

Introduction to Parallel Programming for Multicore/Manycore Clusters

Part VI: Report

Kengo Nakajima
Information Technology Center

SMPindex:

for preconditioning

```

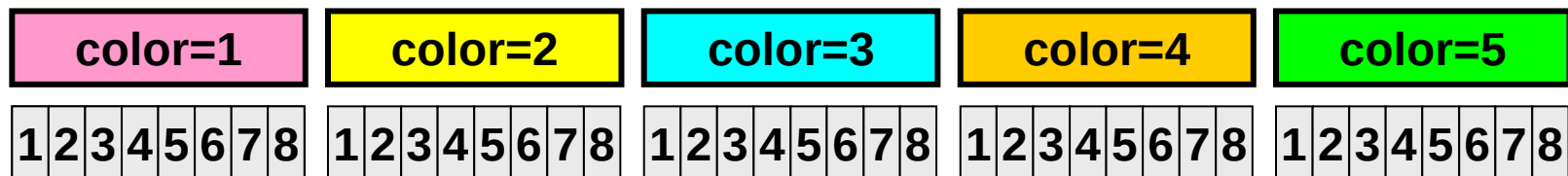
do ic= 1, NCOLORTot
!$omp parallel do ...
  do ip= 1, PEsmptOT
    ip1= (ic-1)*PEsmptOT+ip
    do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      (...)
    enddo
  enddo
!omp end parallel do
enddo

```

Initial Vector



Coloring
(5 colors)
+Ordering



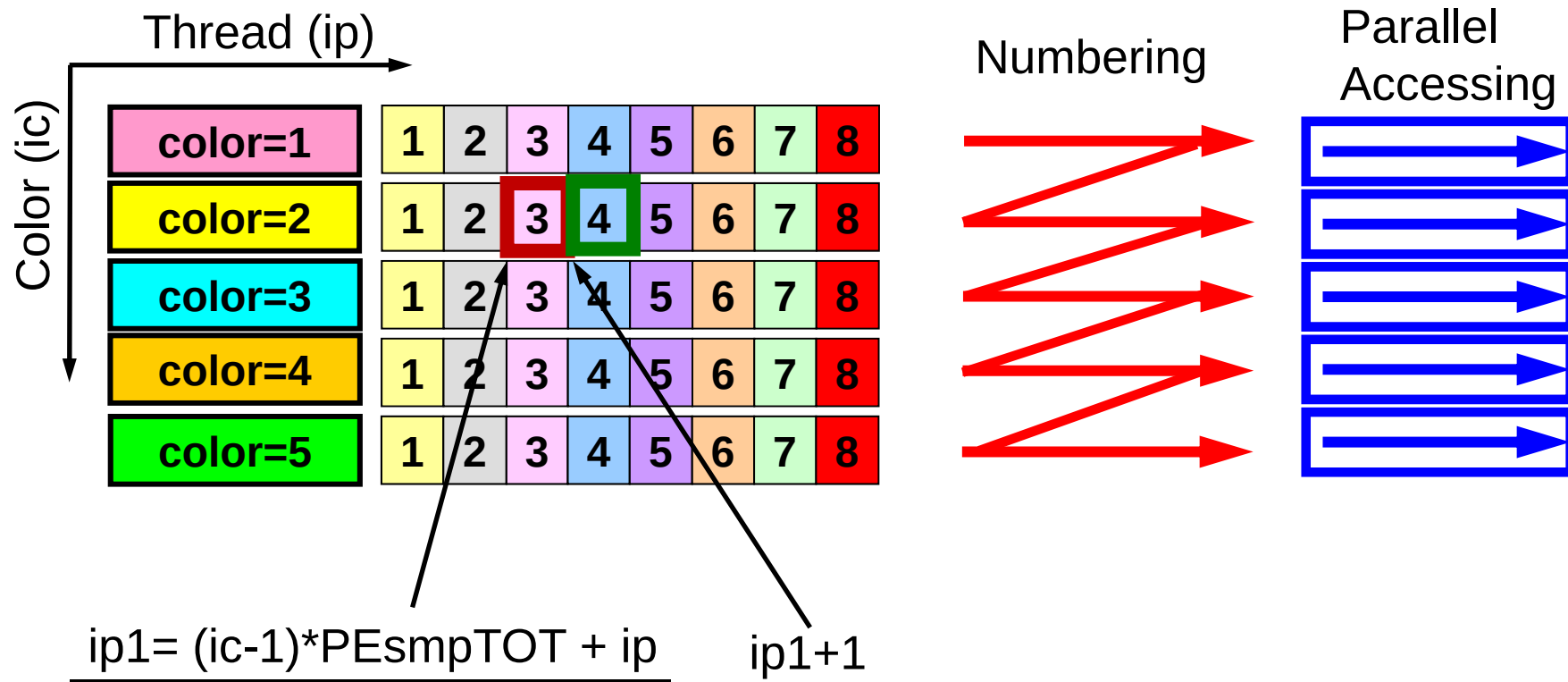
- 5-colors, 8-threads
- Meshes in same color are independent: parallel processing
- Reordering in ascending order according to color ID

SMPindex

```

!$omp parallel private(ip, ip1, i, WVAL, k)
  do ic= 1, NCOLORTot
!$omp do
    do ip= 1, PEsmptOT
      ip1= (ic-1)*PEsmptOT + ip
      do i= SMPindex(ip1-1)+1, SMPindex(ip1) ...

```



SMPindex

$\text{COLORindex}(ic) = 100$
 $\text{COLORindex}(ic+1) = 200$
 $\text{PEsmpTOT} = 8$

$nn1 = 200 - 100 = 100$
 $num = 100 / 8 = 12 \text{ (12.5)}$
 $nr = 100 - 12 * 8 = 4$

$\text{SMPindex}(ip1) = 100$
 $\text{SMPindex}(ip1+1) = 113 \text{ (13 elements in the 1st thread)}$
 $\text{SMPindex}(ip1+2) = 126 \text{ (13 elements in the 2nd thread)}$
 $\text{SMPindex}(ip1+3) = 139 \text{ (13 elements in the 3rd thread)}$
 $\text{SMPindex}(ip1+4) = 152 \text{ (13 elements in the 4th thread)}$
 $\text{SMPindex}(ip1+5) = 164 \text{ (12 elements in the 5th thread)}$
 $\text{SMPindex}(ip1+6) = 176 \text{ (12 elements in the 6th thread)}$
 $\text{SMPindex}(ip1+7) = 188 \text{ (12 elements in the 7th thread)}$
 $\text{SMPindex}(ip1+8) = 200 \text{ (12 elements in the 8th thread)}$

Problems in Reordering

- Coloring
 - MC
 - RCM
 - CM-RCM
- Renumbering is according to color/level ID
- On each thread, numbering is not continuous
 - reduced performance

SMPindex:

for preconditioning

```

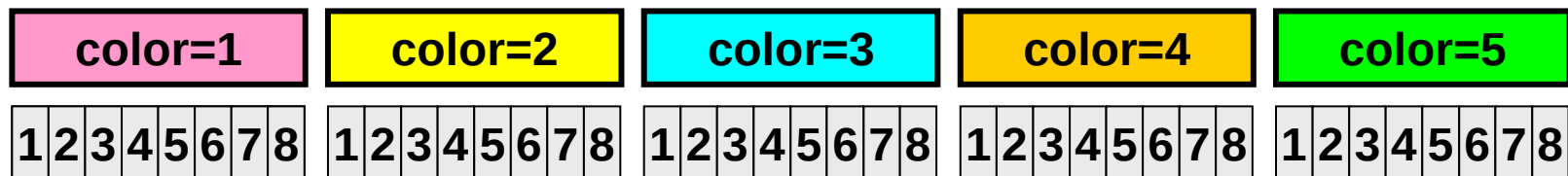
do ic= 1, NCOLORTot
!$omp parallel do ...
  do ip= 1, PEsmptOT
    ip1= (ic-1)*PEsmptOT+ip
    do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      (...)
    enddo
  enddo
!omp end parallel do
enddo

```

Initial Vector



Coloring
(5 colors)
+Ordering



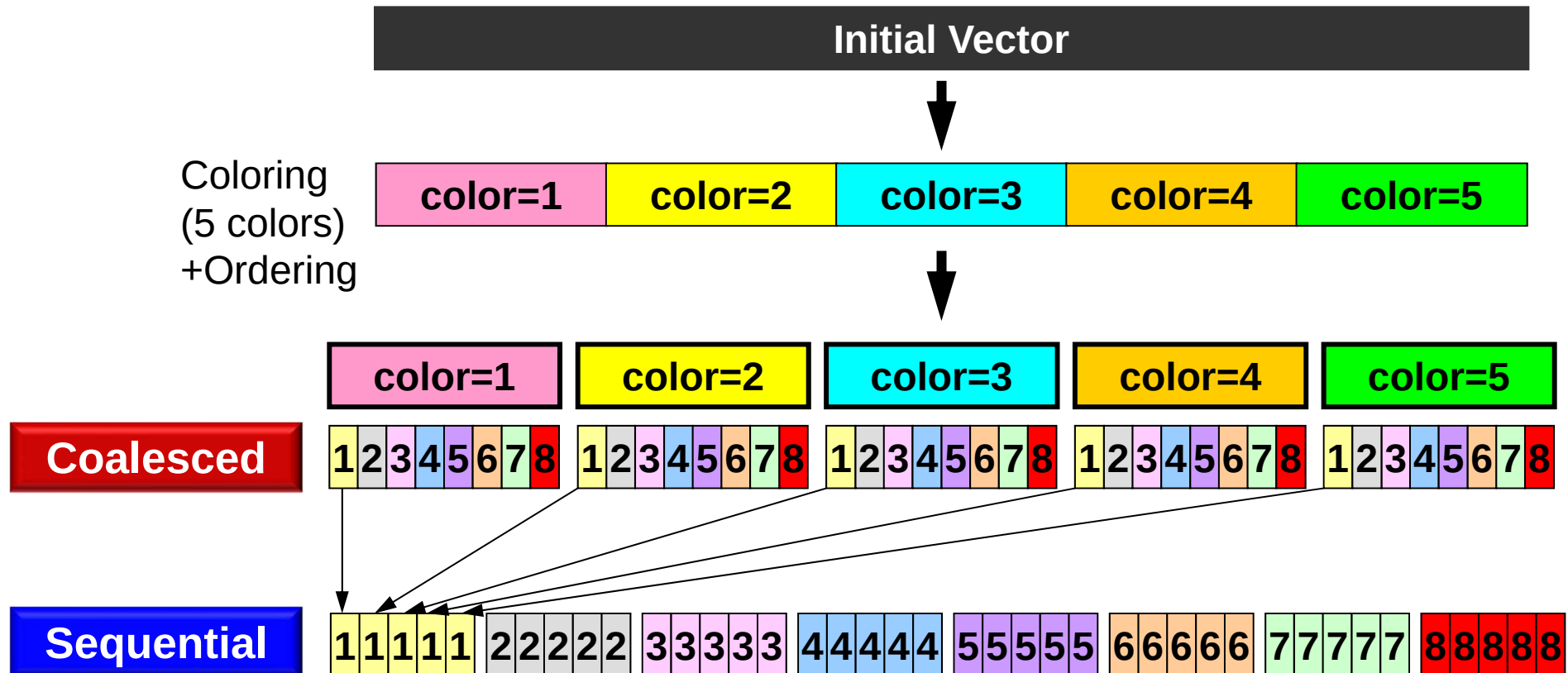
- 5-colors, 8-threads
- Meshes in same color are independent: parallel processing
- Reordering in ascending order according to color ID

Sequential Reordering

- Reordering for continuous memory access on each thread (core)
 - Performance is expected to be better.
 - Continuous address of arrays, such as coefficient matrices
 - Locality (2-page later)

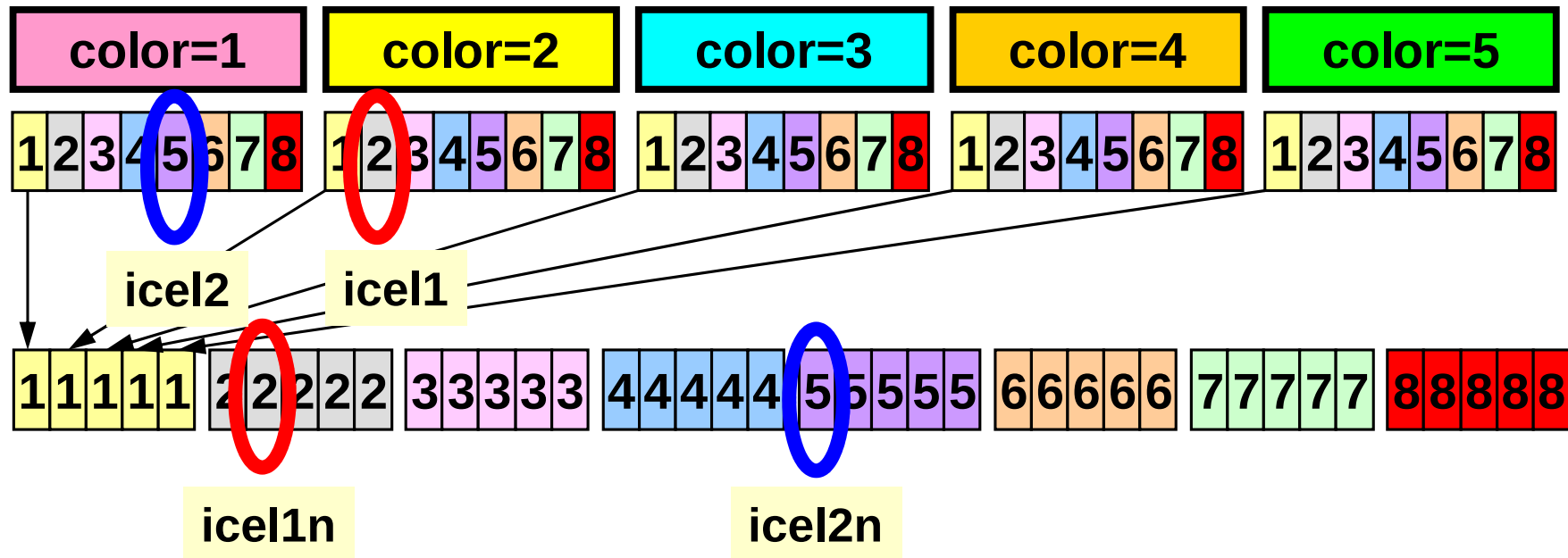
Sequential Reordering

Further reordering for continuous memory access on each thread, 5-color, 8-threads



Inconsistent numbering may occur

- Coalesced
 - $icel1 > icel2$, therefore, $icel2 = itemL(k)$, where $indexL(icel1) + 1 \leq k \leq indexL[icel1 + 1]$
- Sequential
 - $icel1n < icel2n$, but still $icel2n = itemL(k)$, where $indexL(icel1n) + 1 \leq k \leq indexL[icel1n + 2]$



Sequential Reordering

CM-RCM(2), 4-threads

Continuous Data Access on a Thread: Utilization of
Cache, Prefetching

45	10	39	5	35	2	33	1
17	46	11	40	6	36	3	34
53	18	47	12	41	7	37	4
24	54	19	48	13	42	8	38
59	25	55	20	49	14	43	9
29	60	26	56	21	50	15	44
63	30	61	27	57	22	51	16
32	64	31	62	28	58	23	52

CM-RCM(2)



29	18	15	5	11	2	9	1
33	30	19	16	6	12	3	10
45	34	31	20	25	7	13	4
40	46	35	32	21	26	8	14
59	49	47	36	41	22	27	17
53	60	50	48	37	42	23	28
63	54	61	51	57	38	43	24
56	64	55	62	52	58	39	44

Sequential Reordering, 4-threads

Sequential Reordering

CM-RCM(2), 4-threads

1st-Color

■ #0 thread, ■ #1, ■ #2, ■ #3

45	10	39	5	35	2	33	1
17	46	11	40	6	36	3	34
53	18	47	12	41	7	37	4
24	54	19	48	13	42	8	38
59	25	55	20	49	14	43	9
29	60	26	56	21	50	15	44
63	30	61	27	57	22	51	16
32	64	31	62	28	58	23	52

CM-RCM(2)



29	18	15	5	11	2	9	1
33	30	19	16	6	12	3	10
45	34	31	20	25	7	13	4
40	46	35	32	21	26	8	14
59	49	47	36	41	22	27	17
53	60	50	48	37	42	23	28
63	54	61	51	57	38	43	24
56	64	55	62	52	58	39	44

Sequential Reordering, 4-threads

Sequential Reordering

CM-RCM(2), 4-threads

2nd-Color

■ #0 thread, ■ #1, ■ #2, ■ #3

45	10	39	5	35	2	33	1
17	46	11	40	6	36	3	34
53	18	47	12	41	7	37	4
24	54	19	48	13	42	8	38
59	25	55	20	49	14	43	9
29	60	26	56	21	50	15	44
63	30	61	27	57	22	51	16
32	64	31	62	28	58	23	52

CM-RCM(2)

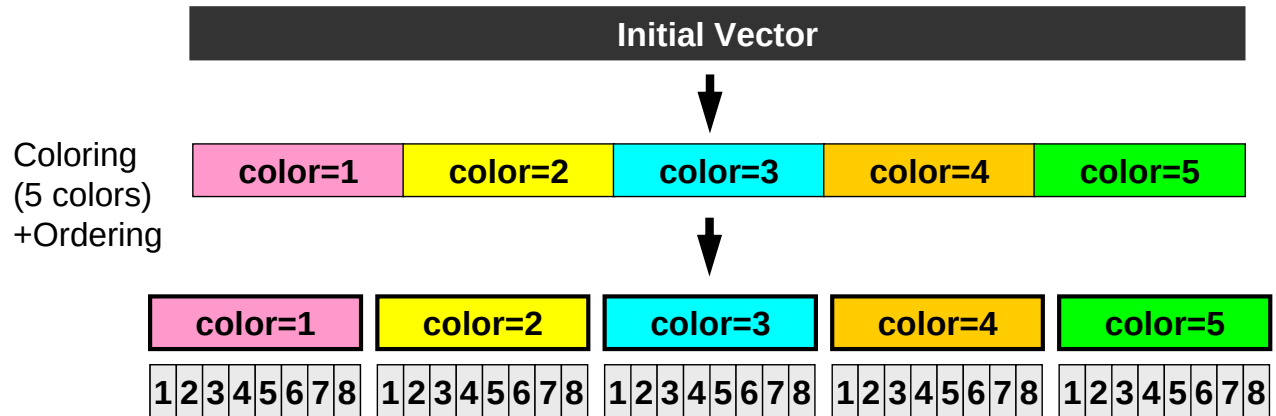


29	18	15	5	11	2	9	1
33	30	19	16	6	12	3	10
45	34	31	20	25	7	13	4
40	46	35	32	21	26	8	14
59	49	47	36	41	22	27	17
53	60	50	48	37	42	23	28
63	54	61	51	57	38	43	24
56	64	55	62	52	58	39	44

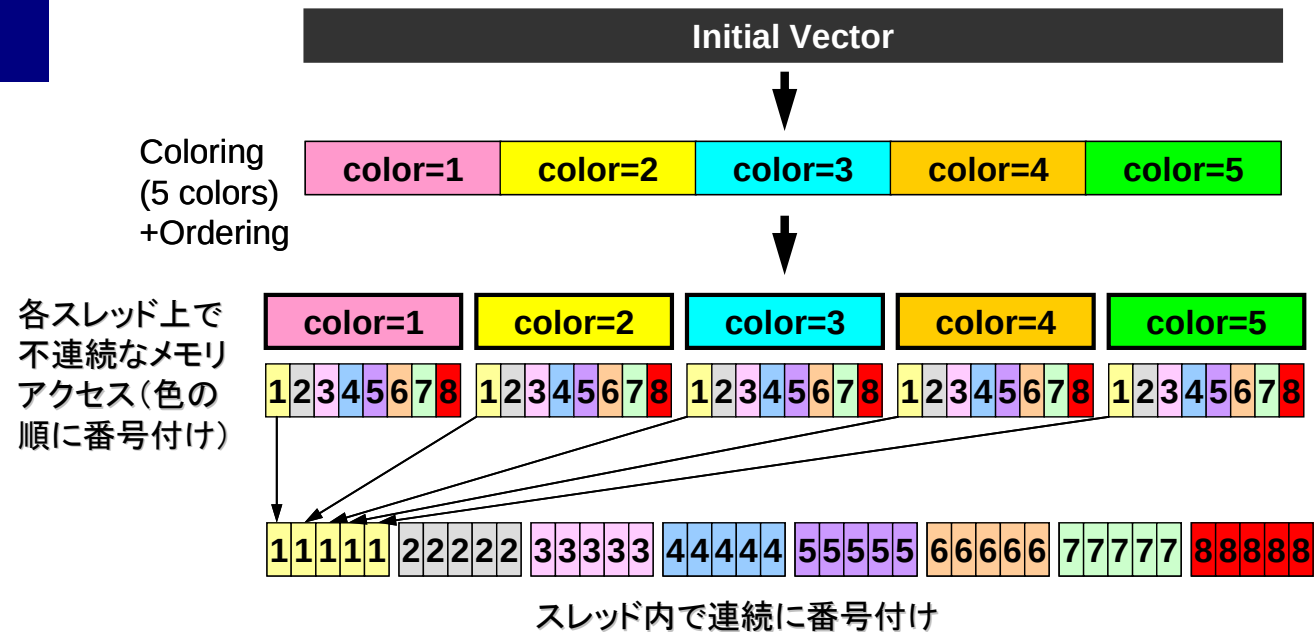
Sequential Reordering, 4-threads

Sequential Reordering

Coalesced
Good for GPU



Sequential



Additional Code

- Location
 - `<$O-L3>: /lustre/gt12/tXXYYY/multicoe/L3`
 - `<$O-L3>/reorder0, <$O-L3>/run`
- Compile & Run
 - Source Code
 - `cd <$O-L3>/reorder0`
 - `make`
 - `<$O-L3>/run/L3-rsol0` execution file
 - Control Data
 - `<$O-L3>/run/INPUT.DAT`
 - Shell Script
 - `<$O-L3>/run/gor.sh`

gor.sh

```
#PBS -q u-lecture2
#PBS -N test
#PBS -l select=1:ncpus=16          ncpus=PEsmpTOT
#PBS -Wgroup_list=gt12
#PBS -l walltime=00:10:00
#PBS -e test2.err
#PBS -o test2.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

export OMP_NUM_THREADS=16          =PEsmpTOT
export KMP_AFFINITY=granularity=fine,compact
numactl ./L3-rsol0
numactl ./L3-rsol0
numactl ./L3-rsol0
```

INPUT.DAT

```

100 100 100          NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00  DX/DY/DZ
1.0e-08             EPSICC
16                  PEsmpTOT
-100                 NCOLORtot
0                    NFLAG
0                    METHOD

```

- **PE_{smpTOT}**
 - Thread Number (= OMP_NUM_THREADS)
- **NCOLOR_{tot}**
 - Reordering Method + Initial Number of Colors/Levels
 - ≥ 2 : MC, =0: CM, =-1: RCM, $-2 \leq$: CMRCM
- **NFLAG**
 - =0: without first-touch, =1: with first-touch
- **METHOD**
 - Loop structure for Mat-Vec
 - =0: conventional way, =1: similar to forward/backward substitution

Sequential Reordering (1/4) Main

```

program MAIN

use STRUCT
use PCG
use solver_ICCG_mc
use solver_ICCG_mc_ft

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

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN

ISET= 0
allocate (WK(ICELTOT))

    if (METHOD.eq.0) then
        call solve_ICCG_mc
&      ( ICELTOT, NPL, NPU, indexL, itemL, indexU, itemU,
&      D, BFORCE, PHI, AL, AU, NCOLORTot, PEsmptTOT,
&      SMPindex_new, EPSICCG, ITR, IER)
    else
        call solve_ICCG_mc_ft
&      ( ICELTOT, NPL, NPU, indexL, itemL, indexU, itemU,
&      D, BFORCE, PHI, AL, AU, NCOLORTot, PEsmptTOT,
&      SMPindex_new, EPSICCG, ITR, IER)
    endif

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

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

call OUTUCD

stop
end

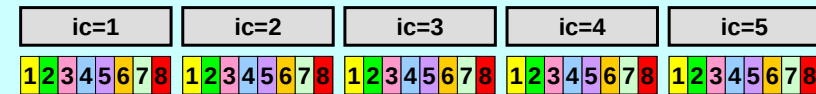
```

```

allocate (SMPindex(0:PEsmptTOT*NCOLORtot))
SMPindex= 0
do ic= 1, NCOLORtot
  nn1= COLORindex(ic) - COLORindex(ic-1)
  num= nn1 / PEsmptTOT
  nr = nn1 - PEsmptTOT*num
  do ip= 1, PEsmptTOT
    if (ip.le.nr) then
      SMPindex((ic-1)*PEsmptTOT+ip)= num + 1
    else
      SMPindex((ic-1)*PEsmptTOT+ip)= num
    endif
  enddo
enddo

```

SMPindex



SMPindex_new



```

allocate (SMPindex_new(0:PEsmptTOT*NCOLORtot))
SMPindex_new(0)= 0
do ic= 1, NCOLORtot
  do ip= 1, PEsmptTOT
    j1= (ic-1)*PEsmptTOT + ip
    j0= j1 - 1
    SMPindex_new((ip-1)*NCOLORtot+ic)= SMPindex(j1)
    SMPindex(j1)= SMPindex(j0) + SMPindex(j1)
  enddo
enddo

do ip= 1, PEsmptTOT
  do ic= 1, NCOLORtot
    j1= (ip-1)*NCOLORtot + ic
    j0= j1 - 1
    SMPindex_new(j1)= SMPindex_new(j0) + SMPindex_new(j1)
  enddo
enddo

```

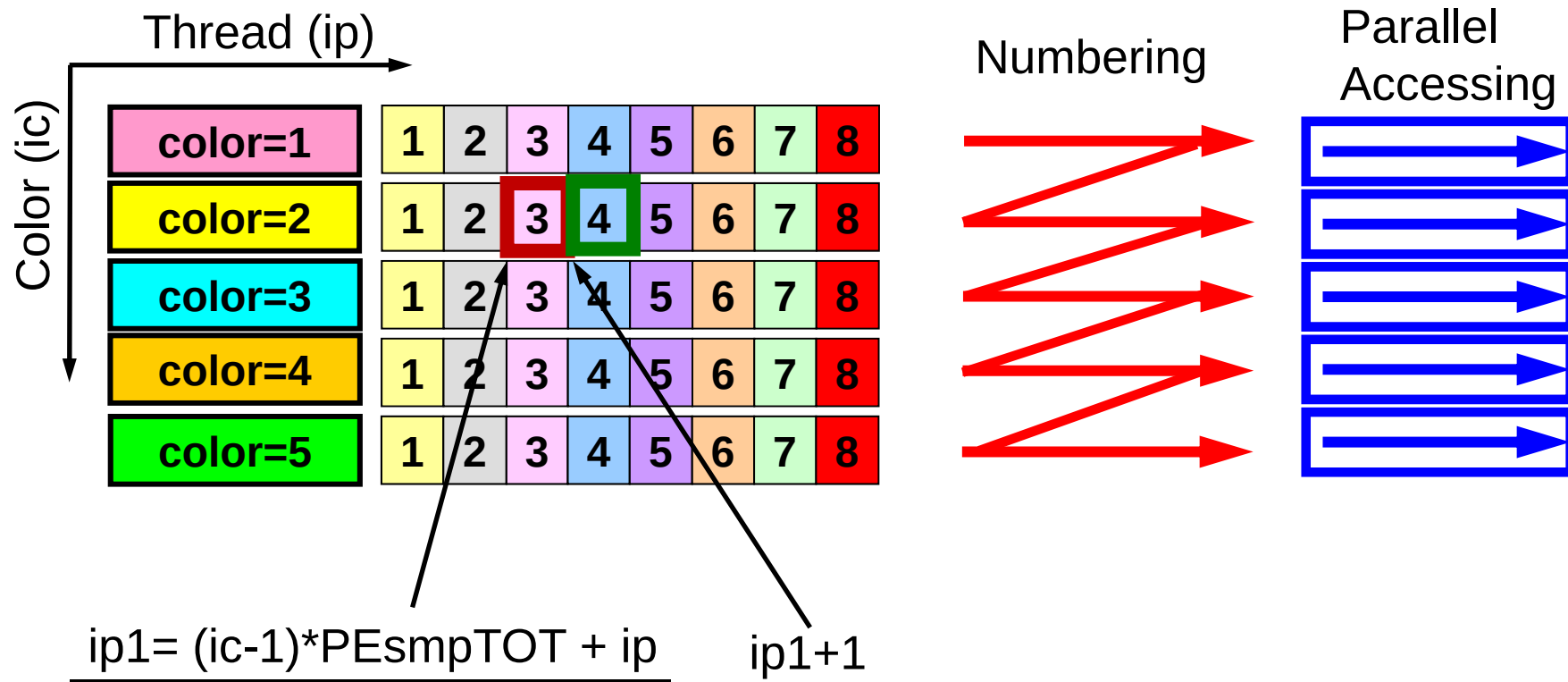
**Sequential
Reordering
(2/4)
poi_gen-1**

Coalesced

```

!$omp parallel private(ip, ip1, i, WVAL, k)
  do ic= 1, NCOLORTot
!$omp do
    do ip= 1, PEsmptOT
      ip1= (ic-1)*PEsmptOT + ip
      do i= SMPindex(ip1-1)+1, SMPindex(ip1) ...

```

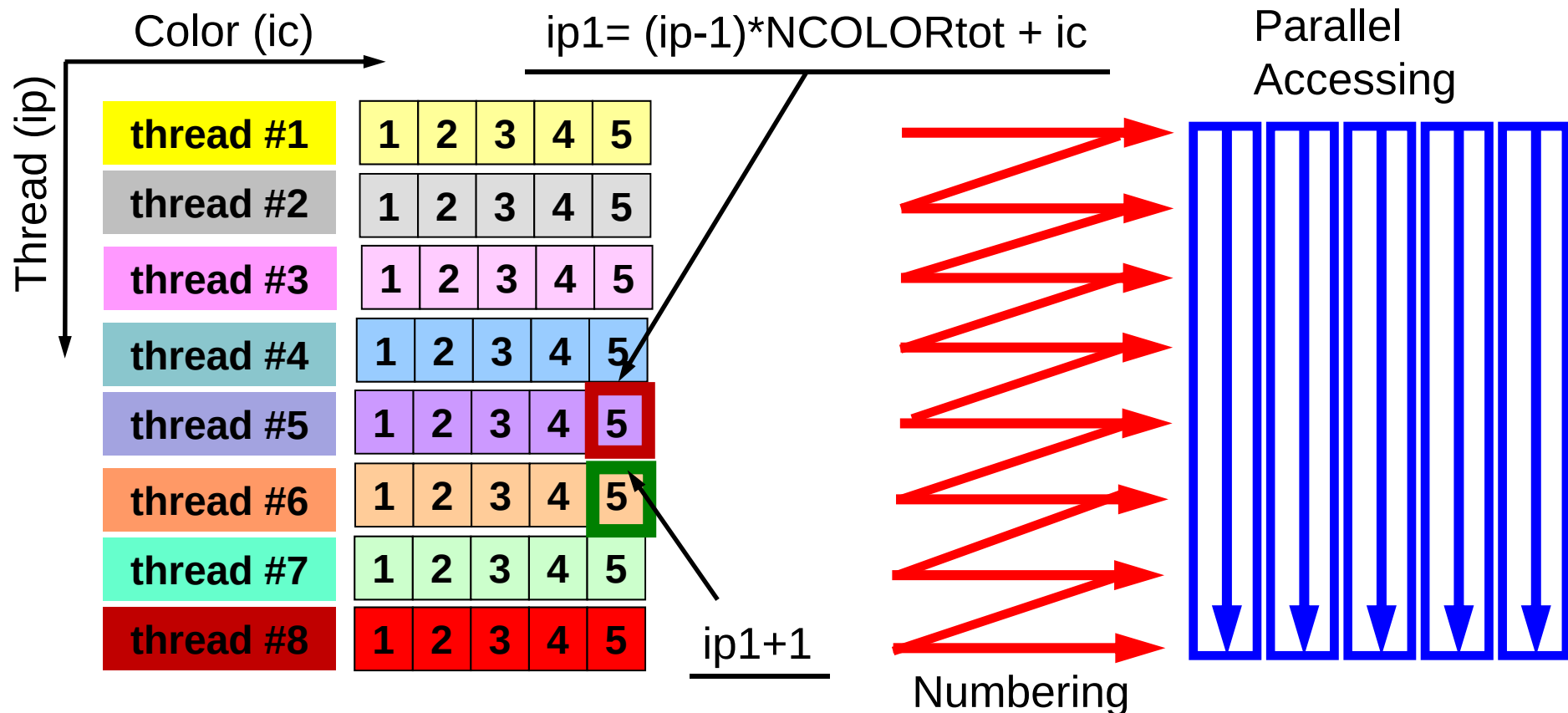


Sequential

```

!$omp parallel private(ip, ip1, i, WVAL, k)
  do ic= 1, NCOLORTot
!$omp do
    do ip= 1, PEsmptOT
      ip1= (ip-1)*NCOLORtot + ic
      do i= SMPindex(ip1-1)+1, SMPindex(ip1) ...

```



```

do ip= 1, PEsmptTOT
  do ic= 1, NCOLORTtot
    icNS= SMPindex_new((ip-1)*NCOLORtot + ic-1)
    ic01= SMPindex((ic-1)*PEsmptTOT + ip-1) + 1
    ic02= SMPindex((ic-1)*PEsmptTOT + ip )
    icou= 0
    do k= ic01, ic02
      icel= NEWtoOLD(k)
      icou= icou + 1
      icelN= icNS + icou
      OLDtoNEWnew(icel )= icelN
      NEWtoOLDnew(icelN)= icel
    enddo
  enddo
enddo

```

Sequential Reordering (3/4) poi_gen-2

```

!$omp parallel do private (ip, icel, ic0, ik0)
  do ip = 1, PEsmptTOT
    do icel= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
      ic0 = NEWtoOLDnew(icel)
      ik0 = OLDtoNEW(ic0)
      indexL(icel)= INL(ik0)
      indexU(icel)= INU(ik0)
    enddo
  enddo

```

OLDtoNEWnew: Original -> Sequential
NEWtoOLDnew: Sequential -> Original

```

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

```

-Original: Initial
-Coalesced
-Sequential

```

!$omp parallel do private (ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6) &
!$omp&                private (coef, j, ii, jj, kk) &
!$omp&                private (ik0, icN10, icN20, icN30, icN40, icN50, icN60)
do ip = 1, PEsmptOT
do icel= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
ic0 = NEWtoOLDnew(icel)
ik0 = OLDtoNEW(ic0)

icN10= NEIBcell(ic0, 1)
icN20= NEIBcell(ic0, 2)
icN30= NEIBcell(ic0, 3)
icN40= NEIBcell(ic0, 4)
icN50= NEIBcell(ic0, 5)
icN60= NEIBcell(ic0, 6)

if (icN50.ne.0) then
icN5= OLDtoNEW(icN50)
coef= RDZ * ZAREA
D(icel)= D(icel) - coef

if (icN5.lt.ik0) then
do j= 1, INL(ik0)
if (IAL(j, ik0).eq. icN5) then
itemL(j+indexL(icel-1))= OLDtoNEWnew(icN50)
AL(j+indexL(icel-1))= coef
exit
endif
enddo
else
do j= 1, INU(ik0)
if (IAU(j, ik0).eq. icN5) then
itemU(j+indexU(icel-1))= OLDtoNEWnew(icN50)
AU(j+indexU(icel-1))= coef
exit
endif
enddo
endif
endif

```

Sequential Reordering (4/4) poi_gen-3

icel: Sequential
ic0: Original
ik0: Coalesced

icN50: Original
icN5 : Coalesced

icN5>ik0: Upper (AU)
icN5<ik0: Lower (AL)

Forward Substitution

```

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

```

Color #1	Thread #1-#(Pe)
Color #2	Thread #1-#(Pe)
Color #3	Thread #1-#(Pe)
Color #4	Thread #1-#(Pe)
	⋮
Color #Nc	Thread #1-#(Pe)

Coalesced

```

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

```

Thread #1	Color #1-#(Nc)
Thread #2	Color #1-#(Nc)
Thread #3	Color #1-#(Nc)
Thread #4	Color #1-#(Nc)
	⋮
Thread # (Pe)	Color #1-#(Nc)

Sequential

Mat-Vec

```

!$omp parallel do private(ip, i, VAL, k)
  do ip= 1, PEsmptOT
    do i = SMPindexG(ip-1)+1, SMPindexG(ip)
      VAL= D(i)*W(i, P)
      do k= indexL(i-1)+1, indexL(i)
        VAL= VAL + AL(k)*W(itemL(k), P)
      enddo
      do k= indexU(i-1)+1, indexU(i)
        VAL= VAL + AU(k)*W(itemU(k), P)
      enddo
      W(i, Q)= VAL
    enddo
  enddo
!$omp end parallel do

```

METHOD=0

```

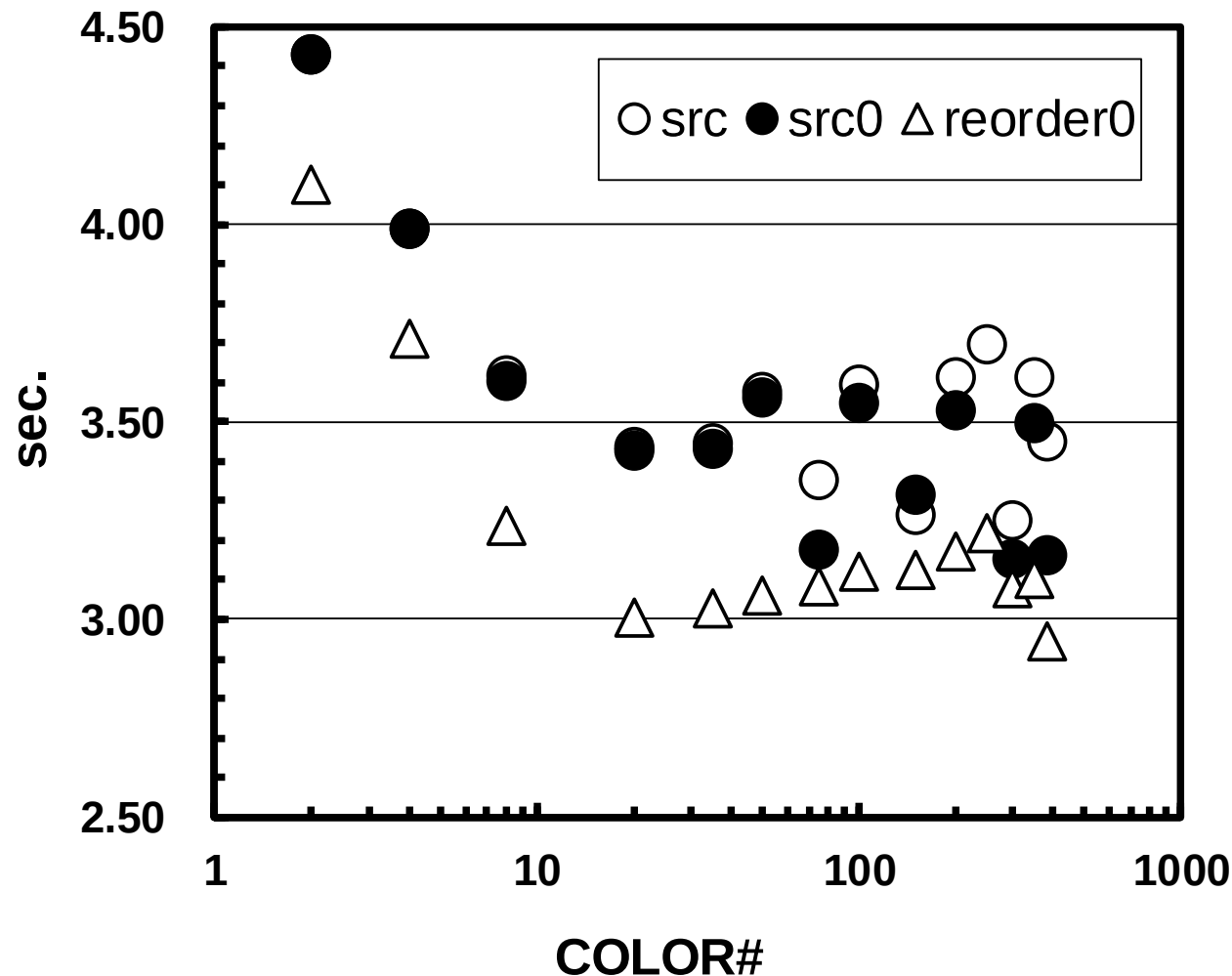
!$omp parallel do private(ip, i, VAL, k)
  do ip= 1, PEsmptOT
    do i= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
      VAL= D(i)*W(i, P)
      do k= indexL(i-1)+1, indexL(i)
        VAL= VAL + AL(k)*W(itemL(k), P)
      enddo
      do k= indexU(i-1)+1, indexU(i)
        VAL= VAL + AU(k)*W(itemU(k), P)
      enddo
      W(i, Q)= VAL
    enddo
  enddo
!$omp end parallel do

```

METHOD=1

Comp. Time for ICCG, CM-RCM

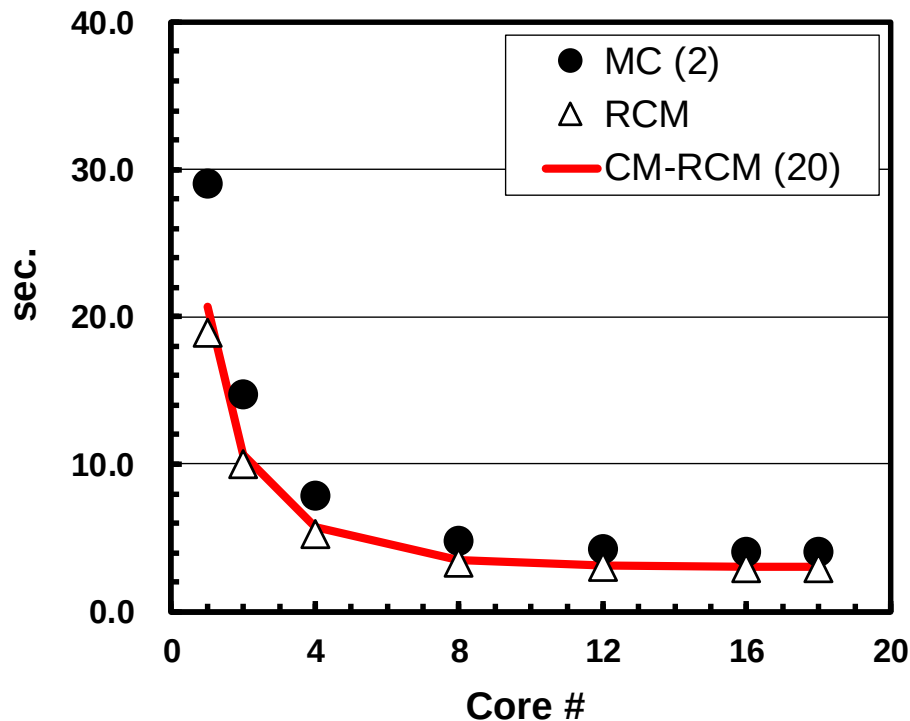
Generally “sequential (reorder0)” is stable and faster than “coalesced (src, src0)”. Effects should be more significant in cases with more colors, but ... (16 threads)



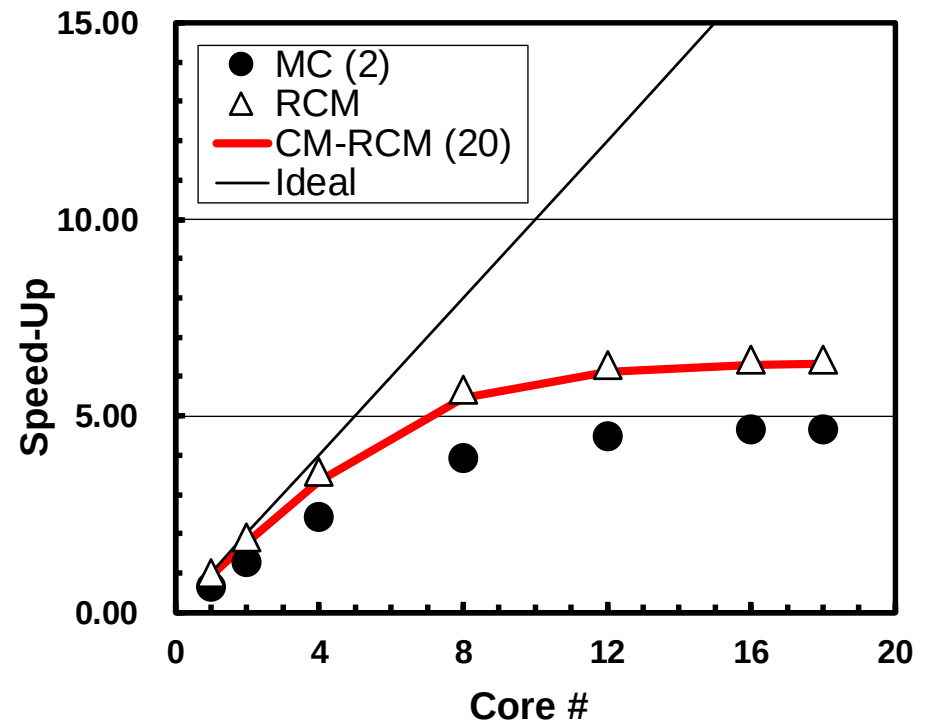
Results on Reedbush-U : 128³

18 cores, reorder0: MC (2-colors) : 424 iter's, 4.05 sec.
RCM (382-levels) : 287 iter's, 2.94 sec.
CM-RCM (Nc=20) : 318 iter's, 2.99 sec.

Computation Time



Speed-Up based on RCM with 1 thread



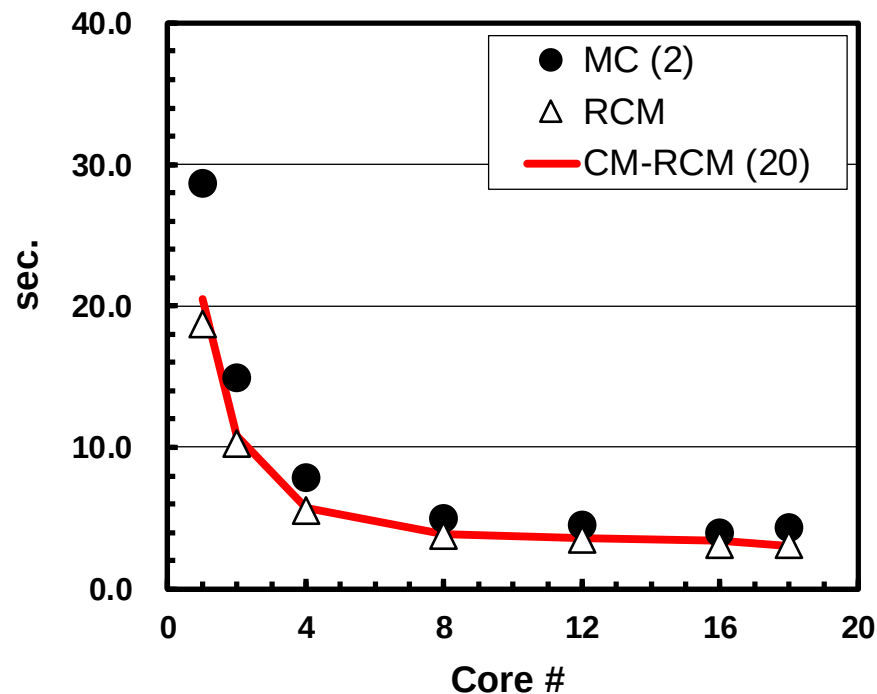
Results on Reedbush-U: 128³

18 cores, src: MC (2-colors) : 424 iter's, 4.38 sec.

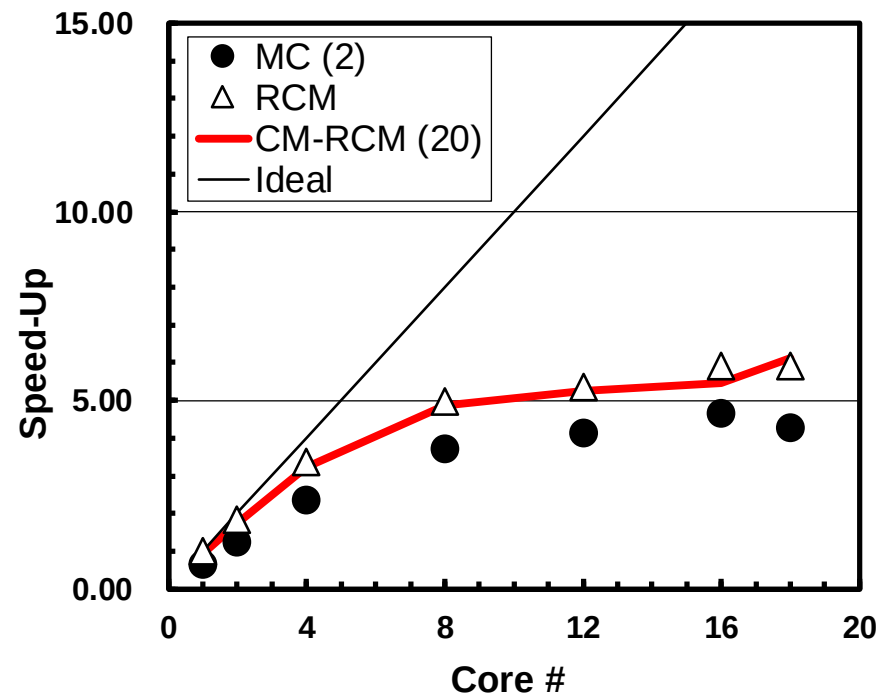
RCM (382-levels) : 287 iter's, 3.16 sec.

CM-RCM (Nc=20) : 318 iter's, 3.06 sec.

Computation Time

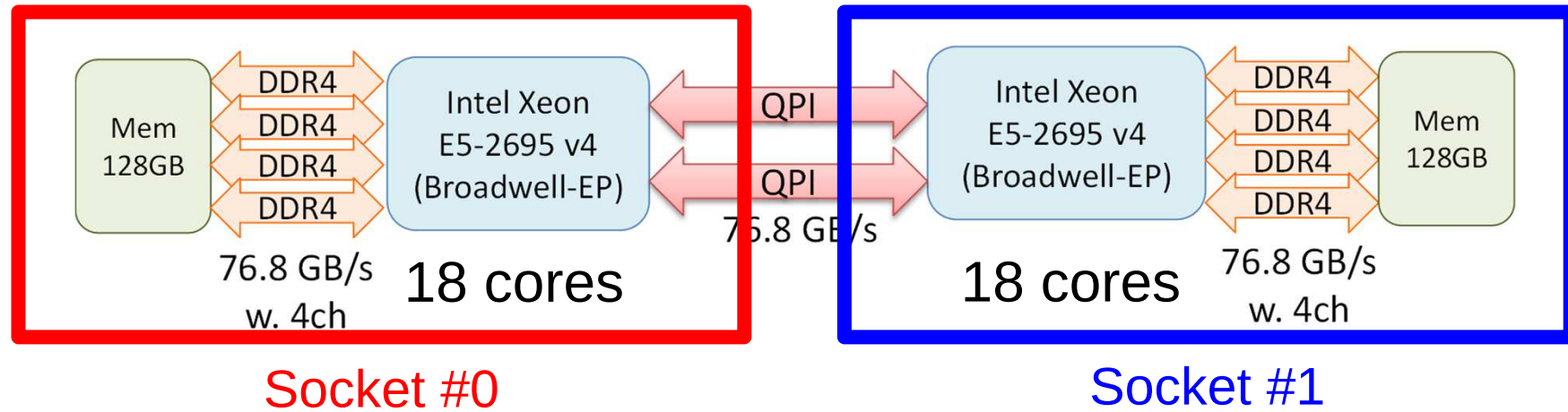


Speed-Up based on RCM with 1 thread



Reedbush-U

1-node: 2-CPU's/sockets



- Each Node of Reedbush-U
 - 2 Sockets (CPU's) of Intel Broadwell-EP
 - Each socket has 18 cores
- Each core of a socket can access to the memory on the other socket : NUMA (Non-Uniform Memory Access)
- Utilization of the local memory is more efficient
- So far, only a single socket has been used
 - Let's utilize both sockets

First Touch Data Placement

“Patterns for Parallel Programming” Mattson, T.G. et al.

To reduce memory traffic in the system, it is important to keep the data close to the PEs that will work with the data (e.g. NUMA control).

On NUMA computers, this corresponds to making sure the pages of memory are allocated and “owned” by the PEs that will be working with the data contained in the page.

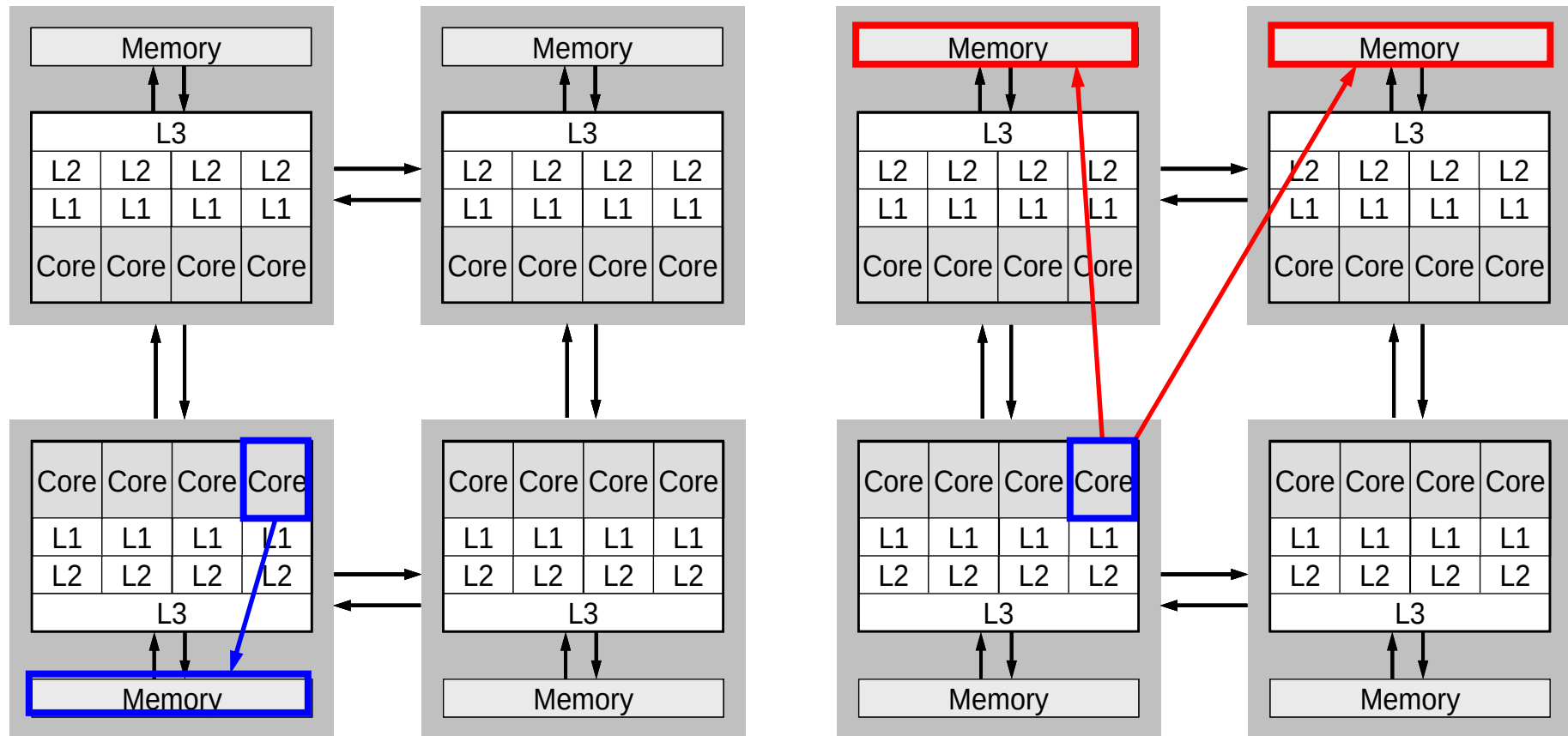
The most common NUMA page-placement algorithm is the “first touch” algorithm, in which the PE first referencing a region of memory will have the page holding that memory assigned to it.

A very common technique in OpenMP program for optimization is to initialize data in parallel using the same loop schedule as will be used later in the computations.

Summary: First Touch Data Placement

- On NUMA architecture (Non-Uniform Memory Access), “pages of memory” are not allocated when variables and arrays are declared/allocated in the program.
- “Pages” are allocated at the local memory of the “socket” for the “core/thread” that first touches the variables and/or arrays.
- If the pages are not on the local memory of the socket for each thread, performance of the program is very bad.
- A very common technique in OpenMP program for optimization is to initialize data in parallel using the same loop schedule as will be used later in the computations.
- You have to consider this if you use two sockets of the Reedbush-U system for a single OpenMP program
 - Not needed for a single socket case

Local/Remote Memory



Local Memory

Remote Memory

Control Data: INPUT.DAT

128 128 128	NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00	DX/DY/DZ
1.0e-08	EPSICC
36	PEsmpTOT
-50	NCOLORTot
0	NFLAG (0 or 1)
0	METHOD

- **PEsmpTOT**
 - Thread Number (= OMP_NUM_THREADS)
- **NCOLORTot**
 - Reordering Method + Initial Number of Colors/Levels
 - ≥ 2 : MC, =0: CM, =-1: RCM, $-2 \leq$: CMRCM
- **NFLAG**
 - =0: without first-touch, =1: with first-touch
- **METHOD**
 - Loop structure for Mat-Vec
 - =0: conventional way, =1: similar to forward/backward substitution

go2.sh: reorder0

```
#PBS -q u-lecture2
#PBS -N test
#PBS -l select=1:ncpus=36          ncpus=PEsmpTOT
#PBS -Wgroup_list=gt12
#PBS -l walltime=00:10:00
#PBS -e test2.err
#PBS -o test2.lst

cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh

export OMP_NUM_THREADS=36          =PEsmpTOT
export KMP_AFFINITY=granularity=fine,compact
numactl ./L3-rsol0
numactl ./L3-rsol0
numactl ./L3-rsol0
```

Array Initialization: NFLAG=0/1 (1/2)

```
!C
!C-- ARRAY init.
      if (NFLAG.eq.0) then
        BFORCE= 0.d0
        PHI    = 0.d0
        D      = 0.d0
        OLDtoNEWnew= 0
        NEWtoOLDnew= 0
      else
!$omp parallel do private (ip, icel)
        do ip= 1, PEsmpTOT
          do icel= SMPindex_new((ip-1)*NCOLORtot+1),
&                SMPindex_new(ip*NCOLORtot)
            BFORCE(icel)= 0.d0
            PHI    (icel)= 0.d0
            D(icel)      = 0.d0
            OLDtoNEWnew(icel)= 0
            NEWtoOLDnew(icel)= 0
          enddo
        enddo
!$omp end parallel do
      endif
```

Pages are allocated at the local memory of the master thread

Pages are allocated at the local memory of each thread

A very common technique in OpenMP program for optimization is to initialize data in parallel using the same loop schedule as will be used later in the computations.

Array Initialization: NFLAG=0/1 (2/2)

```
!C
!C-- ARRAY init.
      if (NFLAG.eq.0) then
        BFORCE= 0.d0
        PHI    = 0.d0
        D      = 0.d0
        OLDtoNEWnew= 0
        NEWtoOLDnew= 0
      else
!$omp parallel do private (ip, icel)
        do ip= 1, PEsmptOT
          do icel= SMPindex_new((ip-1)*NCOLORtot+1),
&                SMPindex_new(ip*NCOLORtot)
            BFORCE(icel)= 0.d0
            PHI    (icel)= 0.d0
            D(icel)      = 0.d0
            OLDtoNEWnew(icel)= 0
            NEWtoOLDnew(icel)= 0
          enddo
        enddo
!$omp end parallel do
      endif
```

Pages are allocated at the local memory of the master thread

Pages are allocated at the local memory of each thread

A very common technique in OpenMP program for optimization is to initialize data in parallel using the same loop schedule as will be used later in the computations.

Results: reoder0, L3-rsol0

128 128 128	NX/NY/NZ
1.00e-0 1.00e-0 1.00e0	DX/DY/DZ
1.0e-08	OMEGA, EPSICCG
36	PE _{smp} TOT
-1	NCOLORTot
0	NFLAG
0	METHOD

- 18 cores 2.94 sec
- 32 cores
 - NFLAG=0 2.51 sec
 - NFLAG=1 2.05 sec
- 36 cores
 - NFLAG=0 2.62 sec
 - NFLAG=1 1.96 sec

Programming Exercise for Report

- Implement “Sequential Reordering”
- Optimize “Two-Socket” cases with 36 cores by “First Touch Data Placement”
- Evaluate the performance of the code, and make comparison with that of the “Coalesced” code
- Original “Coalesced” Code
 - `/lustre/t07XXYY/multicore/L3/src0`
- Questions in the Class on July 23

Report

- Deadline: 17:00 August 21 (Tue), 2018
- Send files via e-mail at **nakajima (at) cc.u-tokyo.ac.jp**
- Report
 - Cover Page: Name and ID must be written.
 - No more than 20 pages including figures and tables (A4).
 - Strategy
 - Structure of the Program, Details of Modification
 - Validation
 - Numerical Experiments
 - Performance Analysis
 - Remarks
 - Source list of the entire program (not included in the 20 pages above)