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、大島聡史、林雅江

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

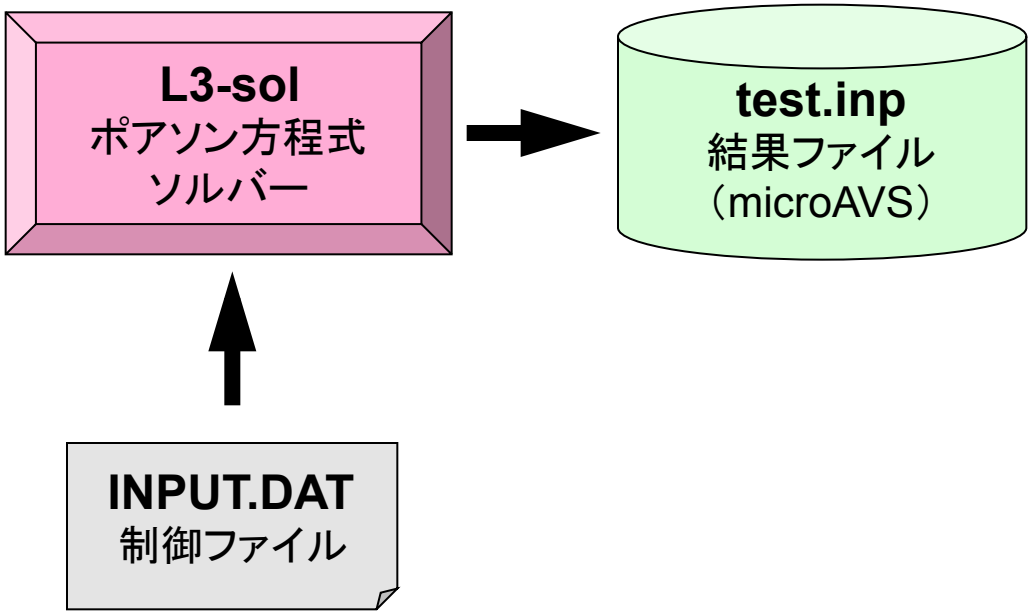
プログラムのありか

- 所在
  - <$L3>/src, <$L3>/run
- コンパイル，実行方法
  - 本体
    - cd <$L3>/src
    - Make
    - <$L3>/run/L3-sol(実行形式)
  - コントロールデータ
    - <$L3>/run/INPUT.DAT
  - 実行用シェル
    - <$L3>/run/x0.sh,x5.sh,x6.sh

実行例

```
% cd <$L3>
% ls
    run    src    ft    reorder    hid
% cd src
% make
% cd ../run
% ls L3-sol
    L3-sol

% <modify “INPUT.DAT”>
% <modify “x0.sh”>

% qsub x0.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

- 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：Sequential Reordering
- 他システムとの比較

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 ⇒ 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 ⇒ 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, PEsmpTOT,                &
     &        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

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 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)
```

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) :: PEsmpTOT

      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

PEsmpTOT：スレッド数

module PCG（これまでとの相違点）

```
module PCG

integer, parameter :: N2= 256
integer :: NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL
integer :: NPL, NPU
integer :: METHOD, ORDER_METHOD

real(kind=8) :: EPSICCG

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

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

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

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

NCOLORtot	色数
COLORindex(0:NCOLORtot)	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**	非零下三角成分(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≧：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
```

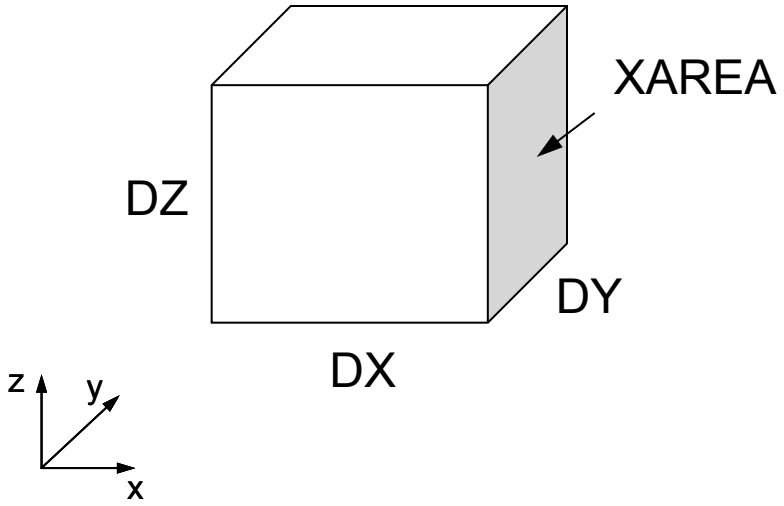
- PEsmpTOT
  - OpenMPスレッド数
- NCOLORtot
  - 色数
  - 「＝0」の場合はCM
  - 「＝-1」の場合はRCM
  - 「≦-2」の場合はCM-RCM

```
100 100 100                          NX/NY/NZ
1.00e-02 5.00e-02 1.00e-02           DX/DY/DZ
1.00e-08                             EPSICCG
16                                   PEsmpTOT
100                                  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)

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

      VOLCEL=  VO
      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 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)
```

poi_gen(1/9)

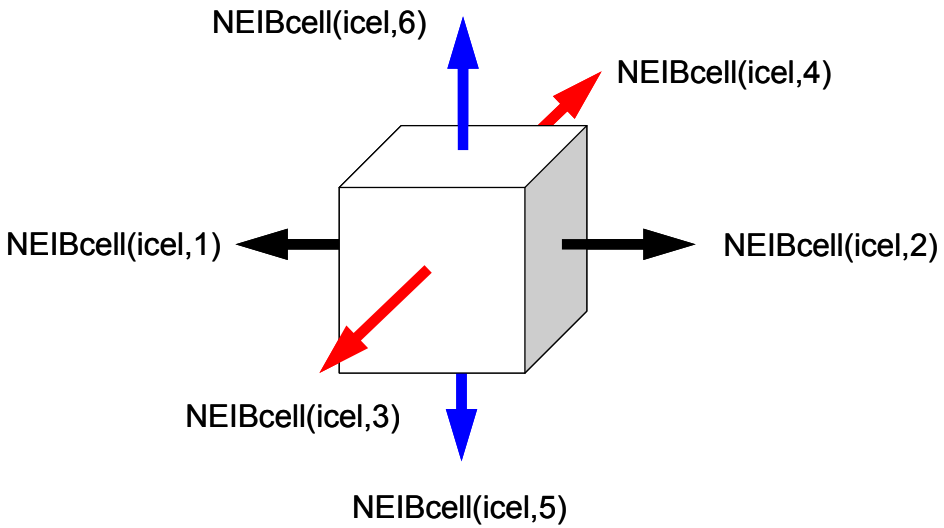
```
      subroutine POI_GEN

      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	非零下三角成分(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 | 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)

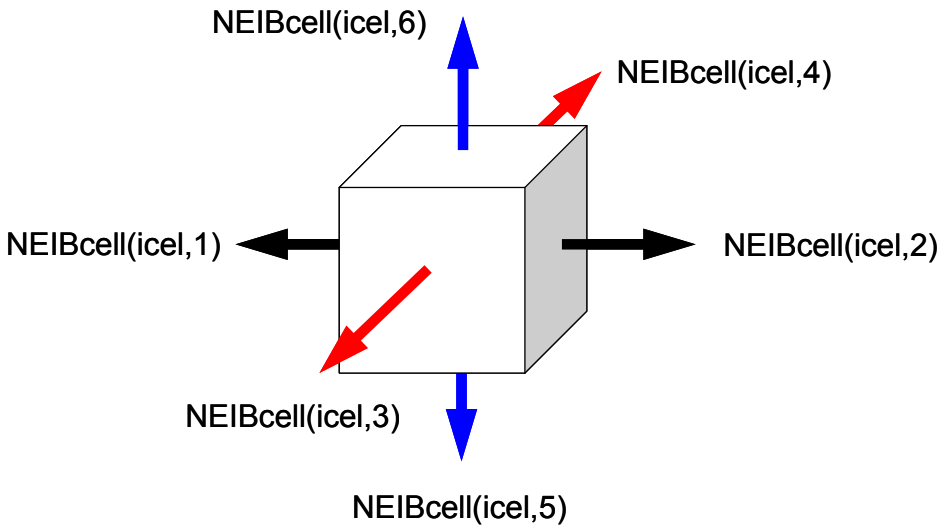


下三角成分

NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,1)= icel – 1

```
!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

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：
COLORindex(ic)-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 ⇒ OpenMP適用

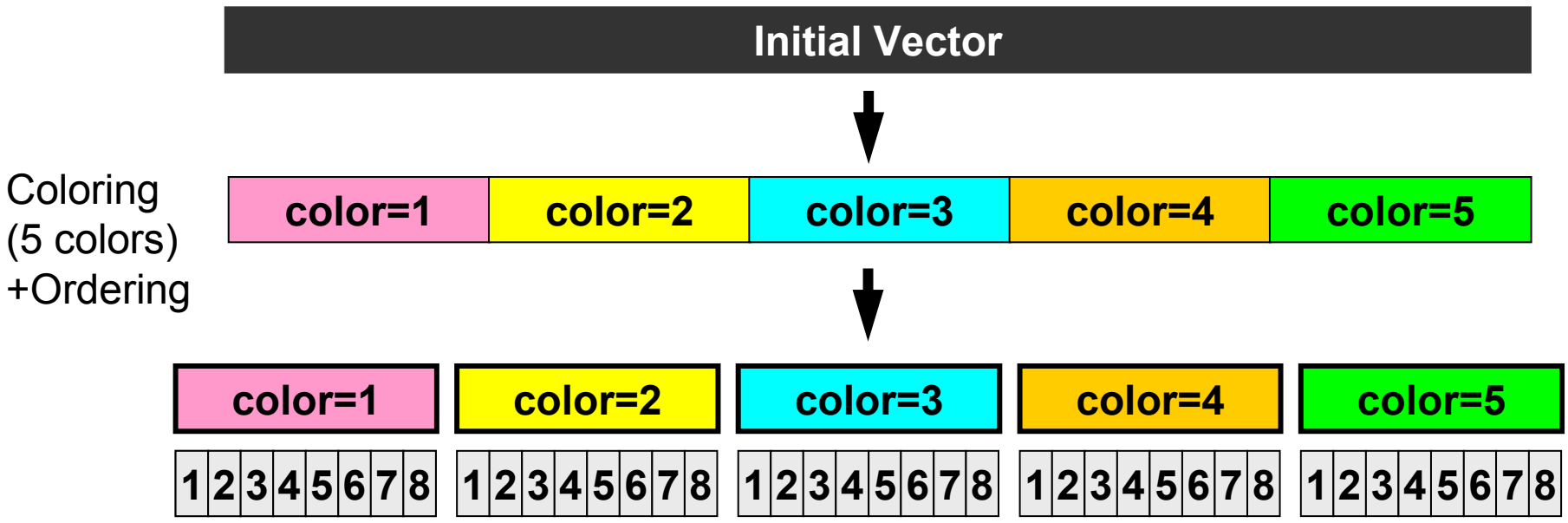
これを更に「PEsmpTOT」で割って
「SMPindex」に割り当てる。

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
```



- 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
```

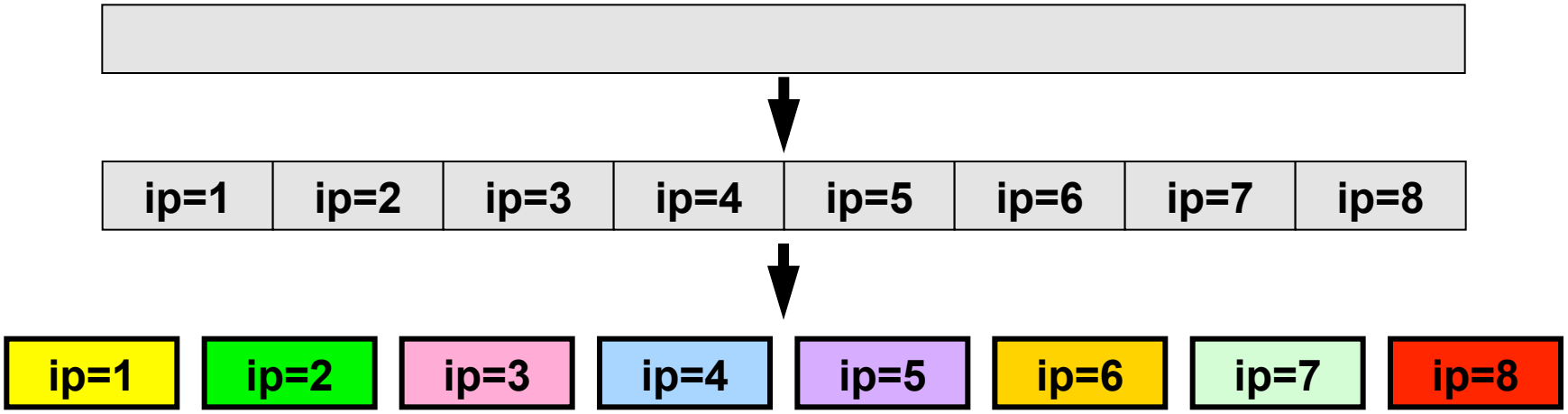
全要素数を「PEsmpTOT」で割って
「SMPindexG」に割り当てる。

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

poi_gen(5/9)

これ以降は新しい番号付けを使用

```
!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

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	非零下三角成分(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, PEsmpTOT
      do icel= SMPindexG(ip-1)+1, SMPindexG(ip)
        ic0 = NEWtoOLD(icel)

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

        VOL0= VOLCEL  (ic0)

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

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

icel: 新しい番号
ic0: 古い番号

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)

        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
```

icel: 新しい番号
ic0: 古い番号

poi_gen(6/9)

新しい番号付けを使用

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

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

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

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

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

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

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

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

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

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

        VOL0= VOLCEL  (ic0)

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

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

poi_gen(6/9)

新しい番号付けを使用

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

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

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

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

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

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

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

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

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

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

        VOL0= VOLCEL  (ic0)

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

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

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

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

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

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

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

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

```
!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= VOLCEL  (ic0)

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

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

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

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

icN5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
```

poi_gen(7/9)

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

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

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

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

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

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

```
      if (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
```

poi_gen(8/9)

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

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

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

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

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

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

poi_gen(9/9)

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

...

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

          if (icN6.lt.icel) then
            do j= 1, INL(icel)
              if (IAL(j,icel).eq.icN6) then
                itemL(j+indexL(icel-1))= icN6
                   AL(j+indexL(icel-1))= coef
                exit
              endif
            enddo
           else
            do j= 1, INU(icel)
              if (IAU(j,icel).eq.icN6) then
                itemU(j+indexU(icel-1))= icN6
                   AU(j+indexU(icel-1))= coef
                exit
              endif
            enddo
          endif
        endif
        ii= XYZ(ic0,1)
        jj= XYZ(ic0,2)
        kk= XYZ(ic0,3)

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

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

もとの座標に従って
BFORCE計算

ii,jj,kk,VOL0はprivate

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

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

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

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

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
```

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

solve_ICCG_mc(4/6)

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

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

for i= 1, 2, …

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

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

  if i=1

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

  else

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

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

  endif

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

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

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

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

  check convergence |r|

end

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

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

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

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
```

solve_ICCG_mc (5/6)

```
!C +----------------------------+
!C | {p} = {z} if      ITER=1   |
!C | BETA= RHO / RHO1  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 / RHO1
!$omp parallel do private(ip,i)
        do ip= 1, PEsmpTOT
        do i = SMPindexG(ip-1)+1, SMPindexG(ip)
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
        enddo
!$omp end parallel do
      endif
!C===

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

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

for i= 1, 2, …

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

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

  if i=1

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

  else

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

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

  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 / RHO1  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 / RHO1
!$omp parallel do private(ip,i)
        do ip= 1, PEsmpTOT
        do i = SMPindexG(ip-1)+1, SMPindexG(ip)
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
        enddo
!$omp end parallel do
      endif
!C===

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

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

for i= 1, 2, …

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

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

  if i=1

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

  else

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

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

  endif

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

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

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

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

  check convergence |r|

end

solve_ICCG_mc (6/6)

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

      ALPHA= RHO / C1
!C===


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

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

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

for i= 1, 2, …

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

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

  if i=1

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

  else

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

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

  endif

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

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

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

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

  check convergence |r|

end

solve_ICCG_mc (6/6)

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

      ALPHA= RHO / C1
!C===


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

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

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

for i= 1, 2, …

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

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

  if i=1

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

  else

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

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

  endif

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

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

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

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

  check convergence |r|

end

solve_ICCG_mc (6/6)

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

      ALPHA= RHO / C1
!C===


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

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

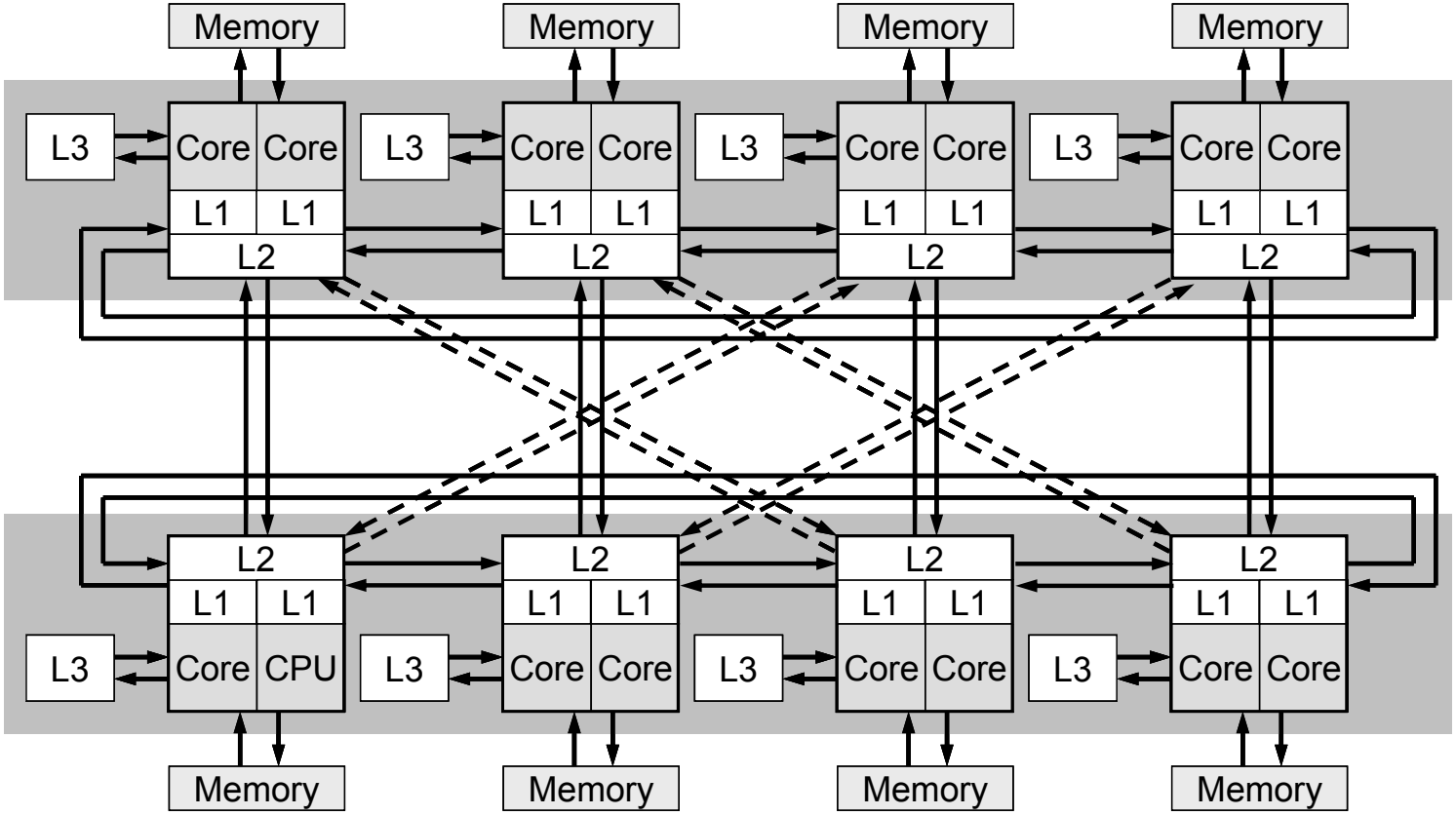
Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$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

- 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：Sequential Reordering
- 他システムとの比較

計算結果

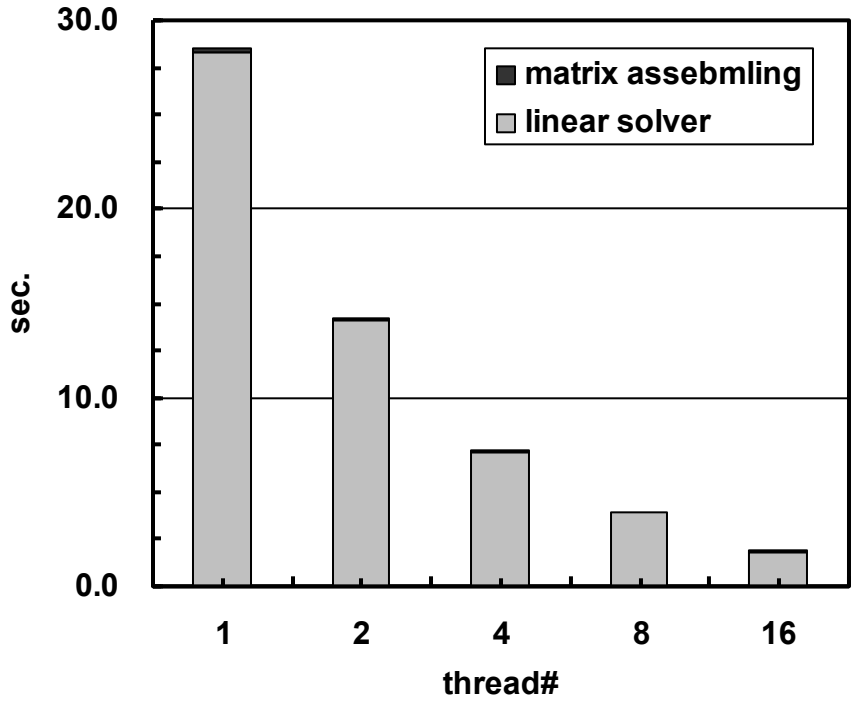
- Hitachi SR11000/J2　1ノード（16コア）
- 100^3要素



計算時間（MC=2）

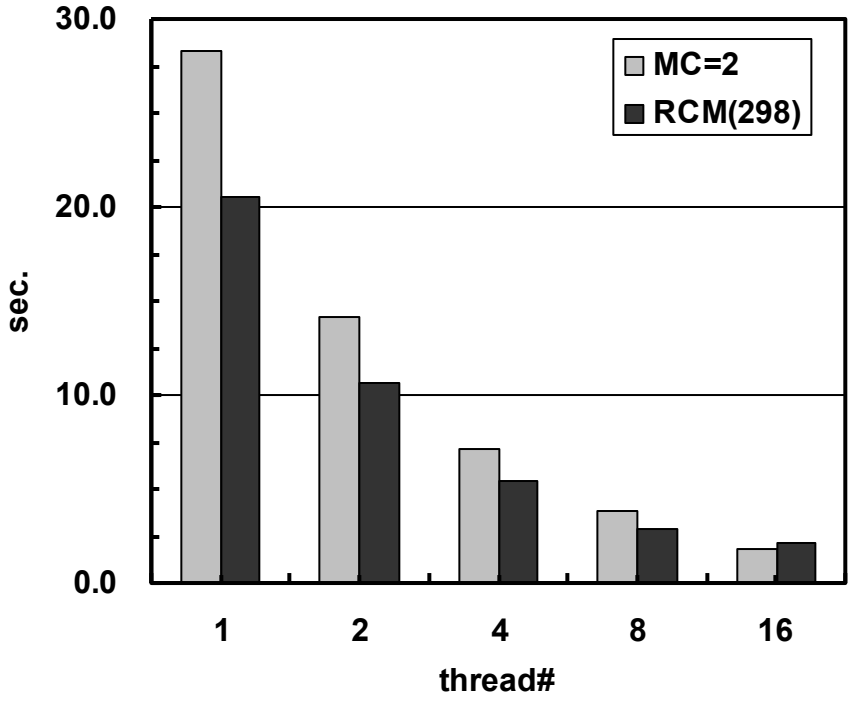
以降，「linear solver」の計算時間のみ問題とする

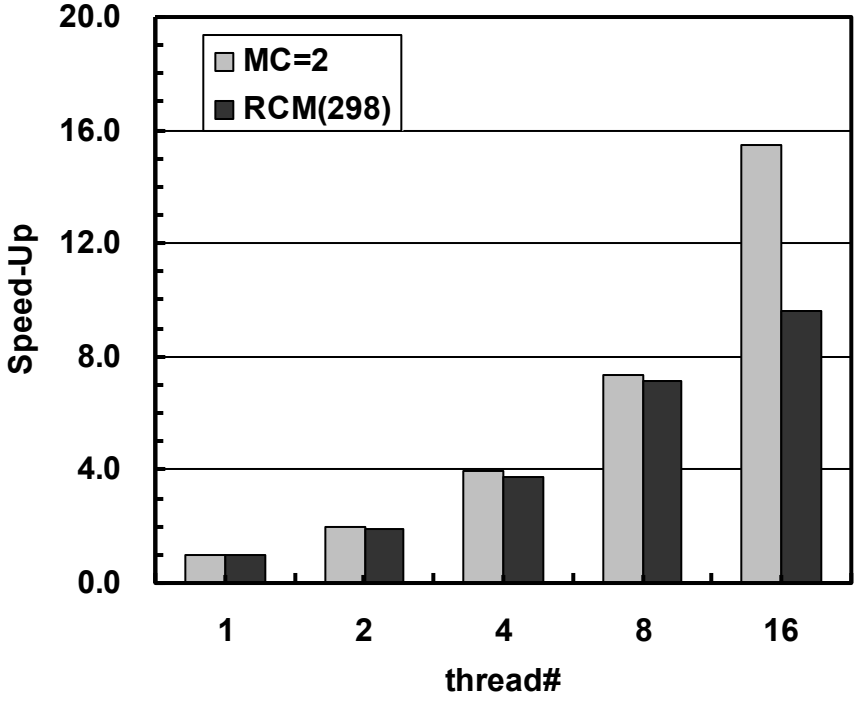
matrix assembling：poi-genの後半：並列化
linear solver：CG



スケーラビリティ

反復回数：MC（2色）：333回，RCM（298レベル）：224回

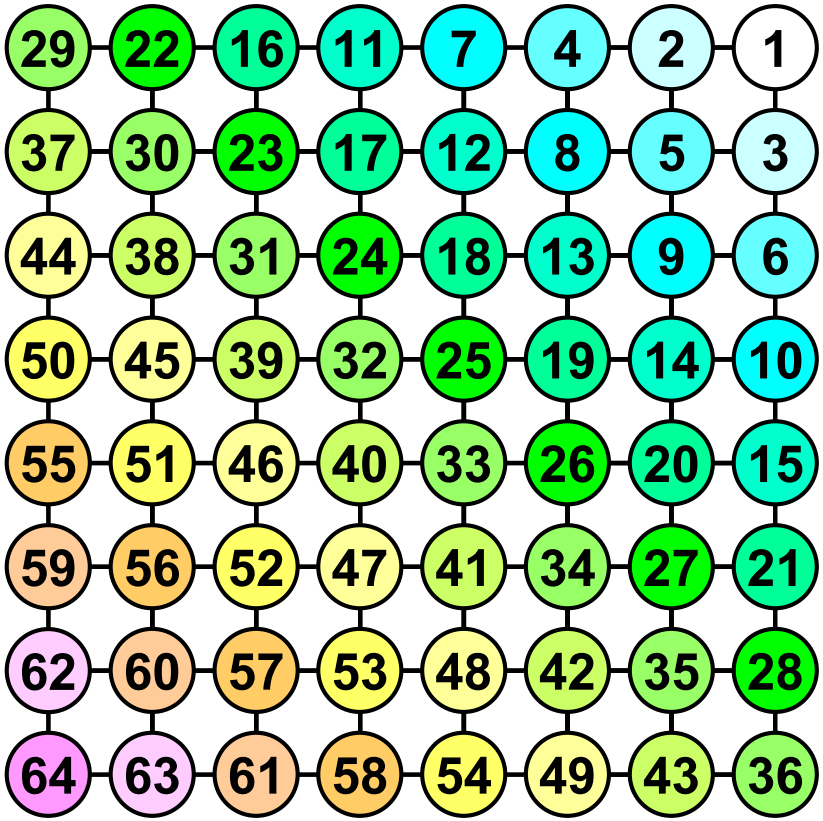




- 反復回数ではRCMの方が少ないのに，コア数が増えると，計算時間が逆転する
- MC=2は良好なスケーラビリティ（16コアで15.5）

原　因

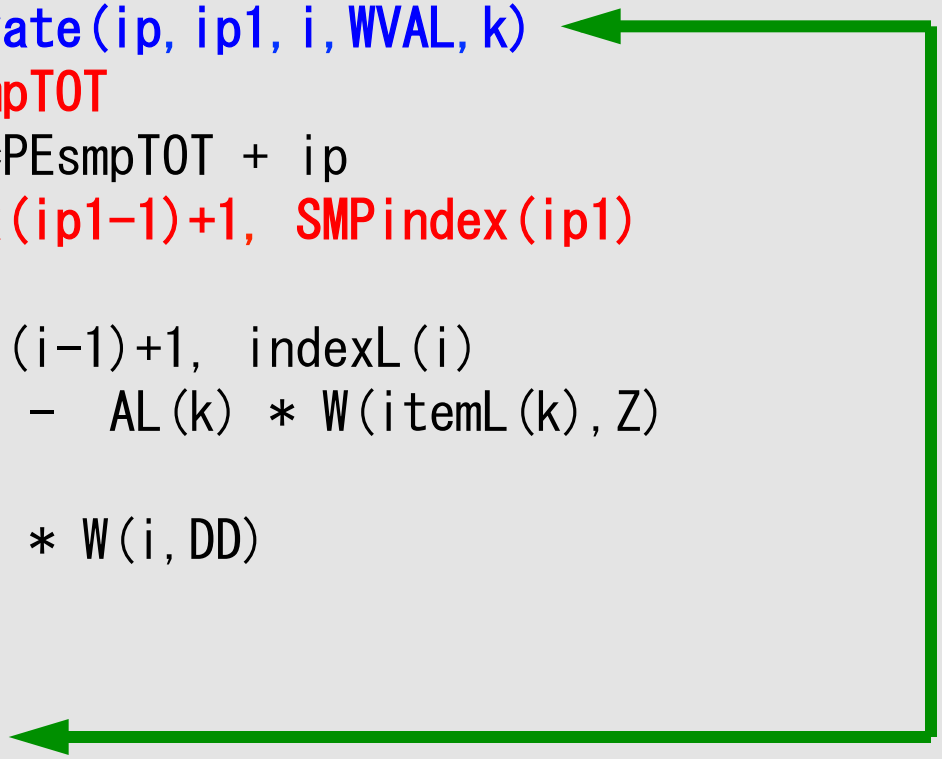
- 色数が多い（298対2）⇒同期オーバーヘッド
- スレッドごとの負荷不均衡
  - RCMでは要素数の少ない「レベル」が必ず存在する



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

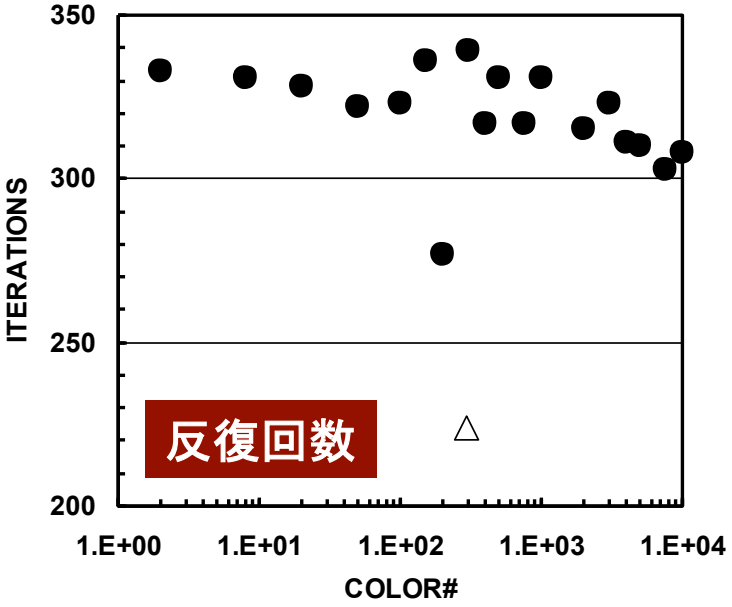
```
      do ic= 1, NCOLORtot
!$omp parallel do private(ip,ip1,i,WVAL,k)
        do ip= 1, PEsmpTOT
          ip1= (ic-1)*PEsmpTOT + ip
        do i= SMPindex(ip1-1)+1, SMPindex(ip1)
          WVAL= W(i,Z)
          do k= indexL(i-1)+1, indexL(i)
            WVAL= WVAL -  AL(k) * W(itemL(k),Z)
     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

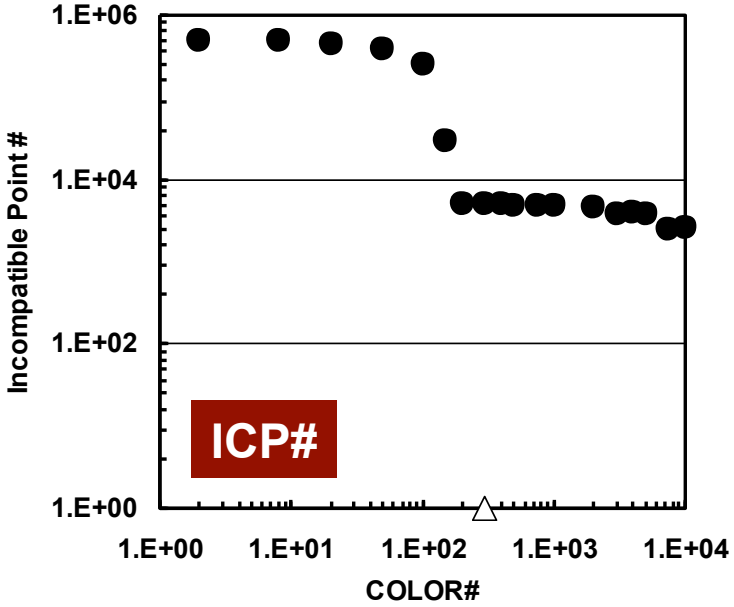


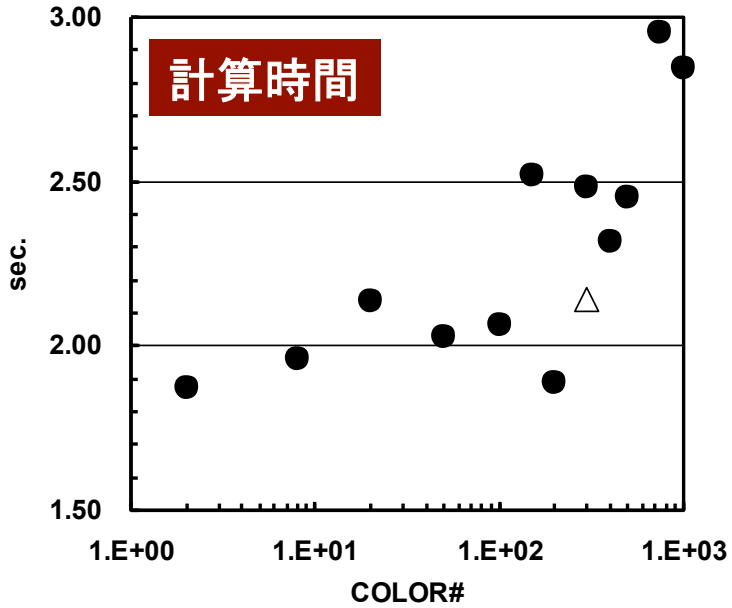
MCとRCMの比較

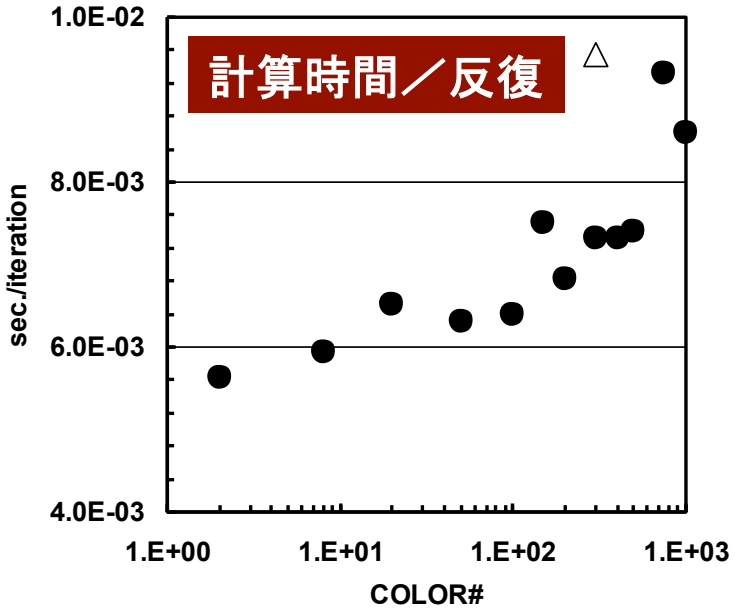
- MC
  - 並列性高い，負荷分散も良い（そのように設定されている）
  - 特に色数少ないと反復回数多い
  - 色数を増やす，反復回数減るが同期オーバーヘッドの影響
- RCM
  - 収束は早いが，レベル数が多く，同期オーバーヘッドの影響受けやすく，コア数が増えると不利
  - 負荷分散もいまーつ
- 反復回数少なくて同期オーバーヘッドの影響が少ない方法が無いものか？
  - 色数が少なくて，かつ反復回数が少ないという都合の良い方法

16コアにおける結果（●：MC，△：RCM）





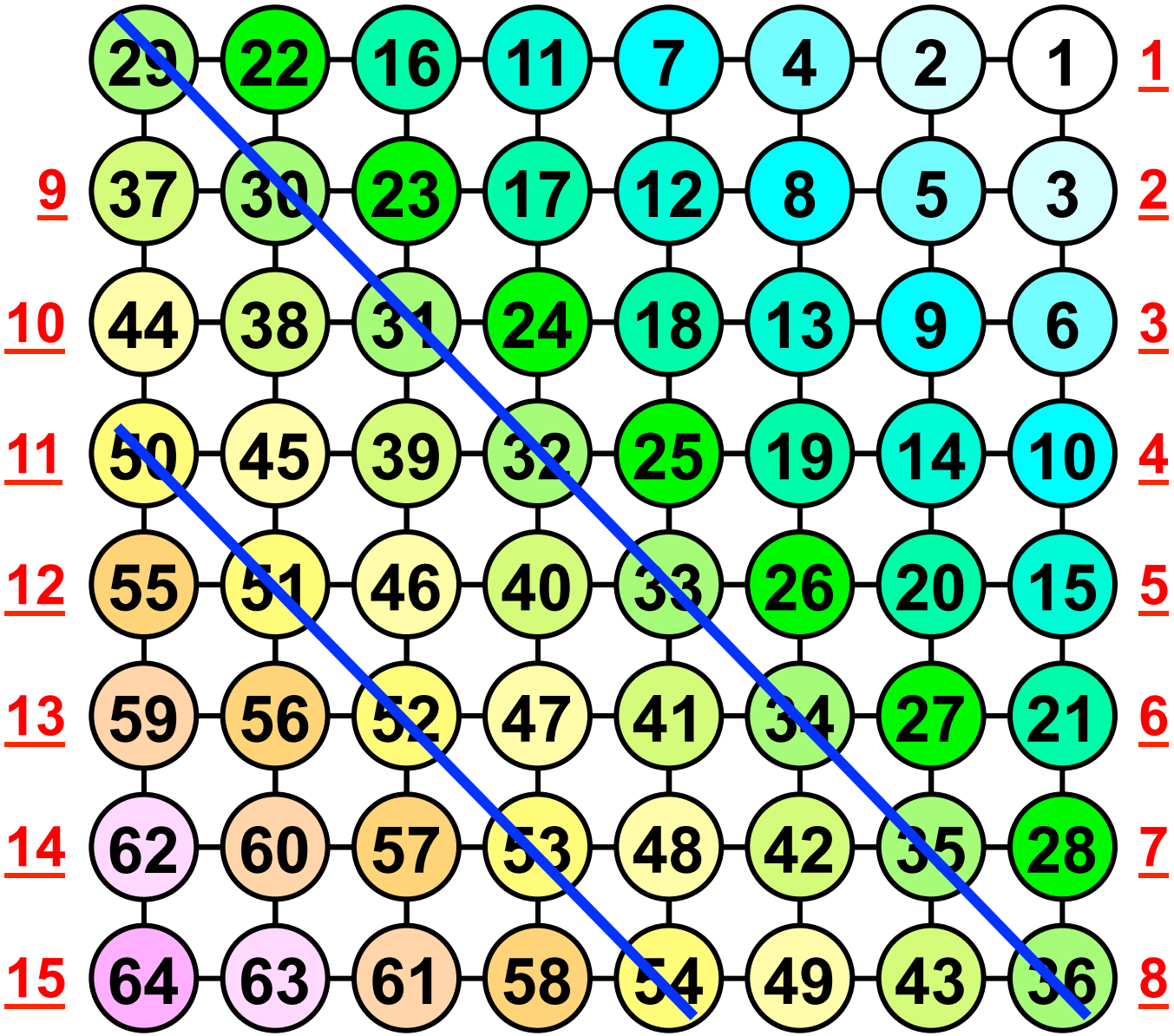




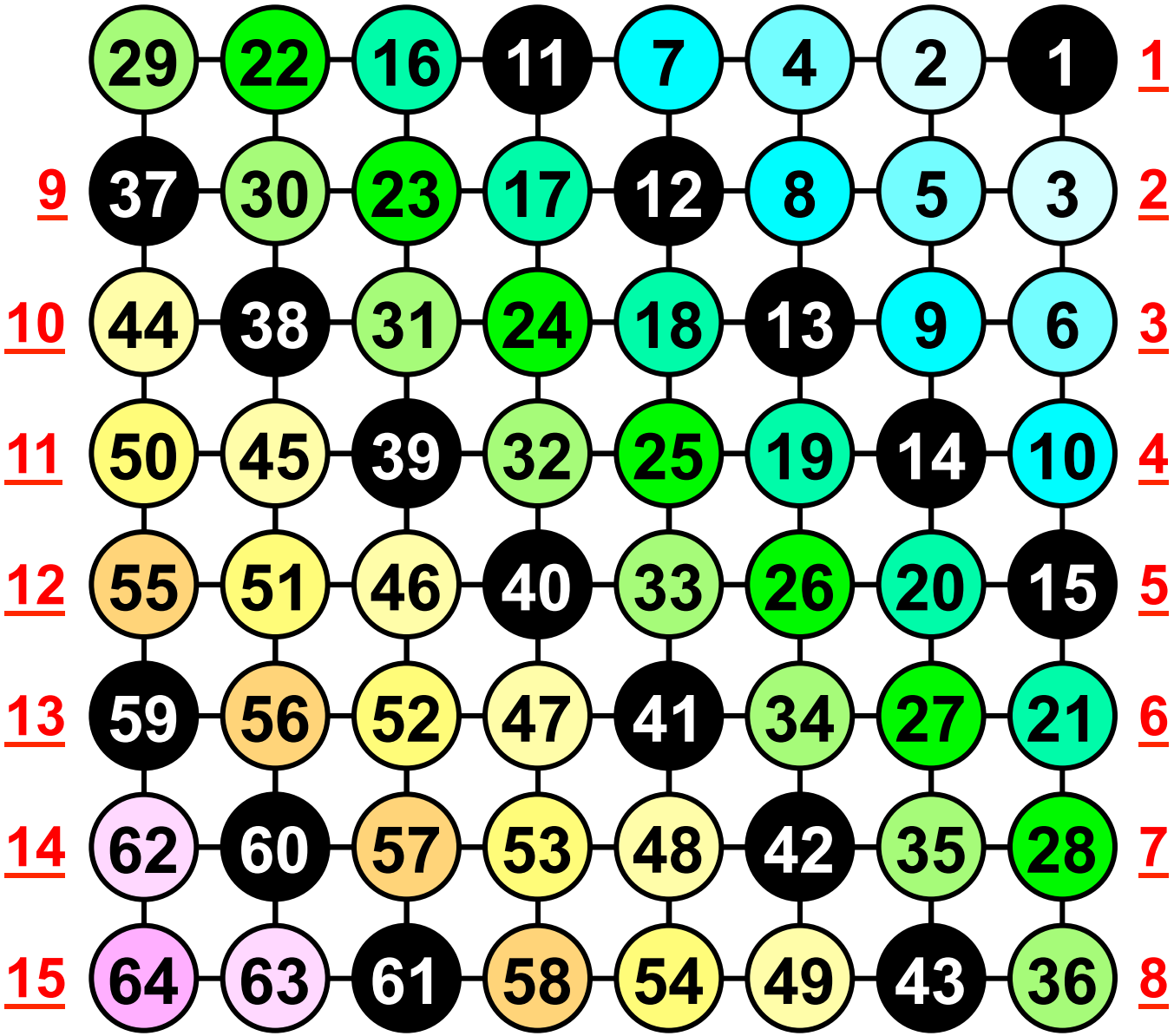
解決策：CM-RCM

- RCM＋Cyclic Mulricoloring〔土肥，襲田，鷲尾他〕
- 手順
  - まずRCMを施す
  - Cyclic Multicoloring（CM）の色数を決める（Nc）
  - RCMの1番目，（Nc+1）番目，（2Nc+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，（Nc+k）番目，（2Nc+k）番目・・・のレベルに属する要素を「k」色に分類する
  - 「k」が「Nc」に達して，要素が「1~Nc」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「Nc」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，Nc=Nc+1として最初からやり直し（ここは少し原始的）

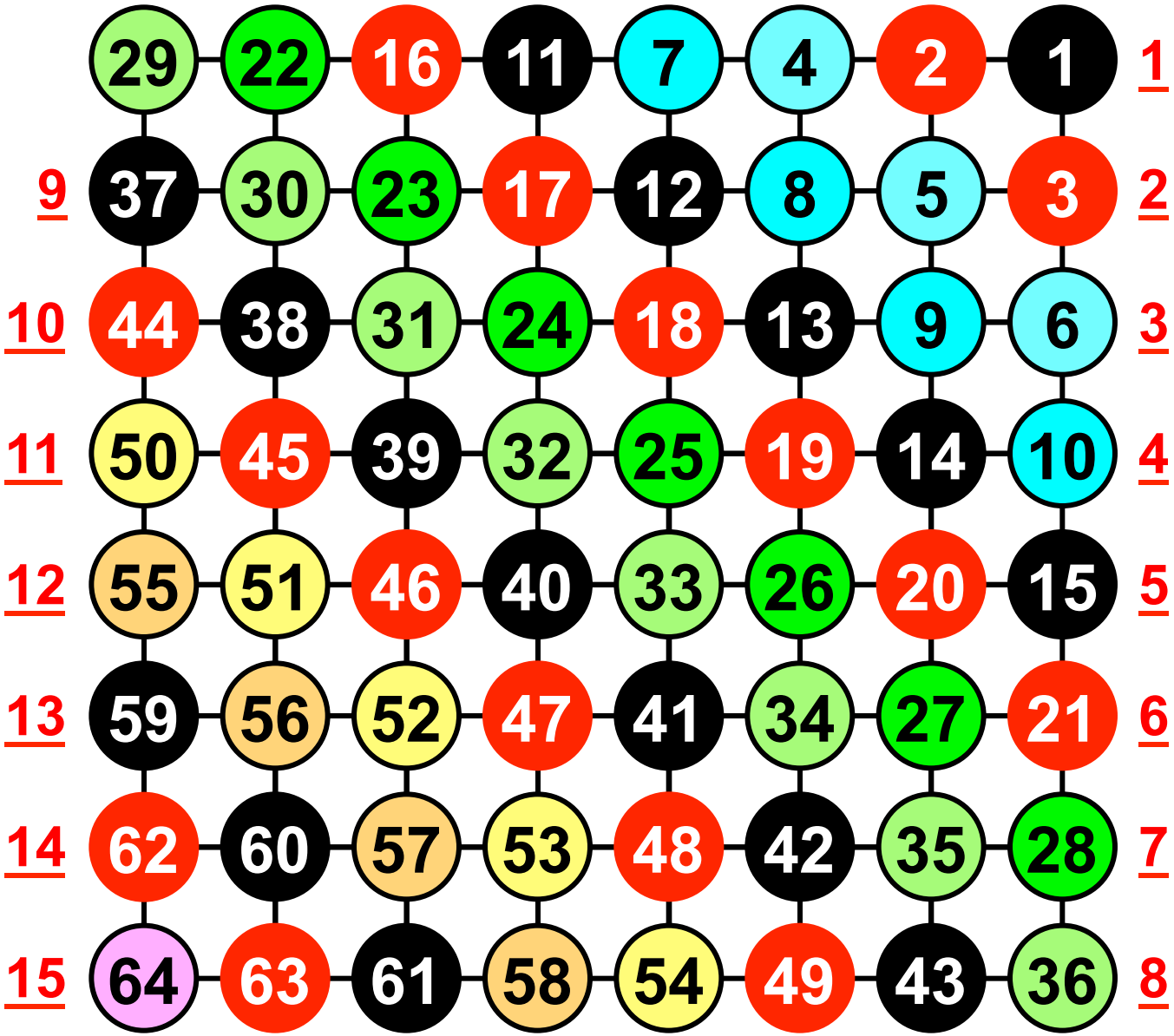
RCM


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

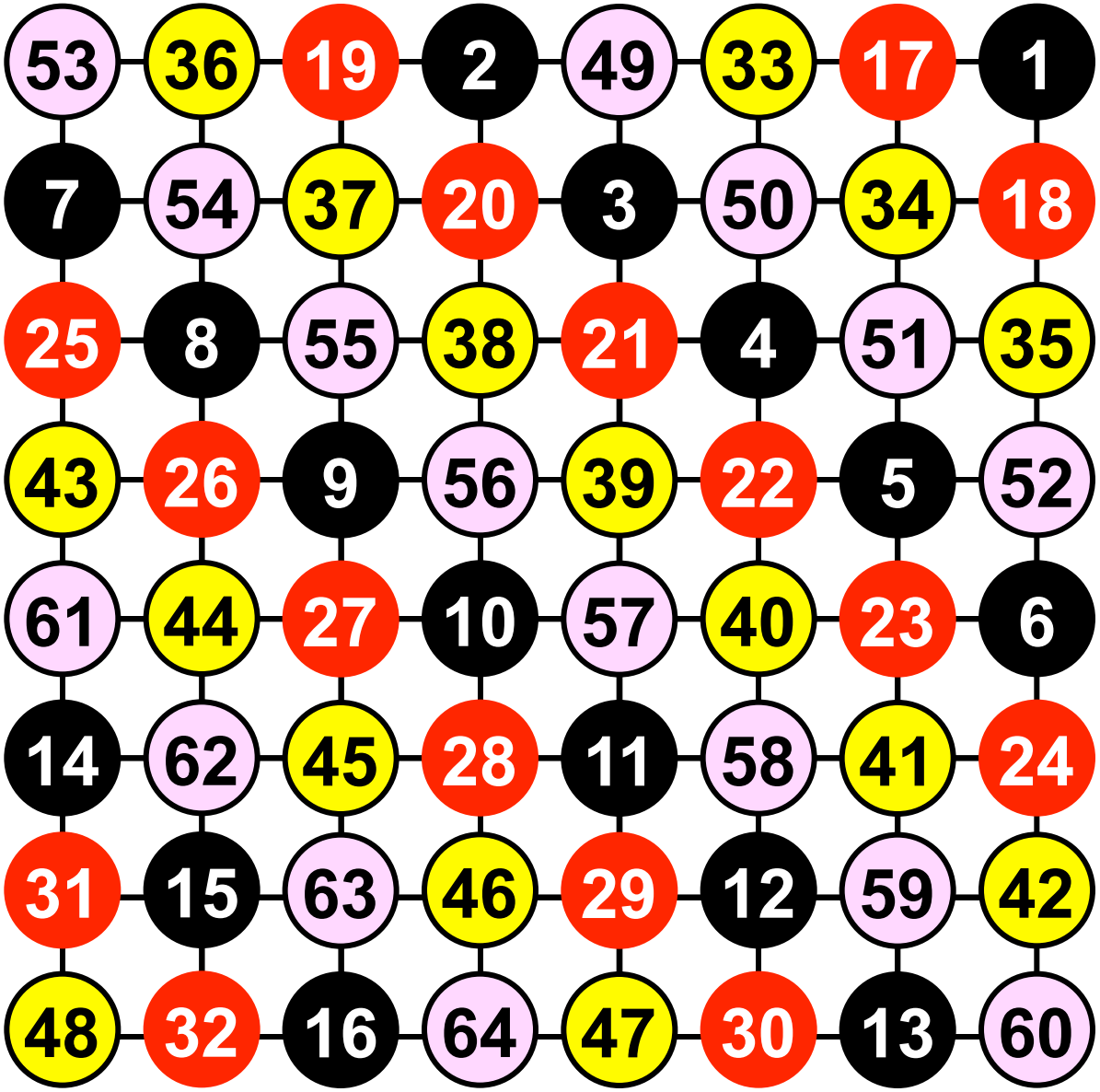
Nc=4, k=1 : 1,5,9,13レベルを選択



Nc=4, k=2：2,6,10,14レベルを選択

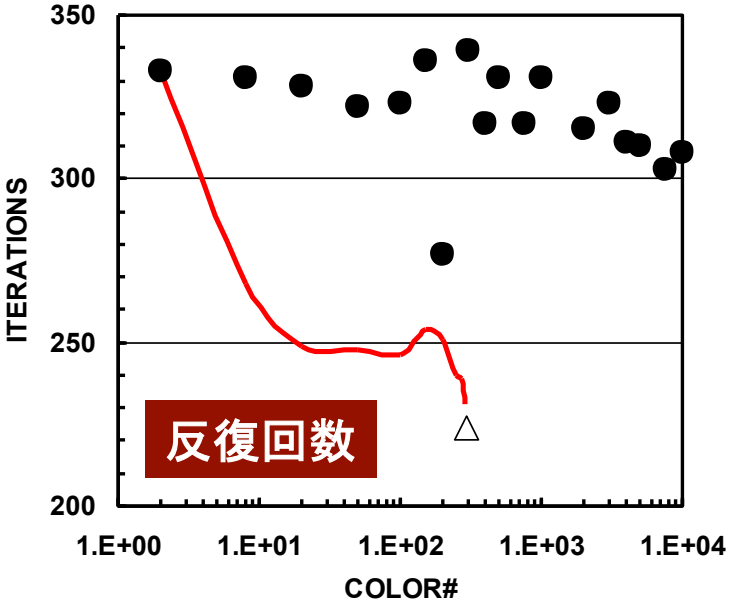


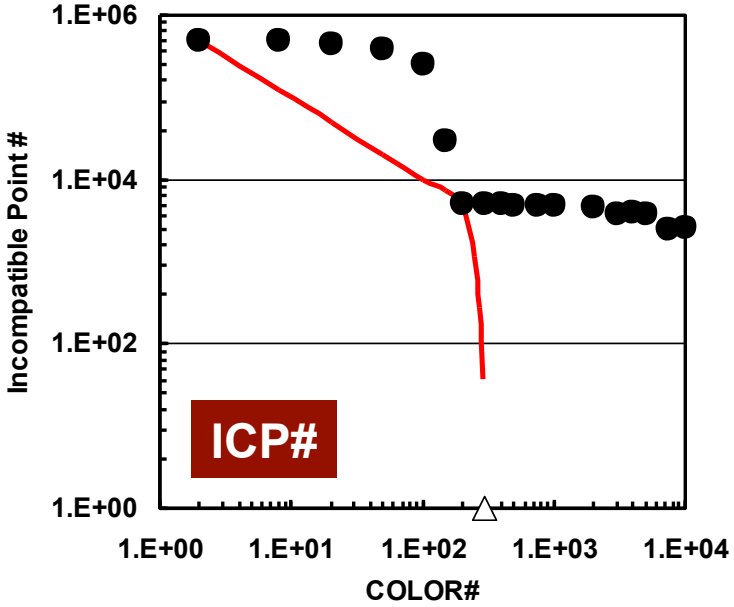
CM-RCM（Nc=4）:「色」の順番に再並替

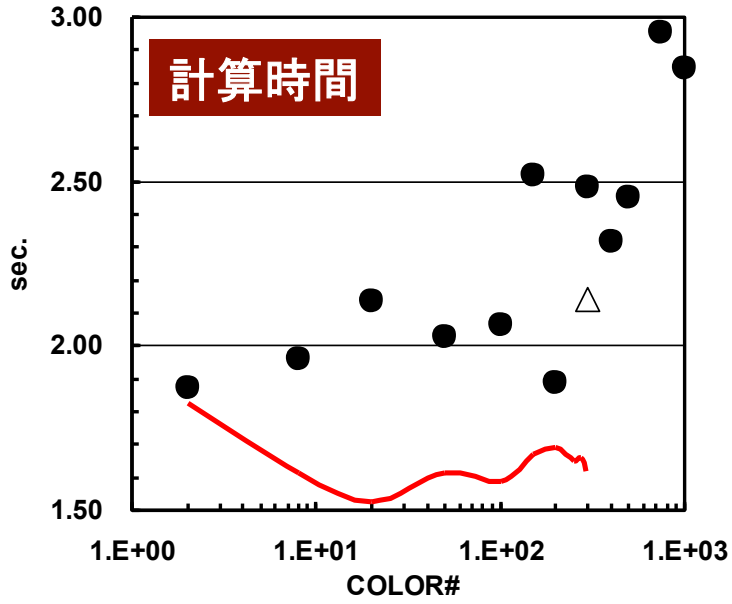


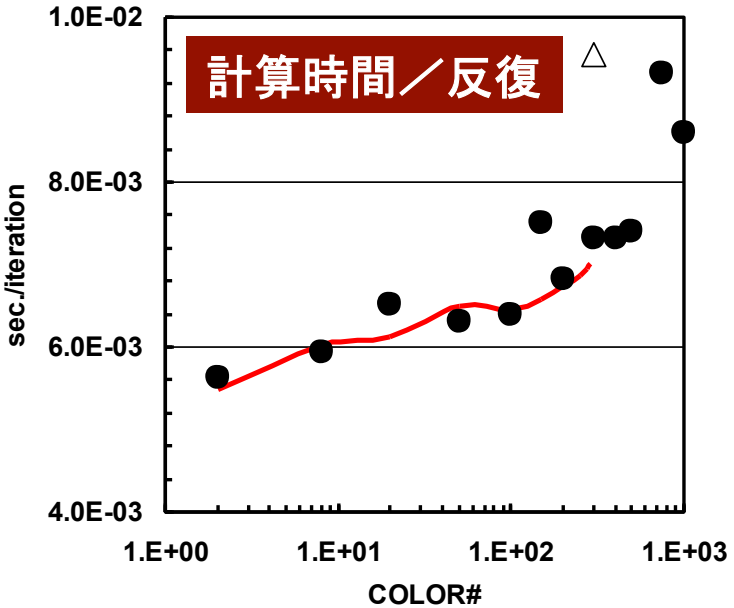
k=1: 16
k=2: 16
k=3: 16
k=4: 16

16コアにおける結果（●：MC，△：RCM）



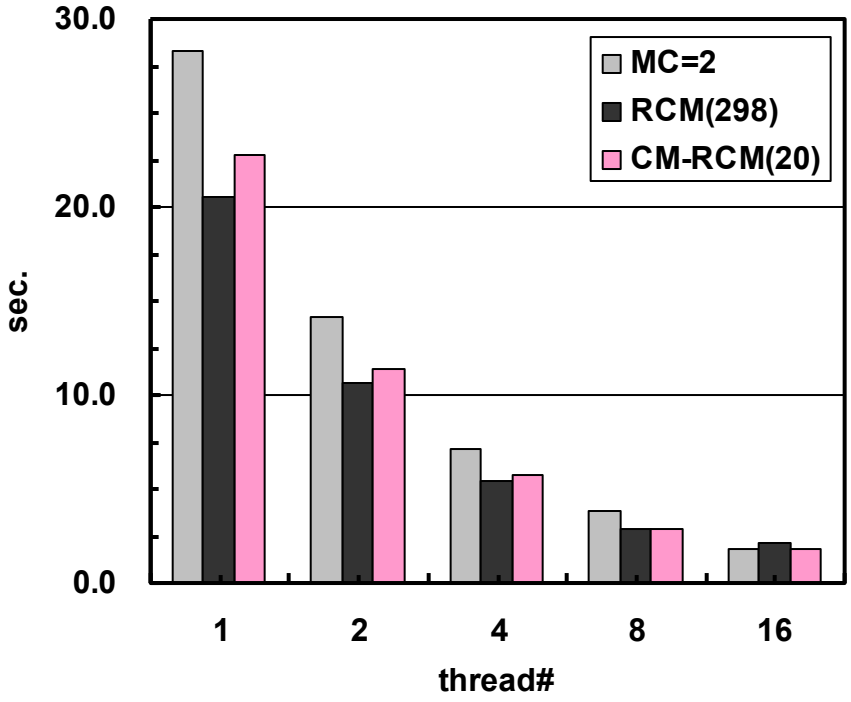


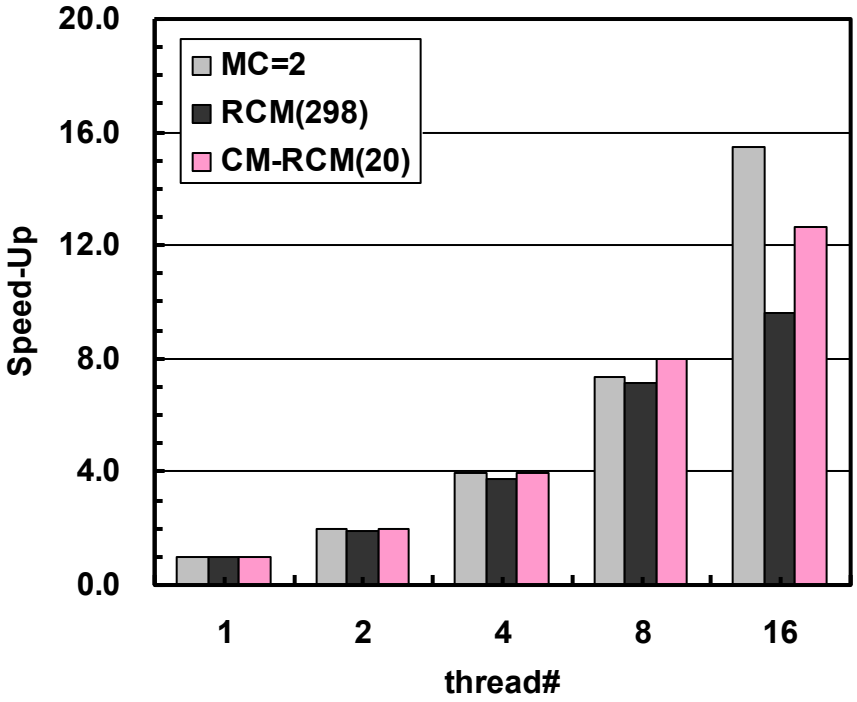




スケーラビリティ

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





16 threads
MC(2): 1.83 sec.
CM-RCM(20): 1.79 sec.

CM-RCM

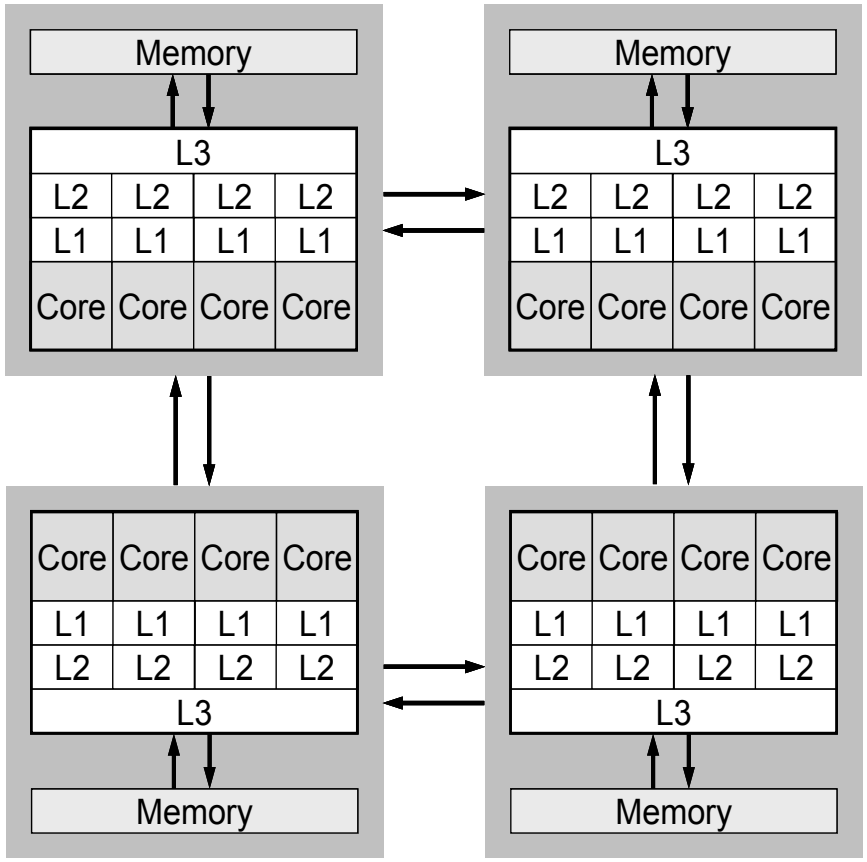
- 少ない色数(Nc)で良好な収束を得られる
- 計算効率も良い
- 実行方法
  - INPUT.DATで「**NCOLORtot=-Nc**」とする
  - L2，L3で有効なオプション(既にL2でも使えていた)

- 実装は「cmrcm.f」を参照ください(本日は説明は省略)

- 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：Sequential Reordering
- 他システムとの比較

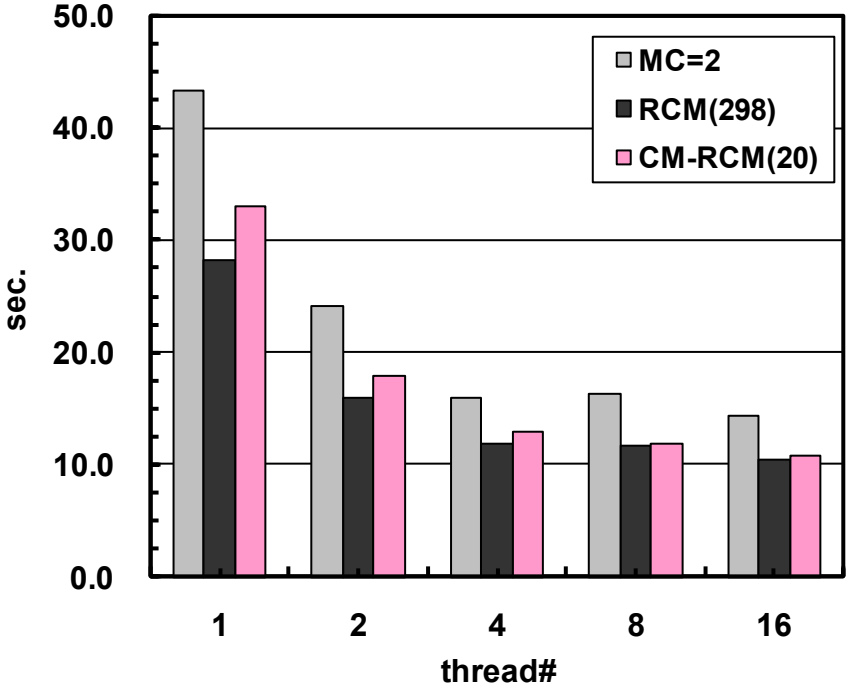
計算結果

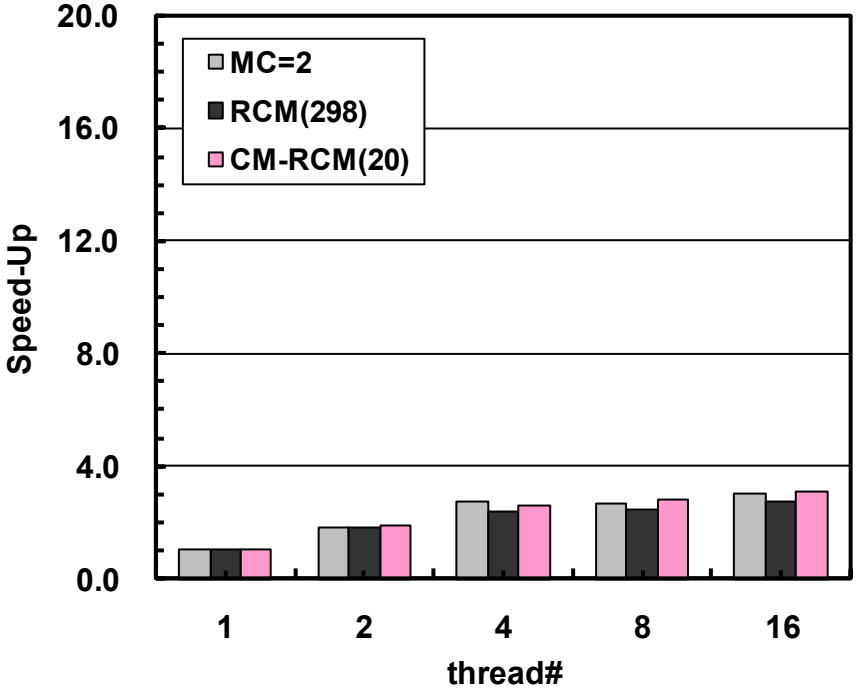
- T2K（東大） 1ノード（4ソケット，16コア）
- 100^3要素



スケーラビリティ：CASE-0

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





実行用データ

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(固定)
100                                  NCOLORtot
```

x0.sh

```
#@$-r test
#@$-q tutorial
#@$-N 1
#@$-J T1
#@$-e err
#@$-o test.lst
#@$-lM 28GB
#@$-lE 00:10:00
#@$-s /bin/sh
#@$

cd $PBS_O_WORKDIR
export OMP_NUM_THREADS= 16 ここでスレッド数を変える
mpirun ./L3-sol
exit
```

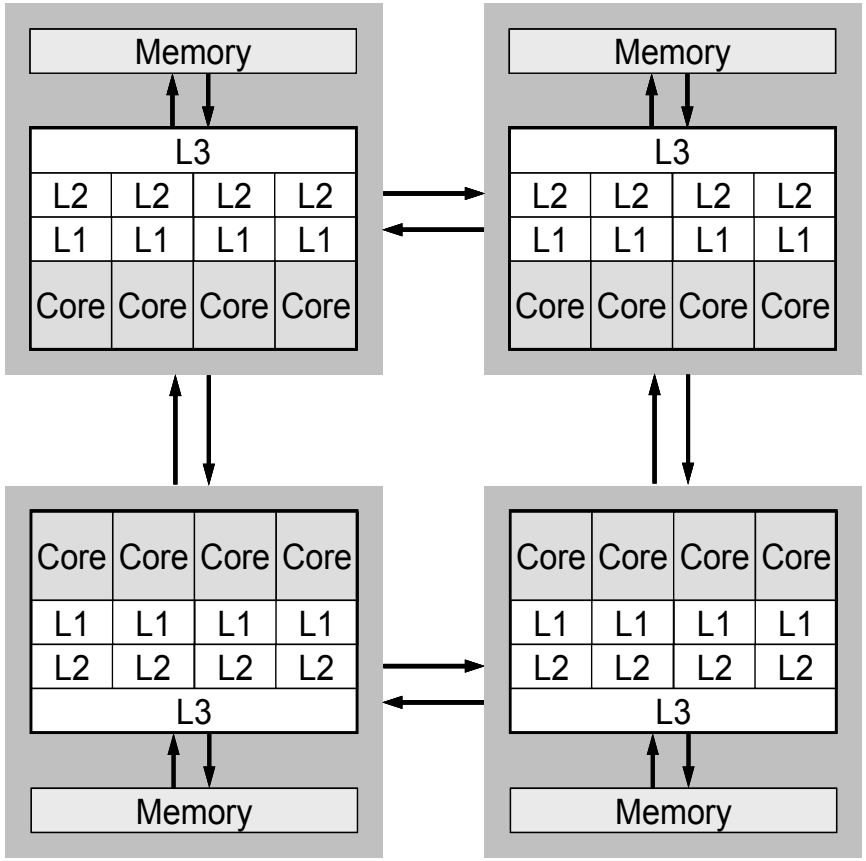
- L2-solへのOpenMPの実装
- Hitachi SR11000/J2での実行
  - 計算結果
  - CM-RCMオーダリング
- T2K（東大）での実行
- **T2K（東大）での性能向上への道**
  - **NUMA　Control**
  - First Touch
  - データ再配置：Sequential Reordering
- 他システムとの比較

性能向上への道

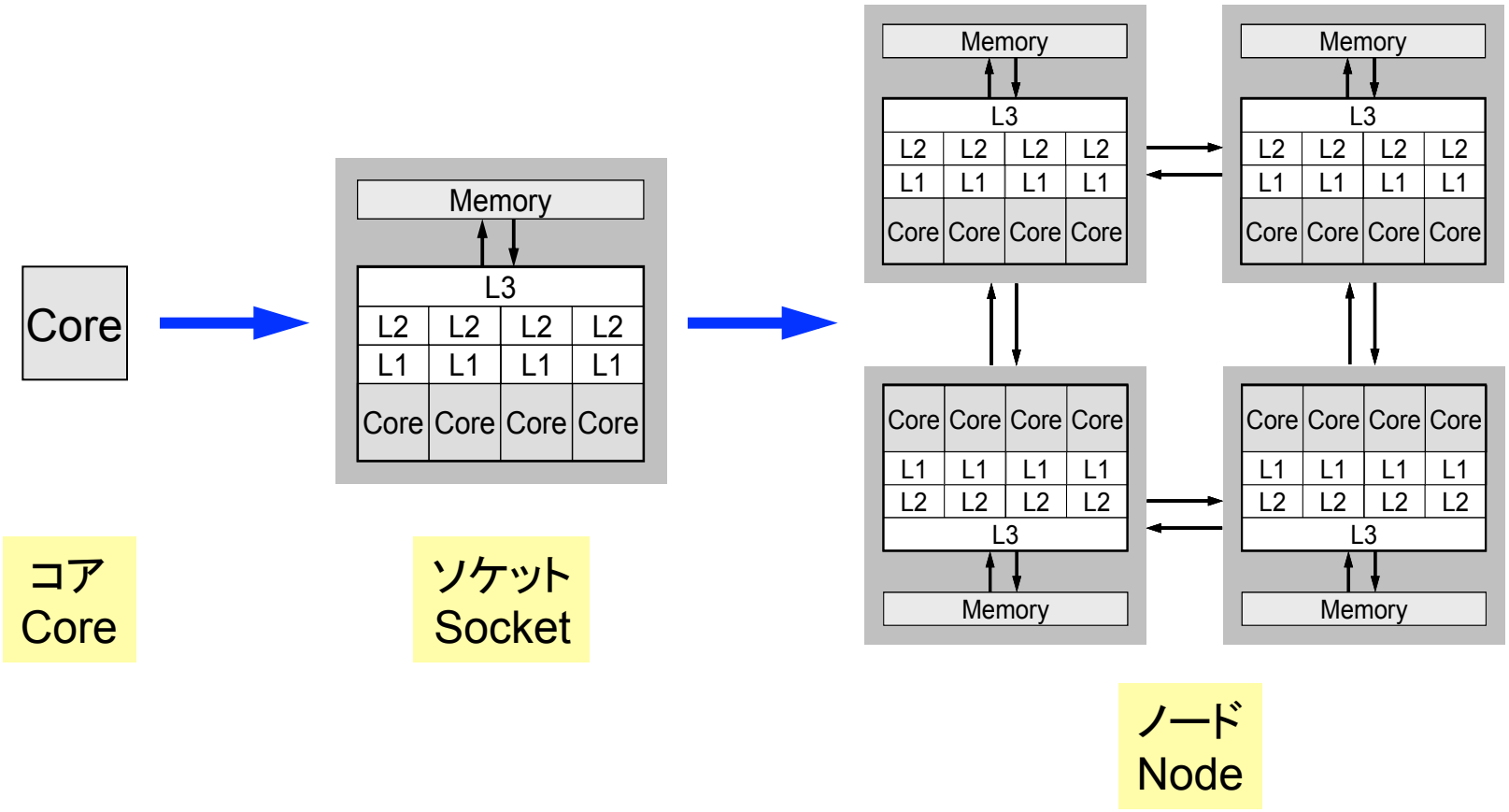
- CASE-0　初期状態
- CASE-1a　NUMA control (policy-a)
- CASE-1b　NUMA control (policy-b)
- CASE-2a　First-touch (a)
- CASE-2b　First-touch (b)
- CASE-3a　First-touch＋データ再配置(a)
- CASE-3b　First-touch＋データ再配置(b)
- CASE-4a　データ再配置(a)
- CASE-4b　データ再配置(b)

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
  - cc-NUMA：cache-coherent-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



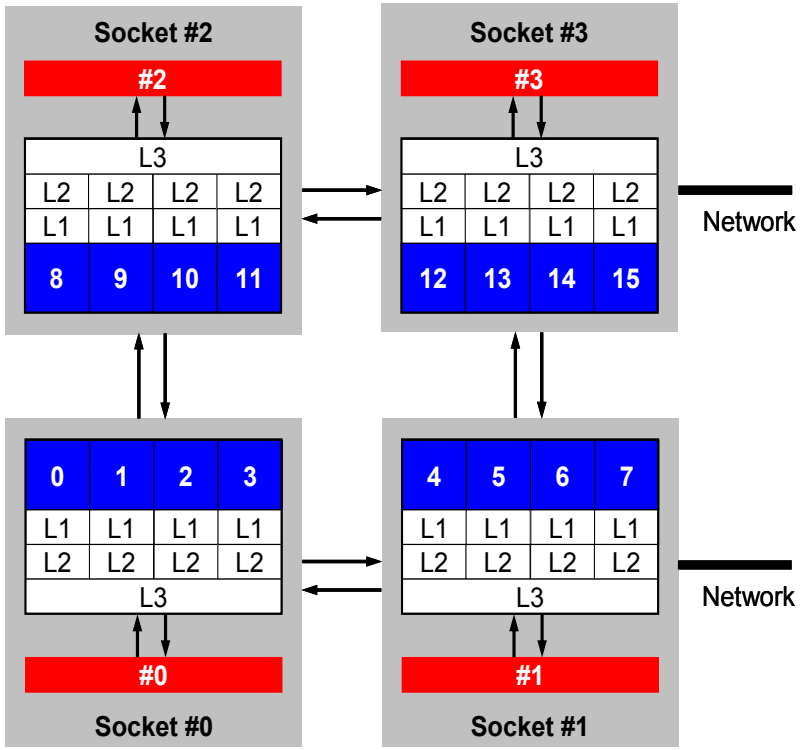
numactl

- 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
- T2K（東大）でも実績有り〔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
```

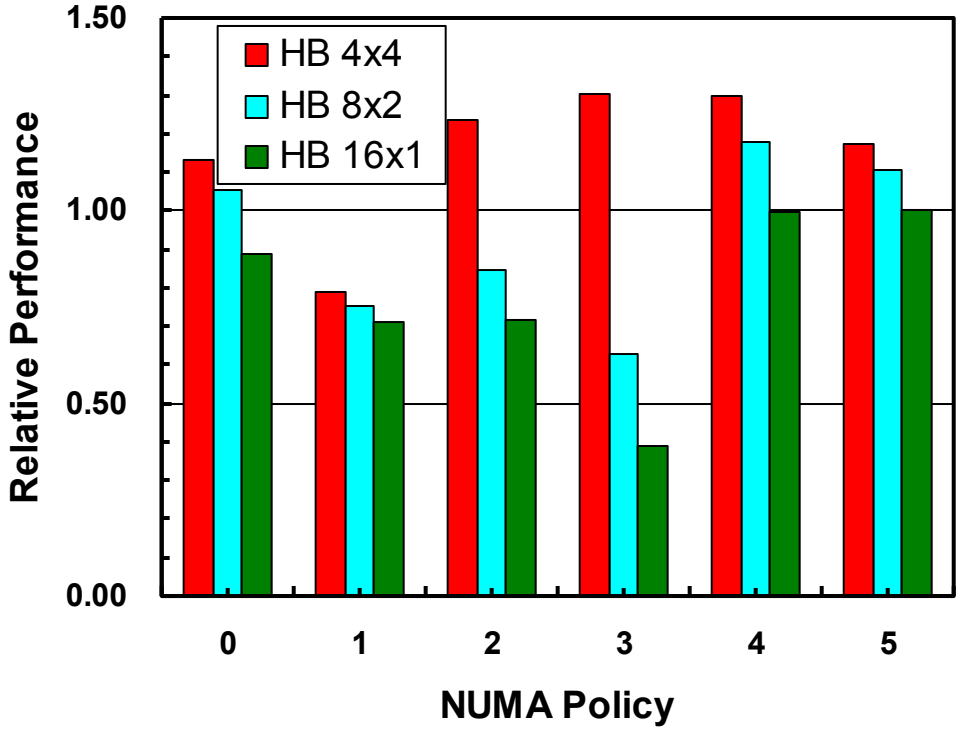
numactl --show

```
>$ numactl --show

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



例：numactlの影響

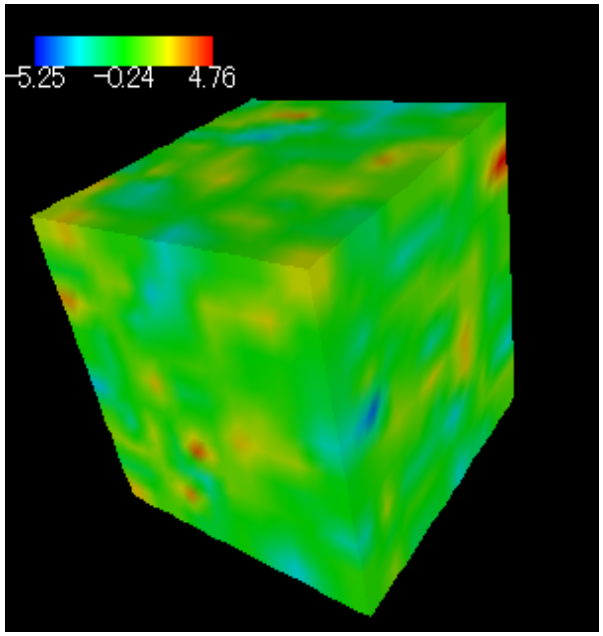
- T2K（東大），有限要素法アプリケーション
- POLICY=0：何も指定しない場合
- 相対性能：大きいほど良い
  - Flat MPIに対する相対性能
- 状況によって，最適な組み合わせは実は異なる（後述）



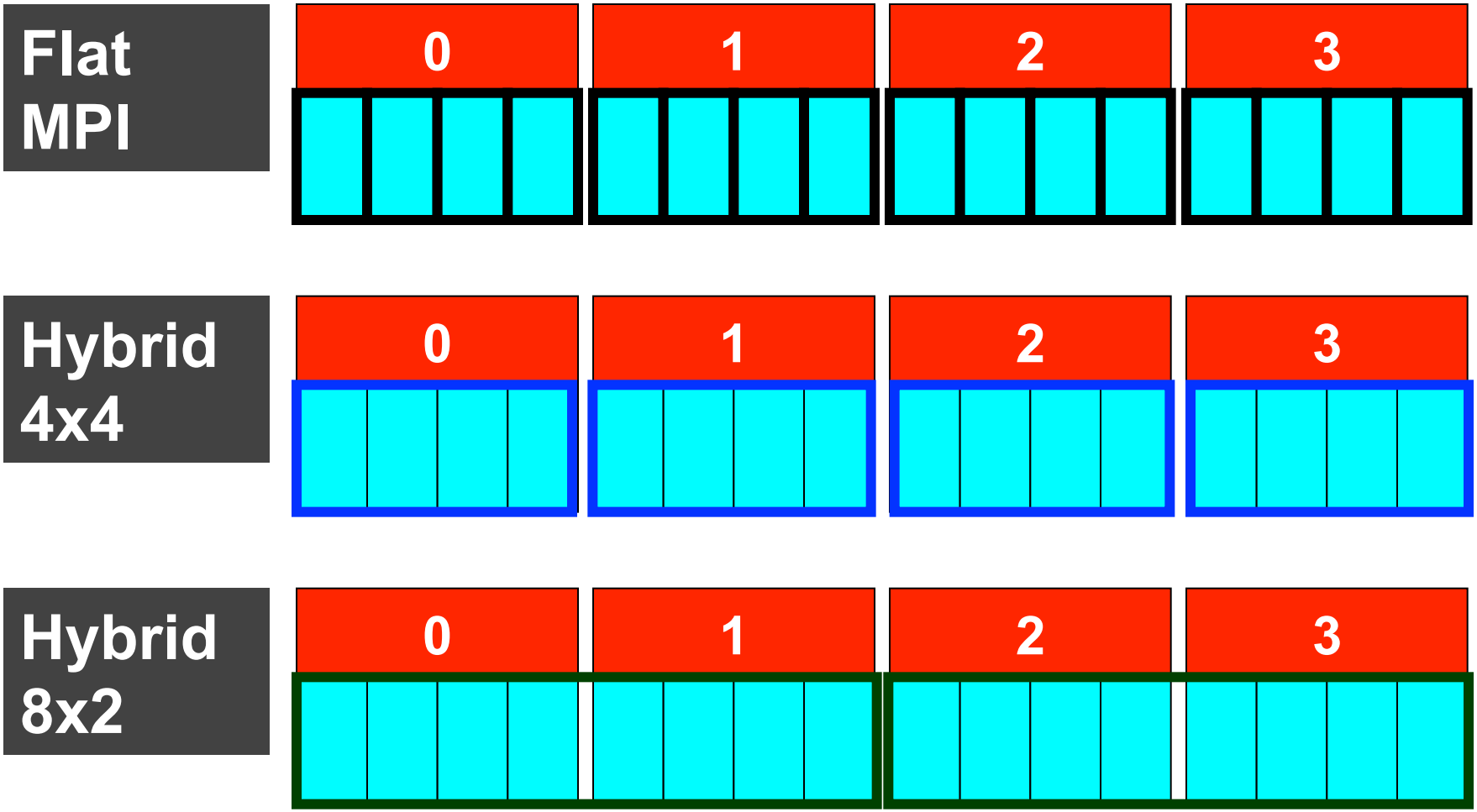

```
#@$-r HID-org
#@$-q h08nkl32
#@$-N 24
#@$-J T4
#@$-e err
#@$-o x384-40-1-a.lst
#@$-lM 27GB
#@$-lE 03:00:00
#@$-s /bin/sh
#@$

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

Target Application

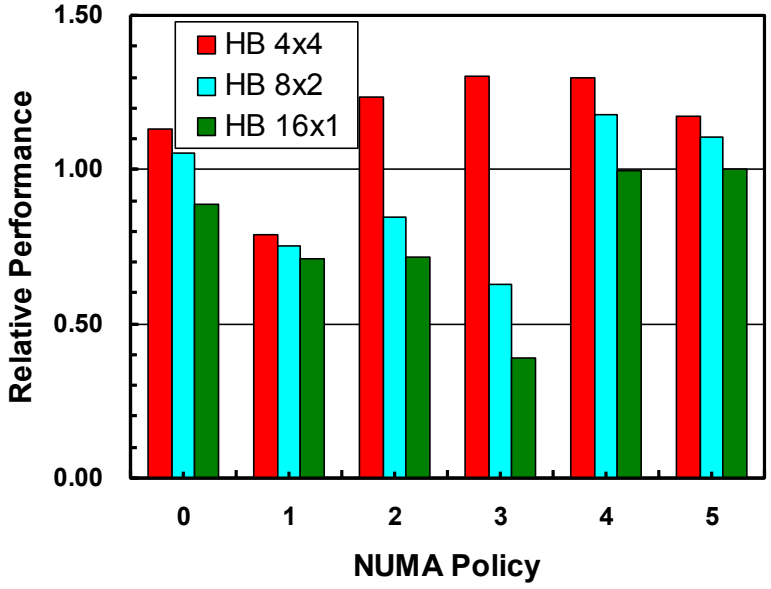
- 3D Elastic Problems with Heterogeneous Material Property
  - $E_{max}=10^3$, $E_{min}=10^{-3}$, $\nu=0.25$
    - generated by “sequential Gauss” algorithm for geo-statistics [Deutsch & Journel, 1998]
  - 128^3 tri-linear hexahedral elements, 6,291,456 DOF
- (SGS+GPBiCG) Iterative Solvers
  - Symmetric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

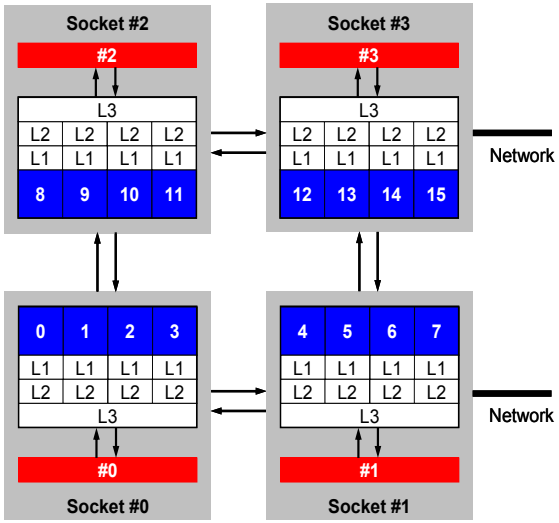


Flat MPI, Hybrid (4x4, 8x2)



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 $@
```

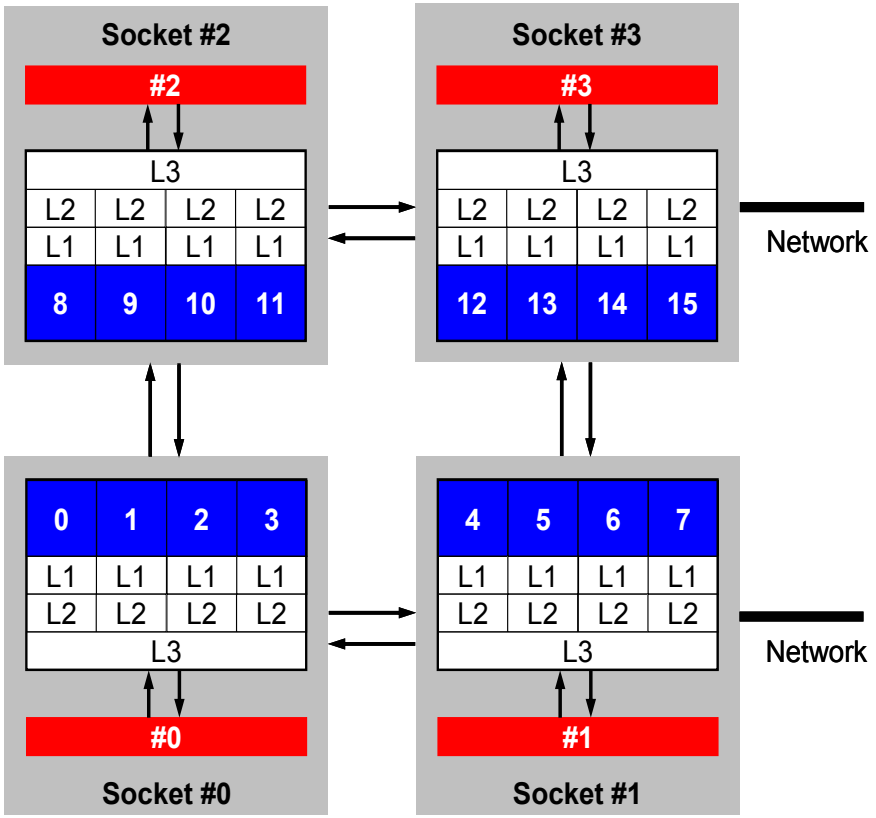
Policy:3
```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --membind=$SOCKET $@
```

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 $@
```

本ケースにおける:numactlのポリシー

- policy-a
  - --localalloc
  - ローカルなソケットのメモリ使用
- policy-b
  - --interleave= all
  - ノード上のメモリをinterleave（分散配置）
  - NUMAは無効になる



本ケースにおける：numactlのポリシー

x5.sh
Policy-a

```
#@$-r test
#@$-q tutorial
#@$-N 1
#@$-J T1
#@$-e err
#@$-o test.lst
#@$-lM 28GB
#@$-lE 00:10:00
#@$-s /bin/sh
#@$

cd $PBS_O_WORKDIR
export OMP_NUM_THREADS= 16
mpirun numactl --localalloc ./sol
exit
```

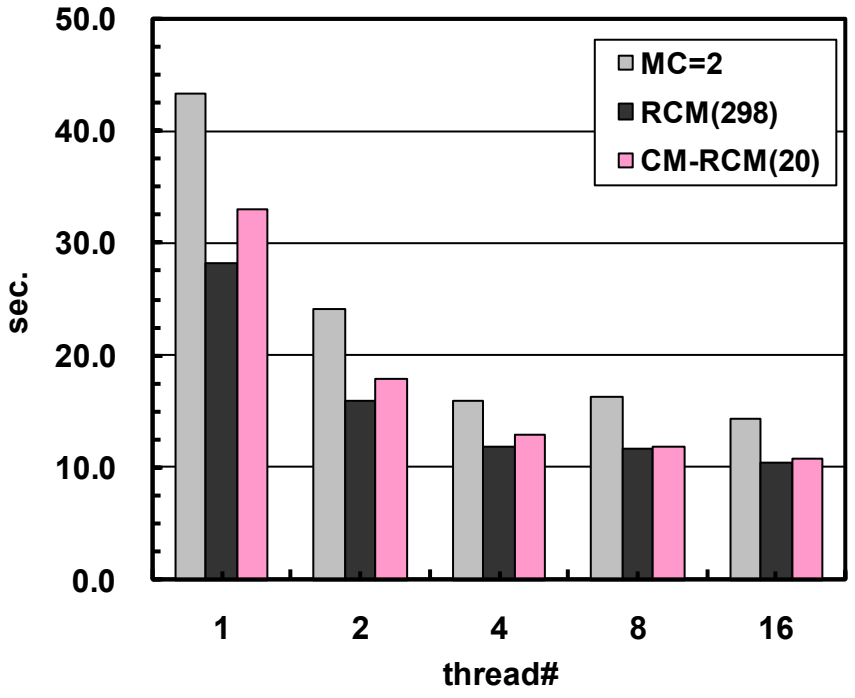
x6.sh
Policy-b

```
cd $PBS_O_WORKDIR
export OMP_NUM_THREADS= 16
mpirun numactl --interleave=all ./sol
exit
```

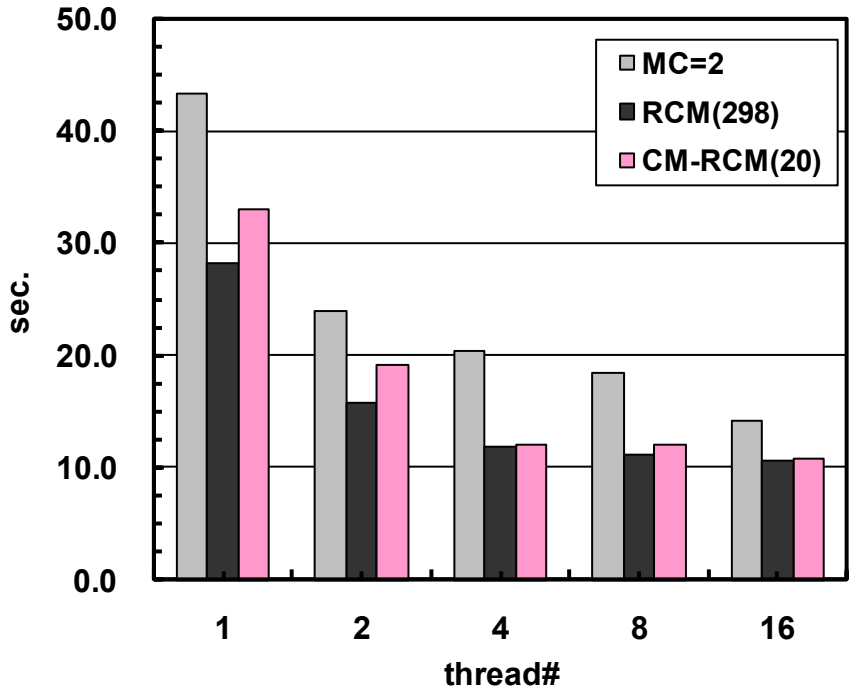
スケーラビリティ：numactlの効果

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

CASE-1a：あまり変わらない



CASE-0

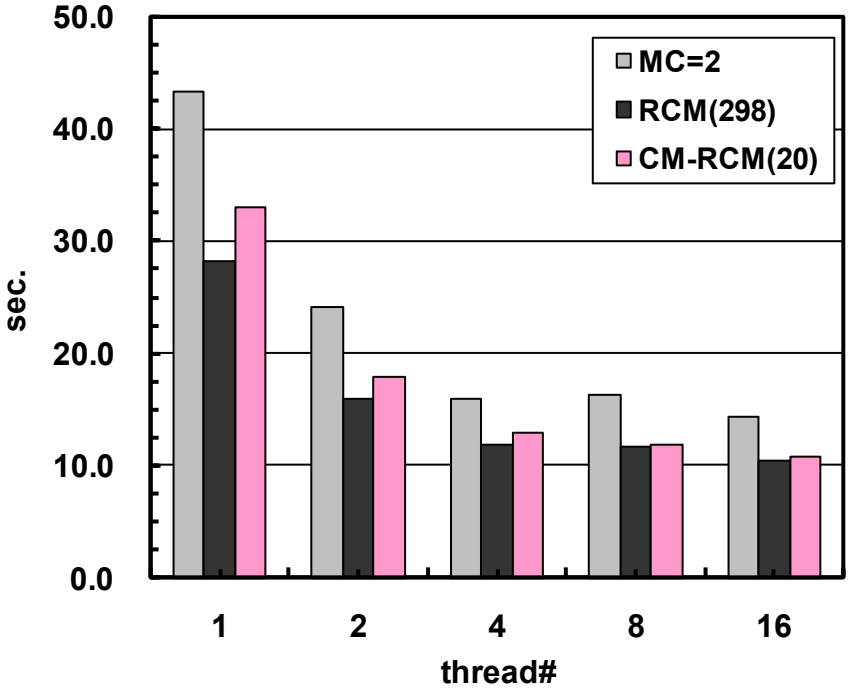


CASE-1a
--localalloc

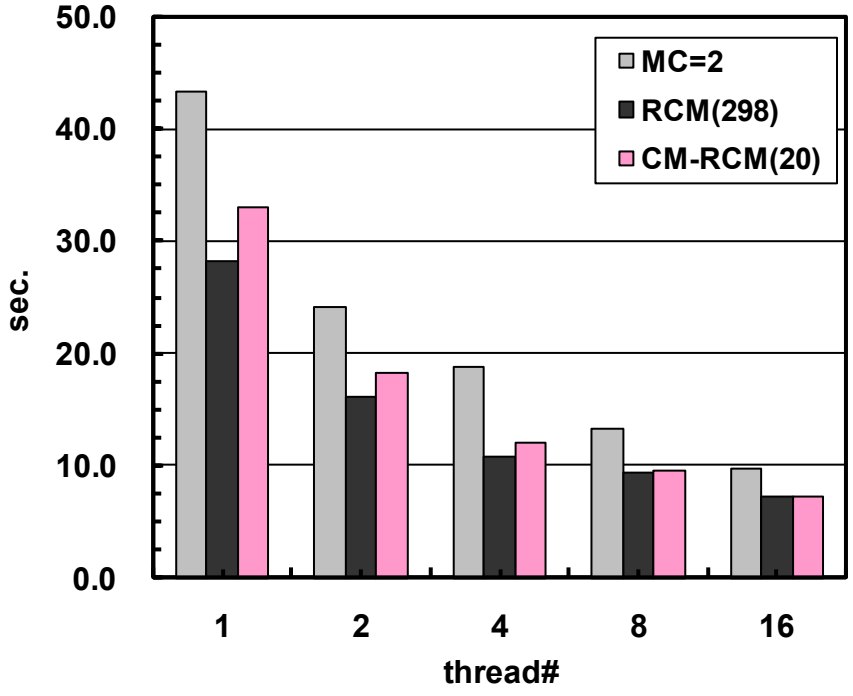
スケーラビリティ：numactlの効果

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

CASE-1b：少し良くなった



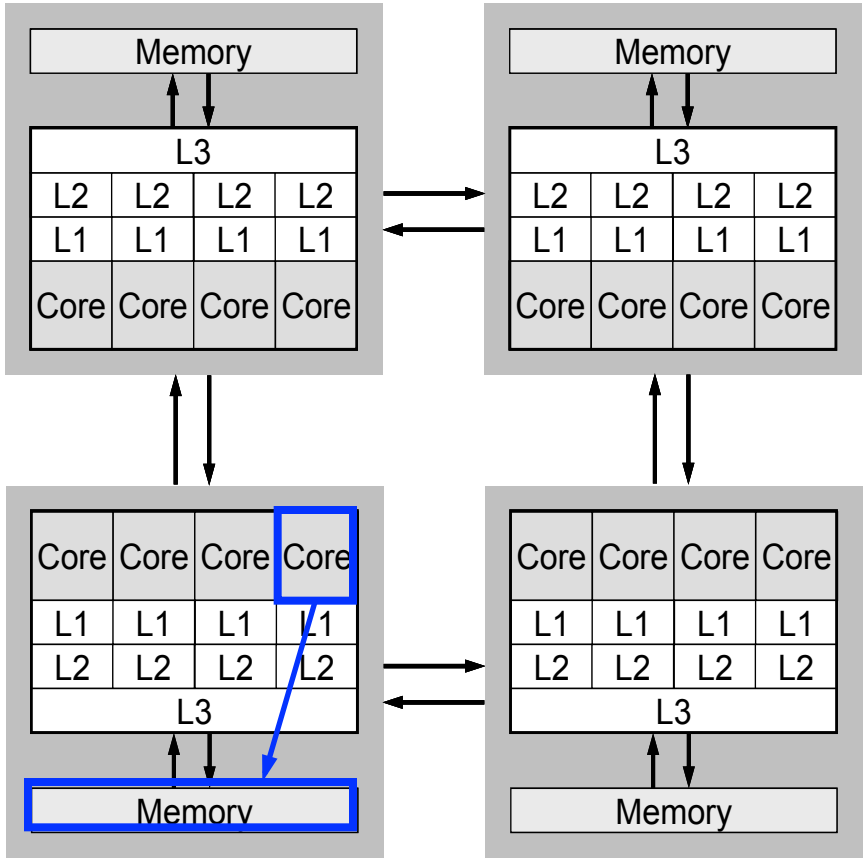
CASE-0



CASE-1b
--interleave=all

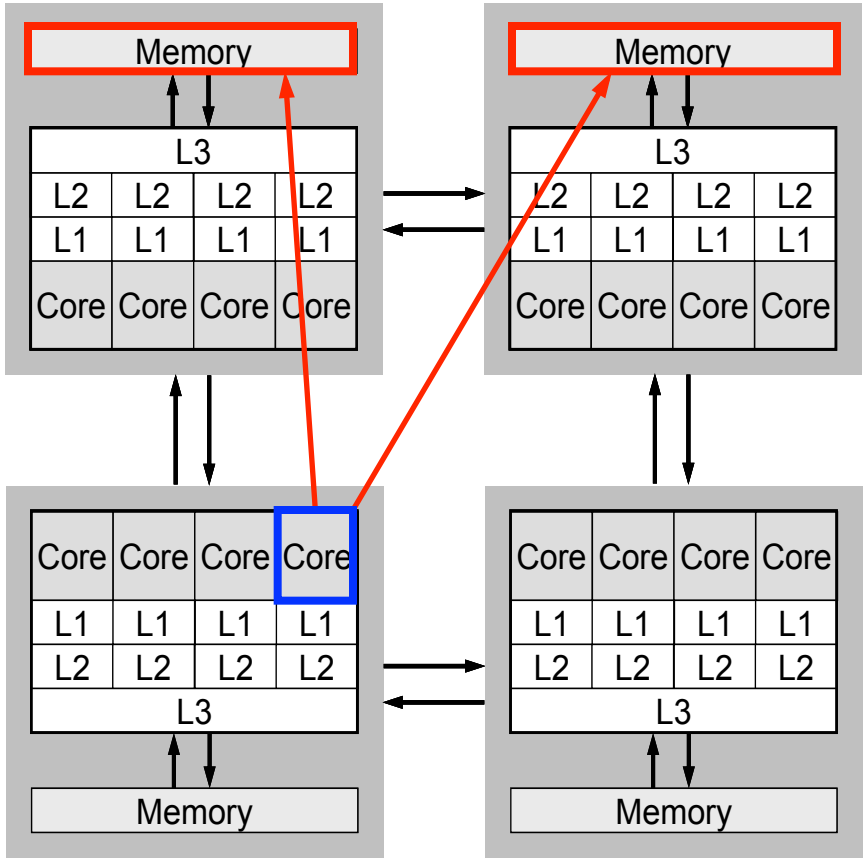
- L2-solへのOpenMPの実装
- Hitachi SR11000/J2での実行
  - 計算結果
  - CM-RCMオーダリング
- T2K（東大）での実行
- **T2K（東大）での性能向上への道**
  - NUMA　Control
  - **First Touch**
  - データ再配置：Sequential Reordering
- 他システムとの比較

ローカルメモリ，遠隔メモリ



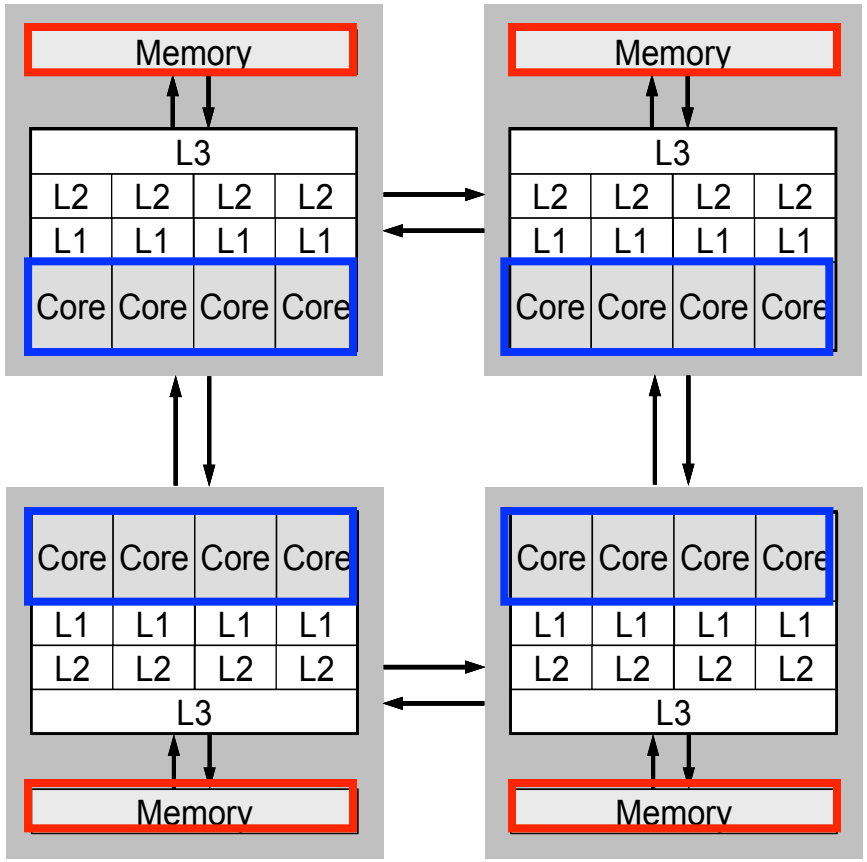
- コアで扱うデータはなるべくローカルなメモリ（コアの属するソケットのメモリ）上にあると効率が良い。

ローカルメモリ，遠隔メモリ



- 異なるソケットにある場合はアクセスに時間がかかる。

ローカルメモリ，遠隔メモリ



- First-touchによって，できるだけローカルなメモリ上にデータを持ってくる。
- NUMAアーキテクチャでは，ある変数を最初にアクセスしたコア（の属するソケット）のローカルメモリ上にその変数の記憶領域（ページファイル）が確保される。
  - 配列の初期化手順によって大幅な性能向上が期待できる。

First Touch Data Placement

"Patterns for Parallel Programming" Mattson, T.G. et al.

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

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

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

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

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

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

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

```
!$omp parallel do private(ip,i,VAL,k)
      do ip= 1, PEsmpTOT
        do i = SMPindexG(ip-1)+1, SMPindexG(ip)
          D(i)=0.d0
          do k= indexL(i-1)+1, indexL(i)
               AL(k)= 0.d0
            itemL(k)= 0
          enddo
          do k= indexU(i-1)+1, indexU(i)
               AU(k)= 0.d0
            itemU(k)= 0
          enddo
        enddo
      enddo
!$omp end parallel do
```

First Touch Data Placement

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

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

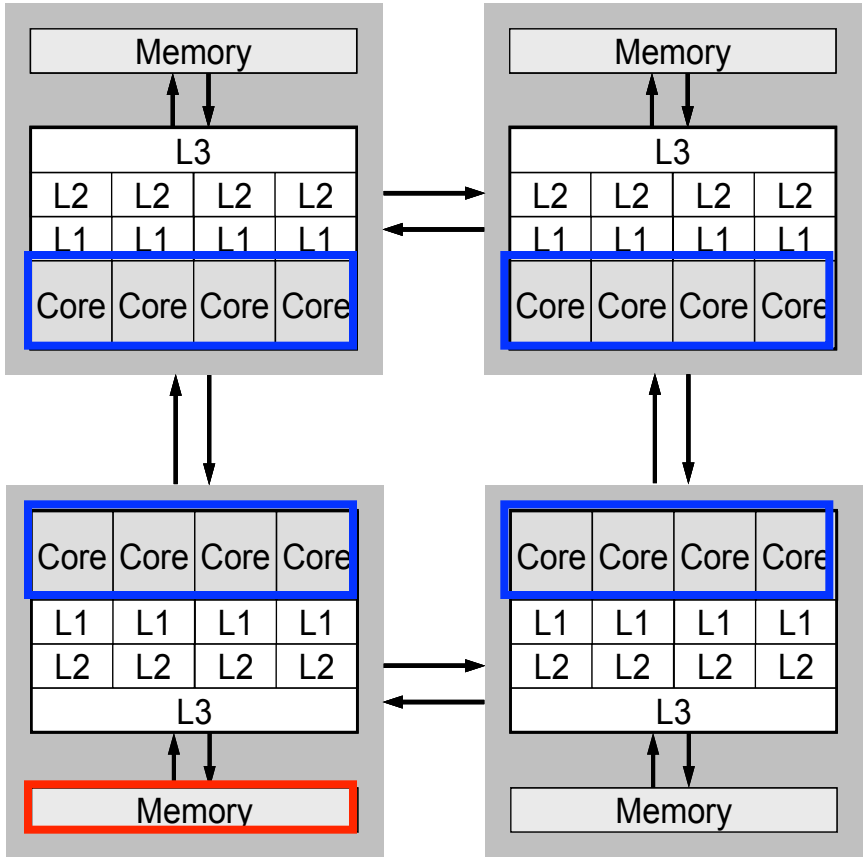
First Touch Data Placement

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

- 青字：ローカルメモリに載ることが保証される変数
- 赤字：ローカルメモリに載ることが保証されない変数（右辺のp）

CASE-0,1における初期化

0番スレッドで初期化：全ての記憶領域が0番スレッドを実行するコアのローカルメモリに確保：効率悪い

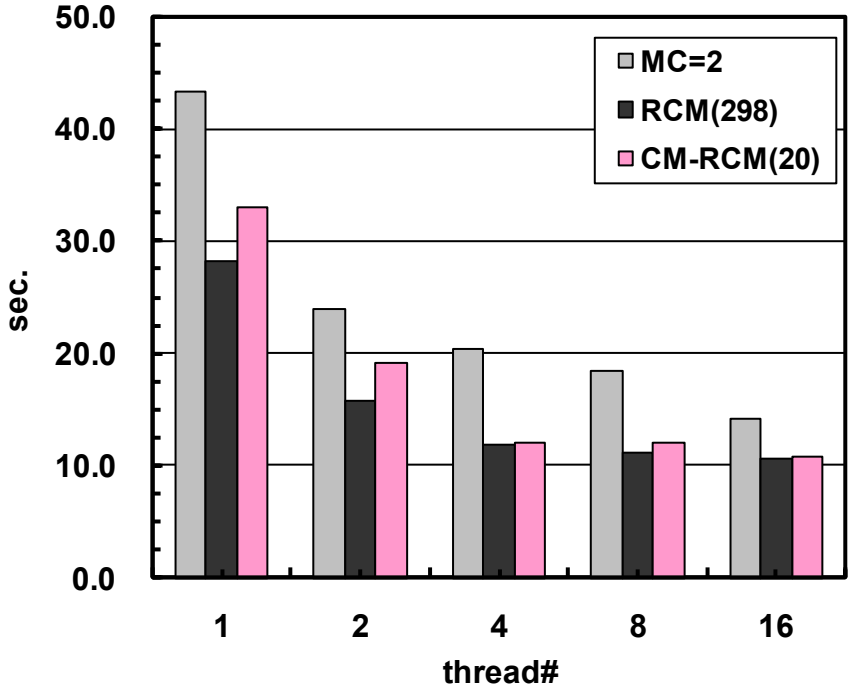



```
D = 0.d0
AL= 0.d0
AU= 0.d0
itemL= 0
itemU= 0
```

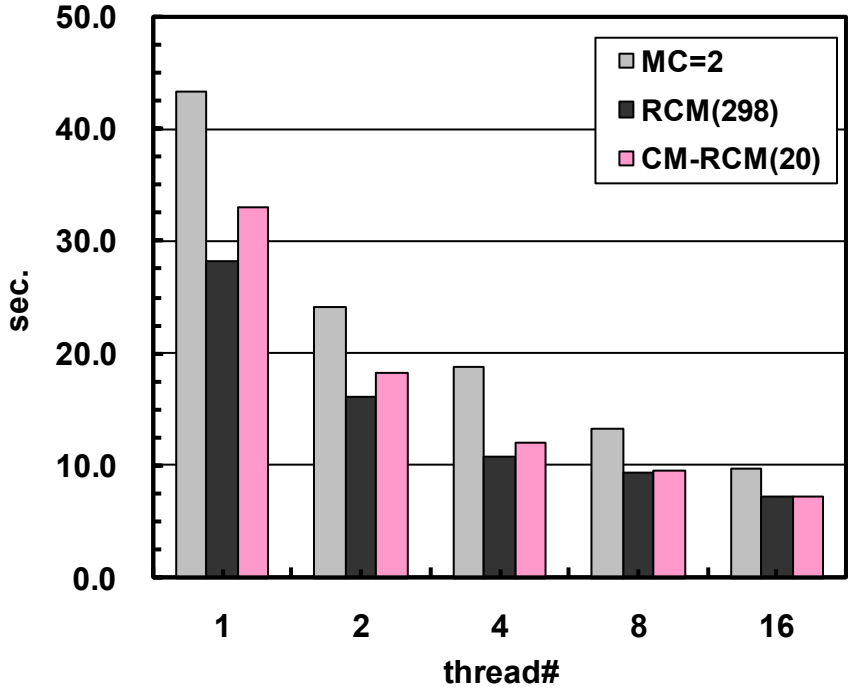
スケーラビリティ：numactlの効果

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

CASE-1a：一つのメモリにアクセス集中，インターリーブした方が性能が良くなる（アクセス負荷が分散）



CASE-1a
--localalloc



CASE-1b
--interleave=all

プログラムのありか

- 所在
  - <$L3>/ft
- コンパイル，実行方法
  - 本体
    - cd <$L3>/ft
    - Make
    - **<$L3>/run/L3-f-sol(実行形式)**
  - メッシュ生成
    - cd <$L3>/run
    - f90 -O mg.f -o mg
  - コントロールデータ
    - <$L3>/run/INPUT.DAT
  - 実行用シェル
    - <$L3>/run/f5.sh,f6.sh

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                         EPSICC
16                              PEsmpTOT
100                             NCOLORtot
1                               NFLAG
1                               METHOD
```

変数名	型	内　容
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
NFLAG	整数	0：First-Touch無し，1：あり
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）

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
  - indexL, indexU, itemL, itemU
  - AL, AU, D, BFORCE, PHI
    - これらを適切に初期化してやればよい
- INL，INU，IAL，IAUは「古い」番号の状態でFirst Touchが行われているが線形ソルバー等での計算には使われない

First Touch: NFLAG=0（無し）

```
      nn = ICELTOT
      allocate (indexL(0:nn), indexU(0:nn))

      indexL(0)= 0
      indexU(0)= 0

      if (NFLAG.eq.0) then
        do icel= 1, ICELTOT
          indexL(icel)= INL(icel)
          indexU(icel)= INU(icel)
        enddo
       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)
            indexL(icel)= INL(icel)
            indexU(icel)= INU(icel)
          enddo
        enddo
        enddo
      endif

      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))
      allocate (BFORCE(nn), D(nn), PHI(nn))

      if (NFLAG.eq.0) then
        BFORCE= 0.d0
        D= 0.d0
        PHI= 0.d0
        itemL= 0
        itemU= 0
           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
            do k= indexL(icel-1)+1, indexL(icel)
              itemL(k)= 0
                 AL(k)= 0.d0
            enddo
            do k= indexU(icel-1)+1, indexU(icel)
              itemU(k)= 0
                 AU(k)= 0.d0
            enddo
          enddo
        enddo
      enddo
      endif
```

First Touch: NFLAG=1（有り）

```
      nn = ICELTOT
      allocate (indexL(0:nn), indexU(0:nn))

      indexL(0)= 0
      indexU(0)= 0

      if (NFLAG.eq.0) then
        do icel= 1, ICELTOT
          indexL(icel)= INL(icel)
          indexU(icel)= INU(icel)
        enddo
       else
        do ic= 1, NCOLORtot
!$omp parallel do private (ip,icel)
        do ip= 1, PEsmpTOT
          do icel= SMPindex((ic-1)*PEsmpTOT+ip-1)+1,
     &             SMPindex((ic-1)*PEsmpTOT+ip)
            indexL(icel)= INL(icel)
            indexU(icel)= INU(icel)
          enddo
        enddo
        enddo
      endif

      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))
      allocate (BFORCE(nn), D(nn), PHI(nn))

      if (NFLAG.eq.0) then
        BFORCE= 0.d0
        D= 0.d0
        PHI= 0.d0
        itemL= 0
        itemU= 0
           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
            do k= indexL(icel-1)+1, indexL(icel)
              itemL(k)= 0
                 AL(k)= 0.d0
            enddo
            do k= indexU(icel-1)+1, indexU(icel)
              itemU(k)= 0
                 AU(k)= 0.d0
            enddo
          enddo
        enddo
      enddo
      endif
```

行列ベクトル積の計算法

METHOD=0
solver_ICCG_mc

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

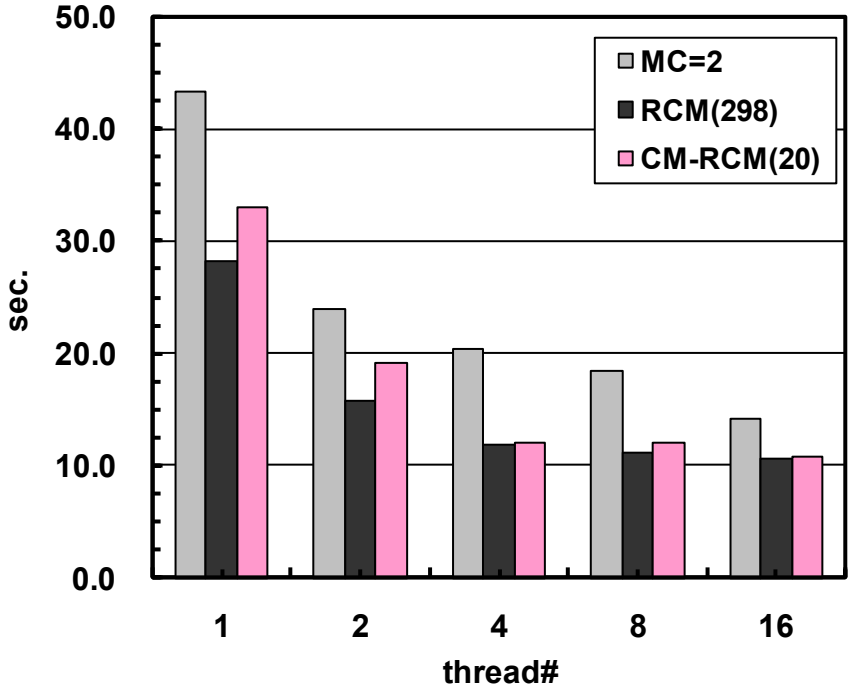
METHOD=1
solver_ICCG_mc_ft

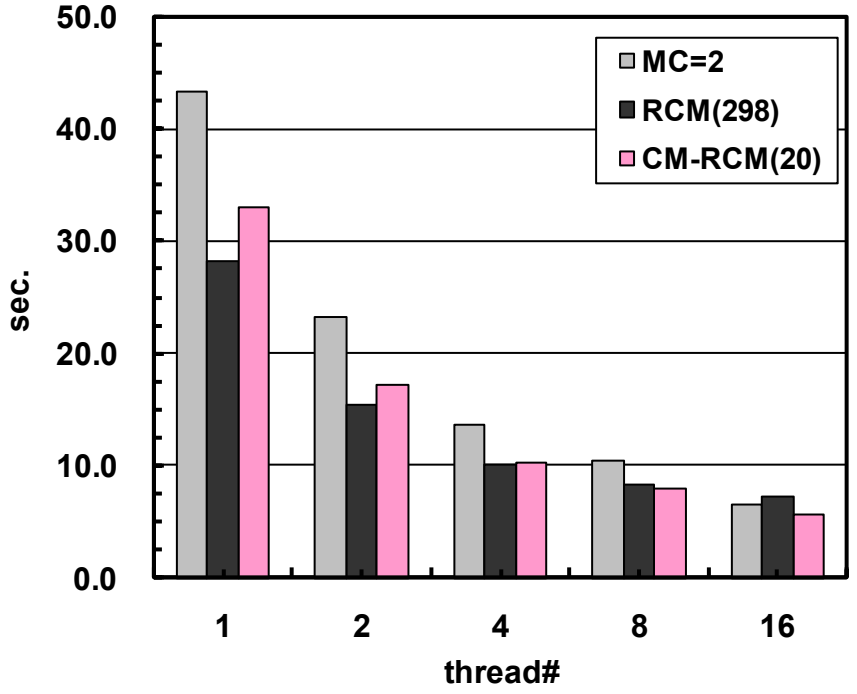
```
      do ic= 1, NCOLORtot
!$omp parallel do private(ip,ip1,i,VAL,k)
        do ip= 1, PEsmpTOT
          ip1= (ic-1)*PEsmpTOT + ip
          do i= SMPindex(ip1-1)+1, SMPindex(ip1)
            VAL= D(i)*W(i,P)
            do k= indexL(i-1)+1, indexL(i)
              VAL= VAL + AL(k)*W(itemL(k),P)
            enddo
            do k= indexU(i-1)+1, indexU(i)
            VAL= VAL + AU(k)*W(itemU(k),P)
          enddo
          W(i,Q)= VAL
        enddo
        enddo
!$omp end parallel do
      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（Nc=20）：249回

CASE-1a⇒2a：改善が見られる





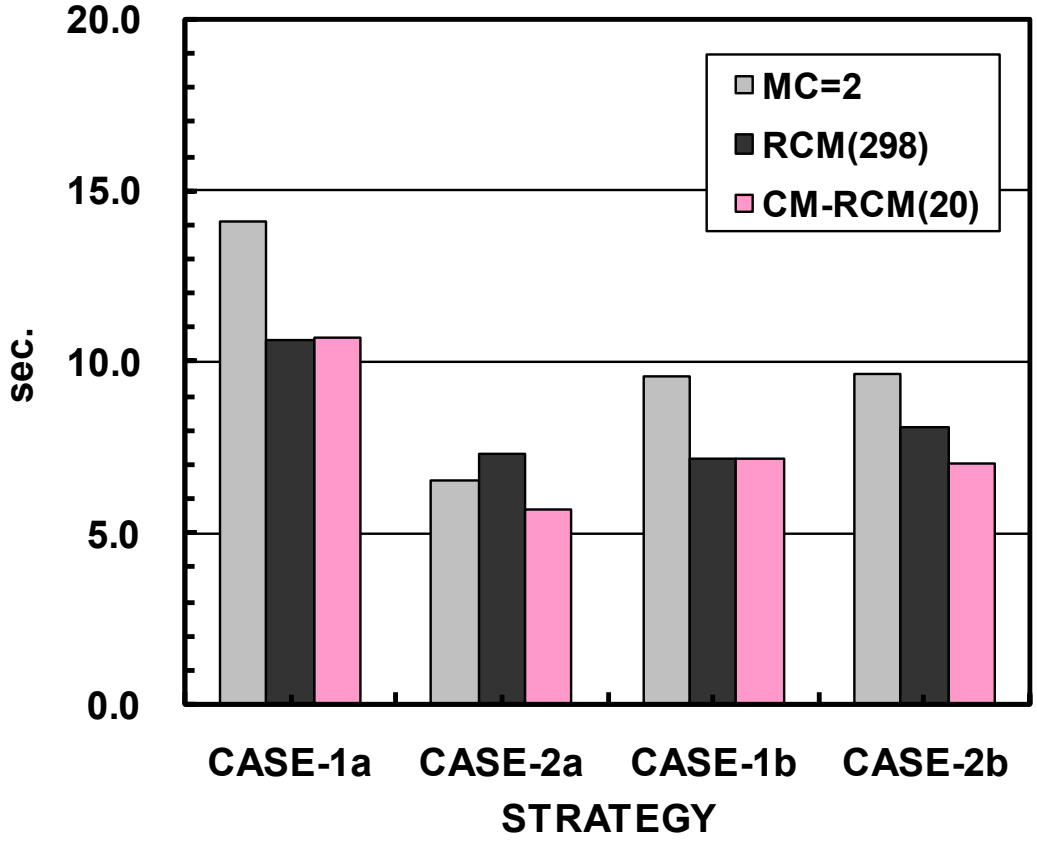
CASE-1a
--localalloc

CASE-2a（f5.sh）
METHOD=1の方が良い

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（Nc=20）：249回

16コア時における比較，CASE-1b⇒2bは変わらず
インターリーブするとFirst Touchは効かなくなる



- 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：Sequential Reordering**
- **他システムとの比較**

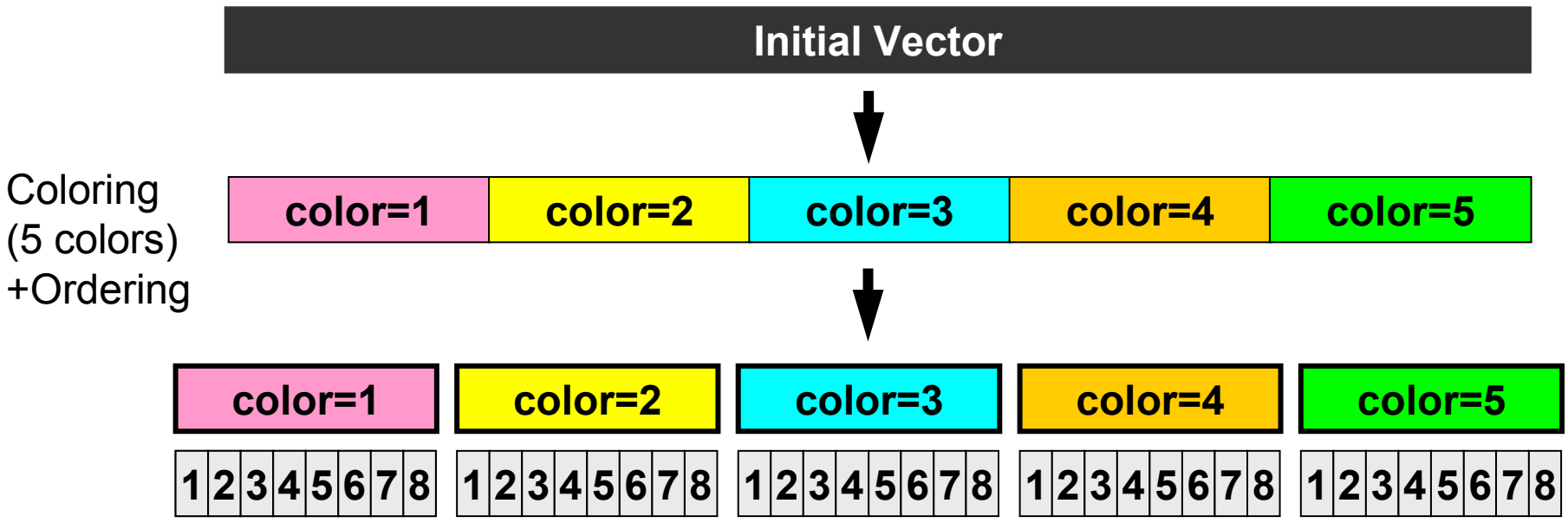
現在のオーダリングの問題

- 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
```



- 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

データ再配置：Sequential Reordering

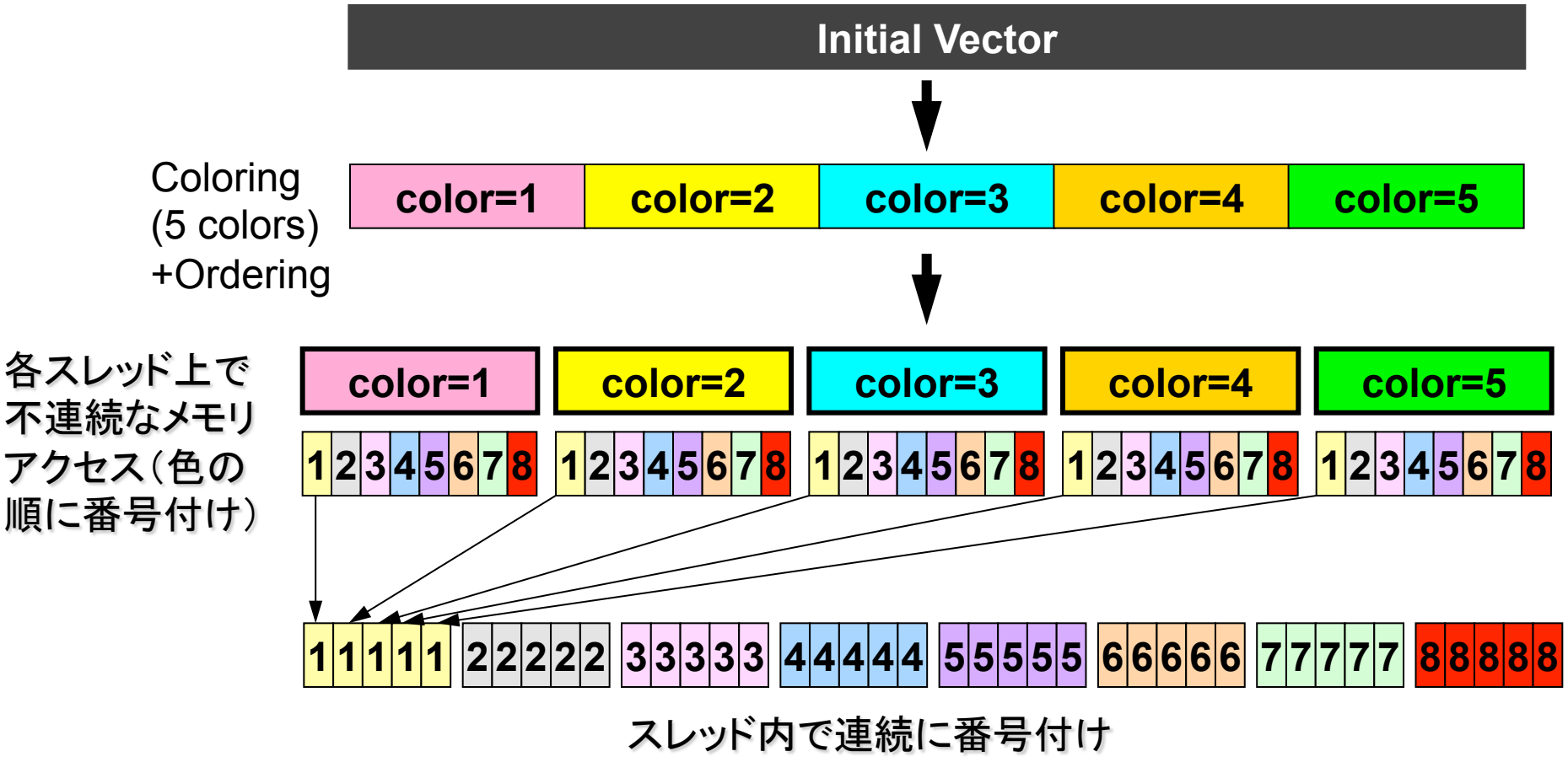
- 同じスレッドで処理するデータをなるべく連続に配置するように更に並び替え
  - 効率の向上が期待される
    - 係数行列等のアドレスが連続になる
    - 局所性が高まる（2ページあと）
- 番号の付け替えによって要素の大小関係は変わるが，上三角，下三角の関係は変えない（もとの計算と反復回数は変わらない）
  - 従って自分より要素番号が大きいのにIAL（下三角）に含まれていたりする

- First-touch + データ再配置：CASE3

データ再配置: Sequential Reordering

各スレッド上でメモリアクセスが連続となるよう更に並び替え

5 colors, 8 threads



データ再配置：Sequential Reordering

CM-RCM(2), 4-threads

番号付けの順番にもよるが，概して同じスレッド上には「近く」の要素が来る。従って，番号も近くなるように連続してふってやると効率良い：局所性，キャッシュ有効利用

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

CM-RCM(2)

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

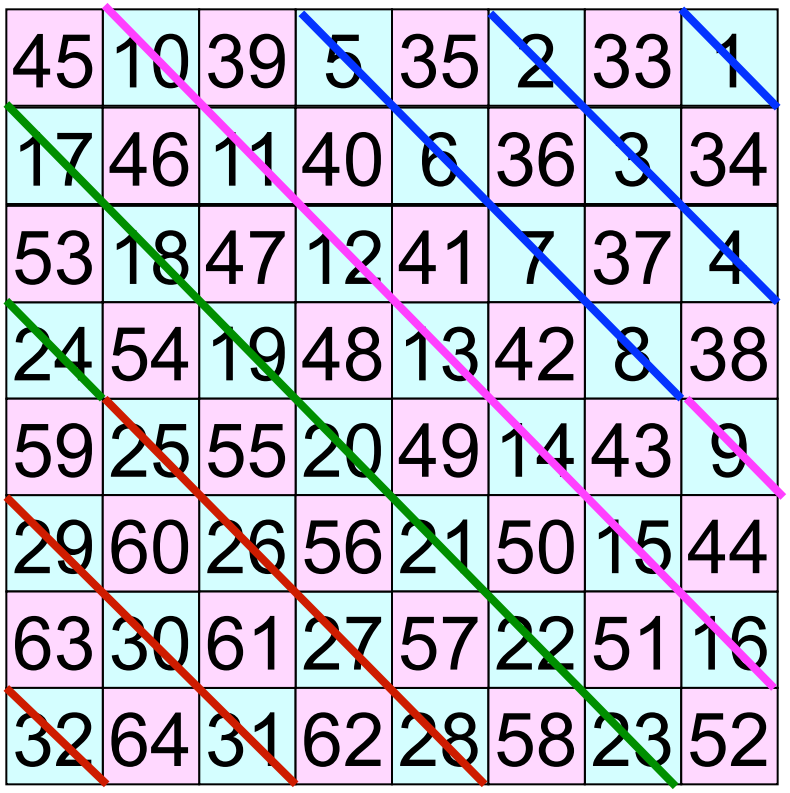
Sequential Reordering, 4-threads

データ再配置：Sequential Reordering

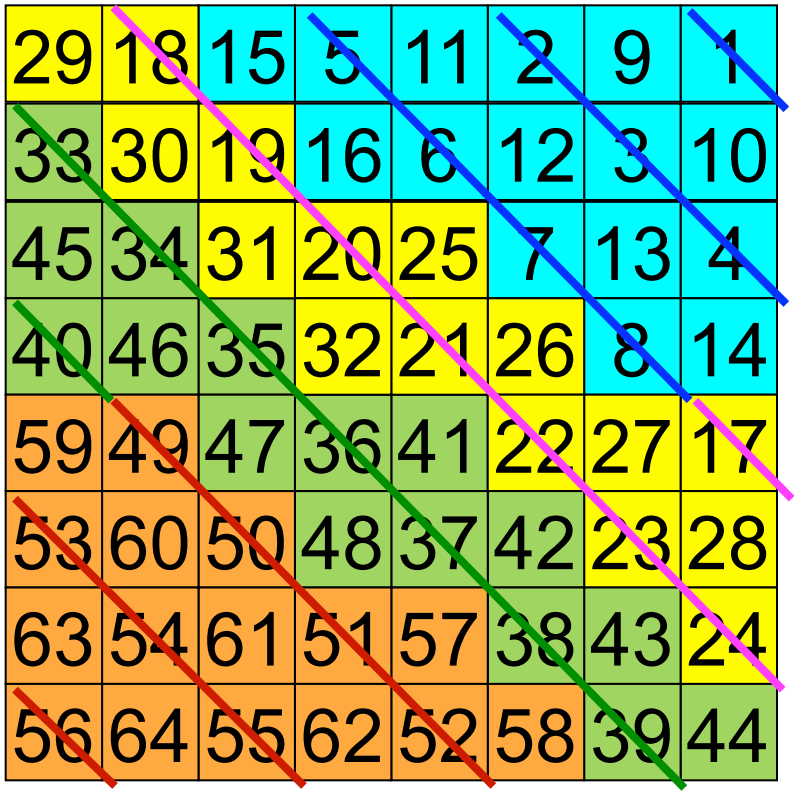
CM-RCM(2), 4-threads

第1色

#0 thread, #1, #2, #3



CM-RCM(2)



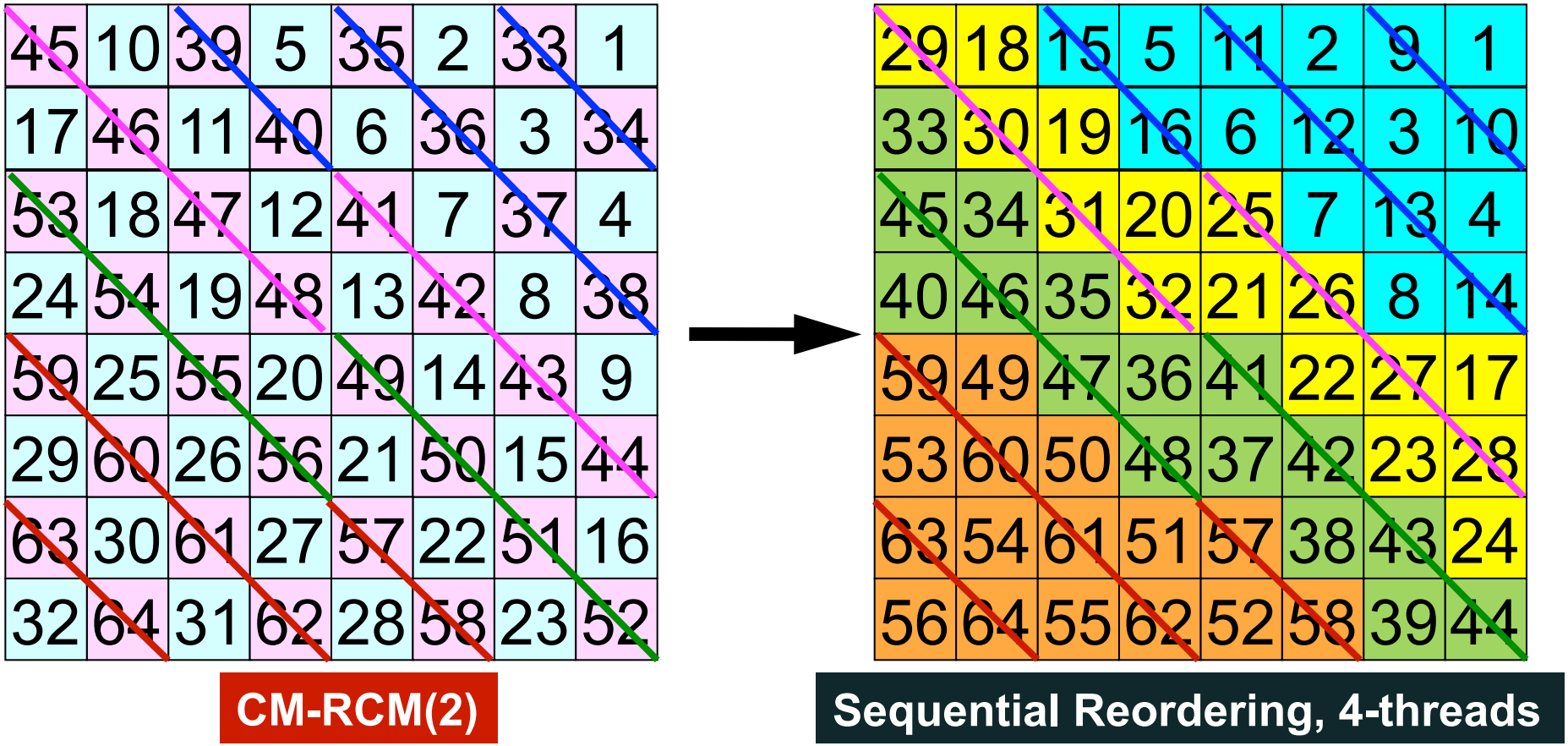
Sequential Reordering, 4-threads

データ再配置：Sequential Reordering

CM-RCM(2), 4-threads

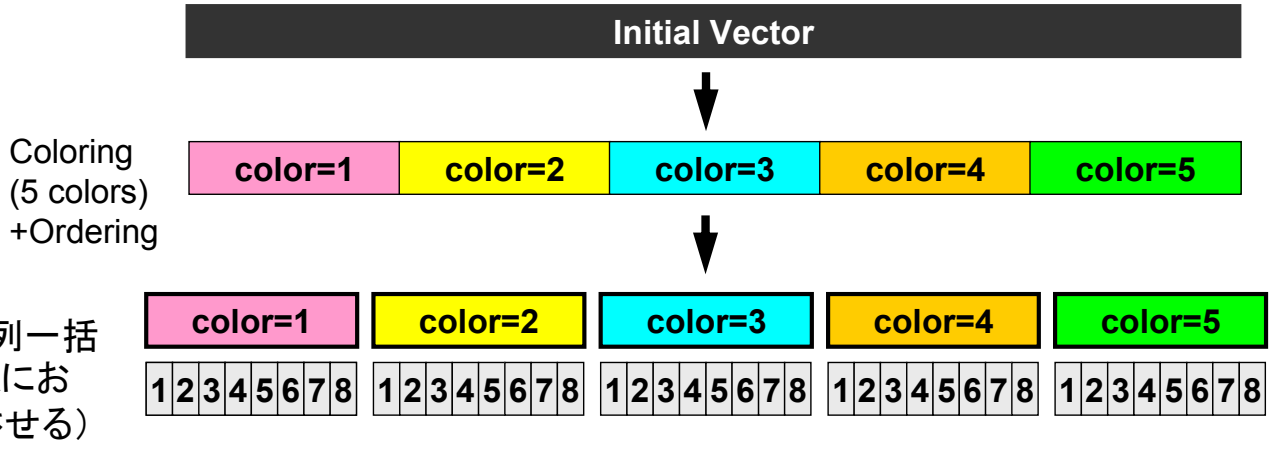
第2色

#0 thread, #1, #2, #3



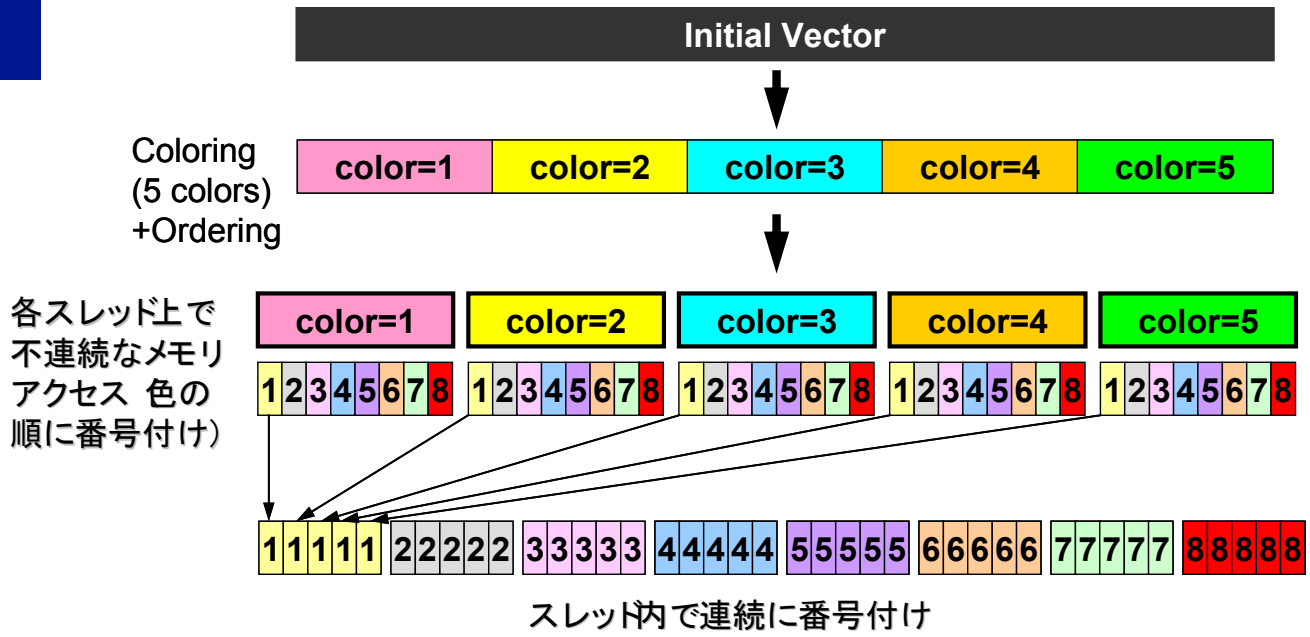
データ再配置: Sequential Reordering

Coalesced
GPUはこちらが
お勧め



それぞれの色について細粒度な並列一括アクセスを行うことで高性能（CUDAにおける各ThreadBlockに各色を担当させる）

Sequential



プログラムのありか

- 所在
  - <$L3>/reorder
- コンパイル，実行方法
  - 本体
    - cd <$L3>/reorder
    - make
    - <$L3>/reorder/L3-r-sol(実行形式)
  - メッシュ生成
    - cd <$L3>/run
    - f90 -O mg.f -o mg
  - コントロールデータ
    - <$L3>/run/INPUT.DAT
  - 実行用シェル
    - <$L3>/run/r5.sh,r6.sh

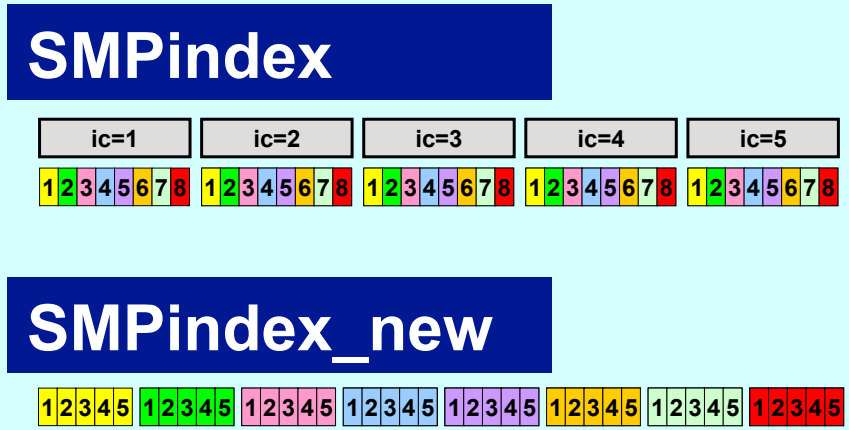
制御データ(INPUT.DAT)

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

変数名	型	内　容
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
NFLAG	整数	0：First-Touch無し，1：あり
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）

再配置：Sequential Reordering

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

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

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



行列ベクトル積の計算法

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

Original

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

New

前進代入の計算法:色ループは外

```
      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
```

Original

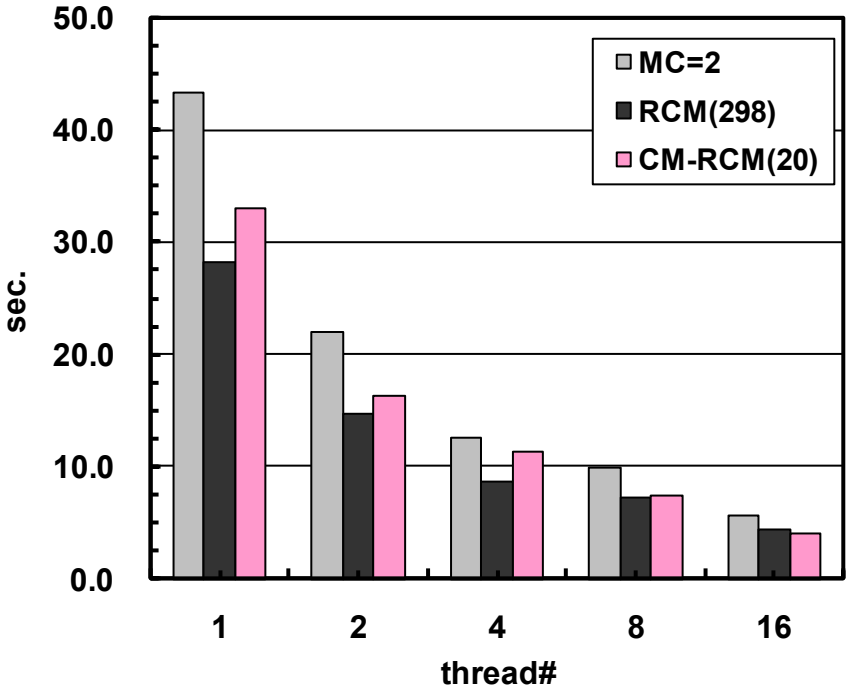
```
      do ic= 1, NCOLORtot
!$omp parallel do private(ip,ip1,i,WVAL,k)
        do ip= 1, PEsmpTOT
          ip1= (ip-1)*NCOLORtot + ic
        do i= SMPindex(ip1-1)+1, SMPindex(ip1)
          WVAL= W(i,Z)
          do k= indexL(i-1)+1, indexL(i)
            WVAL= WVAL -  AL(k) * W(itemL(k),Z)
    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

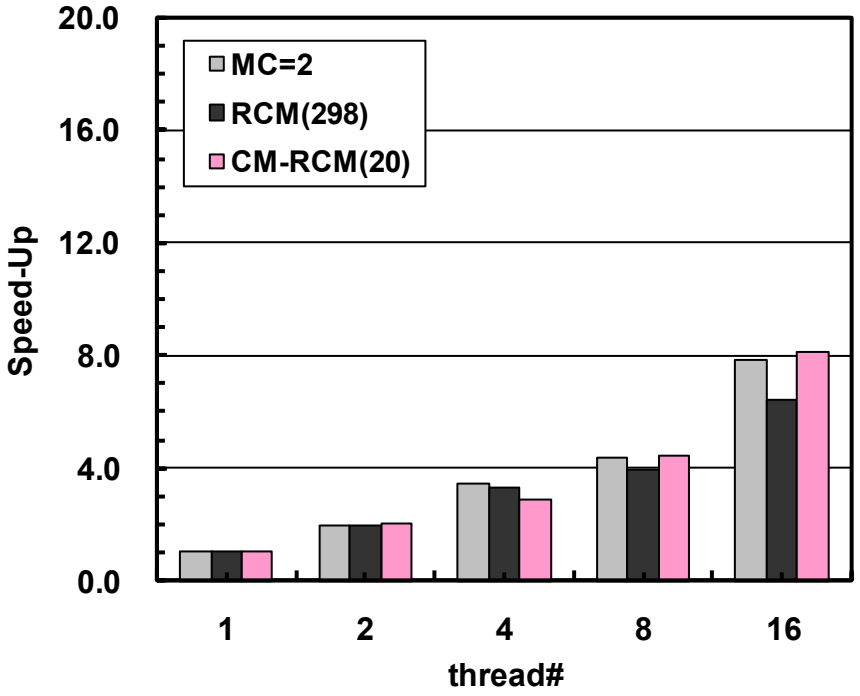
New

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（Nc=20）：249回

CASE-3a（--localalloc）：METHOD=0の方が良い
RCMはコア数増えると性能相対的に低下

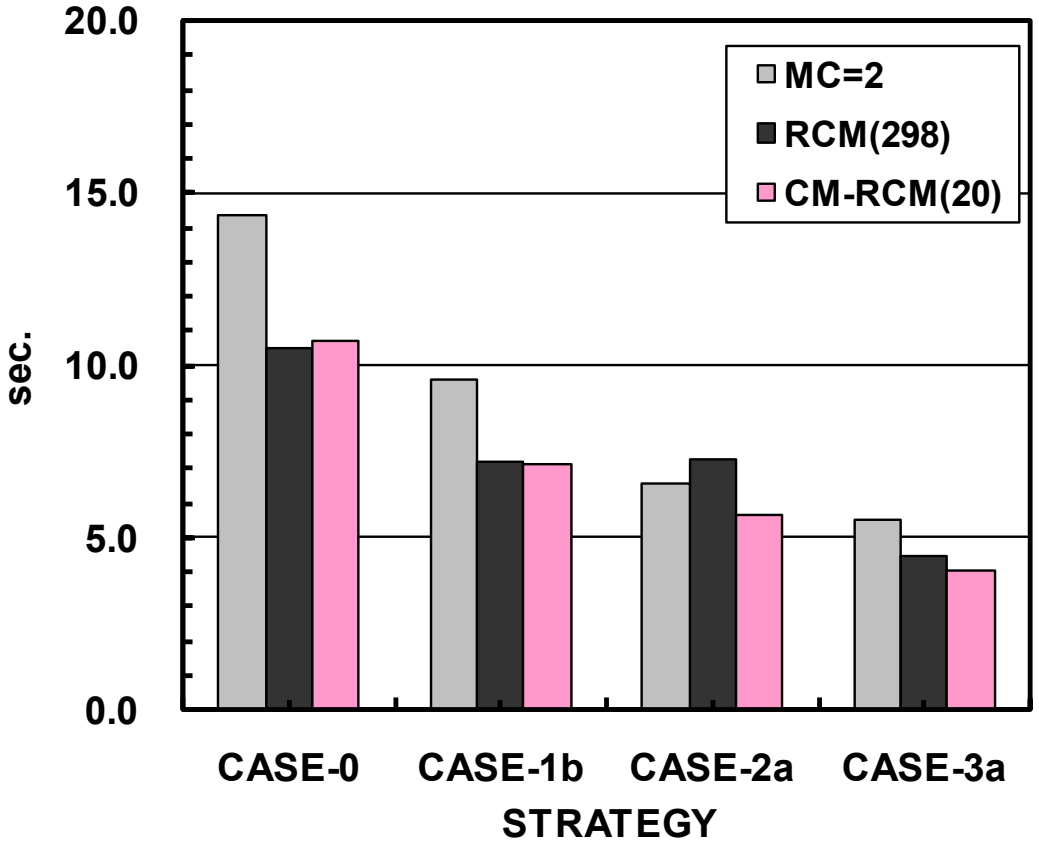




CASE-0⇒CASE-3:改良の履歴

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

16コア時における比較

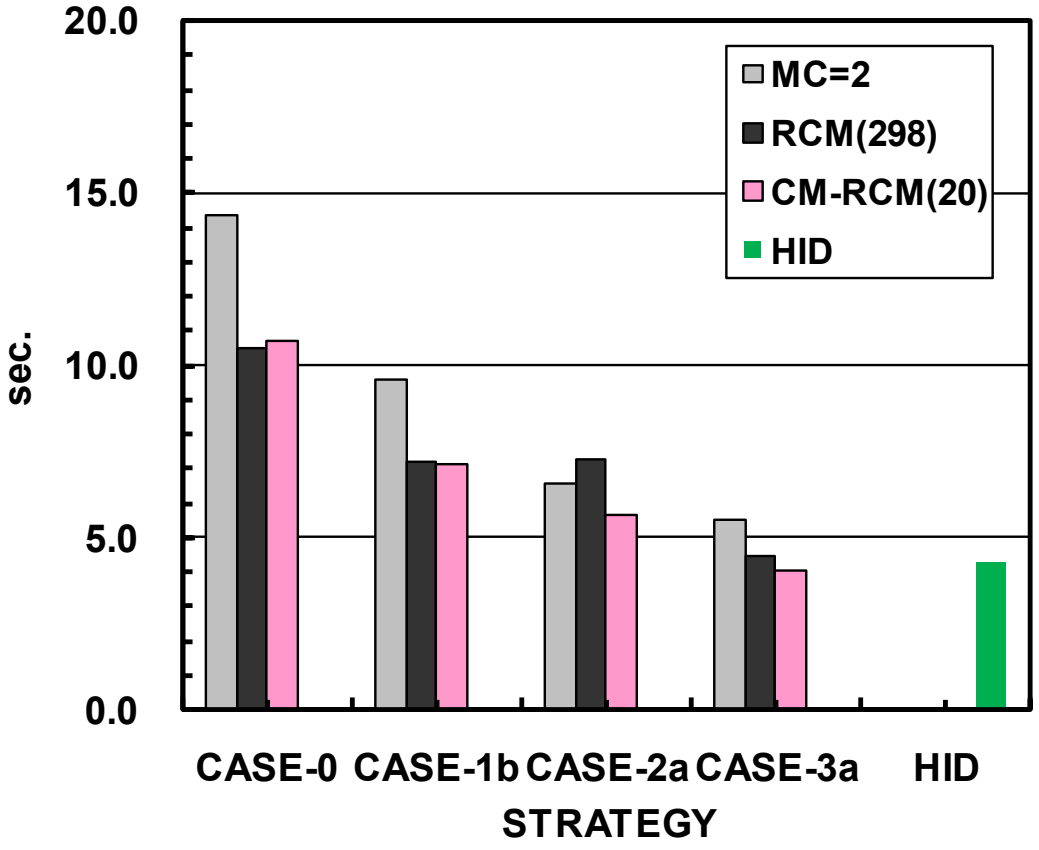


HID

Hierarchical Interface Decompotision

最新の手法：CASE-3aとほぼ同じ

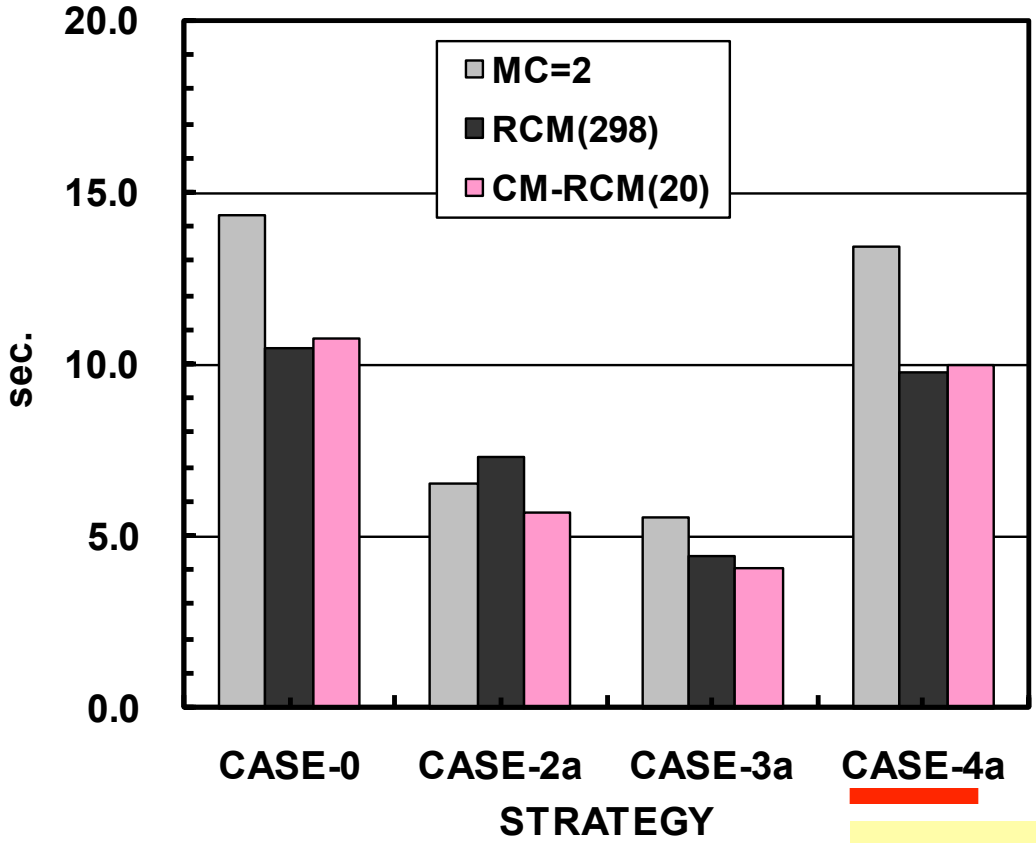
16コア時における比較



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（Nc=20）：249回

16コア時における比較



First-touch無し

- 他システムとの比較

- STREAM Benchmark
- Hitachi SR11000/J2との傾向の比較
- Hopper at Lawrence Berkeley National Laboratoryの結果、Westmereの結果

STREAM ベンチマーク

http://www.streambench.org/

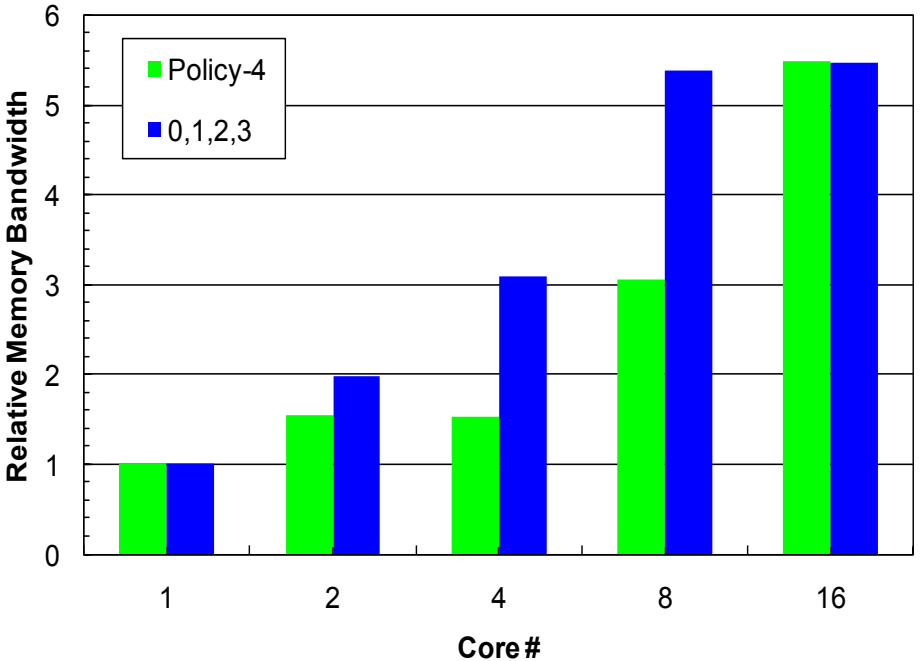
- メモリバンド幅を測定するベンチマーク
  - Copy, Scale, Add, Triad（DAXPYと同じ）

```
-------------------------------------------------
 Double precision appears to have 16 digits of accuracy
 Assuming 8 bytes per DOUBLE PRECISION word
-------------------------------------------------
 Number of processors =           16
 Array size =    2000000
 Offset     =          0
 The total memory requirement is     732.4 MB (     45.8MB/
task)
 You are running each test  10 times
 --
 The *best* time for each test is used
 *EXCLUDING* the first and last iterations
 -----------------------------------------------------
 -----------------------------------------------------
Function     Rate (MB/s)  Avg time   Min time  Max time
Copy:       18334.1898      0.0280      0.0279      0.0280
Scale:      18035.1690      0.0284      0.0284      0.0285
Add:        18649.4455      0.0412      0.0412      0.0413
Triad:      19603.8455      0.0394      0.0392      0.0398
```

Triadの結果（T2K）

Policy-4@T1の性能（メモリバンド幅）を1.00

- MPIの場合，NUMA　Policyによる比較
- 16コア使った場合，メモリの性能は5倍強程度
- FVMのようなMemory-Boundなアプリケーションは，このメモリ性能に全体の性能が左右される
  - CASE-3a: 約8倍@16コア
- 興味のある人は別資料を参照（サンプルプログラムもありますので，自分でやってみてください，後掲ウェブサイトに資料あります）

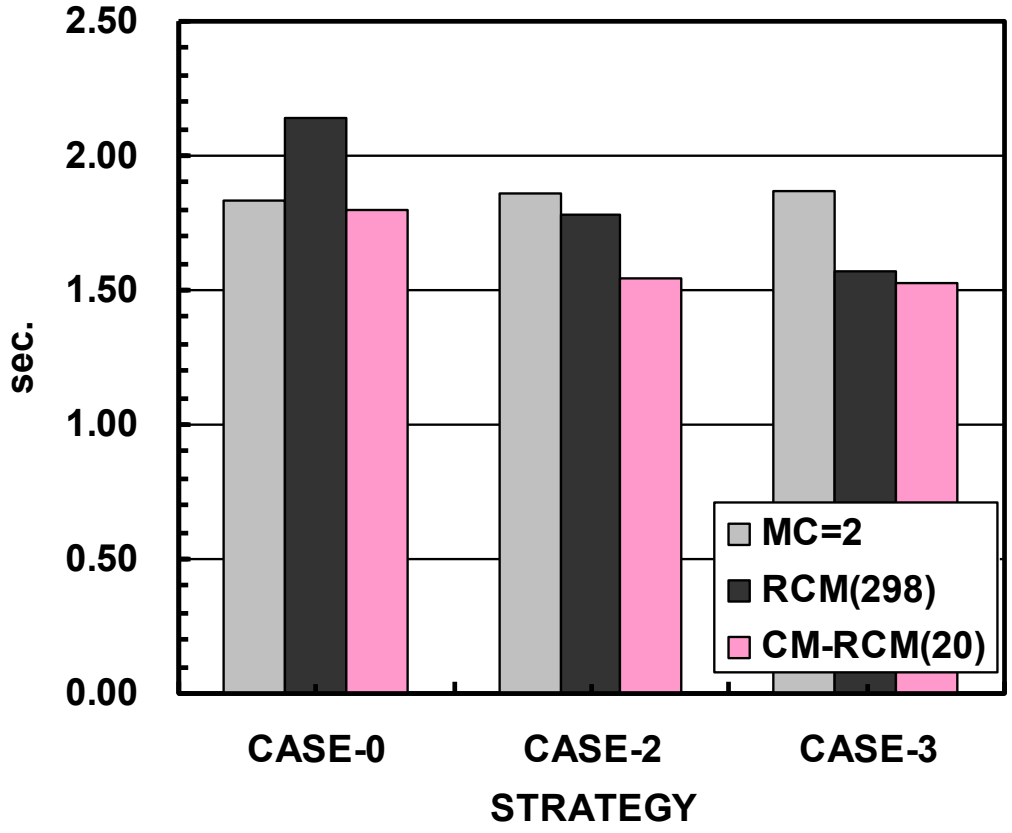


- 他システムとの比較

- STREAM Benchmark
- **Hitachi SR11000/J2との傾向の比較**
- Hopper at Lawrence Berkeley National Laboratoryの結果、Westmereの結果

Hitachi SR11000/J2での結果：16コア

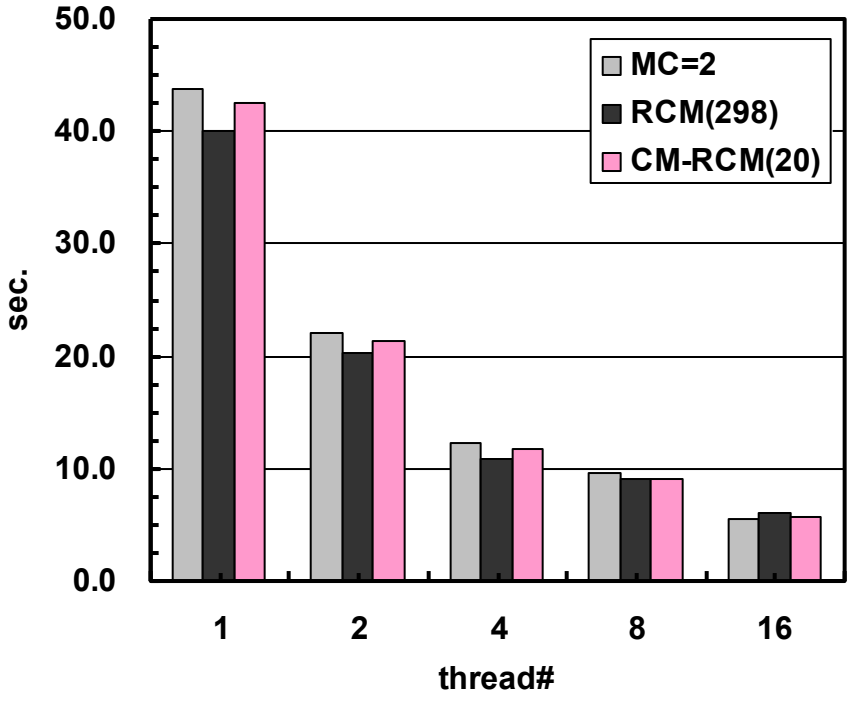
- オーダリング法によって影響が異なる
  - 色数を増やしたときに，First-touch，再配置の効果が出る



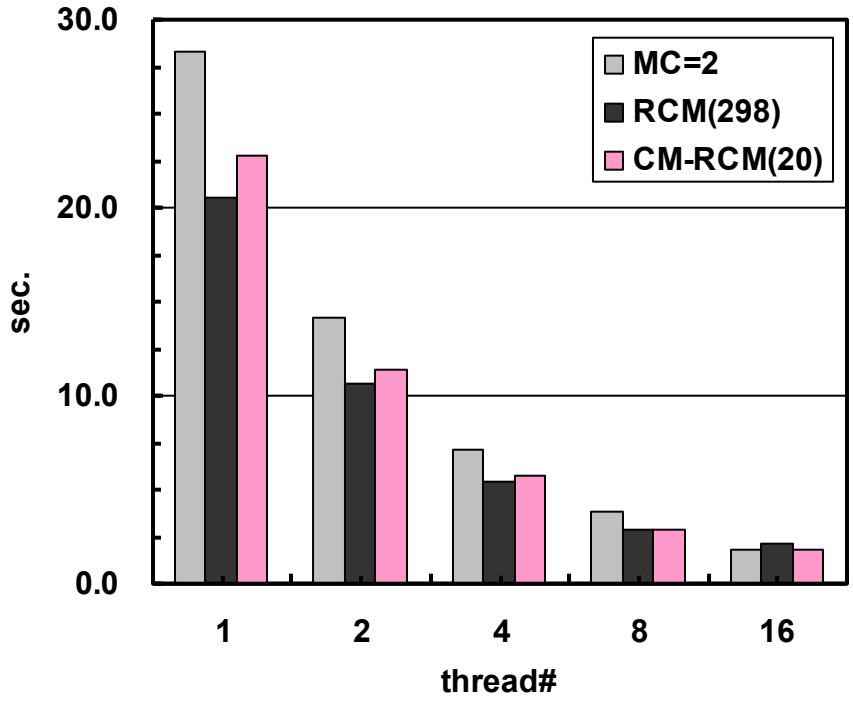
T2KとSR11000（旧バージョン）

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

T2K: CASE-3a

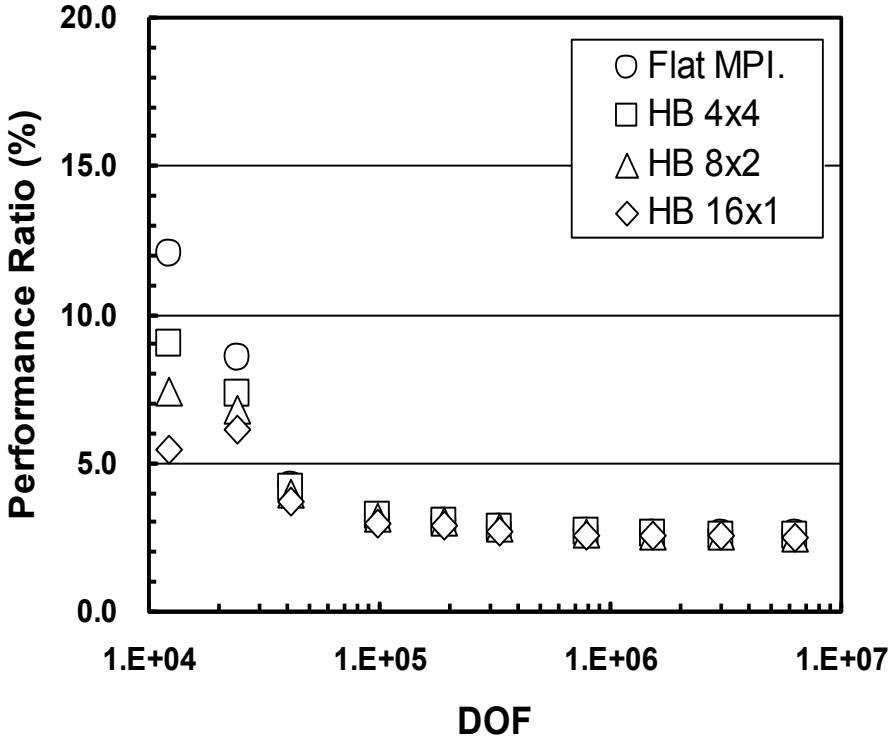


SR11000: CASE-3

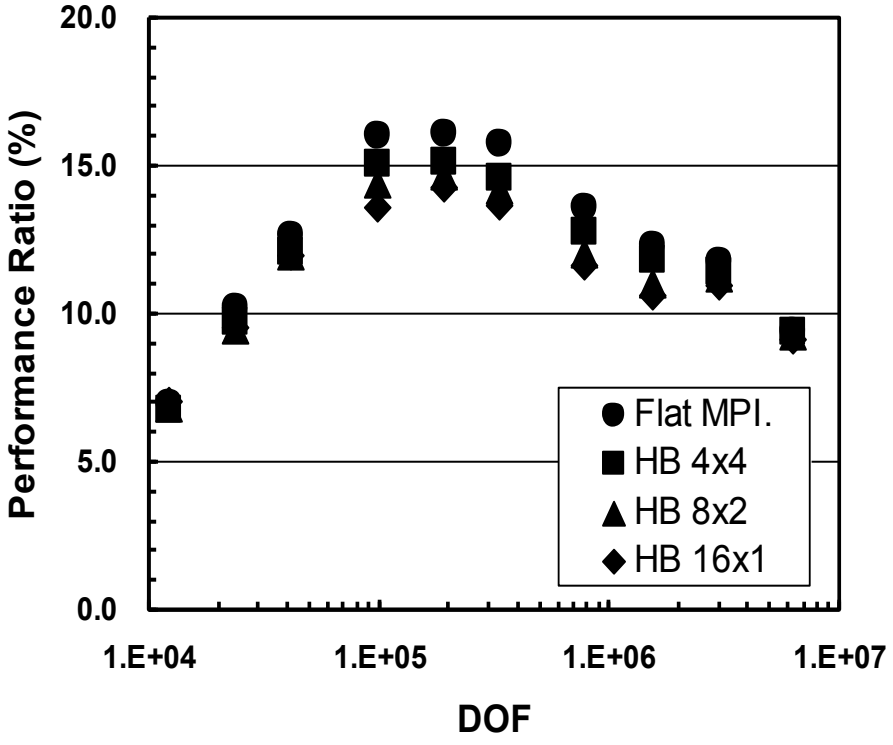


GeoFEM Benchmark 1-node with 16-cores: ICCG

T2K/Tokyo
Opteron 2.3GHz x 16
147.2 GFLOPS/node
20 GB/s for STREAM/Triadd



Hitachi SR11000/J2
Power 5+ 2.3GHz x 16
147.2 GFLOPS/node
100 GB/s for STREAM/Triadd



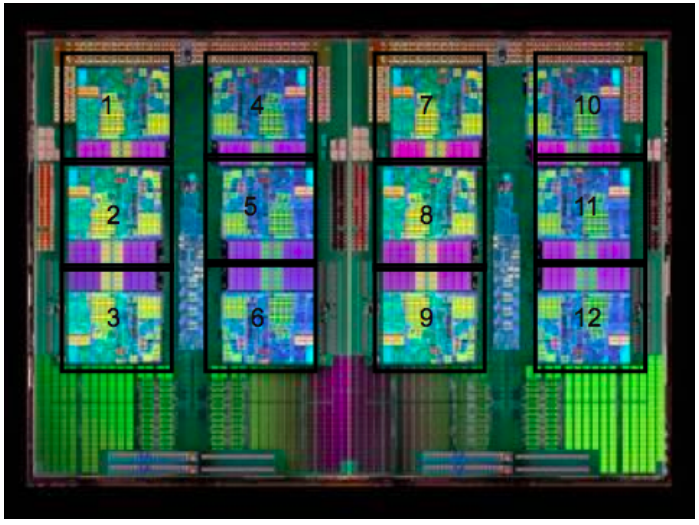
- 他システムとの比較

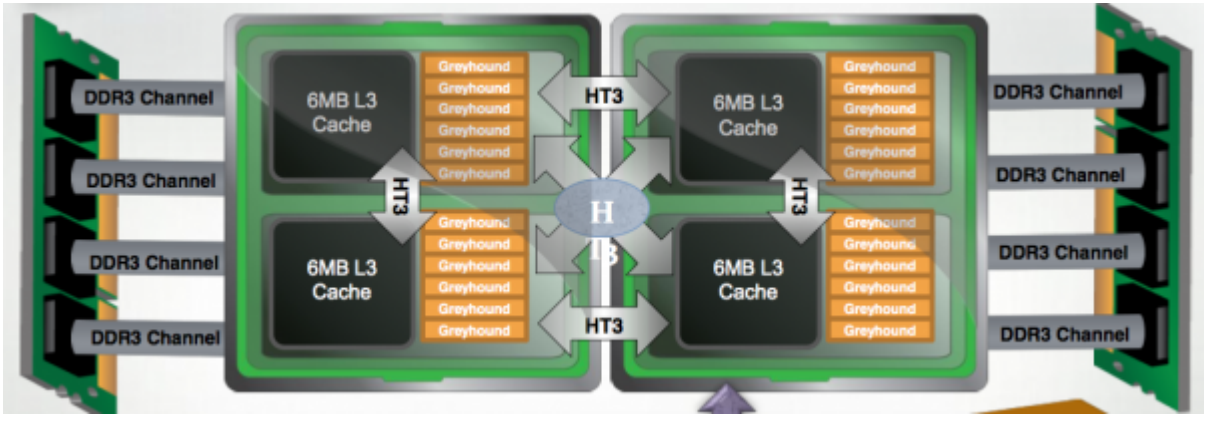
- STREAM Benchmark
- Hitachi SR11000/J2との傾向の比較
- **Hopper at Lawrence Berkeley National Laboratoryの結果、Westmereの結果**

Hopper: Cray XE6 at Lawrence Berkeley Natl. Laboratory

http://www.nersc.gov/systems/hopper-cray-xe6/

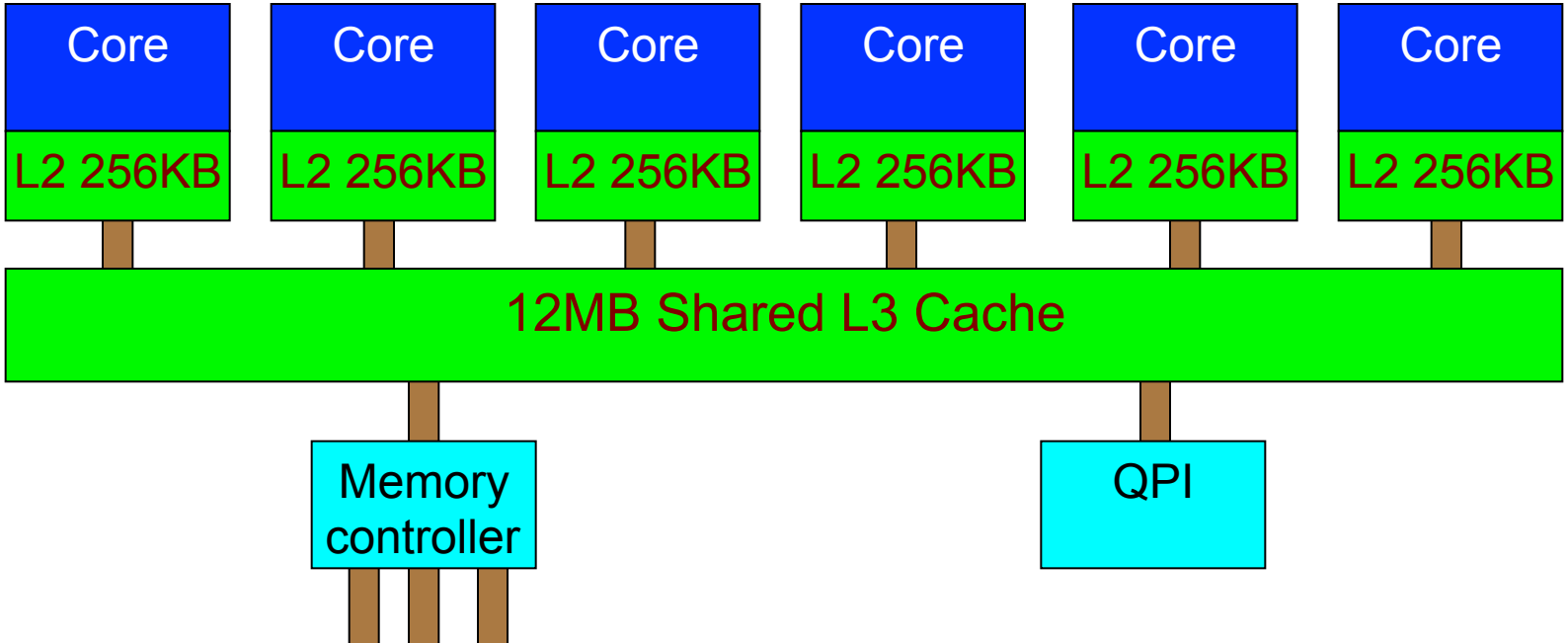
- 6,384 nodes (1.28PFLOPS)
- 2 x 12-core AMD 'MagnyCours' 2.1 GHz processors per node (4 x 6-core die)
- 24 cores per node
- 32/64 GB DDR3 1333 MHz





XeonX5650 Westmere

- Nehalemの拡張版
  - 2.66GHz（TB無効化設定）、 6コア、 32nmプロセス
  - 今回のシステムでは2基搭載（物理12コア、HTは今回は無視）

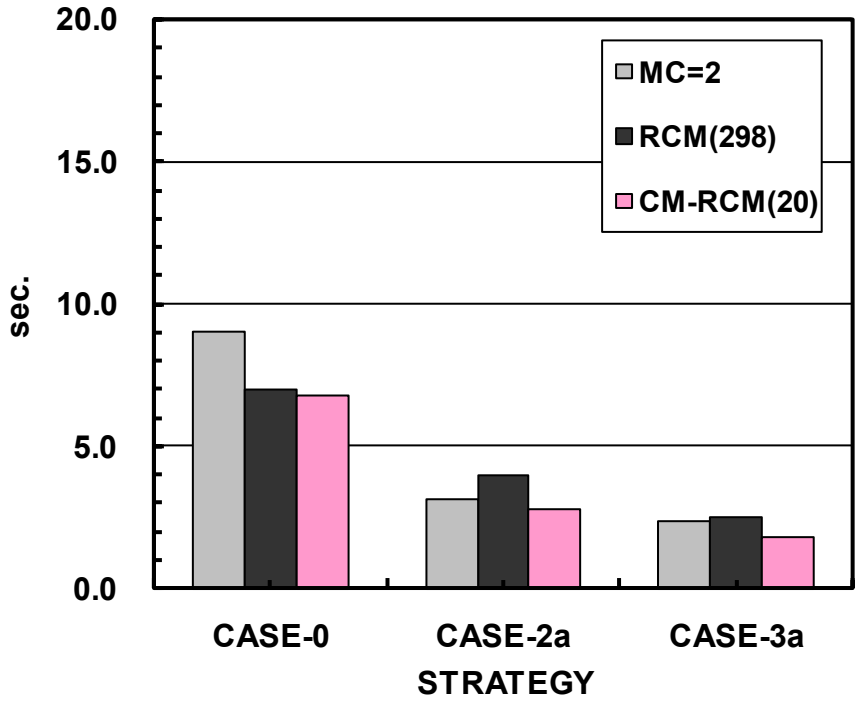


Hopper, T2K, Westmere

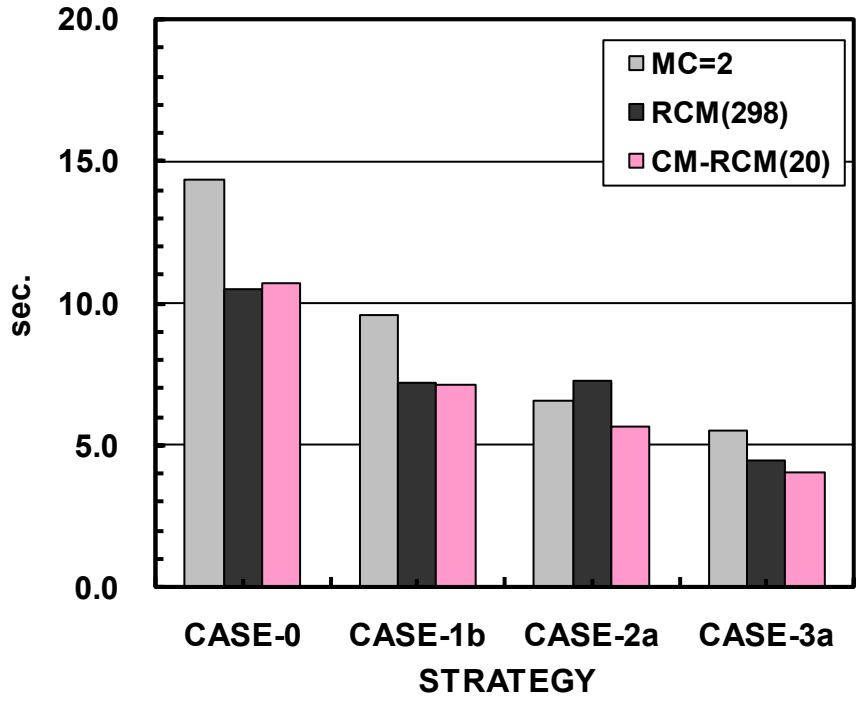
	Hopper	T2K	Xeon X5650 Westmere
Peak Performance/core (GFLOPS)	8.40	9.20	10.68
Core #/Node	24	16	12
Peak Performance/node (GFLOPS)	201.6	147.2	128.0
Triad Performance/node (GB/sec)	52.3	20.0	29.5
Triad Performance/core (GB/sec)	2.18	1.25	2.45
B/F Rate	0.260	0.136	0.229
GeoFEM Bench/ICCG (MFLOPS/core)	469.8	292.8	-

Hopper vs. T2K

Hopper with 24-cores

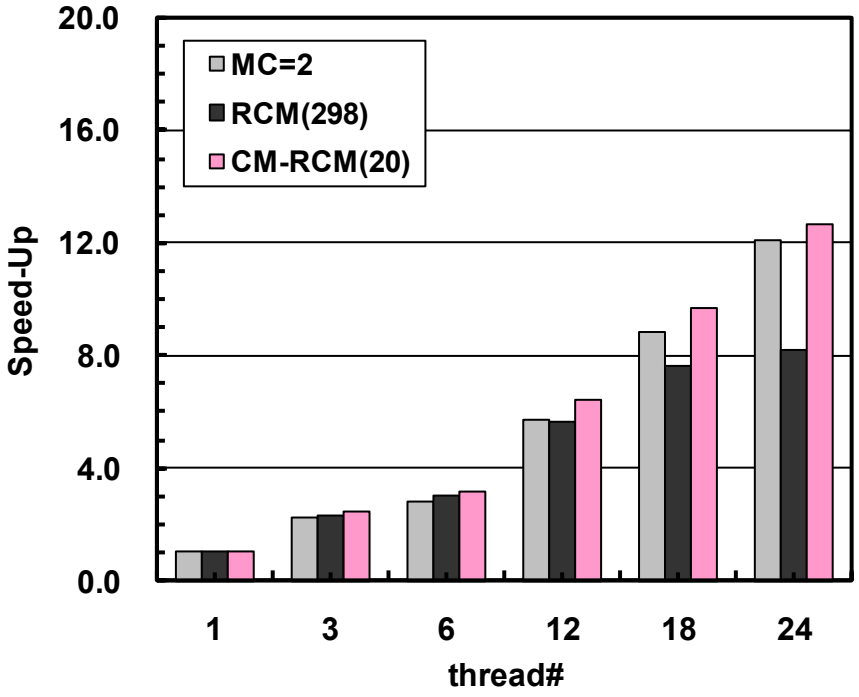


T2K with 16-cores

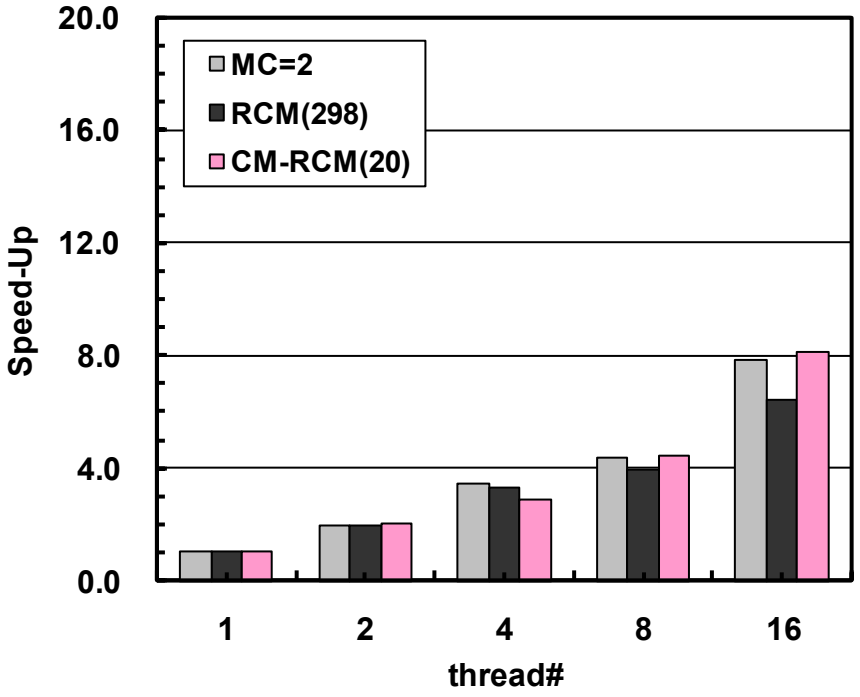


Hopper vs. T2K

Hopper with 24-cores

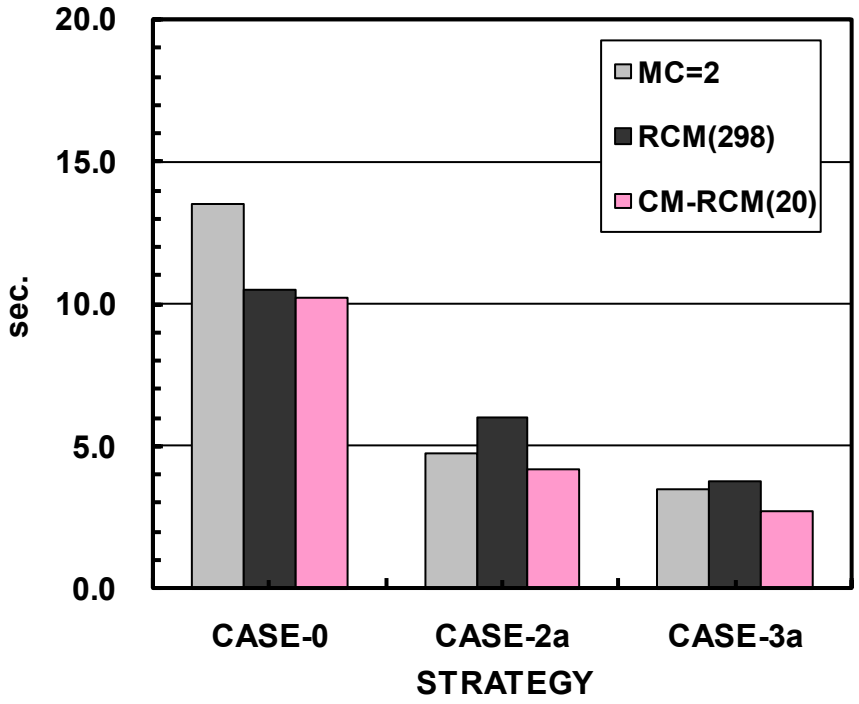


T2K with 16-cores

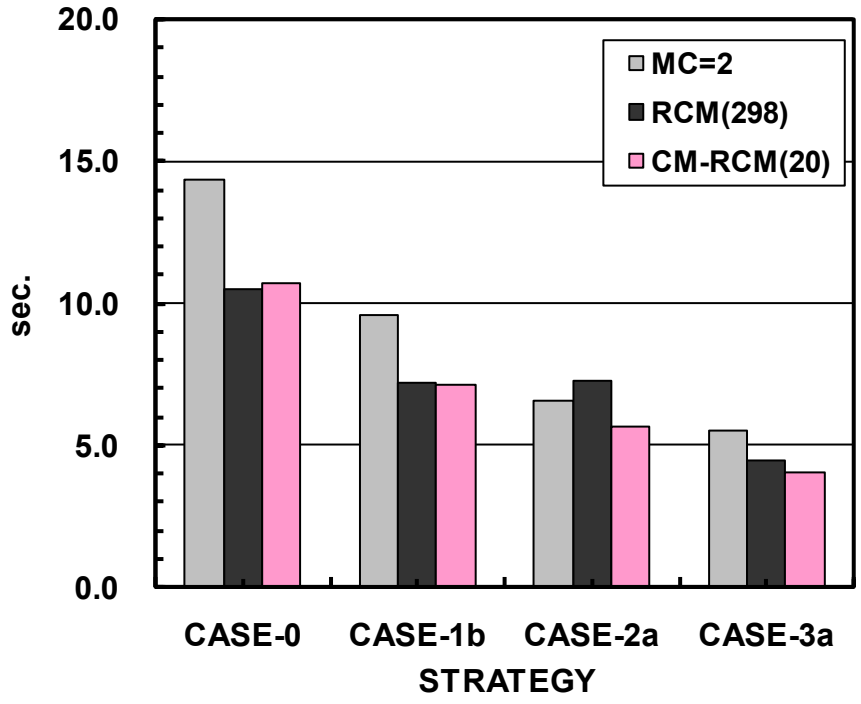


Hopper vs. T2K

Hopper with 16-cores (estimated)

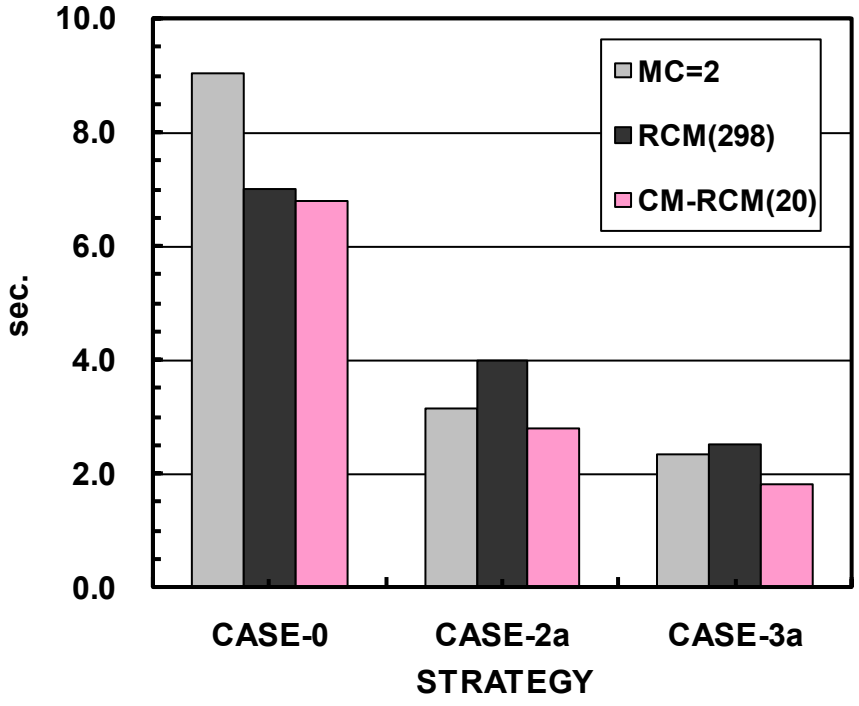


T2K with 16-cores

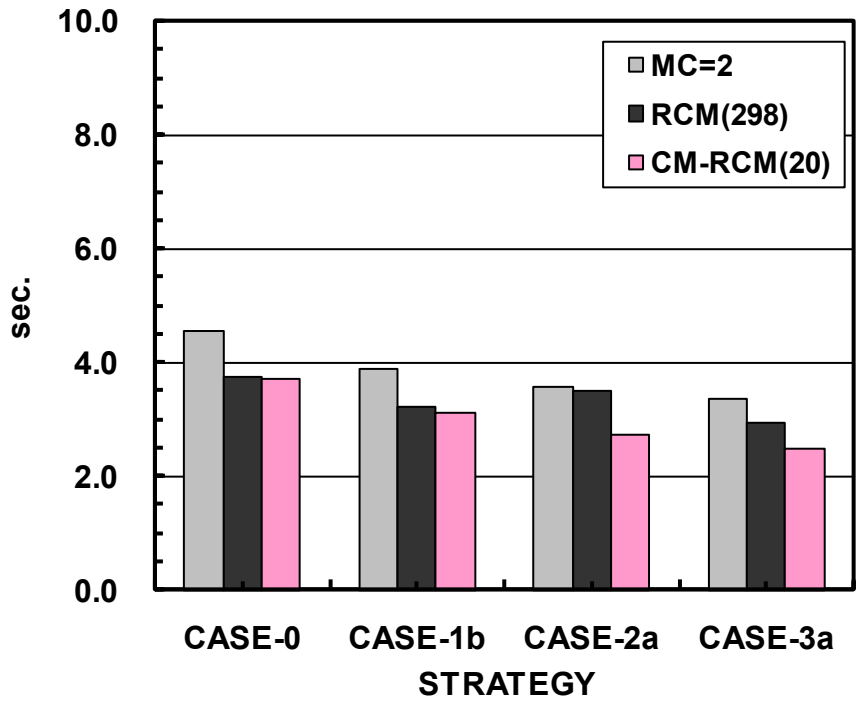


Hopper vs. Westmere

Hopper with 24-cores

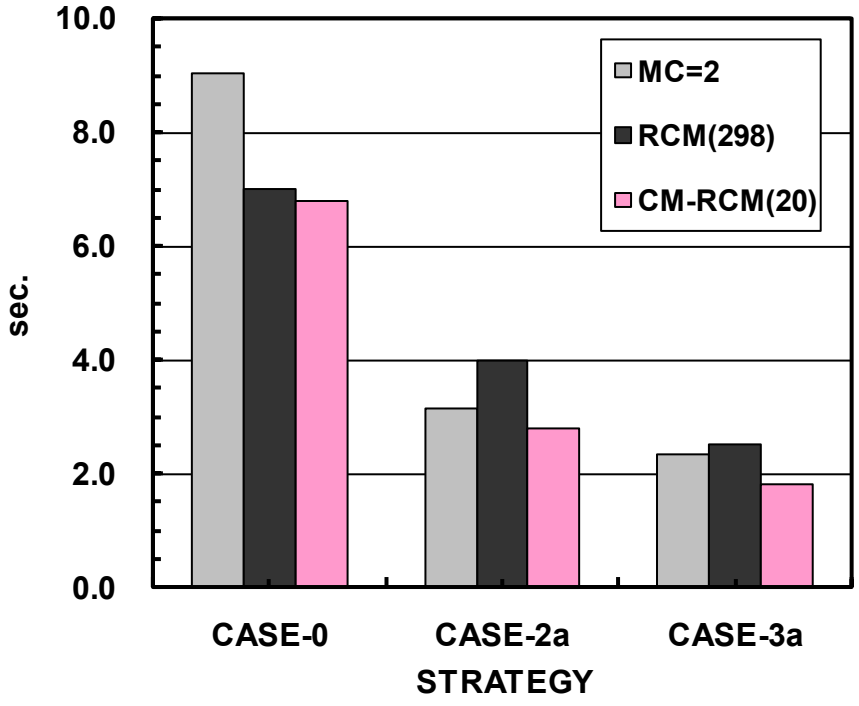


Westmere with 12-cores

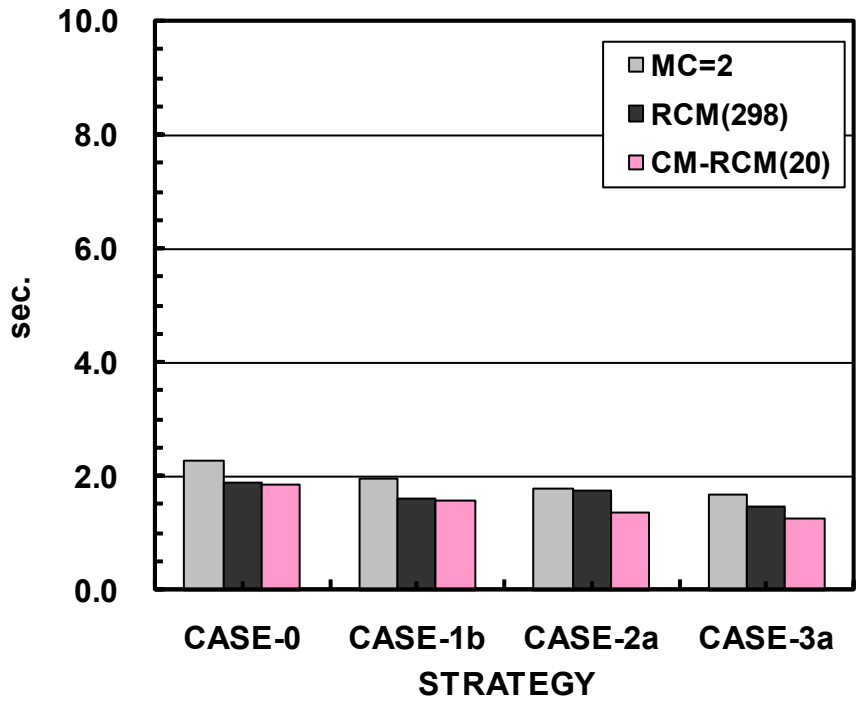


Hopper vs. Westmere

Hopper with 24-cores



Westmere with 24-cores (estimated)



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：Sequential Reordering

今後の動向

- メモリバンド幅と性能のギャップ
  - BYTE/FLOPの低下
- マルチコア化、メニーコア化、アクセラレータ活用
- >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14日（水）17:00まで利用可能
- 最終版の資料はWEBにアップ
  - http://www.cspp.cc.u-tokyo.ac.jp/ohshima/seminars/t2k201112/
- アンケートにお答えください