

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

2008年12月3日・4日

中島研吾

日本応用数理学会 2008年「秋の学校」

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

プログラムのありか

- 所在
 - `<$L3>/src, <$L3>/run`
- コンパイル, 実行方法
 - メッシュ生成
 - `cd <$L3>/run`
 - `f90 -O mg.f, cc -O mg.c`
 - 本体
 - `cd <$L3>/src`
 - `make`
 - コントロールデータ
 - `<$L3>/run/INPUT.DAT`
 - 実行用シェル
 - `<$L3>/run/go.sh`

実行例

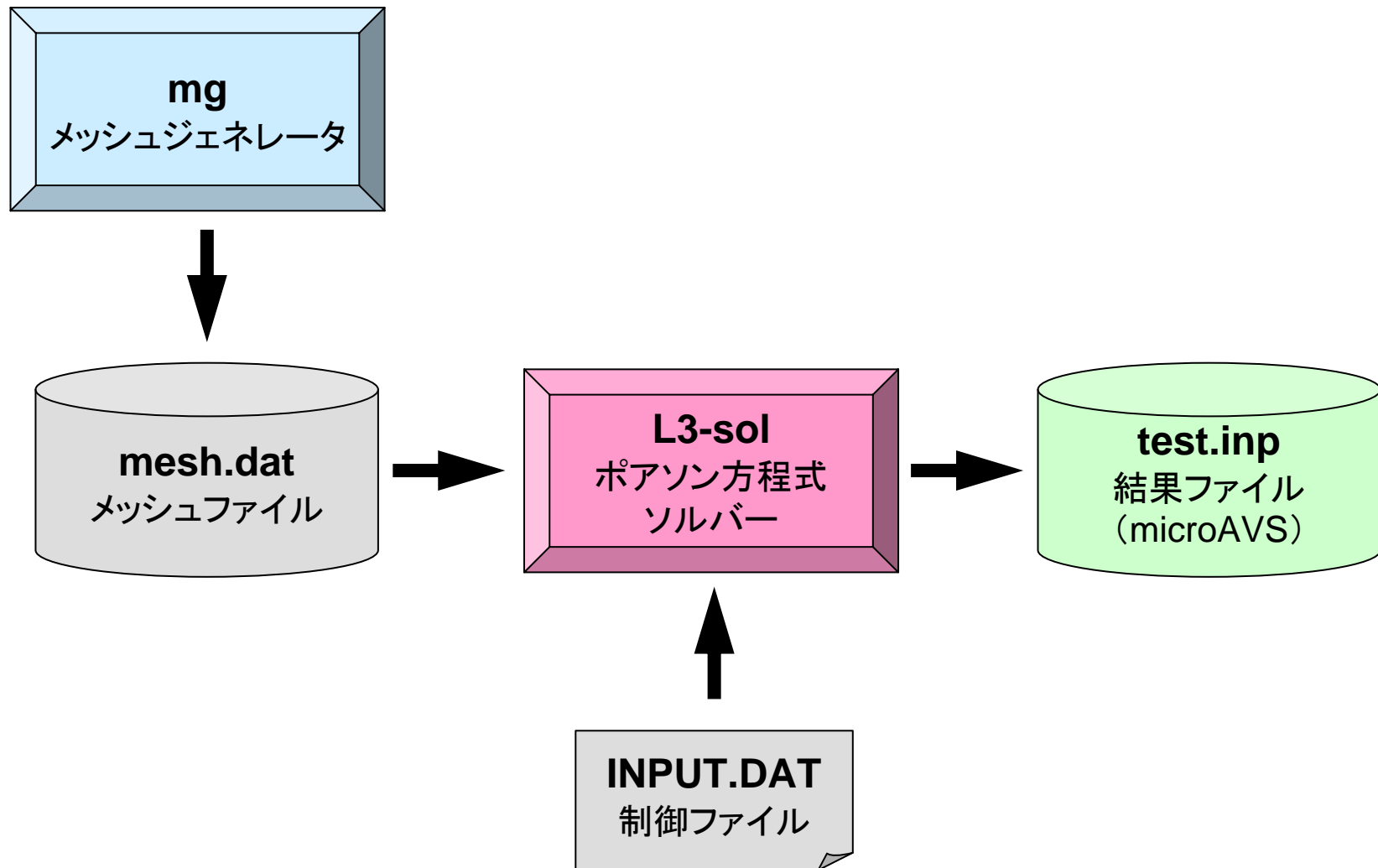
```
% cd <$L3>
% ls
    run    src
% cd src
% make
% cd ../run
% ls L3-sol
    L3-sol
% f90 -O3 mg.f -o mg

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

% qsub go.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)

```

1.00e-00 1.00e-00 1.00e-00      DX/DY/DZ
0.10      1.0e-08                OMEGA, EPSICCG
8                                                PEsmptOT
100                                           NCOLORTot

```

| 変数名 | 型 | 内 容 |
|------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| DX, DY, DZ | 倍精度実数 | 各要素の3辺の長さ (ΔX , ΔY , ΔZ) |
| OMEGA | 倍精度実数 | 不使用 (適当な値を入れておく) |
| 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 |

- 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
- T2K(東大)での実行
- T2K(東大)での性能向上への道
 - NUMA Control
 - First Touch
 - データ再配置

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)

```
!$omp parallel do private(i, VAL, j)
  do i = 1, N
    VAL= D(i)*W(i, P)
    do j= 1, INL(i)
      VAL= VAL + AL(j, i)*W(IAL(j, i), P)
    enddo
    do j= 1, INU(i)
      VAL= VAL + AU(j, i)*W(IAU(j, i), P)
    enddo
    W(i, Q)= VAL
  enddo
!$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
!$omp parallel do private (i, VAL, j)
  do i= COLORindex(icol-1)+1, COLORindex(icol)
    VAL= D(i)
    do j= 1, INL(i)
      VAL= VAL - (AL(j, i)**2) * DD(IAL(j, i))
    enddo
    DD(i)= 1. d0/VAL
  enddo
!$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 MPI_INIT      (ierr)

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN

PHI= 0.d0

call solve_ICCG_mc                                &
&      ( ICELTOT, NL, NU, INL, IAL, INU, IAU, D, BFORCE,      &
&      DELPHI, AL, AU, NCOLORTot, COLORindex,              &
&      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  
  
do icel= 1, ICELTOT  
  PHI(icel)= WK(icel)  
enddo  
  
call OUTUCD  
  
call MPI_FINALIZE (ierr)  
  
stop  
end
```

結果(PHI)をもとの番号
付けにもどす

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 MPI_INIT      (ierr)

  call INPUT
  call POINTER_INIT
  call BOUNDARY_CELL
  call CELL_METRICS
  call POI_GEN

  PHI= 0.d0

  call solve_ICCG_mc                                &
&    ( ICELTOT, NL, NU, INL, IAL, INU, IAU, D, BFORCE,    &
&    DELPHI, AL, AU, NCOLORTot, COLORindex,            &
&    SMPindex, SMPindexG, EPSICCG, ITR, IER)            &
```

module STRUCT

```

module STRUCT

  include 'precision.inc'
  include 'mpif.h'

!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) :: ZmaxGELtot
  integer (kind=kint), dimension(:), allocatable :: BC_INDEX, BC_NOD
  integer (kind=kint), dimension(:), allocatable :: ZmaxCEL

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

!C
!C-- MPI
  integer (kind=kint) :: my_rank, SOLVER_COMM, PETOT, 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 (1/2)

```

module PCG

integer, parameter :: N2= 256
integer :: NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL
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

end module PCG

```

下三角成分：
非対角成分で自分より要素番号
が小さい。

$$IAL(icou,i) < i$$

上三角成分：
非対角成分で自分より要素番号
が大きい。

$$IAU(icou,i) > i$$

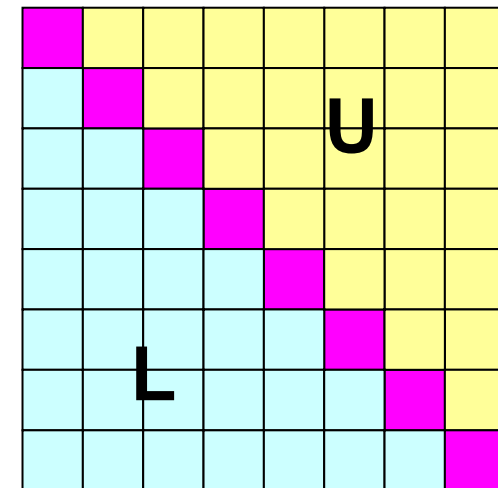
INL (ICELTOT)
IAL (NL, ICELTOT)
INU (ICELTOT)
IAU (NU, ICELTOT)
NU, NL
NUmax, NLmax

NCOLORtot
COLORindex (0:NCOLORtot)

OLDtoNEW, NEWtoOLD

下三角成分の数
下三角成分
上三角成分の数
上三角成分
上下三角成分の最大数 (ここでは6)
未使用

色数
各色に含まれる要素数のインデックス
(COLORindex (icol) - COLORindex (icol-1))
Coloring前後の要素番号対照表



module PCG (1/2)

```

module PCG

integer, parameter :: N2= 256
integer :: NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL
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

end module PCG

```

下三角成分：
非対角成分で自分より要素番号
が小さい。

$$IAL(icou,i) < i$$

上三角成分：
非対角成分で自分より要素番号
が大きい。

$$IAU(icou,i) > i$$

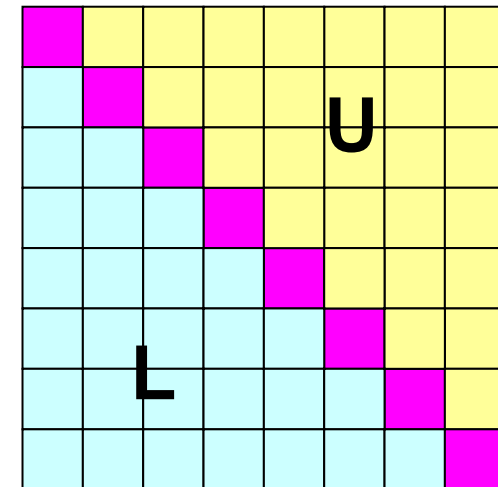
INL (ICELTOT)
IAL (NL, ICELTOT)
INU (ICELTOT)
IAU (NU, ICELTOT)
NU, NL
NUmax, NLmax

NCOLORtot
COLORindex (0:NCOLORtot)

OLDtoNEW, NEWtoOLD

下三角成分の数
下三角成分
上三角成分の数
上三角成分
上下三角成分の最大数 (ここでは6)
未使用

色数
各色に含まれる要素数のインデックス
(COLORindex (icol) - COLORindex (icol-1))
Coloring前後の要素番号対照表



module PCG (2/2)

```

module PCG

integer, parameter :: N2= 256
integer :: NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL
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

end module PCG

```

下三角成分：
非対角成分で自分より要素番号
が小さい。

$$IAL(icou,i) < i$$

上三角成分：
非対角成分で自分より要素番号
が大きい。

$$IAU(icou,i) > i$$

SMP_{index} (0:N_{COLOR}_{tot}*PE_{smp}_{TOT}) スレッド用配列 (後述)
SMP_{indexG} (0:PE_{smp}_{TOT})

EPSICCG

収束打切残差

D (ICELTOT)

係数行列の対角成分

PHI (ICLETOT)

従属変数

BFORCE (ICELTOT)

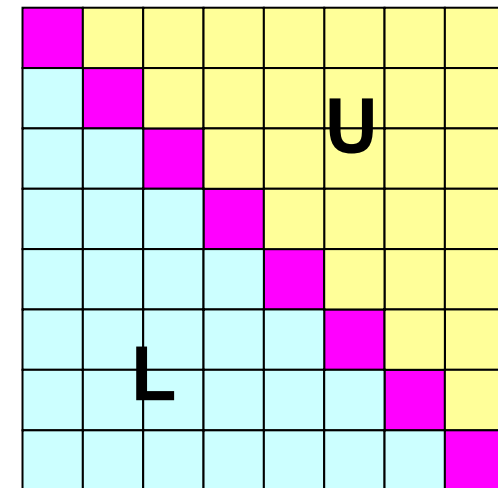
連立一次方程式の右辺ベクトル

AL (NL, ICELTOT)

係数行列の下三角成分

AU (NU, ICELTOT)

係数行列の上三角成分



変数表

| 変数, 配列名 | 型 | 内容 |
|--------------|----|-------------------------------------------------------------------------------------------------------------------------|
| NCOLORtot | 整数 | 入力時にはOrdering手法 (≥ 2 : MC, $=0$: CM, <0 : RCM) 最終的には色数, レベル数が入る |
| ICELTOT | 整数 | 要素数 (=16) |
| NL | 整数 | 各要素の下三角成分の最大数 (=6に設定してある) |
| NU | 整数 | 各要素の上三角成分の最大数 (=6に設定してある) |
| INL(i) | 整数 | 要素 <i>i</i> の下三角成分数 ($<NL$), $i=1\sim ICELTOT$ |
| INU(i) | 整数 | 要素 <i>i</i> の上三角成分数 ($<NU$), $i=1\sim ICELTOT$ |
| IAL(j,i) | 整数 | 要素 <i>i</i> の <i>j</i> 番目の下三角成分に対応する列番号, $i=1\sim ICELTOT$, $j=1\sim INL(i)$ |
| IAU(j,i) | 整数 | 要素 <i>i</i> の <i>j</i> 番目の上三角成分に対応する列番号, $i=1\sim ICELTOT$, $j=1\sim INU(i)$ |
| COLORindex | 整数 | 各色, レベルに含まれる要素数の一次元圧縮配列, $i=0\sim NCOLORtot$, 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(i) | 整数 | 新番号 \Rightarrow 旧番号への参照配列, $i=1\sim ICELTOT$ |
| OLDtoNEW(i) | 整数 | 旧番号 \Rightarrow 新番号への参照配列, $i=1\sim ICELTOT$ |
| PEsmpTOT | 整数 | データ分割数 |
| SMPindex(k) | 整数 | スレッド用補助配列 (データ依存性があるループに使用), $k=$ $0\sim NCOLORtot * PEsmpTOT$ |
| SMPindexG(k) | 整数 | スレッド用補助配列 (データ依存性がないループに使用), $k= 0\sim PEsmpTOT$ |

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 MPI_INIT      (ierr)

  call INPUT
  call POINTER_INIT
  call BOUNDARY_CELL
  call CELL_METRICS
  call POI_GEN

  PHI= 0.d0

  call solve_ICCG_mc                                &
&      ( ICELTOT, NL, NU, INL, IAL, INU, IAU, D, BFORCE,    &
&      DELPHI, AL, AU, NCOLORTot, COLORindex,             &
&      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,*) DX, DY, DZ
      read (11,*) OMEGA, 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
 - 「<0」の場合はRCM

```

1.00e-02 5.00e-02 1.00e-02
0.10      1.0e-08
8
100

```

```

DX/DY/DZ
OMEGA, 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)

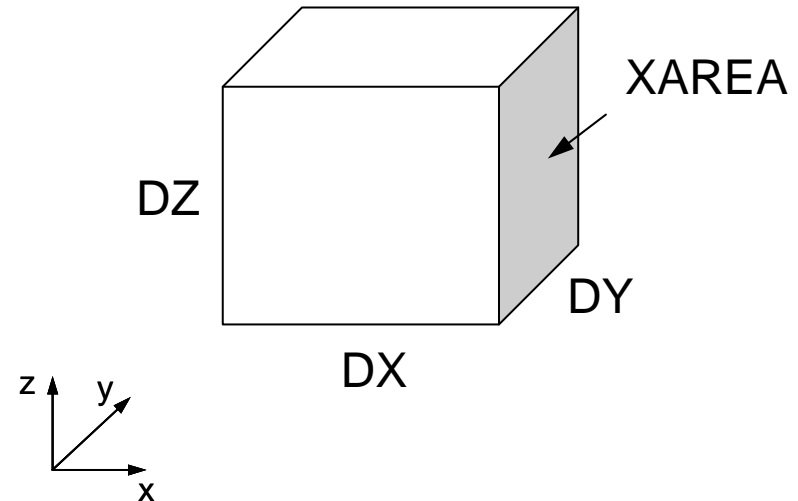
    V0= DX * DY * DZ
    RVO= 1. d0/V0

    VOLCEL= V0
    RVC   = RVO

    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 MPI_INIT      (ierr)

  call INPUT
  call POINTER_INIT
  call BOUNDARY_CELL
  call CELL_METRICS
  call POI_GEN

  PHI= 0.d0

  call solve_ICCG_mc                                &
&    ( ICELTOT, NL, NU, INL, IAL, INU, IAU, D, BFORCE,  &
&    DELPHI, AL, AU, NCOLORTot, COLORindex,          &
&    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), DELPHI(nn))
  allocate (INL(nn), INU(nn), IAL(NL,nn), IAU(NU,nn))
  allocate (AL(NL,nn), AU(NU,nn))

  DELPHI= 0.d0
  D= 0.d0

  INL= 0
  INU= 0
  IAL= 0
  IAU= 0
  AL= 0.d0
  AU= 0.d0
```

配列の宣言

JSLAM 2008.12.03/04

```

!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

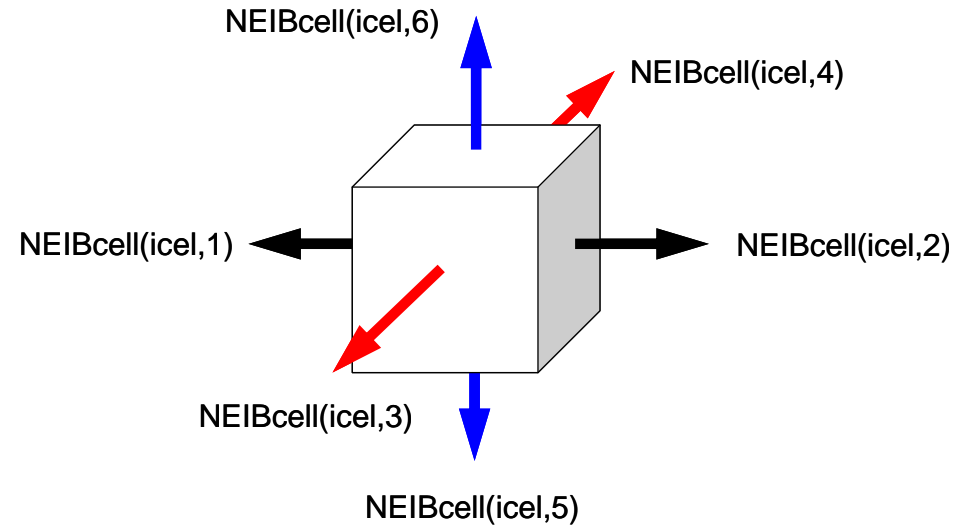
  if (icN2.ne.0.and.icN2.le.ICELTOT) then
    icou= INU(icel) + 1
    IAU(icou, icel)= icN2
    INU(   icel)= icou
  endif

  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===

```

poi_gen(2/9)



下三角成分

$$\text{NEIBcell}(\text{icel}, 5) = \text{icel} - \text{NX} * \text{NY}$$

$$\text{NEIBcell}(\text{icel}, 3) = \text{icel} - \text{NX}$$

$$\text{NEIBcell}(\text{icel}, 1) = \text{icel} - 1$$

JSLAM 2008.12.02/04

```

!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

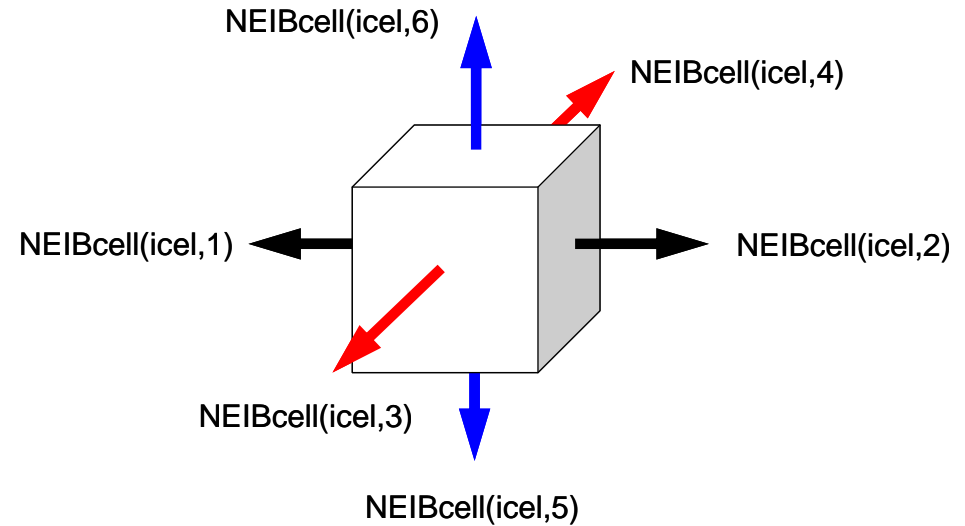
  if (icN2.ne.0.and.icN2.le.ICELTOT) then
    icou= INU(icel) + 1
    IAU(icou,icel)= icN2
    INU(   icel)= icou
  endif

  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===

```

poi_gen(2/9)



上三角成分

$$\text{NEIBcell}(\text{icel},2) = \text{icel} + 1$$

$$\text{NEIBcell}(\text{icel},4) = \text{icel} + \text{NX}$$

$$\text{NEIBcell}(\text{icel},6) = \text{icel} + \text{NX} * \text{NY}$$

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

```

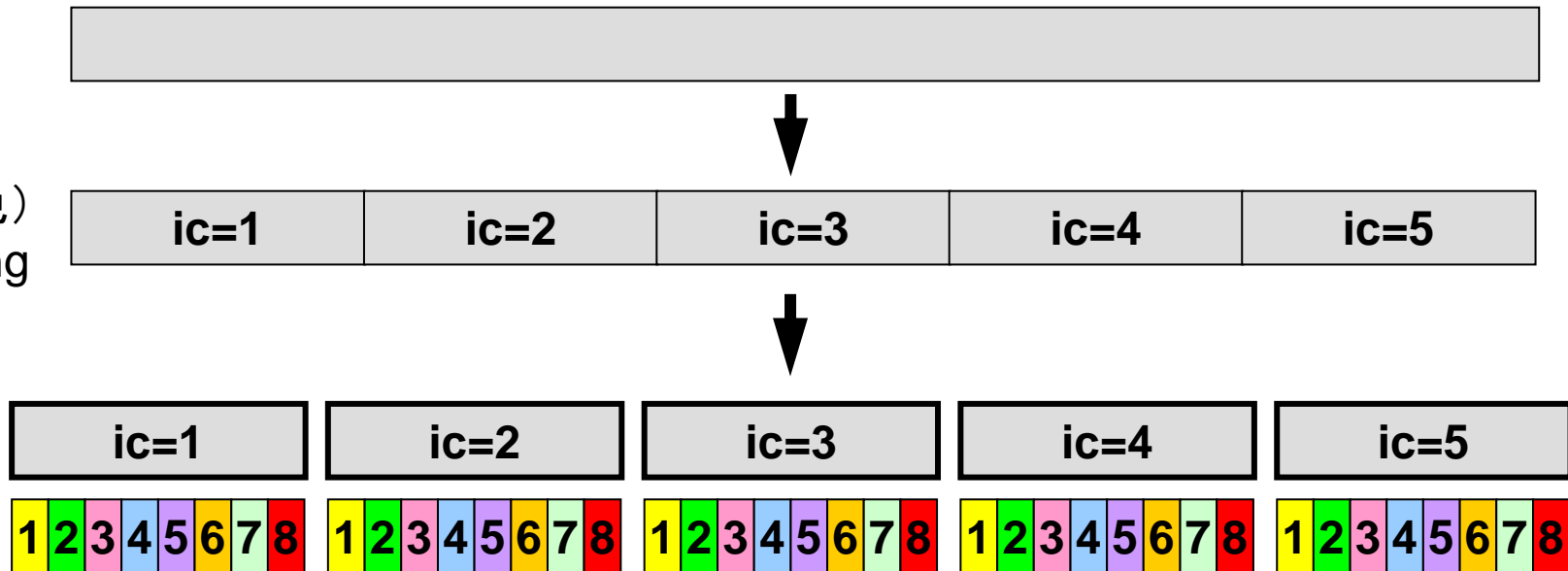
SMPindex: 前処理向け

```

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

```

彩色(5色)
+Ordering



各色内の要素は互いに独立: 並列に計算可能⇒スレッド数で更に分割

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

```

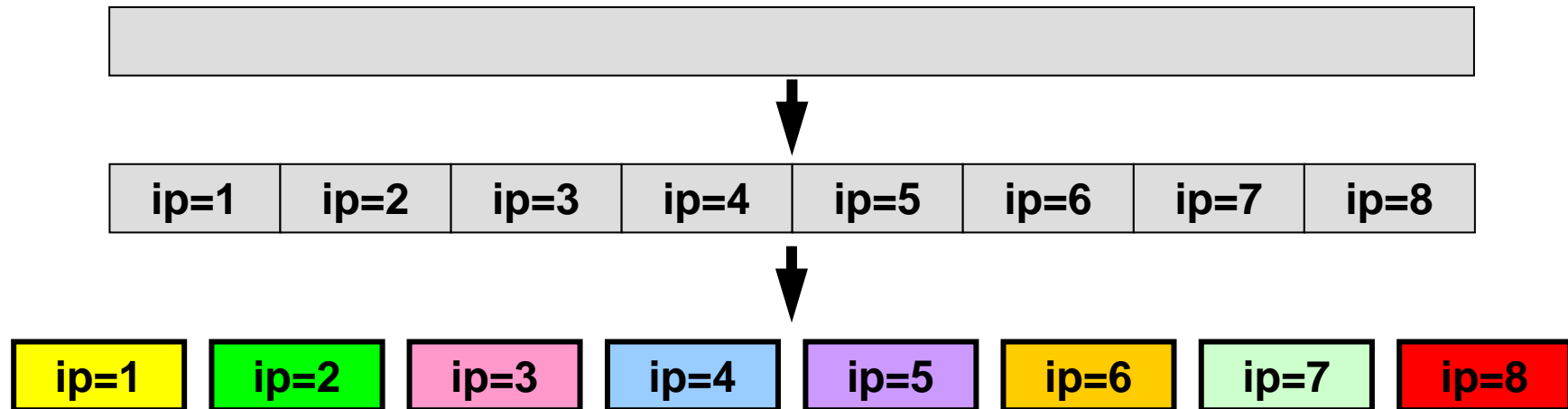
全要素数を「PEsmptOT」で割って
「SMPindexG」に割り当てる。

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

```

icel: 新しい番号
ic0: 古い番号

poi_gen (5/9)

これ以降は新しい
番号付けを使用

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

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)

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

```

icel: 新しい番号
ic0: 古い番号

poi_gen (5/9)

これ以降は新しい
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, 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= VOLGEL (ic0)

  if (icN5.ne.0) then
    icN5= OLDtoNEW(icN5)
    coef= RDZ2 * VOL0
    D(icel)= D(icel) - coef

    if (icN5.lt.icel) then
      do j= 1, INL(icel)
        if (IAL(j, icel).eq.icN5) then
          AL(j, icel)= coef
          exit
        endif
      enddo
    else
      do j= 1, INU(icel)
        if (IAU(j, icel).eq.icN5) then
          AU(j, icel)= coef
          exit
        endif
      enddo
    endif
  endif
endif
endif

```

poi_gen (5/9)

これ以降は新しい
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, 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= VOLGEL (ic0)

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

  if (icN5.lt.icel) then
    do j= 1, INL(icel)
      if (IAL(j, icel).eq.icN5) then
        AL(j, icel)= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq.icN5) then
        AU(j, icel)= 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 (5/9)

これ以降は新しい番号付けを使用

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

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

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

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

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

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

```

!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= VOLGEL (ic0)

  if (icN5.ne.0) then
    icN5= OLDtoNEW(icN5)
    coef= RDZ2 * VOL0
    D(icel)= D(icel) - coef

    if (icN5.lt.icel) then
      do j= 1, INL(icel)
        if (IAL(j, icel).eq.icN5) then
          AL(j, icel)= coef
          exit
        endif
      enddo
    else
      do j= 1, INU(icel)
        if (IAU(j, icel).eq.icN5) then
          AU(j, icel)= coef
          exit
        endif
      enddo
    endif
  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 (5/9)

これ以降は新しい
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
        AL(j, icel)= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq. icN3) then
        AU(j, icel)= 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
        AL(j, icel)= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq. icN1) then
        AU(j, icel)= coef
        exit
      endif
    enddo
  endif
endif
endif

```

poi_gen(6/9)

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

```

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
        AL(j, icel)= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq. icN2) then
        AU(j, icel)= 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
        AL(j, icel)= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq. icN4) then
        AU(j, icel)= coef
        exit
      endif
    enddo
  endif
endif
endif

```

poi_gen(7/9)

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

```
!$omp parallel do private (ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6) &
!$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
      AL(j, icel)= coef
      exit
    endif
  enddo
else
  do j= 1, INU(icel)
    if (IAU(j, icel).eq.icN6) then
      AU(j, icel)= 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==
```

もとの座標に従って
BFORCE計算

ii,jj,kk,VOL0はprivate

poi_gen (8/9)

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 MPI_INIT      (ierr)

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN

PHI= 0. d0

call solve_ICCG_mc      &
&      ( ICELTOT, NL, NU, INL, IAL, INU, IAU, D, BFORCE,      &
&      DELPHI, AL, AU, NCOLORtot, PEsmtot,                  &
&      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, NL, NU, INL, IAL, INU, IAU, D, B, X,          &
&      AL, AU, NCOLORtot, PEsmptTOT, SMPindex, SMPindexG, &
&      EPS, ITR, IER)
  implicit REAL*8 (A-H, O-Z)

  integer :: N, NL, NU, NCOLOR, PEsmptTOT

  real(kind=8), dimension(N)  :: D
  real(kind=8), dimension(N)  :: B
  real(kind=8), dimension(N)  :: X
  real(kind=8), dimension(NL, N) :: AL
  real(kind=8), dimension(NU, N) :: AU

  integer, dimension(N)  :: INL, INU
  integer, dimension(NL, N) :: IAL
  integer, dimension(NU, N) :: IAU
  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,4))

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

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

```

privateに注意。

solve_ICCG_mc(3/6)

```

!C +-----+
!C | {r0}= {b} - [A]{xini} |
!C +-----+
!C===
!$omp parallel do private(ip,i,VAL,j)
do ip= 1, PEsmptOT
do i = SMPindexG(ip-1)+1, SMPindexG(ip)
VAL= D(i)*X(i)
do j= 1, INL(i)
VAL= VAL + AL(j,i)*X(IAL(j,i))
enddo
do j= 1, INU(i)
VAL= VAL + AU(j,i)*X(IAU(j,i))
enddo
W(i,R)= B(i) - VAL
enddo
enddo
!$omp end parallel do

BNRM2= 0.0D0
!$omp parallel do private(ip,i) reduction(+:BNRM2)
do ip= 1, PEsmptOT
do i = SMPindexG(ip-1)+1, SMPindexG(ip)
BNRM2 = BNM2 + B(i) **2
enddo
enddo
!$omp end parallel do
!C===

```

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

for $i = 1, 2, \dots$

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

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

if $i=1$

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

else

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

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

endif

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

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

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

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

check convergence $|\mathbf{r}|$

end

行列ベクトル積

依存性が無い⇒独立に計算可能⇒SMPindexG使用

```
!$omp parallel do private(ip, i, VAL, j)
  do ip= 1, PEsmptOT
    do i = SMPindexG(ip-1)+1, SMPindexG(ip)
      VAL= D(i)*X(i)
      do j= 1, INL(i)
        VAL= VAL + AL(j, i)*X(IAL(j, i))
      enddo
      do j= 1, INU(i)
        VAL= VAL + AU(j, i)*X(IAU(j, i))
      enddo
      W(i, R)= B(i) - VAL
    enddo
  enddo
!$omp end parallel do
```

solve_ICCG_mc(3/6)

```

!C +-----+
!C | {r0} = {b} - [A]{xini} |
!C +-----+
!C===
!$omp parallel do private(ip,i,VAL,j)
do ip= 1, PEsmptOT
  do i = SMPindexG(ip-1)+1, SMPindexG(ip)
    VAL= D(i)*X(i)
    do j= 1, INL(i)
      VAL= VAL + AL(j,i)*X(IAL(j,i))
    enddo
    do j= 1, INU(i)
      VAL= VAL + AU(j,i)*X(IAU(j,i))
    enddo
    W(i,R)= B(i) - VAL
  enddo
enddo
!$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, ...
  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
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} z^{(i)}$ 
  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
```

不完全修正コレスキー分解を使用した 前進後退代入

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

$$\{z\} = (LDL^T)^{-1}\{r\} \longrightarrow \begin{array}{l} (L)\{y\} = \{r\} \\ (DL^T)\{z\} = \{y\} \end{array}$$

不完全修正コレスキー分解を使用した 前進後退代入

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

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

```

!C
!C +-----+
!C | {z} = [Minv] {r} |
!C +-----+
!C===
      do i= 1, N
        W(i, Z) = W(i, R)
      enddo

      do i= 1, N
        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

      do i= N, 1, -1
        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
!C===

```

solve_ICCG_mc(4/6)

```

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

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

do ic= NCOLORTot, 1, -1
!$omp parallel do private(ip, ip1, i, SW, j)
do ip= 1, PEsmptOT
ip1= (ic-1)*PEsmptOT + ip
do i= SMPindex(ip1-1)+1, SMPindex(ip1)
SW = 0.0d0
do j= 1, INU(i)
SW= SW + AU(j, i) * W(IAU(j, i), Z)
enddo
W(i, Z)= W(i, Z) - W(i, DD) * SW
enddo
enddo
!$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
 $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
 $p^{(i)} = z^{(i-1)} + \beta_{i-1} z^{(i)}$
 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)

ここでは「SMPindex」を使う

```

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

do ic= NCOLORTot, 1, -1
!$omp parallel do private(ip, ip1, i, SW, j)
do ip= 1, PEsmptOT
ip1= (ic-1)*PEsmptOT + ip
do i= SMPindex(ip1-1)+1, SMPindex(ip1)
SW = 0.0d0
do j= 1, INU(i)
SW= SW + AU(j, i) * W(IAU(j, i), Z)
enddo
W(i, Z)= W(i, Z) - W(i, DD) * SW
enddo
enddo
!$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
 $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
 $p^{(i)} = z^{(i-1)} + \beta_{i-1} z^{(i)}$
 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)

```

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===

```

ここでは「SMPindex」を使う

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

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

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

solve_ICCG_mc(5/6)

```

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

!C +-----+
!C | {q} = [A] {p} |
!C +-----+
!C===
!$omp parallel do private(ip, i, VAL, j)
      do ip= 1, PEsmptTOT
        do i = SMPindexG(ip-1)+1, SMPindexG(ip)
          VAL= D(i)*W(i, P)
          do j= 1, INL(i)
            VAL= VAL + AL(j, i)*W(IAL(j, i), P)
          enddo
          do j= 1, INU(i)
            VAL= VAL + AU(j, i)*W(IAU(j, i), P)
          enddo
          W(i, Q)= VAL
        enddo
      enddo
!$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(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===
!$omp parallel do private(ip,i,VAL,j)
  do ip= 1, PEsmptOT
    do i = SMPindexG(ip-1)+1, SMPindexG(ip)
      VAL= D(i)*W(i,P)
      do j= 1, INL(i)
        VAL= VAL + AL(j,i)*W(IAL(j,i),P)
      enddo
      do j= 1, INU(i)
        VAL= VAL + AU(j,i)*W(IAU(j,i),P)
      enddo
      W(i,Q)= VAL
    enddo
  enddo
!$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
 $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$
 $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i)}$
 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
!$omp parallel do private(ip,i) reduction(+:C1)
      do ip= 1, PEsmptOT
        do i = COLORg(ip-1)+1, COLORg(ip)
          C1= C1 + W(i,P)*W(i,Q)
        enddo
      enddo
!$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
!$omp parallel do private(ip,i) reduction(+:C1)
  do ip= 1, PEsmptOT
    do i = COLORg(ip-1)+1, COLORg(ip)
      C1= C1 + W(i,P)*W(i,Q)
    enddo
  enddo
!$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

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

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

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

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

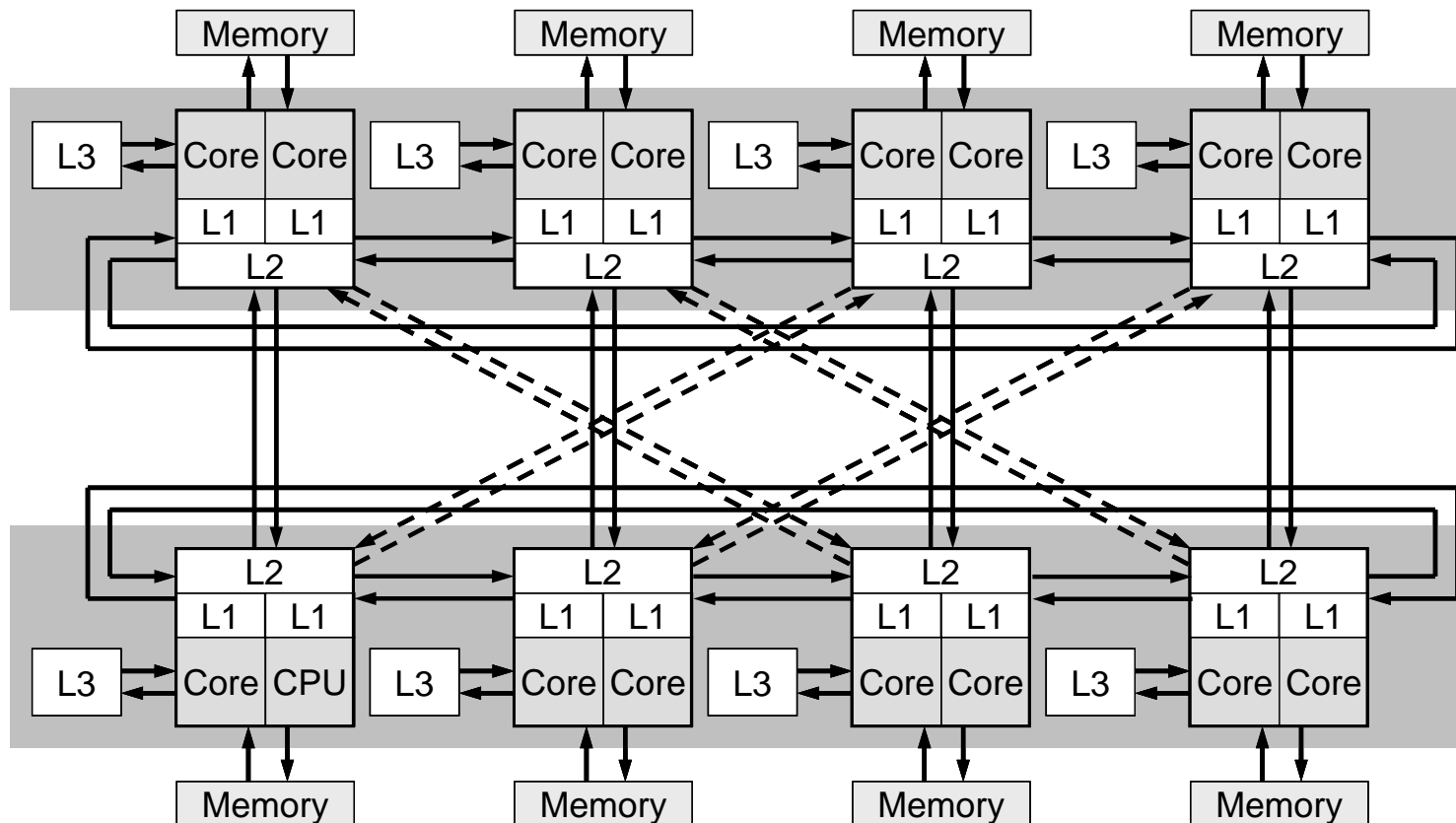
check convergence $|r|$

end

- 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**
- T2K(東大)での実行
- T2K(東大)での性能向上への道
 - NUMA Control
 - First Touch
 - データ再配置

計算結果

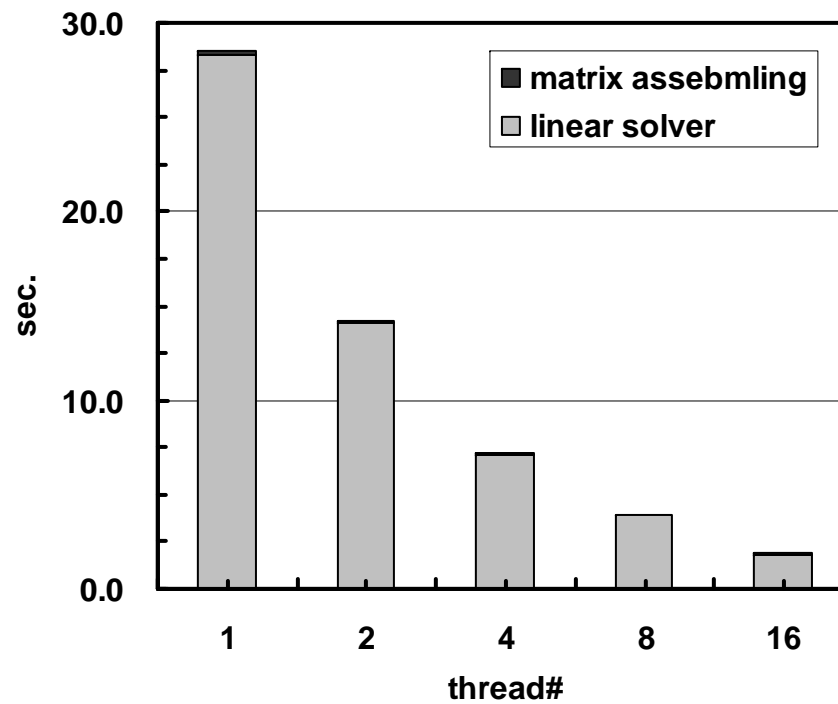
- Hitachi SR11000/J2 1ノード(16コア)
- 100^3 要素



計算時間 (MC=2)

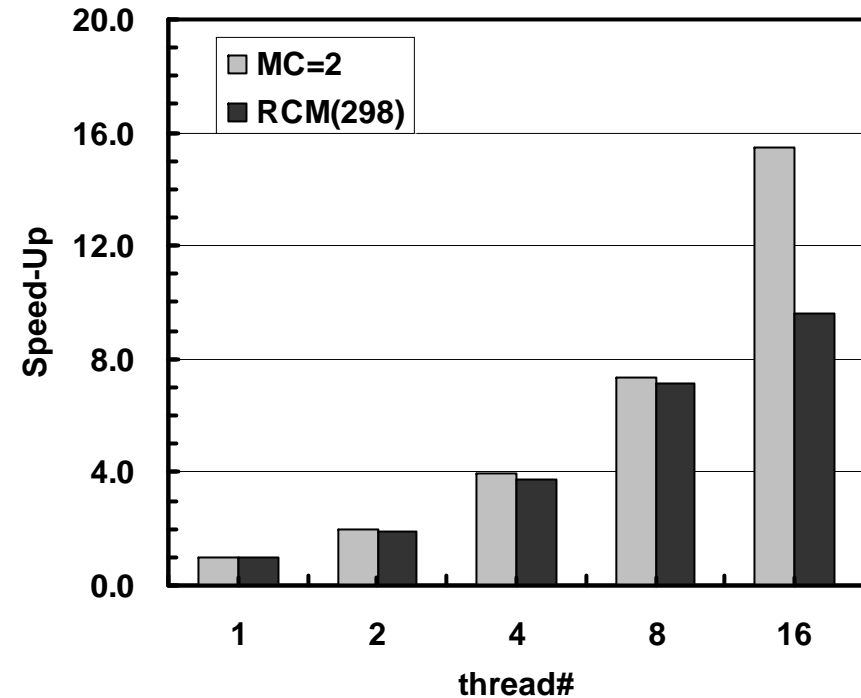
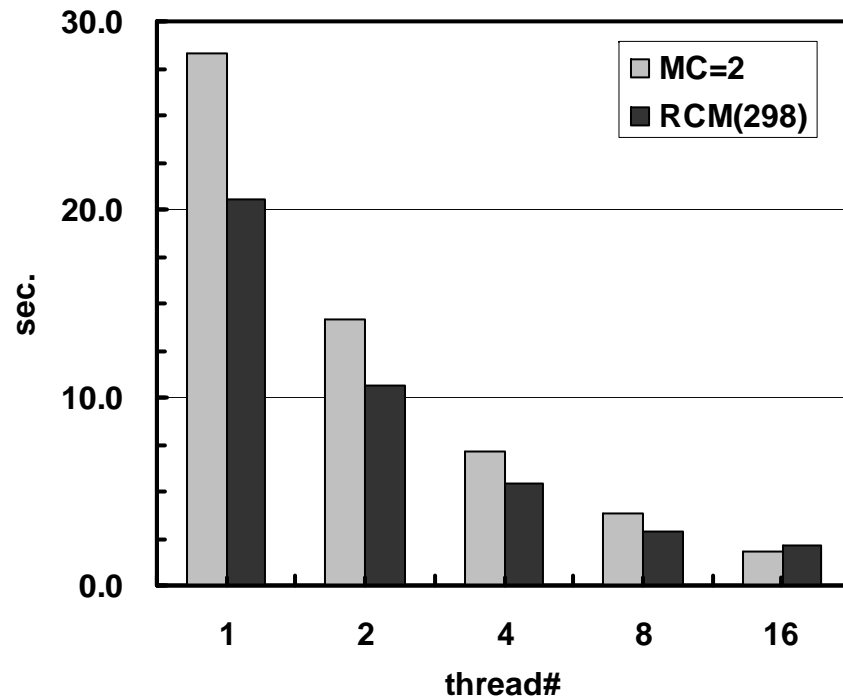
以降, 「linear solver」の計算時間のみ問題とする

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



スケーラビリティ

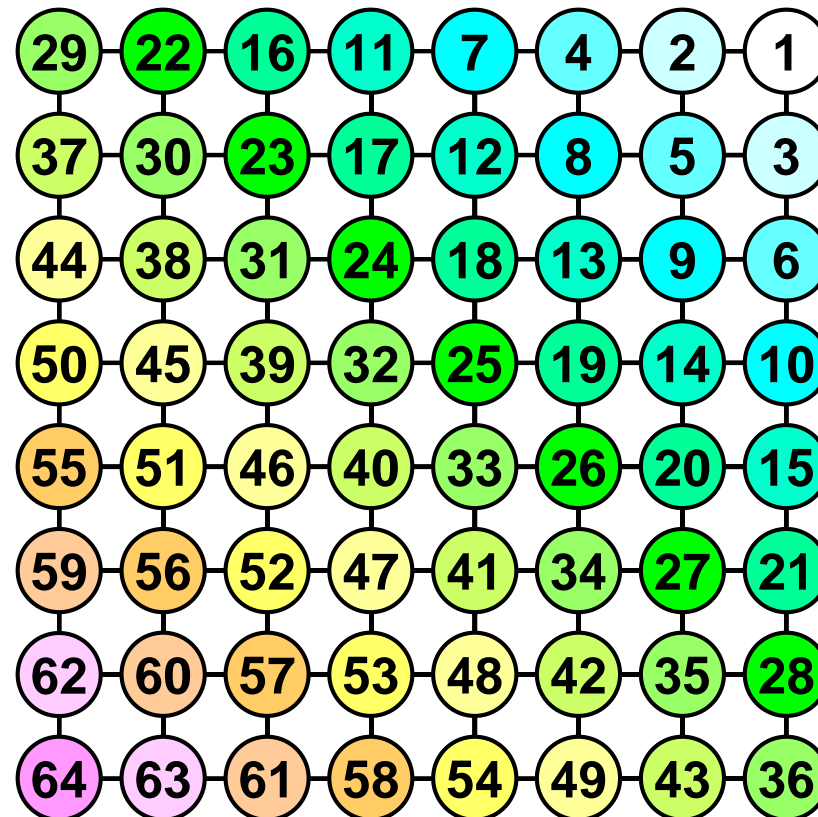
反復回数: MC(2色): 333回, RCM(298レベル): 224回



- 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
- スレッドごとの不可不均衡
 - RCMでは要素数の少ない「レベル」が必ず存在する

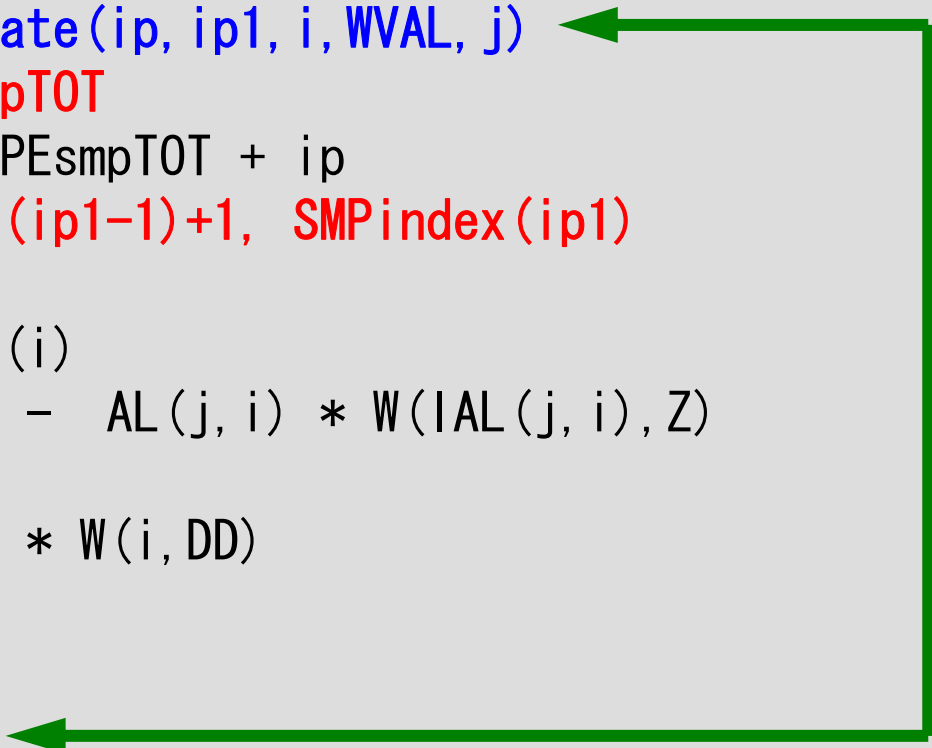


原因はこのようなループにある

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

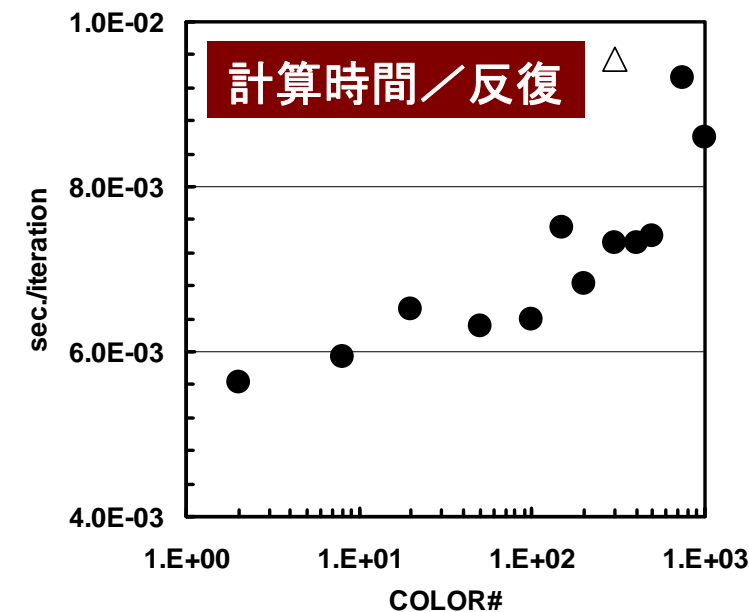
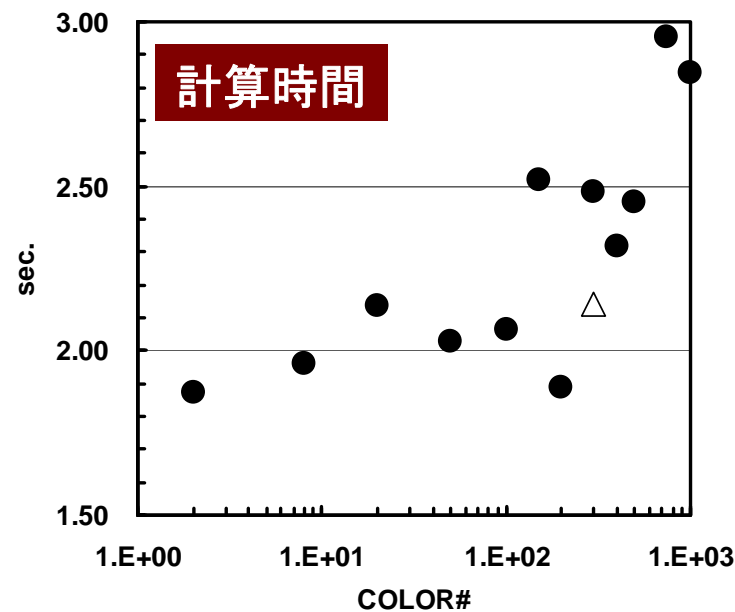
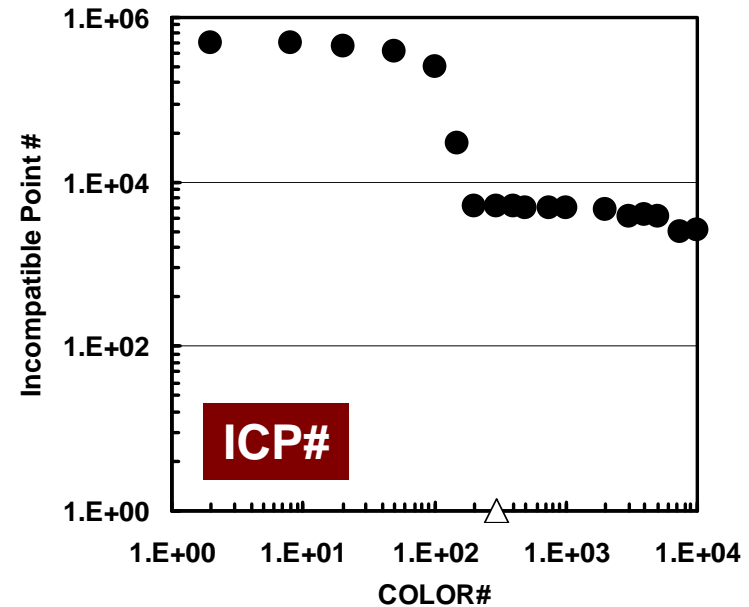
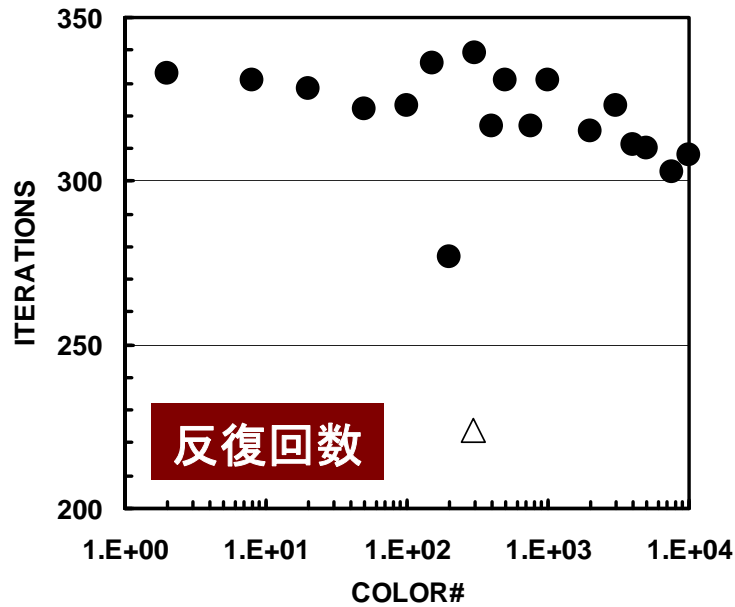
```
do ic= 1, NCOLORTot
!$omp parallel do private(ip, ip1, i, WVAL, j)
  do ip= 1, PEsmptOT
    ip1= (ic-1)*PEsmptOT + ip
    do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      WVAL= W(i, Z)
      do j= 1, INL(i)
        WVAL= WVAL - AL(j, i) * W(IAL(j, i), Z)
      enddo
      W(i, Z)= WVAL * W(i, DD)
  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



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
 - 負荷分散もいま一つ
- 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, かつ反復回数が少ないという都合の良い方法

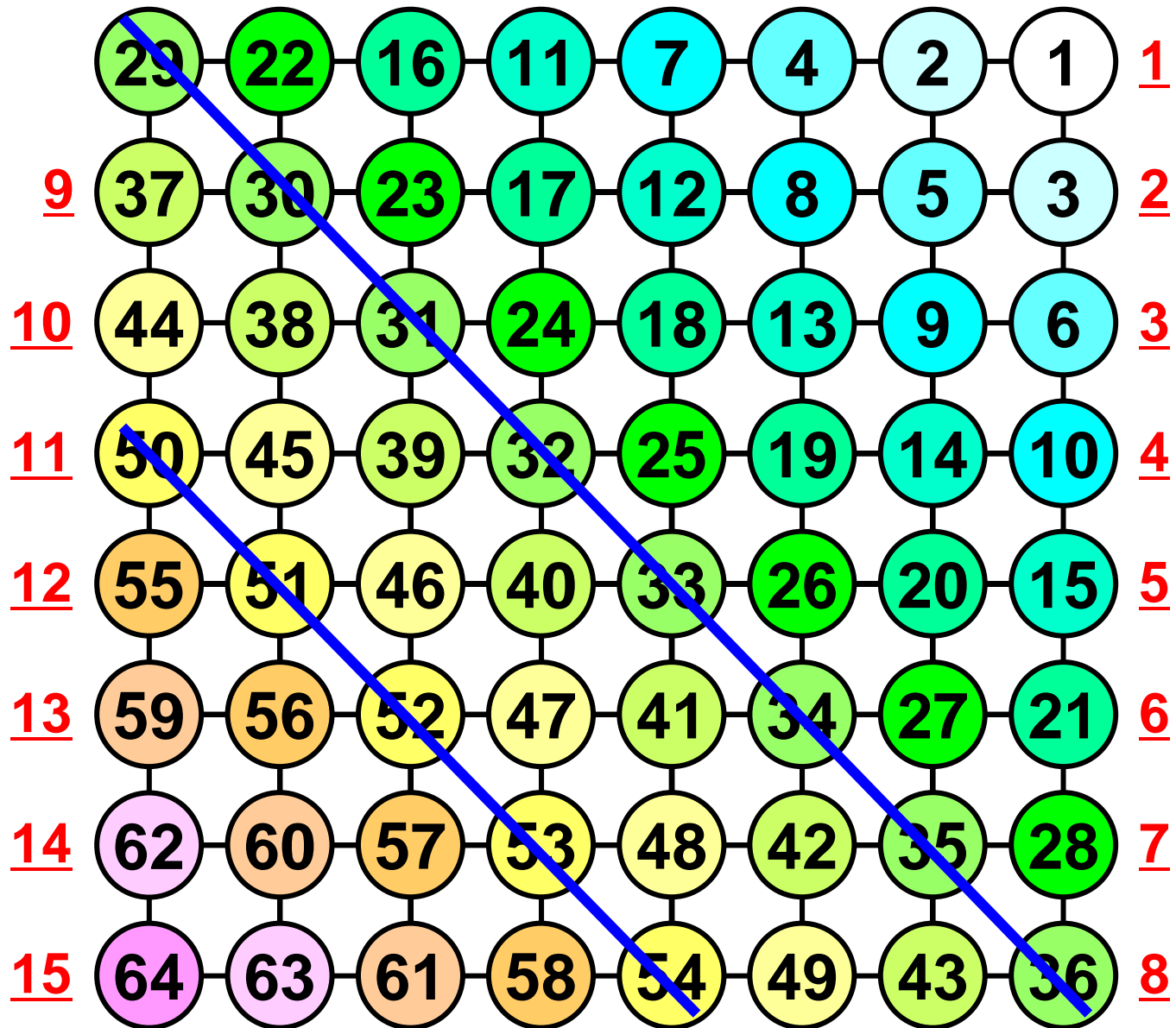
16コアにおける結果 (●: MC, △: RCM)



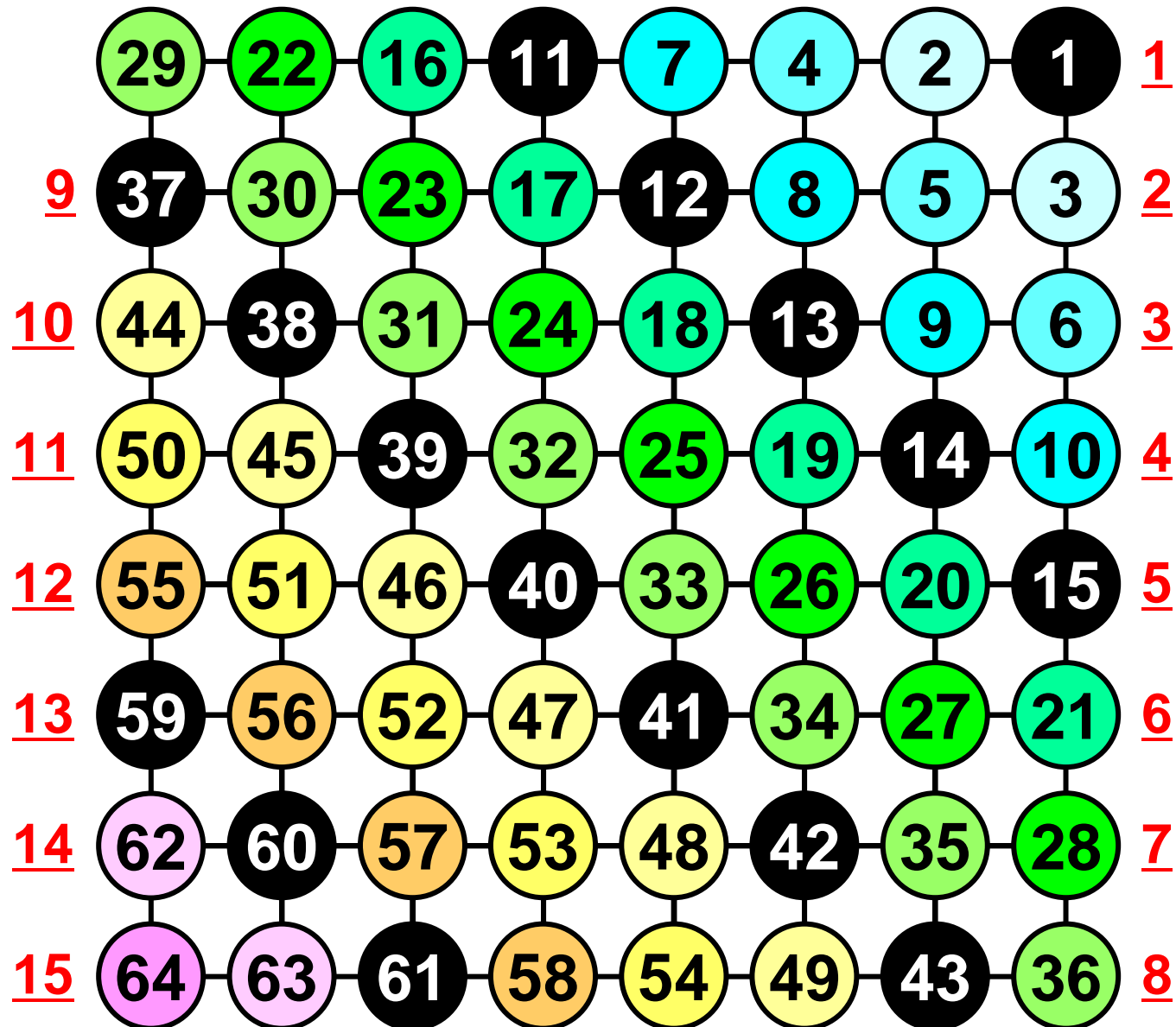
解決策: CM-RCM

- RCM + Cyclic Multicoloring [土井, 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「 $1 \sim N_c$ 」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)

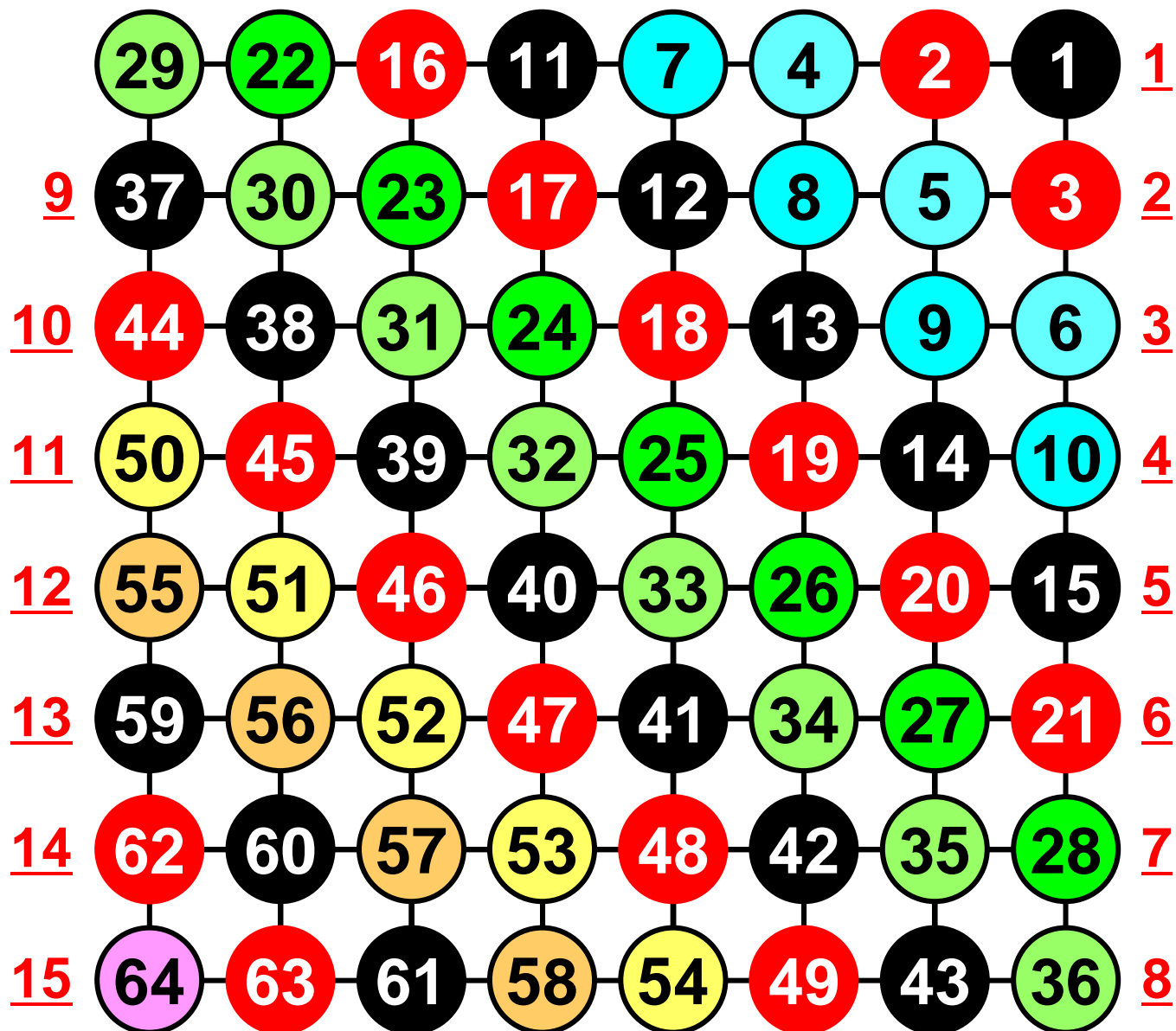
RCM



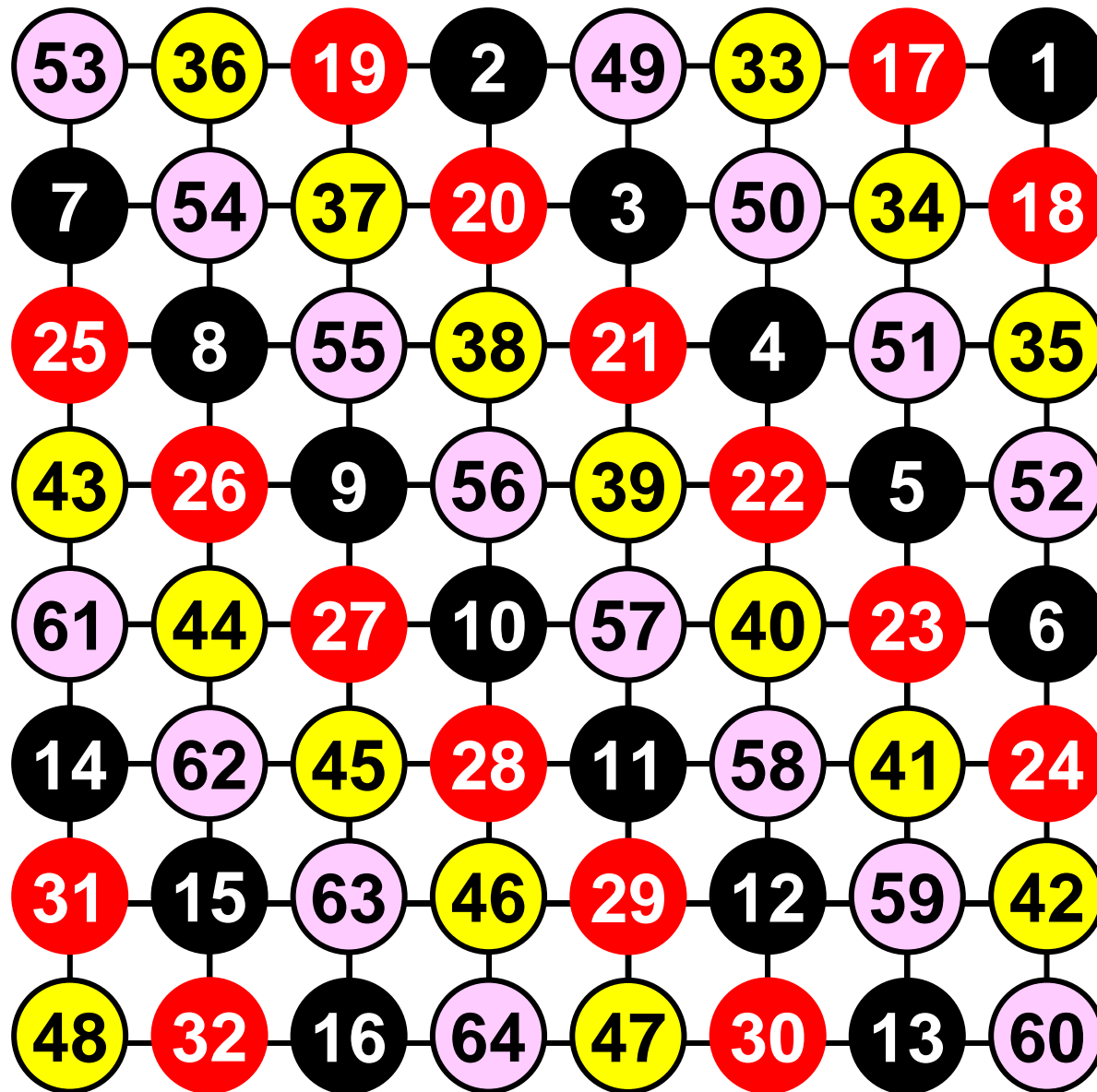
$N_c=4$, $k=1:1,5,9,13$ レベルを選択



$N_c=4$, $k=2:2,6,10,14$ レベルを選択

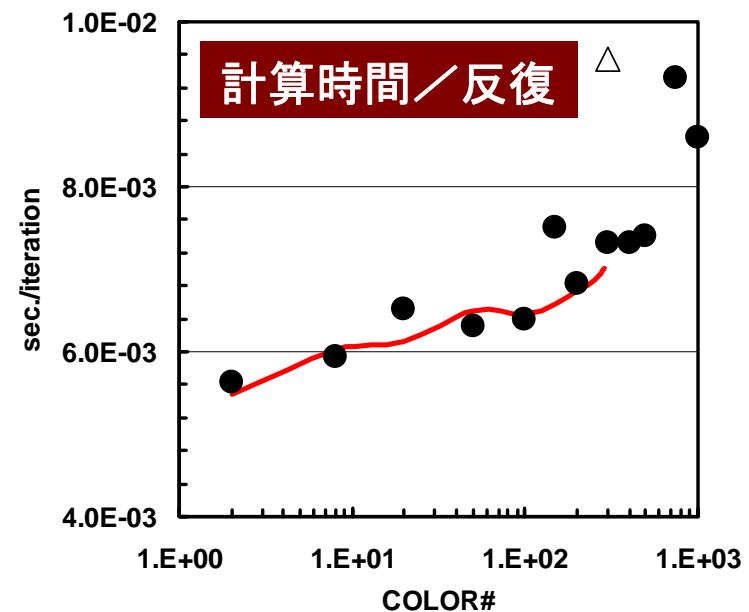
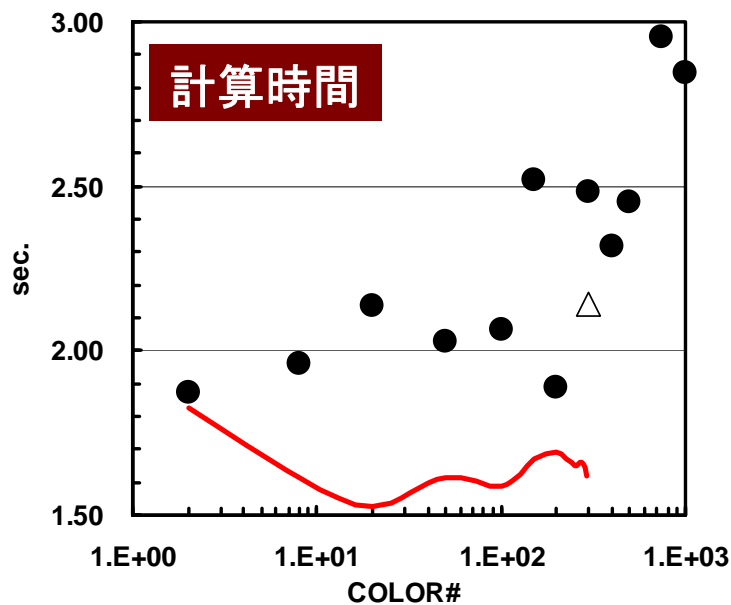
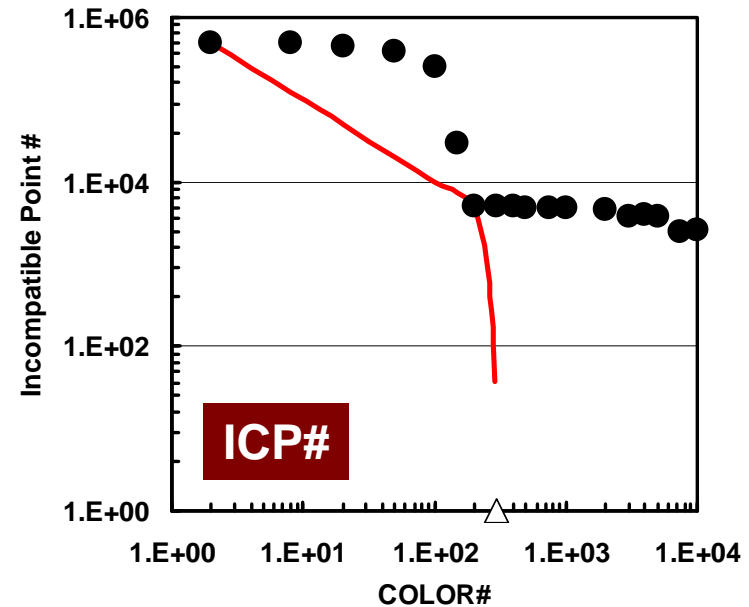
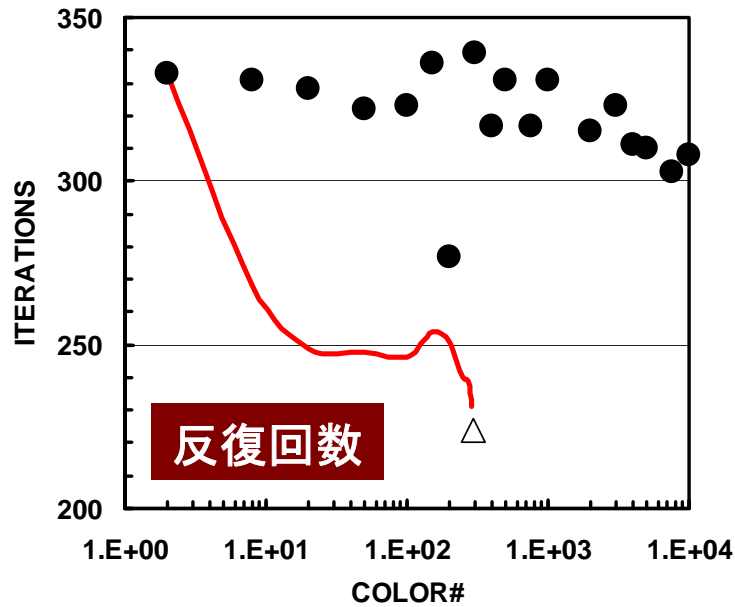


CM-RCM ($N_c=4$) : 「色」の順番に再並替



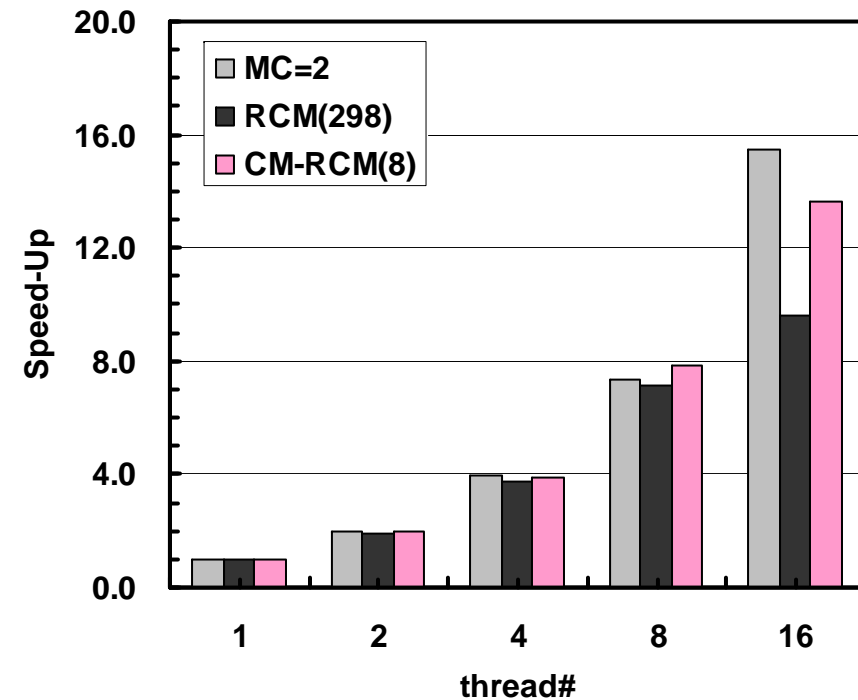
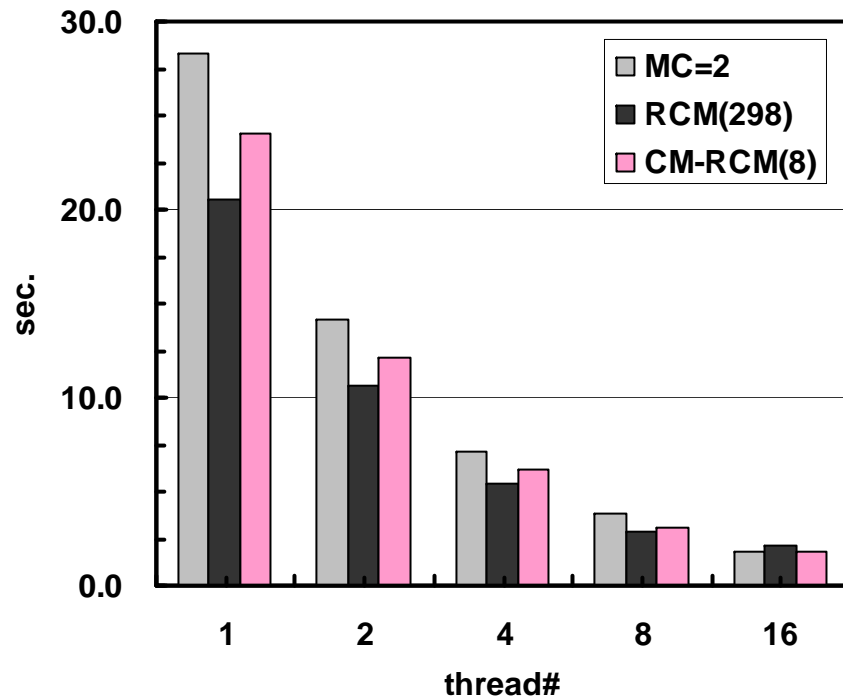
k=1: 16
 k=2: 16
 k=3: 16
 k=4: 16

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 ($N_c=8$) : 268回



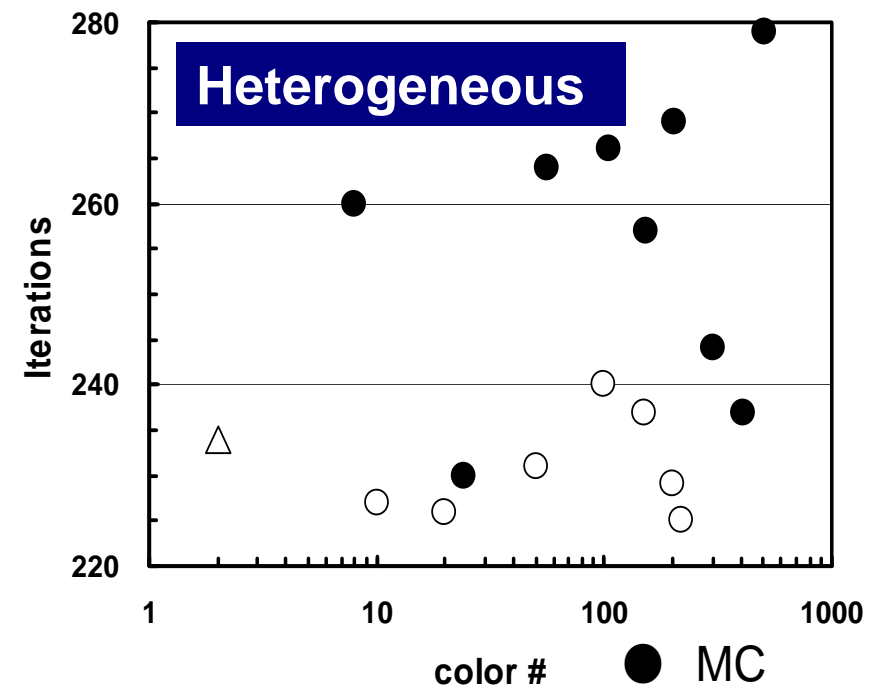
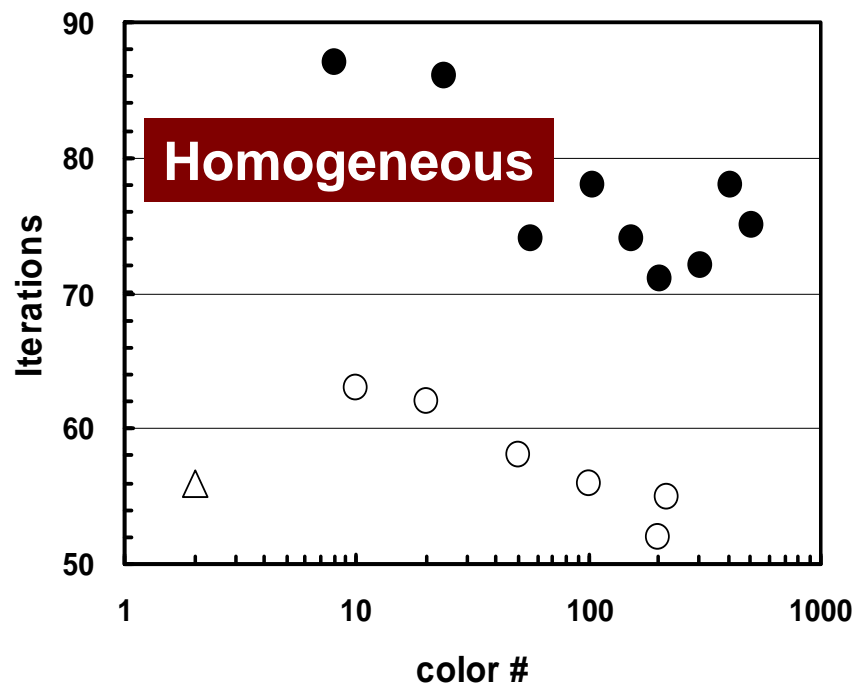
CM-RCM

- 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=- N_c 」とする
 - L2-solver, L3-solverで有効なオプション(但しFORTRAN版のみ)
- 実装は「cmrcm.f」を参照ください(本日は説明は省略)

オーダリング手法の比較

三次元弾性問題

- MCは収束遅い, 不安定(特に不均質(悪条件)問題)
- Cyclic-Mulricoloring + RCM(CM-RCM) が有効 [Washio et al. 2000]



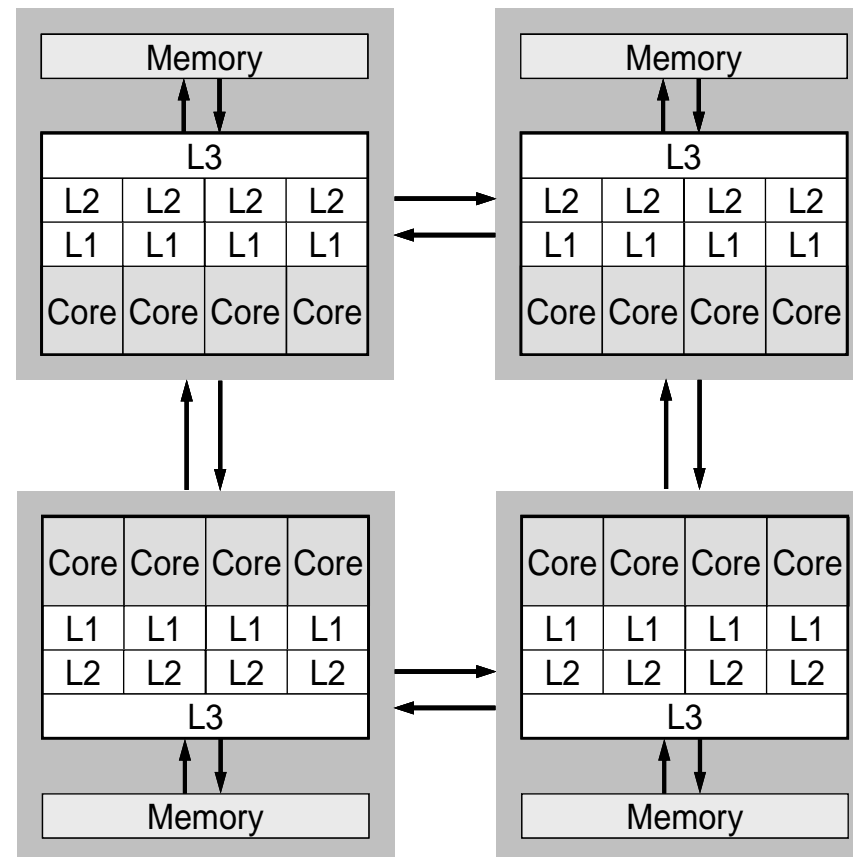
3D Linear-Elastic Problems with 32,768 DOF

- MC
- CM-RCM
- △ No reordering

- 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
- **T2K(東大)での実行**
- T2K(東大)での性能向上への道
 - NUMA Control
 - First Touch
 - データ再配置

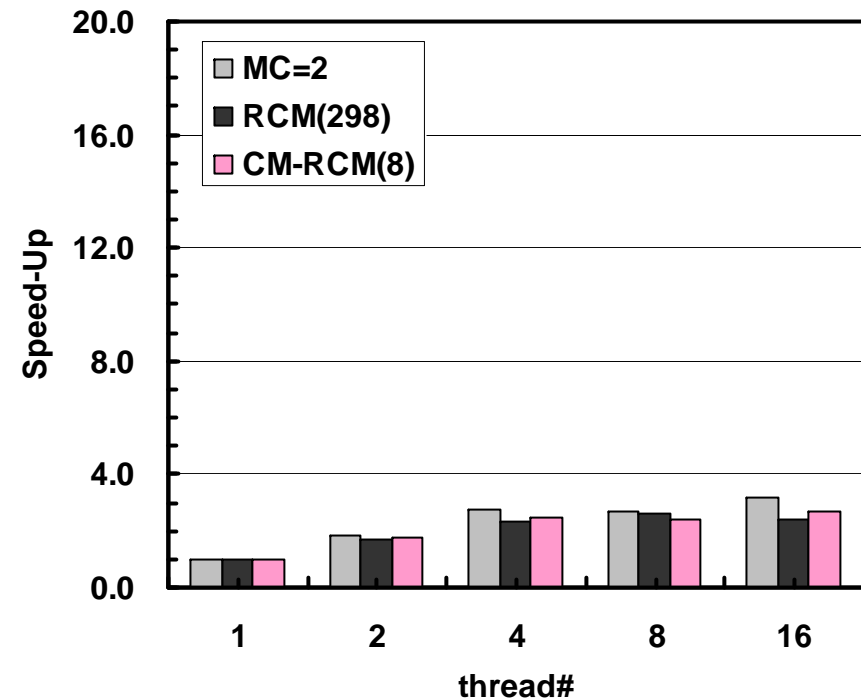
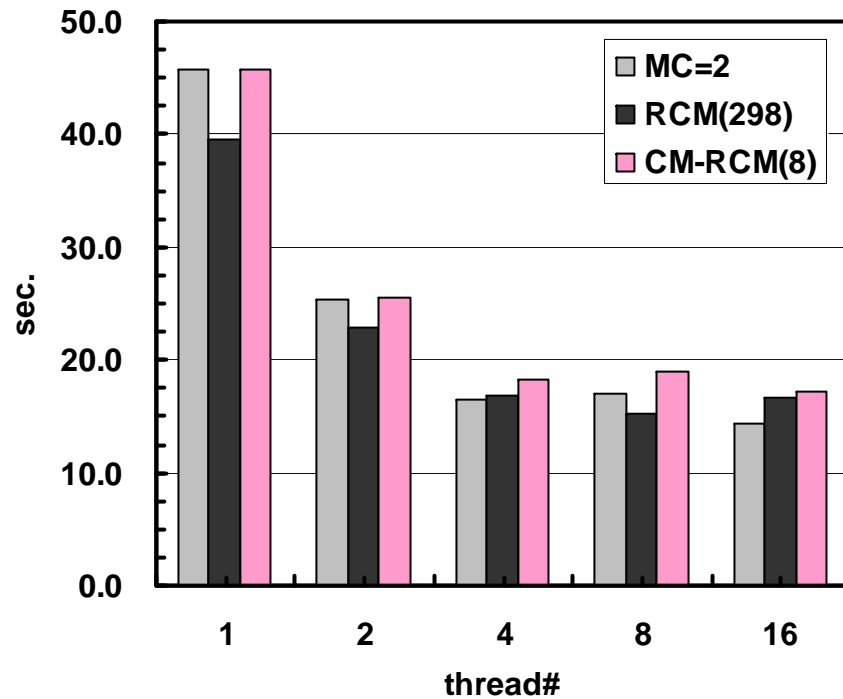
計算結果

- T2K(東大) 1ノード(4ソケット, 16コア)
- 100^3 要素



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 ($N_c=8$) : 268回



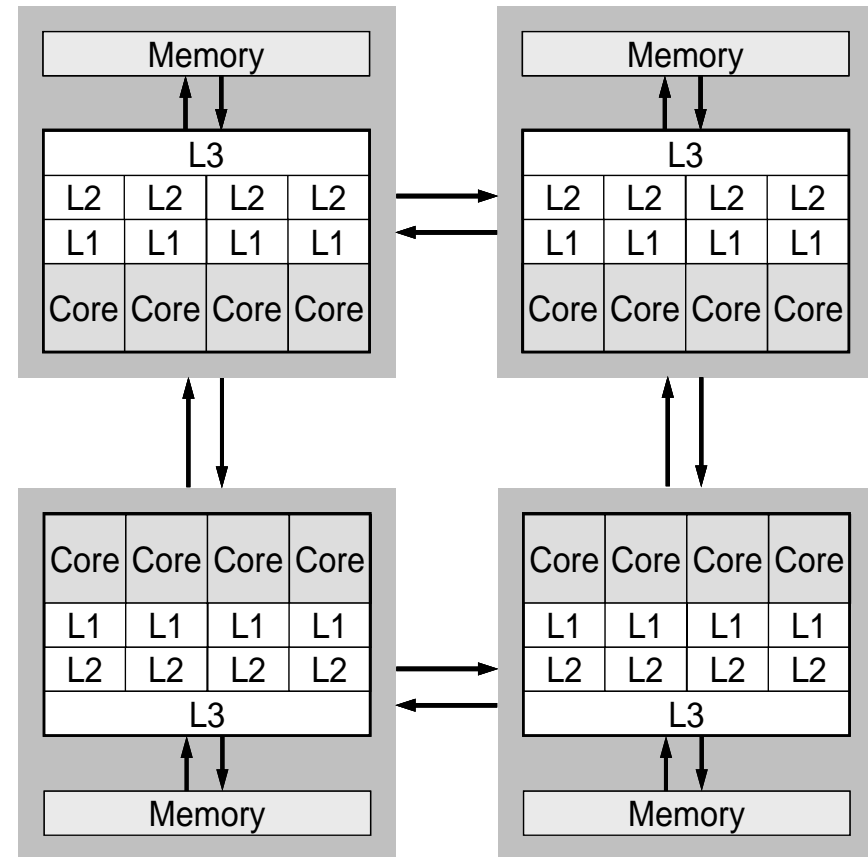
- L2-solへのOpenMPの実装
- Hitachi SR11000/J2での実行
 - 計算結果
 - CM-RCMオーダリング
- T2K(東大)での実行
- **T2K(東大)での性能向上への道**
 - **NUMA Control**
 - First Touch
 - データ再配置

性能向上への道

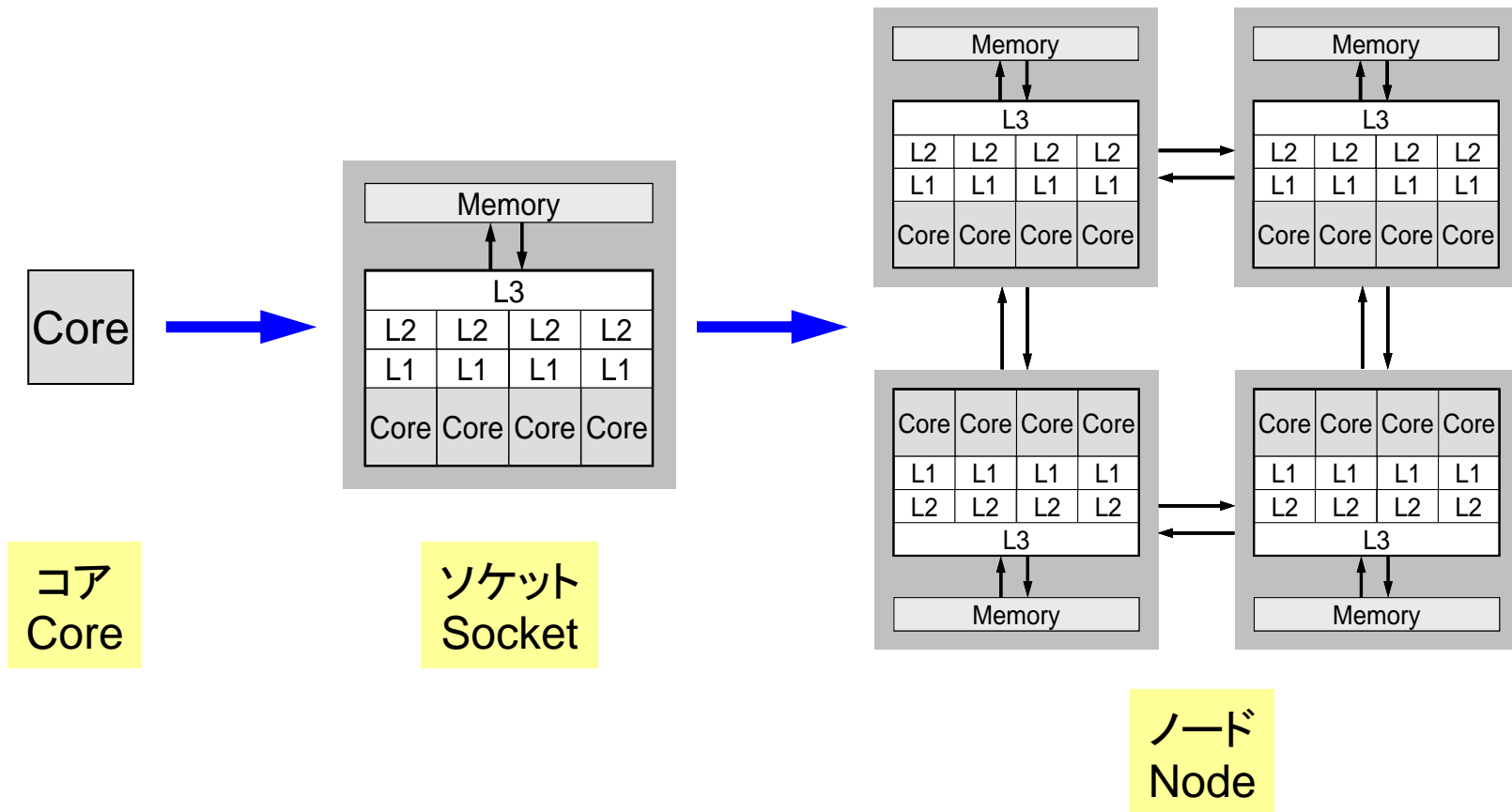
- CASE0 初期状態
- CASE1 NUMA control
- CASE2a First-touch(1)
- CASE2b First-touch(2)
- CASE3 First-touch + データ再配置

Quad Core Opteron: NUMA Architecture

- AMD Quad Core Opteron 2.3GHz
 - Quad Coreのソケット × 4
⇒ 1ノード(16コア)
- 各ソケットがローカルにメモリを持っている
 - NUMA: Non-Uniform Memory Access
 - できるだけ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



NUMA Architecture



numactl

- NUMA(Non Uniform Memory Access) 向けのメモリ割付のためのコマンドライン: Linuxでサポート
 - <http://www.novell.com/collateral/4621437/4621437.pdf>
- T2K(東大)でもある程度実績有り[Nakajima, K. 2008 (IEEE Cluster 2008)]

```
>$ numactl --show

policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
cpubind: 0 1 2 3
nodebind: 0 1 2 3
membind: 0 1 2 3
```

numactl --show

```
>$ numactl --show
```

```
policy: default
```

```
preferred node: current
```

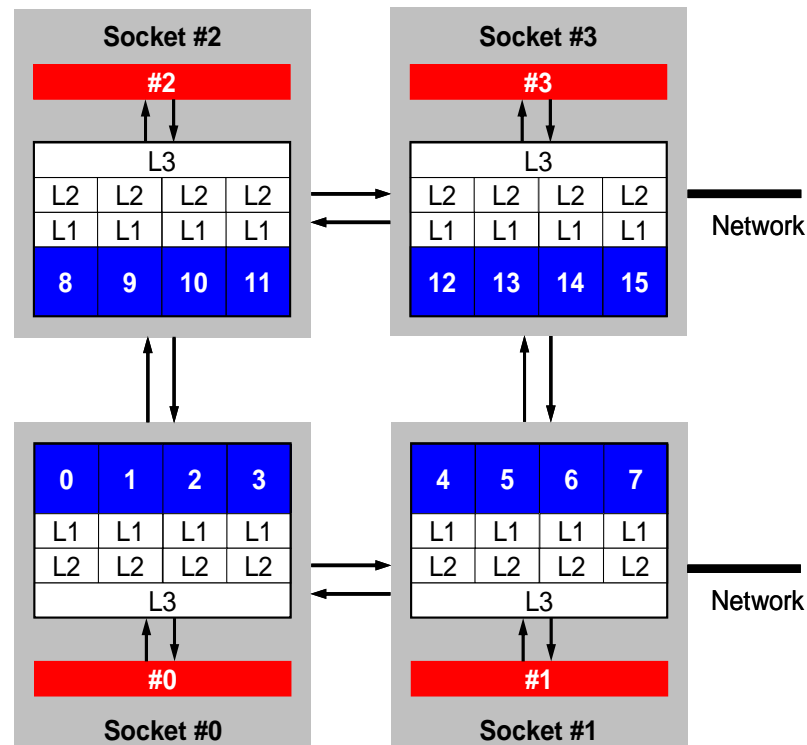
```
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

```
cpubind: 0 1 2 3
```

```
nodebind: 0 1 2 3
```

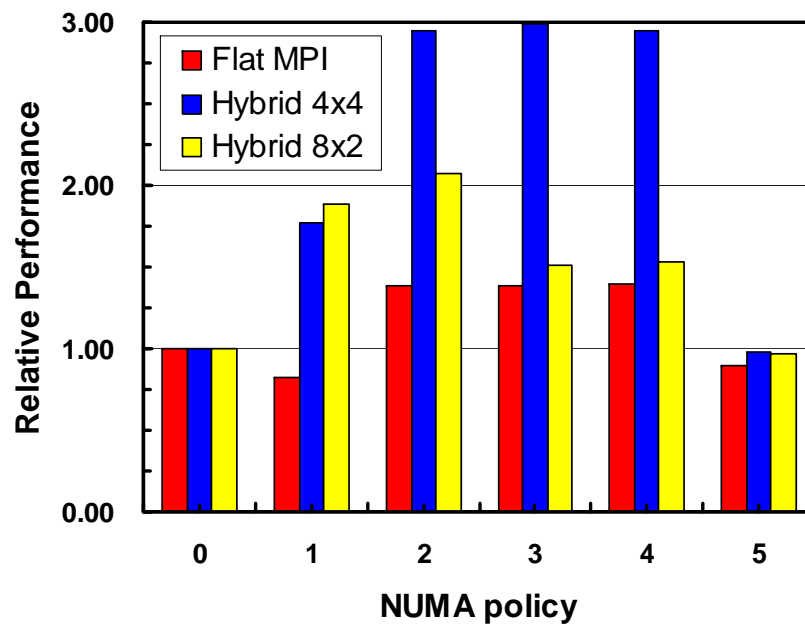
```
membind: 0 1 2 3
```

コア
ソケット
ソケット
メモリ



例: numactlの影響

- T2K(東大), 有限要素法アプリケーション
- POLICY=0: 何も指定しない場合
- 相対性能: 大きいほど良い

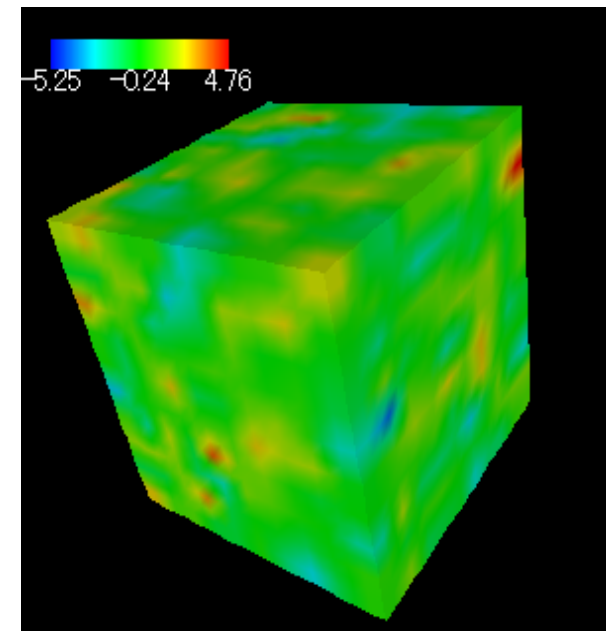


```
#@$-r HID-org
#@$-q h08nk132
#@$-N 24
#@$-J T4
#@$-e err
#@$-o x384-40-1-a.lst
#@$-lM 27GB
#@$-lE 03:00:00
#@$-s /bin/sh
#@$

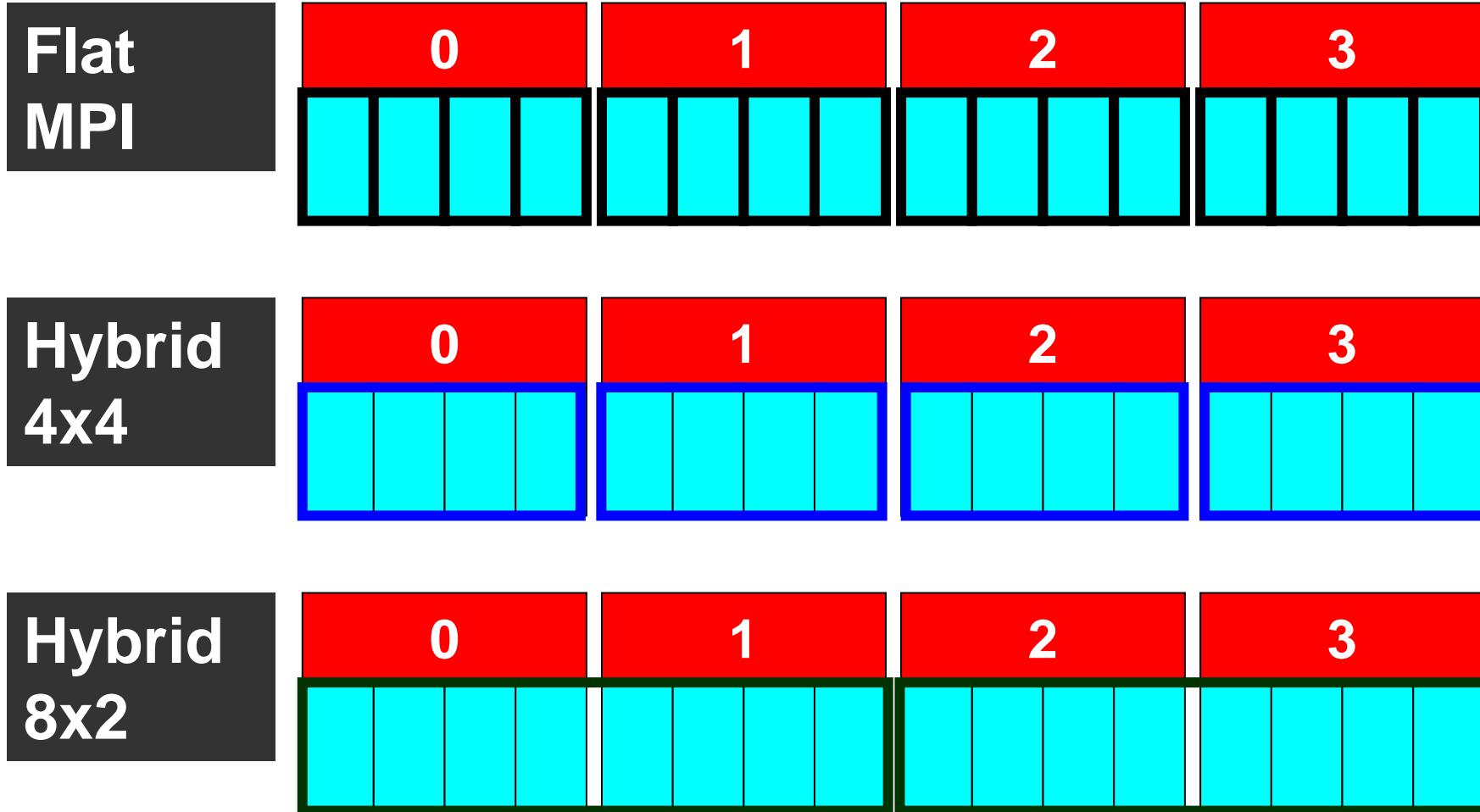
cd /XXX
mpirun ./numarun.sh ./sol
exit
```

Target Application

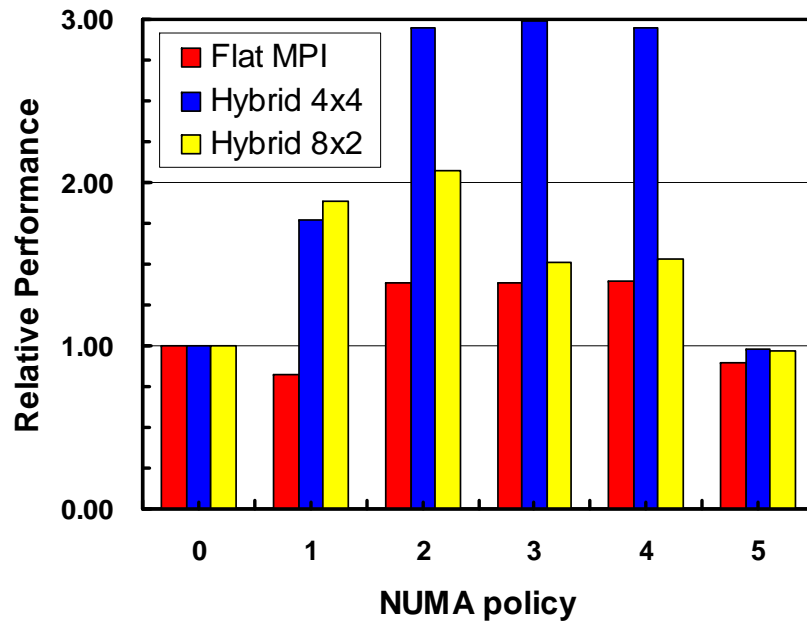
- 3D Elastic Problems with Heterogeneous Material Property
 - $E_{\max}=10^3$, $E_{\min}=10^{-3}$, $\nu=0.25$
 - generated by “sequential Gauss” algorithm for geo-statistics [Deutsch & Journel, 1998]
 - 128^3 tri-linear hexahedral elements, 6,291,456 DOF
- (SGS+GPBiCG) Iterative Solvers
 - Symmetric Gauss-Seidel
 - Original Block Jacobi, HID
- T2K/Tokyo
 - 512 cores (32 nodes)
- FORTARN90 (Hitachi) + MPI
 - Flat MPI, Hybrid (4x4, 8x2)
- Effect of NUMA Control



Flat MPI, Hybrid (4x4, 8x2)



numarun.shの中身



Policy:1

```
#!/bin/bash
MYRANK=$MXMPI_ID MPIのプロセス番号
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --interleave=all $@
```

Policy:2

```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --interleave=$SOCKET $@
```

Policy:3

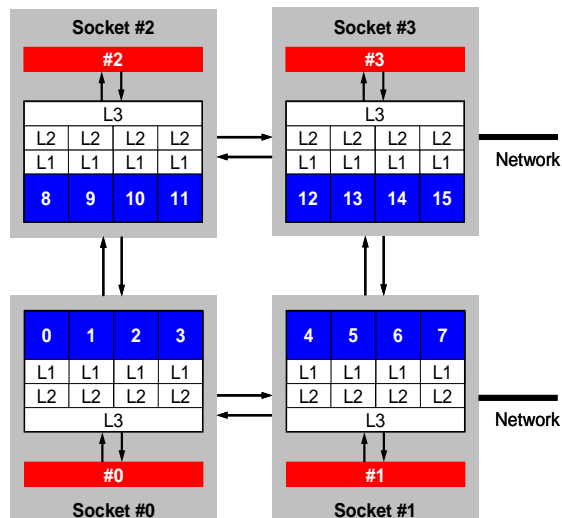
```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --membind=$SOCKET $@
```

Policy:4

```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --localalloc $@
```

Policy:5

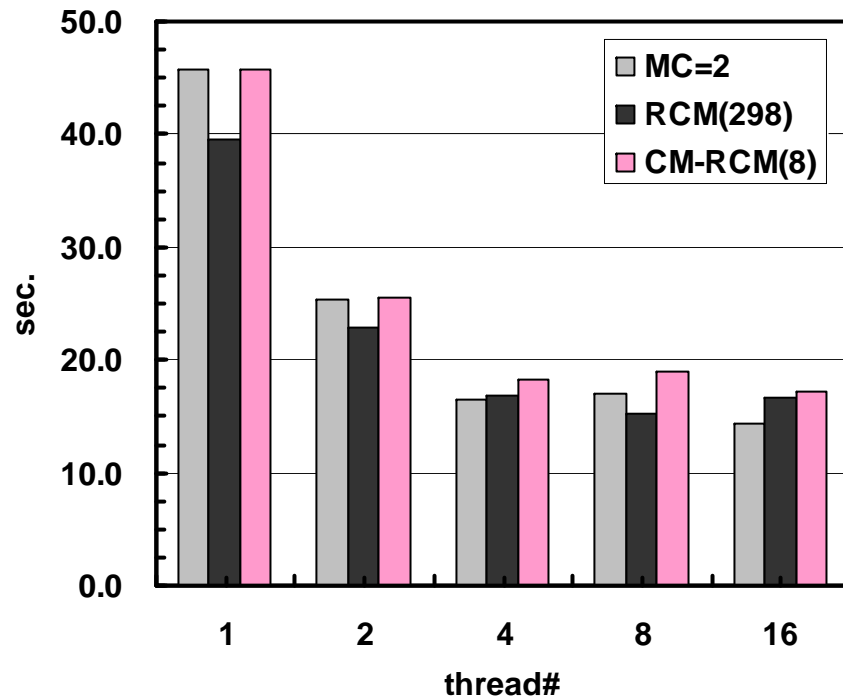
```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --localalloc $@
```



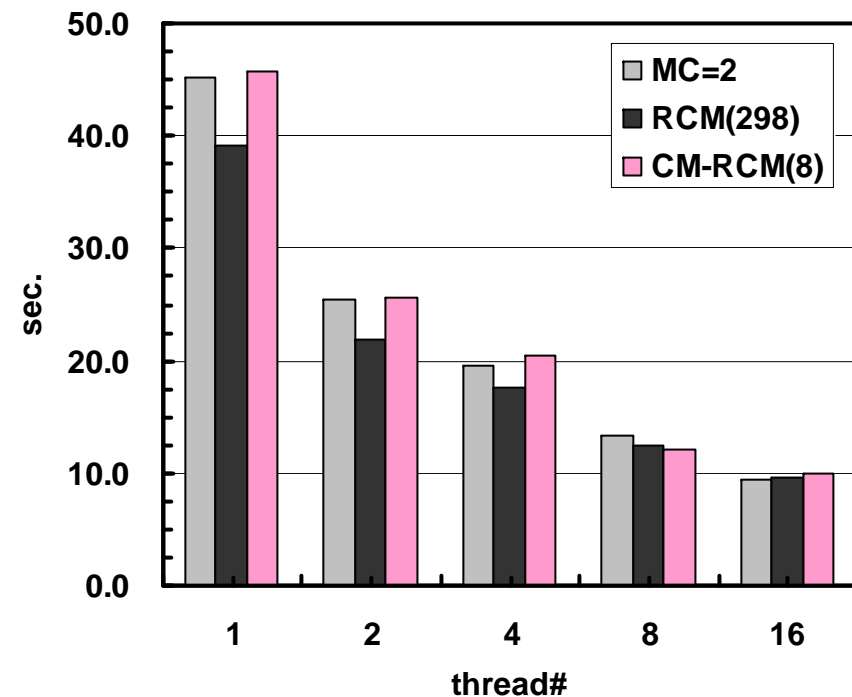
スケーラビリティ: numactlの効果

反復回数: MC (2色): 333回, RCM (298レベル): 224回
CM-RCM ($N_c=8$): 268回

少し良くなった...



numactl無し
CASE0



numactl有り
CASE1

numactlのポリシー

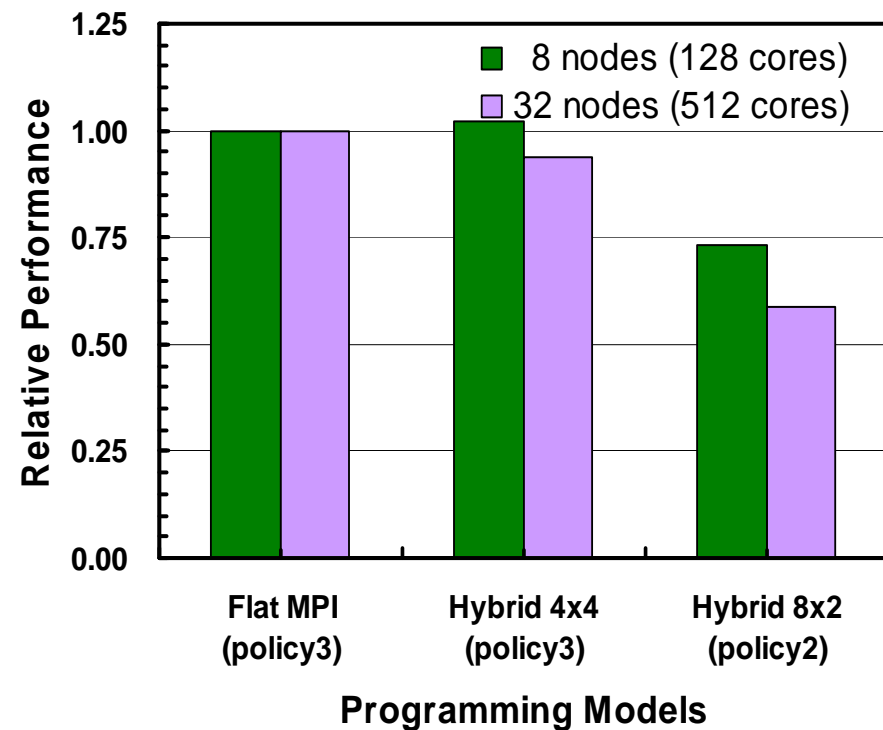
- 2 cores
 - `--cpunodebind= 0,0 --interleave= 0,0`
- 4 cores
 - `--cpunodebind= 0,0,0,0 --interleave= 0,0,0,0`
- 8 cores
 - `--cpunodebind= 0,0,1,1,2,2,3,3`
`--interleave = 0,0,1,1,2,2,3,3`
- 16 cores
 - `--cpunodebind= 0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3`
`--interleave = 0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3`

Effect of NUMA Policy: Best Cases

| Policy ID | command line switches |
|-----------|---------------------------------------------------------------------------|
| 0 | no command line switches |
| 1 | <code>--cpunodebind=\$SOCKET</code> <code>--interleave=all</code> |
| 2 | <code>--cpunodebind=\$SOCKET</code> <code>--interleave=\$SOCKET</code> |
| 3 | <code>--cpunodebind=\$SOCKET</code> <code>--membind=\$SOCKET</code> |
| 4 | <code>--cpunodebind=\$SOCKET</code> <code>--localalloc</code> |
| 5 | <code>--localalloc</code> |

8 nodes, 128 cores
50.3K DOF/core

32 nodes, 512 cores
12.6K DOF/core



- 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
- T2K(東大)での実行
- **T2K(東大)での性能向上への道**
 - NUMA Control
 - **First Touch**
 - データ再配置

プログラムのありか: FORTRANのみ

- 所在
 - `<$L3>/firsttouch`
- コンパイル, 実行方法
 - メッシュ生成
 - `cd <$L3>/firsttouch`
 - `f90 -O mg.f, cc -O mg.c`
 - 本体
 - `cd <$L3>/firsttouch`
 - `make`
 - コントロールデータ
 - `<$L3>/fristtouch/INPUT.DAT`
 - 実行用シェル
 - `<$L3>/firsttouch/go.sh`

制御データ (INPUT.DAT)

```

1.00e-00 1.00e-00 1.00e-00  DX/DY/DZ
0.10      1.0e-08             OMEGA, EPSICC
8                                     PEsmptTOT
100                                NCOLORtot
1  1                                METHOD, NFLAG

```

| 変数名 | 型 | 内容 |
|------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| DX, DY, DZ | 倍精度実数 | 各要素の3辺の長さ (ΔX , ΔY , ΔZ) |
| OMEGA | 倍精度実数 | 不使用 (適当な値を入れておく) |
| EPSICCG | 倍精度実数 | 収束判定値 |
| PEsmptTOT | 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 |
| METHOD | 整数 | 行列ベクトル積のループ構造 (0 : 従来通り, 1 : 前進後退代入と同じ) |
| NFLAG | 整数 | 0 : First-Touch無し, 1 : あり |

First Touch

```
!$omp parallel do private(ip, i, VAL, j)
  do ip= 1, PEsmptTOT
    do i = SMPindexG(ip-1)+1, SMPindexG(ip)
      VAL= D(i)*W(i, P)
      do j= 1, INL(i)
        VAL= VAL + AL(j, i)*W(IAL(j, i), P)
      enddo
      do j= 1, INU(i)
        VAL= VAL + AU(j, i)*W(IAU(j, i), P)
      enddo
      W(i, Q)= VAL
    enddo
  enddo
!$omp end parallel do
```

- 以下の配列に対する処理が必要
 - INL, INU, IAL, IAU
 - AL, AU, D, BFORCE, PHI (X)
- 以下については既に実施済み
 - W (ICCGの中)

現在の計算プロセス in POI_GEN

- メッシュデータより以下を求める
 - INL, INU, IAL, IAU
- Re-Ordering
 - INL, INU, IAL, IAUが入れ替わる
- この時点で新しい番号付けに対して以下を計算
 - AL, AU, D, BFORCE
- INL, INU, IAL, IAUは「古い」番号の状態です。First Touchが行われている
 - 新しい番号付けにおいてFirst Touchを行なう必要がある

First Touch

NFLAG=0

First-touch

無し (定義)

```

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

if (NFLAG.eq.0) then
  BFORCE= 0. d0
  D= 0. d0
  PHI= 0. d0
  INL= INLorg
  INU= INUorg
  IAL= IALorg
  IAU= IAUorg
  AL= 0. d0
  AU= 0. d0
else
  do ic= 1, NCOLORTot
!$omp parallel do private (ip, icel, k)
    do ip= 1, PEsmpTOT
      do icel= SMPindex((ic-1)*PEsmpTOT+ip-1)+1,
& SMPindex((ic-1)*PEsmpTOT+ip)
        D (icel)= 0. d0
        PHI (icel)= 0. d0
        BFORCE(icel)= 0. d0
        INL(icel)= INLorg(icel)
        INU(icel)= INUorg(icel)
        do k= 1, NL
          IAL(k, icel)= IALorg(k, icel)
          AL(k, icel)= 0. d0
        enddo
        do k= 1, NU
          IAU(k, icel)= IAUorg(k, icel)
          AU(k, icel)= 0. d0
        enddo
      enddo
    enddo
  enddo
enddo
endif

```

First Touch NFLAG=1

First-touch 有り (定義)

```

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

if (NFLAG.eq.0) then
  BFORCE= 0. d0
  D= 0. d0
  PHI= 0. d0
  INL= INLorg
  INU= INUorg
  IAL= IALorg
  IAU= IAUorg
  AL= 0. d0
  AU= 0. d0
else
  do ic= 1, NCOLORTot
!$omp parallel do private (ip, icel, k)
    do ip= 1, PEsmptOT
      do icel= SMPindex((ic-1)*PEsmptOT+ip-1)+1,
& SMPindex((ic-1)*PEsmptOT+ip)
        D (icel)= 0. d0
        PHI (icel)= 0. d0
        BFORCE(icel)= 0. d0
        INL(icel)= INLorg(icel)
        INU(icel)= INUorg(icel)
        do k= 1, NL
          IAL(k, icel)= IALorg(k, icel)
          AL(k, icel)= 0. d0
        enddo
        do k= 1, NU
          IAU(k, icel)= IAUorg(k, icel)
          AU(k, icel)= 0. d0
        enddo
      enddo
    enddo
  enddo
enddo
endif

```

行列ベクトル 積の計算法

METHOD=0
solver_ICCG_mc
CASE2a

METHOD=1
solver_ICCG_mc_ft
CASE2b

```
!$omp parallel do private(ip, i, VAL, j)
do ip= 1, PEsmptTOT
  do i = SMPindexG(ip-1)+1, SMPindexG(ip)
    VAL= D(i)*W(i, P)
    do j= 1, INL(i)
      VAL= VAL + AL(j, i)*W(IAL(j, i), P)
    enddo
    do j= 1, INU(i)
      VAL= VAL + AU(j, i)*W(IAU(j, i), P)
    enddo
    W(i, Q)= VAL
  enddo
enddo
!$omp end parallel do
```

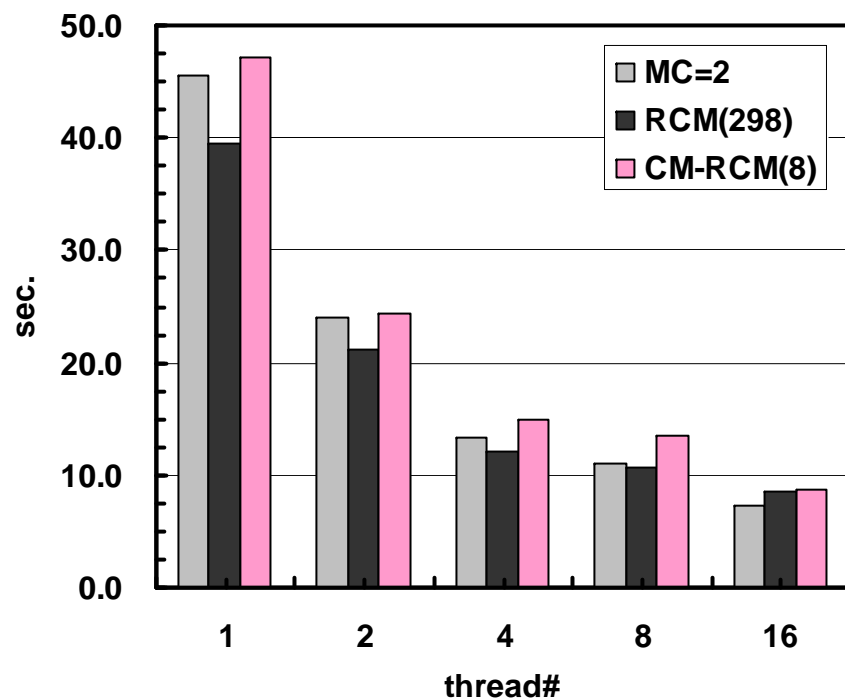
```
do ic= 1, NCOLortot
!$omp parallel do private(ip, ip1, i, VAL, j)
do ip= 1, PEsmptTOT
  ip1= (ic-1)*PEsmptTOT + ip
do i= SMPindex(ip1-1)+1, SMPindex(ip1)
  VAL= D(i)*W(i, P)
  do j= 1, INL(i)
    VAL= VAL + AL(j, i)*W(IAL(j, i), P)
  enddo
  do j= 1, INU(i)
    VAL= VAL + AU(j, i)*W(IAU(j, i), P)
  enddo
  W(i, Q)= VAL
enddo
enddo
!$omp end parallel do
enddo
```

スケーラビリティ: First-touchの効果

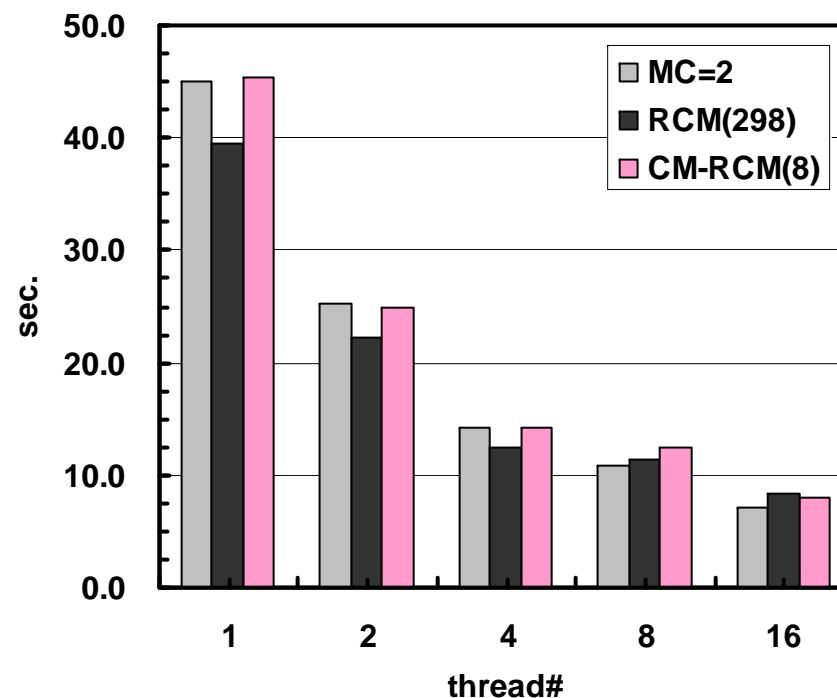
反復回数: MC (2色): 333回, RCM (298レベル): 224回

CM-RCM ($N_c=8$): 268回

更に少し良くなった...



CASE2a
solver_ICCG_mc



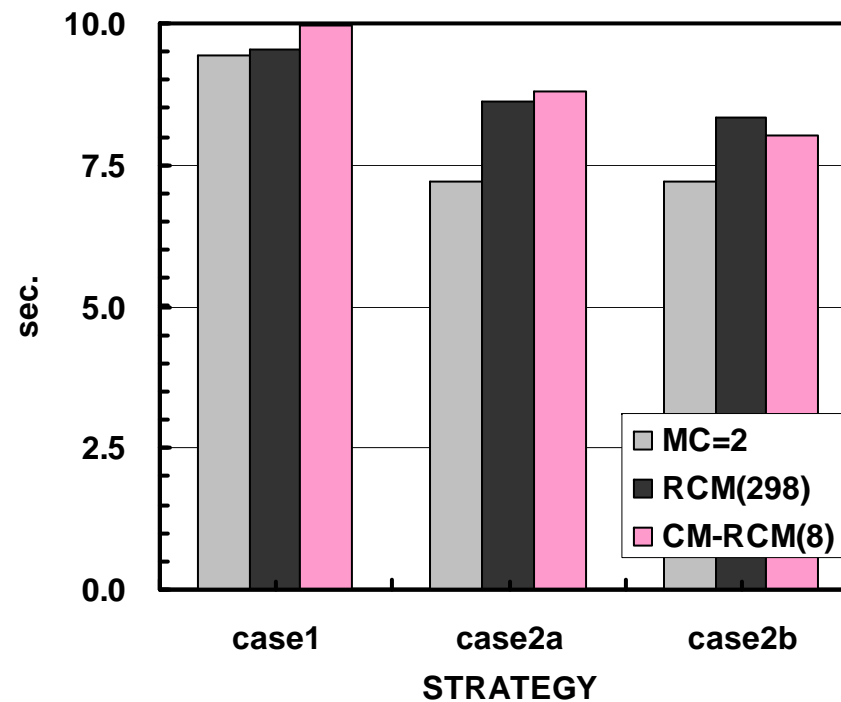
CASE2b
solver_ICCG_mc_ft

スケーラビリティ: First-touchの効果

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

CM-RCM ($N_c=8$): 268回

16コア時における比較



- 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
- T2K(東大)での実行
- **T2K(東大)での性能向上への道**
 - NUMA Control
 - First Touch
 - **データ再配置**

現在のオーダリングの問題

- 色付け
 - MC
 - RCM
 - CM-RCM
- 同じ色に属する要素は独立：並列計算可能
- 「色」の順番に番号付け
- 色内の要素を各スレッドに振り分ける

- 同じスレッド(すなわち同じコア)に属する要素は連続の番号ではない
 - 効率の低下

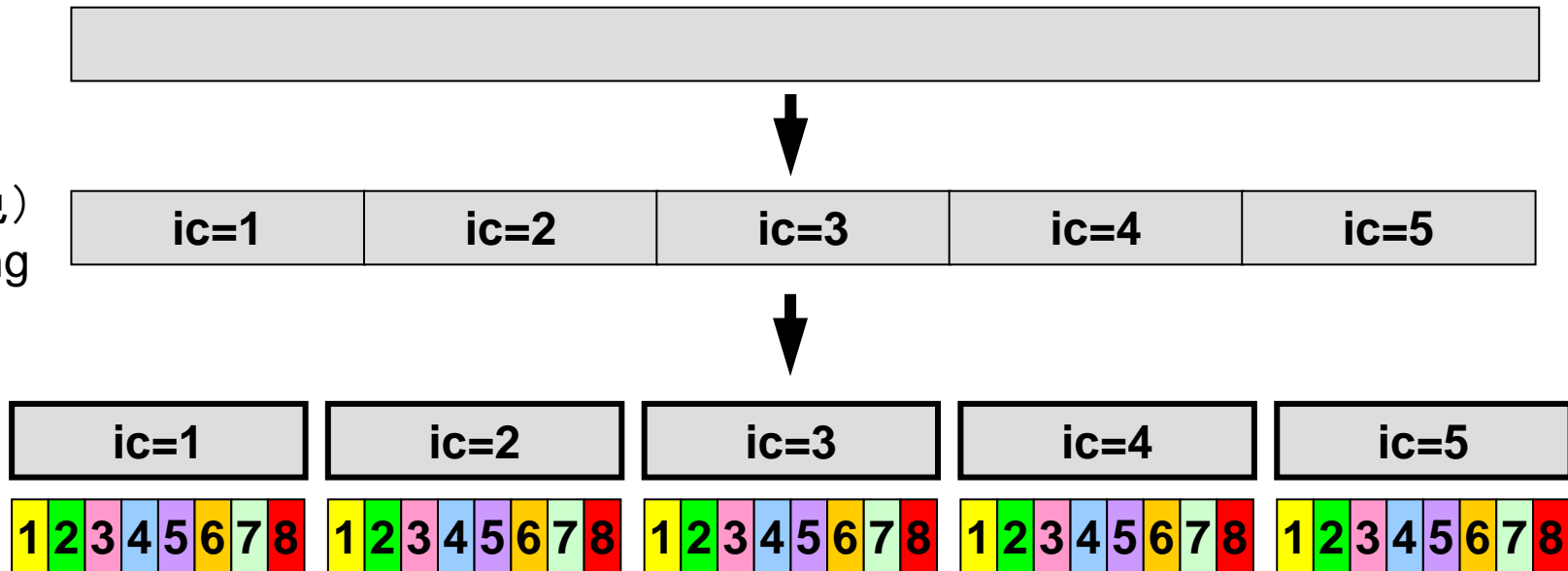
SMPindex: 前処理向け

```

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

```

彩色(5色)
+Ordering



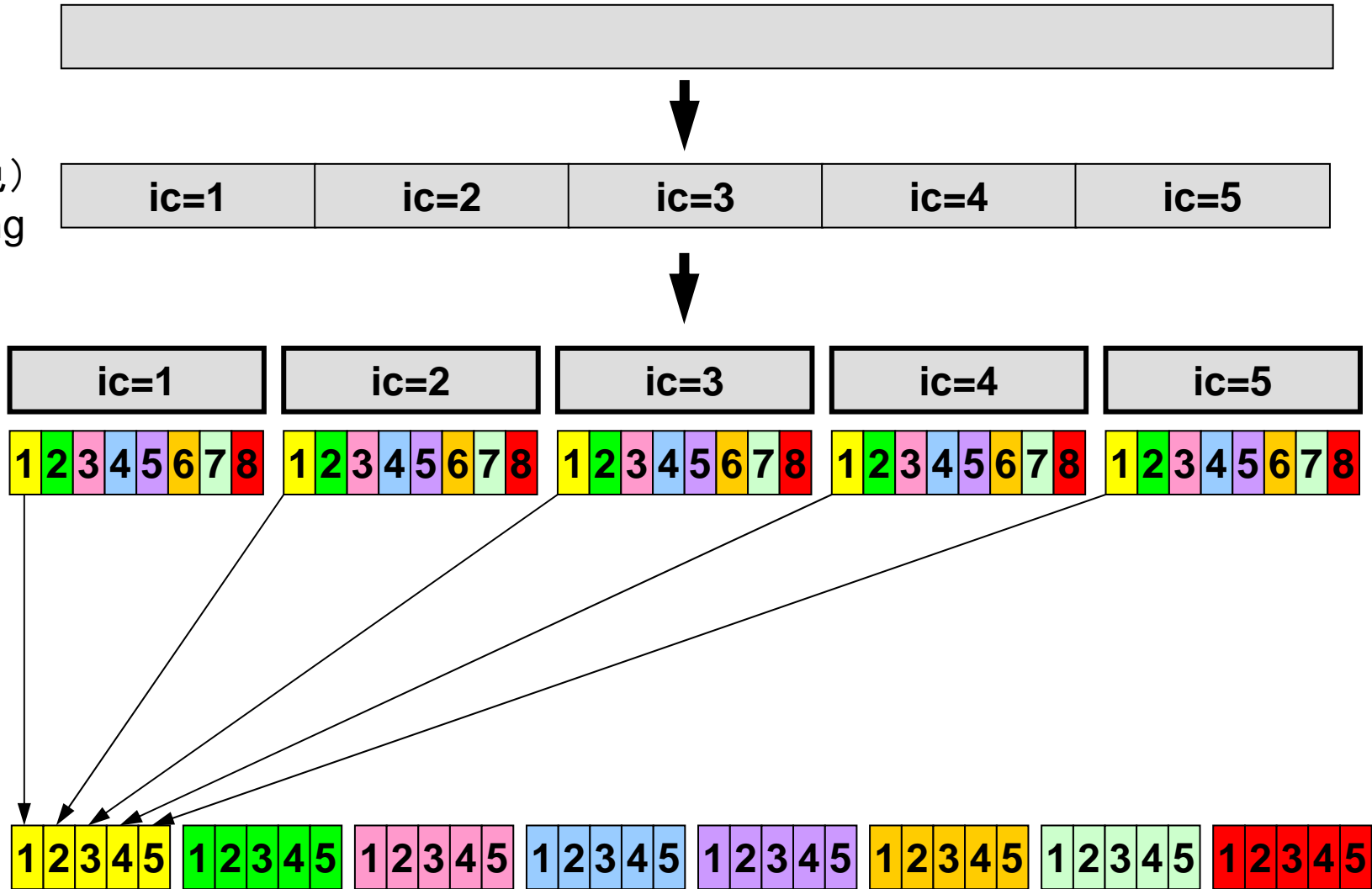
各色内の要素は互いに独立: 並列に計算可能⇒スレッド数で更に分割

データ再配置

- 同じスレッドで処理するデータをなるべく連続に配置するように更に並び替え
 - 効率の向上が期待される
- 番号の付け替えによって要素の大小関係は変わるが、上三角、下三角の関係は変えない(もとの計算と反復回数は変わらない)
 - 従って自分より要素番号が大きいのにIAL(下三角)に含まれていたりする
- First-touch + データ再配置 : CASE3

データ再配置

彩色(5色)
+Ordering



プログラムのありか: FORTRANのみ

- 所在
 - `<$L3>/reorder`
- コンパイル, 実行方法
 - メッシュ生成
 - `cd <$L3>/reorder`
 - `f90 -O mg.f, cc -O mg.c`
 - 本体
 - `cd <$L3>/reorder`
 - `make`
 - コントロールデータ
 - `<$L3>/reorder/INPUT.DAT`
 - 実行用シェル
 - `<$L3>/reorder/go.sh`

制御データ (INPUT.DAT)

```

1.00e-00 1.00e-00 1.00e-00   DX/DY/DZ
0.10      1.0e-08             OMEGA, EPSICC
8                                                PEsmptTOT
100                                              NCOLORtot
1                                                NFLAG

```

| 変数名 | 型 | 内 容 |
|------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| DX, DY, DZ | 倍精度実数 | 各要素の3辺の長さ (ΔX , ΔY , ΔZ) |
| OMEGA | 倍精度実数 | 不使用 (適当な値を入れておく) |
| EPSICCG | 倍精度実数 | 収束判定値 |
| PEsmptTOT | 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 |
| NFLAG | 整数 | 0 : First-Touch無し, 1 : あり |

再配置

```

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

```

SMPindex



SMPindex_new



```

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

```

行列ベクトル積の計算法(内積等も同様)

```

!$omp parallel do private(ip, i, VAL, j)
  do ip= 1, PEsmptOT
    do i = SMPindexG(ip-1)+1, SMPindexG(ip)
      VAL= D(i)*W(i, P)
      do j= 1, INL(i)
        VAL= VAL + AL(j, i)*W(IAL(j, i), P)
      enddo
      do j= 1, INU(i)
        VAL= VAL + AU(j, i)*W(IAU(j, i), P)
      enddo
      W(i, Q)= VAL
    enddo
  enddo
!$omp end parallel do

```

Original

```

!$omp parallel do private(ip, i, VAL, j)
  do ip= 1, PEsmptOT
    do i= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
      VAL= D(i)*W(i, P)
      do j= 1, INL(i)
        VAL= VAL + AL(j, i)*W(IAL(j, i), P)
      enddo
      do j= 1, INU(i)
        VAL= VAL + AU(j, i)*W(IAU(j, i), P)
      enddo
      W(i, Q)= VAL
    enddo
  enddo
!$omp end parallel do

```

New

前進代入の計算法

```

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

```

Original

```

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

```

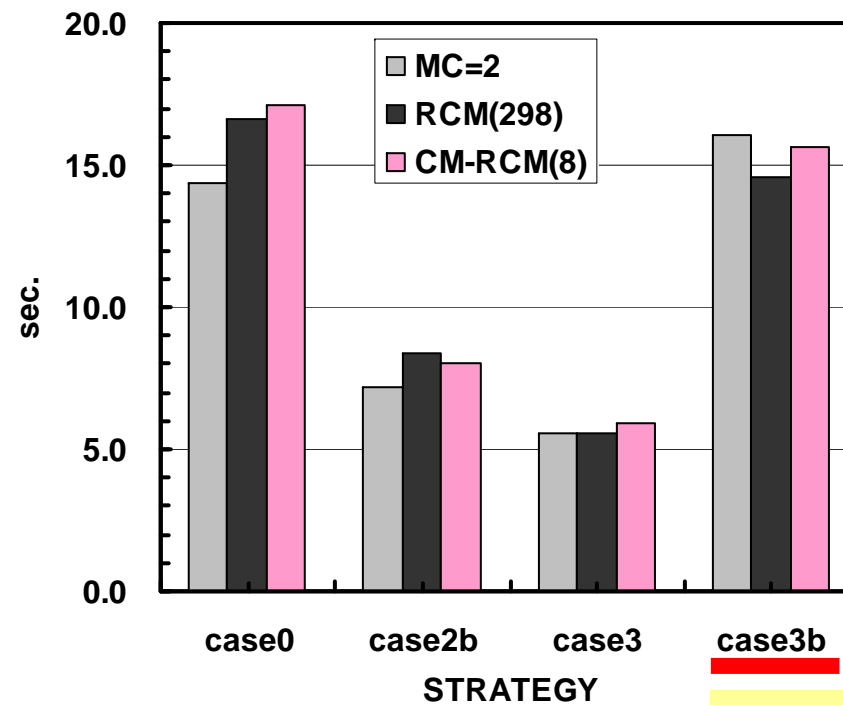
New

First-touch+再配置 の効果

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

CM-RCM ($N_c=8$) : 268回

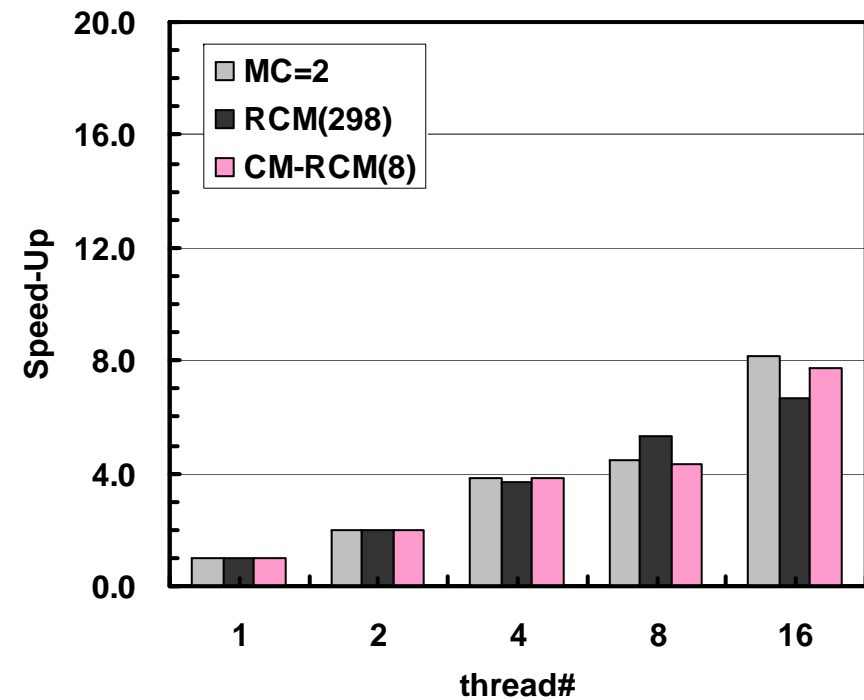
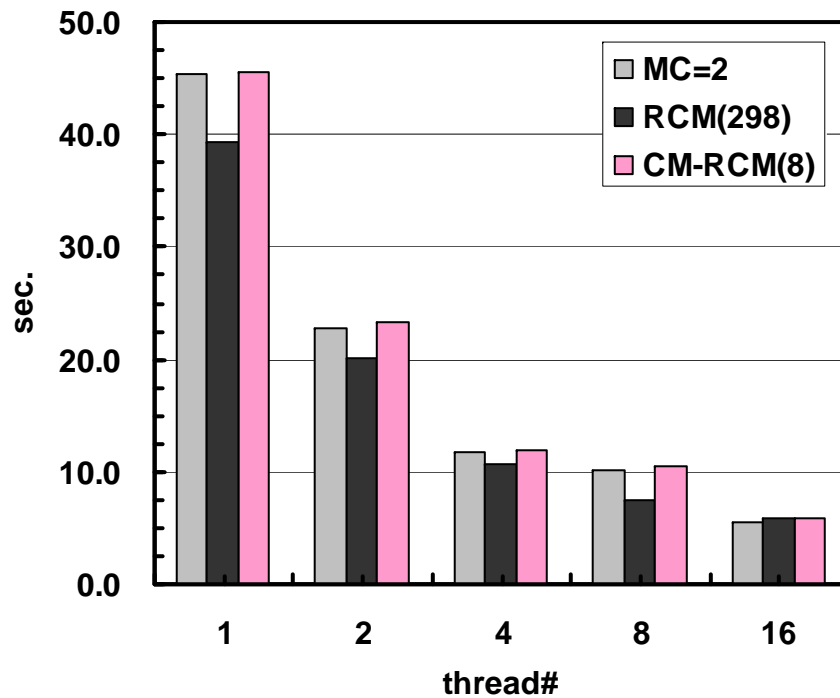
16コア時における比較



First-touch無し

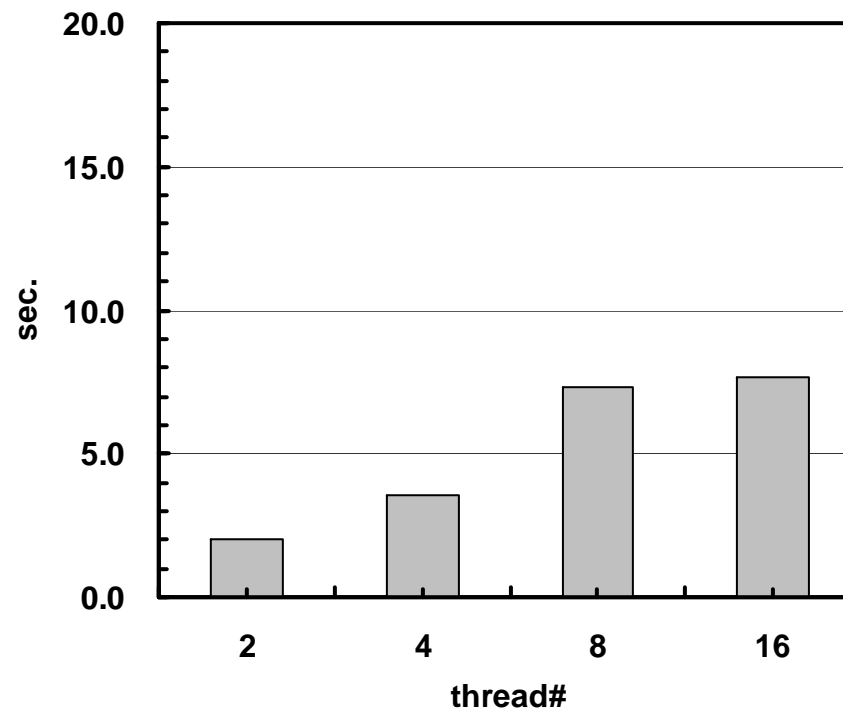
First-touch+再配置 の効果

反復回数: MC(2色): 333回, RCM(298レベル): 224回
CM-RCM($N_c=8$): 268回



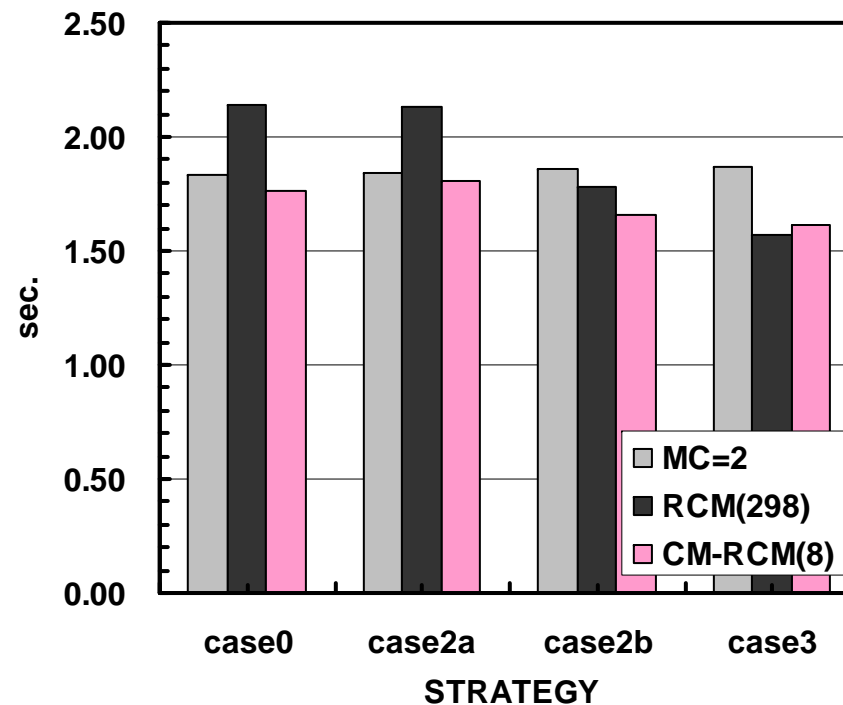
Flat-MPI: 三次元弾性問題の効率 約1.12M DOF

8⇒16コアでほとんど性能向上が見られない
OpenMPと同様にメモリがボトルネックとなっている



Hitachi SR11000/J2での結果: 16コア

- First-touchの有無
 - ほとんど影響ない
- オーダリング法によって影響が異なる



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，T2Kオープンスパコン(東大)を利用した実習

- オーダリングの効果
- First-touch
- データ再配置

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
- 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?
- 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
- 本講習会の内容が少しでも役に立てば幸いである

終わりに

- T2K(東大)
 - 12月7日(日)17:30まで利用可能
- 最終版の資料はWEBにアップ
 - <http://nkl.cc.u-tokyo.ac.jp/seminars/0812-JSIAM/>
- アンケートにお答えください