

# 数値計算アルゴリズム：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+演習

中島研吾

東京大学情報基盤センター

# OpenMP並列化

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

# 4色, 4スレッドの例

## 初期メッシュ

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

# 4色, 4スレッドの例

## 初期メッシュ

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

# 4色, 4スレッドの例 色の順に番号付け

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

# 4色, 4スレッドの例

同じ色の要素は独立: 並列計算可能  
番号順にスレッドに割り当てる

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

# ファイルコピー:FX10

```
>$ cd <$O-TOP>
```

```
>$ cp /home/ss/aics60/C/multicore-c.tar .
```

```
>$ cp /home/ss/aics60/F/multicore-f.tar .
```

```
>$ tar xvf multicore-c.tar
```

```
>$ tar xvf multicore-f.tar
```

```
>$ cd multicore
```

以下のディレクトリが出来ていることを確認

L3 stream

これらを以降 <\$O-L3>, <\$O-stream>

# プログラムのありか on FX10

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



# 実行例

```
% cd <$O-L3>
% ls
    run    src    src0    reorder0

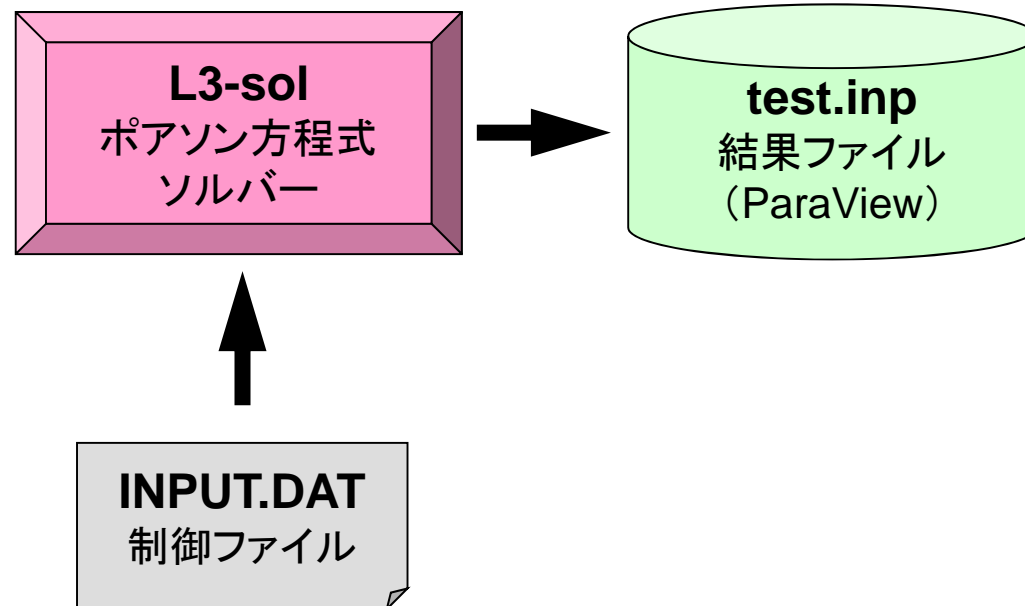
% cd src
% make
% cd ../run
% ls L3-sol
    L3-sol

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

% pjsub go1.sh
```

# プログラムの実行

## プログラム, 必要なファイル等



# 制御データ(INPUT.DAT)

```

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

変数名	型	内 容
NX, NY, NZ	整数	各方向の要素数
DX, DY, DZ	倍精度実数	各要素の3辺の長さ ( $\Delta X$ , $\Delta Y$ , $\Delta Z$ )
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 $\geq 2$ : MC法 (multicolor) , 色数 $= 0$ : CM法 (Cuthill-Mckee) $= -1$ : RCM法 (Reverse Cuthill-Mckee) $\leq -2$ : CM-RCM法

# go1.sh

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=05:00"
#PJM -L "rscgrp=small" または "rscgrp=school"
#PJM -j
#PJM -o "test.lst"

export OMP_NUM_THREADS=16      PEsmptTOTと一致させる
./L3-sol
```

- L2-solへのOpenMPの実装
- 実行例
- 最適化＋演習

# L2-solにOpenMPを適用

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

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

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

- スレッド数をプログラムで制御できるようにしてみよう

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

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

- スレッド数をプログラムで制御できるようにしてみよう



# ICCG法の並列化: OpenMP

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

# プログラムの構成(1/2)

```
program MAIN

use STRUCT
use PCG
use solver_ICCG_mc

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

call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN

PHI= 0.d0

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

# プログラムの構成(2/2)

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

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

結果(PHI)をもとの番号  
付けにもどす

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

```
call OUTUCD
```

```
stop  
end
```

# プログラムの構成

```
program MAIN

  use STRUCT
  use PCG
  use solver_ICCG_mc

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

  call INPUT
  call POINTER_INIT
  call BOUNDARY_CELL
  call CELL_METRICS
  call POI_GEN

  PHI= 0.d0

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

# module STRUCT

```

module STRUCT

  use omp_lib
  include 'precision.inc'

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

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

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

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

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

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

  integer (kind=kint) :: PEsmptTOT

end module STRUCT

```

ICELTOT : 要素数

ICELTOTp : = ICELTOT

N : 節点数

NX, NY, NZ : x, y, z方向要素数

NXP1, NYP1, NZP1 :

& x, y, z方向節点数

& IBNODTOT : NXP1 × NYP1

& XYZ(ICELTOT, 3) : 要素座標 (後述)

NEIBcell(ICELTOT, 6) :

& 隣接要素 (後述)

境界条件関連 : Z=Zmax

PEsmptTOT : スレッド数

# module PCG(これまでとの相違点)

```

module PCG

integer, parameter :: N2= 256
integer :: NUmex, NLmax, NCOLORtot, NCOLORk, NU, NL
integer :: NPL, NPU
integer :: METHOD, ORDER METHOD

real(kind=8) :: EPSICCG

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

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

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

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

```

NCOLORtot

COLORindex(0:NCOLORtot)

色数

各色に含まれる要素数のインデックス

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

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

**SMPindexG(0:PEsmptOT)**

OLDtoNEW, NEWtoOLD

Coloring前後の要素番号対照表

# 変数表(1/2)

配列・変数名	型	内 容
<b>D(N)</b>	<b>R</b>	対角成分, (N:全メッシュ数)
<b>BFORCE(N)</b>	<b>R</b>	右辺ベクトル
<b>PHI(N)</b>	<b>R</b>	未知数ベクトル
<b>indexL(0:N)</b>	<b>I</b>	各行の非零下三角成分数(CRS)
<b>indexU(0:N)</b>	<b>I</b>	各行の非零上三角成分数(CRS)
<b>NPL</b>	<b>I</b>	非零下三角成分総数(CRS)
<b>NPU</b>	<b>I</b>	非零上三角成分総数(CRS)
<b>itemL(NPL)</b>	<b>I</b>	非零下三角成分(列番号)(CRS)
<b>itemU(NPU)</b>	<b>I</b>	非零上三角成分(列番号)(CRS)
<b>AL(NPL)</b>	<b>R</b>	非零下三角成分(係数)(CRS)
<b>AU(NPL)</b>	<b>R</b>	非零上三角成分(係数)(CRS)
<b>NL,NU</b>	<b>I</b>	各行の非零上下三角成分の最大数 (ここでは6)
<b>INL(N)</b>	<b>I</b>	各行の非零下三角成分数
<b>INU(N)</b>	<b>I</b>	各行の非零上三角成分数
<b>IAL(NL,N)</b>	<b>I</b>	各行の非零下三角成分に対応する列番号
<b>IAU(NU,N)</b>	<b>I</b>	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
<b>NCOLORtot</b>	<b>I</b>	入力時にはOrdering手法 ( $\geq 2$ : MC, $=0$ : CM, $=-1$ : RCM, $-2 \leq$ : CMRCM) , 最終的には色数, レベル数が入る
<b>COLORindex(0:NCOLORtot)</b>	<b>I</b>	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。
<b>NEWtoOLD(N)</b>	<b>I</b>	新番号⇒旧番号への参照配列
<b>OLDtoNEW(N)</b>	<b>I</b>	旧番号⇒新番号への参照配列
<b>PEsmpTOT</b>	<b>I</b>	スレッド数
<b>SMPindex (0:NCOLORtot*PEsmpTOT)</b>	<b>I</b>	スレッド用補助配列 (データ依存性がある ループに使用)
<b>SMPindexG(0:PEsmpTOT)</b>	<b>I</b>	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, PEsmptTOT,
  &      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
  - 「 $\leq -2$ 」の場合はCM-RCM

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

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

# cell\_metrics

```

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

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

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

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

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

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

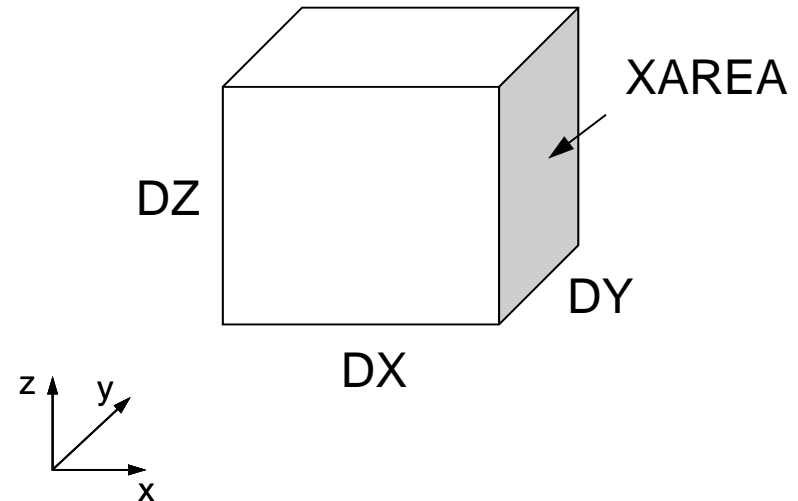
      V0= DX * DY * DZ
      RV0= 1. d0/V0

      VOLCEL= V0
      RVC    = RV0

      return
      end

```

## 計算に必要な諸パラメータ



# プログラムの構成

```
program MAIN

  use STRUCT
  use PCG
  use solver_ICCG_mc

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

  call INPUT
  call POINTER_INIT
  call BOUNDARY_CELL
  call CELL_METRICS
  call POI_GEN

  PHI= 0.d0

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

# poi\_gen (1/9)

```
subroutine POI_GEN

  use STRUCT
  use PCG

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

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

  NU= 6
  NL= 6

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

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

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

# 配列の宣言

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

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

! C===

A central cube represents a cell. Six arrows point away from its faces, representing its neighbors:

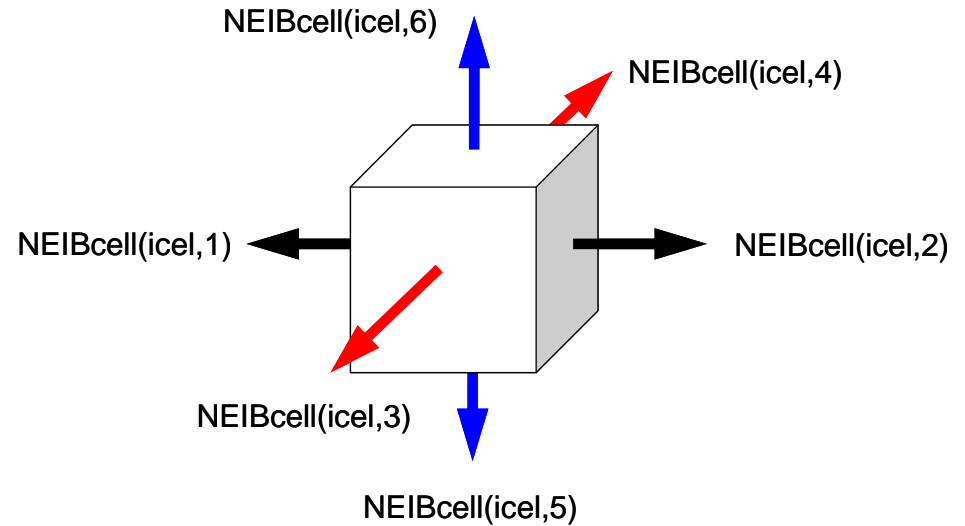
- Top face: Blue arrow pointing up, labeled  $\text{NEIBcell}(\text{icel},6)$ .
- Bottom face: Blue arrow pointing down, labeled  $\text{NEIBcell}(\text{icel},5)$ .
- Left face: Black arrow pointing left, labeled  $\text{NEIBcell}(\text{icel},1)$ .
- Right face: Black arrow pointing right, labeled  $\text{NEIBcell}(\text{icel},2)$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

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

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

! C===

# poi\_gen (2/9)



## 上三角成分

```
NEIBcell(icel,2)= icel + 1
```

NEIBcell(icel,4)= icel + NX

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



# poi\_gen (3/9)

```

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

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

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

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

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

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

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

```

並べ替えの実施 :

NCOLORtot > 1 : Multicolor

NCOLORtot = 0 : CM

NCOLORtot = -1 : RCM

NCOLORtot < -1 : CM-RCM

# poi\_gen (4/9)

```

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

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

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

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

```

!C==

各色内の要素数：

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

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

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

前処理で使用

```

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

```

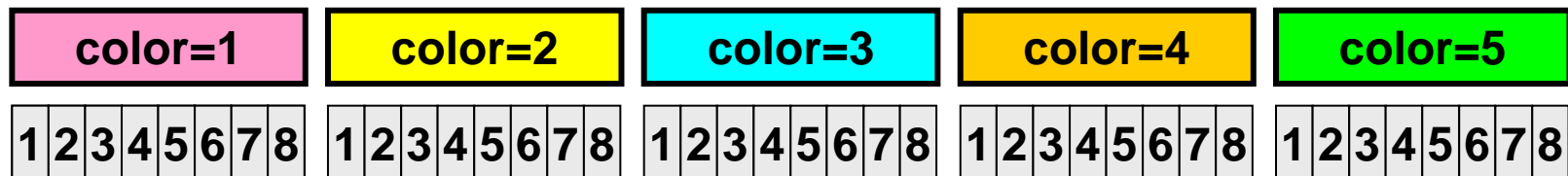
# SMPindex: 前処理向け

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

Initial Vector



Coloring  
(5 colors)  
+Ordering



- 5色, 8スレッドの例
- 同じ「色」に属する要素は独立⇒並列計算可能
- 色の順番に並び替え

# poi\_gen(4/9)

```

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

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

```

```

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

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

```

!C==

```

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

```

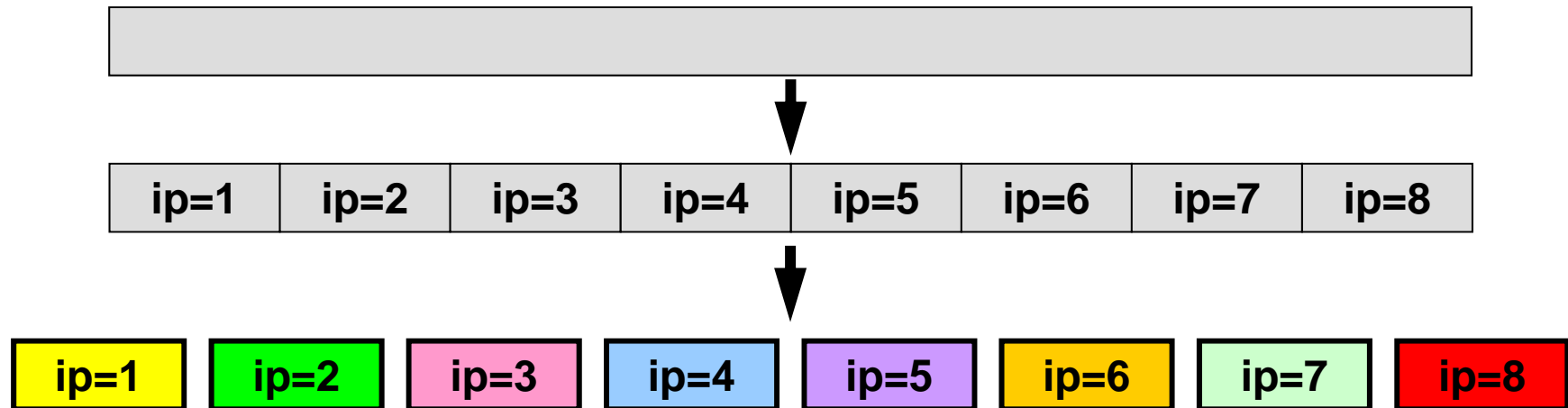
全要素数を「PEsmpTOT」で割って  
「SMPindexG」に割り当てる。

内積，行列ベクトル積，DAXPYで使用

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

# SMPindexG

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



各スレッドで独立に計算: 行列ベクトル積, 内積, DAXPY等

```

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

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

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

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

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

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

```

## 配列の宣言

# poi\_gen(5/9)

これ以降は新しい  
番号付けを使用

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

```

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

```

```

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

```

```

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

```

**icel: 新しい番号**  
**ic0: 古い番号**

```

  VOL0= VOLCEL (ic0)

```

```

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

```

```

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

```

## poi\_gen (6/9)

### 新しい番号付けを使用

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

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

```

```

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

```

**icel: 新しい番号**  
**ic0: 古い番号**

```

  VOL0= VOLCEL (ic0)

```

```

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

```

```

  if (icN5.lt.icel) then
    do j= 1, INL(icel)
      if (IAL(j, icel).eq.icN5) then
        itemL(j+indexL(icel-1))= icN5
        AL(j+indexL(icel-1))= coef
        exit
      endif
    enddo
  else
    do j= 1, INU(icel)
      if (IAU(j, icel).eq.icN5) then
        itemU(j+indexU(icel-1))= icN5
        AU(j+indexU(icel-1))= coef
        exit
      endif
    enddo
  endif
endif
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, PEsmptTOT
do icel= SMPindexG(ip-1)+1, SMPindexG(ip)
ic0 = NEWtoOLD(icel)

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

VOL0= VOLCEL (ic0)

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

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

```

## poi\_gen (6/9)

### 新しい番号付けを使用

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

```

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

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

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

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

VOL0= VOLCEL (ic0)

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

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

```

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

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

icN5がicelより小さければ下三角成分

## poi\_gen (6/9)

### 新しい番号付けを使用

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

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

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

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

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

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

```

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

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

VOL0= VOLCEL (ic0)

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

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

```

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

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

icN5がicelより大き  
ければ上三角成分

## poi\_gen (6/9)

### 新しい番号付けを使用

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

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

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

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

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

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

```

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

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

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

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

```

## poi\_gen(7/9)

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

```

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

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

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

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

```

## poi\_gen(8/9)

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

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

...

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

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

もとの座標に従って  
BFORCE計算

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

ii,jj,kk,VOL0はprivate

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

## poi\_gen (9/9)

係数の計算(境界面以外)

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

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

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

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

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

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

# プログラムの構成

```
program MAIN

  use STRUCT
  use PCG
  use solver_ICCG_mc

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

  call INPUT
  call POINTER_INIT
  call BOUNDARY_CELL
  call CELL_METRICS
  call POI_GEN

  PHI= 0.d0

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

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



# solve\_ICCG\_mc(1/6)

```

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

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

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

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

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

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

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

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

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

```

# solve\_ICCG\_mc(2/6)

```

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

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

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

```

不完全修正  
コレスキー分解

# 不完全修正コレスキー分解

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

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

```

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

```

# 不完全修正コレスキー分解：並列版

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

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

```

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

```

privateに注意。

# solve\_ICCG\_mc(3/6)

```

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

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

```

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

```

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

```

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

for  $i = 1, 2, \dots$

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

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

if  $i=1$

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

else

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

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

endif

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

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

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

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

  check convergence  $|\mathbf{r}|$

end

# 内積: SMPindexG使用, reduction

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



```

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

```

# solve\_ICCG\_mc (4/6)

```

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

```

```

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

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

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

```

ここでは「SMPindex」を使う

## solve\_ICCG\_mc (4/6)

```

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

```

```

      ITR= N
      do L= 1, ITR
      !C
      !C +-----+
      !C | {z}= [Minv] {r} |
      !C +-----+
      !C==
      !$omp parallel do private(ip, i)
      do ip= 1, PEsmptOT
      do i = SMPindexG(ip-1)+1, SMPindexG(ip)
      W(i, Z)= W(i, R)
      enddo
      enddo
      !$omp end parallel do
      do ic= 1, NCOLORTot
      !$omp parallel do private(ip, ip1, i, WVAL, k)
      do ip= 1, PEsmptOT
      ip1= (ic-1)*PEsmptOT + ip
      do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      WVAL= W(i, Z)
      do k= indexL(i-1)+1, indexL(i)
      WVAL= WVAL - AL(k) * W(itemL(k), Z)
      enddo
      W(i, Z)= WVAL * W(i, DD)
      enddo
      enddo
      !$omp end parallel do
      enddo
      do ic= NCOLORTot, 1, -1
      !$omp parallel do private(ip, ip1, i, SW, k)
      do ip= 1, PEsmptOT
      ip1= (ic-1)*PEsmptOT + ip
      do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      SW = 0.0d0
      do k= indexU(i-1)+1, indexU(i)
      SW= SW + AU(k) * W(itemU(k), Z)
      enddo
      W(i, Z)= W(i, Z) - W(i, DD) * SW
      enddo
      enddo
      !$omp end parallel do
      enddo
      !C==

```

ここでは「SMPindex」を使う

# solve\_ICCG\_mc (4/6)

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

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

前進代入  
Forward Substitution

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

後退代入  
Backward Substitution

# 前進代入: SMPindex使用

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

```

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

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

```

# solve\_ICCG\_mc (5/6)

```

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

```

```

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

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

```

# solve\_ICCG\_mc (5/6)

```

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

```

```

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

      ALPHA= RHO / C1
!C==

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

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

```

# solve\_ICCG\_mc (6/6)

```

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

```

```

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

      ALPHA= RHO / C1
!C==

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

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

```

# solve\_ICCG\_mc (6/6)

```

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

```



```

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

      ALPHA= RHO / C1
!C===

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

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

```

# solve\_ICCG\_mc (6/6)

```

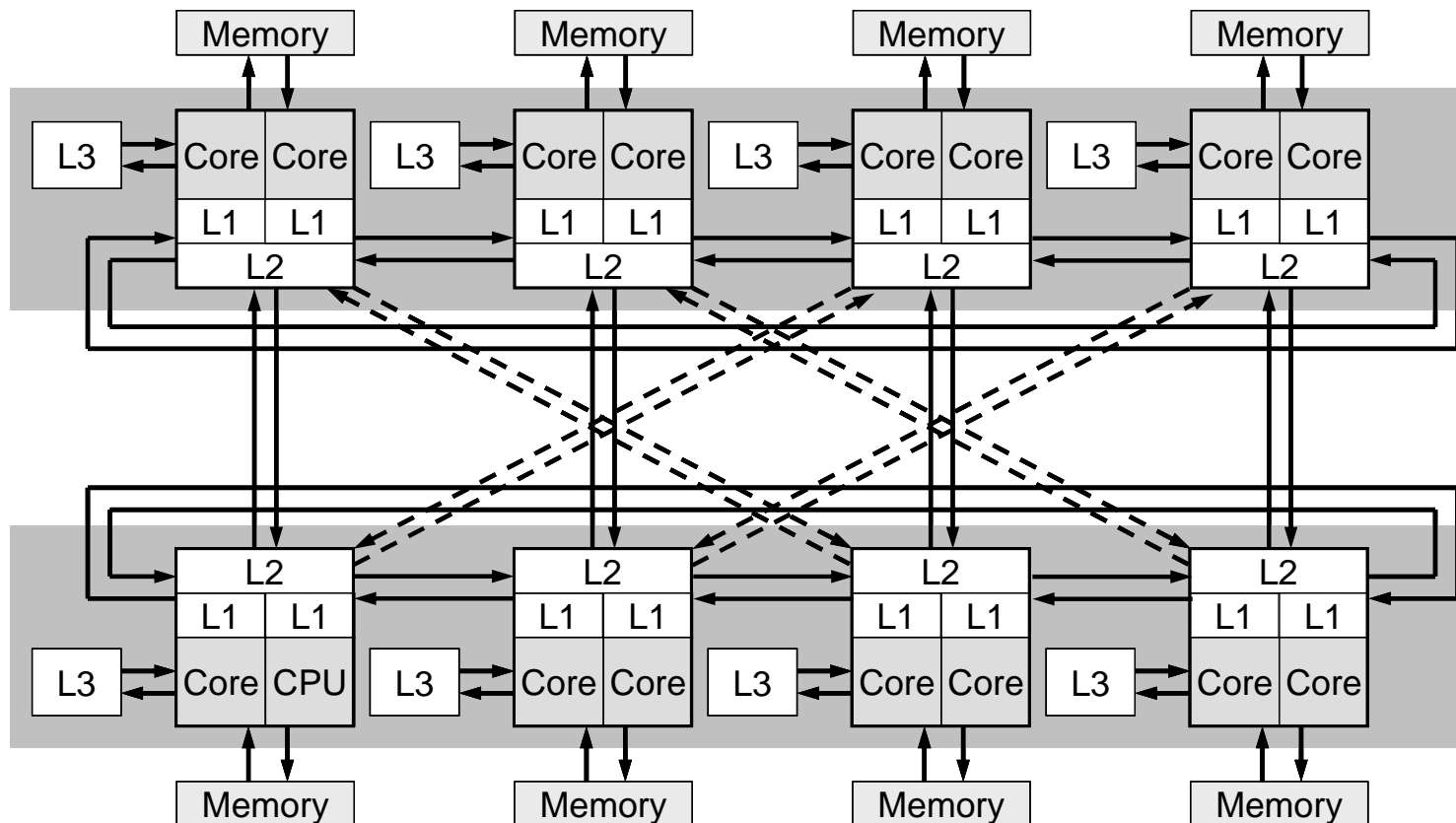
Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
  solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
   $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
  if  $i=1$ 
     $p^{(1)} = z^{(0)}$ 
  else
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  endif
   $q^{(i)}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

```

- L2-solへのOpenMPの実装
- 実行例
- 最適化＋演習

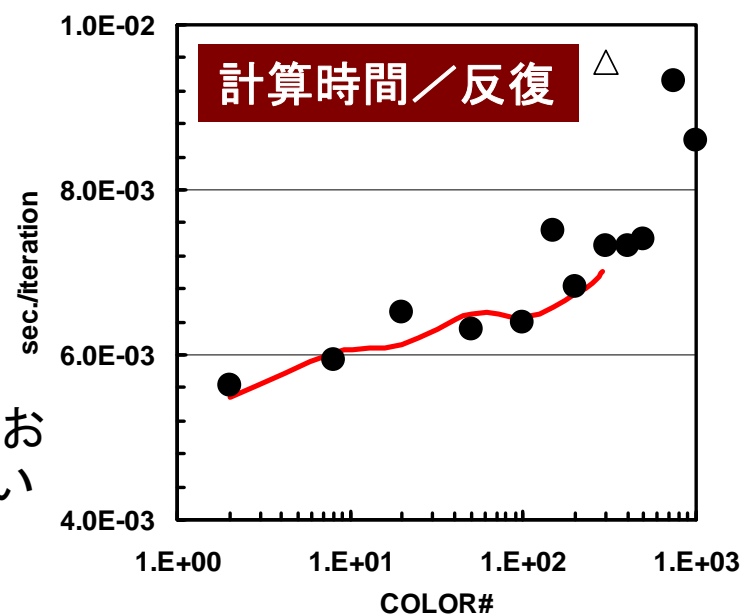
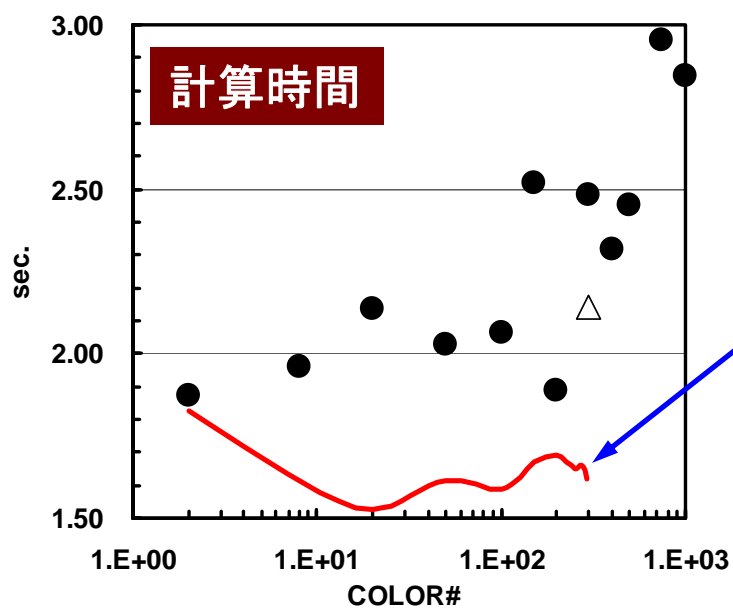
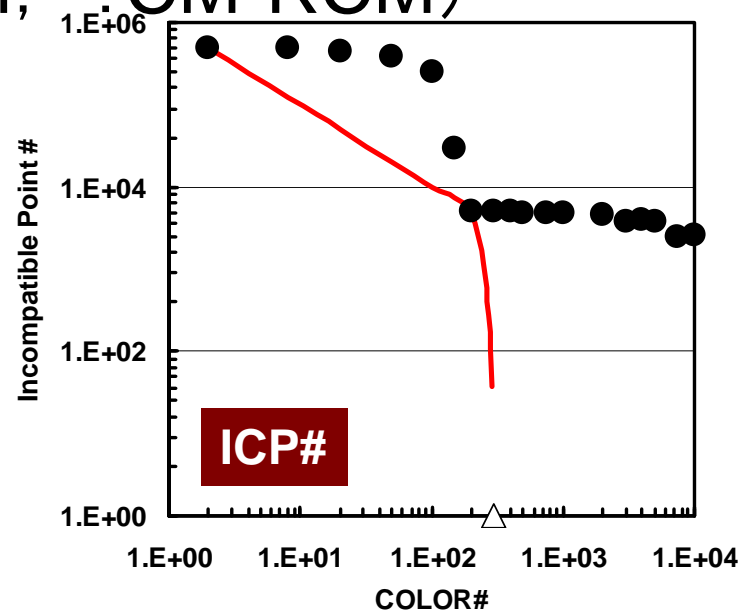
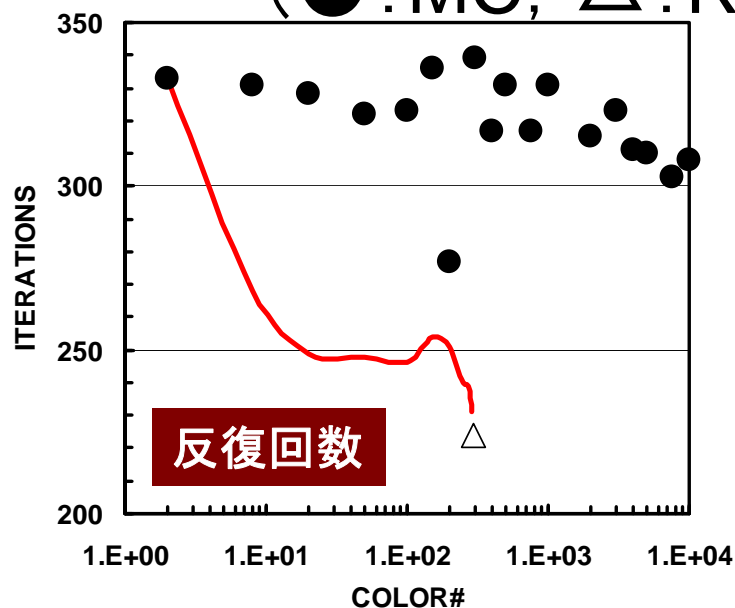
# 計算結果

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



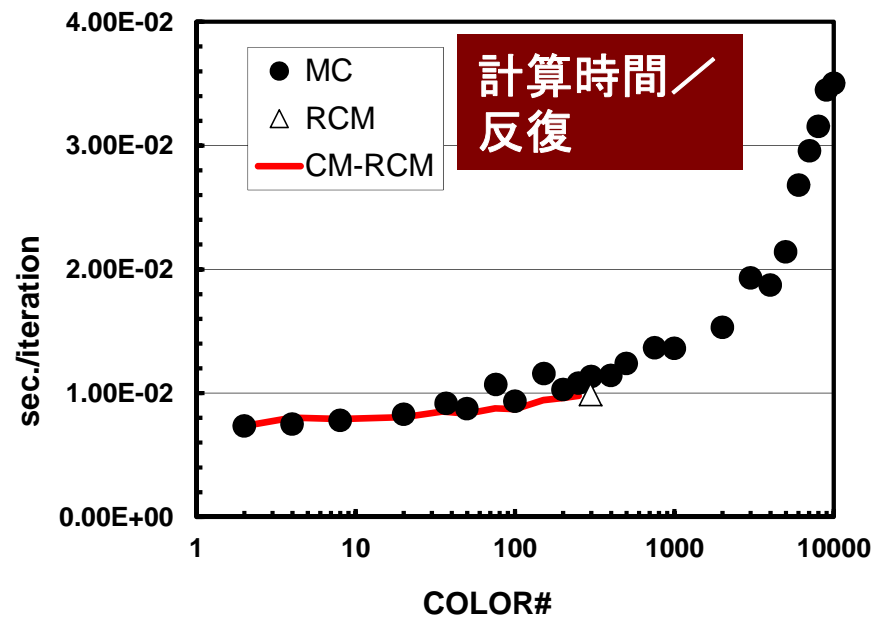
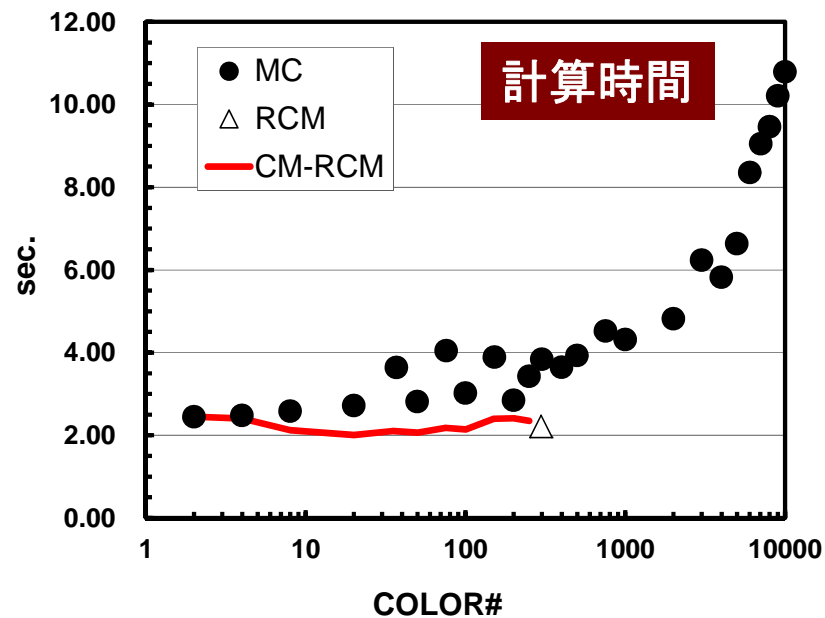
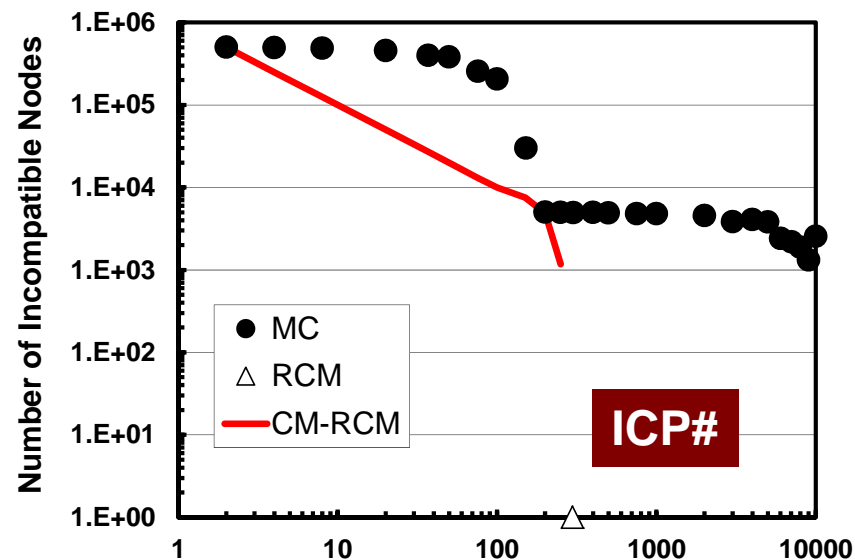
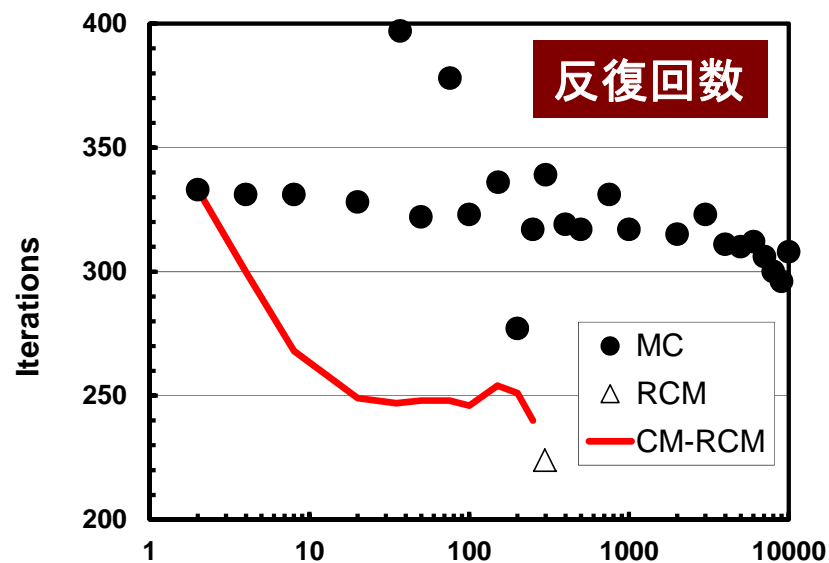
# SR11000, 16コアにおける結果, $100^3$

(●: MC, △: RCM, - : CM-RCM)



# FX10, 16コアにおける結果, $100^3$

(●:MC, △:RCM, -:CM-RCM)



- L2-solへのOpenMPの実装
- 実行例
- 最適化＋演習

- マルチコア版コードの実行
- 更なる最適化
- STREAM
- プロファイラ, コンパイルリスト分析等

# コンパイル・実行

```
>$ cd <$O-L3>/src  
>$ make  
>$ ls ../run/L3-sol
```

L3-sol

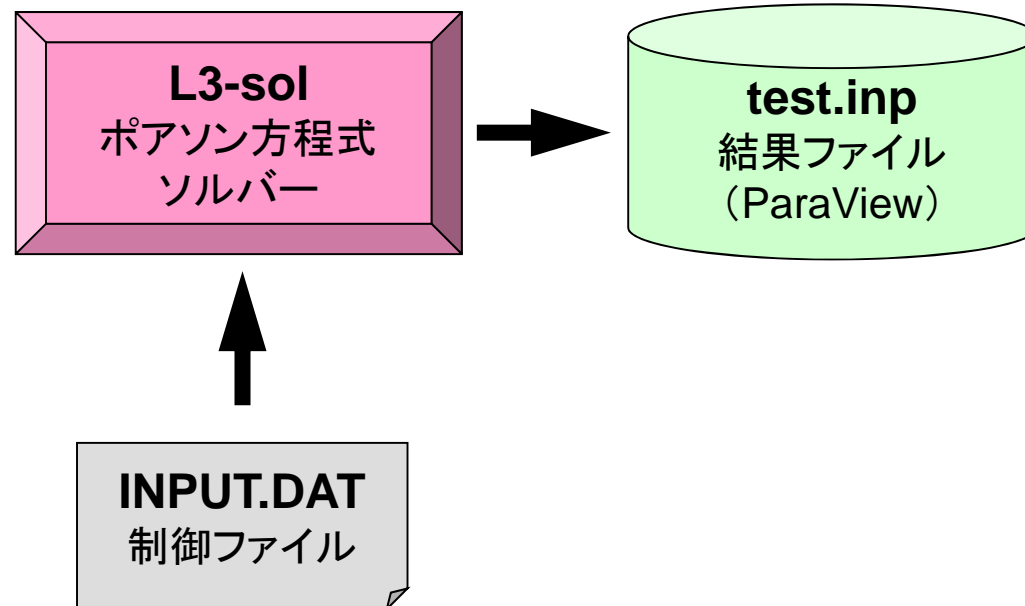
```
>$ cd ../run
```

```
>$ pjsub gol.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
-10                 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 ( $\Delta X$ , $\Delta Y$ , $\Delta Z$ )
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 $\geq 2$ : MC法 (multicolor) , 色数 $= 0$ : CM法 (Cuthill-Mckee) $= -1$ : RCM法 (Reverse Cuthill-Mckee) $\leq -2$ : CM-RCM法

# go1.sh

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:10:00"
#PJM -L "rscgrp=small"
#PJM -j
#PJM -o "test.lst"
```

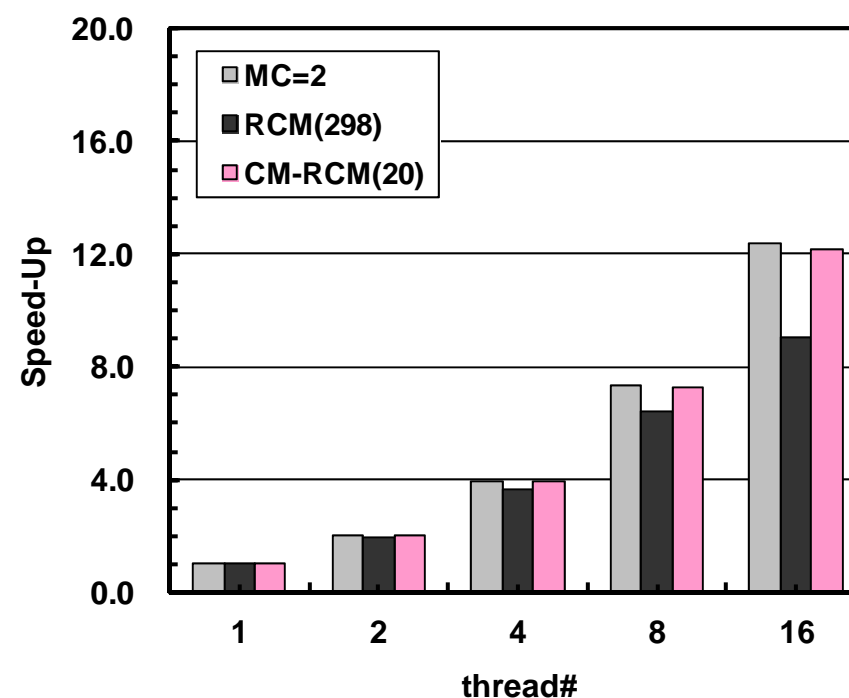
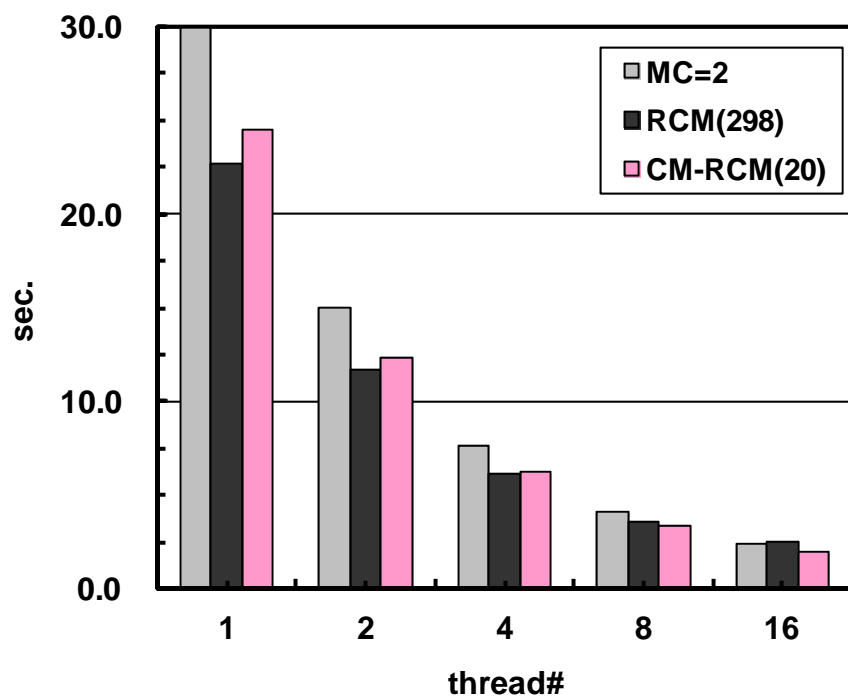
標準出力ファイル名

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

スレッド数, 通常 =PEsmpTOT

# 計算結果(FX10@東大): $10^6$ 要素

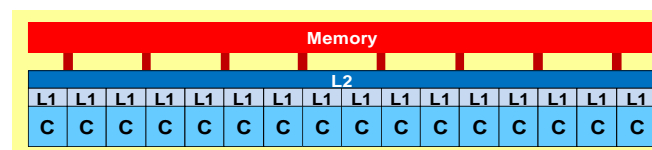
反復回数: MC(2色): 333回, RCM(298レベル): 224回  
CM-RCM( $N_c=20$ ): 249回



16 threads

MC(2): 2.42 sec.

CM-RCM(20): 2.01 sec.



# 実習(1)

- 色々なケースでやってみよう
  - 問題サイズ
  - スレッド数
  - 色数, 色分け法 (MC, RCM, CM-RCM)

- マルチコア版コードの実行
- 更なる最適化
  - その1: **OpenMP Statement**
  - その2: Sequential Reordering
  - その3: ELL
- STREAM
- プロファイラ, コンパイルリスト分析等

# 前進代入: 現状の並列化(Fortran)

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

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

# 前進代入:現状の並列化(C)

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

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



# 前進代入: Overhead削減 (Fortran)

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

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

# 前進代入: Overhead削減(C)

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

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

# プログラム類

```
% cd <$0-L3>  
% ls  
    run  reorder0  src  src0
```

```
% cd src0
```

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

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

```
% pjsub go0.sh
```

# 計算結果 : L3-sol0が速い

## $N=128^3$

	L3-sol	L3-sol0
NCOLORtot= -20 CM-RCM (20) 318 Iterations	5.69 sec.	5.44 sec.
NCOLORtot= -1 RCM (382 levels) 287 Iterations	6.54 sec.	6.37 sec.

- マルチコア版コードの実行
- 更なる最適化
  - その1: OpenMP Statement
  - **その2: Sequential Reordering**
  - **その3: ELL**
- STREAM
- プロファイラ, コンパイルリスト分析等

# 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

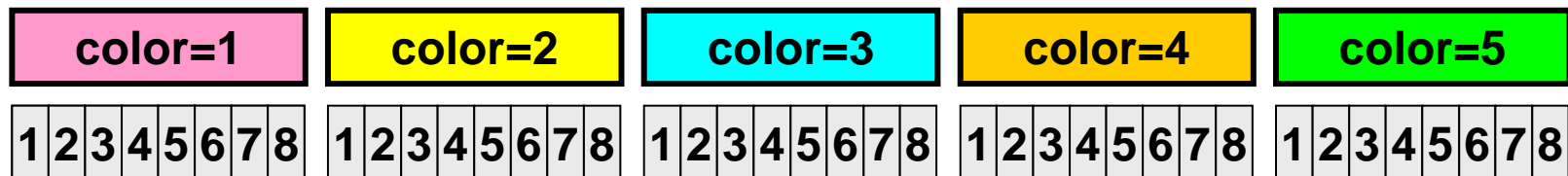
# SMPindex: 前処理向け

```
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
```

Initial Vector



Coloring  
(5 colors)  
+Ordering



- 5色, 8スレッドの例
- 同じ「色」に属する要素は独立⇒並列計算可能
- 色の順番に並び替え

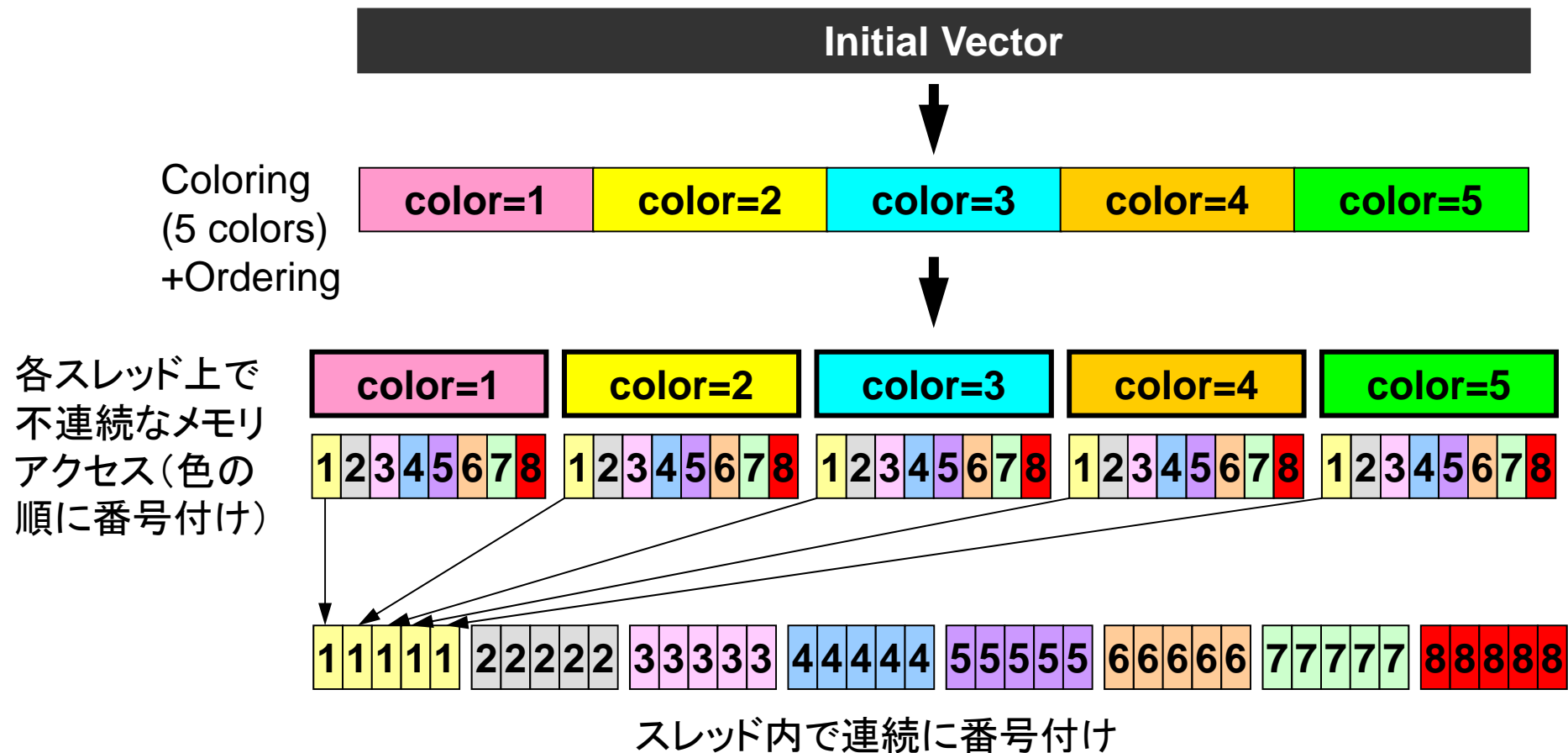
# データ再配置: Sequential Reordering

- 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



# 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

スレッド上のデータ連続性: キャッシュ有効利用, プリフェッチが効きやすくなる

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

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2色

■ #0 thread, ■ #1, ■ #2, ■ #3

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

CM-RCM(2)

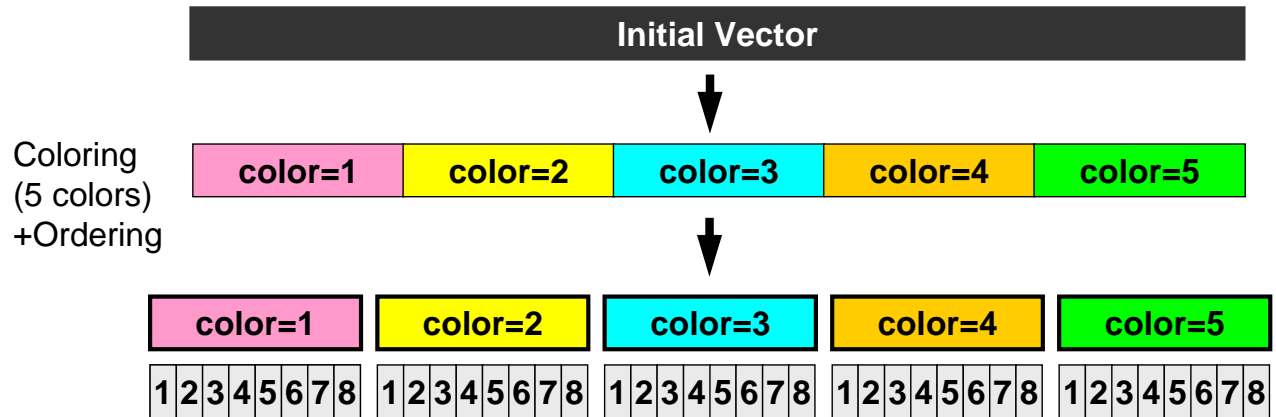


29	18	15	5	11	2	9	1
33	30	19	16	6	12	3	10
45	34	31	20	25	7	13	4
40	46	35	32	21	26	8	14
59	49	47	36	41	22	27	17
53	60	50	48	37	42	23	28
63	54	61	51	57	38	43	24
56	64	55	62	52	58	39	44

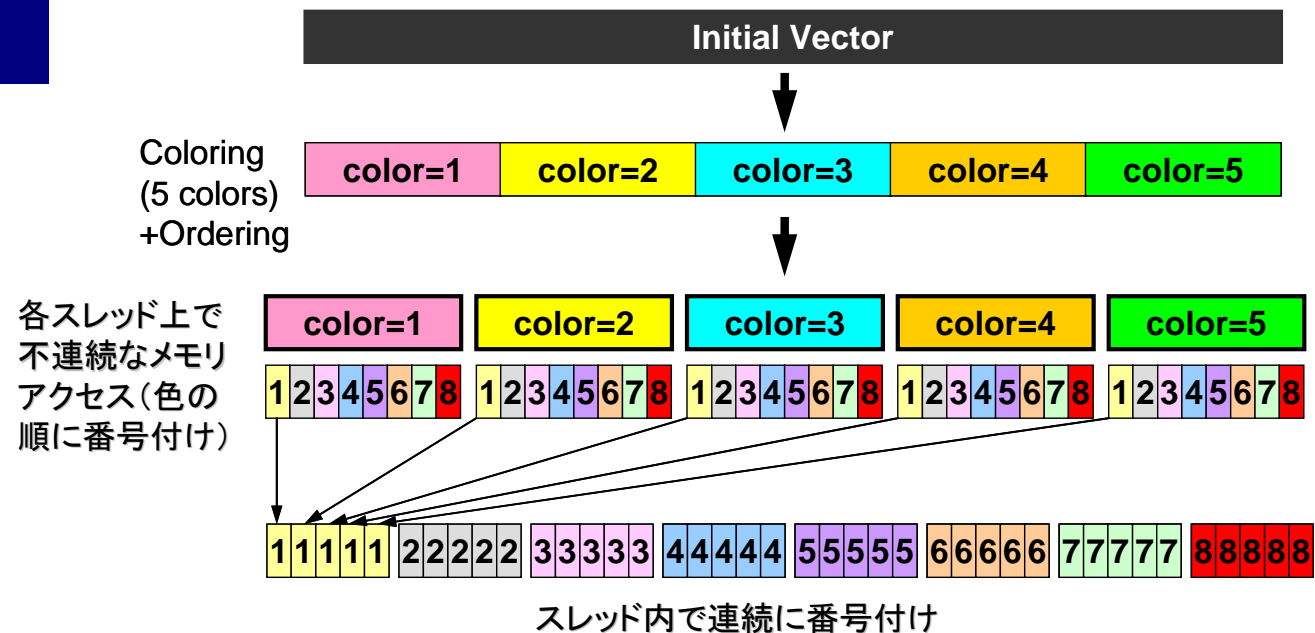
Sequential Reordering, 4-threads

# データ再配置: Sequential Reordering

Coalesced  
GPUはこちらが  
お勧め



Sequential



# プログラムのありか

- 所在
  - `<$L3>/reorder0`
- コンパイル, 実行方法
  - 本体
    - `cd <$L3>/reorder0`
    - `make`
    - `<$L3>/reorder0/L3-rsol0`(実行形式)
  - コントロールデータ
    - `<$L3>/run/INPUT.DAT`
  - 実行用シェル
    - `<$L3>/run/gor.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              PEsmptTOT
100             NCOLORTot
0               NFLAG
0               METHOD

```

変数名	型	内 容
PEsmptTOT	整数	データ分割数
NCOLORtot	整数	Ordering手法と色数 $\geq 2$ : MC法 (multicolor) , 色数 $= 0$ : CM法 (Cuthill-Mckee) $= -1$ : RCM法 (Reverse Cuthill-Mckee) $\leq -2$ : CM-RCM法
NFLAG	整数	0 : First-Touch無し, 1 : あり 今回は無関係
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)

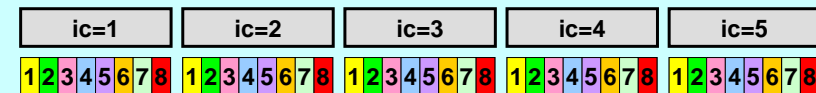
# 再配置: Sequential Reordering

```

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

```

## SMPindex



## SMPindex\_new



```

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

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

```



# 行列ベクトル積の計算法 (METHOD=0)

```

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

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

**Original**

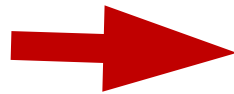
```
!$omp parallel private(ip, ip1, i, WVAL, k)
  do ic= 1, NCOLORTot
!$omp do
  do ip= 1, PEsmptOT
    ip1= (ip-1)*NCOLORTot + ic
    do i= SMPindex(ip1-1)+1, SMPindex(ip1)
      WVAL= W(i, Z)
      do k= indexL(i-1)+1, indexL(i)
        WVAL= WVAL - AL(k) * W(itemL(k), Z)
      enddo
      W(i, Z)=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

**New**

# 行列格納方法

**ELL: ループ長固定, プリフェッチ効きやすい**

$$\begin{bmatrix} 1 & 3 & 0 & 0 & 0 \\ 1 & 2 & 5 & 0 & 0 \\ 4 & 1 & 3 & 0 & 0 \\ 0 & 3 & 7 & 4 & 0 \\ 1 & 0 & 0 & 0 & 5 \end{bmatrix}$$



1	3	
1	2	5
4	1	3
3	7	4
1	5	

(a) CRS

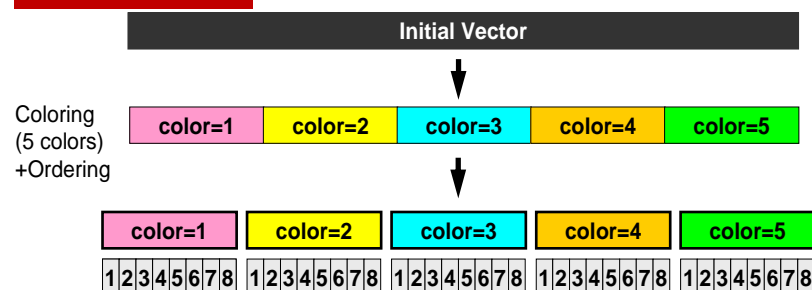
1	3	0
1	2	5
4	1	3
3	7	4
1	5	0

(b) ELL

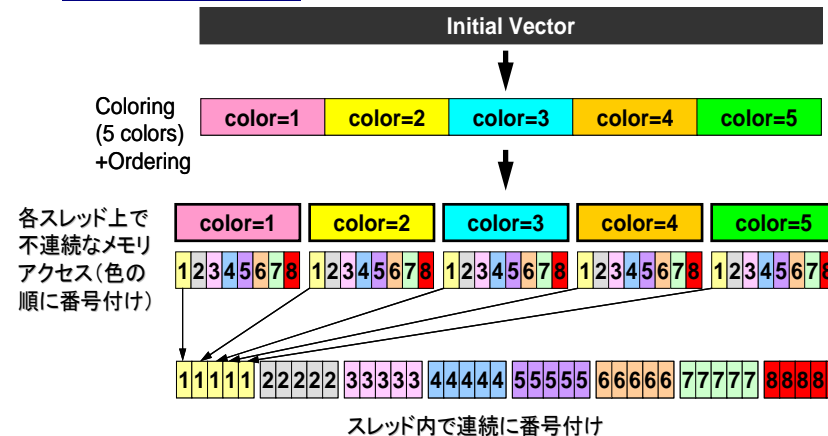
# 実施ケース

		並列化向け色付け手法	番号付け	First Touch Data Placement	係数行列格納形式
src0	Case-1	CM-RCM	Coalesced (図 4 (a))	無し	CRS
reorder0	Case-2		Sequential (図 4 (b))	有り	
ELL	Case-3				ELL

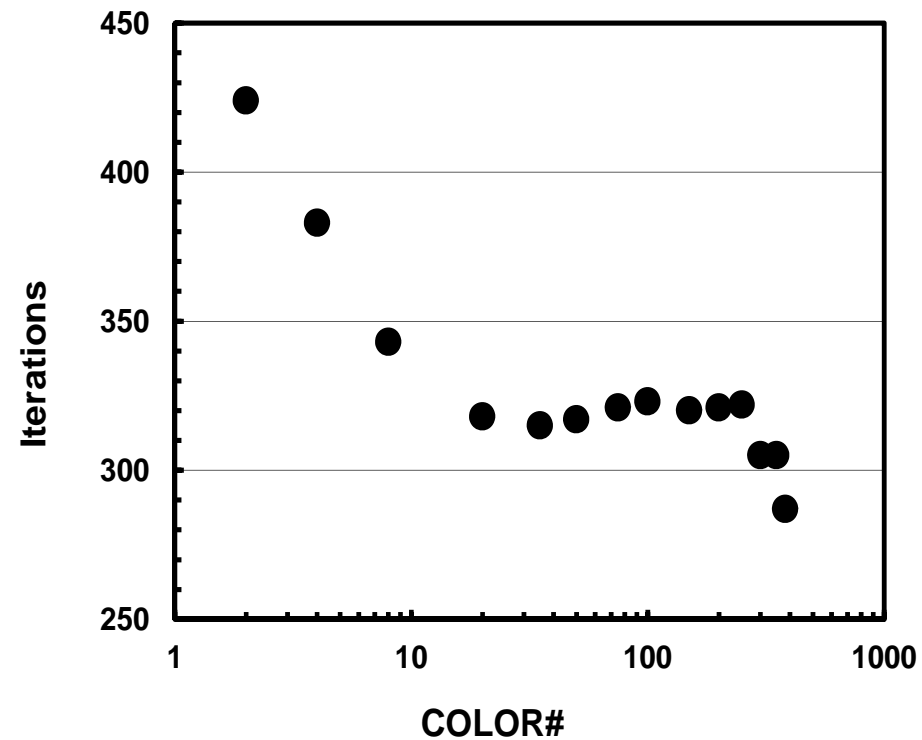
## Coalesced



## Sequential

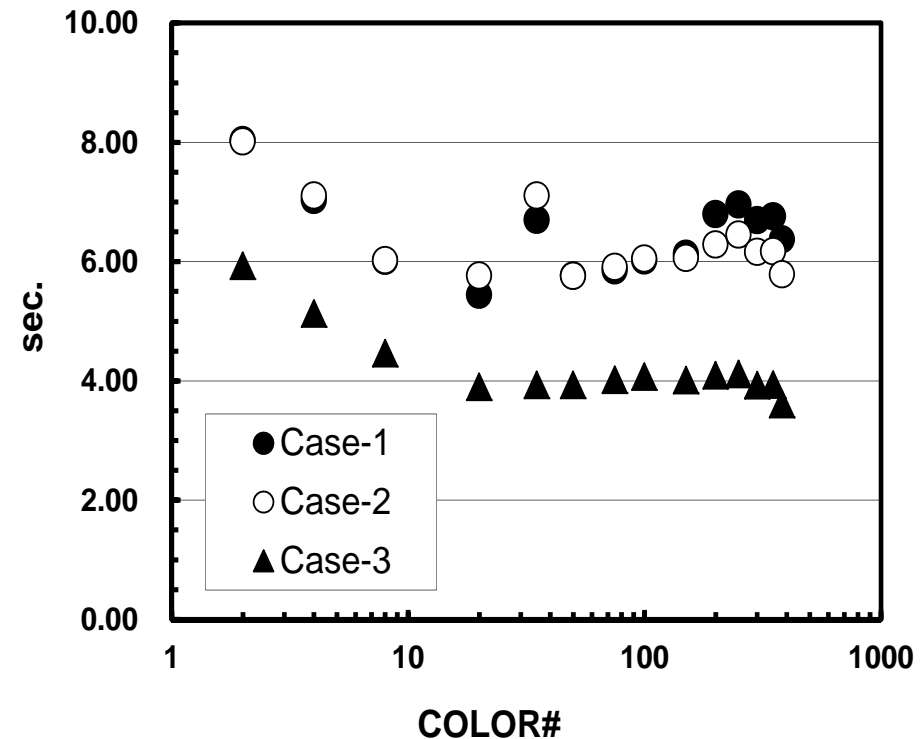


# 計算結果：色数～反復回数 (CM-RCM)



# 計算結果：色数～計算時間：FX10

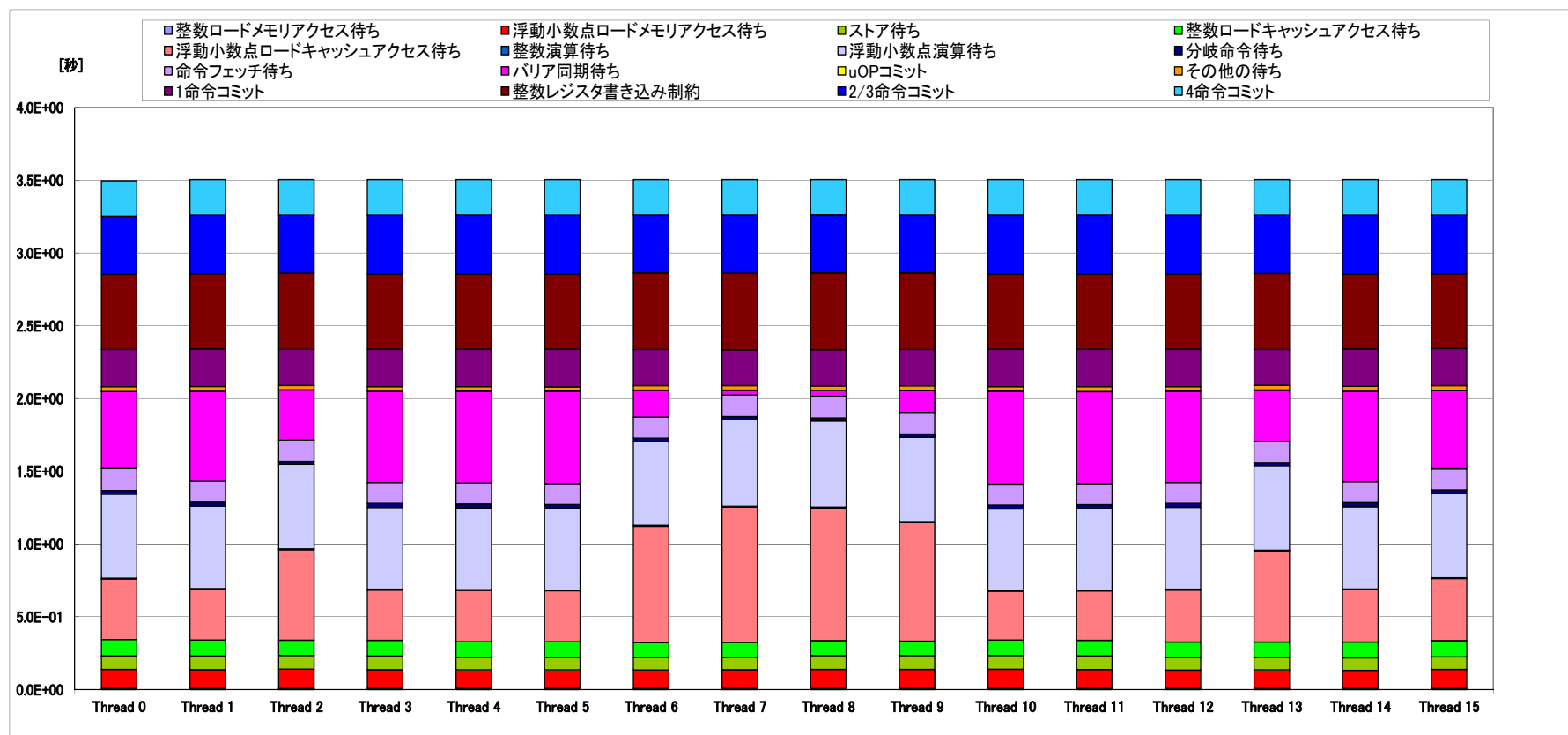
- CASE-1(src0)⇒  
CASE-2(reorder0)
  - 色数が大きくなると多少効果あり
  - 色数が多くなるほど、色当たり、スレッド当たりの計算量は減少
    - CASE-2は色が変わってもスレッド上のデータが連続
  - First Touchの効果はナシ
- ELLの効果は大



Case-1: src0  
Case-2: reorder0  
Case-3: reorder0 + ELL

# Fujitsu FX10: CASE-1, CM-RCM(2)

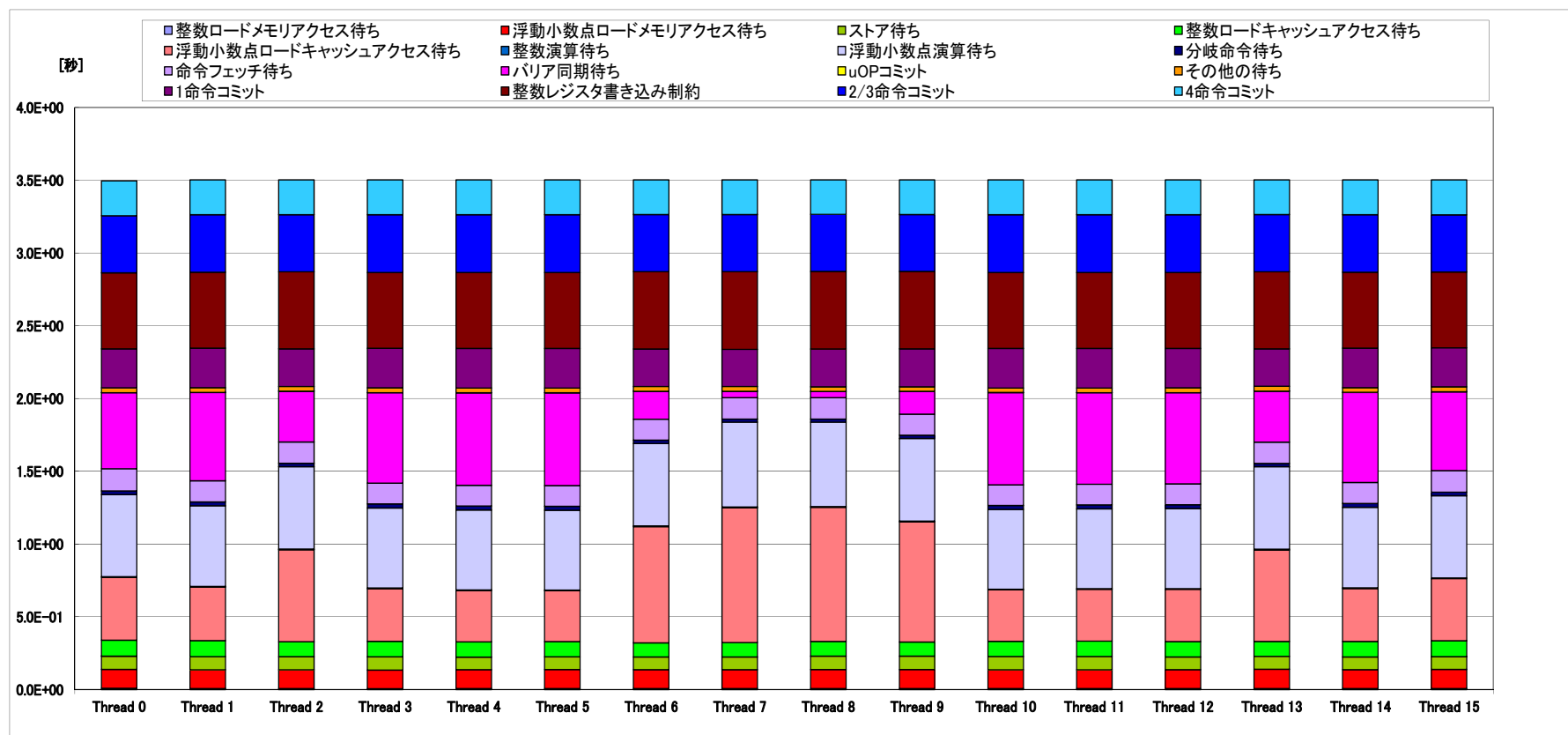
L1デマンドミス: 25.6%, Mem. Throughput: 41.8GB/sec.  
前処理部分(前進後退代入)



**src0: CRS, Coalesced**

# Fujitsu FX10: CASE-2, CM-RCM(2)

## 25.6%, 41.8GB/sec.

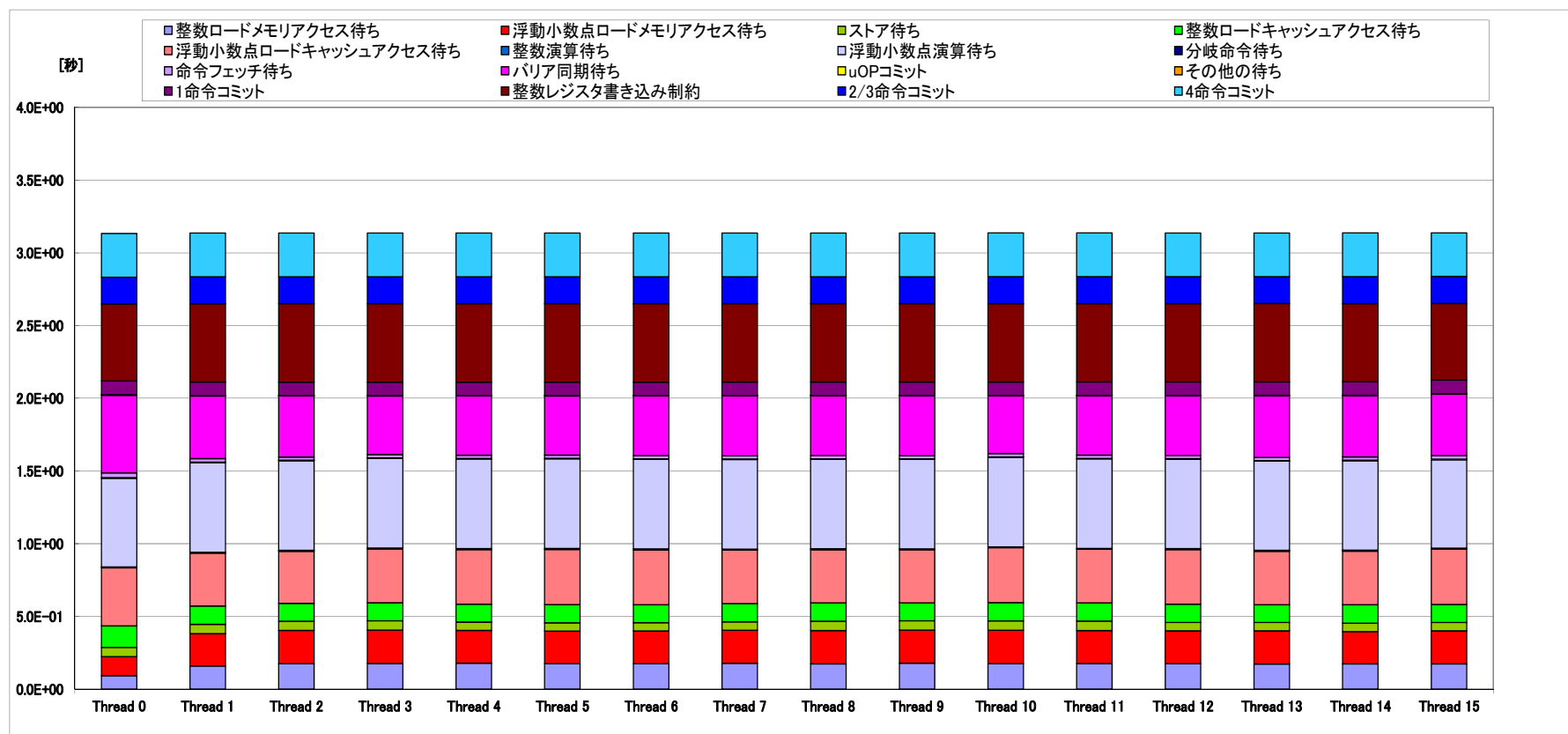


**reorder0: CRS, Sequential**



# Fujitsu FX10: CASE-1, CM-RCM(382)

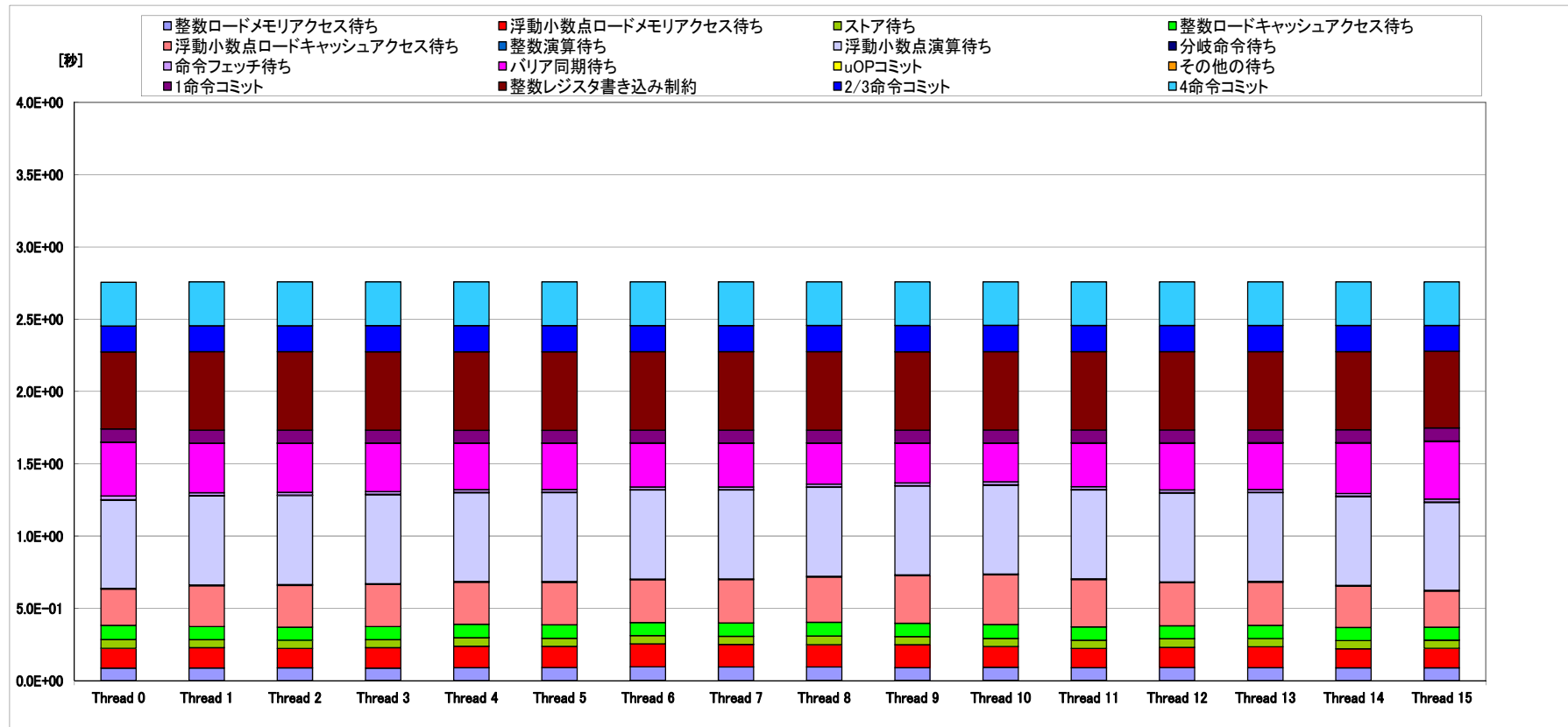
## 37.7%, 28.7GB/sec.



**src0: CRS, Coalesced**

# Fujitsu FX10: CASE-2, CM-RCM(382)

## 29.3%, 32.6GB/sec.



**reorder0: CRS, Sequential**

# 計算結果のまとめ: Fujitsu FX10

## 詳細プロファイラ

上段: L1デマンドミス率

下段: メモリースループット

	src0 CASE-1 CRS+ Coalesced	reorder0 CASE-2 CRS+ Sequential	CASE-3 ELL+ Sequential
CM-RCM(2)	25.5 %	25.6 %	5.42 %
	41.8 GB/sec.	41.8 GB/sec.	64.0 GB/sec.
CM-RCM(382)	37.7 %	29.3 %	16.5 %
	28.7 GB/sec.	32.6 GB/sec.	52.2 GB/sec.

# 計算結果のまとめ: Fujitsu FX10

## 詳細プロファイル

### 上段: CM-RCM(20), 下段: CM-RCM(382)

	Instructions	SIMD (%)	Memory Access Throughput (%)
Case-2 CRS	$1.83 \times 10^{11}$	7.17	50.2
	$1.83 \times 10^{11}$	6.90	44.5
Case-3 ELL	$6.71 \times 10^{10}$	16.3	69.8
	$5.96 \times 10^{10}$	16.2	67.0

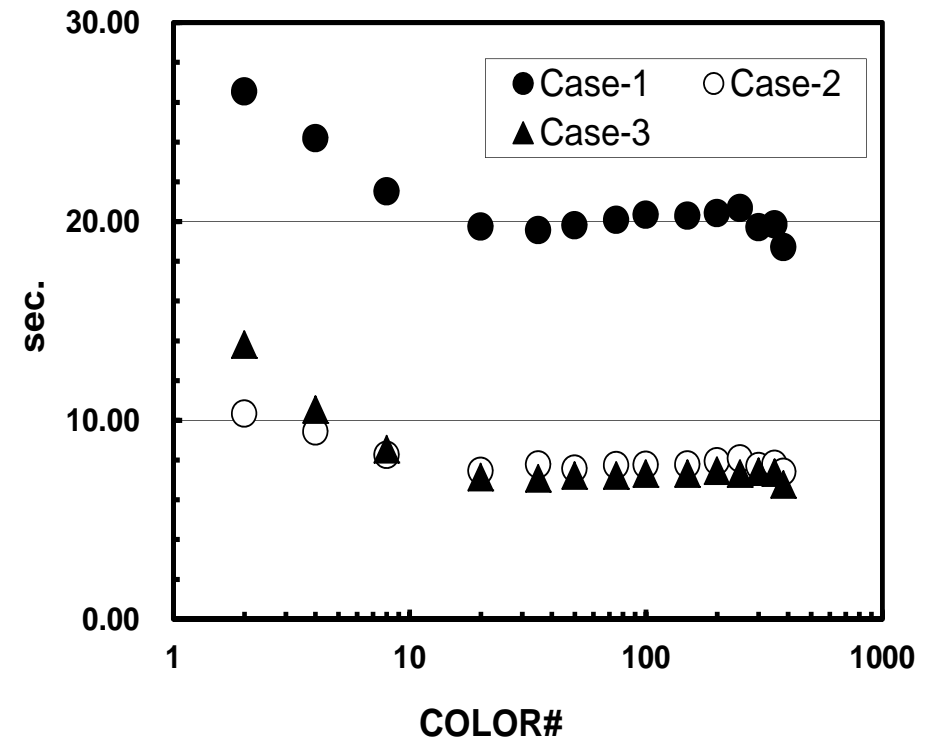
Case-1: src0

Case-2: reorder0

Case-3: reorder0 + ELL

# 計算結果：色数～計算時間：Cray XE6

- CASE-1(src0)⇒  
CASE-2(reorder0)の変化大
  - NUMA向け最適化の効果
  - First Touchとの合わせ技ではあるが
- CRS⇒ELLの効果はそれほど顕著では無い



Case-1: src0  
Case-2: reorder0  
Case-3: reorder0 + ELL

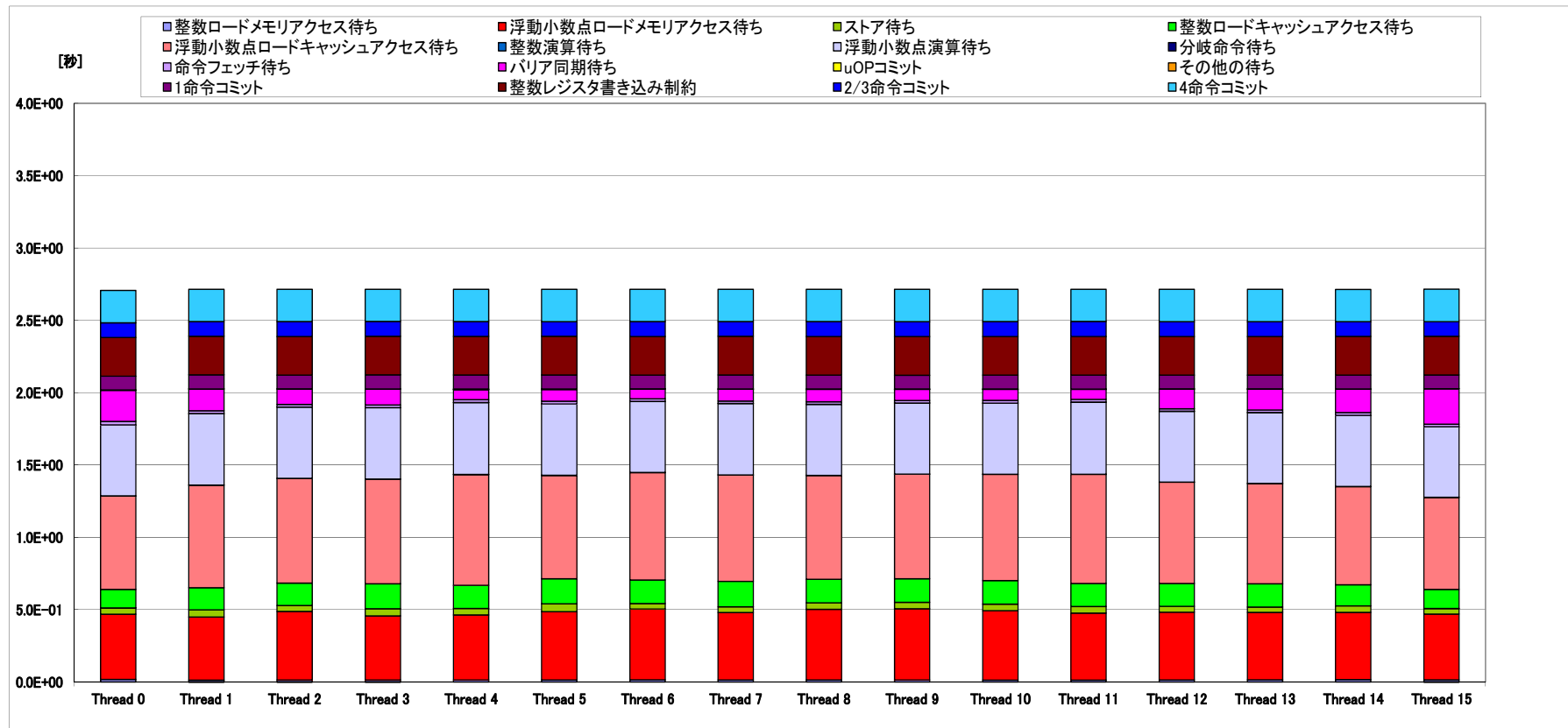
# 計算結果のまとめ

		CM-RCM(20)		CM-RCM(382) = RCM	
		計算時間 (秒)	一反復当たり計算時間 (秒)	計算時間 (秒)	一反復当たり計算時間 (秒)
Fujitsu FX10	Case-1	5.44	$1.71 \times 10^{-2}$	6.37	$2.22 \times 10^{-2}$
	Case-2	5.76	$1.81 \times 10^{-2}$	5.78	$2.02 \times 10^{-2}$
	Case-3	3.90	$1.23 \times 10^{-2}$	3.61	$1.26 \times 10^{-2}$
Cray XE6	Case-1	19.7	$6.26 \times 10^{-2}$	18.7	$6.52 \times 10^{-2}$
	Case-2	7.45	$2.34 \times 10^{-2}$	7.40	$2.58 \times 10^{-2}$
	Case-3	7.14	$2.25 \times 10^{-2}$	6.77	$2.36 \times 10^{-2}$

Case-1: src0  
Case-2: reorder0  
Case-3: reorder0 + ELL

# Fujitsu FX10: CASE-3, CM-RCM(2)

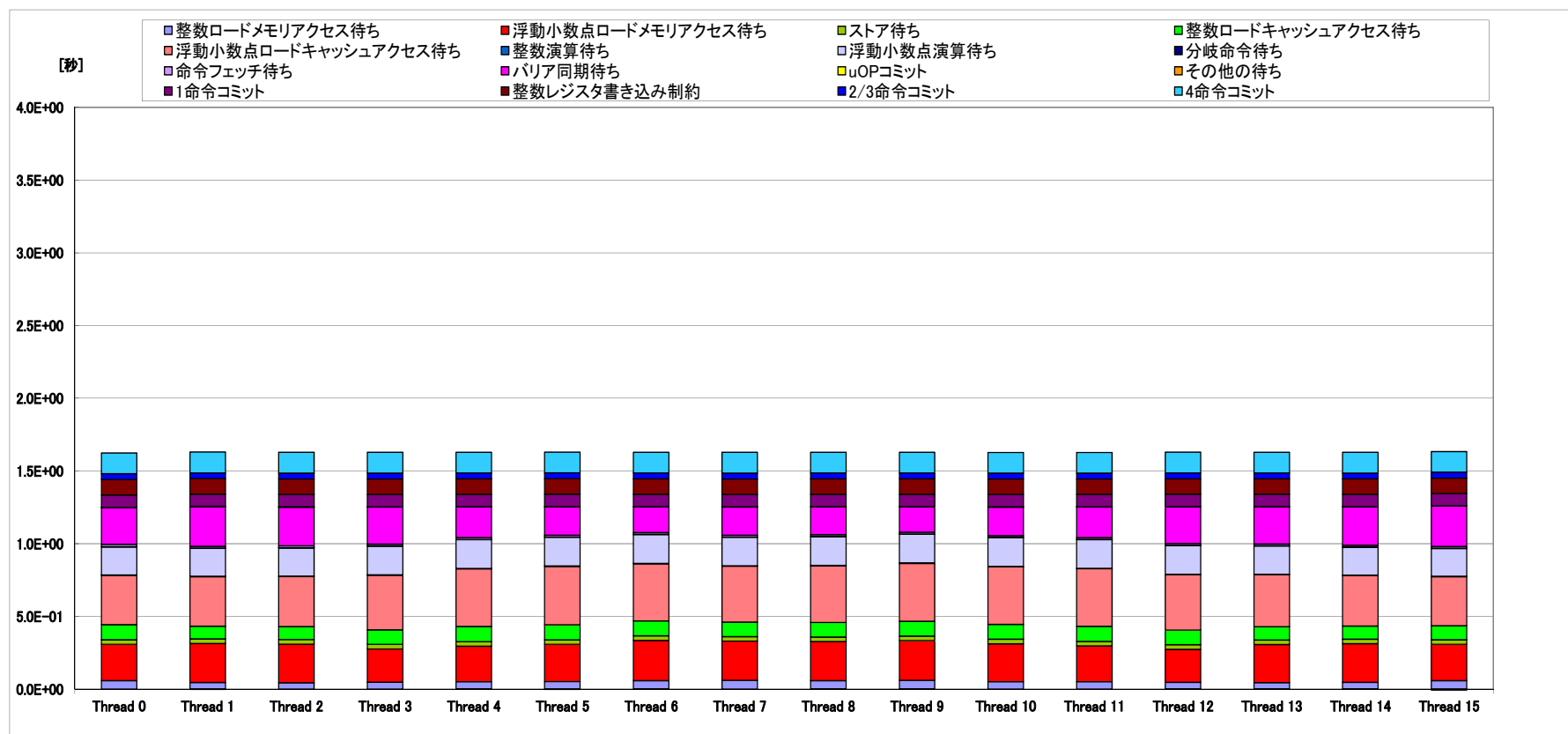
## 5.4%, 64.0GB/sec.



**ELL, Sequential**

# Fujitsu FX10: CASE-3, CM-RCM(382)

## 16.5%, 52.2GB/sec.



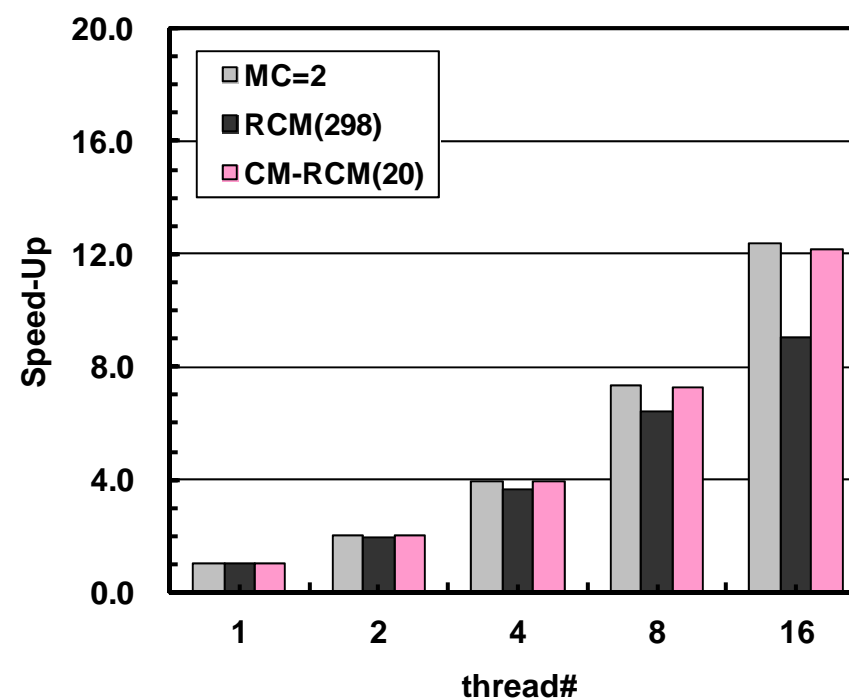
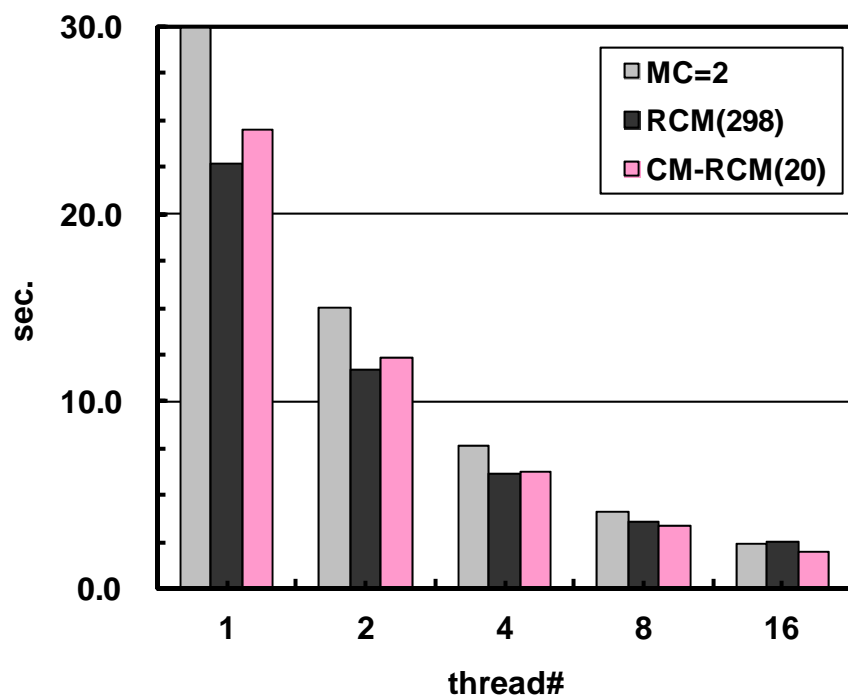
**ELL, Sequential**



- マルチコア版コードの実行
- 更なる最適化
- **STREAM**
- プロファイラ, コンパイルリスト分析等

# 計算結果(FX10@東大): $10^6$ 要素

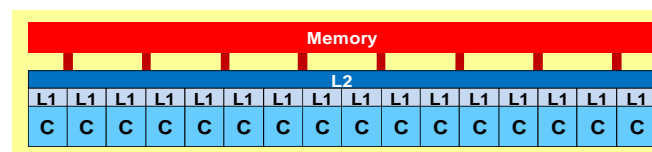
反復回数: MC(2色): 333回, RCM(298レベル): 224回  
CM-RCM( $N_c=20$ ): 249回, L3-sol



16 threads

MC(2): 2.42 sec.

CM-RCM(20): 2.01 sec.



# 何故16倍にならないか？

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

疎行列：非零成分のみ記憶  
⇒メモリへの負担大  
(memory-bound)：間接参照  
(差分, FEM, FVM)

$$\{Y\} = [A] \{X\}$$

```
do i= 1, N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    kk= item(k)
    Y(i)= Y(i) + AMAT(k)*X(kk)
  enddo
enddo
```

# 行列ベクトル積：密行列⇒とても簡単 メモリへの負担も小さい

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,N-1} & a_{1,N} \\ a_{21} & a_{22} & & a_{2,N-1} & a_{2,N} \\ \cdots & & \cdots & & \cdots \\ a_{N-1,1} & a_{N-1,2} & & a_{N-1,N-1} & a_{N-1,N} \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N-1} & a_{N,N} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{bmatrix}$$

$$\{Y\} = [A] \{X\}$$

```
do j= 1, N
  Y(j)= 0. d0
  do i= 1, N
    Y(j)= Y(j) + A(i, j)*X(i)
  enddo
enddo
```

# GeoFEM Benchmark

ICCG法の性能(固体力学向け)

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

疎行列ソルバー: Memory-Bound

# 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:  $c(i) = a(i)$
  - Scale:  $c(i) = s * b(i)$
  - Add:  $c(i) = a(i) + b(i)$
  - Triad:  $c(i) = a(i) + s * b(i)$

-----  
Double precision appears to have 16 digits of accuracy  
Assuming 8 bytes per DOUBLE PRECISION word  
-----

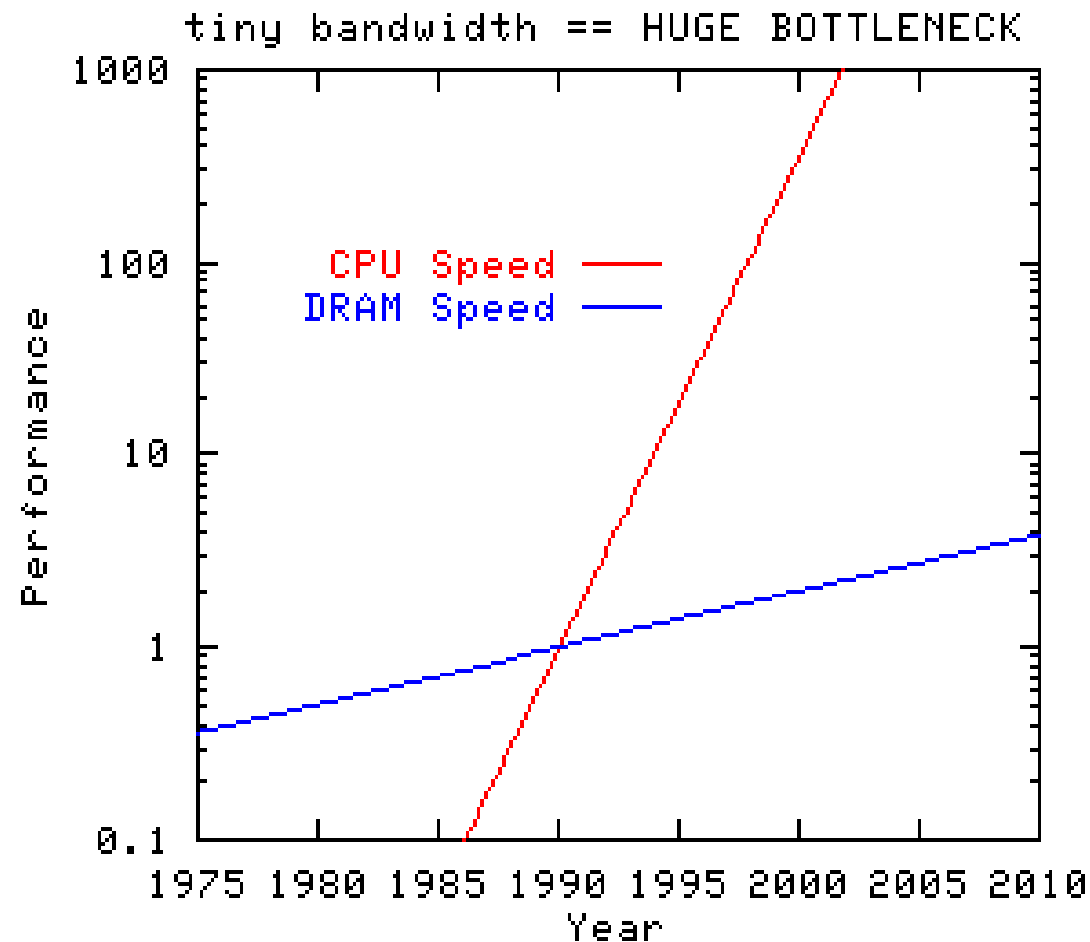
Number of processors = 16  
Array size = 2000000  
Offset = 0  
The total memory requirement is 732.4 MB  
( 45.8MB/task)  
You are running each test 10 times  
--

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

# マイクロプロセッサの動向

## CPU性能, メモリバンド幅のギャップ





# 実行: OpenMPバージョン

```
>$ cd <$O-stream>  
>$ pjsub go.sh
```

# go.sh

```
#!/bin/sh
#PJM -L "rscgrp=tutorial"
#PJM -L "node=1"
#PJM -L "elapsed=10:00"
#PJM -j
```

```
export PATH=...
export LD_LIBRARY_PATH=...
export PARALLEL=16
export OMP_NUM_THREADS=16
```

スレッド数 (1-16)

```
./stream.out > 16-01.lst 2>&1
```

出力ファイル名

# Triadの結果

## <\$O-stream>/\*.lst

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

## 実習(2)

- 実際にやってみよ
- スレッド数を変える
- 1CPU版, MPI版もある
  - FORTRAN, C
  - STREAMのサイト

- マルチコア版コードの実行
- 更なる最適化
- STREAM
- プロファイラ, コンパイルリスト分析等
  - 利用支援ポータル⇒ドキュメント閲覧⇒プログラム開発支援ツール⇒プロファイラ使用手引書⇒「3章: 詳細プロファイラ」

## デフォルト

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

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

## Kshortloop

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

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

# コンパイル・実行

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

# 最新のコンパイラではC/C++からも リスト出力可能

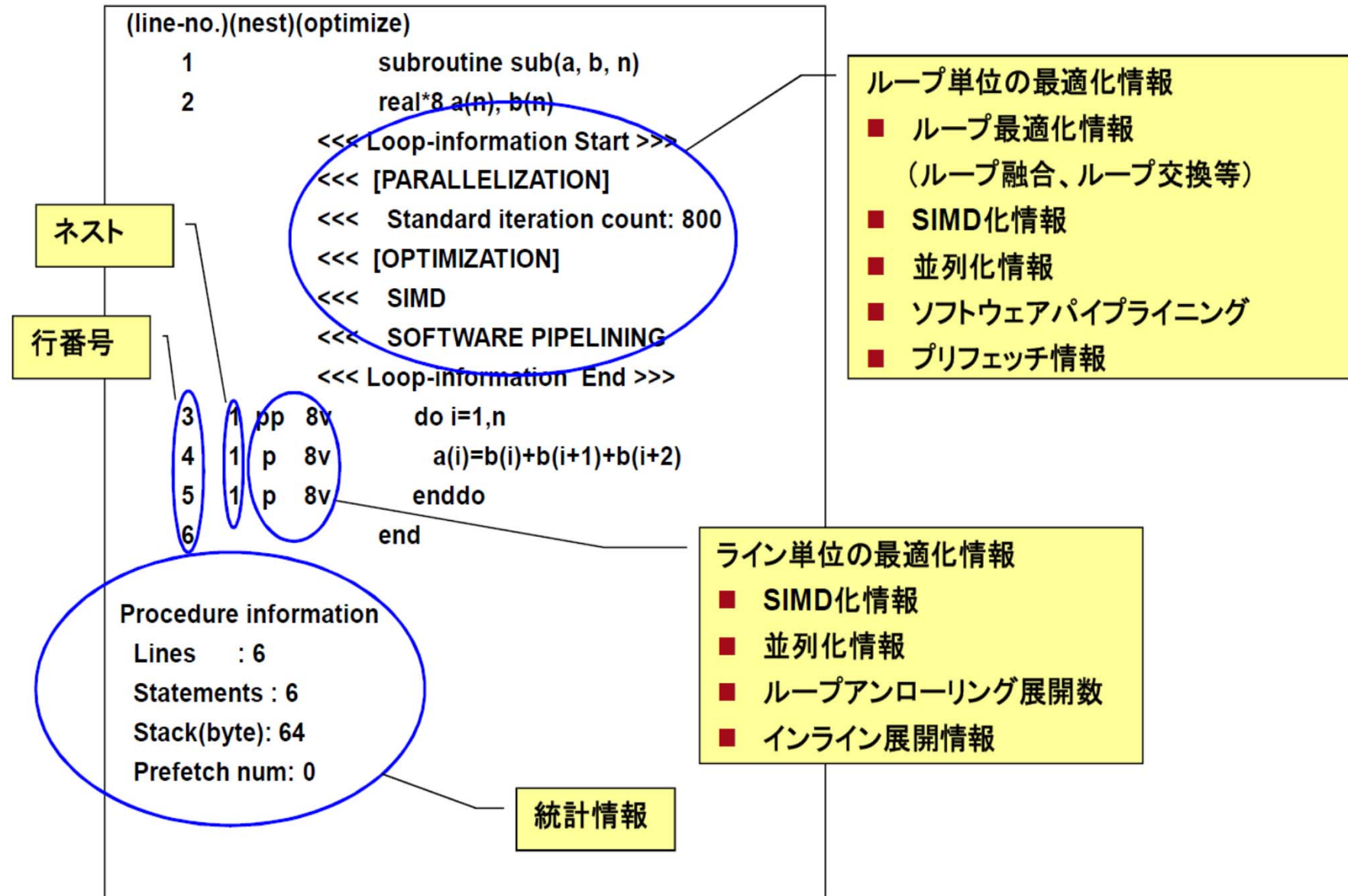
## Fortran/C/C++共通

- Nlst=p 標準の最適化情報（デフォルト）
- Nlst=t 詳細な最適化情報

## Fortranのみ

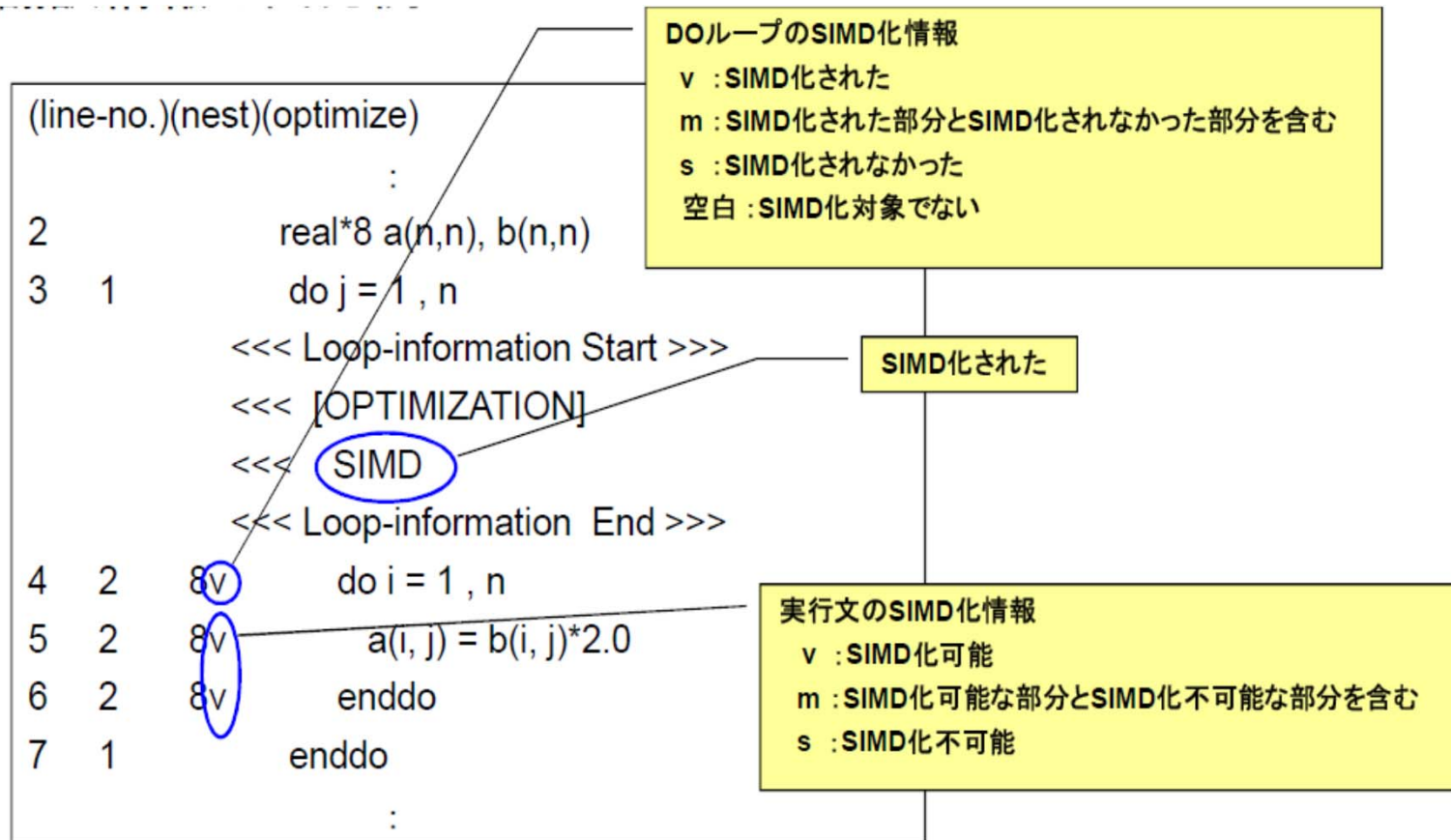
- Nlst=a 名前の属性情報
- Nlst=d 派生型の構成情報
- Nlst=i インクルードされたファイルのプログラムリスト  
およびインクルードファイル名一覧
- Nlst=m 自動並列化の状況をOpenMP指示文によって表現した  
原始プログラム出力
- Nlst=x 名前および文番号の相互参照情報

# \* .lstの見方

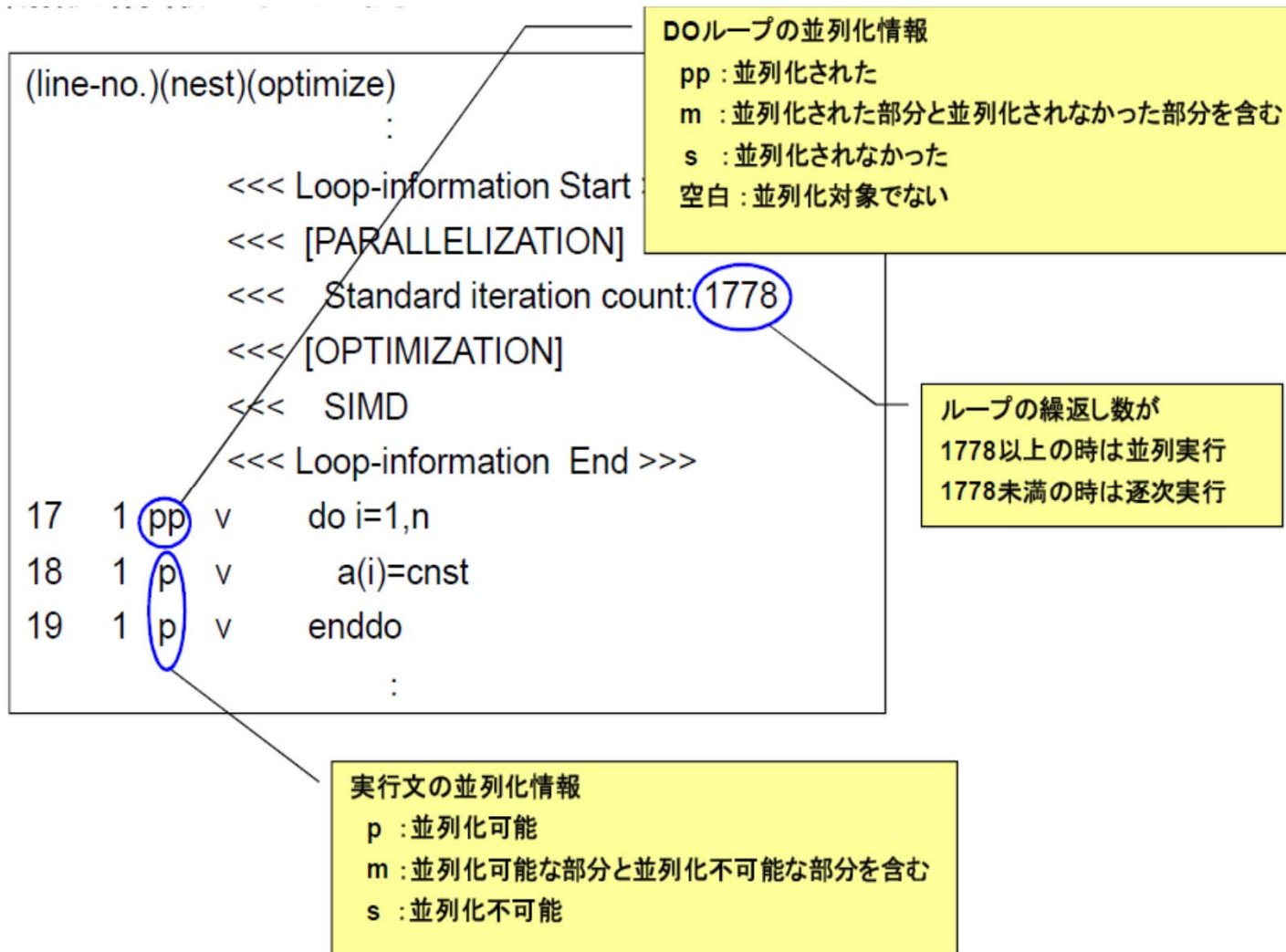




# SIMD情報



# 自動並列化情報



# solver\_ICCG\_mc.lst (src)

```

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

```

## 3.5 精密PA可視化機能(Excel形式)

### (1/3) 測定範囲指定, コンパイル・リンク

```
call start_collection ("SpMV")
!$omp parallel do private(ip, i, VAL, k)
do ip= 1, PEsmptTOT
  do i= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
    VAL= D(i)*W(i, P)
    do k= 1, 3
      VAL= VAL + AL(k, i)*W(itemL(k, i), P)
    enddo
    do k= 1, 3
      VAL= VAL + AU(k, i)*W(itemU(k, i), P)
    enddo
    W(i, Q)= VAL
  enddo
enddo
!$omp end parallel do
call stop_collection ("SpMV")
```

## 3.5 精密PA可視化機能(Excel形式)

(2/3) データ収集:7回実行  
ディレクトリ名: pa1~pa7, -Hpa=1~7

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:05:00"
#PJM -L "rscgrp=debug"
#PJM -g "pz0088"
#PJM -j
#PJM -o "3.lst"
#PJM --mpi "proc=1"

export OMP_NUM_THREADS=16
fapp -C -d pa1 -lhwm -Hpa=1 ./sol-r3k
```

## 3.5 精密PA可視化機能(Excel形式)

### (3/3) データ解析:変換+Excelシート(p.80)

```
fapppx -A -d pa1 -o output_prof_1.csv -tcsv -Hpa  
fapppx -A -d pa2 -o output_prof_2.csv -tcsv -Hpa  
fapppx -A -d pa3 -o output_prof_3.csv -tcsv -Hpa  
fapppx -A -d pa4 -o output_prof_4.csv -tcsv -Hpa  
fapppx -A -d pa5 -o output_prof_5.csv -tcsv -Hpa  
fapppx -A -d pa6 -o output_prof_6.csv -tcsv -Hpa  
fapppx -A -d pa7 -o output_prof_7.csv -tcsv -Hpa
```

# まとめ

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

# 今後の動向

- メモリバンド幅と性能のギャップ
  - BYTE/FLOP, 中々縮まらない
- マルチコア化, メニーコア化
  - Intel Xeon/Phi
- $>10^5$ 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