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

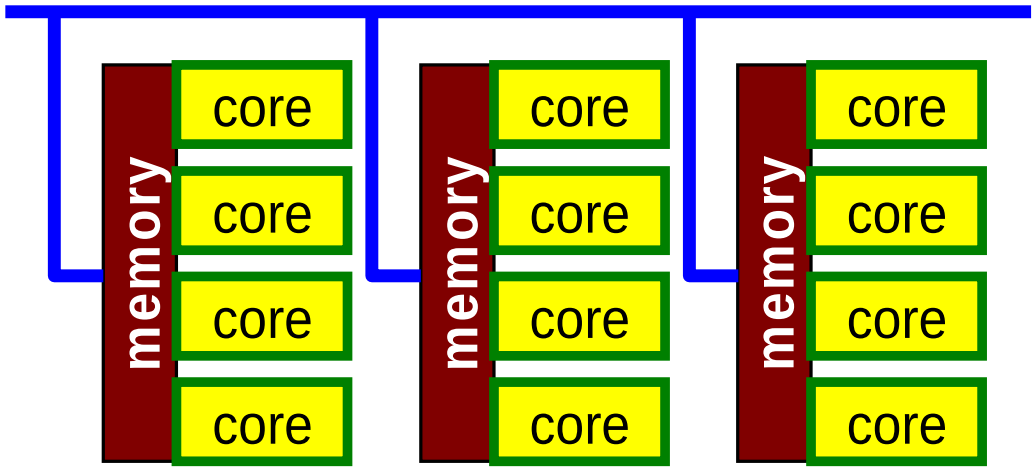
Kengo Nakajima
Information Technology Center
The University of Tokyo

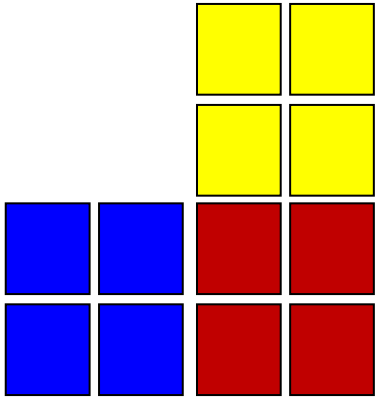
Hybrid Parallel Programming Model

- 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

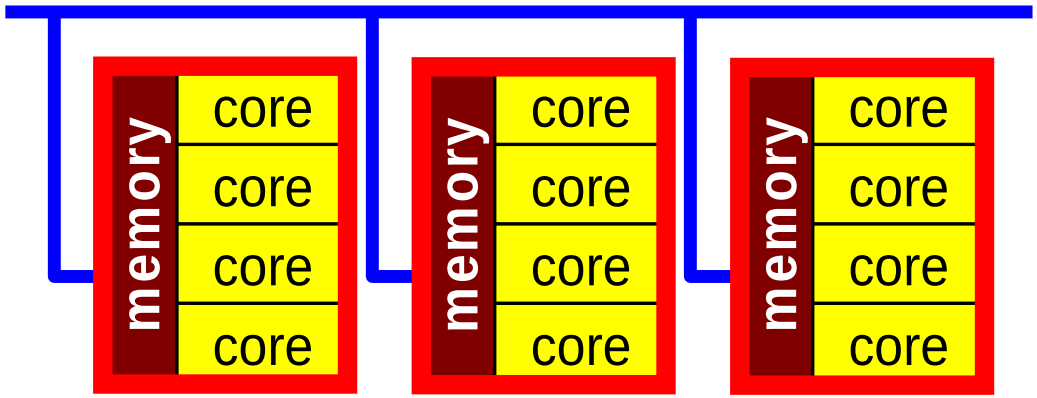
Flat MPI vs. Hybrid

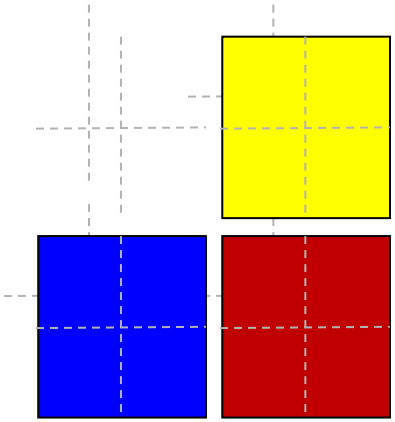
Flat-MPI：Each Core -> Independent





Hybrid：Hierarchal Structure

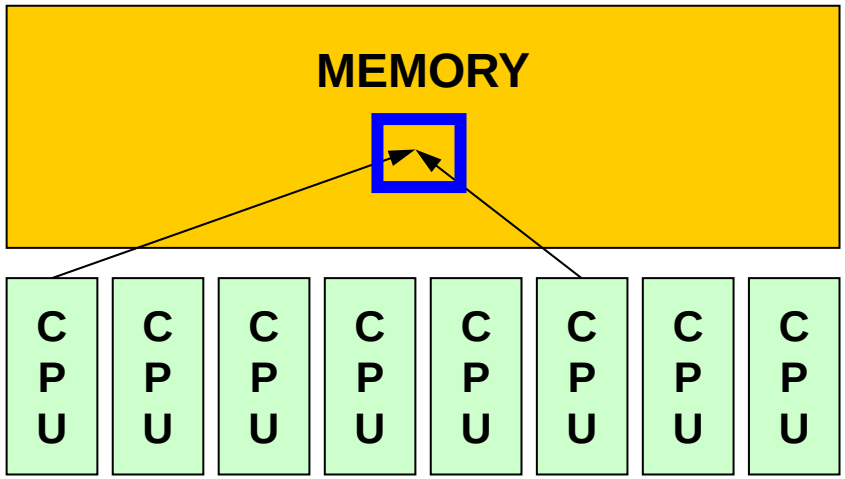




Background

- 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/19s/
- 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

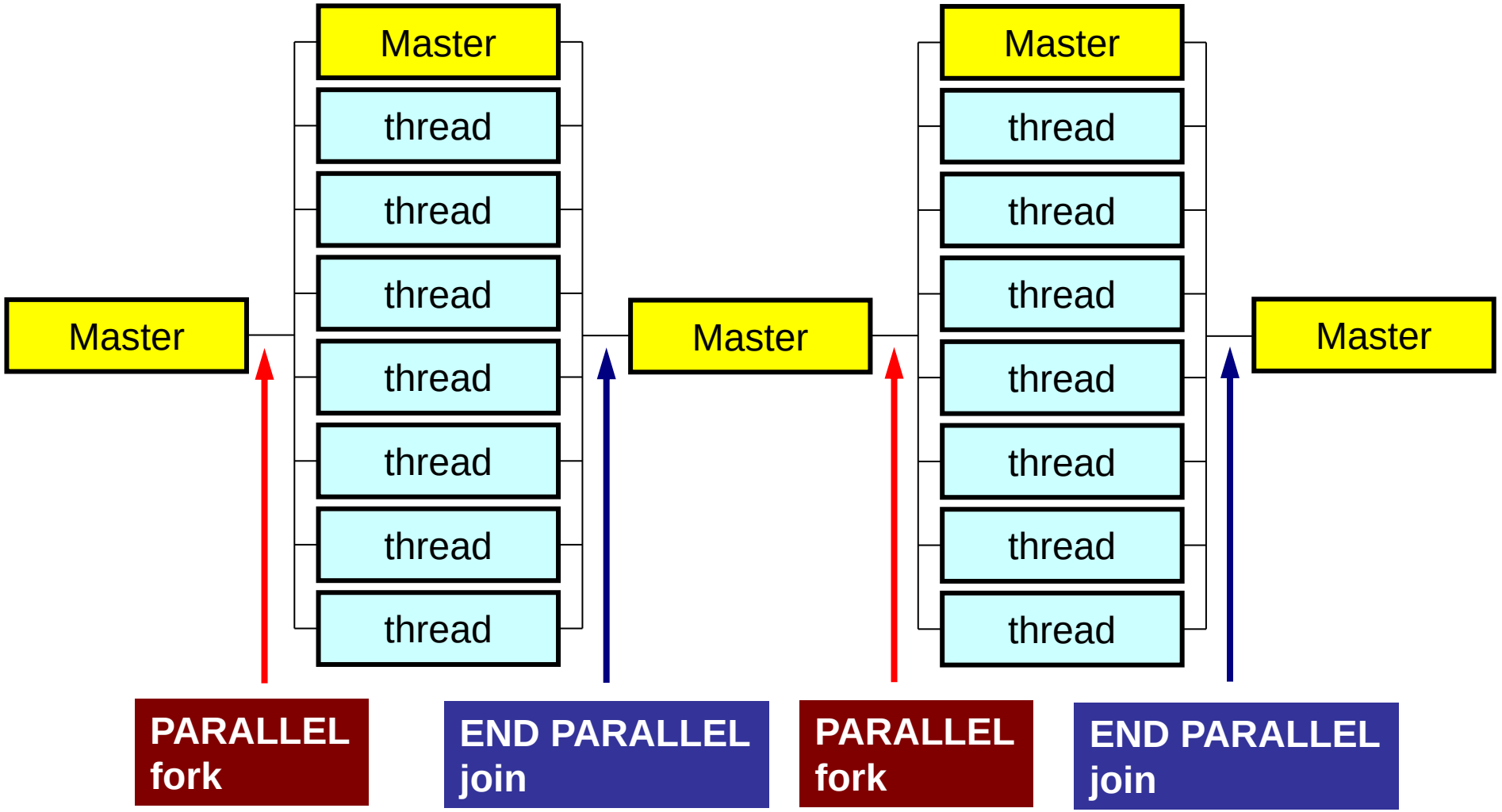
- An API (Application Programming Interface) for multi-platform shared-memory parallel programming in C/C++ and Fortran
  - Current version: 4.X (5.0 is already announced)
    - GPU, Accelerators: close to OpenACC
- Background
  - Merger of Cray and SGI in 1996
    - Separated later, … but both are now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

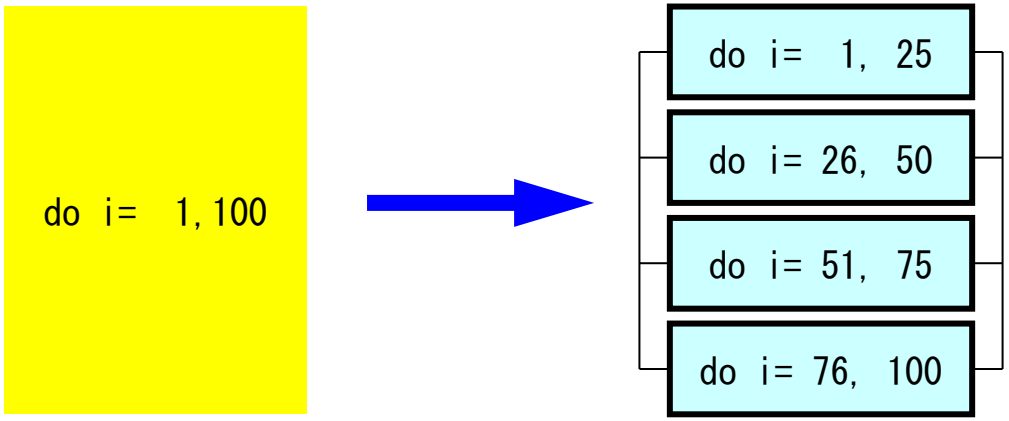


Number of Threads

- **OMP_NUM_THREADS**
  - How to change ?
    - bash(.bashrc)
    - csh(.cshrc)

```
export OMP_NUM_THREADS=8
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
```

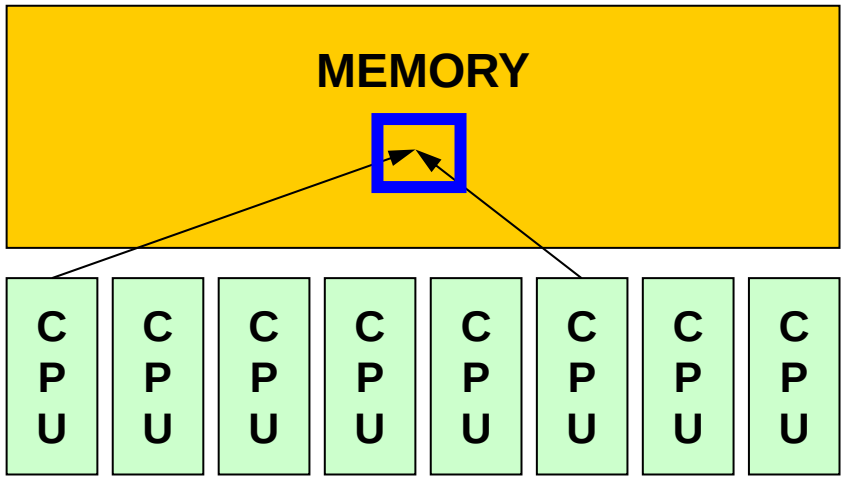
OpenMP/Direceives Matrix/Vector Products

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

Features of OpenMP

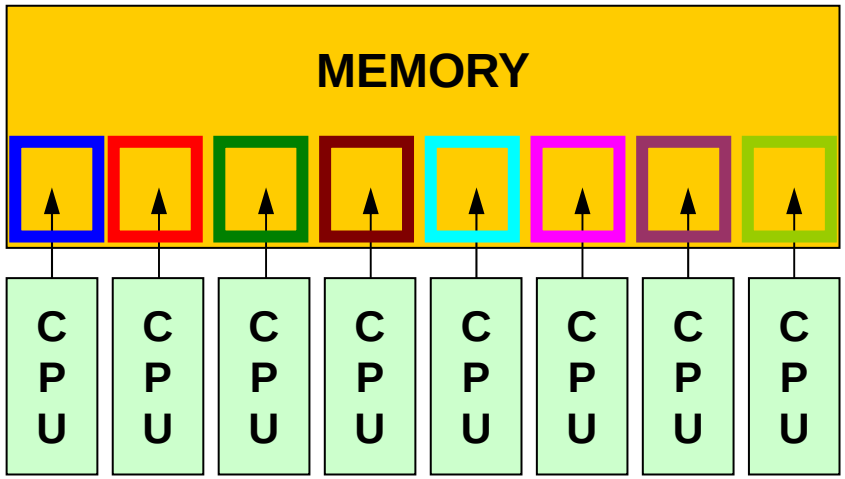
- Directives
  - Loops right after the directives are parallelized.
  - If the compiler does not support OpenMP, directives are considered as just comments.
- Nothing happen without explicit directives
  - Different from “automatic parallelization/vectorization”
  - Something wrong may happen by un-proper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

```
!$OMP PARALLEL DO PRIVATE (i)
      do i= 1, N
        B(i)= (A(i) + B(i)) * 0.50
      enddo
!$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) call omp_set_num_threads (num_threads)	Setting Thread #

OpenMP for Dot Products

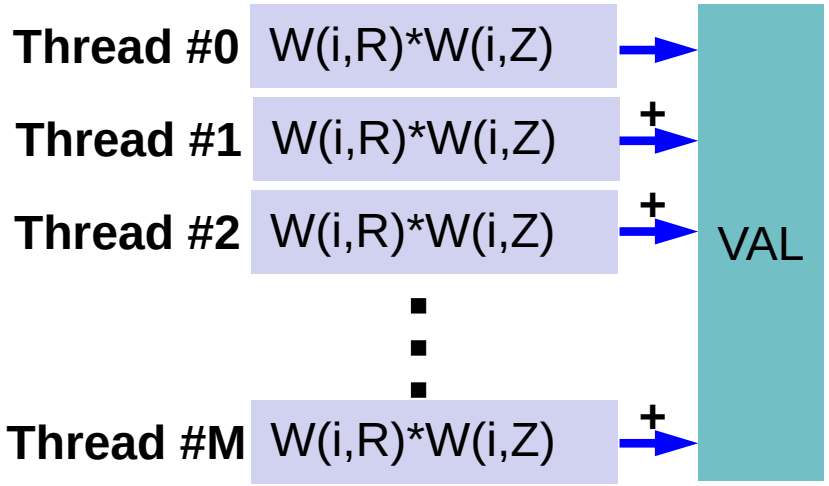
```
VAL= 0.d0
do i= 1, N
  VAL= VAL + W(i,R) * W(i,Z)
enddo
```

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

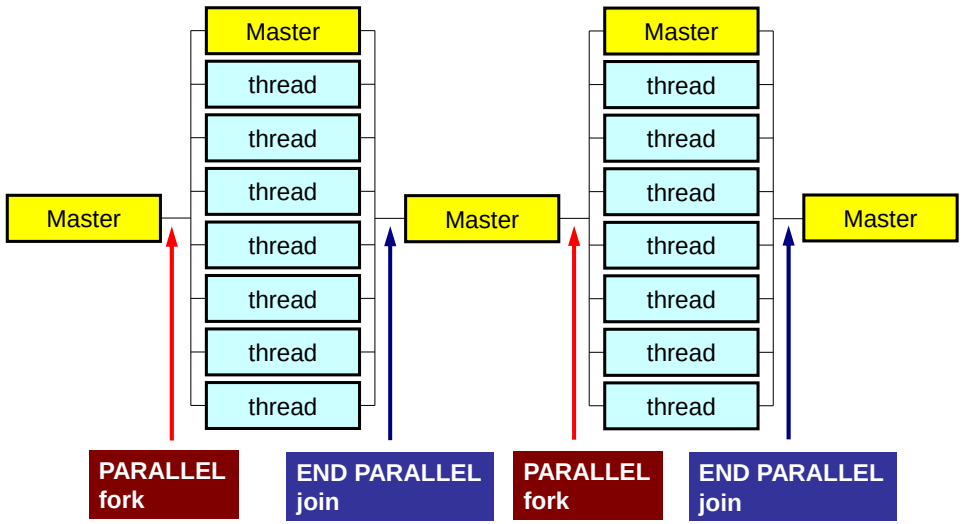
This is rather better for GPU and (very) many-core architectures: simpler structure of loops

"!$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 /work/gt62/t62XXX/pFEM/pfem3d/src1
>$ make
>$ cd ../run
>$ ls sol1
      sol1

>$ cd ../pmesh

<Parallel Mesh Generation>

>$ cd ../run

<modify tLLxMMxN.sh>

>$ pjsub tLLxMMxN.sh
```

Makefile（Fortran）

```
F90        = mpiifort
F90LINKER = $(F90)
LIB_DIR    =
INC_DIR    =
OPTFLAGS   = -align array64byte -O3 -axCORE-AVX512 -qopenmp
FFLAGS = $(OPTFLAGS)
FLIBS =
F90LFLAGS=
#
TARGET = ../run/sol1
default: $(TARGET)
OBJS =¥
pfem_util.o …

$(TARGET):  $(OBJS)
        $(F90LINKER) $(OPTFLAGS) -o $(TARGET) $(OBJS) $(F90LFLAGS)
clean:
        /bin/rm -f *.o $(TARGET) *~ *.mod
.f.o:
        $(F90) $(FFLAGS) $(INC_DIR) -c  $*.f
.f90.o:
        $(F90) $(FFLAGS) $(INC_DIR) -c  $*.f90
.SUFFIXES: .f90 .f
```

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)

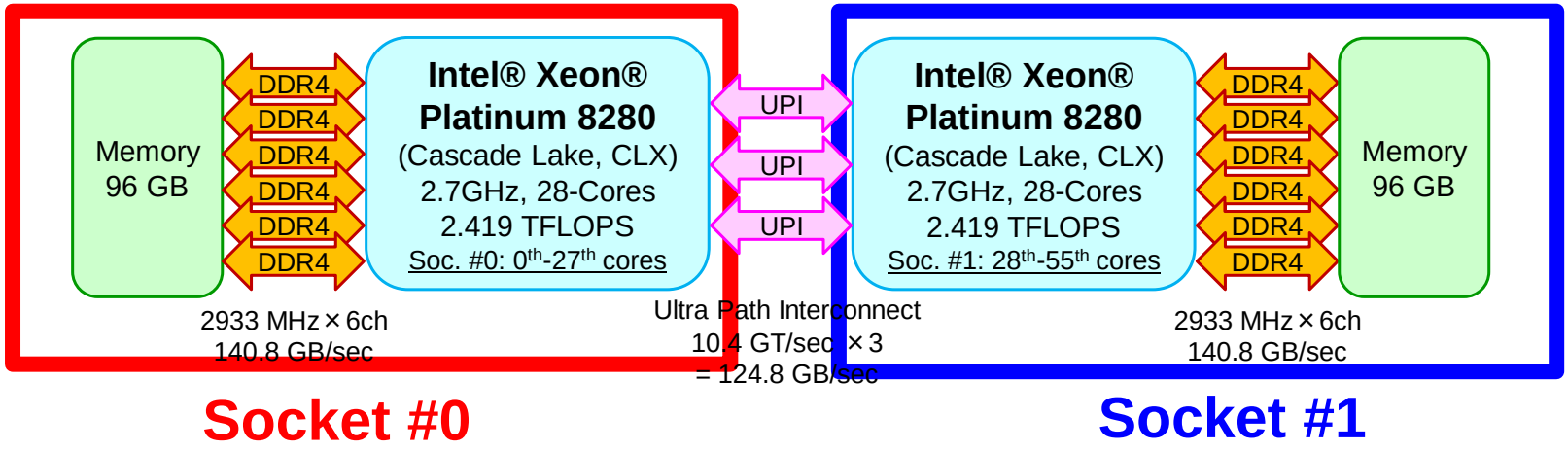
```
!$omp parallel do private(i)
      do i= 1, N
         X(i)  = X (i)    + ALPHA * WW(i,P)
        WW(i,R)= WW(i,R) - ALPHA * WW(i,Q)
      enddo
```

```
      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
```



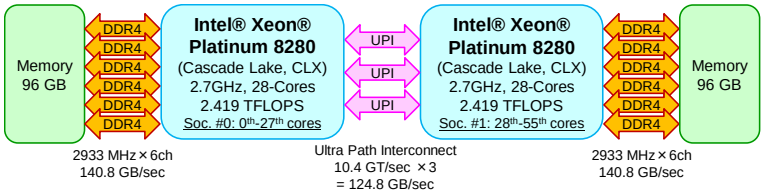
HB M x N

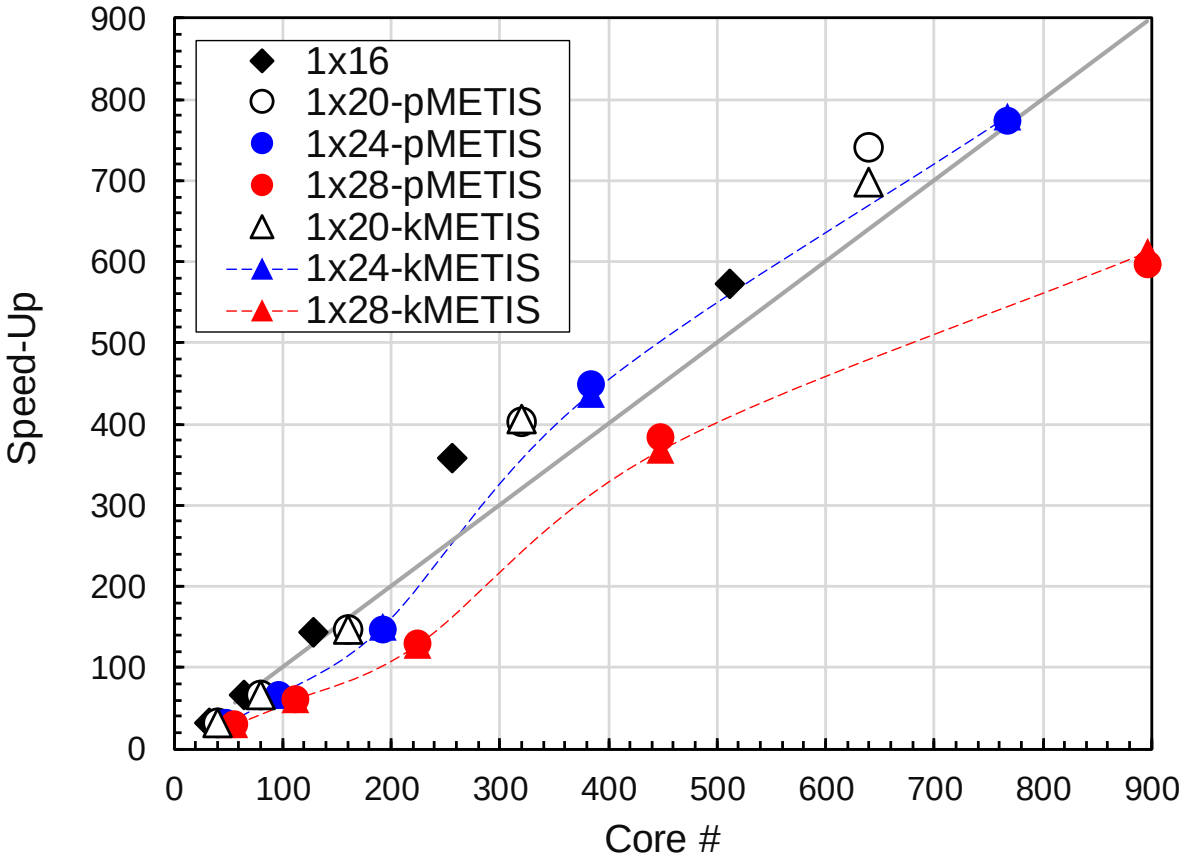
Number of OpenMP threads per a single MPI process

Number of MPI process per a single “socket”

Example: Strong Scaling: Fortran Flat MPI

- 128×128×128 nodes, 2,097,152 DOF
- 32～896 cores, Linear Solver





- ◆ 1x16 proc's for each Socket
  1x16x2 (=32) for each Node
- ○ 1x20x2 (=40), pMETIS
- ● 1x24x2 (=48), pMETIS
- ● 1x28x2 (=56), pMETIS
- △ 1x20x2 (=40), kMETIS
- ▲ 1x24x2 (=48), kMETIS
- ▲ 1x28x2 (=56), kMETIS

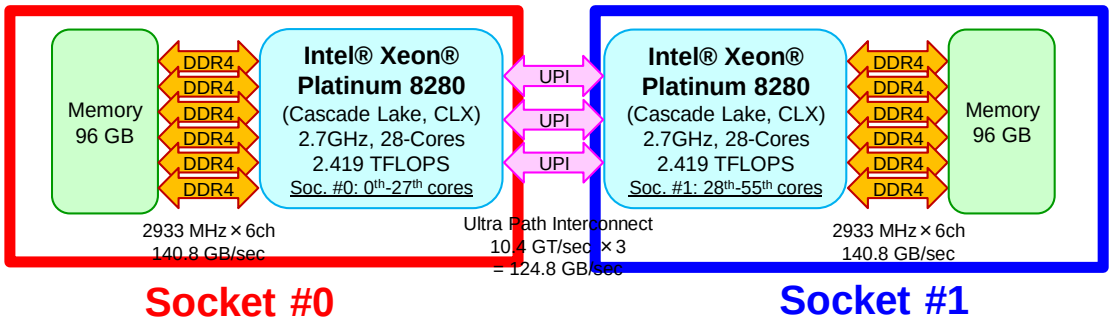
Performance of ◆ without NUMA at 32-cores= 32.0

HB (2or3) x 8, 8-nodes, 128-proc's

t02x08x2.sh t03x08x2.sh

mesh.inp

```
128 128 128
  8   4   4
pcube
```



t02x08x2.sh, HB 2x8

16/28 cores on ea. Soc.
2-threads x 8-proc's

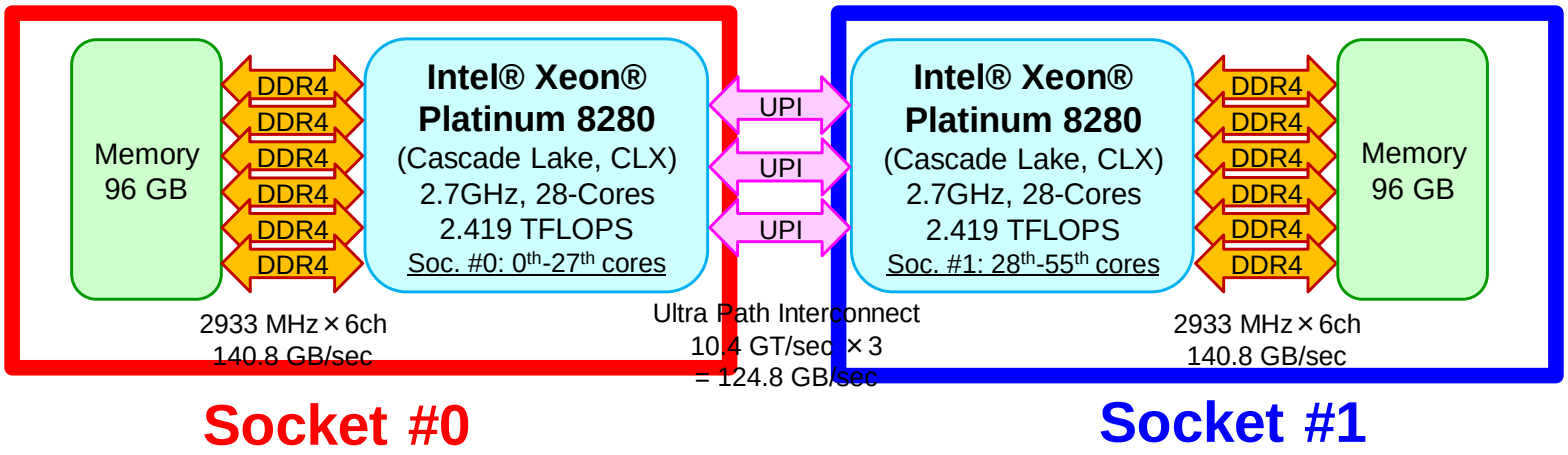
```
#!/bin/sh
#PJM -N "HB02x08x2
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=128
#PJM --omp thread=2
#PJM -L elapse=00:15:00
#PJM -g gt62
#PJM -j
#PJM -e err
#PJM -o t02x08x2_0001.lst

mpiexec.hydra -n ${PJM_MPI_PROC} ./sol1
mpiexec.hydra -n ${PJM_MPI_PROC} numactl -l ./sol1

export KMP_AFFINITY=granularity=fine,compact
mpiexec.hydra -n ${PJM_MPI_PROC} ./sol1
mpiexec.hydra -n ${PJM_MPI_PROC} numactl -l ./sol1
```

t03x08x2.sh, HB 3x8

24/28 cores on ea. Soc.
3-threads x 8-proc's

```
#!/bin/sh
#PJM -N "HB03x08x2
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=128
#PJM --omp thread=3
#PJM -L elapse=00:15:00
#PJM -g gt62
#PJM -j
#PJM -e err
#PJM -o t03x08x2_0001.lst

mpiexec.hydra -n ${PJM_MPI_PROC} ./sol1
mpiexec.hydra -n ${PJM_MPI_PROC} numactl -l ./sol1

export KMP_AFFINITY=granularity=fine,compact
mpiexec.hydra -n ${PJM_MPI_PROC} ./sol1
mpiexec.hydra -n ${PJM_MPI_PROC} numactl -l ./sol1
```

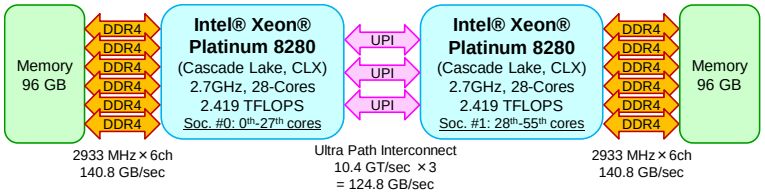
- export KMP_AFFINITY=granularity=fine,compact
  - If the total number of cores are much smaller than 56 on each node, cores on Socke #0 are used prior to those on Socket #1.
  - **This improves performance if number of M in HB MxN (OpenMP Thread # on each MPI Process) is rather larger.**

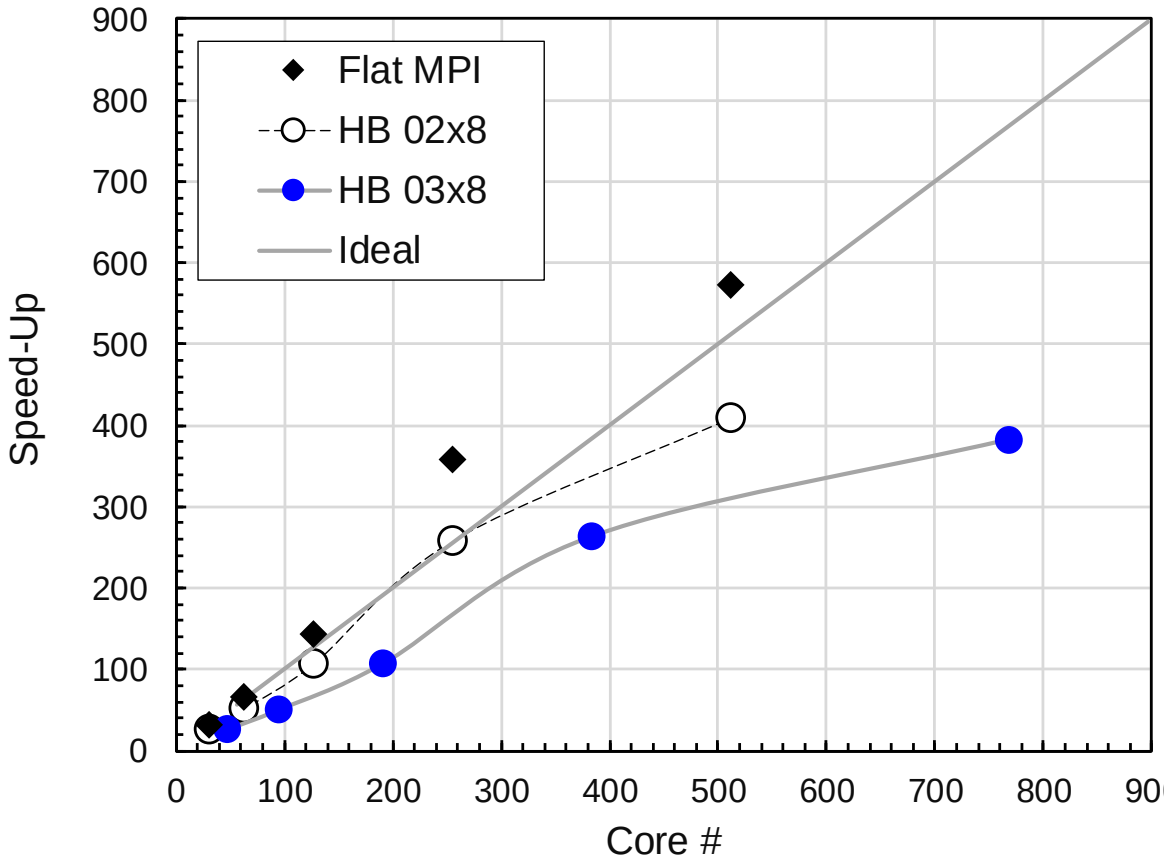


Example: Strong Scaling: Fortran

HB 2/3 x 8, 8-MPI Proc's on Soc.

- 128 × 128 × 128 nodes, 2,097,152 DOF
- 1-16 nodes, Linear Solver
- Best at 16-nodes





◆ Flat MPI
16 core's for each Socket
32 core's for each Node

○ HB 2x8
2x8= 16-cores/Soc,
32-cores/Node

● HB 3x8
3x8= 24-cores/Soc
48-cores/Node

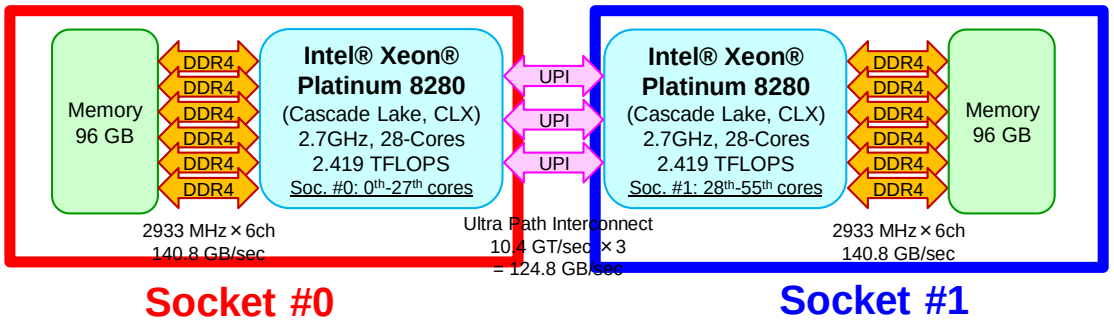
Performance of ◆ without NUMA at 32-cores= 32.0

HB (5or6or7) x 4, 8-nodes, 64-proc's

t05x04x2.sh t06x04x2.sh, t07x04x2.sh

mesh.inp

```
128 128 128
  4   4   4
pcube
```



t05x04x2.sh, HB 5x4
20/28 cores on ea. Soc.
5-threads x 4-proc's

```
#!/bin/sh
#PJM -N "HB05x04x2"
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=64
#PJM --omp thread=5
```

t06x04x2.sh, HB 6x4
24/28 cores on ea. Soc.
6-threads x 4-proc's

```
#!/bin/sh
#PJM -N "HB06x04x2"
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=64
#PJM --omp thread=6
```

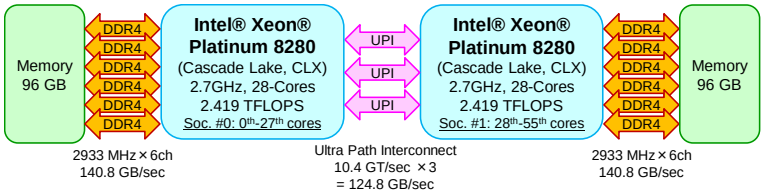
```
#!/bin/sh
#PJM -N "HB07x04x2"
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=64
#PJM --omp thread=7
```

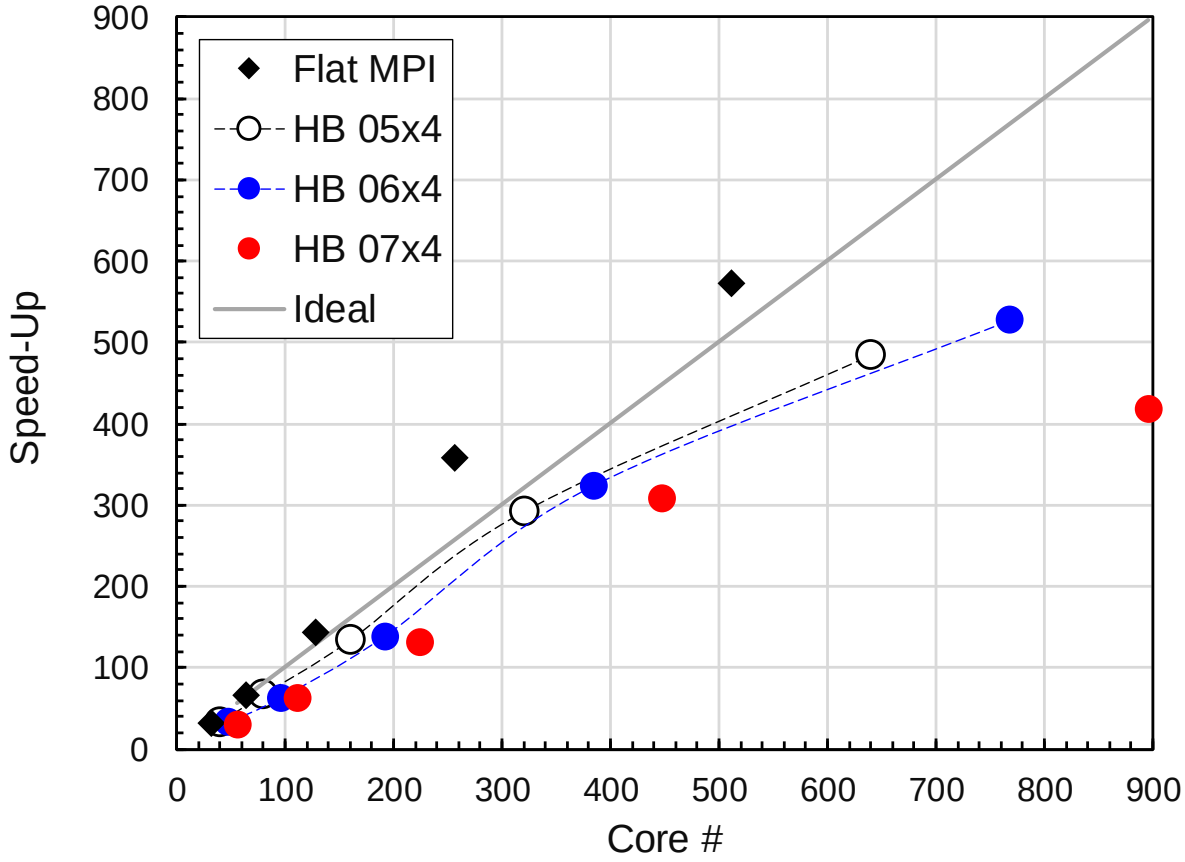
t07x04x2.sh, HB 7x4
28/28 cores on ea. Soc.
7-threads x 4-proc's

Example: Strong Scaling: Fortran

HB 5/6/7 x 4, 4-MPI Proc's on Soc.

- 128 × 128 × 128 nodes, 2,097,152 DOF
- 1-16 nodes, Linear Solver
- Best at 16-nodes





- ◆ Flat MPI
  16 core's for each Socket
  32 core's for each Node
- ○ HB 5x4
  5x4= 20-cores/Socket
  40-cores/Node
- ● HB 6x4
  6x4= 24-cores/Socket
  48-cores/Node
- ● HB 7x4
  7x4= 28-cores/Socket
  56-cores/Node

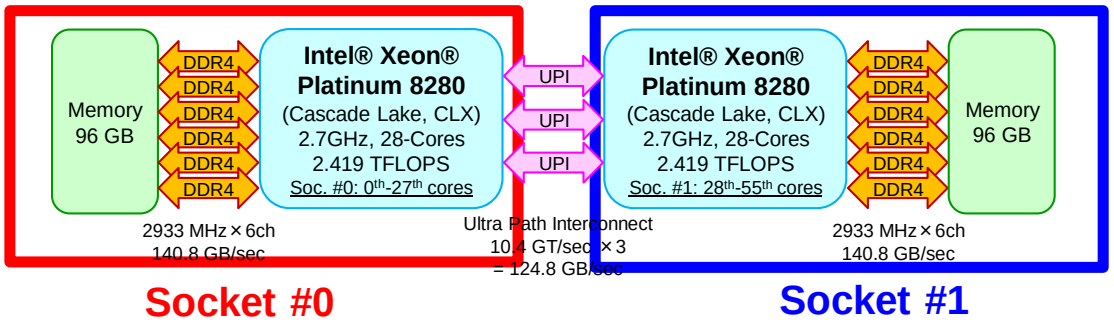
Performance of ◆ without NUMA at 32-cores= 32.0

HB (10/12/14) x 2, 8-nodes, 32-proc's

t10x02x2.sh t12x02x2.sh, t14x02x2.sh

mesh.inp

```
128 128 128
  4   4   2
pcube
```



t10x02x2.sh, HB 10x2

20/28 cores on ea. Soc.
10-threads x 2-proc's

```
#!/bin/sh
#PJM -N "HB10x02x2"
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=32
#PJM --omp thread=10
```

t12x02x2.sh, HB 12x2

24/28 cores on ea. Soc.
12-threads x 2-proc's

```
#!/bin/sh
#PJM -N "HB12x02x2"
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=32
#PJM --omp thread=12
```

t14x02x2.sh, HB 14x2

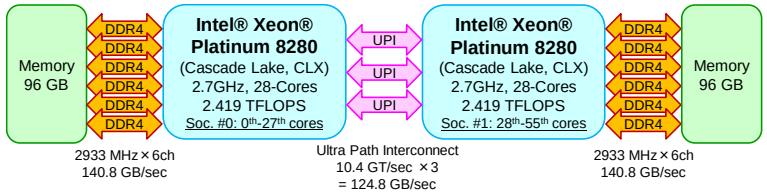
28/28 cores on ea. Soc.
14-threads x 2-proc's

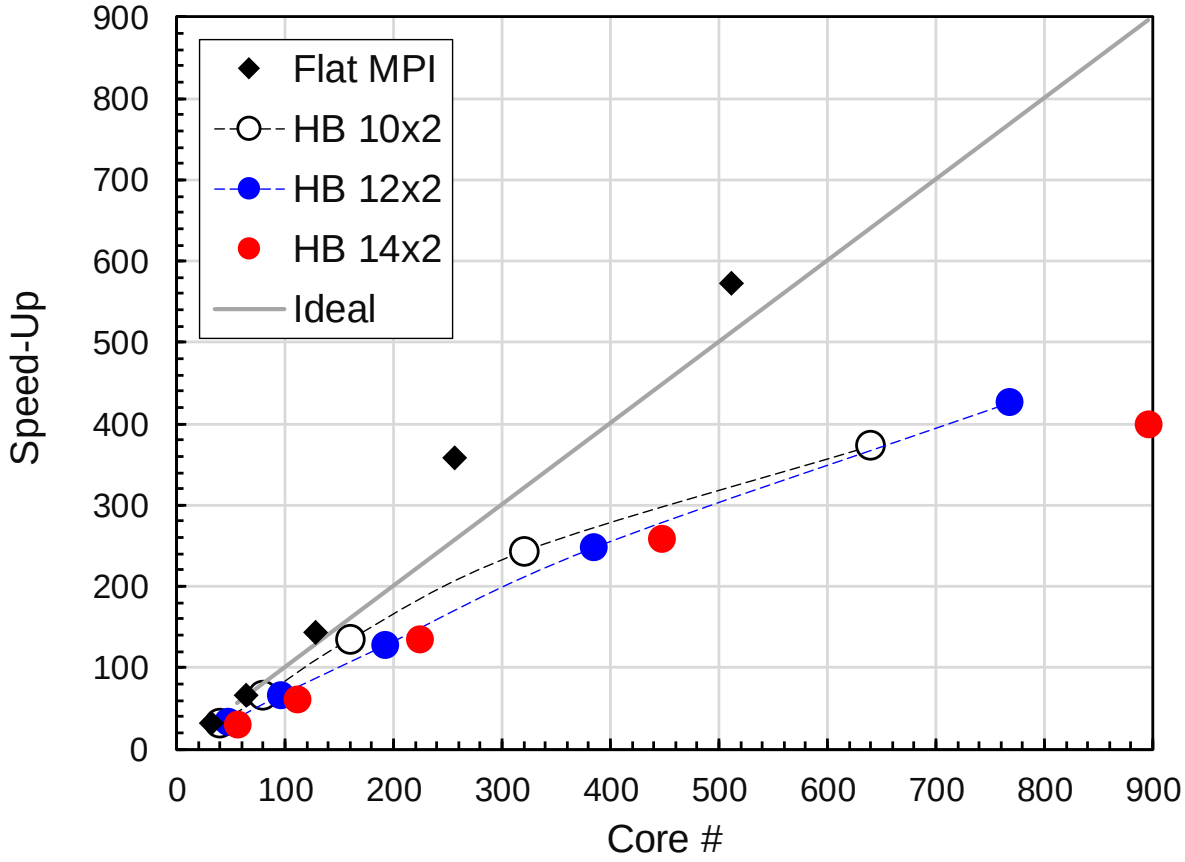
```
#!/bin/sh
#PJM -N "HB14x02x2"
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=32
#PJM --omp thread=14
```

Example: Strong Scaling: Fortran

HB 10/12/14 x 2, 2-MPI Proc's on Soc.

- 128 × 128 × 128 nodes, 2,097,152 DOF
- 1-16 nodes, Linear Solver
- Best at 16-nodes





- ◆ Flat MPI
  16 core's for each Socket
  32 core's for each Node
- ○ HB 10x2
  10x2= 20-cores/Socket
  40-cores/Node
- ● HB 12x2
  12x2= 24-cores/Socket
  48-cores/Node
- ● HB 14x2
  14x2= 28-cores/Socket
  56-cores/Node

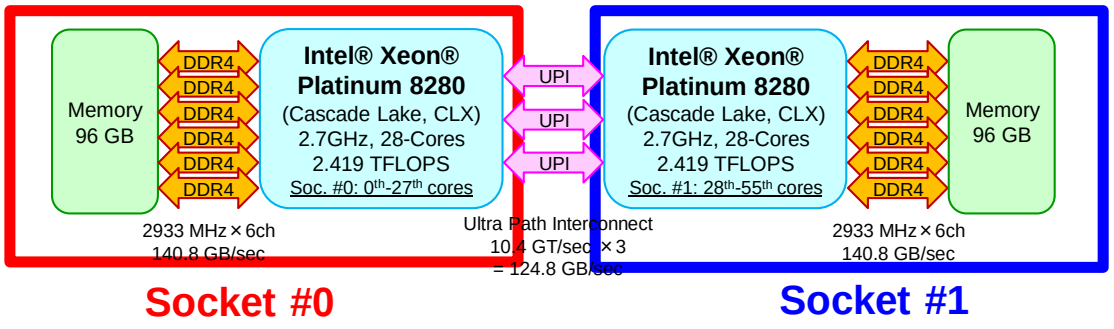
Performance of ◆ without NUMA at 32-cores= 32.0

HB (20/24/28) x 1, 8-nodes, 16-proc's

t20x01x2.sh t24x01x2.sh, t28x01x2.sh

mesh.inp

```
128 128 128
  4   2   2
pcube
```



t20x01x2.sh, HB 20x1
20/28 cores on ea. Soc.
20-threads x 1-proc's

```
#!/bin/sh
#PJM -N "HB20x01x2"
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=16
#PJM --omp thread=20
```

t24x01x2.sh, HB 24x1
24/28 cores on ea. Soc.
24-threads x 1-proc's

```
#!/bin/sh
#PJM -N "HB24x01x2"
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=16
#PJM --omp thread=24
```

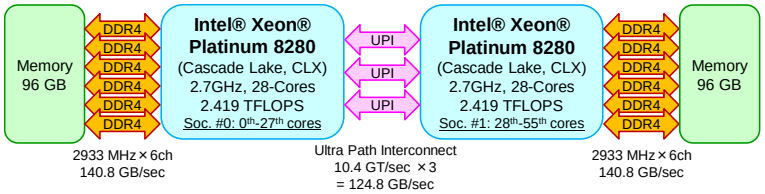
```
#!/bin/sh
#PJM -N "HB28x01x2"
#PJM -L rscgrp=lecture2
#PJM -L node=8
#PJM --mpi proc=16
#PJM --omp thread=28
```

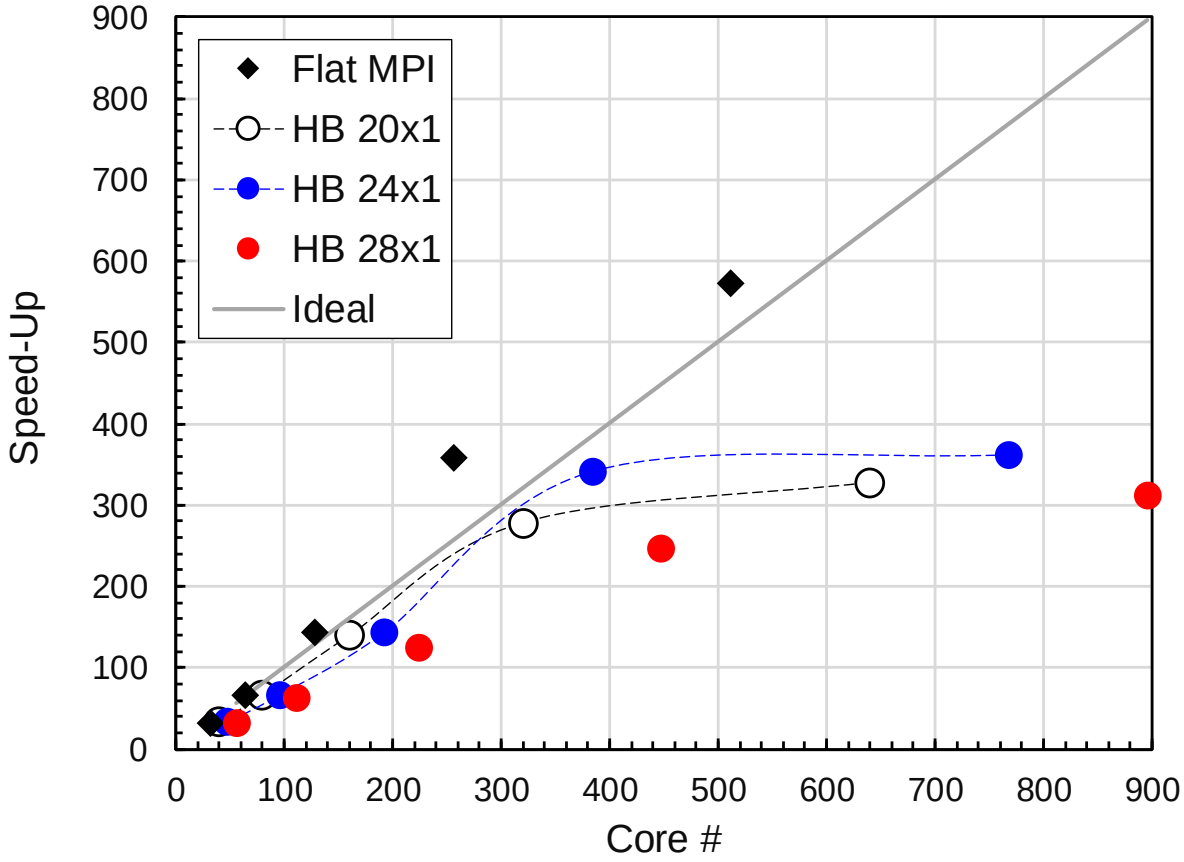
t28x01x2.sh, HB 28x1
28/28 cores on ea. Soc.
28-threads x 1-proc's

Example: Strong Scaling: Fortran

HB 20/24/28 x 1, 1-MPI Proc's on Soc.

- 128 × 128 × 128 nodes, 2,097,152 DOF
- 1-16 nodes, Linear Solver
- Best at 16-nodes





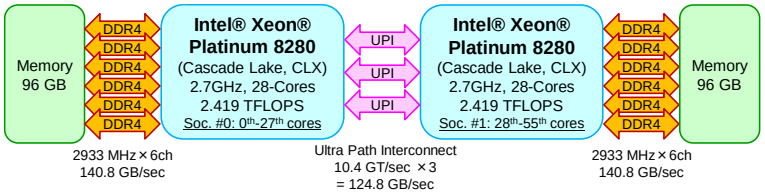
- ◆ Flat MPI
  16 core's for each Socket
  32 core's for each Node
- ○ HB 20x1
  20x1= 20-cores/Socket
  40-cores/Node
- ● HB 24x1
  24x1= 24-cores/Socket
  48-cores/Node
- ● HB 28x2
  28x1= 28-cores/Socket
  56-cores/Node

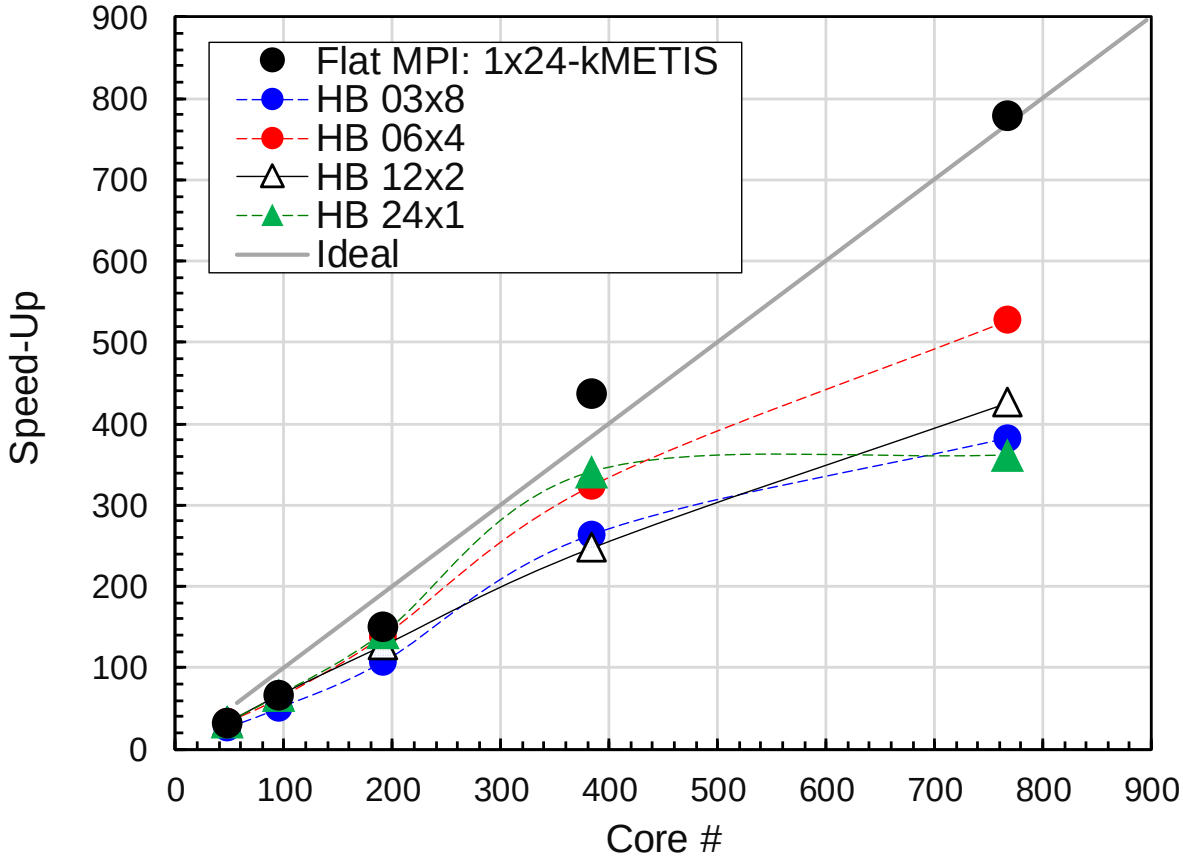
Performance of ◆ without NUMA at 32-cores= 32.0

Example: Strong Scaling: Fortran

Best Cases: 24-cores/Socket, 48-cores/Node

- 128 × 128 × 128 nodes, 2,097,152 DOF
- 1-16 nodes, Linear Solver
- Best at 16-nodes





- Flat MPI 1x24 k-METIS
  24 core's for each Socket
  48 core's for each Node
- HB 3x8
- HB 6x4
- HB 12x2
- HB 24x1

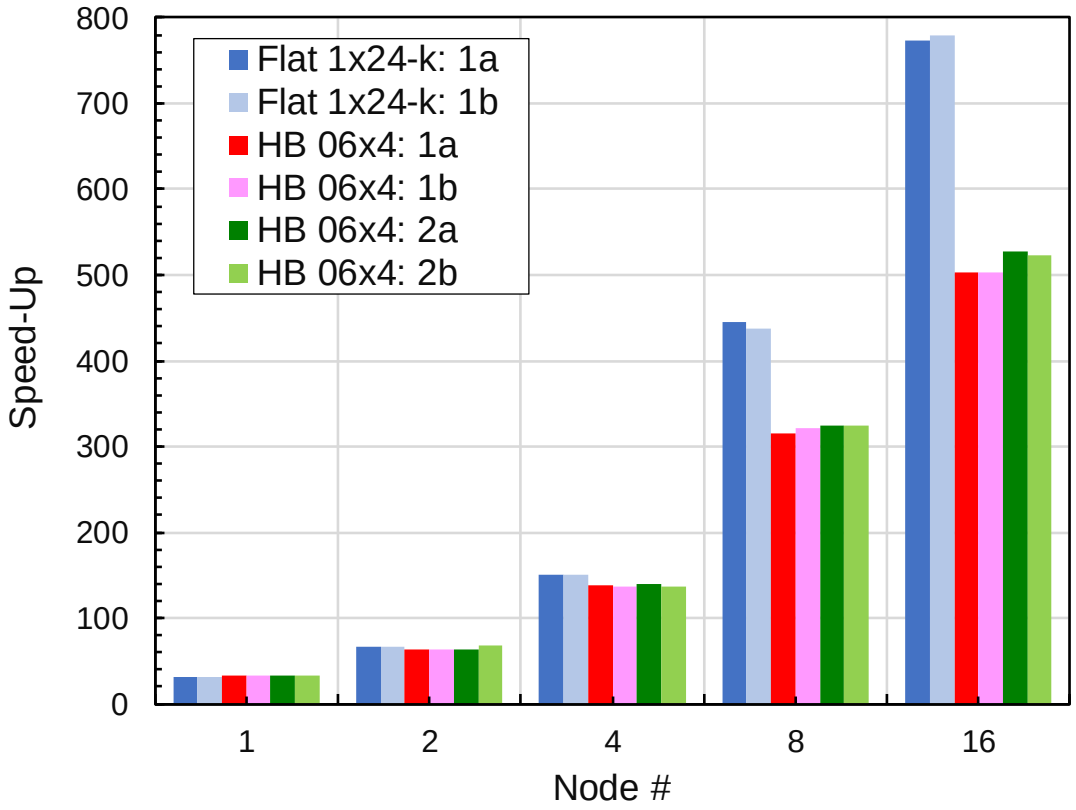
Performance of ◆ (Flat MPI 1x16) without NUMA at 32-cores= 32.0

Example: Strong Scaling: Fortran

Best Cases: 24-cores/Socket, 48-cores/Node

- 1a: No Options
- 1b: numactl –l
- 2a: KMP_AFFINITY
- 2b: KMP_AFFINITY + numactl -l

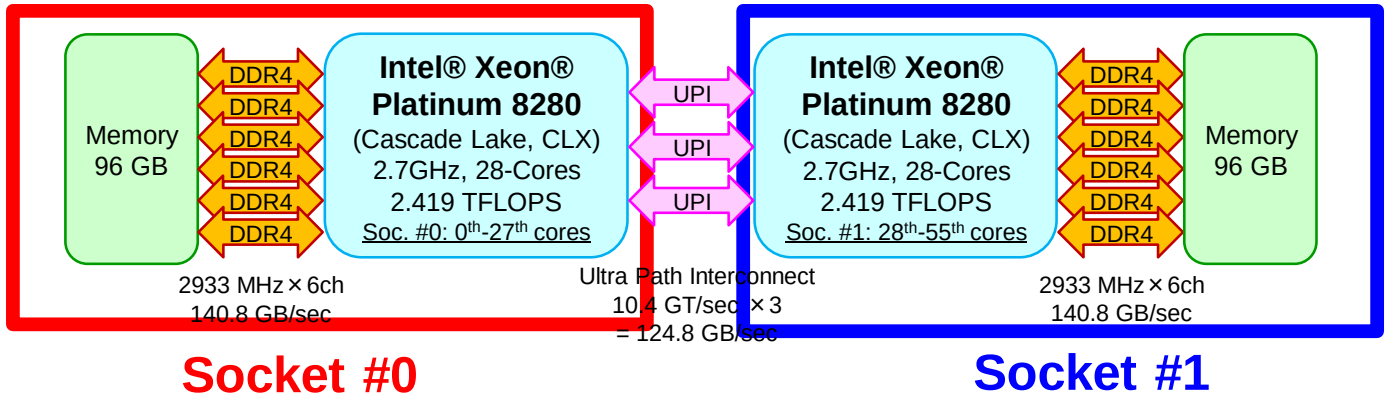
```
mpiexec.hydra -n ${PJM_MPI_PROC} ./sol1                1a
mpiexec.hydra -n ${PJM_MPI_PROC} numactl -l ./sol1     1b

export KMP_AFFINITY=granularity=fine,compact
mpiexec.hydra -n ${PJM_MPI_PROC} ./sol1                2a
mpiexec.hydra -n ${PJM_MPI_PROC} numactl -l ./sol1     2b
```



Performance of ◆ (Flat MPI 1x16) without NUMA at 32-cores= 32.0

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 number of computing nodes is larger.
  - Fewer number of MPI processes.
  - Behavior of HB in many MPI proc's on OBCX is under investigation
- 1 MPI Process/Node is possible: NUMA (S1/S2)

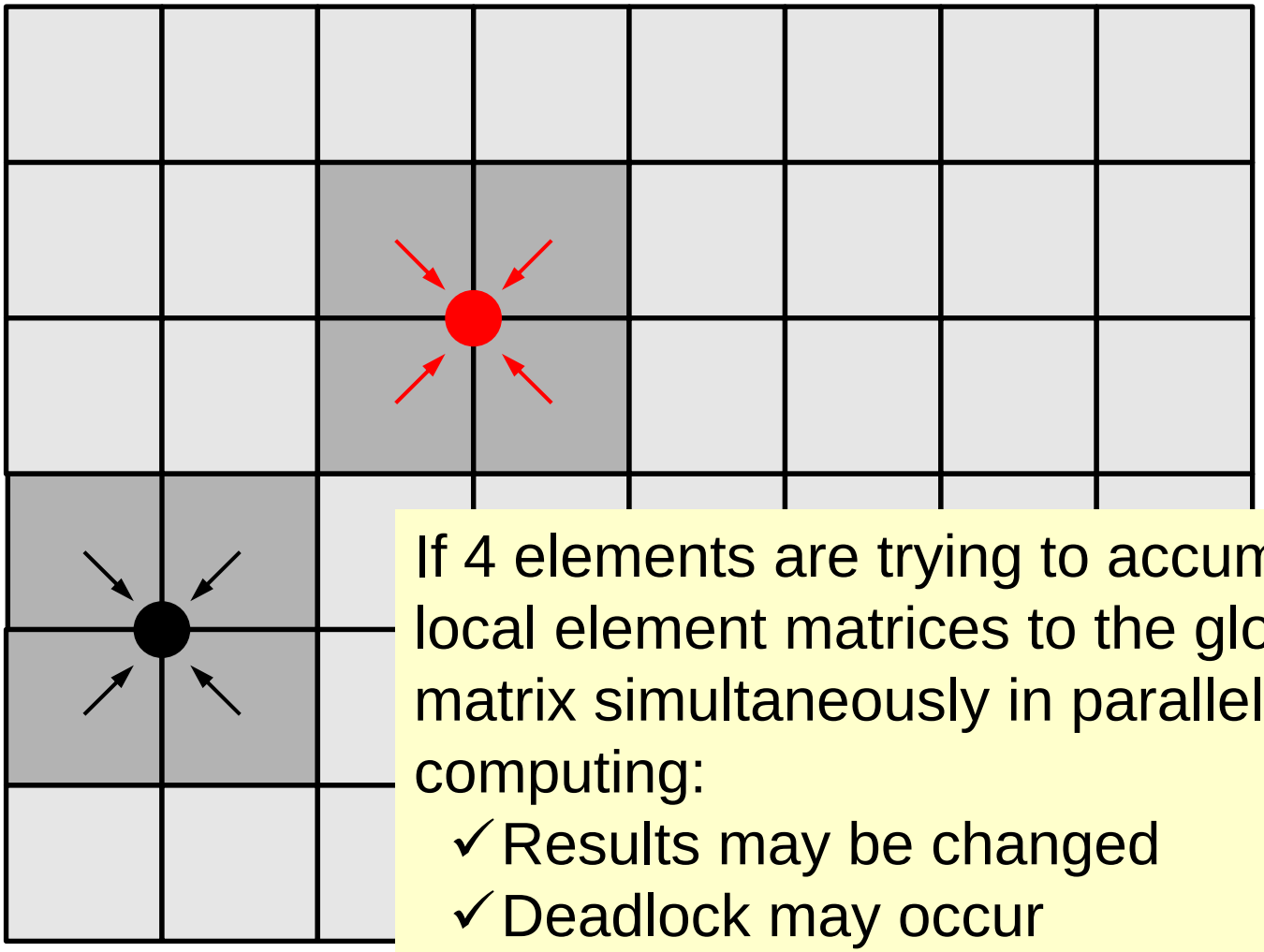


How to apply multi-threading

- 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



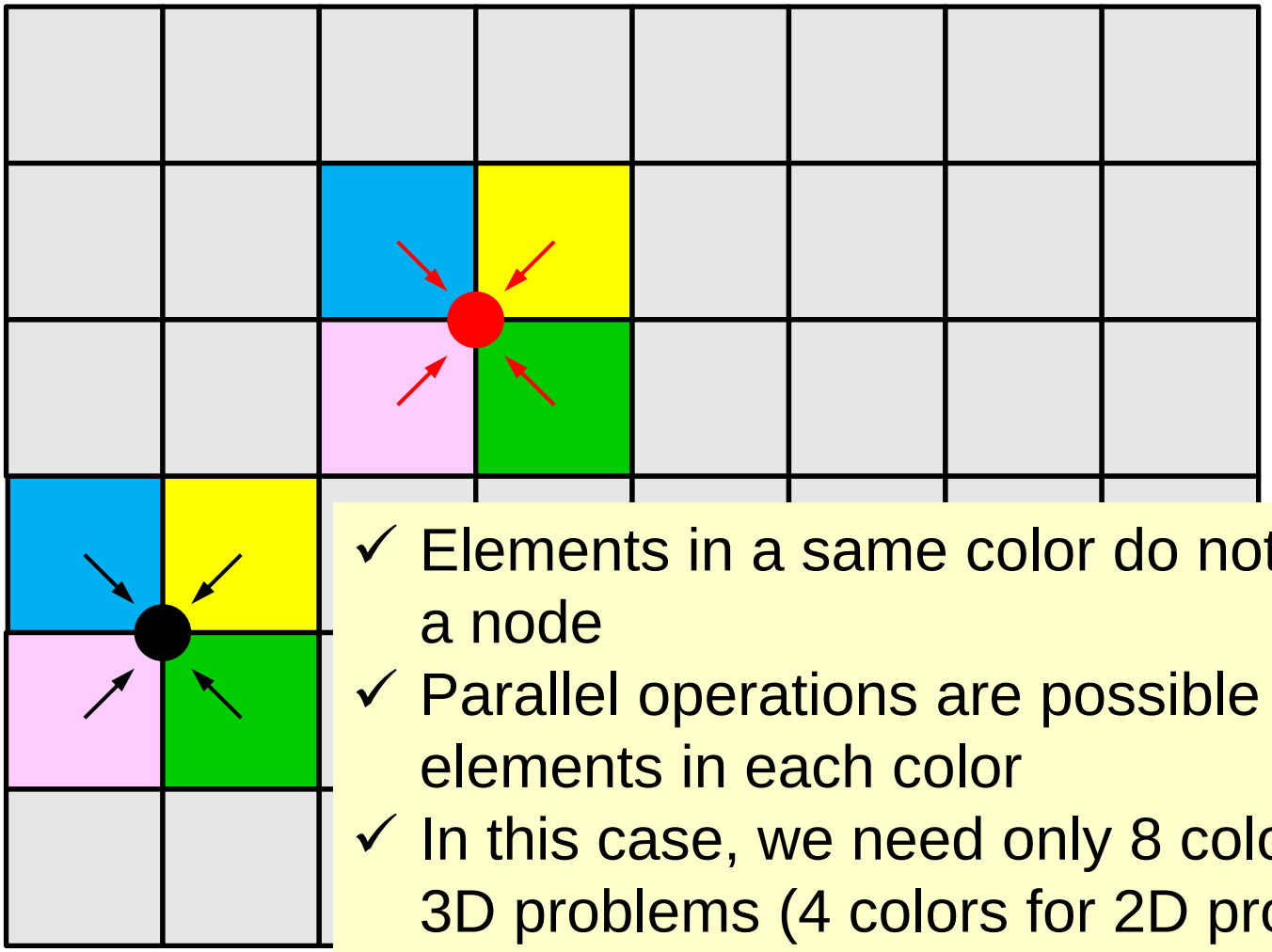
If 4 elements are trying to accumulate local element matrices to the global matrix simultaneously in parallel computing:

- ✓Results may be changed
- ✓Deadlock may occur

How to apply multi-threading

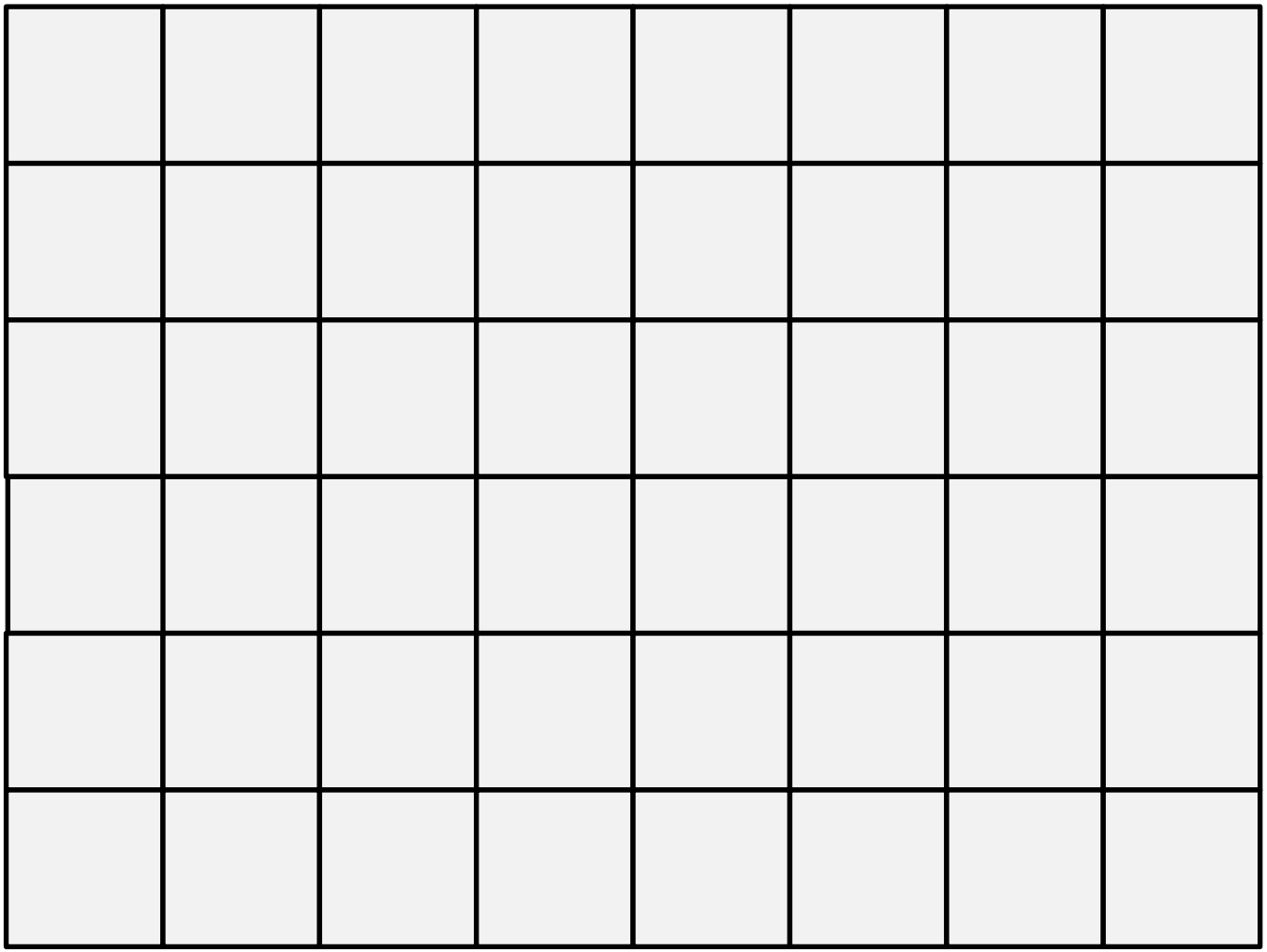
- CG Solver
  - Just insert OpenMP directives
  - ILU/IC preconditioning is much more difficult
- MAT_ASS (mat_ass_main, mat_ass_bc)
  - Coloring
    - Elements in a same color do not share a node
    - Parallel operations are possible for elements in each color
    - In this case, we need only 8 colors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

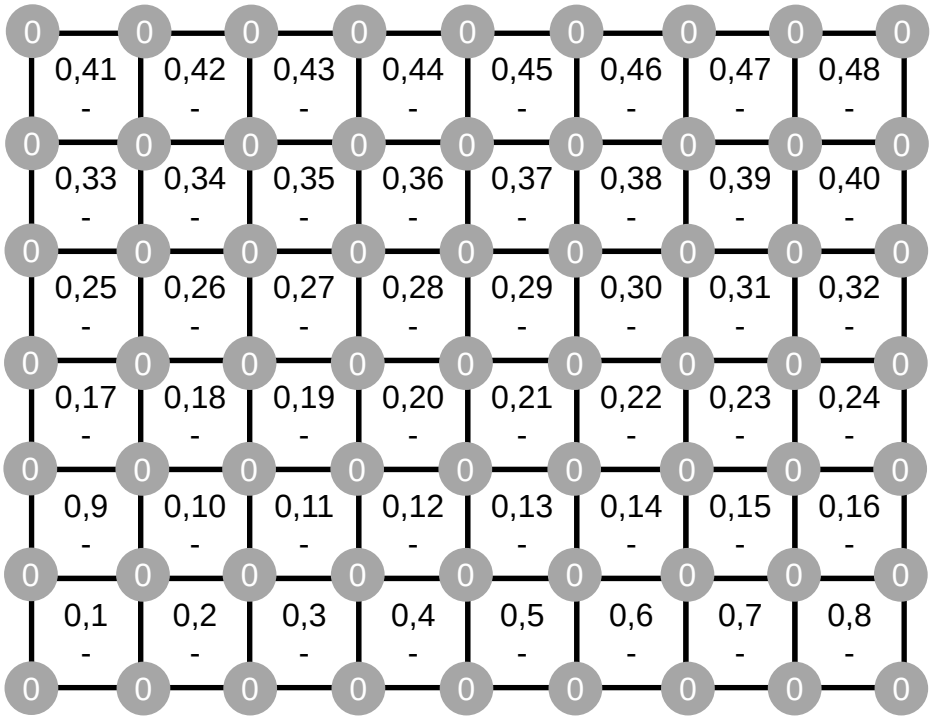
Target: 8x6= 48-meshes



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 =0: Not flagged in the current color =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



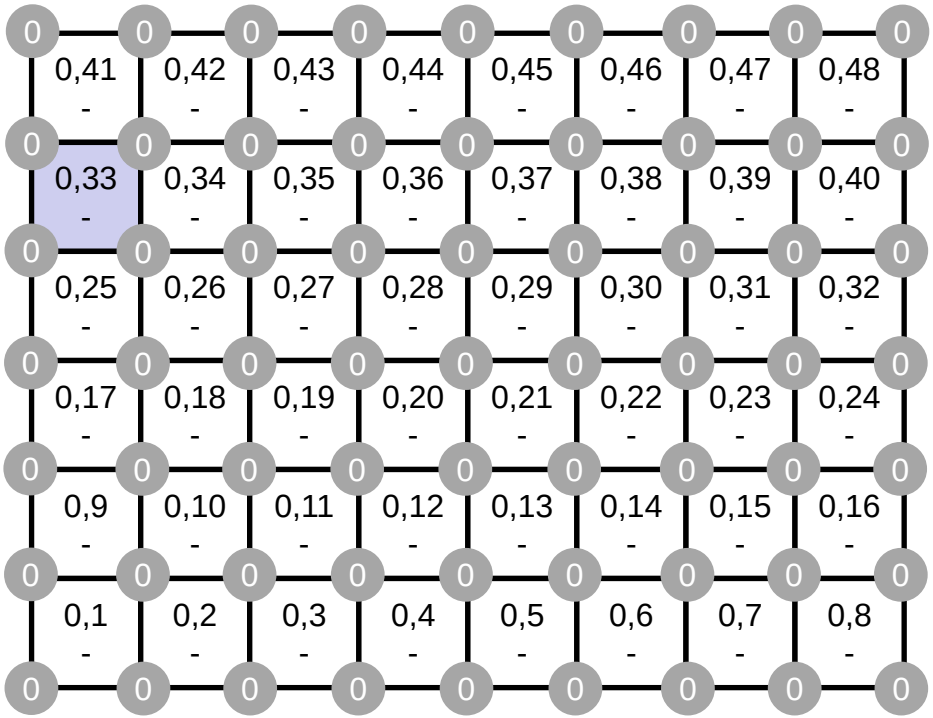
Coloring (2D) (1/7)

```
      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
          W1(in2)= 1
        enddo
      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)

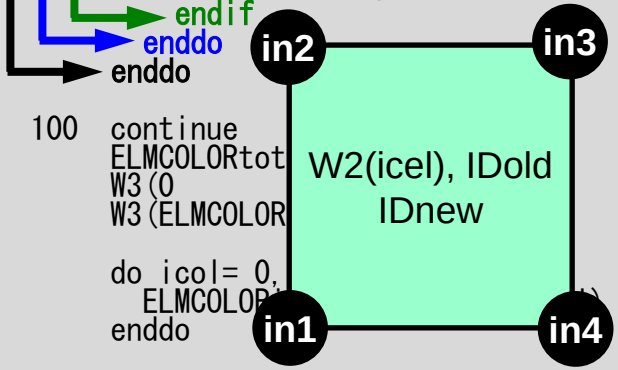
Name	Size	Content
ELMCOLORindex	**0:NP**	Number of Elements in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**0:NP**	Accumulated # of Colored Elem's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



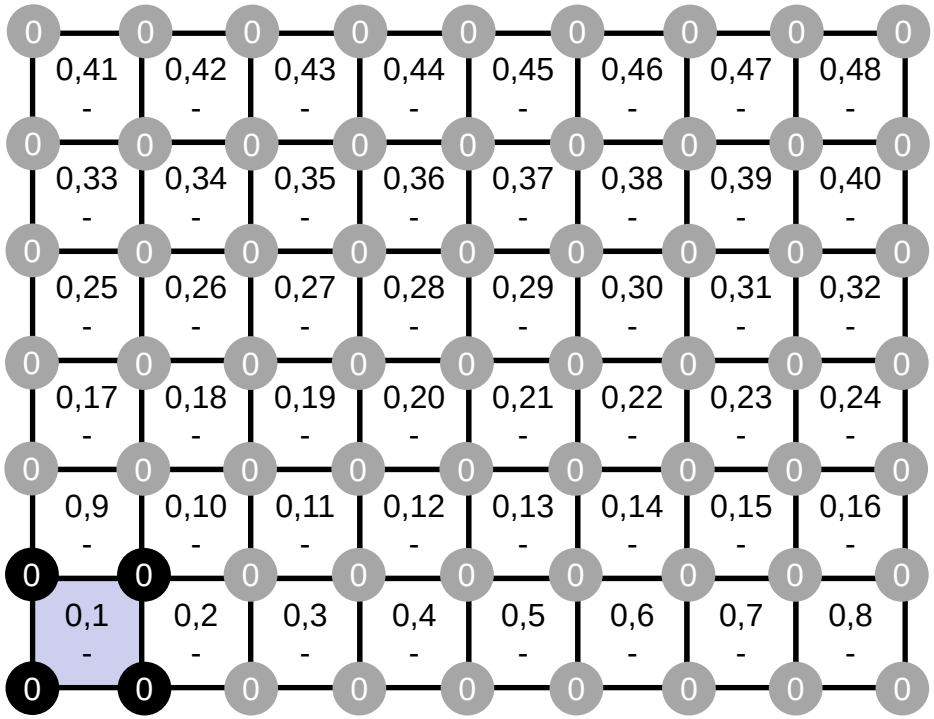
Coloring (2D) (2/7)

```
      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 =0: Not flagged in the current color =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

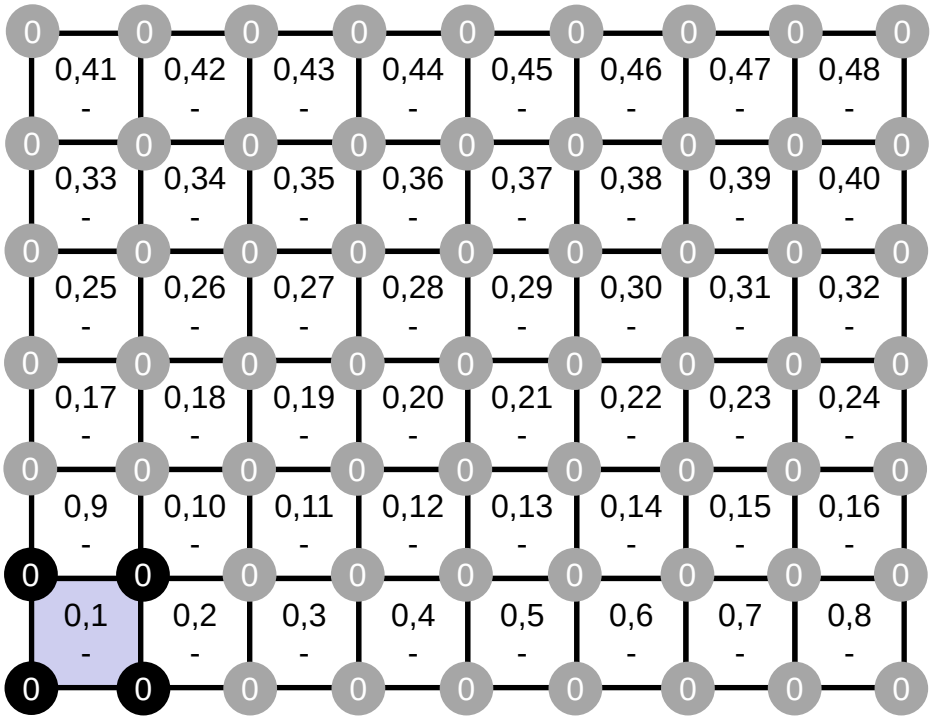


Coloring (2D) (2/7)

```
      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

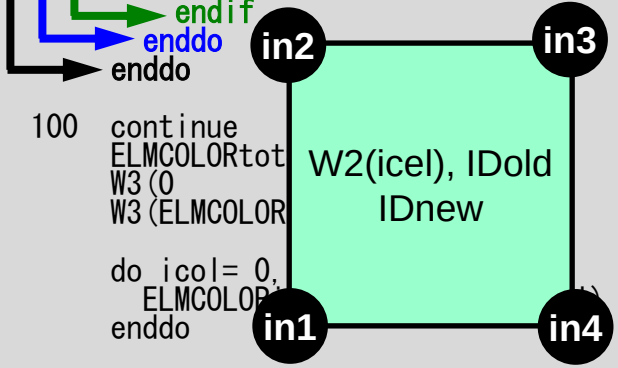
Name	Size	Content
ELMCOLORindex	**0:NP**	Number of Elements in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**0:NP**	Accumulated # of Colored Elem's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



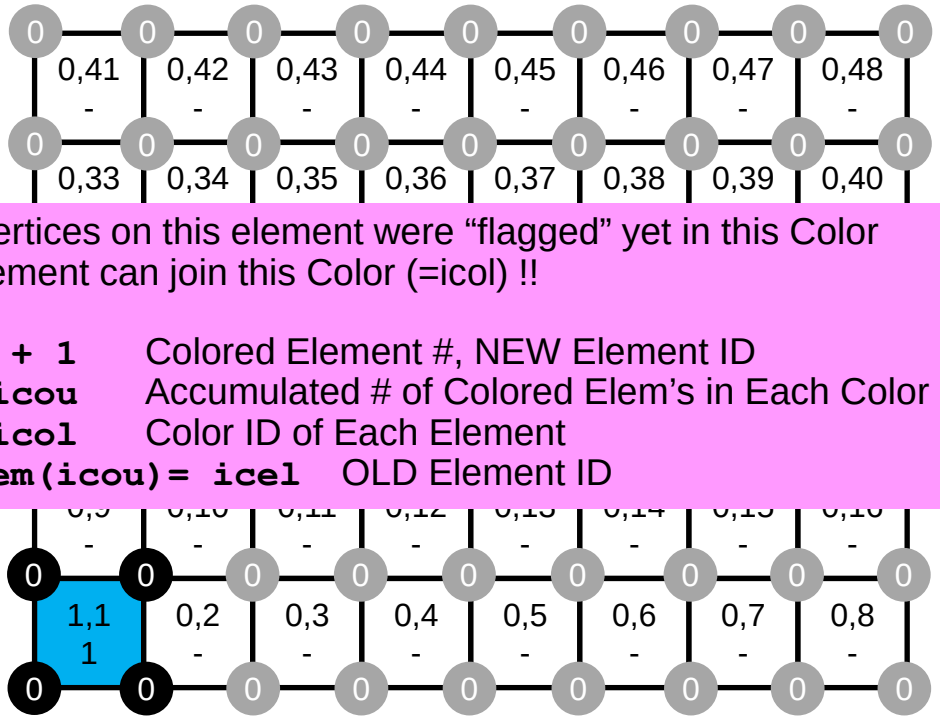
Coloring (2D) (2/7)

```
      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 =0: Not flagged in the current color =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



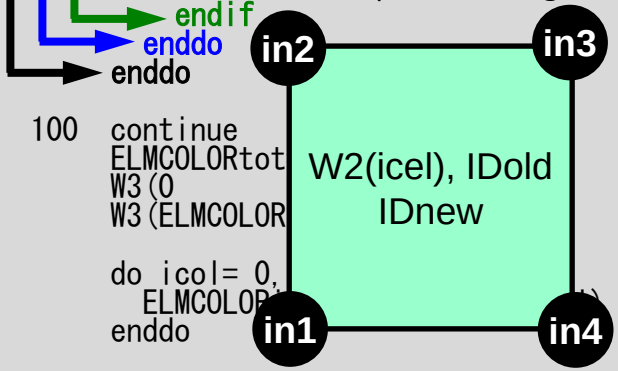
Because no vertices on this element were "flagged" yet in this Color (=icol), this element can join this Color (=icol) !!

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)

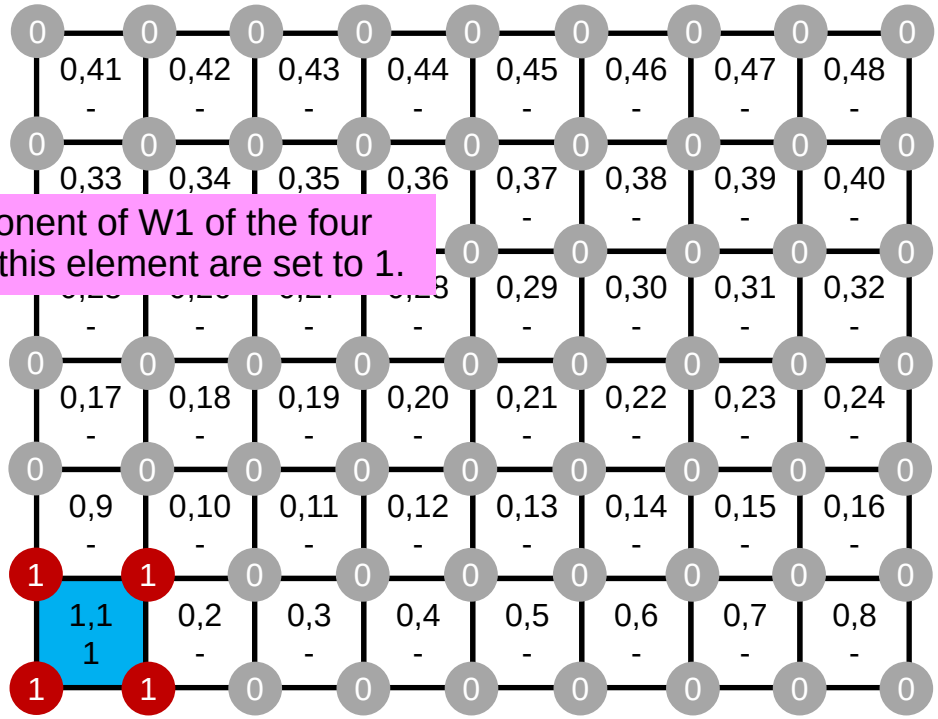
```
      allocate (ELMCOLORindex(0:NP))

      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)  (=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=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 =0: Not flagged in the current color =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.

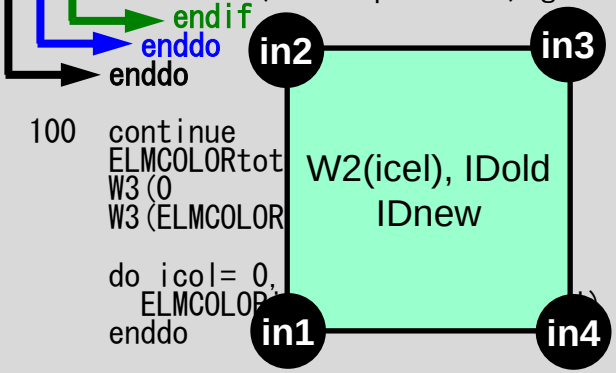


Coloring (2D) (3/7)

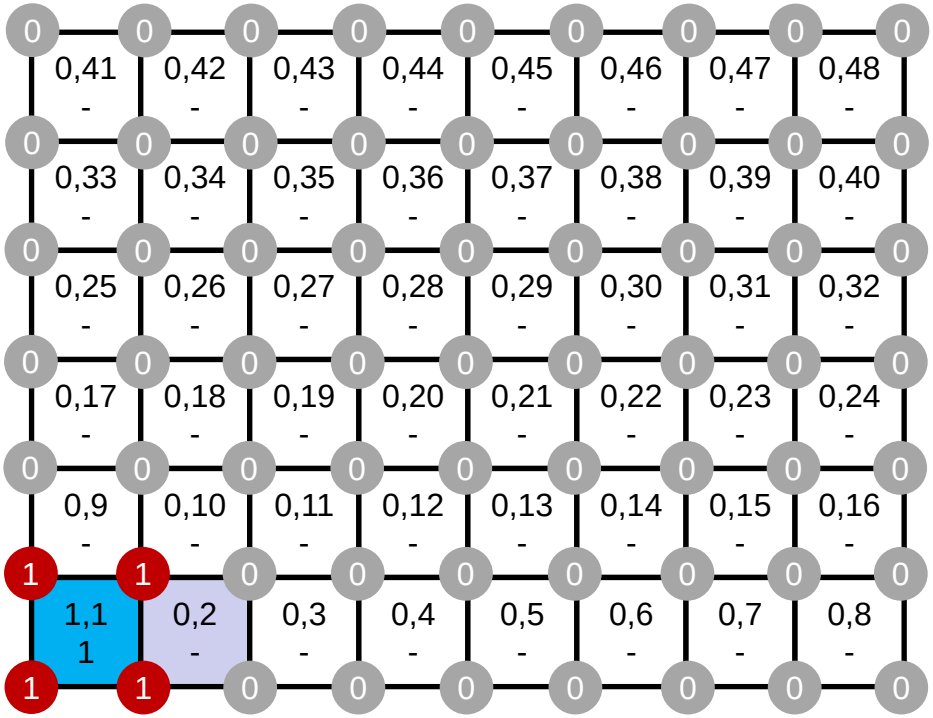
```
      allocate (ELMCOLORindex(0:NP))

      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)



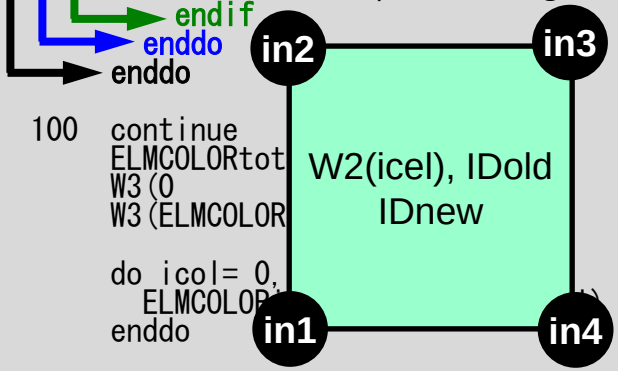
Name	Size	Content
ELMCOLORindex	**0:NP**	Number of Elements in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**0:NP**	Accumulated # of Colored Elem’s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



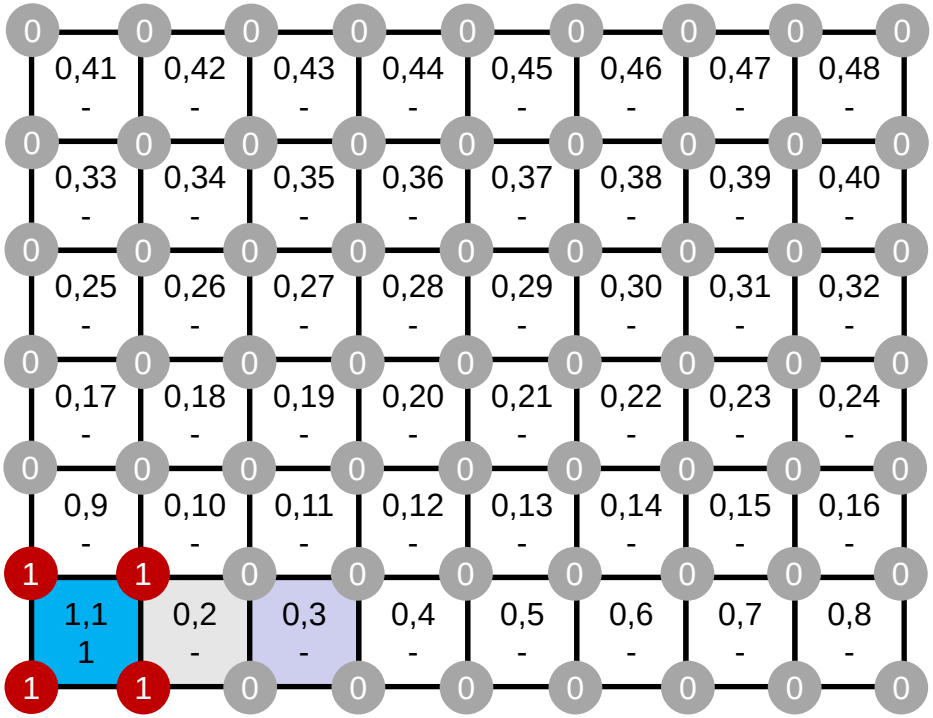
Coloring (2D) (4/7)

```
      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



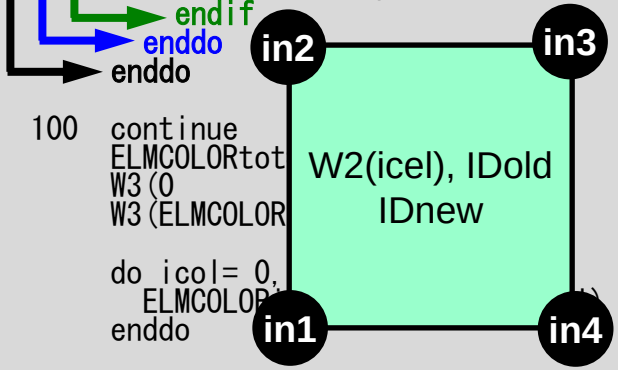
Name	Size	Content
ELMCOLORindex	**0:NP**	Number of Elements in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**0:NP**	Accumulated # of Colored Elem's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



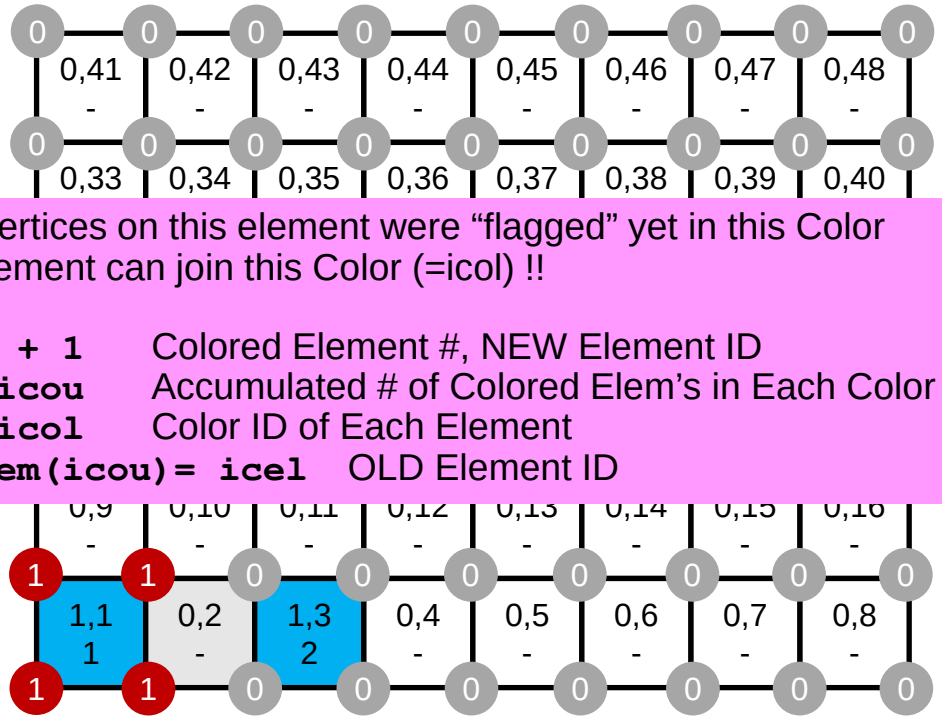
Coloring (2D) (4/7)

```
      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 =0: Not flagged in the current color =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



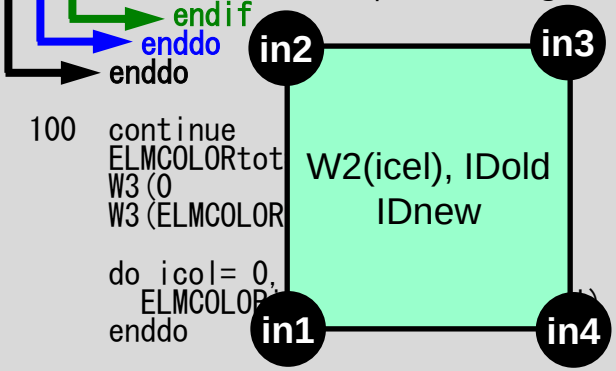
Because no vertices on this element were "flagged" yet in this Color (=icol), this element can join this Color (=icol) !!

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)

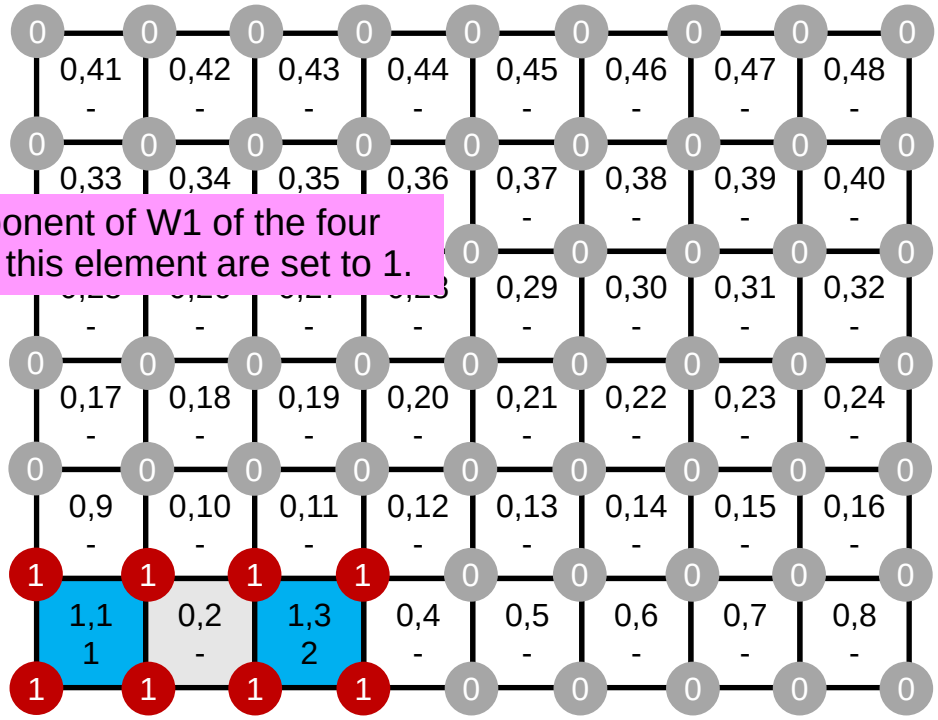
```
      allocate (ELMCOLORindex(0:NP))

      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) (=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 =0: Not flagged in the current color =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

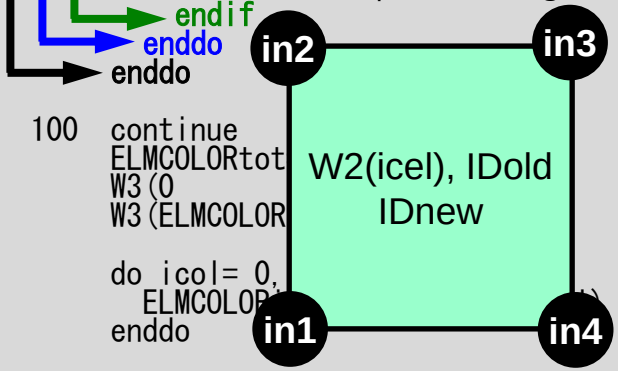
Each component of W1 of the four vertices on this element are set to 1.



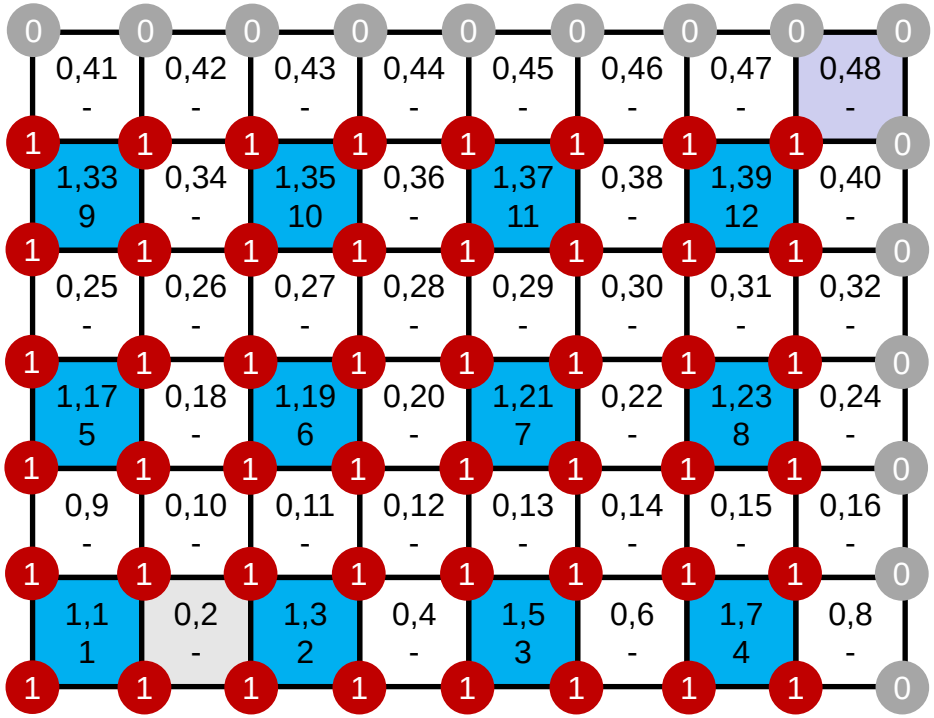
Coloring (2D) (5/7)

```
      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 =0: Not flagged in the current color =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



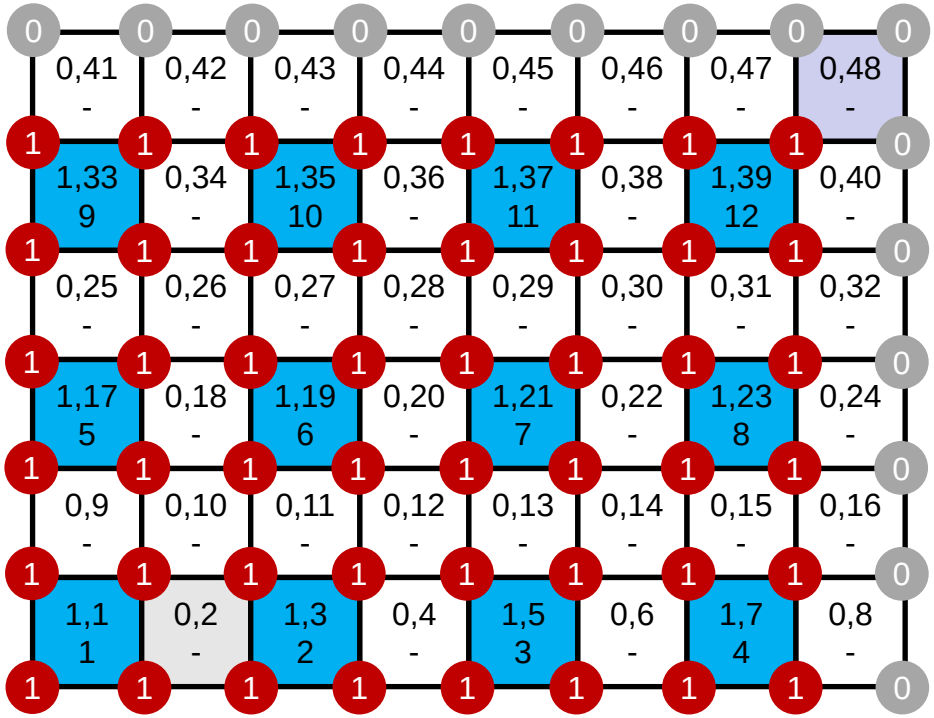
Coloring (2D) (5/7)

```
      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

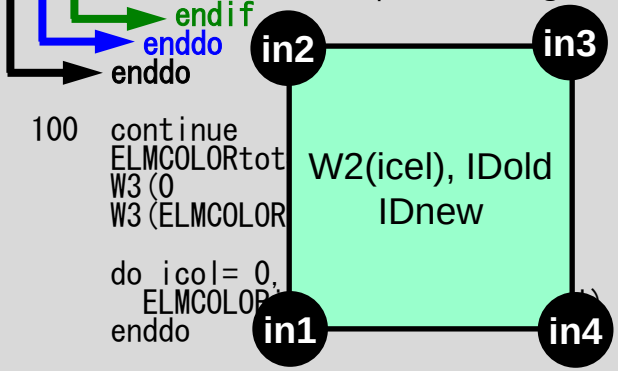
Name	Size	Content
ELMCOLORindex	**0:NP**	Number of Elements in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 (0, 1)	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 [0]	**ICELTOT**	Color ID of Each Element
W3	**0:NP**	Accumulated # of Colored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



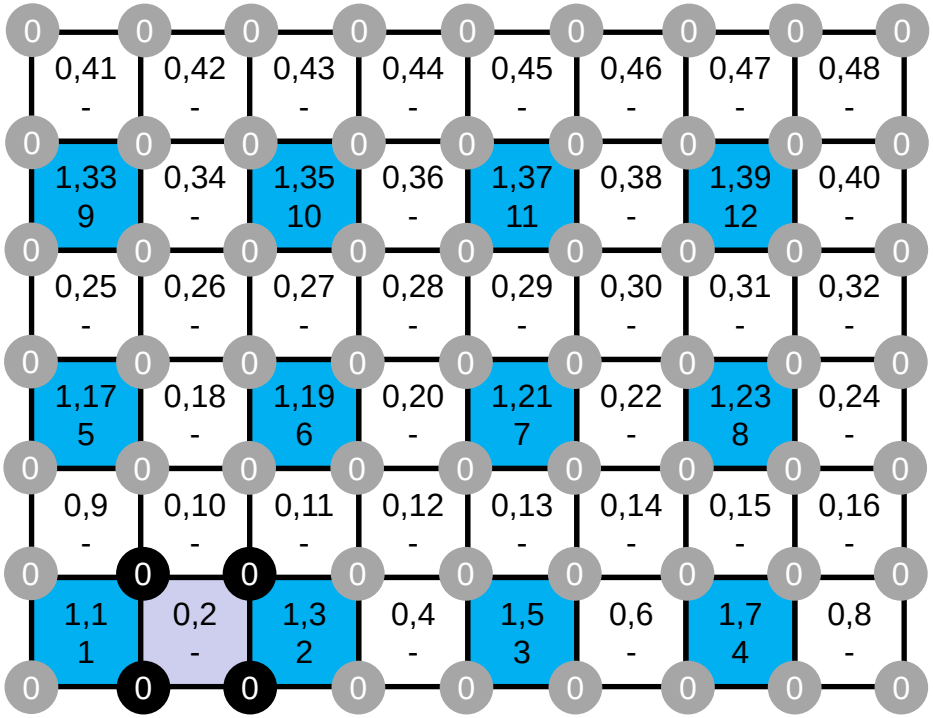
Coloring (2D) (6/7)

```
      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



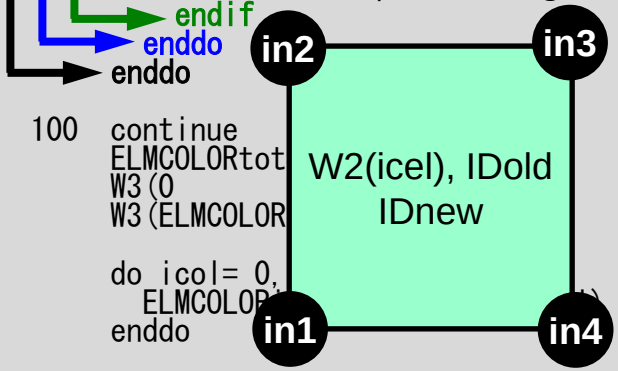
Name	Size	Content
ELMCOLORindex	**0:NP**	Number of Elements in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**0:NP**	Accumulated # of Colored Elem's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



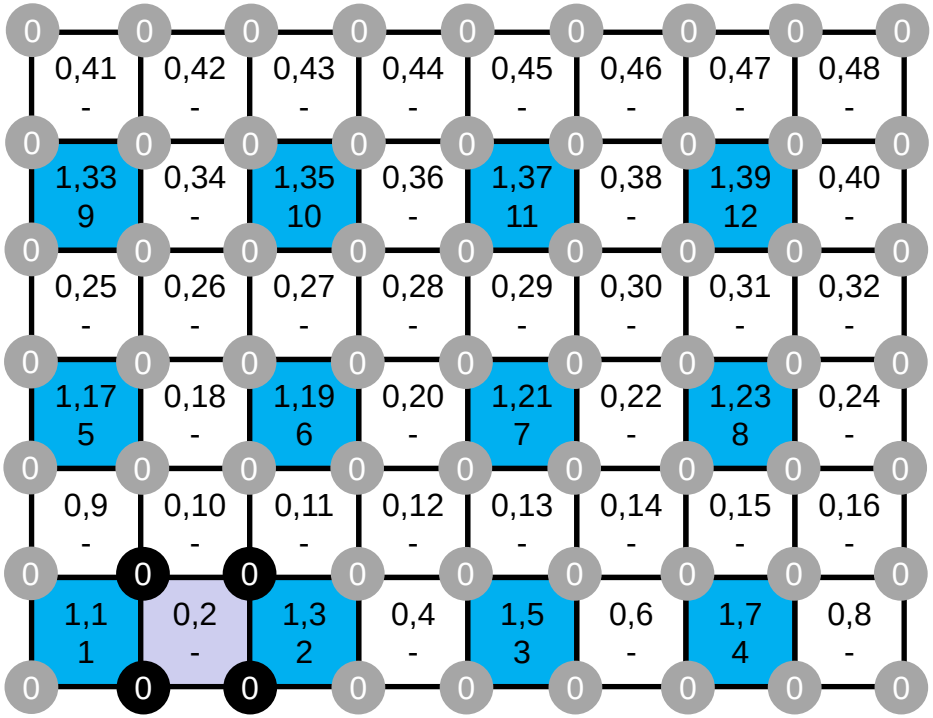
Coloring (2D) (6/7)

```
      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



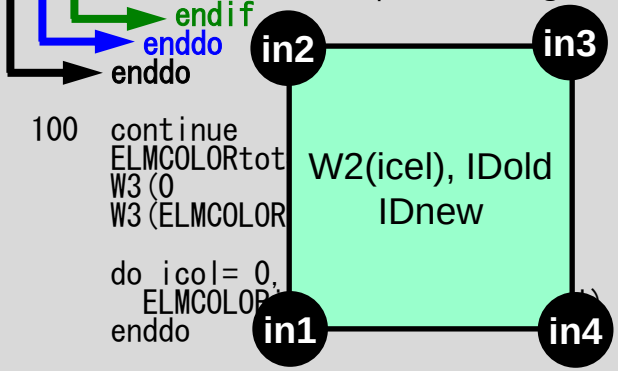
Name	Size	Content
ELMCOLORindex	**0:NP**	Number of Elements in Each Color
ELMCOLORitem	**ICELTOT**	OLD Element ID in Each Color
W1 0 1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2 0	**ICELTOT**	Color ID of Each Element
W3	**0:NP**	Accumulated # of Colored Elem's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



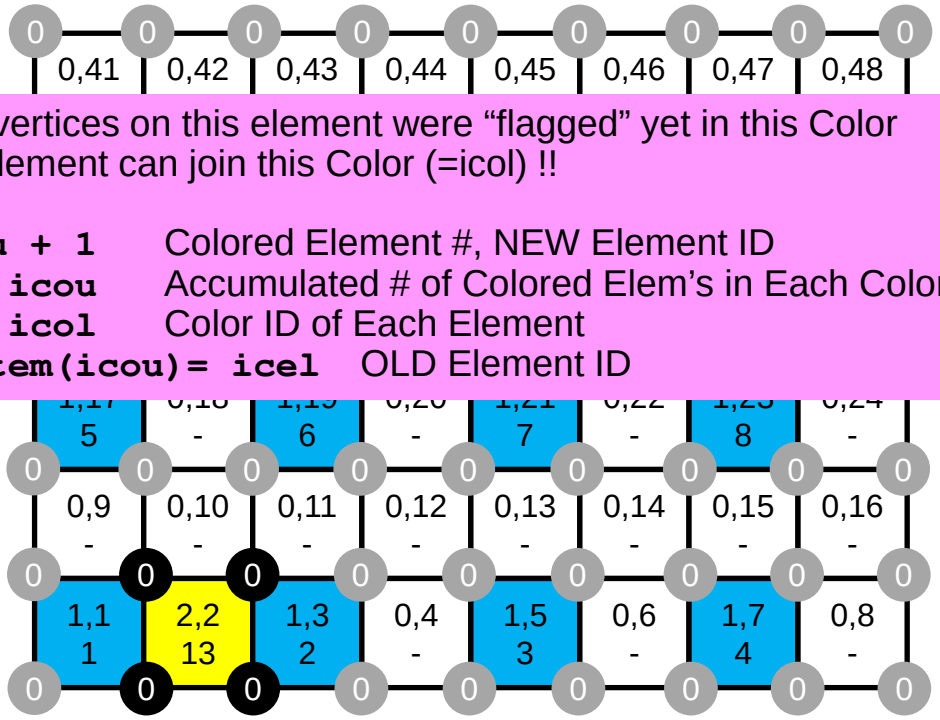
Coloring (2D) (6/7)

```
      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 =0: Not flagged in the current color =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



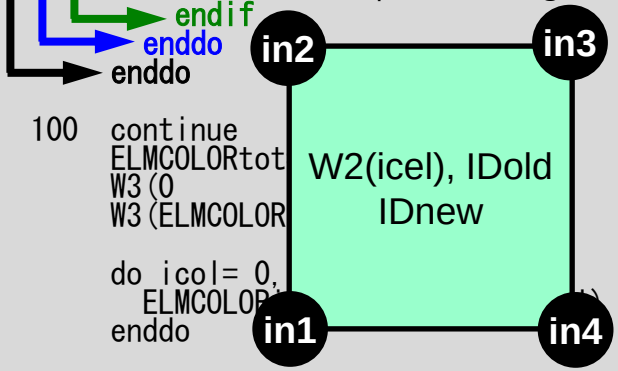
Because no vertices on this element were "flagged" yet in this Color (=icol), this element can join this Color (=icol) !!

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)

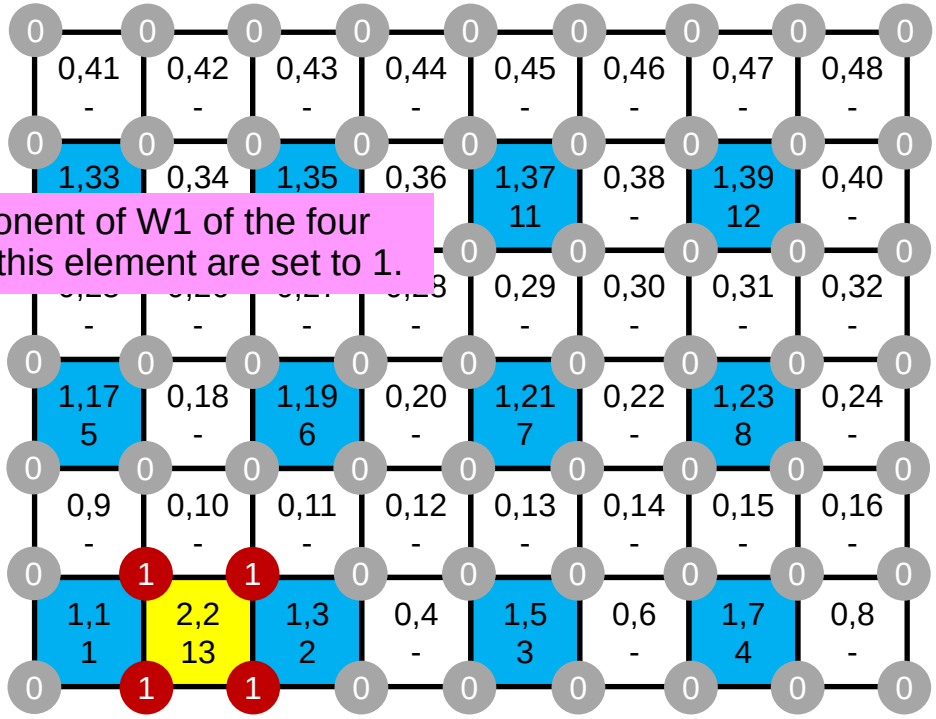
```
      allocate (ELMCOLORindex(0:NP))

      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) (=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 =0: Not flagged in the current color =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

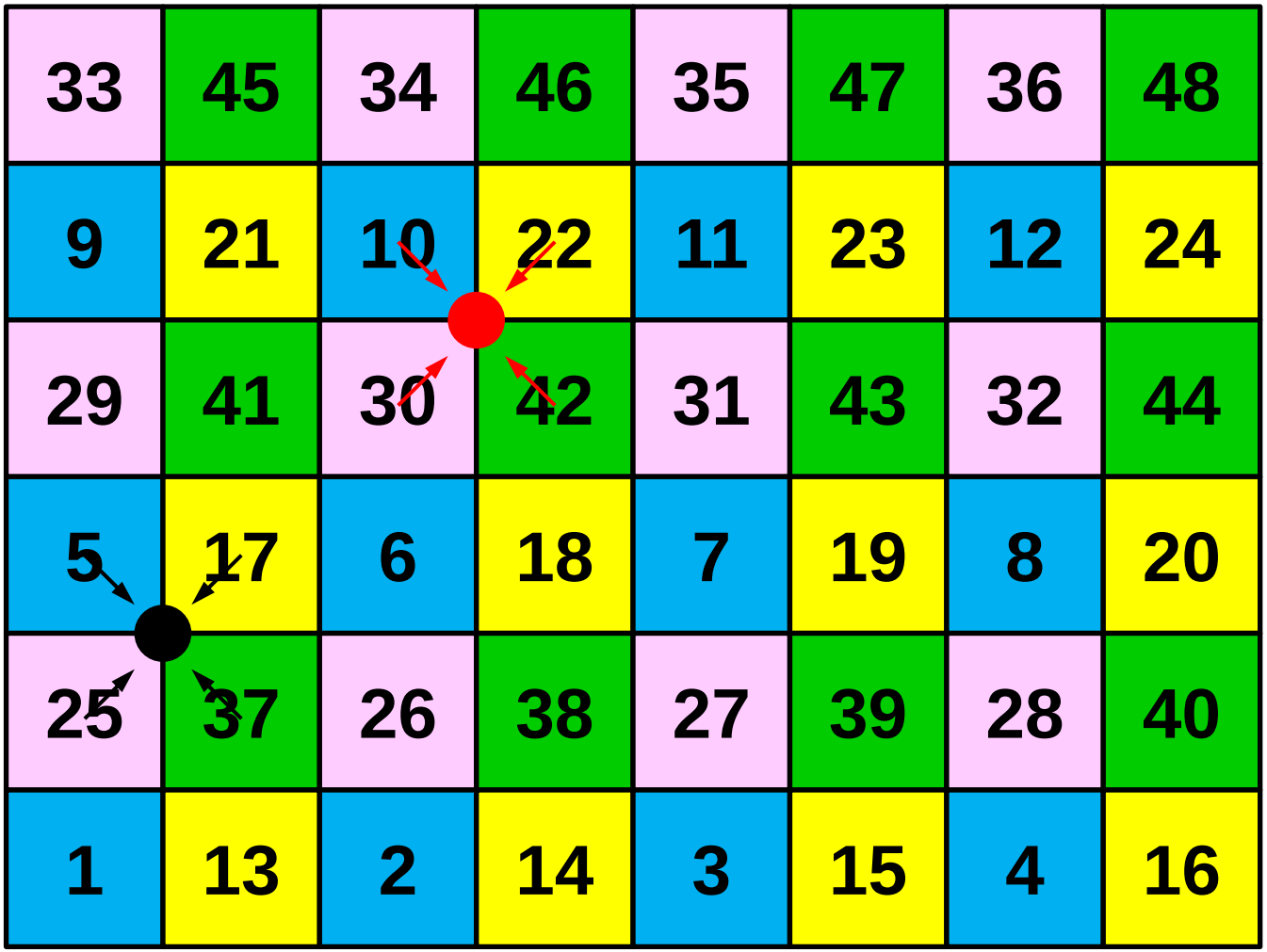
Each component of W1 of the four vertices on this element are set to 1.



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 =0: Not flagged in the current color =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,QVO,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
W1	**NP**	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2	**ICELTOT**	Color ID of Each Element
W3	**0:NP**	Accumulated # of Colored Elem's 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