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

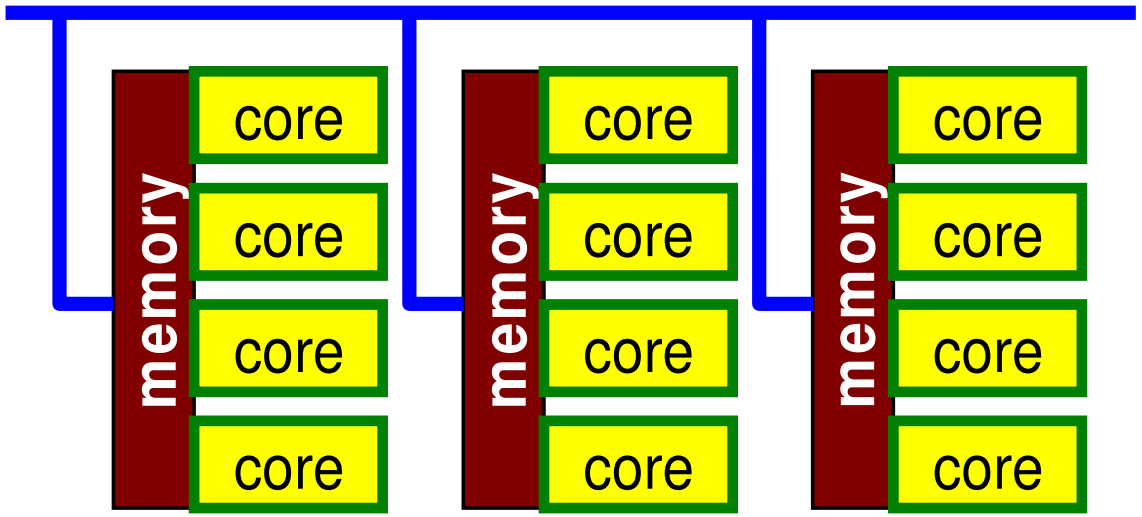
Kengo Nakajima
RIKEN R-CCS

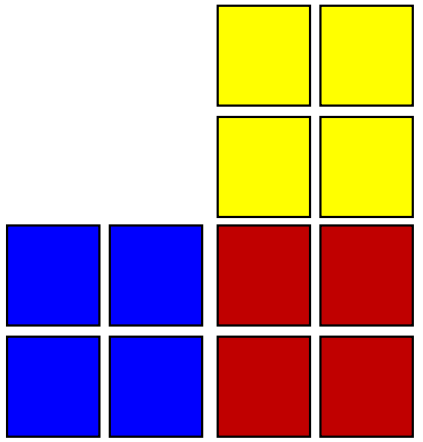
Hybrid Parallel Programming Model

- 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

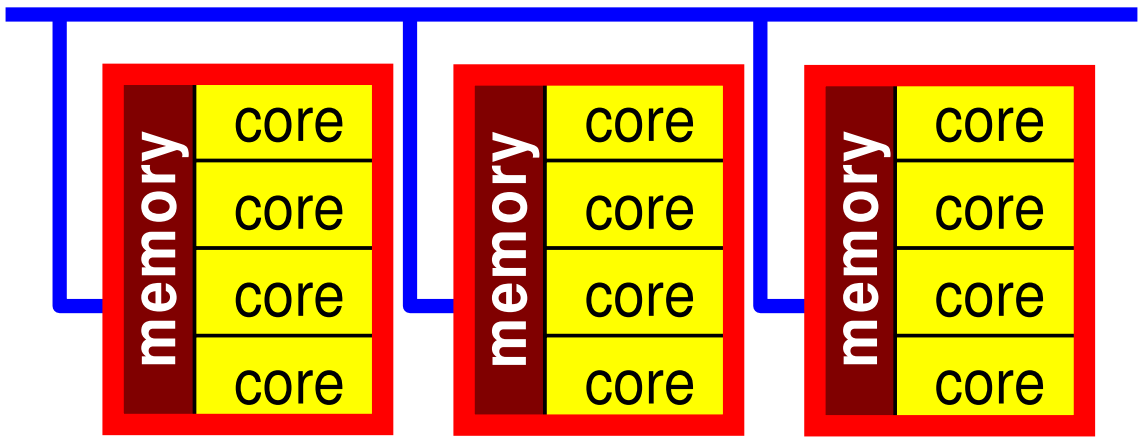
Flat MPI vs. Hybrid

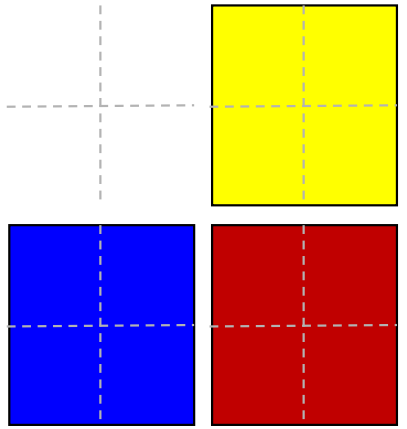
Flat-MPI：Each Core -> Independent





Hybrid：Hierarchal Structure

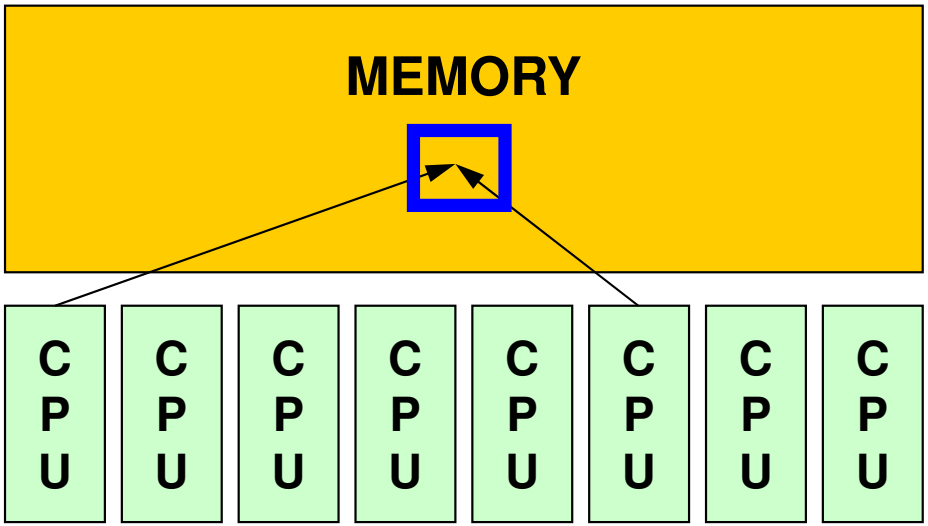




Background

- 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

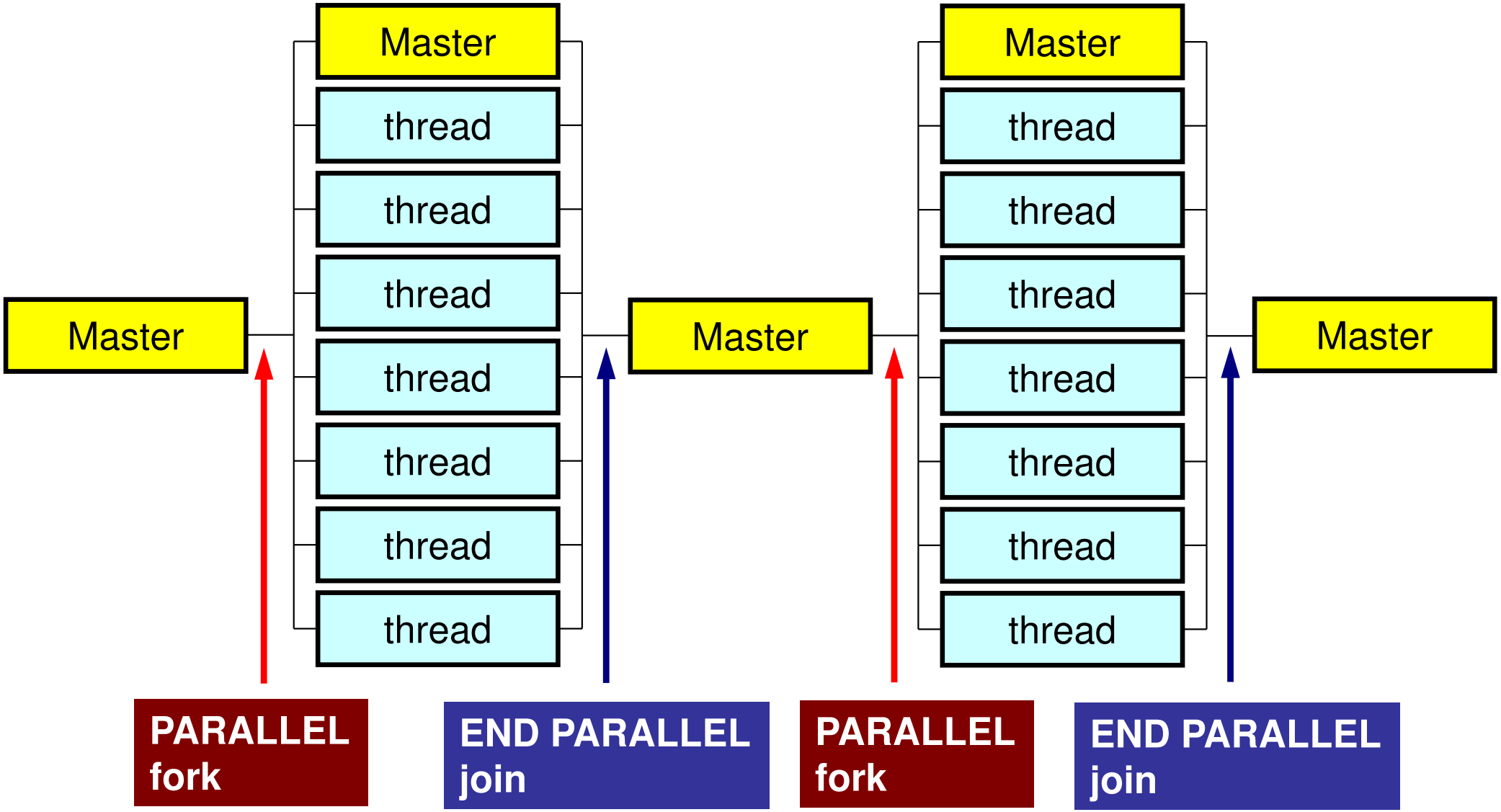
- An API (Application Programming Interface) for multi-platform shared-memory parallel programming in C/C++ and Fortran
  - Current version: 4.X (5.0 is already announced)
    - GPU, Accelerators: close to OpenACC
- Background
  - Merger of Cray and SGI in 1996
    - Separated later, … but both are now 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

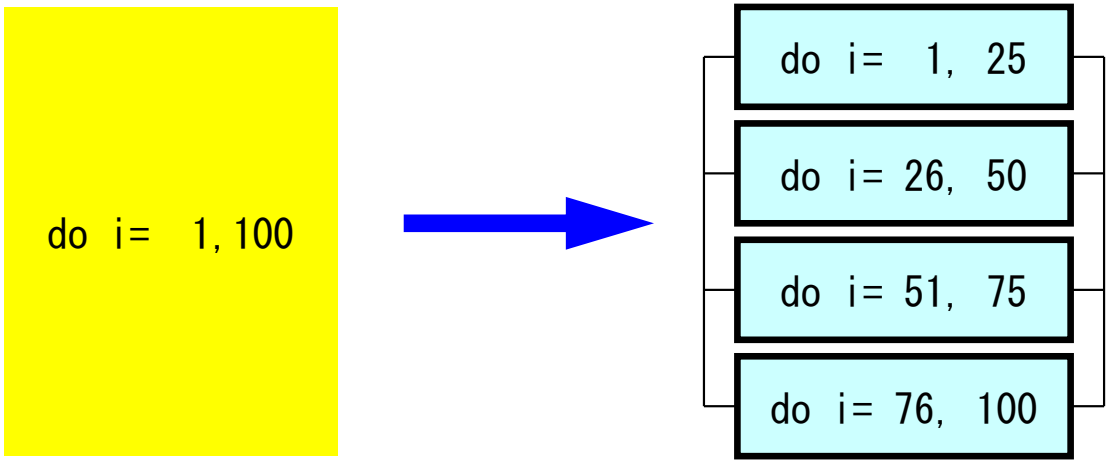
Fork-Join Parallel Execution Model



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

```
#pragma omp parallel for private (i)
 for (i=0; i<N; i++) {
       X[i] = 0.0;
       W[0][i] = 0.0;
       W[1][i] = 0.0;
       W[2][i] = 0.0;
 }
```

Dot Products

```
RHO = 0.0;
#pragma omp parallel for private (i)
 reduction (+:RHO)
 for(i=0; i<N; i++) {
     RHO += W[R][i] * W[Z][i];
 }
```

DAXPY

```
#pragma omp parallel for private (i)
 for (i=0; i<N; i++) {
    Y[i] = Y[i} + alpha*X[i];
 }
```

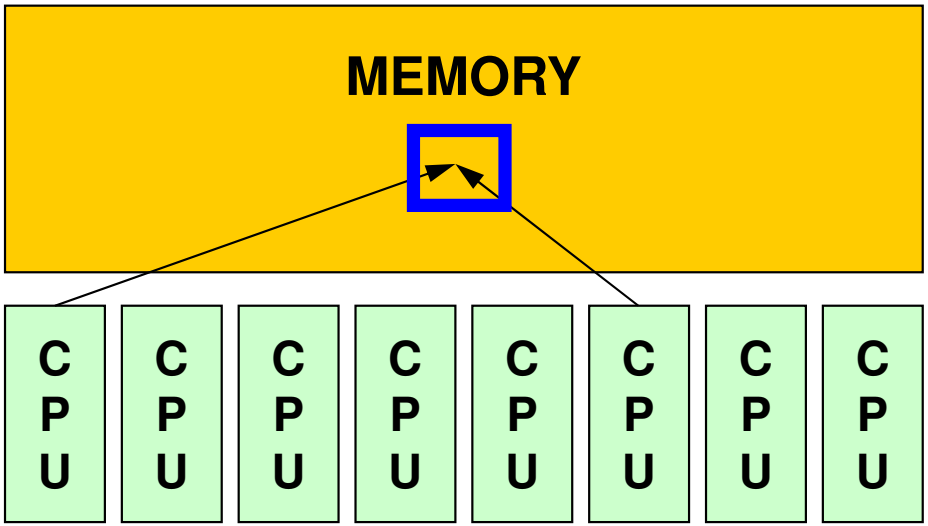
OpenMP/Direceives Matrix/Vector Products

```
#pragma omp parallel for private (i,VAL,j)
 for(i=0; i<N; i++) {
      VAL = D[i] * W[P][i];
      for(j=indexL[i]; j<indexL[i+1]; j++) {
          VAL += AL[j] * W[P][itemL[j]-1];
      }

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

Features of OpenMP

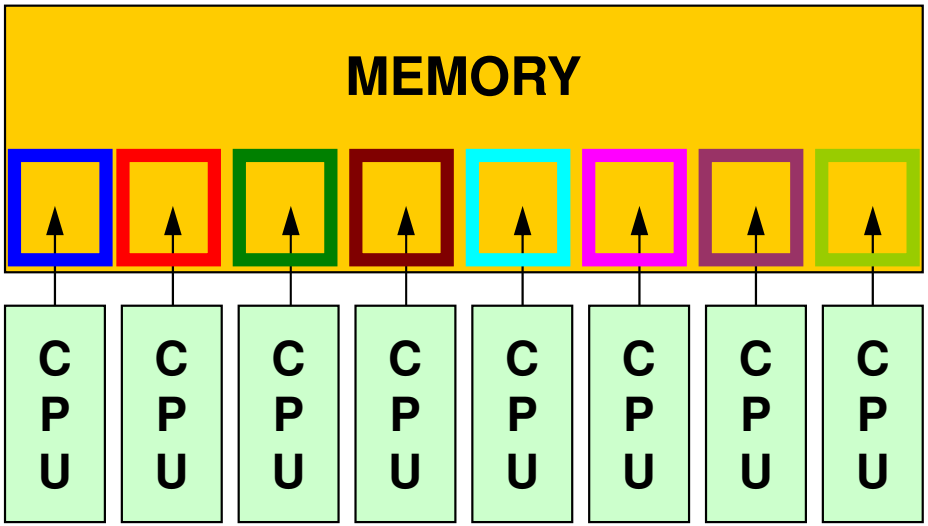
- Directives
  - Loops right after the directives are parallelized.
  - If the compiler does not support OpenMP, directives are considered as just comments.
- Nothing happen without explicit directives
  - Different from “automatic parallelization/vectorization”
  - Something wrong may happen by un-proper 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.)

- “for” loops with “#pragma omp parallel for”
- Global (Shared) Variables, Private Variables
  - Default: Global (Shared)
  - Dot Products: reduction

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

```
RHO = 0.0;
#pragma omp parallel for private (i) reduction (+:RHO)
 for(i=0; i<N; i++) {
     RHO += W[R][i] * W[Z][i];
 }
```

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, COMP, *$OMP
    - !$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

```
#pragma omp parllel for private (i)
 for(i=0; i<N; i++){
     B[i]= (A[i] + B[i]) * 0.50;
 }
```

- 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

Example-2: REDUCTION

```
#pragma omp parallel default(private) reduction(+:A,B)
 for(i=0; i<N; i++){
      err= work(Alocal, Blocal);
      A= A + Alocal;
      B= B + Blocal;
 }
```

Functions which can be used with OpenMP

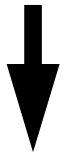
Name	Functions
int omp_get_num_threads (void)	Total Thread #
int omp_get_thread_num (void)	Thread ID
double omp_get_wtime (void)	= MPI_Wtime
void omp_set_num_threads (int num_threads) call omp_set_num_threads (num_threads)	Setting Thread #

OpenMP for Dot Products

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

OpenMP for Dot Products

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



```
    VAL= 0.0;
#pragma omp parallel for private (i) reduction(+:VAL)
    for(i=0; i<N; i++){
      VAL= VAL + W[R][i] * W[Z][i];
    }
```

Directives are just inserted.

OpenMP for Dot Products

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

```
      VAL= 0.0;
#pragma omp parallel for private (i) reduction(+:VAL)
      for(i=0; i<N; i++){
        VAL= VAL + W[R][i] * W[Z][i];
      }
```

Directives are just inserted.

```
       VAL= 0.0;
#pragma omp parallel for private (i,ip)  reduction(+:VAL)
      for(ip=0; ip<PEsmpTOT; ip++){
          for (i=INDEX[ip]; i<INDEX[ip+1]; i++){
             VAL= VAL + W[R][i] * W[Z][i];
          }
      }
```

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= 0.0;
#pragma omp parallel for private (i,ip) reduction(+:VAL)
    for(ip=0; ip<PEsmpTOT; ip++){
        for (i=INDEX[ip]; i<INDEX[ip+1]; i++){
           VAL= VAL + W[R][i] * W[Z][i];
        }
    }
```

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

INDEX[0]= 0
INDEX[1]= 25
INDEX[2]= 50
INDEX[3]= 75
INDEX[4]= 100

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.

NOT good for GPU's

Matrix-Vector Multiply

```
for(i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexLU[i]; j<indexLU[i+1]; j++) {
        VAL += AMAT[j] * W[P][itemLU[j]-1];
    }
    W[Q][i] = VAL;
}
```

Matrix-Vector Multiply

```
#pragma omp parallel for private(ip,i,VAL,j)
 for (ip=0; ip<PEsmpTOT; ip++){
       for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++){
        VAL = D[i] * W[P][i];
        for(j=indexLU[i]; j<indexLU[i+1]; j++) {
            VAL += AMAT[j] * W[P][itemLU[j]-1];
        }
        W[Q][i] = VAL;       }
 }
```

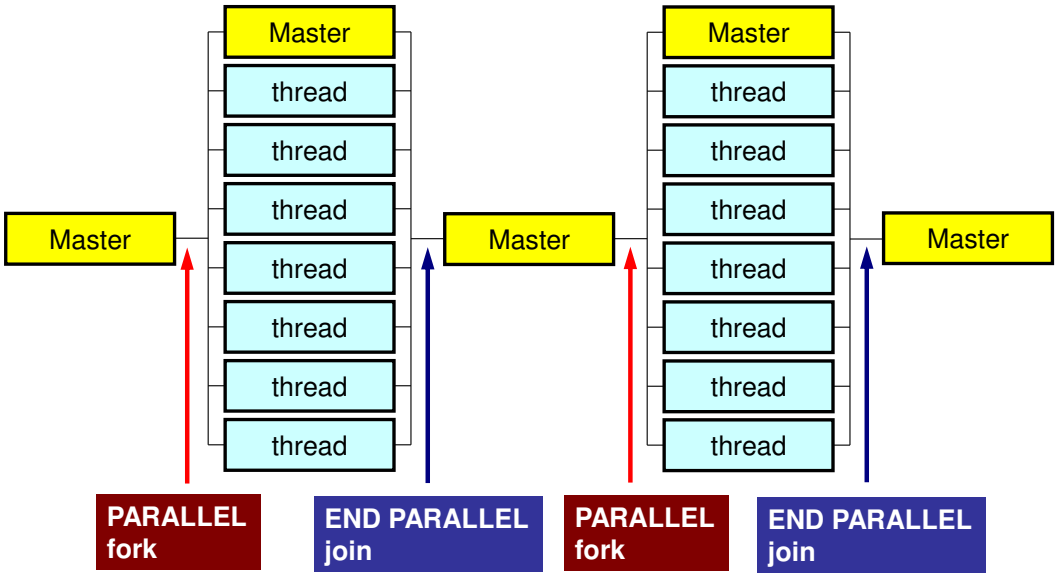
Matrix-Vector Multiply: Other Approach

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

```
#pragma omp parallel for private(i,VAL,j)
 for(i=0; i<N; i++){
      VAL = D[i] * W[P][i];
      for(j=indexLU[i]; j<indexLU[i+1]; j++) {
          VAL += AMAT[j] * W[P][itemLU[j]-1];
      }
      W[Q][i] = VAL;
 }
```

omp parallel (do)

- “omp parallel-omp end parallel” = “fork-join”
- If you have many loops, these “fork-join’s” cause overheads
- 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(C)

```
CC     = mpifccpx
OPTFLAGS= -Kfast,openmp -Nclang
LIBS =
LFLAGS=
#
TARGET = ../run/sol1
default: $(TARGET)
OBJS =¥
        test1.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 solve11.o solver_CG.o solver_SR.o¥
        output_ucd.o pfem_finalize.o allocate.o util.o

$(TARGET):  $(OBJS)
        $(CC) $(OPTFLAGS) -o $@ $(OBJS) $(LFLAGS)
.c.o:
        $(CC) $(OPTFLAGS) -c  $*.c
clean:
        /bin/rm -f *.o $(TARGET) *~ *.mod
```

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

C(solver_CG)

```
#pragma omp parallel for private (i)
    for(i=0;i<N;i++){
      X [i]    +=  ALPHA *WW[P][i];
      WW[R][i]+= -ALPHA *WW[Q][i];
    }
```

```
    DNRM20= 0.e0;
#pragma omp parallel for private (i) reduction (+:DNRM20)
    for(i=0;i<N;i++){
      DNRM20+=WW[R][i]*WW[R][i];
    }
```

```
#pragma omp parallel for private (j,i,k,WVAL)
    for( j=0;j<N;j++){
      WVAL= D[j] * WW[P][j];
      for(k=indexLU[j];k<indexLU[j+1];k++){
        i=itemLU[k];
        WVAL+= AMAT[k] * WW[P][i];
    }
    WW[Q][j]=WVAL;
```

solver_SR (send)

```
 for( neib=1;neib<=NEIBPETOT;neib++){
    istart=EXPORT_INDEX[neib-1];
    inum  =EXPORT_INDEX[neib]-istart;
#pragma omp parallel for private (k,ii)
    for( k=istart;k<istart+inum;k++){
      ii= EXPORT_ITEM[k];
      WS[k]= X[ii-1];
    }
    MPI_Isend(&WS[istart],inum,MPI_DOUBLE,
              NEIBPE[neib-1],0,MPI_COMM_WORLD,&req1[neib-1]);
  }
```

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
```

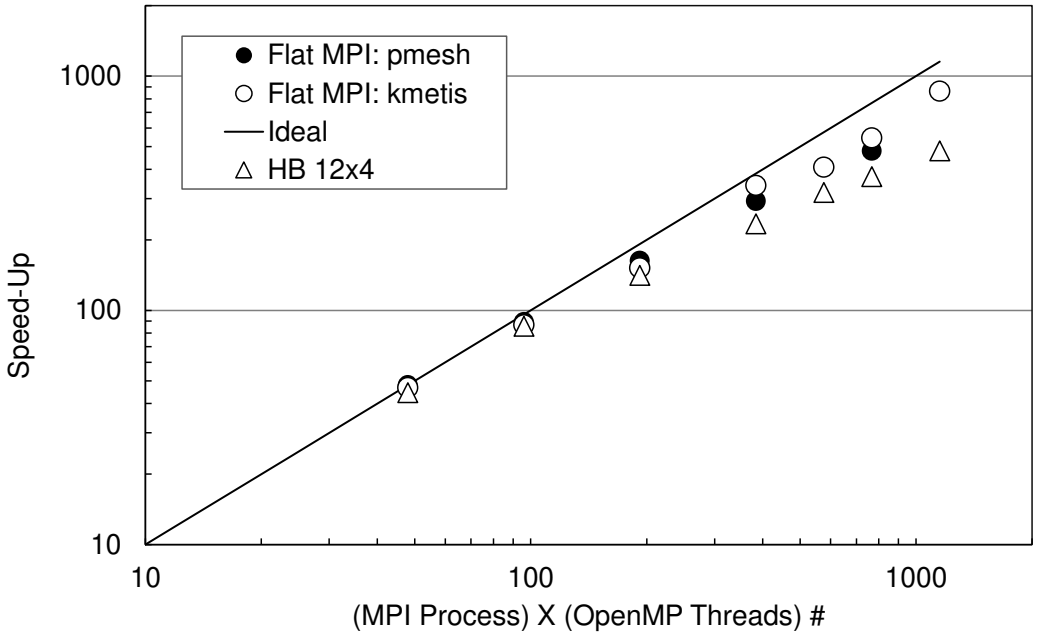
b08.sh

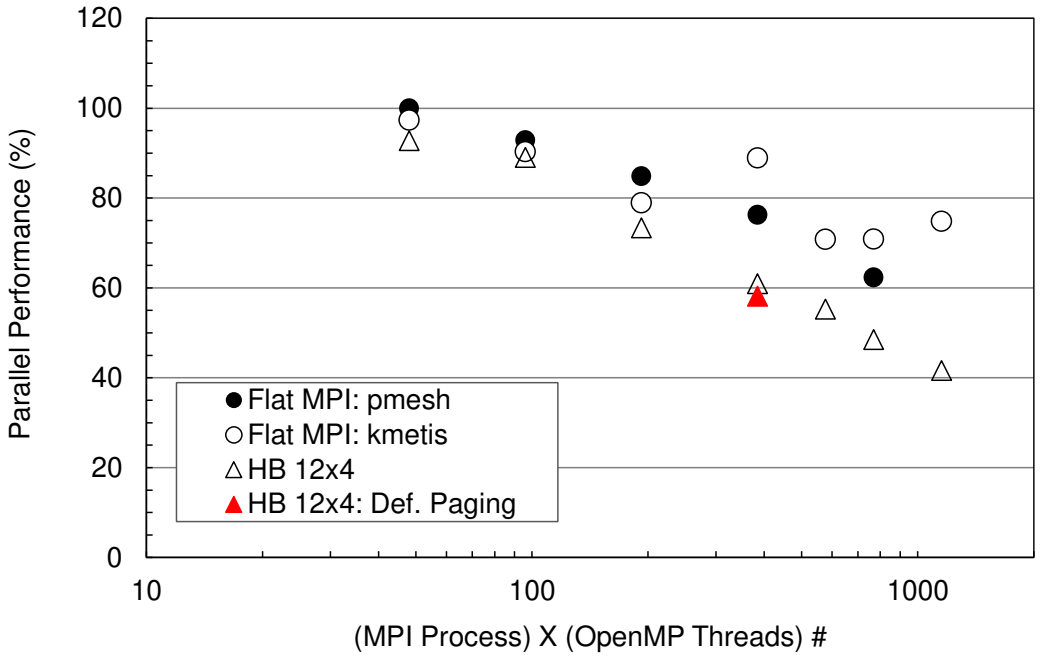
```
export XOS_MMM_L_PAGING_POLICY=
demand:demand:demand
```

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

Example: Strong Scaling: C

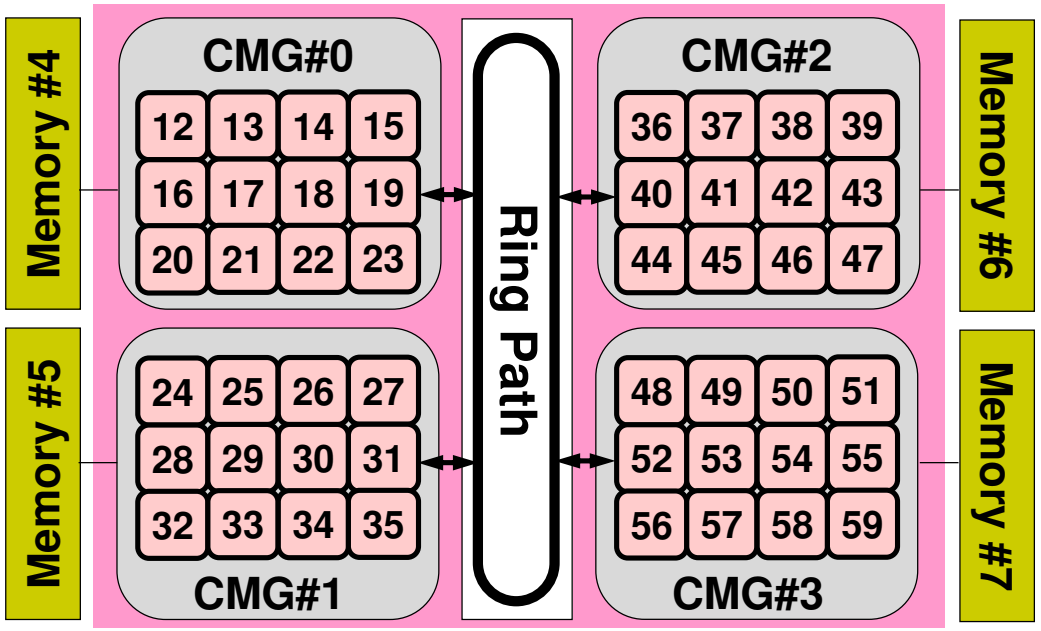
- 256×256×192 nodes, 12,582,912 DOF
- 48～1,152 cores (1~24 nodes)
- Linear Solver





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
```

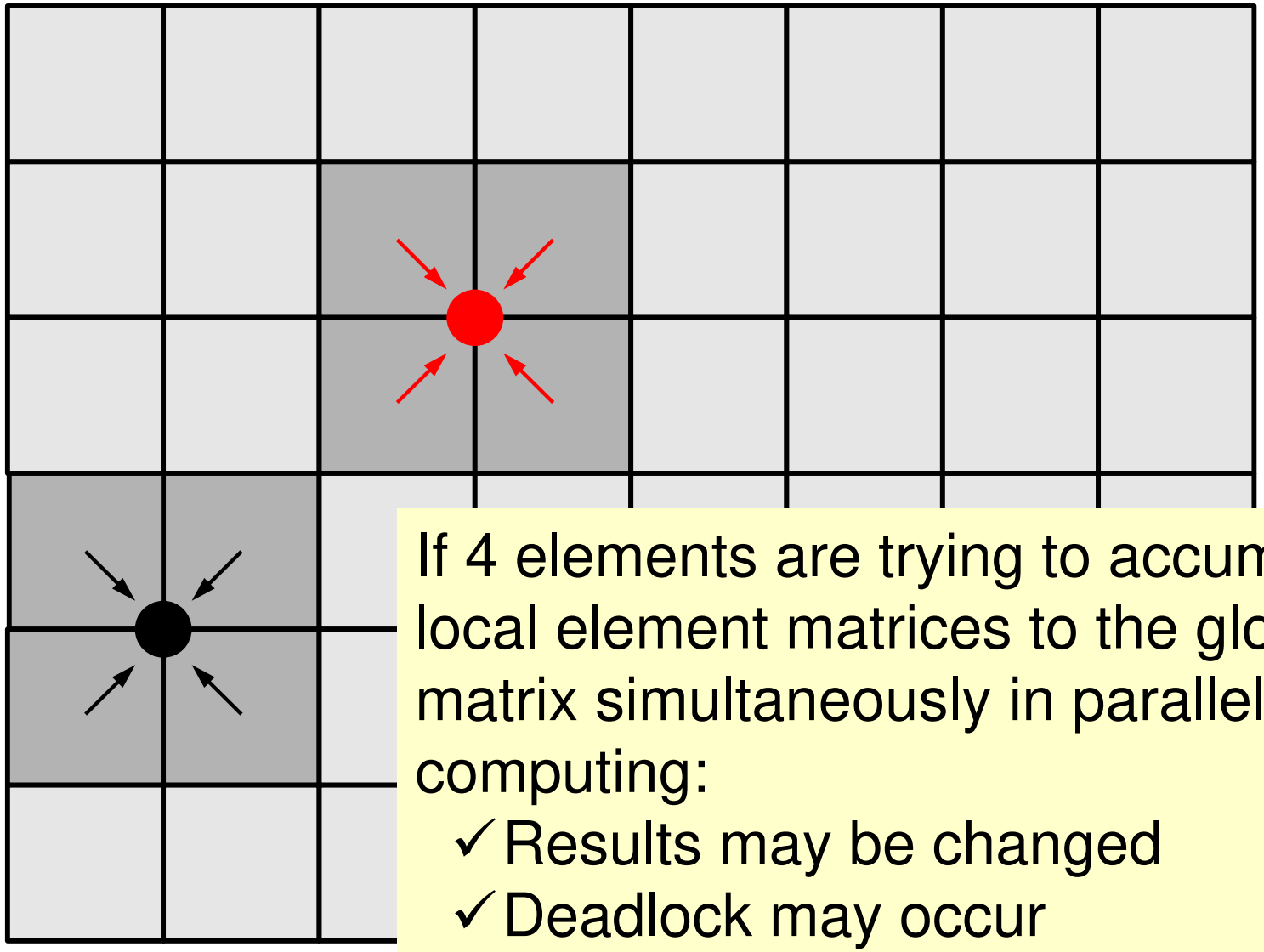
24-nodes
```
256 256 192
  8   4   3
pcube
```

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



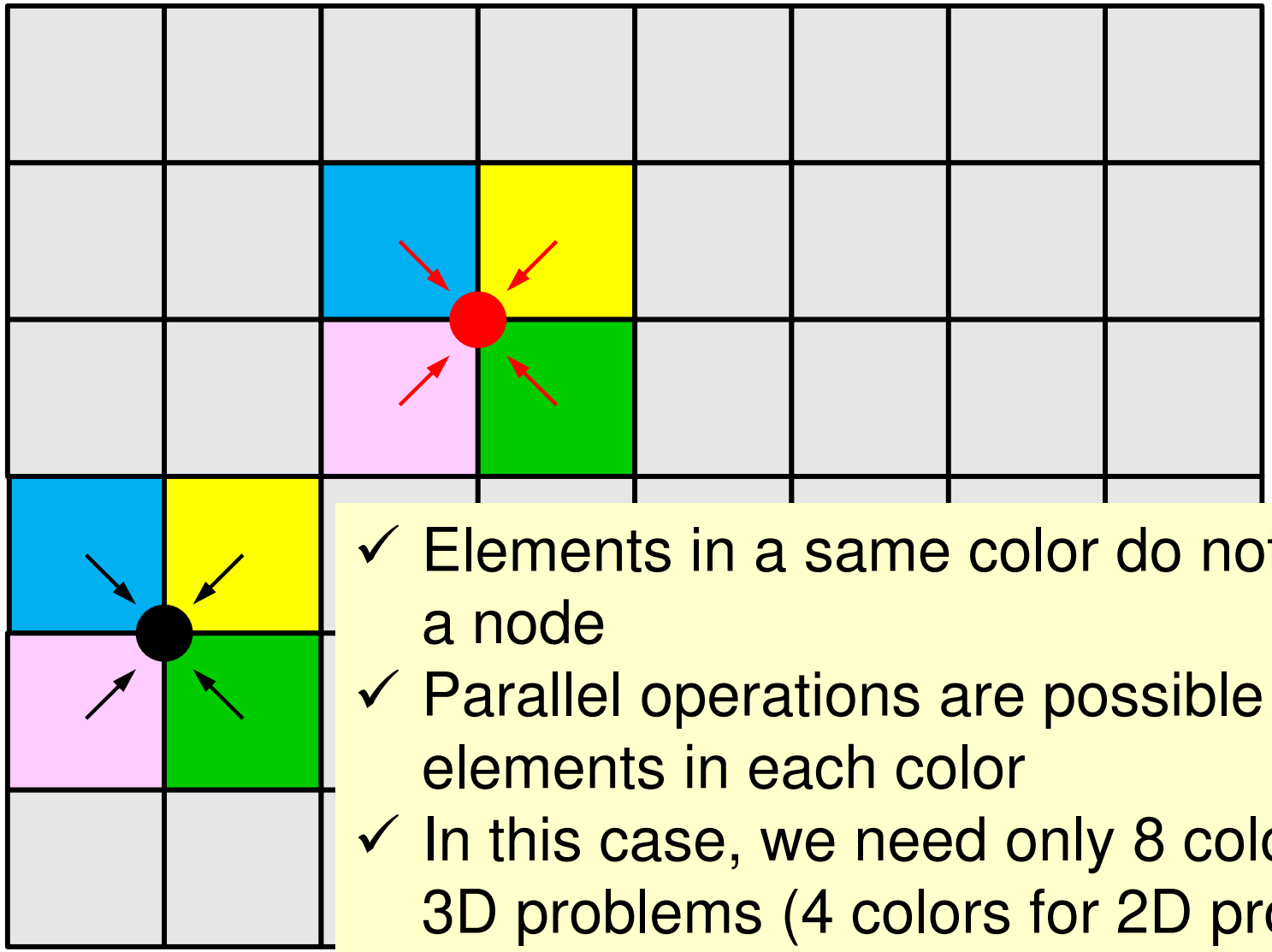
If 4 elements are trying to accumulate local element matrices to the global matrix simultaneously in parallel computing:

- ✓Results may be changed
- ✓Deadlock may occur

How to apply multi-threading

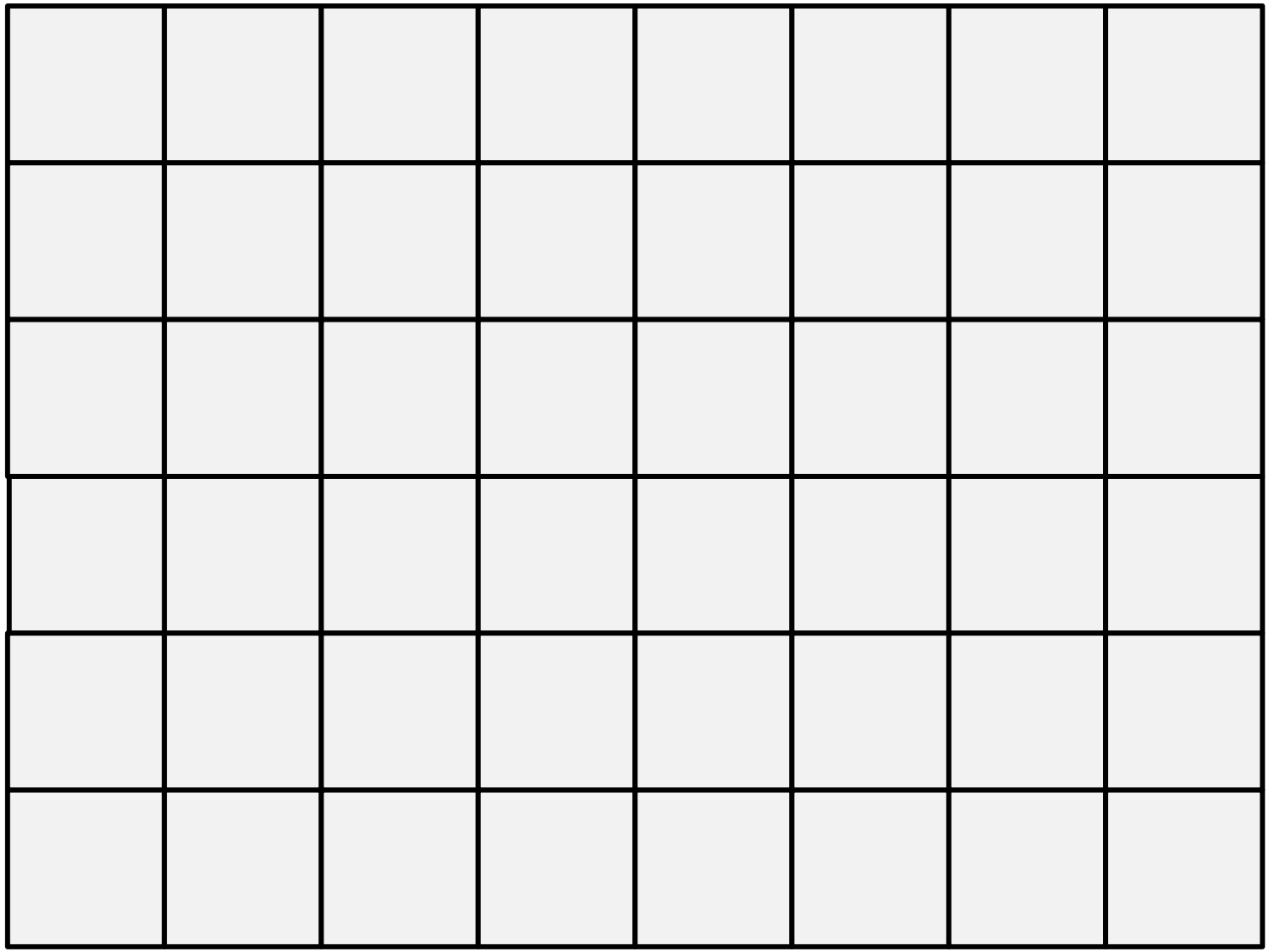
- CG Solver
  - Just insert OpenMP directives
  - ILU/IC preconditioning is much more difficult
- MAT_ASS (mat_ass_main, mat_ass_bc)
  - Coloring
    - Elements in a same color do not share a node
    - Parallel operations are possible for elements in each color
    - In this case, we need only 8 colors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

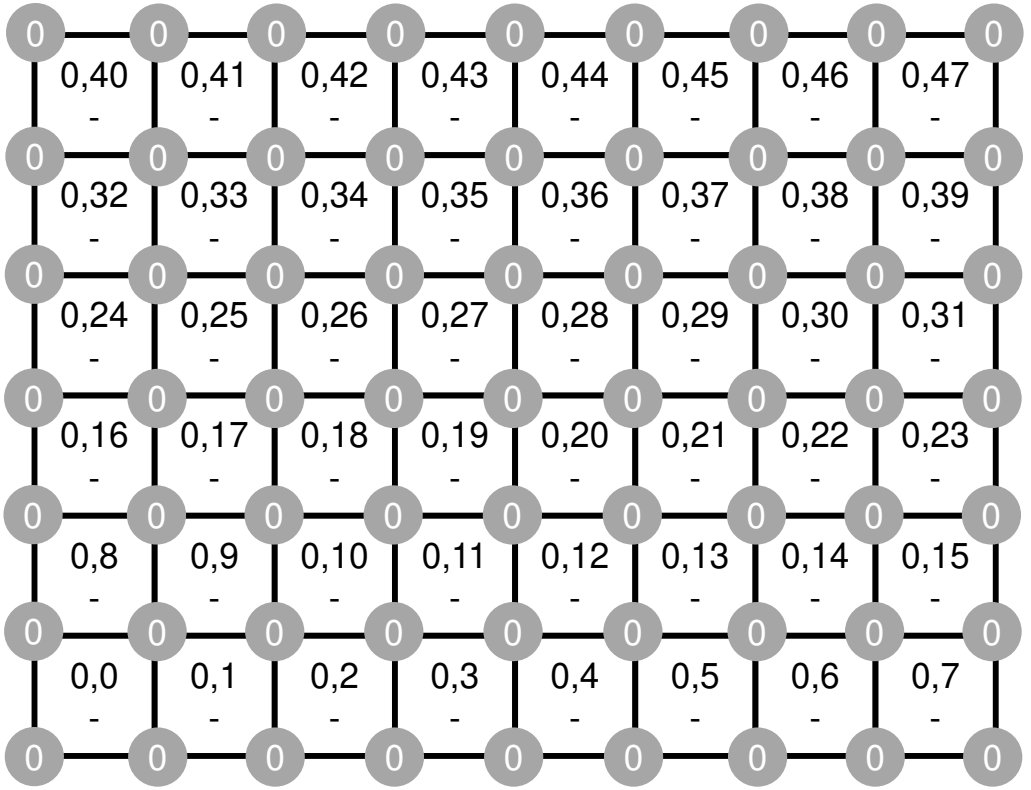
Target: 8x6= 48-meshes



Coloring (2D) (1/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

  for(icol=1;icol<NP;icol++) {
    for(i=0;i<NP;i++) {
      W1[i]=0;
    }
    for(icel=0;icel<ICELTOT;icel++) {
    if (W2[icel]== 0){
      in1=ICELNOD[icel][0]-1;
      in2=ICELNOD[icel][1]-1;
      in3=ICELNOD[icel][2]-1;
      in4=ICELNOD[icel][3]-1;
      ip1= W1[in1];
      ip2= W1[in2];
      ip3= W1[in3];
      ip4= W1[in4];

      isum= ip1+ip2+ip3+ip4;
      if (isum==0) {
        W3[icol]= icou + 1;
        W2[icel]= icol;
        ELMCOLORitem[icou]= icel;
        icou= icou + 1;

        W1[in1]= 1;
        W1[in2]= 1;
        W1[in3]= 1;
        W1[in4]= 1;
        if (icou==ICELTOT) goto expoint;
      }
    }
    }
  }

expoint:
  ELMCOLORtot= icol;
  W3[0]= 0;
  W3[ELMCOLORtot]= ICELTOT;

  for(icol=0;icol<ELMCOLORtot+1;icol++) {
    ELMCOLORindex[icol]= W3[icol];
  }
```

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



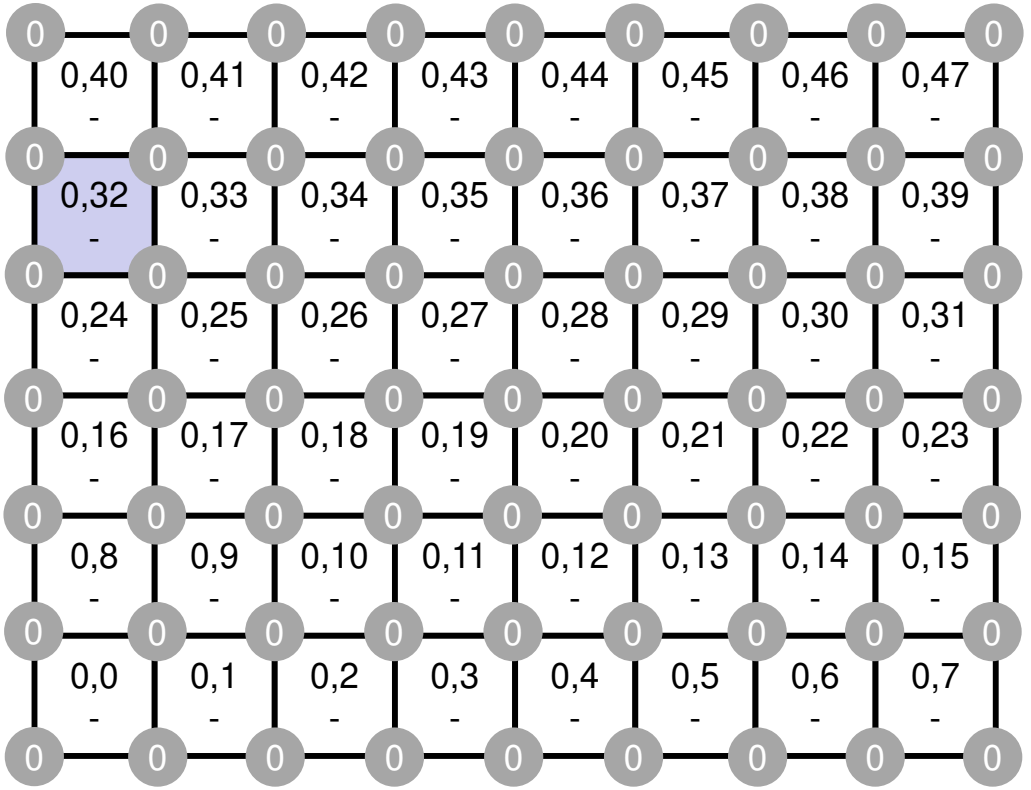
Coloring (2D) (1/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

  for(icol=1;icol<NP;icol++) {
    for(i=0;i<NP;i++) {
      W1[i]=0;
    }
    for(icel=0;icel<ICELTOT;icel++) {
      in= W2[icel];
      if(W1[in]==0) {
        icou= icou + 1;
        W1[in]= 1;
        IDold[icou]= icel;
        IDnew[icel]= icou;
      }
    }
    W3[icol]= icou;
  }

expoint:
  ELMCOLORtot= icol;
  W3[0]= 0;
  W3[ELMCOLORtot]= ICELTOT;

  for(icol=0;icol<ELMCOLORtot+1;icol++) {
    ELMCOLORindex[icol]= W3[icol];
  }
```

W2[icel], IDold
IDnew

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

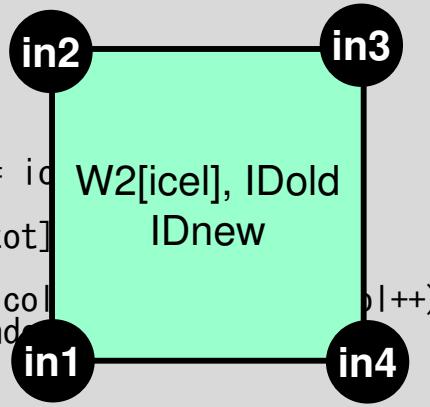
Name	Size	Content
ELMCOLORindex	**NP+1**	Element # in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**NP+1**	Accumulated # of Colored Elem's in each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



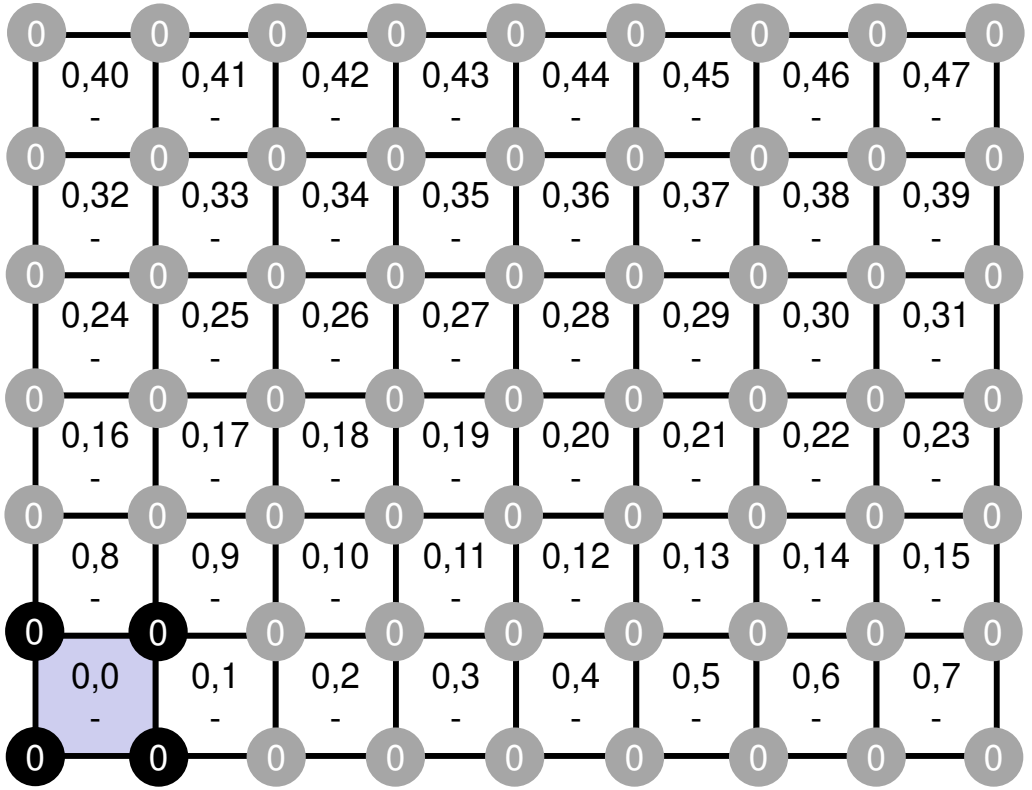
Coloring (2D) (2/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

for(icol=1;icol<NP;icol++) {
  for(i=0;i<NP;i++) {
    W1[i]=0;
  }
  for(icel=0;icel<ICELTOT;icel++) {
  if (W2[icel]== 0){
    in1=ICELNOD[icel][0]-1;
    in2=ICELNOD[icel][1]-1;
    in3=ICELNOD[icel][2]-1;
    in4=ICELNOD[icel][3]-1;
    ip1=W1[in1];        (=0)
    ip2=W1[in2];        (=0)
    ip3=W1[in3];        (=0)
    ip4=W1[in4];        (=0)

    isum= ip1+ip2+ip3+ip4;
    if (isum==0) {
      W3[icol]= icou + 1;
      W2[icel]= icol;
      ELMCOLORitem[icou]= icel;
      icou= icou + 1;

      W1[in1]= 1;
      W1[in2]= 1;
      W1[in3]= 1;
      W1[in4]= 1;
      if (icou==ICELTOT) goto expoint;
    }
  }
  }
}

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=1
icel=0



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

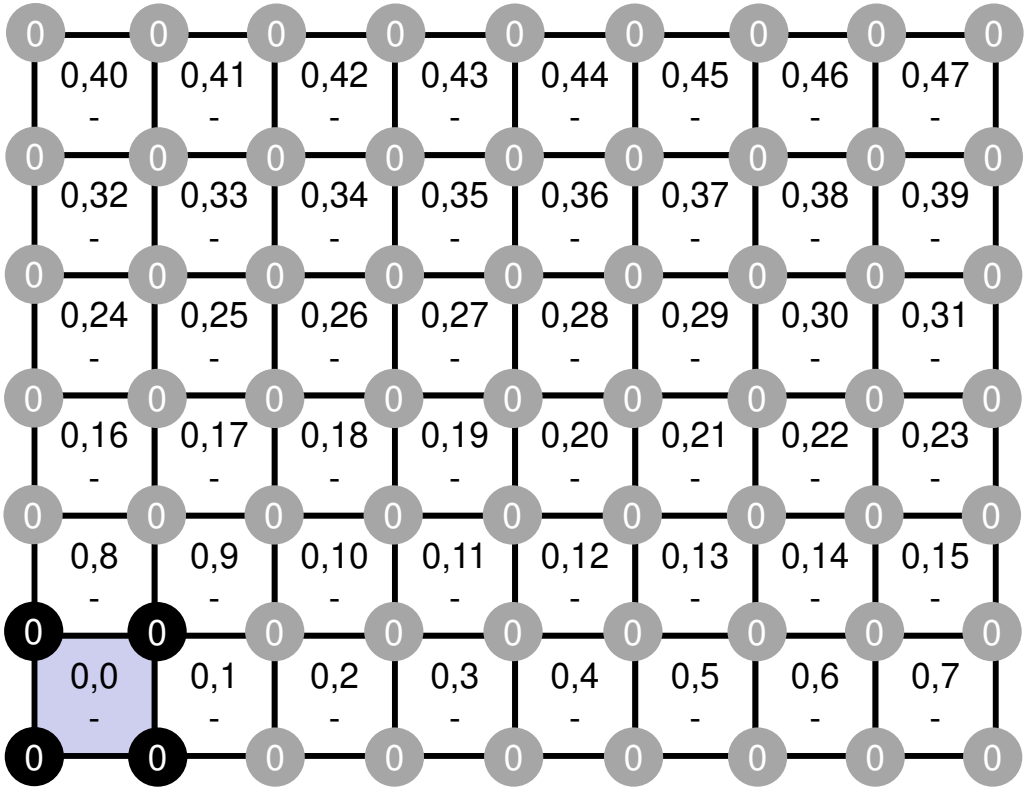


Coloring (2D) (2/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

for(icol=1;icol<NP;icol++) {
  for(i=0;i<NP;i++) {
    W1[i]=0;
  }
  for(icel=0;icel<ICELTOT;icel++) {
  if (W2[icel]== 0){
    in1=ICELNOD[icel][0]-1;
    in2=ICELNOD[icel][1]-1;
    in3=ICELNOD[icel][2]-1;
    in4=ICELNOD[icel][3]-1;
    ip1=W1[in1];       (=0)
    ip2=W1[in2];       (=0)
    ip3=W1[in3];       (=0)
    ip4=W1[in4];       (=0)

    isum= ip1+ip2+ip3+ip4;
    if (isum==0) {
      W3[icol]= icou + 1;
      W2[icel]= icol;
      ELMCOLORitem[icou]= icel;
      icou= icou + 1;

      W1[in1]= 1;
      W1[in2]= 1;
      W1[in3]= 1;
      W1[in4]= 1;
      if (icou==ICELTOT) goto expoint;
    }
  }
  }
}

expoint:
  ELMCOLORtot= icol;
  W3[0]= 0;
  W3[ELMCOLORtot]= ICELTOT;

  for(icol=0;icol<ELMCOLORtot+1;icol++) {
    ELMCOLORindex[icol]= W3[icol];
  }
```

icol=1
icel=0

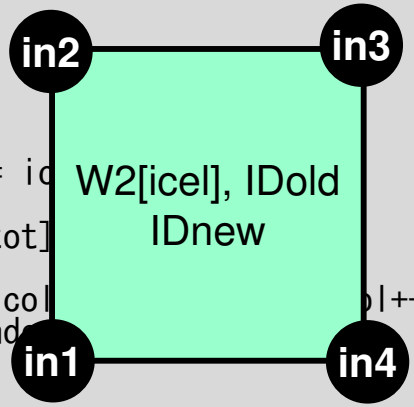
Name	Size	Content
ELMCOLORindex	**NP+1**	Element # in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**NP+1**	Accumulated # of Colored Elem's in each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



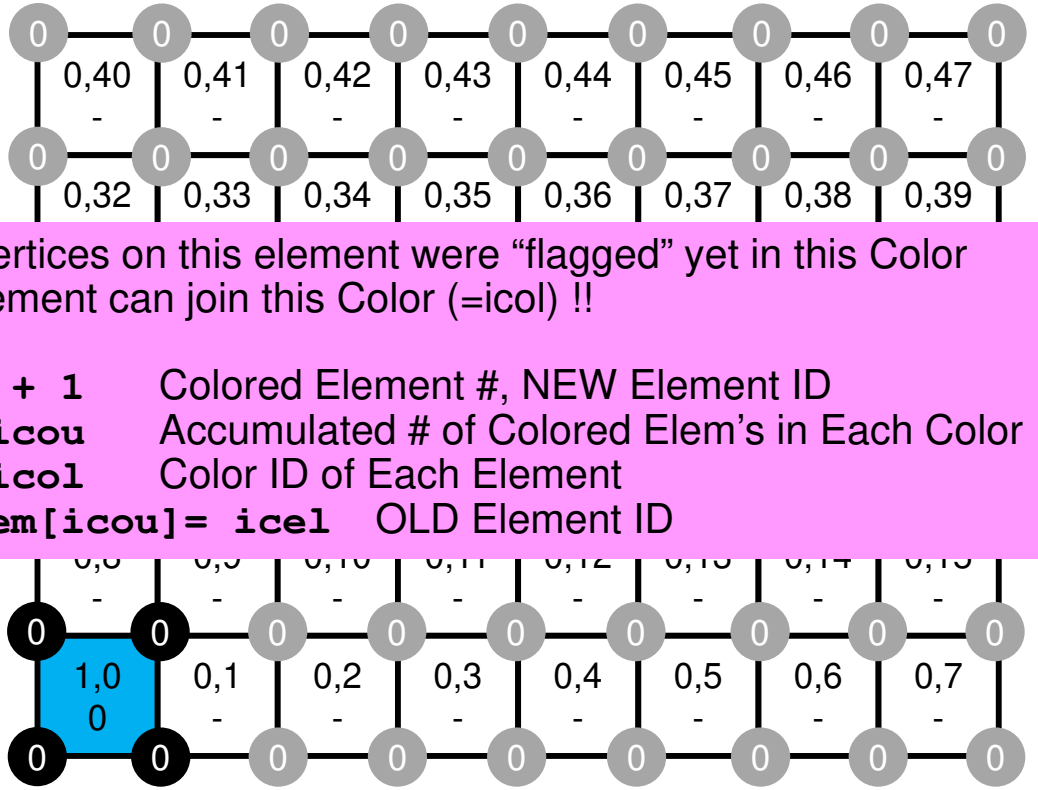
Coloring (2D) (2/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

 for(icol=1;icol<NP;icol++) {
   for(i=0;i<NP;i++) {
     W1[i]=0;
   }
   for(icel=0;icel<ICELTOT;icel++) {
   if (W2[icel]== 0){
     in1=ICELNOD[icel][0]-1;
     in2=ICELNOD[icel][1]-1;
     in3=ICELNOD[icel][2]-1;
     in4=ICELNOD[icel][3]-1;
     ip1=W1[in1];        (=0)
     ip2=W1[in2];        (=0)
     ip3=W1[in3];        (=0)
     ip4=W1[in4];        (=0)

     isum= ip1+ip2+ip3+ip4;
     if (isum==0) {
       W3[icol]= icou + 1;
       W2[icel]= icol;
       ELMCOLORitem[icou]= icel;
       icou= icou + 1;

       W1[in1]= 1;
       W1[in2]= 1;
       W1[in3]= 1;
       W1[in4]= 1;
       if (icou==ICELTOT) goto expoin
     }
   }
 }

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=1
icel=0



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



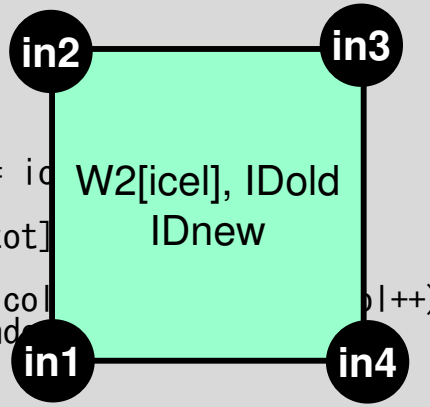
Because no vertices on this element were "flagged" yet in this Color (=icol), this element can join this Color (=icol) !!

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

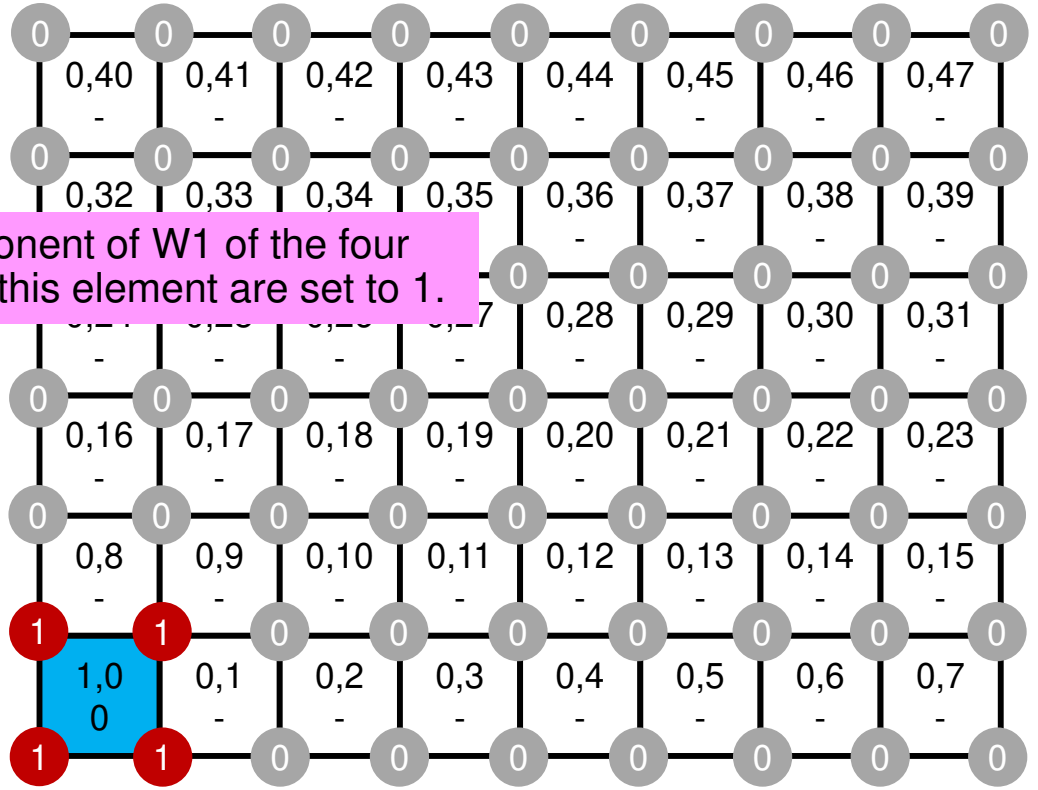
Coloring (2D) (2/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

 for(icol=1;icol<NP;icol++) {
   for(i=0;i<NP;i++) {
     W1[i]=0;
   }
   for(icel=0;icel<ICELTOT;icel++) {
   if (W2[icel]== 0){
     in1=ICELNOD[icel][0]-1;
     in2=ICELNOD[icel][1]-1;
     in3=ICELNOD[icel][2]-1;
     in4=ICELNOD[icel][3]-1;
     ip1=W1[in1];
     ip2=W1[in2];
     ip3=W1[in3];
     ip4=W1[in4];

     isum= ip1+ip2+ip3+ip4;
     if (isum==0) {
       W3[icol]= icou + 1;
       W2[icel]= icol;
       ELMCOLORitem[icou]= icel;
       icou= icou + 1;

       W1[in1]= 1;
       W1[in2]= 1;
       W1[in3]= 1;
       W1[in4]= 1;
       if (icou==ICELTOT) goto expoint;
     }
   }
   }
 }

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=1
icel=0



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



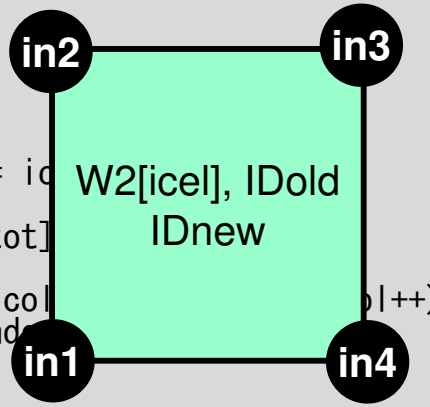
Each component of W1 of the four vertices on this element are set to 1.

Coloring (2D) (3/7)

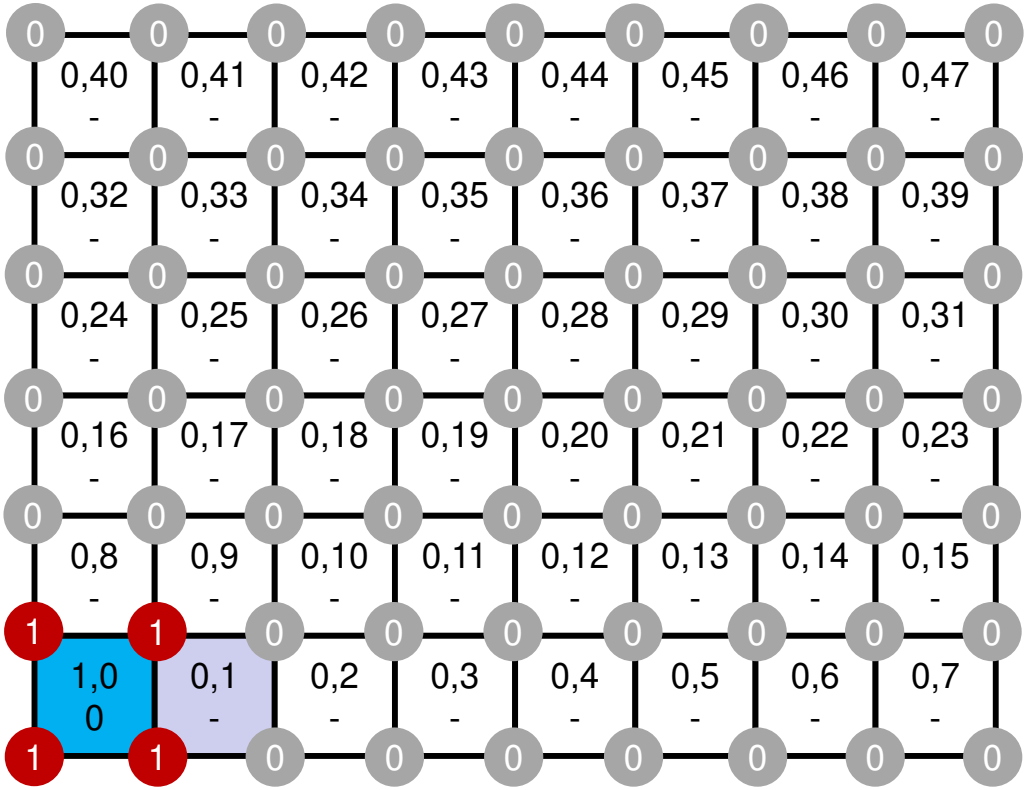
```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

for(icol=1;icol<NP;icol++) {
  for(i=0;i<NP;i++) {
    W1[i]=0;
  }
  for(icel=0;icel<ICELTOT;icel++) {
  if (W2[icel]== 0){
    in1=ICELNOD[icel][0]-1;
    in2=ICELNOD[icel][1]-1;
    in3=ICELNOD[icel][2]-1;
    in4=ICELNOD[icel][3]-1;
    ip1=W1[in1];        (=1)
    ip2=W1[in2];        (=1)
    ip3=W1[in3];        (=0)
    ip4=W1[in4];        (=0)

    isum= ip1+ip2+ip3+ip4; (=2)
    if (isum==0) {
      if (icou==ICELTOT) goto expoint;
    }
  }
  }
}

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=1
icel=1

- ✓ 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=1) cannot join this color (icol=1)



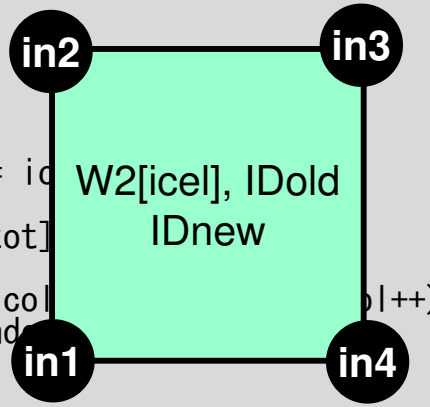
Name	Size	Content
ELMCOLORindex	**NP+1**	Element # in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**NP+1**	Accumulated # of Colored Elem’s in each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



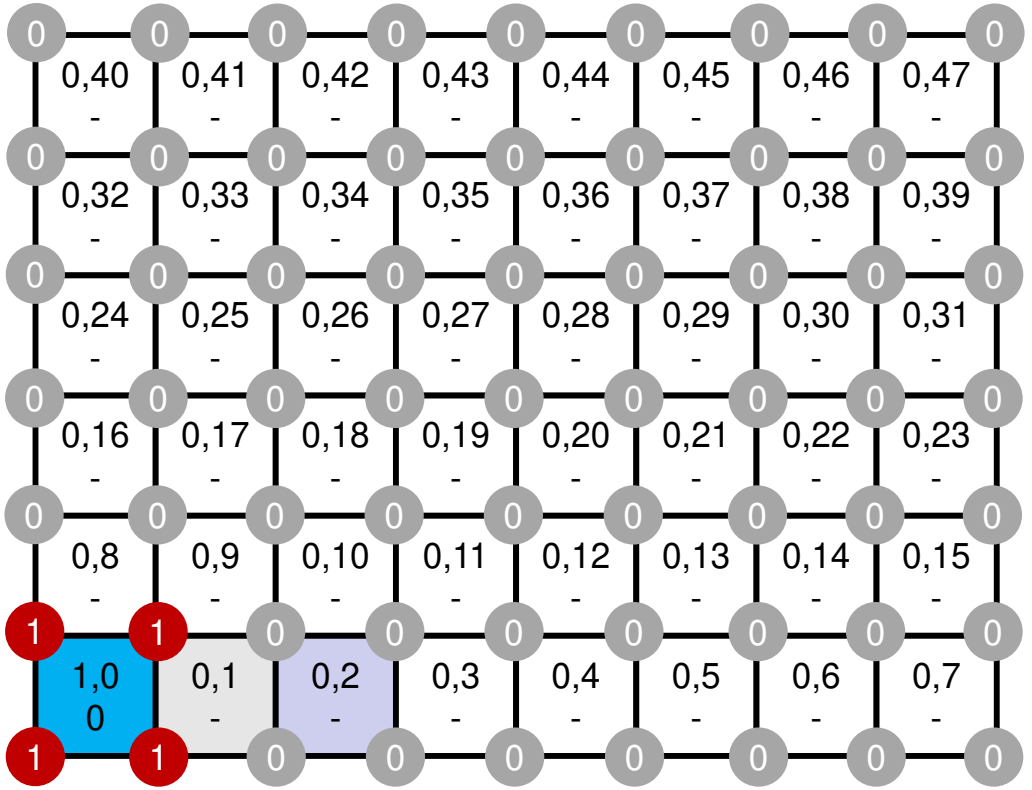
Coloring (2D) (4/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

for(icol=1;icol<NP;icol++) {
  for(i=0;i<NP;i++) {
    W1[i]=0;
  }
  for(icel=0;icel<ICELTOT;icel++) {
  if (W2[icel]== 0){
    in1=ICELNOD[icel][0]-1;
    in2=ICELNOD[icel][1]-1;
    in3=ICELNOD[icel][2]-1;
    in4=ICELNOD[icel][3]-1;
    ip1=W1[in1];        (=0)
    ip2=W1[in2];        (=0)
    ip3=W1[in3];        (=0)
    ip4=W1[in4];        (=0)

    isum= ip1+ip2+ip3+ip4; (=0)
    if (isum==0) {
      W3[icol]= icou + 1;
      W2[icel]= icol;
      ELMCOLORitem[icou]= icel;
      icou= icou + 1;

      W1[in1]= 1;
      W1[in2]= 1;
      W1[in3]= 1;
      W1[in4]= 1;
      if (icou==ICELTOT) goto expoint;
    }
  }
  }
}

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=1
icel=2



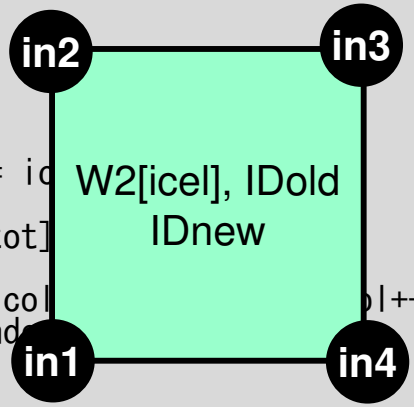
Name	Size	Content
ELMCOLORindex	**NP+1**	Element # in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**NP+1**	Accumulated # of Colored Elem's in each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



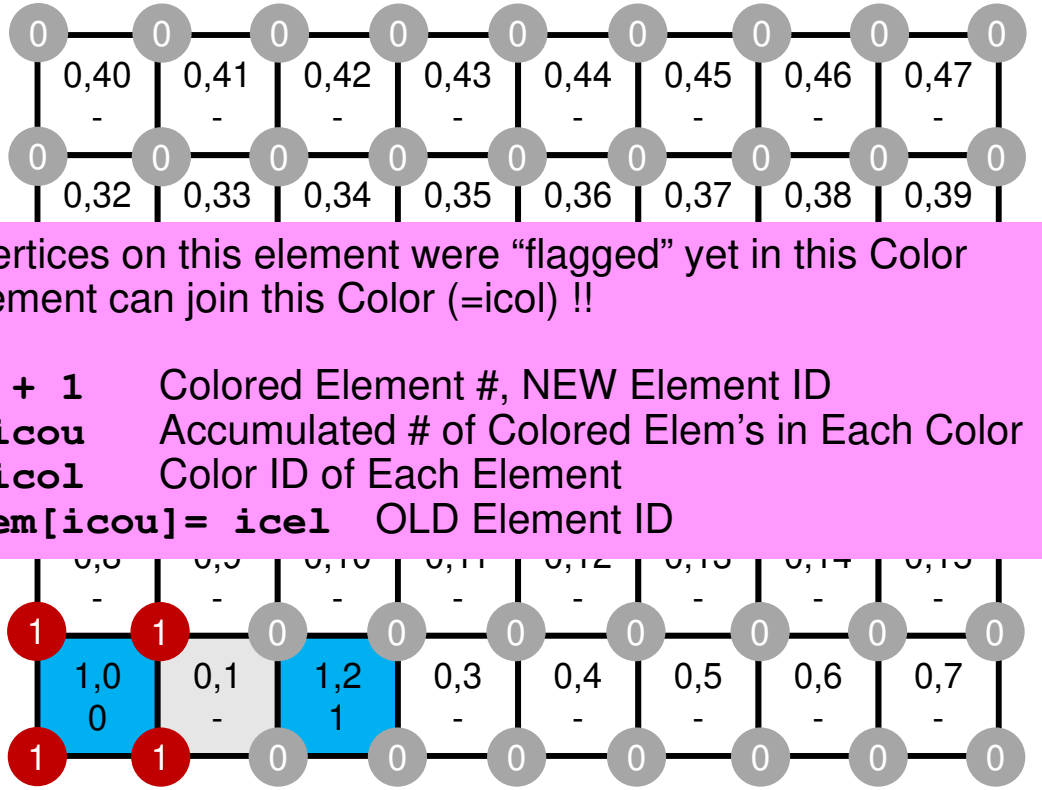
Coloring (2D) (4/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

 for(icol=1;icol<NP;icol++) {
   for(i=0;i<NP;i++) {
     W1[i]=0;
   }
   for(icel=0;icel<ICELTOT;icel++) {
   if (W2[icel]== 0){
     in1=ICELNOD[icel][0]-1;
     in2=ICELNOD[icel][1]-1;
     in3=ICELNOD[icel][2]-1;
     in4=ICELNOD[icel][3]-1;
     ip1=W1[in1];        (=0)
     ip2=W1[in2];        (=0)
     ip3=W1[in3];        (=0)
     ip4=W1[in4];        (=0)

     isum= ip1+ip2+ip3+ip4; (=0)
     if (isum==0) {
       W3[icol]= icou + 1;
       W2[icel]= icol;
       ELMCOLORitem[icou]= icel;
       icou= icou + 1;

       W1[in1]= 1;
       W1[in2]= 1;
       W1[in3]= 1;
       W1[in4]= 1;
       if (icou==ICELTOT) goto expoin
     }
   }
 }

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=1
icel=2



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



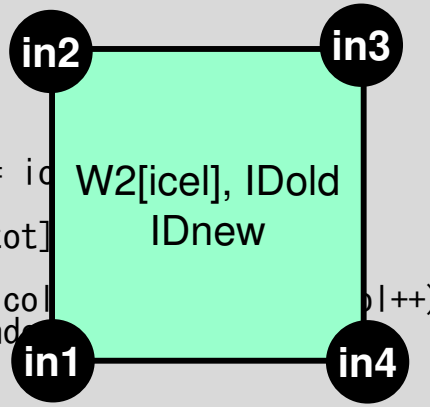
Because no vertices on this element were "flagged" yet in this Color (=icol), this element can join this Color (=icol) !!

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)

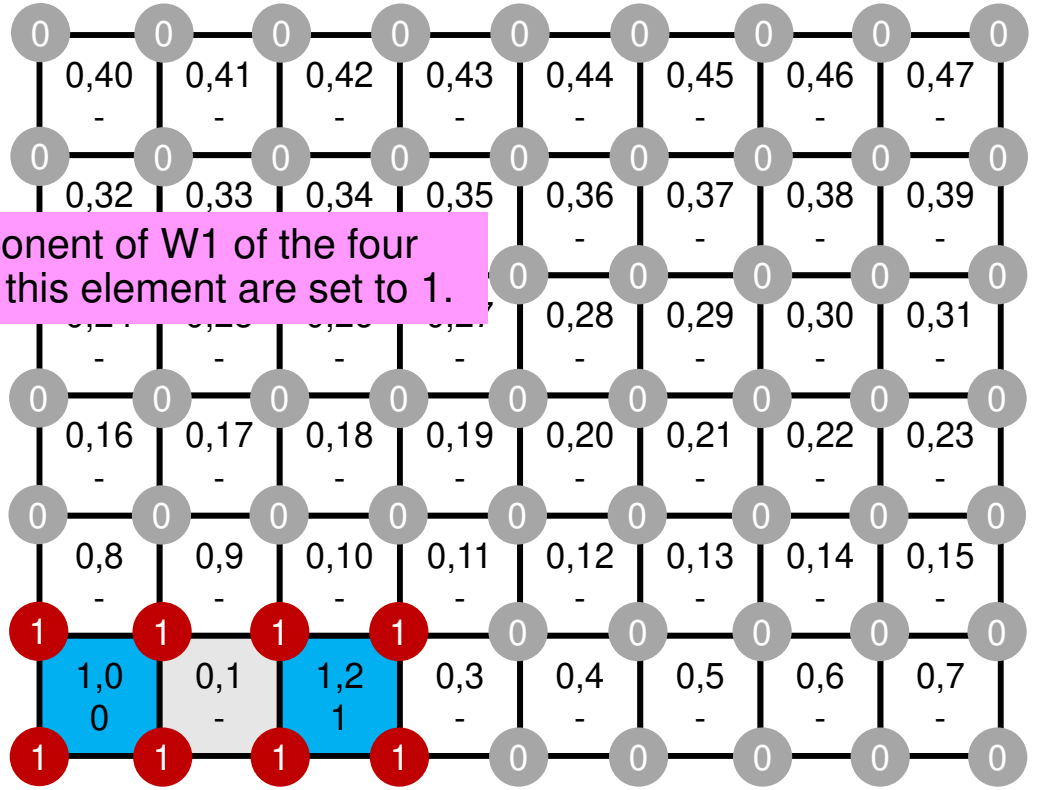
```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

 for(icol=1;icol<NP;icol++) {
   for(i=0;i<NP;i++) {
     W1[i]=0;
   }
   for(icel=0;icel<ICELTOT;icel++) {
   if (W2[icel]== 0){
     in1=ICELNOD[icel][0]-1;
     in2=ICELNOD[icel][1]-1;
     in3=ICELNOD[icel][2]-1;
     in4=ICELNOD[icel][3]-1;
     ip1=W1[in1];
     ip2=W1[in2];
     ip3=W1[in3];
     ip4=W1[in4];

     isum= ip1+ip2+ip3+ip4; (=0)
     if (isum==0) {
       W3[icol]= icou + 1;
       W2[icel]= icol;
       ELMCOLORitem[icou]= icel;
       icou= icou + 1;

       W1[in1]= 1;
       W1[in2]= 1;
       W1[in3]= 1;
       W1[in4]= 1;
       if (icou==ICELTOT) goto expoint;
     }
   }
   }
 }

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=1
icel=2



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

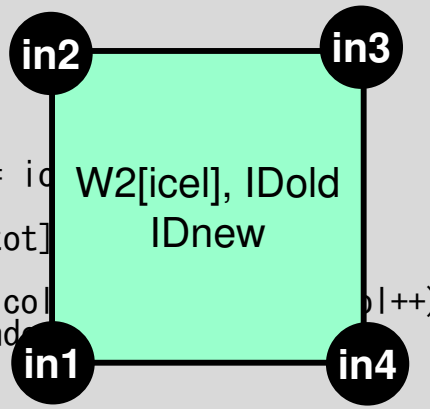
Each component of W1 of the four vertices on this element are set to 1.



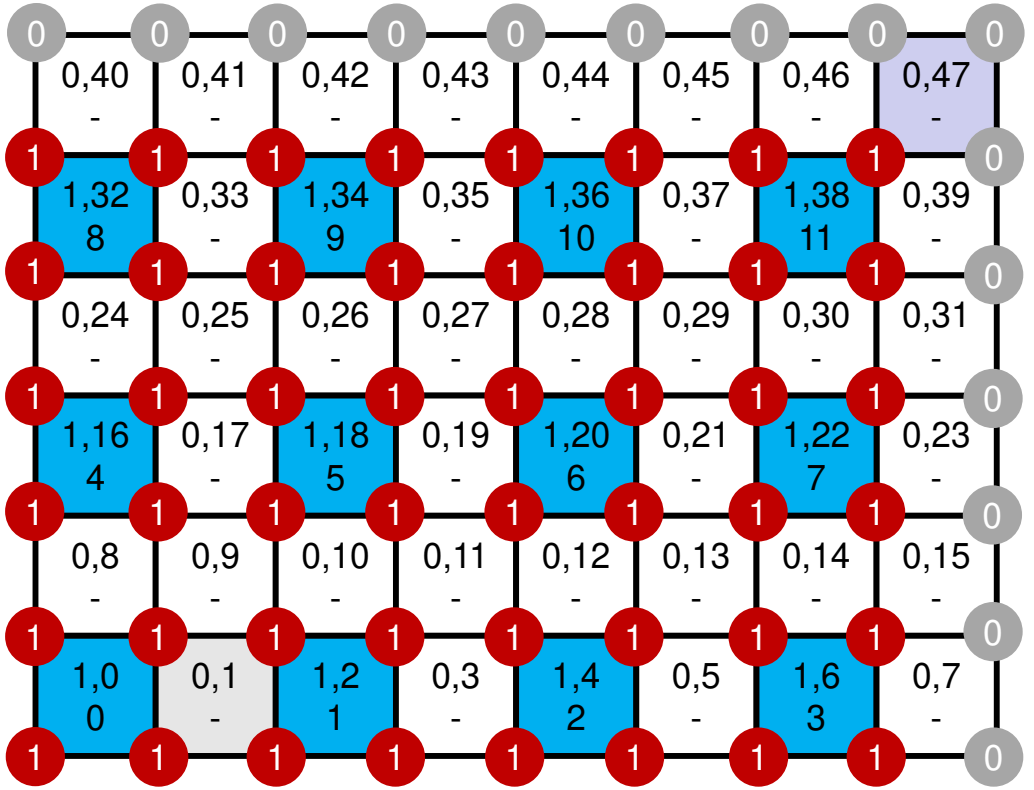
Coloring (2D) (5/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1]
W1[NP], W2[ICELTOT]
W1=0; W2=0; W3=0

icou= 0;

for(icol=1;icol<NP;icol++) {
  for(i=0;i<NP;i++) {
    W1[i]=0;
  }
  for(icel=0;icel<ICELTOT;icel++) {
  if (W2[icel]== 0){
    in1=ICELNOD[icel][0]-1;
    in2=ICELNOD[icel][1]-1;
    in3=ICELNOD[icel][2]-1;
    in4=ICELNOD[icel][3]-1;
    ip1=W1[in1];        (=1)
    ip2=W1[in2];        (=0)
    ip3=W1[in3];        (=0)
    ip4=W1[in4];        (=0)

    isum= ip1+ip2+ip3+ip4; (=1)
    if (isum==0) {
      W3[icol]= icou + 1;
      W2[icel]= icol;
      ELMCOLORitem[icou]= icel;
      icou= icou + 1;

      W1[in1]= 1;
      W1[in2]= 1;
      W1[in3]= 1;
      W1[in4]= 1;
      if (icou==ICELTOT) goto expoint;
    }
  }
  }
}

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=1
icel=ICELTOT-1 (=47)



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



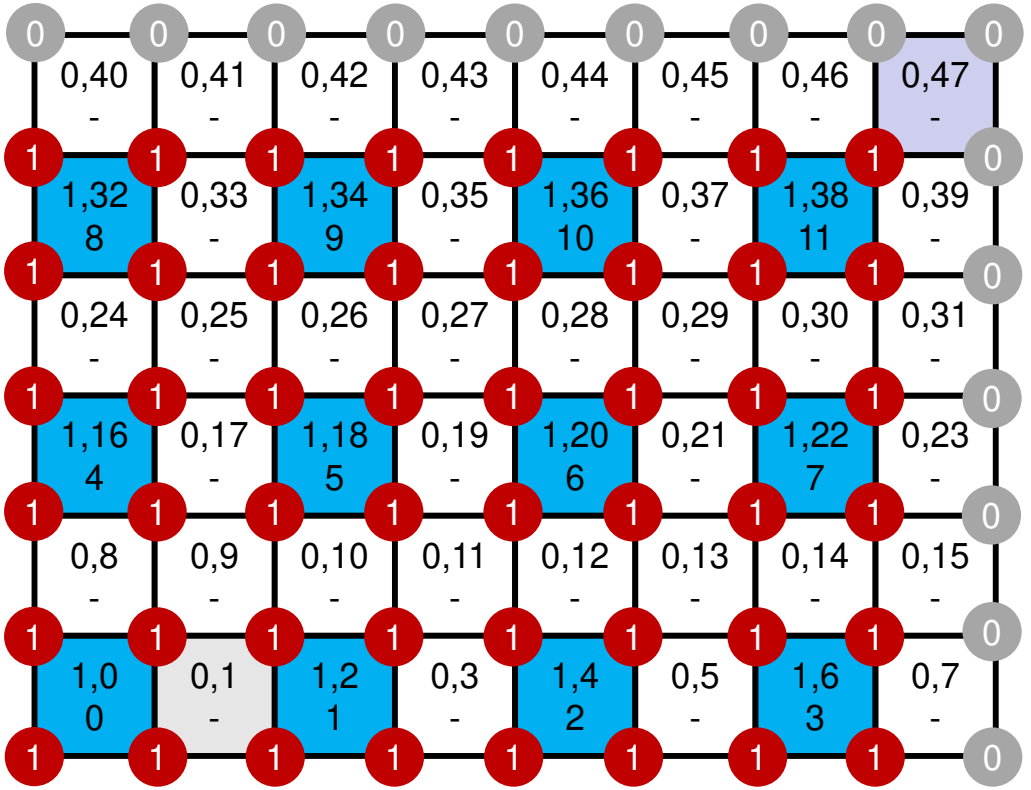
Coloring (2D) (5/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1
W1[NP], W2[ICELTO
W1=0; W2=0; W3=0

icou= 0;

for(icol=1;icol<NP;icol++) {
  for(i=0;i<NP;i++) {
    W1[i]=0;
  }
for(icel=0;icel<ICELTOT;icel++) {
if (W2[icel]== 0){
  in1=ICELNOD[icel][0]-1;
  in2=ICELNOD[icel][1]-1;
  in3=ICELNOD[icel][2]-1;
  in4=ICELNOD[icel][3]-1;
  ip1=W1[in1];        (=1)
  ip2=W1[in2];        (=0)
  ip3=W1[in3];        (=0)
  ip4=W1[in4];        (=0)

  isum= ip1+ip2+ip3+ip4; (=1)
  if (isum==0) {
    W3[icol]= icou + 1;
    W2[icel]= icol;
    ELMCOLORitem[icou]= icel;
    icou= icou + 1;

    W1[in1]= 1;
    W1[in2]= 1;
    W1[in3]= 1;
    W1[in4]= 1;
    if (icou==ICELTOT) goto expoint;
  }
}
}
}

expoi
  ELM
  W3[
  W3[

  for
    E
  }
```

icol=1
icel=ICELTOT-1 (=47)

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

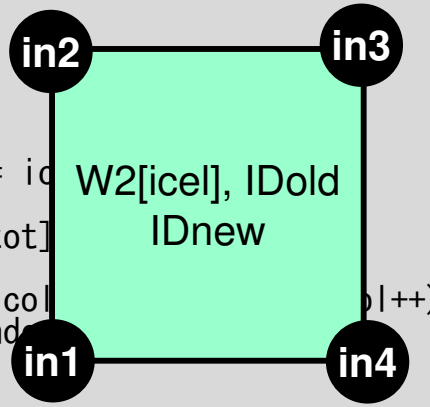
Name	Size	Content
ELMCOLORindex	**NP+1**	Element # in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**NP+1**	Accumulated # of Colored Elem's in each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



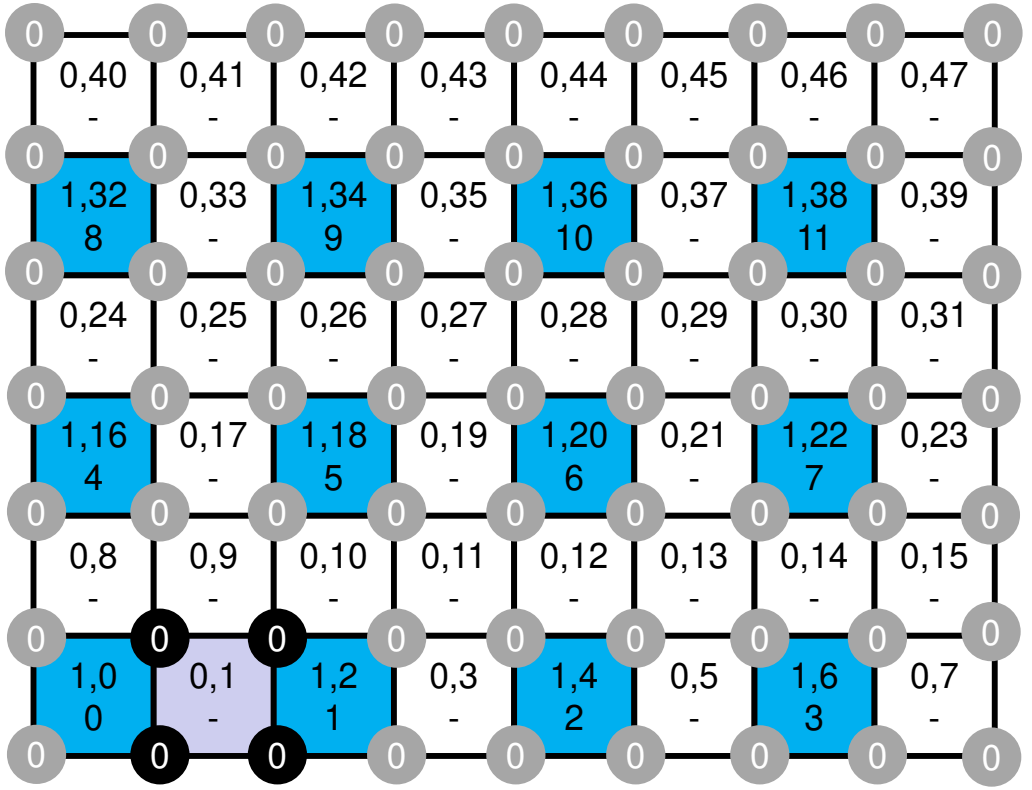
Coloring (2D) (6/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

for(icol=1;icol<NP;icol++) {
  for(i=0;i<NP;i++) {
    W1[i]=0;
  }
  for(icel=0;icel<ICELTOT;icel++) {
  if (W2[icel]== 0){
    in1=ICELNOD[icel][0]-1;
    in2=ICELNOD[icel][1]-1;
    in3=ICELNOD[icel][2]-1;
    in4=ICELNOD[icel][3]-1;
    ip1=W1[in1];        (=0)
    ip2=W1[in2];        (=0)
    ip3=W1[in3];        (=0)
    ip4=W1[in4];        (=0)

    isum= ip1+ip2+ip3+ip4;
    if (isum==0) {
      W3[icol]= icou + 1;
      W2[icel]= icol;
      ELMCOLORitem[icou]= icel;
      icou= icou + 1;

      W1[in1]= 1;
      W1[in2]= 1;
      W1[in3]= 1;
      W1[in4]= 1;
      if (icou==ICELTOT) goto expoint;
    }
  }
  }
}

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=2
Icel=1



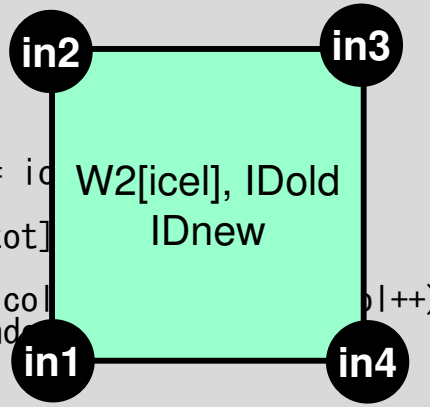
Name	Size	Content
ELMCOLORindex	**NP+1**	Element # in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**NP+1**	Accumulated # of Colored Elem's in each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



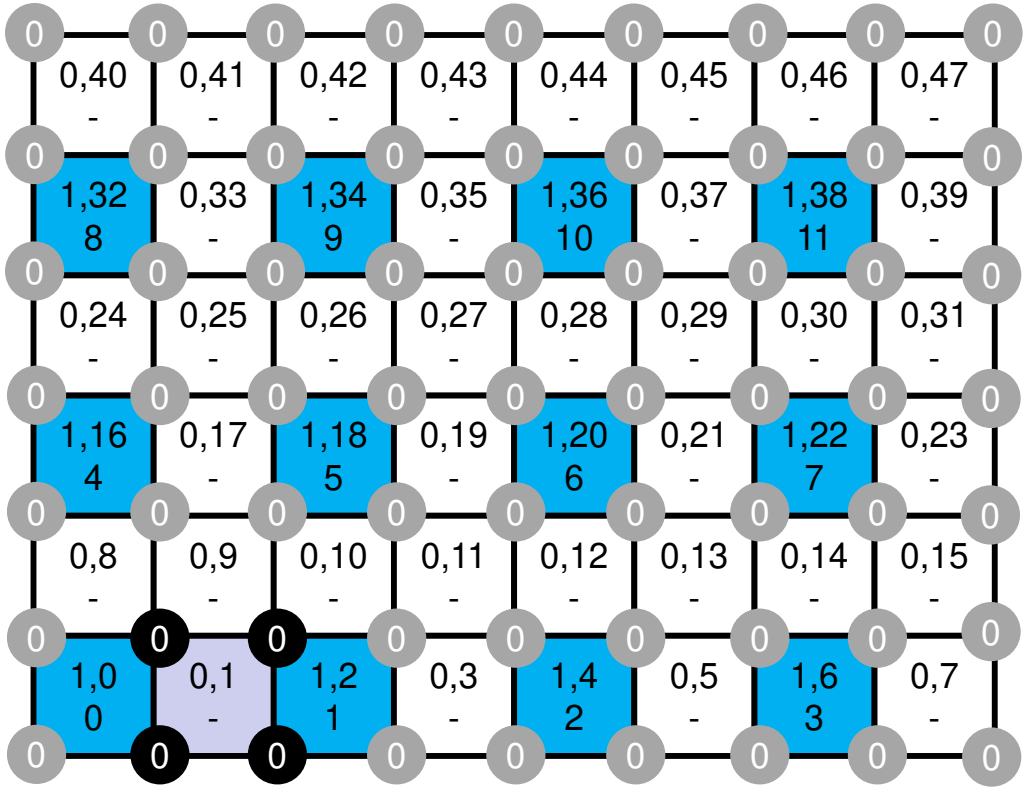
Coloring (2D) (6/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

for(icol=1;icol<NP;icol++) {
  for(i=0;i<NP;i++) {
    W1[i]=0;
  }
  for(icel=0;icel<ICELTOT;icel++) {
  if (W2[icel]== 0){
    in1=ICELNOD[icel][0]-1;
    in2=ICELNOD[icel][1]-1;
    in3=ICELNOD[icel][2]-1;
    in4=ICELNOD[icel][3]-1;
    ip1=W1[in1];        (=0)
    ip2=W1[in2];        (=0)
    ip3=W1[in3];        (=0)
    ip4=W1[in4];        (=0)

    isum= ip1+ip2+ip3+ip4; (=0)
    if (isum==0) {
      W3[icol]= icou + 1;
      W2[icel]= icol;
      ELMCOLORitem[icou]= icel;
      icou= icou + 1;

      W1[in1]= 1;
      W1[in2]= 1;
      W1[in3]= 1;
      W1[in4]= 1;
      if (icou==ICELTOT) goto expoint;
    }
  }
  }
}

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=2
Icel=1



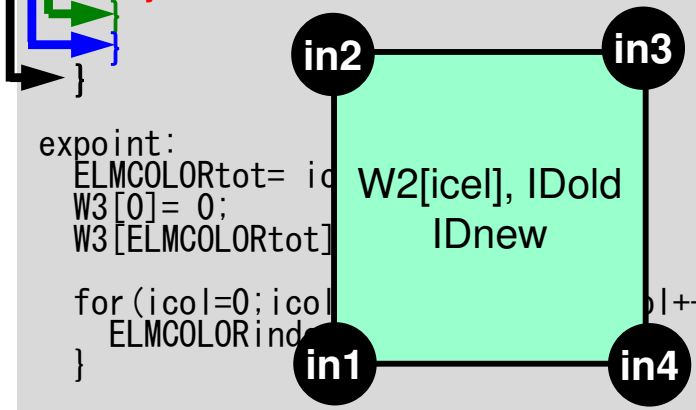
Name	Size	Content
ELMCOLORindex	**NP+1**	Element # in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**NP+1**	Accumulated # of Colored Elem's in each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



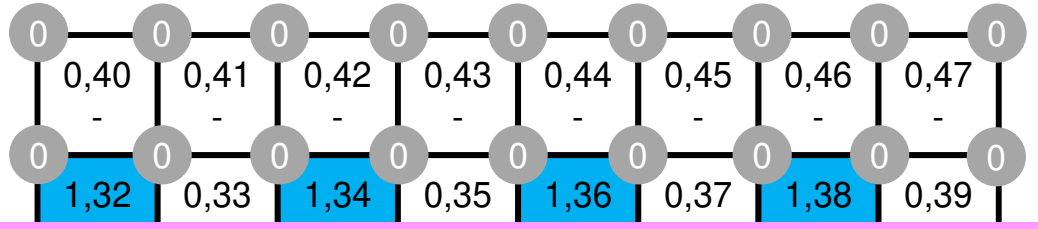
Coloring (2D) (6/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

 for(icol=1;icol<NP;icol++) {
   for(i=0;i<NP;i++) {
     W1[i]=0;
   }
   for(icel=0;icel<ICELTOT;icel++) {
   if (W2[icel]== 0){
     in1=ICELNOD[icel][0]-1;
     in2=ICELNOD[icel][1]-1;
     in3=ICELNOD[icel][2]-1;
     in4=ICELNOD[icel][3]-1;
     ip1=W1[in1];        (=0)
     ip2=W1[in2];        (=0)
     ip3=W1[in3];        (=0)
     ip4=W1[in4];        (=0)

     isum= ip1+ip2+ip3+ip4; (=0)
     if (isum==0) {
       W3[icol]= icou + 1;
       W2[icel]= icol;
       ELMCOLORitem[icou]= icel;
       icou= icou + 1;

       W1[in1]= 1;
       W1[in2]= 1;
       W1[in3]= 1;
       W1[in4]= 1;
       if (icou==ICELTOT) goto expoi
     }
   }
   }
 }

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=2
Icel=1

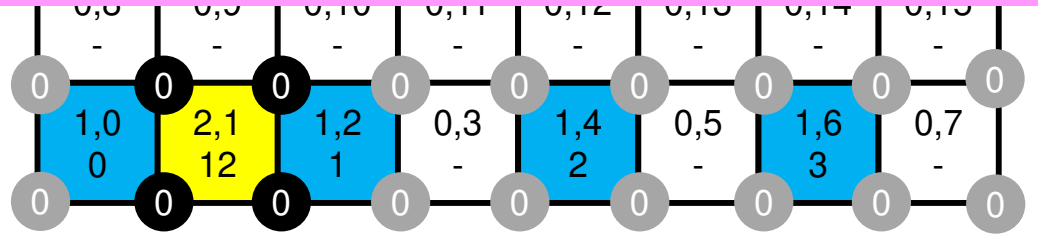


Name	Size	Content
ELMCOLORindex	**NP+1**	Element # in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**NP+1**	Accumulated # of Colored Elem's in each 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) !!

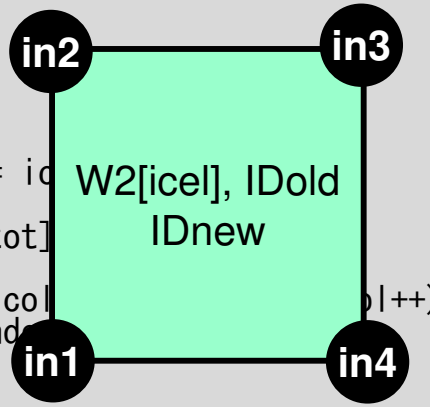
icou= icou + 1 Colored Element #, NEW Element ID
W3[icol]= icou Accumulated # of Colored Elem's in Each Color
W2[icel]= icol Color ID of Each Element
ELMCOLORitem[icou]= icel OLD Element ID



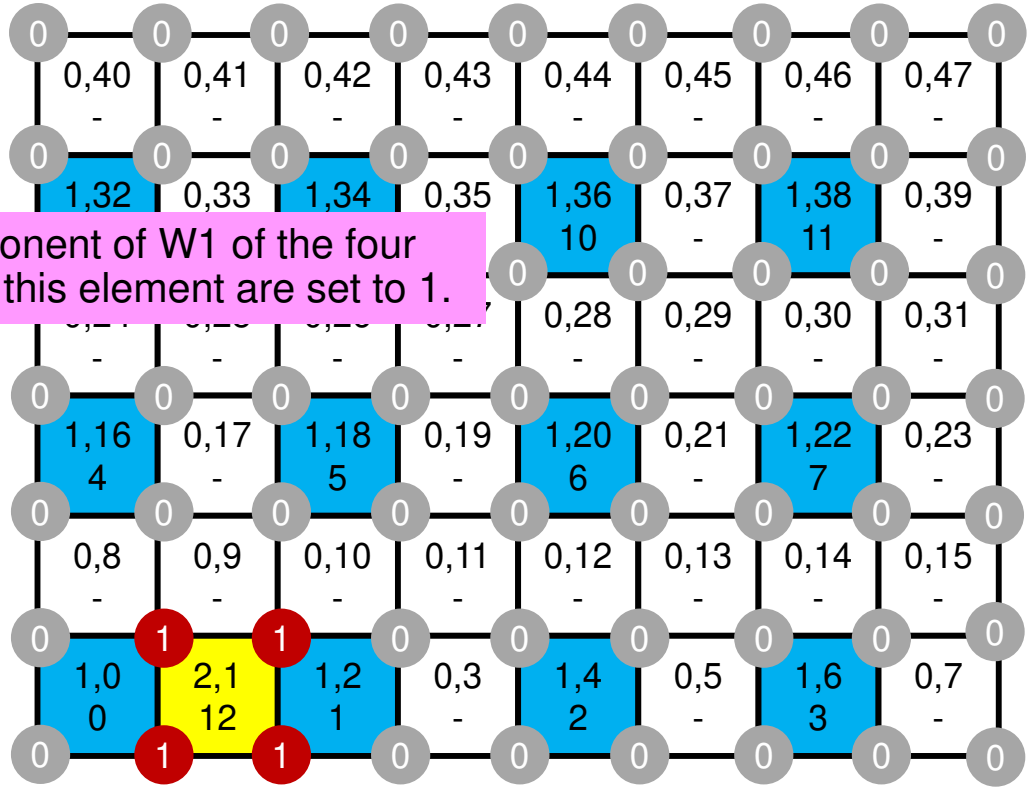
Coloring (2D) (6/7)

```
allocate_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

for(icol=1;icol<NP;icol++) {
  for(i=0;i<NP;i++) {
    W1[i]=0;
  }
  for(icel=0;icel<ICELTOT;icel++) {
  if (W2[icel]== 0){
    in1=ICELNOD[icel][0]-1;
    in2=ICELNOD[icel][1]-1;
    in3=ICELNOD[icel][2]-1;
    in4=ICELNOD[icel][3]-1;
    ip1=W1[in1];
    ip2=W1[in2];
    ip3=W1[in3];
    ip4=W1[in4];

    isum= ip1+ip2+ip3+ip4; (=0)
    if (isum==0) {
      W3[icol]= icou + 1;
      W2[icel]= icol;
      ELMCOLORitem[icou]= icel;
      icou= icou + 1;

      W1[in1]= 1;
      W1[in2]= 1;
      W1[in3]= 1;
      W1[in4]= 1;
      if (icou==ICELTOT) goto expoint;
    }
  }
  }
}

expoint:
  ELMCOLORtot= ic
  W3[0]= 0;
  W3[ELMCOLORtot]

  for(icol=0;icol                    ol++) {
    ELMCOLORind
  }
```

icol=2
icel=1



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

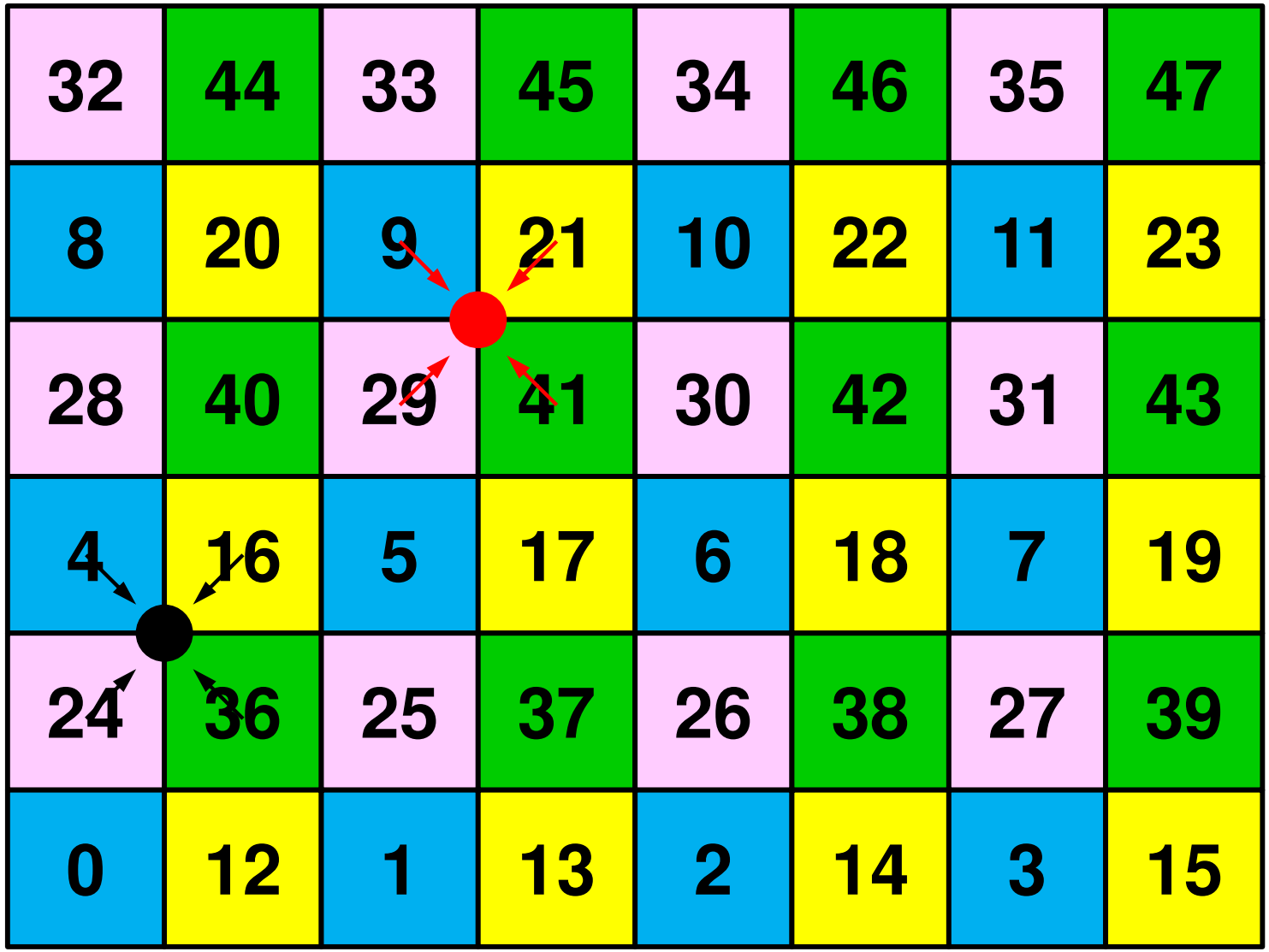


Each component of W1 of the four vertices on this element are set to 1.

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_vector(KINT)

ELMCOLRindex[NP+1], ELMCOLORitem[ICELTOT]
W1[NP], W2[ICELTOT], W3[NP+1]
W1=0; W2=0; W3=0; ELMCOLORindex[0]=0;

icou= 0;

for(icol=1;icol<NP;icol++) {
  for(i=0;i<NP;i++) {
    W1[i]=0;
  }
  for(icel=0;icel<ICELTOT;icel++) {
  if (W2[icel]== 0){
    in1=ICELNOD[icel][0]-1;
    in2=ICELNOD[icel][1]-1;
    in3=ICELNOD[icel][2]-1;
    in4=ICELNOD[icel][3]-1;
    ip1=W1[in1];
    ip2=W1[in2];
    ip3=W1[in3];
    ip4=W1[in4];

    isum= ip1+ip2+ip3+ip4;
    if (isum==0) {
      W3[icol]= icou + 1;
      W2[icel]= icol;
      ELMCOLORitem[icou]= icel;
      icou= icou + 1;

      W1[in1]= 1;
      W1[in2]= 1;
      W1[in3]= 1;
      W1[in4]= 1;
      if (icou==ICELTOT) goto expoint;
    }
  }
  }
}

expoint:
  ELMCOLORtot= icol;
  W3[0]= 0;
  W3[ELMCOLORtot]= ICELTOT;

  for(icol=0;icol<ELMCOLORtot+1;icol++) {
    ELMCOLORindex[icol]= W3[icol];
  }
```

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

32	44	33	45	34	46	35	47
8	20	9	21	10	22	11	23
28	40	29	41	30	42	31	43
4	16	5	17	6	18	7	19
24	36	25	37	26	38	27	39
0	12	1	13	2	14	3	15

Multi-Threaded Matrix Assembling Procedure

```
  for( icol=1; icol< ELMCOLORtot+1; icol++){

#pragma omp parallel for private
(icel0,icel,in1,in2,in3,in4,in5,in6,in7,in8,¥
 nodLOCAL,ie,je,ip,jp,kp,kk,iiS,iiE,k,¥
 DETJ,PNX,PNY,PNZ,PNXi,PNYi,PNZi,PNXj,PNYj,PNZj,COEFij,SHi,¥
 X1,X2,X3,X4,X5,X6,X7,X8,Y1,Y2,Y3,Y4,Y5,Y6,Y7,Y8,¥
 Z1,Z2,Z3,Z4,Z5,Z6,Z7,Z8,QVC,QV0,ipn,jpn,kpn,coef)

  for( icel0=ELMCOLORindex[icol-1]; icel0< ELMCOLORindex[icol]; icel0++){
    icel = ELMCOLORitem[icel0]; icel0: NEW Elem. ID, icel: OLD Elem. ID

    in1=ICELNOD[icel][0];
    in2=ICELNOD[icel][1];
    in3=ICELNOD[icel][2];
    in4=ICELNOD[icel][3];
    in5=ICELNOD[icel][4];
    in6=ICELNOD[icel][5];
    in7=ICELNOD[icel][6];
    in8=ICELNOD[icel][7];...
```

Name	Size	Content
ELMCOLORindex	**NP+1**	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

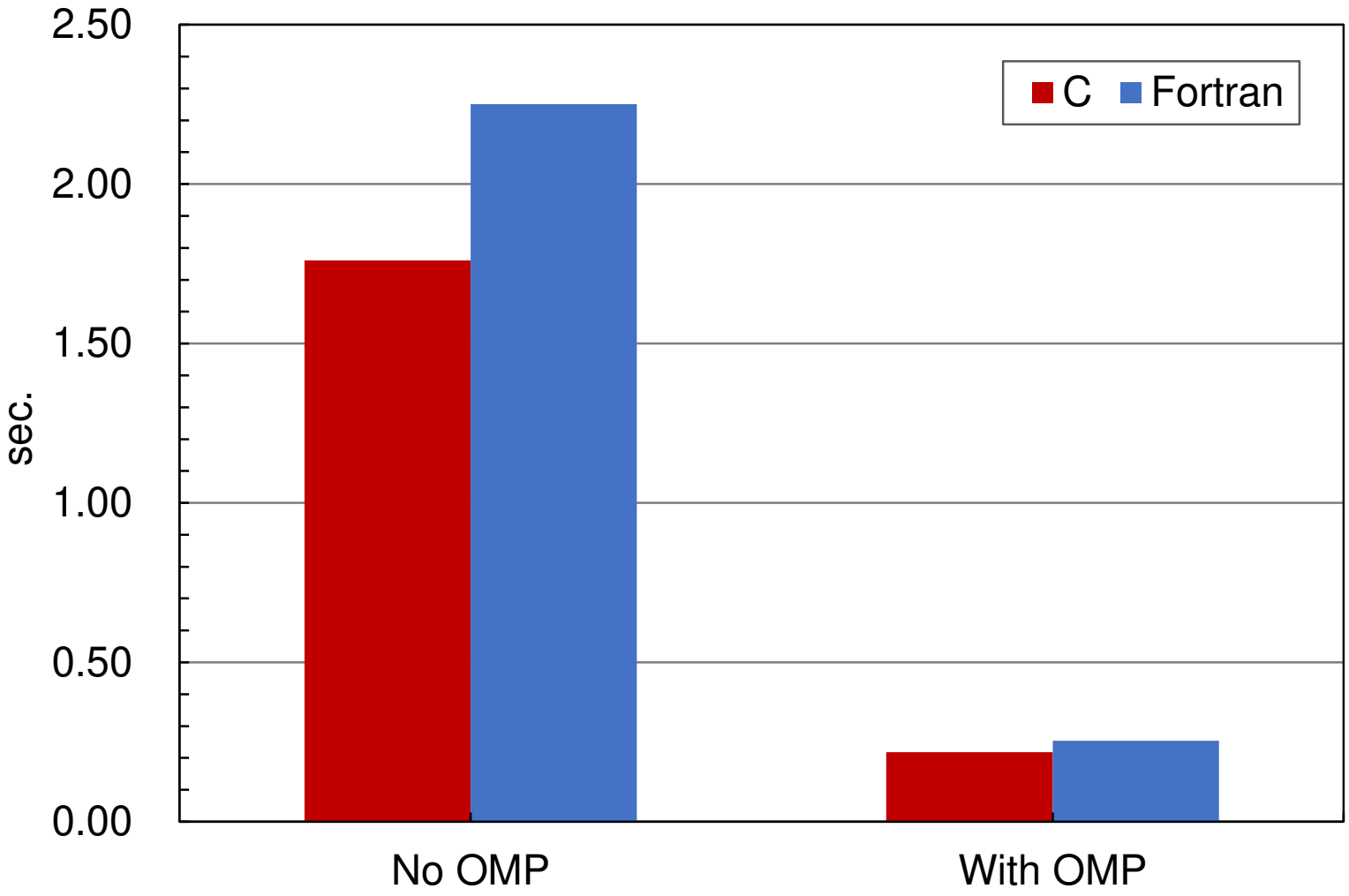
HB 12x4: 48 processes

```
#!/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);
  }
```