

科学技術計算のための マルチコアプログラミング入門

第Ⅲ部：OpenMPによる並列化

中島研吾

東京大学情報基盤センター

OpenMP並列化

- L2-solをOpenMPによって並列化する。
 - 並列化にあたってはスレッド数を「PEsmpTOT」によってプログラム内で調節できる方法を適用する
- 基本方針
 - 同じ「色」(または「レベル」)内の要素は互いに独立, したがって並列計算(同時処理)が可能

4色, 4スレッドの例

初期メッシュ

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色, 4スレッドの例

初期メッシュ

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色, 4スレッドの例 色の順に番号付け

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色, 4スレッドの例

同じ色の要素は独立: 並列計算可能
番号順にスレッドに割り当てる

	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

プログラムのありか on Oakleaf-FX

- 所在
 - `<$0-L3>/src`, `<$0-L3>/run`
- コンパイル, 実行方法
 - 本体
 - `cd <$0-L3>/src`
 - `make`
 - `<$0-L3>/run/L3-sol`(実行形式)
 - コントロールデータ
 - `<$0-L3>/run/INPUT.DAT`
 - 実行用シェル
 - `<$0-L3>/run/go1.sh`

実行例

```
% cd <$0-L3>
% ls
      run      src      src0      src      short (Fortranのみ)

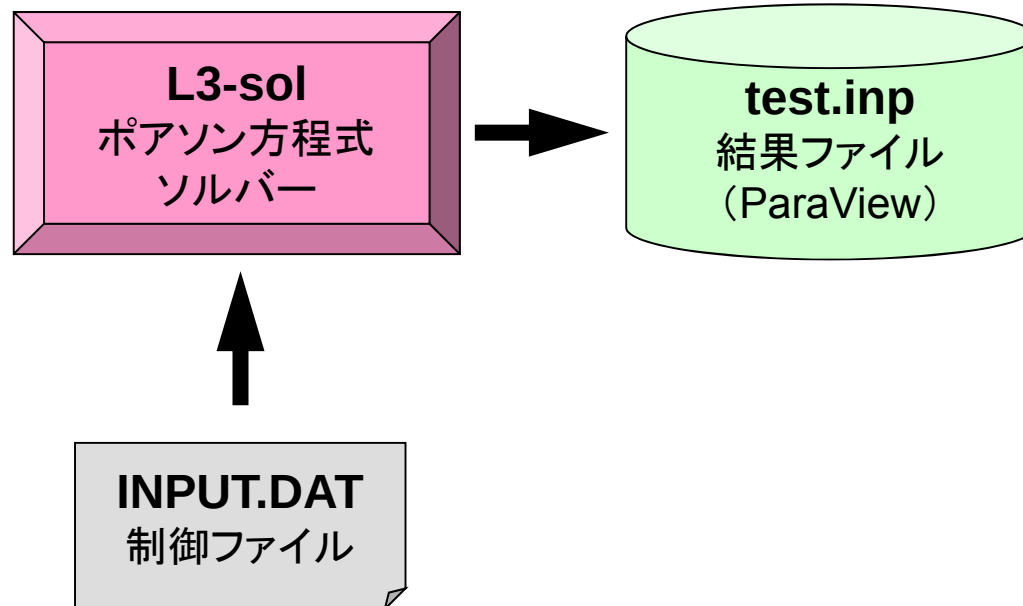
% cd src
% make
% cd ../run
% ls L3-sol
      L3-sol

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

% pjsub go1.sh
```


プログラムの実行

プログラム, 必要なファイル等



制御データ(INPUT.DAT)

```

100 100 100          NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00  DX/DY/DZ
1.0e-08             EPSICCG
16                  PEsmptOT
100                 NCOLORTot
  
```

変数名	型	内 容
NX, NY, NZ	整数	各方向の要素数
DX, DY, DZ	倍精度実数	各要素の3辺の長さ (ΔX , ΔY , ΔZ)
EPSICCG	倍精度実数	収束判定値
PEsmptOT	整数	データ分割数
NCOLORTot	整数	Ordering手法と色数 ≥ 2 : MC法 (multicolor) , 色数 $= 0$: CM法 (Cuthill-Mckee) $= -1$: RCM法 (Reverse Cuthill-Mckee) ≤ -2 : CM-RCM法

go1.sh

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapsed=00:10:00"
#PJM -L "rscgrp=lecture"
#PJM -g "gt61"
#PJM -j
#PJM -o "arcm.lst"

export OMP_NUM_THREADS=16
./L3-sol
```

- L2-solへのOpenMPの実装
- Hitachi SR11000/J2での実行
 - 計算結果
 - CM-RCMオーダリング
- FX10での実行

L2-solにOpenMPを適用

- ICCGソルバーへの適用を考慮すると
- 内積, DAXPY, 行列ベクトル積
 - もともとデータ依存性無し $\Rightarrow$ straightforwardな適用可能
- 前処理(修正不完全コレスキー分解, 前進後退代入)
 - 同じ色内は依存性無し $\Rightarrow$ 色内では並列化可能

実はこのようにしてDirectiveを 直接挿入しても良いのだが・・・(1/2)

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

- スレッド数をプログラムで制御できるようにしてみよう

実はこのようにしてDirectiveを 直接挿入しても良いのだが・・・(2/2)

```
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
```

- スレッド数をプログラムで制御できるようにしてみよう

ICCG法の並列化: OpenMP

- 内積: OK
- DAXPY: OK
- 行列ベクトル積: OK
- 前処理

プログラムの構成(1/2)

```
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)  &
```

プログラムの構成(2/2)

```
allocate (WK(ICELTOT))
```

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

結果(PHI)をもとの番号
付けにもどす

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

```
call OUTUCD
```

```
stop  
end
```

プログラムの構成

```
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)  &
```

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 : 要素数

ICELTOTp : = ICELTOT

N : 節点数

NX, NY, NZ : x, y, z方向要素数

NXP1, NYP1, NZP1 :

& x, y, z方向節点数

& IBNODTOT : NXP1 × NYP1

& XYZ(ICELTOT, 3) : 要素座標 (後述)

NEIBcell(ICELTOT, 6) :

& 隣接要素 (後述)

境界条件関連 : Z=Zmax

PEsmptTOT : スレッド数

module PCG(これまでとの相違点)

```

module PCG

integer, parameter :: N2= 256
integer :: NUmex, 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

COLORindex(0:NCOLORtot)

色数

各色に含まれる要素数のインデックス

(COLORindex(icol)-COLORindex(icol-1))

SMPindex(0:NCOLORtot*PEsmptOT) スレッド用配列 (後述)

SMPindexG(0:PEsmptOT)

OLDtoNEW, NEWtoOLD

Coloring前後の要素番号対照表

変数表(1/2)

配列・変数名	型	内 容
D(N)	R	対角成分, (N:全メッシュ数)
BFORCE(N)	R	右辺ベクトル
PHI(N)	R	未知数ベクトル
indexL(0:N)	I	各行の非零下三角成分数(CRS)
indexU(0:N)	I	各行の非零上三角成分数(CRS)
NPL	I	非零下三角成分総数(CRS)
NPU	I	非零上三角成分総数(CRS)
itemL(NPL)	I	非零下三角成分(列番号)(CRS)
itemU(NPU)	I	非零上三角成分(列番号)(CRS)
AL(NPL)	R	非零下三角成分(係数)(CRS)
AU(NPL)	R	非零上三角成分(係数)(CRS)
NL, NU	I	各行の非零上下三角成分の最大数 (ここでは6)
INL(N)	I	各行の非零下三角成分数
INU(N)	I	各行の非零上三角成分数
IAL(NL, N)	I	各行の非零下三角成分に対応する列番号
IAU(NU, N)	I	各行の非零上三角成分に対応する列番号

変数表(2/2)

配列・変数名	型	内 容
NCOLORtot	I	入力時にはOrdering手法 (≥ 2 : MC, $=0$: CM, $=-1$: RCM, $-2 \leq$: CMRCM) , 最終的には色数, レベル数が入る
COLORindex(0:NCOLORtot)	I	各色, レベルに含まれる要素数の 一次元圧縮配列, COLORindex(icol-1)+1から COLORindex(icol)までの要素がicol番 目の色 (レベル) に含まれる。
NEWtoOLD(N)	I	新番号⇒旧番号への参照配列
OLDtoNEW(N)	I	旧番号⇒新番号への参照配列
PEsmpTOT	I	スレッド数
SMPindex (0:NCOLORtot*PEsmpTOT)	I	スレッド用補助配列 (データ依存性がある ループに使用)
SMPindexG(0:PEsmpTOT)	I	スレッド用補助配列 (データ依存性が無い ループに使用)

プログラムの構成

```
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, PEsmptOT,
&      SMPindex, SMPindexG, EPSICCG, ITR, IER) &
```


input

「IPNUT.DAT」の読み込み

```
!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
 - OpenMPスレッド数
- NCOLORtot
 - 色数
 - 「=0」の場合はCM
 - 「=-1」の場合はRCM
 - 「 ≤ -2 」の場合はCM-RCM

```
100 100 100
1.00e-02 5.00e-02 1.00e-02
1.00e-08
16
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)

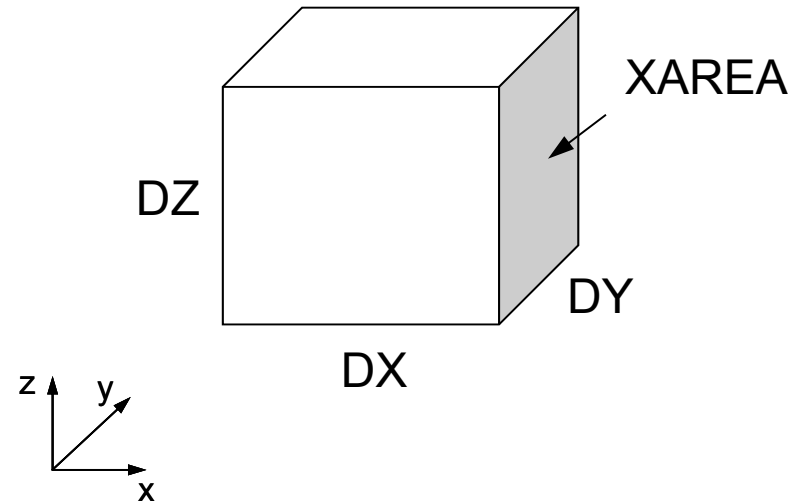
      V0= DX * DY * DZ
      RV0= 1. d0/V0

      VOLCEL= V0
      RVC    = RV0

      return
      end

```

計算に必要な諸パラメータ



プログラムの構成

```
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
```

配列の宣言

配列・変数名	型	内 容
D (N)	R	対角成分, (N:全メッシュ数)
BFORCE (N)	R	右辺ベクトル
PHI (N)	R	未知数ベクトル
indexL (0:N)	I	各行の非零下三角成分数(CRS)
indexU (0:N)	I	各行の非零上三角成分数(CRS)
NPL	I	非零下三角成分総数(CRS)
NPU	I	非零上三角成分総数(CRS)
itemL (NPL)	I	非零下三角成分(列番号)(CRS)
itemU (NPU)	I	非零上三角成分(列番号)(CRS)
AL (NPL)	R	非零下三角成分(係数)(CRS)
AU (NPL)	R	非零上三角成分(係数)(CRS)

$$\begin{array}{c} | \text{C} \text{ } \dot{+} - \\ | \text{C} \text{ } \equiv \equiv \\ | \end{array}$$

! C==

A 3D cube representing a central cell. Six arrows point away from the cube's faces, each labeled with a neighboring cell:

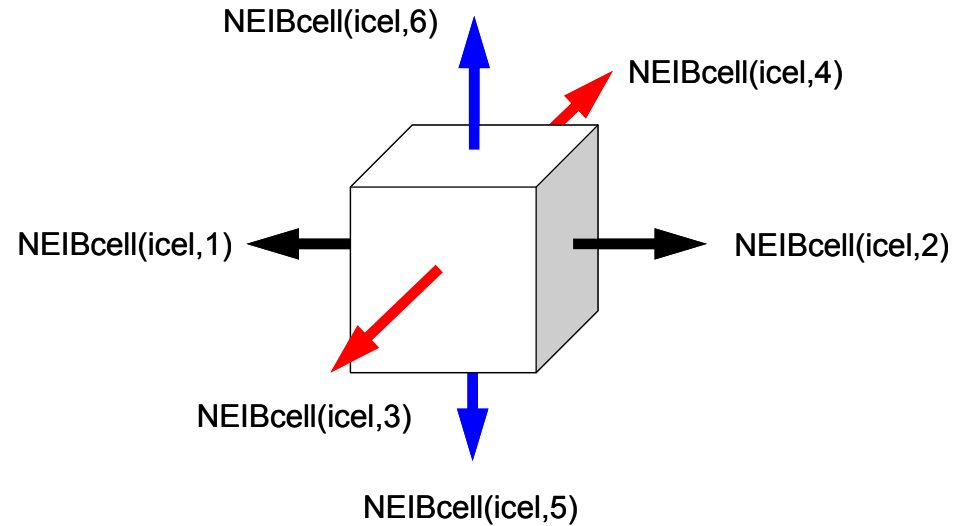
- Top face: Blue arrow pointing up, labeled $\text{NEIBcell}(\text{icel},6)$
- Bottom face: Blue arrow pointing down, labeled $\text{NEIBcell}(\text{icel},5)$
- Left face: Black arrow pointing left, labeled $\text{NEIBcell}(\text{icel},1)$
- Right face: Black arrow pointing right, labeled $\text{NEIBcell}(\text{icel},2)$
- Front face: Red arrow pointing down-left, labeled $\text{NEIBcell}(\text{icel},3)$
- Back face: Red arrow pointing up-right, labeled $\text{NEIBcell}(\text{icel},4)$

```
NEIBcell(icol,5)= icol - NX*NY
NEIBcell(icol,3)= icol - NX
NEIBcell(icol,1)= icol - 1
```

$$\begin{array}{l} |C \\ |C \\ |C \\ |C \\ |C= \end{array}$$

! C===

poi_gen (2/9)



上三角成分

```
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

```

並べ替えの実施 :

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

```

!C==

各色内の要素数：

$\text{COLORindex}(ic) - \text{COLORindex}(ic-1)$

同じ色内の要素は依存性が無いため、
並列に計算可能 $\Rightarrow$ OpenMP適用

これを更に「PEsmpTOT」で割って
「SMPindex」に割り当てる。

前処理で使用

```

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

```

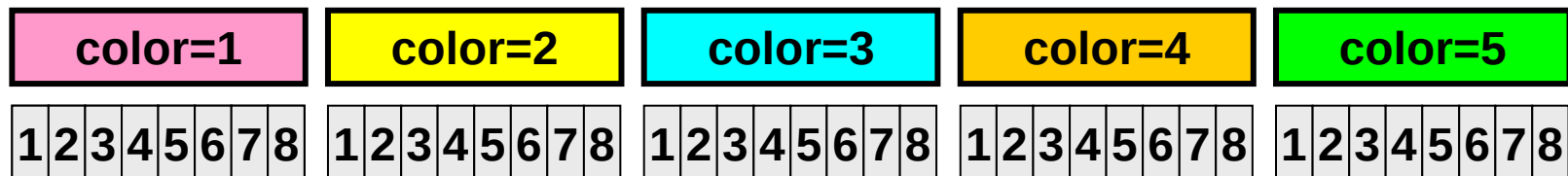
SMPindex: 前処理向け

```
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色, 8スレッドの例
- 同じ「色」に属する要素は独立⇒並列計算可能
- 色の順番に並び替え

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

```

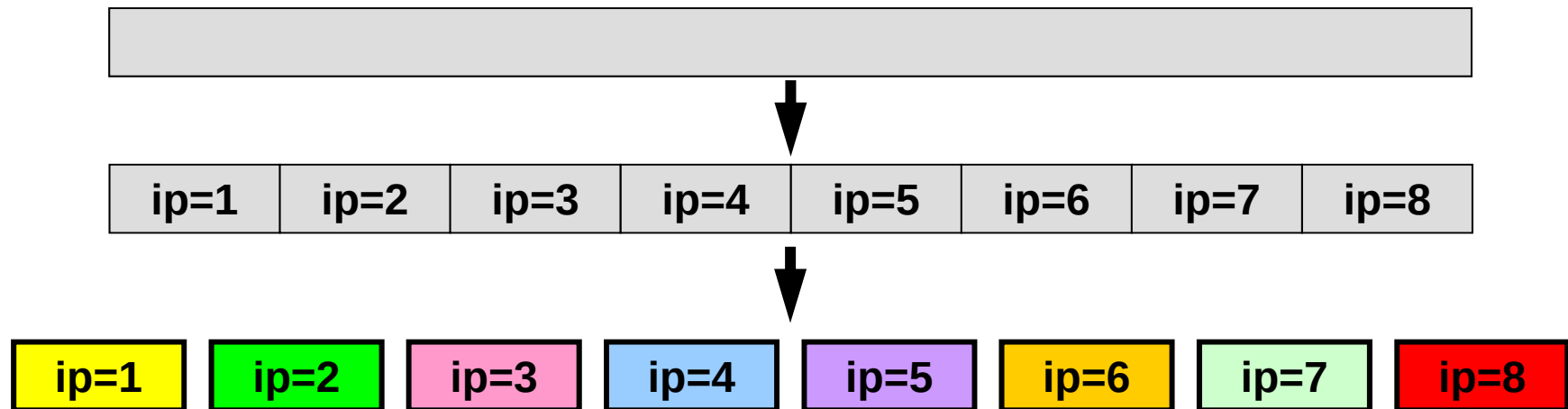
全要素数を「PEsmpTOT」で割って
「SMPindexG」に割り当てる。

内積，行列ベクトル積，DAXPYで使用

これを使用すれば，実は，
「poi_gen(2/9)」の部分も並列化可能
「poi_gen(5/9)」以降では実際に使用

SMPindexG

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



各スレッドで独立に計算: 行列ベクトル積, 内積, DAXPY等

```

!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===

```

配列の宣言

poi_gen(5/9)

これ以降は新しい
番号付けを使用

配列・変数名	型	内 容
D (N)	R	対角成分, (N:全メッシュ数)
BFORCE (N)	R	右辺ベクトル
PHI (N)	R	未知数ベクトル
indexL(0:N)	I	各行の非零下三角成分数(CRS)
indexU(0:N)	I	各行の非零上三角成分数(CRS)
NPL	I	非零下三角成分総数(CRS)
NPU	I	非零上三角成分総数(CRS)
itemL(NPL)	I	非零下三角成分(列番号)(CRS)
itemU(NPU)	I	非零上三角成分(列番号)(CRS)
AL(NPL)	R	非零下三角成分(係数)(CRS)
AU(NPL)	R	非零上三角成分(係数)(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, PEsmptTOT
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: 新しい番号
ic0: 古い番号

```

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

```

poi_gen (6/9)

新しい番号付けを使用

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

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

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

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

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

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

係数の計算: 並列に実施可能

SMPindexG を使用

private宣言に注意

```
!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: 新しい番号
ic0: 古い番号

```

  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

```

poi_gen (6/9)

新しい番号付けを使用

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


```

!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, PEsmptTOT
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
endif
endif

```

poi_gen (6/9)

新しい番号付けを使用

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

```

!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, PEsmptTOT
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
endif
endif

```

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

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

icN5がicelより小さければ下三角成分

poi_gen (6/9)

新しい番号付けを使用

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

```

!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, PEsmptTOT
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
endif
endif

```

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

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

icN5がicelより大き
ければ上三角成分

poi_gen (6/9)

新しい番号付けを使用

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

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

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

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

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

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

```

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
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)

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

```
!$omp parallel do private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6) &
!$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)
```

もとの座標に従って
BFORCE計算

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

ii,jj,kk,VOL0はprivate

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

poi_gen (9/9)

係数の計算(境界面以外)

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

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

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

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

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

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

プログラムの構成

```
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, PEsmptOT,              &
&    SMPindex, SMPindexG, EPSICCG, ITR, IER)
```

この時点で、係数、右辺ベクトル
ともに、「新しい」番号にしたがって
計算、記憶されている。

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, PEsmptOT, SMPindex, SMPindexG, &
&      EPS, ITR, IER)
&

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

      integer :: N, NL, NU, NCOLORTot, PEsmptOT

      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*PEsmptOT) :: SMPindex
      integer, dimension(0:PEsmptOT) :: 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,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

```

不完全修正
コレスキー分解

不完全修正コレスキー分解

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

$W(i, DD):$	d_i
$D(i):$	a_{ii}
$IAL(j, i):$	k
$AL(j, i):$	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

```

不完全修正コレスキー分解：並列版

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

$W(i, DD):$	d_i
$D(i):$	a_{ii}
$IAL(j, i):$	k
$AL(j, i):$	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

```

privateに注意。

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

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

行列ベクトル積

依存性が無い⇒独立に計算可能⇒SMPindexG使用

```
!$omp parallel do private(ip, i, VAL, k)
  do ip= 1, PEsmptOT
    do i = SMPindexG(ip-1)+1, SMPindexG(ip)
      VAL= D(i)*X(i)
      do k= indexL(i-1)+1, indexL(i)
        VAL= VAL + AL(k)*X(itemL(k))
      enddo
      do k= indexU(i-1)+1, indexU(i)
        VAL= VAL + AU(k)*X(itemU(k))
      enddo
      W(i, R)= B(i) - VAL
    enddo
  enddo
!$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 $r^{(0)} = b - [A]x^{(0)}$

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

内積: SMPindexG使用, 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
```

```

      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==

```

solve_ICCG_mc (4/6)

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i= 1, 2, ...
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  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==

```

ここでは「SMPindex」を使う

solve_ICCG_mc (4/6)

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i= 1, 2, ...
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  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==

```

ここでは「SMPindex」を使う

solve_ICCG_mc (4/6)

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

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

前進代入
Forward Substitution

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

後退代入
Backward Substitution

前進代入: SMPindex使用

```
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
```

```

!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

```

```

!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

```

```

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

      ALPHA= RHO / C1
!C==

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

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

```

solve_ICCG_mc (6/6)

```

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

```

```

!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==

```

solve_ICCG_mc (6/6)

```

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

```

```

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

      ALPHA= RHO / C1
!C==

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

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

```

solve_ICCG_mc (6/6)

```

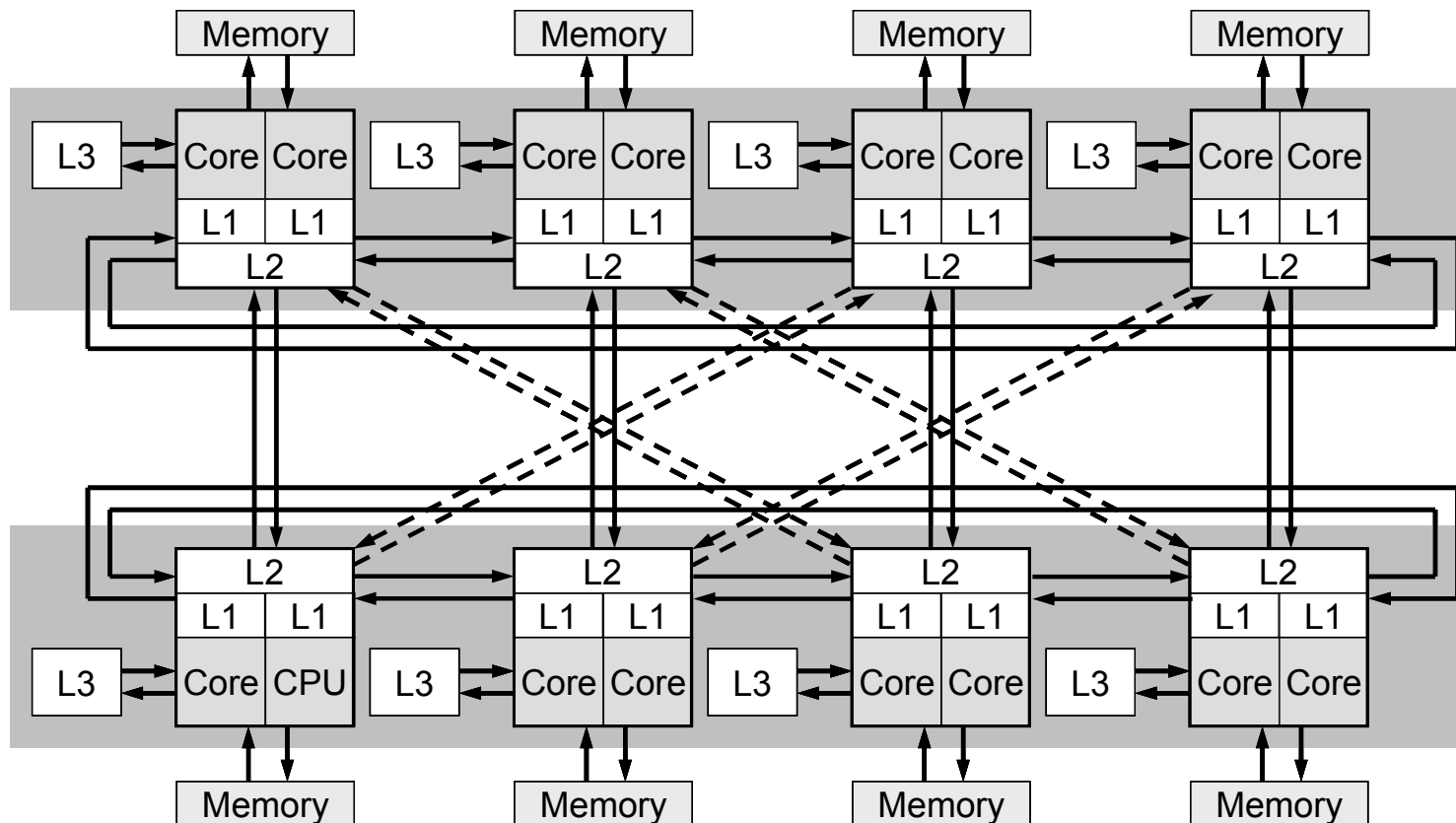
Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for i= 1, 2, ...
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  if i=1
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)} = [A]p^{(i)}$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

```


- L2-solへのOpenMPの実装
- **Hitachi SR11000/J2での実行**
 - 計算結果
 - CM-RCMオーダリング
- FX10での実行

計算結果

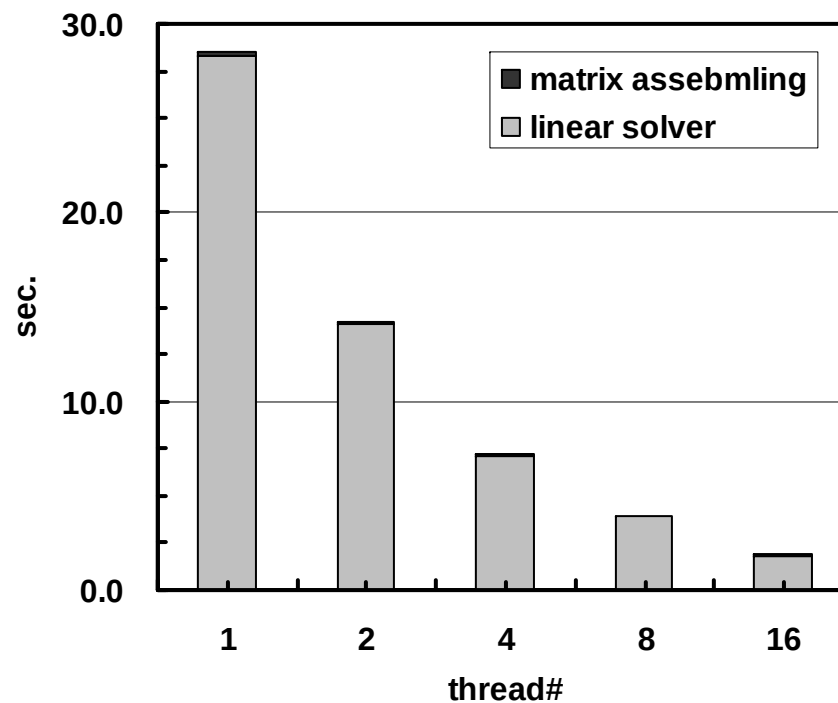
- Hitachi SR11000/J2 1ノード(16コア)
- 100^3 要素



計算時間 (MC=2)

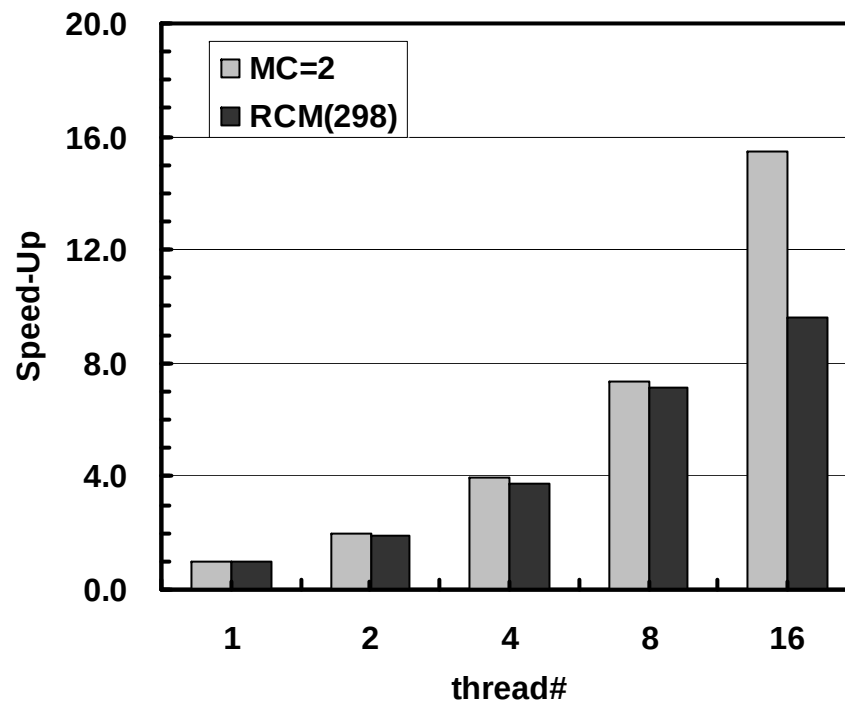
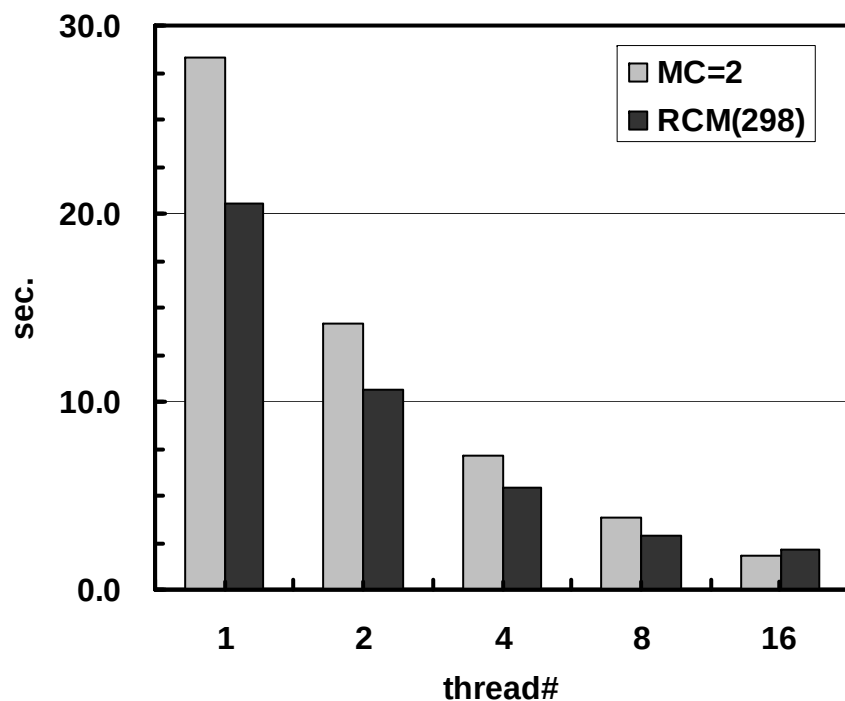
以降, 「linear solver」の計算時間のみ問題とする

matrix assembling: poi-genの後半: 並列化
linear solver: CG



スケーラビリティ

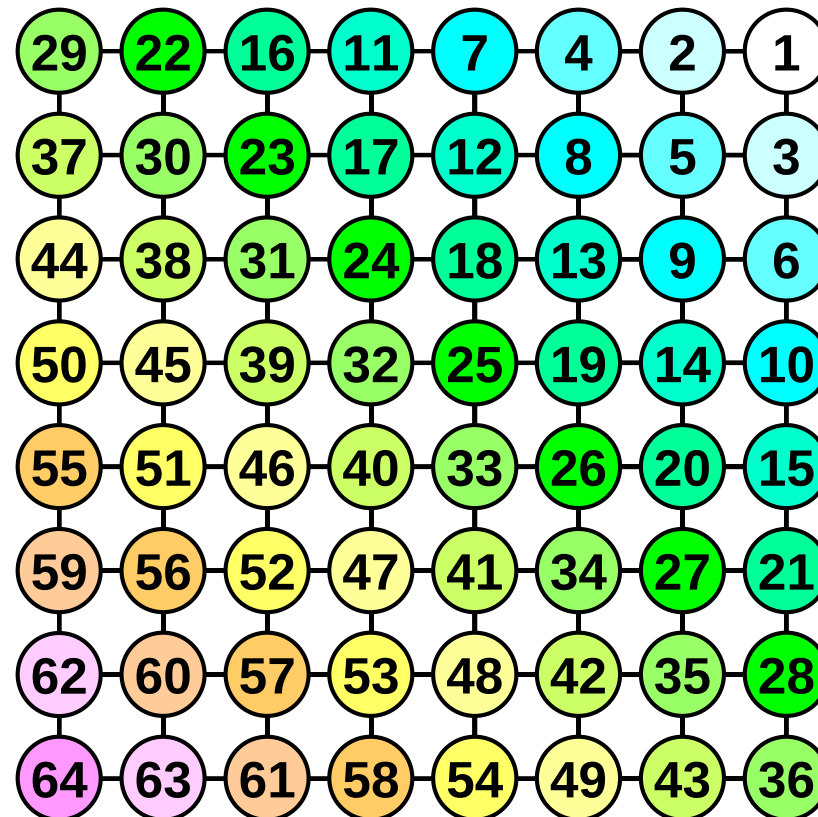
反復回数: MC(2色): 333回, RCM(298レベル): 224回



- 反復回数ではRCMの方が少ないのに, コア数が増えると, 計算時間が逆転する
- MC=2は良好なスケーラビリティ(16コアで15.5)

原因

- 色数が多い(298対2)⇒同期オーバーヘッド
- スレッドごとの負荷不均衡
 - RCMでは要素数の少ない「レベル」が必ず存在する

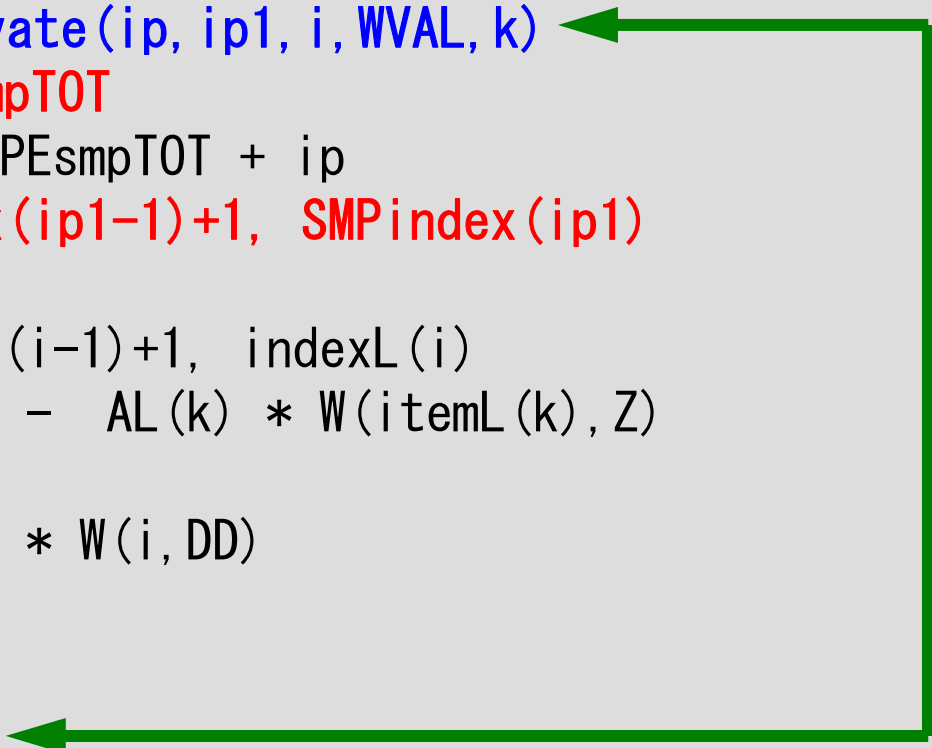


原因はこのようなループにある

色数増加⇒同期オーバーヘッド増加

必ずある「色」の計算が終わってから次の「色」に行く

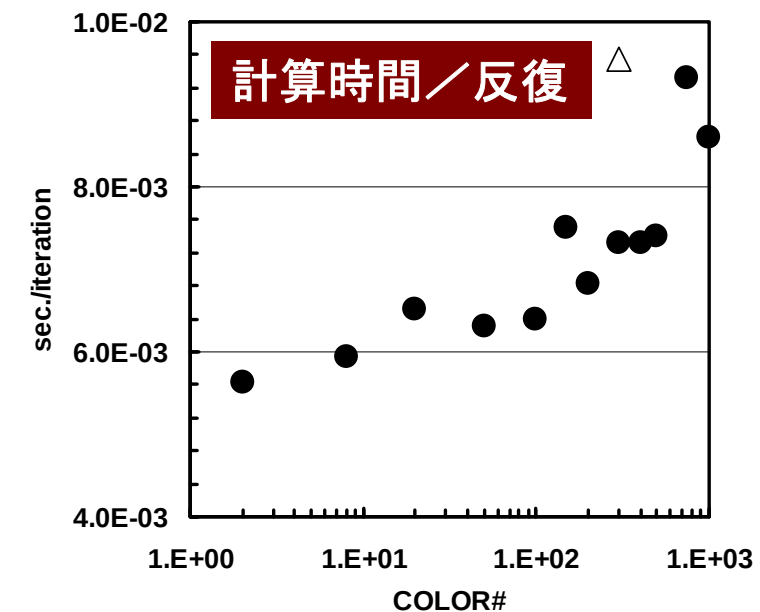
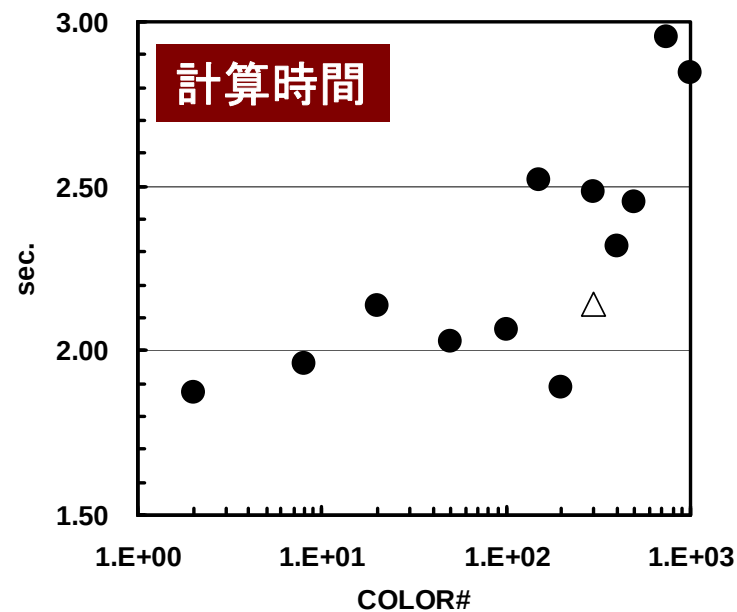
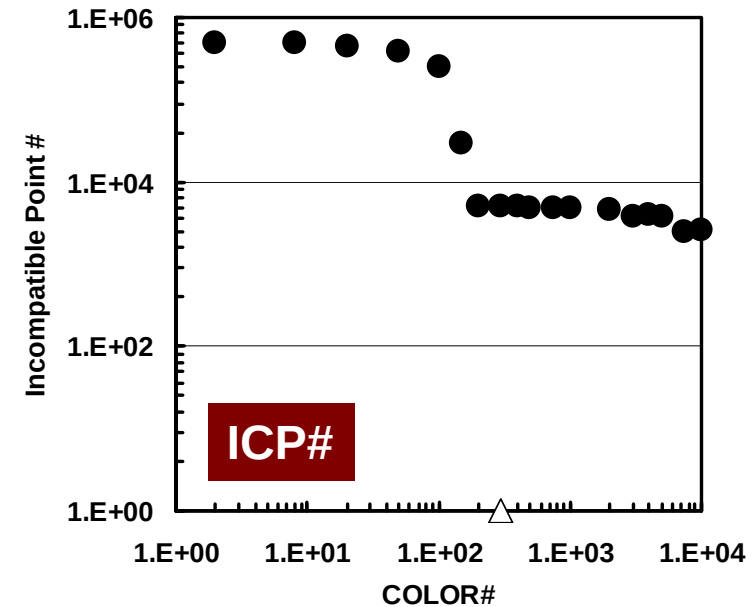
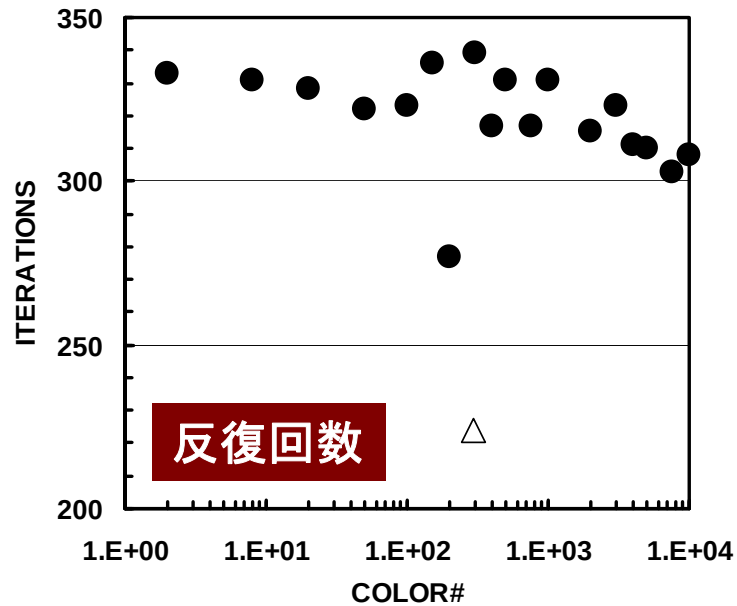
```
do ic= 1, NCOLORTot
!$omp parallel do private(ip, ip1, i, WVAL, k)
  do ip= 1, PEsmptOT
    ip1= (ic-1)*PEsmptOT + ip
    do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      WVAL= W(i, Z)
      do k= indexL(i-1)+1, indexL(i)
        WVAL= WVAL - AL(k) * W(itemL(k), Z)
      enddo
      W(i, Z)= WVAL * W(i,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



MCとRCMの比較

- MC
 - 並列性高い, 負荷分散も良い(そのように設定されている)
 - 特に色数少ないと反復回数多い
 - 色数を増やす, 反復回数減るが同期オーバーヘッドの影響
- RCM
 - 収束は早い, レベル数が多く, 同期オーバーヘッドの影響受けやすく, コア数が増えると不利
 - 負荷分散もいまいち
- 反復回数少なくて同期オーバーヘッドの影響が少ない方法が無いものか?
 - 色数が少なくて, かつ反復回数が少ないという都合の良い方法

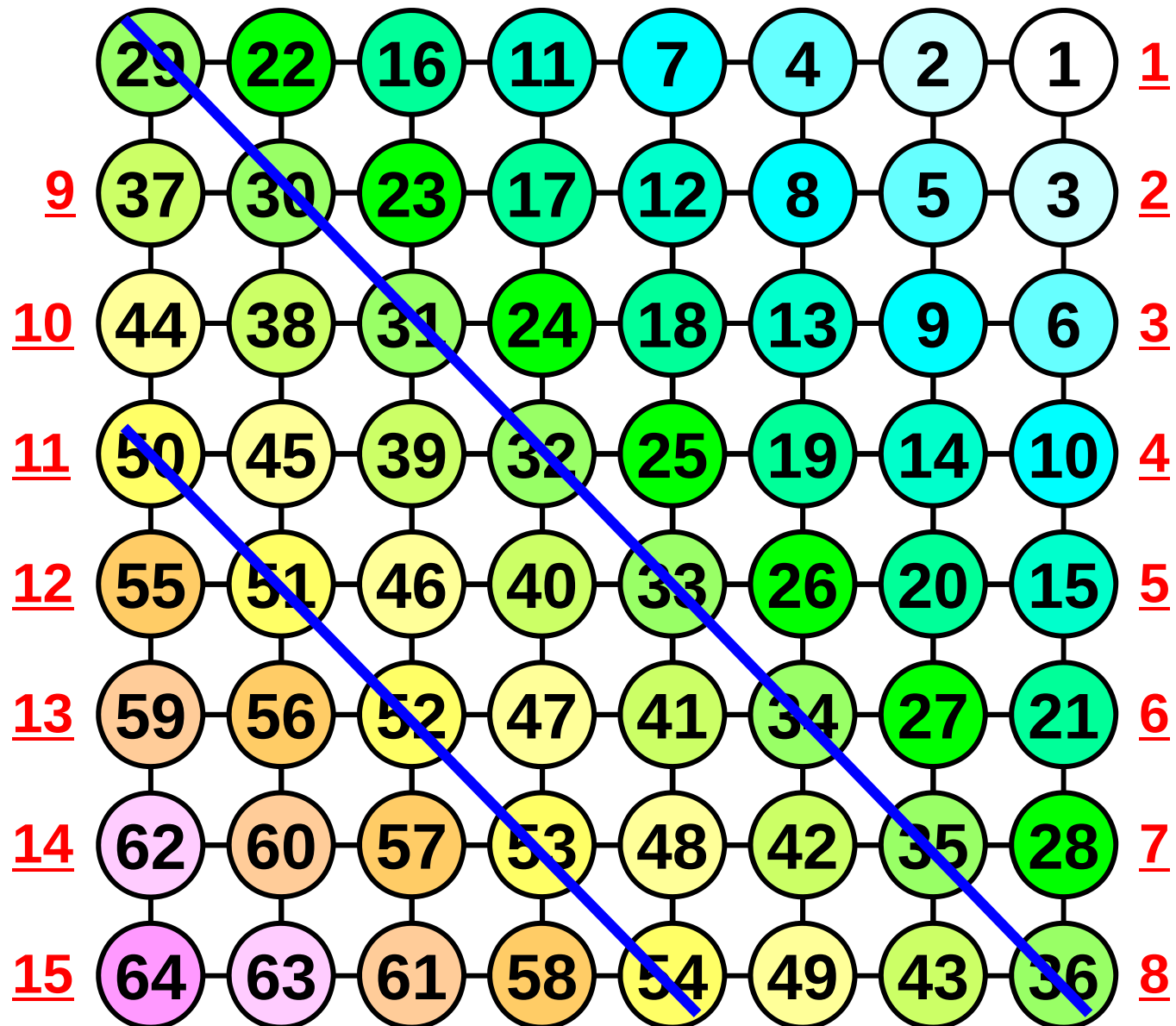
16コアにおける結果(●:MC, △:RCM)



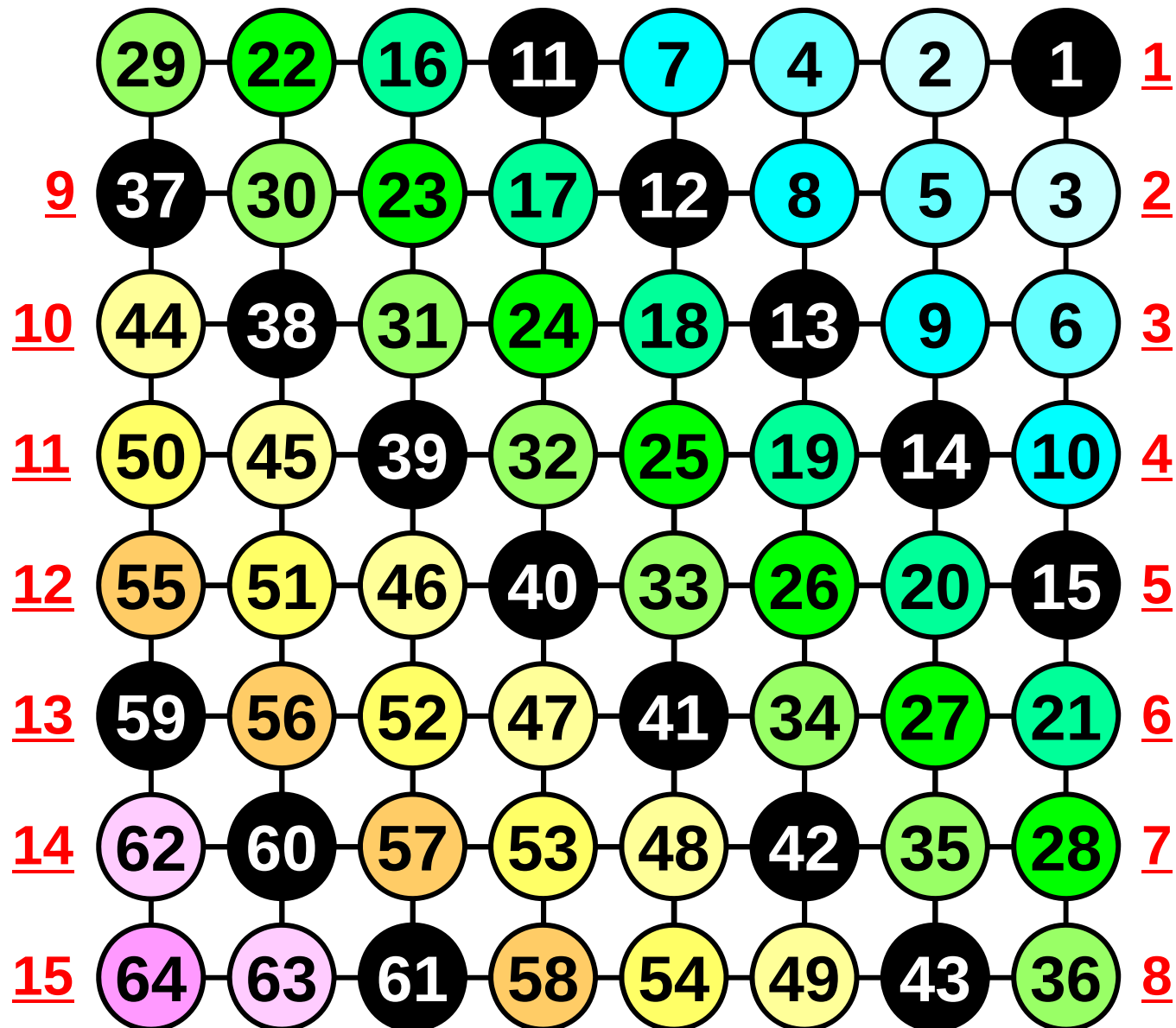
解決策: CM-RCM

- RCM + Cyclic Multicoloring [土肥, 襲田, 鷺尾他]
- 手順
 - まずRCMを施す
 - Cyclic Multicoloring (CM) の色数を決める(N_c)
 - RCMの1番目, (N_c+1)番目, ($2N_c+1$)番目...のレベルに属する要素を「1」色に分類する
 - RCMの k 番目, (N_c+k)番目, ($2N_c+k$)番目...のレベルに属する要素を「 k 」色に分類する
 - 「 k 」が「 N_c 」に達して, 要素が「1~ N_c 」で色付けされたら完了
 - あとはMCのときと同じように, 色の順番に再番号付
 - RCMの各レベルに対して「 N_c 」のサイクルで再色付けを実施している
 - もし同じ色の要素の中に依存性が見つかったら, $N_c=N_c+1$ として最初からやり直し(ここは少し原始的)

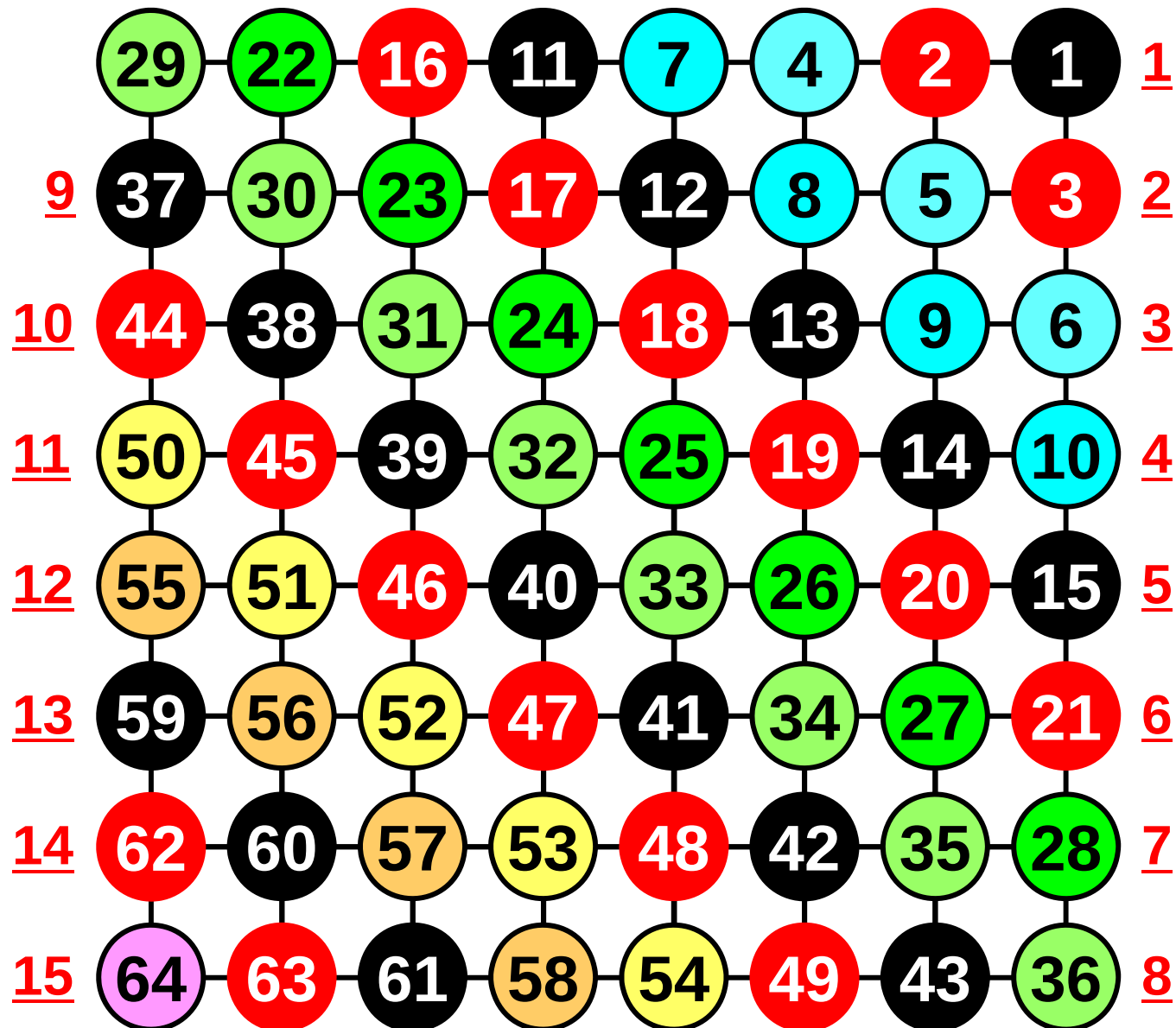
RCM



$N_c=4$, $k=1:1,5,9,13$ レベルを選択

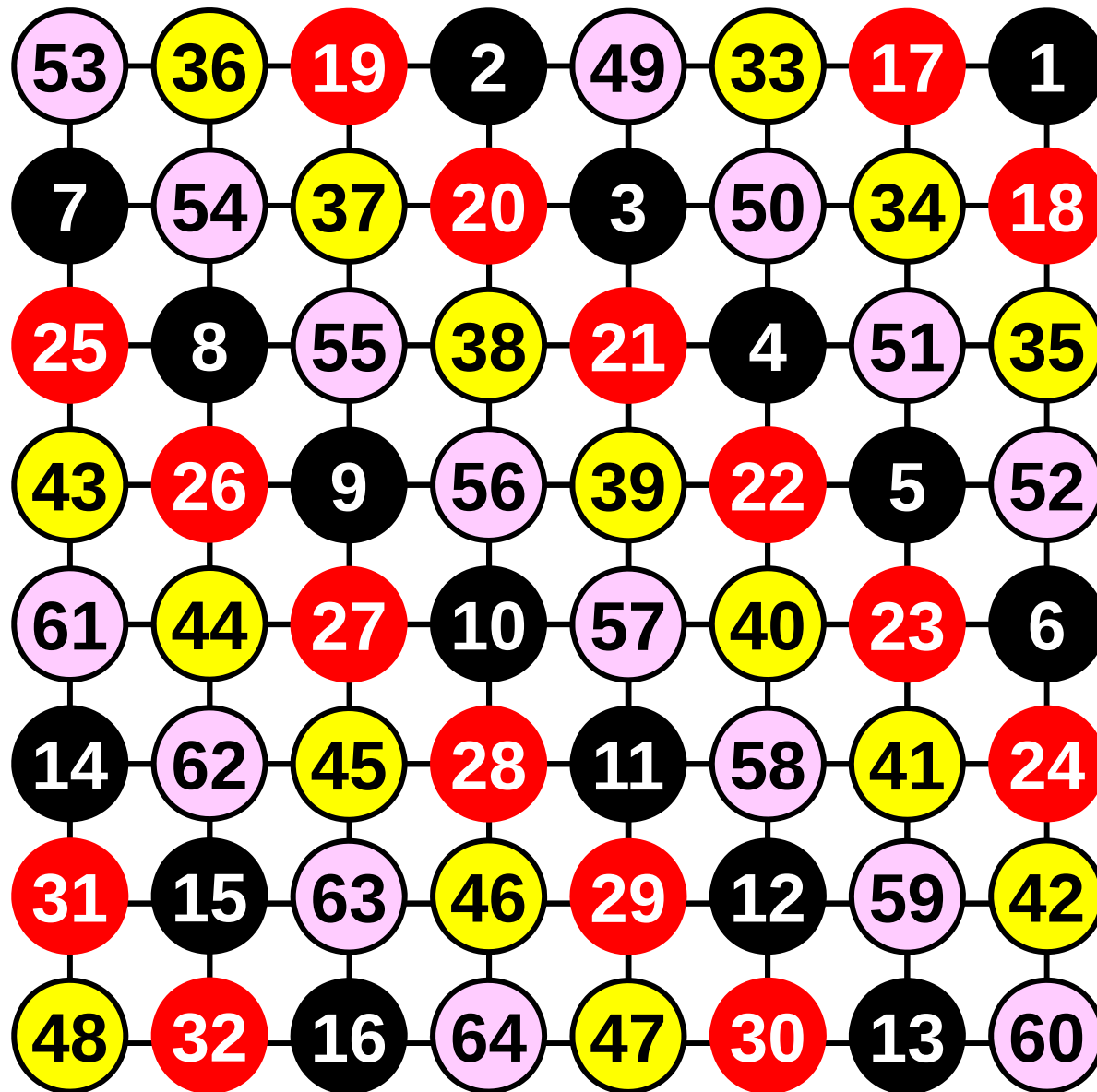


$N_c=4$, $k=2:2,6,10,14$ レベルを選択

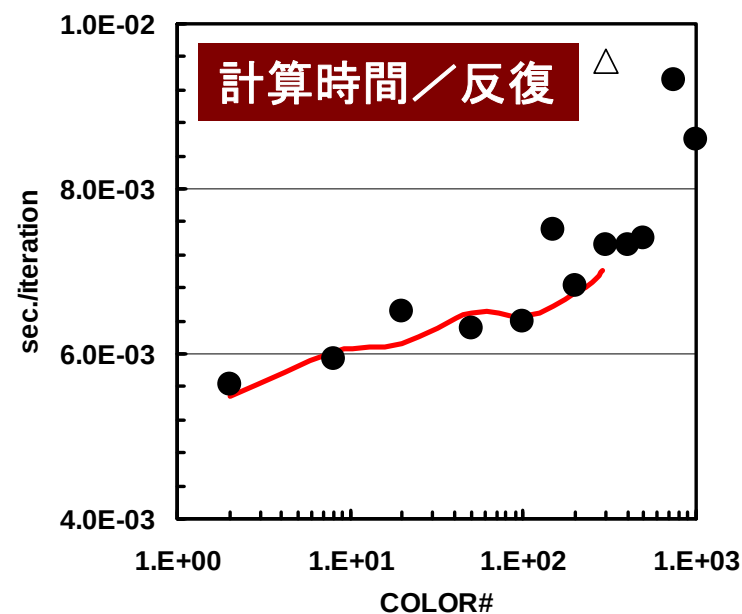
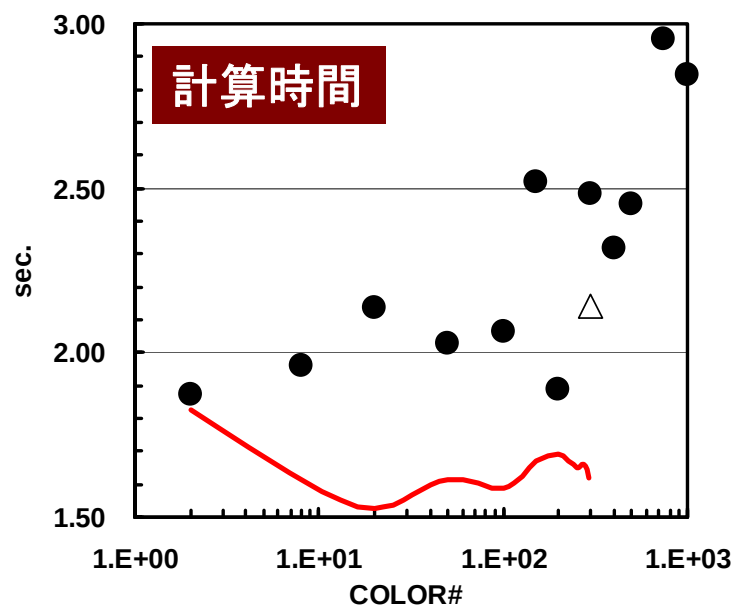
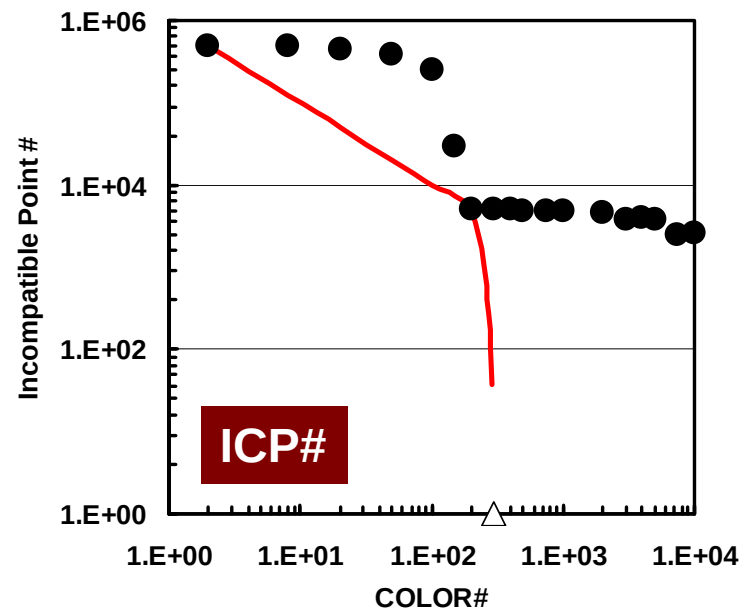
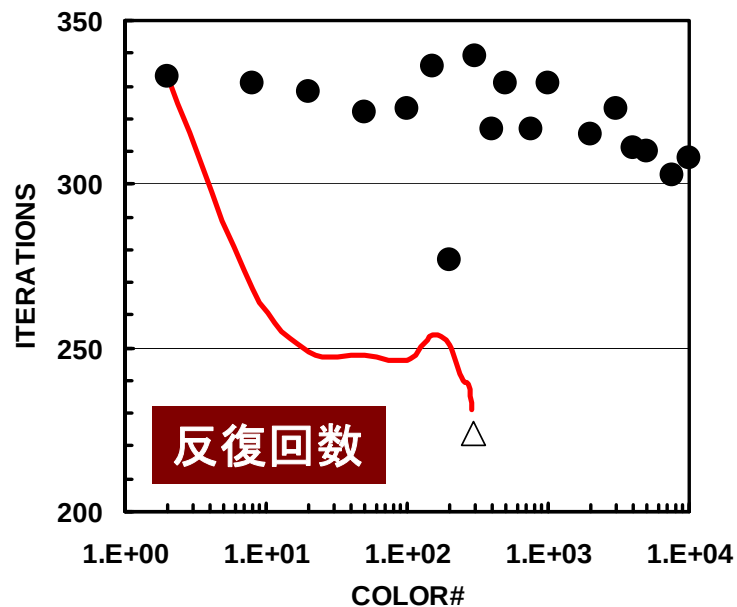


CM-RCM ($N_c=4$): 「色」の順番に再並替

k=1: 16
k=2: 16
k=3: 16
k=4: 16

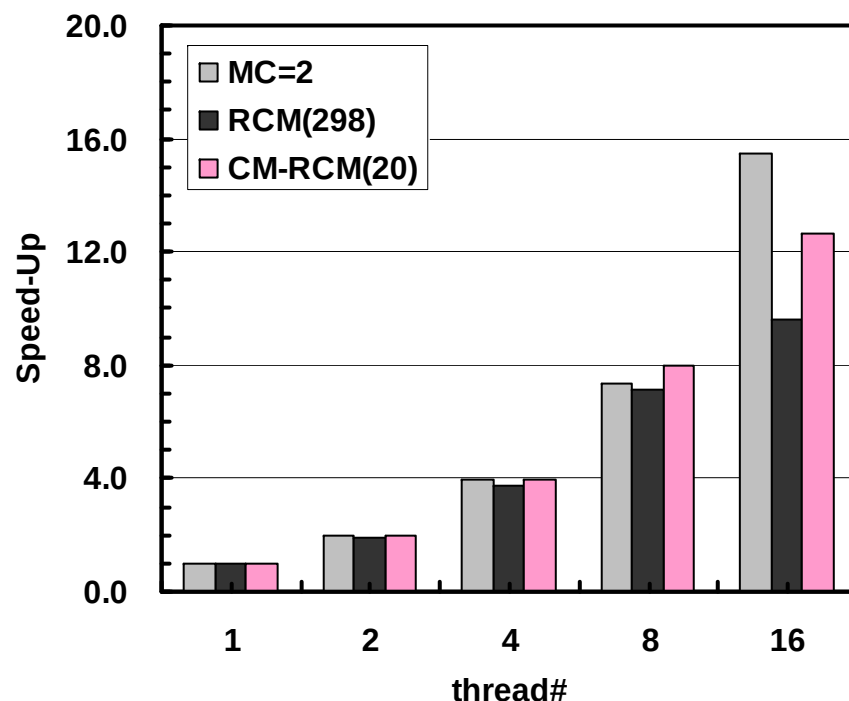
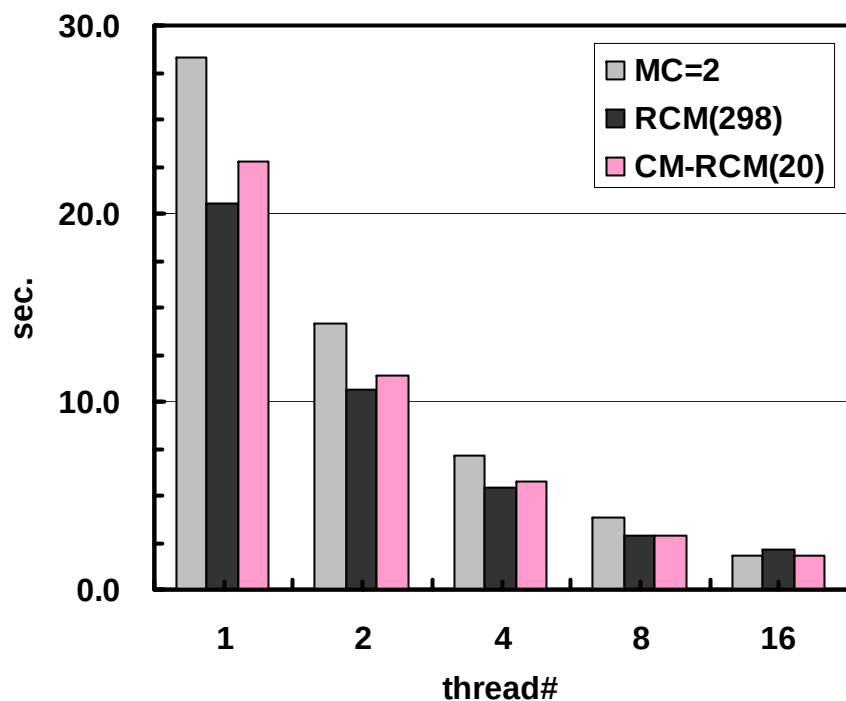


16コアにおける結果 (●: MC, △: RCM)



スケーラビリティ

反復回数: MC(2色): 333回, RCM(298レベル): 224回
CM-RCM($N_c=20$): 249回



16 threads

MC(2): 1.83 sec.

CM-RCM(20): 1.79 sec.

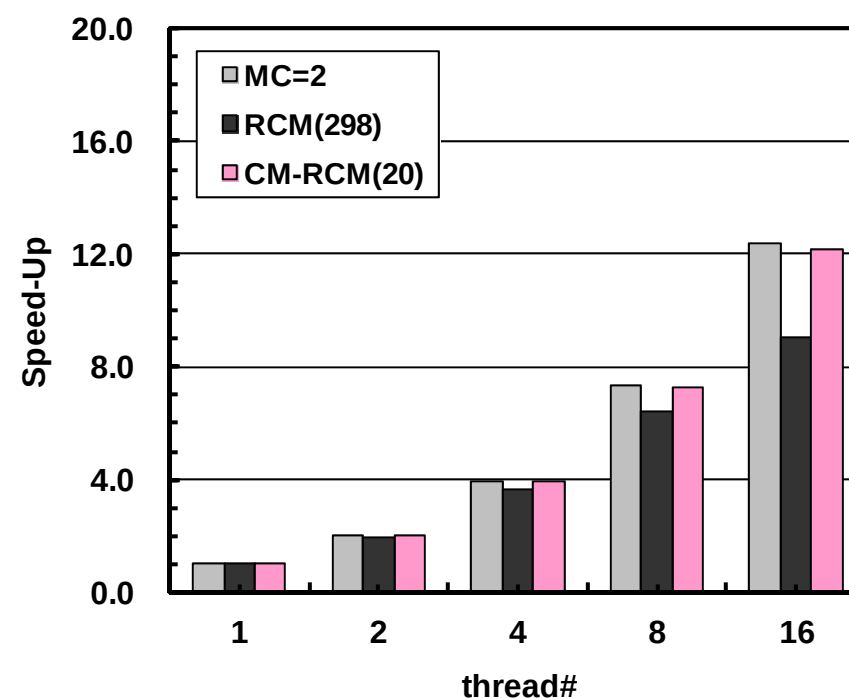
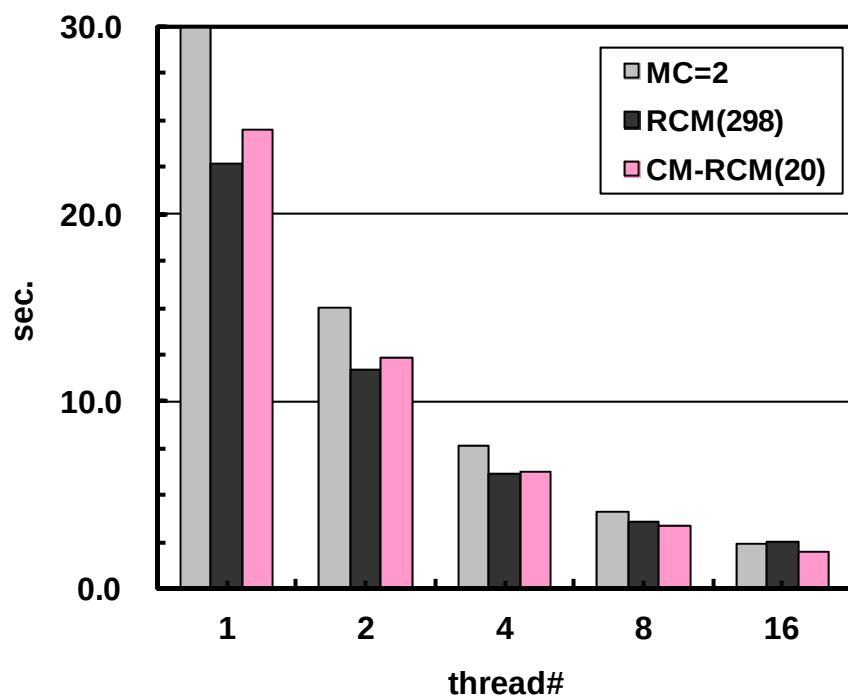
CM-RCM

- 少ない色数(N_c)で良好な収束を得られる
- 計算効率も良い
- 実行方法
 - INPUT.DATで「**NCOLORtot=-Nc**」とする
 - L2, L3で有効なオプション(既にL2でも使っていた)
- 実装は「cmrcm.f」を参照ください

- L2-solへのOpenMPの実装
- Hitachi SR11000/J2での実行
 - 計算結果
 - CM-RCMオーダリング
- **FX10での実行**

計算結果(FX10@東大): 10^6 要素

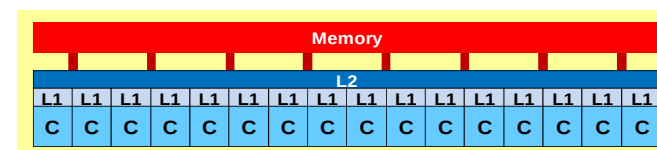
反復回数: MC(2色): 333回, RCM(298レベル): 224回
CM-RCM($N_c=20$): 249回



16 threads

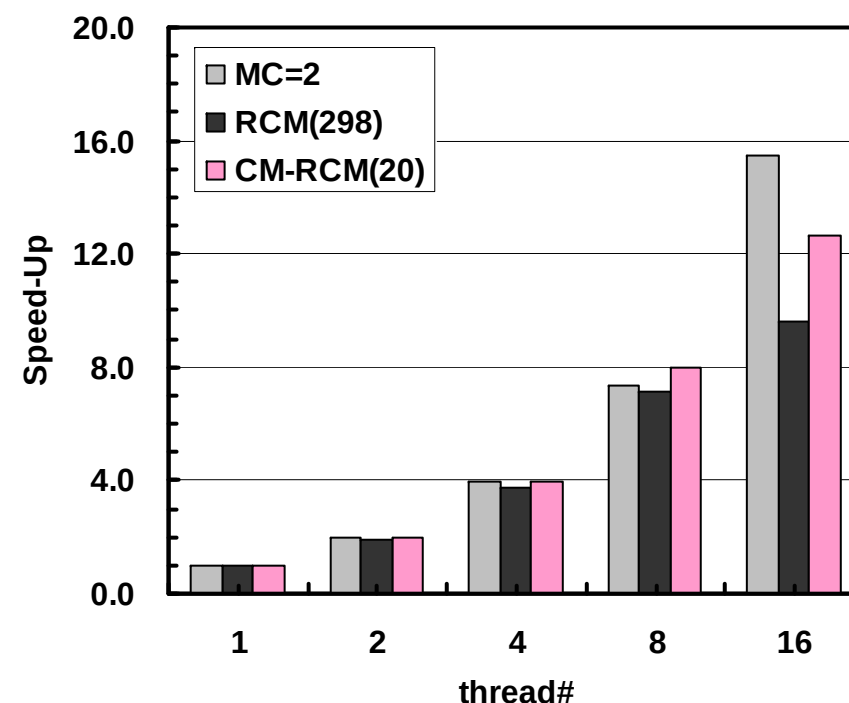
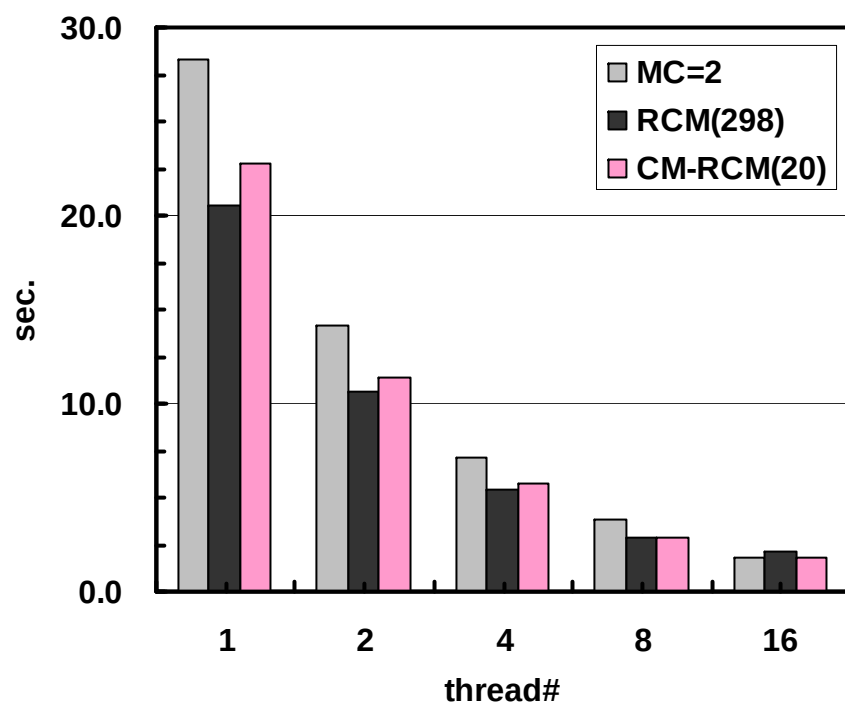
MC(2): 2.42 sec.

CM-RCM(20): 2.01 sec.



計算結果(SR11000): 10^6 要素

反復回数: MC(2色): 333回, RCM(298レベル): 224回
CM-RCM($N_c=20$): 249回



16 threads

MC(2): 1.83 sec.

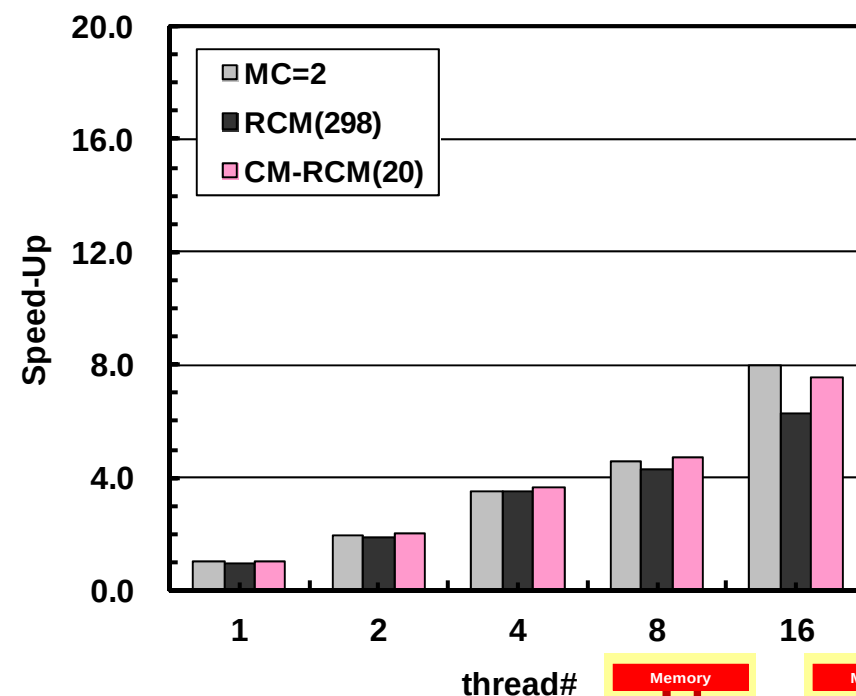
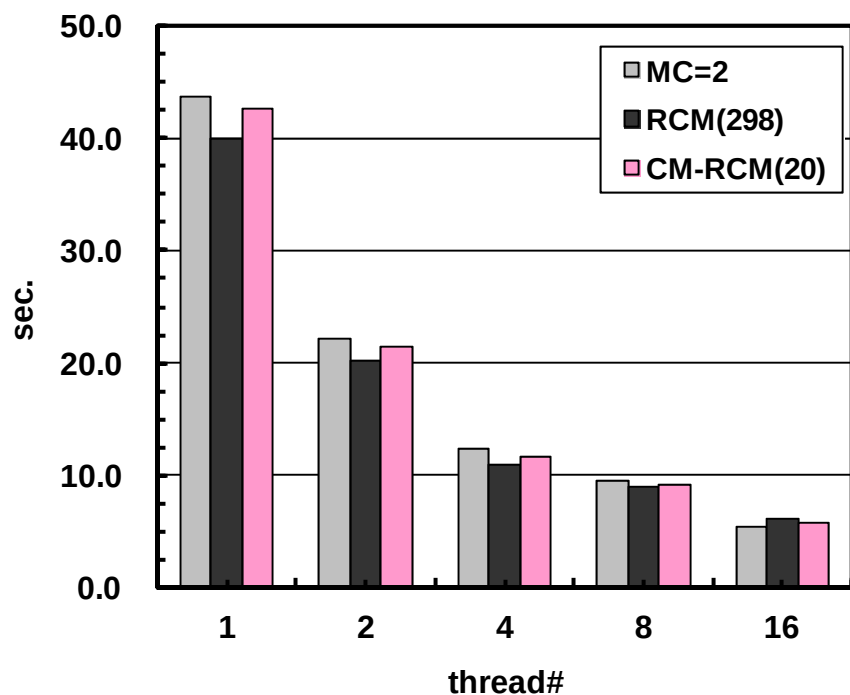
CM-RCM(20): 1.79 sec.

計算結果(T2K東大): 10^6 要素

反復回数: MC(2色): 333回, RCM(298レベル): 224回

CM-RCM($N_c=20$): 249回

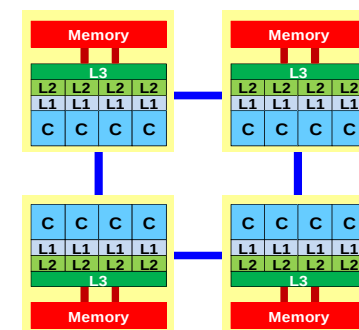
ccNUMA向け最適化



16 threads

MC(2): 5.48 sec.

CM-RCM(20): 5.71 sec.



GeoFEM Benchmark

ICCG法の性能(固体力学向け)

	SR11K/J2	SR16K/M1	T2K	FX10	京
Core #/Node	16	32	16	16	8
Peak Performance (GFLOPS)	147.2	980.5	147.2	236.5	128.0
STREAM Triad (GB/s)	101.0	264.2	20.0	64.7	43.3
B/F	0.686	0.269	0.136	0.274	0.338
GeoFEM (GFLOPS)	19.0	72.7	4.69	16.0	11.0
% to Peak	12.9	7.41	3.18	6.77	8.59
LLC/core (MB)	18.0	4.00	2.00	0.75	0.75

疎行列ソルバー: Memory-Bound

何故16倍にならないか？

- メモリ競合
- 16スレッドがメモリにアクセスすると, 1スレッドの場合と比較して, スレッド当り(コア当り)メモリ性能は低下
- 疎行列はmemory-boundなためその傾向がより顕著
 - 疎行列計算の高速化: 研究途上の課題
- 問題規模が比較的小さい

STREAM/Triadの結果(既出)

Thread #	MB/sec.	Speed-up
1	8606.14	1.00
2	16918.81	1.97
4	34170.72	3.97
8	59505.92	6.91
16	64714.32	7.52

まとめ

- 「有限体積法から導かれる疎行列を対象としたICCG法」を題材とした, データ配置, reorderingなど, 科学技術計算のためのマルチコアプログラミングにおいて重要なアルゴリズムについての講習
- 更に理解を深めるための, FX10を利用した実習
- オーダリングの効果

今後の動向

- メモリバンド幅と性能のギャップ
 - BYTE/FLOP, 中々縮まらない
- マルチコア化, メニーコア化
- $>10^5$ コアのシステム
 - Exascale: $>10^8$
- オーダリング
 - グラフ情報だけでなく, 行列成分の大きさの考慮も必要か?
 - 最適な色数の選択: 研究課題 (特に悪条件問題)
- OpenMP+MPIのハイブリッド⇒一つの有力な選択
 - プロセス内 (OpenMP) の最適化が最もcritical
- 本講義の内容が少しでも役に立てば幸いである

補足・実習

- 更なる最適化
- コンパイルオプション
 - FORTRANのみ
 - コンパイルリストの分析
- 実行

前進代入：現状の並列化(Fortran)

```
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
```

- 「!omp parallel」でスレッド(～16)の生成, 消滅が発生
 - 色ごとにこの部分を通る
 - 多少のオーバーヘッドがある
- 色数が増えるとオーバーヘッドが増す

前進代入:現状の並列化(C)

```
for(ic=0; ic<NCOLORtot; ic++) {  
  #pragma omp parallel for private (ip, ip1, i, WVAL, j)  
    for(ip=0; ip<PEsmpTOT; ip++) {  
      ip1 = ic * PEsmpTOT + ip;  
      for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++){  
        WVAL = W[Z][i];  
        for(j=indexL[i]; j<indexL[i+1]; j++){  
          WVAL -= AL[j] * W[Z][itemL[j]-1];  
        }  
        W[Z][i] = WVAL * W[DD][i];  
      }  
    }  
}
```

- 「#pragma omp parallel」でスレッド(～16)の生成, 消滅が発生
 - 色ごとにこの部分を通る
 - 多少のオーバーヘッドがある
- 色数が増えるとオーバーヘッドが増す

前進代入: Overhead削減 (Fortran)

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

- このようにすることによって, スレッド生成を前進代入に入る前の一回で済ませることができる
- 「!omp do」のループが並列化

前進代入: Overhead削減(C)

```
#pragma omp parallel private (ip, ip1, i, WVAL, j)
for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp for
    for(ip=0; ip<PEsmpTOT; ip++) {
        ip1 = ic * PEsmpTOT + ip;
        for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++){
            WVAL = W[Z][i];
            for(j=indexL[i]; j<indexL[i+1]; j++){
                WVAL -= AL[j] * W[Z][itemL[j]-1];
            }
            W[Z][i] = WVAL * W[DD][i];
        }
    }
}
```

- このようにすることによって, スレッド生成を前進代入に入る前の一回で済ませることができる
- 「#pragma omp for」のループが並列化

プログラム類

```
% cd <$0-L3>
% ls
    run    src    src0    src    short (Fortranのみ)

% cd src0

% make
% cd ../run
% ls L3-sol0
    L3-sol0

% <modify "INPUT.DAT">
% <modify "go0.sh">

% pjsub go0.sh
```

計算結果：色数が多いと少し速い

$N=128^3$

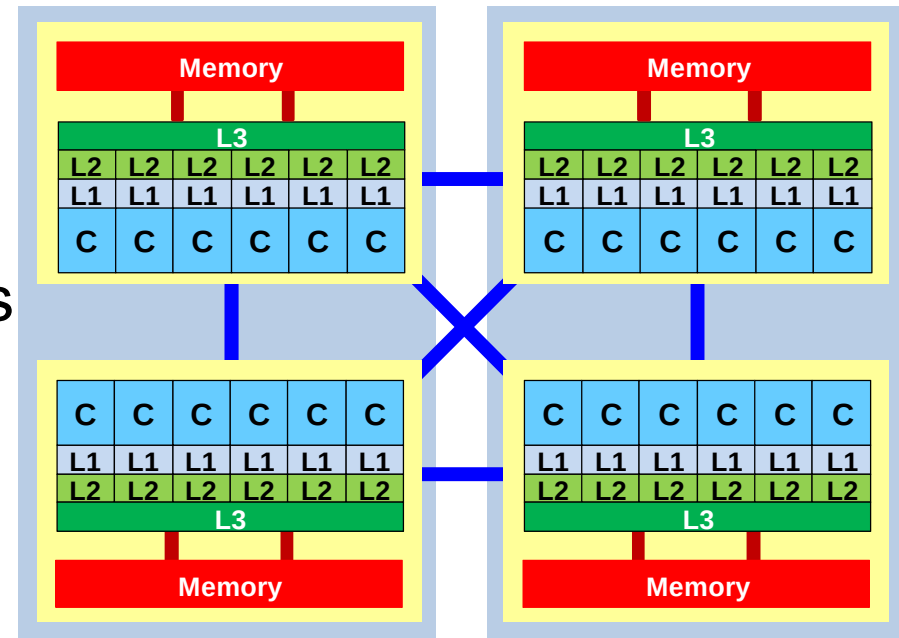
	L3-sol	L3-sol-0
NCOLORtot= -20 CM-RCM (20) 318 Iterations	5.69 sec.	5.67 sec.
NCOLORtot= -1 RCM (382 levels) 287 Iterations	6.54 sec.	6.38 sec.

- 更なる最適化
- コンパイルオプション
 - FORTRANのみ
 - コンパイルリストの分析
- 実行

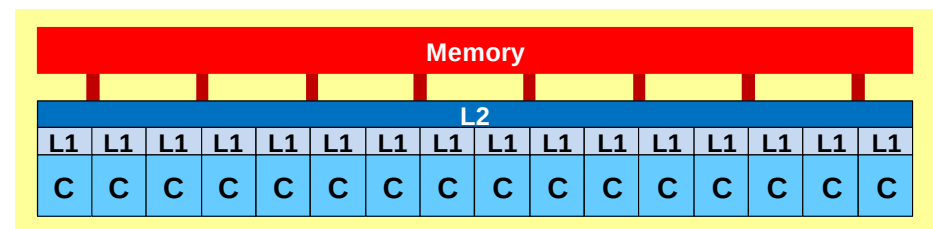
ccNUMA Architecture

- Multicore-Multisocket
- 各ソケットがローカルにメモリを持っている
 - cc-NUMA: cache-coherent-Non-Uniform Memory Access
 - ローカルのメモリをアクセスして計算するようなプログラミング, データ配置, 実行時制御 (numactl) が必要
 - T2K, Cray XE6 etc.
- 特殊な最適化
 - First-Touch Data Placement
 - Sequential Reordering
- 京, FX10 → Flat, UMA

Cray XE6 (Hopper)



Fujitsu FX10 (Oakleaf-FX)



First Touch Data Placement

配列のメモリ・ページ:

最初にtouchしたコアのローカルメモリ上に確保

計算と同じ順番で初期化

```

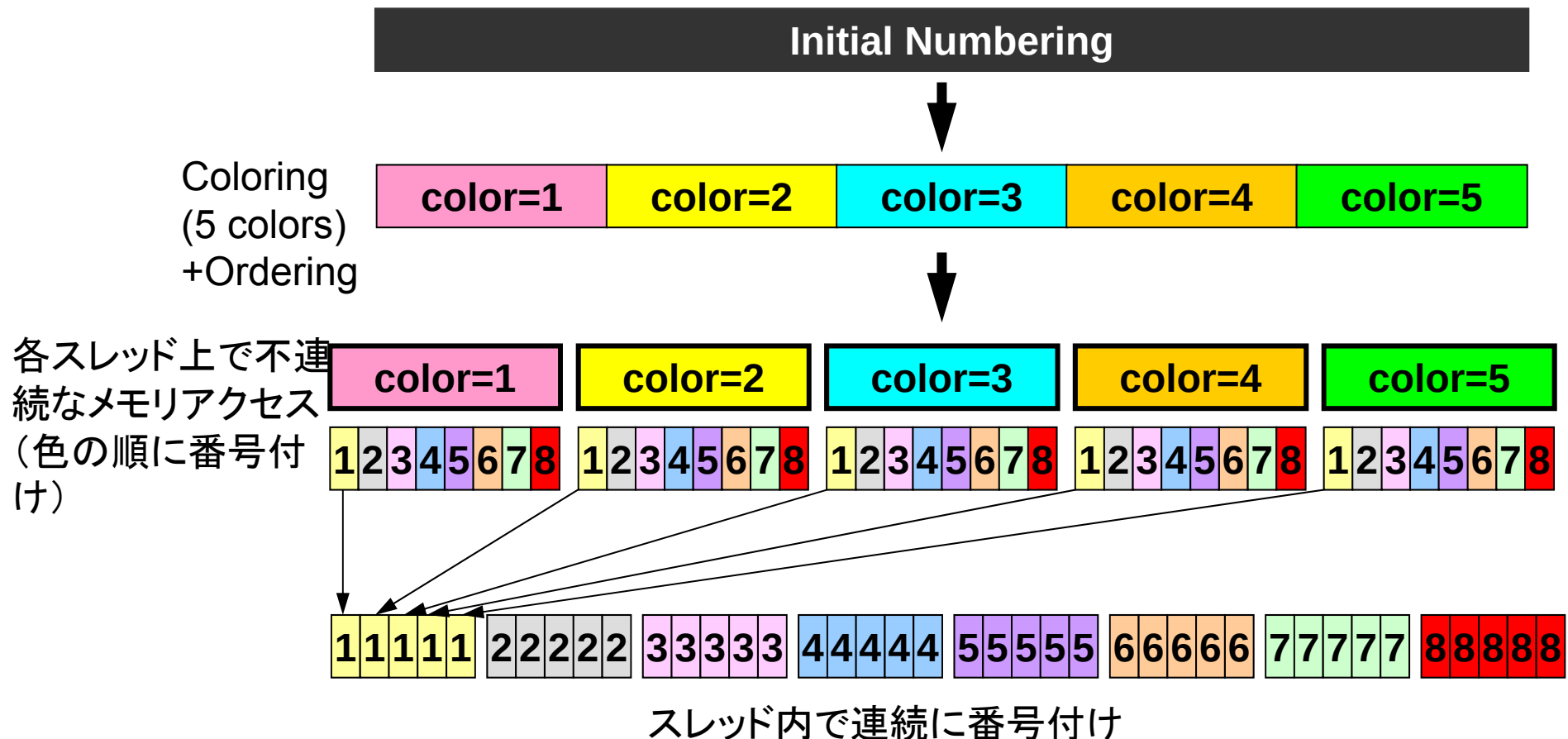
do lev= 1, LEVELtot
  do ic= 1, COLortot(lev)
!$omp parallel do private(ip,i,j,isL,ieL,isU,ieU)
    do ip= 1, PESmpTOT
      do i = STACKmc(ip,ic-1,lev)+1, STACKmc(ip,ic,lev)
        RHS(i)= 0.d0; X(i)= 0.d0; D(i)= 0.d0

        isL= indexL(i-1)+1
        ieL= indexL(i)
        do j= isL, ieL
          itemL(j)= 0; AL(j)= 0.d0
        enddo

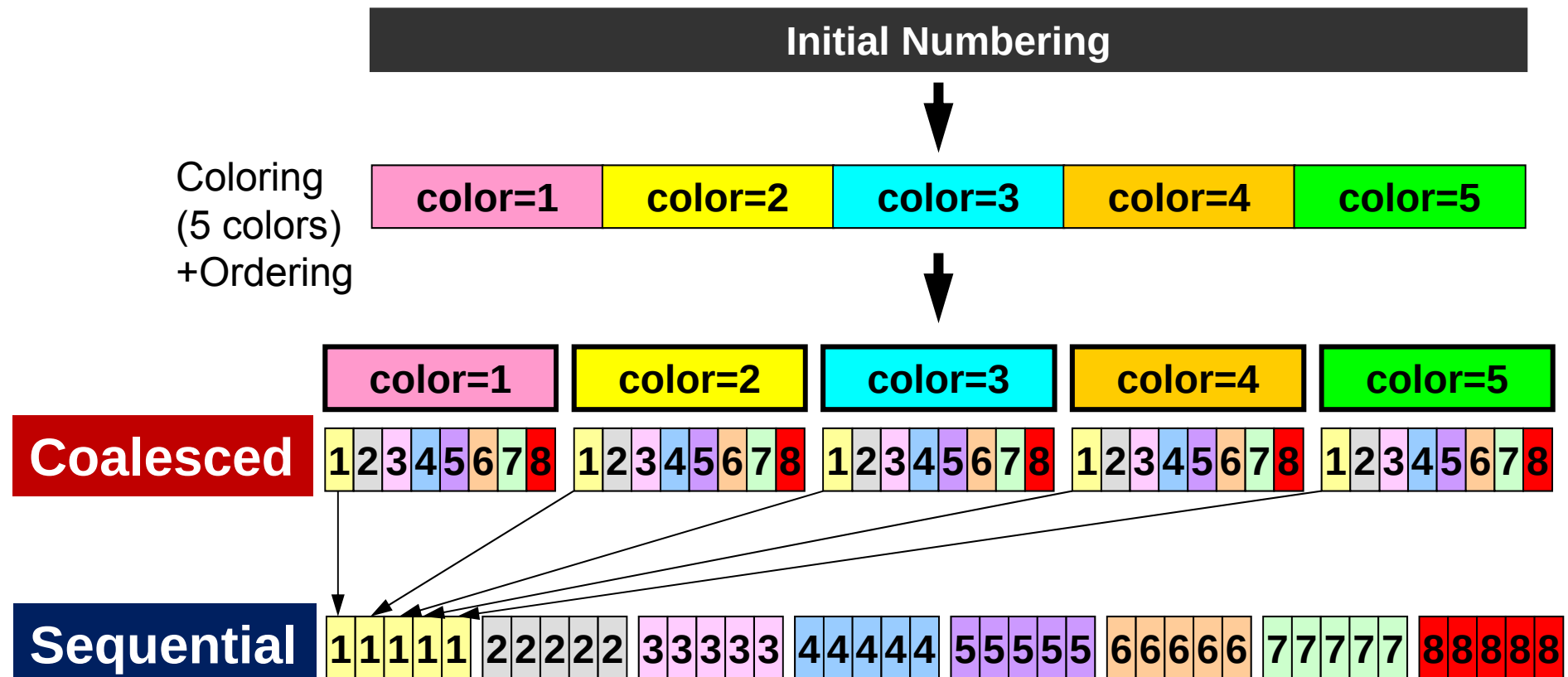
        isU= indexU(i-1)+1
        ieU= indexU(i)
        do j= isU, ieU
          itemU(j)= 0; AU(j)= 0.d0
        enddo
      enddo
    enddo
  enddo
!$omp omp end parallel do
enddo
enddo

```

各スレッド上でメモリアクセスが連続となる ように更なる並び替え, 「sequential」 5 colors, 8 threads



各スレッド上でメモリアクセスが連続となる ように更なる並び替え, 「sequential」 5 colors, 8 threads



Hopper & Oakleaf-FX

	Cray XE6 Hopper	FX10 Oakleaf-FX
Peak Performance/core (GFLOPS)	8.40	14.78
Core #/Node	24	16
Peak Performance/node (GFLOPS)	201.6	236.5
Peak Memory Bandwidth/node (GB/sec)	85.3 8xDDR3 1333MHz	85.3 8xDDR3 1333MHz
Triad Performance/node (GB/sec)	52.3	64.7
Triad Performance/core (GB/sec)	2.18	4.04
B/F Rate	0.260	0.274
GeoFEM Bench/ICCG (MFLOPS/core)	469.8	1011.3
GeoFEM Bench/ICCG (GFLOPS/node)	11.28	16.18

Optimizations for ccNUMA are not effective on Oakleaf-FX

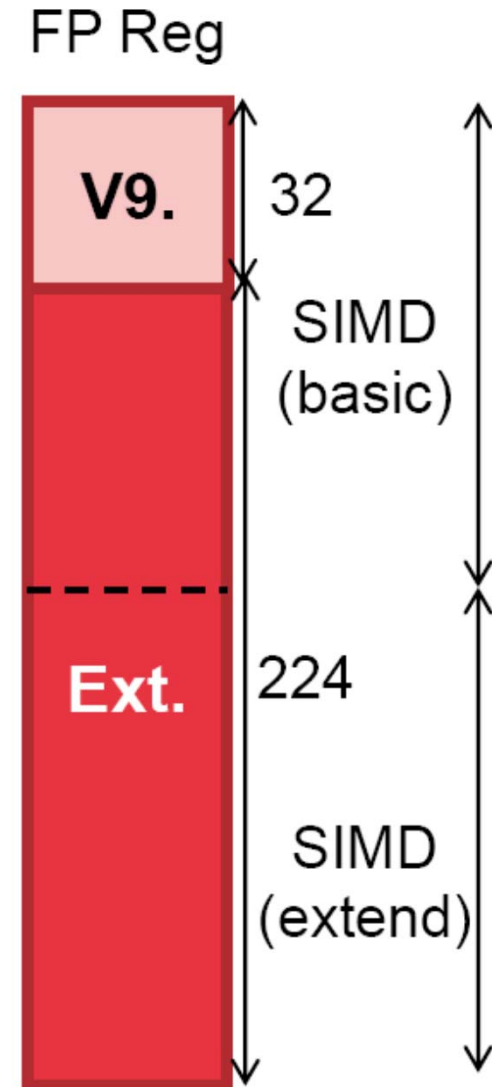
Poisson's Equations, ICCG/CM-RCM(2), 128^3 unknowns
Time for ICCG solver until convergence (318 iter's).

	Cray XE6 (Hopper)	Oakleaf-FX (not optimum compiler options)
Original	18.1 sec	5.62 sec
+ First Touch	7.81 sec	5.94 sec
+ Sequential Reordering	5.04 sec	5.84 sec

HPC-ACE

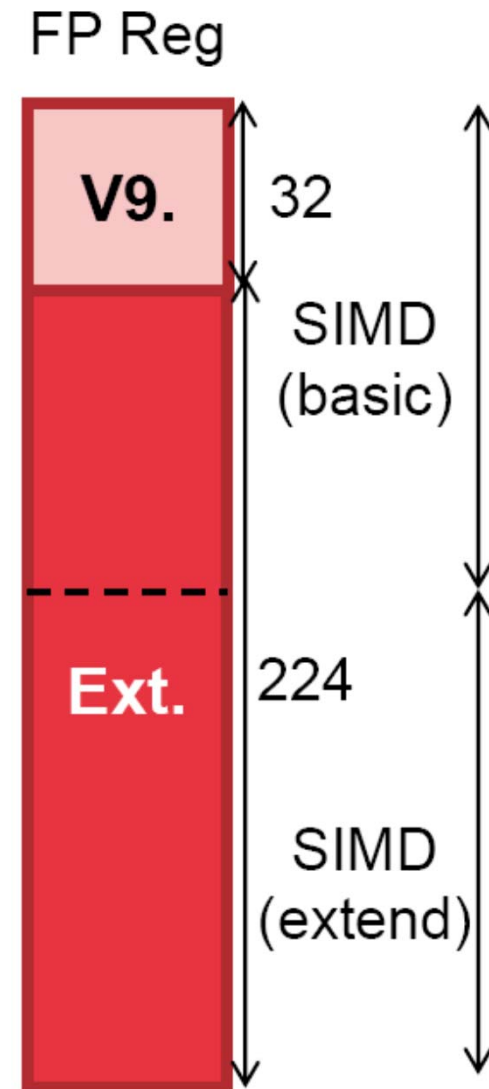
High Performance Computing – Arithmetic Computational Extensions

- Enhanced instruction set for the SPARC-V9 instruction set arch.
 - High-Performance & Power-Aware
- Extended Number of Registers
 - FP Registers: 32→256
- S/W Controllable Cache
 - “Sector Cache”
 - for keeping reusable data sets on cache
- High-Performance, Efficient
 - Optimized FP functions
 - Conditional Operation

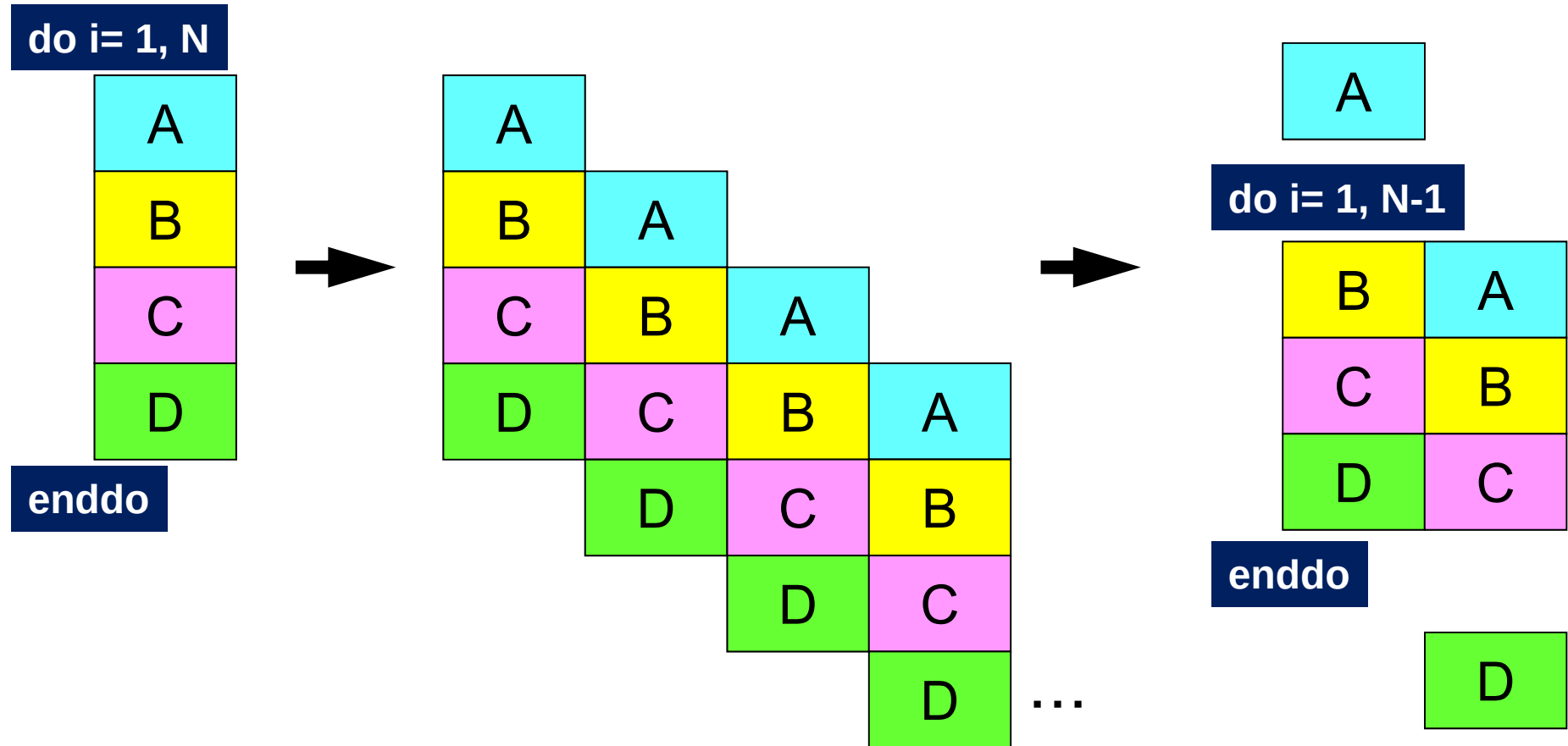


SPARC64™ IXfx

- HPC-ACE (High Performance Computing – Arithmetic Computational Extensions)
 - Extended number of registers
 - FP Registers: 32→256
 - Software Pipelining is useful
 - S/W controllable “sector” cache
- UMA, not NUMA
- H/W barrier for high-speed synchronization of on-chip cores



Pipelining + SIMD

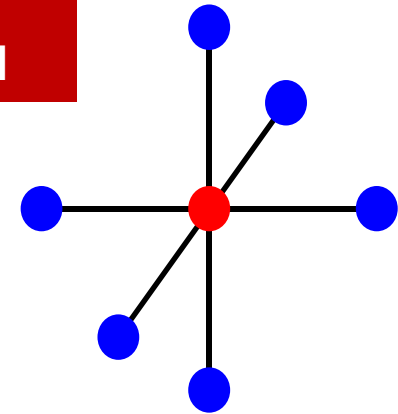


もしこのような実行が可能であれば...

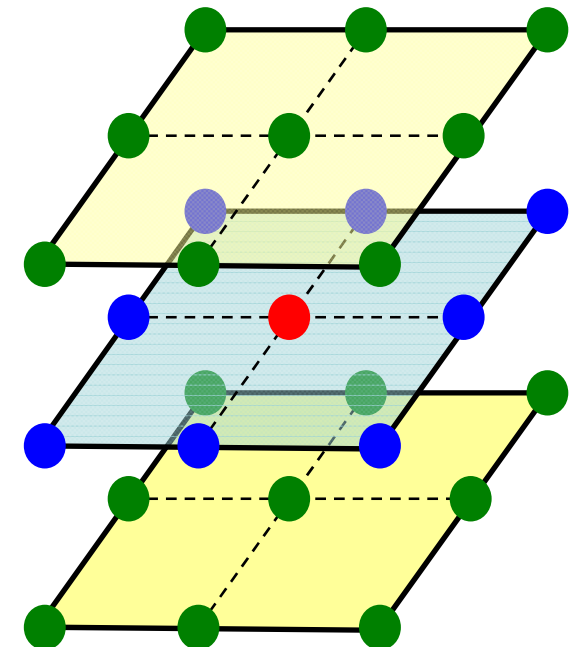
現行コードの問題

- 最内側ループの回転数(ループ長)が短い
 - いわゆる7点ステンシル, 最大でも6
 - 大小関係を考慮する上下三角成分では色数が少ない場合は6に近いが, 色数が増えると3程度
 - Software Pipeliningが効きにくい
- 有限要素法(GeoFEM Benchmark)だと性能が出やすい
 - 27点ステンシル

7-pt.
Stencil



27-pt. Stencil



Hopper & Oakleaf-FX: 27-pt. stencil

	Hopper	Oakleaf FX
Peak Performance/core (GFLOPS)	8.40	14.78
Core #/Node	24	16
Peak Performance/node (GFLOPS)	201.6	236.5
Peak Memory Bandwidth/node (GB/sec)	85.3 8xDDR3 1333MHz	85.3 8xDDR3 1333MHz
Triad Performance/node (GB/sec)	52.3	64.7
Triad Performance/core (GB/sec)	2.18	4.04
B/F Rate	0.260	0.274
GeoFEM Bench/ICCG (GFLOPS/node)	11.28	16.18
GeoFEM Bench/ICCG (MFLOPS/core)	469.8	1011.3

最内ループ回転数が少ない場合

- コンパイラ: 専用オプション (Fortranのみ)
 - -Kshortloop= N (N:2-10), -Knoshortloop (default)
- 最内側ループの強制Unrolling (N/2段)
- 以下は抑止
 - ソフトウェアパイプライニング
 - ループブロッキング
 - ループストライピング
- 回転数が10より大きいときに指定すると性能が低下する場合あり
- 結果 (128³, CM-RCM(20)) on FX10@東大
 - Default: 5.62 sec.
 - Kshortloop: 5.39 sec. (N=2), 5.39 (N=3), 5.61 (N=4)
 - それでもCray XE6より遅い

デフォルト

```
>$ cd <$0-L3>/src
>$ make
>$ ls ../run/L3-sol
    L3-sol
>$ cd ../run
>$ pjsub go1.sh
```

```
F90          = frtpx
F90OPTFLAGS= -Kfast,openmp -Qt
F90FLAGS    =$(F90OPTFLAGS)
```

Kshort loop

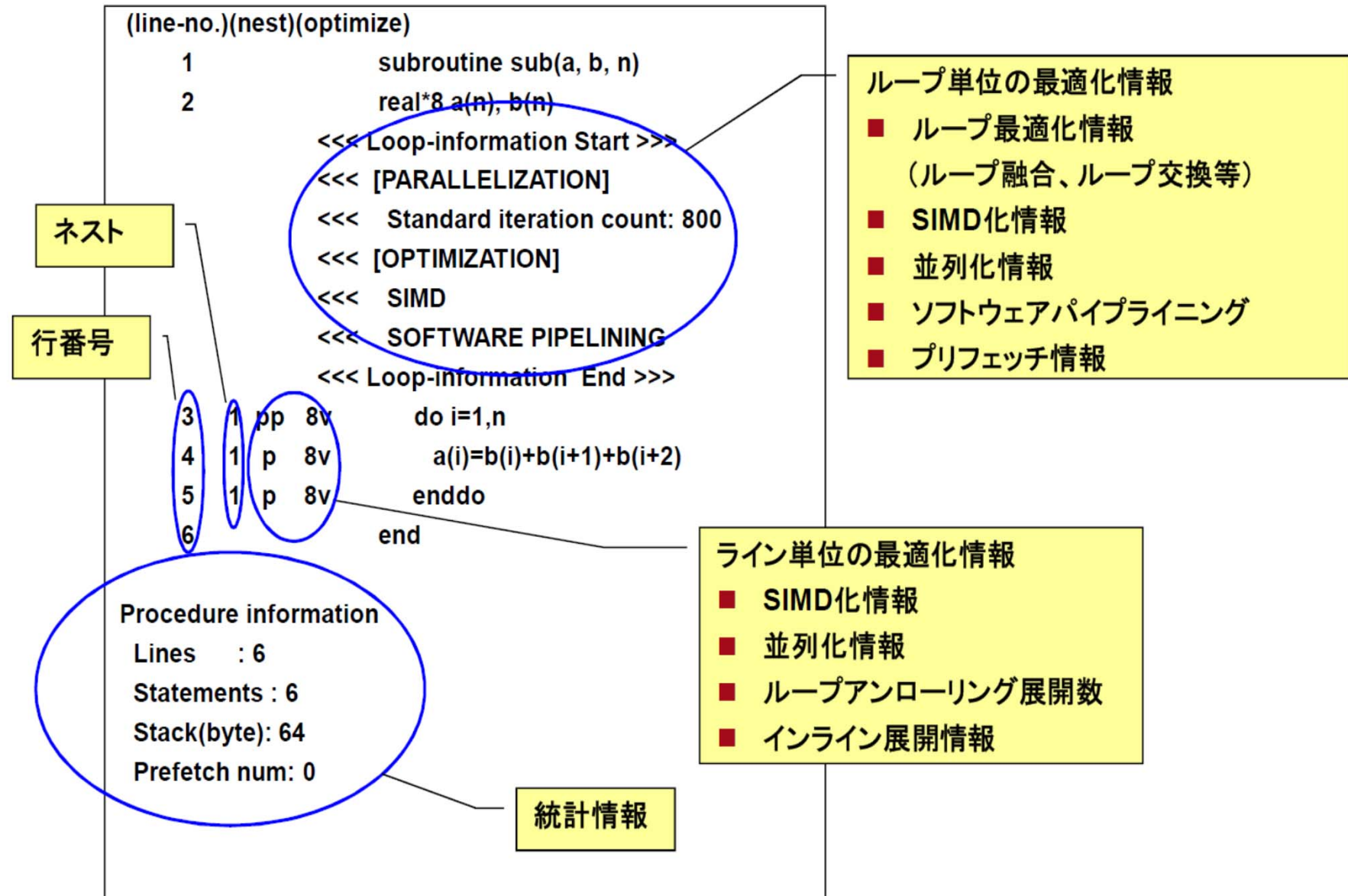
```
>$ cd <$0-L3>/short
>$ make
>$ ls ../run/L3-sols
    L3-sols
>$ cd ../run
>$ pjsub gos.sh
```

```
F90          = frtpx
F90OPTFLAGS= -Kfast,openmp -Kshortloop=3 -Qt
F90FLAGS    =$(F90OPTFLAGS)
```

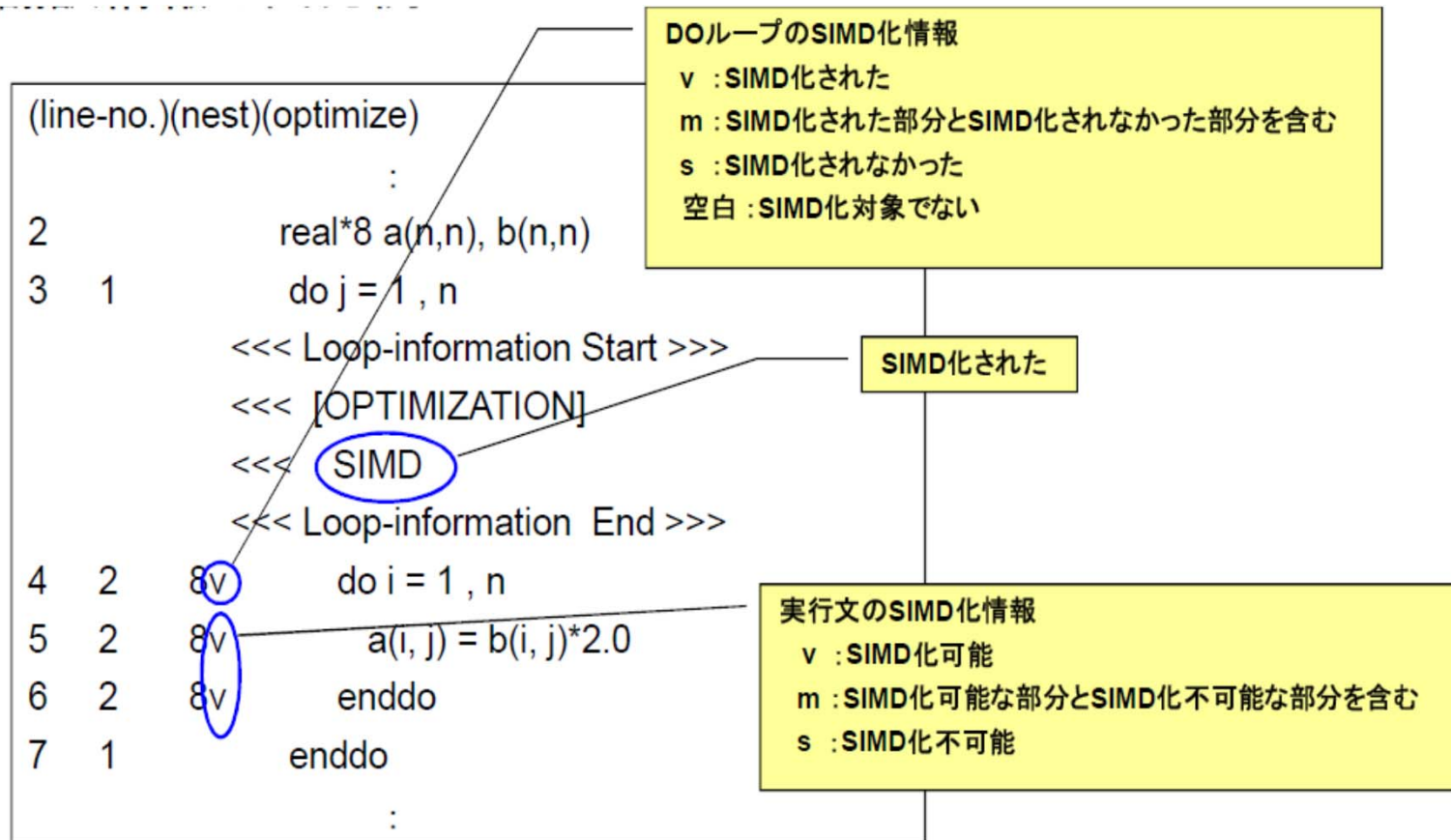
コンパイル・実行

- -Qt
 - コンパイルリスト出力
 - *.lst
- Cでは「-Qt」は使えません, 「-Nsrc」を使ってください。
 - 画面に出てしまいますが

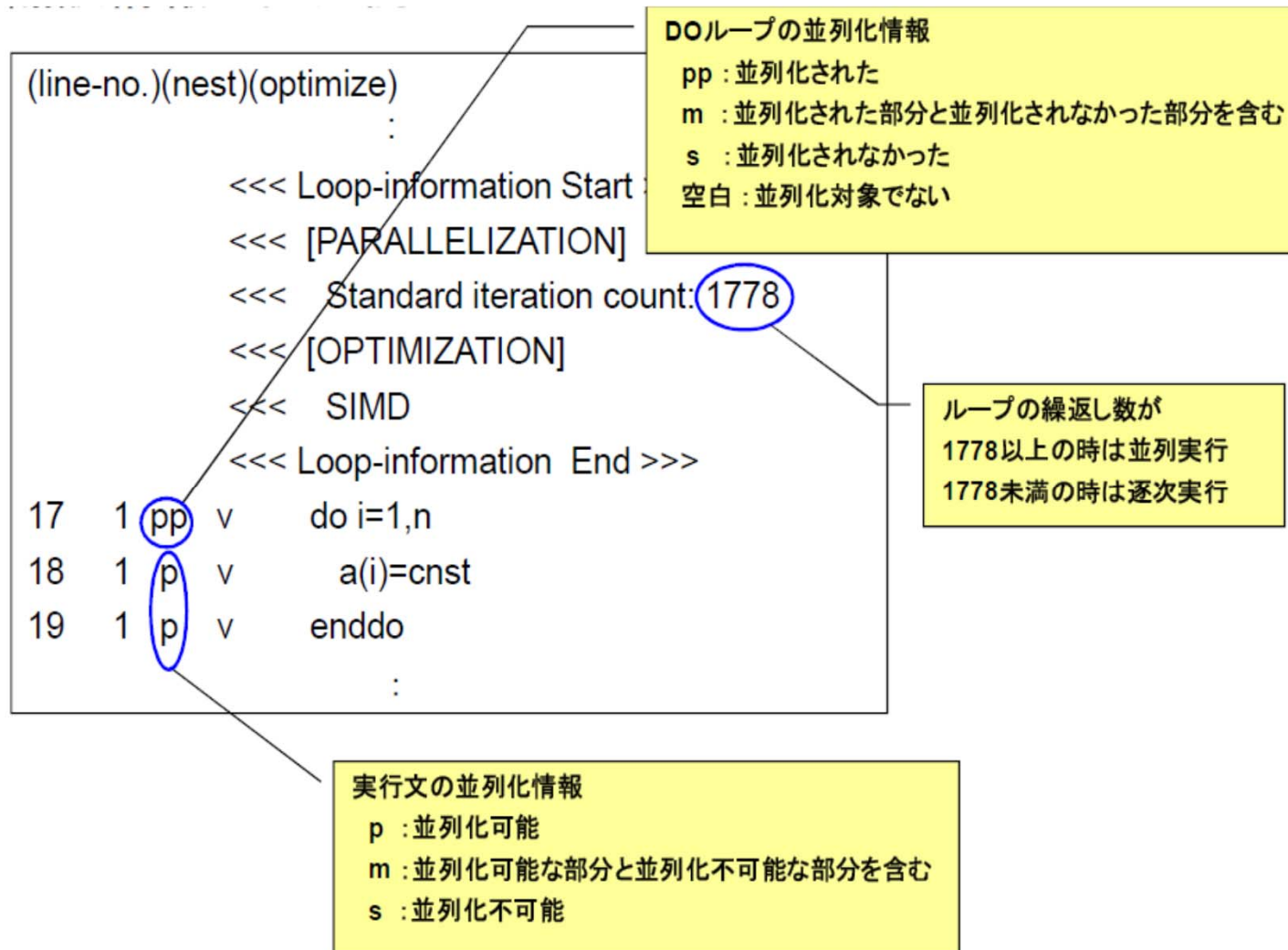
* .lstの見方



SIMD情報



自動並列化情報



solver_ICCG_mc.lst (src)

```

101      1      !C
102      1      !C +-----+
103      1      !C | {z}= [Minv]{r} |
104      1      !C +-----+
105      1      !C===
106      1
107      1      !$omp parallel do private(ip,i)
108      2      p      do ip= 1, PESmpTOT
<<< Loop-information Start >>>
<<< [OPTIMIZATION]
<<< SIMD
<<< SOFTWARE PIPELINING
<<< Loop-information End >>>
109      3      p      8v      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
110      3      p      8v      W(i,Z)= W(i,R)
111      3      p      8v      enddo
112      2      p      enddo
113      1      !$omp end parallel do
114      1
115      1      Stime= omp_get_wtime()
116      1      call fapp_start ("precond", 1, 1)
117      2      do ic= 1, NCOLRtot
118      2      !$omp parallel do private(ip,ip1,i,WVAL,k)
119      3      p      do ip= 1, PESmpTOT
120      3      p      ip1= (ic-1)*PESmpTOT + ip
121      4      p      do i= SMPindex(ip1-1)+1, SMPindex(ip1)
122      4      p      WVAL= W(i,Z)
<<< Loop-information Start >>>
<<< [OPTIMIZATION]
<<< SIMD
<<< SOFTWARE PIPELINING
<<< Loop-information End >>>
123      5      p      4v      do k= indexL(i-1)+1, indexL(i)
124      5      p      4v      WVAL= WVAL - AL(k) * W(itemL(k),Z)
125      5      p      4v      enddo
126      4      p      W(i,Z)= WVAL * W(i,DD)
127      4      p      enddo
128      3      p      enddo
129      2      !$omp end parallel do
130      2      enddo

```

solver_ICCG_mc.lst (short)

SOFTWARE PIPELININGが無い

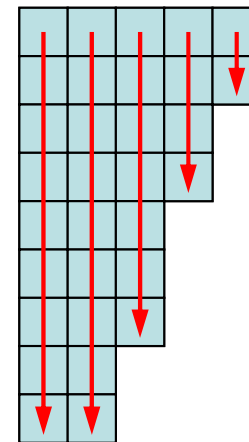
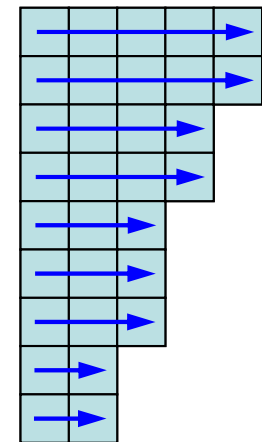
```

101      1      !C
102      1      !C +-----+
103      1      !C | {z}= [Minv]{r} |
104      1      !C +-----+
105      1      !C==
106      1
107      1      !$omp parallel do private(ip,i)
108      2      2      p      do ip= 1, PESmpTOT
<<< Loop-information Start >>>
<<< [OPTIMIZATION]
<<< SIMD
<<< Loop-information End >>>
109      3      3      p      v      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
110      3      3      p      v      w(i,Z)= w(i,R)
111      3      3      p      v      enddo
112      2      2      p      enddo
113      1      !$omp end parallel do
114      1
115      1      Stime= omp_get_wtime()
116      1      call fapp_start ("precond", 1, 1)
117      2      3
118      2      3      !$omp parallel do private(ip,ip1,i,WVAL,k)
119      3      3      p      3      do ip= 1, PESmpTOT
120      3      3      p      3      ip1= (ic-1)*PESmpTOT + ip
121      4      3      p      3      do i= SMPindex(ip1-1)+1, SMPindex(ip1)
122      4      3      p      3      WVAL= w(i,Z)
<<< Loop-information Start >>>
<<< [OPTIMIZATION]
<<< SIMD
<<< Loop-information End >>>
123      5      5      p      3v      do k= indexL(i-1)+1, indexL(i)
124      5      5      p      3v      WVAL= WVAL - AL(k) * w(itemL(k),Z)
125      5      5      p      3v      enddo
126      4      4      p      3      w(i,Z)= WVAL * w(i,DD)
127      4      4      p      3      enddo
128      3      3      p      3      enddo
129      2      2      3      !$omp end parallel do
130      2      2      3      enddo

```

現 状

- コンパイラに頼り過ぎたくない, しかも今は速くない
- 最内ループ長を長くとれるような工夫
 - 現状は比較的naïveな実装
- ベクトルプロセッサ向けコード(下図のDJDS)
- プロファイラ情報の利用
- コンパイラの成熟度に問題アリ
 - FORTRANはそれでもかなり良い
 - C, C++
- NUMA向け最適化
 - 京, FX10には効かない
 - とは言え, やらなくて良いわけではない
 - 将来のアーキテクチャ

DJDS**DCRS**

- 更なる最適化
- コンパイルオプション
 - FORTRANのみ
 - コンパイルリストの分析
- 実行

実習

- 色々なケースでやってみよう
 - 問題サイズ
 - スレッド数
 - 色数, 色分け法 (MC, RCM, CM-RCM)
- Kshortloop=Nの値を変えて見よ
- 色数によって効き具合がどう変わるか？