

Introduction to Parallel Programming for Multicore/Manycore Clusters

Part B3: Parallel ICCG by OpenMP

Kengo Nakajima
Information Technology Center
The University of Tokyo

Parallel Version: OpenMP

- OpenMP version of L2-sol
 - Number of threads= “PEsmpTOT”
 - can be controlled in the program
- Fundamental Idea
 - Meshes in a same color/level are independent, therefore parallel/concurrent processing is possible for these meshes.

4-Colors, 4-Threads

Initial Mesh

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

4-Colors, 4-Threads

Initial Mesh

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

4-Colors, 4-Threads

Renumbering according to Color ID

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

4-Colors, 4-Threads

Meshes in a same color/level are independent, therefore parallel/concurrent processing is possible for these meshes, renumbered meshes are assigned to

threads

	45	61	46	62	47	63	48	64
thread #3	13	29	14	30	15	31	16	32
	41	57	42	58	43	59	44	60
thread #2	9	25	10	26	11	27	12	28
	37	53	38	54	39	55	40	56
thread #1	5	21	6	22	7	23	8	24
	33	49	34	50	35	51	36	52
thread #0	1	17	2	18	3	19	4	20

How to Run

```
>$ cd /work/gt80/t80xxx

>$ cp /work/gt00/z30088/ompf.tar .
>$ tar xvf ompc.tar

>$ cd ompf
>$ ls
    run    src    src0    reorder0

>$ cd src
>$ module load fj
>$ make
>$ cd ../run
>$ ls L3-sol
    L3-sol

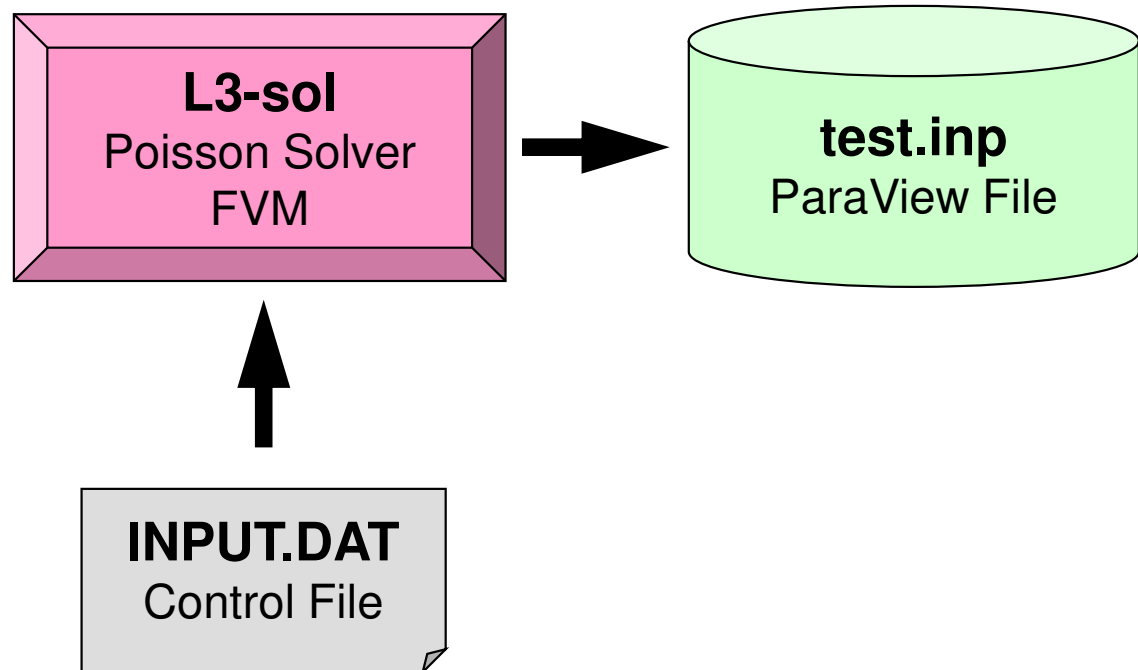
<modify "INPUT.DAT">
<modify "go1.sh">

>$ pjsub go1.sh
```

Files on Odyssey

- Location
 - **<\$O-L3>: /work/gt80/t80XXX/ompf**
 - <\$O-L3>/src, <\$O-L3>/run
- Compile & Run
 - Source Code
 - `cd <$O-L3>/src`
 - `make`
 - `<$O-L3>/run/L3-sol` execution file
 - Control Data
 - `<$O-L3>/run/INPUT.DAT`
 - Shell Script
 - `<$O-L3>/run/go1.sh`

Running the Program



Control Data: INPUT.DAT

```
128 128 128
1.00e-00 1.00e-00 1.00e-00
1.0e-08
24
-10
```

```
NX/NY/NZ
DX/DY/DZ
EPSICCG
PEsmpTOT
NCOLORtot
```

- **NX, NY, NZ**

- Number of meshes in X/Y/Z dir.

- **DX, DY, DZ**

- Size of meshes

- **EPSICCG**

- Convergence Criteria for ICCG

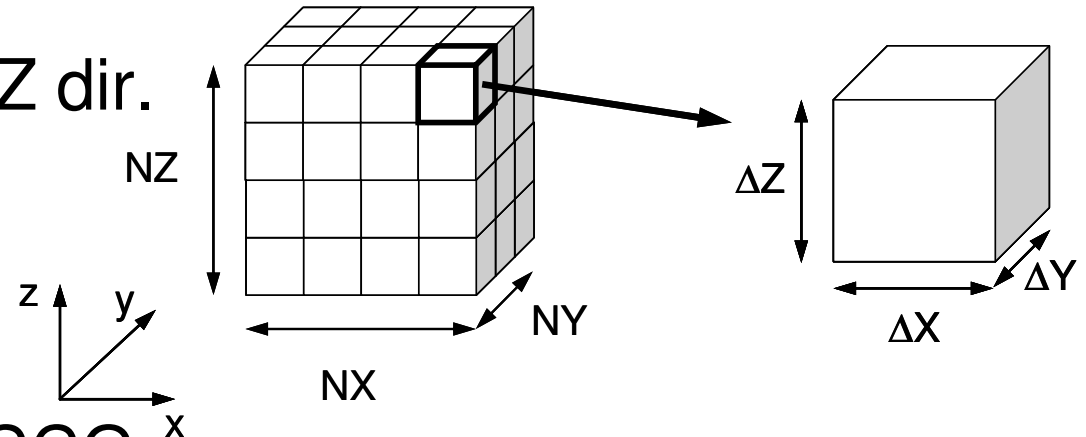
- **PEsmpTOT**

- Thread Number (**--omp thread=XX**)

- **NCOLORtot**

- Reordering Method + Initial Number of Colors/Levels

- ≥ 2 : MC, =0: CM, =-1: RCM, $-2 \leq$: CMRCM



Job Script: go1.sh

- `/work/gt80/t80XXX/ompf/run/go1.sh`
- Scheduling + Shell Script

```
#!/bin/sh
#PJM -N "go1"                Job Name (not required)
#PJM -L rscgrp=lecture0-o    Name of Queue (Resource Group)
#PJM -L node=1              Node # (=1)
#PJM --omp thread=12        Thread # (= PEsmptTOT)
#PJM -L elapse=00:15:00     Elapsed Computation Time
#PJM -g gt80                Group Name (Wallet)
#PJM -j
#PJM -e err                 Standard Error
#PJM -o test1.lst           Standard Output

module load fj
export OMP_NUM_THREADS=12    Thread # (--omp thread=XX)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl ./L3-sol
numactl -C 12-23 -m 4 ./L3-sol
```

- Applying OpenMP to L2-sol
- Examples
- Optimization + Exercise

Applying OpenMP to “L2-sol”

- on ICCG solver
- Dot Products, DAXPY, Mat-Vec
 - NO data dependency: Just insert directives
- Preconditioning (IC Factorization, Forward/Backward Substitution)
 - NO data dependency in same color: Parallel processing is possible for meshes in same color

Just inserting directives works fine, but ... (1/2) (Mat-Vec)

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

- Thread number cannot be handled in the program
- This may work better on GPU and manycore's

Just inserting directives works fine, but ... (2/2) (Forward Substitution)

```
do icol= 1, NCOLORtot
!$omp parallel do private (i, VAL, k)
  do i= COLORindex(icol-1)+1, COLORindex(icol)
    VAL= D(i)
    do k= indexL(i-1)+1, indexL(i)
      VAL= VAL - (AL(k)**2) * DD(itemL(k))
    enddo
    DD(i)= 1. d0/VAL
  enddo
!$omp end parallel do
enddo
```

- Thread number cannot be handled in the program
- This may work better on GPU and manycore's

Parallelize ICCG Method by OpenMP

- Dot Product: OK
- DAXPY: OK
- Matrix-Vector Multiply: OK
- Preconditioning

Main Program

```

program MAIN

use STRUCT
use PCG
use solver_ICCG_mc

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

PHI= 0.d0

call solve_ICCG_mc
&      ( ICELTOT, NPL, NPU, indexL, itemL, indexU, itemU, D,      &
&      BFORCE, PHI, AL, AU, NCOLORTOT, PEsmptTOT,              &
&      SMPindex, SMPindexG, EPSICCG, ITR, IER)

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

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

call OUTUCD

stop
end

```

module STRUCT

```

module STRUCT

  use omp_lib
  include 'precision.inc'

!C
!C-- METRICS & FLUX
  integer (kind=kint) :: ICELTOT, ICELTOTp, N
  integer (kind=kint) :: NX, NY, NZ, NXP1, NYP1, NZP1, IBNODTOT
  integer (kind=kint) :: NXc, NYc, NZc

  real (kind=kreal) ::
&    DX, DY, DZ, XAREA, YAREA, ZAREA, RDX, RDY, RDZ,
&    RDX2, RDY2, RDZ2, R2DX, R2DY, R2DZ

  real (kind=kreal), dimension(:), allocatable ::
&    VOLCEL, VOLNOD, RVC, RVN

  integer (kind=kint), dimension(:,:), allocatable ::
&    XYZ, NEIBcell

!C
!C-- BOUNDARYs
  integer (kind=kint) :: ZmaxCELtot
  integer (kind=kint), dimension(:), allocatable :: BC_INDEX, BC_NOD
  integer (kind=kint), dimension(:), allocatable :: ZmaxCEL

!C
!C-- WORK
  integer (kind=kint), dimension(:,:), allocatable :: IWKX
  real (kind=kreal), dimension(:,:), allocatable :: FCV

  integer (kind=kint) :: PEsmptTOT

end module STRUCT

```

ICELTOT :

Number of meshes (NX x NY x NZ)

N :

Number of modes

NX, NY, NZ :

&
& Number of meshes in x/y/z directions

NXP1, NYP1, NZP1 :

&
Number of nodes in x/y/z directions

IBNODTOT :

= NXP1 x NYP1

XYZ (ICELTOT, 3) :

Location of meshes

NEIBcell (ICELTOT, 6) :

Neighboring meshes

PEsmptTOT :

Number of threads

module PCG

```

module PCG

integer, parameter :: N2= 256
integer :: NUmax, NLmax, NCOLORTot, NCOLORK, NU, NL
integer :: NPL, NPU
integer :: METHOD, ORDER_METHOD

real(kind=8) :: EPSICCG

real(kind=8), dimension(:), allocatable :: D, PHI, BFORCE
real(kind=8), dimension(:), allocatable :: AL, AU

integer, dimension(:), allocatable :: INL, INU, COLORindex
integer, dimension(:), allocatable :: SMPindex, SMPindexG
integer, dimension(:), allocatable :: OLDtoNEW, NEWtoOLD

integer, dimension(:, :), allocatable :: IAL, IAU

integer, dimension(:), allocatable :: indexL, itemL
integer, dimension(:), allocatable :: indexU, itemU
end module PCG

```

NCOLORTot Total number of colors/levels
COLORindex Index of number of meshes in each color/level
(0:NCOLORTot) (COLORindex(icol) - COLORindex(icol-1))

SMPindex (0:NCOLORTot*PEsmpTOT)

SMPindexG (0:PEsmpTOT)

OLDtoNEW, NEWtoOLD Reference table before/after renumbering

Variables/Arrays for Matrix (1/2)

Name	Type	Content
D (N)	R	Diagonal components of the matrix (N= ICELTOT)
BFORCE (N)	R	RHS vector
PHI (N)	R	Unknown vector
indexL (0:N) , indexU (0:N)	I	# of L/U non-zero off-diag. comp. (CRS)
NPL, NPU	I	Total # of L/U non-zero off-diag. comp. (CRS)
itemL (NPL) , itemU (NPU)	I	Column ID of L/U non-zero off-diag. comp. (CRS)
AL (NPL) , AU (NPU)	R	L/U non-zero off-diag. comp. (CRS)

Name	Type	Content
NL, NU	I	MAX. # of L/U non-zero off-diag. comp. for each mesh (=6)
INL (N) , INU (N)	I	# of L/U non-zero off-diag. comp.
IAL (NL, N) , IAU (NU, N)	I	Column ID of L/U non-zero off-diag. comp.

Variables/Arrays for Matrix (2/2)

Name	Type	Content
NCOLORtot	I	<u>Input:</u> reordering method + initial number of colors/levels ≥ 2 : MC, =0: CM, =-1: RCM, $-2 \leq$: CMRCM <u>Output:</u> Final number of colors/levels
COLORindex (0:NCOLORtot)	I	Number of meshes at each color/level 1D compressed array Meshes in icolth color/level are stored in this array from COLORindex(icol-1)+1 to COLORindex(icol)
NEWtoOLD (N)	I	Reference array from New to Old numbering
OLDtoNEW (N)	I	Reference array from Old to New numbering
PEsmpTOT	I	Number of Threads
SMPindex (0:NCOLORtot*PEsmpTOT)	I	Array for OpenMP Operations (for Loops with Data Dependency)
SMPindexG (0:PEsmpTOT)	I	Array for OpenMP Operations (for Loops without Data Dependency)

Main Program

```

program MAIN

use STRUCT
use PCG
use solver_ICCG_mc

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

PHI= 0. d0

call solve_ICCG_mc                                     &
&      ( ICELTOT, NPL, NPU, indexL, itemL, indexU, itemU, D,      &
&      BFORCE, PHI, AL, AU, NCOLORTot, PEsmptTOT,              &
&      SMPindex, SMPindexG, EPSICCG, ITR, IER)

```

input: reading INPUT.DAT

```
!C
!C***
!C*** INPUT
!C***
!C
!C INPUT CONTROL DATA
!C
      subroutine INPUT
      use STRUCT
      use PCG
      implicit REAL*8 (A-H, O-Z)
      character*80 CNTFIL
!C
!C-- CNTL. file
      open (11, file='INPUT.DAT', status='unknown')
      read (11,*) NX, NY, NZ
      read (11,*) DX, DY, DZ
      read (11,*) EPSICCG
      read (11,*) PEsmpTOT
      read (11,*) NCOLORtot
      close (11)
!C==
      return
      end
```

- **PEsmpTOT**
 - Thread Number
- **NCOLORtot**
 - Reordering Method + Initial Number of Colors/Levels
 - ≥ 2 : MC
 - $= 0$: CM
 - $= -1$: RCM
 - $-2 \leq$: CMRCM

```
100 100 100
1.00e-02 5.00e-02 1.00e-02
1.00e-08
24
100
```

```
NX/NY/NZ
DX/DY/DZ
EPSICCG
PEsmpTOT
NCOLORtot
```

cell_metrics

```

!C
!C***
!C*** CELL_METRICS
!C***
!C
      subroutine CELL_METRICS
      use STRUCT
      use PCG
      implicit REAL*8 (A-H, O-Z)

!C
!C-- ALLOCATE
      allocate (VOLCEL(ICELTOT))
      allocate (RVC(ICELTOT))

!C
!C-- VOLUME, AREA, PROJECTION etc.
      XAREA= DY * DZ
      YAREA= DX * DZ
      ZAREA= DX * DY

      RDX= 1. d0 / DX
      RDY= 1. d0 / DY
      RDZ= 1. d0 / DZ

      RDX2= 1. d0 / (DX**2)
      RDY2= 1. d0 / (DY**2)
      RDZ2= 1. d0 / (DZ**2)

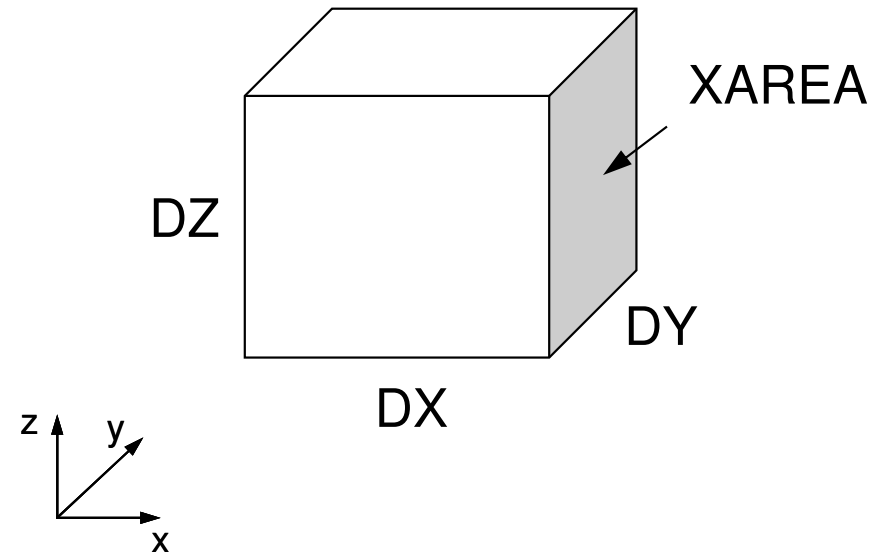
      R2DX= 1. d0 / (0. 50d0*DX)
      R2DY= 1. d0 / (0. 50d0*DY)
      R2DZ= 1. d0 / (0. 50d0*DZ)

      V0= DX * DY * DZ
      RV0= 1. d0/V0

      VOLCEL= V0
      RVC    = RV0

      return
      end

```



Main Program

```

program MAIN

  use STRUCT
  use PCG
  use solver_ICCG_mc

  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

  PHI= 0.d0

  call solve_ICCG_mc
&      ( ICELTOT, NPL, NPU, indexL, itemL, indexU, itemU, D,
&      BFORCE, PHI, AL, AU, NCOLORTot, PEsmptTOT,
&      SMPindex, SMPindexG, EPSICCG, ITR, IER)
&

```

poi_gen (1/9)

```

subroutine POI_GEN

  use STRUCT
  use PCG

  implicit REAL*8 (A-H, O-Z)

!C
!C-- INIT.
  nn = ICELTOT
  nnp= ICELTOTp

  NU= 6
  NL= 6

  allocate (BFORCE(nn), D(nn), PHI(nn))
  allocate (INL(nn), INU(nn), IAL(NL, nn), IAU(NU, nn))

  PHI   = 0. d0
  D     = 0. d0
  BFORCE= 0. d0

  INL= 0
  INU= 0
  IAL= 0
  IAU= 0

```

poi_gen (2/9)

```

!C
!C +-----+
!C | CONNECTIVITY |
!C +-----+
!C===

```

```

do icel= 1, ICELTOT

```

```

  icN1= NEIBcell( icel, 1)
  icN2= NEIBcell( icel, 2)
  icN3= NEIBcell( icel, 3)
  icN4= NEIBcell( icel, 4)
  icN5= NEIBcell( icel, 5)
  icN6= NEIBcell( icel, 6)

```

```

  if (icN5.ne.0.and.icN5.le.ICELTOT) then
    icou= INL( icel) + 1
    IAL( icou, icel)= icN5
    INL(      icel)= icou
  endif

```

```

  if (icN3.ne.0.and.icN3.le.ICELTOT) then
    icou= INL( icel) + 1
    IAL( icou, icel)= icN3
    INL(      icel)= icou
  endif

```

```

  if (icN1.ne.0.and.icN1.le.ICELTOT) then
    icou= INL( icel) + 1
    IAL( icou, icel)= icN1
    INL(      icel)= icou
  endif

```

```

  if (icN2.ne.0.and.icN2.le.ICELTOT) then
    icou= INU( icel) + 1
    IAU( icou, icel)= icN2
    INU(      icel)= icou
  endif

```

```

  if (icN4.ne.0.and.icN4.le.ICELTOT) then
    icou= INU( icel) + 1
    IAU( icou, icel)= icN4
    INU(      icel)= icou
  endif

```

```

  if (icN6.ne.0.and.icN6.le.ICELTOT) then
    icou= INU( icel) + 1
    IAU( icou, icel)= icN6
    INU(      icel)= icou
  endif

```

```

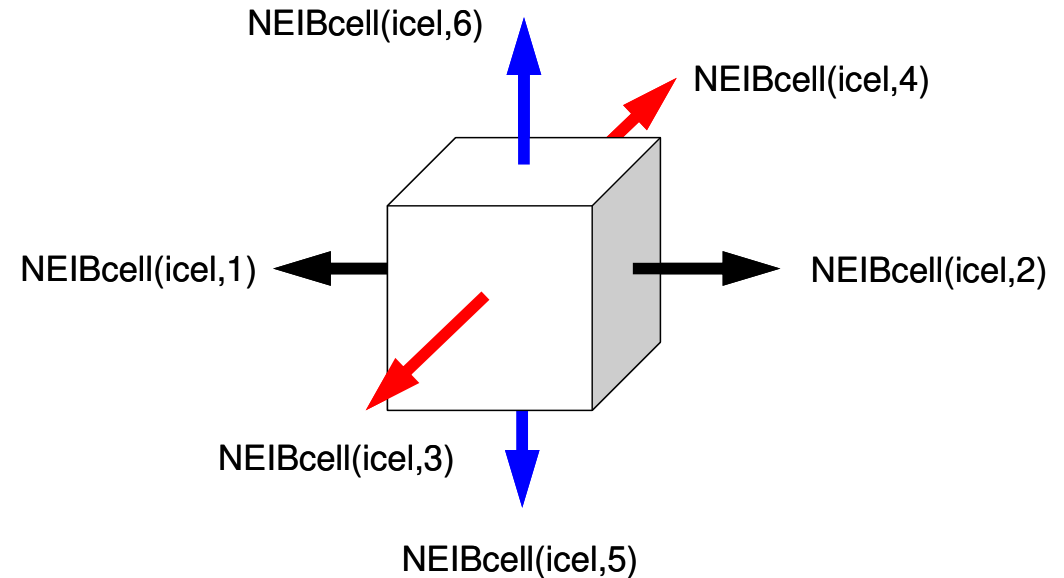
enddo

```

```

!C===

```



Lower Triangular Part

NEIBcell(icel,5)= icel – NX*NY

NEIBcell(icel,3)= icel – NX

NEIBcell(icel,1)= icel – 1

poi_gen (2/9)

```
!C
!C +-----+
!C | CONNECTIVITY |
!C +-----+
!C ==
```

```
do icel= 1, ICELTOT
  icN1= NEIBcell( icel, 1)
  icN2= NEIBcell( icel, 2)
  icN3= NEIBcell( icel, 3)
  icN4= NEIBcell( icel, 4)
  icN5= NEIBcell( icel, 5)
  icN6= NEIBcell( icel, 6)

  if (icN5.ne.0.and.icN5.le.ICELTOT) then
    icou= INL( icel) + 1
    IAL( icou, icel)= icN5
    INL(      icel)= icou
  endif

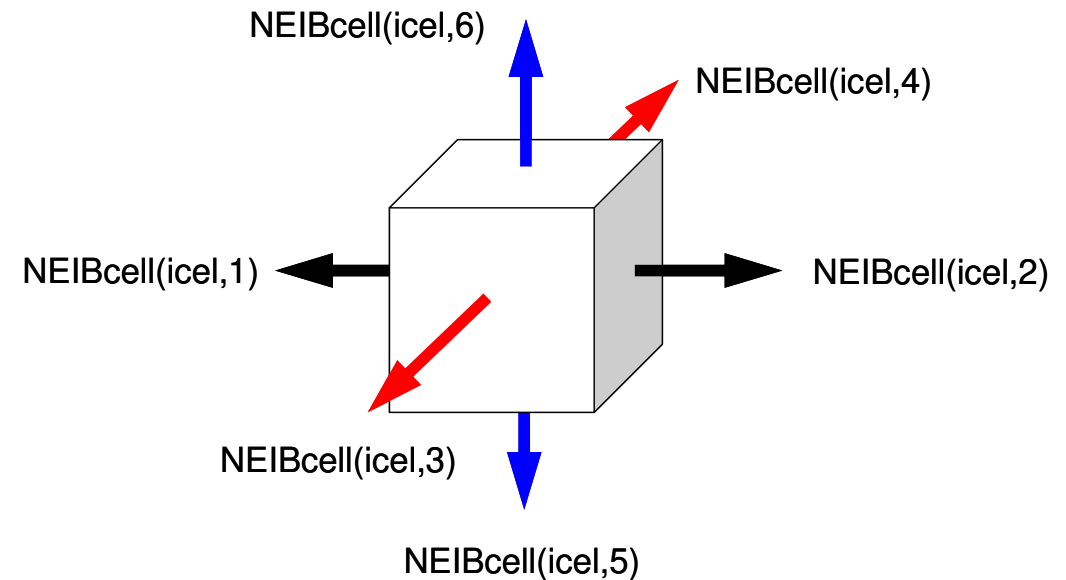
  if (icN3.ne.0.and.icN3.le.ICELTOT) then
    icou= INL( icel) + 1
    IAL( icou, icel)= icN3
    INL(      icel)= icou
  endif

  if (icN1.ne.0.and.icN1.le.ICELTOT) then
    icou= INL( icel) + 1
    IAL( icou, icel)= icN1
    INL(      icel)= icou
  endif

  if (icN2.ne.0.and.icN2.le.ICELTOT) then
    icou= INU( icel) + 1
    IAU( icou, icel)= icN2
    INU(      icel)= icou
  endif

  if (icN4.ne.0.and.icN4.le.ICELTOT) then
    icou= INU( icel) + 1
    IAU( icou, icel)= icN4
    INU(      icel)= icou
  endif

  if (icN6.ne.0.and.icN6.le.ICELTOT) then
    icou= INU( icel) + 1
    IAU( icou, icel)= icN6
    INU(      icel)= icou
  endif
enddo
!C ==
```



Upper Triangular Part

```
NEIBcell(icel, 2)= icel + 1
NEIBcell(icel, 4)= icel + NX
NEIBcell(icel, 6)= icel + NX*NY
```

poi_gen (3/9)

```

!C
!C +-----+
!C | MULTICOLORING |
!C +-----+
!C==
      allocate (OLDtoNEW(ICELTOT), NEWtoOLD(ICELTOT))
      allocate (COLORindex(0:ICELTOT))

111  continue
      write (*, '(//a,i8,a)') 'You have', ICELTOT, ' elements.'
      write (*, '( a )') 'How many colors do you need?'
      write (*, '( a )') '#COLOR must be more than 2 and'
      write (*, '( a,i8 )') '#COLOR must not be more than', ICELTOT
      write (*, '( a )') 'CM if #COLOR .eq. 0'
      write (*, '( a )') 'RCM if #COLOR .eq. -1'
      write (*, '( a )') 'CMRCM if #COLOR .le. -2'
      write (*, '( a )') '=>'

      if (NCOLORtot.gt.0) then
        call MC (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&              NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW)
      endif

      if (NCOLORtot.eq.0) then
        call CM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&              NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW)
      endif

      if (NCOLORtot.eq.-1) then
        call RCM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&              NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW)
      endif

      if (NCOLORtot.lt.-1) then
        call CMRCM (ICELTOT, NL, NU, INL, IAL, INU, IAU,
&              NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW)
      endif

      write (*, '(//a,i8,// )') '### FINAL COLOR NUMBER', NCOLORtot

```

Reordering

NCOLORtot > 1: Multicolor

NCOLORtot = 0: CM

NCOLORtot = -1: RCM

NCOLORtot < -1: CM-RCM

poi_gen (4/9)

```

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

do ic= 1, NCOLORtot
  do ip= 1, PEsmpTOT
    j1= (ic-1)*PEsmpTOT + ip
    j0= j1 - 1
    SMPindex(j1)= SMPindex(j0) + SMPindex(j1)
  enddo
enddo

allocate (SMPindexG(0:PEsmpTOT))
SMPindexG= 0
nn= ICELTOT / PEsmpTOT
nr= ICELTOT - nn*PEsmpTOT
do ip= 1, PEsmpTOT
  SMPindexG(ip)= nn
  if (ip.le.nr) SMPindexG(ip)= nn + 1
enddo

do ip= 1, PEsmpTOT
  SMPindexG(ip)= SMPindexG(ip-1) + SMPindexG(ip)
enddo

```

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

```

SMPindex:

for preconditioning

```

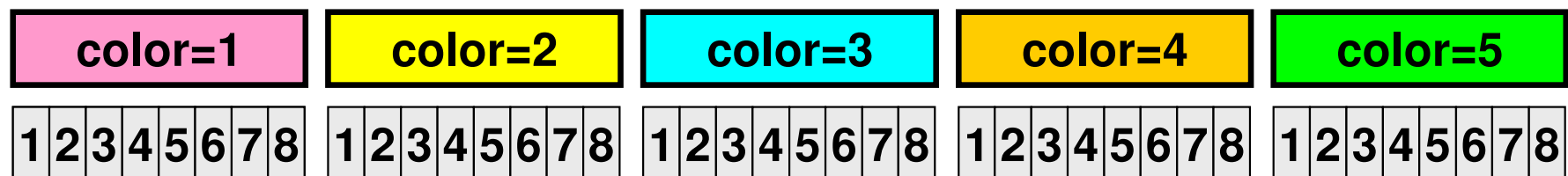
do ic= 1, NCOLORTot
!$omp parallel do ...
  do ip= 1, PEsmpTOT
    ip1= (ic-1)*PEsmpTOT+ip
    do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      (...)
    enddo
  enddo
!omp end parallel do
enddo

```

Initial Vector



Coloring
(5 colors)
+Ordering



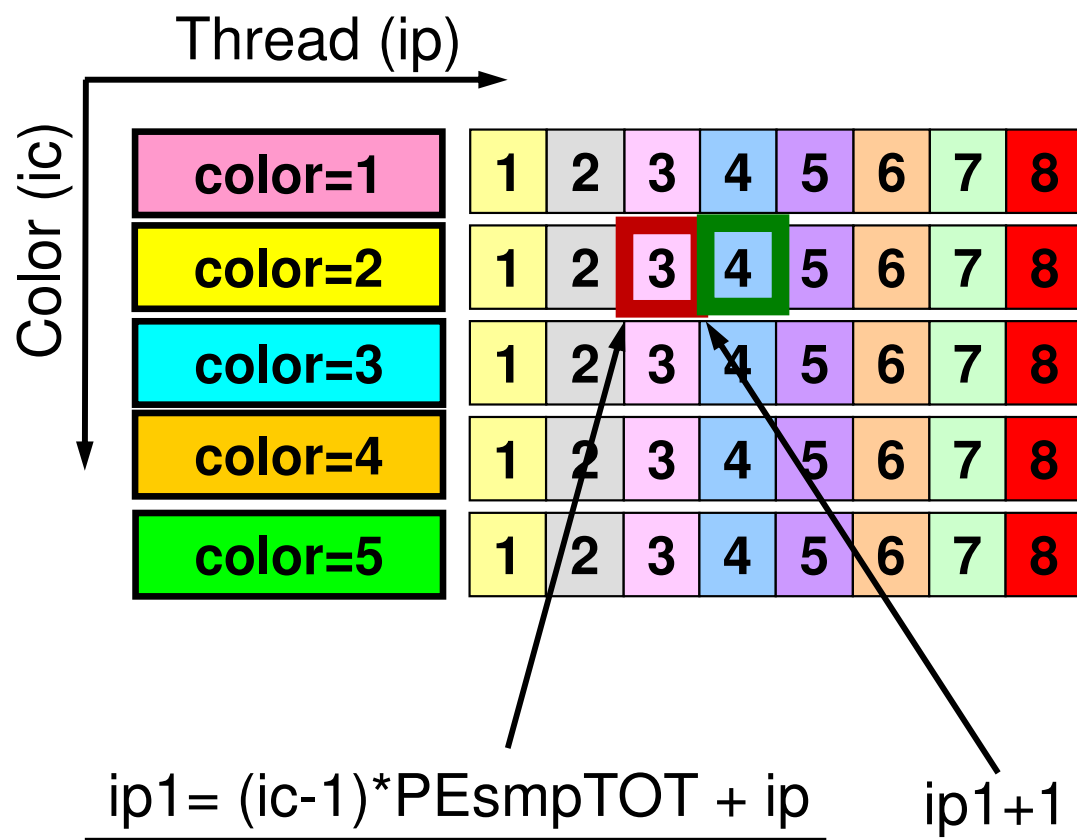
- 5-colors, 8-threads
- Meshes in same color are independent: parallel processing
- Reordering in ascending order according to color ID

SMPindex

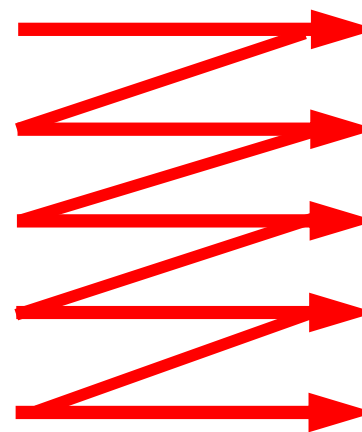
```

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

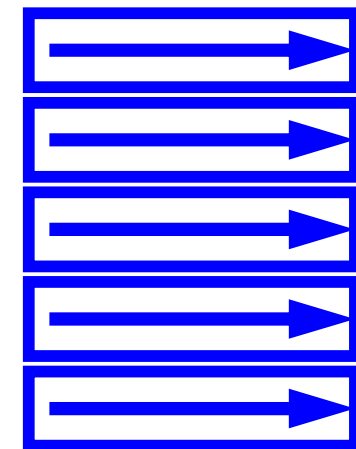
```



Numbering



Parallel Accessing



SMPindex

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

$nn1 = 200 - 100 = 100$
 $num = 100 / 8 = 12 \text{ (12.5)}$
 $nr = 100 - 12 * 8 = 4$
 $ip0 = (ic - 1) * \text{PEsmpTOT} \quad (ic: \text{ starting at } 1)$

$\text{SMPindex}(ip0) = 100$
 $\text{SMPindex}(ip0+1) = 113 \text{ (13 elements in the 1st thread)}$
 $\text{SMPindex}(ip0+2) = 126 \text{ (13 elements in the 2nd thread)}$
 $\text{SMPindex}(ip0+3) = 139 \text{ (13 elements in the 3rd thread)}$
 $\text{SMPindex}(ip0+4) = 152 \text{ (13 elements in the 4th thread)}$
 $\text{SMPindex}(ip0+5) = 164 \text{ (12 elements in the 5th thread)}$
 $\text{SMPindex}(ip0+6) = 176 \text{ (12 elements in the 6th thread)}$
 $\text{SMPindex}(ip0+7) = 188 \text{ (12 elements in the 7th thread)}$
 $\text{SMPindex}(ip0+8) = 200 \text{ (12 elements in the 8th thread)}$

poi_gen (4/9)

```

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

do ic= 1, NCOLORtot
  do ip= 1, PEsmpTOT
    j1= (ic-1)*PEsmpTOT + ip
    j0= j1 - 1
    SMPindex(j1)= SMPindex(j0) + SMPindex(j1)
  enddo
enddo

```

```

allocate (SMPindexG(0:PEsmpTOT))
SMPindexG= 0
nn= ICELTOT / PEsmpTOT
nr= ICELTOT - nn*PEsmpTOT
do ip= 1, PEsmpTOT
  SMPindexG(ip)= nn
  if (ip.le.nr) SMPindexG(ip)= nn + 1
enddo

```

```

do ip= 1, PEsmpTOT
  SMPindexG(ip)= SMPindexG(ip-1) + SMPindexG(ip)
enddo

```

!C===

```

!$omp parallel do ...
  do ip= 1, PEsmpTOT
    do i= SMPindexG(ip-1)+1, SMPindexG(ip)
      (...)
    enddo
  enddo
!$omp end parallel do

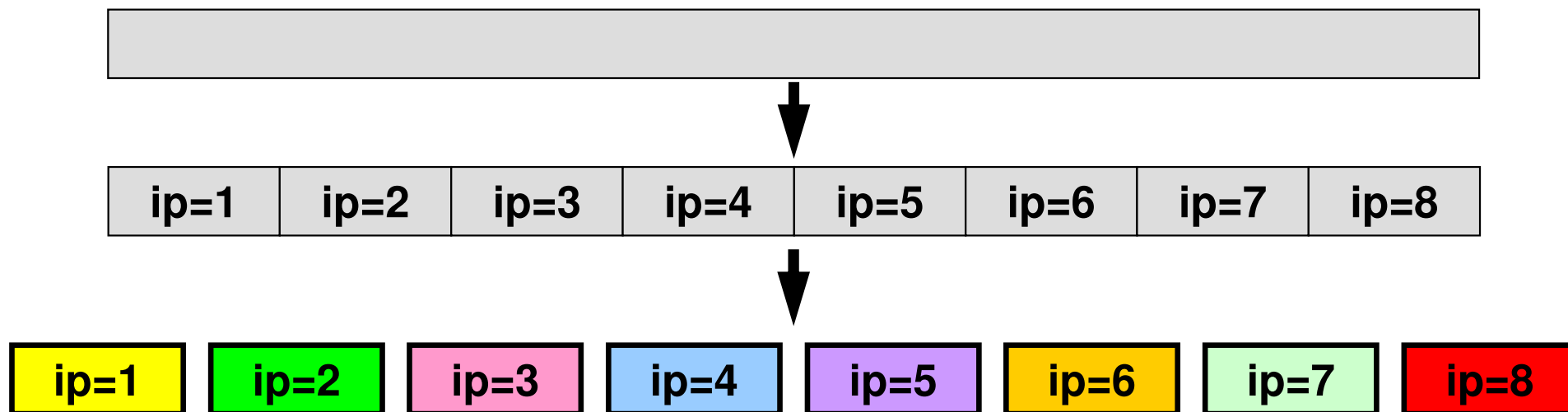
```

SMPindexG:

for Dot-products, DAXPY,
Mat-vec, and Poi-gen

SMPindexG

```
!$omp parallel do ...  
  do ip= 1, PEsmptOT  
    do i= SMPindexG(ip-1)+1, SMPindexG(ip)  
      (...)  
    enddo  
  enddo  
!$omp end parallel do
```



for Dot-products, DAXPY, Mat-vec, and Poi-gen

```

!C
!C--- 1D array
nn = ICELTOT
allocate (indexL(0:nn), indexU(0:nn))
indexL= 0
indexU= 0

do icel= 1, ICELTOT
  indexL(icel)= INL(icel)
  indexU(icel)= INU(icel)
enddo

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

NPL= indexL(ICELTOT)
NPU= indexU(ICELTOT)

allocate (itemL(NPL), AL(NPL))
allocate (itemU(NPU), AU(NPU))

itemL= 0
itemU= 0
  AL= 0. d0
  AU= 0. d0
!C===

```

```

do i= 1, N

  VAL= D(i)*p(i)

  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL + AL(k)*p(itemL(k))
  enddo

  do k= indexU(i-1)+1, indexU(i)
    VAL= VAL + AU(k)*p(itemU(k))
  enddo

  q(i)= VAL

enddo

```

poi_gen (5/9)

New numbering is applied
after this point

Name	Type	Content
D (N)	R	Diagonal components of the matrix (N= ICELTOT)
BFORCE (N)	R	RHS vector
PHI (N)	R	Unknown vector
indexL (0:N) , indexU (0:N)	I	# of L/U non-zero off-diag. comp. (CRS)
NPL, NPU	I	Total # of L/U non-zero off-diag. comp. (CRS)
itemL (NPL) , itemU (NPU)	I	Column ID of L/U non-zero off-diag. comp. (CRS)
AL (NPL) , AU (NPU)	R	L/U non-zero off-diag. comp. (CRS)

```

!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C===

```

```

!$omp parallel do private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6) &
!$omp& private (VOL0, coef, j, ii, jj, kk)

```

```

do ip = 1, PEsmptOT
do icel= SMPindexG(ip-1)+1, SMPindexG(ip)
  ic0 = NEWtoOLD(icel)

```

```

icN1= NEIBcell (ic0, 1)
icN2= NEIBcell (ic0, 2)
icN3= NEIBcell (ic0, 3)
icN4= NEIBcell (ic0, 4)
icN5= NEIBcell (ic0, 5)
icN6= NEIBcell (ic0, 6)

```

icel: New ID
ic0: Old ID

```

VOL0= VOLCEL (ic0)

```

```

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

```

```

  if (icN5.lt.icel) then
    do j= 1, INL(icel)
      if (IAL(j, icel).eq. icN5) then
        itemL(j+indexL(icel-1))= icN5
        AL(j+indexL(icel-1))= coef
      exit
    endif
  enddo

```

```

  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq. icN5) then
        itemU(j+indexU(icel-1))= icN5
        AU(j+indexU(icel-1))= coef
      exit
    endif
  enddo
endif
endif
endif

```

poi_gen (6/9)

New numbering is applied

$$\frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z$$

Coef. Matrix: Parallel, “SMPindexG” “private”

```
!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C===

!$omp parallel do private (ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6) &
!$omp&                private (VOL0, coef, j, ii, jj, kk)

  do ip = 1, PEsmptOT
  do icel= SMPindexG(ip-1)+1, SMPindexG(ip)
    ic0 = NEWtoOLD(icel)

    icN1= NEIBcell (ic0, 1)
    icN2= NEIBcell (ic0, 2)
    icN3= NEIBcell (ic0, 3)
    icN4= NEIBcell (ic0, 4)
    icN5= NEIBcell (ic0, 5)
    icN6= NEIBcell (ic0, 6)

    VOL0= VOLCEL  (ic0)
  ...
```

```

!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C===

```

```

!$omp parallel do private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6) &
!$omp& private (VOL0, coef, j, ii, jj, kk)

```

```

do ip = 1, PEsmptOT
do icel= SMPindexG(ip-1)+1, SMPindexG(ip)
  ic0 = NEWtoOLD(icel)

```

```

icN1= NEIBcell (ic0, 1)
icN2= NEIBcell (ic0, 2)
icN3= NEIBcell (ic0, 3)
icN4= NEIBcell (ic0, 4)
icN5= NEIBcell (ic0, 5)
icN6= NEIBcell (ic0, 6)

```

icel: New ID
ic0: Old ID

```

VOL0= VOLCEL (ic0)

```

```

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

```

```

  if (icN5.lt.icel) then
    do j= 1, INL(icel)
      if (IAL(j, icel).eq. icN5) then
        itemL(j+indexL(icel-1))= icN5
        AL(j+indexL(icel-1))= coef
      exit
    endif
  enddo

```

```

  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq. icN5) then
        itemU(j+indexU(icel-1))= icN5
        AU(j+indexU(icel-1))= coef
      exit
    endif
  enddo
endif
endif
endif

```

poi_gen (6/9)

New numbering is applied

$$\begin{aligned}
 & \frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
 & \frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
 & \frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
 & \frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
 & \frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y + \\
 & \frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z
 \end{aligned}$$

```

!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C===

!$omp parallel do private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6) &
!$omp& private (VOL0, coef, j, ii, jj, kk)

```

```

do ip = 1, PEsmptOT
do icel= SMPindexG(ip-1)+1, SMPindexG(ip)
  ic0 = NEWtoOLD(icel)

```

```

  icN1= NEIBcell (ic0, 1)
  icN2= NEIBcell (ic0, 2)
  icN3= NEIBcell (ic0, 3)
  icN4= NEIBcell (ic0, 4)
  icN5= NEIBcell (ic0, 5)
  icN6= NEIBcell (ic0, 6)

```

```

VOL0= VOLCEL (ic0)

```

```

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

```

```

    if (icN5.lt.icel) then
      do j= 1, INL(icel)
        if (IAL(j, icel).eq. icN5) then
          itemL(j+indexL(icel-1))= icN5
          AL(j+indexL(icel-1))= coef
          exit
        endif
      enddo
    else

```

```

      do j= 1, INU(icel)
        if (IAU(j, icel).eq. icN5) then
          itemU(j+indexU(icel-1))= icN5
          AU(j+indexU(icel-1))= coef
          exit
        endif
      enddo
    endif
  enddo
endif
endif

```

poi_gen (6/9)

New numbering is applied

$$\frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z$$


```

!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C===
!$omp parallel do private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6) &
!$omp& private (VOL0, coef, j, ii, jj, kk)

```

```

do ip = 1, PEsmptOT
do icel= SMPindexG(ip-1)+1, SMPindexG(ip)
ic0 = NEWtoOLD(icel)

```

```

icN1= NEIBcel|(ic0, 1)
icN2= NEIBcel|(ic0, 2)
icN3= NEIBcel|(ic0, 3)
icN4= NEIBcel|(ic0, 4)
icN5= NEIBcel|(ic0, 5)
icN6= NEIBcel|(ic0, 6)

```

```

VOL0= VOLCEL (ic0)

```

```

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

```

```

if (icN5.lt.icel) then
do j= 1, INL(icel)
if (IAL(j, icel).eq.icN5) then
itemL(j+indexL(icel-1))= icN5
AL(j+indexL(icel-1))= coef
exit
endif
enddo

```

```

else
do j= 1, INU(icel)
if (IAU(j, icel).eq.icN5) then
itemU(j+indexU(icel-1))= icN5
AU(j+indexU(icel-1))= coef
exit
endif
enddo
endif
endif

```

$$RDZ = \frac{1}{\Delta z}$$

$$ZAREA = \Delta x \Delta y$$

**icN5 < icel
Lower Part**

poi_gen (6/9)

New numbering is applied

$$\frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z$$

```

!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C===
!$omp parallel do private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6) &
!$omp&          private (VOL0, coef, j, ii, jj, kk)

```

```

do ip = 1, PEsmptOT
do icel= SMPindexG(ip-1)+1, SMPindexG(ip)
  ic0 = NEWtoOLD(icel)

```

```

  icN1= NEIBcel|(ic0, 1)
  icN2= NEIBcel|(ic0, 2)
  icN3= NEIBcel|(ic0, 3)
  icN4= NEIBcel|(ic0, 4)
  icN5= NEIBcel|(ic0, 5)
  icN6= NEIBcel|(ic0, 6)

```

```

  VOL0= VOLCEL (ic0)

```

```

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

```

```

  if (icN5.lt.icel) then
    do j= 1, INL(icel)
      if (IAL(j, icel).eq. icN5) then
        itemL(j+indexL(icel-1))= icN5
        AL(j+indexL(icel-1))= coef
        exit
      endif
    enddo

```

```

  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq. icN5) then
        itemU(j+indexU(icel-1))= icN5
        AU(j+indexU(icel-1))= coef
        exit
      endif
    enddo
  endif
endif
endif

```

$$RDZ = \frac{1}{\Delta z}$$

$$ZAREA = \Delta x \Delta y$$

**icN5 > icel
Upper Part**

poi_gen (6/9)

New numbering is applied

$$\frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z$$

```

if (icN3.ne.0) then
  icN3= OLDtoNEW(icN3)
  coef= RDY * YAREA
  D(icel)= D(icel) - coef

  if (icN3.lt.icel) then
    do j= 1, INL(icel)
      if (IAL(j, icel).eq.icN3) then
        itemL(j+indexL(icel-1))= icN3
        AL(j+indexL(icel-1))= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq.icN3) then
        itemU(j+indexU(icel-1))= icN3
        AU(j+indexU(icel-1))= coef
        exit
      endif
    enddo
  endif
endif

if (icN1.ne.0) then
  icN1= OLDtoNEW(icN1)
  coef= RDX * XAREA
  D(icel)= D(icel) - coef

  if (icN1.lt.icel) then
    do j= 1, INL(icel)
      if (IAL(j, icel).eq.icN1) then
        itemL(j+indexL(icel-1))= icN1
        AL(j+indexL(icel-1))= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq.icN1) then
        itemU(j+indexU(icel-1))= icN1
        AU(j+indexU(icel-1))= coef
        exit
      endif
    enddo
  endif
endif

```

poi_gen (7/9)

$$\begin{aligned}
& \frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
& \frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
& \frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
& \frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
& \frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y + \\
& \frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z
\end{aligned}$$

```

if (icN2.ne.0) then
  icN2= OLDtoNEW(icN2)
  coef= RDX * XAREA
  D(icel)= D(icel) - coef

  if (icN2.lt.icel) then
    do j= 1, INL(icel)
      if (IAL(j, icel).eq.icN2) then
        itemL(j+indexL(icel-1))= icN2
        AL(j+indexL(icel-1))= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq.icN2) then
        itemU(j+indexU(icel-1))= icN2
        AU(j+indexU(icel-1))= coef
        exit
      endif
    enddo
  endif
endif

if (icN4.ne.0) then
  icN4= OLDtoNEW(icN4)
  coef= RDY * YAREA
  D(icel)= D(icel) - coef

  if (icN4.lt.icel) then
    do j= 1, INL(icel)
      if (IAL(j, icel).eq.icN4) then
        itemL(j+indexL(icel-1))= icN4
        AL(j+indexL(icel-1))= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq.icN4) then
        itemU(j+indexU(icel-1))= icN4
        AU(j+indexU(icel-1))= coef
        exit
      endif
    enddo
  endif
endif
endif

```

poi_gen (8/9)

$$\begin{aligned}
& \frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
& \frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
& \frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
& \frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
& \frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y + \\
& \frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z
\end{aligned}$$

poi_gen (9/9)

```
!$omp parallel do private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6) &
!$omp&          private (VOL0, coef, j, ii, jj, kk)
```

```
...
```

```
if (icN6.ne.0) then
  icN6= OLDtoNEW(icN6)
  coef= RDZ * ZAREA
  D(icel)= D(icel) - coef
```

```
if (icN6.lt.icel) then
  do j= 1, INL(icel)
    if (IAL(j, icel).eq.icN6) then
      itemL(j+indexL(icel-1))= icN6
      AL(j+indexL(icel-1))= coef
    exit
  endif
enddo
```

```
else
  do j= 1, INU(icel)
    if (IAU(j, icel).eq.icN6) then
      itemU(j+indexU(icel-1))= icN6
      AU(j+indexU(icel-1))= coef
    exit
  endif
enddo
```

```
endif
endif
ii= XYZ(ic0, 1)
jj= XYZ(ic0, 2)
kk= XYZ(ic0, 3)
```

```
BFORCE(icel)= -dfloat(ii+jj+kk) * VOL0
```

```
enddo
enddo
```

```
!$omp end parallel do
!C===
```

BFORCE
using original
mesh ID

ii,jj,kk,VOL0:
private

$$\frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y = f_{icel} \Delta x \Delta y \Delta z$$

Main Program

```

program MAIN

use STRUCT
use PCG
use solver_ICCG_mc

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

PHI= 0. d0

call solve_ICCG_mc
&      ( ICELTOT, NPL, NPU, indexL, itemL, indexU, itemU, D,
&      BFORCE, PHI, AL, AU, NCOLORTot, PEsmptTOT,
&      SMPindex, SMPindexG, EPSICCG, ITR, IER)
&

```

solve_ICCG_mc (1/6)

```

!C***
!C*** module solver_ICCG_mc
!C***
!
      module solver_ICCG_mc
      contains
!C
!C*** solve_ICCG
!C
subroutine solve_ICCG_mc
      &      (N, NPL, NPU, indexL, itemL, indexU, itemU, D, B, X,      &
      &      AL, AU, NCOLORTot, PEsmptTOT, SMPindex, SMPindexG,      &
      &      EPS, ITR, IER)

      implicit REAL*8 (A-H, O-Z)

      integer :: N, NL, NU, NCOLORTot, PEsmptTOT

      real(kind=8), dimension(N) :: D
      real(kind=8), dimension(N) :: B
      real(kind=8), dimension(N) :: X

      real(kind=8), dimension(NPL) :: AL
      real(kind=8), dimension(NPU) :: AU

      integer, dimension(0:N) :: indexL, indexU
      integer, dimension(NPL) :: itemL
      integer, dimension(NPU) :: itemU

      integer, dimension(0:NCOLORTot*PEsmptTOT) :: SMPindex
      integer, dimension(0:PEsmptTOT) :: SMPindexG

      real(kind=8), dimension(:, :), allocatable :: W

      integer, parameter :: R= 1
      integer, parameter :: Z= 2
      integer, parameter :: Q= 2
      integer, parameter :: P= 3
      integer, parameter :: DD= 4

```

solve_ICCG_mc (2/6)

```

!C
!C +-----+
!C |  INIT  |
!C +-----+
!C===
      allocate (W(N+128,4))

!$omp parallel do private(ip,i)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        X(i) = 0. d0
        W(i,2)= 0. 0D0
        W(i,3)= 0. 0D0
        W(i,4)= 0. 0D0
      enddo
      enddo
!$omp end parallel do

      do ic= 1, NCOLORTot
!$omp parallel do private(ip,ip1,i,VAL,k)
      do ip= 1, PEsmptOT
        ip1= (ic-1)*PEsmptOT + ip
      do i= SMPindex(ip1-1)+1, SMPindex(ip1)
        VAL= D(i)
        do k= indexL(i-1)+1, indexL(i)
          VAL= VAL - (AL(k)**2) *
W(itemL(k),DD)
        enddo
        W(i,DD)= 1. d0/VAL
      enddo
      enddo
!$omp end parallel do
      enddo

```

**Incomplete
“Modified” Cholesky
Factorization**

Incomplete “Modified” Cholesky Factorization

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} a_{ik}^2 \cdot d_k \right)^{-1} = l_{ii}^{-1}$$

$W[DD][i]:$	d_i
$D[i]:$	a_{ii}
$itemL[j]:$	k
$AL[j]:$	a_{ik}

```

do i= 1, N
  VAL= D(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL - (AL(k)**2) * W(itemL(k), DD)
  enddo
  W(i, DD)= 1. d0/VAL
enddo

```

Incomplete “Modified” Cholesky Factorization: Parallel Version

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} a_{ik}^2 \cdot d_k \right)^{-1} = l_{ii}^{-1}$$

$W[DD][i]:$	d_i
$D[i]:$	a_{ii}
$itemL[j]:$	k
$AL[j]:$	a_{ik}

```

do ic= 1, NCOLORTot
!$omp parallel do private(ip, ip1, i, VAL, k)
  do ip= 1, PEsmptOT
    ip1= (ic-1)*PEsmptOT + ip
    do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      VAL= D(i)
      do k= indexL(i-1)+1, indexL(i)
        VAL= VAL - (AL(k)**2) * W(itemL(k), DD)
      enddo
      W(i, DD)= 1. d0/VAL
    enddo
  enddo
!$omp end parallel do
enddo

```

solve_ICCG_mc (3/6)

```

!C +-----+
!C | {r0}= {b} - [A]{xini} |
!C +-----+
!C===
!$omp parallel do private(ip, i, VAL, k)
    do ip= 1, PEsmptTOT
    do i = SMPindexG(ip-1)+1,
        & SMPindexG(ip)
        VAL= D(i)*X(i)
        do k= indexL(i-1)+1, indexL(i)
            VAL= VAL + AL(k)*X(itemL(k))
        enddo
        do k= indexU(i-1)+1, indexU(i)
            VAL= VAL + AU(k)*X(itemU(k))
        enddo
        W(i, R)= B(i) - VAL
    enddo
    enddo
!$omp end parallel do

    BNRM2= 0.0D0
!$omp parallel do private(ip, i)
reduction(+:BNRM2)
    do ip= 1, PEsmptTOT
    do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        BNRM2 = BNRM2 + B(i) **2
    enddo
    enddo
!$omp end parallel do
!C===

```

Compute $\mathbf{r}^{(0)} = \mathbf{b} - [\mathbf{A}]\mathbf{x}^{(0)}$

```

for i= 1, 2, ...
    solve [M] z(i-1) = r(i-1)
    ρi-1 = r(i-1) z(i-1)
    if i=1
        p(1) = z(0)
    else
        βi-1 = ρi-1 / ρi-2
        p(i) = z(i-1) + βi-1 p(i-1)
    endif
    q(i) = [A] p(i)
    αi = ρi-1 / p(i) q(i)
    x(i) = x(i-1) + αi p(i)
    r(i) = r(i-1) - αi q(i)
    check convergence |r|
end

```

Mat-Vec

NO Data Dependency: SMPindexG

```

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

```

solve_ICCG_mc (3/6)

```

!C +-----+
!C | {r0}= {b} - [A]{xini} |
!C +-----+
!C===
!$omp parallel do private(ip, i, VAL, k)
  do ip= 1, PEsmptOT
  do i = SMPindexG(ip-1)+1, SMPindexG(ip)
    VAL= D(i)*X(i)
    do k= indexL(i-1)+1, indexL(i)
      VAL= VAL + AL(k)*X(itemL(k))
    enddo
    do k= indexU(i-1)+1, indexU(i)
      VAL= VAL + AU(k)*X(itemU(k))
    enddo
    W(i, R)= B(i) - VAL
  enddo
enddo
!$omp end parallel do

BNRM2= 0.0D0
!$omp parallel do private(ip, i) reduction(+:BNRM2)
  do ip= 1, PEsmptOT
  do i = SMPindexG(ip-1)+1, SMPindexG(ip)
    BNRM2 = BNRM2 + B(i) **2
  enddo
enddo
!$omp end parallel do
!C===

```

Compute $\mathbf{r}^{(0)} = \mathbf{b} - [\mathbf{A}]\mathbf{x}^{(0)}$

```

for i= 1, 2, ...
  solve [M] z(i-1) = r(i-1)
  ρi-1 = r(i-1) z(i-1)
  if i=1
    p(1) = z(0)
  else
    βi-1 = ρi-1 / ρi-2
    p(i) = z(i-1) + βi-1 p(i-1)
  endif
  q(i) = [A]p(i)
  αi = ρi-1 / p(i) q(i)
  x(i) = x(i-1) + αip(i)
  r(i) = r(i-1) - αiq(i)
  check convergence |r|
end

```

Dot Products: SMPindexG, reduction

```
BNRM2= 0.0D0
!$omp parallel do private(ip, i) reduction(+:BNRM2)
  do ip= 1, PEsmptOT
    do i = SMPindexG(ip-1)+1, SMPindexG(ip)
      BNRM2 = BNRM2 + B(i) **2
    enddo
  enddo
!$omp end parallel do
```

solve_ICCG_mc (4/6)

```

      ITR= N
      do L= 1, ITR
!C
!C +-----+
!C | {z}= [Minv] {r} |
!C +-----+
!C===
!$omp parallel do private(ip,i)
      do ip= 1, PEsmptTOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        W(i,Z)= W(i,R)
      enddo
    enddo
!$omp end parallel do

      do ic= 1, NCOLORTot
!$omp parallel do private(ip,ip1,i,WVAL,j)
      do ip= 1, PEsmptTOT
        ip1= (ic-1)*PEsmptTOT + ip
        do i= SMPindex(ip1-1)+1, SMPindex(ip1)
          WVAL= W(i,Z)
          do j= 1, INL(i)
            WVAL= WVAL - AL(j,i) * W(IAL(j,i),Z)
          enddo
          W(i,Z)= WVAL * W(i,DD)
        enddo
      enddo
!$omp end parallel do
    enddo

      do ic= NCOLORTot, 1, -1
!$omp parallel do private(ip,ip1,i,SW,j)
      do ip= 1, PEsmptTOT
        ip1= (ic-1)*PEsmptTOT + ip
        do i= SMPindex(ip1-1)+1, SMPindex(ip1)
          SW = 0.0d0
          do j= 1, INU(i)
            SW= SW + AU(j,i) * W(IAU(j,i),Z)
          enddo
          W(i,Z)= W(i,Z) - W(i,DD) * SW
        enddo
      enddo
!$omp end parallel do
    enddo
!C===

```

Compute $r^{(0)} = b - [A]x^{(0)}$

for $i = 1, 2, \dots$

solve $[M]z^{(i-1)} = r^{(i-1)}$

$\rho_{i-1} = r^{(i-1)} z^{(i-1)}$

if $i=1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

endif

$q^{(i)} = [A]p^{(i)}$

$\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check convergence $|r|$

end

solve_ICCG_mc (4/6)

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i= 1, 2, ...
    solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$ 
    if i=1
         $p^{(1)} = z^{(0)}$ 
    else
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    endif
     $q^{(i)} = [A]p^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    check convergence  $|r|$ 
end

```

```

ITR= N
do L= 1, ITR
!C
!C +-----+
!C | {z}= [Minv] {r} |
!C +-----+
!C==
!$omp parallel do private(ip,i)
do ip= 1, PEsmptOT
do i= SMPindexG(ip-1)+1, SMPindexG(ip)
W(i,Z)= W(i,R)
enddo
enddo
!$omp end parallel do

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

do ic= NCOLORTot, 1, -1
!$omp parallel do private(ip,ip1,i,SW,k)
do ip= 1, PEsmptOT
ip1= (ic-1)*PEsmptOT + ip
do i= SMPindex(ip1-1)+1, SMPindex(ip1)
SW = 0.0d0
do k= indexU(i-1)+1, indexU(i)
SW= SW + AU(k) * W(itemU(k),Z)
enddo
W(i,Z)= W(i,Z) - W(i,DD) * SW
enddo
enddo
!$omp end parallel do
enddo
!C==

```

SMPindex

solve_ICCG_mc (4/6)

$$(M)\{z\} = (LDL^T)\{z\} = \{r\}$$

$$(L)\{z\} = \{r\}$$

Forward Substitution

$$(DL^T)\{z\} = \{z\}$$

Backward Substitution

```

      ITR= N
      do L= 1, ITR
!C
!C +-----+
!C | {z}= [Minv] {r} |
!C +-----+
!C==
!$omp parallel do private(ip,i)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        W(i,Z)= W(i,R)
      enddo
      enddo
!$omp end parallel do
      do ic= 1, NCOLORTot
!$omp parallel do private(ip,ip1,i,WVAL,k)
        do ip= 1, PEsmptOT
          ip1= (ic-1)*PEsmptOT + ip
          do i= SMPindex(ip1-1)+1, SMPindex(ip1)
            WVAL= W(i,Z)
            do k= indexL(i-1)+1, indexL(i)
              WVAL= WVAL - AL(k) * W(itemL(k),Z)
            enddo
            W(i,Z)= WVAL * W(i,DD)
          enddo
        enddo
      enddo
!$omp end parallel do
      enddo

      do ic= NCOLORTot, 1, -1
!$omp parallel do private(ip,ip1,i,SW,k)
        do ip= 1, PEsmptOT
          ip1= (ic-1)*PEsmptOT + ip
          do i= SMPindex(ip1-1)+1, SMPindex(ip1)
            SW = 0.0d0
            do k= indexU(i-1)+1, indexU(i)
              SW= SW + AU(k) * W(itemU(k),Z)
            enddo
            W(i,Z)= W(i,Z) - W(i,DD) * SW
          enddo
        enddo
      enddo
!$omp end parallel do
      enddo
!C==

```

SMPindex

Forward Substitution: SMPindex

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

solve_ICCG_mc

(5/6)

```

!C +-----+
!C | {p} = {z} if      ITER=1 |
!C | BETA= RHO / RH01 otherwise |
!C +-----+
!C===
      if ( L.eq.1 ) then
!$omp parallel do private(ip,i)
          do ip= 1, PEsmptOT
              do i = SMPindexG(ip-1)+1, SMPindexG(ip)
                  W(i,P)= W(i,Z)
              enddo
          enddo
!$omp end parallel do
      else
          BETA= RHO / RH01
!$omp parallel do private(ip,i)
          do ip= 1, PEsmptOT
              do i = SMPindexG(ip-1)+1, SMPindexG(ip)
                  W(i,P)= W(i,Z) + BETA*W(i,P)
              enddo
          enddo
!$omp end parallel do
      endif
!C===

!C +-----+
!C | {q}= [A] {p} |
!C +-----+
!C===
!$omp parallel do private(ip,i,VAL,k)
      do ip= 1, PEsmptOT
          do i = SMPindexG(ip-1)+1, SMPindexG(ip)
              VAL= D(i)*W(i,P)
              do k= indexL(i-1)+1, indexL(i)
                  VAL= VAL + AL(k)*W(itemL(k),P)
              enddo
              do k= indexU(i-1)+1, indexU(i)
                  VAL= VAL + AU(k)*W(itemU(k),P)
              enddo
              W(i,Q)= VAL
          enddo
      enddo
!$omp end parallel do
!C===

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i= 1, 2, ...
    solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    if i=1
         $p^{(1)} = z^{(0)}$ 
    else
         $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
         $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
    endif
     $q^{(i)} = [A]p^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    check convergence  $|r|$ 
end

```

```

!C +-----+
!C | {p} = {z} if ITER=1 |
!C | BETA= RHO / RH01 otherwise |
!C +-----+
!C===
      if ( L.eq.1 ) then
!$omp parallel do private(ip,i)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        W(i,P)= W(i,Z)
      enddo
      enddo
!$omp end parallel do
      else
        BETA= RHO / RH01
!$omp parallel do private(ip,i)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        W(i,P)= W(i,Z) + BETA*W(i,P)
      enddo
      enddo
!$omp end parallel do
      endif
!C===

!C +-----+
!C | {q}= [A] {p} |
!C +-----+
!C===
!$omp parallel do private(ip,i,VAL,k)
  do ip= 1, PEsmptOT
    do i = SMPindexG(ip-1)+1, SMPindexG(ip)
      VAL= D(i)*W(i,P)
      do k= indexL(i-1)+1, indexL(i)
        VAL= VAL + AL(k)*W(itemL(k),P)
      enddo
      do k= indexU(i-1)+1, indexU(i)
        VAL= VAL + AU(k)*W(itemU(k),P)
      enddo
      W(i,Q)= VAL
    enddo
  enddo
!$omp end parallel do
!C===

```

solve_ICCG_mc (5/6)

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i= 1, 2, ...
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  if i=1
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```

solve_ICCG_mc (6/6)

```

!C
!C +-----+
!C | ALPHA= RHO / {p} {q} |
!C +-----+
!C===
      C1= 0. d0
!$omp parallel do private(ip,i) reduction(+:C1)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        C1= C1 + W(i,P)*W(i,Q)
      enddo
      enddo
!$omp end parallel do

      ALPHA= RHO / C1
!C===

!C
!C +-----+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +-----+
!C===
!$omp parallel do private(ip,i)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        X(i) = X(i) + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo
      enddo
!$omp end parallel do

      DNRM2= 0. d0
!$omp parallel do private(ip,i) reduction(+:DNRM2)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        DNRM2= DNRM2 + W(i,R)**2
      enddo
      enddo
!$omp end parallel do
!C===

```

Compute $r^{(0)} = b - [A]x^{(0)}$
for $i = 1, 2, \dots$
 solve $[M]z^{(i-1)} = r^{(i-1)}$
 $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$
 if $i=1$
 $p^{(1)} = z^{(0)}$
 else
 $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
 $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
 endif
 $q^{(i)} = [A]p^{(i)}$
 $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$
 $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
 $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
 check convergence $|r|$
end

solve_ICCG_mc (6/6)

```

!C
!C +-----+
!C | ALPHA= RHO / {p} {q} |
!C +-----+
!C===
      C1= 0.d0
!$omp parallel do private(ip,i) reduction(+:C1)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        C1= C1 + W(i,P)*W(i,Q)
      enddo
      enddo
!$omp end parallel do

      ALPHA= RHO / C1
!C===

!C
!C +-----+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +-----+
!C===
!$omp parallel do private(ip,i)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        X(i) = X(i) + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo
      enddo
!$omp end parallel do

      DNRM2= 0.d0
!$omp parallel do private(ip,i) reduction(+:DNRM2)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        DNRM2= DNRM2 + W(i,R)**2
      enddo
      enddo
!$omp end parallel do
!C===

```

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i= 1, 2, ...
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  if i=1
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$ 
   $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$ 
   $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
   $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
  check convergence  $|r|$ 
end

```

solve_ICCG_mc (6/6)

```

!C
!C +-----+
!C | ALPHA= RHO / {p} {q} |
!C +-----+
!C===
      C1= 0.d0
!$omp parallel do private(ip,i) reduction(+:C1)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        C1= C1 + W(i,P)*W(i,Q)
      enddo
      enddo
!$omp end parallel do

      ALPHA= RHO / C1
!C===

!C
!C +-----+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +-----+
!C===
!$omp parallel do private(ip,i)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        X(i) = X(i) + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo
      enddo
!$omp end parallel do

      DNRM2= 0.d0
!$omp parallel do private(ip,i) reduction(+:DNRM2)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
        DNRM2= DNRM2 + W(i,R)**2
      enddo
      enddo
!$omp end parallel do
!C===

```

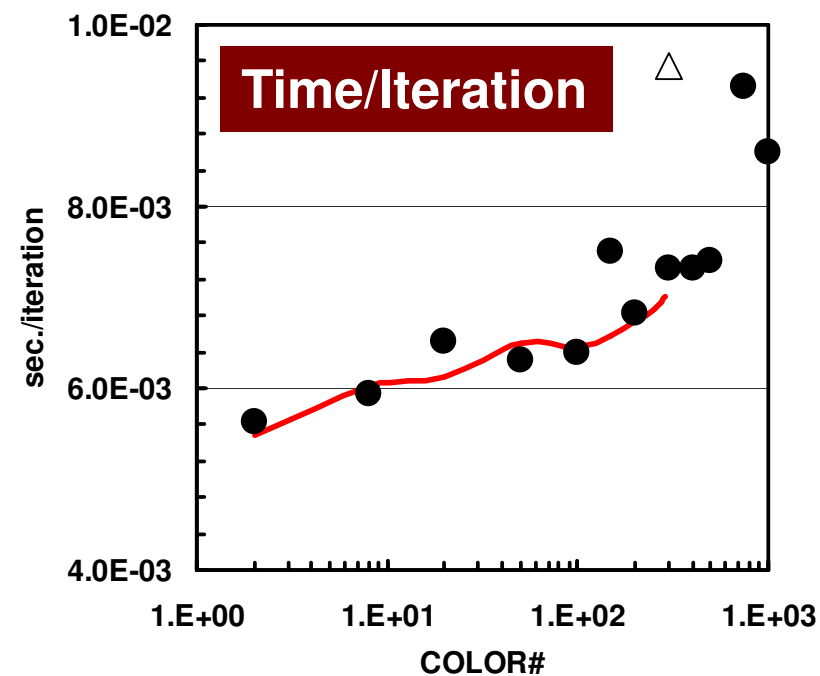
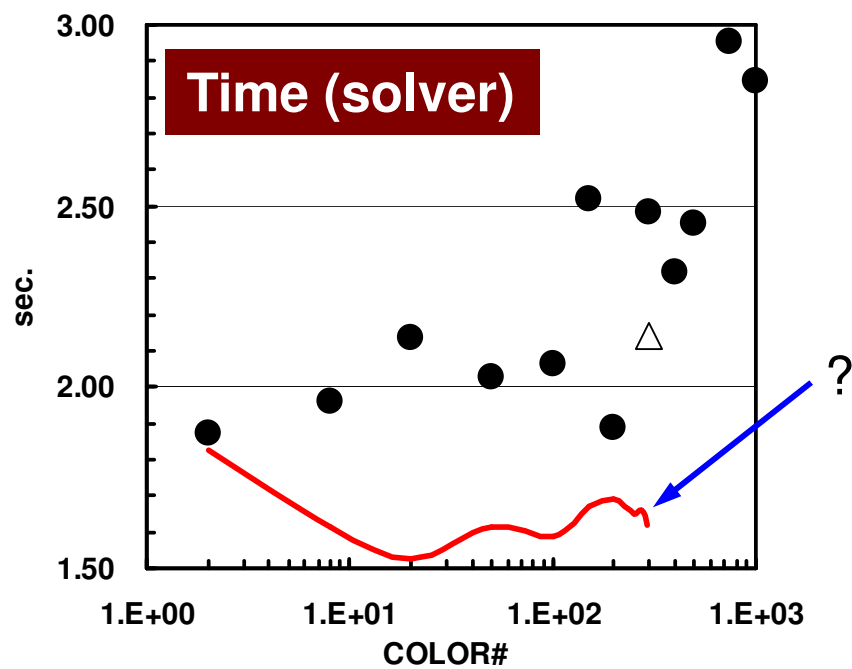
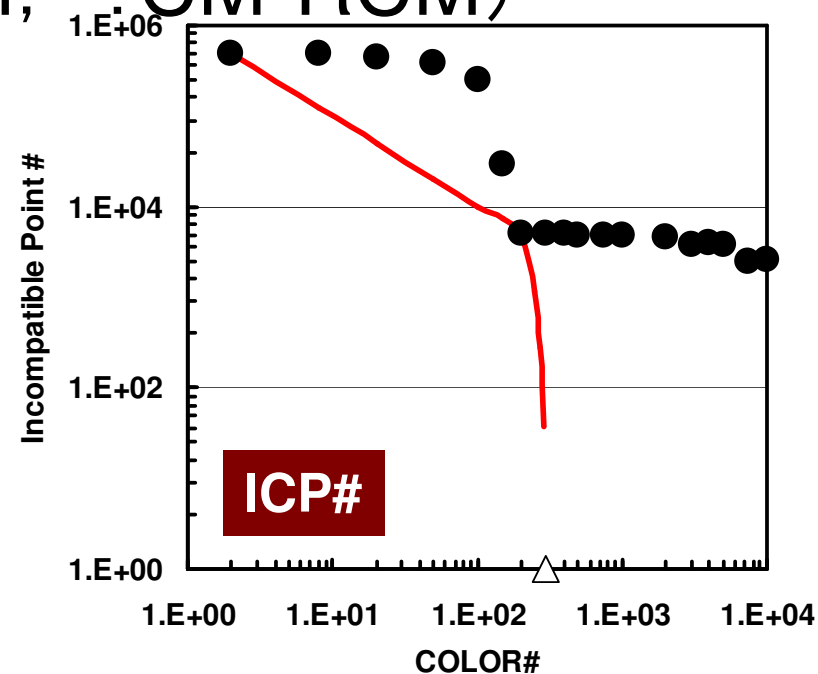
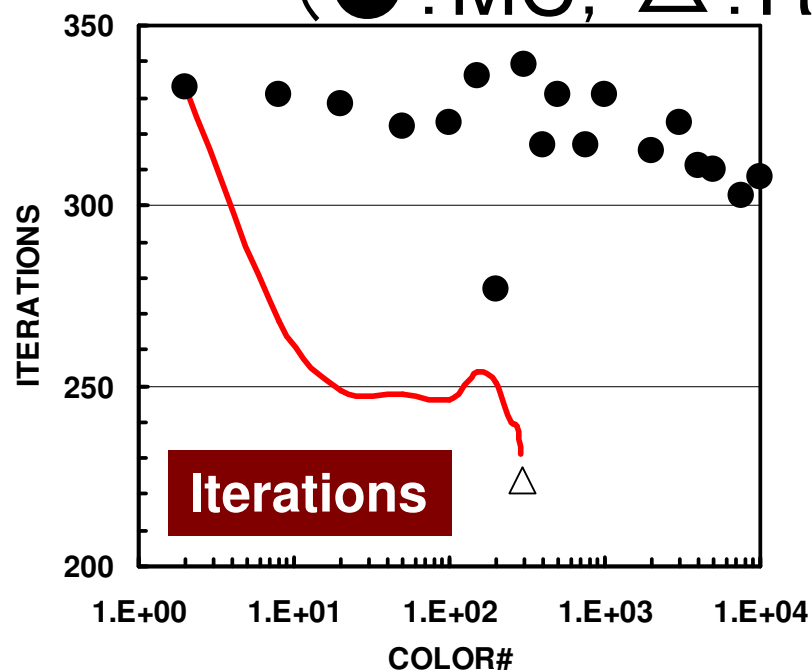
Compute $r^{(0)} = b - [A]x^{(0)}$
for $i = 1, 2, \dots$
 solve $[M]z^{(i-1)} = r^{(i-1)}$
 $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$
 if $i=1$
 $p^{(1)} = z^{(0)}$
 else
 $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
 $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
 endif
 $q^{(i)} = [A]p^{(i)}$
 $\alpha_i = \rho_{i-1} / p^{(i)} q^{(i)}$
 $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
 $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
 check convergence |r|
end

- Applying OpenMP to L2-sol
- **Examples**
- Optimization + Exercise

-
- The diagram illustrates a 2D mesh network topology for a 2x4 processor array. The array consists of eight processing elements (PEs) arranged in two rows and four columns. Each PE contains a Core, L1 cache, and L2 cache. The top row PEs have a Core and L1 cache, while the bottom row PEs have a Core and CPU. Each PE is connected to its four neighbors (up, down, left, right) via L2-L2 links. Additionally, there are L3-L3 links between adjacent PEs in each row. The network is connected to a Memory block at the top and bottom. Dashed lines indicate crossbar connections between the L2 caches of the top and bottom rows.

SR11000, 1-node/16-cores, 100^3

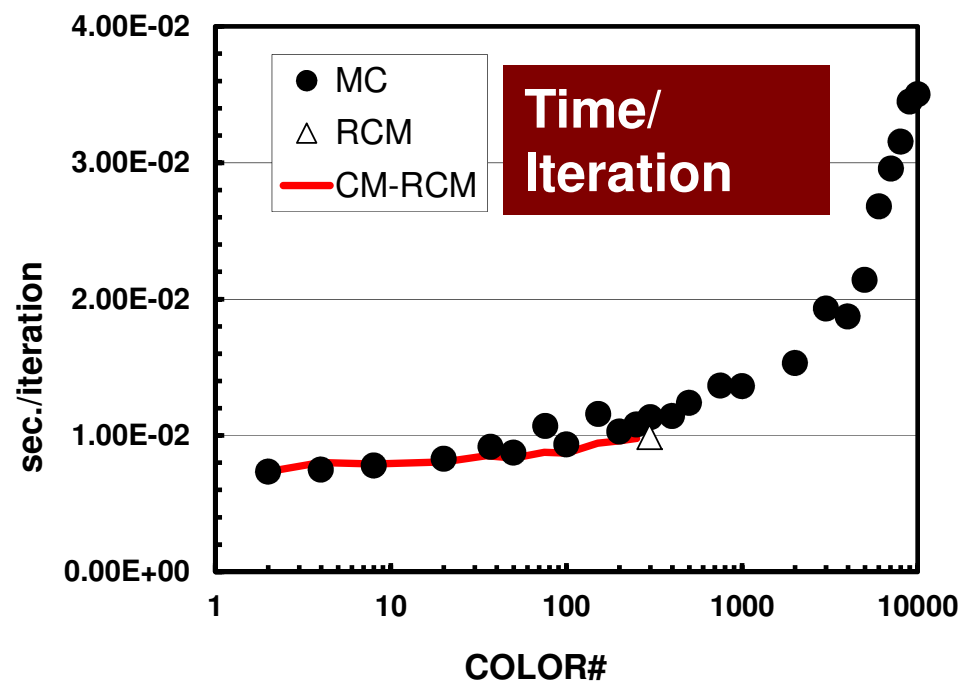
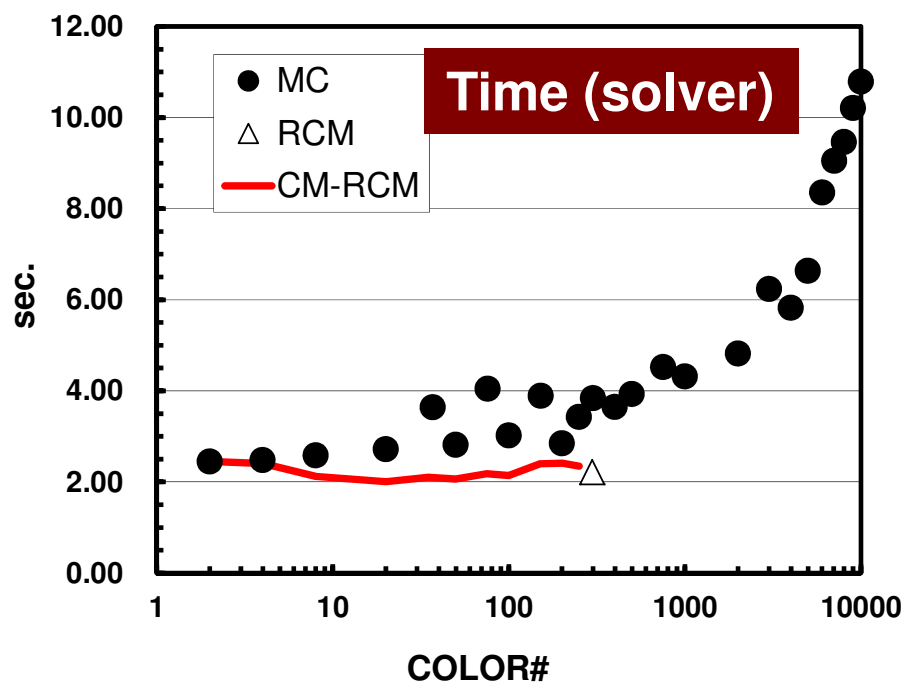
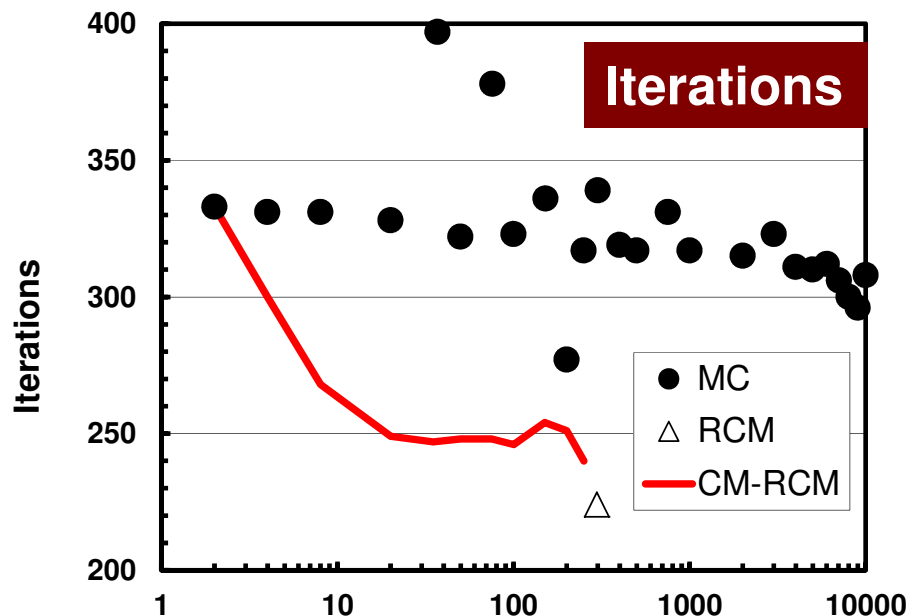
(●: MC, △: RCM, - : CM-RCM)



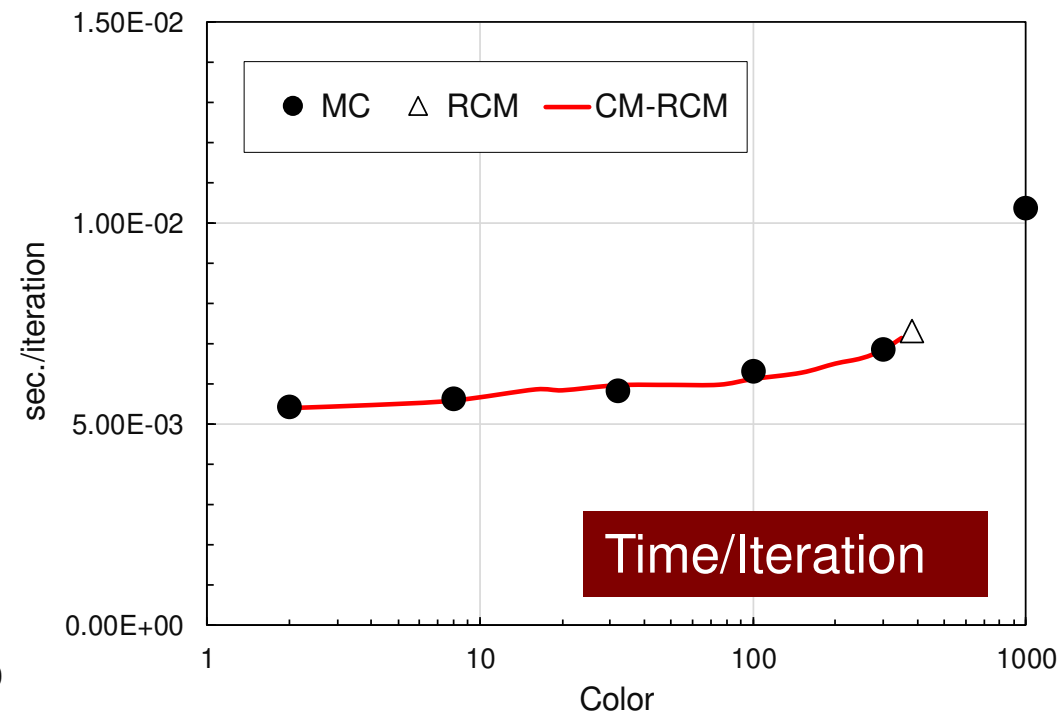
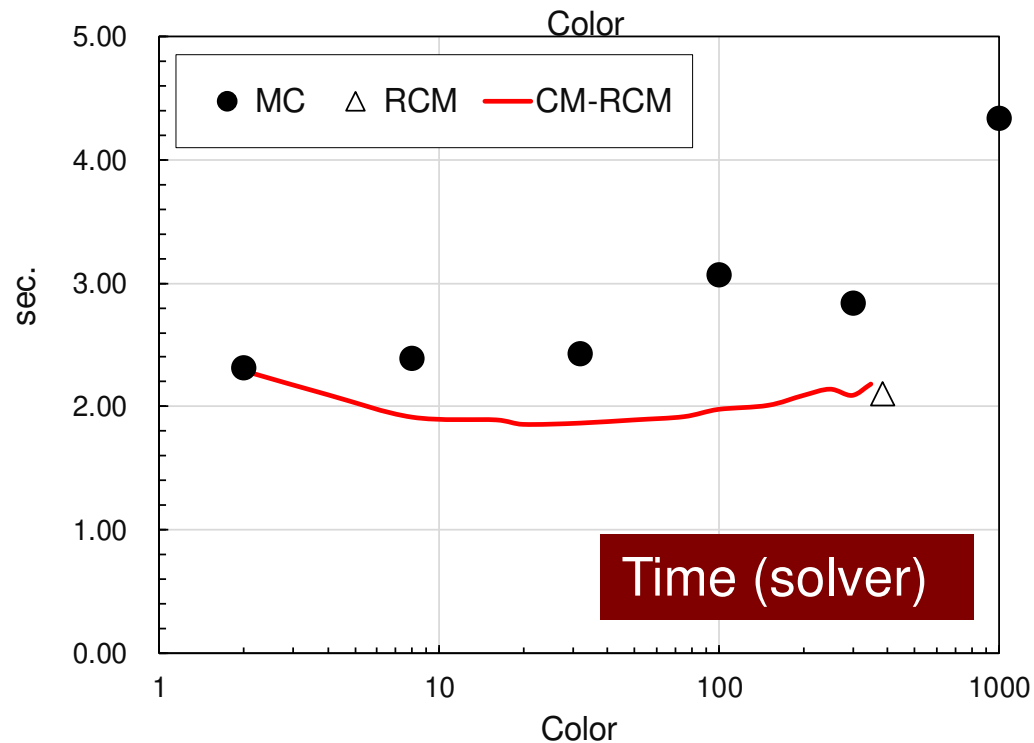
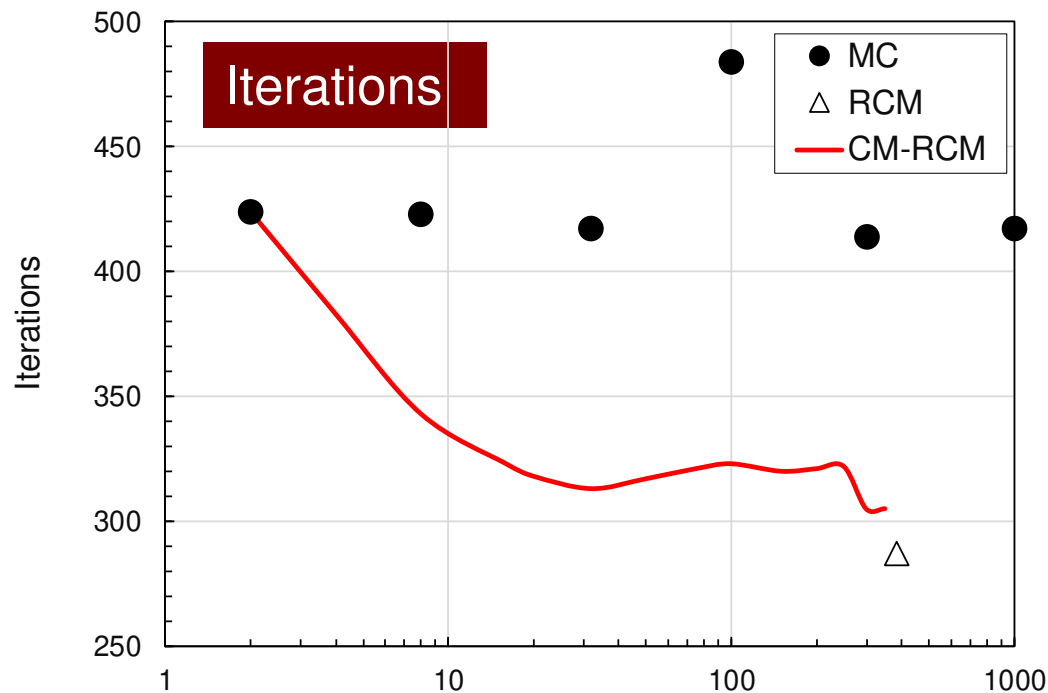
FX10, 1-node/16-cores, 100^3

(●: MC, △: RCM, - : CM-RCM)

- Fujitsu PRIMEHPC FX10
 - ✓ Oakleaf-FX, Oakbridge-CX
 - ✓ Commercial Version of K
- Apr. 2012-Mar. 2018

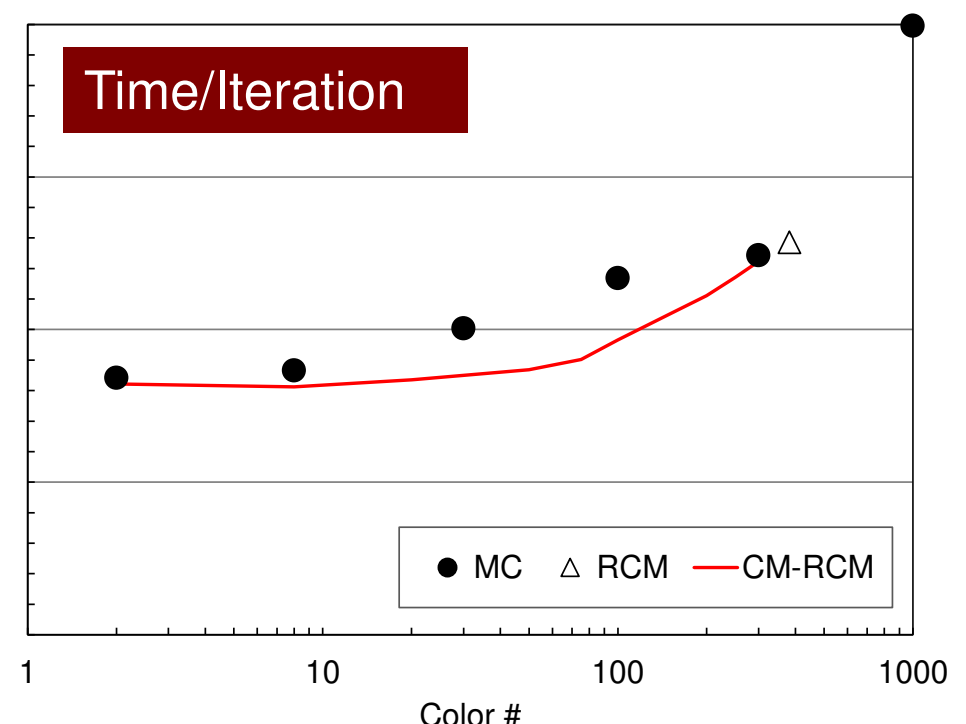
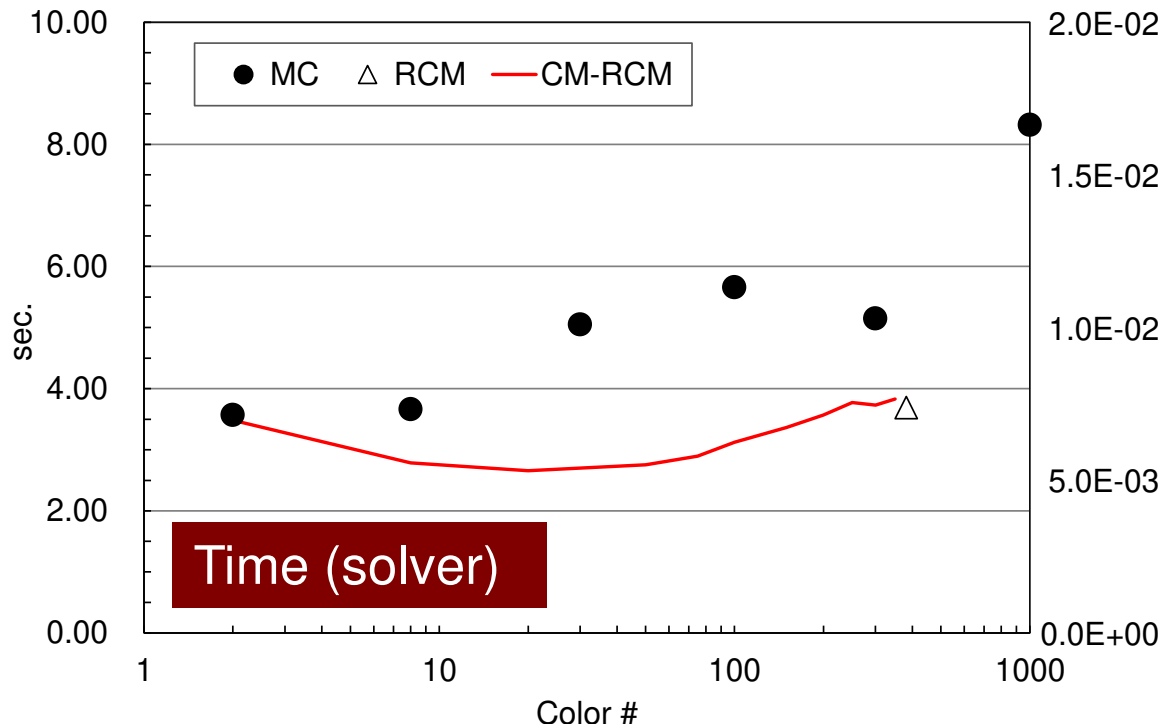
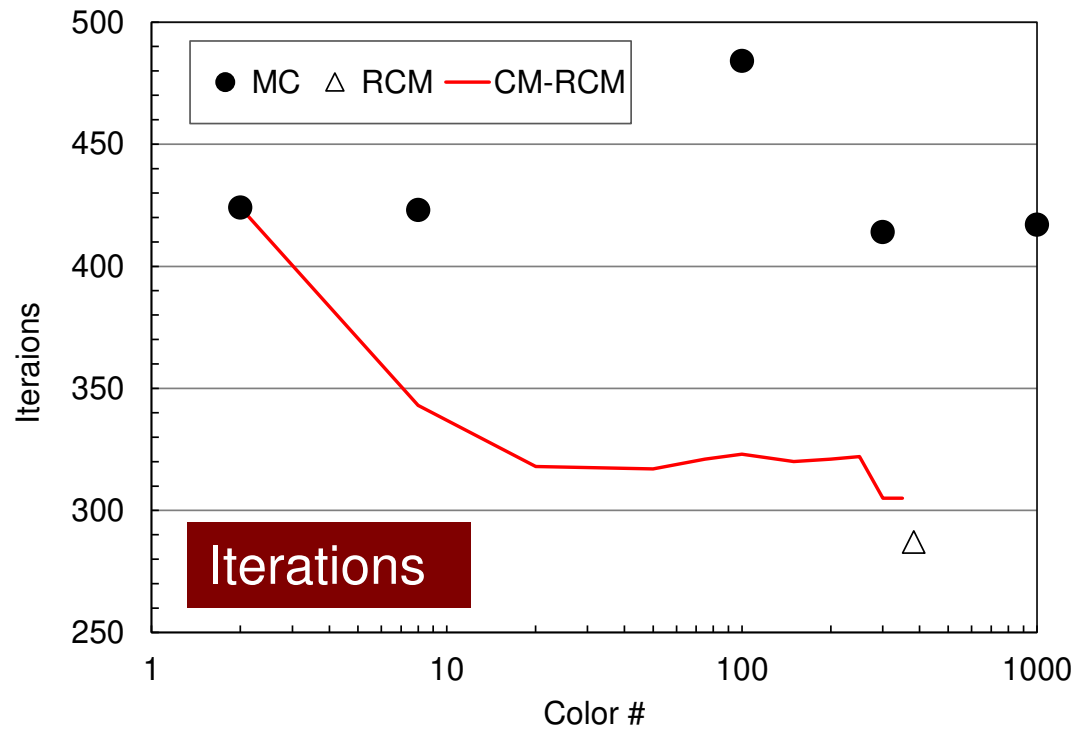


OBCX,
1-socket/24-cores,
 128^3
(● : MC, △ : RCM, - : CM-RCM)



Odyssey

1-CMG/12-cores,
128³
(● : MC, △ : RCM, - : CM-RCM)



- Applying OpenMP to L2-sol
- Examples
- **Optimization + Exercise**

- Running the Code
- Further Optimization

Compile & Run

```
>$ cd /work/gt80/t80XXX/ompf/src  
>$ module load fj
```

```
>$ make  
>$ ls ../run/L3-sol
```

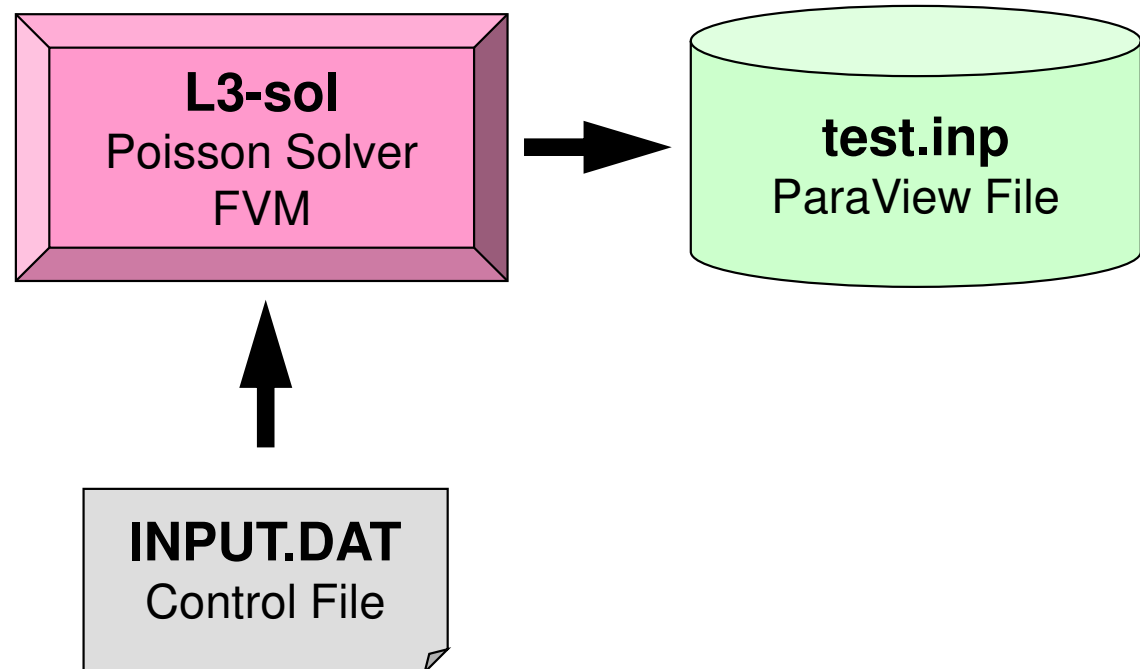
L3-sol

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

(modify INPUT.DAT, go1.sh)

```
>$ pjsub go1.sh
```


Running L3-sol



Control Data: INPUT.DAT

```
128 128 128
1.00e-00 1.00e-00 1.00e-00
1.0e-08
12
-10
```

```
NX/NY/NZ
DX/DY/DZ
EPSICCG
PEsmpTOT
NCOLOrTot
```

- **NX, NY, NZ**

- Number of meshes in X/Y/Z dir.

- **DX, DY, DZ**

- Size of meshes

- **EPSICCG**

- Convergence Criteria for ICCG

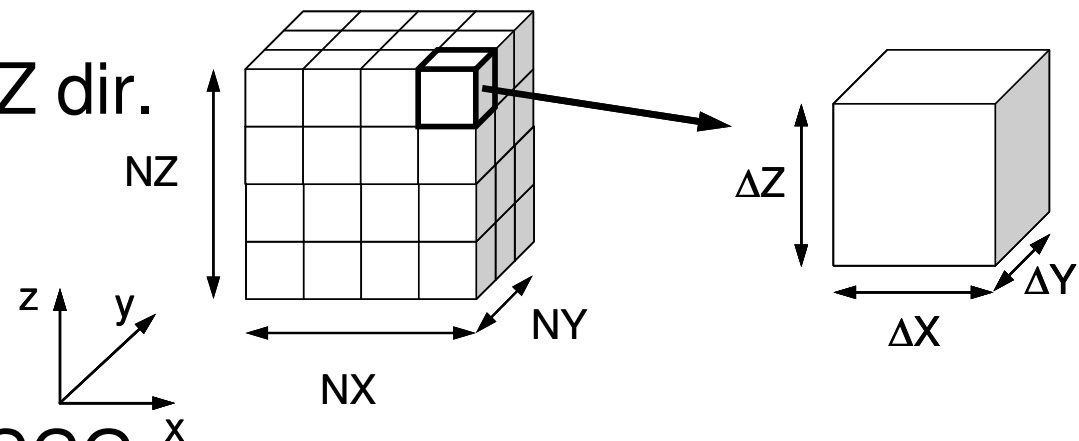
- **PEsmpTOT**

- Thread Number (**--omp thread=XX**)

- **NCOLOrTot**

- Reordering Method + Initial Number of Colors/Levels

- ≥ 2 : MC, =0: CM, =-1: RCM, $-2 \leq$: CMRCM



go1.sh

```
#!/bin/sh
#PJM -N "go1"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=12                (=PEsmpTOT)
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o test1.lst

module load fj
export OMP_NUM_THREADS=12           (=PEsmpTOT)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

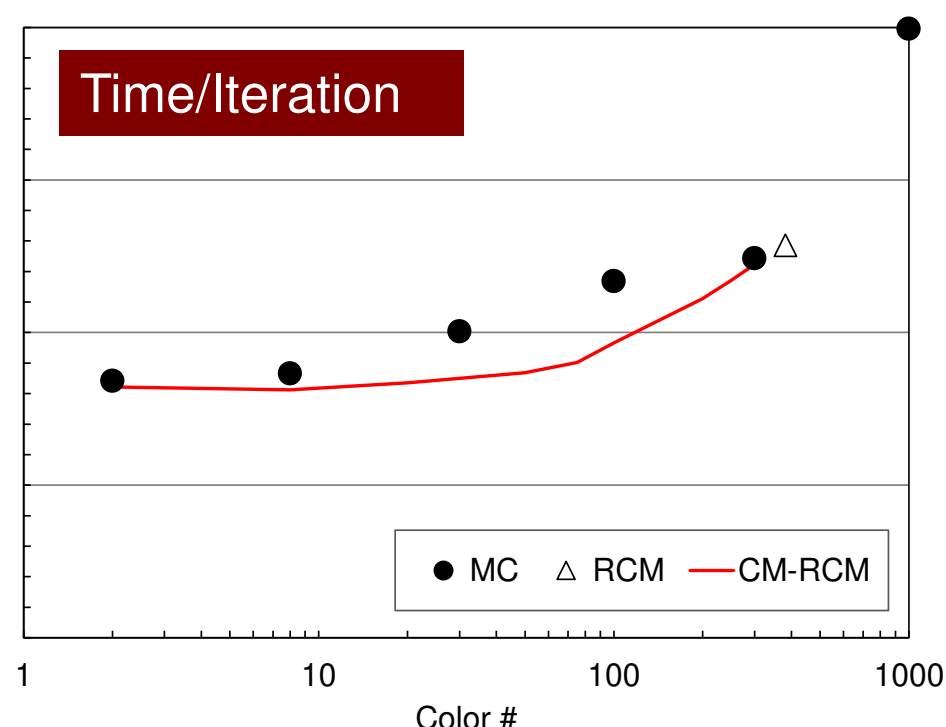
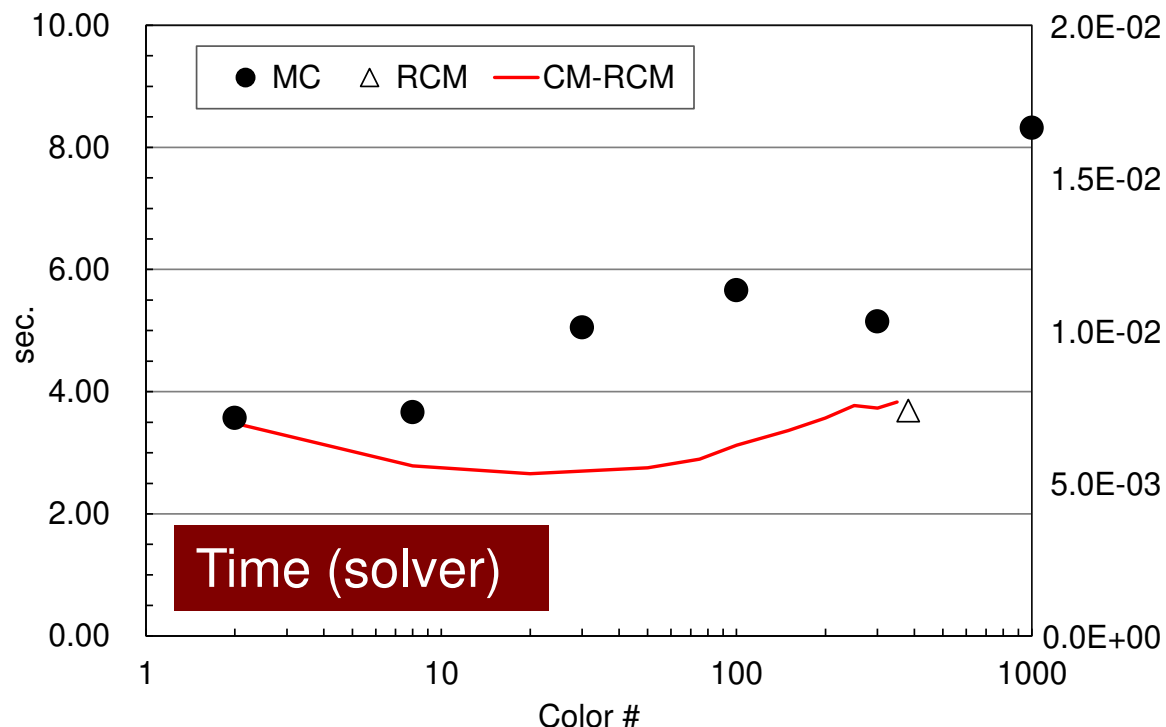
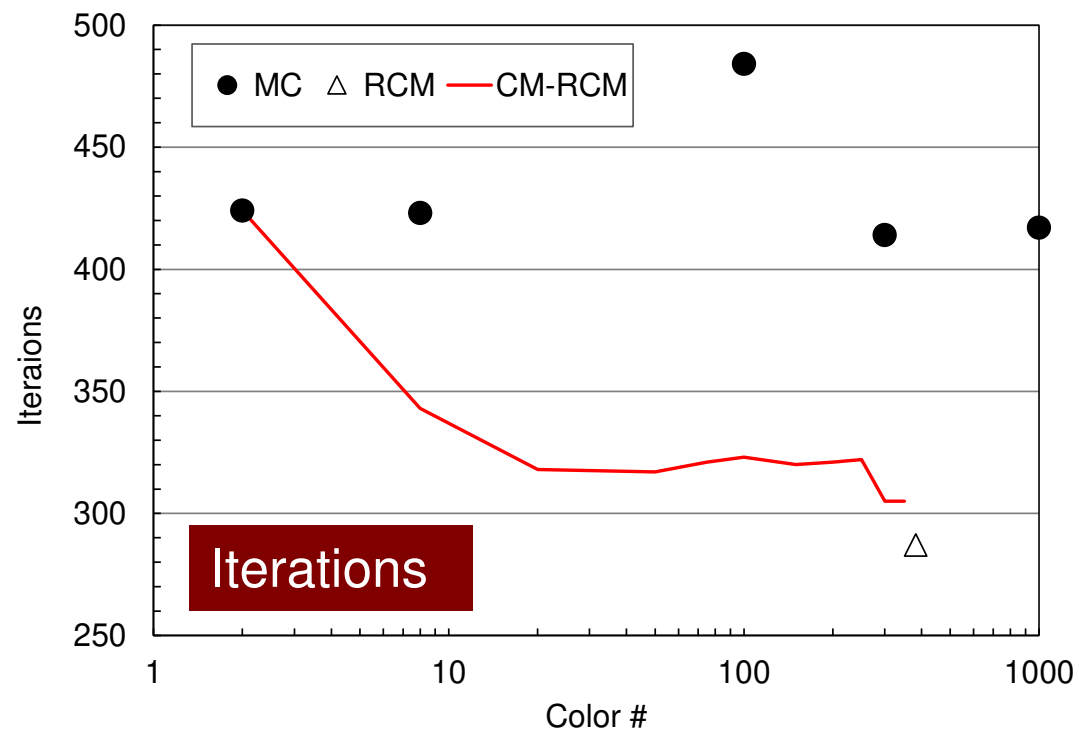
numactl -l ./L3-sol
numactl -C 12-23 -m 4 ./L3-sol
```

Odyssey

1-CMG/12-cores,

128^3

(● : MC, △ : RCM, - : CM-RCM)



- Running the Code
- **Further Optimization**
 - **OpenMP Statement**
 - Sequential Reordering

Forward Subst.: Current Impl. (F)

```

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

```

- At “**!omp parallel**”, generation and corruption of threads (up to 28) occurs: Fork-Join
 - In each color, this occurs
 - Some overhead
- Overhead increases, if number of color increases.

For. Subst.: Reduced Overhead (F)

```

!$omp parallel private(ic,ip,ip1,i,WVAL,k)
  do ic= 1, NCOLORTot
!$omp do
    do ip= 1, PEsmptTOT
      ip1= (ic-1)*PEsmptTOT + 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
  endd
!$omp end parallel

```

- Generation of threads occurs just once before starting forward substitutions.
- Loops with “!omp do” are parallelized.

Programs (src0)

```
% cd /work/gt80/t80XXX/ompf/src0
```

```
% module load fj
```

```
% make
```

```
% cd ../run
```

```
% ls L3-sol0
```

```
    L3-sol0
```

```
<modify "INPUT.DAT">
```

```
<modify "go0.sh">
```

```
% pjsub go0.sh
```


go0.sh

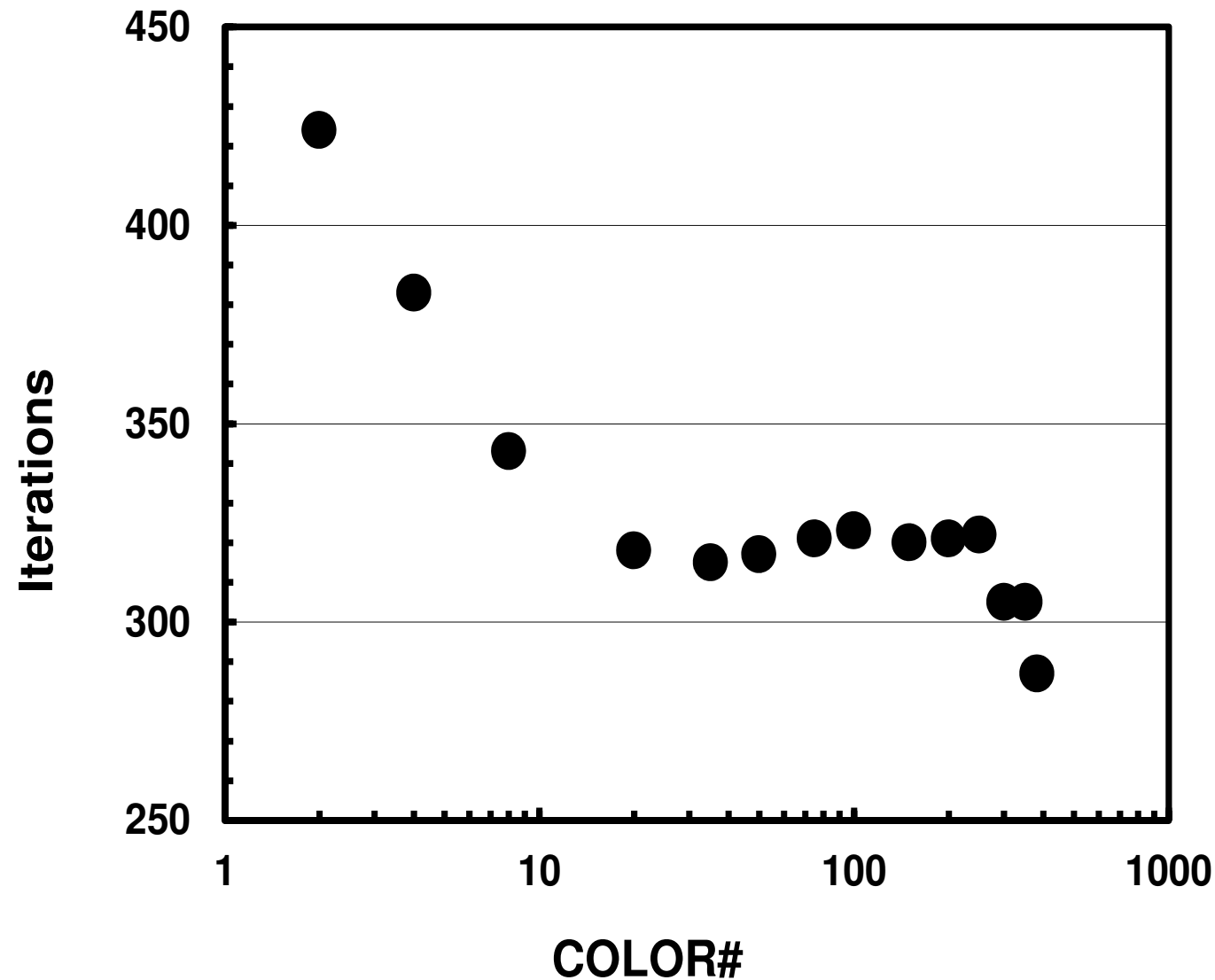
```
#!/bin/sh
#PJM -N "go0"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=12                (=PEsmpTOT)
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o test0.lst

module load fj
export OMP_NUM_THREADS=12           (=PEsmpTOT)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl -l ./L3-sol0
numactl -C 12-23 -m 4 ./L3-sol0
```

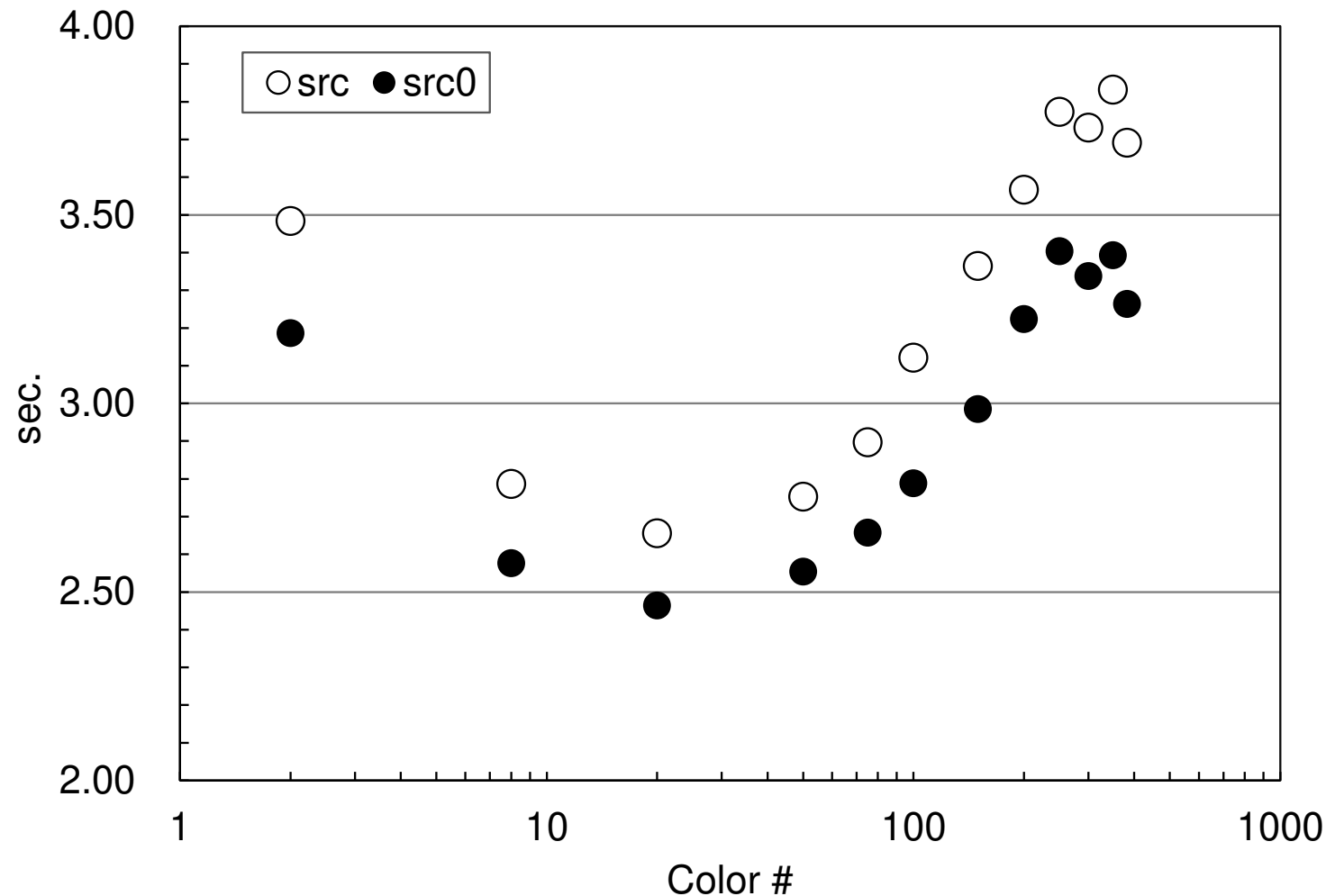
Color#~Iterations for CM-RCM

128^3 case



Time for ICCG Solver: CM-RCM

“src” becomes slower if color# is larger: overhead of fork-join, unstable for many colors (12 threads, C)



- Running the Code
- **Further Optimization**
 - OpenMP Statement
 - **Sequential Reordering**

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

```

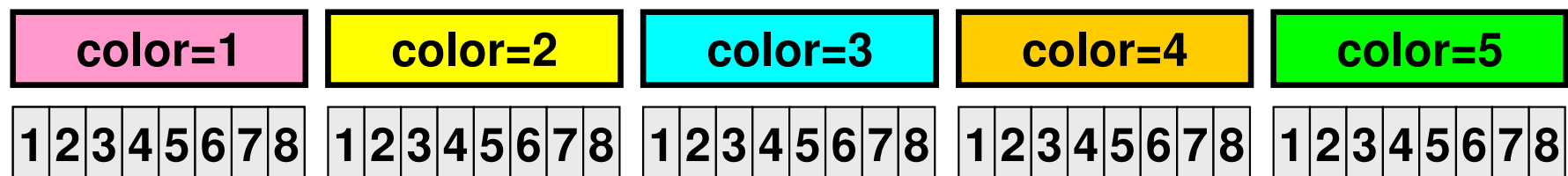
do ic= 1, NCOLORTot
!$omp parallel do ...
  do ip= 1, PEsmpTOT
    ip1= (ic-1)*PEsmpTOT+ip
    do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      (...)
    enddo
  enddo
!omp end parallel do
enddo

```

Initial Vector



Coloring
(5 colors)
+Ordering



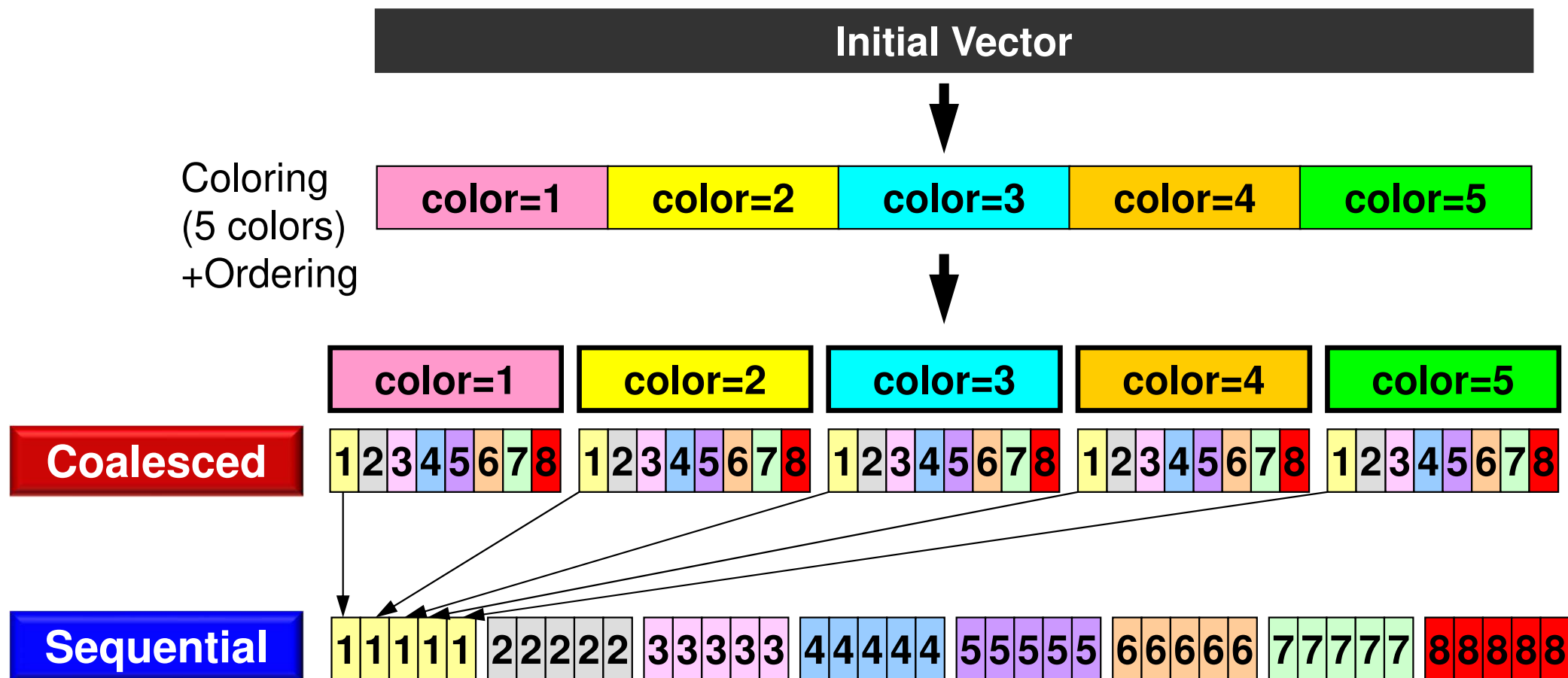
- 5-colors, 8-threads
- Meshes in same color are independent: parallel processing
- Reordering in ascending order according to color ID

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)
- Inconsistent numbering
 - $\text{itemL}(k) > \text{icel}$
 - $\text{indexL}(\text{icel}-1)+1 \leq k \leq \text{indexL}(\text{icel})$

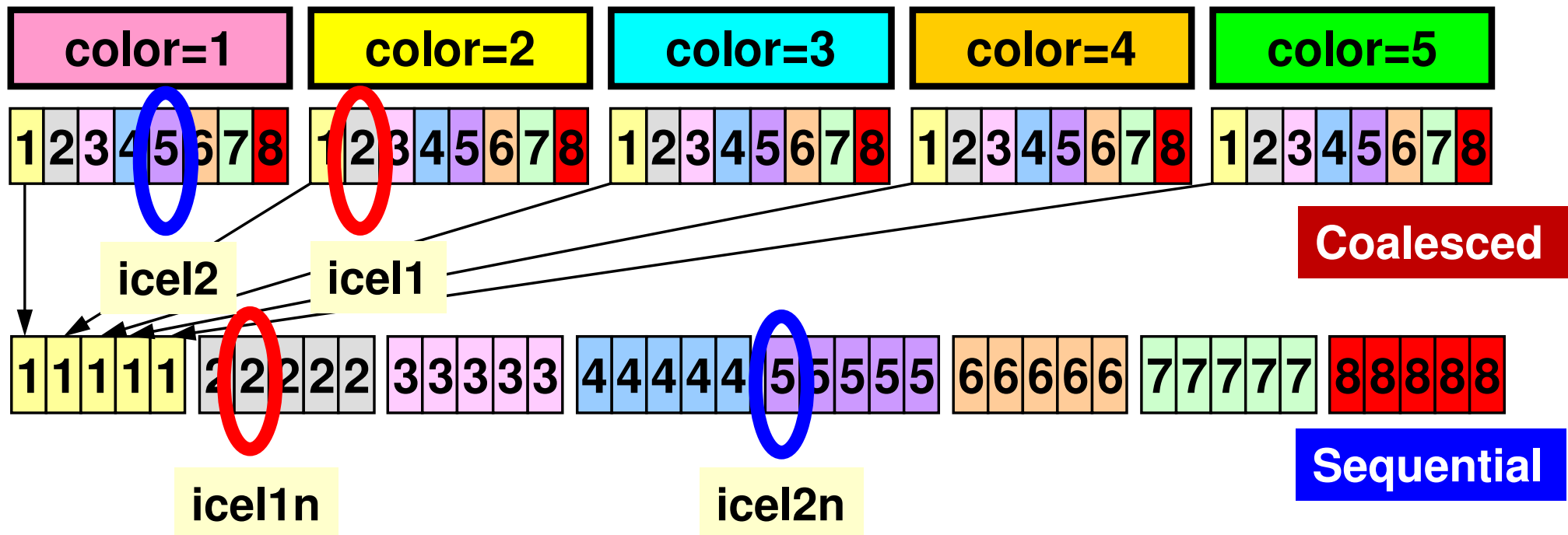
Sequential Reordering

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



Inconsistent numbering may occur

- Coalesced
 - $ice11 > ice12$, therefore, $ice12 = itemL(k)$, where $indexL(ice11) + 1 \leq k \leq indexL[ice11+1]$
- Sequential
 - $ice11n < ice12n$, but still $ice12n = itemL(k)$, where $indexL(ice11n) + 1 \leq k \leq indexL[ice11n+2]$



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)



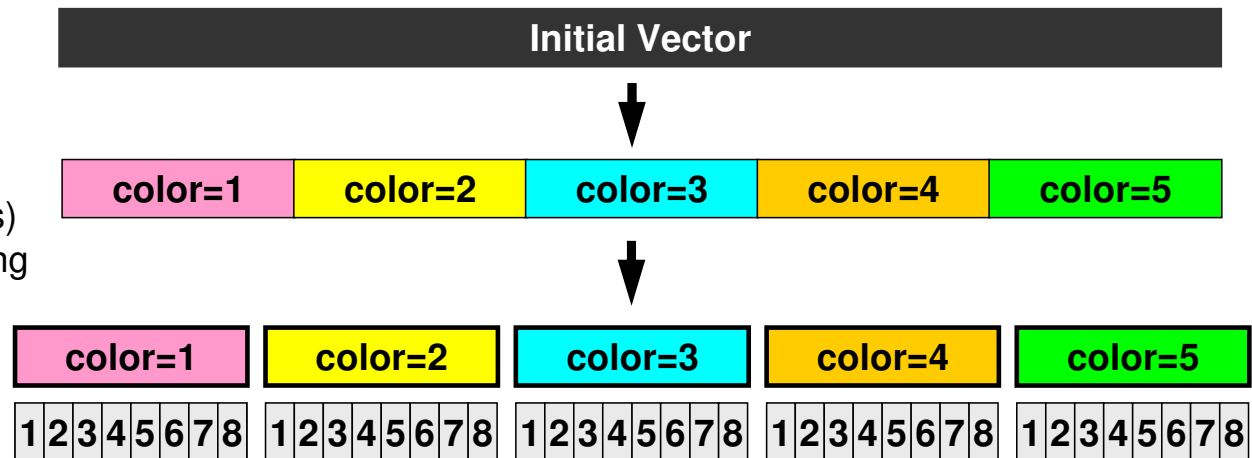
29	18	15	5	11	2	9	1
33	30	19	16	6	12	3	10
45	34	31	20	25	7	13	4
40	46	35	32	21	26	8	14
59	49	47	36	41	22	27	17
53	60	50	48	37	42	23	28
63	54	61	51	57	38	43	24
56	64	55	62	52	58	39	44

Sequential Reordering, 4-threads

Sequential Reordering

**Coalesced
Good for GPU**

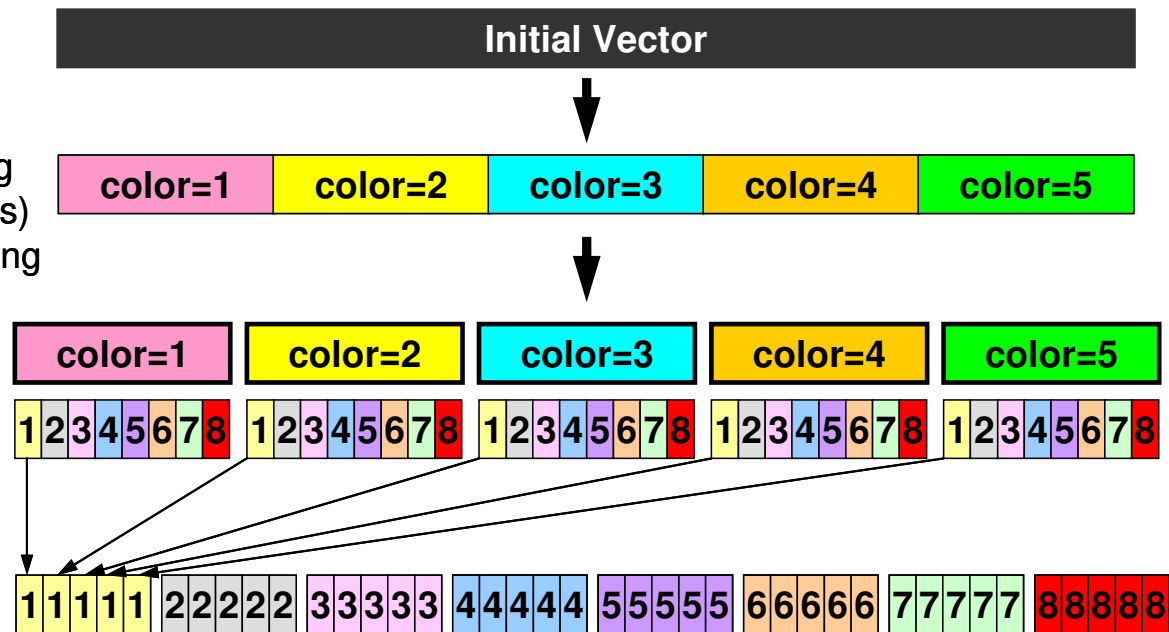
Coloring
(5 colors)
+Ordering



Sequential

Coloring
(5 colors)
+Ordering

各スレッド上で
不連続なメモリ
アクセス(色の
順に番号付け)



スレッド内で連続に番号付け

Programs (reorder0)

```
% cd /work/gt80/t80XXX/ompf/reorder0
% module load fj

% make
% cd ../run
% ls L3-rsol0
    L3-rsol0

<modify "INPUT.DAT">
<modify "gor.sh">

% pjsub gor.sh
```

gor.sh

```
#!/bin/sh
#PJM -N "gor"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=12                (=PEsmpTOT)
#PJM -L elapse=00:15:00
#PJM -g gt80
#PJM -j
#PJM -e err
#PJM -o testr.lst

module load fj
export OMP_NUM_THREADS=12           (=PEsmpTOT)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl -l ./L3-rsol0
numactl -C 12-23 -m 4 ./L3-rsol0
```

INPUT.DAT

```

128 128 128          NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00  DX/DY/DZ
1.0e-08             EPSICC
12                  PEsmptTOT
-10                 NCOLORTot
0                   NFLAG
0                   METHOD

```

- **PEsmptTOT**
 - Thread Number (**--omp thread=XX**)
- **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

Sequential Reordering (1/5) 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, PEsmptOT,
&      SMPindex_new, EPSICCG, ITR, IER)
  else
    call solve_ICCG_mc_ft
&      ( ICELTOT, NPL, NPU, indexL, itemL, indexU, itemU,
&      D, BFORCE, PHI, AL, AU, NCOLORtot, PEsmptOT,
&      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:PEsmpTOT*NCOLORtot))
SMPindex= 0
do ic= 1, NCOLORtot
  nn1= COLORindex(ic) - COLORindex(ic-1)
  num= nn1 / PEsmpTOT
  nr = nn1 - PEsmpTOT*num
  do ip= 1, PEsmpTOT
    if (ip.le.nr) then
      SMPindex((ic-1)*PEsmpTOT+ip)= num + 1
    else
      SMPindex((ic-1)*PEsmpTOT+ip)= num
    endif
  enddo
enddo

```

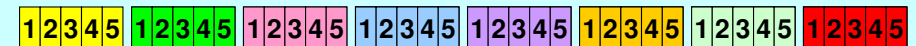
```

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

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

```

SMPindex
Coalesced

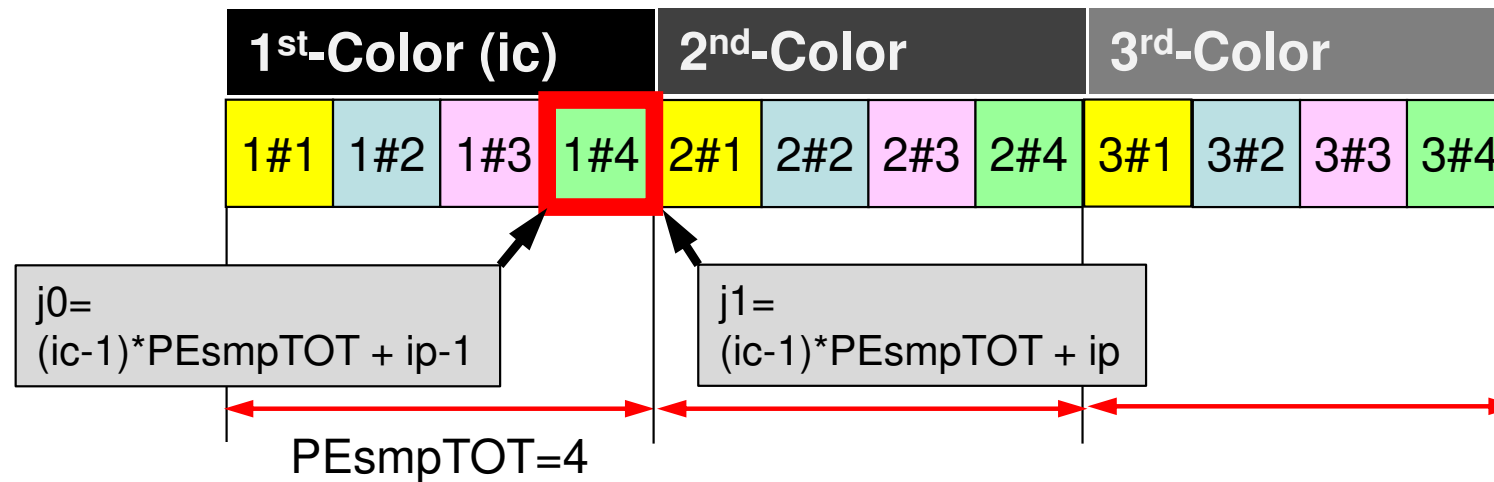
SMPindex_new
Sequential


Sequential Reordering (2/5) poi_gen-1

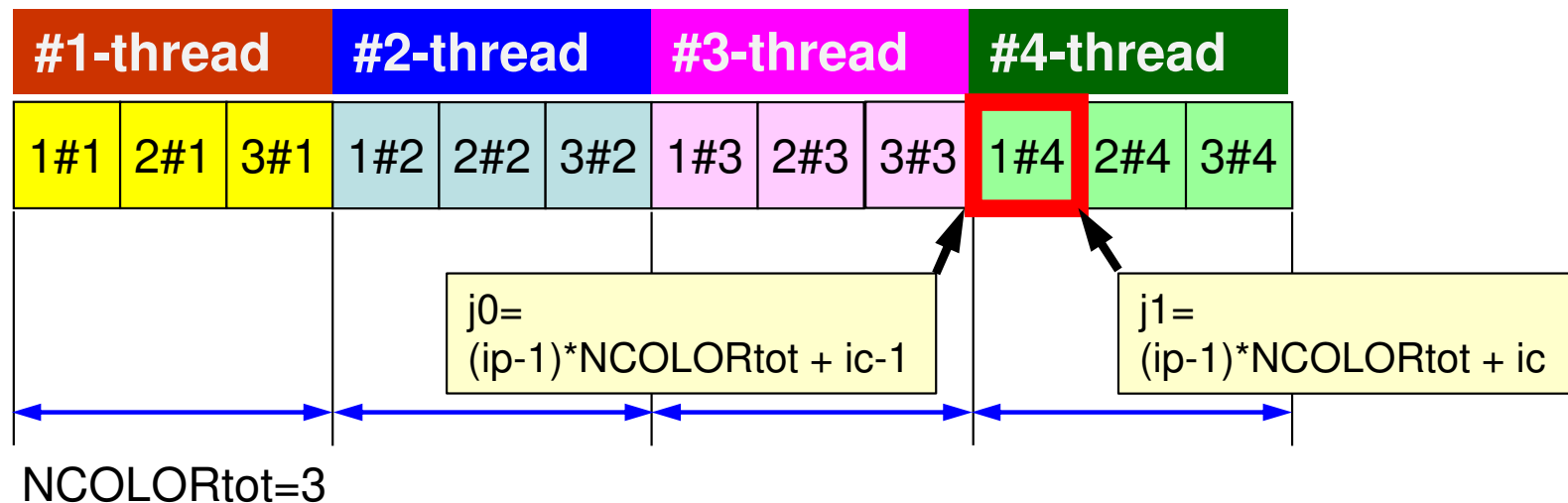
SMPIndex

ic#ip

ic: 1 ~ NCOLORtot
ip: 1 ~ PEsmptTOT



SMPIndex_new

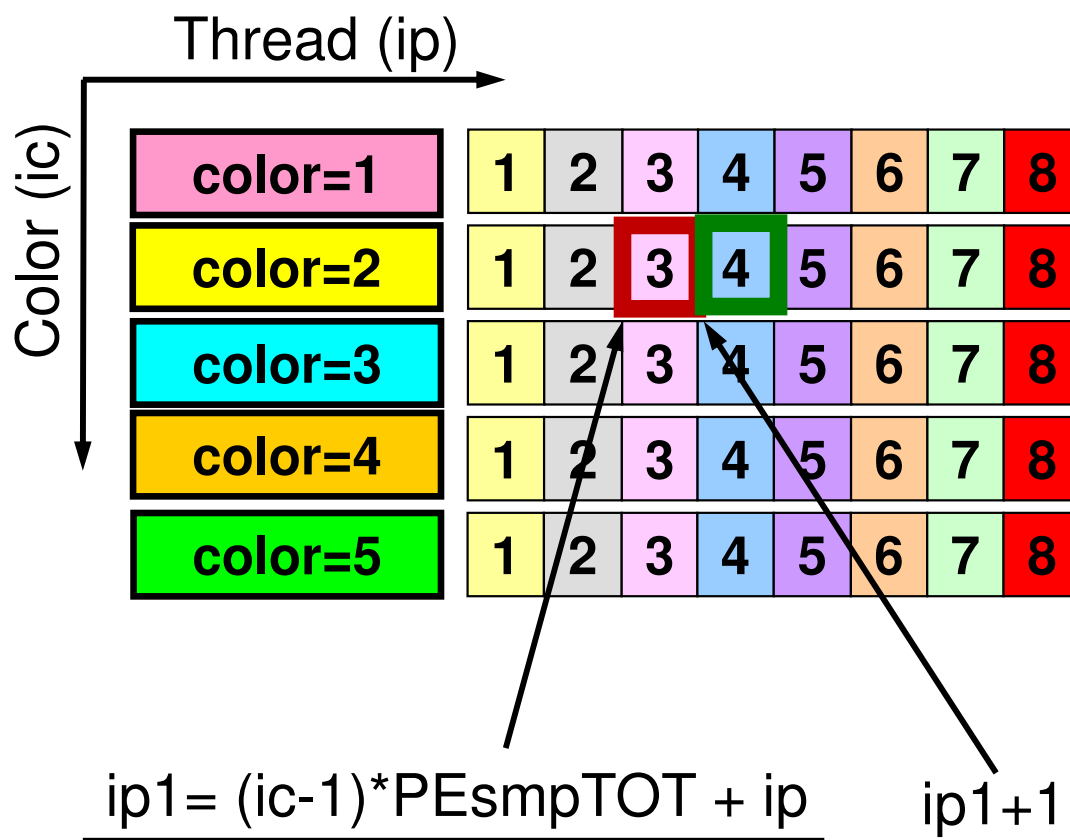


Coalesced

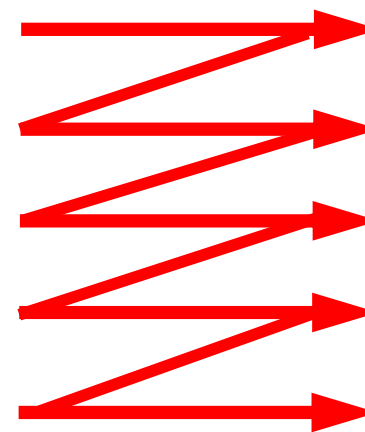
```

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

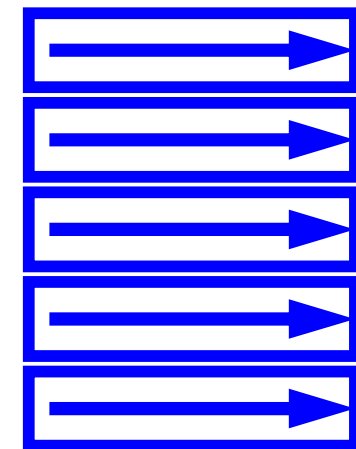
```



Numbering



Parallel Accessing

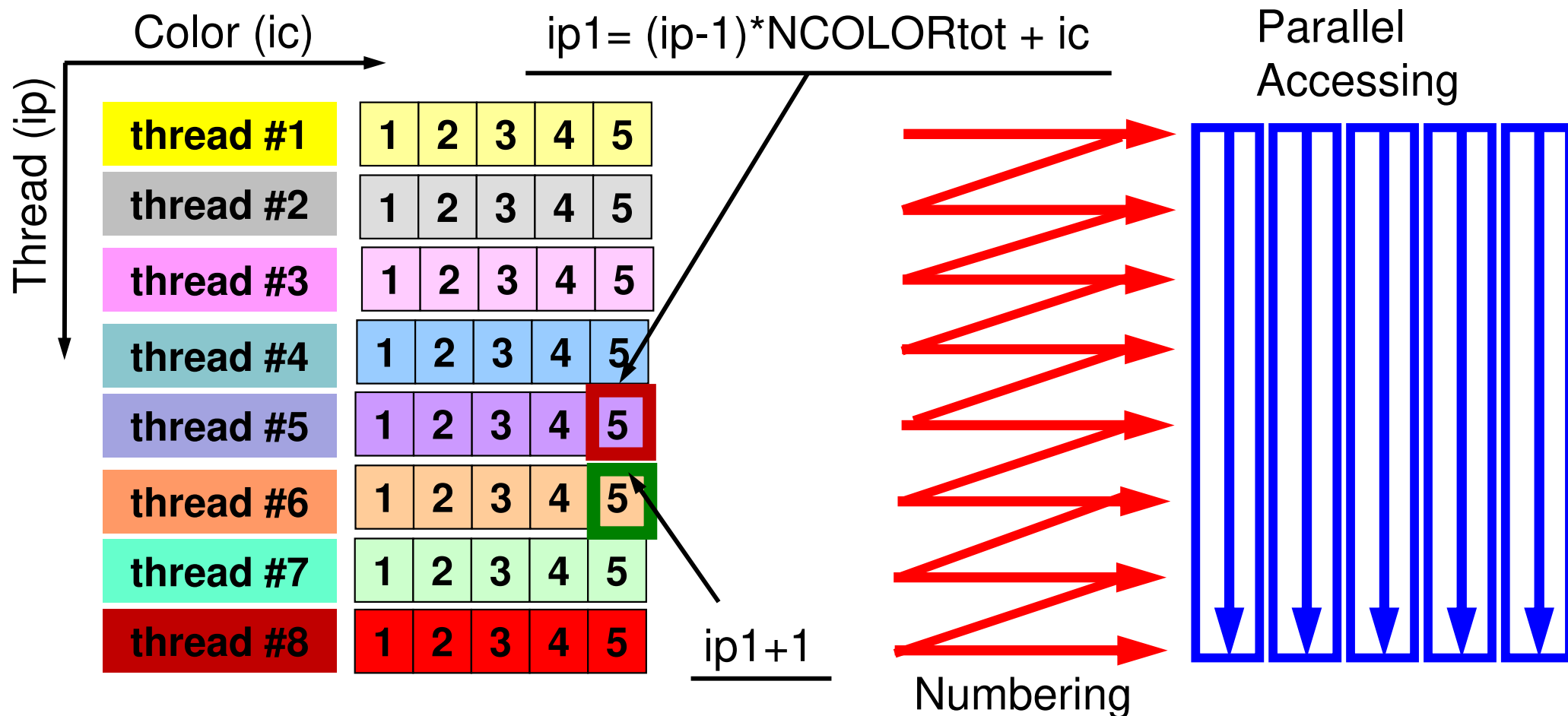


Sequential

```

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

```



Sequential Reordering (3/5) poi_gen-2

```
do ip= 1, PEsmptOT
  do ic= 1, NCOLORTot
    icNS= SMPindex_new((ip-1)*NCOLORTot + ic-1)
    ic01= SMPindex((ic-1)*PEsmptOT + ip-1) + 1
    ic02= SMPindex((ic-1)*PEsmptOT + 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
```

OLDtoNEWnew: Original -> Sequential
NEWtoOLDnew: Sequential -> Original
-Original: Initial icel
-Sequential icelN

```
!$omp parallel do private (ip, icel, ic0, ik0)
  do ip = 1, PEsmptOT
    do icel= SMPindex_new((ip-1)*NCOLORTot)+1, SMPindex_new(ip*NCOLORTot)
      ic0 = NEWtoOLDnew(icel)
      ik0 = OLDtoNEW(ic0)
      indexL(icel)= INL(ik0)
      indexU(icel)= INU(ik0)
    enddo
  enddo
```

Sequential

Coalesced

enddo
enddo

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

-Original: Initial ic0
-Coalesced ik0
-Sequential icel

Sequential Reordering (4/5)

poi_gen-3

```

NPL= indexL(ICELTOT)
NPU= indexU(ICELTOT)

allocate (itemL(NPL), AL(NPL))
allocate (itemU(NPU), AU(NPU))

if (NFLAG.eq.0) then
  itemL= 0
  itemU= 0
  AL= 0.d0
  AU= 0.d0
else
!$omp parallel do private (ip, icel, k)
  do ip= 1, PEsmptOT
    do icel= SMPindex_new((ip-1)*NCOLORtot+1),
&          SMPindex_new(ip*NCOLORtot)
      do k= indexL(icel-1)+1, indexL(icel)
        itemL(k)= 0
        AL(k)= 0.d0
      enddo
      do k= indexU(icel-1)+1, indexU(icel)
        itemU(k)= 0
        AU(k)= 0.d0
      enddo
    enddo
  enddo
!$omp end parallel do
endif

```

```

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

```

Sequential Reordering (5/5) poi_gen-4

icel: Sequential
ic0: Original
ik0: Coalesced

icN50: Original
icN5 : Coalesced

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

Forward Substitution

```

!$omp parallel private(ic, ip, ip1, i, WVAL, k)
  do ic= 1, NCOLORTot
!$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

```

!$omp parallel private(ic, ip, ip1, i, WVAL, k)
  do ic= 1, NCOLORTot
!$omp do
    do ip= 1, PEsmptOT
      ip1= (ip-1)*NCOLORTot + ic
      do i= SMPindex_new(ip1-1)+1, SMPindex_new(ip1)
        WVAL= W(i, Z)
        do k= indexLnew(i-1)+1, indexLnew(i)
          WVAL= WVAL - ALnew(k) * W(itemLnew(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

```

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

```

METHOD=0

```

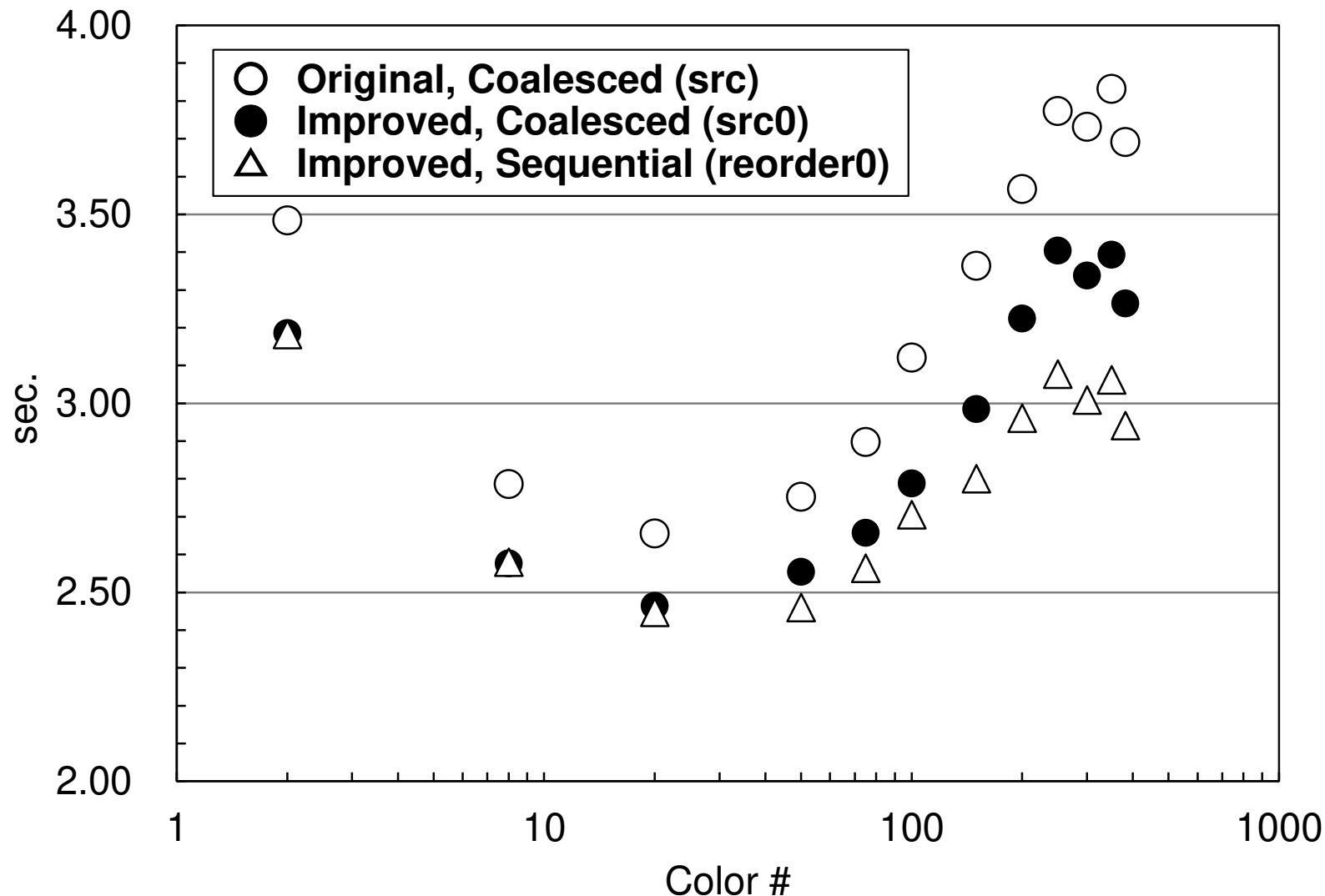
!$omp parallel do private(ip, i, VAL, k)
  do ip= 1, PEsmptTOT
    do i= SMPindex_new((ip-1)*NCOLORtot)+1, SMPindex_new(ip*NCOLORtot)
      VAL= D(i)*W(i, P)
      do k= indexLnew(i-1)+1, indexLnew(i)
        VAL= VAL + ALnew(k)*W(itemLnew(k), P)
      enddo
      do k= indexUnew(i-1)+1, indexUnew(i)
        VAL= VAL + AUnew(k)*W(itemUnew(k), P)
      enddo
      W(i, Q)= VAL
    enddo
  enddo
!$omp end parallel do

```

METHOD=1

Comp. Time for ICCG, CM-RCM

Generally “sequential (reorder0)” is stable and faster than “coalesced (src, src0)”. Effects are more significant in cases with more colors (12 threads, C)



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.
 - ✓ Page/Memory Page/Virtual Page: A fixed-length continuous block of virtual memory, smallest unit of data for memory management in a virtual memory OS
- 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 4 CMG's of the Odyssey system for a single OpenMP program
 - If you don't care, all pages are created at the local memory of CMG#0
 - Not needed for a single CMG case

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
48	PEsmpTOT
-50	NCOLORtot
0	NFLAG (0 or 1)
0	METHOD

- **PEsmpTOT**
 - Thread Number (**--omp thread=XX**)
- **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

g.sh: reorder0

```
#!/bin/sh
#PJM -N "go0"
#PJM -L rscgrp=lecture0-o
#PJM -L node=1
#PJM --omp thread=48                (=PEsmpTOT)
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test1.lst

module load fj
export OMP_NUM_THREADS=48            (=PEsmpTOT)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl -l ./L3-rsol0
numactl -C 12-59 -m 4-7 ./L3-rsol0
```


Array Initialization: NFLAG=0/1 (1/3)

poi_gen.f

```
!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 (2/3)

```
if (NFLAG.eq.0) then
  do icel= 1, ICELTOT
    indexL(icel)= 0
    indexU(icel)= 0
  enddo
else
  !$omp parallel do private (ip, icel, k)
    do ip= 1, PEsmpTOT
      do icel= SMPindex_new((ip-1)*NCOLORtot+1),
        &      SMPindex_new(ip*NCOLORtot)
        indexL(icel)= 0
        indexU(icel)= 0
      enddo
    enddo
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 (3/3)

```
if (NFLAG.eq. 0) then
  itemL= 0
  itemU= 0
  AL= 0. d0
  AU= 0. d0
else
  !$omp parallel do private (ip, icel, k)
    do ip= 1, PEsmptOT
      do icel= SMPindex_new((ip-1)*NCOLORtot+1),
        & SMPindex_new(ip*NCOLORtot)
        do k= indexL(icel-1)+1, indexL(icel)
          itemL(k)= 0
          AL(k)= 0. d0
        enddo
        do k= indexU(icel-1)+1, indexU(icel)
          itemU(k)= 0
          AU(k)= 0. d0
        enddo
      enddo
    enddo
  !$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.

Sequential Reordering (4/5)

poi_gen-3

```

NPL= indexL(ICELTOT)
NPU= indexU(ICELTOT)

allocate (itemL(NPL), AL(NPL))
allocate (itemU(NPU), AU(NPU))

if (NFLAG.eq.0) then
  itemL= 0
  itemU= 0
  AL= 0.d0
  AU= 0.d0
else
  !$omp parallel do private (ip, icel, k)
    do ip= 1, PESmpTOT
      do icel= SMPindex_new((ip-1)*NCOLORtot+1),
        &      SMPindex_new(ip*NCOLORtot)
        do k= indexL(icel-1)+1, indexL(icel)
          itemL(k)= 0
          AL(k)= 0.d0
        enddo
        do k= indexU(icel-1)+1, indexU(icel)
          itemU(k)= 0
          AU(k)= 0.d0
        enddo
      enddo
    enddo
  !$omp end parallel do
endif

```

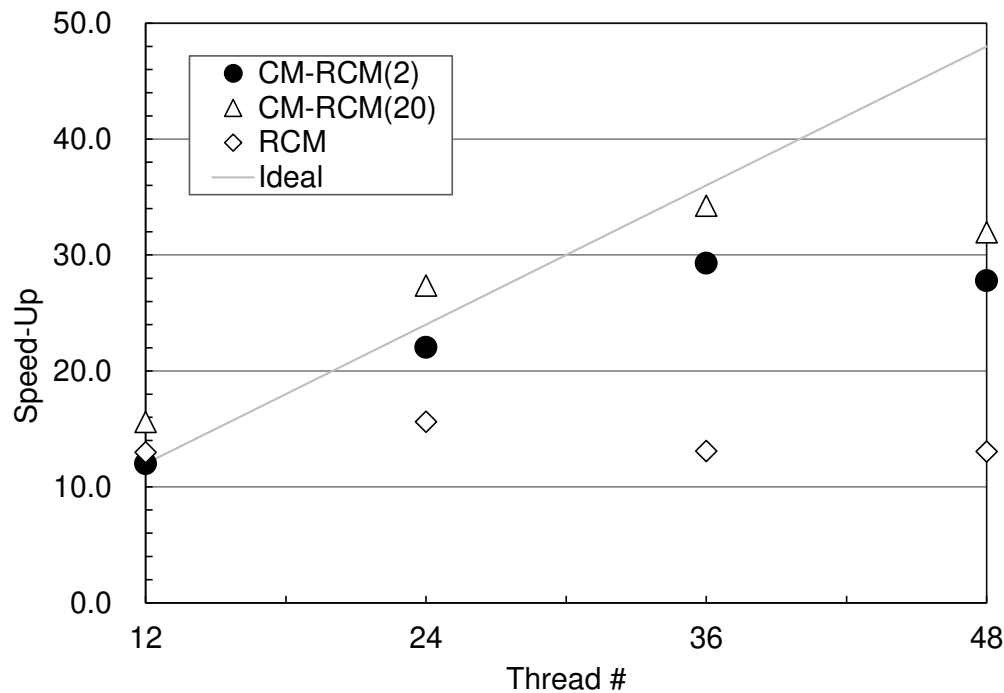
Pages are allocated at the local memory of the master thread

Pages are allocated at the local memory of each thread

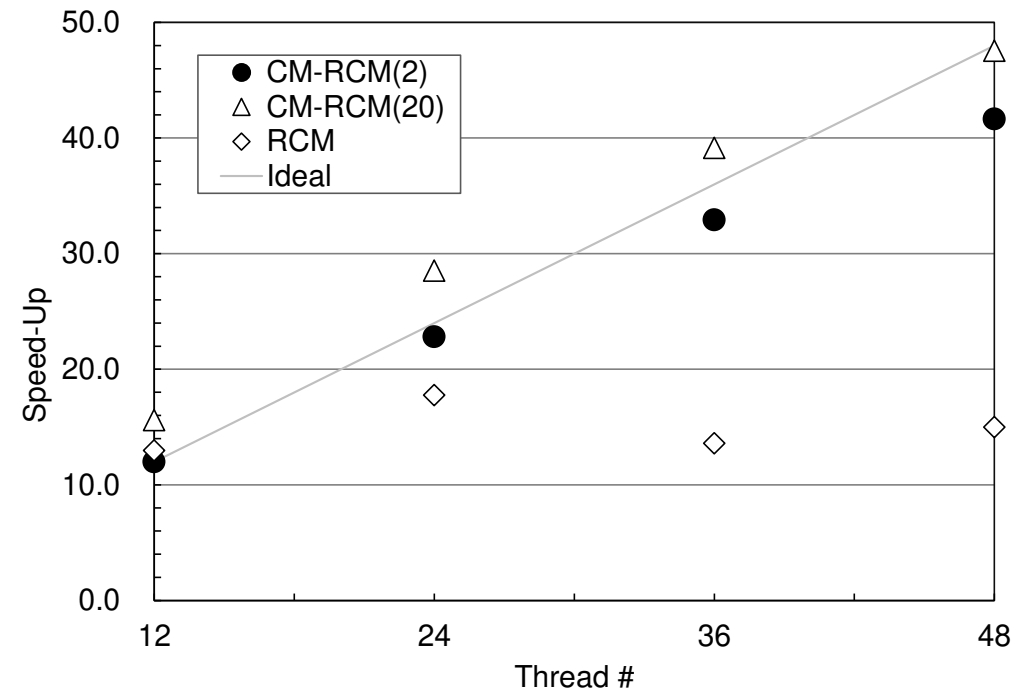
Results: reoder0, L3-rsol0, N=128³, C

based on the performance of CM-RCM(2) with
12 threads/without First Touch

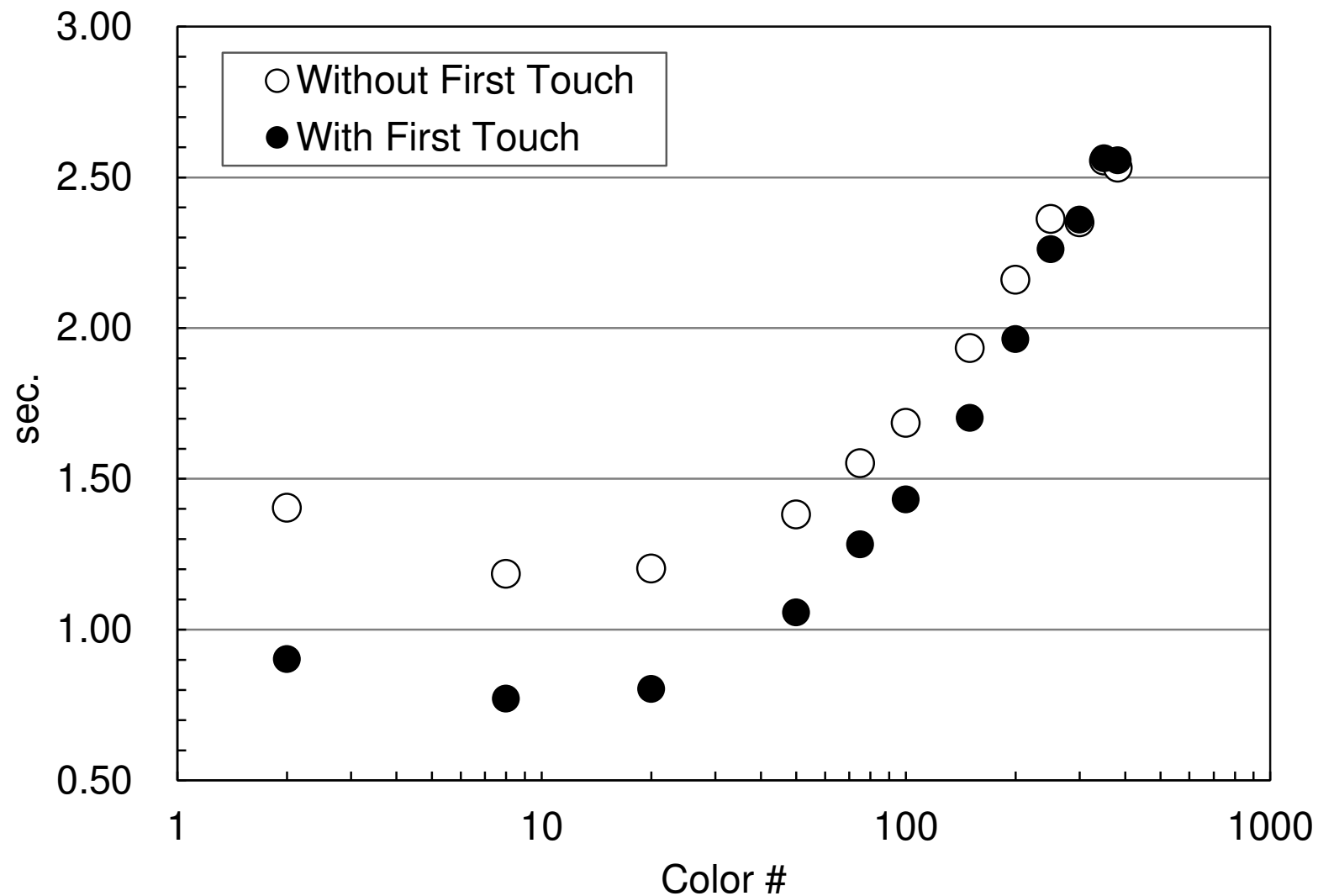
Without First-Touch



With First-Touch



Results: reoder0, L3-rsol0, N=128³, C 48 threads



Summary

- Material: ICCG solver for sparse matrices derived from FVM applications (Finite Volume Method).
- Parallelization on a single node of OBCX using OpenMP
 - Data Placement
 - Reordering
- Effects of reordering
- First-Touch Data Placement

Future Directions

- Gap between performance of CPU & memory
 - BYTE/FLOP
- Multicore/Manycore
 - Intel Xeon/Phi, GPU with OpenACC
- Supercomputer system with $>10^5$ cores
 - Exascale: $>10^8$
- Reordering/Ordering
 - Intensity of components of matrices should be also considered (not only the connectivity information)
 - Selection of optimum number of colors: research topic, especially for ill-conditioned problems
- OpenMP/MPI Hybrid -> One of effective choices
 - Optimization for OpenMP is the most critical
 - **Winter School: Parallel FEM using OpenMP/MPI**

Fortran Code

```
% cd /work/gt80/t80xxx
% module load fj

% cp /work/gt80/z30088/makeNEW.tar .
% tar xvf makeNEW.tar
% cd ompf/src
% make -f make-n clean
% make -f make-n
% cd ../src0
% make -f make-n clean
% make -f make-n
% cd ../reorder0
% make -f make-n clean
% make -f make-n

% cd ../run
% ls *NEW
      L3-rsol0-NEW  L3-sol0-NEW  L3-sol-NEW
```

make-n

```
F90      = frtpx
F90OPTFLAGS= -Kno swp, openmp, nosimd, nounroll
F90FLAGS =$(F90OPTFLAGS)

.SUFFIXES:
.SUFFIXES: .o .f .f90 .c
#
.f90.o:; $(F90) -c $(F90FLAGS) $(F90OPTFLAG) $<
.f.o:; $(F90) -c -loglist $(F90FLAGS) $(F90OPTFLAG) $<
#
OBJS = ¥
solver_ICCG_mc.o solver_ICCG_mc_ft.o struct.o pcg.o ¥
boundary_cell.o cell_metrics.o ¥
input.o main.o poi_gen.o pointer_init.o outucd.o mc.o cm.o rcm.o
cmrcm.o

TARGET = ../run/L3-rsol0-NEW

all: $(TARGET)

$(TARGET): $(OBJS)
    $(F90) $(F90FLAGS) -o $(TARGET) ¥
    $(OBJS) ¥
    $(F90FLAGS)

clean:
    rm -f *.o $(TARGET) *.mod *~ PI* *.log *.lst
```

**All options for optimization
are suppressed**

Effects of Compiler Options

- Original: `-Kfast, openmp -KSVE`
- NEW: `-Knoswp, openmp, nosimd, nounroll`
- If the most inner-loop is small (3-6), optimization of Fujitsu's Fortran compiler does not work well.
- $N=128^3$, CM-RCM(20), 48-threads, First-Touch
 - reorder0
 - 1.484 sec. -> 1.354 sec.
- **reorder0+ELL**
 - 0.674. sec -> 1.467 sec.