

科学技術計算のための  
マルチコアプログラミング入門  
C言語編

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



# 実行例

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

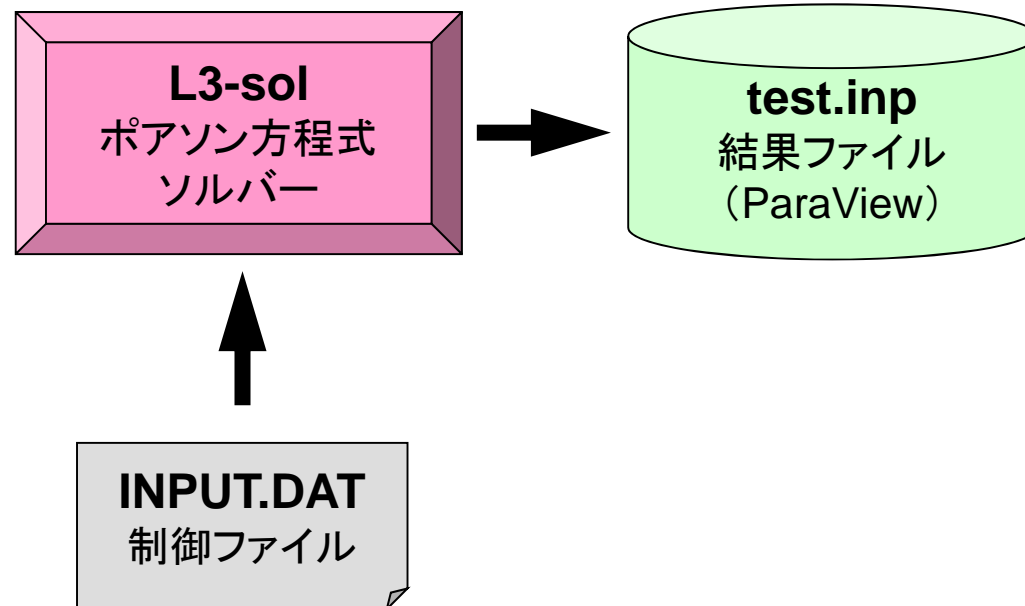
% cd src
% make
% cd ../run
% ls L3-sol
    L3-sol

% <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



# Control Data: 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      PEsmptOTと一致させる
./L3-sol
```

# ジョブスクリプト

- `<$O-L3>/run/go1.sh`
- スケジューラへの指令 + シェルスクリプト

```
#!/bin/sh
#PJM -L "node=1"          ノード数
#PJM -L "elapsed=00:05:00" 実行時間
#PJM -L "rscgrp=school"     実行キュー名
#PJM -j
#PJM -o "test.lst"         標準出力ファイル名

export OMP_NUM_THREADS=16

./L3-sol                  実行ファイル名
```

# ジョブ投入, 確認等

- ジョブの投入 `pjsub` スクリプト名
- ジョブの確認 `pjstat`
- ジョブの取り消し・強制終了 `pjdel` ジョブID
- キューの状態の確認 `pjstat --rsc`
- キューの詳細構成 `pjstat --rsc -x`
- 実行中のジョブ数 `pjstat --rsc -b`
- 同時実行・投入可能数 `pjstat --limit`

```
[z30088@oakleaf-fx-6 S2-ref]$ pjstat
```

```
Oakleaf-FX scheduled stop time: 2012/09/28(Fri) 09:00:00 (Remain: 31days 20:01:46)
```

JOB_ID	JOB_NAME	STATUS	PROJECT	RSCGROUP	START_DATE	ELAPSE	TOKEN	NODE:COORD
334730	go. sh	RUNNING	gt61	lecture	08/27 12:58:08	00:00:05	0.0	1

- 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)

```
#pragma omp parallel for private(i, VAL, j)
for (i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for (j=indexL[i]; j<indexL[i+1]; j++) {
        VAL += AL[j] * W[P][itemL[j]-1];
    }
    for (j=indexU[i]; j<indexU[i+1]; j++) {
        VAL += AU[j] * W[P][itemU[j]-1];
    }
    W[Q][i] = VAL;
}
```

- 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
- GPU, メニィコアではこのままの方が良い場合もある

# 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)

```
for (ic=0; ic<NCOLORtot; ic++) {  
#pragma omp parallel for private (i,WVAL,j)  
    for (i=COLORindex[ic]; i<COLORindex[ic+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];  
    }  
}
```

- 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
- GPU, メニィコアではこのままの方が良い場合もある

# ICCG法の並列化: OpenMP

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

# Main Program

```

#include <stdio.h> ...

int
main()
{
    double *WK;
    int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
    double xN, xL, xU; Stime, Etime;

    if(INPUT()) goto error;
    if(POINTER_INIT()) goto error;
    if(BOUNDARY_CELL()) goto error;
    if(CELL_METRICS()) goto error;
    if(POI_GEN()) goto error;

    ISET = 0;
    WK = (double *)malloc(sizeof(double)*ICELTOT);
    if(WK == NULL) {
        fprintf(stderr, "Error: %s\n", strerror(errno));
        goto error;
    }
    Stime = omp_get_wtime();
    if(solve_ICCG_mc(ICELTOT, NL, NU, indexL, itemL, indexU, itemU,
                    D, BFORCE, PHI, AL, AU, NCOLORTot, PEsmptOT,
                    SMPindex, SMPindexG, EPSICCG, &ITR, &IER)) goto error;
    Etime = omp_get_wtime();
    for(ic0=0; ic0<ICELTOT; ic0++) {
        icel = NEWtoOLD[ic0];
        WK[icel-1] = PHI[ic0];
    }
    for(icel=0; icel<ICELTOT; icel++) {
        PHI[icel] = WK[icel];
    }
    if(OUTUCD()) goto error;
    return 0;
error:
    return -1;
}

```

# struct.h

```

#ifndef __H_STRUCT
#define __H_STRUCT

#include <omp.h>

int ICELTOT, ICELTOTp, N;
int NX, NY, NZ, NXP1, NYP1, NZP1, IBNODTOT;
int NXc, NYc, NZc;

double DX, DY, DZ, XAREA, YAREA, ZAREA;
double RDX, RDY, RDZ, RDX2, RDY2, RDZ2, R2DX, R2DY, R2DZ;
double *VOLCEL, *VOLNOD, *RVC, *RVN;

int **XYZ, **NEIBcell;

int ZmaxCELTot;
int *BC_INDEX, *BC_NOD;
int *ZmaxCEL;

int **IWKX;
double **FCV;

int my_rank, PETOT, PEsmptOT;

#endif /* __H_STRUCT */

```

## ICELTOT:

Number of meshes (NX x NY x NZ)

## N:

Number of modes

## NX, NY, NZ:

Number of meshes in x/y/z directions

## NXP1, NYP1, NZP1:

Number of nodes in x/y/z directions

## IBNODTOT:

= NXP1 x NYP1

## XYZ[ICELTOT][3]:

Location of meshes

## NEIBcell[ICELTOT][6]:

Neighboring meshes

## PEsmptOT:

Number of threads

# pcg.h

```

#ifndef __H_PCG
#define __H_PCG
    static int N2 = 256;
    int NUmax, NLmax, NCOLORTtot, NCOLORK, NU,
NL;

    int METHOD, ORDER_METHOD;
    double EPSICCG;

    double *D, *PHI, *BFORCE;
    double *AL, *AU;

    int *INL, *INU, *COLORindex;
    int *indexL, *indexU;
    int *SMPindex, *SMPindexG;
    int *OLDtoNEW, *NEWtoOLD;
    int **IAL, **IAU;
    int *itemL, *itemU;
    int NPL, NPU;
#endif /* __H_PCG */

```

**NCOLOR<sub>tot</sub>**            Total number of colors/levels  
**COLOR<sub>index</sub>**            Index of number of meshes in each color/level  
**[NCOLOR<sub>tot</sub>+1]**    (COLOR<sub>index</sub>[icol+1]- COLOR<sub>index</sub>[icol])

**SMP<sub>index</sub> [NCOLOR<sub>tot</sub>\*PE<sub>smp</sub>TOT+1]**  
**SMP<sub>index</sub>G [PE<sub>smp</sub>TOT+1]**

**OLDtoNEW, NEWtoOLD**    Reference table before/after renumbering

# Variables/Arrays for Matrix (1/2)

Name	Type	Content
<b>D[N]</b>	<b>R</b>	Diagonal components of the matrix (N= ICELTOT)
<b>BFORCE[N]</b>	<b>R</b>	RHS vector
<b>PHI[N]</b>	<b>R</b>	Unknown vector
<b>indexL[N+2]</b> <b>indexU[N+2]</b>	<b>I</b>	# of L/U non-zero off-diag. comp. (CRS)
<b>NPL, NPU</b>	<b>I</b>	Total # of L/U non-zero off-diag. comp. (CRS)
<b>itemL[NPL]</b> <b>itemU[NPU]</b>	<b>I</b>	Column ID of L/U non-zero off-diag. comp. (CRS)
<b>AL[NPL]</b> <b>AU[NPU]</b>	<b>R</b>	L/U non-zero off-diag. comp. (CRS)

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

# Variables/Arrays for Matrix (2/2)

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



# Main Program

```

#include <stdio.h> ...

int
main()
{
    double *WK;
    int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
    double xN, xL, xU; Stime, Etime;

    if(INPUT()) goto error;
    if(POINTER_INIT()) goto error;
    if(BOUNDARY_CELL()) goto error;
    if(CELL_METRICS()) goto error;
    if(POI_GEN()) goto error;

    ISET = 0;
    WK = (double *)malloc(sizeof(double)*ICELTOT);
    if(WK == NULL) {
        fprintf(stderr, "Error: %s\n", strerror(errno));
        goto error;
    }
    Stime = omp_get_wtime();
    if(solve_ICCG_mc(ICELTOT, NL, NU, indexL, itemL, indexU, itemU,
                    D, BFORCE, PHI, AL, AU, NCOLORTot, PEsmptOT,
                    SMPindex, SMPindexG, EPSICCG, &ITR, &IER)) goto error;

    Etime = omp_get_wtime();
    for(ic0=0; ic0<ICELTOT; ic0++) {
        icel = NEWtoOLD[ic0];
        WK[icel-1] = PHI[ic0];
    }
    for(icel=0; icel<ICELTOT; icel++) {
        PHI[icel] = WK[icel];
    }
    if(OUTUCD()) goto error;
    return 0;
error:
    return -1;
}

```

# input: reading INPUT.DAT

```
#include <stdio.h>; <stdlib.h>; <string.h>; <errno.h>
#include "struct_ext.h"; "pcg_ext.h"; "input.h"

extern int
INPUT(void)
{
#define BUF_SIZE 1024

char line[BUF_SIZE];
char CNTFIL[81];
double OMEGA;
FILE *fp11;

if((fp11 = fopen("INPUT.DAT", "r")) == NULL) {
    fprintf(stderr, "Error: %s\n", strerror(errno));
    return -1;
}
sscanf(line, "%d%d%d", &NX, &NY, &NZ);
sscanf(line, "%d", &METHOD);
sscanf(line, "%le%le%le", &DX, &DY, &DZ);
sscanf(line, "%le", &EPSICCG);
sscanf(line, "%d", &PEsmptOT);
sscanf(line, "%d", &NCOLORtot);

fclose(fp11);
return 0;
}
```

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

```
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

```

#include <stdio.h> ...

extern int
CELL_METRICS(void)
{
    double V0, RVO;
    int i;
    VOLCEL =
    (double *)allocate_vector(sizeof(double), ICELTOT);
    RVC =
    (double *)allocate_vector(sizeof(double), ICELTOT);

    XAREA = DY * DZ;
    YAREA = DZ * DX;
    ZAREA = DX * DY;

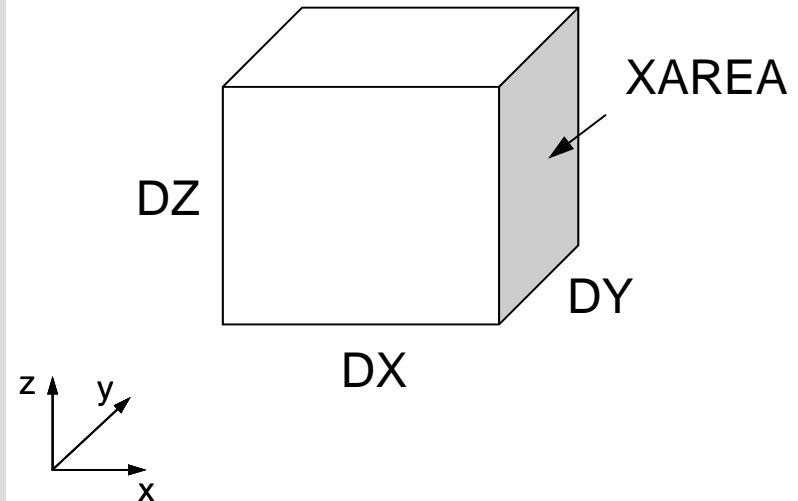
    RDX = 1.0 / DX;
    RDY = 1.0 / DY;
    RDZ = 1.0 / DZ;

    RDX2 = 1.0 / (pow(DX, 2.0));
    RDY2 = 1.0 / (pow(DY, 2.0));
    RDZ2 = 1.0 / (pow(DZ, 2.0));
    R2DX = 1.0 / (0.5 * DX);
    R2DY = 1.0 / (0.5 * DY);
    R2DZ = 1.0 / (0.5 * DZ);

    V0 = DX * DY * DZ;
    RVO = 1.0 / V0;

    for(i=0; i<ICELTOT; i++) {
        VOLCEL[i] = V0;
        RVC[i] = RVO;
    }
    return 0; }

```



# Main Program

```

#include <stdio.h> ...

int
main()
{
    double *WK;
    int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
    double xN, xL, xU; Stime, Etime;

    if(INPUT()) goto error;
    if(POINTER_INIT()) goto error;
    if(BOUNDARY_CELL()) goto error;
    if(CELL_METRICS()) goto error;
    if(POI_GEN()) goto error;

    ISET = 0;
    WK = (double *)malloc(sizeof(double)*ICELTOT);
    if(WK == NULL) {
        fprintf(stderr, "Error: %s\n", strerror(errno));
        goto error;
    }
    Stime = omp_get_wtime();
    if(solve_ICCG_mc(ICELTOT, NL, NU, indexL, itemL, indexU, itemU,
                    D, BFORCE, PHI, AL, AU, NCOLORTot, PEsmptOT,
                    SMPindex, SMPindexG, EPSICCG, &ITR, &IER)) goto error;
    Etime = omp_get_wtime();
    for(ic0=0; ic0<ICELTOT; ic0++) {
        icel = NEWtoOLD[ic0];
        WK[icel-1] = PHI[ic0];
    }
    for(icel=0; icel<ICELTOT; icel++) {
        PHI[icel] = WK[icel];
    }
    if(OUTUCD()) goto error;
    return 0;
error:
    return -1;
}

```

# poi\_gen (1/9)

```
#include "allocate.h"
extern int
POI_GEN(void)
{ int nn;
  int ic0, icN1, icN2, icN3, icN4, icN5, icN6;
  int i, j, k, ib, ic, ip, icel, icou, icol, icouG;
  int ii, jj, kk, nn1, num, nr, j0, j1;
  double coef, VOL0, S1t, E1t;
  int isL, ieL, isU, ieU;
  NL=6; NU= 6;
  IAL = (int **)allocate_matrix(sizeof(int), ICELTOT, NL);
  IAU = (int **)allocate_matrix(sizeof(int), ICELTOT, NU);
  BFORCE = (double *)allocate_vector(sizeof(double), ICELTOT);
  D       = (double *)allocate_vector(sizeof(double), ICELTOT);
  PHI     = (double *)allocate_vector(sizeof(double), ICELTOT);
  INL     = (int *)allocate_vector(sizeof(int), ICELTOT);
  INU     = (int *)allocate_vector(sizeof(int), ICELTOT);

  for (i = 0; i < ICELTOT ; i++) {
    BFORCE[i]=0.0;
    D[i]     =0.0; PHI[i]=0.0;
    INL[i]   = 0; INU[i] = 0;
    for (j=0; j<6; j++) {
      IAL[i][j]=0; IAU[i][j]=0;
    }
  }
  for (i = 0; i <= ICELTOT ; i++) {
    indexL[i] = 0; indexU[i] = 0;
  }
}
```

```
/******
  allocate matrix                               allocate.c
  *****/
void** allocate_matrix(int size, int m, int n)
{
  void **aa;
  int i;
  if ( ( aa=(void **)malloc( m * sizeof(void*) ) ) == NULL ) {
    fprintf(stdout, "Error:Memory does not enough! aa in matrix %n");
    exit(1);
  }
  if ( ( aa[0]=(void *)malloc( m * n * size ) ) == NULL ) {
    fprintf(stdout, "Error:Memory does not enough! in matrix %n");
    exit(1);
  }
  for (i=1; i<m; i++) aa[i]=(char*)aa[i-1]+size*n;
  return aa;
}
```

```

for (icel=0; icel<ICELTOT; icel++) {
  icN1 = NEIBcell[icel][0];
  icN2 = NEIBcell[icel][1];
  icN3 = NEIBcell[icel][2];
  icN4 = NEIBcell[icel][3];
  icN5 = NEIBcell[icel][4];
  icN6 = NEIBcell[icel][5];

  if(icN5 != 0) {
    icou = INL[icel] + 1;
    IAL[icel][icou-1] = icN5;
    INL[icel] = icou;
  }

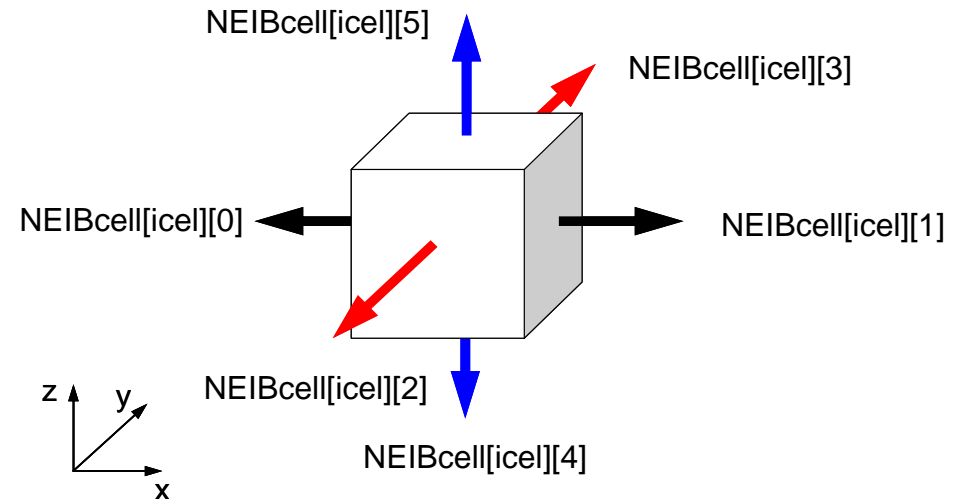
  if(icN3 != 0) {
    icou = INL[icel] + 1;
    IAL[icel][icou-1] = icN3;
    INL[icel] = icou;
  }
  if(icN1 != 0) {
    icou = INL[icel] + 1;
    IAL[icel][icou-1] = icN3;
    INL[icel] = icou;
  }
  if(icN2 != 0) {
    icou = INU[icel] + 1;
    IAU[icel][icou-1] = icN2;
    INU[icel] = icou;
  }

  if(icN4 != 0) {
    icou = INU[icel] + 1;
    IAU[icel][icou-1] = icN4;
    INU[icel] = icou;
  }

  if(icN6 != 0) {
    icou = INU[icel] + 1;
    IAU[icel][icou-1] = icN6;
    INU[icel] = icou;
  }
}

```

## poi\_gen (2/9)



### Lower Triangular Part

```

NEIBcell[icel][4]= icel - NX*NY + 1
NEIBcell[icel][2]= icel - NX      + 1
NEIBcell[icel][0]= icel - 1      + 1

```

“icel” starts at 0

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

“IAL” starts at 1

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

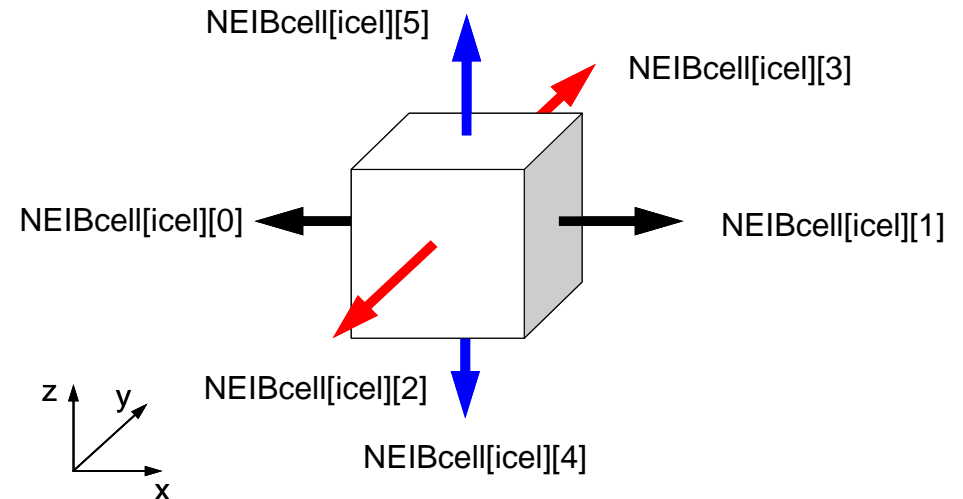
```

for (icel=0; icel<ICELTOT; icel++) {
  icN1 = NEIBcell[icel][0];
  icN2 = NEIBcell[icel][1];
  icN3 = NEIBcell[icel][2];
  icN4 = NEIBcell[icel][3];
  icN5 = NEIBcell[icel][4];
  icN6 = NEIBcell[icel][5];

  if (icN5 != 0) {
    icou = INL[icel] + 1;
    IAL[icel][icou-1] = icN5;
    INL[icel] = icou;
  }
  if (icN3 != 0) {
    icou = INL[icel] + 1;
    IAL[icel][icou-1] = icN3;
    INL[icel] = icou;
  }
  if (icN1 != 0) {
    icou = INL[icel] + 1;
    IAL[icel][icou-1] = icN3;
    INL[icel] = icou;
  }
  if (icN2 != 0) {
    icou = INU[icel] + 1;
    IAU[icel][icou-1] = icN2;
    INU[icel] = icou;
  }
  if (icN4 != 0) {
    icou = INU[icel] + 1;
    IAU[icel][icou-1] = icN4;
    INU[icel] = icou;
  }
  if (icN6 != 0) {
    icou = INU[icel] + 1;
    IAU[icel][icou-1] = icN6;
    INU[icel] = icou;
  }
}

```

## poi\_gen (2/9)



### Upper Triangular Part

```

NEIBcell[icel][1]= icel + 1      + 1
NEIBcell[icel][3]= icel + NX    + 1
NEIBcell[icel][5]= icel + NX*NY + 1

```

“icel” starts at 0

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

“IAU” starts at 1

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

# poi\_gen (3/9)

## Reordering

NCOLORtot > 1: Multicolor

NCOLORtot = 0: CM

NCOLORtot = -1: RCM

NCOLORtot < -1: CM-RCM

```
N111:
fprintf(stderr, "%n%nYou have%8d elements%n", ICELTOT);
fprintf(stderr, "How many colors do you need ?%n");
fprintf(stderr, "  #COLOR must be more than 2 and%n");
fprintf(stderr, "  #COLOR must not be more than%8d%n", ICELTOT);
fprintf(stderr, "  if #COLOR= 0 then CM ordering%n");
fprintf(stderr, "  if #COLOR=-1 then RCM ordering%n");
fprintf(stderr, "  if #COLOR<-1 then CMRCM ordering%n");
fprintf(stderr, "=>%n");
fscanf(stdin, "%d", &NCOLORtot);
if(NCOLORtot == 1 && NCOLORtot > ICELTOT) goto N111;

OLDtoNEW = (int *)calloc(ICELTOT, sizeof(int));
if(OLDtoNEW == NULL) {
    fprintf(stderr, "Error: %s%n", strerror(errno));
    return -1;
}
NEWtoOLD = (int *)calloc(ICELTOT, sizeof(int));
if(NEWtoOLD == NULL) {
    fprintf(stderr, "Error: %s%n", strerror(errno));
    return -1;
}
COLORindex = (int *)calloc(ICELTOT+1, sizeof(int));
if(COLORindex == NULL) {
    fprintf(stderr, "Error: %s%n", strerror(errno));
    return -1;
}

if(NCOLORtot > 0) {
    MC(ICELTOT, NL, NU, INL, IAL, INU, IAU,
        &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
} else if(NCOLORtot == 0) {
    CM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
        &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
} else if(NCOLORtot == -1) {
    RCM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
        &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
} else if(NCOLORtot < -1) {
    CMRCM(ICELTOT, NL, NU, INL, IAL, INU, IAU,
        &NCOLORtot, COLORindex, NEWtoOLD, OLDtoNEW);
}

fprintf(stderr, "%n# TOTAL COLOR number%8d%n", NCOLORtot);
return 0;
}
```



```

SMPindex = (int *) allocate_vector(sizeof(int),
NCOLORtot*PEsmpTOT+1);
memset(SMPindex, 0,
sizeof(int)*(NCOLORtot*PEsmpTOT+1));

for(ic=1; ic<=NCOLORtot; ic++) {
    nn1 = COLORindex[ic] - COLORindex[ic-1];
    num = nn1 / PEsmpTOT;
    nr = nn1 - PEsmpTOT * num;
    for(ip=1; ip<=PEsmpTOT; ip++) {
        if(ip <= nr) {
            SMPindex[(ic-1)*PEsmpTOT+ip] = num + 1;
        } else {
            SMPindex[(ic-1)*PEsmpTOT+ip] = num;
        }
    }
}

for(ic=1; ic<=NCOLORtot; ic++) {
    for(ip=1; ip<=PEsmpTOT; ip++) {
        j1 = (ic-1) * PEsmpTOT + ip;
        j0 = j1 - 1;
        SMPindex[j1] += SMPindex[j0];
    }
}

```

```

SMPindexG = (int *) allocate_vector
PEsmpTOT+1);
memset(SMPindexG, 0, sizeof(int)*(PE

nn = ICELTOT / PEsmpTOT;
nr = ICELTOT - nn * PEsmpTOT;
for(ip=1; ip<=PEsmpTOT; ip++) {
    SMPindexG[ip] = nn;
    if(ip <= nr) {SMPindexG[ip] +=
}
for(ip=1; ip<=PEsmpTOT; ip++) {
    SMPindexG[ip] += SMPindexG[ip-1],
}

```

```

for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for ...
    for(ip=0; ip<PEsmpTOT; ip++) {
        ip1 = ic * PEsmpTOT + ip;
        for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
            (...)
        }
    }
}

```

# poi\_gen (4/9)

## SMPindex:

### for preconditioning

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  $\Rightarrow$  OpenMP適用

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

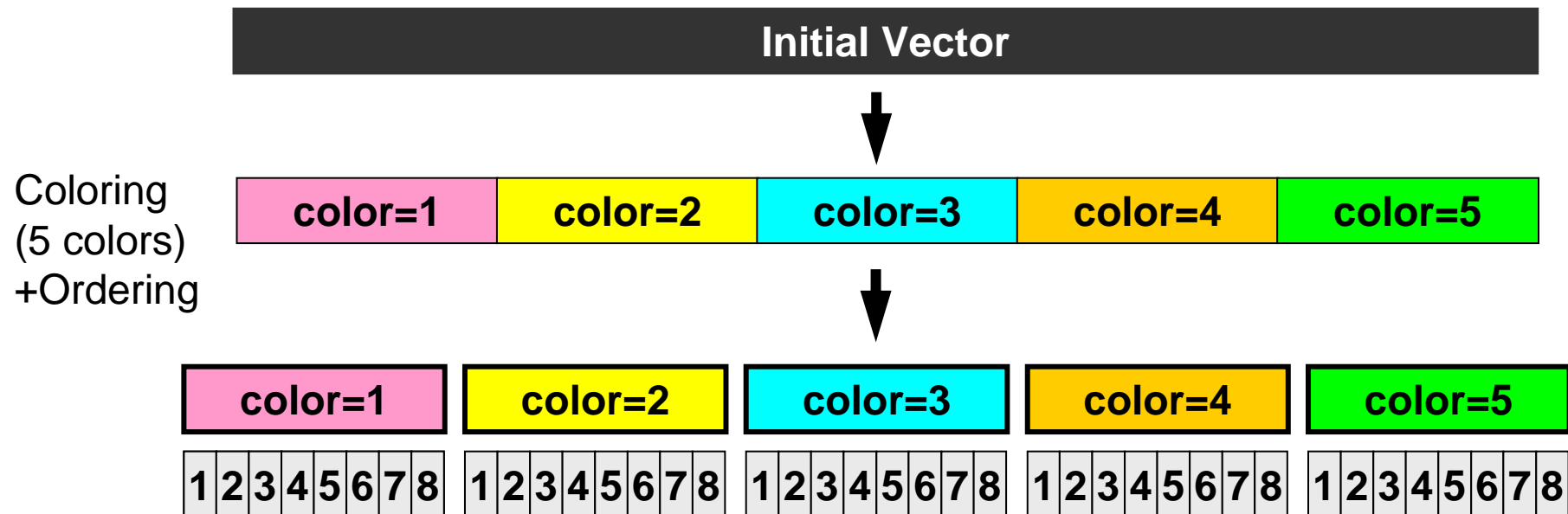
**前処理で使用**

# SMPindex: 前処理向け

```

for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for ...
  for(ip=0; ip<PEsmpTOT; ip++) {
    ip1 = ic * PEsmpTOT + ip;
    for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
      (...)
    }
  }
}

```



- 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)

```
SMPindex = (int *) allocate_vector(sizeof(int),
NCOLORtot*PEsmptTOT+1);
memset(SMPindex, 0,
sizeof(int)*(NCOLORtot*PEsmptTOT+1));

for(ic=1; ic<=NCOLORtot; ic++) {
    nn1 = COLORindex[ic] - COLORindex[ic-1];
    num = nn1 / PEsmptTOT;
    nr = nn1 - PEsmptTOT * num;
    for(ip=1; ip<=PEsmptTOT; ip++) {
        if(ip <= nr) {
            SMPindex[(ic-1)*PEsmptTOT+ip] = num + 1;
        } else {
            SMPindex[(ic-1)*PEsmptTOT+ip] = num;
        }
    }
}

for(ic=1; ic<=NCOLORtot; ic++) {
    for(ip=1; ip<=PEsmptTOT; ip++) {
        j1 = (ic-1) * PEsmptTOT + ip;
        j0 = j1 - 1;
        SMPindex[j1] += SMPindex[j0];
    }
}
```

```
#pragma omp parallel for ...
for(ip=0; ip<PEsmptTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip]; i++) {
        (...)
    }
}
```

```
SMPindexG = (int *) allocate_vector(sizeof(int),
PEsmptTOT+1);
memset(SMPindexG, 0, sizeof(int)*(PEsmptTOT+1));

nn = ICELTOT / PEsmptTOT;
nr = ICELTOT - nn * PEsmptTOT;
for(ip=1; ip<=PEsmptTOT; ip++) {
    SMPindexG[ip] = nn;
    if(ip <= nr) {SMPindexG[ip] += 1;}
}
for(ip=1; ip<=PEsmptTOT; ip++) {
    SMPindexG[ip] += SMPindexG[ip-1];
}
```

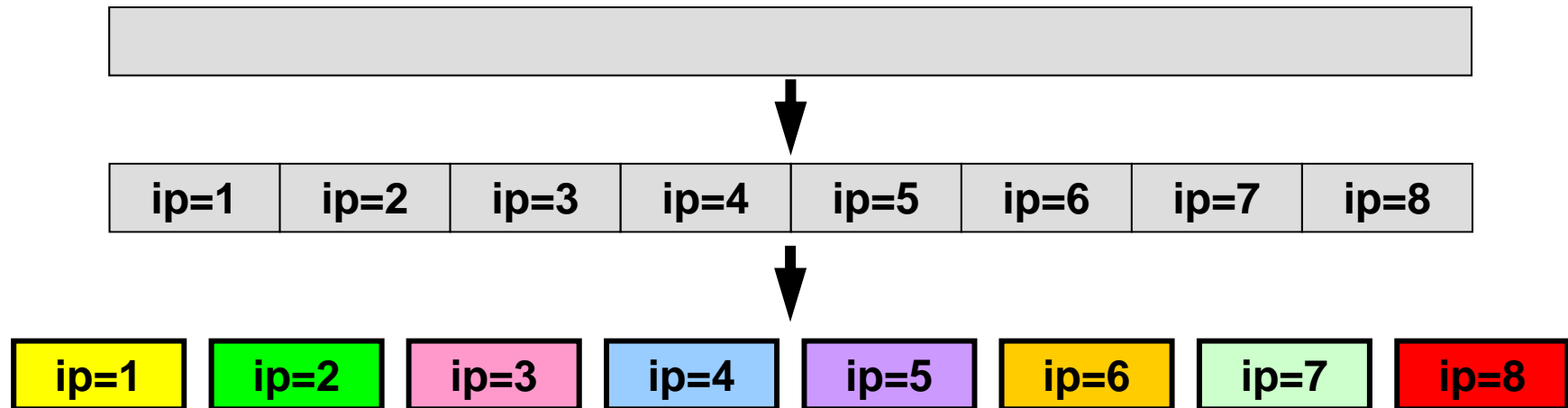
全要素数を「PEsmptTOT」で割って  
「SMPindexG」に割り当てる。

**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

```
#pragma omp parallel for ...  
for(ip=0; ip<PEsmpTOT; ip++) {  
    for(i=SMPindexG[ip]; i<SMPindexG[ip]; i++) {  
        (...)  
    }  
}
```



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

```

indexL =
(int *)allocate_vector(sizeof(int), ICELTOT+1);
indexU =
(int *)allocate_vector(sizeof(int), ICELTOT+1);

for(i=0; i<ICELTOT; i++){
    indexL[i+1]=indexL[i]+INL[i];
    indexU[i+1]=indexU[i]+INU[i];
}
NPL = indexL[ICELTOT];
NPU = indexU[ICELTOT];

itemL = (int *)allocate_vector(sizeof(int), NPL);
itemU = (int *)allocate_vector(sizeof(int), NPU);
AL =
(double *)allocate_vector(sizeof(double), NPL);
AU =
(double *)allocate_vector(sizeof(double), NPU);

memset(itemL, 0, sizeof(int)*NPL);
memset(itemU, 0, sizeof(int)*NPU);
memset(AL, 0.0, sizeof(double)*NPL);
memset(AU, 0.0, sizeof(double)*NPU);

```

```

for(i=0; i<ICELTOT; i++){
    for(k=0; k<INL[i]; k++){
        kk= k + indexL[i];
        itemL[kk]= IAL[i][k];
    }
    for(k=0; k<INU[i]; k++){
        kk= k + indexU[i];
        itemU[kk]= IAU[i][k];
    }
}

```

```

free(INL); free(INU);
free(IAL); free(IAU);

```

“itemL” / “itemU”  
start at 1

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Name	Type	Content
D[N]	R	Diagonal components of the matrix (N= ICELTOT)
BFORCE[N]	R	RHS vector
PHI[N]	R	Unknown vector
indexL[N+1] indexU[N+1]	I	# of L/U non-zero off-diag. comp. (CRS)
NPL, NPU	I	Total # of L/U non-zero off-diag. comp. (CRS)
itemL[NPL] itemU[NPU]	I	Column ID of L/U non-zero off-diag. comp. (CRS)
AL[NPL] AU[NPU]	R	L/U non-zero off-diag. comp. (CRS)

```

for (i=0; i<N; i++) {
    q[i]= D[i] * p[i];
    for (j=indexL[i]; j<indexL[i+1]; j++) {
        q[i] += AL[j] * p[itemL[j]-1];
    }
    for (j=indexU[i]; j<indexU[i+1]; j++) {
        q[i] += AU[j] * p[itemU[j]-1];
    }
}

```

```

S1t = omp_get_wtime();
#pragma omp parallel for private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6, coef, j,
ii, jj, kk, isL, ieL, isU, ieU)

for(ip=0; ip<PEsmptTOT; ip++) {
for(icel=SMPindexG[ip]; icel<SMPindexG[ip+1]; icel++) {

    ic0 = NEWtoOLD[icel];
    icN1 = NEIBcell[ic0-1][0];
    icN2 = NEIBcell[ic0-1][1];
    icN3 = NEIBcell[ic0-1][2];
    icN4 = NEIBcell[ic0-1][3];
    icN5 = NEIBcell[ic0-1][4];
    icN6 = NEIBcell[ic0-1][5];
    VOLO = VOLCEL[ic0];

    isL = indexL[icel ];   ieL = indexL[icel+1];
    isU = indexU[icel ];   ieU = indexU[icel+1];

    if(icN5 != 0) {
        icN5 = OLDtoNEW[icN5-1];
        coef = RDZ * ZAREA;
        D[icel] -= coef;

        if(icN5-1 < icel) {
            for(j=isL; j<ieL; j++) {
                if(itemL[j] == icN5) {
                    AL[j] = coef;
                    break;
                }
            }
        } else {
            for(j=isU; j<ieU; j++) {
                if(itemU[j] == icN5) {
                    AU[j] = coef;
                    break;
                }
            }
        }
    }
}
...

```

**icel: New ID**  
**ic0: Old ID**

## poi\_gen (6/9)

### 新しい番号付けを使用

$$\frac{\phi_{neib[icel][0]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][1]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][2]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][3]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][4]} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib[icel][5]} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0$$

# 係数の計算：並列に実施可能 SMPindexG を使用 private宣言に注意

```
#pragma omp parallel for private  
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6, coef, j, ii,  
jj, kk, isL, ieL, isU, ieU)  
  
for (ip=0; ip<PEsmpTOT; ip++) {  
for (icel=SMPindexG[ip]; icel<SMPindexG[ip+1]; icel++) {  
  
    ic0 = NEWtoOLD[icel];  
    icN1 = NEIBcell[ic0-1][0];
```

```
S1t = omp_get_wtime();
#pragma omp parallel for private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6, coef, j,
ii, jj, kk, isL, ieL, isU, ieU)
```

```
for(ip=0; ip<PEmpTOT; ip++) {
for(icel=SMPindexG[ip]; icel<SMPindexG[ip+1]; icel++) {
```

```
ic0 = NEWtoOLD[icel];
icN1 = NEIBcell[ic0-1][0];
icN2 = NEIBcell[ic0-1][1];
icN3 = NEIBcell[ic0-1][2];
icN4 = NEIBcell[ic0-1][3];
icN5 = NEIBcell[ic0-1][4];
icN6 = NEIBcell[ic0-1][5];
VOLO = VOLCEL[ic0];
```

**icel: New ID**  
**ic0: Old ID**

```
isL = indexL[icel ];   ieL = indexL[icel+1];
isU = indexU[icel ];   ieU = indexU[icel+1];
```

```
if(icN5 != 0) {
icN5 = OLDtoNEW[icN5-1];
coef = RDZ * ZAREA;
D[icel] -= coef;

if(icN5-1 < icel) {
for(j=isL; j<ieL; j++) {
if(itemL[j] == icN5) {
AL[j] = coef;
break;
}
}
} else {
for(j=isU; j<ieU; j++) {
if(itemU[j] == icN5) {
AU[j] = coef;
break;
}
}
}
}
}
```

...

## poi\_gen (6/9)

### New numbering applied

$$\frac{\phi_{neib[icel][0]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][1]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][2]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][3]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][4]} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib[icel][5]} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0$$



```

S1t = omp_get_wtime();
#pragma omp parallel for private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6, coef, j,
ii, jj, kk, isL, ieL, isU, ieU)

for(ip=0; ip<PEsmptOT; ip++) {
for(icel=SMPindexG[ip]; icel<SMPindexG[ip+1]; icel++) {

    ic0 = NEWtoOLD[icel];
    icN1 = NEIBcell[ic0-1][0];
    icN2 = NEIBcell[ic0-1][1];
    icN3 = NEIBcell[ic0-1][2];
    icN4 = NEIBcell[ic0-1][3];
    icN5 = NEIBcell[ic0-1][4];
    icN6 = NEIBcell[ic0-1][5];
    VOLO = VOLCEL[ic0];

    isL = indexL[icel  ];    ieL = indexL[icel+1];
    isU = indexU[icel  ];    ieU = indexU[icel+1];

    if(icN5 != 0) {
        icN5 = OLDtoNEW[icN5-1];
        coef = RDZ * ZAREA;
        D[icel] -= coef;

        if(icN5-1 < icel) {
            for(j=isL; j<ieL; j++) {
                if(itemL[j] == icN5) {
                    AL[j] = coef;
                    break;
                }
            }
        } else {
            for(j=isU; j<ieU; j++) {
                if(itemU[j] == icN5) {
                    AU[j] = coef;
                    break;
                }
            }
        }
    }
}
}
...

```

## poi\_gen (6/9)

### New numbering applied

$$\frac{\phi_{neib[icel][0]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][1]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][2]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][3]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][4]} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib[icel][5]} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0$$

```

S1t = omp_get_wtime();
#pragma omp parallel for private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6, coef, j,
ii, jj, kk, isL, ieL, isU, ieU)

```

```

for(ip=0; ip<PEsmptTOT; ip++) {
for(icel=SMPindexG[ip]; icel<SMPindexG[ip+1]; icel++) {

```

```

ic0 = NEWtoOLD[icel];
icN1 = NEIBcell[ic0-1][0];
icN2 = NEIBcell[ic0-1][1];
icN3 = NEIBcell[ic0-1][2];
icN4 = NEIBcell[ic0-1][3];
icN5 = NEIBcell[ic0-1][4];
icN6 = NEIBcell[ic0-1][5];
VOLO = VOLCEL[ic0];

```

```

isL = indexL[icel ];   ieL = indexL[icel+1];
isU = indexU[icel ];   ieU = indexU[icel+1];

```

```

if(icN5 != 0) {
icN5 = OLDtoNEW[icN5-1];
coef = RDZ * ZAREA;
D[icel] -= coef;

```

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

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

```

if(icN5-1 < icel) {
for(j=isL; j<ieL; j++) {
if(itemL[j] == icN5) {
AL[j] = coef;
break;
}
}
}

```

**icN5 < icel  
Lower Part**

```

} else {
for(j=isU; j<ieU; j++) {
if(itemU[j] == icN5) {
AU[j] = coef;
break;
}
}
}
}
}

```

...

## poi\_gen (6/9)

### New numbering applied

$$\frac{\phi_{neib[icel][0]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][1]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][2]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][3]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][4]} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib[icel][5]} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0$$

```

S1t = omp_get_wtime();
#pragma omp parallel for private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6, coef, j,
ii, jj, kk, isL, ieL, isU, ieU)

```

```

for(ip=0; ip<PEsmptOT; ip++) {
for(icel=SMPindexG[ip]; icel<SMPindexG[ip+1]; icel++) {

```

```

ic0 = NEWtoOLD[icel];
icN1 = NEIBcell[ic0-1][0];
icN2 = NEIBcell[ic0-1][1];
icN3 = NEIBcell[ic0-1][2];
icN4 = NEIBcell[ic0-1][3];
icN5 = NEIBcell[ic0-1][4];
icN6 = NEIBcell[ic0-1][5];
VOLO = VOLCEL[ic0];

```

```

isL = indexL[icel ];   ieL = indexL[icel+1];
isU = indexU[icel ];   ieU = indexU[icel+1];

```

```

if(icN5 != 0) {
icN5 = OLDtoNEW[icN5-1];
coef = RDZ * ZAREA;
D[icel] -= coef;

```

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

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

```

if(icN5-1 < icel) {
for(j=isL; j<ieL; j++) {
if(itemL[j] == icN5) {
AL[j] = coef;
break;
}
}
}

```

```

} else {
for(j=isU; j<ieU; j++) {
if(itemU[j] == icN5) {
AU[j] = coef;
break;
}
}
}
}

```

**icN5 > icel  
Upper Part**

## poi\_gen (6/9)

### New numbering applied

$$\frac{\phi_{neib[icel][0]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][1]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][2]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][3]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][4]} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib[icel][5]} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0$$

```

if(icN3 != 0) {
  icN3 = OLDtoNEW[icN3-1];
  coef = RDY * YAREA;
  D[icel] -= coef;

  if(icN3-1 < icel) {
    for(j=isL; j<ieL; j++) {
      if(itemL[j] == icN3) {
        AL[j] = coef;
        break; }
    }
  } else {
    for(j=isU; j<ieU; j++) {
      if(itemU[j] == icN3) {
        AU[j] = coef;
        break; }
    }
  }
}

if(icN1 != 0) {
  icN1 = OLDtoNEW[icN1-1];
  coef = RDX * XAREA;
  D[icel] -= coef;

  if(icN1-1 < icel) {
    for(j=isL; j<ieL; j++) {
      if(itemL[j] == icN1) {
        AL[j] = coef;
        break;}
    }
  } else {
    for(j=isU; j<ieU; j++) {
      if(itemU[j] == icN1) {
        AU[j] = coef;
        break;}
    }
  }
}
}

```

## poi\_gen (7/9)

$$\begin{aligned}
& \frac{\phi_{neib[icel][0]} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
& \frac{\phi_{neib[icel][1]} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
& \frac{\phi_{neib[icel][2]} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
& \frac{\phi_{neib[icel][3]} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
& \frac{\phi_{neib[icel][4]} - \phi_{icel}}{\Delta z} \Delta x \Delta y + \\
& \frac{\phi_{neib[icel][5]} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0
\end{aligned}$$

```

if(icN2 != 0) {
  icN2 = OLDtoNEW[icN2-1];
  coef = RDX * XAREA;
  D[icel] -= coef;

  if(icN2-1 < icel) {
    for(j=isL; j<ieL; j++) {
      if(itemL[j] == icN2) {
        AL[j] = coef;
        break;}
    }
  } else {
    for(j=isU; j<ieU; j++) {
      if(itemU[j] == icN2) {
        AU[j] = coef;
        break;}
    }
  }
}

if(icN4 != 0) {
  icN4 = OLDtoNEW[icN4-1];
  coef = RDY * YAREA;
  D[icel] -= coef;

  if(icN4-1 < icel) {
    for(j=isL; j<ieL; j++) {
      if(itemL[j] == icN4) {
        AL[j] = coef;
        break; }
    }
  } else {
    for(j=isU; j<ieU; j++) {
      if(itemU[j] == icN4) {
        AU[j] = coef;
        break; }
    }
  }
}
}

```

## poi\_gen (8/9)

$$\begin{aligned}
& \frac{\phi_{neib[icel][0]} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
& \frac{\phi_{neib[icel][1]} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
& \frac{\phi_{neib[icel][2]} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
& \frac{\phi_{neib[icel][3]} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
& \frac{\phi_{neib[icel][4]} - \phi_{icel}}{\Delta z} \Delta x \Delta y + \\
& \frac{\phi_{neib[icel][5]} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0
\end{aligned}$$

```
#pragma omp parallel for private
(ip, icel, ic0, icN1, icN2, icN3, icN4, icN5, icN6,
coef, j, ii, jj, kk, isL, ieL, isU, ieU)
```

```
...
```

```
if(icN6 != 0) {
  icN6 = OLDtoNEW[icN5-1];
  coef = RDZ * ZAREA;
  D[icel] -= coef;

  if(icN6-1 < icel) {
    for(j=isL; j<ieL; j++) {
      if(itemL[j] == icN6) {
        AL[j] = coef;
        break;
      }
    }
  } else {
    for(j=isU; j<ieU; j++) {
      if(itemU[j] == icN6) {
        AU[j] = coef;
        break;
      }
    }
  }
}
```

```
ii = XYZ[ic0][0];
jj = XYZ[ic0][1];
kk = XYZ[ic0][2];
```

```
BFORCE[icel]= -(double) (ii+jj+kk) * VOL0;
```

**BFORCE**  
using original  
mesh ID

**ii,jj,kk,VOL0:**  
private

```
}
```

## poi\_gen (9/9)

$$\frac{\phi_{neib[icel][0]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][1]} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib[icel][2]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][3]} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib[icel][4]} - \phi_{icel}}{\Delta z} \Delta x \Delta y +$$

$$\frac{\phi_{neib[icel][5]} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0$$

# Main Program

```

#include <stdio.h> ...

int
main()
{
    double *WK;
    int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
    double xN, xL, xU; Stime, Etime;

    if(INPUT()) goto error;
    if(POINTER_INIT()) goto error;
    if(BOUNDARY_CELL()) goto error;
    if(CELL_METRICS()) goto error;
    if(POI_GEN()) goto error;

    ISET = 0;
    WK = (double *)malloc(sizeof(double)*ICELTOT);
    if(WK == NULL) {
        fprintf(stderr, "Error: %s\n", strerror(errno));
        goto error;
    }
    Stime = omp_get_wtime();
    if(solve_ICCG_mc(ICELTOT, NL, NU, indexL, itemL, indexU, itemU,
                    D, BFORCE, PHI, AL, AU, NCOLORTot, PEsmptOT,
                    SMPindex, SMPindexG, EPSICCG, &ITR, &IER)) goto error;
    Etime = omp_get_wtime();
    for(ic0=0; ic0<ICELTOT; ic0++) {
        icel = NEWtoOLD[ic0];
        WK[icel-1] = PHI[ic0];
    }
    for(icel=0; icel<ICELTOT; icel++) {
        PHI[icel] = WK[icel];
    }
    if(OUTUCD()) goto error;
    return 0;
error:
    return -1;
}

```

# solve\_ICCG\_mc (1/6)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <math.h> etc.

#include "solver_ICCG.h"

extern int
solve_ICCG_mc(int N, int NL, int NU, int *indexL, int *itemL, int *indexU,
              int *itemU,
              double *D, double *B, double *X, double *AL, double *AU,
              int NCOLORTot, int *COLORindex,
              int PEsmptTOT, int *SMPindex, int *SMPindexG,
              double EPS, int *ITR, int *IER)
{
    double **W;
    double VAL, BNRM2, WVAL, SW, RHO, BETA, RHO1, C1, DNRM2, ALPHA, ERR;
    int i, j, ic, ip, L, ip1;
    int R = 0;
    int Z = 1;
    int Q = 1;
    int P = 2;
    int DD = 3;
```



# solve\_ICCG\_mc (2/6)

```

W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
    fprintf(stderr, "Error: %s\n", strerror(errno));
    return -1;
}

for(i=0; i<4; i++) {
    W[i] = (double *)malloc(sizeof(double)*N);
    if(W[i] == NULL) {
        fprintf(stderr, "Error: %s\n",
            strerror(errno)); return -1;
    }
}

#pragma omp parallel for private (ip, i)
for(ip=0; ip<PEsmptOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        X[i] = 0.0;
        W[1][i] = 0.0;
        W[2][i] = 0.0;
        W[3][i] = 0.0;
    }
}

for(ic=0; ic<NCOLORtot; ic++) {
    #pragma omp parallel for private (ip, ip1, i, VAL, j)
    for(ip=0; ip<PEsmptOT; ip++) {
        ip1 = ic * PEsmptOT + ip;
        for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
            VAL = D[i];
            for(j=indexL[i]; j<indexL[i+1]; j++) {
                VAL -= AL[j]*AL[j]*W[DD][itemL[j] - 1];
            }
            W[DD][i] = 1.0 / VAL;
        }
    }
}

```

Incomplete "Modified"  
Cholesky  
Factorization

# Incomplete “Modified” Cholesky Factorization

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

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

```

for (i=0; i<N; i++) {
    VAL = D[i];
    for (j=indexL[i]; j<indexL[i+1]; j++) {
        VAL -= AL[j]*AL[j]*W[DD][itemL[j] - 1];
    }
    W[DD][i] = 1.0 / VAL;
}

```

# Incomplete “Modified” Cholesky Factorization: Parallel Version

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

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

```

for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for private (ip, ip1, i, VAL, j)
    for(ip=0; ip<PEsmpTOT; ip++) {
        ip1 = ic * PEsmpTOT + ip;
        for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
            VAL = D[i];
            for(j=indexL[i]; j<indexL[i+1]; j++) {
                VAL -= AL[j]*AL[j]*W[DD][itemL[j] - 1];
            }
            W[DD][i] = 1.0 / VAL;
        }
    }
}

```

privateに注意。

# solve\_ICCG\_mc (3/6)

```

#pragma omp parallel for private (ip, i, VAL, j)
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        VAL = D[i] * X[i];

        for(j=indexL[i]; j<indexL[i+1]; j++) {
            VAL += AL[j] * X[itemL[j]-1];
        }
        for(j=indexU[i]; j<indexU[i+1]; j++) {
            VAL += AU[j] * X[itemU[j]-1];
        }
    }
    W[R][i] = B[i] - VAL;
}

BNRM2 = 0.0;
#pragma omp parallel for private (ip, i)
                    reduction (+:BNRM2)
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        BNRM2 += B[i]*B[i];
    }
}

```

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

```

for i= 1, 2, ...
    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

```

# Mat-Vec

## NO Data Dependency: SMPindexG

```
#pragma omp parallel for private (ip, i, VAL, j)
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        VAL = D[i] * X[i];

        for(j=indexL[i]; j<indexL[i+1]; j++) {
            VAL += AL[j] * X[itemL[j]-1];
        }
        for(j=indexU[i]; j<indexU[i+1]; j++) {
            VAL += AU[j] * X[itemU[j]-1];
        }
    }
    W[R][i] = B[i] - VAL;
}
```

# solve\_ICCG\_mc (3/6)

```

#pragma omp parallel for private (ip, i, VAL, j)
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        VAL = D[i] * X[i];

        for(j=indexL[i]; j<indexL[i+1]; j++) {
            VAL += AL[j] * X[itemL[j]-1];
        }
        for(j=indexU[i]; j<indexU[i+1]; j++) {
            VAL += AU[j] * X[itemU[j]-1];
        }
    }
    W[R][i] = B[i] - VAL;
}

BNRM2 = 0.0;
#pragma omp parallel for private (ip, i)
                        reduction (+:BNRM2)
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        BNRM2 += B[i]*B[i];
    }
}

```

```

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

```

# Dot Products: SMPindexG, reduction

```
BNRM2 = 0.0;
#pragma omp parallel for private (ip,i)
                        reduction (+:BNRM2)
for (ip=0; ip<PEsmpTOT; ip++) {
    for (i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        BNRM2 += B[i]*B[i];
    }
}
```

```

*ITR = N;
for(L=0; L<(*ITR); L++) {

#pragma omp parallel for private(ip, i)
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        W[Z][i] = W[R][i];
    }
}

for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for private (ip, ip1, i, VAL, 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];
    }
}
}

for(ic=NCOLORtot-1; ic>=0; ic--) {
#pragma omp parallel for private (ip, ip1, i, VAL, j)
for(ip=0; ip<PEsmpTOT; ip++) {
    ip1 = ic * PEsmpTOT + ip;
    for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
        SW = 0.0;
        for(j=indexU[i]; j<indexU[i+1]; j++) {
            SW += AU[j] * W[Z][itemU[j]-1];
        }
        W[Z][i] = W[Z][i] - W[DD][i] * SW;
    }
}
}
}
}

```

# solve\_ICCG\_mc (4/6)

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



```

*ITR = N;
for(L=0; L<(*ITR); L++) {

#pragma omp parallel for private(ip, i)
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        W[Z][i] = W[R][i];
    }
}

SMPindex

for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for private (ip, ip1, i, VAL, 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];
    }
}
}

for(ic=NCOLORtot-1; ic>=0; ic--) {
#pragma omp parallel for private (ip, ip1, i, VAL, j)
for(ip=0; ip<PEsmpTOT; ip++) {
    ip1 = ic * PEsmpTOT + ip;
    for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
        SW = 0.0;
        for(j=indexU[i]; j<indexU[i+1]; j++) {
            SW += AU[j] * W[Z][itemU[j]-1];
        }
        W[Z][i] = W[Z][i] - W[DD][i] * SW;
    }
}
}

```

# 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;
for(L=0; L<(*ITR); L++) {

#pragma omp parallel for private(ip, i)
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        W[Z][i] = W[R][i];
    }
}

```

**SMPindex**

```

for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for private (ip, ip1, i, VAL, 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];
    }
}

```

```

for(ic=NCOLORtot-1; ic>=0; ic--) {
#pragma omp parallel for private (ip, ip1, i, VAL, j)
for(ip=0; ip<PEsmpTOT; ip++) {
    ip1 = ic * PEsmpTOT + ip;
    for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
        SW = 0.0;
        for(j=indexU[i]; j<indexU[i+1]; j++) {
            SW += AU[j] * W[Z][itemU[j]-1];
        }
        W[Z][i] = W[Z][i] - W[DD][i] * SW;
    }
}
}

```

# solve\_ICCG\_mc (4/6)

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

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

Forward Substitution

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

Backward Substitution

# Forward Substitution: SMPindex

```
for(ic=0; ic<NCOLORtot; ic++) {  
#pragma omp parallel for private (ip, ip1, i, VAL, 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];  
    }  
}  
}
```

# solve\_ICCG\_mc

## (5/6)

```

/*****
* {p} = {z} if ITER=0 *
* BETA = RHO / RH01 otherwise *
*****/

if(L == 0) {
#pragma omp parallel for private(ip, i)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      W[P][i] = W[Z][i];
    }
  }
} else {
  BETA = RHO / RH01;
#pragma omp parallel for private(ip, i)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      W[P][i] = W[Z][i] + BETA * W[P][i];
    }
  }
}

/*****
* {q} = [A] {p} *
*****/

#pragma omp parallel for private(ip, i)
for(ip=0; ip<PEsmpTOT; ip++) {
  for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
      VAL += AL[j] * W[P][itemL[j]-1];
    }
    for(j=indexU[i]; j<indexU[i+1]; j++) {
      VAL += AU[j] * W[P][itemU[j]-1];
    }
    W[Q][i] = VAL;
  }
}

```

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

for  $i = 1, 2, \dots$

  solve  $[M]z^{(i-1)} = r^{(i-1)}$

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

if  $i=1$

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

else

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

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

endif

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

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

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

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

  check convergence  $|r|$

end

# solve\_ICCG\_mc

## (5/6)

```

/*****
* {p} = {z} if ITER=0 *
* BETA = RHO / RH01 otherwise *
*****/

if(L == 0) {
#pragma omp parallel for private(ip, i)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      W[P][i] = W[Z][i];
    }
  }
} else {
  BETA = RHO / RH01;
#pragma omp parallel for private(ip, i)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      W[P][i] = W[Z][i] + BETA * W[P][i];
    }
  }
}

/*****
* {q} = [A] {p} *
*****/

#pragma omp parallel for private(ip, i)
for(ip=0; ip<PEsmpTOT; ip++) {
  for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
      VAL += AL[j] * W[P][itemL[j]-1];
    }
    for(j=indexU[i]; j<indexU[i+1]; j++) {
      VAL += AU[j] * W[P][itemU[j]-1];
    }
    W[Q][i] = VAL;
  }
}

```

```

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

```

# solve\_ICCG\_mc (6/6)

```

/*****
* ALPHA = RHO / {p} {q} *
*****/
C1 = 0.0;
#pragma omp parallel for private(ip, i) reduction(+:C1)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      C1 += W[P][i] * W[Q][i];
    }
  }
ALPHA = RHO / C1;

/*****
* {x} = {x} + ALPHA * {p} *
* {r} = {r} - ALPHA * {q} *
*****/
#pragma omp parallel for private(ip, i)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      X[i] += ALPHA * W[P][i];
      W[R][i] -= ALPHA * W[Q][i];
    }
  }

DNRM2 = 0.0;
#pragma omp parallel for private(ip, i)
  reduction(+:DNRM2)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      DNRM2 += W[R][i]*W[R][i];
    }
  }

```

```

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

```

# solve\_ICCG\_mc (6/6)

```

/*****
* ALPHA = RHO / {p} {q} *
*****/
C1 = 0.0;
#pragma omp parallel for private(ip, i) reduction(+:C1)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      C1 += W[P][i] * W[Q][i];
    }
  }
ALPHA = RHO / C1;

/*****
* {x} = {x} + ALPHA * {p} *
* {r} = {r} - ALPHA * {q} *
*****/
#pragma omp parallel for private(ip, i)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      X[i] += ALPHA * W[P][i];
      W[R][i] -= ALPHA * W[Q][i];
    }
  }

DNRM2 = 0.0;
#pragma omp parallel for private(ip, i)
  reduction(+:DNRM2)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      DNRM2 += W[R][i]*W[R][i];
    }
  }

```

```

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

```

# solve\_ICCG\_mc (6/6)

```

/*****
* ALPHA = RHO / {p} {q} *
*****/
C1 = 0.0;
#pragma omp parallel for private(ip, i) reduction(+:C1)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      C1 += W[P][i] * W[Q][i];
    }
  }
ALPHA = RHO / C1;

/*****
* {x} = {x} + ALPHA * {p} *
* {r} = {r} - ALPHA * {q} *
*****/
#pragma omp parallel for private(ip, i)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      X[i] += ALPHA * W[P][i];
      W[R][i] -= ALPHA * W[Q][i];
    }
  }

DNRM2 = 0.0;
#pragma omp parallel for private(ip, i)
  reduction(+:DNRM2)
  for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
      DNRM2 += W[R][i]*W[R][i];
    }
  }

```

```

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

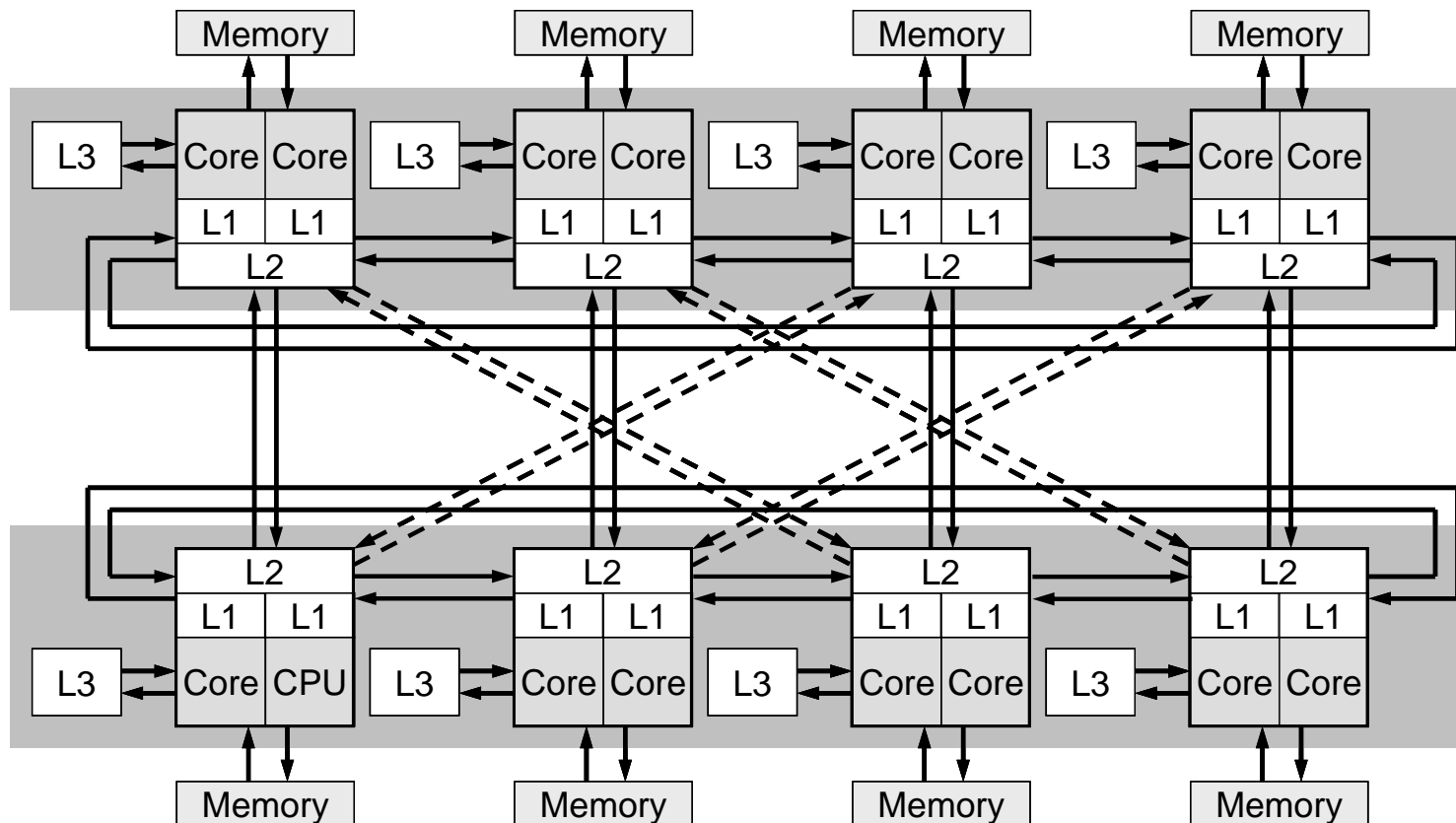
```



- L2-solへのOpenMPの実装
- 実行例
- 最適化＋演習

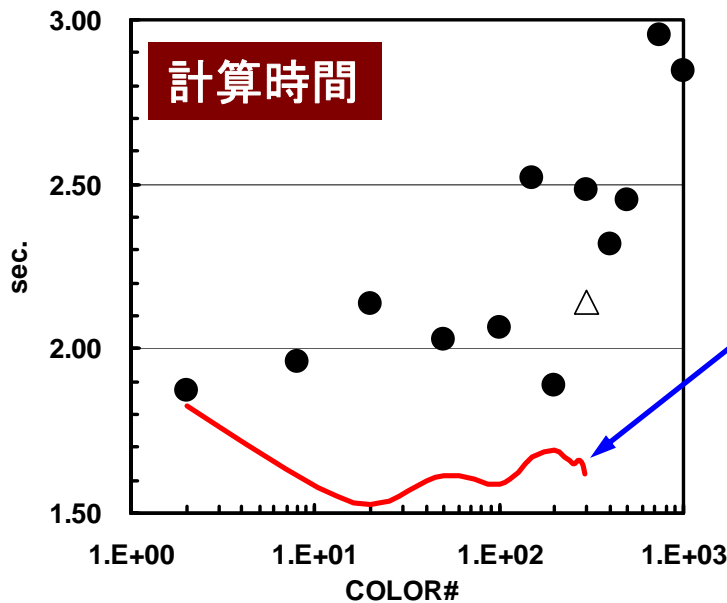
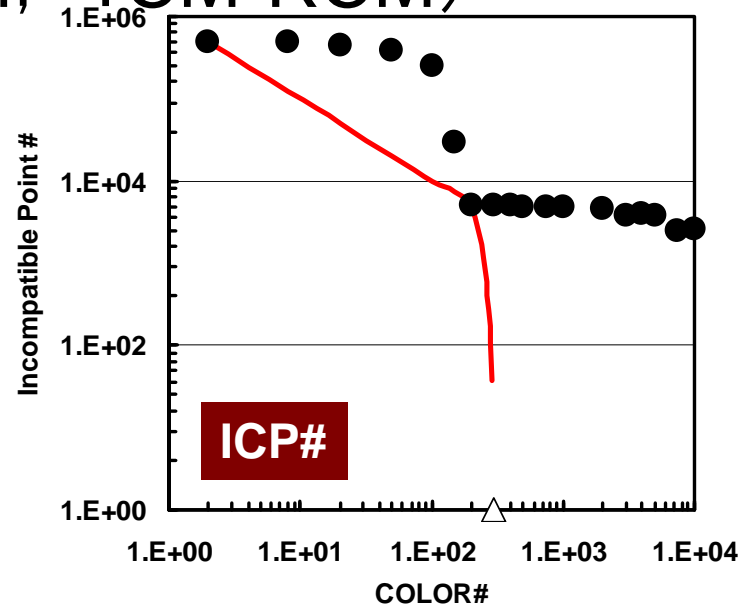
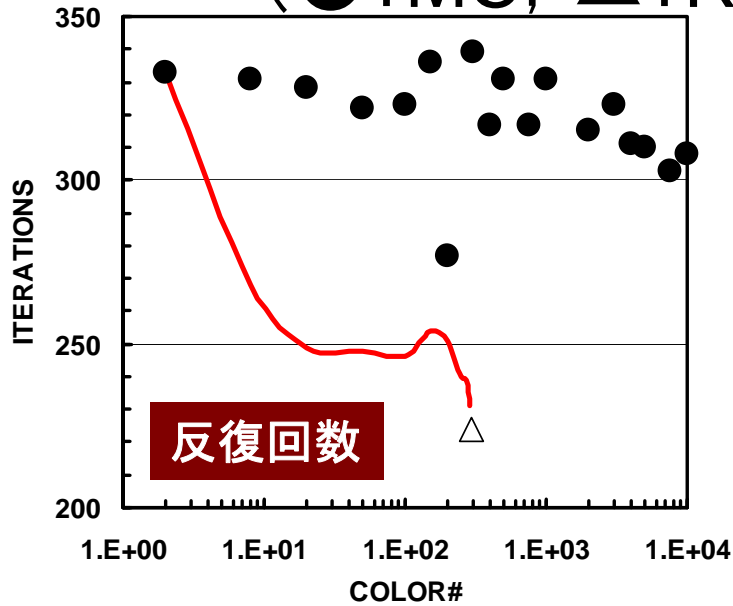
# 計算結果

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

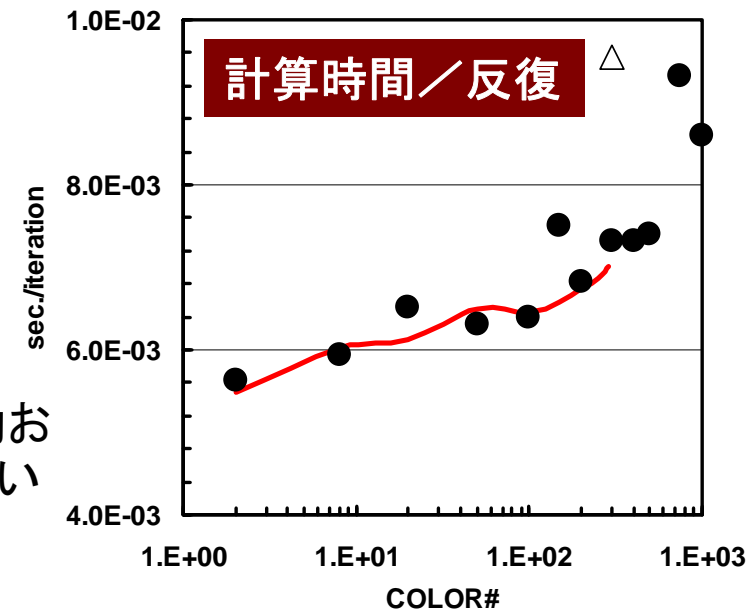


# SR11000, 16コアにおける結果, $100^3$

(●:MC, △:RCM, -:CM-RCM)

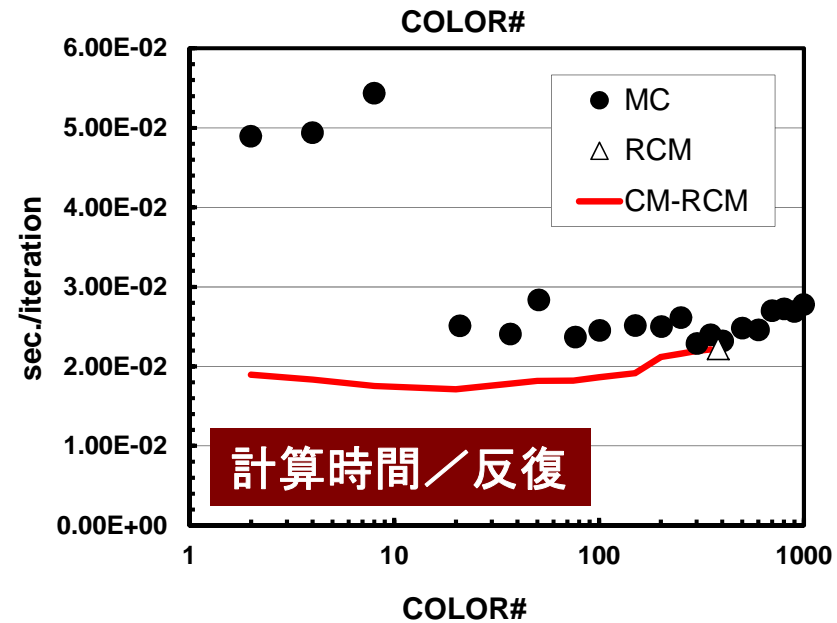
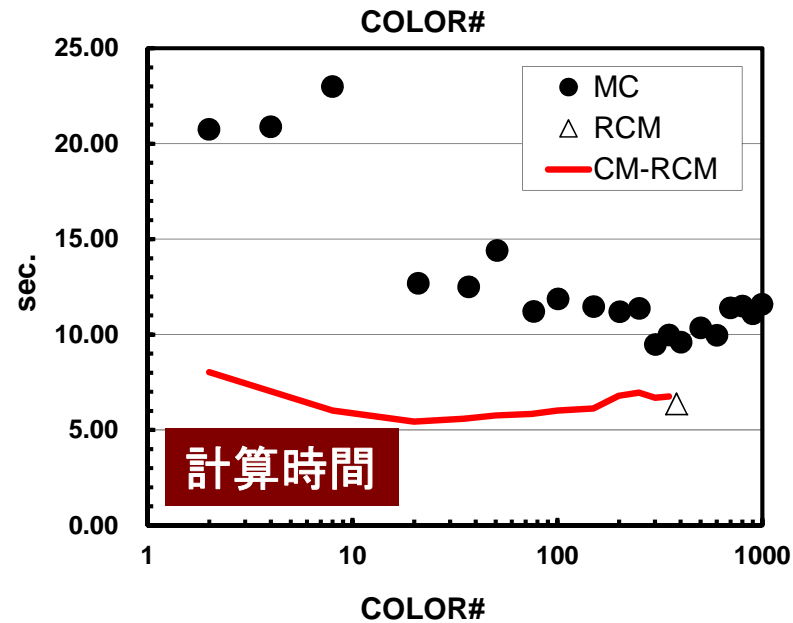
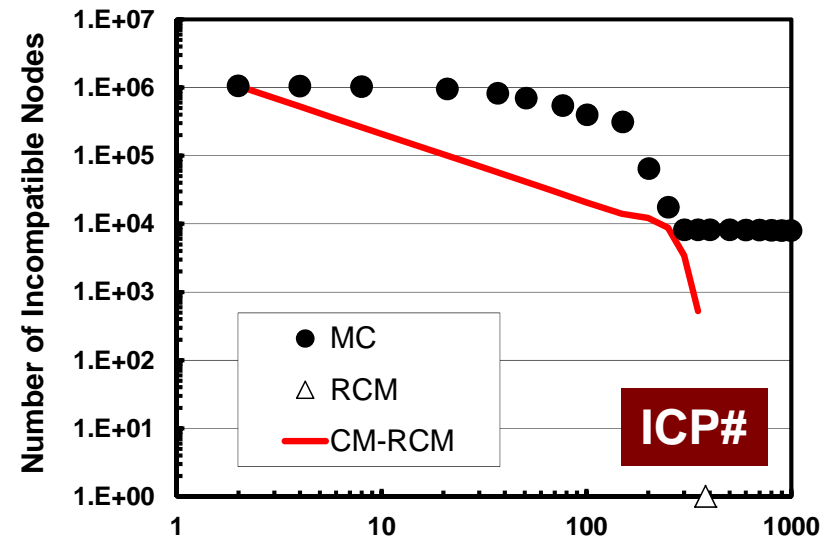
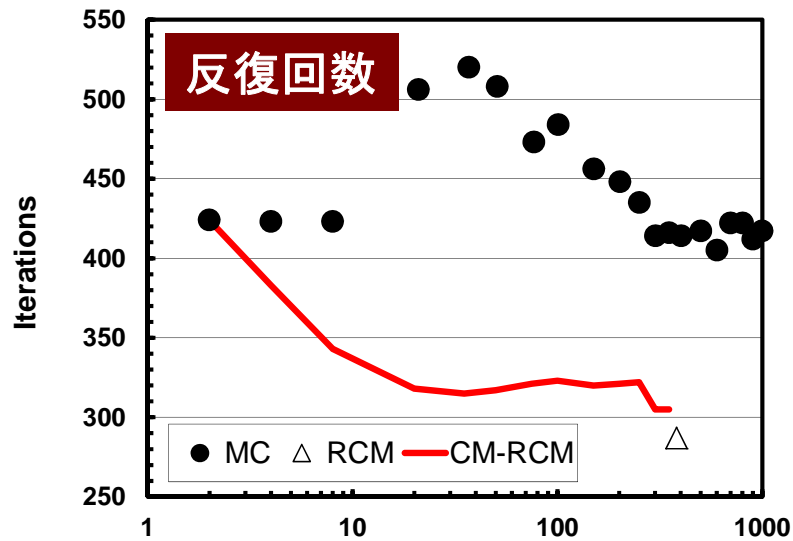


挙動おかしい



# FX10, 16コアにおける結果, $128^3$

(●:MC, △:RCM, -:CM-RCM)



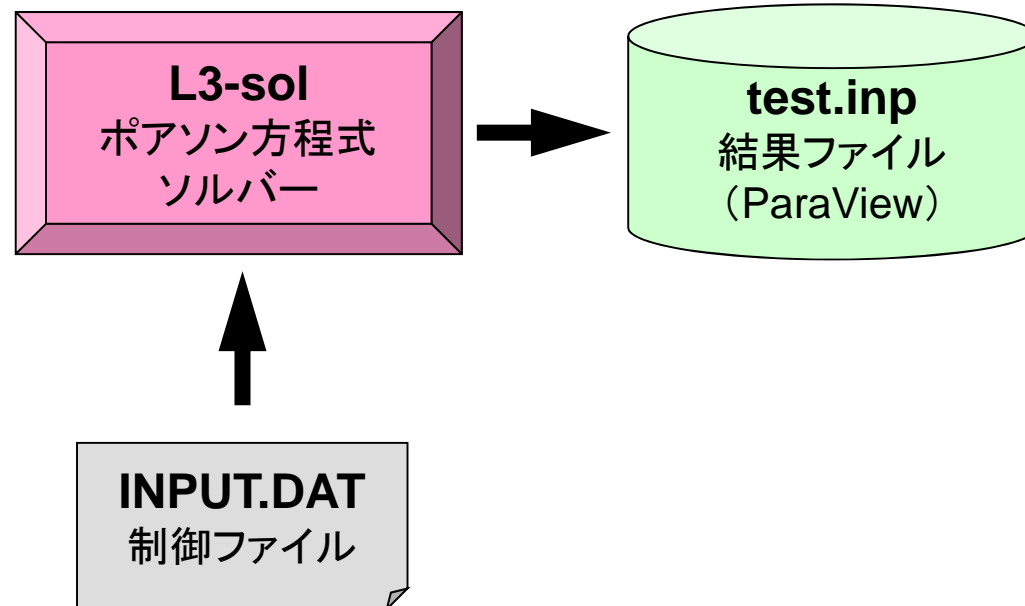
- L2-solへのOpenMPの実装
- 実行例
- 最適化＋演習

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

# Running L3-sol





# 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

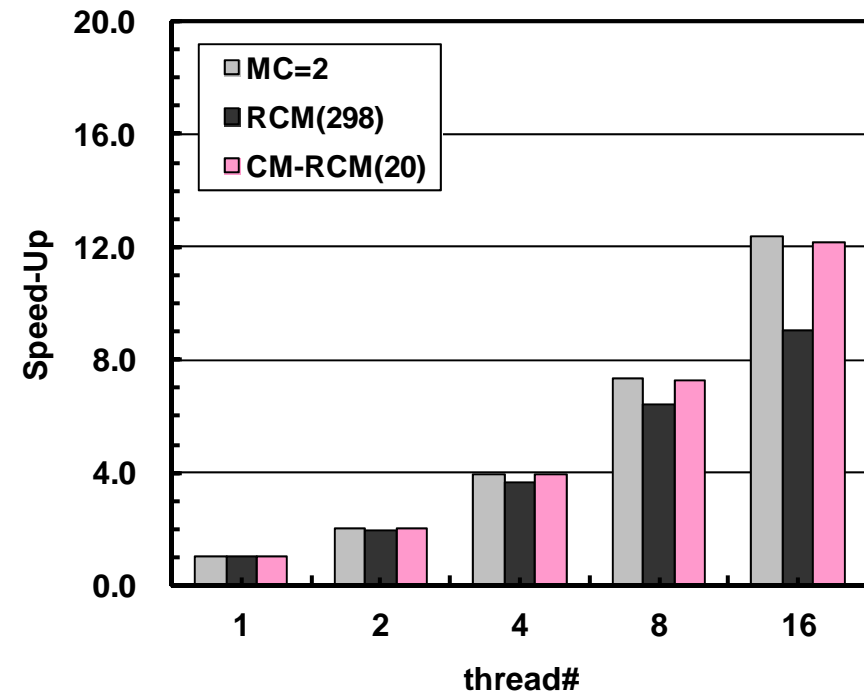
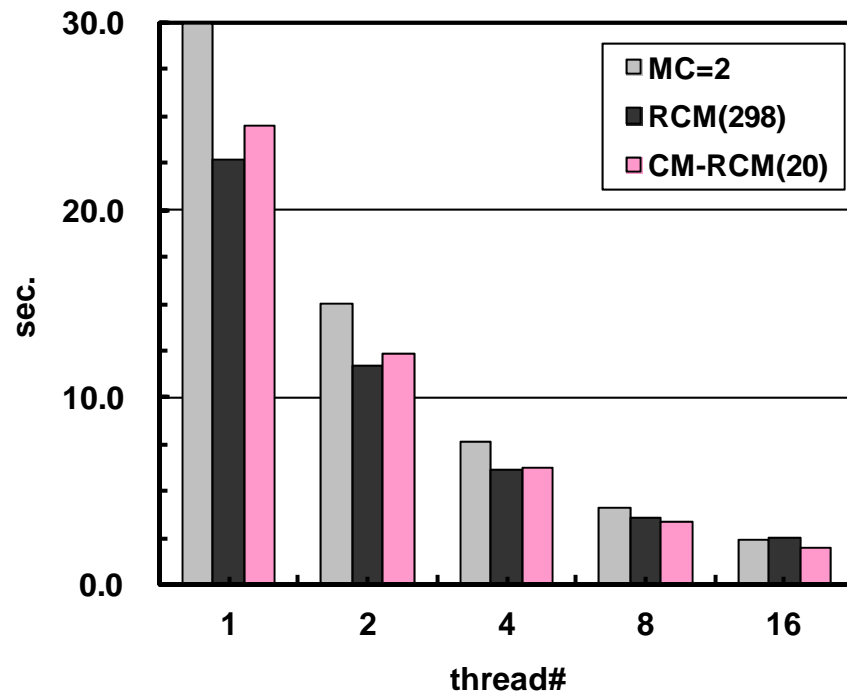
```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:10:00"
#PJM -L "rscgrp=small"
#PJM -j
#PJM -o "test.lst"
export OMP_NUM_THREADS=16
./L3-sol
```

標準出力ファイル名

スレッド数, 通常 =PEsmpTOT

# 計算結果 (FX10@東大): $10^6$ 要素

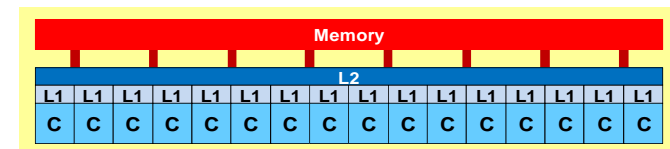
反復回数: MC (2色): 333回, RCM (298レベル): 224回  
 CM-RCM ( $N_c=20$ ): 249回



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(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(C)

```
#pragma omp parallel private (ic, 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<sup>3</sup>

	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

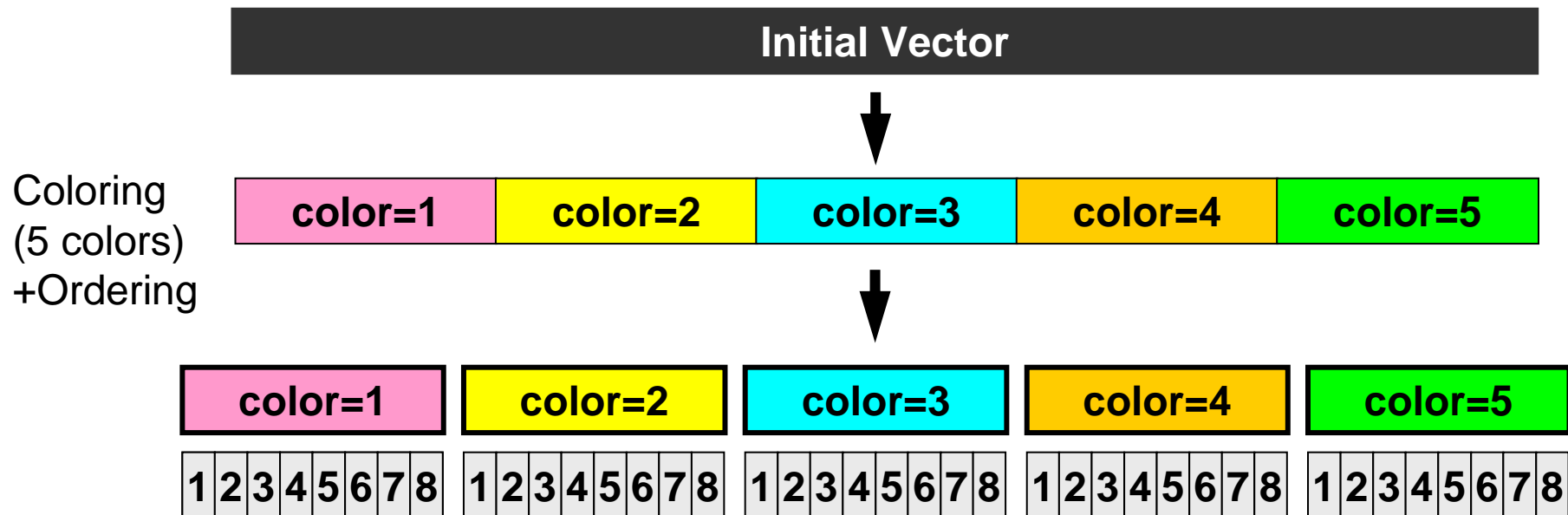
- 色付け
  - MC
  - RCM
  - CM-RCM
- 同じ色に属する要素は独立：並列計算可能
- 「色」の順番に番号付け
- 色内の要素を各スレッドに振り分ける
  
- 同じスレッド(すなわち同じコア)に属する要素は連続の番号ではない
  - 効率の低下

# SMPindex: 前処理向け

```

for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for ...
    for(ip=0; ip<PEsmpTOT; ip++) {
        ip1 = ic * PEsmpTOT + ip;
        for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
            (...)
        }
    }
}

```



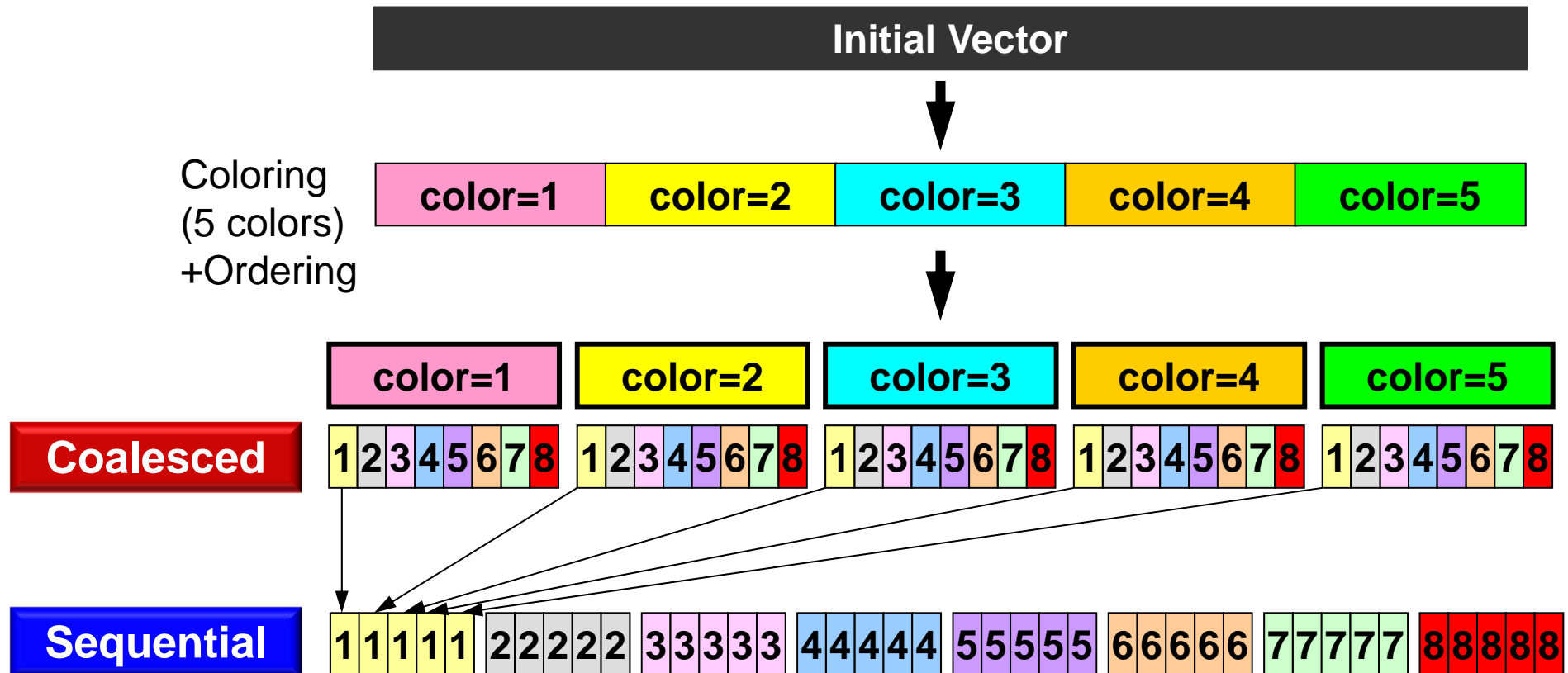
- 5色, 8スレッドの例
- 同じ「色」に属する要素は独立⇒並列計算可能
- 色の順番に並び替え

# 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

1<sup>st</sup>-Color

■ #0 thread, ■ #1, ■ #2, ■ #3

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

CM-RCM(2)



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

Sequential Reordering, 4-threads



# データ再配置: Sequential Reordering

CM-RCM(2), 4-threads

2<sup>nd</sup>-Color

■ #0 thread, ■ #1, ■ #2, ■ #3

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

CM-RCM(2)

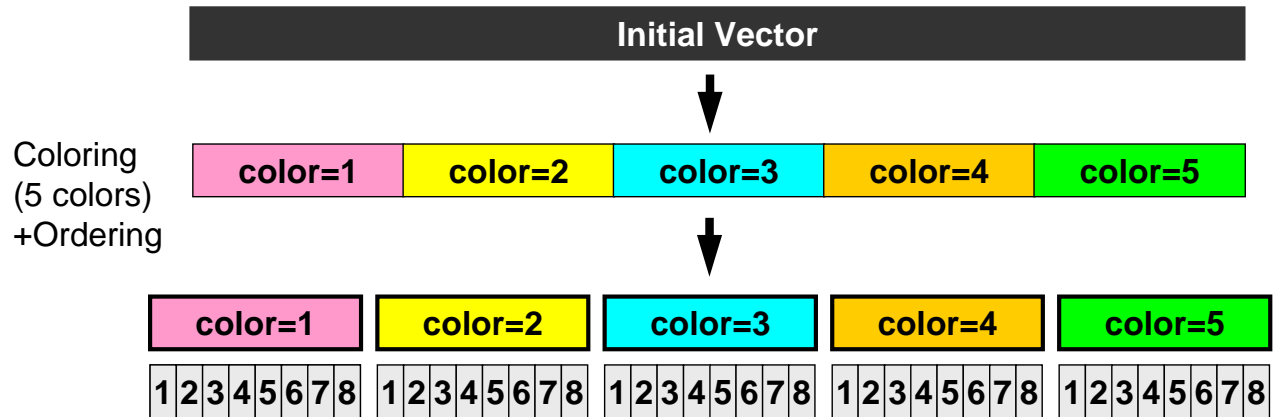


29	18	15	5	11	2	9	1
33	30	19	16	6	12	3	10
45	34	31	20	25	7	13	4
40	46	35	32	21	26	8	14
59	49	47	36	41	22	27	17
53	60	50	48	37	42	23	28
63	54	61	51	57	38	43	24
56	64	55	62	52	58	39	44

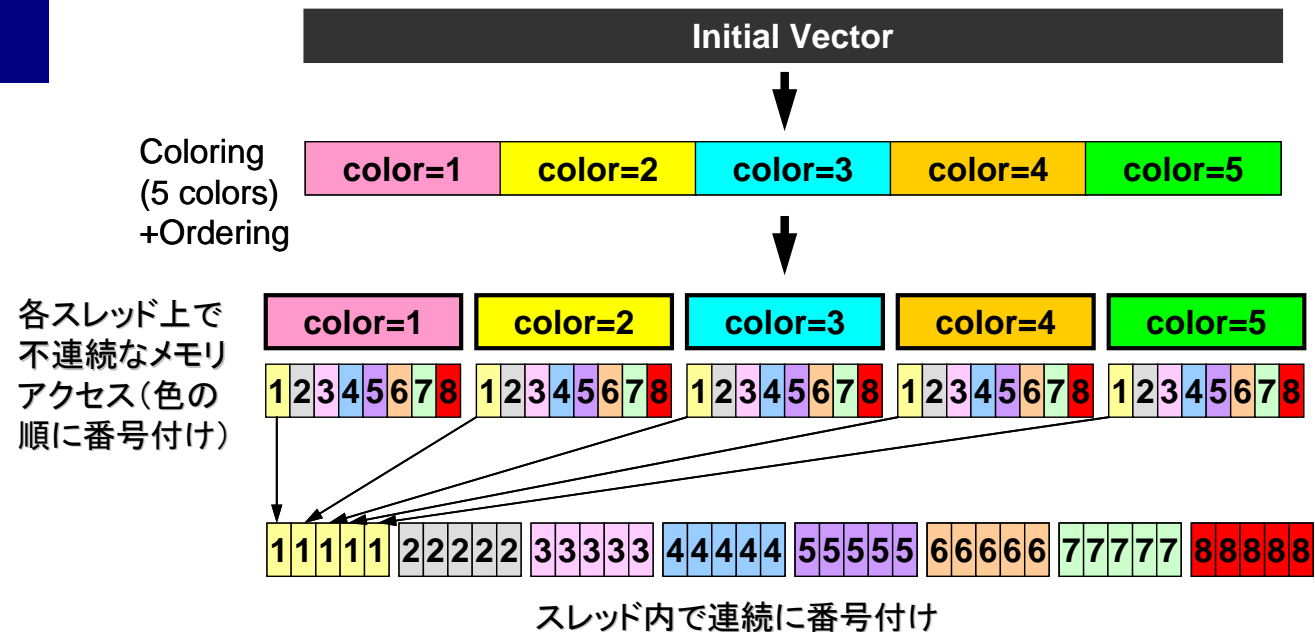
Sequential Reordering, 4-threads

# データ再配置: Sequential Reordering

**Coalesced  
Good for GPU**



**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

```

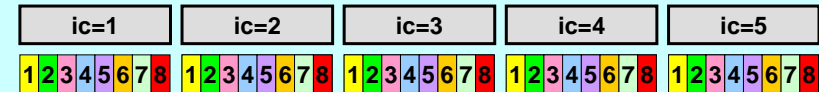
変数名	型	内容
PEsmptTOT	整数	データ分割数
NCOLORtot	整数	Ordering手法と色数 $\geq 2$ : MC法 (multicolor) , 色数 $= 0$ : CM法 (Cuthill-Mckee) $= -1$ : RCM法 (Reverse Cuthill-Mckee) $\leq -2$ : CM-RCM法
NFLAG	整数	0 : First-Touch無し, 1 : あり 今回は無関係
METHOD	整数	行列ベクトル積のループ構造 (0 : 従来通り, 1 : 前進後退代入と同じ)

# Sequential Reordering

```
SMPindex = (int *) allocate_vector(sizeof(int), NCOLORTot * PEsmptTOT + 1);
memset(SMPindex, 0, sizeof(int)*(NCOLORTot*PEsmptTOT+1));
```

```
for(ic=1; ic<=NCOLORTot; ic++) {
    nn1 = COLORindex[ic] - COLORindex[ic-1];
    num = nn1 / PEsmptTOT;
    nr = nn1 - PEsmptTOT * num;
    for(ip=1; ip<=PEsmptTOT; ip++) {
        if(ip <= nr) {
            SMPindex[(ic-1)*PEsmptTOT+ip] = num + 1;
        } else {
            SMPindex[(ic-1)*PEsmptTOT+ip] = num;
        }
    }
}
```

## SMPindex



## SMPindex\_new



```
SMPindex_new = (int *) allocate_vector(sizeof(int), NCOLORTot * PEsmptTOT + 1);
memset(SMPindex_new, 0, sizeof(int)*(NCOLORTot*PEsmptTOT+1));
```

```
for(ic=1; ic<=NCOLORTot; ic++) {
    for(ip=1; ip<=PEsmptTOT; ip++) {
        j1 = (ic-1)*PEsmptTOT + ip;
        j0 = j1-1;
        SMPindex_new[(ip-1)*NCOLORTot+ic] = SMPindex[j1];
        SMPindex[j1] = SMPindex[j0] + SMPindex[j1];
    }
}
```

```
for(ip=1; ip<=PEsmptTOT; ip++) {
    for(ic=1; ic<=NCOLORTot; ic++) {
        j1 = (ip-1) * NCOLORTot + ic;
        j0 = j1 - 1;
        SMPindex_new[j1] += SMPindex_new[j0];
    }
}
```

# 行列ベクトル積の計算法

```
#pragma omp parallel for private(ip, i)
for(ip=0; ip<PEsmpTOT; ip++) {
  for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
      VAL += AL[j] * W[P][itemL[j]-1];
    }
    for(j=indexU[i]; j<indexU[i+1]; j++) {
      VAL += AU[j] * W[P][itemU[j]-1];
    }
    W[Q][i] = VAL;
  }
}
```

**METHOD=0**

```
#pragma omp parallel for private (ip1, i, VAL, j)
for(ip=0; ip<PEsmpTOT; ip++) {
  for(i=SMPindex[ip*NCOLORtot]; i<SMPindex[(ip+1)*NCOLORtot]; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
      VAL += AL[j] * W[P][itemL[j]-1];
    }
    for(j=indexU[i]; j<indexU[i+1]; j++) {
      VAL += AU[j] * W[P][itemU[j]-1];
    }
    W[Q][i] = VAL;
  }
}
```

**METHOD=1**

# 前進代入の計算法：色ループは外

```

for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for private (ip, ip1, i, VAL, 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];
    }
  }
}

```

Original

```

for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for private (ip1, i, WVAL, j)
  for(ip=0; ip<PEsmpTOT; ip++) {
    ip1 = ip * NCOLORtot + ic;
    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];
    }
  }
}

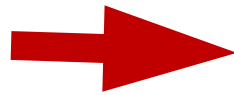
```

New

# 行列格納方法

**ELL (Ellpack-ltpack):** 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

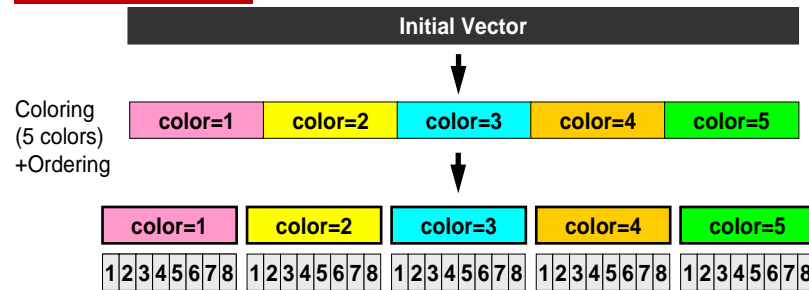
(b) ELL



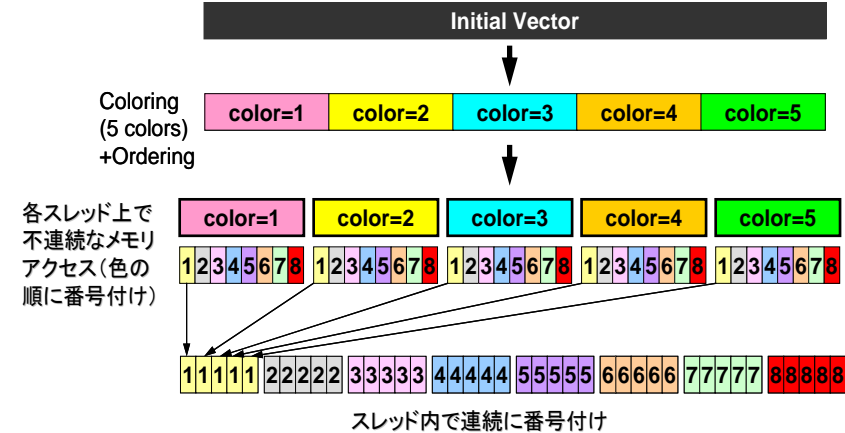
# Cases: $128^3$ meshes

	Coloring	Further Reordering	First Touch Data Placement	Matrix Storage Format
<b>src0</b>	Case-1	CM-RCM	Coalesced ( $\boxtimes$ 4 (a))	CRS
<b>reorder0</b>	Case-2		Sequential ( $\boxtimes$ 4 (b))	
<b>ELL</b>	Case-3			ELL

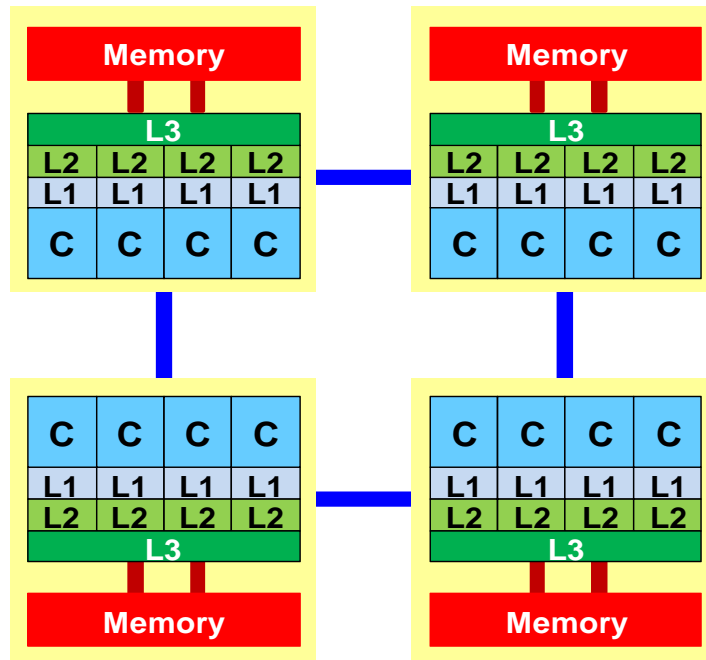
## Coalesced



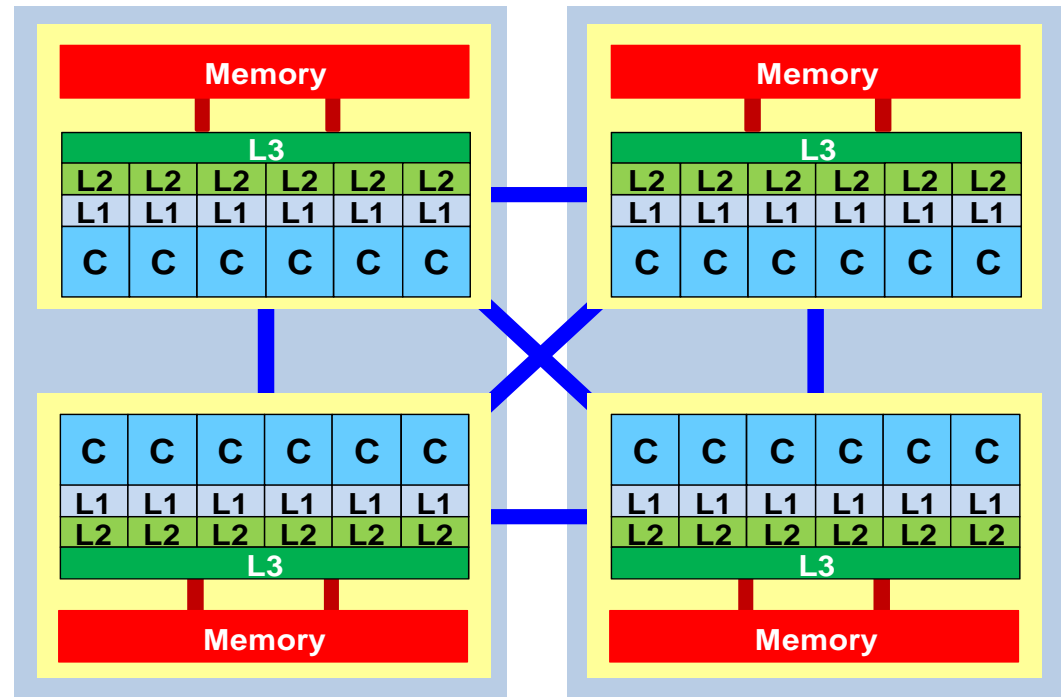
## Sequential



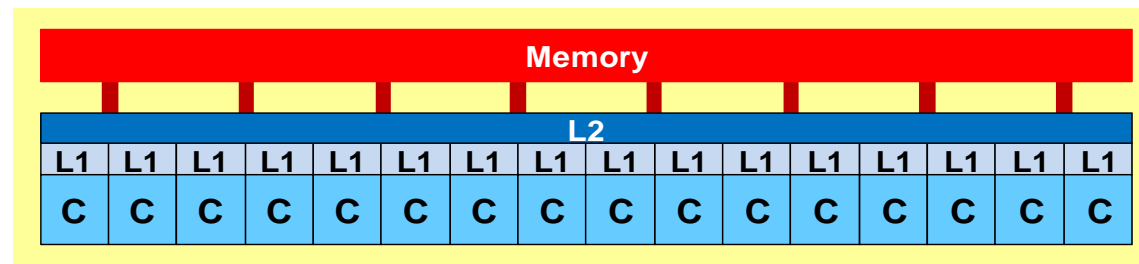
## T2K/Tokyo



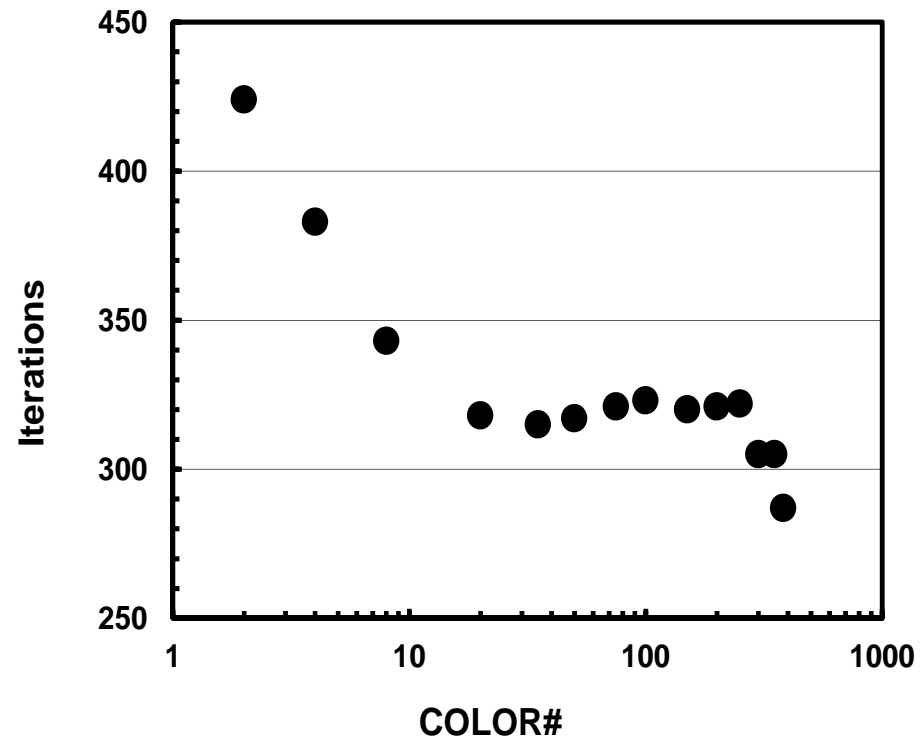
## Cray XE6 (Hopper)



## Fujitsu FX10 (Oakleaf-FX)

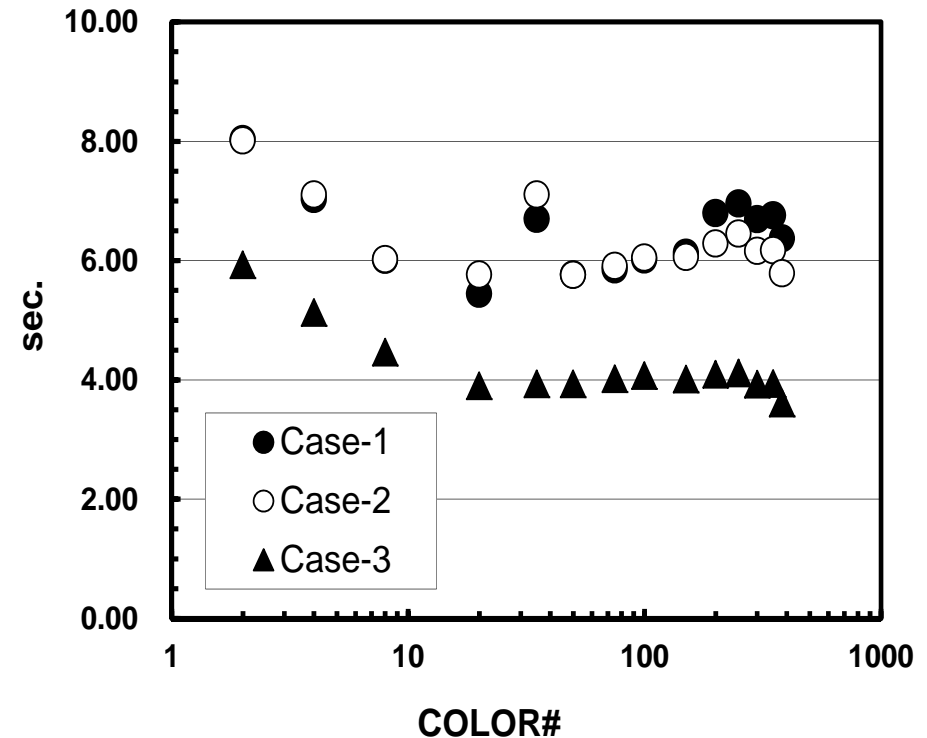


# 計算結果：色数～反復回数 (CM-RCM)



# 計算結果：色数～計算時間：FX10

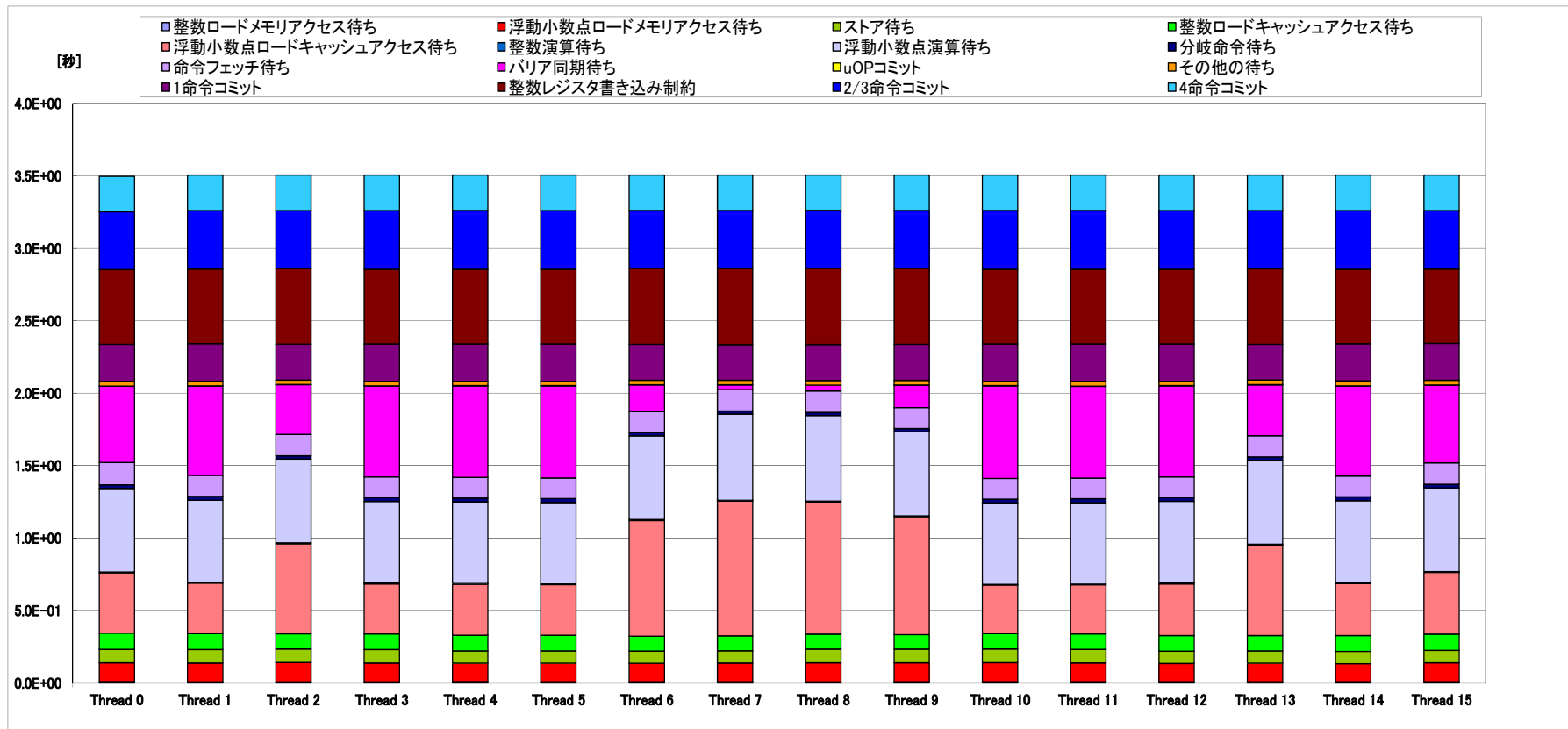
- CASE-1(src0)⇒  
CASE-2(reorder0)
  - 色数が大きくなると多少効果あり
  - 色数が多くなるほど、色当たり、スレッド当たりの計算量は減少
    - CASE-2は色が変わってもスレッド上のデータが連続
  - First Touchの効果はナシ
- ELLの効果は大



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

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

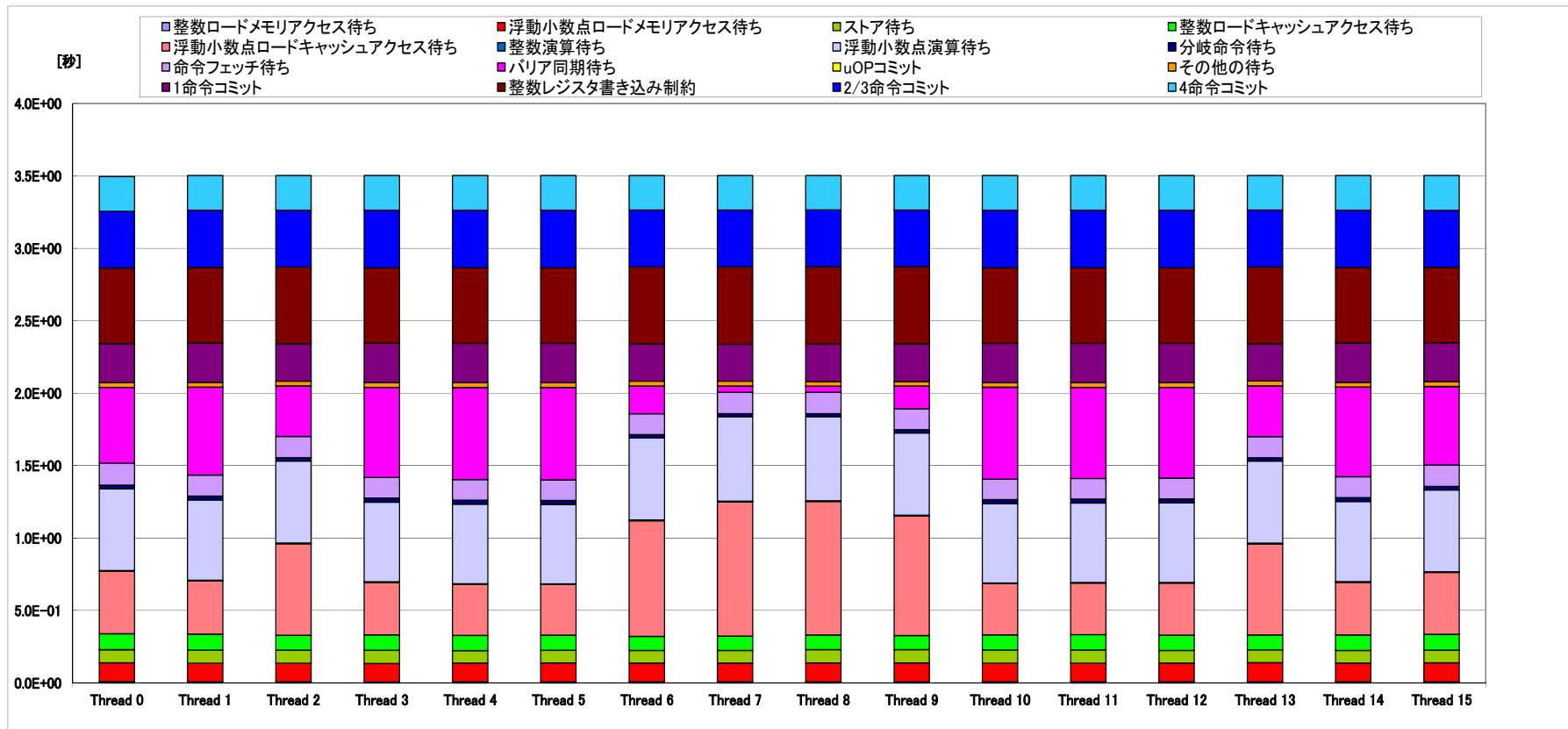
L1デマンドミス: 25.6%, Mem. Throughput: 41.8GB/sec.  
前処理部分(前進後退代入)



**src0: CRS, Coalesced**

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

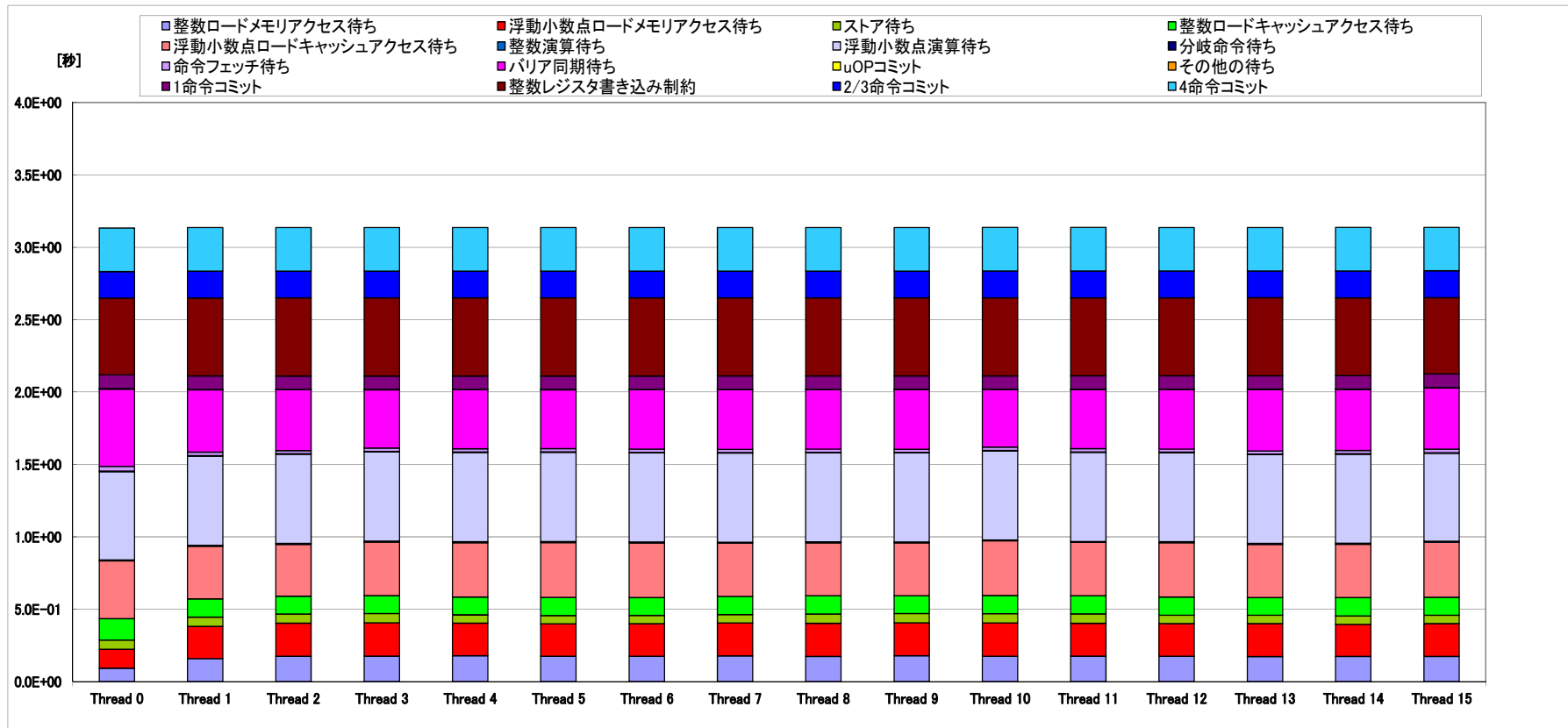
## 25.6%, 41.8GB/sec.



**reorder0: CRS, Sequential**

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

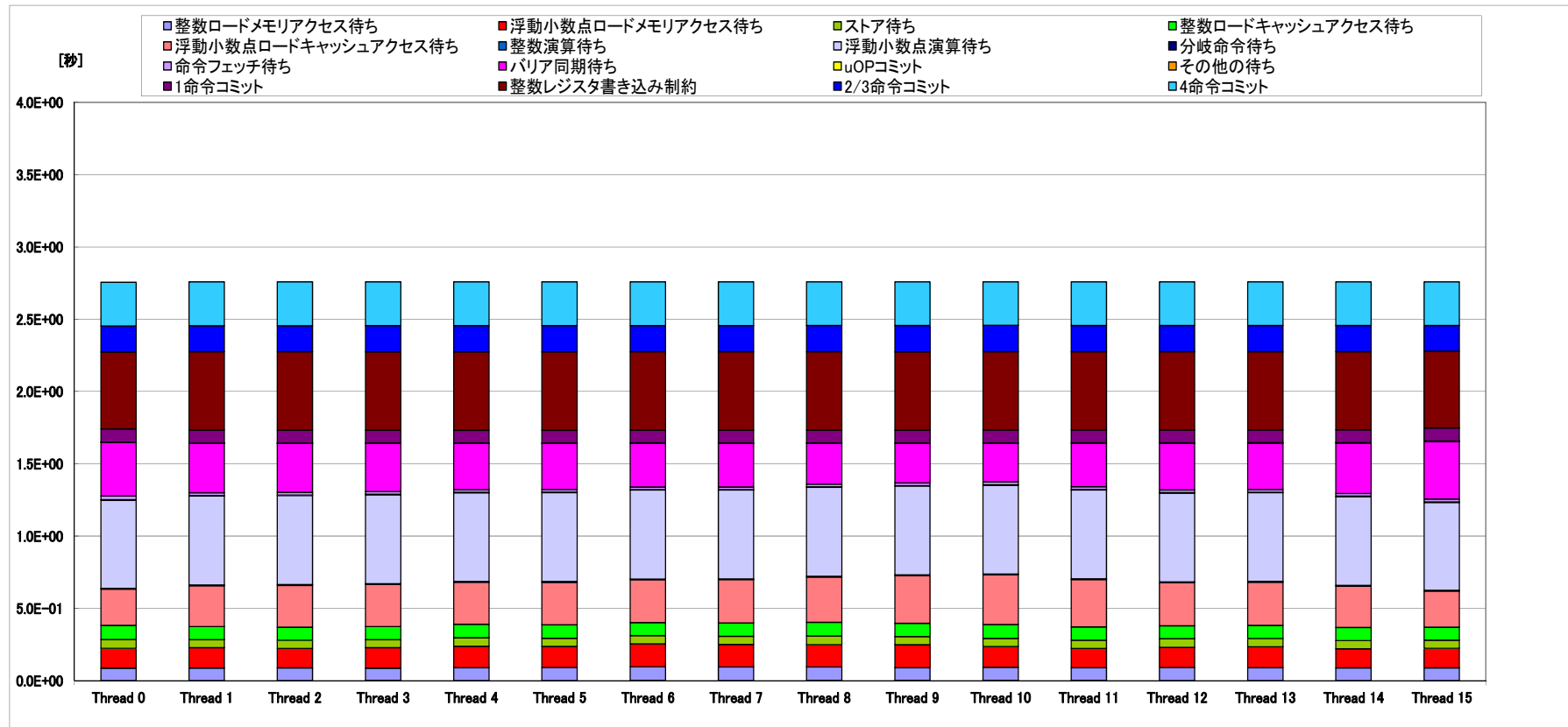
## 37.7%, 28.7GB/sec.



**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

## 詳細プロファイラ

上段: CM-RCM(20), 下段: CM-RCM(382)

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

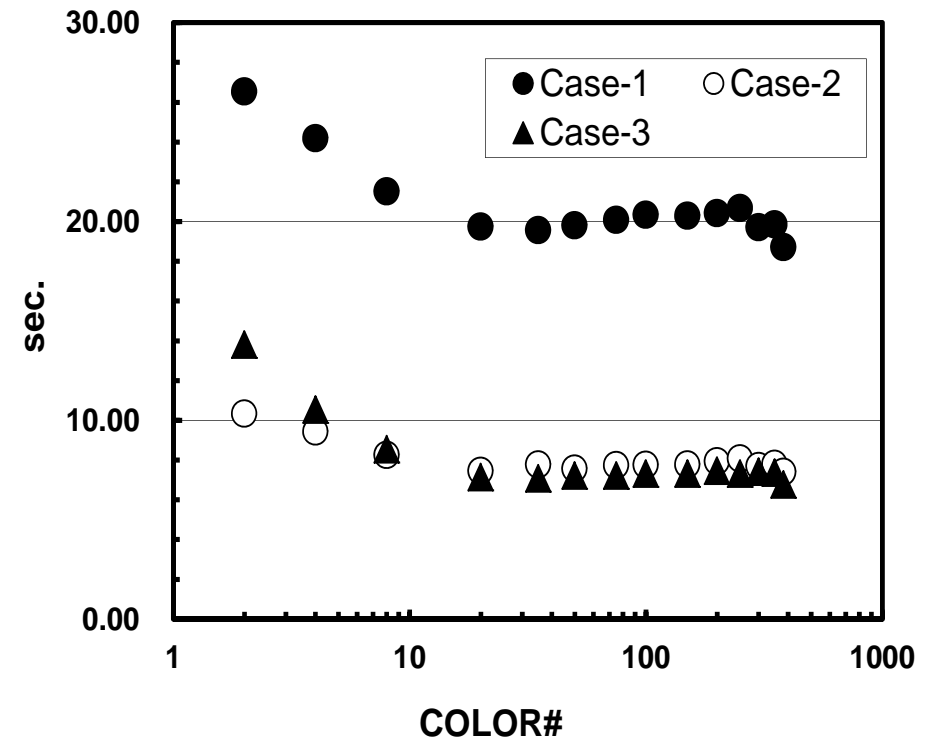
Case-1: src0

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

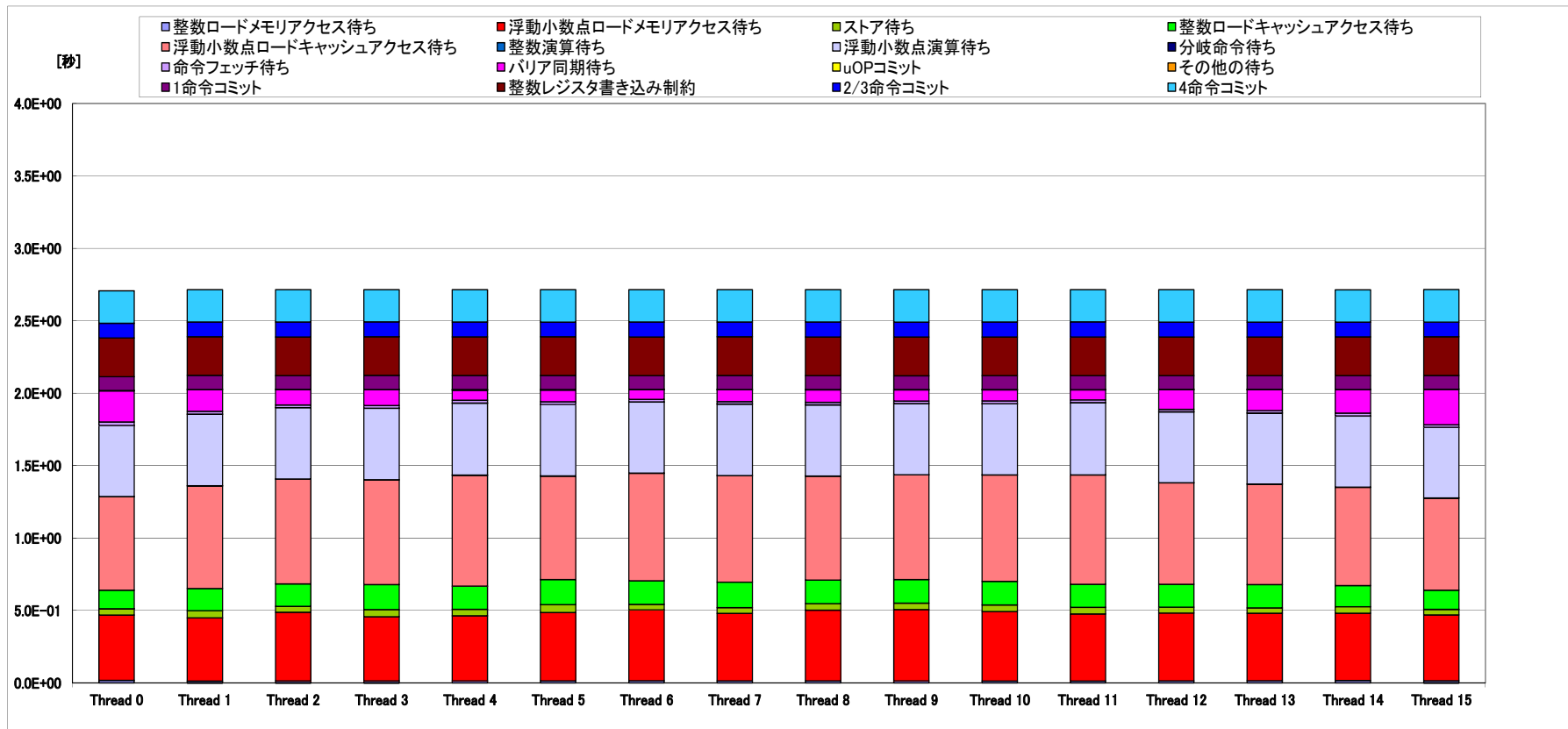
# 計算結果のまとめ

		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)

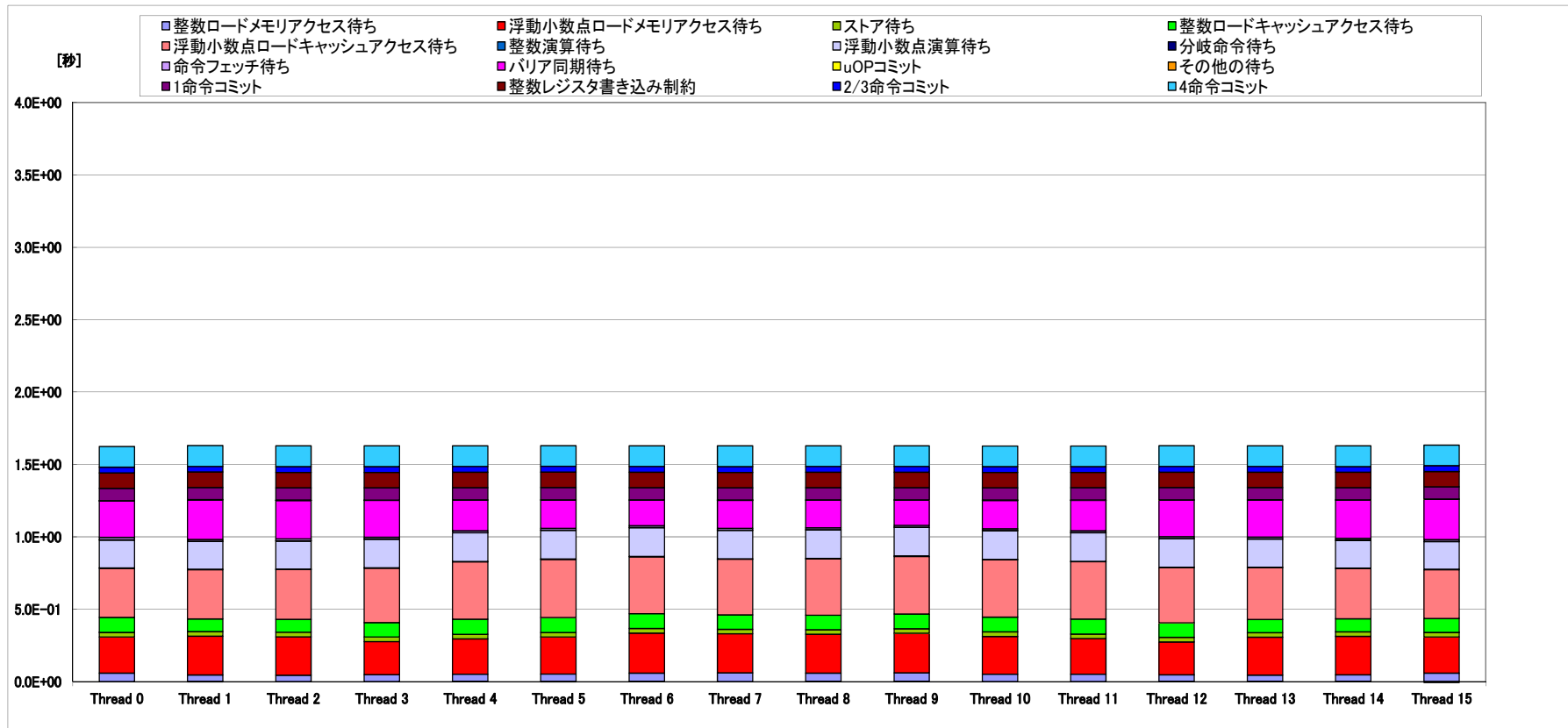
## 5.4%, 64.0GB/sec.



**ELL, Sequential**

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

## 16.5%, 52.2GB/sec.



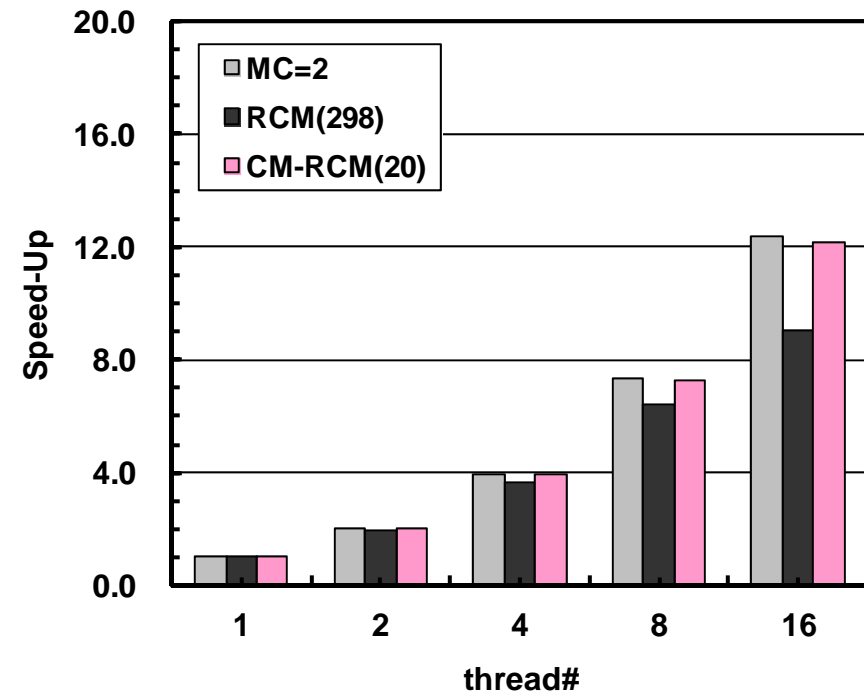
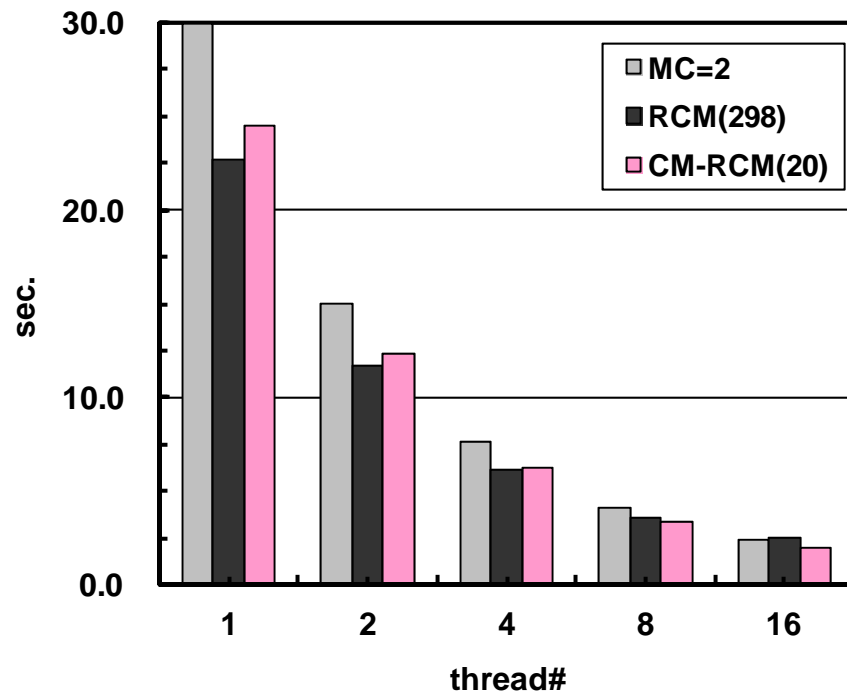
**ELL, Sequential**

- 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

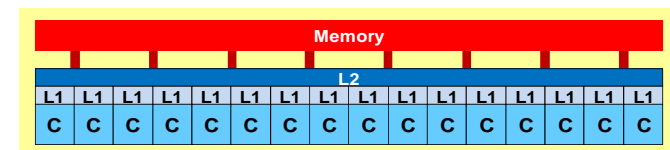
CM-RCM ( $N_c=20$ ): 249回, L3-sol



16 threads

MC(2): 2.42 sec.

CM-RCM(20): 2.01 sec.





# 何故16倍にならないか？

- 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・密行列

```
for (i=0; i<N; i++) {  
    Y[i] = Diag[i] * X[i];  
    for (k=Index[i]; k<Index[i+1]; k++) {  
        Y[i] += AMat[k]*X[Item[k]];  
    }  
}
```

```
for (j=0; j<N; j++) {  
    Y[j] = 0.0;  
    for (i=0; i<N; i++) {  
        Y[j] += A[j][i]*X[i];  
    }  
}
```

- “X” in RHS
  - 密行列: 連続アクセス, キャッシュ有効利用
  - 疎行列: 連続性は保証されず, キャッシュを有効に活用できず
    - より「memory-bound」

# 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 benchmark

<http://www.cs.virginia.edu/stream/>

- 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
-----
```

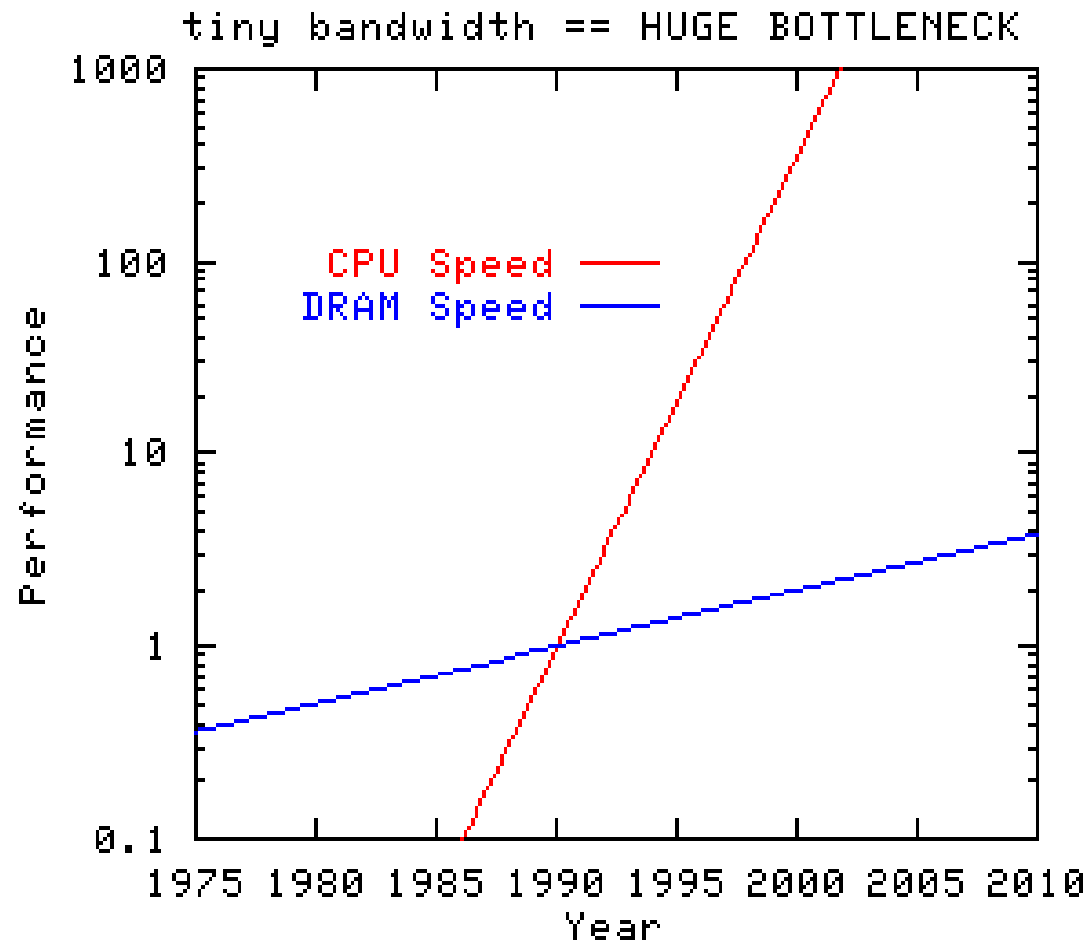
```
Number of processors =          16
Array size =      2000000
Offset      =          0
The total memory requirement is      732.4 MB
(      45.8MB/task)
You are running each test  10 times
--
```

```
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 run.sh
```

# run.sh

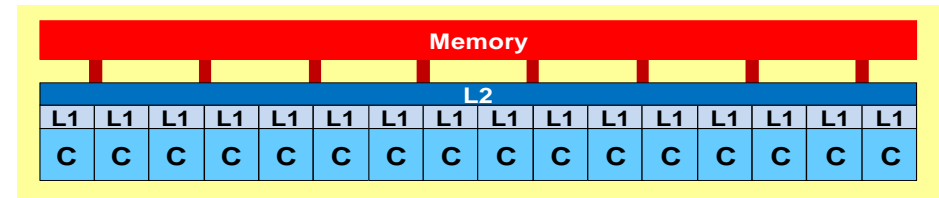
```
#!/bin/sh
#PJM -L "rscgrp=school"
#PJM -L "node=1"
#PJM -L "elapse=10:00"
#PJM -j

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
```

# Results of Triad

<\$O-stream>/stream/\*.lst  
Peak is 85.3 GB/sec., 75%



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

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

## Kshortloop

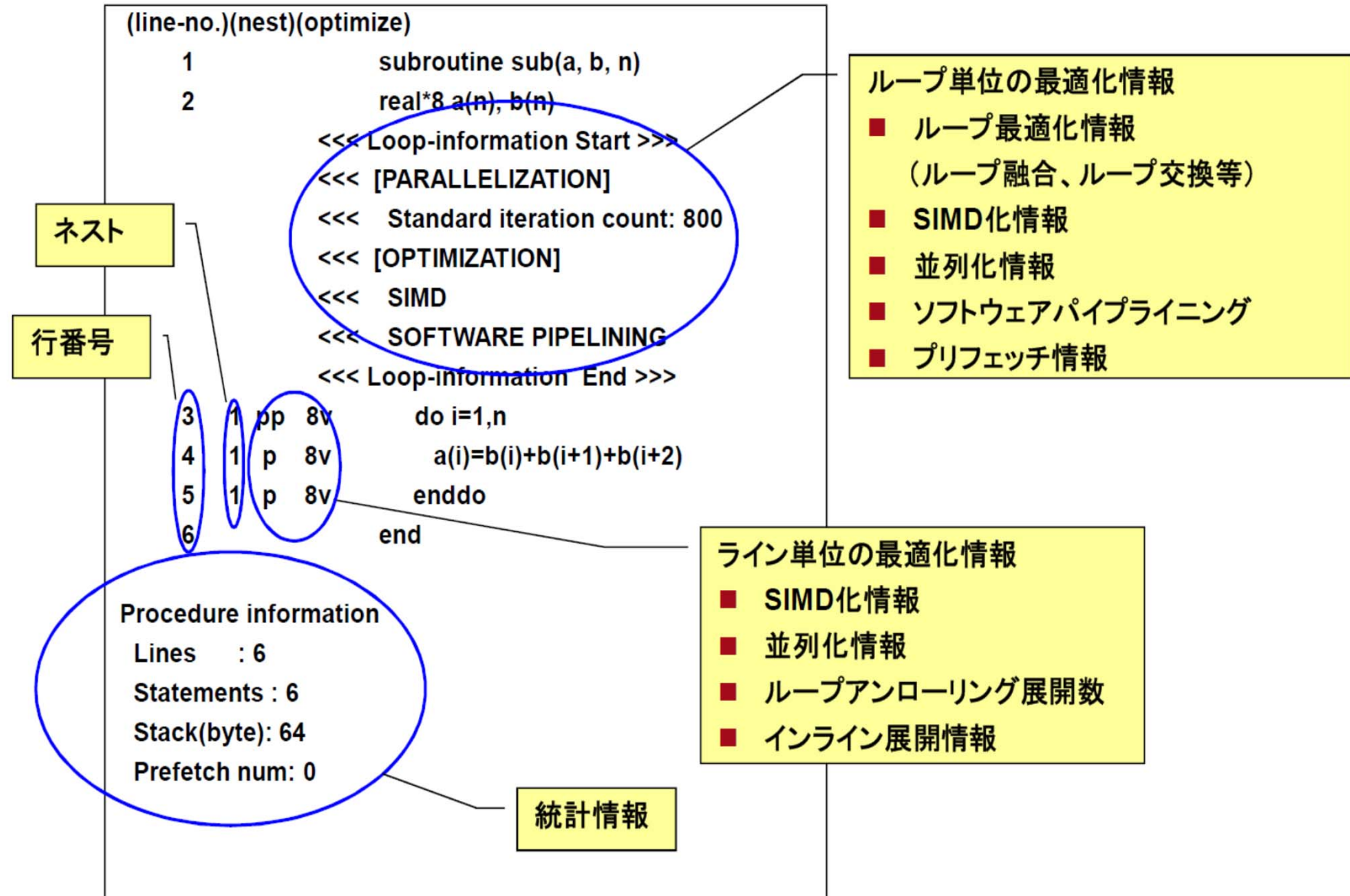
```
>$ cd <${0-L3}>/short
>$ make
>$ ls ../run/L3-sols
   L3-sols
>$ cd ../run
>$ pjsub gos.sh
```

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

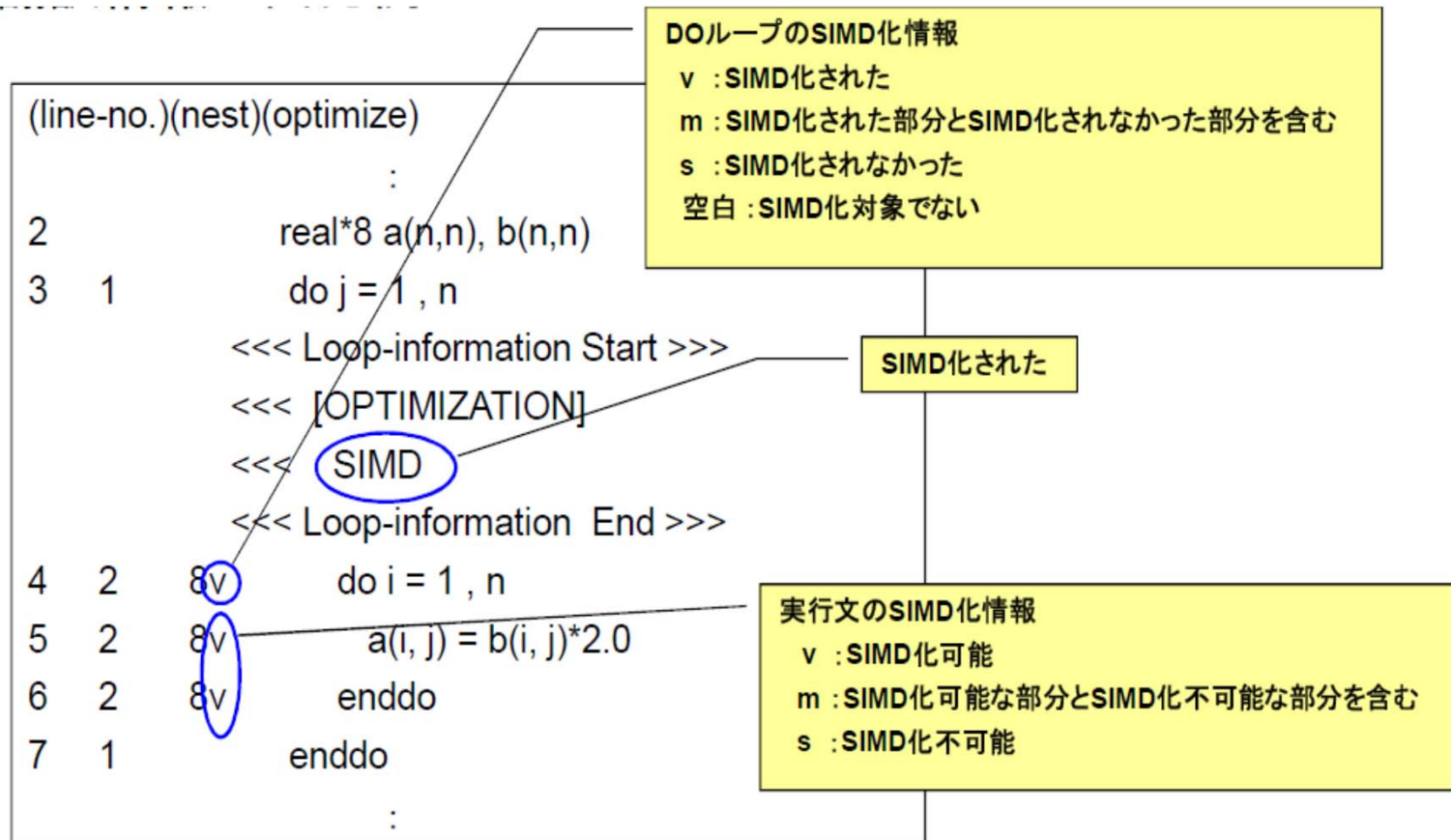
# コンパイル・実行

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

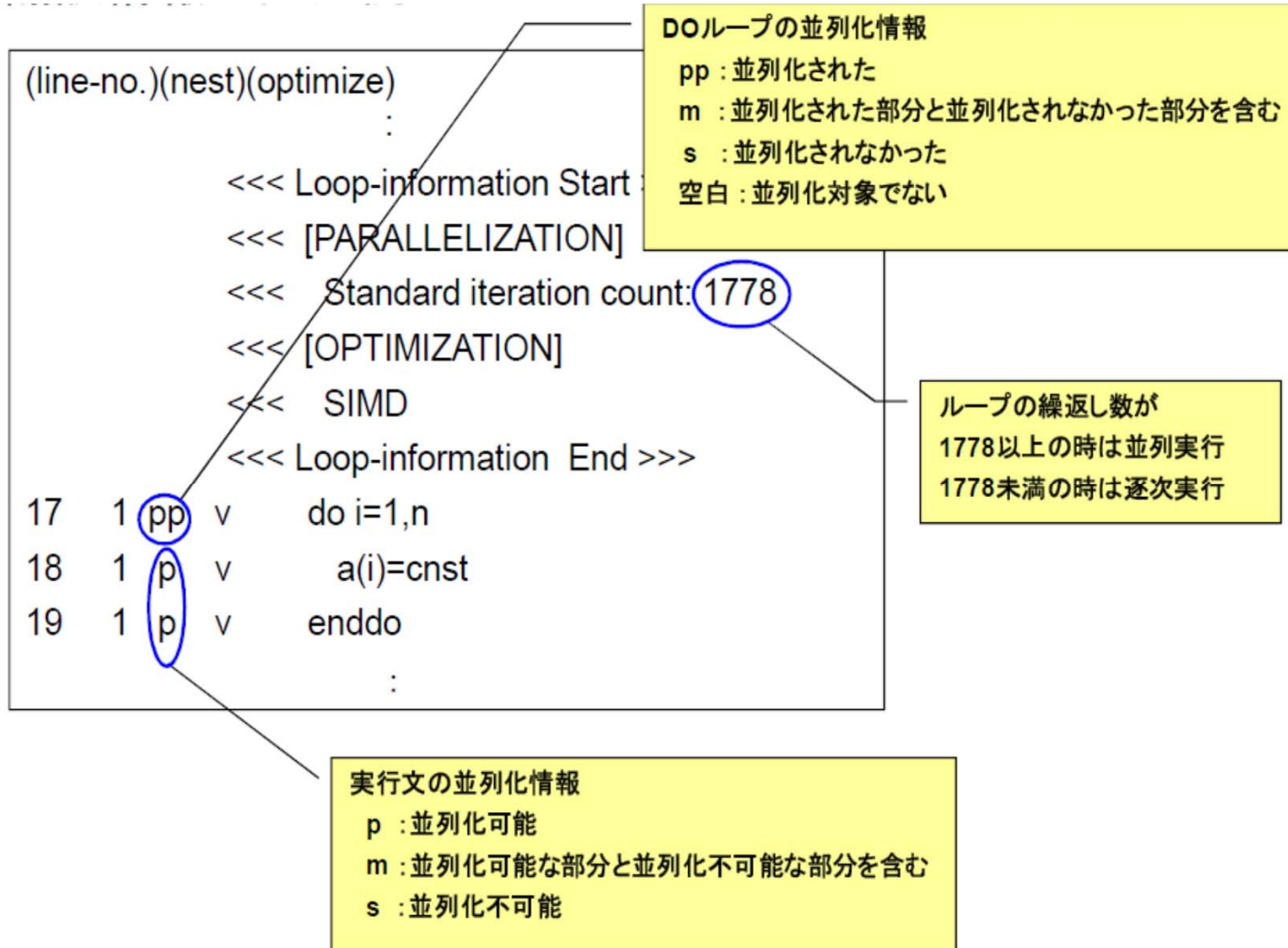
# \* .lstの見方



# SIMD情報



# 自動並列化情報



# solver\_ICCG\_mc.lst (src)

```

101      1          !C
102      1          !C +-----+
103      1          !C | {z}= [Minv]{r} |
104      1          !C +-----+
105      1          !C===
106      1
107      1          !$omp parallel do private(ip,i)
108      2      p          do ip= 1, PESmpTOT
<<< Loop-information Start >>>
<<< [OPTIMIZATION]
<<<     SIMD
<<<     SOFTWARE PIPELINING
<<< Loop-information End >>>
109      3      p      8v          do i = SMPindexG(ip-1)+1, SMPindexG(ip)
110      3      p      8v          W(i,Z)= W(i,R)
111      3      p      8v          enddo
112      2      p          enddo
113      1          !$omp end parallel do
114      1
115      1          Stime= omp_get_wtime()
116      1          call fapp_start ("precond", 1, 1)
117      2          do ic= 1, 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")
!$omp parallel do private(ip, i, VAL, k)
do ip= 1, PEsmptOT
  do i= SMPindex((ip-1)*NCOLORtot)+1, SMPindex(ip*NCOLORtot)
    VAL= D(i)*W(i, P)
    do k= 1, 3
      VAL= VAL + AL(k, i)*W(itemL(k, i), P)
    enddo
    do k= 1, 3
      VAL= VAL + AU(k, i)*W(itemU(k, i), P)
    enddo
    W(i, Q)= VAL
  enddo
enddo
!$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

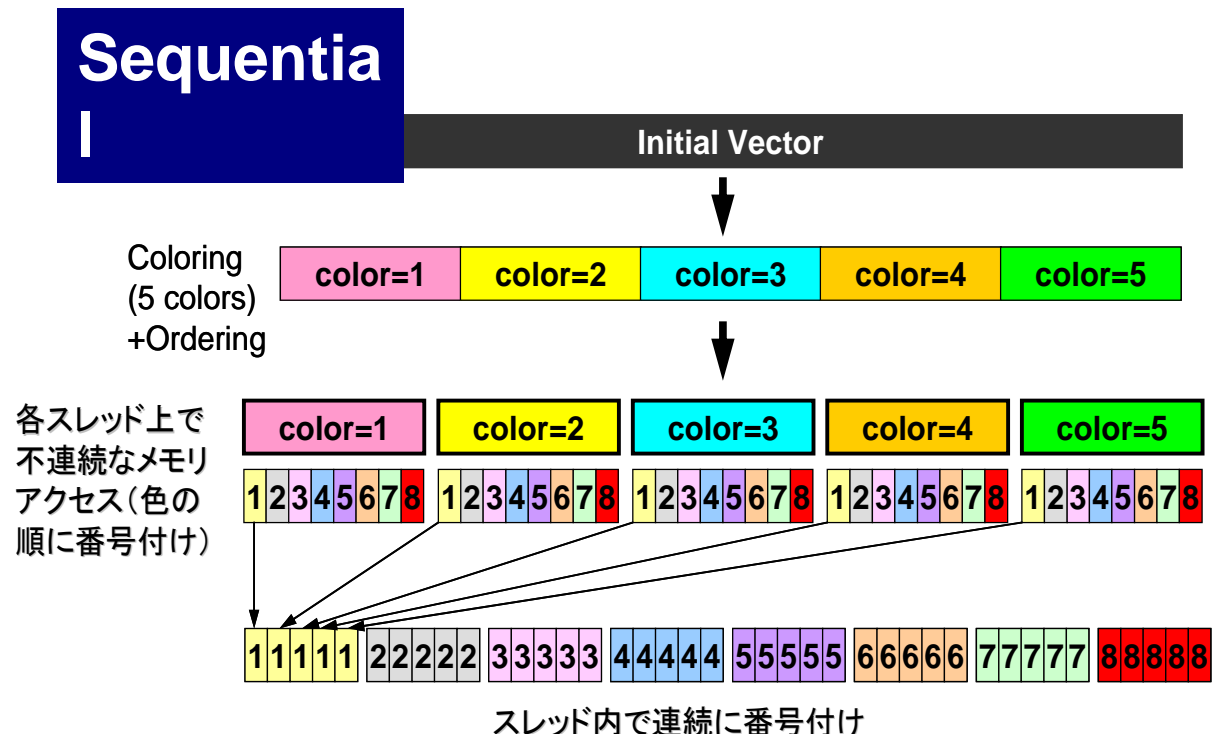
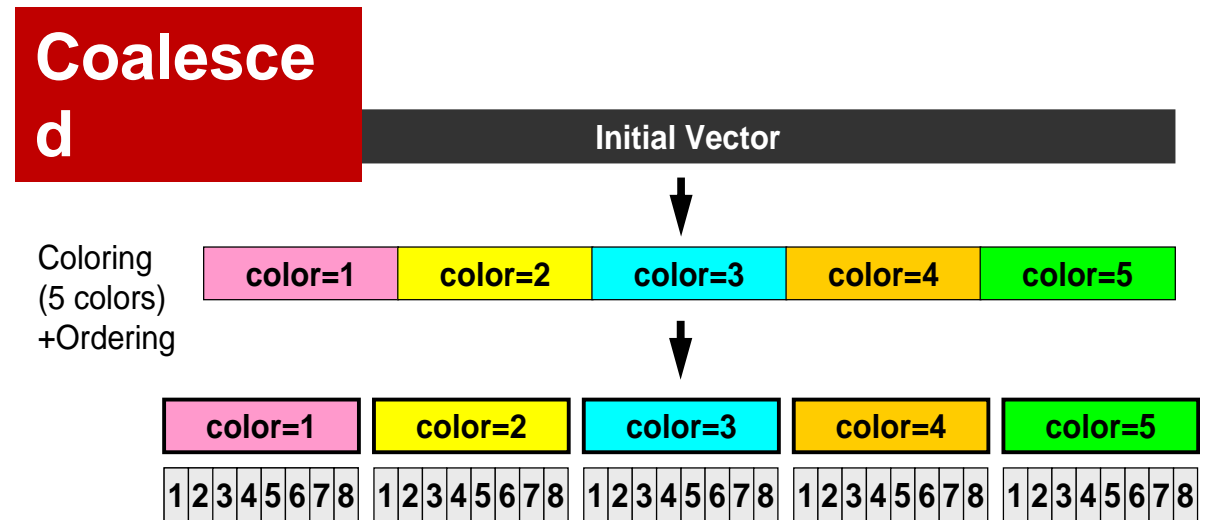
# 今後の動向

- 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

略 称	FX10	MIC	IvyB
名 称	Fujitsu SPARC64 IX fx	Intel Xeon Phi 5110P (Knights Corner)	Intel Xeon E5-2680 v2 (Ivy-Bridge-EP)
動作周波数 (GHz)	1.848	1.053	2.80
コア数 (有効スレッド 数)	16 (16)	60 (240)	10 (20)
使用スレッド数	16	240	10
メモリ種別	DDR3	GDDR5	DDR3
理論演算性能 (GFLOPS)	236.5	1,010.9	224.0
主記憶容量 (GB)	32	8	64
理論メモリ性能 (GB/sec.)	85.1	320	59.7
キャッシュ構成	L1:32KB/core L2:12MB/socket	L1:32KB/core L2:512KB/core	L1:32KB/core L2:256KB/core L3:25MB/socket
コンパイルオプション	-Kfast, openmp	-O3 -openmp - mmic -align array64byte	-O3 -openmp -ipo -xAVX -align array32byte

# Coalesced, Sequential

- NUMA向け最適化
  - First Touch Data Placement
    - 計算と同じ順番で初期化
  - Sequential Reordering (再オーダリング)
    - 色の順番にリオーダリング (Coalesced)
    - 更にスレッド上で番号が連続となるように再番号付け



	Numbering	行列格納形式	外側ループ	その他
AR-0	Coalesced	CRS	行方向 (Row-wise)	
AR-1		Sliced ELL		
AR-2				間接メモリアクセス
AC-1			列方向 (Column-wise)	
AC-2			ブロック化あり	
BR-0	Sequential	CRS	行方向 (Row-wise)	
BR-1		Sliced ELL		
BC-1			列方向 (Column-wise)	
BC-2				ブロック化

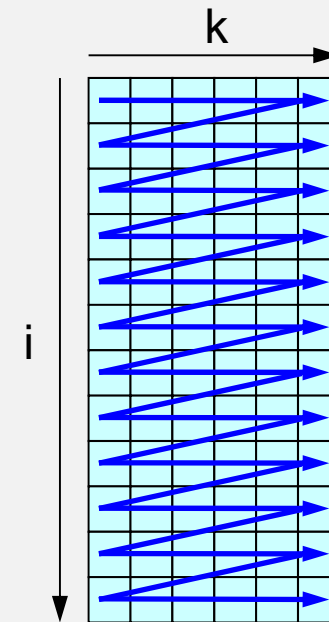
# ELL: 外側ループ: 行方向: Row-wise

従来, 中島がやってきたやり方: CRS with fixed length  
前進代入のループ

```

!$omp parallel
  do icol= 1, NCOLORTot
!$omp do
  do ip = 1, PEsmptOT
    do i= Index(ip-1, icol)+1, Index(ip, icol)
      do k= 1, 6
         $Z(i) = Z(i) - AML(k, i) * Z(IAML(k, i))$ 
      enddo
       $Z(i) = Z(i) / DD(i)$ 
    enddo
  enddo
!omp end parallel

```





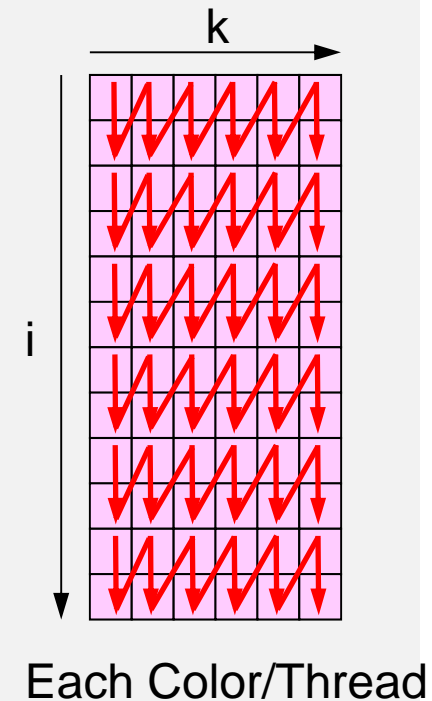
# ELL: 外側ループ: 列方向: Column-wise

こちらが本来のELL: Jagged Diagonal  
係数行列のアクセス不連続

```

!$omp parallel
  do icol= 1, NCOLORTot
!$omp do
  do ip = 1, PEsmptOT
    do k= 1, 6
      do i= Index(ip-1, icol)+1, Index(ip, icol)
        Z(i) = Z(i) + AML (i, k)*Z(IAML(i, k))
      enddo
    enddo
    do i= Index(ip-1, icol)+1, Index(ip, icol)
      Z(i) = Z(i) / DD(i)
    enddo
  enddo
!omp end parallel

```



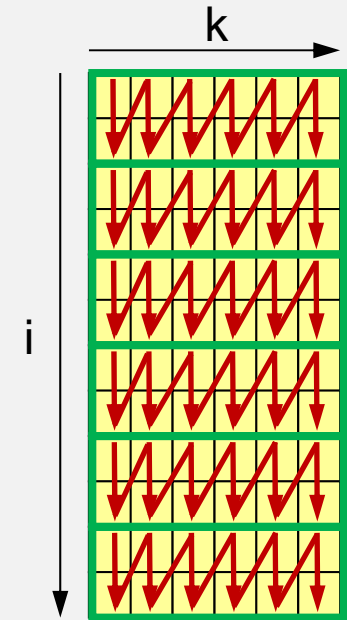
# ELL: 外側ループ: 列方向: Column-wise

ブロック化版, 各色・スレッドごとに別々のブロックで  
係数行列を記憶

```

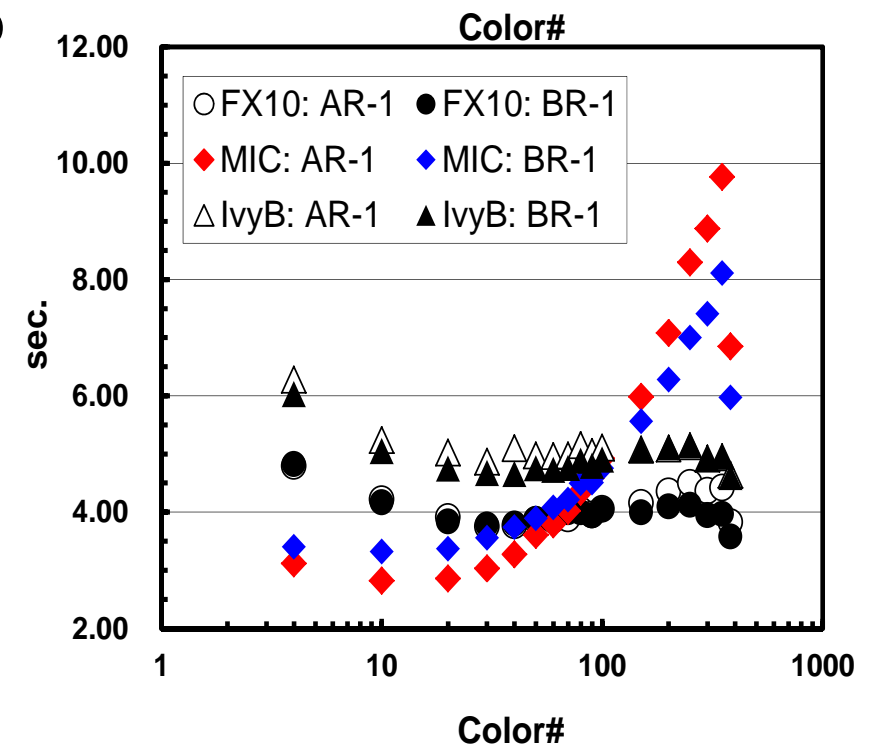
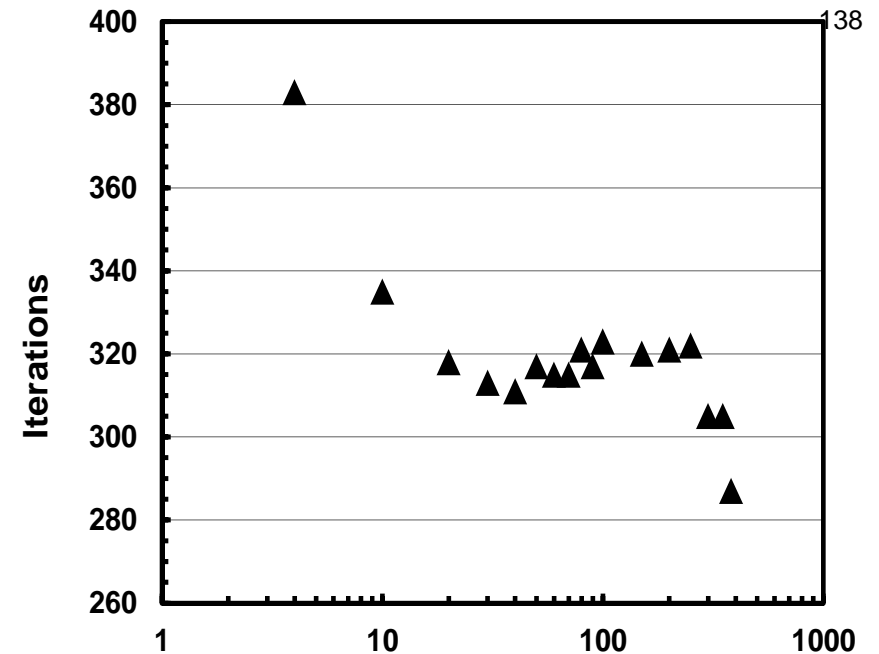
!$omp parallel
  do icol= 1, NCOLORTot
!$omp do
  do ip = 1, PEsmptTOT
    blkID= (ip-1)*NCOLORtot + ip
    do k= 1, 6
      do i= IndexB(ip-1,blkID,icol)+1, &
          IndexB(ip ,blkID,icol)
        locID= i - IndexB(ip-1,blkID,icol)
        Z(i)= Z(i) +
          AMLb(locID, k, blkID)* X(IAMLb(locID, k, blkID))
      enddo
    enddo
    do i= IndexB(ip-1,blkID,icol)+1, IndexB(ip ,blkID,icol)
      Z(i)= Z(i) / DD(i)
    enddo
  enddo
!omp end parallel

```



# 色数と計算時間 (Solver)

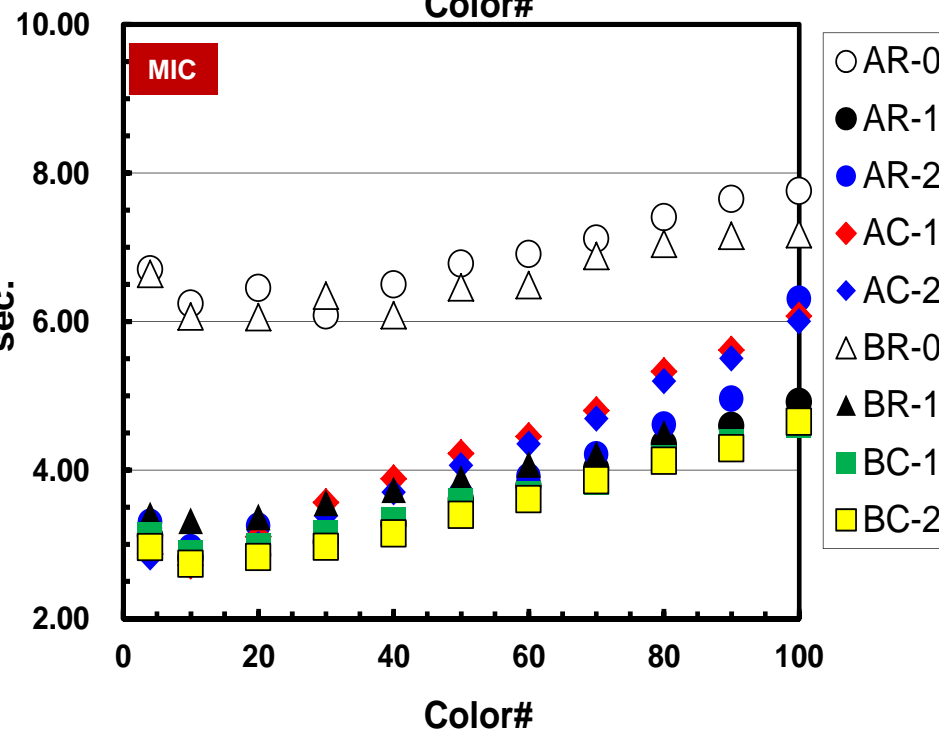
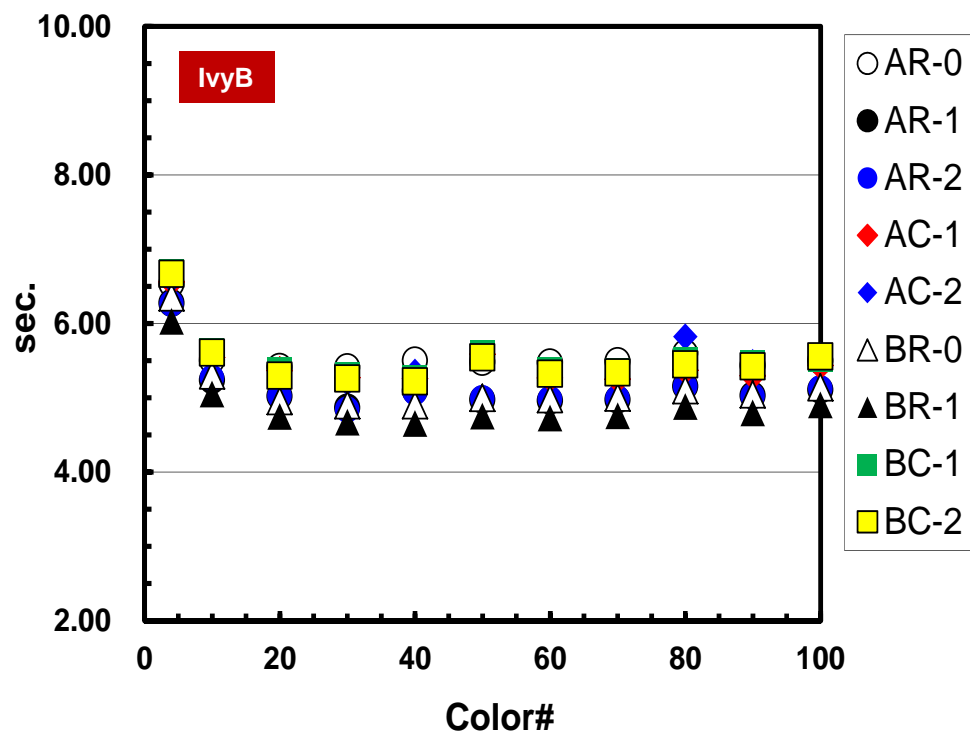
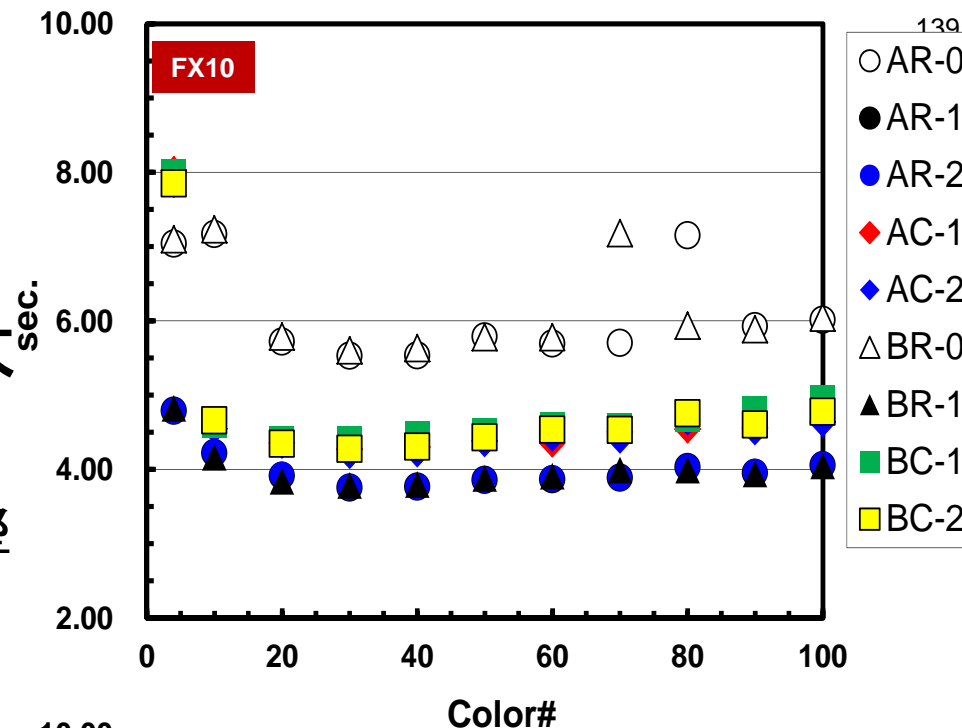
- ELLの場合
  - AR-1: Coalesced + Row-wise
  - BR-1: Sequential + Row-wise
- 色数が減ると反復回数が減るが、同期オーバーヘッドは増加
- MICはこれが顕著
- 計算時間を考慮した最適色数はアーキテクチャによって異なる
- FX10, IvyBはRCMが良い



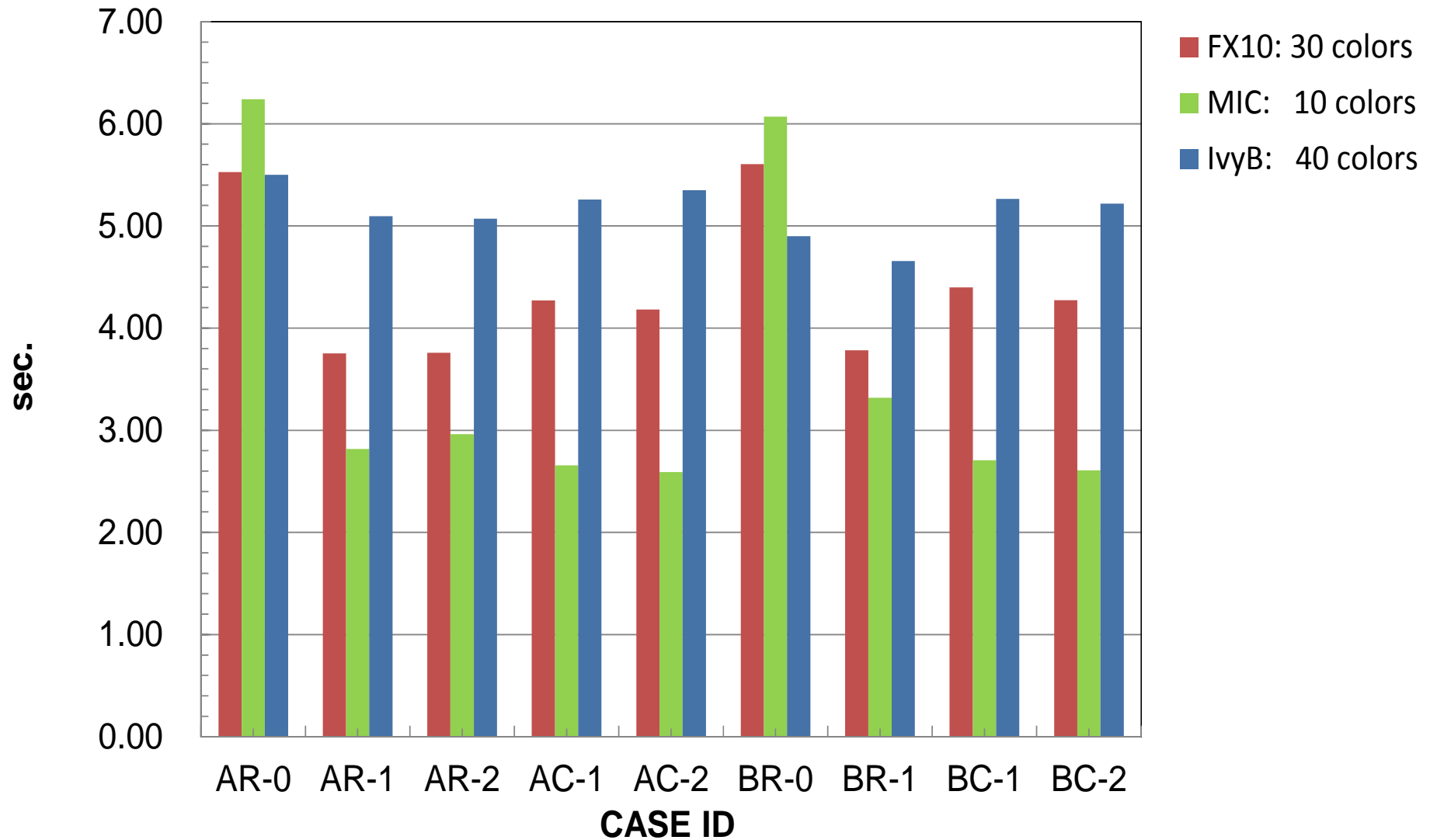
# 色数と計算時間

## 100色まで

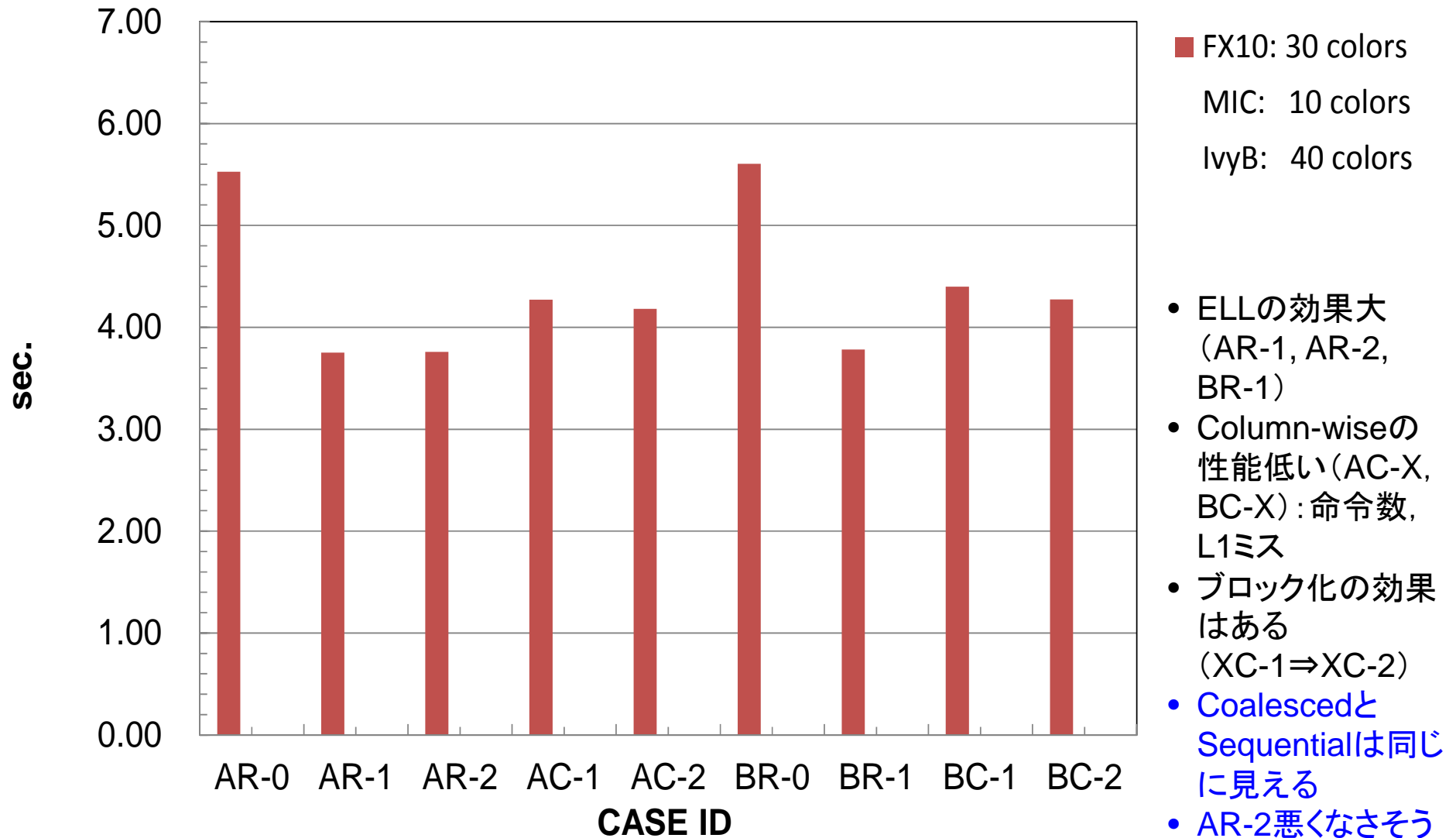
- FX10, MICはELLの効果大 (CRS遅い: AR-0, BR-0)
- IvyBは効果少, 色数の影響もほとんど無い



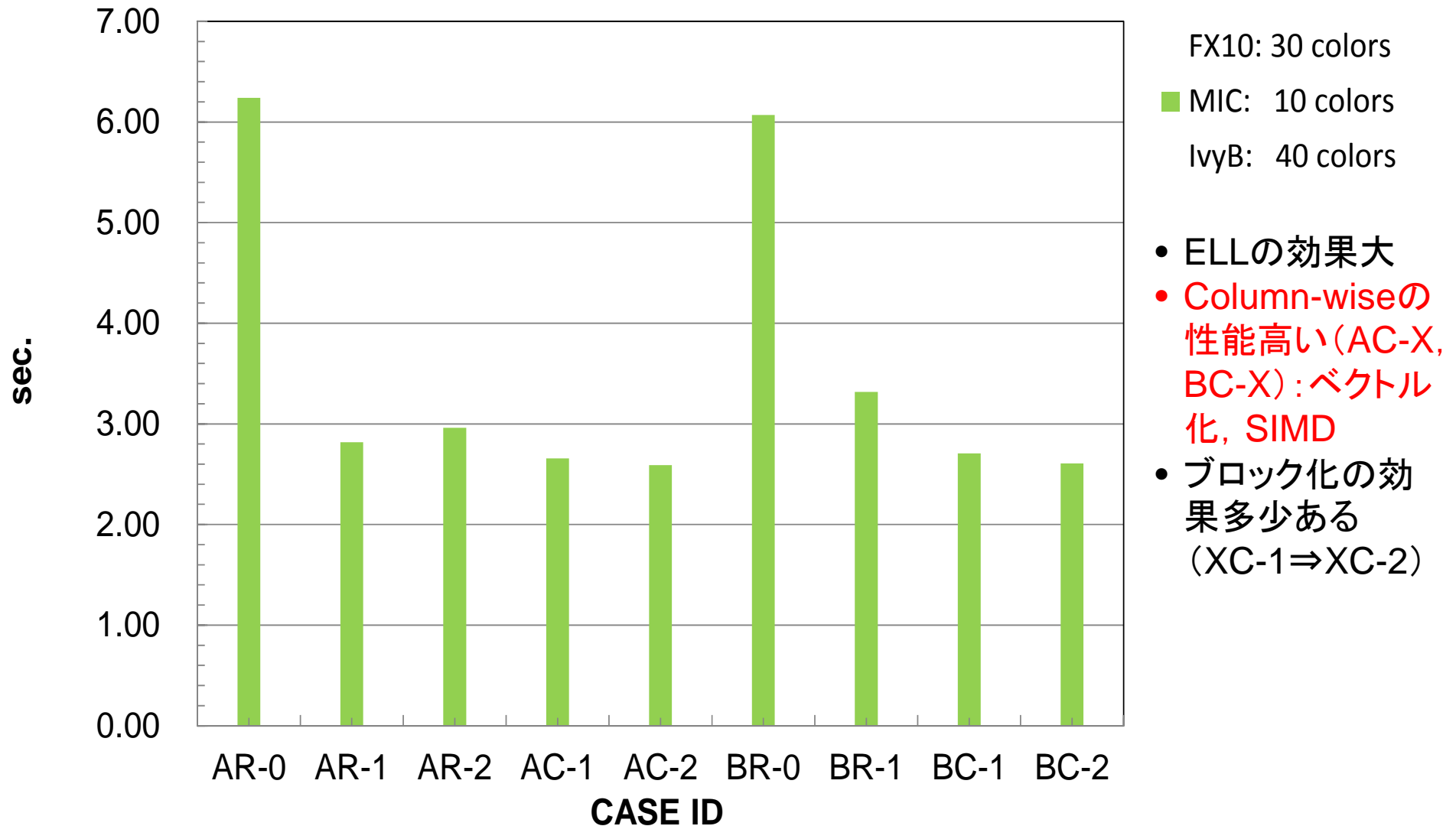
# 各手法の結果 (Solver計算時間)



# 各手法の結果 (Solver計算時間):FX10



# 各手法の結果 (Solver計算時間):MIC



# 各手法の結果 (Solver計算時間): IvyB

