OpenMPによるマルチコア・メニィコア並列プログラミング入門 C言語

Part-B3：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

How to Run

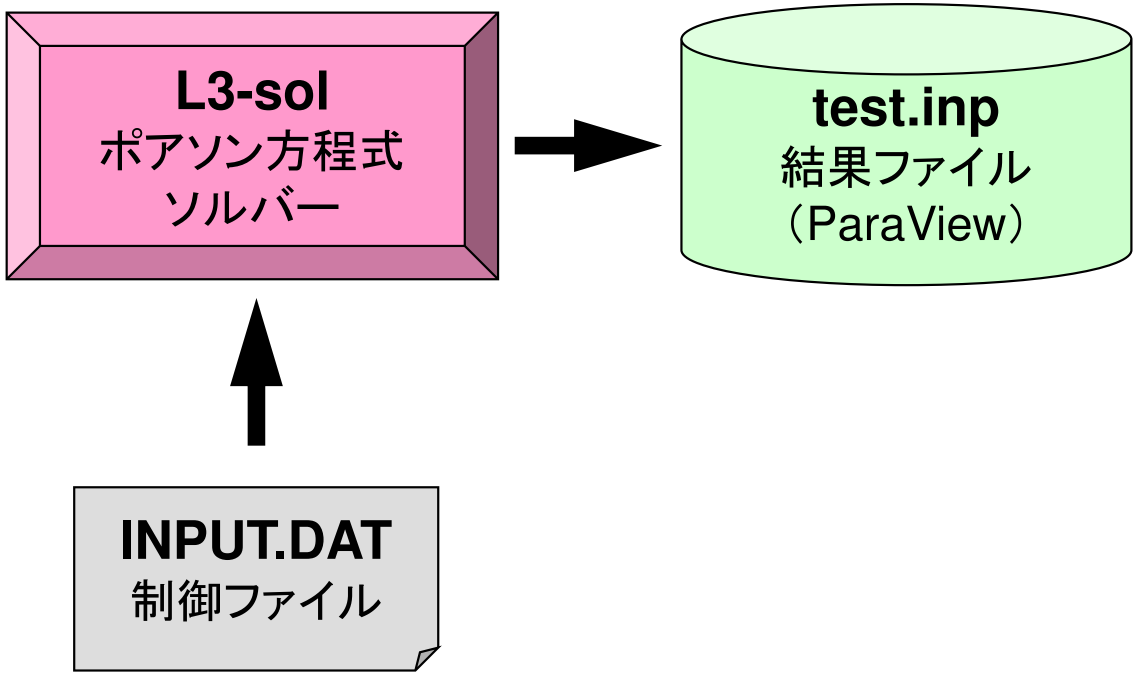
```
>$ cd /work/gt00/t00XXX

>$ cp /work/gt00/z30088/ompc.tar .
>$ tar xvf ompc.tar

>$ cd ompc
>$ ls
    run   src   src0   reorder0

>$ cd src
>$ module load fj
>$ make
>$ cd ../run
>$ ls L3-sol
    L3-sol

<modify “INPUT.DAT”>
<modify “go1.sh”>

>$ pjsub go1.sh
```

Files on Odyssey

- Location
  - **<$O-L3>: /work/gt00/t00XXX/ompc**
  - <$O-L3>/src, <$O-L3>/run
- Compile & Run
  - Source Code
    - cd <$O-L3>/src
    - make
    - <$O-L3>/run/L3-sol execution file
  - Control Data
    - <$O-L3>/run/INPUT.DAT
  - Shell Script
    - <$O-L3>/run/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）

```
128 128 128                         NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00          DX/DY/DZ
1.0e-08                             EPSICCG
24                                  PEsmpTOT
100                                 NCOLORtot
```

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

Job Script: go1.sh

- **/work/gt00/t00XXX/ompc/run/go1.sh**
- Scheduling + Shell Script

```
#!/bin/sh
#PJM -N "go1"                     Job Name (not required)
#PJM -L rscgrp=tutorial-o         Name of Queue (Resource Group)
#PJM -L node=1                    Node # (=1)
#PJM --omp thread=12              Thread # (= PEsmpTOT)
#PJM -L elapse=00:15:00           Elapsed Computation Time
#PJM -g gt00                      Group Name (Wallet)
#PJM -j
#PJM -e err                       Standard Error
#PJM -o test1.lst                 Standard Output

module load fj
export OMP_NUM_THREADS=12         Thread #(--omp thread=XX)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl ./L3-sol
numactl -C 12-23 -m 4 ./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 ⇒ straightforwardな適用可能
- 前処理（修正不完全コレスキー分解，前進後退代入）
  - 同じ色内は依存性無し ⇒ 色内では並列化可能

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

```
#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
要素数（NX×NY×NZ）

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

NEIBcell[ICELTOT][6]
隣接要素

PEsmpTOT
スレッド数

pcg.h

```
#ifndef __H_PCG
#define __H_PCG
        static int N2 = 256;
        int NUmax, NLmax, NCOLORtot, 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 */
```

NCOLORtot Total number of colors/levels

COLORindex [NCOLORtot+1] Index of number of meshes in each color/level (COLORindex[icol+1]- COLORindex[icol])

SMPindex [NCOLORtot*PEsmpTOT+1]

SMPindexG[PEsmpTOT+1]

OLDtoNEW, NEWtoOLD Reference table before/after renumbering

変数表(1/2)

配列・変数名	型	内　容
D[N]	R	対角成分, (N:全メッシュ数)
BFORCE[N]	R	右辺ベクトル
PHI[N]	R	未知数ベクトル
indexL[N+2] indexU[N+2]	I	各行の非零下三角成分数(CRS)
NPL, NPU	I	各行の非零上三角成分数(CRS)
itemL[NPL] itemU[NPU]	I	非零下三角成分総数(CRS)
AL[NPL] AU[NPU]	R	非零上三角成分総数(CRS)

配列・変数名	型	内　容
NL,NU	I	各行の非零上下三角成分の最大数 (ここでは6)
INL[N] INU[N]	I	各行の非零下三角成分数
IAL[N][NL] IAU[N][NU]	I	各行の非零上三角成分数

変数表(2/2)

配列・変数名	型	内 容
NCOLORtot	**I**	入力時にはOrdering手法（≧2：MC，=0：CM，=-1：RCM，-2≧：CMRCM）， 最終的には色数，レベル数が入る
COLORindex [NCOLORtot+1]	**I**	各色，レベルに含まれる要素数の一次元圧縮配列， COLORindex(icol-1)+1から COLORindex(icol)までの要素がicol番目の色（レベル）に含まれる。
NEWtoOLD[N]	**I**	新番号⇒旧番号への参照配列
OLDtoNEW[N]	**I**	旧番号⇒新番号への参照配列
PEsmpTOT	**I**	スレッド数
SMPindex [NCOLORtot*PEsmpTOT+1]	**I**	スレッド用補助配列（データ依存性があるループに使用）
SMPindexG[PEsmpTOT+1]	**I**	スレッド用補助配列（データ依存性が無いループに使用）

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:「IPNUT.DAT」の読み込み

```
#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;
}
```

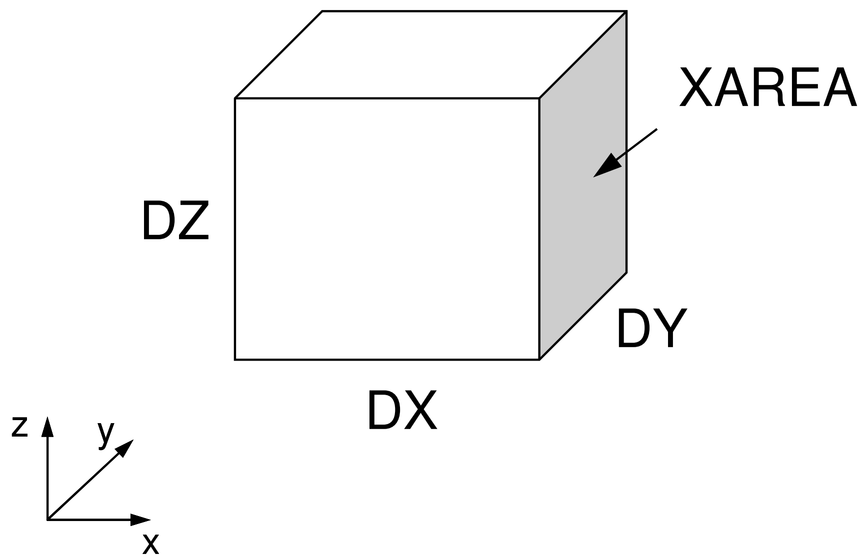
- PEsmpTOT
  - OpenMPスレッド数
- NCOLORtot
  - 色数
  - 「＝0」の場合はCM
  - 「＝-1」の場合はRCM
  - 「≦-2」の場合はCM-RCM

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

cell_metrics

```
#include <stdio.h> ...

extern int
CELL_METRICS(void)
{        double V0, RV0;
         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;
RV0 = 1.0 / V0;

for(i=0; i<ICELTOT; i++) {
    VOLCEL[i] = V0;
    RVC[i]    = RV0;
}
return 0; }
```

計算に必要な諸パラメータ



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)

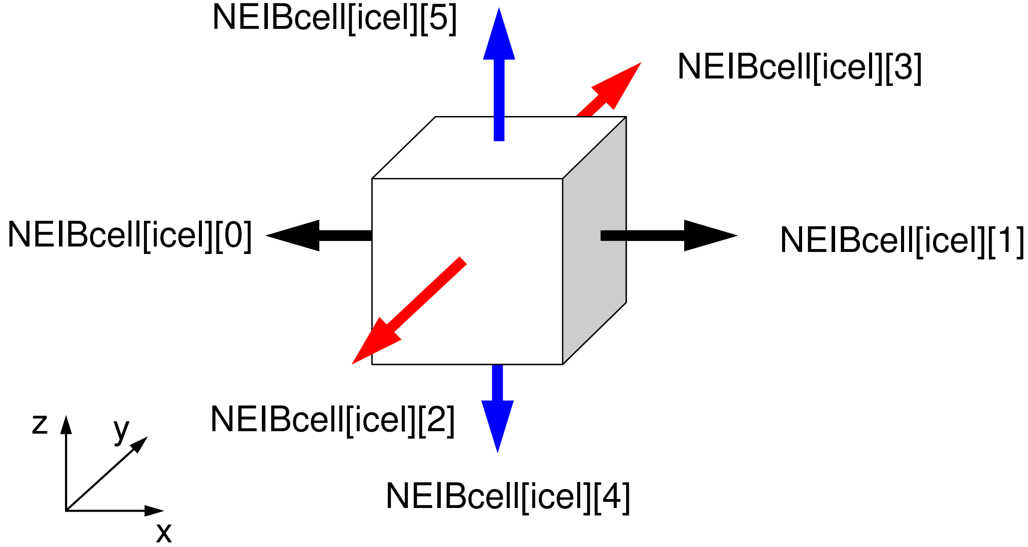
```
#include "allocate.h"
extern int
POI_GEN(void)
{ int nn;
  int ic0, icN1, icN2, icN3, icN4, icN5, icN6;
  int i, j, k, ib, ic, ip, icel, icou, icol, icouG;
  int ii, jj, kk, nn1, num, nr, j0, j1;
  double coef, VOL0, S1t, E1t;
  int isL, ieL, isU, ieU;
  NL=6; NU= 6;
  IAL = (int **)allocate_matrix(sizeof(int),ICELTOT,NL);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

<u>allocate.c</u>

```
/**************************
  allocate matrix
**************************/
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;
}
```

poi_gen (2/9)

```
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;
       }
}
```



下三角成分
NEIBcell[icel][4]= icel – NX*NY + 1
NEIBcell[icel][2]= icel – NX + 1
NEIBcell[icel][0]= icel – 1 + 1

“icel” starts at 0

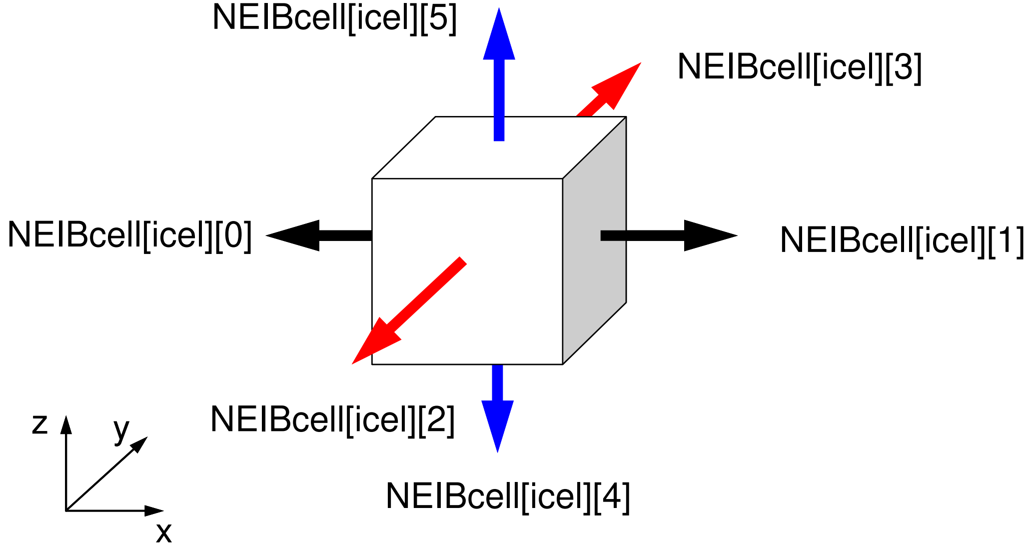
12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

“IAL” starts at 1

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

poi_gen (2/9)

```
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;
       }
}
```



上三角成分

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)

```
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;
}
```

並べ替えの実施：

NCOLORtot > 1 : Multicolor
NCOLORtot = 0 : CM
NCOLORtot =-1 : RCM
NCOLORtot <-1 : CM-RCM

poi_gen (4/9)

```
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];
}
```

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 ⇒ OpenMP適用

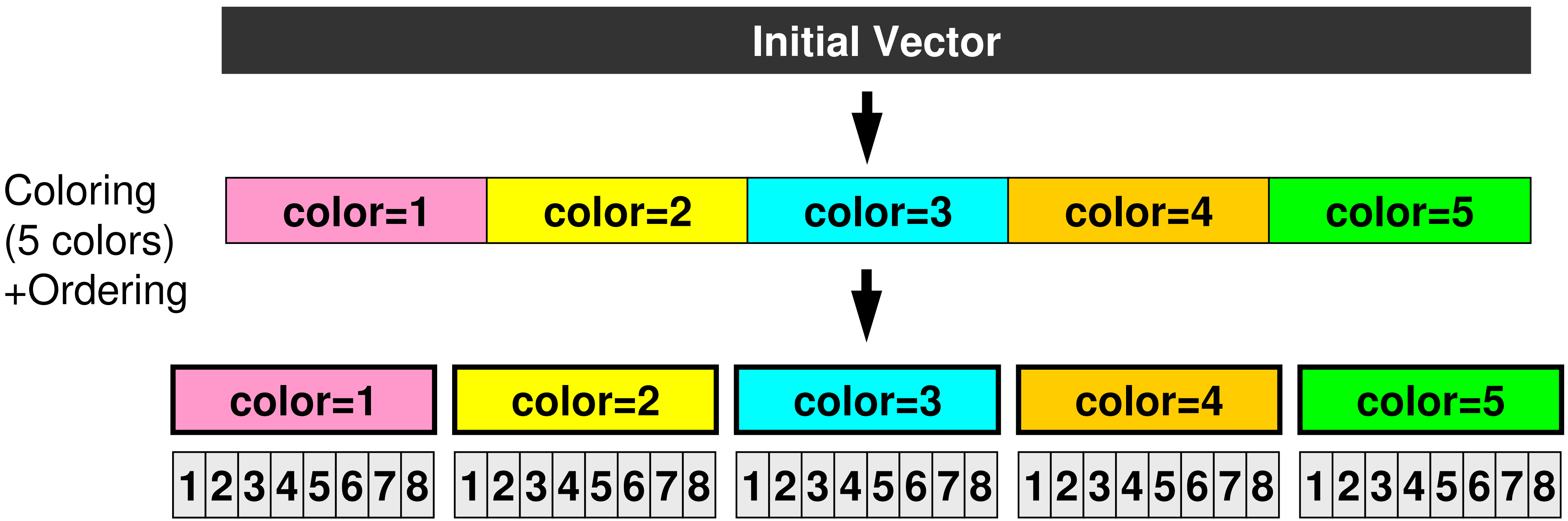
これを更に「PEsmpTOT」で割って
「SMPindex」に割り当てる。

前処理で使用

```
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++) {
            (...)
        }
    }
}
```

SMPindex：前処理向け

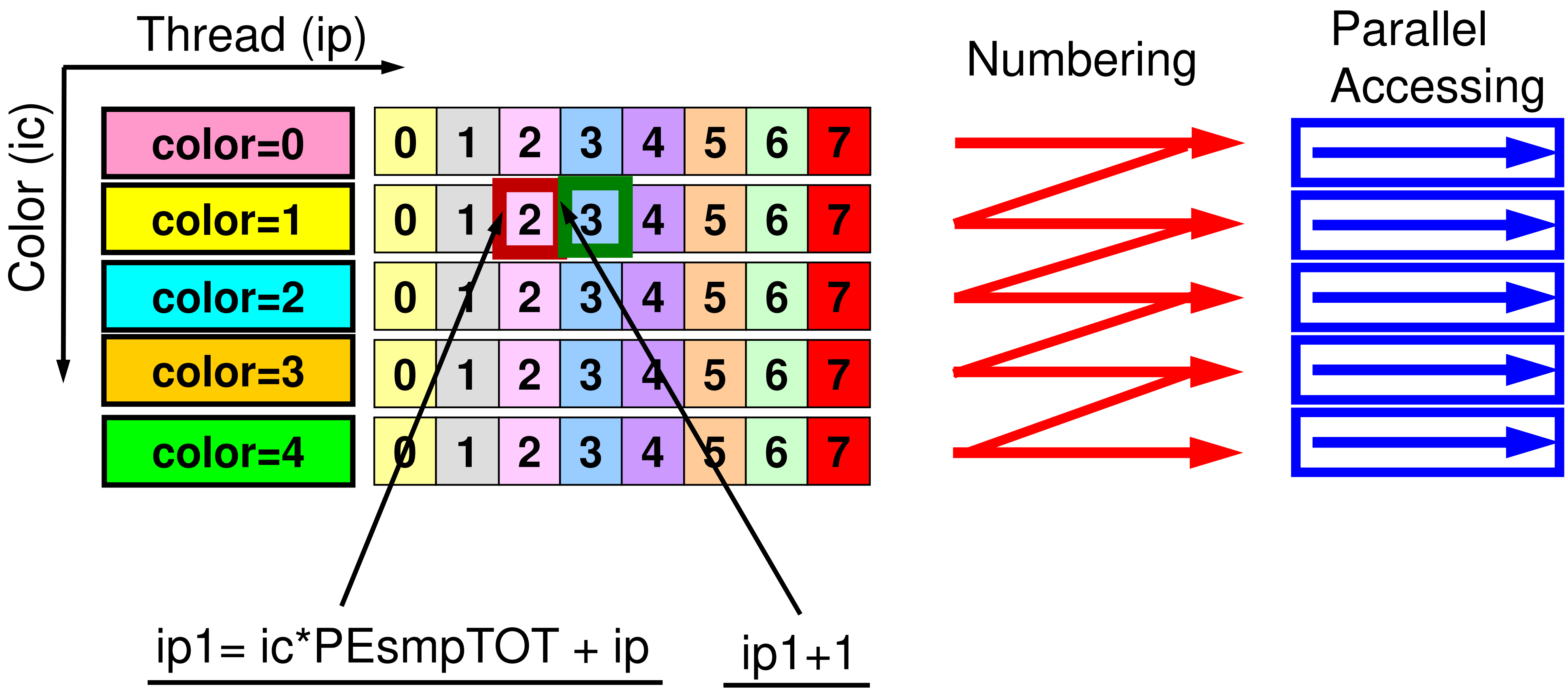
```
for(ic=0; ic<NCOLORtot; ic++) {
#pragma omp parallel for ...
    for(ip=0; ip<PEsmpTOT; ip++) {
        ip1 = ic * PEsmpTOT + ip;
        for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {
            (...)
        }
    }
}
```



- 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

SMPindex

```
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++) {…
```



SMPindex

```
COLORindex[ic  ]= 100
COLORindex[ic+1]= 200
PEsmpTOT        =   8

nn1= 200-100    = 100
num= 100 / 8    =  12 (12.5)
nr = 100 - 12*8=   4
ip0= ic*PEsmpTOT (ic: starting at 0)

SMPindex[ip0  ]= 100
SMPindex[ip0+1]= 113 (13 elements in the 1st thread)
SMPindex[ip0+2]= 126 (13 elements in the 2nd thread)
SMPindex[ip0+3]= 139 (13 elements in the 3rd thread)
SMPindex[ip0+4]= 152 (13 elements in the 4th thread)
SMPindex[ip0+5]= 164 (12 elements in the 5th thread)
SMPindex[ip0+6]= 176 (12 elements in the 6th thread)
SMPindex[ip0+7]= 188 (12 elements in the 7th thread)
SMPindex[ip0+8]= 200 (12 elements in the 8th thread)
```

poi_gen (4/9)

```
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(sizeof(int),
                         PEsmpTOT+1);
memset(SMPindexG, 0, sizeof(int)*(PEsmpTOT+1));

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

```
#pragma omp parallel for ...
for(ip=0; ip<PEsmpTOT; ip++) {
    for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++) {
        (...)
    }
}
```

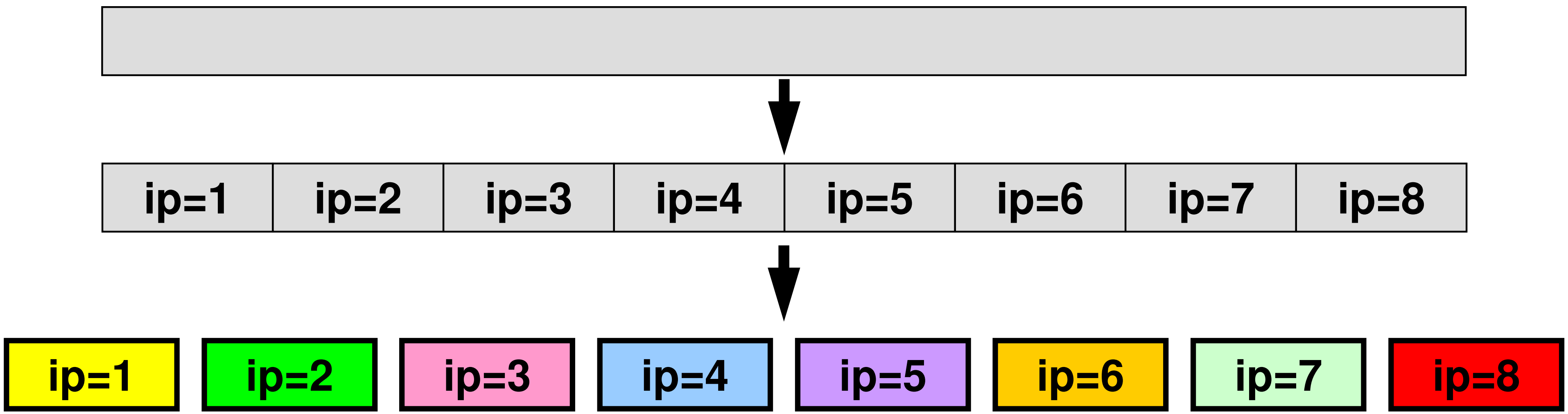
全要素数を「PEsmpTOT」で割って「SMPindexG」に割り当てる。

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

```
#pragma omp parallel for ...
for (ip=0; ip<PEsmpTOT; ip++) {
    for (i=SMPindexG[ip]; i<SMPindexG[ip+1]; 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);
```

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[N+2] **indexU[N+2]**	**I**	各行の非零下三角成分数(CRS)
NPL, NPU	**I**	各行の非零上三角成分数(CRS)
itemL[NPL] **itemU[NPU]**	**I**	非零下三角成分総数(CRS)
AL[NPL] **AU[NPU]**	**R**	非零上三角成分総数(CRS)

"itemL" /"itemU" start at 1

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

```
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];
   }
}
```

poi_gen (6/9)

新しい番号付けを使用

```
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];
     VOL0 = 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

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

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

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

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

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

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

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];
```

poi_gen (6/9)

新しい番号付けを使用

```
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];
    VOL0 = 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

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

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

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

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

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

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

poi_gen (6/9)
新しい番号付けを使用

```
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];
    VOL0 = 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;
              }
          }
       }
    }
...
```

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

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

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

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

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

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

poi_gen (6/9)

新しい番号付けを使用

```
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];
    VOL0 = 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;
              }
          }
       }
    }
...
```

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

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

icN5 < icel
Lower Part

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

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

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

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

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

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

poi_gen (6/9)

新しい番号付けを使用

```
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];
    VOL0 = 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;
              }
          }
       }
    }
...
```

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

$$\mathrm{ZAREA} = \Delta x \Delta y$$

icN5 > icel
Upper Part

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

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

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

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

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

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

poi_gen (7/9)

```
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;}
      }
   }
}
```

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

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

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

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

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

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

poi_gen (8/9)

```
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; }
      }
   }
}
```

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

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

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

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

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

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

poi_gen (9/9)

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

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

}
```

もとの座標に従ってBFORCE計算

ii,jj,kk,VOL0: private

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

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;
}
```

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)

```
#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 PEsmpTOT, 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;
        }
    }
}
```

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[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;
}
```

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

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

行列ベクトル積

依存性が無い⇒独立に計算可能⇒SMPindexG使用

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

```
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) + αip(i)
      r(i)= r(i-1) - αiq(i)
      check convergence |r|
end
```

内積：SMPindexG使用，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];
    }
}
```

solve_ICCG_mc (4/6)

```
*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,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];
    }
}
}
for(ic=NCOLORtot-1; ic>=0; ic--){
#pragma omp parallel for private (ip,ip1,i,SW,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;
     }
}
}
```

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

solve_ICCG_mc (4/6)

```
*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,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];
    }
}
}
for(ic=NCOLORtot-1; ic>=0; ic--){
#pragma omp parallel for private (ip,ip1,i,SW,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;
     }
}
}
```

SMPindex

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

solve_ICCG_mc (4/6)

```
*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,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];
    }
}
}
for(ic=NCOLORtot-1; ic>=0; ic--){
#pragma omp parallel for private (ip,ip1,i,SW,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;
     }
}
}
```

SMPindex

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

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

前進代入
Forward Substitution

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

後退代入
Backward Substitution

前進代入:SMPindex使用

```
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];
    }
}
}
```

solve_ICCG_mc (5/6)

```
/********************************
 * {p}  = {z} if      ITER=0    *
 * BETA = RHO / RHO1  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 / RHO1;
#pragma omp parallel for private(ip,i)
   for(ip=0; ip<PEsmpTOT; ip++) {
       for(i=SMPindexG[ip];]; 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 $r^{(0)}= b-[A]x^{(0)}$
<u>for</u> i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  <u>if</u> i=1
   $\mathbf{p^{(1)}= z^{(0)}}$
  <u>else</u>
   $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
   $\mathbf{p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}}$
  <u>endif</u>
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence $|r|$
<u>end</u>

solve_ICCG_mc (5/6)

```
/***********************************
 * {p}  = {z} if      ITER=0    *
 * BETA = RHO / RHO1  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 / RHO1;
#pragma omp parallel for private(ip,i)
   for(ip=0; ip<PEsmpTOT; ip++) {
        for(i=SMPindexG[ip];]; 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 $i= 1, 2, \ldots$

  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

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

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

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

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

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

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

solve_ICCG_mc (6/6)

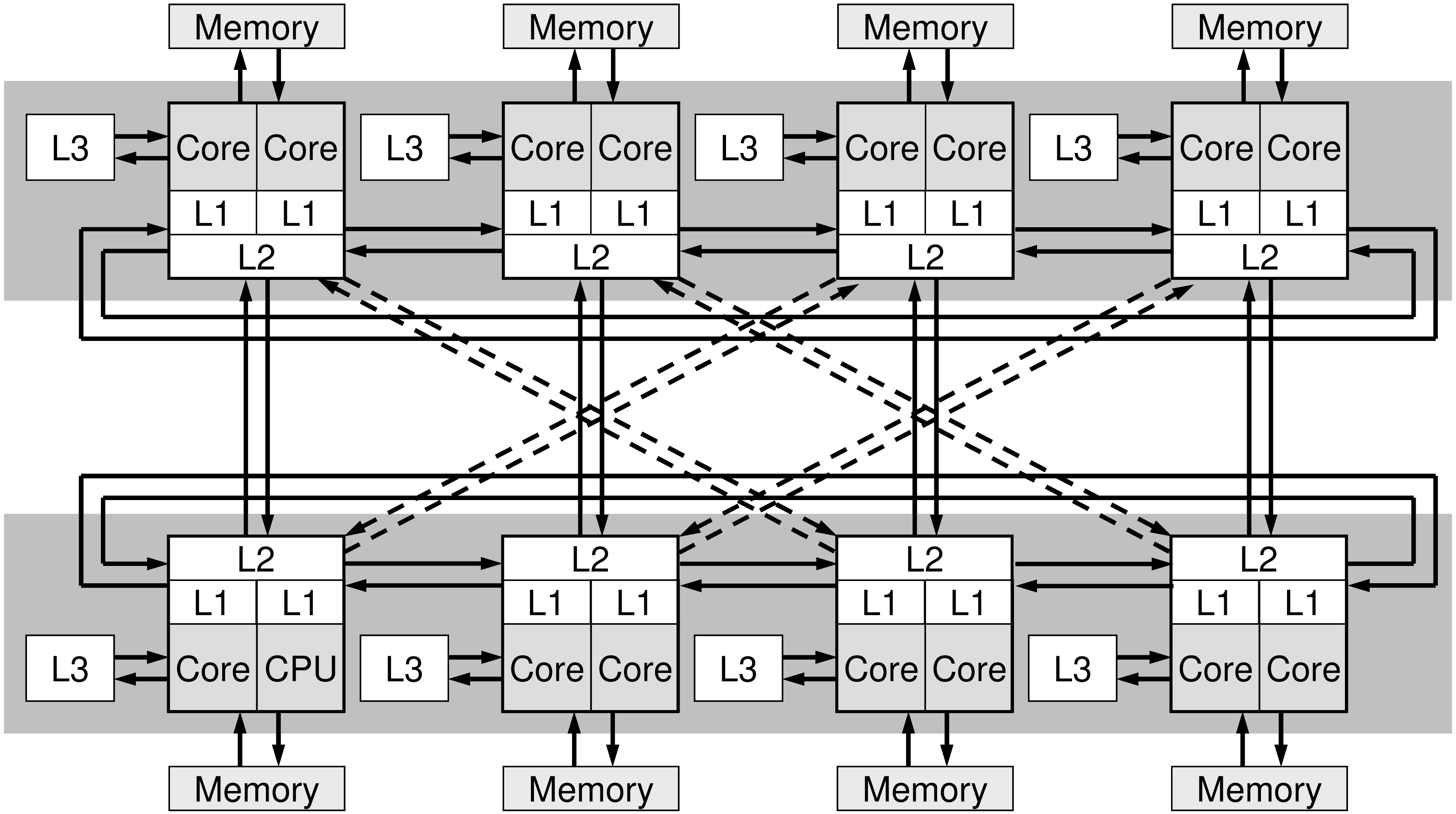
```
/************************
 * ALPHA = RHO / {p}{q} *
 ************************/
   C1 = 0.0;
#pragma omp parallel for private(ip,i)reduction(+:C1)
   for(ip=0; ip<PEsmpTOT; ip++) {
       for(i=SMPindexG[ip]; i<SMPindexG[ip+1]; i++){
           C1 += W[P][i] * W[Q][i];
       }
   }
   ALPHA = RHO / 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 $r^{(0)}= b-[A]x^{(0)}$
<u>for</u> i= 1, 2, …
    solve $[M]z^{(i-1)}= r^{(i-1)}$
    $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
    <u>if</u> i=1
      $p^{(1)}= z^{(0)}$
     <u>else</u>
      $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
      $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
    <u>endif</u>
    $q^{(i)}= [A]p^{(i)}$
    $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
    $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
    $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
    check convergence |r|
<u>end</u>

- L2-solへのOpenMPの実装
- 実行例
- 最適化＋演習

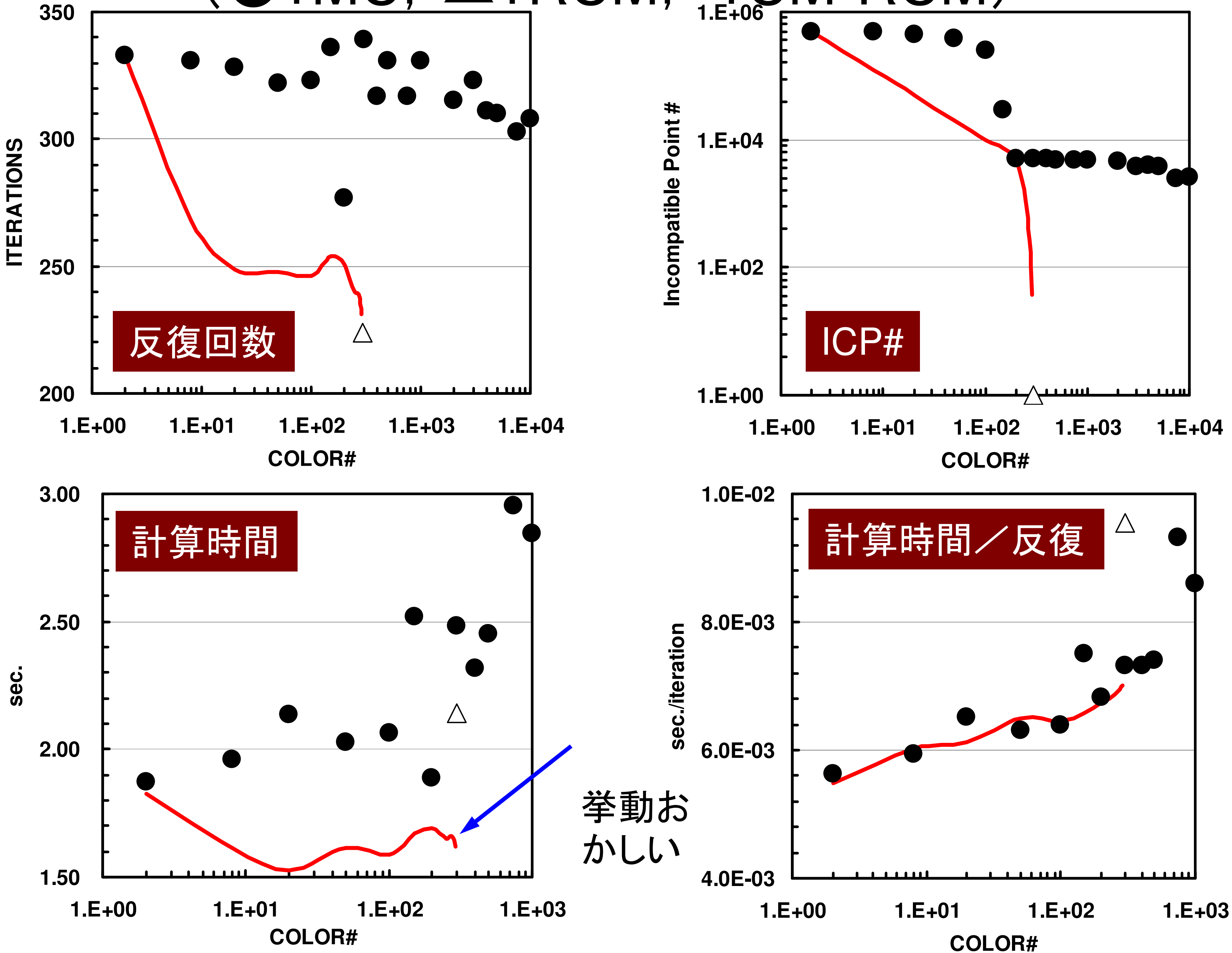
計算結果

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



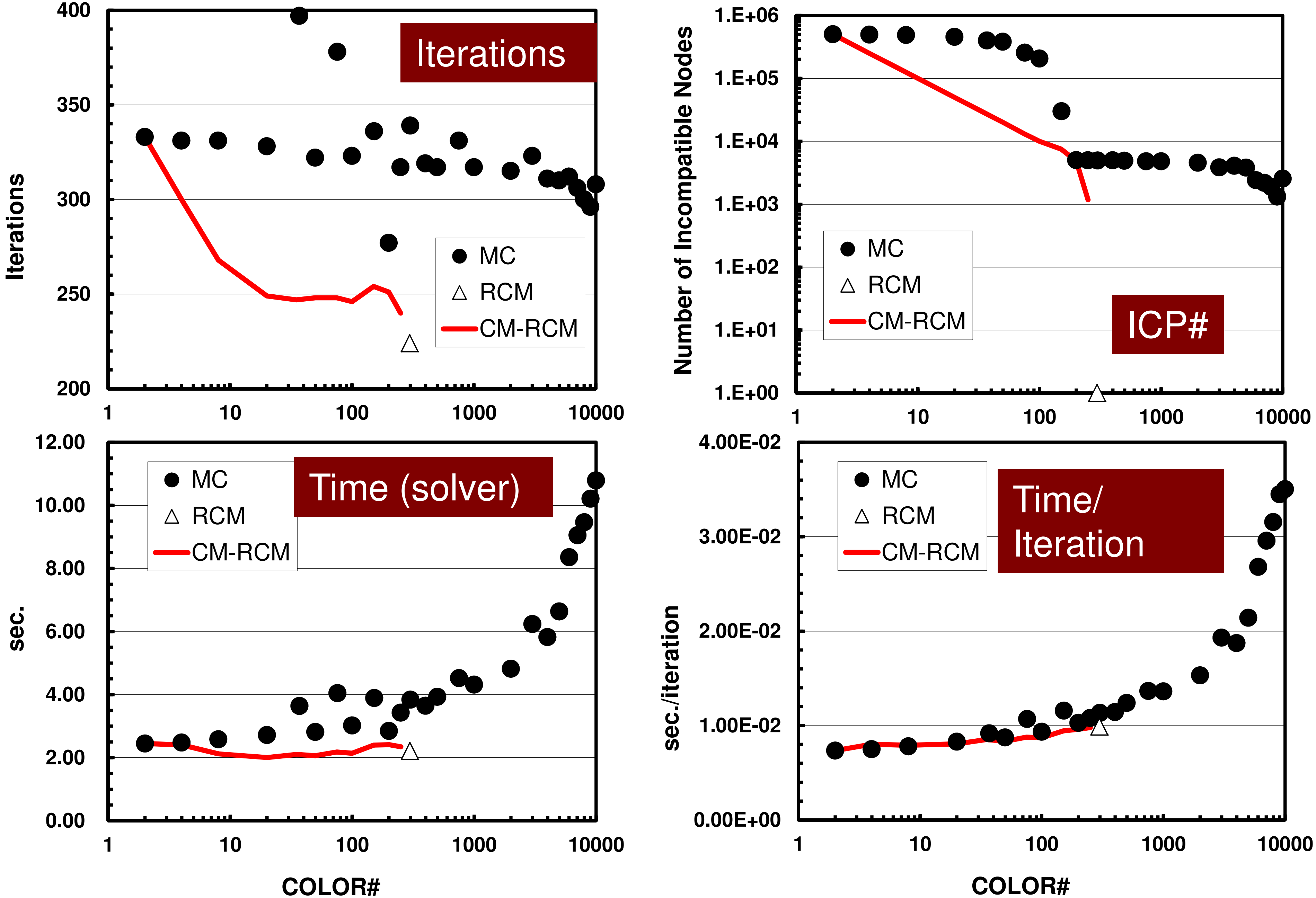
SR11000，16コアにおける結果，100^3

（●：MC，△：RCM，-：CM-RCM）



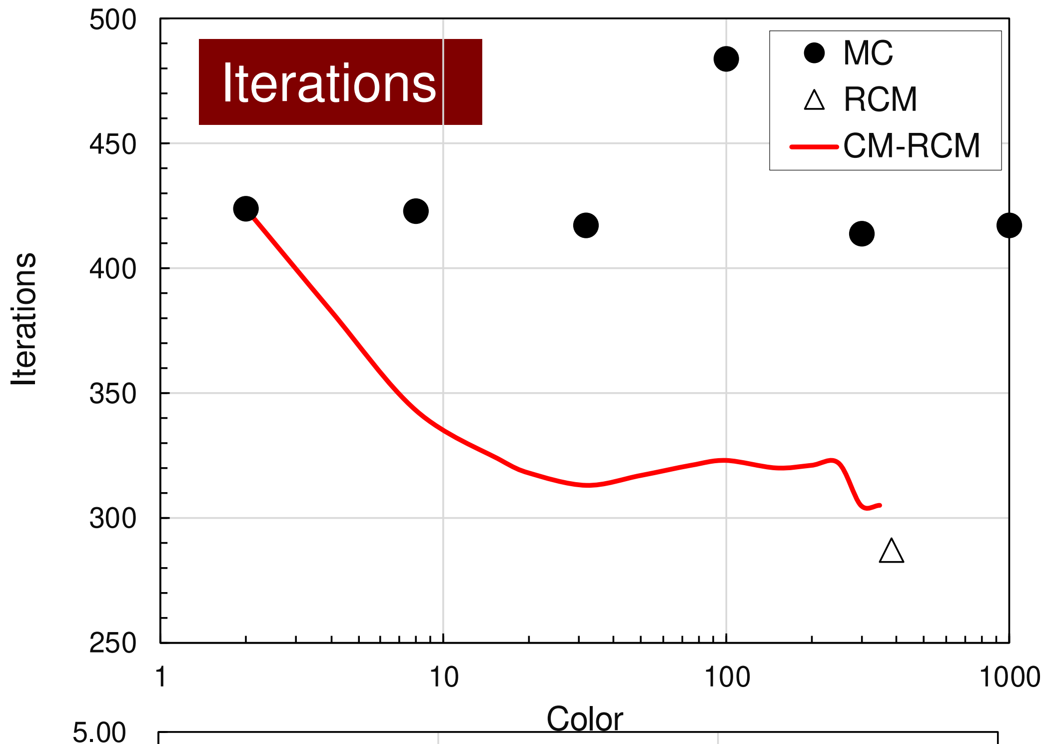
FX10, 1-node/16-cores, 100^3

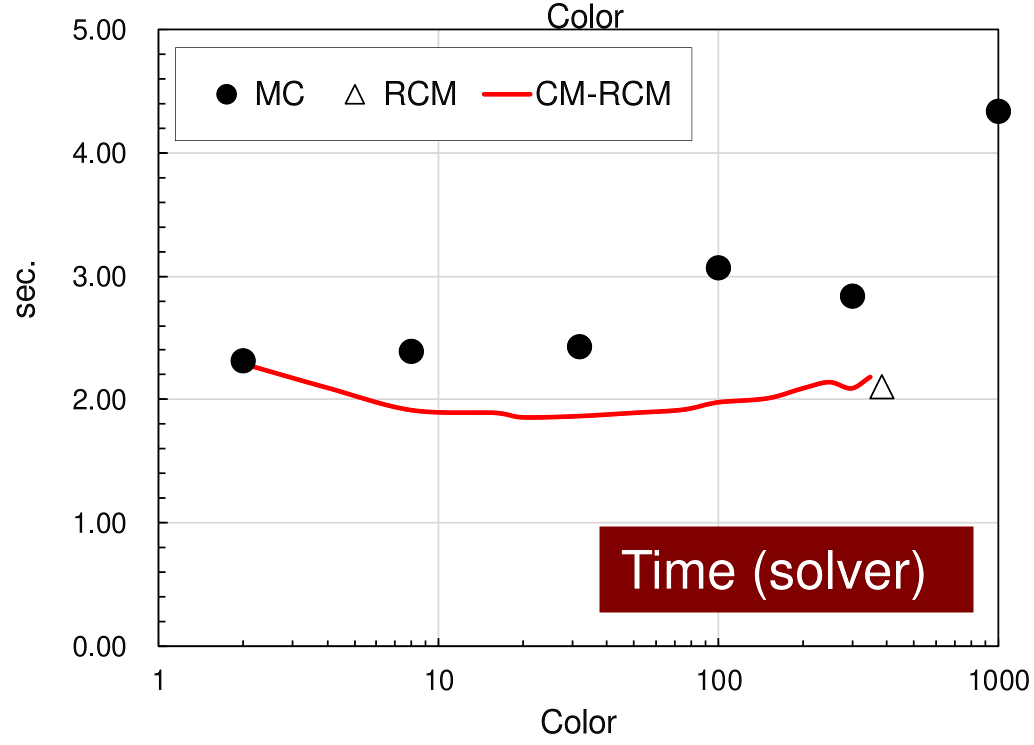
(●:MC, △:RCM, -:CM-RCM)

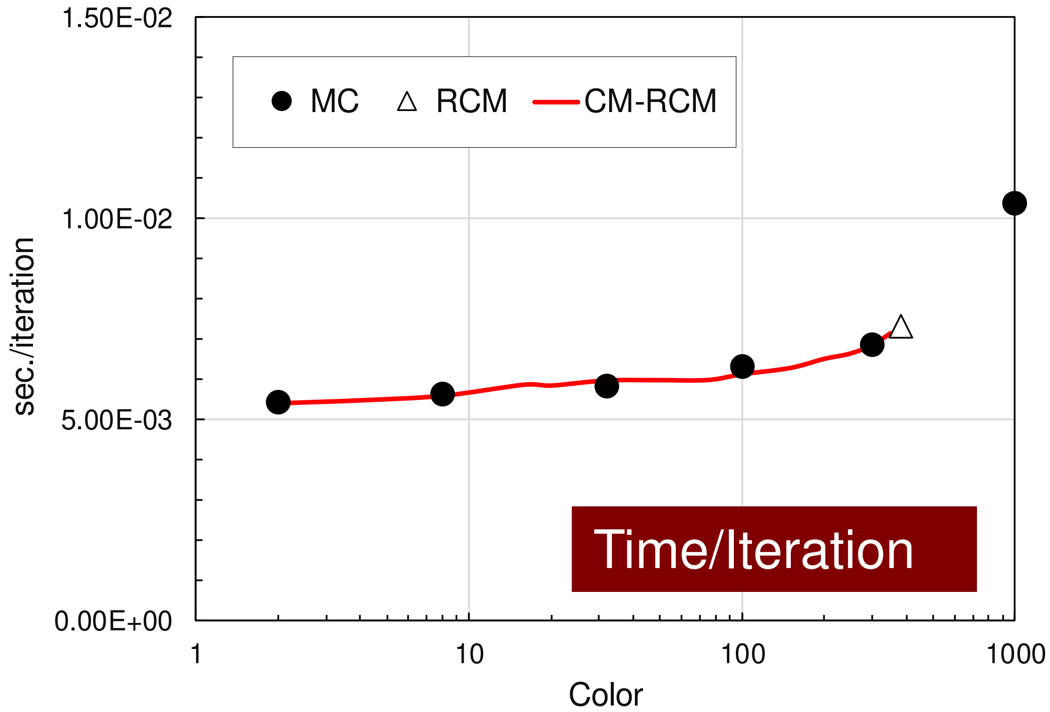


OBCX, 1-soket/24-cores, 128^3

(●：MC, △：RCM, -：CM-RCM)



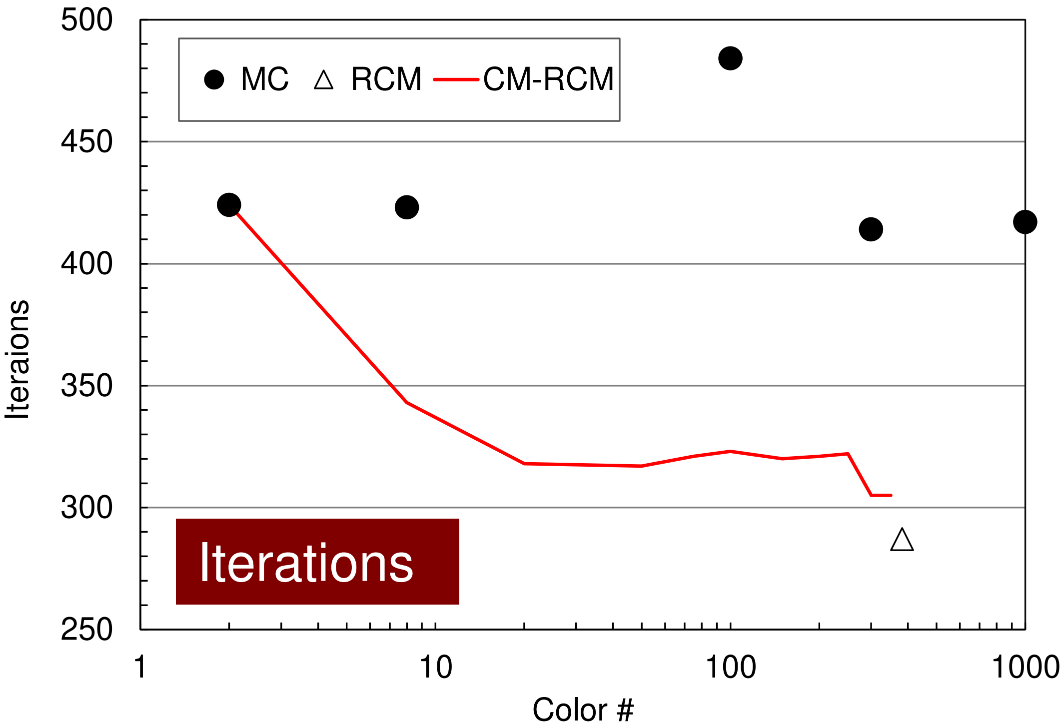


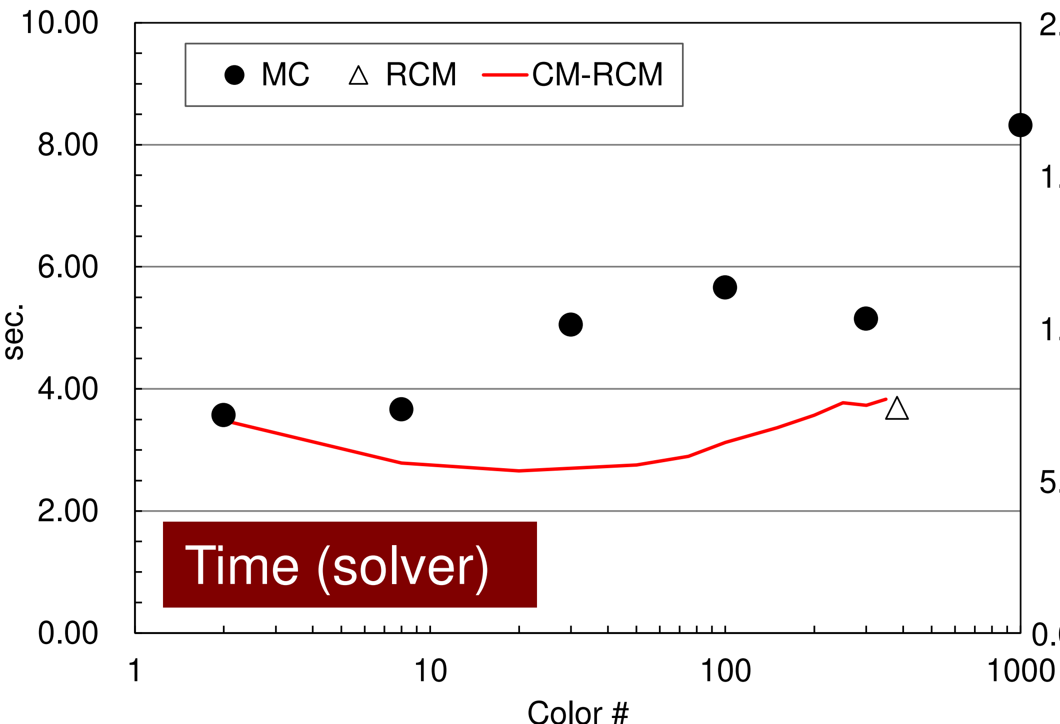


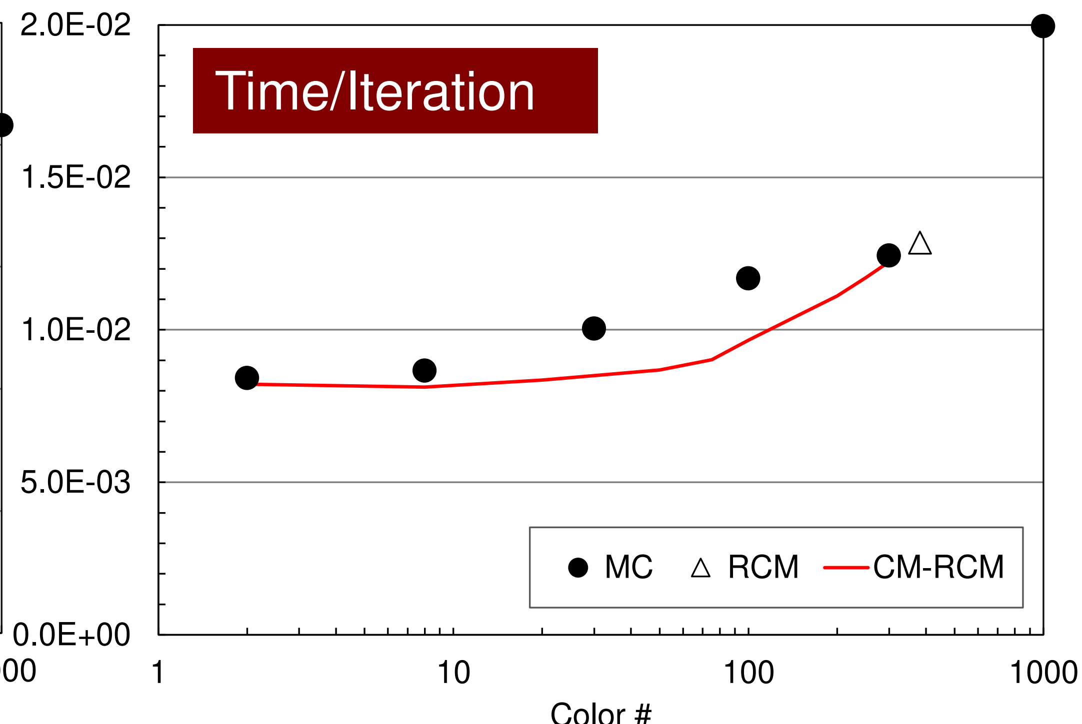
Odyssey

1-CMG/12-cores, 128^3

(●: MC, △: RCM, -: CM-RCM)







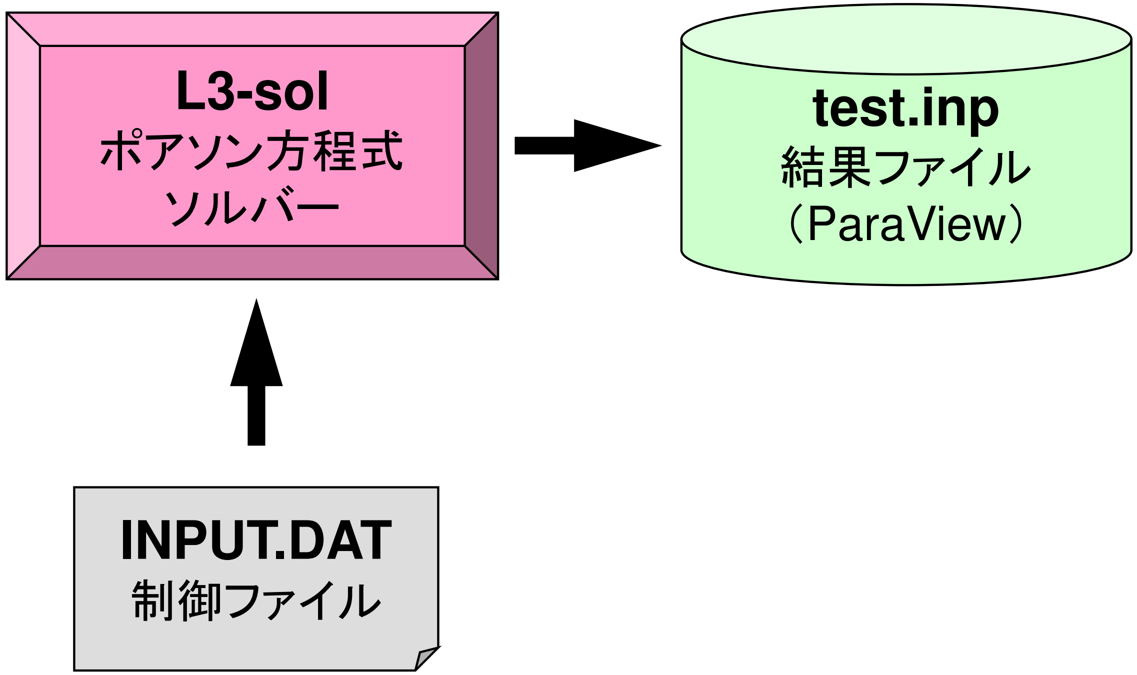
- L2-solへのOpenMPの実装
- 実行例
- 最適化＋演習

- マルチコア版コードの実行
- 更なる最適化

コンパイル・実行

```
>$ cd /work/gt00/t00XXX/ompc/src
>$ module load fj

>$ make
>$ ls ../run/L3-sol

  L3-sol

>$ cd ../run

(modify INPUT.DAT, 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)

```
128 128 128                              NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00               DX/DY/DZ
1.0e-08                                  EPSICCG
24                                       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 (ΔX, ΔY, ΔZ)
EPSICCG	倍精度実数	収束判定値
PEsmpTOT	整数	データ分割数(スレッド数)
NCOLORtot	整数	Ordering手法と色数 ≧2:MC法 (multicolor), 色数 =0:CM法 (Cuthill-Mckee) =-1:RCM法 (Reverse Cuthill-Mckee) ≦-2:CM-RCM法

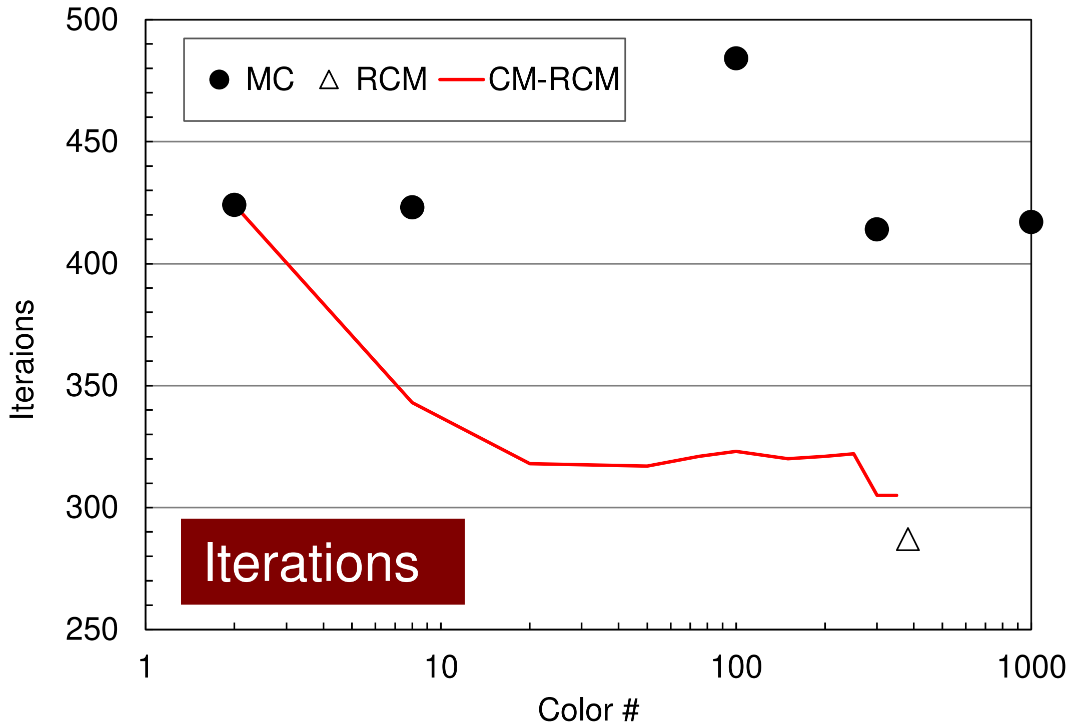
go1.sh

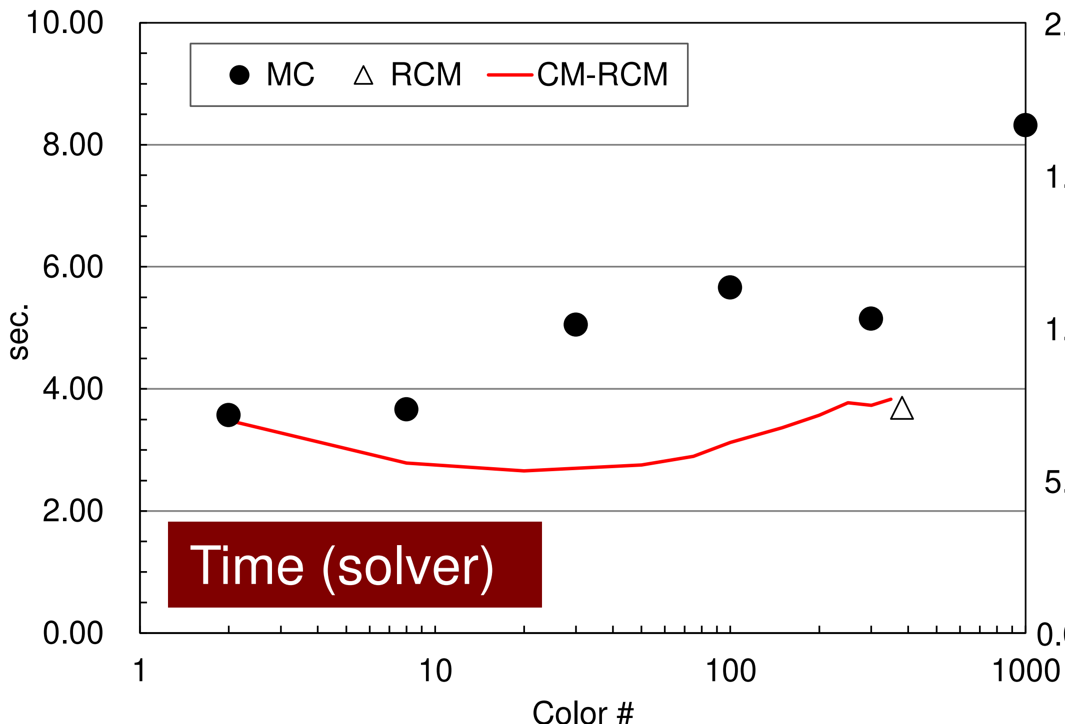
```
#!/bin/sh
#PJM -N "test"
#PJM -L rscgrp=tutorial
#PJM -L node=1
#PJM --omp thread=24     (= PEsmpTOT)
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test1.lst

export KMP_AFFINITY=granularity=fine,compact
./L3-sol
```

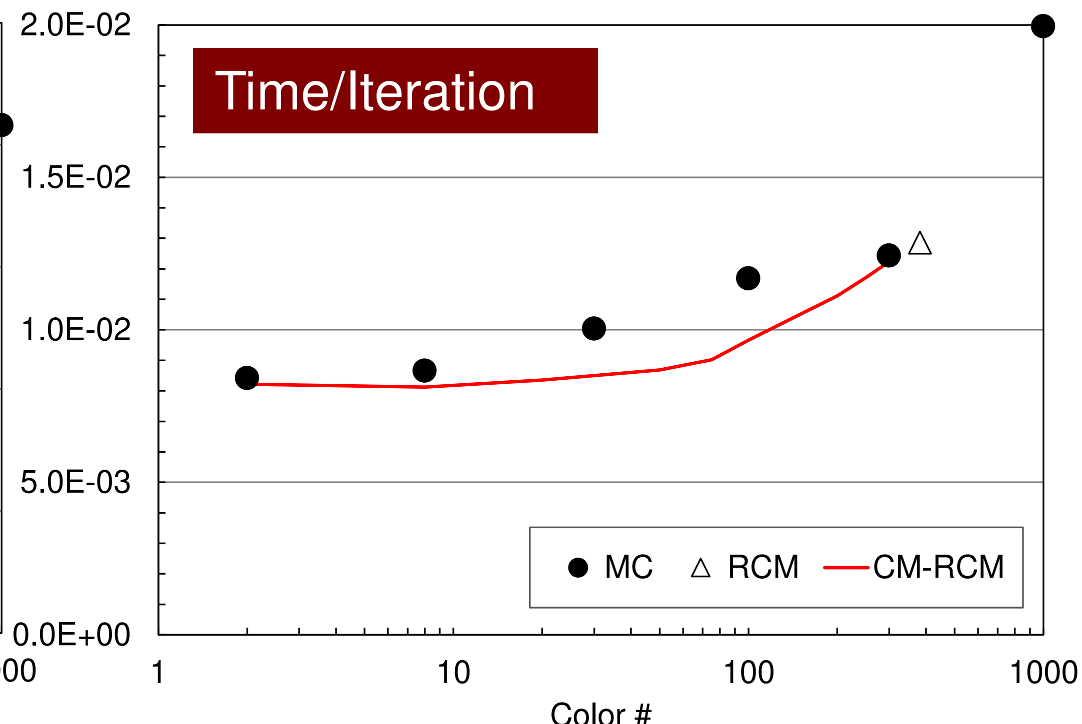
Odyssey

1-CMG/12-cores, 128^3

(●: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

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

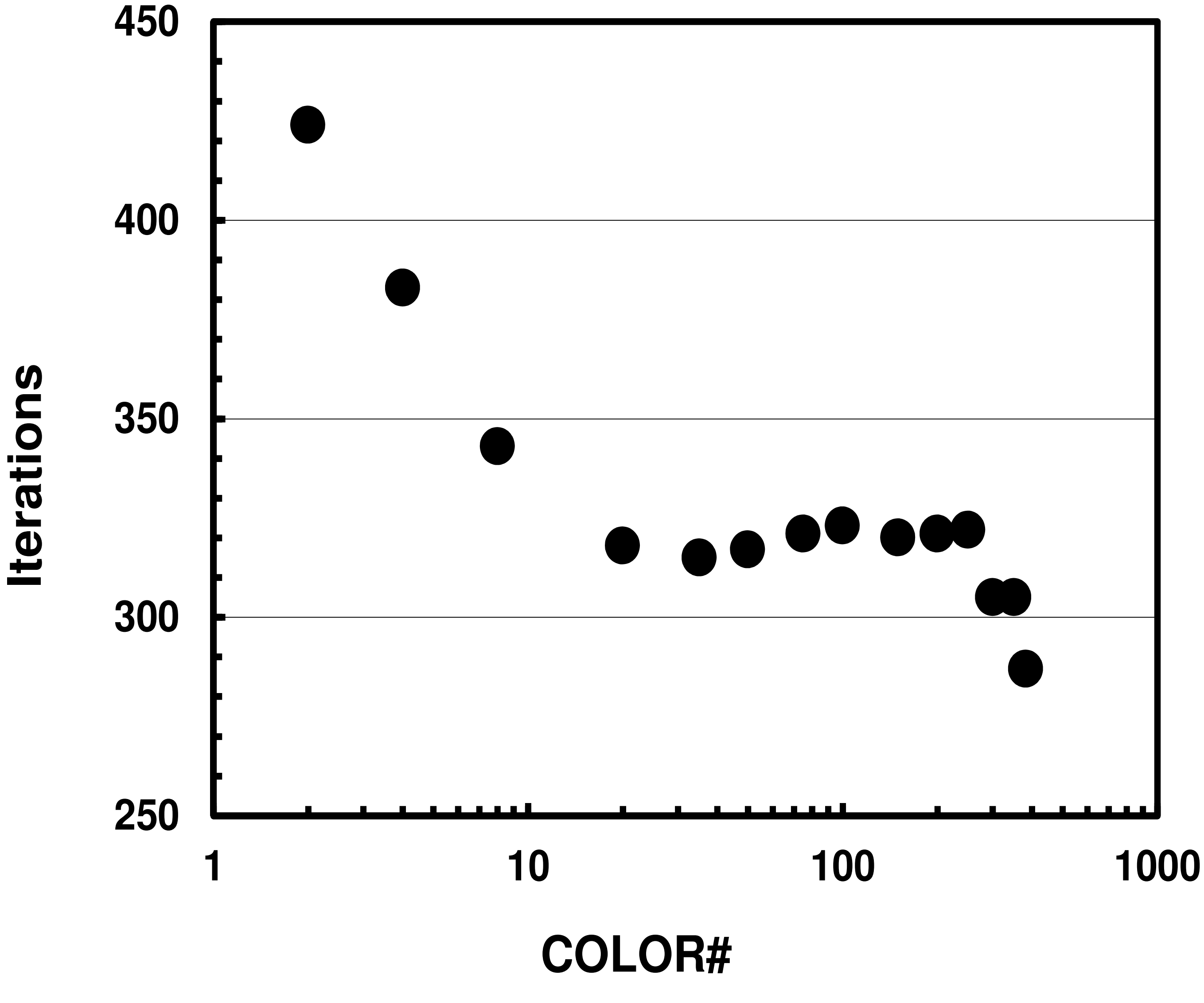
Programs (src0)

```
% cd /work/gt00/t00XXX/ompc/src0
% module load fj

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

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

% pjsub go0.sh
```

go0.sh

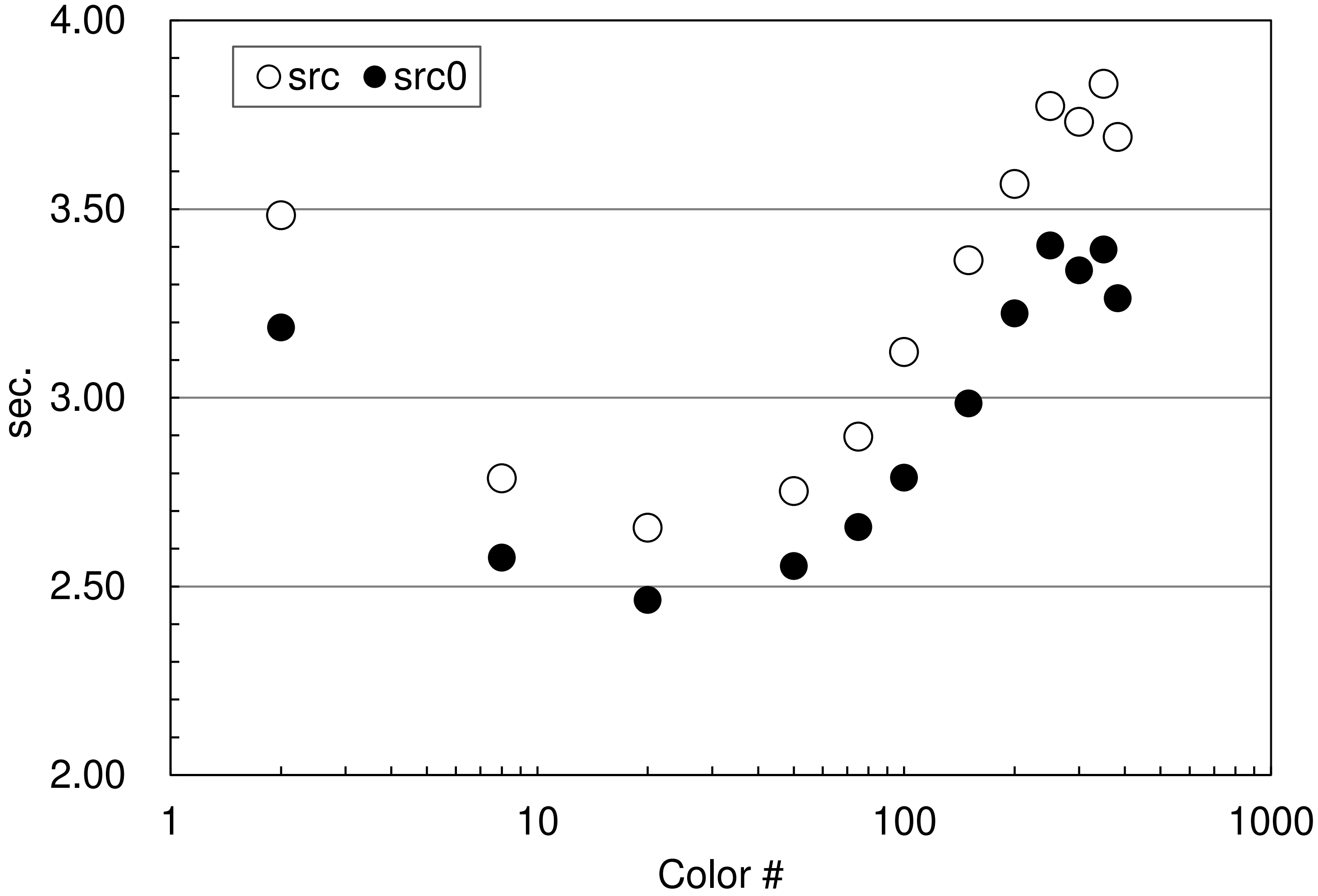
```
#!/bin/sh
#PJM -N "go0"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --omp thread=12            (=PEsmpTOT)
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test0.lst

module load fj
export OMP_NUM_THREADS=12      (=PEsmpTOT)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl -l ./L3-sol0
numactl -C 12-23 -m 4 ./L3-sol0
```

Color#~Iterations for CM-RCM 128^3 case



Time for ICCG Solver: CM-RCM

"src" becomes slower if color# is larger: overhead of fork-join, unstable for many colors (12 threads, C)



- 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**

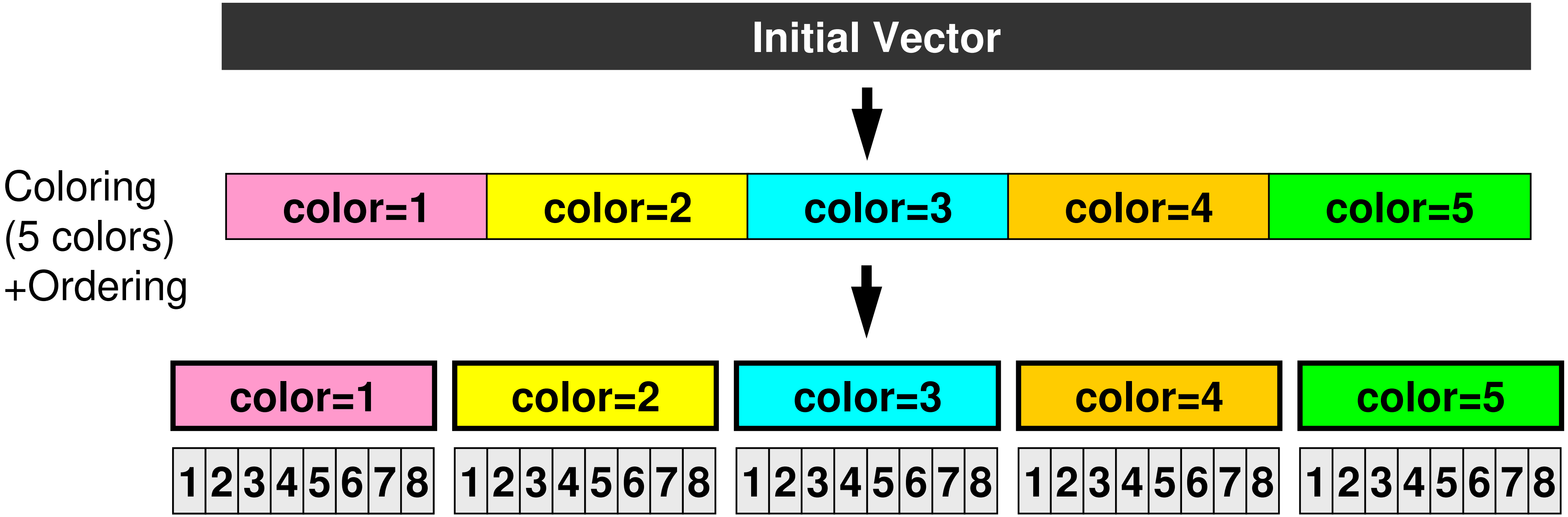
現在のオーダリングの問題

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

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

SMPindex：前処理向け

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



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

データ再配置：Sequential Reordering

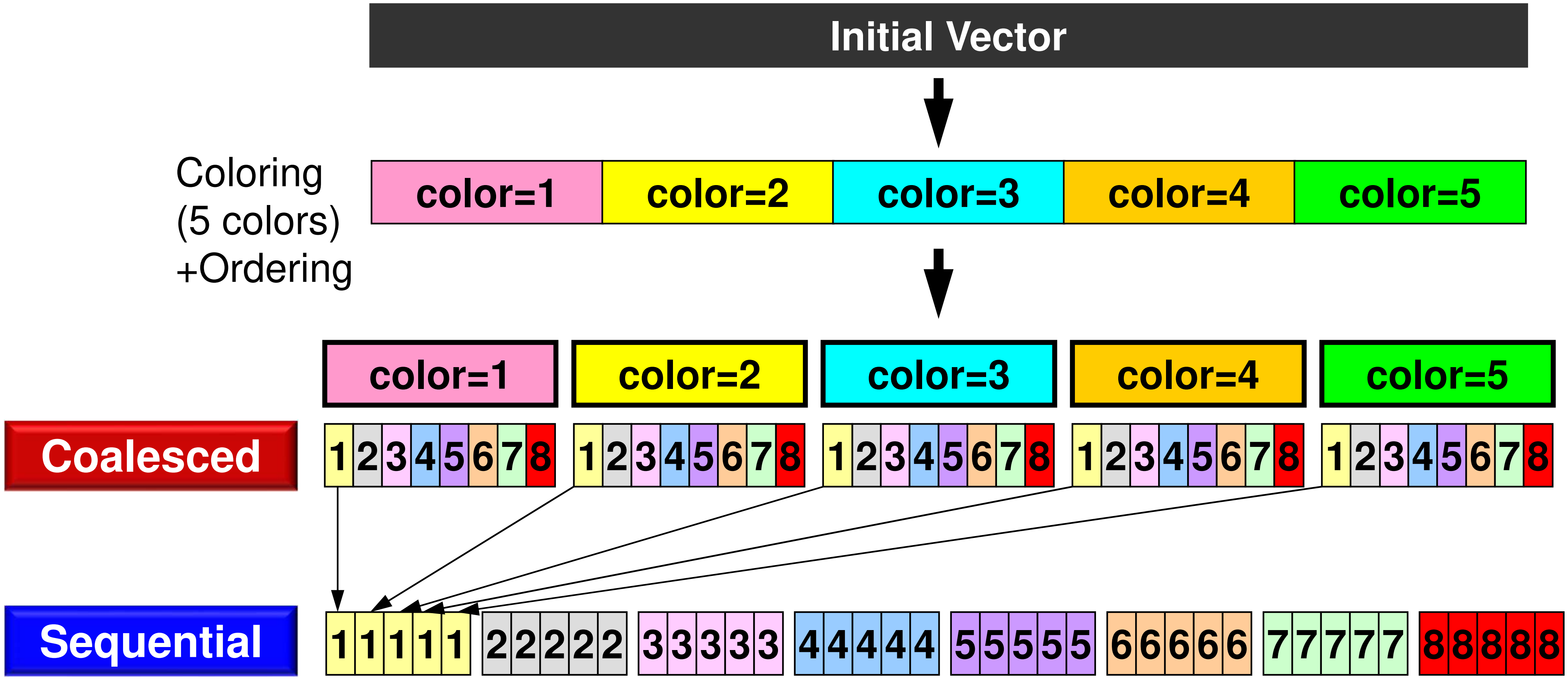
- 同じスレッドで処理するデータをなるべく連続に配置するように更に並び替え
  - 効率の向上が期待される
    - 係数行列等のアドレスが連続になる
    - 局所性が高まる（2ページあと）

- 番号の付け替えによって要素の大小関係は変わるが，上三角，下三角の関係は変えない（もとの計算と反復回数は変わらない）
  - 従って自分より要素番号が大きいのにIAL（下三角）に含まれていたりする

データ再配置: Sequential Reordering

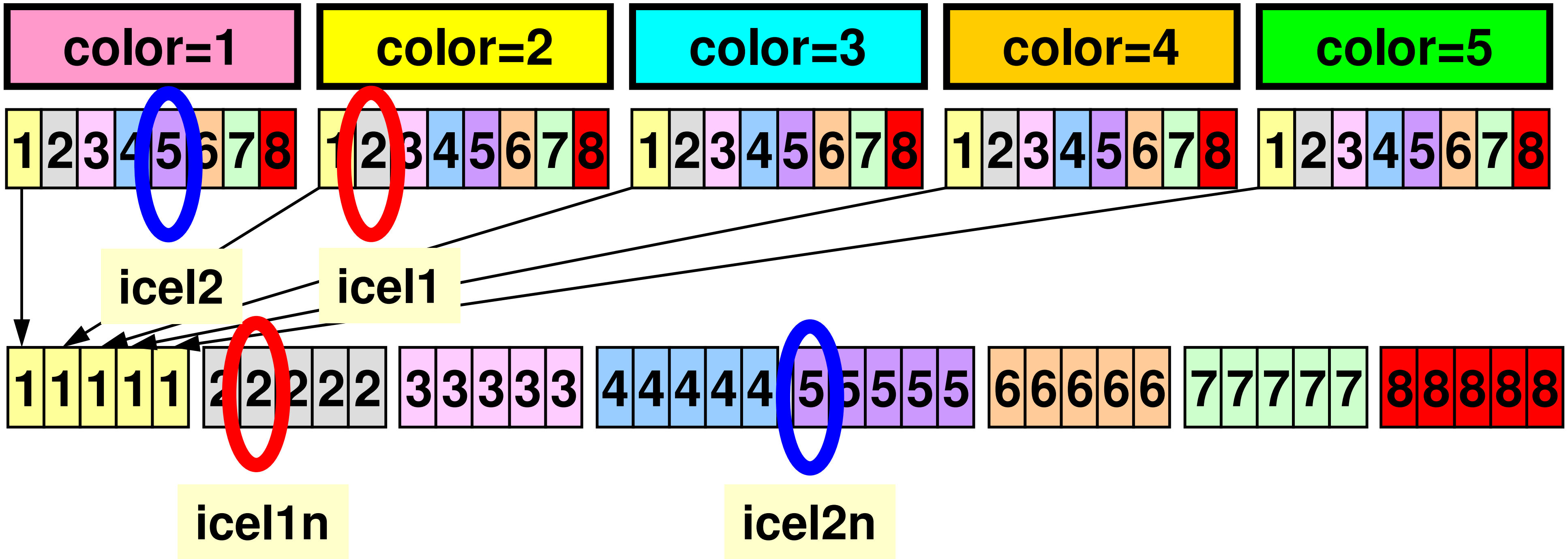
各スレッド上でメモリアクセスが連続となるよう更に並び替え

5 colors, 8 threads



番号付けがinconsistentになる可能性がある

- Coalesced
  - `icel1 > icel2`, therefore, `icel2= itemL[k]`, where `indexL[icel1]`≦`k`<`indexL[icel1+1]`
- Sequential
  - `icel1n < icel2n`, but still `icel2n= itemL[k]`, where `indexL[icel1n]`≦`k`<`indexL[icel1n+2]`



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

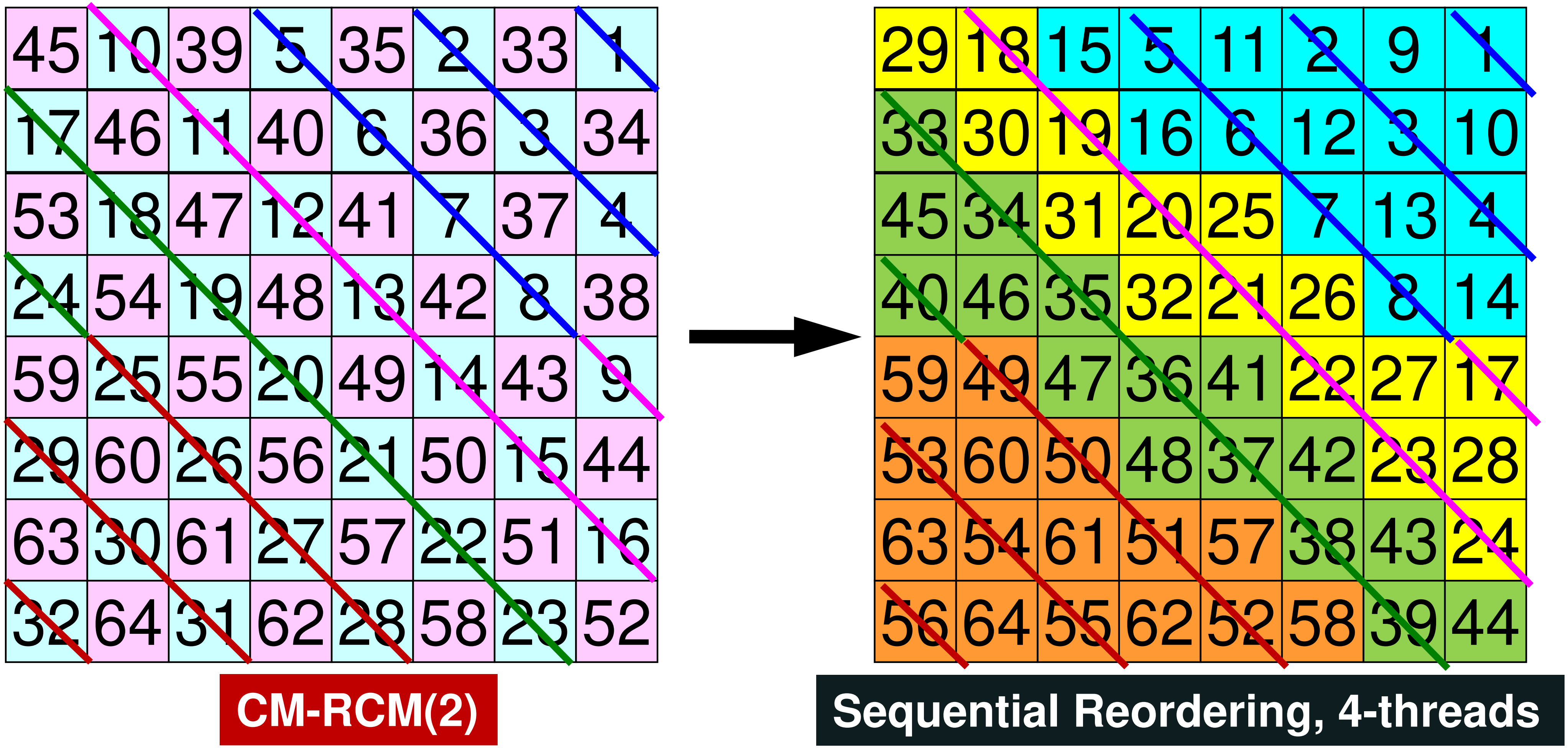
Sequential Reordering, 4-threads

データ再配置：Sequential Reordering

CM-RCM(2), 4-threads

第1色

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



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

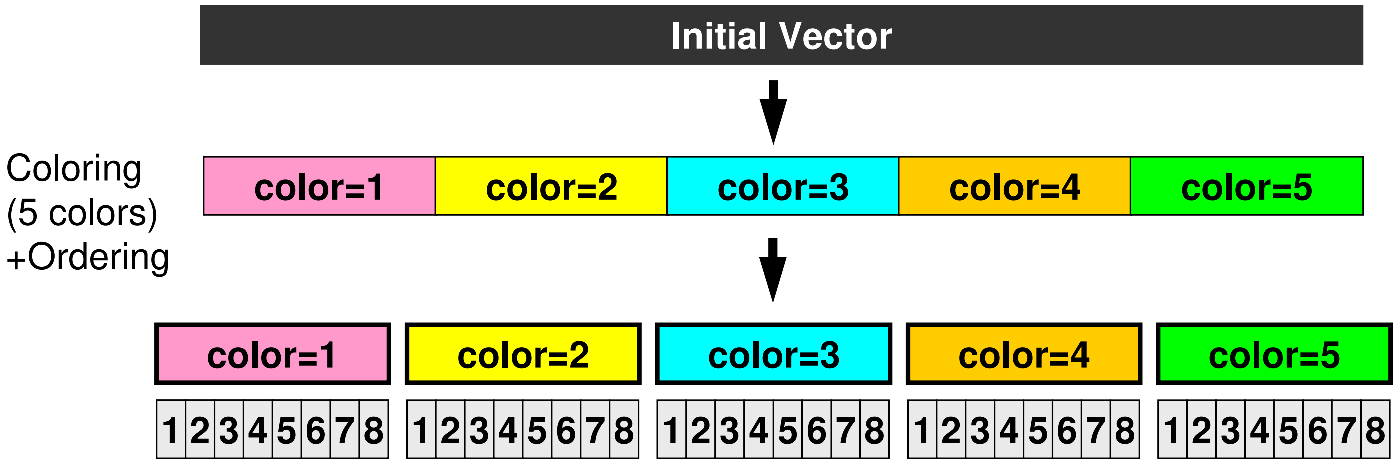
CM-RCM(2)

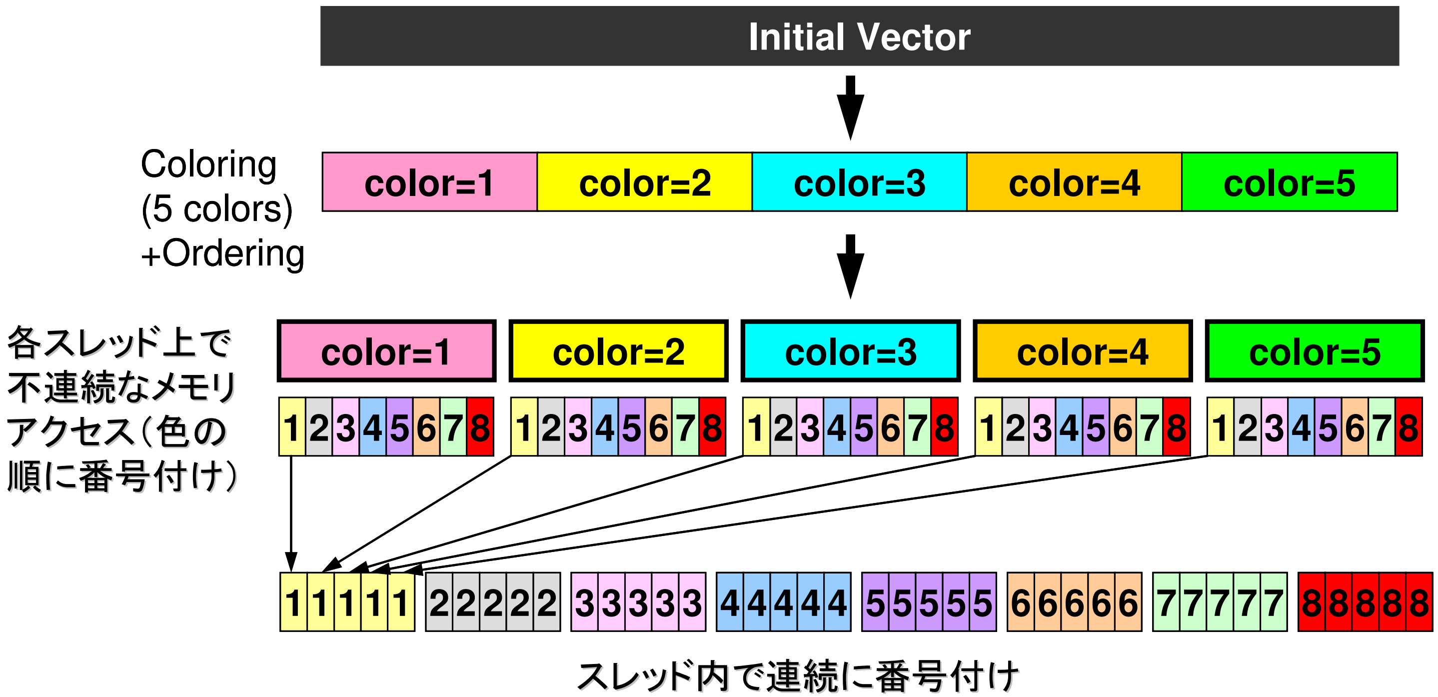
→

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

Programs (reorder0)

```
% cd /work/gt00/t00XXX/ompc/reorder0
% module load fj

% make
% cd ../run
% ls L3-rsol0
    L3-rsol0

<modify “INPUT.DAT”>
<modify “gor.sh”>

% pjsub gor.sh
```

gor.sh

```
#!/bin/sh
#PJM -N "gor"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --omp thread=12                 (=PEsmpTOT)
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o testr.lst

module load fj
export OMP_NUM_THREADS=12            (=PEsmpTOT)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl -l ./L3-rsol0
numactl -C 12-23 -m 4 ./L3-rsol0
```

制御データ（INPUT.DAT）

```
128 128 128                       NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00         DX/DY/DZ
1.0e-08                            EPSICC
24                                 PEsmpTOT
-100                               NCOLORtot
0                                  NFLAG
0                                  METHOD
```

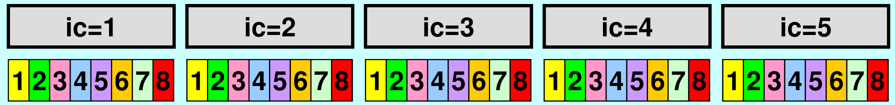
変数名	型	内　容
PEsmpTOT	整数	データ分割数（スレッド数）
NCOLORtot	整数	Ordering手法と色数 ≧2：MC法（multicolor），色数 =0：CM法（Cuthill-Mckee） =-1：RCM法（Reverse Cuthill-Mckee） ≦-2：CM-RCM法
NFLAG	整数	0：First-Touch無し，1：あり 今回は無関係
METHOD	整数	行列ベクトル積のループ構造 （0：従来通り，1：前進後退代入と同じ）

Sequential Reordering (1/5) Main

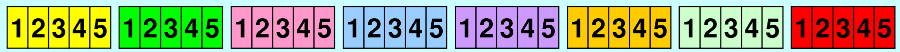
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "struct.h"
#include "pcg.h"
(…)

int main(){
        double *WK;
        int ISET, ITR, IER;
        int icel, ic0, i;
        double 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;
        if(METHOD == 0 ){
                if(solve_ICCG_mc(ICELTOT, NL, NU, indexLnew, itemLnew,
                        indexUnew, itemUnew, D, BFORCE, PHI, ALnew, AUnew,
                        NCOLORtot, PEsmpTOT, SMPindex_new, EPSICCG,
                        &ITR, &IER)) goto error;
        }else if (METHOD == 1){
                if(solve_ICCG_mc_ft(ICELTOT, NL, NU, indexLnew, itemLnew,
                        indexUnew, itemUnew, D, BFORCE, PHI, ALnew, AUnew,
                        NCOLORtot, PEsmpTOT, SMPindex_new, EPSICCG,
                        &ITR, &IER)) goto error;
        }

        for(ic0=0; ic0<ICELTOT; ic0++){
                icel = NEWtoOLDnew[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;
}
```

```
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;
        }
    }
}
```

SMPindex

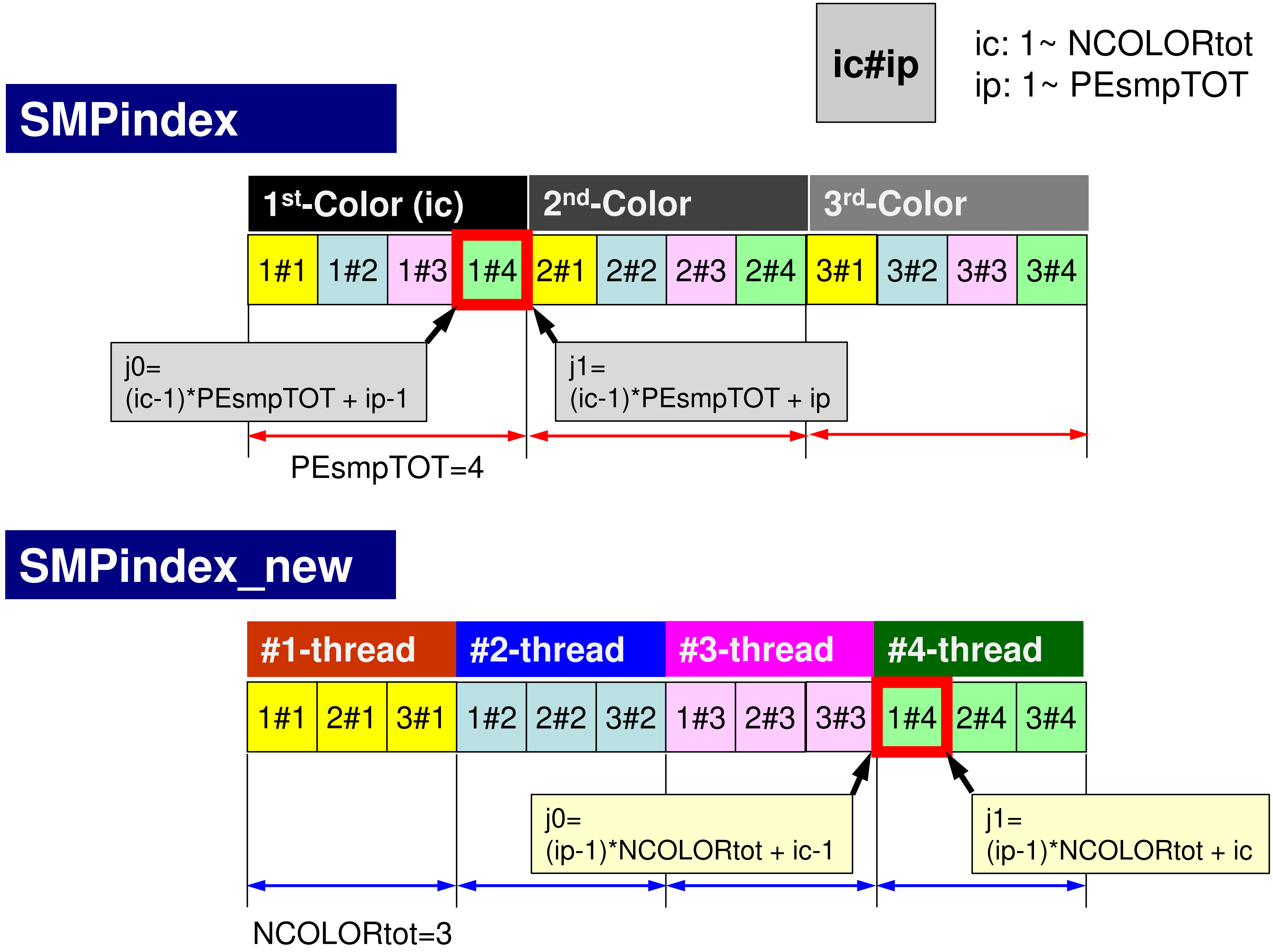


SMPindex_new



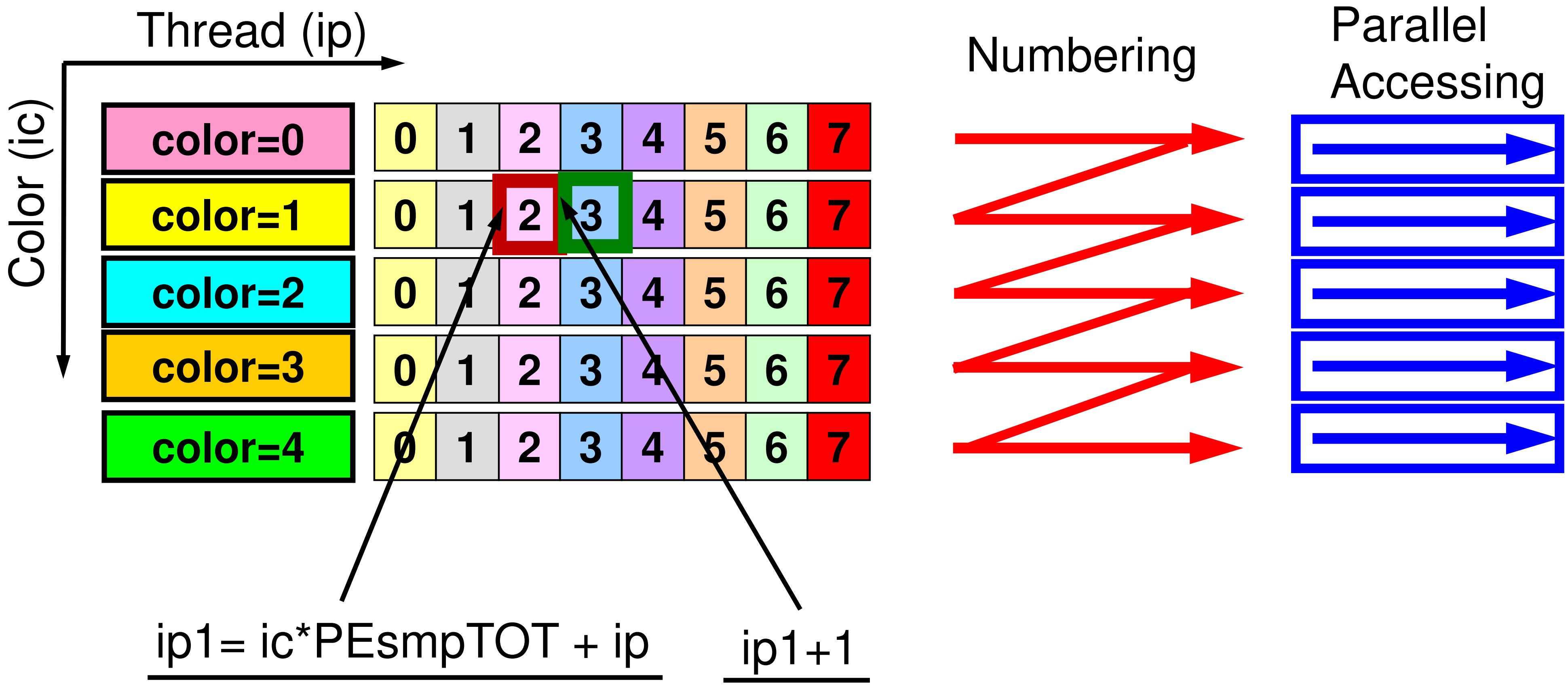

```
SMPindex_new = (int *) allocate_vector(sizeof(int), NCOLORtot * PEsmpTOT + 1);
memset(SMPindex_new, 0, sizeof(int)*(NCOLORtot*PEsmpTOT+1));

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

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

Sequential Reordering (2/5) poi_gen-1


ic#ip
ic: 1~ NCOLORtot
ip: 1~ PEsmpTOT
SMPindex
1st-Color (ic)
2nd-Color
3rd-Color
1#1 1#2 1#3 1#4 2#1 2#2 2#3 2#4 3#1 3#2 3#3 3#4
j0=
(ic-1)*PEsmpTOT + ip-1
j1=
(ic-1)*PEsmpTOT + ip
PEsmpTOT=4
SMPindex_new
#1-thread
#2-thread
#3-thread
#4-thread
1#1 2#1 3#1 1#2 2#2 3#2 1#3 2#3 3#3 1#4 2#4 3#4
j0=
(ip-1)*NCOLORtot + ic-1
j1=
(ip-1)*NCOLORtot + ic
NCOLORtot=3

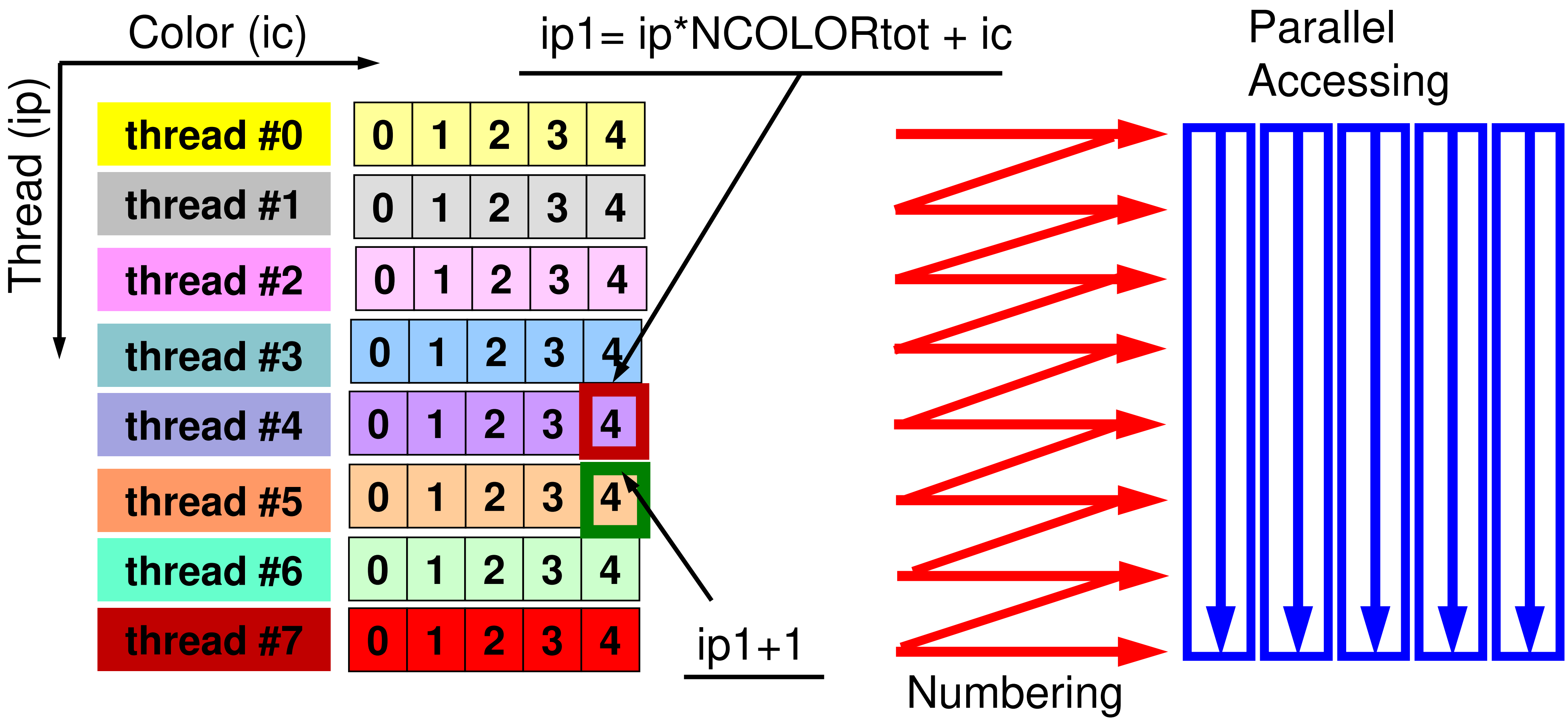
Coalesced

```
#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++) {…
```



Sequential

```
#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 = ip * NCOLORtot + ic;
        for(i=SMPindex[ip1]; i<SMPindex[ip1+1]; i++) {…
```



Sequential Reordering (3/5) poi_gen-2

```
for(ip=0; ip<PEsmpTOT; ip++){
    for(ic=0; ic<NCOLORtot; ic++){
        icNS = SMPindex_new[ip*NCOLORtot + ic];
        ic01 = SMPindex[ic*PEsmpTOT + ip];
        ic02 = SMPindex[ic*PEsmpTOT + ip+1];
        icou = 0;
        for(k=ic01; k<ic02; k++){
            icel=NEWtoOLD[k];
            icou = icou +1;
            icelN=icNS+icou;
            OLDtoNEWnew[icel-1] = icelN;
            NEWtoOLDnew[icelN-1]= icel;
        }
    }
}
```

OLDtoNEWnew: Original -> Sequential
NEWtoOLDnew: Sequential -> Original
-Original: Initial icel
-Sequential icelN

```
indexLnew = (int *)allocate_vector(sizeof(int),ICELTOT+1);
indexUnew = (int *)allocate_vector(sizeof(int),ICELTOT+1);
INLnew = (int *)allocate_vector( sizeof(int),ICELTOT);
INUnew = (int *)allocate_vector( sizeof(int),ICELTOT);
indexLnew_org = (int *)allocate_vector(sizeof(int),ICELTOT+1);
indexUnew_org = (int *)allocate_vector(sizeof(int),ICELTOT+1);

for(ip=1; ip<=PEsmpTOT; ip++) {
    id1 = ip    *NCOLORtot;
    id2 = (ip-1)*NCOLORtot;

    for(icel=SMPindex_new[id2]+1; icel<=SMPindex_new[id1]; icel++) {
        ic0  = NEWtoOLDnew[icel-1];
        ik0  = OLDtoNEW[ic0-1];
        INLnew[icel-1] =INL[ik0-1];
        INUnew[icel-1] =INU[ik0-1];
    }
}
```

Sequential **Coalesced**

-Original: Initial ic0
-Coalesced ik0
-Sequential icel

```
for(i=1; i<=ICELTOT; i++){
    indexLnew_org[i]=indexLnew_org[i-1]+INLnew[i-1];
    indexUnew_org[i]=indexUnew_org[i-1]+INUnew[i-1];
}
```

Sequential Reordering (4/5) poi_gen-3

```
/*************
* ARRAY init.
*************/
        if(NFLAG == 0){
                for(i=0; i<ICELTOT; i++) {
                        BFORCE[i] = 0.0;
                        D[i]      = 0.0;
                        PHI[i]    = 0.0;
                }
                for(i=0; i<=ICELTOT; i++) {
                        indexLnew[i] = indexLnew_org[i];
                        indexUnew[i] = indexUnew_org[i];

                }
                for(i=0; i<NPL; i++) {
                        itemLnew[i] = 0;
                        ALnew[i] = 0.0;
                }
                for(i=0; i<NPU; i++) {
                        itemUnew[i] = 0;
                        AUnew[i] = 0.0;
                }

        }else {
                indexLnew[0]=0;
                indexUnew[0]=0;
#pragma omp parallel for private (icel, j)
                for(ip=1; ip<=PEsmpTOT; ip++){
                        for(icel = SMPindex_new[(ip-1)*NCOLORtot]+1; icel<=SMPindex_new[ip*NCOLORtot]; icel++){
                                BFORCE[icel-1] = 0.0;
                                PHI[icel-1] = 0.0;
                                D[icel-1] = 0.0;
                                indexLnew[icel]=indexLnew_org[icel];
                                indexUnew[icel]=indexUnew_org[icel];

                                for(j=indexLnew_org[icel-1];j<indexLnew_org[icel];j++){
                                        itemLnew[j]=0;
                                        ALnew[j] = 0.0;
                                }
                                for(j=indexUnew_org[icel-1];j<indexUnew_org[icel];j++){
                                        itemUnew[j]=0;
                                        AUnew[j] = 0.0;
                                }
                        }
                }
        }
```

Sequential Reordering (5/5) poi_gen-4

```
#pragma omp parallel for private (icel, id1, id2 …)
for(ip=1; ip<=PEsmpTOT; ip++) {
    id1= ip    *NCOLORtot; id2= (ip-1)*NCOLORtot;
    for(icel=SMPindex_new[id2]+1; icel<=SMPindex_new[id1]; icel++) {
        ic0  = NEWtoOLDnew[icel-1];
        ik0  = OLDtoNEW[ic0-1];
        icN10 = NEIBcell[ic0-1][0];
(…)
        icN50 = NEIBcell[ic0-1][4];
        icN60 = NEIBcell[ic0-1][5];
        isL=indexL[ik0-1]; ieL=indexL[ik0  ];
        isU=indexU[ik0-1]; ieU=indexU[ik0  ];

        if(icN50 != 0) {
           icN5 = OLDtoNEW[icN50-1];
           coef = RDZ * ZAREA;
           D[icel-1] -= coef;
           if(icN5 < ik0) {
              for(j=isL; j<ieL; j++) {
                  if(itemL[j] == icN5) {
                     j_new=indexLnew[icel-1]+j-isL;
                     ALnew[j_new]    = coef;
                     itemLnew[j_new] = OLDtoNEWnew[icN50-1];
                     break;
                  }
              }
           } else {
              for(j=isU; j<ieU; j++) {
                  if(itemU[j] == icN5) {
                     j_new=indexUnew[icel-1]+j-isU;
                     AUnew[j_new] = coef;
                     itemUnew[j_new] = OLDtoNEWnew[icN50-1];
                     break;
                  }
              }
           }
    }
```

icel: Sequential
ic0: Original
ik0: Coalesced

icN50: Original
icN5 : Coalesced

icN5>ik0: Upper (AU)
icN5<ik0: Lower (AL)

前進代入の計算法

```
#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];
        }
    }
}
```

Color #1	Thread #0-#(Pe-1)
Color #2	Thread #0-#(Pe-1)
Color #3	Thread #0-#(Pe-1)
Color #4	Thread #0-#(Pe-1)
:	
Color #Nc	Thread #0-#(Pe-1)

Coalesced

```
#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 = 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];
       }
   }
}
```

Thread #0	Color #1-#(Nc)
Thread #1	Color #1-#(Nc)
Thread #2	Color #1-#(Nc)
Thread #3	Color #1-#(Nc)
:	
Thread #(Pe-1)	Color #1-#(Nc)

Sequential

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;
    }
}
```

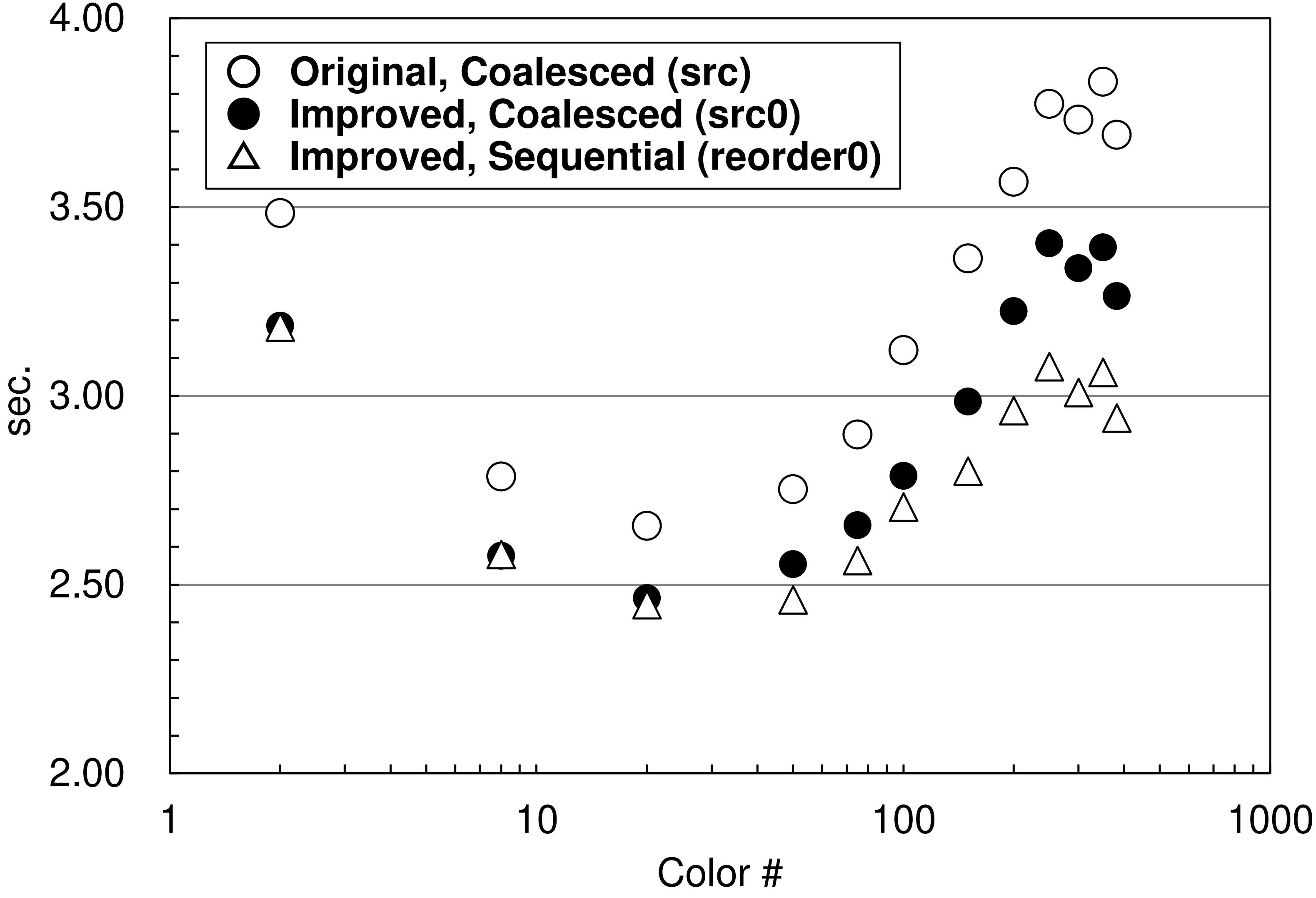
METHOD=0

```
#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

Comp. Time for ICCG, CM-RCM

Generally “sequential (reorder0)” is stable and faster than “coalesced (src, src0)”. Effects aremore significant in cases with more colors (12 threads, C)



データをローカルメモリに置く方法（1/2）

- NUMAアーキテクチャでは，プログラムにおいて変数や配列を宣言した時点では，物理的メモリ上に記憶領域は確保されず，ある変数を最初にアクセスしたコア（の属するソケット）のローカルメモリ上に，その変数の記憶領域（ページ）が確保される。
- これをFirst Touch Data Placement（First Touch）と呼び，配列の初期化手順により得られる性能が大幅に変化する場合があるため，注意が必要である。
- 例えばある配列を初期化する場合，特に指定しなければ0番のソケットで初期化が行われるため，記憶領域は0番ソケットのローカルメモリ上に確保される。

データをローカルメモリに置く方法(2/2)

- したがって，他のソケットでこの配列のデータをアクセスする場合には，必ず0番ソケットのメモリにアクセスする必要があるため，高い性能を得ることは困難である。
- 配列の初期化を，実際の計算の手順にしたがってOpenMPを使って並列に実施すれば，実際に計算を担当するソケットのメモリにその配列の担当部分の記憶領域が確保され，より効率的に計算を実施することができる。
- 1ソケットしか使用しない場合はこのような配慮は不要
  - OpenMP/MPIハイブリッドで1CPU(ソケット)当りに1つのMPIプロセスを使用する場合も同様

First Touch Data Placement

“Patterns for Parallel Programming” Mattson, T.G. et al.

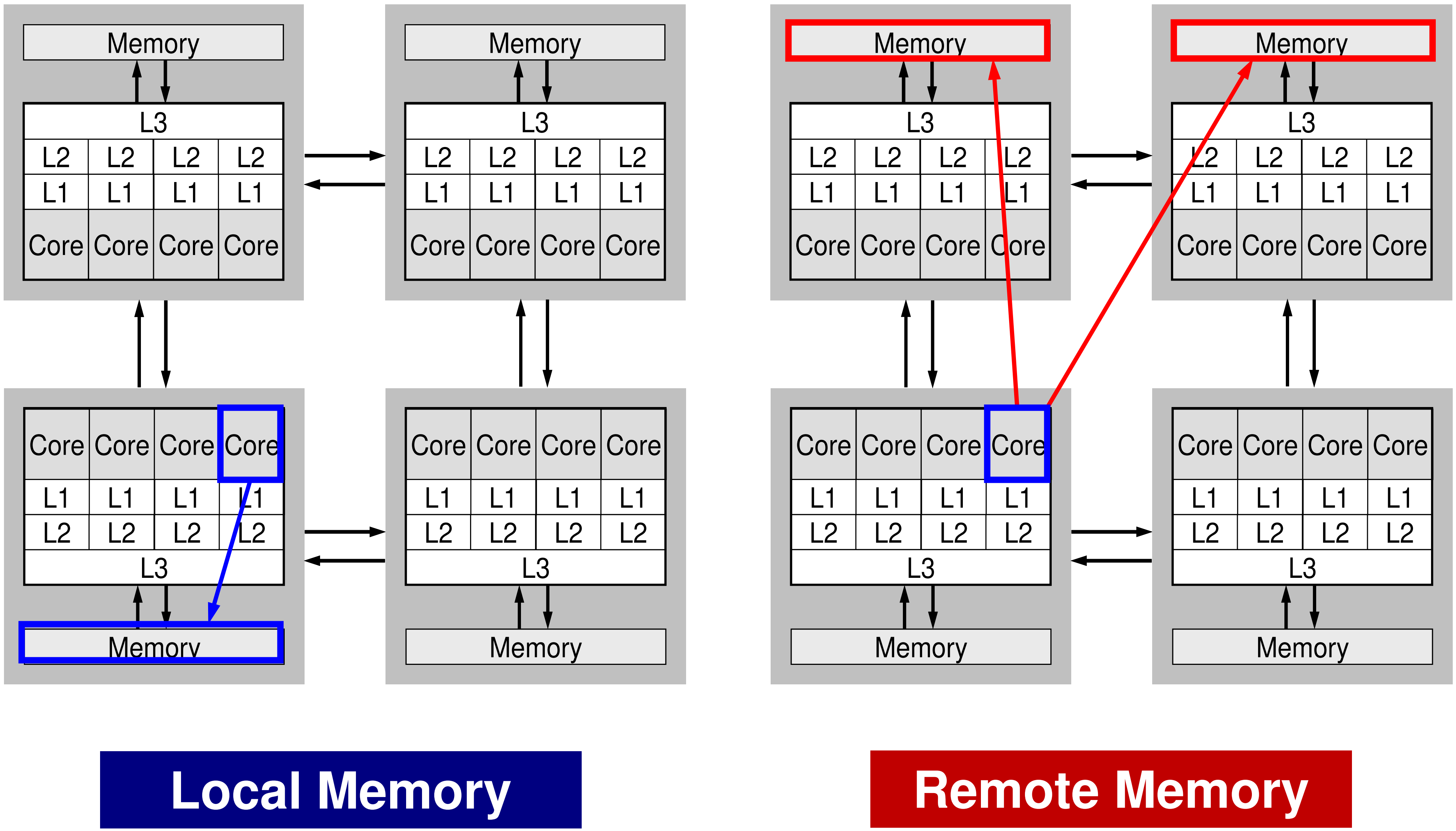
To reduce memory traffic in the system, it is important to keep the data close to the PEs that will work with the data (e.g. NUMA control).

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

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

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

Local/Remote Memory



制御データ(INPUT.DAT)

```
128 128 128                          NX/NY/NZ
1.00e-00 1.00e-00 1.00e-00           DX/DY/DZ
1.0e-08                              EPSICC
48                                   PEsmpTOT
-50                                  NCOLORtot
0                                    NFLAG (0 or 1)
0                                    METHOD
```

変数名	型	内　容
PEsmpTOT	整数	データ分割数（スレッド数）
NCOLORtot	整数	Ordering手法と色数 ≧2：MC法（multicolor），色数 =0：CM法（Cuthill-Mckee） =-1：RCM法（Reverse Cuthill-Mckee） ≦-2：CM-RCM法
NFLAG	整数	0：First-Touch無し，1：あり
METHOD	整数	行列ベクトル積のループ構造 （0：従来通り，1：前進後退代入と同じ）

g.sh: reorder0

```
#!/bin/sh
#PJM -N "go0"
#PJM -L rscgrp=tutorial-o
#PJM -L node=1
#PJM --omp thread=48                (=PEsmpTOT)
#PJM -L elapse=00:15:00
#PJM -g gt00
#PJM -j
#PJM -e err
#PJM -o test1.lst

module load fj
export OMP_NUM_THREADS=48      (=PEsmpTOT)
export XOS_MMM_L_PAGING_POLICY=demand:demand:demand

numactl -l ./L3-rsol0
numactl -C 12-59 -m 4-7 ./L3-rsol0
```

配列の初期化（NFLAG=0/1）（1/3） poi_gen.c

```
  if(NFLAG == 0){
                for(i=0; i<ICELTOT; i++) {
                        OLDtoNEWnew[i] = 0;
                        NEWtoOLDnew[i] = 0;
                }
        }else {
#pragma omp parallel for private (icel, j)
                for(ip=1; ip<=PEsmpTOT; ip++){
                        for(icel = SMPindex_new[(ip-1)*NCOLORtot]+1;
                            icel<= SMPindex_new[ip*NCOLORtot]; icel++){
                                OLDtoNEWnew[icel-1] = 0;
                                NEWtoOLDnew[icel-1] = 0;
                        }
                }
        }
```

Pages are allocated at the local memory of the master thread

Pages are allocated at the local memory of each thread

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

配列の初期化（NFLAG=0/1）（2/3）poi_gen.c

```
if(NFLAG == 0){
   for(i=0; i<ICELTOT; i++) {
       BFORCE[i] = 0.0;
       D[i]      = 0.0;
       PHI[i]    = 0.0;
   }
   for(i=0; i<=ICELTOT; i++) {
       indexLnew[i] = indexLnew_org[i];
       indexUnew[i] = indexUnew_org[i];
   }
   for(i=0; i<NPL; i++) {
       itemLnew[i] = 0;
       ALnew[i] = 0.0;
   }
   for(i=0; i<NPU; i++) {
       itemUnew[i] = 0;
       AUnew[i] = 0.0;
   }

   }else {
```

Pages are allocated at the local memory of the master thread

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

配列の初期化（NFLAG=0/1）（3/3）

```
    }else {
            indexLnew[0]=0;
            indexUnew[0]=0;
#pragma omp parallel for private (icel, j)
     for(ip=1; ip<=PEsmpTOT; ip++){
         for(icel = SMPindex_new[(ip-1)*NCOLORtot]+1;
             icel<=SMPindex_new[ip*NCOLORtot]; icel++){
             BFORCE[icel-1] = 0.0;
             PHI[icel-1] = 0.0;
             D[icel-1] = 0.0;
             indexLnew[icel]=indexLnew_org[icel];
             indexUnew[icel]=indexUnew_org[icel];

             for(j=indexLnew_org[icel-1];j<indexLnew_org[icel];j++){
                 itemLnew[j]=0;
                 ALnew[j] = 0.0;
             }
             for(j=indexUnew_org[icel-1];j<indexUnew_org[icel];j++){
                 itemUnew[j]=0;
                 AUnew[j] = 0.0;
             }
         }
     }
  }
```

Pages are allocated at the local memory of each thread

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

Sequential Reordering (4/5) poi_gen-3

```
/*************
* ARRAY init.
*************/
        if(NFLAG == 0){
                for(i=0; i<ICELTOT; i++) {
                        BFORCE[i] = 0.0;
                        D[i]      = 0.0;
                        PHI[i]    = 0.0;
                }
                for(i=0; i<=ICELTOT; i++) {
                        indexLnew[i] = indexLnew_org[i];
                        indexUnew[i] = indexUnew_org[i];

                }
                for(i=0; i<NPL; i++) {
                        itemLnew[i] = 0;
                        ALnew[i] = 0.0;
                }
                for(i=0; i<NPU; i++) {
                        itemUnew[i] = 0;
                        AUnew[i] = 0.0;
                }

        }else {
                indexLnew[0]=0;
                indexUnew[0]=0;
#pragma omp parallel for private (icel, j)
                for(ip=1; ip<=PEsmpTOT; ip++){
                        for(icel = SMPindex_new[(ip-1)*NCOLORtot]+1; icel<=SMPindex_new[ip*NCOLORtot]; icel++){
                                BFORCE[icel-1] = 0.0;
                                PHI[icel-1] = 0.0;
                                D[icel-1] = 0.0;
                                indexLnew[icel]=indexLnew_org[icel];
                                indexUnew[icel]=indexUnew_org[icel];

                                for(j=indexLnew_org[icel-1];j<indexLnew_org[icel];j++){
                                        itemLnew[j]=0;
                                        ALnew[j] = 0.0;
                                }
                                for(j=indexUnew_org[icel-1];j<indexUnew_org[icel];j++){
                                        itemUnew[j]=0;
                                        AUnew[j] = 0.0;
                                }
                        }
                }
        }
```

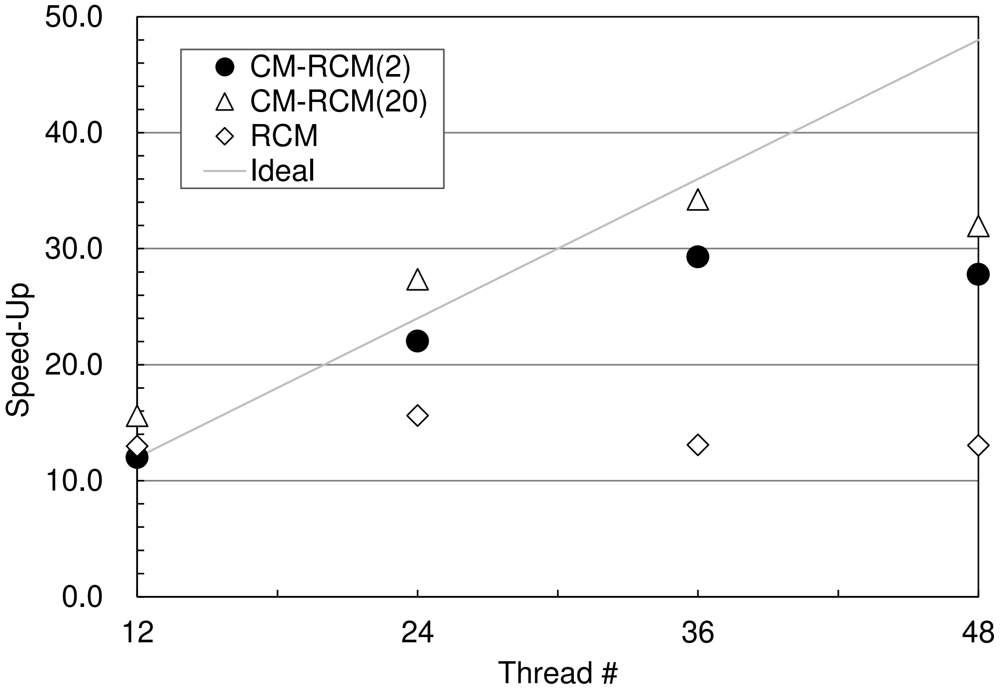
Pages are allocated at the local memory of the master thread

Pages are allocated at the local memory of each thread

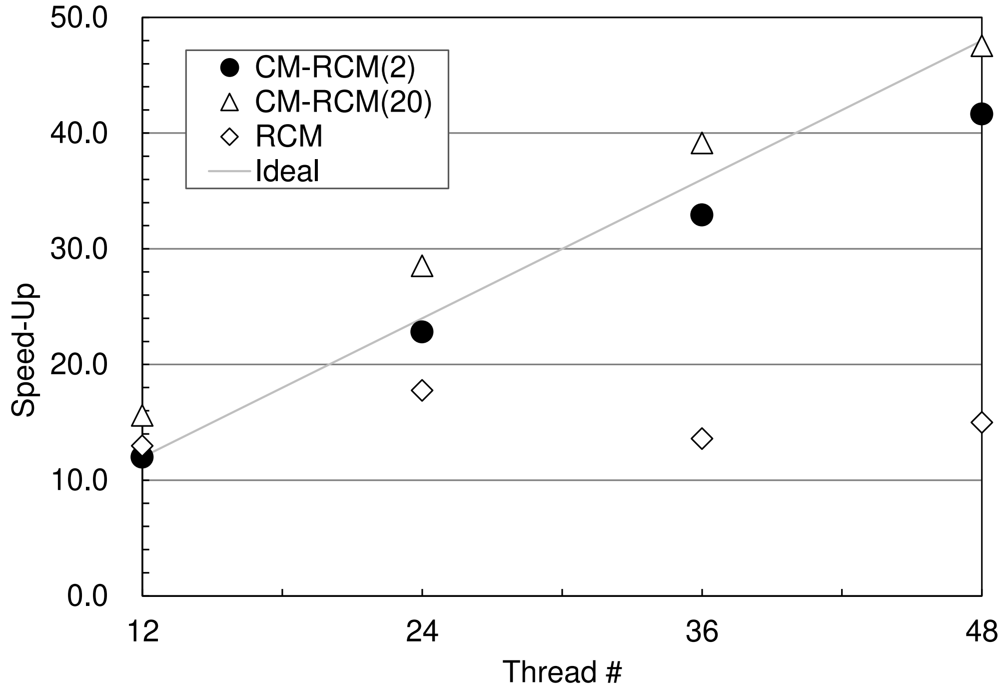
Results: reoder0, L3-rsol0, $N=128^3$, C

based on the performance of CM-RCM(2) with 12 threads/without First Touch

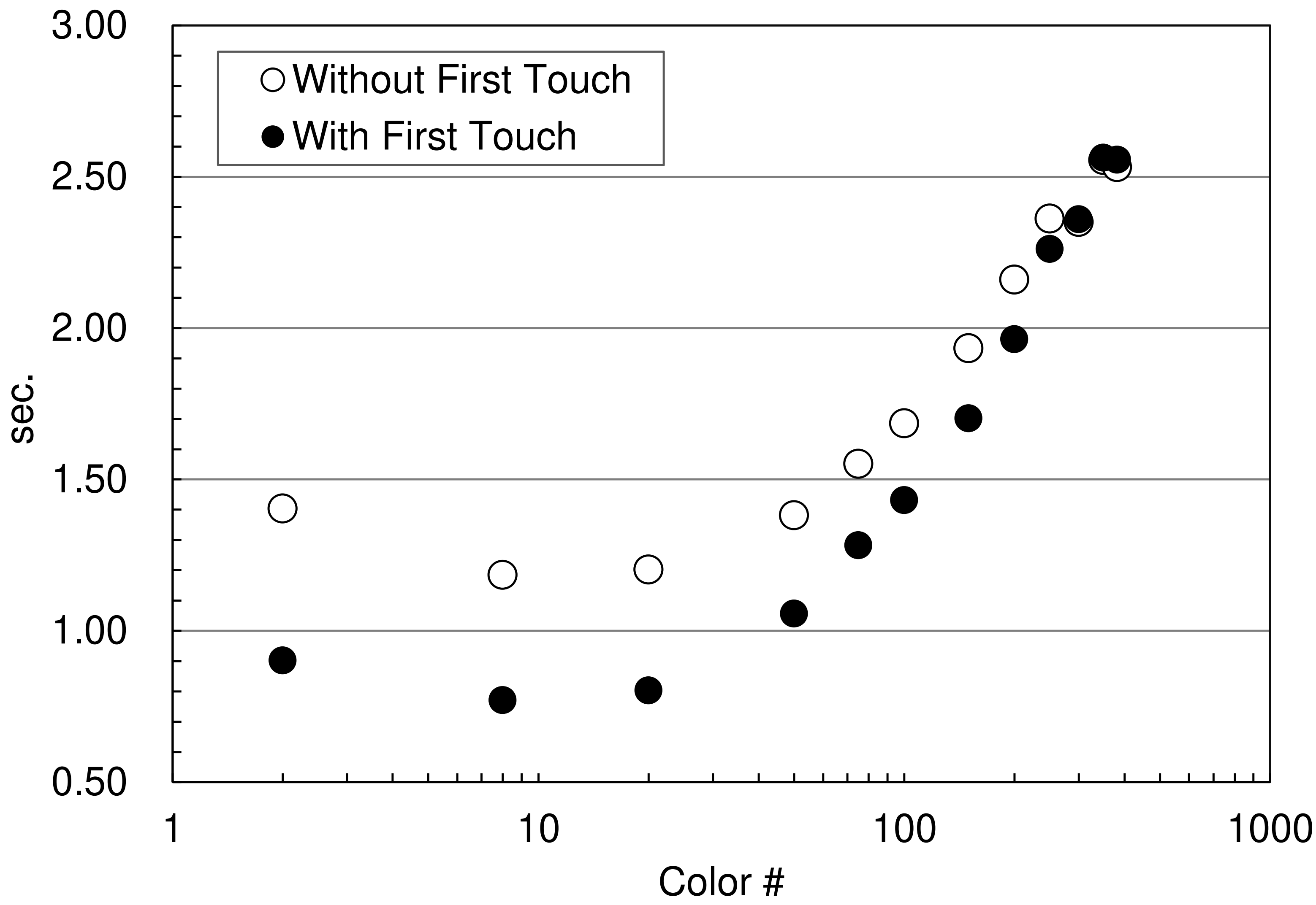
Without First-Touch



With First-Touch



Results: reoder0, L3-rsol0, $N=128^3$, C 48 threads



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，OBCXを利用した実習

- オーダリングの効果
- First-Touch Data Placement

今後の動向

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

この後

- UIDは一ヶ月有効
- 何かあったらいつでも気軽に質問してください
- 利用方法等に関する質問は相談窓口ではなく中島へ
- 一ヶ月間に限りZoomでの個別のセッション（一人最大45分～60分程度）にも対応しますので，ご連絡ください。
- プログラム類は研究等にご利用いただいて結構です。商用プログラムの一部になる場合，これを元にしたプログラムでビジネスをやる場合のみご相談ください。
- アンケートもよろしく