

並列有限要素法による
三次元定常熱伝導解析プログラム
OpenMP+ハイブリッド並列化(1/2)
Fortran編

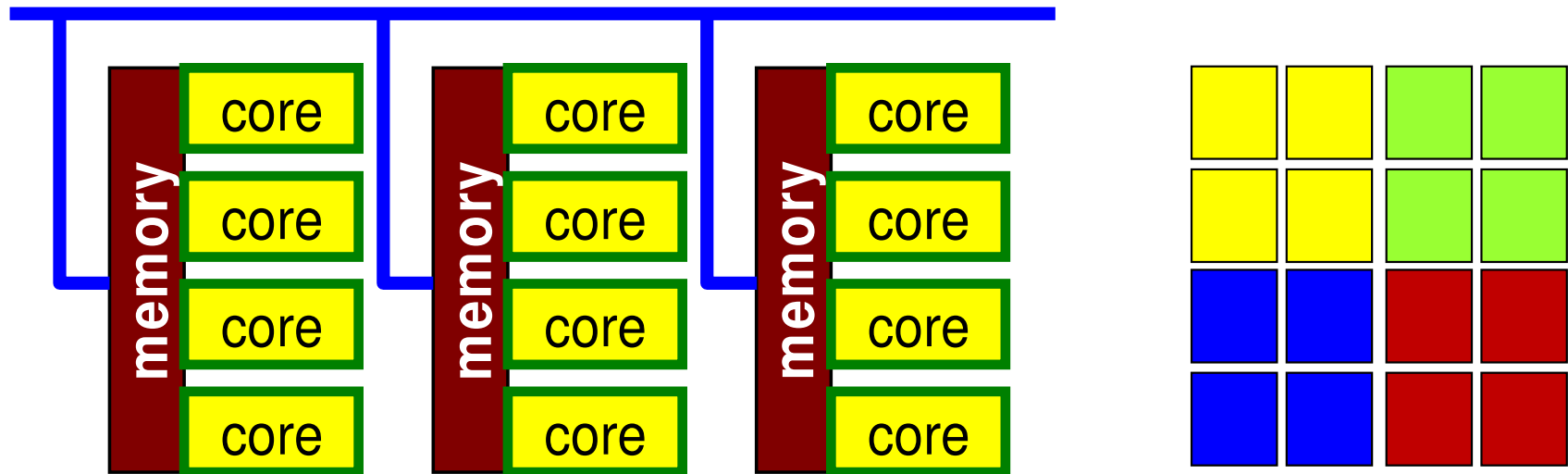
中島 研吾
東京大学情報基盤センター

Hybrid並列プログラミング

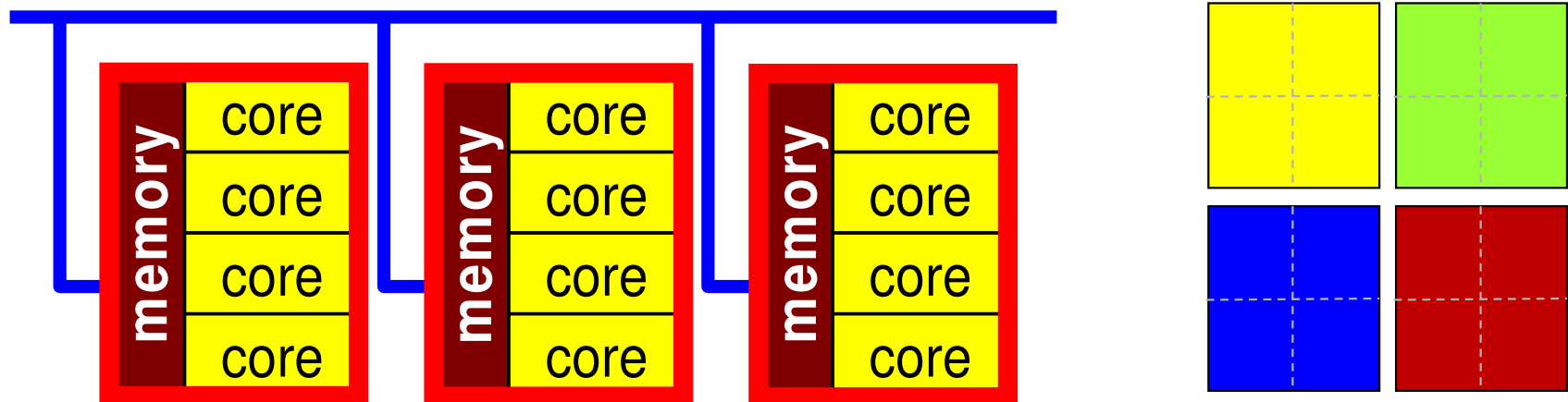
- スレッド並列+メッセージパッシング
 - OpenMP+ MPI
 - CUDA + MPI, OpenACC + MPI
- OpenMPがMPIより簡単ということはない
 - データ依存性のない計算であれば、機械的にOpenMP指示文を入れれば良い
 - NUMAになるとより複雑：First Touch Data Placement

Flat MPI vs. Hybrid

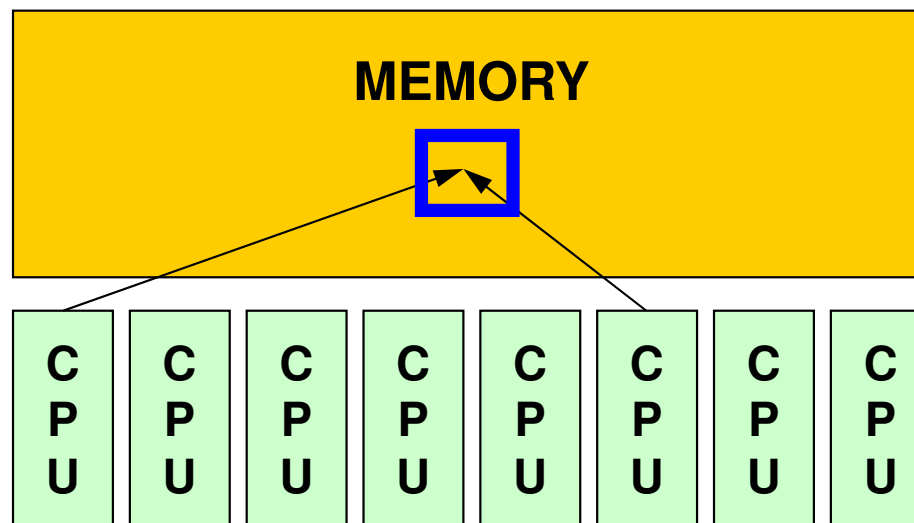
Flat-MPI: Each Core -> Independent



Hybrid: Hierarchical Structure



共有メモリ型計算機



- SMP

- Symmetric Multi Processors

- 複数のCPU(コア)で同じメモリ空間を共有するアーキテクチャ

OpenMPとは(1/2)

<http://www.openmp.org>

- 共有メモリ型並列計算機用のDirectiveの統一規格
 - MPIやHPFに比べると遅く1996年頃から活動開始
 - 現在 Ver.4.X
- 背景
 - CrayとSGIの合併(1996)
 - ASCI計画の開始(1995)
 - Accelerated Strategic Computing Initiative (ASCI) -> Advanced Simulation and Computing Program (ASC)
 - ASCI: 核実験のシミュレーションによる代替
 - 計算機開発, シミュレーションソフトウェア
 - SMPクラスタにフォーカス
 - ASCI Red (Intel), Blue Pacific/White/Purple/BlueGene (IBM), Blue Mountain (SGI)
 - SMPクラスタ向けの並列プログラミングの共通API (Application Program Interface)が必要

OpenMPとは(2/2)

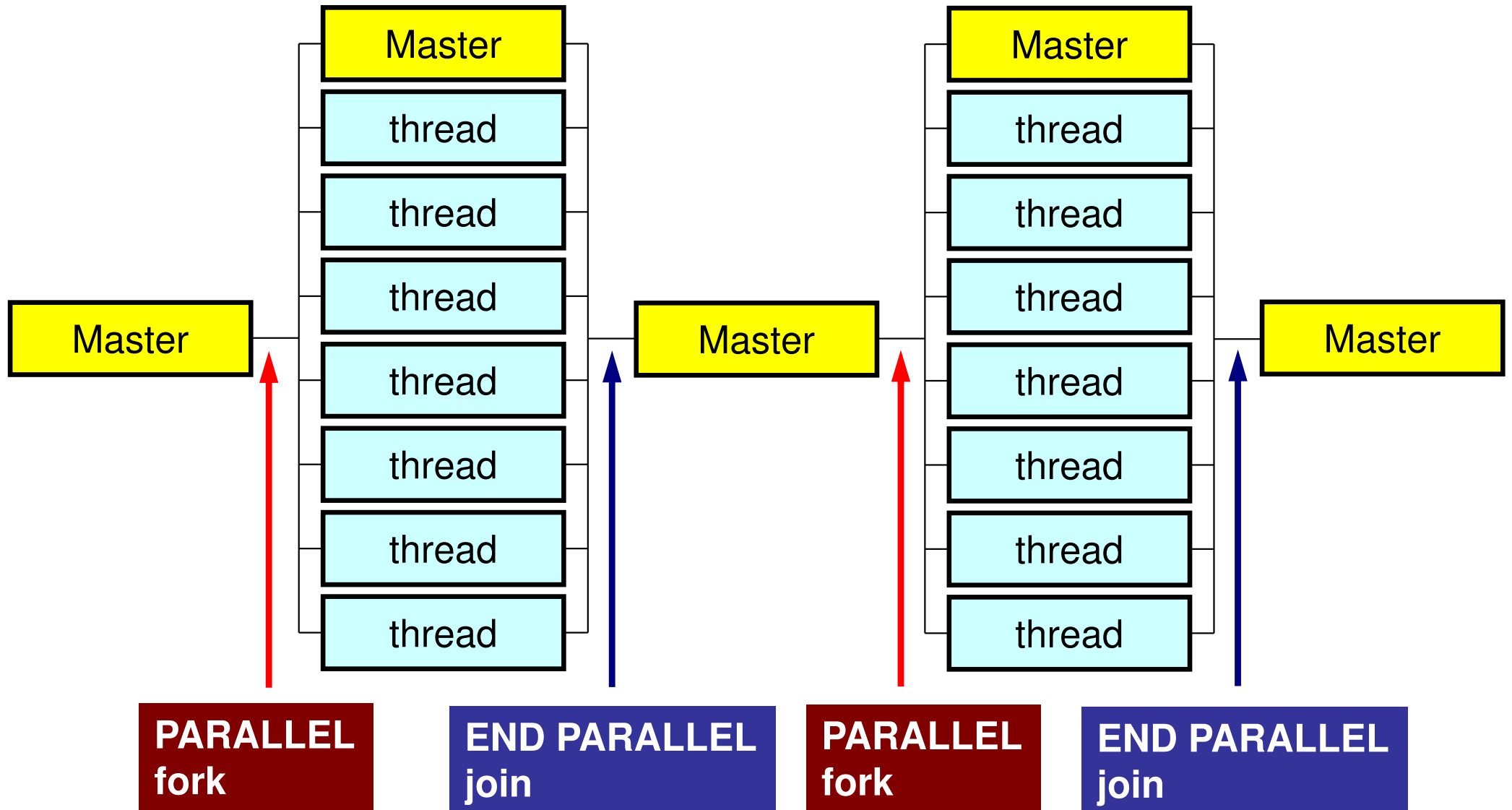
<http://www.openmp.org>

- 主な計算機ベンダーが集まって [OpenMP ARB](#)を結成し、1997年にはもう規格案ができていたそうである
 - SC98ではすでにOpenMPのチュートリアルがあったし、すでにSGI Origin2000でOpenMP-MPIハイブリッドのシミュレーションをやっている例もあった。
- OpenMPはFortran版とC/C++版の規格が全く別々に進められてきた。
 - Ver.2.5で言語間の仕様を統一
- Ver.4.0ではGPU, Intel-MIC等Co-Processor, Accelerator環境での動作も考慮
 - OpenACCに近づいている

OpenMPの概要

- 基本的仕様
 - プログラムを並列に実行するための動作をユーザーが明示
 - OpenMP実行環境は、依存関係、衝突、デッドロック、競合条件、結果としてプログラムが誤った実行につながるような問題に関するチェックは要求されていない。
 - プログラムが正しく実行されるよう構成するのはユーザーの責任である。
- 実行モデル
 - fork-join型並列モデル
 - 当初はマスタスレッドと呼ばれる単一プログラムとして実行を開始し、「PARALLEL」、「END PARALLEL」ディレクティブの対で並列構造を構成する。並列構造が現れるとマスタスレッドはスレッドのチームを生成し、そのチームのマスタとなる。
 - いわゆる「入れ子構造」も可能であるが、ここでは扱わない

Fork-Join 型並列モデル



スレッド数

- 環境変数 **OMP_NUM_THREADS**

- 値の変え方

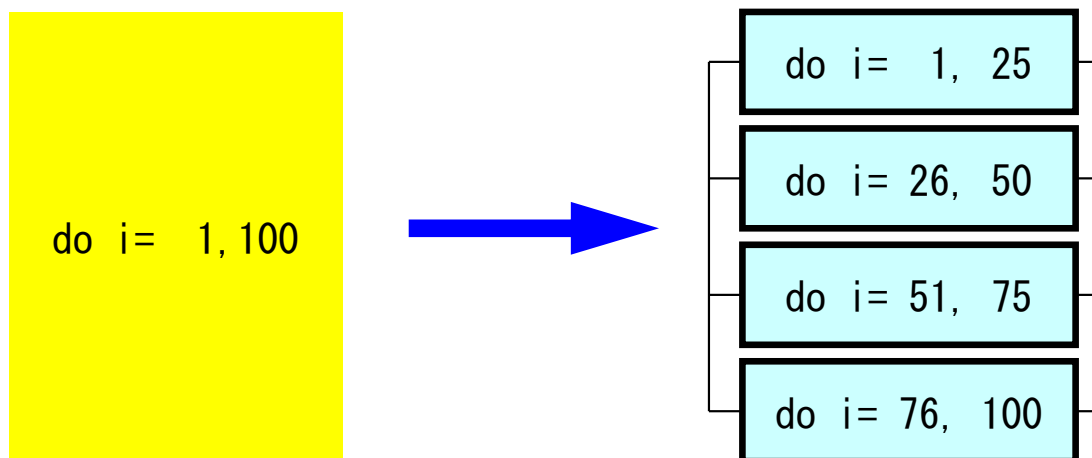
- bash(.bashrc)

- ```
export OMP_NUM_THREADS=8
```

- csh(.cshrc)

- ```
setenv OMP_NUM_THREADS 8
```

- たとえば, **OMP_NUM_THREADS=4**とすると, 以下のように **i=1~100**のループが4分割され, 同時に実行される。



OpenMPに関する情報

- OpenMP Architecture Review Board (ARB)
 - <http://www.openmp.org>
- 参考文献
 - 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)最新!
- 富士通他による翻訳：（OpenMP 3.0）必携！
 - <http://www.openmp.org/mp-documents/OpenMP30spec-ja.pdf>

OpenMPの特徴

- ディレクティブ（指示行）の形で利用
 - 挿入直後のループが並列化される
 - コンパイラがサポートしていなければ, コメントとみなされる

OpenMP/Directives

Array Operations

Simple Substitution

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

Dot Products

```
!$omp parallel do private(i)
!$omp&      reduction(+:RH0)
  do i= iS, iE
    RH0= RH0 + W(i, R)*W(i, Z)
  enddo
!$omp end parallel do
```

DAXPY

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

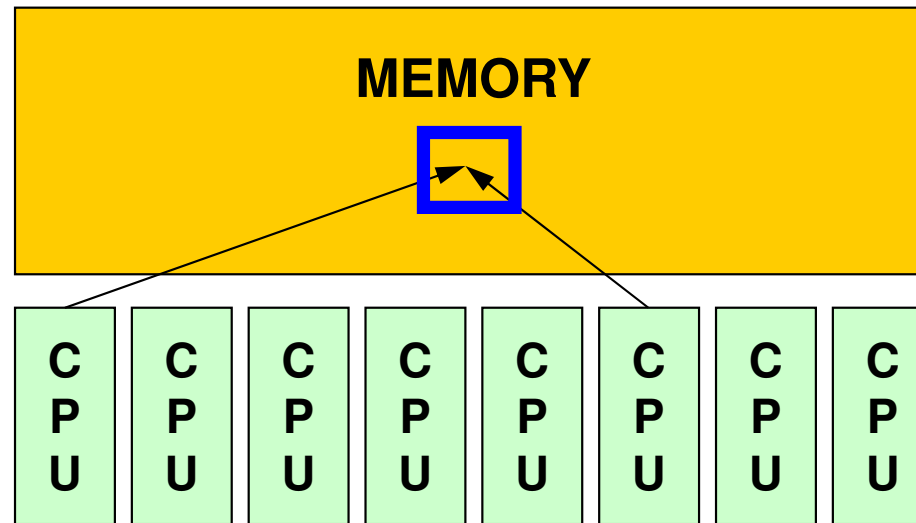
OpenMP/Directives Matrix/Vector Products

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

OpenMPの特徴

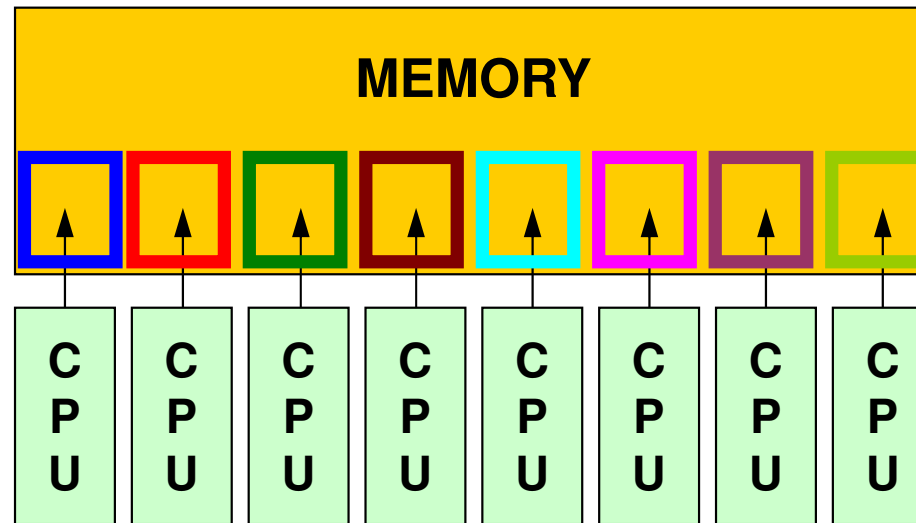
- ディレクティブ（指示行）の形で利用
 - 挿入直後のループが並列化される
 - コンパイラがサポートしていなければ、コメントとみなされる
- **何も指定しなければ、何もしない**
 - 「自動並列化」, 「自動ベクトル化」とは異なる。
 - 下手なことをするとおかしな結果になる: ベクトル化と同じ
 - データ分散等 (Ordering) は利用者の責任
- 共有メモリユニット内のプロセッサ数に応じて、「Thread」が立ち上がる
 - 「Thread」: MPIでいう「プロセス」に相当する。
 - 普通は「Thread数 = 共有メモリユニット内プロセッサ数, コア数」であるが最近のアーキテクチャではHyper Threading (HT) がサポートされているものが多い (1コアで2-4スレッド)

メモリ競合 (Memory Contention)



- 複雑な処理をしている場合，複数のスレッドがメモリ上の同じアドレスにあるデータを同時に更新する可能性がある。
 - 複数のCPUが配列の同じ成分を更新しようとする。
 - メモリを複数のコアで共有しているためこのようなことが起こりうる。
 - 場合によっては答えが変わる

メモリ競合 (Memory Contention) (続き)



- 本演習で扱っている例は，そのようなことが生じないよう，各スレッドが同時に同じ成分を更新するようなことはないようにする（実はそもそも無い）。
 - これはユーザーの責任でやること，である。
- データ依存性 (Data Dependency)
- コア数 (スレッド数) が増えるほど，メモリへの負担が増えて，処理速度は低下する (メモリ飽和)。

OpenMPの特徴(続き)

- 基本は「!omp parallel for」～「!omp end parallel do」
- 変数について, グローバル/sharedな変数と, Thread内でローカルな「private」な変数に分けられる。
 - デフォルトは「global/shared」
 - 内積を求める場合は「reduction」を使う

```
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
```

W(:, :), R, Z, PEsmptOT
などはグローバル変数

FORTRANとC

```
use omp_lib
```

```
!$omp parallel do shared(n, x, y) private(i)  
  do i= 1, n  
    x(i)= x(i) + y(i)  
  enddo  
!$ omp end parallel do
```

```
#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];  
}
```

本講義における方針

- OpenMPは多様な機能を持っているが、それらの全てを逐一教えることはしない。
 - 講演者も全てを把握、理解しているわけではない。
- 数値解析に必要な最低限の機能のみ学習する。
 - 具体的には、講義で扱っているPCG法によるポアソン方程式ソルバーを動かすために必要な機能のみについて学習する
 - それ以外の機能については、自習、質問のこと(全てに答えられるとは限らない)。

最初にやること

- `use omp_lib` FORTRAN
- `#include <omp.h>` C
- 様々な環境変数, インタフェースの定義 (OpenMP3.0以降でサポート)

OpenMPディレクティブ (FORTRAN)

```
sentinel directive_name [clause[,] clause...]
```

- 大文字小文字は区別されない。
- sentinel
 - 接頭辞
 - FORTRANでは「!\$OMP」、「C\$OMP」、「*\$OMP」、但し自由ソース形式では「!\$OMP」のみ。
 - 継続行にはFORTRANと同じルールが適用される。以下はいずれも「!\$OMP PARALLEL DO SHARED(A,B,C)」

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

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

OpenMPディレクティブ(C)

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

- 継続行は「\」
- 小文字を使用(変数名以外)

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

PARALLEL DO

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

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

- 多重スレッドによって実行される領域を定義し、DOループの並列化を実施する。
- 並び項目 (clause) : よく利用するもの
 - private (list)
 - shared (list)
 - default (private|shared|none)
 - reduction ({operation|intrinsic}: list)

REDUCTION

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

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

- 「MPI_REDUCE」のようなものと思えばよい
- Operator
 - +, *, -, .AND., .OR., .EQV., .NEQV.
- Intrinsic
 - MAX, MIN, IAND, IOR, IEQR

実例A1: 簡単なループ

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

- ループの繰り返し変数(ここでは「i」)はデフォルトで private なので, 明示的に宣言は不要。
- 「END PARALLEL DO」は省略可能。
 - C言語ではそもそも存在しない

実例A2: 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」は省略可能。

OpenMP使用時に呼び出すことのできる 関数群

関数名	内容
<code>int omp_get_num_threads (void)</code>	スレッド総数
<code>int omp_get_thread_num (void)</code>	自スレッドのID
<code>double omp_get_wtime (void)</code>	MPI_Wtimeと同じ
<code>void omp_set_num_threads (int num_threads)</code> <code>call omp_set_num_threads (num_threads)</code>	スレッド数設定

OpenMPを適用するには？(内積)

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

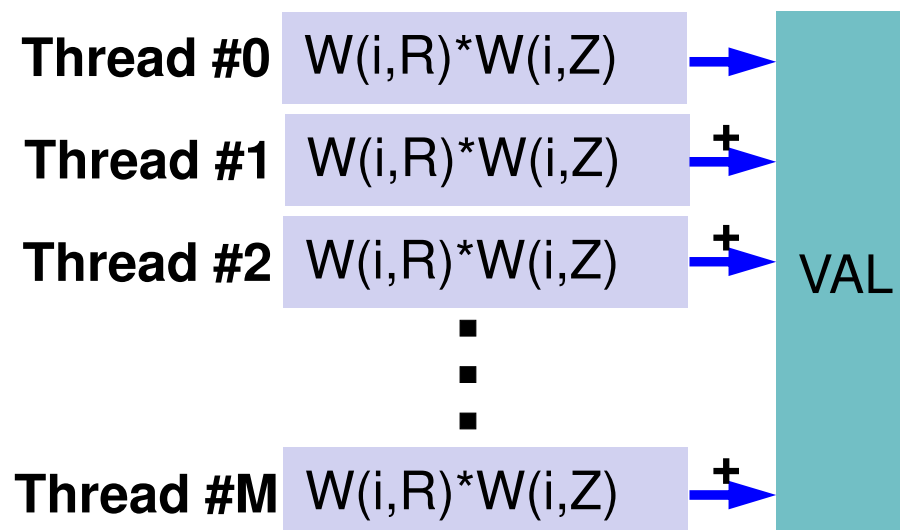
OpenMPを適用するには？(内積)

```
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
```

OpenMPディレクティブの挿入
これでも並列計算は可能



OpenMPを適用するには？(内積)

```
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
```

OpenMPディレクティブの挿入
これでも並列計算は可能



```
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
```

多重ループの導入
PEsmptOT:スレッド数
あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施
(別に効率がよくなるわけではない)

OpenMPを適用するには？(内積)

```
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
```

OpenMPディレクティブの挿入
これでも並列計算は可能



```
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
```

多重ループの導入
PEsmptOT:スレッド数
あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施

PEsmptOT個のスレッドが立ち上がり、
並列に実行

OpenMPを適用するには？(内積)

```
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
```

多重ループの導入

PEsmptOT:スレッド数

あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施

PEsmptOT個のスレッドが立ち上がり、
並列に実行

各要素が計算されるスレッドを
指定できる

例えば, N=100, PEsmptOT=4とすると:

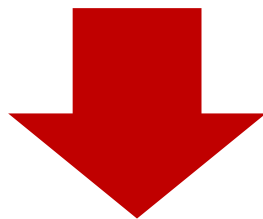
```
INDEX(0)= 0
INDEX(1)= 25
INDEX(2)= 50
INDEX(3)= 75
INDEX(4)= 100
```


Matrix-Vector Multiply

```

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

```



```

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

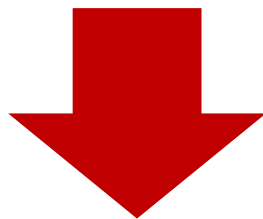
```

Matrix-Vector Multiply

```

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

```



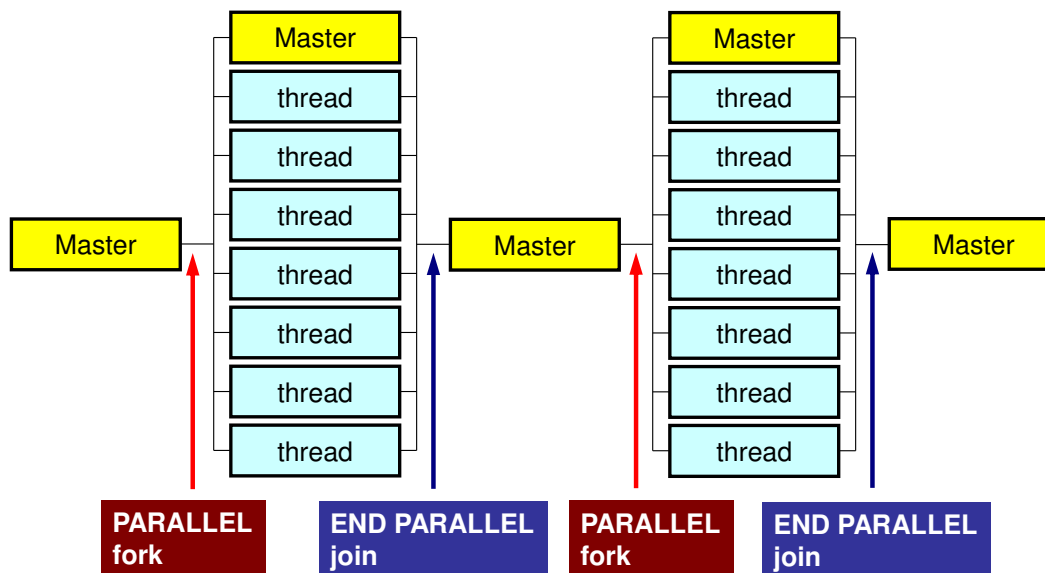
```

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

```

omp parallel (do)

- omp parallel-omp end parallelはそのたびにスレッドを生成, 消滅させる : fork-join
- ループが連続するとオーバーヘッドになる。
- 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
```

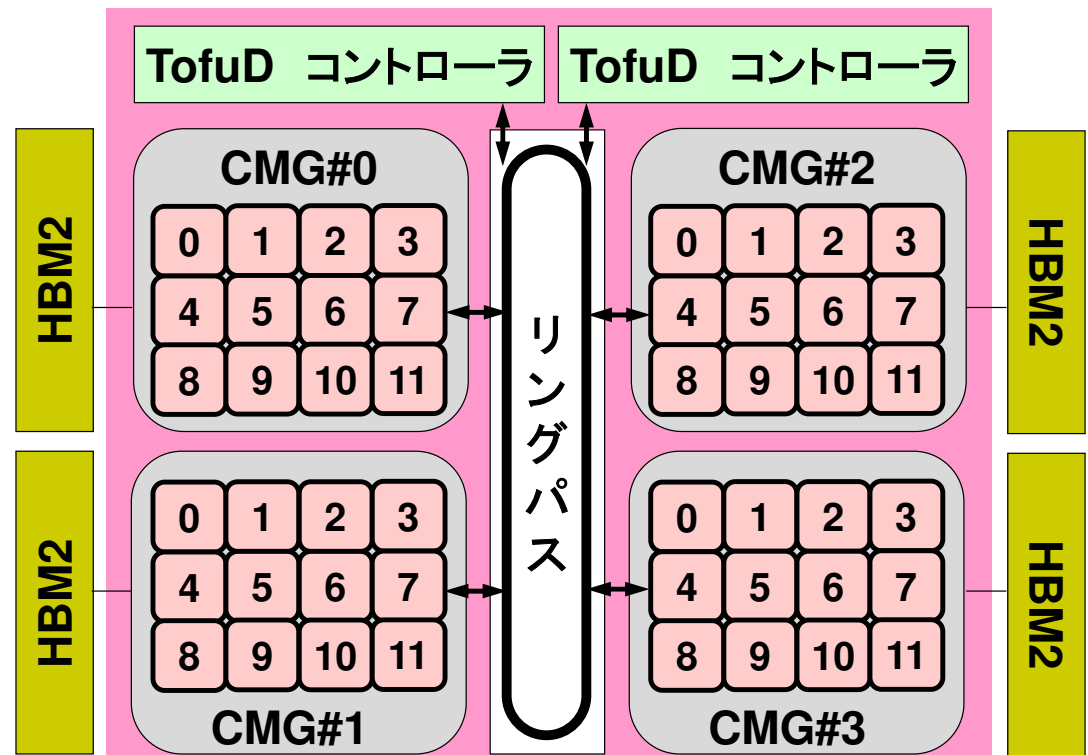
```
    do i= 1, N
```

```
...
```

```
!$omp end parallel required
```

ここでの目標

- CGソルバー (solver_CG, solver_SR) のOpenMPによるマルチスレッド化, Hybrid並列化
- 行列生成部 (mat_ass_main, mat_ass_bc) のマルチスレッド化, Hybrid並列化
- CMG当たり1MPIプロセス
 - 12スレッド/プロセス
 - 4プロセス/ノード
 - HB 12x4



OpenMP (Only Solver) (F-C)

```
>$ cd /work/gt00/t00XXX/pFEM/pfem3d/src1
```

```
>$ module load fj
```

```
>$ make
```

```
>$ cd ../run
```

```
>$ ls sol1  
sol1
```

```
>$ cd ../pmesh
```

```
<Parallel Mesh Generation>
```

```
>$ cd ../run
```

```
<modify bXX.sh>
```

```
>$ pjsub bXX.sh
```

Makefile (Fortran)

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

```
OPTFLAGS = -Kfast, openmp
```

```
FFLAGS = $(OPTFLAGS)
```

```
FLIBS =
```

```
F90LFLAGS=
```

```
#
```

```
TARGET = ../run/sol1
```

```
default: $(TARGET)
```

```
OBJS =¥
```

```
pfem_util.o ¥
```

```
solver_SR.o solver_CG.o ¥
```

```
solver11.o test1.o util.o pfem_init.o input_cntl.o input_grid.o
```

```
define_file_name.o¥
```

```
mat_con0.o mat_con1.o mat_ass_main.o mat_ass_bc.o pfem_finalize.o
```

```
output_ucd.o
```

How to apply multi-threading

- CG Solver
 - OpenMP指示文を挿入するのみ
 - ILU/IC 前処理の場合はもっと難しい
- MAT_ASS (mat_ass_main, mat_ass_bc)
 - データ依存性あり
 - 複数要素による1節点への足し込みが並列計算時に同時に発生することを避ける必要がある
 - 答えが変わる, もしくはDead Lockが生じる可能性がある
 - 色づけ : Coloring
 - 同じ色に彩色された要素は節点を共有しない
 - 同じ色の要素には並列計算が可能
 - 本問題の場合, 三次元では8色, 二次元では4色必要
 - 色づけ部分の計算はexpensive : 並列化困難

FORTRAN (solver_CG)

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

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

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


solver_SR (send)

```

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

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

```

pmesh: 8-nodes, 384-cores

Flat MPI: 384 processes

mesh.inp

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

mg.sh

```
#!/bin/sh
#PJM -N "pmg"
#PJM -L rscgrp=tutorial-o
#PJM -L node=8
#PJM --mpi proc=384
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o pmg.lst
```

```
module load fj
module load fjmpi
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=tutorial-o
#PJM -L node=8
#PJM --mpi proc=32
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o pmg.lst
```

```
module load fj
module load fjmpi
mpiexec ./pmesh
rm wk.*
```

pFEM: 8-nodes, 384-cores

Flat MPI: 384 processes

a08.sh

```
#!/bin/sh
#PJM -N "flat-08"
#PJM -L rscgrp=tutorial-o
#PJM -L node=8
#PJM --mpi proc=384
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o a08.lst

module load fj
module load fjmpi

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

pFEM: 8-nodes, 384-cores

HB 12x4: 32 processes

b08.sh

```
#!/bin/sh
#PJM -N "hb-04"
#PJM -L rscgrp=tutorial-o
#PJM -L node=8
#PJM --mpi proc=32
#PJM --omp thread=12
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o b08.lst

module load fj
module load fjmpi
export OMP_NUM_THREADS=12 (==--omp thread)

mpiexec ./sol1
mpiexec numactl -l ./sol1

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

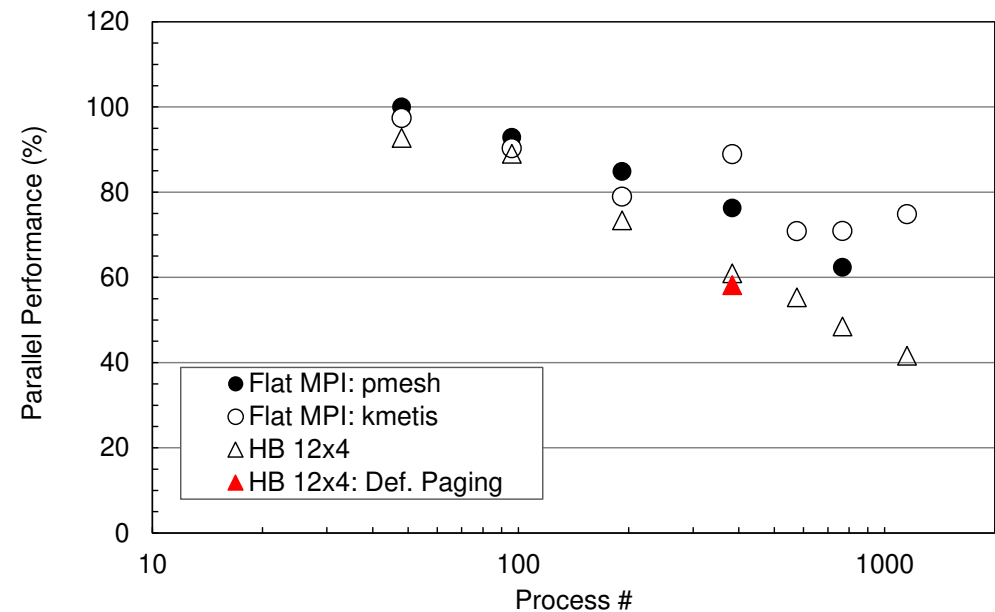
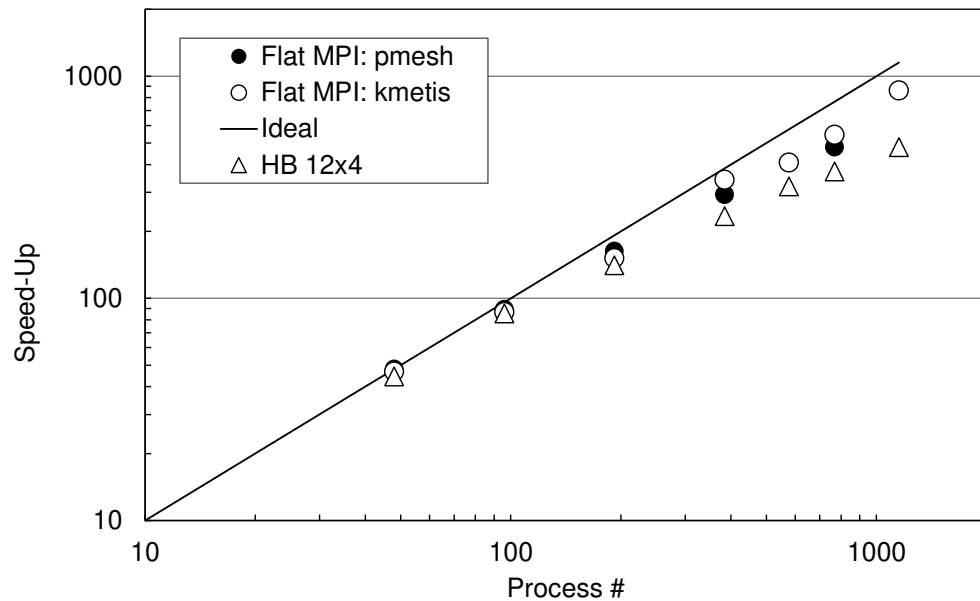
b08.sh

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

環境変数名	指定値(__はdefault)	説明
XOS_MMM_L_PAGING_POLICY	[demand <u>prepage</u>] [demand <u>prepage</u>] [demand <u>prepage</u>]	<p>各メモリ領域のページング方式(ページの割り当て契機)を選択する設定</p> <ul style="list-style-type: none"> ✓ demand: デマンドページング方式 ✓ prepage: プリページング方式 <p>本変数はコロン区切りで3つのメモリ領域のページング方式を指定:</p> <ul style="list-style-type: none"> ✓ 第1指定: 静的データの .bss 領域, 静的データの .data 領域はページング方式指定の対象外で常に prepage ✓ 第2指定: スタック領域・スレッドスタック領域 ✓ 第3 指定は動的メモリ確保領域 <p>指定値以外の値を指定した場合は, 「prepage:demand:prepage」指定とみなす。</p> <p>複数CMGをまたぐ場合はdemandを推奨!</p>

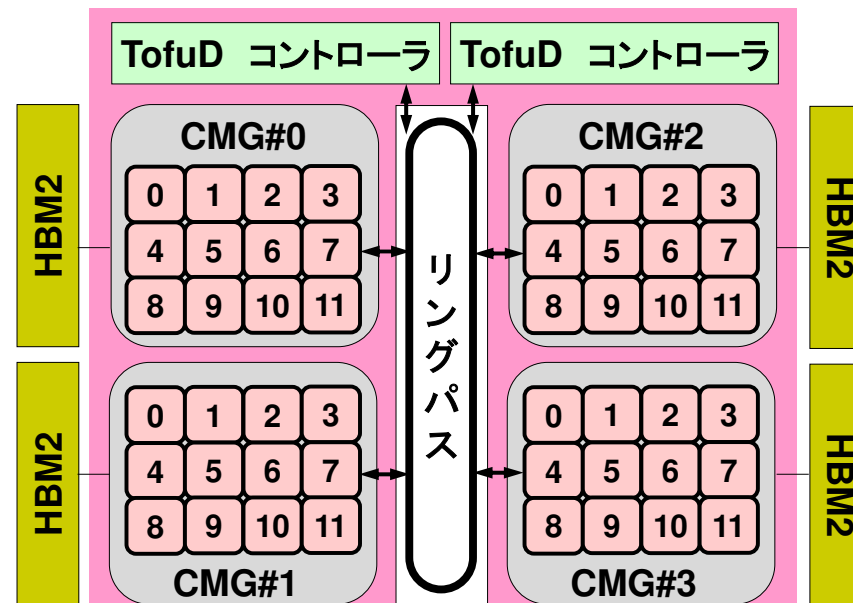
Example: Strong Scaling: C

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



Flat MPI vs. Hybrid

- アプリ・HWの特性，問題サイズ等に依存
- 一般的にノード数が少ない場合には疎行列ソルバーを含むアプリではFlat MPIの性能が良い
 - メモリ性能
- ノード数が増えるとHybridが良くなると言われるが
 - MPIプロセス数は少なくなる
- NUMAに配慮すれば，CMGをまたいだOpenMP並列化も可能



mesh.inp

Flat MPI

1-node

```
256 256 192
  4   4   3
pcube
```

8-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
 - OpenMP指示文を挿入するのみ
 - ILU/IC 前処理の場合はもっと難しい
- MAT_ASS (mat_ass_main, mat_ass_bc)
 - データ依存性あり
 - 複数要素による1節点への足し込みが並列計算時に同時に発生することを避ける必要がある
 - 答えが変わる, もしくはDead Lockが生じる可能性がある
 - 色づけ : Coloring
 - 同じ色に彩色された要素は節点を共有しない
 - 同じ色の要素には並列計算が可能
 - 本問題の場合, 三次元では8色, 二次元では4色必要
 - 色づけ部分の計算はexpensive : 並列化困難

OpenMP (Solver+Mat-Ass.) (F-C)

```
>$ cd /work/gt00/t00XXX/pFEM/pfem3d/src2
```

```
>$ module load fj
```

```
>$ make
```

```
>$ cd ../run
```

```
>$ ls sol2  
sol2
```

```
>$ cd ../pmesh
```

```
<Parallel Mesh Generation>
```

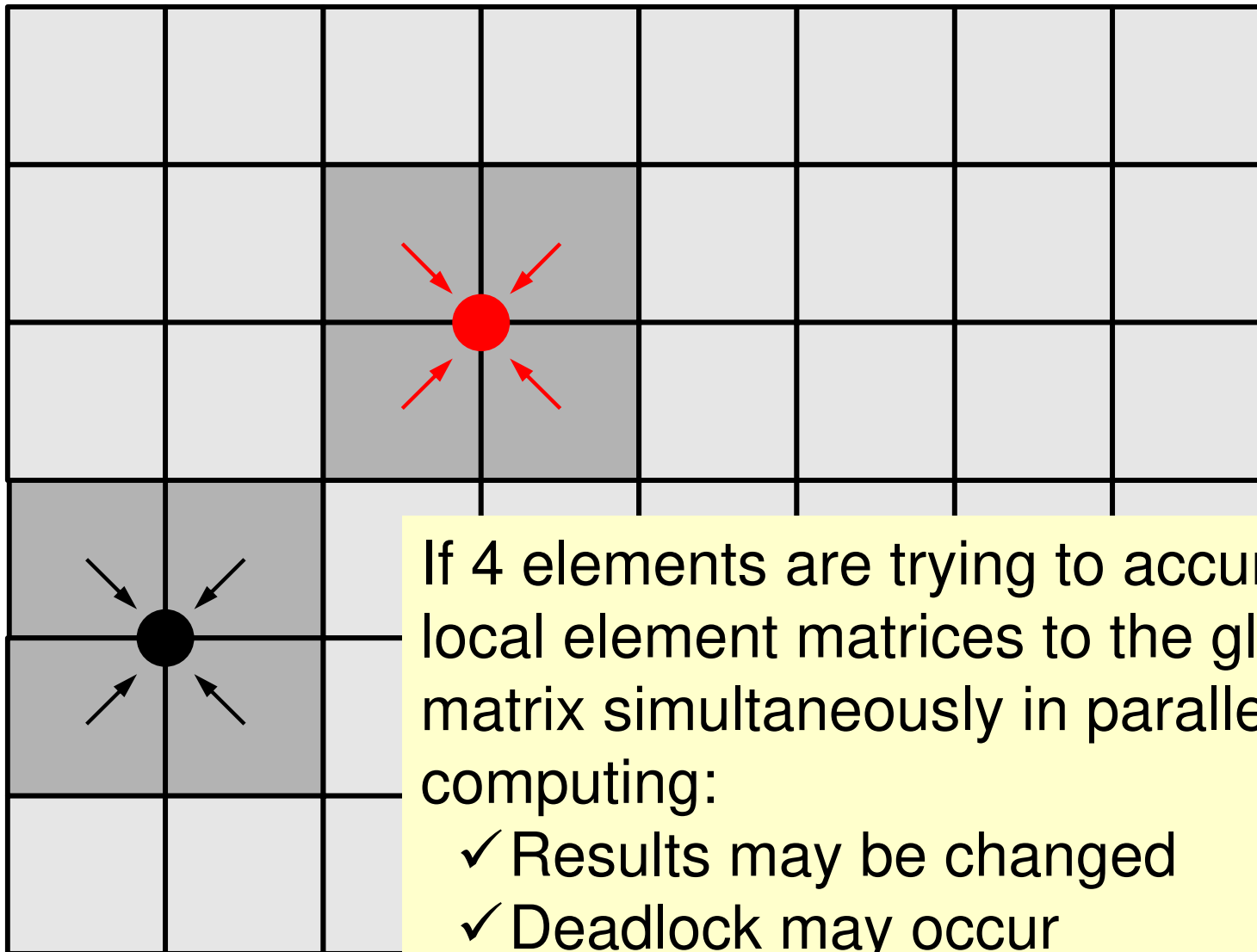
```
>$ cd ../run
```

```
<modify bXX.sh>
```

```
>$ pjsub bXX.sh
```

Mat_Ass: Data Dependency

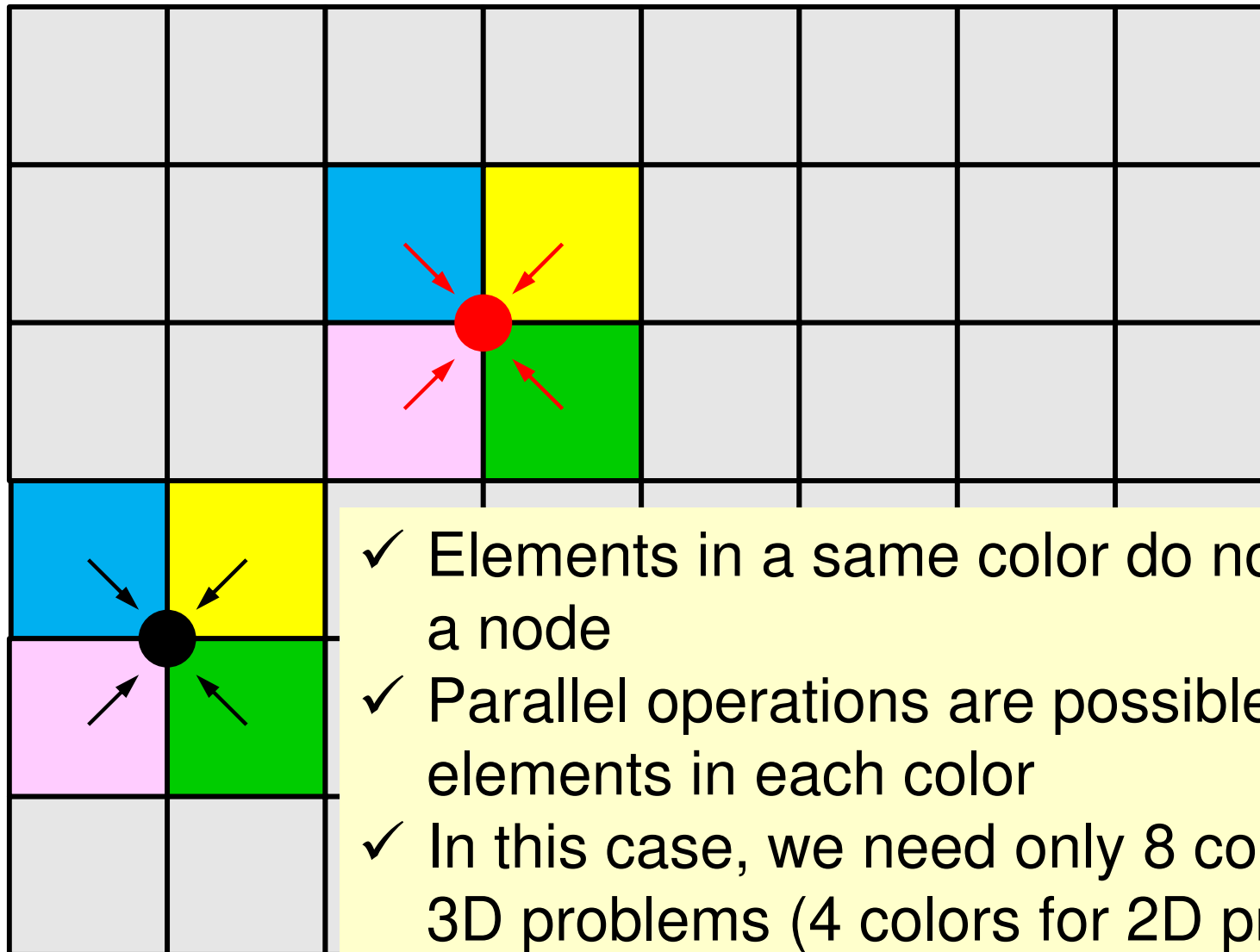
Each Node is shared by 4-Elements in 2D



How to apply multi-threading

- CG Solver
 - OpenMP指示文を挿入するのみ
 - ILU/IC 前処理の場合はもっと難しい
- MAT_ASS (mat_ass_main, mat_ass_bc)
 - 色づけ : Coloring
 - 同じ色に彩色された要素は節点を共有しない
 - 同じ色の要素には並列計算が可能
 - 本問題の場合, 三次元では8色, 二次元では4色必要

Mat_Ass: Data Dependency



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
    endif
    if (icou.eq.ICELTOT) goto 100
  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	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



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(icou)= 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
```

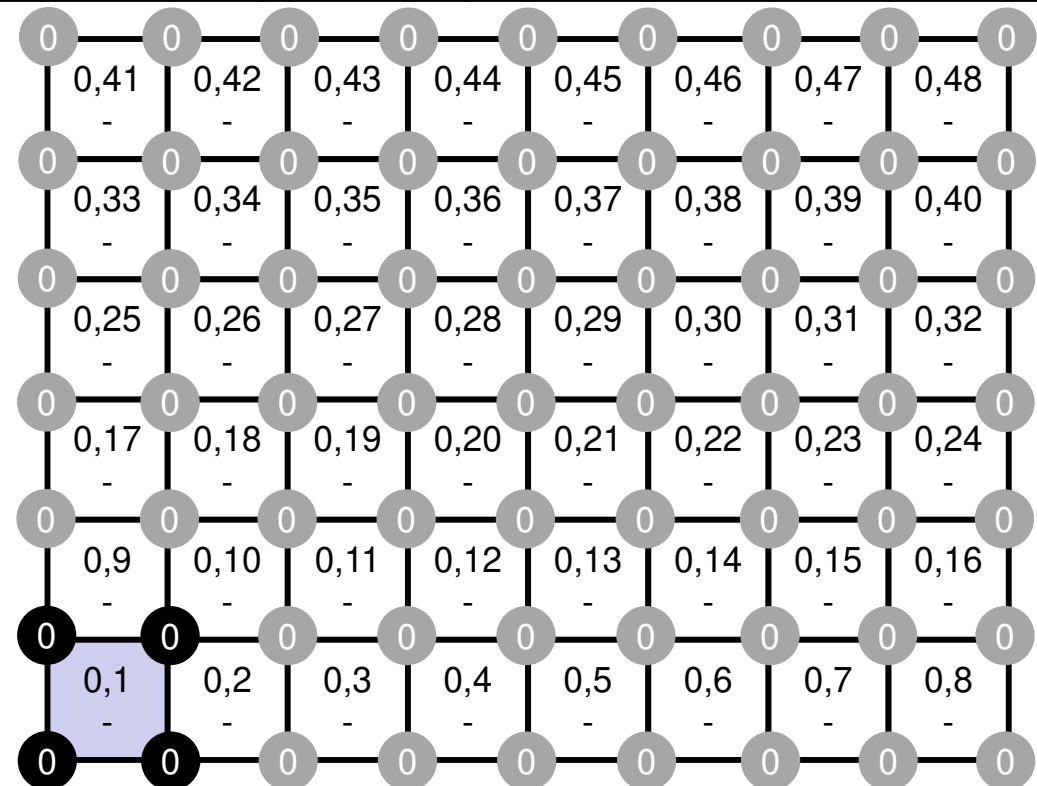
```
do icol= 0, ELMCOLORtot
```

```
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	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



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
```

icol=1
icel=1

```
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
```

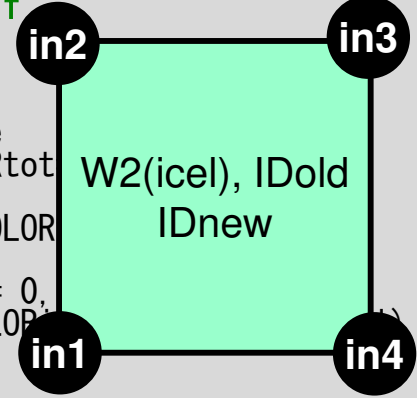
```
      ELMCOLORitem(icou)= icel
      W1(in1)= 1
      W1(in2)= 1
      W1(in3)= 1
      W1(in4)= 1
```

```
    endif
    if (icou.eq.ICELTOT) goto
```

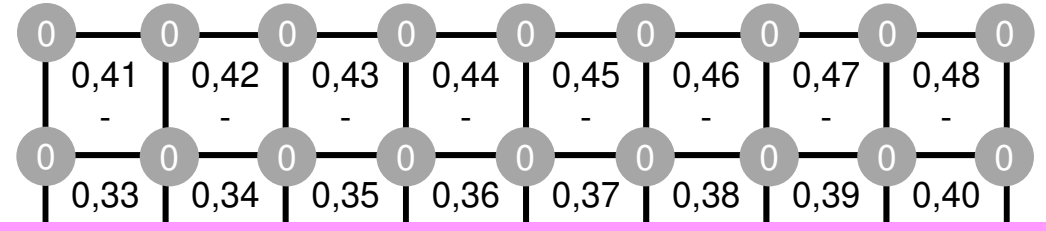
```
  enddo
enddo
```

```
100 continue
ELMCOLORtot
W3(0)
W3(ELMCOLOR
```

```
do icol= 0,
ELMCOLOR
enddo
```

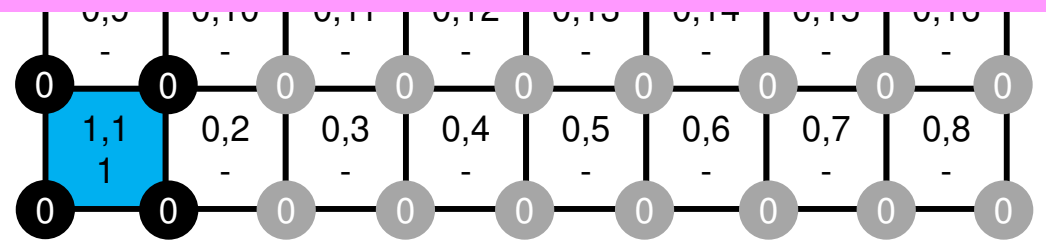


Name	Size	Content
ELMCOLORindex	0 : NP	Number of Elements in Each Color
ELMCOLORitem	ICELTOT	OLD Element ID in Each Color
W1	NP	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2	ICELTOT	Color ID of Each Element
W3	0 : NP	Accumulated # of Colored Elem's in each Color
ELMCOLORtot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



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
```

icol=1
icel=1

```
do icol= 1, NP
  do i= 1, NP
    W1(i)= 0
  enddo
```

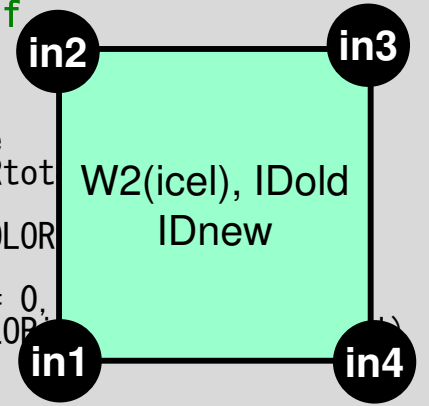
```
do icel= 1, ICELTOT
  if (W2(icel).eq.0) then
    in1= ICELNOD(icel, 1)
    in2= ICELNOD(icel, 2)
    in3= ICELNOD(icel, 3)
    in4= ICELNOD(icel, 4)
```

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

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

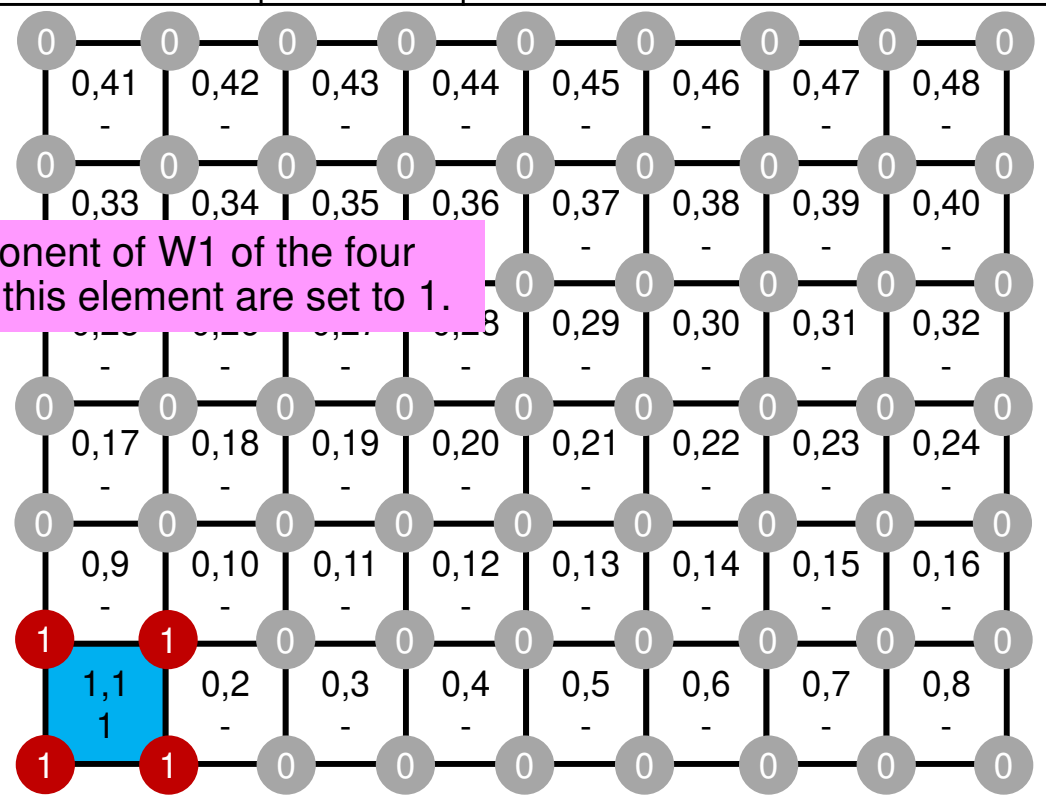
```
      ELMCOLORitem(icou)= icel
      W1(in1)= 1
      W1(in2)= 1
      W1(in3)= 1
      W1(in4)= 1
```

```
    endif
    if (icou.eq.ICELTOT) goto 100
  endif
enddo
```



```
100 continue
ELMCOLORtot
W3(0
W3(ELMCOLOR
do icol= 0,
ELMCOLOR
enddo
```

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



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
```

**icol=1
icel=2**

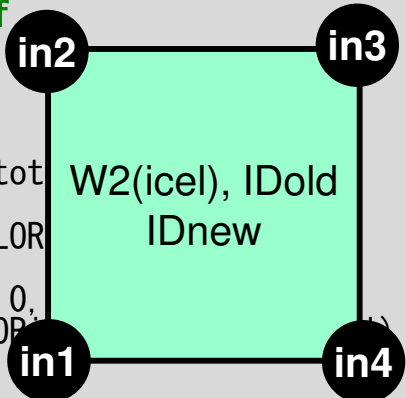
```
do icol= 1, NP
  do i= 1, NP
    W1(i)= 0
  enddo
  do icel= 1, ICELTOT
    if (W2(icel).eq.0) then
      in1= ICELNOD (icel, 1)
      in2= ICELNOD (icel, 2)
      in3= ICELNOD (icel, 3)
      in4= ICELNOD (icel, 4)

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

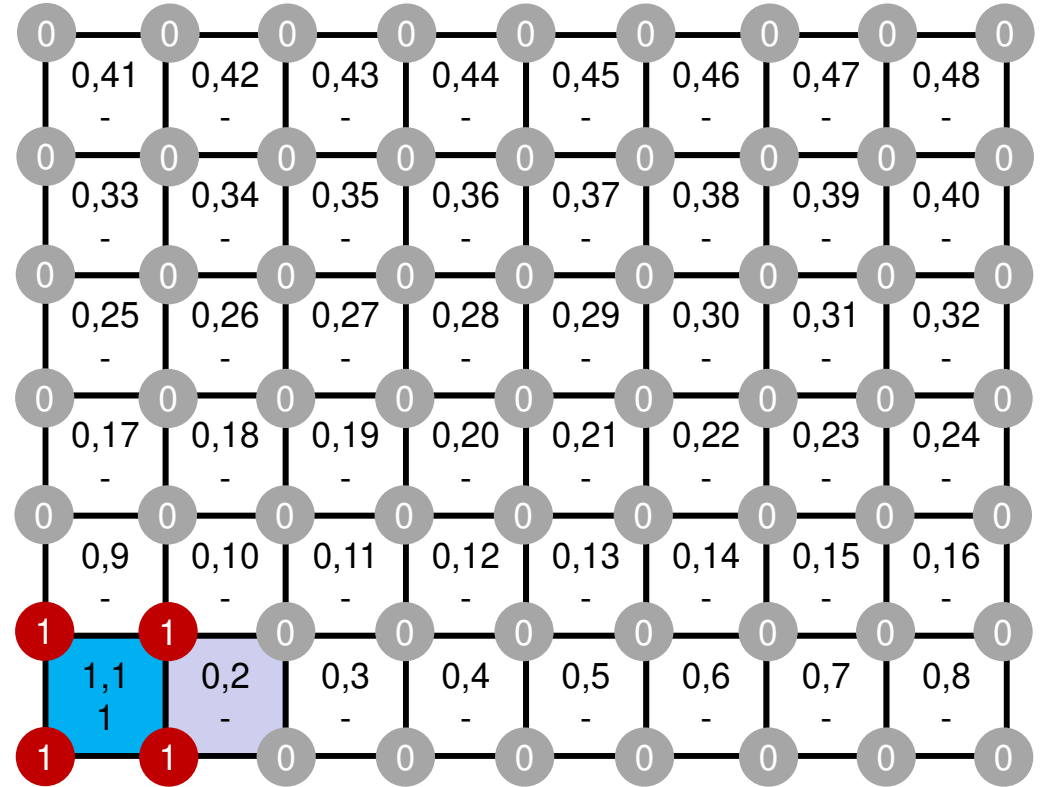
      isum= ip1 + ip2 + ip3 + ip4 (=2)
      if (isum.eq.0) then
        icou= icou + 1
        W3(icol)= icou
      endif
    endif
  enddo
enddo
```

- ✓ 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)

```
100 continue
ELMCOLORtot
W3(0)
W3 (ELMCOLOR
do icol= 0,
  ELMCOLOR
enddo
```



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



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
```

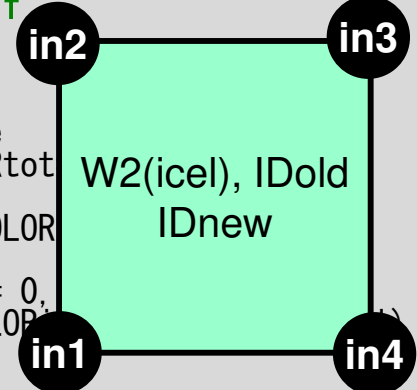
**icol=1
icel=3**

```
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)

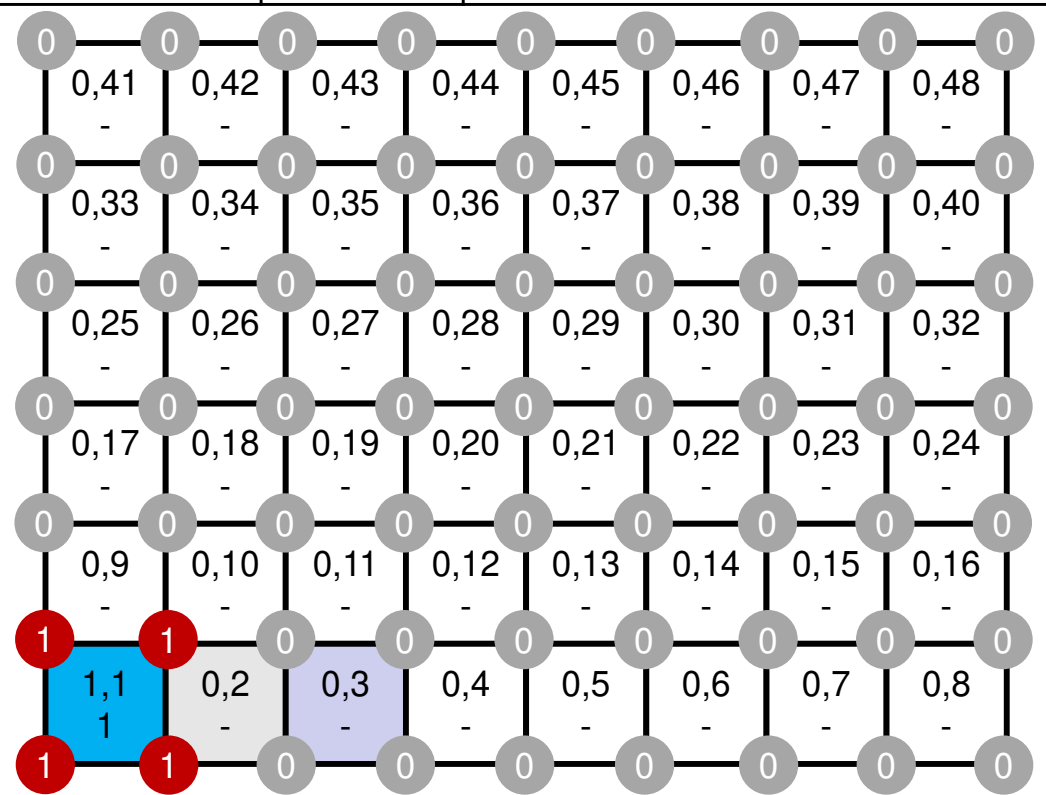
      isum= ip1 + ip2 + ip3 + ip4 (=0)
      if (isum.eq.0) then
        icou= icou + 1
        W3(icol)= icou
        W2(icel)= icol

        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
```

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



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
```

**icol=1
icel=3**

```
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
```

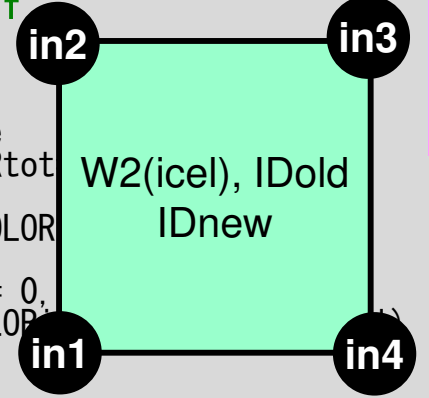
```
    ELMCOLORitem(icou)= icel
    W1(in1)= 1
    W1(in2)= 1
    W1(in3)= 1
    W1(in4)= 1
```

```
  endif
  if (icou.eq.ICELTOT) goto 100
endif
```

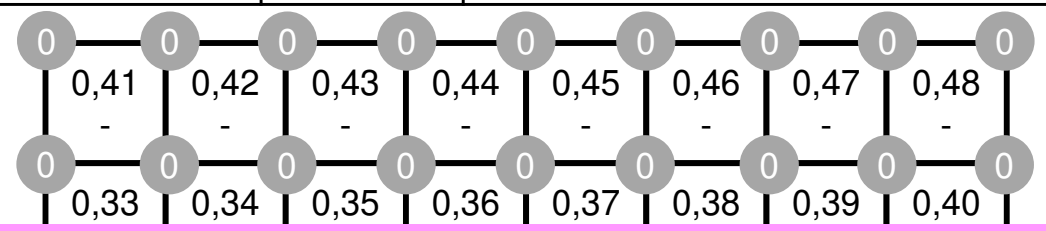
```
enddo
```

```
100 continue
ELMCOLORtot= icou
W3(0)= icou
W3(ELMCOLORtot)= icou
```

```
do icol= 0, icou
  ELMCOLORitem(icol)= 0
enddo
```

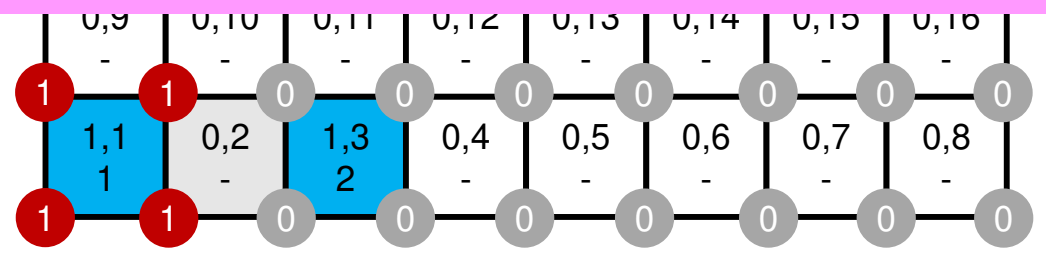


Name	Size	Content
ELMCOLORindex	0 : NP	Number of Elements in Each Color
ELMCOLORitem	ICELTOT	OLD Element ID in Each Color
W1	NP	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2	ICELTOT	Color ID of Each Element
W3	0 : NP	Accumulated # of Colored Elem's in each Color
ELMCOLORtot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



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
```

**icol=1
icel=3**

```
do icol= 1, NP
  do i= 1, NP
    W1(i)= 0
  enddo
```

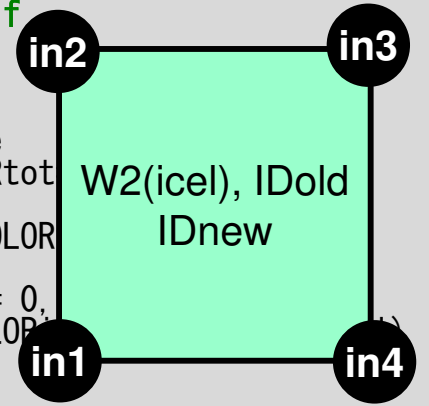
```
do icel= 1, ICELTOT
  if (W2(icel).eq.0) then
    in1= ICELNOD(icel, 1)
    in2= ICELNOD(icel, 2)
    in3= ICELNOD(icel, 3)
    in4= ICELNOD(icel, 4)
```

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

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

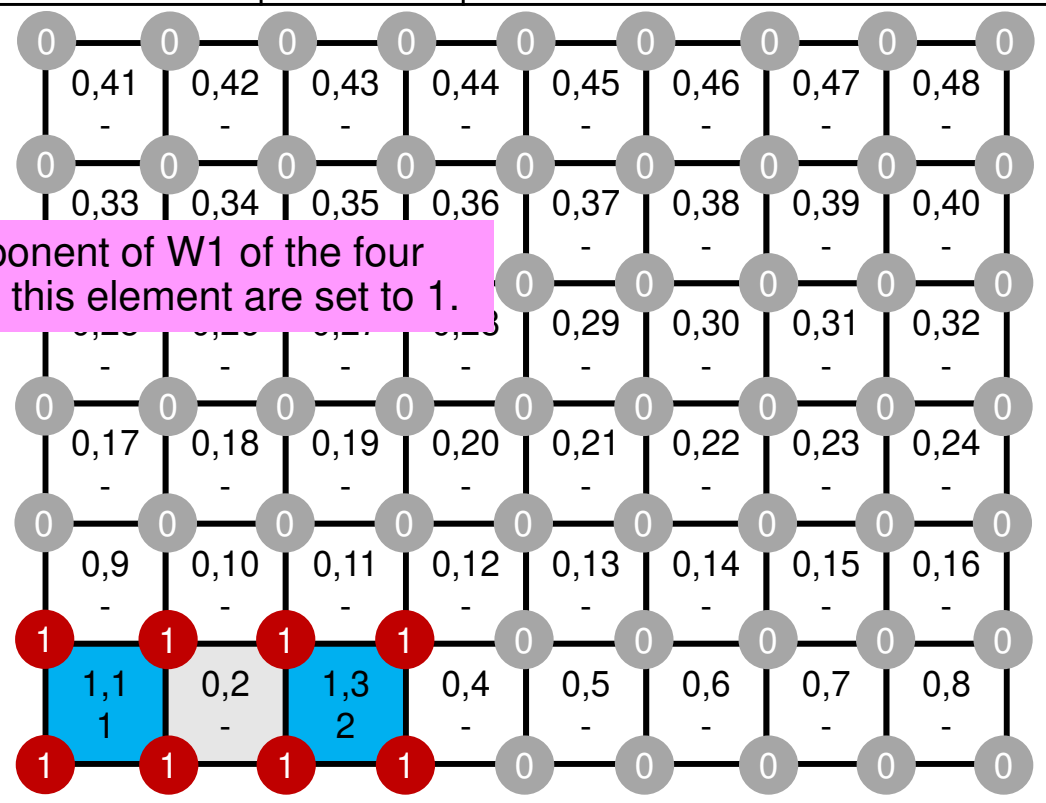
```
      ELMCOLORitem(icou)= icel
      W1(in1)= 1
      W1(in2)= 1
      W1(in3)= 1
      W1(in4)= 1
    endif
```

```
  endif
  if (icou.eq.ICELTOT) goto 100
enddo
```



```
100 continue
ELMCOLORtot
W3(0
W3(ELMCOLOR
do icol= 0,
  ELMCOLOR
enddo
```

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



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

icol=1
icel=ICELTOT (=48)

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

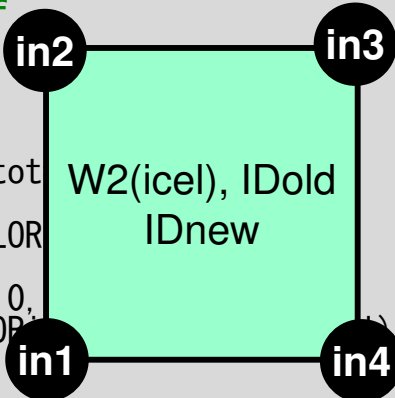
ELMCOLORitem(icou)= icel
W1(in1)= 1
W1(in2)= 1
W1(in3)= 1
W1(in4)= 1

endif
if (icou.eq. ICELTOT) goto 100
endif

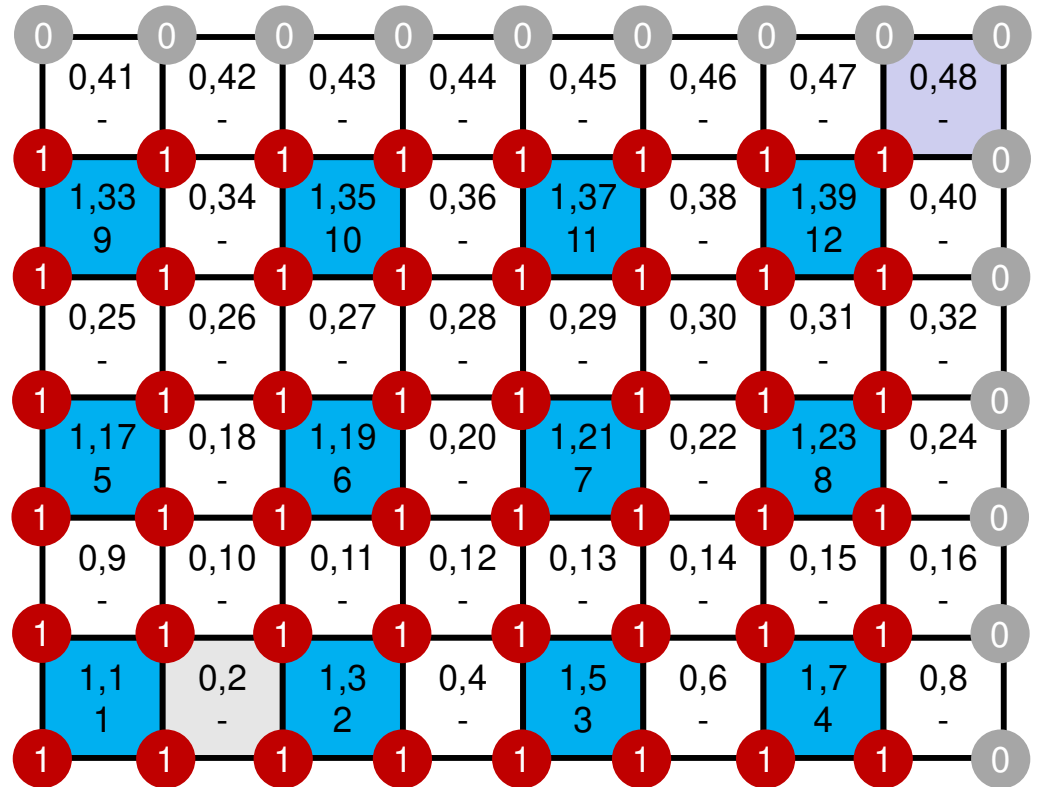
enddo
enddo

100 continue
ELMCOLORtot
W3(0
W3 (ELMCOLOR

do icol= 0,
ELMCOLOR
enddo



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



Coloring (2D) (5/7)

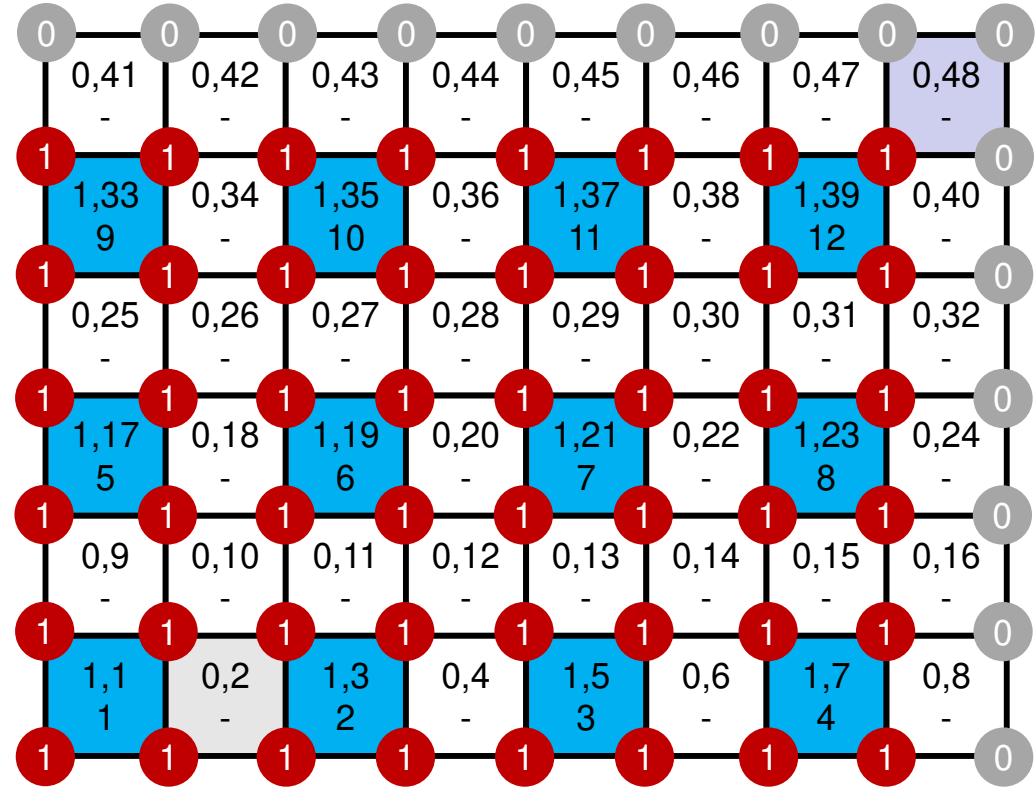
```
allocate (ELMCOLORindex(0:NP))
allocate (E
allocate (W
icol=1
icel=ICELTOT (=48)
W1=0; W2=0;
icou= 0
do icol= 1, NP
do i= 1, NP
W1(i)= 0
enddo
do icel= 1, ICELTOT
if (W2(icel).eq.0) then
in1= ICELNOD (icel, 1)
in2= ICELNOD (icel, 2)
in3= ICELNOD (icel, 3)
in4= ICELNOD (icel, 4)

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
```

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



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

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(icou)= 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
```

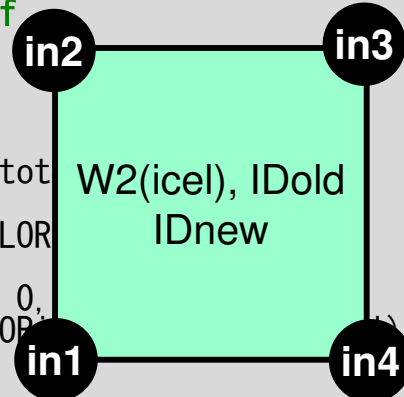
```
W3 (ELMCOLOR
```

```
do icol= 0,
```

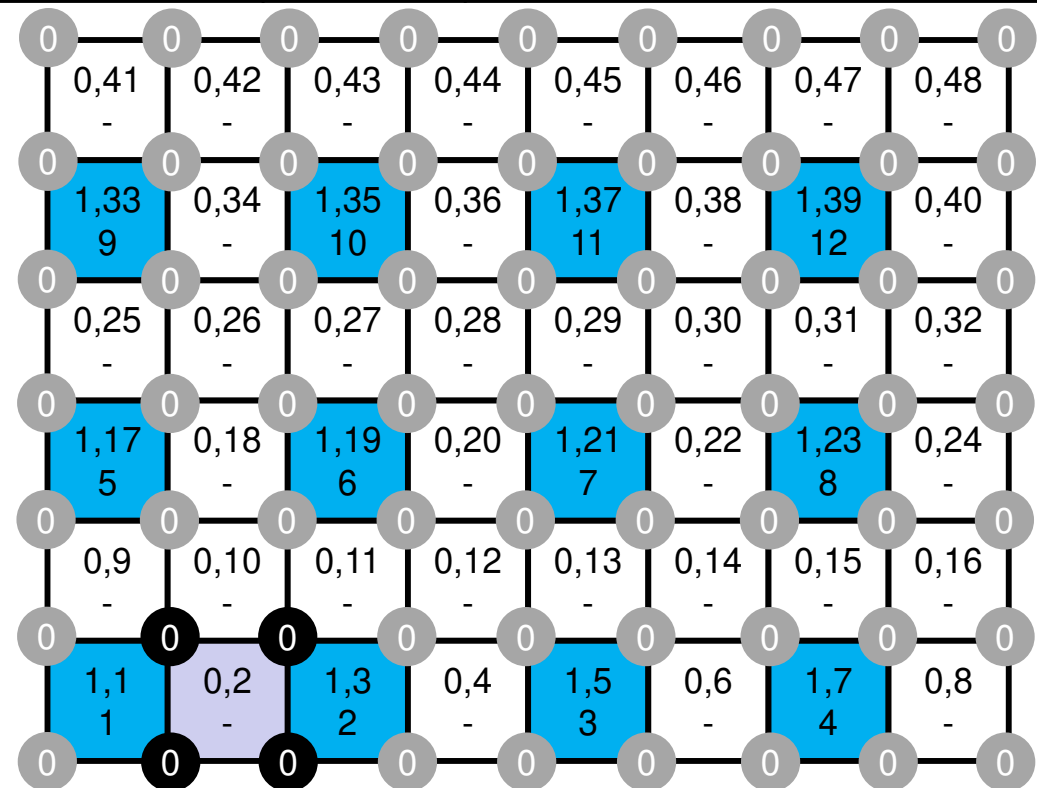
```
ELMCOLOR
```

```
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	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



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
```

icol=2
icel=2

```
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)
```

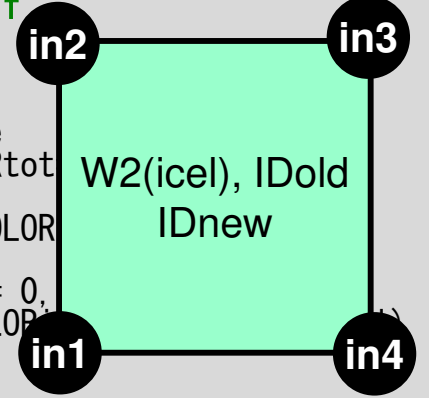
```
    ip1= W1(in1) (=0)
    ip2= W1(in2) (=0)
    ip3= W1(in3) (=0)
    ip4= W1(in4) (=0)
```

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

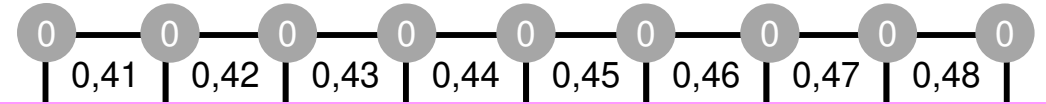
```
      ELMCOLORitem(icou)= icel
      W1(in1)= 1
      W1(in2)= 1
      W1(in3)= 1
      W1(in4)= 1
    endif
```

```
  endif
  if (icou.eq.ICELTOT) goto 1
enddo
```

```
100 continue
ELMCOLORtot= icou
W3(0)= icou
W3(ELMCOLORtot)= icou
do icol= 0, icou
  ELMCOLORitem(icol)= icel
enddo
```

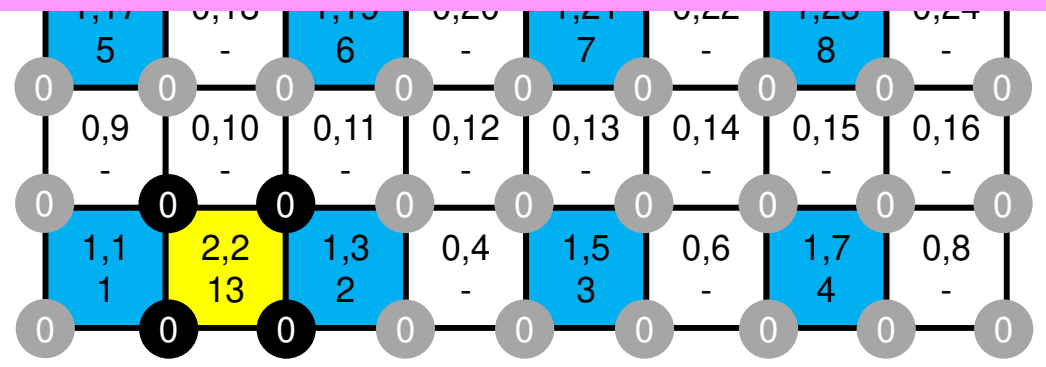


Name	Size	Content
ELMCOLORindex	0 : NP	Number of Elements in Each Color
ELMCOLORitem	ICELTOT	OLD Element ID in Each Color
W1	NP	Flag of Each Node =0: Not flagged in the current color =1: Already Flagged
W2	ICELTOT	Color ID of Each Element
W3	0 : NP	Accumulated # of Colored Elem's in each Color
ELMCOLORtot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



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
```

icol=2
icel=2

```
do icol= 1, NP
  do i= 1, NP
    W1(i)= 0
  enddo
```

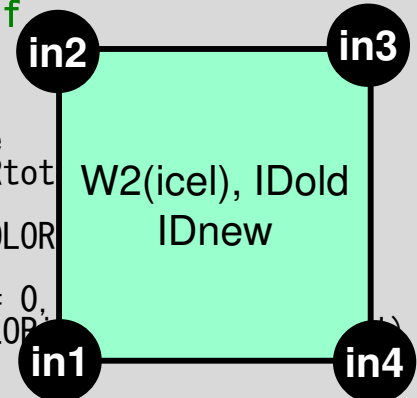
```
do icel= 1, ICELTOT
  if (W2(icel).eq.0) then
    in1= ICELNOD (icel, 1)
    in2= ICELNOD (icel, 2)
    in3= ICELNOD (icel, 3)
    in4= ICELNOD (icel, 4)
```

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

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

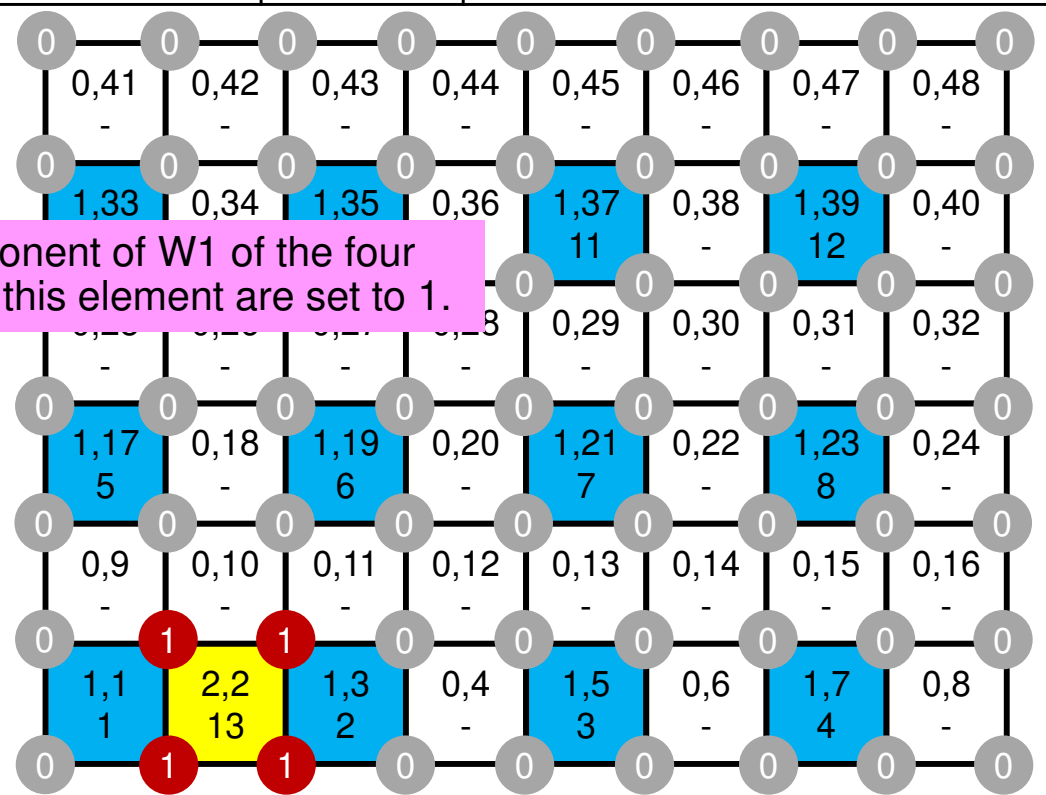
```
      ELMCOLORitem(icou)= icel
      W1 (in1)= 1
      W1 (in2)= 1
      W1 (in3)= 1
      W1 (in4)= 1
    endif
```

```
  endif
  if (icou.eq. ICELTOT) goto 100
enddo
```



```
100 continue
ELMCOLORtot
W3(0
W3 (ELMCOLOR
do icol= 0,
  ELMCOLOR
enddo
```

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



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

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

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)

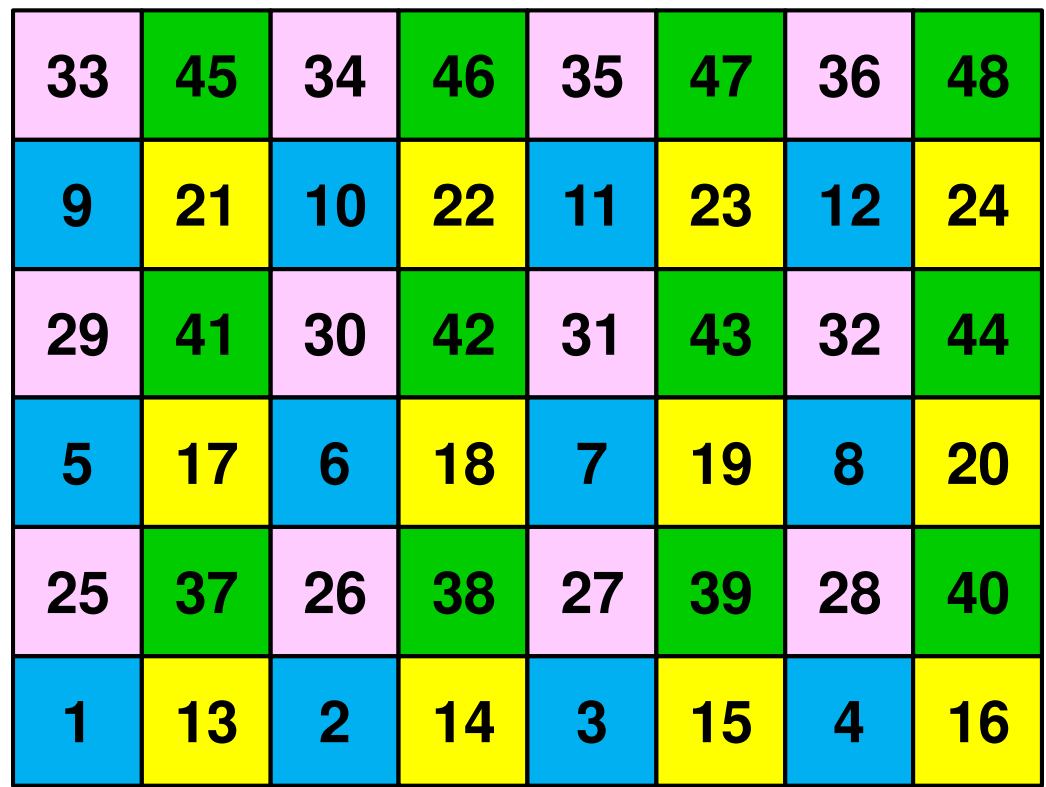
      isum= ip1 + ip2 + ip3 + ip4
      if (isum.eq.0) then
        icou= icou + 1
        W3(icol)= icou
        W2(icel)= icol

        ELMCOLORitem(icou)= icel
        W1 (in1)= 1
        W1 (in2)= 1
        W1 (in3)= 1
        W1 (in4)= 1
      endif
    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	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



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

```

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

...

How to apply multi-threading

- CG Solver
 - OpenMP指示文を挿入するのみ
 - ILU/IC 前処理の場合はもっと難しい
- MAT_ASS (mat_ass_main, mat_ass_bc)
 - データ依存性あり
 - 複数要素による1節点への足し込みが並列計算時に同時に発生することを避ける必要がある
 - 答えが変わる, もしくはDead Lockが生じる可能性がある
 - 色づけ : Coloring
 - 同じ色に彩色された要素は節点を共有しない
 - 同じ色の要素には並列計算が可能
 - 本問題の場合, 三次元では8色, 二次元では4色必要
 - 色づけ部分の計算はexpensive : 並列化困難

x12.sh

```
#!/bin/sh
#PJM -N "hb-12"
#PJM -L rscgrp=tutorial-o
#PJM -L node=12
#PJM --mpi proc=48
#PJM --omp thread=12
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o x12.lst

module load fj
module load fjmpi

export OMP_NUM_THREADS=12
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

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

