**12-nodes**
```
256 256 192
  4   4   3
pcube
```

**2-nodes**
```
256 256 192
  2   2   2
pcube
```

**16-nodes**
```
256 256 192
  4   4   4
pcube
```

**4-nodes**
```
256 256 192
  4   2   2
pcube
```

**24-nodes**
```
256 256 192
  4   4   6
pcube
```

**8-nodes**
```
256 256 192
  4   4   2
pcube
```

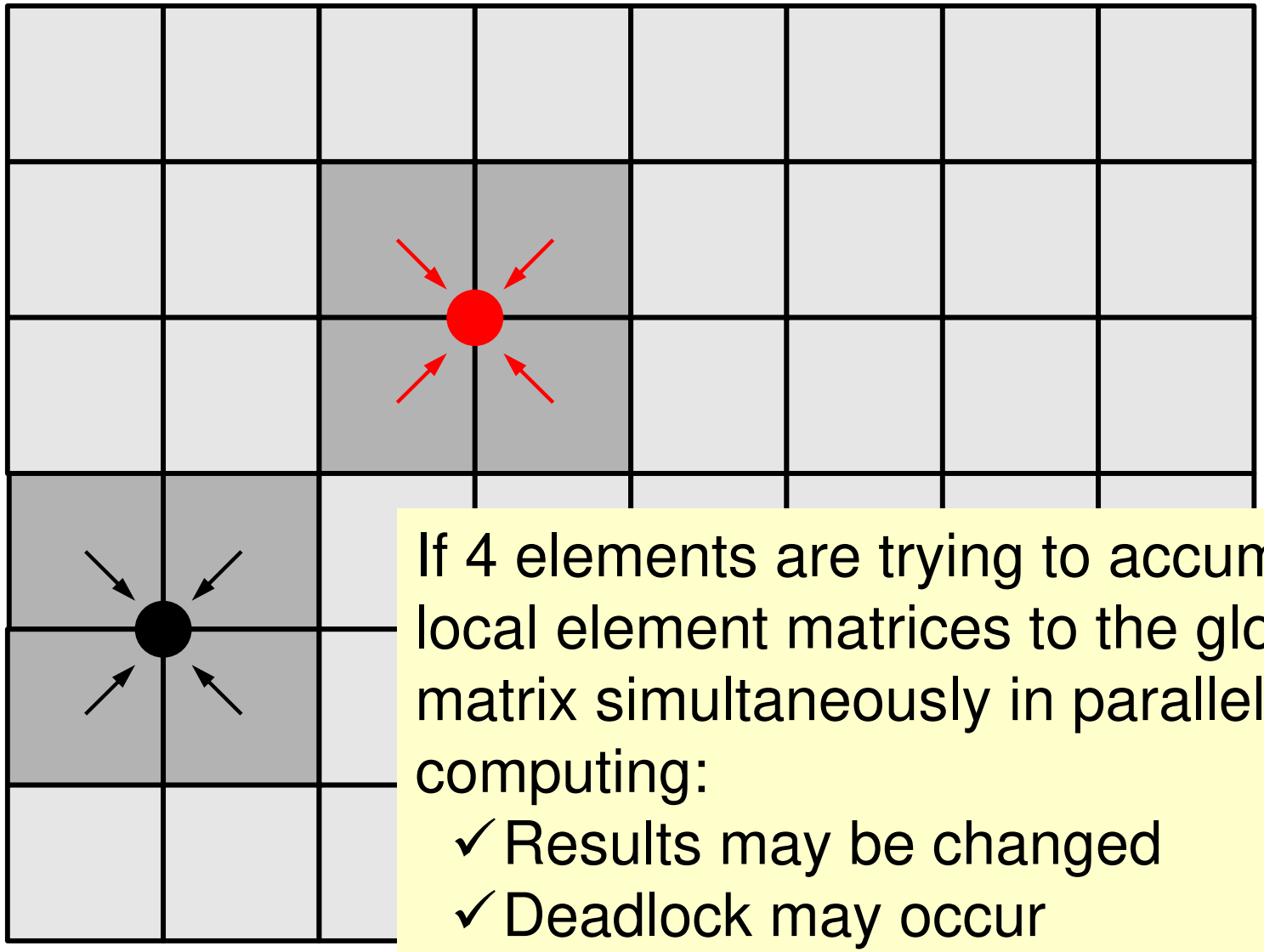
**24-nodes**
```
256 256 192
  8   4   3
pcube
```

# How to apply multi-threading

- CG Solver
  - Just insert OpenMP directives
  - ILU/IC preconditioning is much more difficult
- MAT_ASS (mat_ass_main, mat_ass_bc)
  - Data Dependency
  - Each Node is shared by 4/8-Elements (in 2D/3D)
  - If 4 elements are trying to accumulate local element matrices to the global matrix simultaneously in parallel computing:
    - Results may be changed
    - Deadlock may occur

# Mat_Ass: Data Dependency

Each Node is shared by 4-Elements in 2D



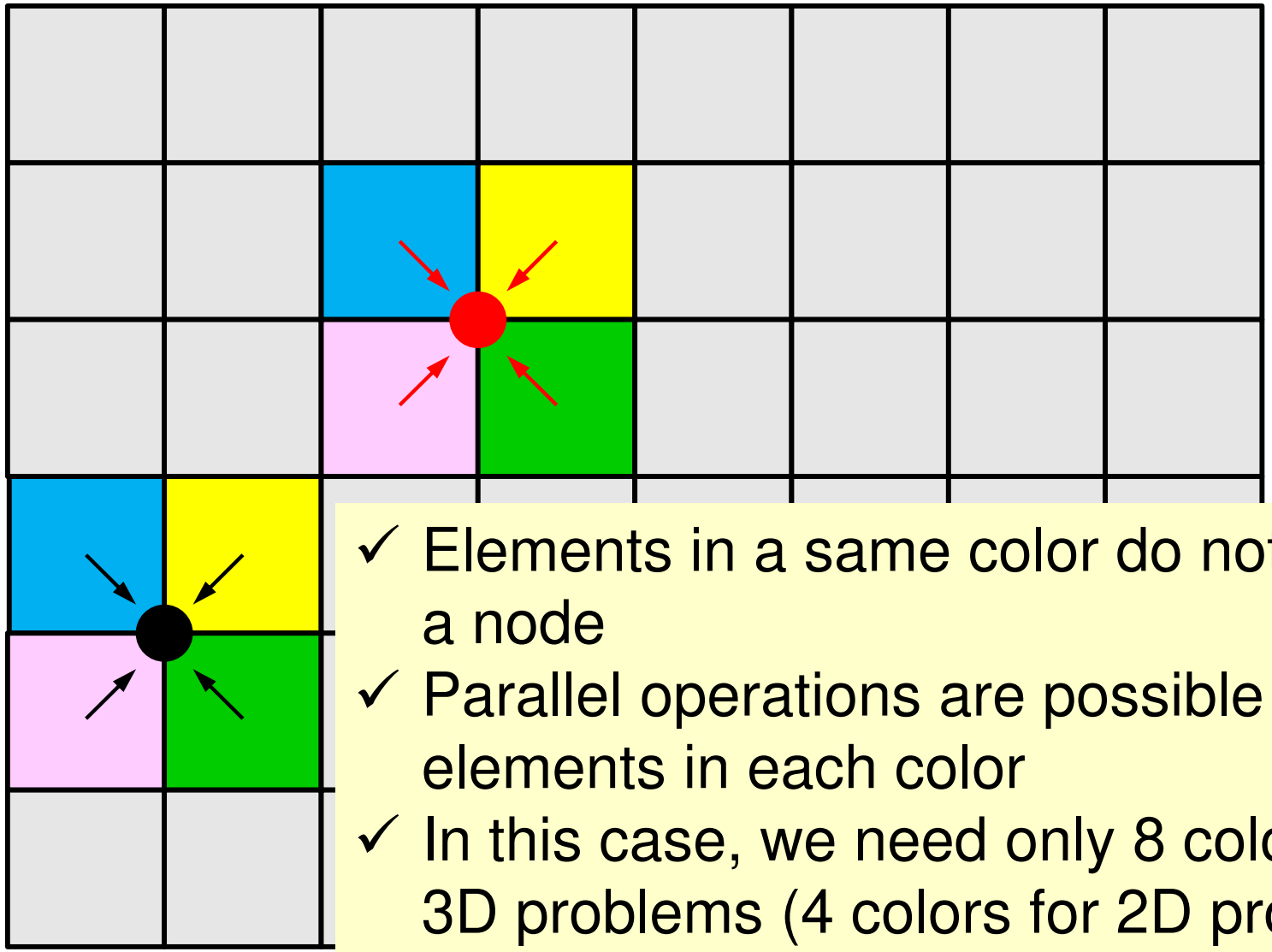
If 4 elements are trying to accumulate local element matrices to the global matrix simultaneously in parallel computing:

- ✓Results may be changed
- ✓Deadlock may occur

# How to apply multi-threading

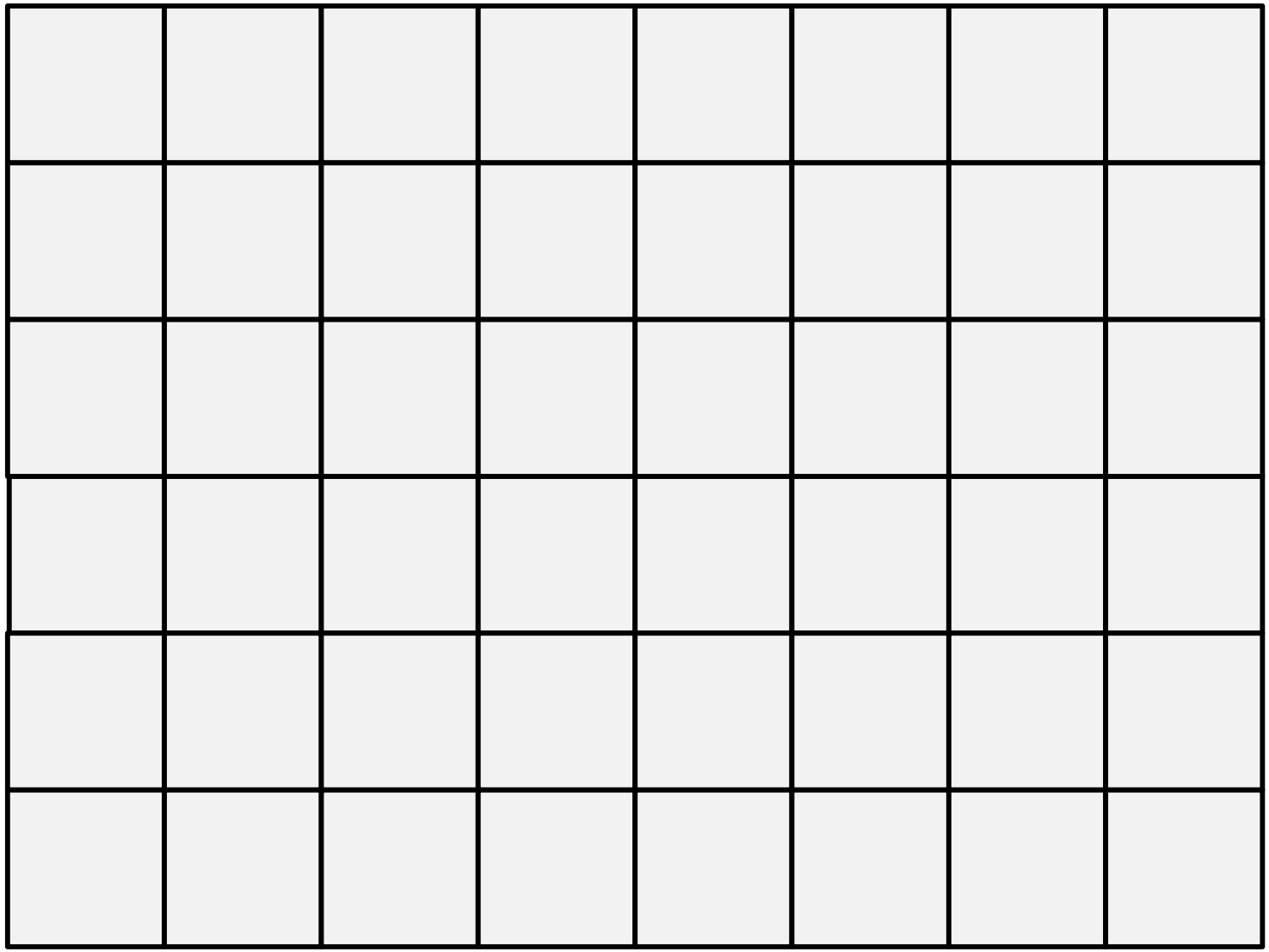
- CG Solver
  - Just insert OpenMP directives
  - ILU/IC preconditioning is much more difficult
- MAT_ASS (mat_ass_main, mat_ass_bc)
  - Coloring
    - Elements in a same color do not share a node
    - Parallel operations are possible for elements in each color
    - In this case, we need only 8 colors for 3D problems (4 colors for 2D problems)

# Mat_Ass: Data Dependency



- ✓ Elements in a same color do not share a node
- ✓ Parallel operations are possible for elements in each color
- ✓ In this case, we need only 8 colors for 3D problems (4 colors for 2D problems)

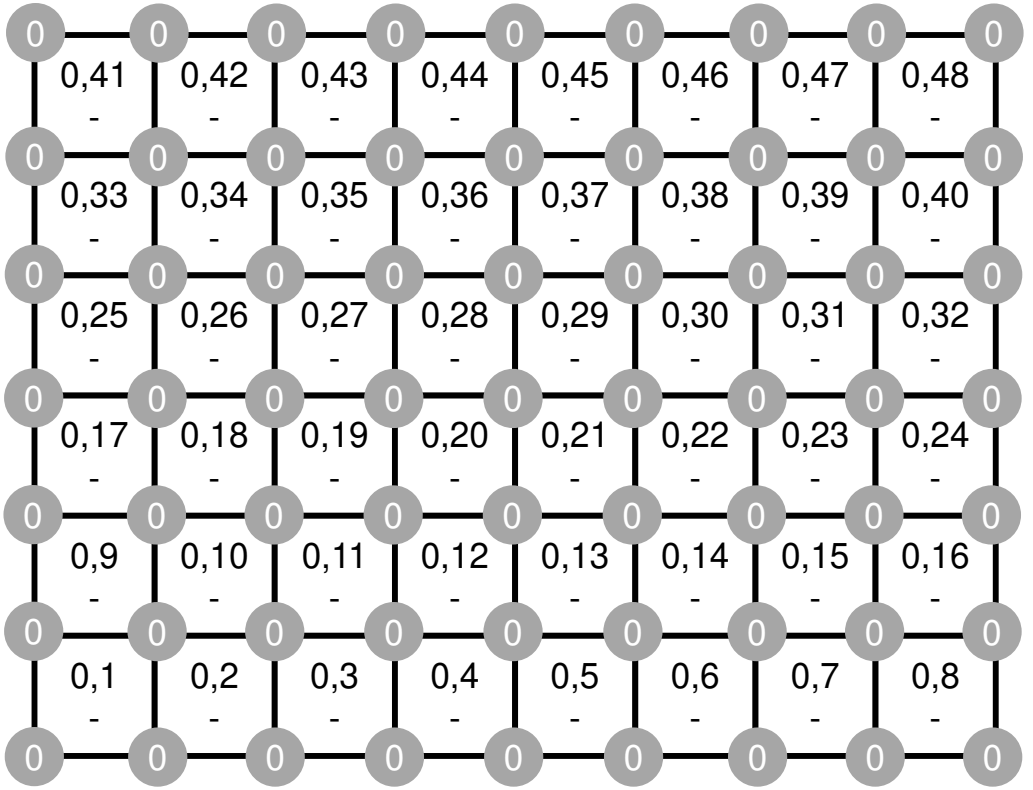
# Target: 8x6= 48-meshes



# Coloring (2D) (1/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1)
            ip2= W1(in2)
            ip3= W1(in3)
            ip4= W1(in4)

            isum= ip1 + ip2 + ip3 + ip4
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

100   continue
      ELMCOLORtot= icol
      W3(0           )= 0
      W3(ELMCOLORtot)= ICELTOT

      do icol= 0, ELMCOLORtot
        ELMCOLORindex(icol)= W3(icol)
      enddo
```

| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |



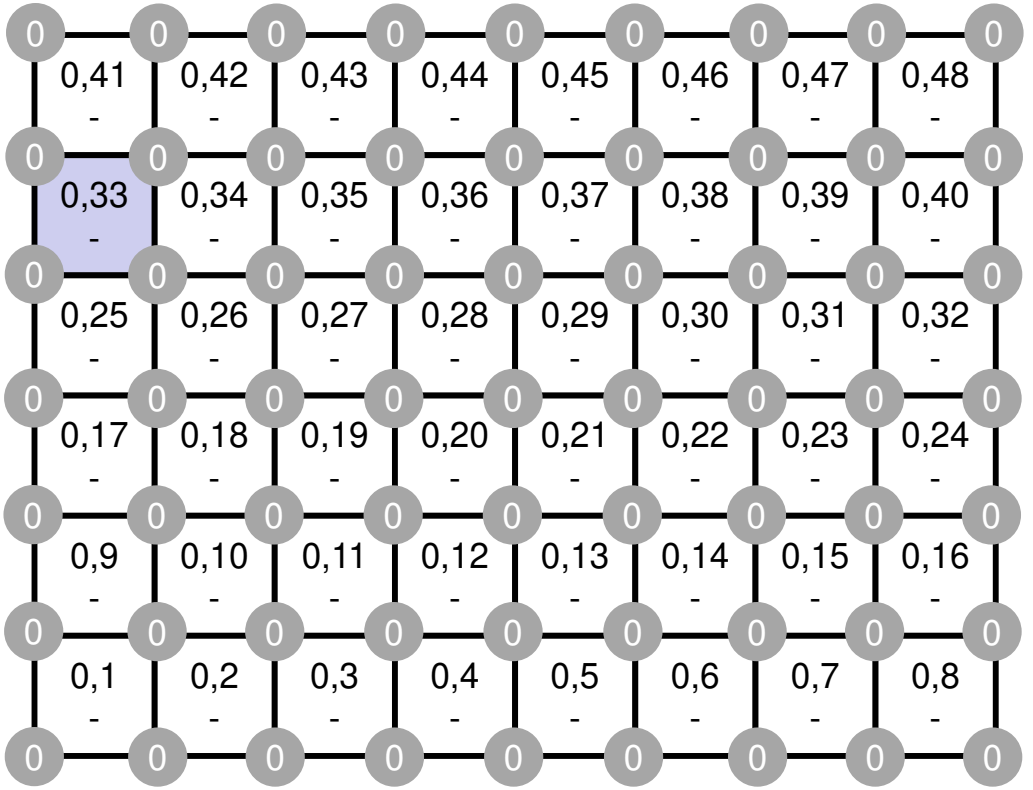
# Coloring (2D) (1/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          in1= IDold(icel,1)
          in2= IDold(icel,2)
          in3= IDold(icel,3)
          in4= IDold(icel,4)
          W1(in1)= 1
          W1(in2)= 1
          W1(in3)= 1
          W1(in4)= 1
          if (W2(icel).eq.0) then
            W2(icel)= icol
            IDnew(icel,1)= in1
            IDnew(icel,2)= in2
            IDnew(icel,3)= in3
            IDnew(icel,4)= in4
            icou= icou + 1
          endif
        enddo
        if (icou.eq.ICELTOT) goto 100
      enddo

  100 continue
      ELMCOLORtot= icol
      W3(0           )= 0
      W3(ELMCOLORtot)= ICELTOT

      do icol= 0, ELMCOLORtot
        ELMCOLORindex(icol)= W3(icol)
      enddo
```

W2(icel), IDold
IDnew

**W2**: Color ID of the Element
**IDold**: Element ID (Original)
**IDnew**: Element ID (New)

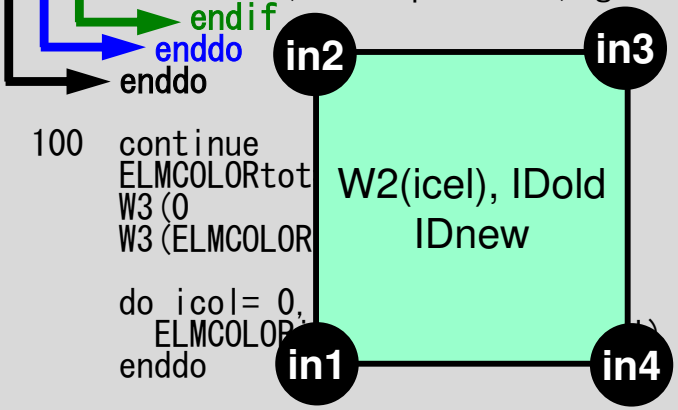
| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |



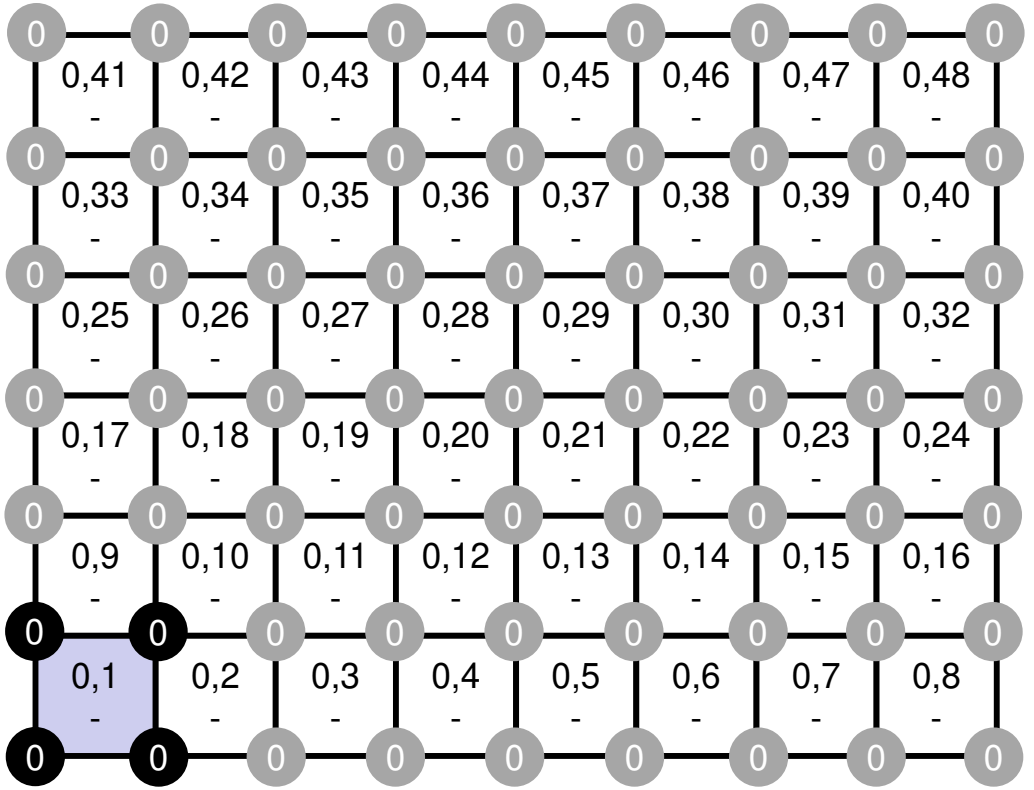
# Coloring (2D) (2/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1) (=0)
            ip2= W1(in2) (=0)
            ip3= W1(in3) (=0)
            ip4= W1(in4) (=0)

            isum= ip1 + ip2 + ip3 + ip4
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=1**
**icel=1**




| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |

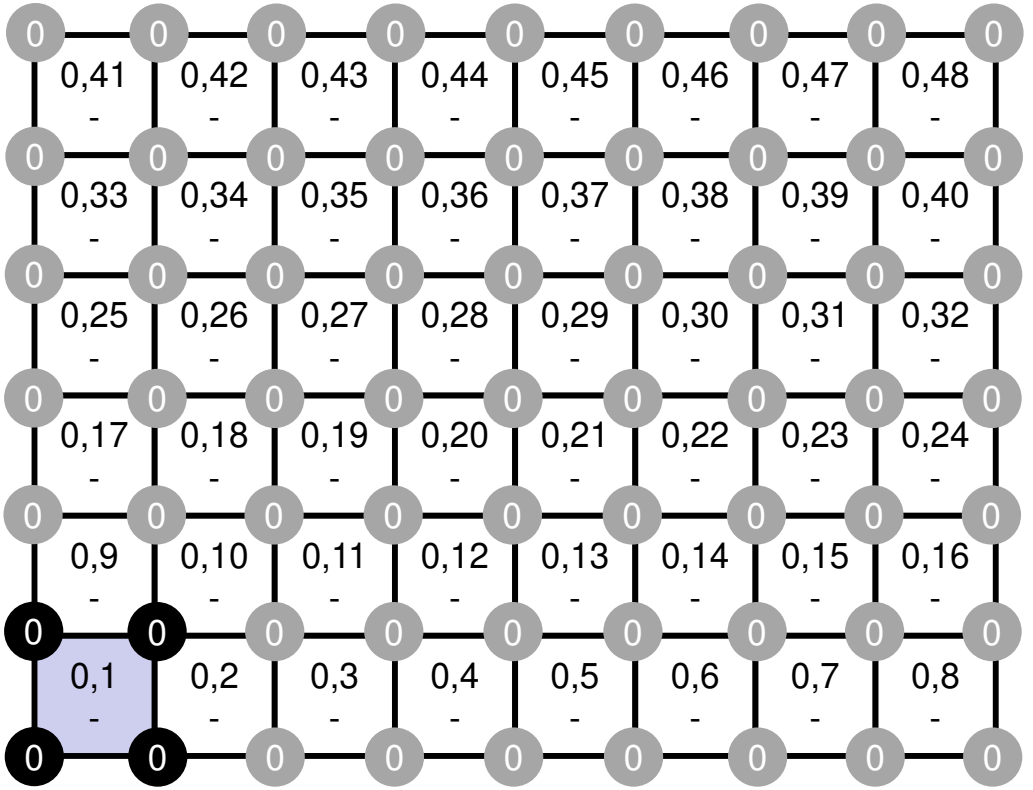


# Coloring (2D) (2/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1) (=0)
            ip2= W1(in2) (=0)
            ip3= W1(in3) (=0)
            ip4= W1(in4) (=0)

            isum= ip1 + ip2 + ip3 + ip4 (=0)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot= icol
      IWKX(0           ,3)= 0
      IWKX(ELMCOLORtot,3)= ICELTOT

      do icol= 0, ELMCOLORtot
        ELMCOLORindex(icol)= W3(icol)
      enddo
```

**icol=1**
**icel=1**

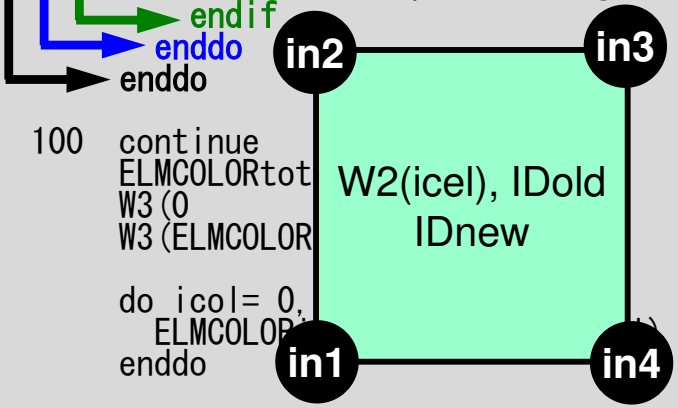
| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** (0 / 1) | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** [0] | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |



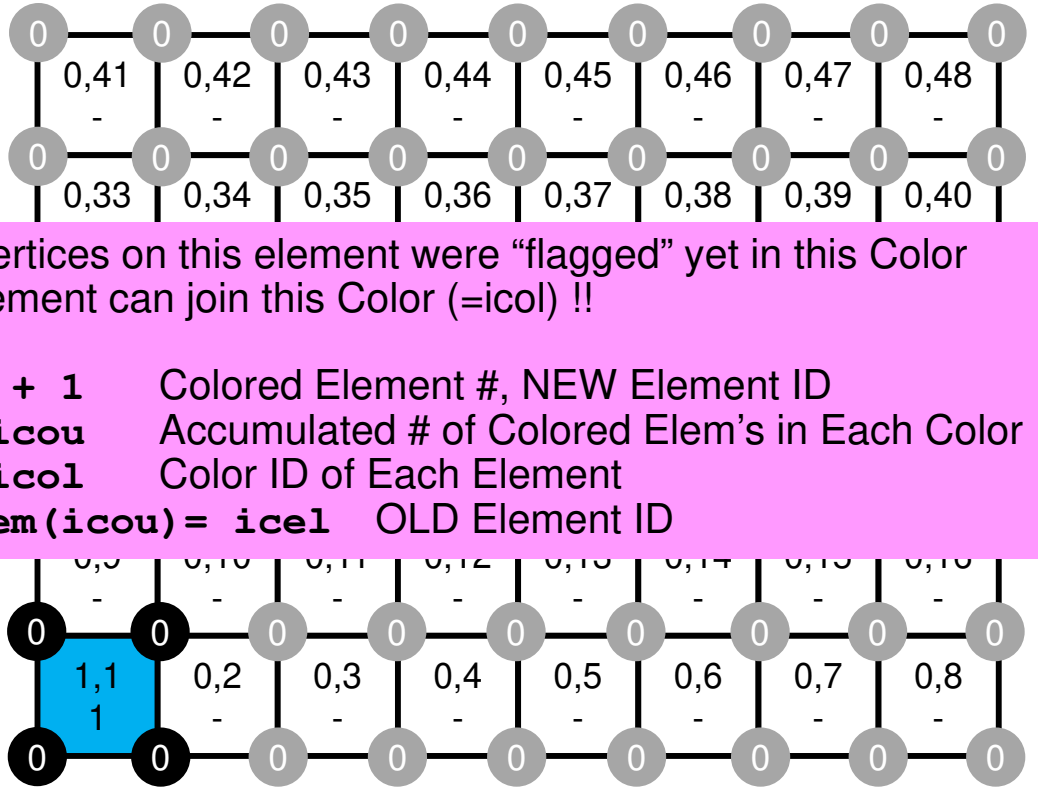
# Coloring (2D) (2/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1) (=0)
            ip2= W1(in2) (=0)
            ip3= W1(in3) (=0)
            ip4= W1(in4) (=0)

            isum= ip1 + ip2 + ip3 + ip4 (=0)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=1**
**icel=1**




| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |



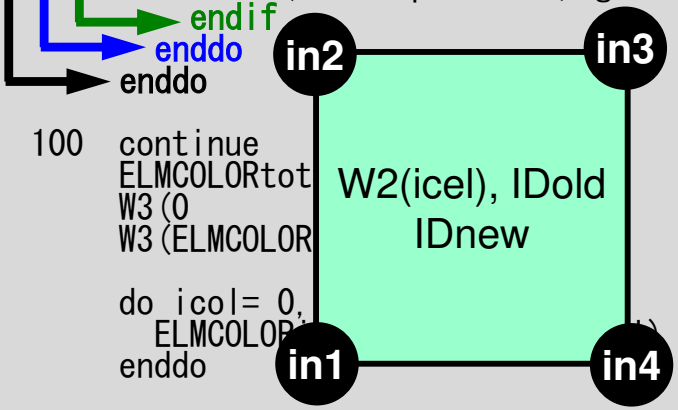

Because no vertices on this element were "flagged" yet in this Color (=icol), this element can join this Color (=icol) !!

**icou= icou + 1** Colored Element #, NEW Element ID
**W3(icol)= icou** Accumulated # of Colored Elem's in Each Color
**W2(icel)= icol** Color ID of Each Element
**ELMCOLORitem(icou)= icel** OLD Element ID

# Coloring (2D) (2/7)

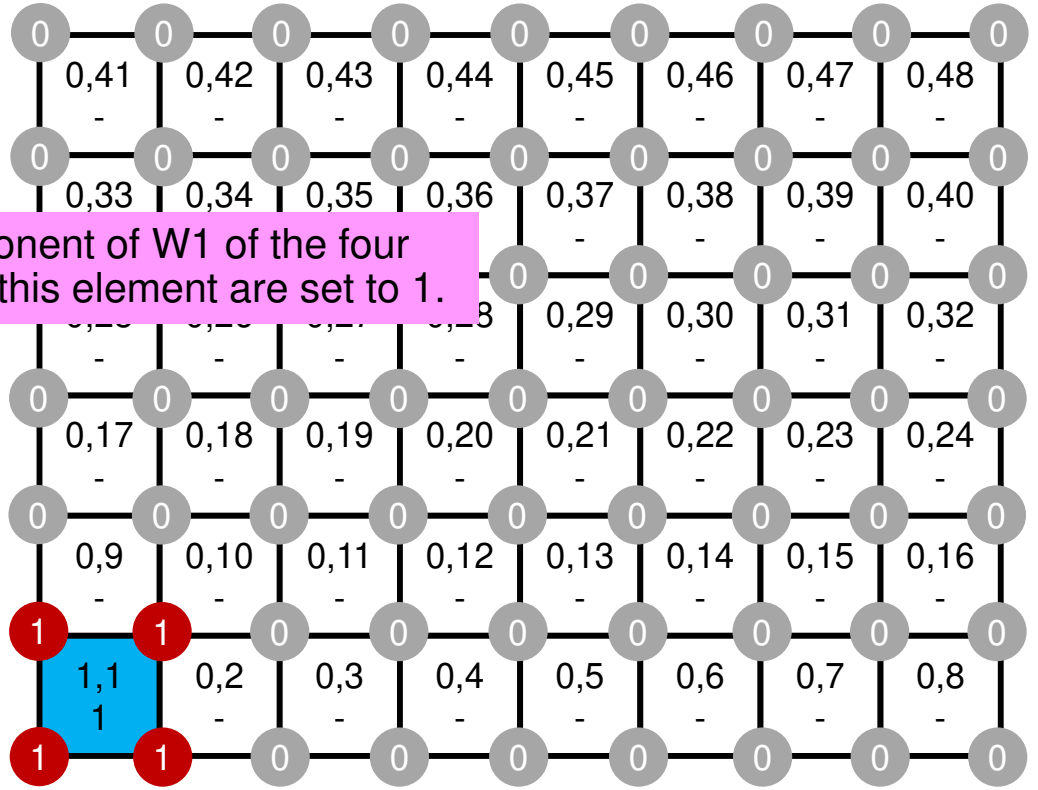
```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1)
            ip2= W1(in2)
            ip3= W1(in3)
            ip4= W1(in4)

            isum= ip1 + ip2 + ip3 + ip4 (=0)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=1**
**icel=1**




| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |

Each component of W1 of the four vertices on this element are set to 1.

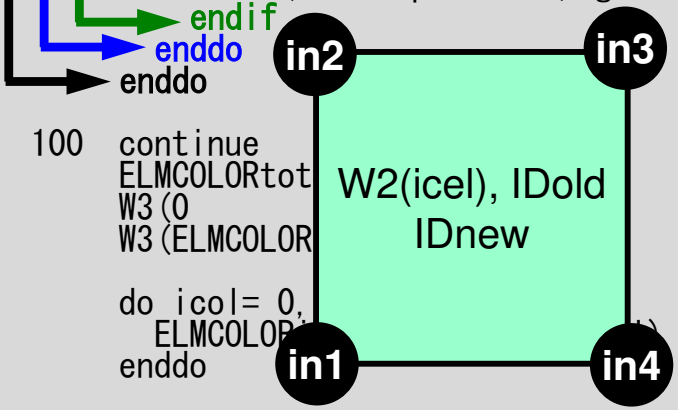


# Coloring (2D) (3/7)

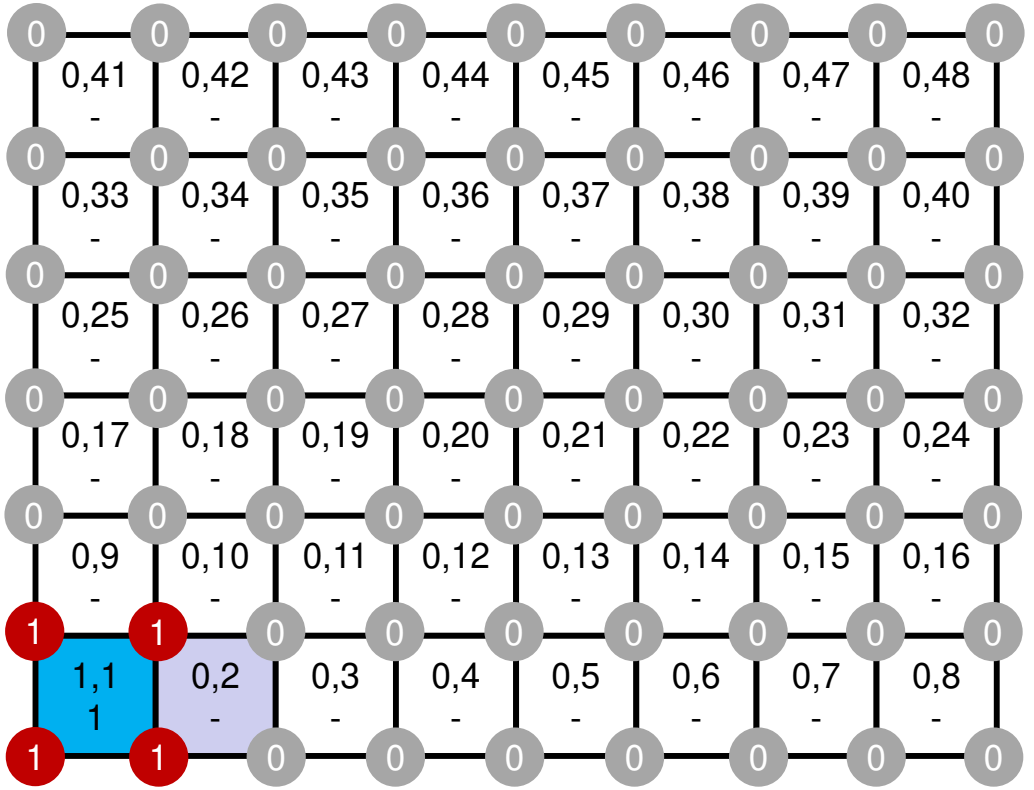
```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1) (=1)
            ip2= W1(in2) (=1)
            ip3= W1(in3) (=0)
            ip4= W1(in4) (=0)

            isum= ip1 + ip2 + ip3 + ip4 (=2)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
          endif
        enddo
      enddo

100   continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=1**
**icel=2**

- ✓ 2 of 4 vertices on this element are already “flagged” (isum=2)
- ✓ Elements in a same color do not share a node
- ✓ Therefore, this element (icel=2) cannot join this color (icol=1)



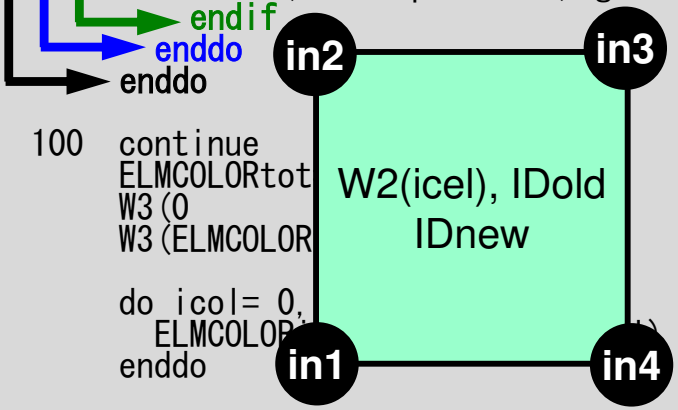

| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem’s in each Color |
| **ELMCOLORtot** | | Total # of Colors |



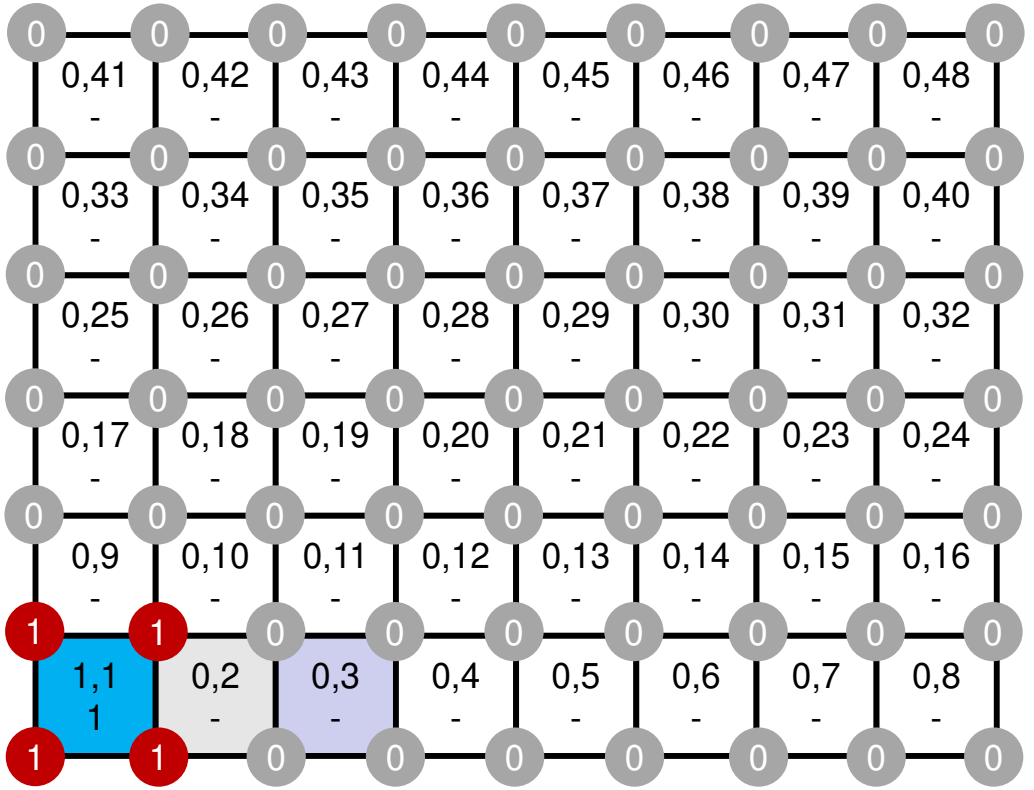
# Coloring (2D) (4/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1) (=0)
            ip2= W1(in2) (=0)
            ip3= W1(in3) (=0)
            ip4= W1(in4) (=0)

            isum= ip1 + ip2 + ip3 + ip4 (=0)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=1**
**icel=3**



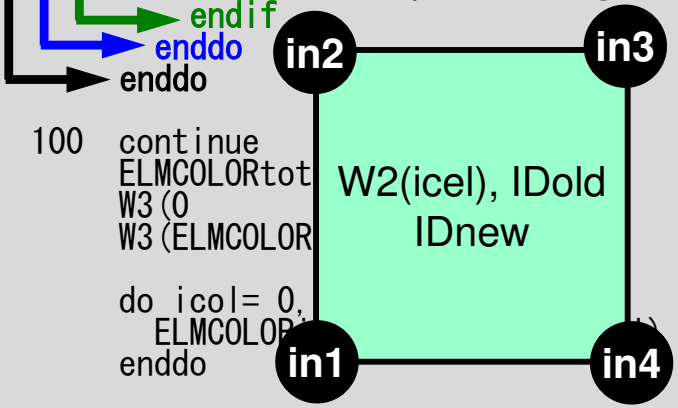

| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |



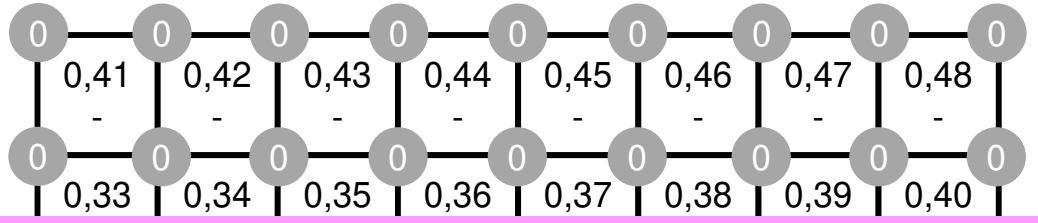
# Coloring (2D) (4/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1) (=0)
            ip2= W1(in2) (=0)
            ip3= W1(in3) (=0)
            ip4= W1(in4) (=0)

            isum= ip1 + ip2 + ip3 + ip4 (=0)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=1**
**icel=3**

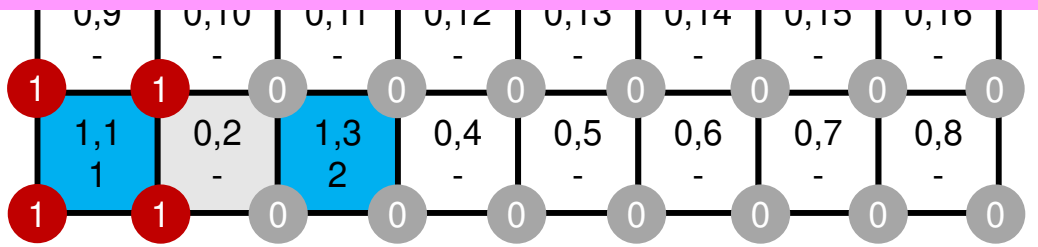



| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |




Because no vertices on this element were "flagged" yet in this Color (=icol), this element can join this Color (=icol) !!

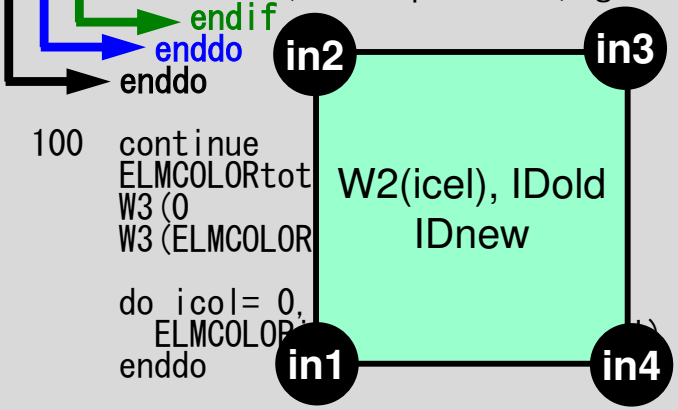
| | |
|---|---|
| **icou= icou + 1** | Colored Element #, NEW Element ID |
| **W3(icol)= icou** | Accumulated # of Colored Elem's in Each Color |
| **W2(icel)= icol** | Color ID of Each Element |
| **ELMCOLORitem(icou)= icel** | OLD Element ID |



# Coloring (2D) (4/7)

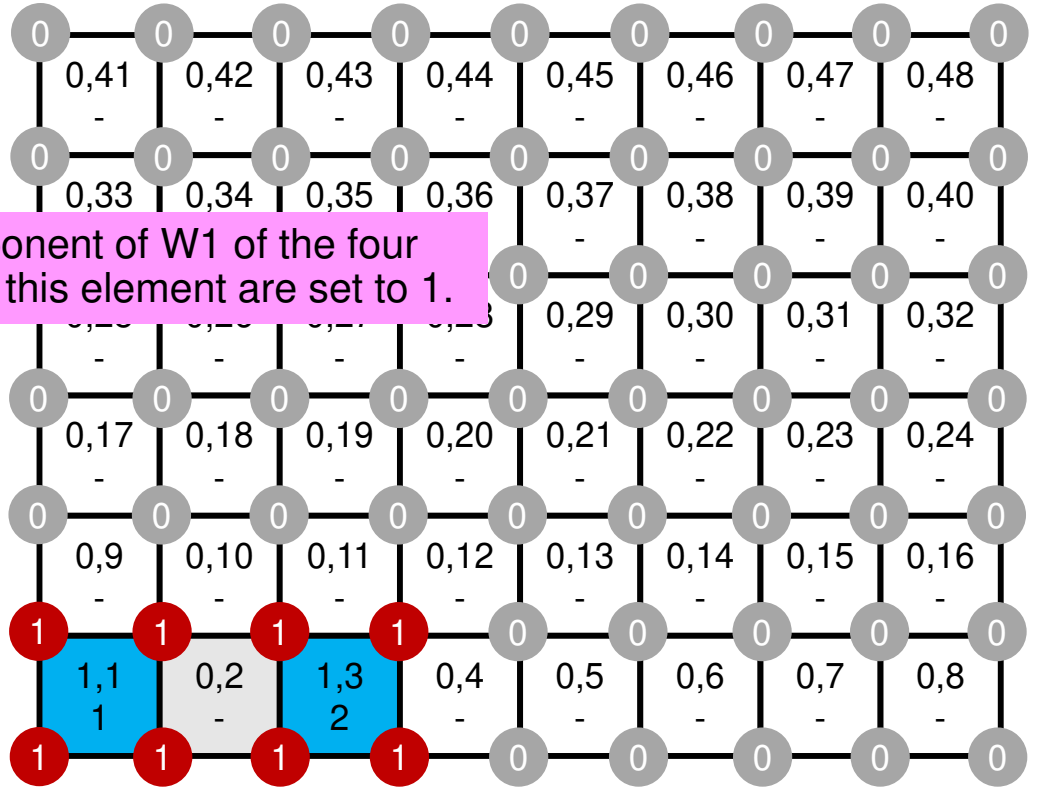
```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1)
            ip2= W1(in2)
            ip3= W1(in3)
            ip4= W1(in4)

            isum= ip1 + ip2 + ip3 + ip4 (=0)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=1**
**icel=3**




| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |

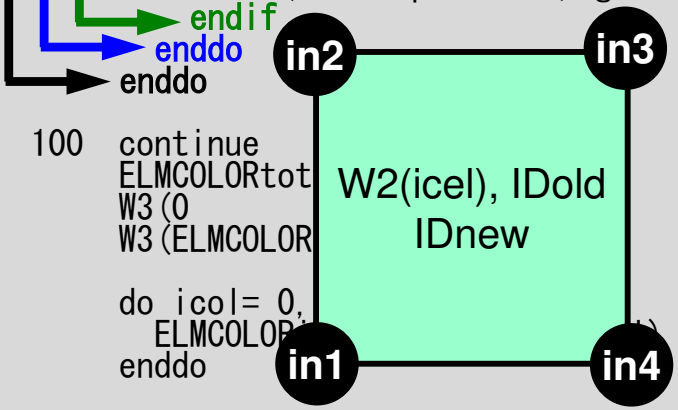
Each component of W1 of the four vertices on this element are set to 1.



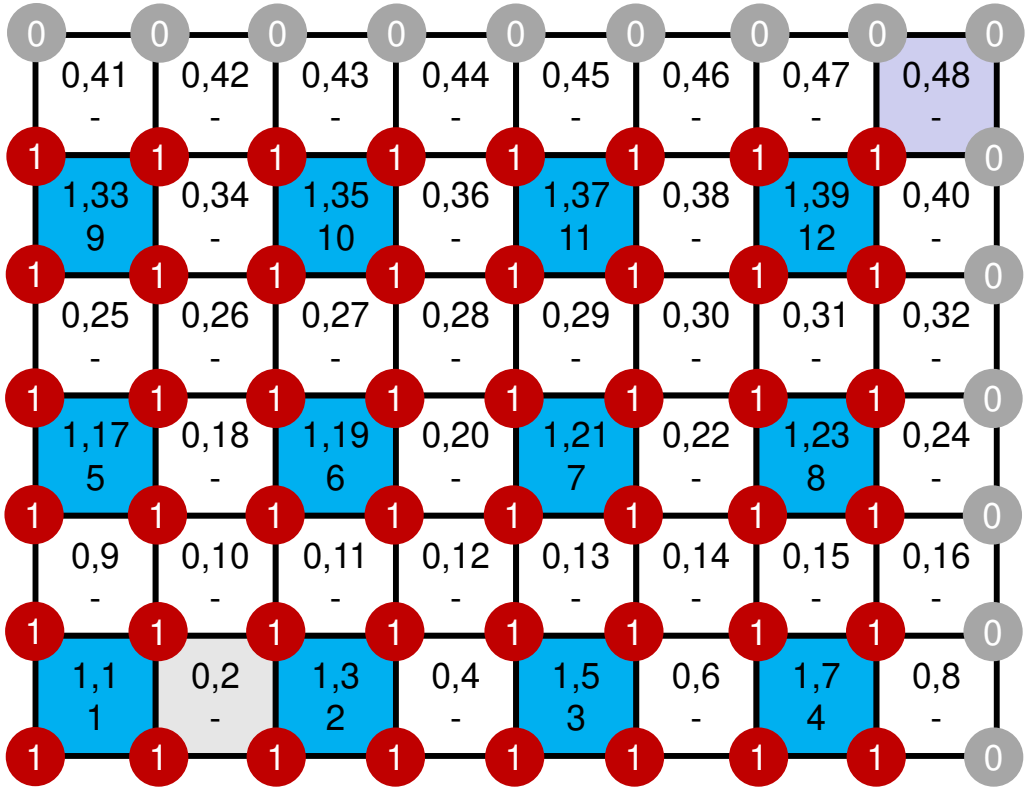
# Coloring (2D) (5/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (E
      allocate (W

      W1=0; W2=0;
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1) (=1)
            ip2= W1(in2) (=0)
            ip3= W1(in3) (=0)
            ip4= W1(in4) (=0)

            isum= ip1 + ip2 + ip3 + ip4 (=1)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=1**
**icel=ICELTOT (=48)**




| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |



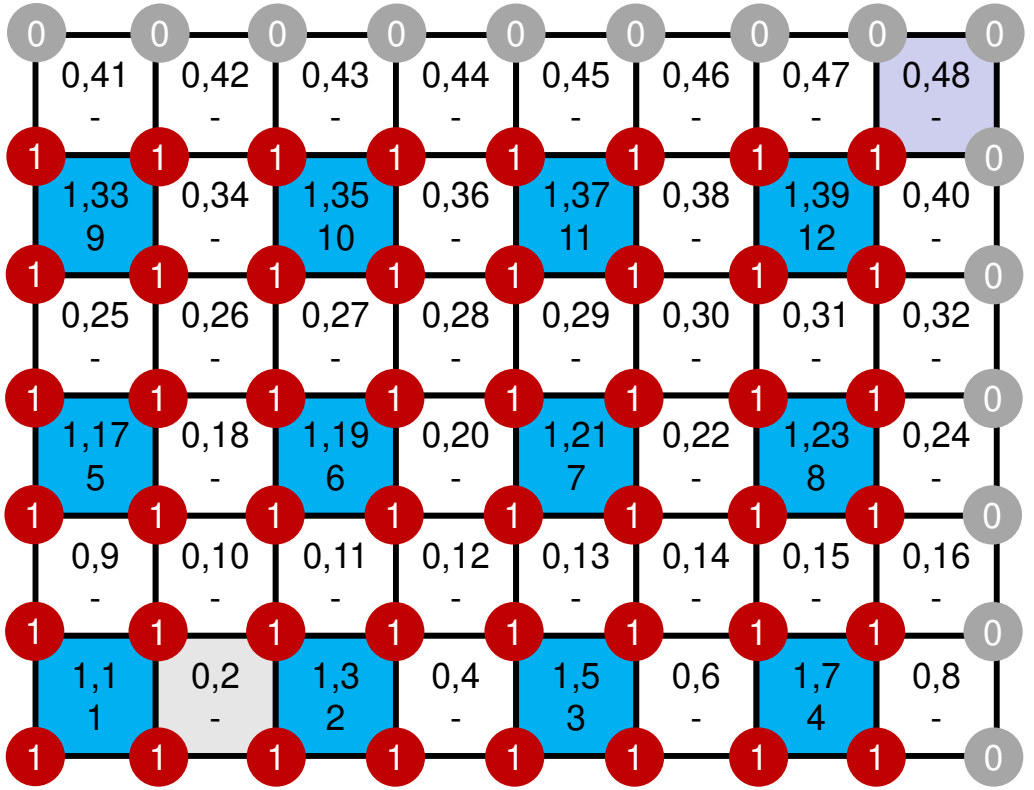
# Coloring (2D) (5/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (E
      allocate (W

      W1=0; W2=0;
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1) (=1)
            ip2= W1(in2) (=0)
            ip3= W1(in3) (=0)
            ip4= W1(in4) (=0)

            isum= ip1 + ip2 + ip3 + ip4 (=1)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

  100
```

**icol=1**
**icel=ICELTOT (=48)**

- ✓ Elements in a same color do not share a node
- ✓ Parallel operations are possible for elements in each color

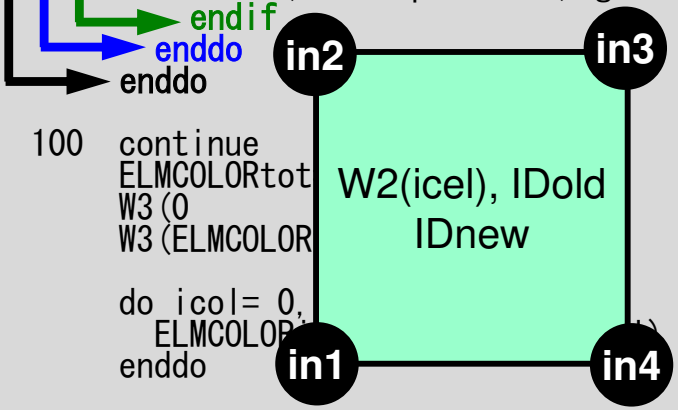
| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** (0) (1) | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** [0] | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |



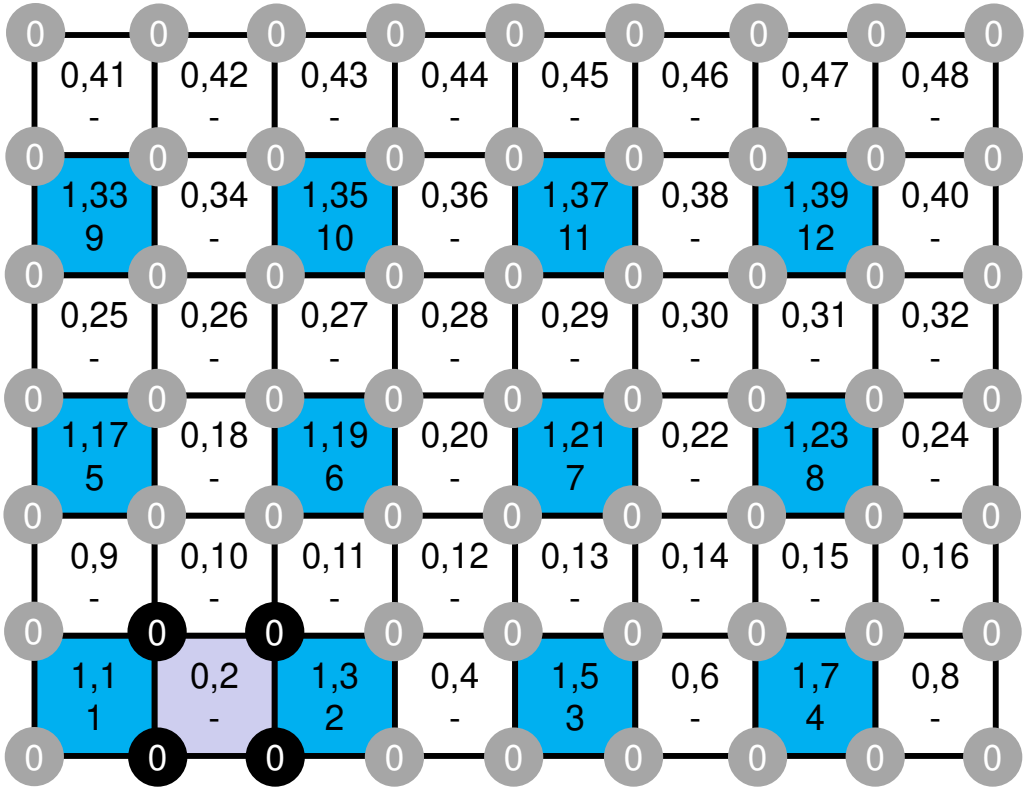
# Coloring (2D) (6/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1)  (=0)
            ip2= W1(in2)  (=0)
            ip3= W1(in3)  (=0)
            ip4= W1(in4)  (=0)

            isum= ip1 + ip2 + ip3 + ip4
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=2**
**icel=2**



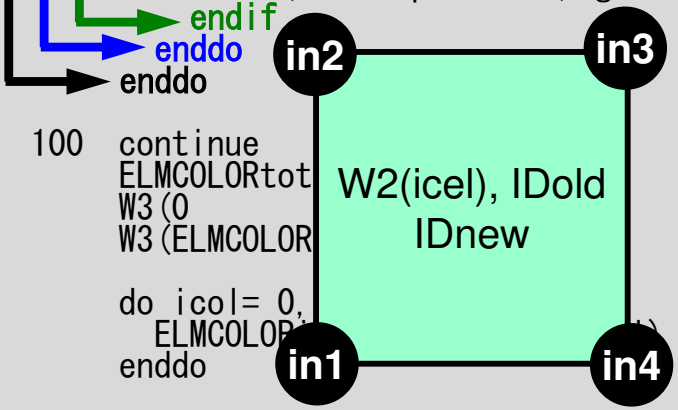

| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |



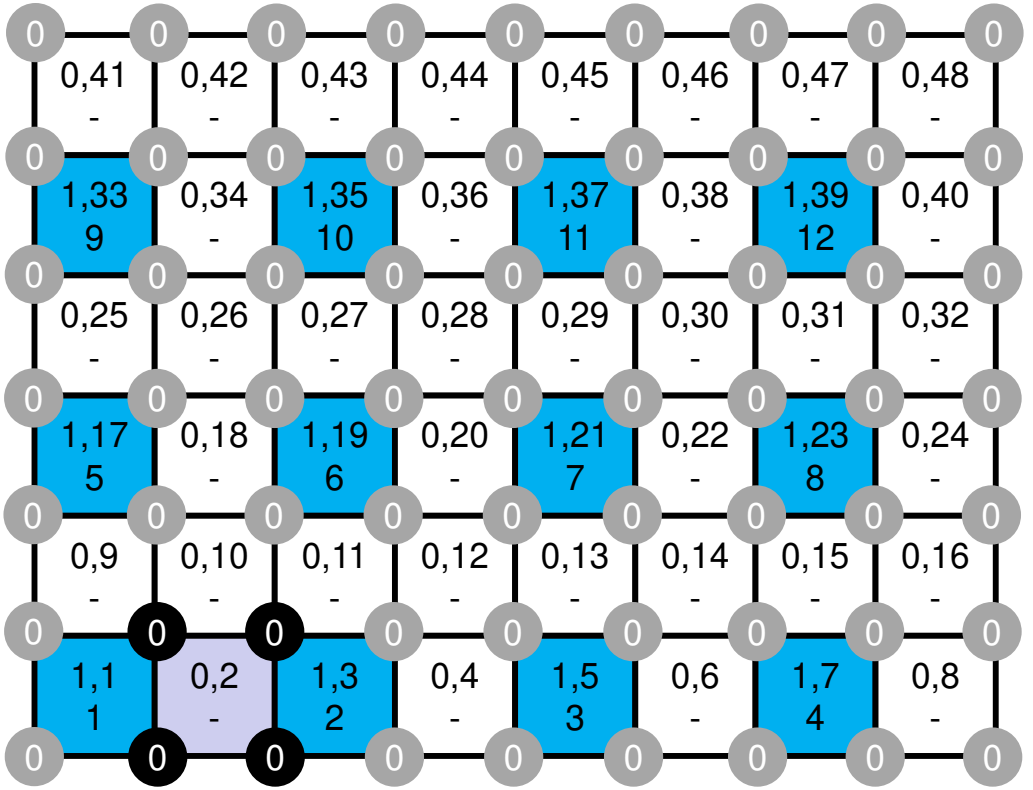
# Coloring (2D) (6/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1)  (=0)
            ip2= W1(in2)  (=0)
            ip3= W1(in3)  (=0)
            ip4= W1(in4)  (=0)

            isum= ip1 + ip2 + ip3 + ip4 (=0)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=2**
**icel=2**



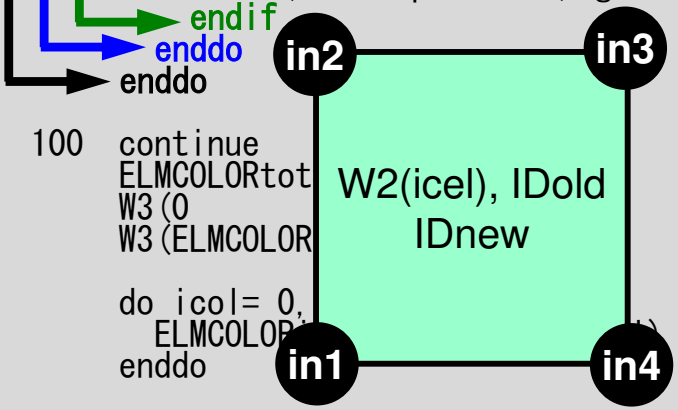

| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |



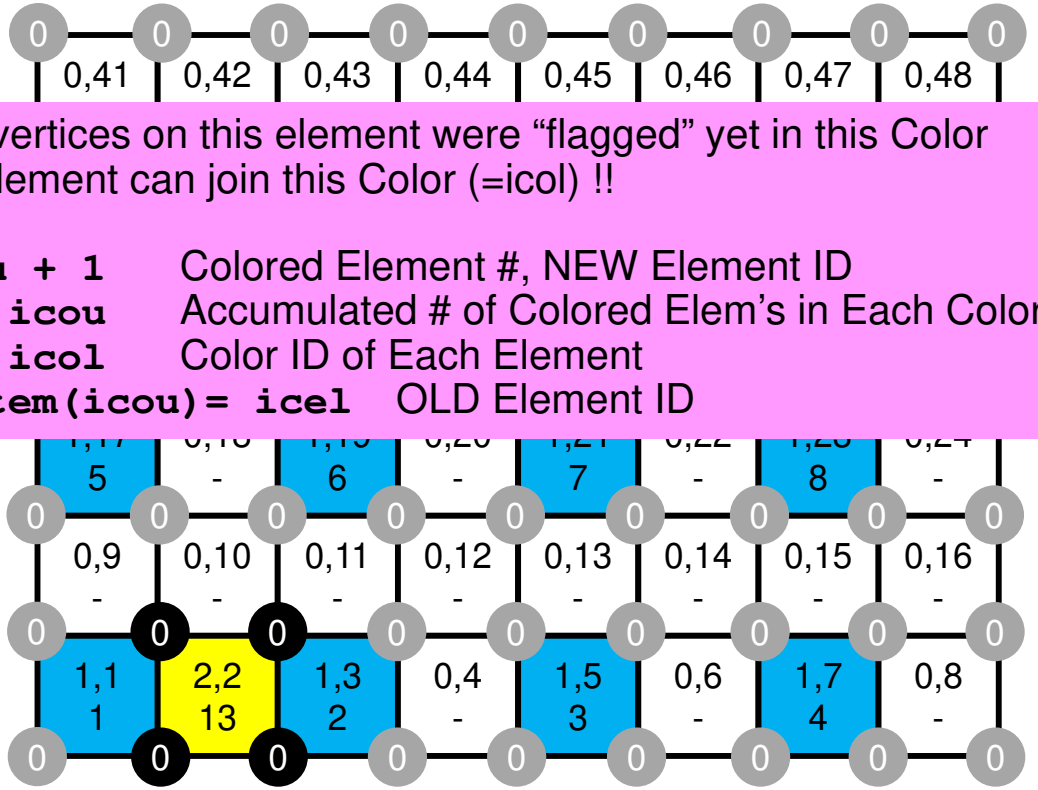
# Coloring (2D) (6/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1) (=0)
            ip2= W1(in2) (=0)
            ip3= W1(in3) (=0)
            ip4= W1(in4) (=0)

            isum= ip1 + ip2 + ip3 + ip4 (=0)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 1
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=2**
**icel=2**




| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |



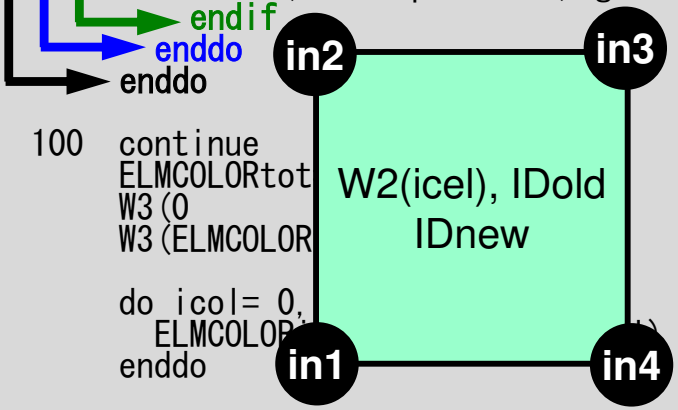

Because no vertices on this element were "flagged" yet in this Color (=icol), this element can join this Color (=icol) !!

**icou= icou + 1** Colored Element #, NEW Element ID
**W3(icol)= icou** Accumulated # of Colored Elem's in Each Color
**W2(icel)= icol** Color ID of Each Element
**ELMCOLORitem(icou)= icel** OLD Element ID

# Coloring (2D) (6/7)

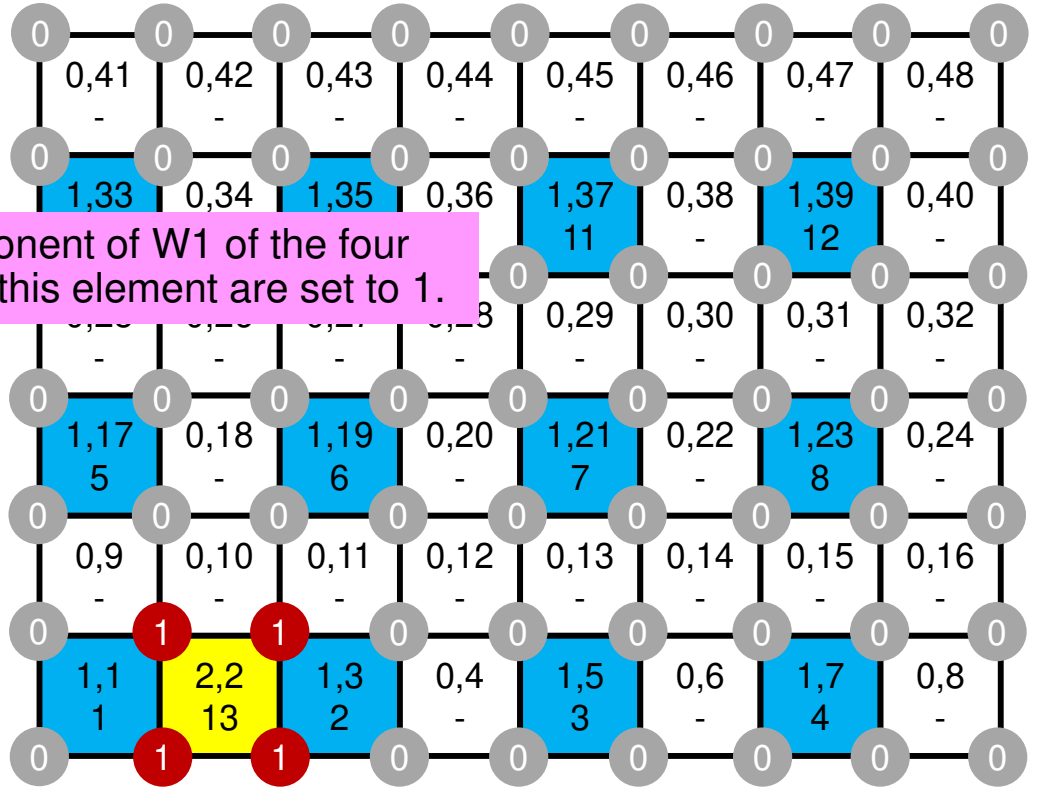
```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1)
            ip2= W1(in2)
            ip3= W1(in3)
            ip4= W1(in4)

            isum= ip1 + ip2 + ip3 + ip4 (=0)
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

  100 continue
      ELMCOLORtot
      W3(0
      W3(ELMCOLOR

      do icol= 0,
        ELMCOLOR
      enddo
```

**icol=2**
**icel=2**




| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |

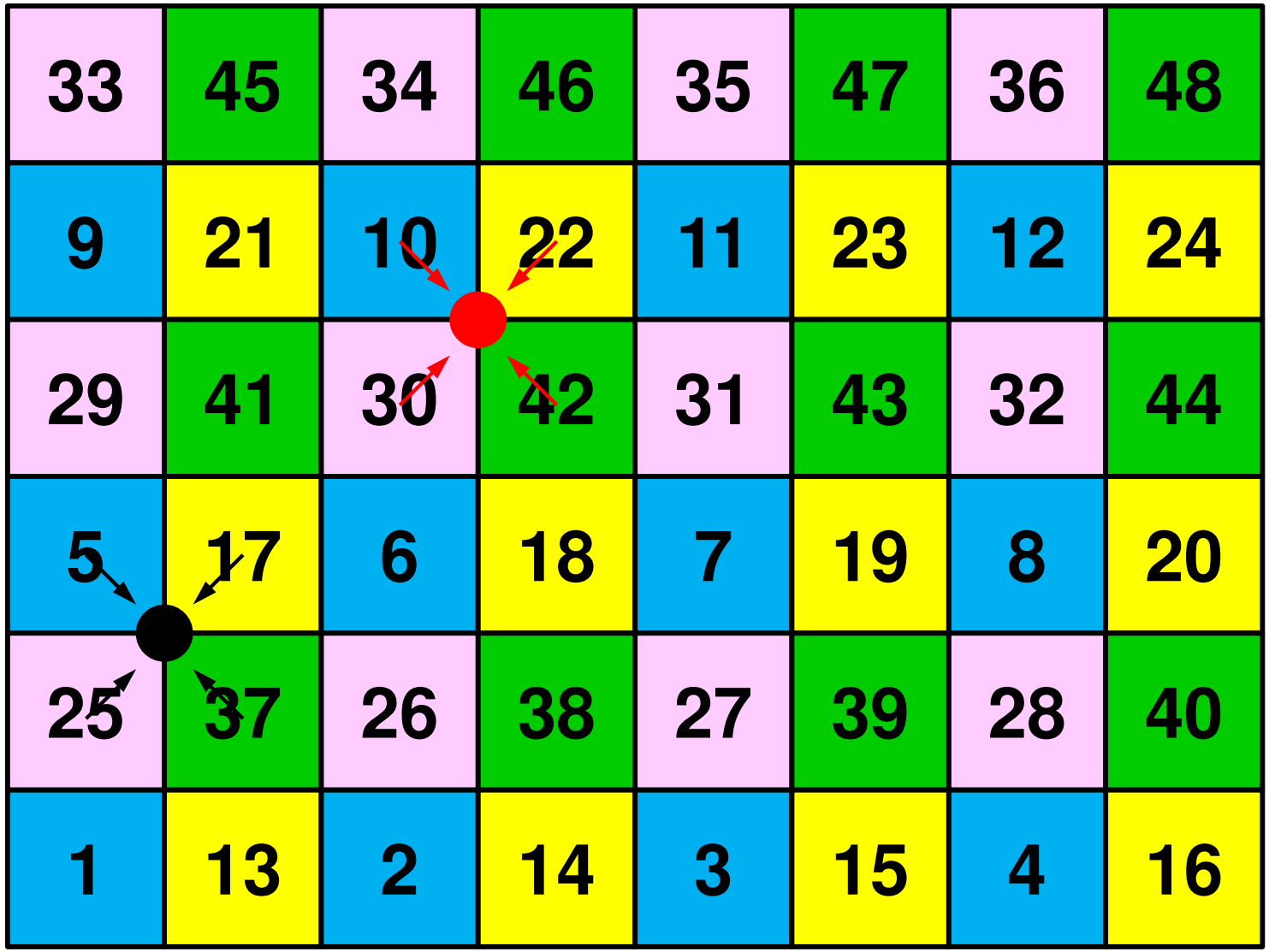
Each component of W1 of the four vertices on this element are set to 1.



# Multi-Threading: Mat_Ass

Parallel operations are possible for elements in same color (they are independent)

Colors of elements sharing a node are different



# Coloring (2D) (7/7)

```
      allocate (ELMCOLORindex(0:NP))

      allocate (ELMCOLORitem (ICELTOT))
      allocate (W1(NP), W2(ICELTOT), W3(NP))

      W1=0; W2=0; W3=0
      icou= 0
      do icol= 1, NP
        do i= 1, NP
          W1(i)= 0
        enddo
        do icel= 1, ICELTOT
          if (W2(icel).eq.0) then
            in1= ICELNOD(icel,1)
            in2= ICELNOD(icel,2)
            in3= ICELNOD(icel,3)
            in4= ICELNOD(icel,4)

            ip1= W1(in1)
            ip2= W1(in2)
            ip3= W1(in3)
            ip4= W1(in4)

            isum= ip1 + ip2 + ip3 + ip4
            if (isum.eq.0) then
              icou= icou + 1
              W3(icol)= icou
              W2(icel)= icol

              ELMCOLORitem(icou)= icel
              W1(in1)= 1
              W1(in2)= 1
              W1(in3)= 1
              W1(in4)= 1
            endif
            if (icou.eq.ICELTOT) goto 100
          endif
        enddo
      enddo

100   continue
      ELMCOLORtot= icol
      W3(0           )= 0
      W3(ELMCOLORtot)= ICELTOT

      do icol= 0, ELMCOLORtot
        ELMCOLORindex(icol)= W3(icol)
      enddo
```

| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **W1** 0 1 | **NP** | Flag of Each Node<br>=0: Not flagged in the current color<br>=1: Already Flagged |
| **W2** 0 | **ICELTOT** | Color ID of Each Element |
| **W3** | **0:NP** | Accumulated # of Colored Elem's in each Color |
| **ELMCOLORtot** | | Total # of Colors |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 33 | 45 | 34 | 46 | 35 | 47 | 36 | 48 |
| 9 | 21 | 10 | 22 | 11 | 23 | 12 | 24 |
| 29 | 41 | 30 | 42 | 31 | 43 | 32 | 44 |
| 5 | 17 | 6 | 18 | 7 | 19 | 8 | 20 |
| 25 | 37 | 26 | 38 | 27 | 39 | 28 | 40 |
| 1 | 13 | 2 | 14 | 3 | 15 | 4 | 16 |

# Multi-Threaded Matrix Assembling Procedure

```
      do icol= 1, ELMCOLORtot
!$omp parallel do private (icel0,icel)                                    &
!$omp&            private (in1,in2,in3,in4,in5,in6,in7,in8)               &
!$omp&            private (nodLOCAL,ie,je,ip,jp,kk,iiS,iiE,k)             &
!$omp&            private (DETJ,PNX,PNY,PNZ,QVC,QV0,COEFij,coef,SHi)      &
!$omp&            private (PNXi,PNYi,PNZi,PNXj,PNYj,PNZj,ipn,jpn,kpn)     &
!$omp&            private (X1,X2,X3,X4,X5,X6,X7,X8)                       &
!$omp&            private (Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8)                       &
!$omp&            private (Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8,CONDO)
      do icel0= ELMCOLORindex(icol-1)+1, ELMCOLORindex(icol)
        icel= ELMCOLORitem(icel0)
        in1= ICELNOD(icel,1)
        in2= ICELNOD(icel,2)
        in3= ICELNOD(icel,3)
        in4= ICELNOD(icel,4)
        in5= ICELNOD(icel,5)
        in6= ICELNOD(icel,6)
        in7= ICELNOD(icel,7)
        in8= ICELNOD(icel,8)
...
```

**icel0**: NEW Elem. ID, **icel**: OLD Elem. ID

| Name | Size | Content |
|---|---|---|
| **ELMCOLORindex** | **0:NP** | Number of Elements in Each Color |
| **ELMCOLORitem** | **ICELTOT** | OLD Element ID in Each Color |
| **ELMCOLORtot** | | Total # of Colors |

# How to apply multi-threading

- CG Solver
  - Just insert OpenMP directives
  - ILU/IC preconditioning is much more difficult
- MAT_ASS (mat_ass_main, mat_ass_bc)
  - Data Dependency
  - Each Node is shared by 4/8-Elements (in 2D/3D)
  - If 4 elements are trying to accumulate local element matrices to the global matrix simultaneously in parallel computing:
    - Results may be changed
    - Deadlock may occur
  - Actually, “coloring” process is very difficult to be parallelized: research topic

# OpenMP（Matrix Ass.）（F・C）

```
>$ cd /home/ra020019/<Your-UID>/pFEM/pfem3d/src2
>$ make
>$ cd ../run
>$ ls sol2
      sol2

>$ cd ../pmesh

<Parallel Mesh Generation>

>$ cd ../run

<modify x12.sh>

>$ pjsub x12.sh
```

# mesh.inp

## Flat MPI

**1-node**
```
256 256 192
  4   4   3
pcube
```

**12-nodes**
```
MeTiS
```

**2-nodes**
```
256 256 192
  8   4   3
pcube
```

**16-nodes**
```
256 256 192
  8   8  12
pcube
```

**4-nodes**
```
256 256 192
  8   8   3
pcube
```

**24-nodes**
```
MeTiS
```

**8-nodes**
```
256 256 192
  8   8   6
pcube
```

## HB 12x4

**1-node**
```
256 256 192
  2   2   1
pcube
```

**12-nodes**
```
256 256 192
  4   4   3
pcube
```

**2-nodes**
```
256 256 192
  2   2   2
pcube
```

**16-nodes**
```
256 256 192
  4   4   4
pcube
```

**4-nodes**
```
256 256 192
  4   2   2
pcube
```

**24-nodes**
```
256 256 192
  4   4   6
pcube
```

**8-nodes**
```
256 256 192
  4   4   2
pcube
```

**24-nodes**
```
256 256 192
  8   4   3
pcube
```

# x12.sh

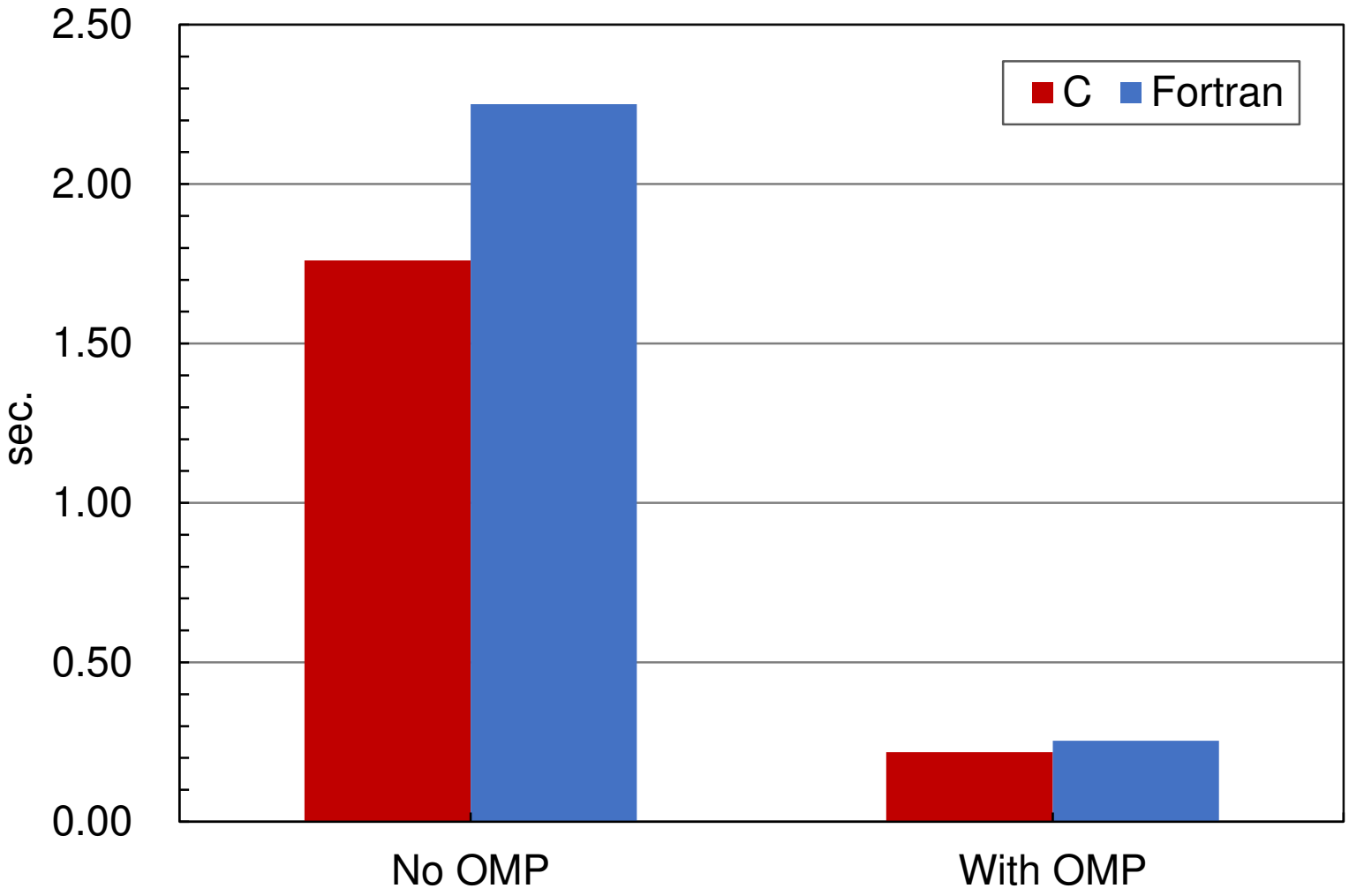
## HB 12x4: 48 processes

```
#!/bin/sh
#PJM -N "hb-12"
#PJM -L “rscgrp=small”
#PJM -L “node=12:torus”
#PJM --mpi “max-proc-per-node=4
#PJM -L elapse=00:15:00
#PJM -g ra020019
#PJM -j
#PJM -e err
#PJM -o x12.lst

export OMP_NUM_THREADS=12
mpiexec ./sol2
mpiexec numactl -l ./sol2
```

# Speed-up of Mat-Ass-Main N=256x256x192, 12-nodes

## 8x~9x times by 12-threads

## No OMP: src1, With OMP: src2



## Fortran

```
      START_TIME= MPI_WTIME()

      call MAT_ASS_MAIN
      call MAT_ASS_BC

      END_TIME= MPI_WTIME()
      if (my_rank.eq.0) then
        write (*, '("*** matrix ass.  ", 1pe16.6, " sec.",/)')          &
     &          END_TIME-START_TIME
      endif
!C===
```

## C Language

```
  START_TIME= MPI_Wtime();

  MAT_ASS_MAIN();
  MAT_ASS_BC()  ;

  END_TIME= MPI_Wtime();

  if (my_rank == 0) {
    fprintf(stdout,"*** matrix ass.  %e sec.¥n",END_TIME-START_TIME);
    fprintf(fp_log,"*** matrix ass.  %e sec.¥n",END_TIME-START_TIME);
  }
```