

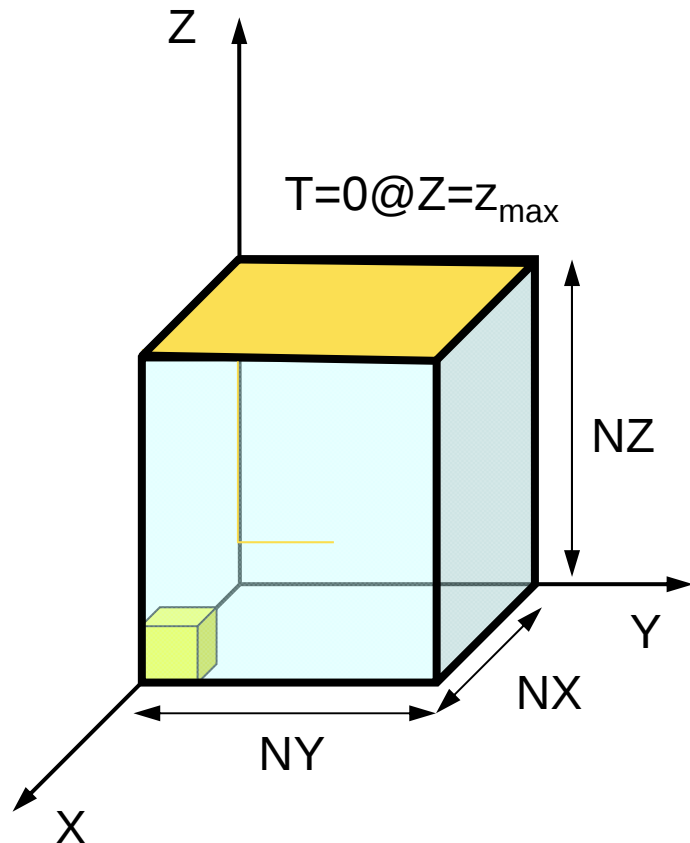
並列有限要素法による 三次元定常熱伝導解析プログラム (2/2) +課題P1出題 C言語編

2012年夏季集中講義
中島 研吾

並列計算プログラミング (616-2057) ・ 先端計算機演習 (616-4009)

対象とする問題：三次元定常熱伝導

$$\frac{\partial}{\partial x} \left(\lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(\lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left(\lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$



- 定常熱伝導＋発熱
- 一様な熱伝導率 λ
- 直方体
 - 一辺長さ1の立方体（六面体）要素
 - 各方向にNX・NY・NZ個
- 境界条件
 - $T=0@Z=z_{\max}$
- 体積当たり発熱量は位置（メッシュの中心の座標 x_c, y_c ）に依存
 - $\dot{Q}(x, y, z) = QVOL|x_c + y_c|$

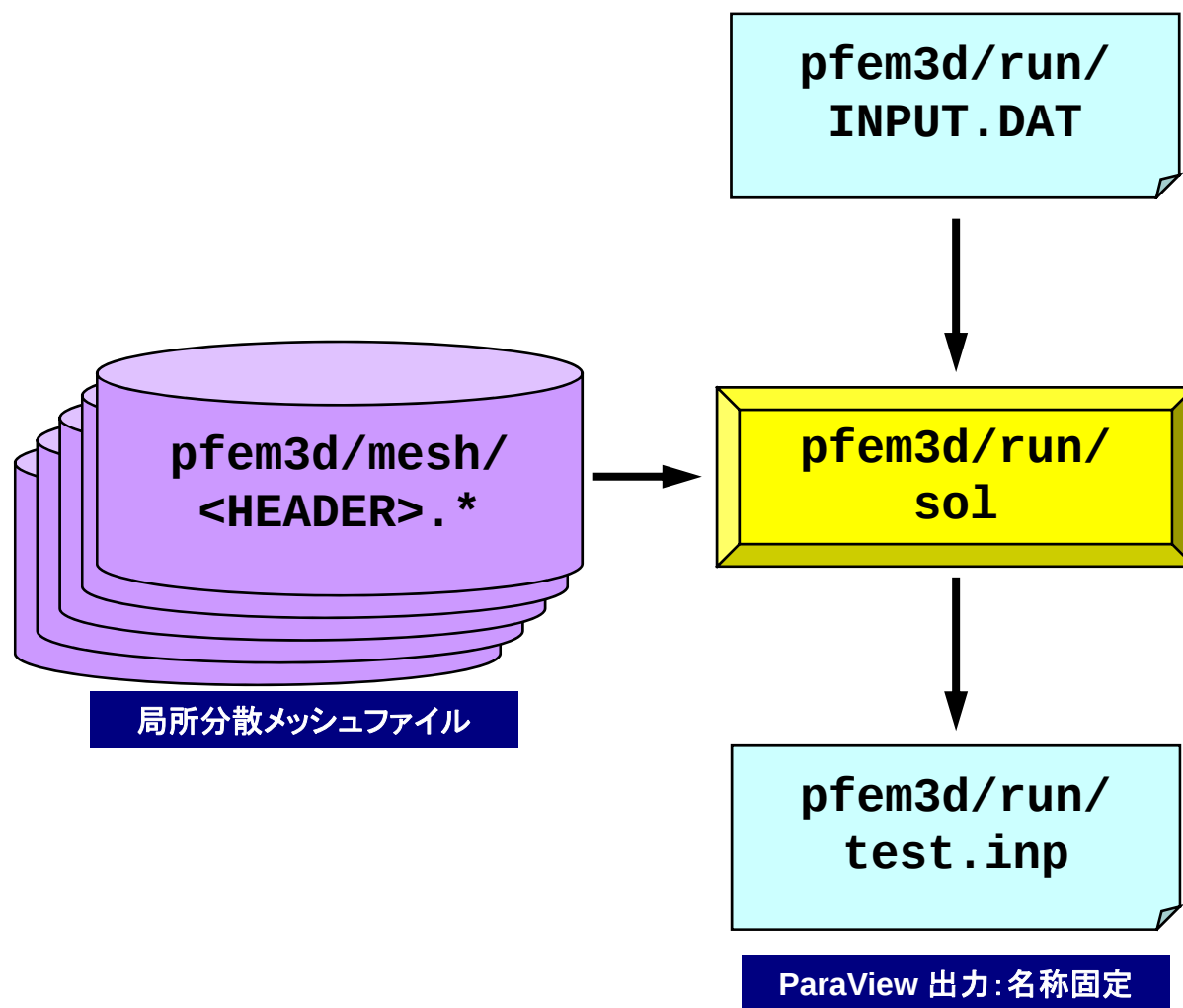
有限要素法の処理

- 支配方程式
- ガラーキン法：弱形式
- 要素単位の積分
 - 要素マトリクス生成
- 全体マトリクス生成
- 境界条件適用
- 連立一次方程式

並列有限要素法の処理：プログラム

- 初期化
 - 制御変数読み込み
 - 座標読み込み⇒要素生成 (N:節点数, NE: 要素数)
 - 配列初期化 (全体マトリクス, 要素マトリクス)
 - 要素⇒全体マトリクスマッピング (Index, Item)
- マトリクス生成
 - 要素単位の処理 (do icel= 1, NE)
 - 要素マトリクス計算
 - 全体マトリクスへの重ね合わせ
 - 境界条件の処理
- 連立一次方程式
 - 共役勾配法 (CG)

並列有限要素法の手順（並列計算実行）



制御ファイル：INPUT.DAT

```
../mesh/aaa  HEADER
2000          ITER
1.0  1.0      COND,  QVOL
1.0e-08      RESID
```

- HEADER : 局所分散ファイルヘッダ名
 <HEADER>.my_rank
- ITER : 反復回数上限
- COND : 熱伝導率
- QVOL : 体積当たり発熱量係数
- RESID : 反復法の収束判定値

$$\frac{\partial}{\partial x} \left(\lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(\lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left(\lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$

$$\dot{Q}(x, y, z) = QVOL |x_C + y_C|$$

<\$O-TOP>/pfem3d/run/go.sh

```
#!/bin/sh
```

```
#PJM -L "node=1"
```

ノード数 (≤ 12)

```
#PJM -L "elapsed=00:10:00"
```

実行時間 (≤ 15 分)

```
#PJM -L "rscgrp=lecture"
```

実行キュー名 (lecture1)

```
#PJM -g "gt61"
```

「財布」変更不可

```
#PJM -
```

```
#PJM -o "test.lst"
```

標準出力

```
#PJM --mpi "proc=8"
```

MPIプロセス数 (≤ 192)

```
mpiexec ./sol
```

8分割

"node=1"

"proc=8"

16分割

"node=1"

"proc=16"

32分割

"node=2"

"proc=32"

64分割

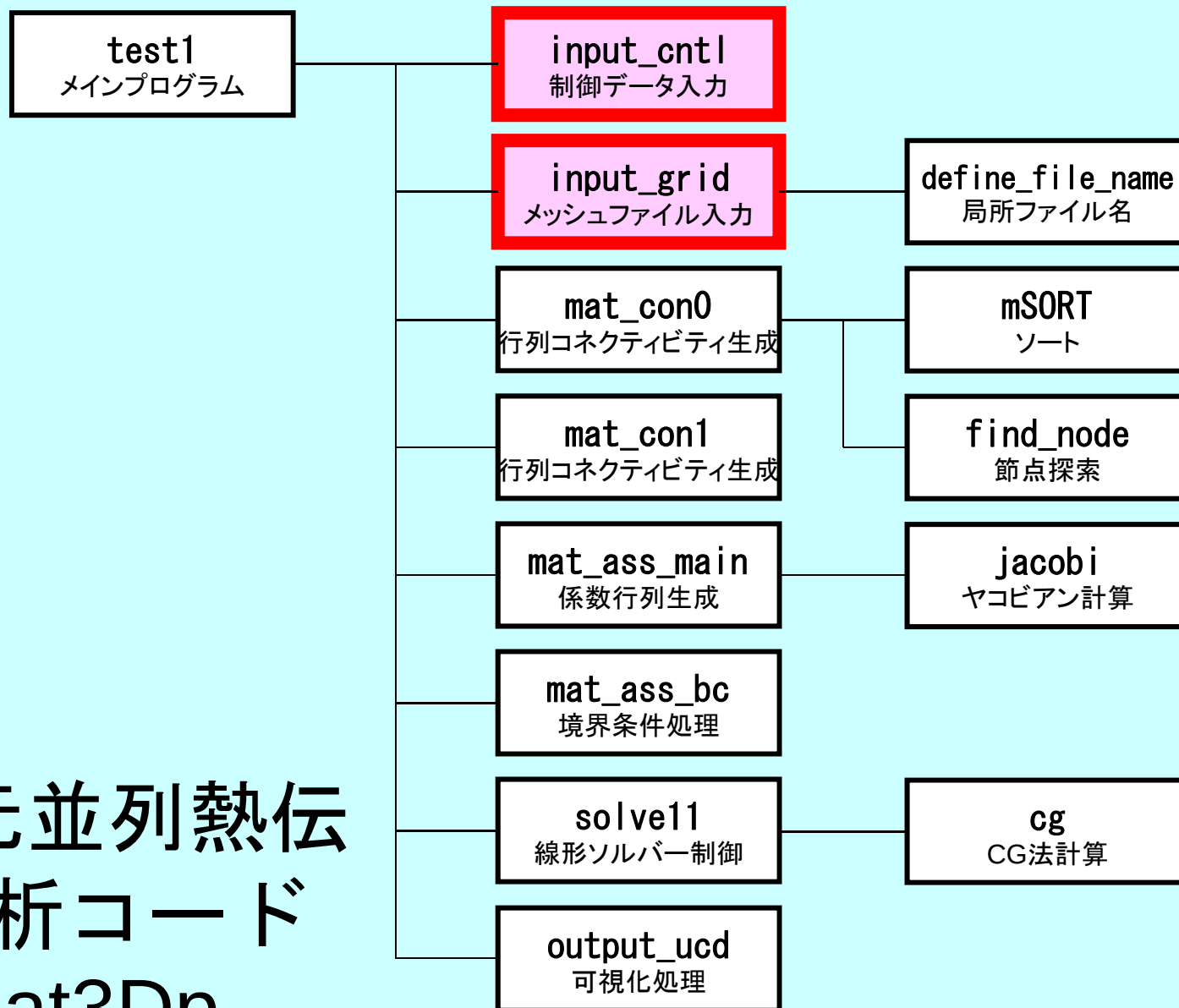
"node=4"

"proc=64"

192分割

"node=12"

"proc=192"



三次元並列熱伝
導解析コード
heat3Dp

全体処理

```
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"
extern void PFEM_INIT(int, char**);
extern void INPUT_CNTL();
extern void INPUT_GRID();
extern void MAT_CONO();
extern void MAT_CON1();
extern void MAT_ASS_MAIN();
extern void MAT_ASS_BC();
extern void SOLVE11();
extern void OUTPUT_UCD();
extern void PFEM_FINALIZE();
int main(int argc, char* argv[])
{
    double START_TIME, END_TIME;

    PFEM_INIT(argc, argv);

    INPUT_CNTL();
    INPUT_GRID();

    MAT_CONO();
    MAT_CON1();

    MAT_ASS_MAIN();
    MAT_ASS_BC();

    SOLVE11();

    OUTPUT_UCD();
    PFEM_FINALIZE();
}
```

Global変数表 : pfem_util.h (1/4)

変数名	種別	サイズ	I/O	内 容
fname	C	[80]	I	メッシュファイル名
N, NP	I		I	節点数 (N : 内点, NP : 内点+外点)
ICELTOT	I		I	要素数
NODGRPtot	I		I	節点グループ数
XYZ	R	[NP][3]	I	節点座標
ICELNOD	I	[ICELTOT][8]	I	要素コネクティビティ
NODGRP_INDEX	I	[NODGRPtot+1]	I	各節点グループに含まれる節点数 (累積)
NODGRP_ITEM	I	[NODGRP_INDEX[NODGRPtot+1]]	I	節点グループに含まれる節点
NODGRP_NAME	C80	[NODGRP_INDEX[NODGRPtot+1]]	I	節点グループ名
NLU	I		O	各節点非対角成分数
NPLU	I		O	非対角成分総数
D	R	[NP]	O	全体行列 : 対角ブロック
B, X	R	[NP]	O	右辺ベクトル, 未知数ベクトル

Global変数表 : pfem_util.h (2/4)

変数名	種別	サイズ	I/O	内 容
AMAT	R	[NP]	O	全体行列 : 非零非対角成分
indexLU	I	[NP+1]	O	全体行列 : 非零非対角成分数
itemLU	I	[NPLU]	O	全体行列 : 非零非対角成分 (列番号)
INLU	I	[NP]	O	各節点の非零非対角成分数
IALU	I	[NP][NLU]	O	各節点の非零非対角成分数 (列番号)
IWKX	I	[NP][2]	O	ワーク用配列
ITER, ITERactual	I		I	反復回数の上限, 実際の反復回数
RESID	R		I	打ち切り誤差 (1.e-8に設定)

Global変数表 : pfem_util.h (3/4)

変数名	種別	サイズ	I/O	内 容
08th	R		I	=0.125
PNQ, PNE, PNT	R	[2][2][8]	O	各ガウス積分点における $\frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta} (i=1\sim 8)$
POS, WEI	R	[2]	O	各ガウス積分点の座標, 重み係数
NCOL1, NCOL2	I	[100]	O	ソート用ワーク配列
SHAPE	R	[2][2][2][8]	O	各ガウス積分点における形状関数 N_i ($i=1\sim 8$)
PNX, PNY, PNZ	R	[2][2][2][8]	O	各ガウス積分点における $\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z} (i=1\sim 8)$
DETJ	R	[2][2][2]	O	各ガウス積分点におけるヤコビアン行列式
COND, QVOL	R		I	熱伝導率, 体積当たり発熱量係数

Global変数表 : pfem_util.h (4/4)

変数名	種別	サイズ	I/O	内 容
PETOT	I		O	領域数 (MPI プロセス数)
my_rank	I		O	MPI プロセス番号
errno	I		O	エラーフラグ
NEIBPETOT	I		I	隣接領域数
NEIBPE	I	[NEIBPETOT]	I	隣接領域番号
IMPORT_INDEX EXPORT_INEDX	I	[NEIBPETOT+1]	I	送信, 受信テーブルのサイズ (一次元圧縮配列)
IMPORT_ITEM	I	[NPimport]	I	受信テーブルの内容 (NPimport=IMPORT_INDEX[NEIBPETOT])
EXPORT_ITEM	I	[NPexport]	I	受信テーブルの内容 (NPexport=EXPORT_INDEX[NEIBPETOT])
ICELTOT_INT	I		I	領域所属要素数
intELEM_list	I	[ICELTOT_INT]	I	領域所属要素のリスト: 可視化に使用

開始, 終了 : MPI_Init/Finalize

```
#include "pfem_util.h"
void PFEM_INIT(int argc, char* argv[])
{
    int i;
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &PETOT);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    for(i=0; i<100; i++) pfemRarray[i]=0.0;
    for(i=0; i<100; i++) pfemIarray[i]=0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "pfem_util.h"

void PFEM_FINALIZE()
{
    MPI_Finalize();

    if( my_rank == 0 ){
        fprintf(stdout, "* normal termination\n");
        exit(0);
    }
}
```

制御ファイル入力 : INPUT_CNTL

```
#include <stdio.h>
#include <stdlib.h>
#include "pfem_util.h"
/** **/
void INPUT_CNTL()
{
    FILE *fp;

    if( my_rank == 0 ){
        if( (fp=fopen("INPUT.DAT","r")) == NULL ){
            fprintf(stdout,"input file cannot be opened!¥n");
            exit(1);
        }

        fscanf(fp,"%s",HEADER);
        fscanf(fp,"%d",&ITER);
        fscanf(fp,"%lf %lf", &COND, &QVOL);
        fscanf(fp,"%lf", &RESID);
        fclose(fp);
    }

    MPI_Bcast(HEADER, 80, MPI_CHAR, 0, MPI_COMM_WORLD);
    MPI_Bcast(&ITER, 1, MPI_INTEGER, 0, MPI_COMM_WORLD);
    MPI_Bcast(&COND, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(&QVOL, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Bcast(&RESID, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    pfemRarray[0]= RESID;
    pfemIarray[0]= ITER;
}
```

メッシュ入力 : INPUT_GRID (1/3)

```

#include <stdio.h>
#include <stdlib.h>
#include "pfem_util.h"
#include "allocate.h"
/** external functions **/
extern void ERROR_EXIT (int, int);
extern void DEFINE_FILE_NAME (char*, char*, int);
/** **/
void INPUT_GRID()
{
    FILE *fp;
    int i, j, k, ii, kk, kkk, nn, icel, iS, iE, ic0;
    int NTYPE, IMAT;
    int idummy;

    DEFINE_FILE_NAME(HEADER, fname, my_rank);
    if( (fp=fopen(fname, "r")) == NULL){
        fprintf(stdout, "input file cannot be opened!¥n");
        exit(1);}

    /**
    NEIB-PE
    **/

    fscanf(fp, "%d", &kkk);
    fscanf(fp, "%d", &NEIBPETOT);

    NEIBPE=(int*)allocate_vector(sizeof(int), NEIBPETOT);
    for(i=0; i<NEIBPETOT; i++) fscanf(fp, "%d", &NEIBPE[i]);

    for(i=0; i<NEIBPETOT; i++) {
        if( NEIBPE[i] > PETOT-1 ){
            ERROR_EXIT (202, my_rank);}
    }
}

```

分散メッシュファイル名 : DEFINE_FILE_NAME HEADER+ランク番号

```
#include <stdio.h>
#include <string.h>
void DEFINE_FILE_NAME (char HEADERo[], char filename[], int my_rank)
{
    char string[80];
    sprintf(string, ".%d", my_rank);
    strcpy(filename, HEADERo);
    strcat(filename, string);
}
```

allocate, deallocate関数

```
#include <stdio.h>
#include <stdlib.h>
void* allocate_vector(int size, int m)
{
    void *a;
    if ( ( a=(void *)malloc( m * size ) ) == NULL ) {
        fprintf(stdout, "Error:Memory does not enough! in vector %n");
        exit(1);
    }
    return a;
}

void deallocate_vector(void *a)
{
    free( a );
}

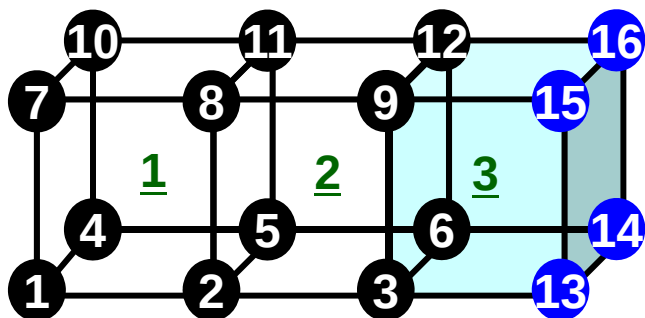
void** allocate_matrix(int size, int m, int n)
{
    void **aa;
    int i;
    if ( ( aa=(void **)malloc( m * sizeof(void*) ) ) == NULL ) {
        fprintf(stdout, "Error:Memory does not enough! aa in matrix %n");
        exit(1);
    }
    if ( ( aa[0]=(void *)malloc( m * n * size ) ) == NULL ) {
        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;
}

void deallocate_matrix(void **aa)
{
    free( aa );
}
```

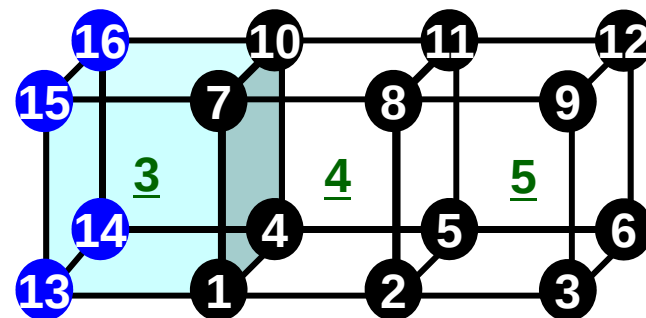
allocateをFORTRAN並みに
簡単にやるための関数

局所番号付け：節点（隣接領域）

aaa.1



aaa.0



1				
1				
0				
16		12		
1	1	0.00	0.00	0.00
2	1	1.00	0.00	0.00
3	1	2.00	0.00	0.00
4	1	0.00	1.00	0.00
5	1	1.00	1.00	0.00
6	1	2.00	1.00	0.00
7	1	0.00	0.00	1.00
8	1	1.00	0.00	1.00
9	1	2.00	0.00	1.00
10	1	0.00	1.00	1.00
11	1	1.00	1.00	1.00
12	1	2.00	1.00	1.00
1	0	3.00	0.00	0.00
4	0	3.00	1.00	0.00
7	0	3.00	0.00	1.00
10	0	3.00	1.00	1.00

0				
1				
1				
16		12		
1	0	3.00	0.00	0.00
2	0	4.00	0.00	0.00
3	0	5.00	0.00	0.00
4	0	3.00	1.00	0.00
5	0	4.00	1.00	0.00
6	0	5.00	1.00	0.00
7	0	3.00	0.00	1.00
8	0	4.00	0.00	1.00
9	0	5.00	0.00	1.00
10	0	3.00	1.00	1.00
11	0	4.00	1.00	1.00
12	0	5.00	1.00	1.00
3	1	2.00	0.00	0.00
6	1	2.00	1.00	0.00
9	1	2.00	0.00	1.00
12	1	2.00	1.00	1.00

メッシュ入力 : INPUT_GRID (2/3)

```

/**
**/
NODE

fscanf(fp, "%d %d", &NP, &N);

XYZ = (KREAL**) allocate_matrix(sizeof(KREAL), NP, 3);
NODE_ID = (KINT **) allocate_matrix(sizeof(KINT ), NP, 2);

for (i=0; i<NP; i++) {
    for (j=0; j<3; j++) {
        XYZ[i][j]=0.0;
    }
}

for (i=0; i<NP; i++) {
    fscanf(fp, "%d %d %lf %lf %lf", &NODE_ID[i][0], &NODE_ID[i][1], &XYZ[i][0], &XYZ[i][1], &XYZ[i][2]);
}

/**
**/
ELEMENT

fscanf(fp, "%d %d", &ICELTOT, &ICELTOT_INT);

ICELNOD = (KINT**) allocate_matrix(sizeof(KINT), ICELTOT, 8);
intELEM_list = (KINT*) allocate_vector(sizeof(KINT), ICELTOT);
ELEM_ID = (KINT**) allocate_matrix(sizeof(KINT), ICELTOT, 2);

for (i=0; i<ICELTOT; i++) fscanf(fp, "%d", &NTYPE);

for (icel=0; icel<ICELTOT; icel++) {
    fscanf(fp, "%d %d %d %d %d %d %d %d %d %d",
        &ELEM_ID[icel][0], &ELEM_ID[icel][1],
        &IMAT,
        &ICELNOD[icel][0], &ICELNOD[icel][1], &ICELNOD[icel][2], &ICELNOD[icel][3],
        &ICELNOD[icel][4], &ICELNOD[icel][5], &ICELNOD[icel][6], &ICELNOD[icel][7]);
}

for (ic0=0; ic0<ICELTOT_INT; ic0++) fscanf(fp, "%d", &intELEM_list[ic0]);

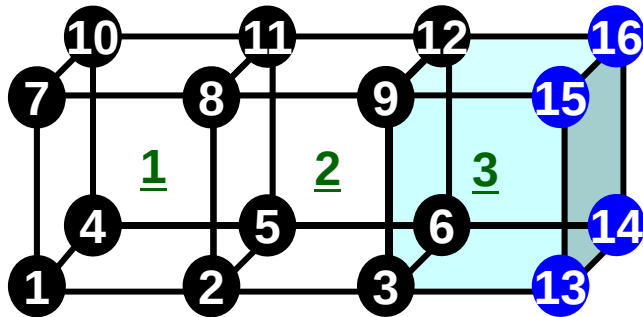
```

局所番号付け：節点

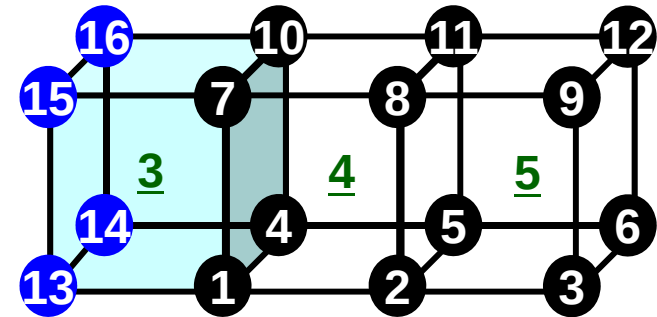
- 局所番号は各領域「1」から番号付け
 - 1CPUの場合と同じプログラムを使用可能：SPMD
 - 要素番号も同じように「1」から番号付け
- 内点⇒外点という順番で番号付け
- Double Numbering
 - 本来の所属領域での局所節点番号
 - 所属領域番号

局所番号付け：節点

aaa.1



aaa.0

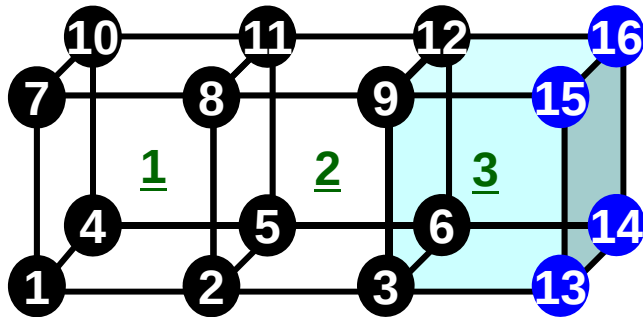


1				
1				
0				
16		12		
1	1	0.00	0.00	0.00
2	1	1.00	0.00	0.00
3	1	2.00	0.00	0.00
4	1	0.00	1.00	0.00
5	1	1.00	1.00	0.00
6	1	2.00	1.00	0.00
7	1	0.00	0.00	1.00
8	1	1.00	0.00	1.00
9	1	2.00	0.00	1.00
10	1	0.00	1.00	1.00
11	1	1.00	1.00	1.00
12	1	2.00	1.00	1.00
1	0	3.00	0.00	0.00
4	0	3.00	1.00	0.00
7	0	3.00	0.00	1.00
10	0	3.00	1.00	1.00

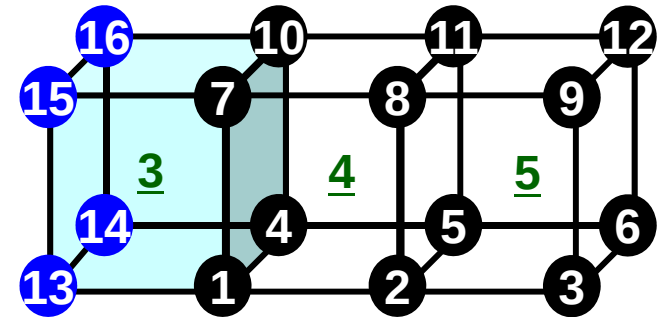
0				
1				
1				
16		12	(総節点数, 内点数)	
1	0	3.00	0.00	0.00
2	0	4.00	0.00	0.00
3	0	5.00	0.00	0.00
4	0	3.00	1.00	0.00
5	0	4.00	1.00	0.00
6	0	5.00	1.00	0.00
7	0	3.00	0.00	1.00
8	0	4.00	0.00	1.00
9	0	5.00	0.00	1.00
10	0	3.00	1.00	1.00
11	0	4.00	1.00	1.00
12	0	5.00	1.00	1.00
3	1	2.00	0.00	0.00
6	1	2.00	1.00	0.00
9	1	2.00	0.00	1.00
12	1	2.00	1.00	1.00

局所番号付け：節点

aaa.1



aaa.0



1					
1					
0					
16		12			
1	1	0.00	0.00	0.00	①
2	1	1.00	0.00	0.00	②
3	1	2.00	0.00	0.00	③
4	1	0.00	1.00	0.00	④
5	1	1.00	1.00	0.00	⑤
6	1	2.00	1.00	0.00	⑥
7	1	0.00	0.00	1.00	⑦
8	1	1.00	0.00	1.00	⑧
9	1	2.00	0.00	1.00	⑨
10	1	0.00	1.00	1.00	⑩
11	1	1.00	1.00	1.00	⑪
12	1	2.00	1.00	1.00	⑫
1	0	3.00	0.00	0.00	⑬
4	0	3.00	1.00	0.00	⑭
7	0	3.00	0.00	1.00	⑮
10	0	3.00	1.00	1.00	⑯

所属領域とそこでの番号

座標

0					
1					
1					
16		12			
1	0	3.00	0.00	0.00	①
2	0	4.00	0.00	0.00	②
3	0	5.00	0.00	0.00	③
4	0	3.00	1.00	0.00	④
5	0	4.00	1.00	0.00	⑤
6	0	5.00	1.00	0.00	⑥
7	0	3.00	0.00	1.00	⑦
8	0	4.00	0.00	1.00	⑧
9	0	5.00	0.00	1.00	⑨
10	0	3.00	1.00	1.00	⑩
11	0	4.00	1.00	1.00	⑪
12	0	5.00	1.00	1.00	⑫
3	1	2.00	0.00	0.00	⑬
6	1	2.00	1.00	0.00	⑭
9	1	2.00	0.00	1.00	⑮
12	1	2.00	1.00	1.00	⑯

所属領域とそこでの番号

座標

メッシュ入力 : INPUT_GRID (2/3)

```

/**
**/
NODE

fscanf(fp, "%d %d", &NP, &N);

XYZ = (KREAL**) allocate_matrix(sizeof(KREAL), NP, 3);
NODE_ID = (KINT **) allocate_matrix(sizeof(KINT ), NP, 2);

for (i=0; i<NP; i++) {
    for (j=0; j<3; j++) {
        XYZ[i][j]=0.0;
    }
}

for (i=0; i<NP; i++) {
    fscanf(fp, "%d %d %lf %lf %lf", &NODE_ID[i][0], &NODE_ID[i][1], &XYZ[i][0], &XYZ[i][1], &XYZ[i][2]);
}

/**
**/
ELEMENT

fscanf(fp, "%d %d", &ICELTOT, &ICELTOT_INT);

ICELNOD = (KINT**) allocate_matrix(sizeof(KINT), ICELTOT, 8);
intELEM_list = (KINT*) allocate_vector(sizeof(KINT), ICELTOT);
ELEM_ID = (KINT**) allocate_matrix(sizeof(KINT), ICELTOT, 2);

for (i=0; i<ICELTOT; i++) fscanf(fp, "%d", &NTYPE);

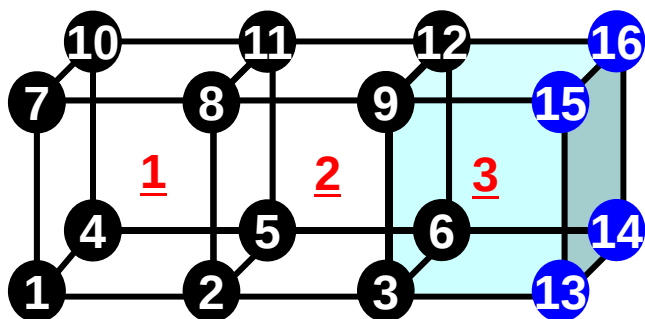
for (icel=0; icel<ICELTOT; icel++) {
    fscanf(fp, "%d %d %d %d %d %d %d %d %d %d",
        &ELEM_ID[icel][0], &ELEM_ID[icel][1],
        &IMAT,
        &ICELNOD[icel][0], &ICELNOD[icel][1], &ICELNOD[icel][2], &ICELNOD[icel][3],
        &ICELNOD[icel][4], &ICELNOD[icel][5], &ICELNOD[icel][6], &ICELNOD[icel][7]);
}

for (ic0=0; ic0<ICELTOT_INT; ic0++) fscanf(fp, "%d", &intELEM_list[ic0]);

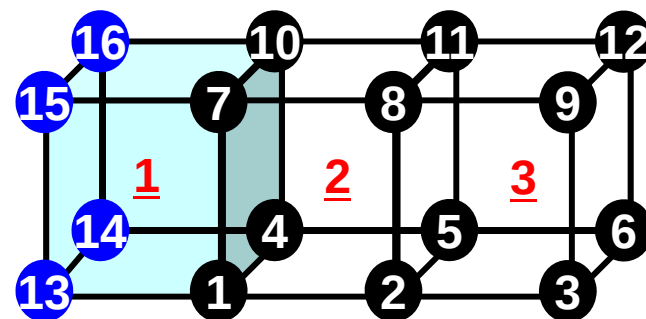
```

局所番号付け：要素

aaa.1



aaa.0

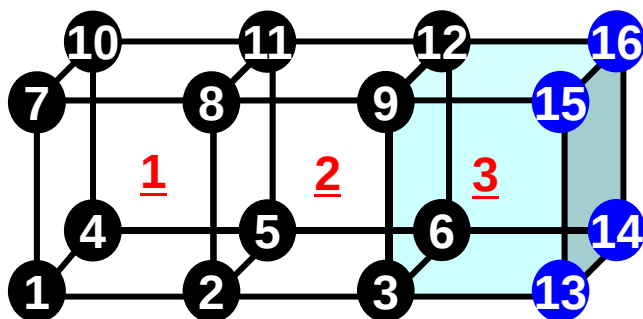


3	2									
361	361	361								
1	1	1	1	2	5	4	7	8	11	10
2	1	1	2	3	6	5	8	9	12	11
1	0	1	3	13	14	6	9	15	16	12
1	2									

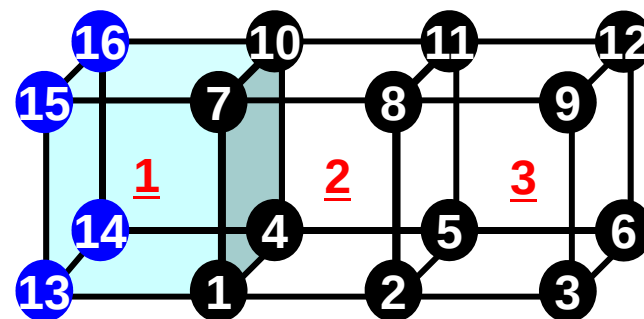
3	3									
361	361	361								
1	0	1	13	1	4	14	15	7	10	16
2	0	1	1	2	5	4	7	8	11	10
3	0	1	2	3	6	5	8	9	12	11
1	2	3								

局所番号付け：要素

aaa.1

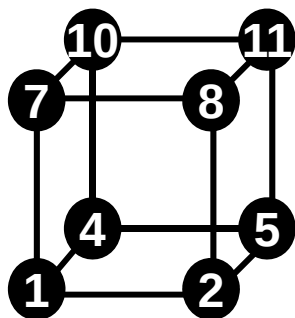


aaa.0



3	2										
361	361	361									
1	1	1	1	2	5	4	7	8	11	10	
2	1	1	2	3	6	5	8	9	12	11	
1	0	1	3	13	14	6	9	15	16	12	
1	2										

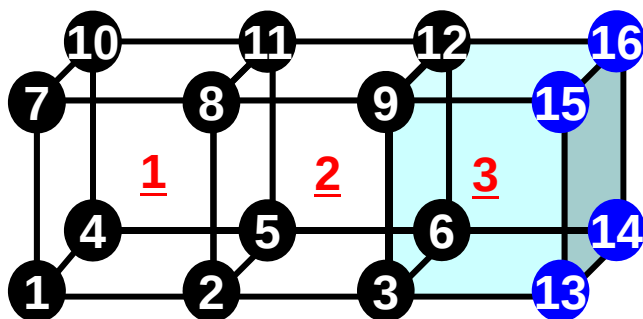
3	3	(全要素, 領域所属要素)									
361	361	361									
1	0	1	13	1	4	14	15	7	10	16	
2	0	1	1	2	5	4	7	8	11	10	
3	0	1	2	3	6	5	8	9	12	11	
1	2	3									



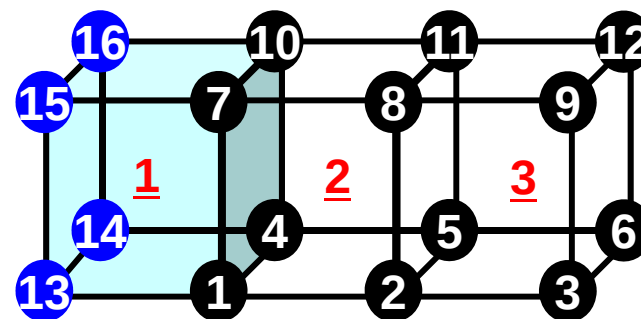
- 要素が所属する領域
 - 8個の節点の所属する領域によって決定
 - 全て「内点」であれば、節点と同じ領域
 - 「外点」を含む場合は、節点の所属領域番号の最も若い領域に属する
 - 本ケースのオーバーラップ要素は「0」領域に所属
 - OUTPUT_UCDで利用

局所番号付け：要素

aaa.1



aaa.0

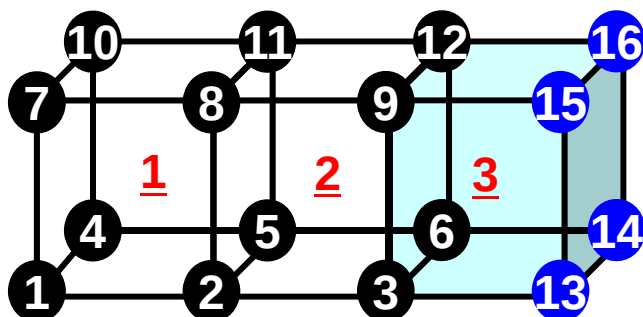


	<u>3</u>	2								
361	361	361								
1	1	1	1	2	5	4	7	8	11	10
2	1	1	2	3	6	5	8	9	12	11
1	0	1	3	13	14	6	9	15	16	12
1	2									

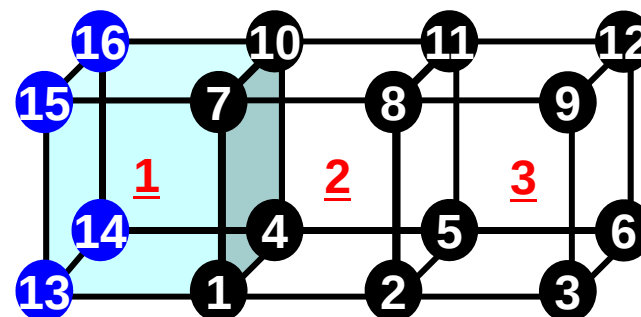
3	3									
361	361	361	(要素タイプ, 全要素)							
1	0	1	13	1	4	14	15	7	10	16
2	0	1	1	2	5	4	7	8	11	10
3	0	1	2	3	6	5	8	9	12	11
1	2	3								

局所番号付け：要素

aaa.1



aaa.0

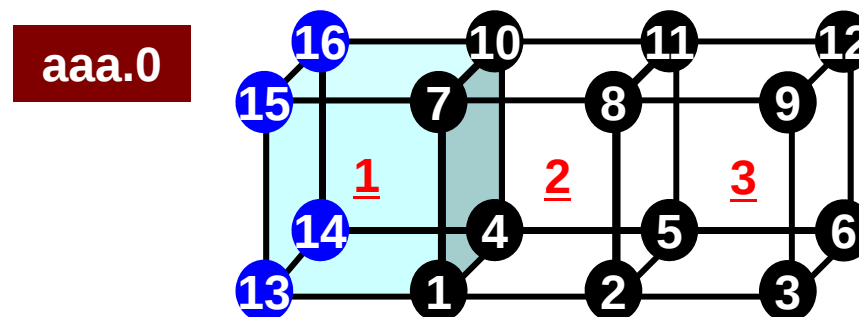
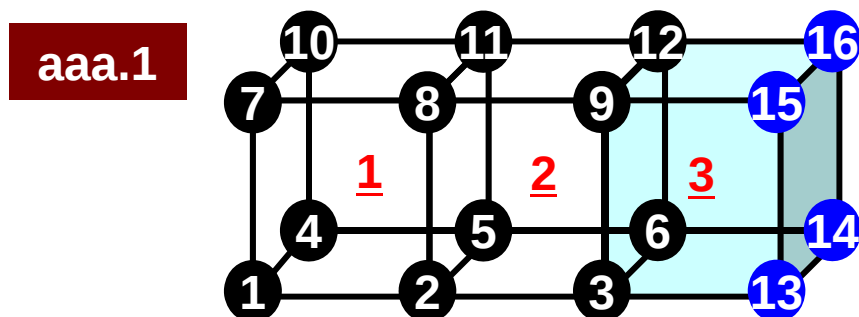


<u>3</u>	2											
361	361	361										
1	1	1	1	2	5	4	7	8	11	10	<u>1</u>	
2	1	1	2	3	6	5	8	9	12	11	<u>2</u>	
1	0	1	3	13	14	6	9	15	16	12	<u>3</u>	
1	2											

<u>3</u>	3											
361	361	361										
1	0	1	13	1	4	14	15	7	10	16	<u>1</u>	
2	0	1	1	2	5	4	7	8	11	10	<u>2</u>	
3	0	1	2	3	6	5	8	9	12	11	<u>3</u>	
1	2	3										

- 要素についても Double Numbering
 - 本来の所属領域での局所要素番号
 - 所属領域番号
- 材料番号
- 8個の節点
- 以降の計算では下線付の「局所要素番号」を使用

局所番号付け：要素

[illegible][illegible]

- aaa.1
 - 1, 2 の要素が「領域所属要素」
- aaa.0
 - 1, 2, 3 の要素が「領域所属要素」

メッシュ入力 : INPUT_GRID (3/3)

```

/**
    COMMUNICATION table
**/

IMPORT_INDEX=(int*)allocate_vector(sizeof(int), NEIBPETOT+1);
EXPORT_INDEX=(int*)allocate_vector(sizeof(int), NEIBPETOT+1);

for(i=0;i<NEIBPETOT+1;++i) IMPORT_INDEX[i]=0;
for(i=0;i<NEIBPETOT+1;++i) EXPORT_INDEX[i]=0;

if( PETOT != 1 ) {
    for(i=1;i<=NEIBPETOT;i++) fscanf(fp, "%d", &IMPORT_INDEX[i]);
    nn=IMPORT_INDEX[NEIBPETOT];
    if( nn > 0 ) IMPORT_ITEM=(int*)allocate_vector(sizeof(int), nn);
    for(i=0;i<nn;i++) fscanf(fp, "%d %d", &IMPORT_ITEM[i], &dummy);

    for(i=1;i<=NEIBPETOT;i++) fscanf(fp, "%d", &EXPORT_INDEX[i]);
    nn=EXPORT_INDEX[NEIBPETOT];
    if( nn > 0 ) EXPORT_ITEM=(int*)allocate_vector(sizeof(int), nn);
    for(i=0;i<nn;i++) fscanf(fp, "%d", &EXPORT_ITEM[i]);}

/**
    NODE grp. info.
**/

fscanf(fp, "%d", &NODGRPtot);

NODGRP_INDEX=(KINT*)allocate_vector(sizeof(KINT), NODGRPtot+1);
NODGRP_NAME =(CHAR80*)allocate_vector(sizeof(CHAR80), NODGRPtot);
for(i=0;i<NODGRPtot+1;i++) NODGRP_INDEX[i]=0;

for(i=0;i<NODGRPtot;i++) fscanf(fp, "%d", &NODGRP_INDEX[i+1]);
nn=NODGRP_INDEX[NODGRPtot];
NODGRP_ITEM=(KINT*)allocate_vector(sizeof(KINT), nn);

for(k=0;k<NODGRPtot;k++) {
    iS= NODGRP_INDEX[k];
    iE= NODGRP_INDEX[k+1];
    fscanf(fp, "%s", NODGRP_NAME[k].name);
    nn= iE - iS;
    if( nn != 0 ) {
        for(kk=iS;kk<iE;kk++) fscanf(fp, "%d", &NODGRP_ITEM[kk]);}
}

```

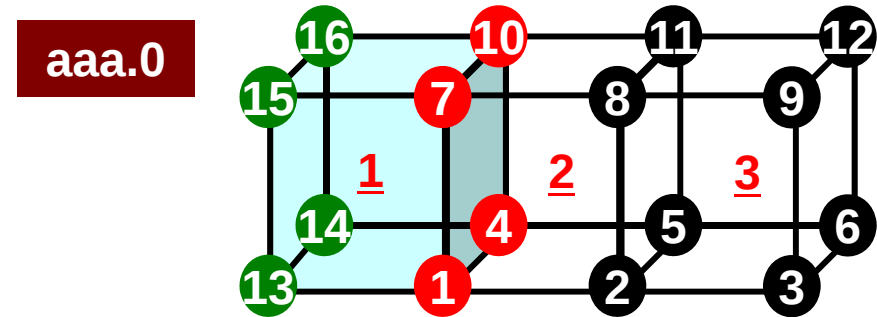
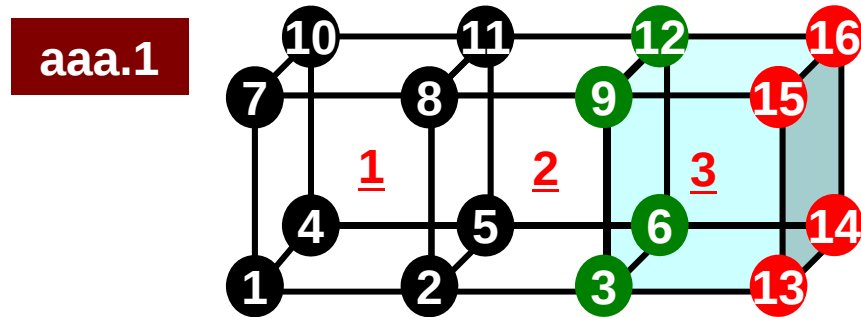
領域間通信 一般化された通信テーブル

- 「通信」とは「外点」の情報を, その「外点」が本来属している領域から得ることである。
- 「通信テーブル」とは領域間の外点の関係の情報を記述したもの。
 - 「送信テーブル (export)」, 「受信テーブル (import)」がある。
- 送信側: 「境界点」として送る
- 受信側: 「外点」として受け取る

一般化された通信テーブル:送信

- 送信相手
 - NeibPETot, NeibPE[neib]
- それぞれの送信相手に送るメッセージサイズ
 - export_index[neib], neib= 0, NeibPETot
- 「境界点」番号
 - export_item[k], k= 0, export_index[NeibPETot]-1
- それぞれの送信相手に送るメッセージ
 - SendBuf[k], k= 0, export_index[NeibPETot]-1

通信テーブル(送信)



```

4
13
14
15
16
4
3
6
9
12
0
0
0
0

```

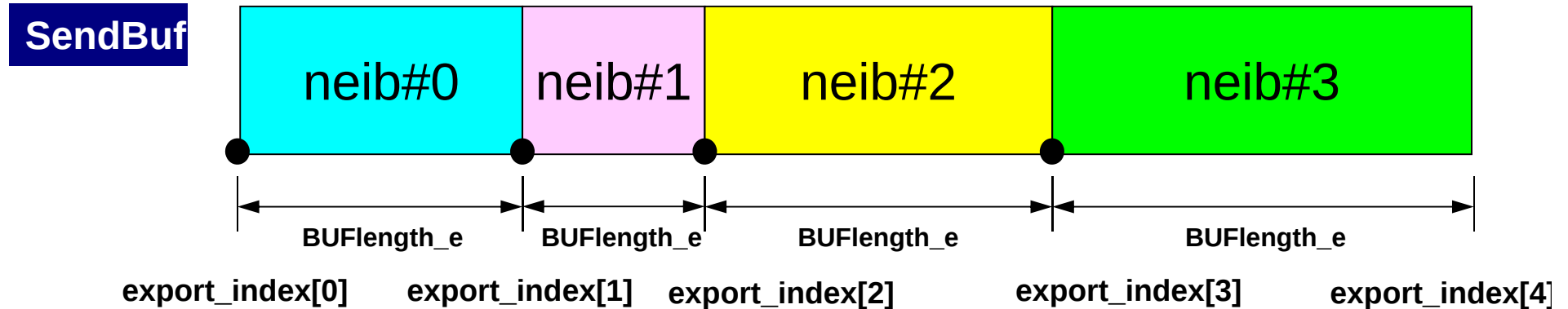
```

4
13
14
15
16
4
1
4
7
10
1
1
1
1
export_index(neib): 送信節点数
export_item: 節点番号

```

- export_index 各隣接領域に送信する外点の数
(累積数)
 - 現在: 隣接領域数は1
- export_item 境界点の番号

送信 (MPI_Isend/Irecv/Waitall)



export_index[neib] ~ export_index[neib+1]-1番目のexport_itemがneib番目の隣接領域に送信される

```
for (neib=0; neib<NeibPETot;neib++){
    for (k=export_index[neib];k<export_index[neib+1];k++){
        kk= export_item[k];
        SendBuf[k]= VAL[kk];
    }
}
```

送信バッファへの代入

```
for (neib=0; neib<NeibPETot; neib++){
    tag= 0;
    iS_e= export_index[neib];
    iE_e= export_index[neib+1];
    BUFlength_e= iE_e - iS_e

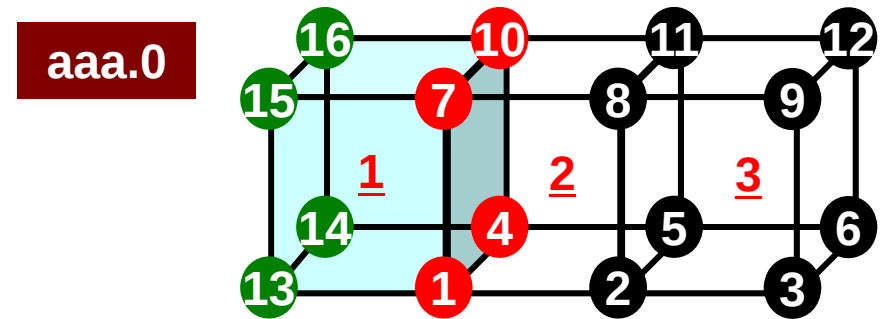
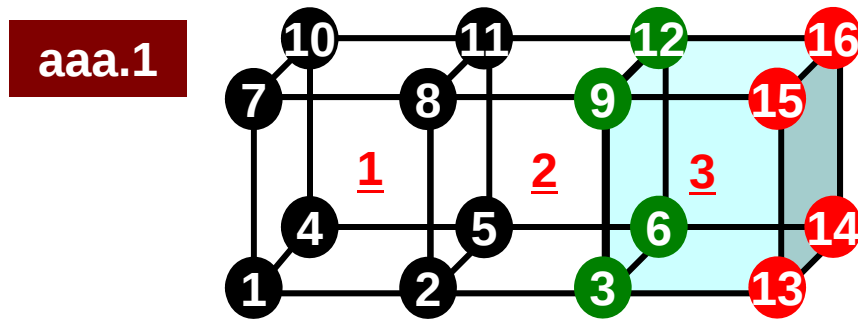
    ierr= MPI_Isend
        (&SendBuf[iS_e], BUFlength_e, MPI_DOUBLE, NeibPE[neib], 0,
         MPI_COMM_WORLD, &ReqSend[neib])
}
```

```
MPI_Waitall(NeibPETot, ReqSend, StatSend);
```

一般化された通信テーブル: 受信

- 受信相手
 - NeibPETot , NeibPE[neib]
- それぞれの受信相手から受け取るメッセージサイズ
 - import_index[neib], neib= 0, NeibPETot
- 「外点」番号
 - import_item[k], k= 0, import_index[NeibPETot]-1
- それぞれの受信相手から受け取るメッセージ
 - RecvBuf[k], k= 0, import_index[NeibPETot]-1

通信テーブル(受信)



```

4
13
14
15
16
4
3
6
9
12

```

```

0
0
0
0

```

```

4
13
14
15
16
4
1
7
10
import_index(neib)受信節点数
import_item  節点番号, 領域
export_index(neib)
export_item

```

- import_index 各隣接領域から受信する外点の数 (累積数)
 - 現在: 隣接領域数は1
- import_item 外点の番号, 所属領域

受信 (MPI_Isend/Irecv/Waitall)

```

for (neib=0; neib<NeibPETot; neib++){
    tag= 0;
    iS_i= import_index[neib];
    iE_i= import_index[neib+1];
    BUFlength_i= iE_i - iS_i

    ierr= MPI_Irecv
        (&RecvBuf[iS_i], BUFlength_i, MPI_DOUBLE, NeibPE[neib], 0,
         MPI_COMM_WORLD, &ReqRecv[neib])
}

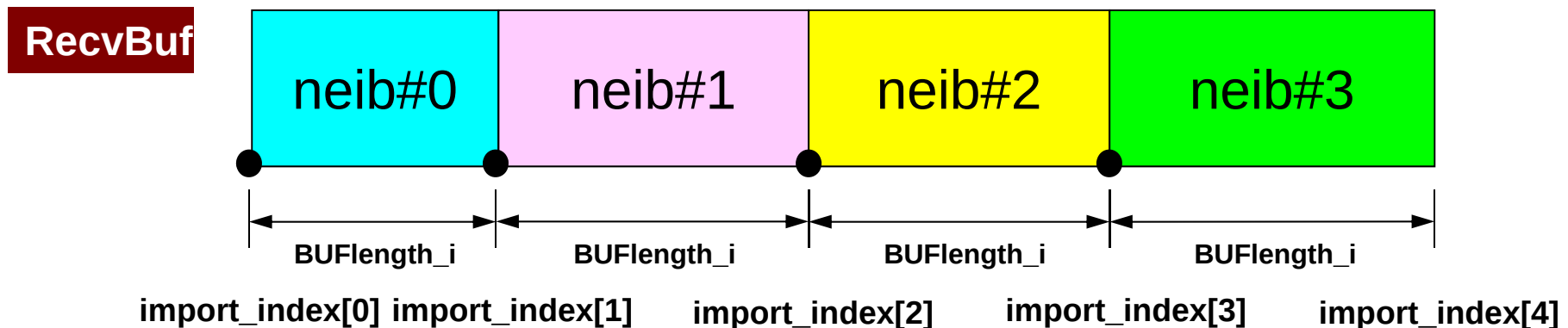
MPI_Waitall(NeibPETot, ReqRecv, StatRecv);

for (neib=0; neib<NeibPETot;neib++){
    for (k=import_index[neib];k<import_index[neib+1];k++){
        kk= import_item[k];
        VAL[kk]= RecvBuf[k];
    }
}

```

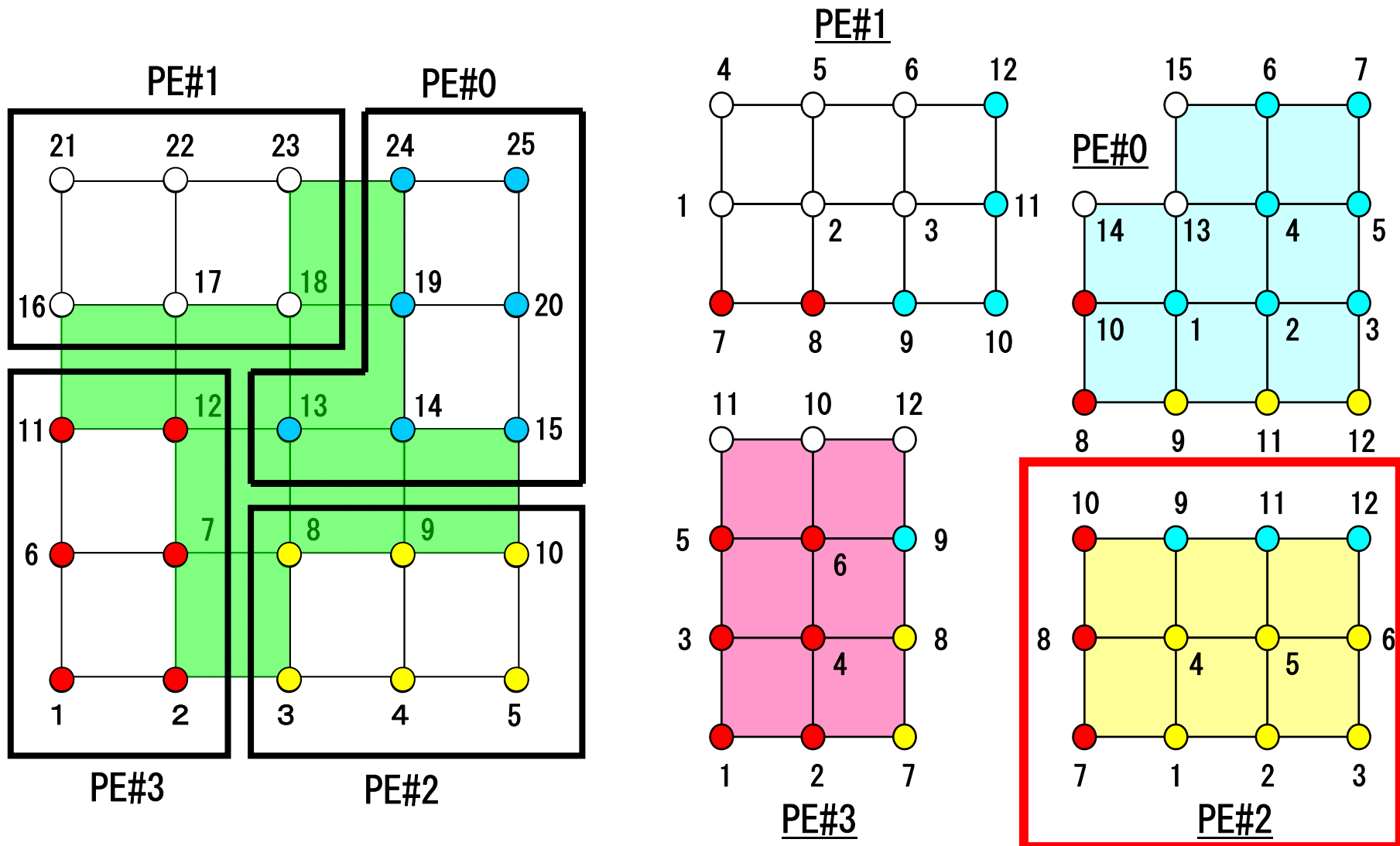
受信バッファからの代入

import_index[neib] ~ import_index[neib+1]-1番目のimport_itemがneib番目の隣接領域から受信される

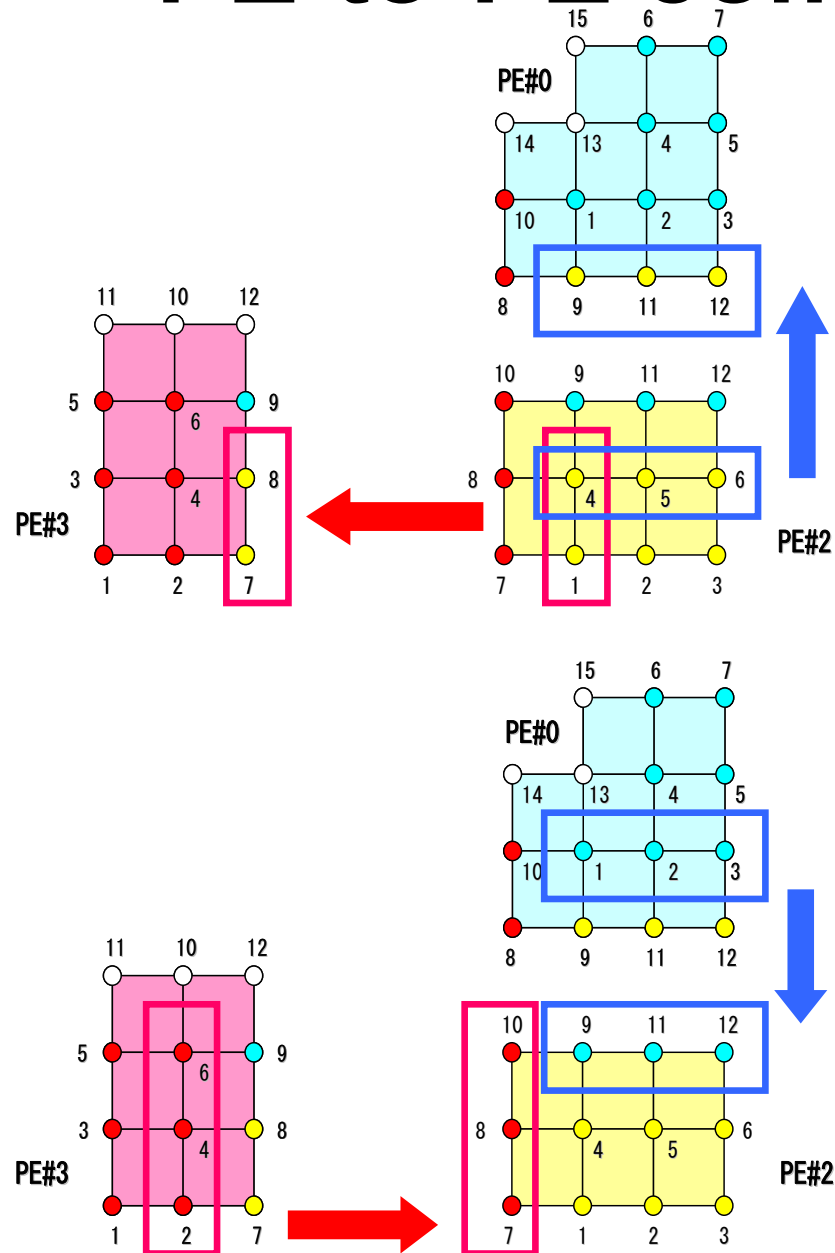


Node-based Partitioning

internal nodes - elements - external nodes



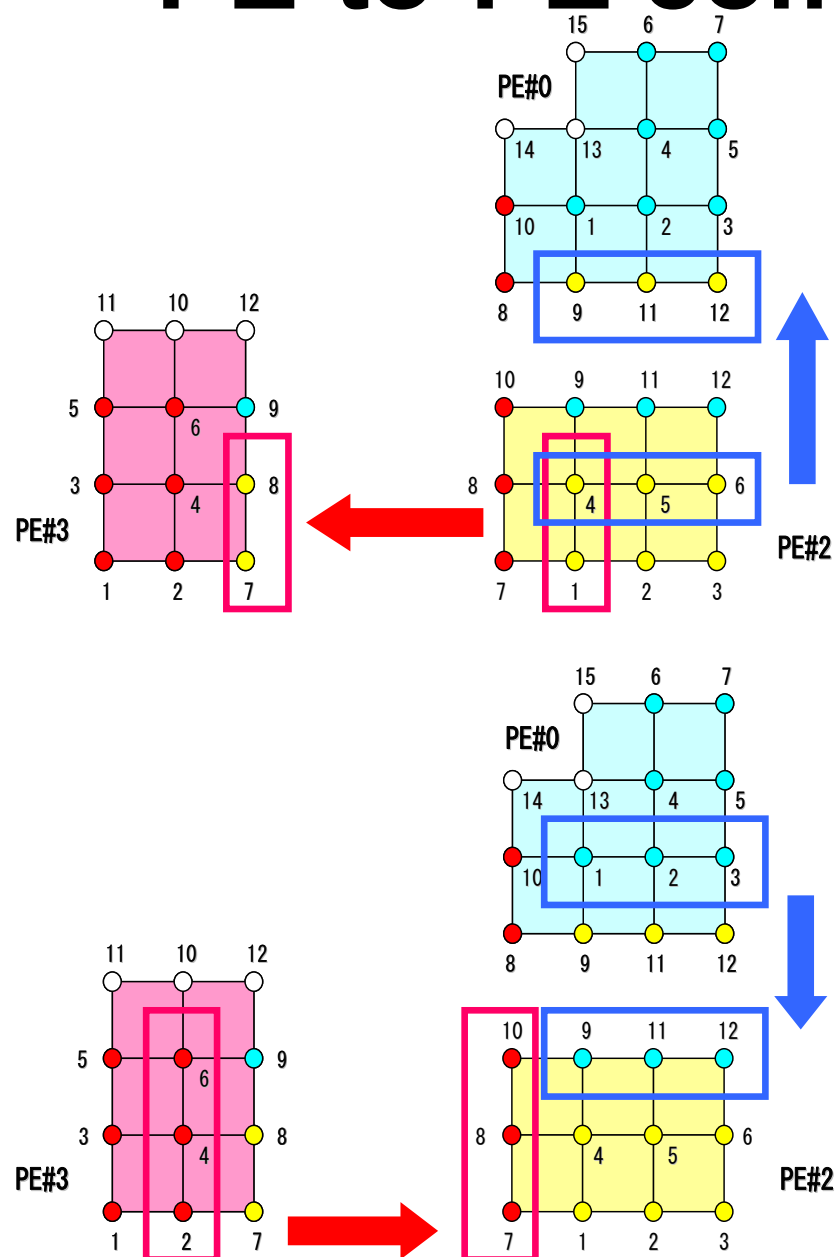
PE-to-PE comm. : Local Data



(中略)

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

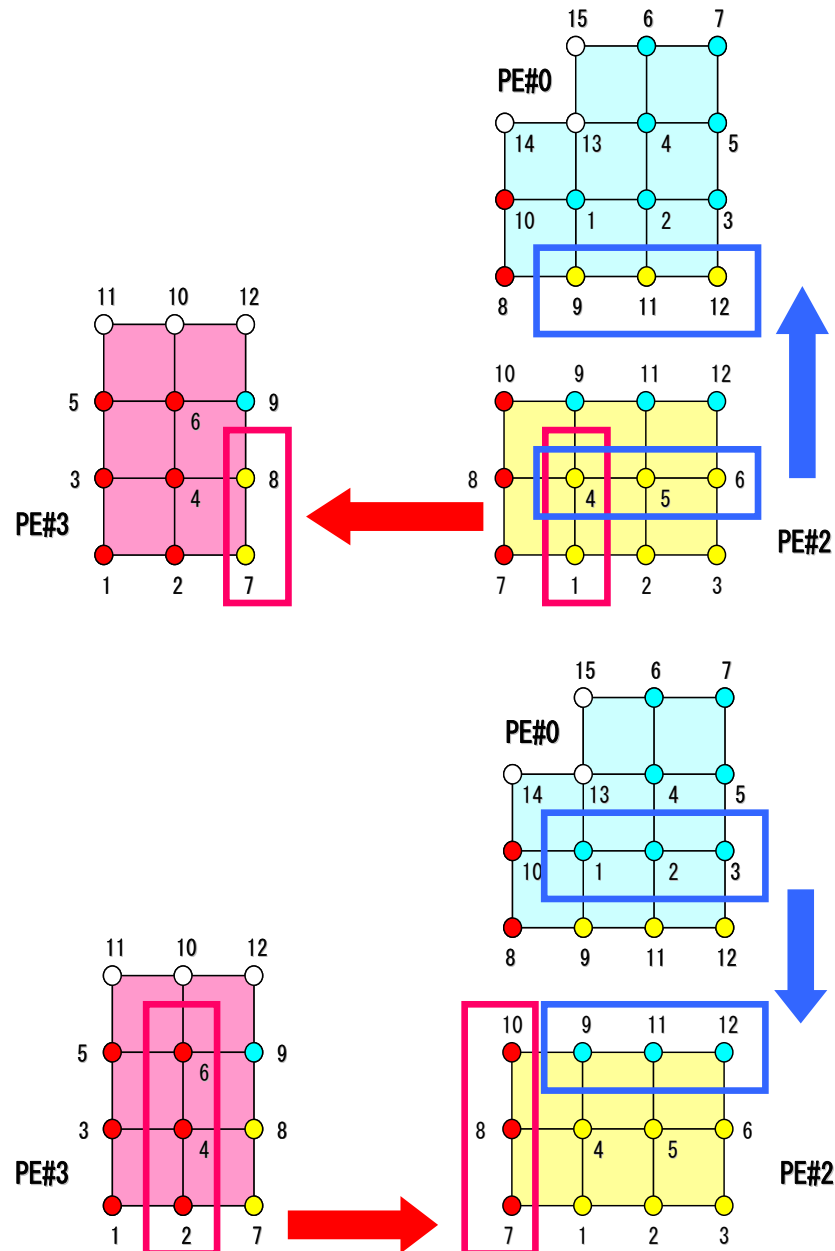
PE-to-PE comm. : Local Data



2	領域ID
2	隣接領域数
3	隣接領域
(中略)	0
3	6
7	3
8	3
10	3
9	0
11	0
12	0
2	0
1	5
4	
4	
5	
6	

NEIBPE= 2
 NEIBPE[0]=3, NEIBPE[1]= 0

PE-to-PE comm. : SEND



(中略)

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

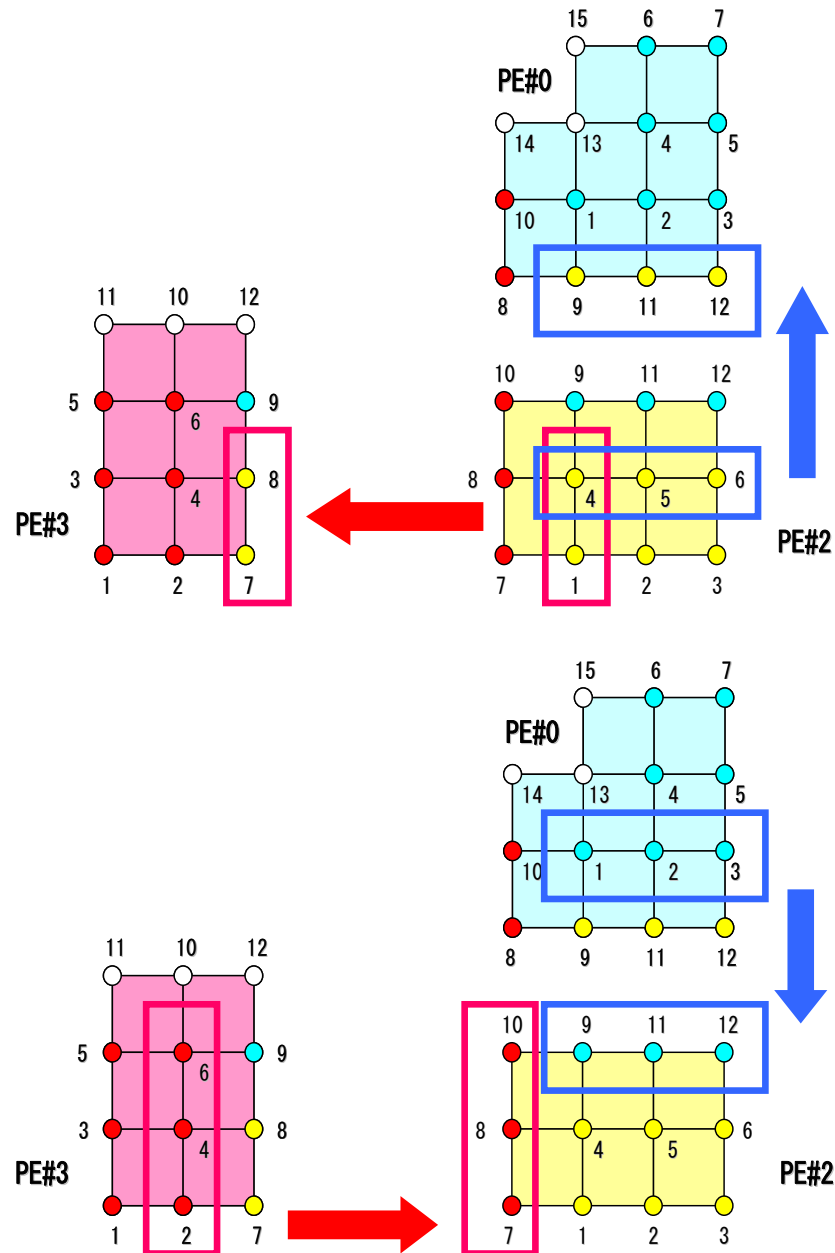
export_index

```
export_index[0]= 0
export_index[1]= 2
export_index[2]= 2+3 = 5
```

```
export_item[0-4]=1,4,4,5,6
```

4番の節点は2つの領域に送られる

PE-to-PE comm. : RECV



(中略)

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

import_index

```
import_index[0]= 0
import_index[1]= 3
import_index[2]= 3+3 = 6
```

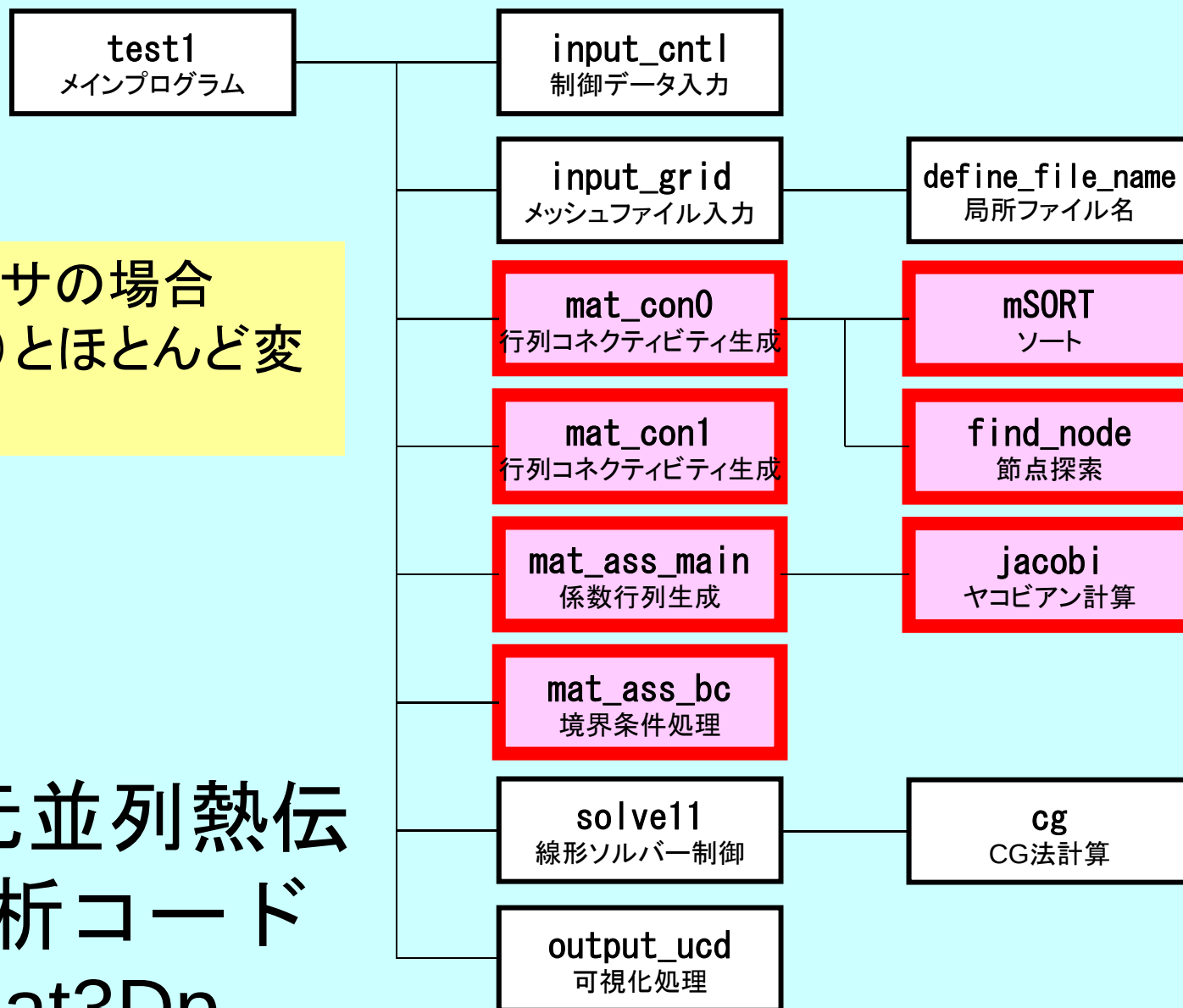
```
import_item[0-5]=7,8,10,9,11,12
```

並列有限要素法の処理：プログラム

- 初期化
 - 制御変数読み込み
 - 座標読み込み⇒要素生成 (N:節点数, NE: 要素数)
 - 配列初期化 (全体マトリクス, 要素マトリクス)
 - 要素⇒全体マトリクスマッピング (Index, Item)
- マトリクス生成
 - 要素単位の処理 (do icel= 1, NE)
 - 要素マトリクス計算
 - 全体マトリクスへの重ね合わせ
 - 境界条件の処理
- 連立一次方程式
 - 共役勾配法 (CG)

1プロセッサの場合
(heat3D)とほとんど変
わらない

三次元並列熱伝 導解析コード heat3Dp



マトリクス生成まで

- 一次元の場合は, index, itemに関連した情報を簡単に作ることができた
 - 非ゼロ非対角成分の数は2
 - 番号が自分に対して : +1と-1
- 三次元の場合はもっと複雑
 - 非ゼロ非対角ブロックの数は7~26 (現在の形状)
 - 実際はもっと複雑
 - 前以て, 非ゼロ非対角ブロックの数はわからない

マトリクス生成まで

- 一次元の場合は, index, itemに関連した情報を簡単に作ることができた
 - 非ゼロ非対角成分の数は2
 - 番号が自分に対して: +1と-1
- 三次元の場合はもっと複雑
 - 非ゼロ非対角ブロックの数は7~26 (現在の形状)
 - 実際はもっと複雑
 - 前以て, 非ゼロ非対角ブロックの数はわからない
- INLU[NP], IALU[NP][NLU] を使って非ゼロ非対角成分数を予備的に勘定する

全体処理

```
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"
extern void PFEM_INIT(int, char**);
extern void INPUT_CNTL();
extern void INPUT_GRID();
extern void MAT_CONO();
extern void MAT_CON1();
extern void MAT_ASS_MAIN();
extern void MAT_ASS_BC();
extern void SOLVE11();
extern void OUTPUT_UCD();
extern void PFEM_FINALIZE();
int main(int argc, char* argv[])
{
    double START_TIME, END_TIME;

    PFEM_INIT(argc, argv);

    INPUT_CNTL();
    INPUT_GRID();

    MAT_CONO();
    MAT_CON1();

    MAT_ASS_MAIN();
    MAT_ASS_BC();

    SOLVE11();

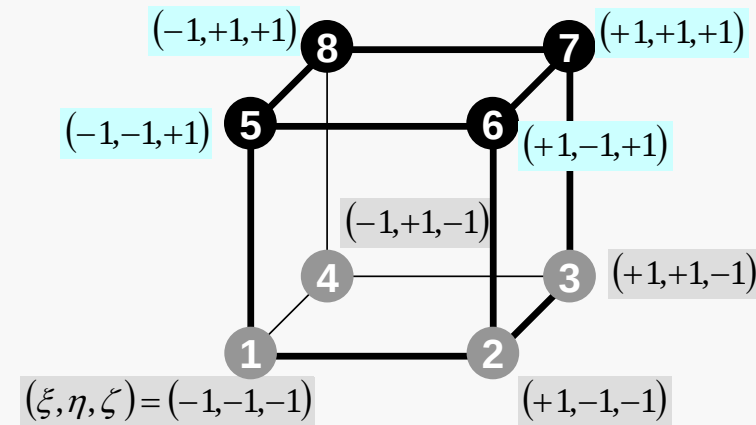
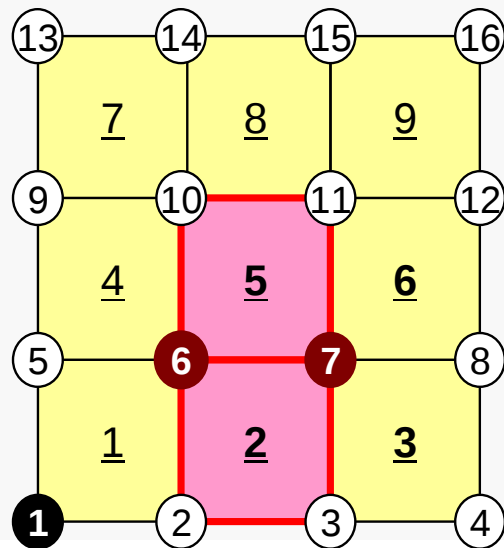
    OUTPUT_UCD();
    PFEM_FINALIZE();
}
```

MAT_CON0: INLU, IALU生成

MAT_CON1: indexLU, itemLU生成

MAT_CON0 : 全体構成

```
do icel= 1, ICELTOT
  8節点相互の関係から,
  INL, INU, IAL, IAUを生成
  (FIND_NODE)
enddo
```



行列コネクティビティ生成： MAT_CONO (1/4)

```
/**
** MAT_CONO
**/

#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"

extern FILE *fp_log;
/** external functions */
extern void mSORT(int*, int*, int);
/** static functions */
static void FIND_TS_NODE (int, int);

void MAT_CONO ()
{
    int i, j, k, icel, in;
    int in1, in2, in3, in4, in5, in6, in7, in8;
    int NN;

    NLU= 26;

    INLU=(KINT* ) allocate_vector (sizeof (KINT), NP);
    IALU=(KINT**) allocate_matrix (sizeof (KINT), NP, NLU);

    for (i=0; i<NP; i++) INLU[i]=0;
    for (i=0; i<NP; i++) for (j=0; j<NLU; j++) IALU[i][j]=0;
```

NLU:

各節点における
非ゼロ非対角成分
の最大数
(接続する節点数)

今の問題の場合は
わかっているので、
このようにできる

不明の場合の実装:
⇒レポート課題

行列コネクティビティ生成： MAT_CONO (1/4)

```

/**
** MAT_CONO
**/

#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"

extern FILE *fp_log;
/** external functions */
extern void mSORT(int*, int*, int);
/** static functions */
static void FIND_TS_NODE (int, int);

void MAT_CONO()
{
    int i, j, k, icel, in;
    int in1, in2, in3, in4, in5, in6, in7, in8;
    int NN;

    NLU= 26;

    INLU=(KINT* ) allocate_vector(sizeof(KINT), NP);
    IALU=(KINT**) allocate_matrix(sizeof(KINT), NP, NLU);

    for (i=0; i<NP; i++) INLU[i]=0;
    for (i=0; i<NP; i++) for (j=0; j<NLU; j++) IALU[i][j]=0;

```

変数名	サイズ	内 容
INLU	[NP]	各節点の非零非対角成分数
IALU	[NP] {[NLU]}	各節点の非零非対角成分（列番号）

行列コネクティビティ生成： MAT_CON0 (2/4)

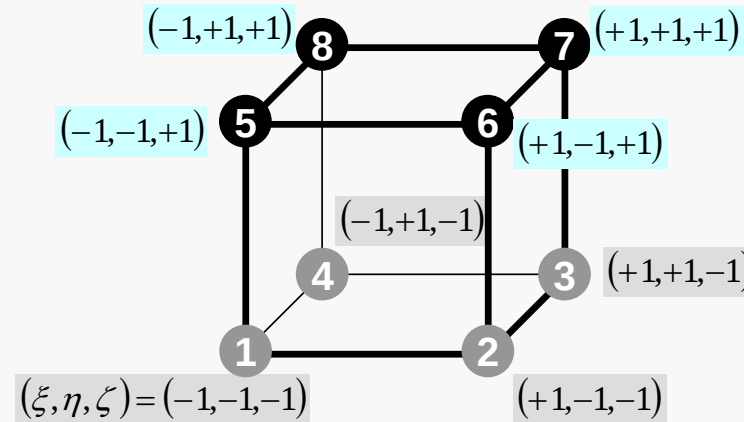
```
for( ice|=0;ice|< ICELTOT;ice|++) {
```

```
  in1=ICELNOD[ice|][0];
  in2=ICELNOD[ice|][1];
  in3=ICELNOD[ice|][2];
  in4=ICELNOD[ice|][3];
  in5=ICELNOD[ice|][4];
  in6=ICELNOD[ice|][5];
  in7=ICELNOD[ice|][6];
  in8=ICELNOD[ice|][7];
```

```
  FIND_TS_NODE (in1, in2);
  FIND_TS_NODE (in1, in3);
  FIND_TS_NODE (in1, in4);
  FIND_TS_NODE (in1, in5);
  FIND_TS_NODE (in1, in6);
  FIND_TS_NODE (in1, in7);
  FIND_TS_NODE (in1, in8);
```

```
  FIND_TS_NODE (in2, in1);
  FIND_TS_NODE (in2, in3);
  FIND_TS_NODE (in2, in4);
  FIND_TS_NODE (in2, in5);
  FIND_TS_NODE (in2, in6);
  FIND_TS_NODE (in2, in7);
  FIND_TS_NODE (in2, in8);
```

```
  FIND_TS_NODE (in3, in1);
  FIND_TS_NODE (in3, in2);
  FIND_TS_NODE (in3, in4);
  FIND_TS_NODE (in3, in5);
  FIND_TS_NODE (in3, in6);
  FIND_TS_NODE (in3, in7);
  FIND_TS_NODE (in3, in8);
```



節点探索：FIND_TS_NODE

INL,INU,IAL,IAU探索：一次元ではこの部分は手動

```

/**
 *** FIND_TS_NODE
 **/

static void FIND_TS_NODE (int ip1, int ip2)
{
    int kk, icou;

    for (kk=1; kk<=INLU[ip1-1]; kk++) {
        if (ip2 == IALU[ip1-1][kk-1]) return;
    }

    icou=INLU[ip1-1]+1;
    IALU[ip1-1][icou-1]=ip2;
    INLU[ip1-1]=icou;

    return;
}

```

変数名	サイズ	内 容
INLU	[N]	各節点の非零非対角成分数
IALU	[N] {[NLU]}	各節点の非零非対角成分（列番号）

節点探索 : FIND_TS_NODE

一次元ではこの部分は手動

```
/**
 *** FIND_TS_NODE
 **/

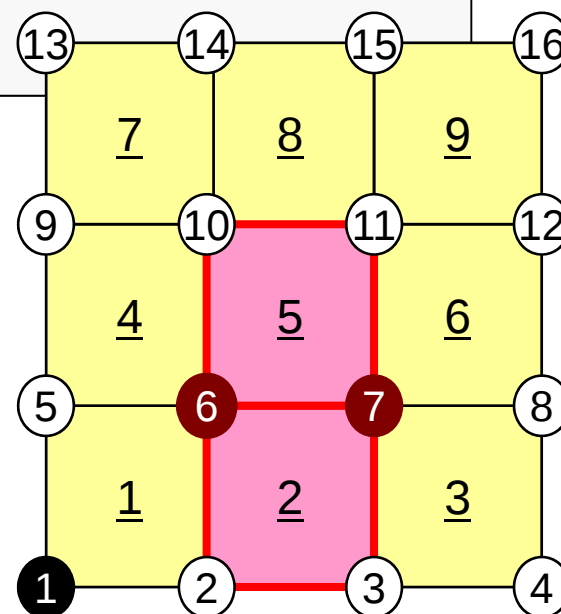
static void FIND_TS_NODE (int ip1, int ip2)
{
    int kk, icou;

    for (kk=1; kk<=INLU[ip1-1]; kk++) {
        if (ip2 == IALU[ip1-1][kk-1]) return;
    }

    icou=INLU[ip1-1]+1;
    IALU[ip1-1][icou-1]=ip2;
    INLU[ip1-1]=icou;

    return;
}
```

既にIALUに含まれている
場合は、次のペアへ



節点探索：FIND_TS_NODE

一次元ではこの部分は手動

```

/**
 *** FIND_TS_NODE
 **/

static void FIND_TS_NODE (int ip1, int ip2)
{
    int kk, icou;

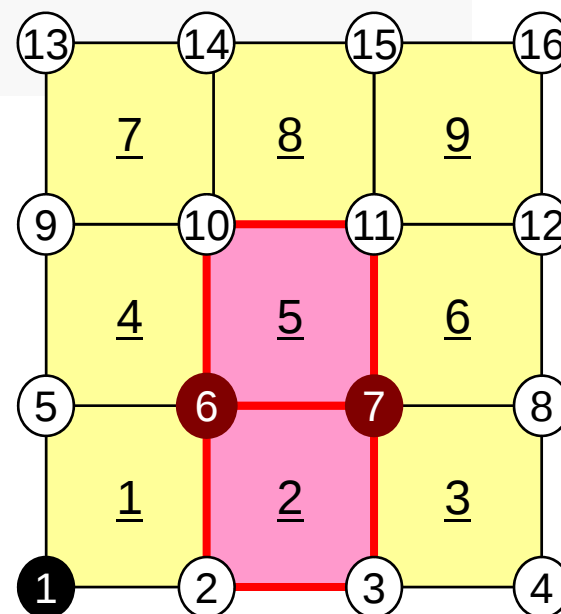
    for (kk=1; kk<=INLU[ip1-1]; kk++) {
        if (ip2 == IALU[ip1-1][kk-1]) return;
    }

    icou=INLU[ip1-1]+1;
    IALU[ip1-1][icou-1]=ip2;
    INLU[ip1-1]=icou;

    return;
}

```

IALUに含まれていない
場合は, INLUに1を加えて
IALUに格納



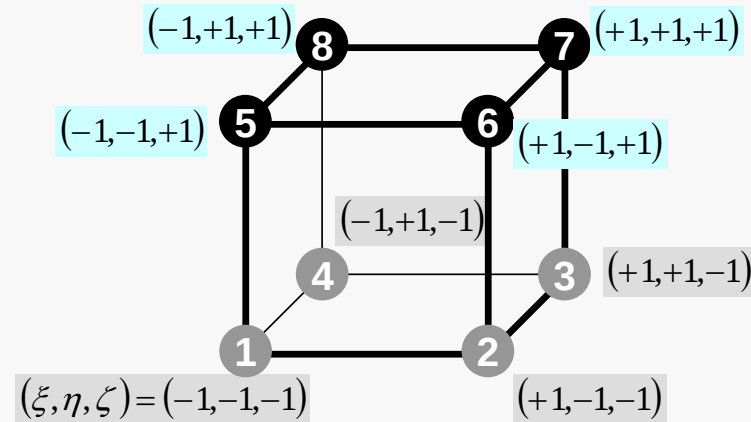
行列コネクティビティ生成： MAT_CON0 (3/4)

```
FIND_TS_NODE (in4, in1);
FIND_TS_NODE (in4, in2);
FIND_TS_NODE (in4, in3);
FIND_TS_NODE (in4, in5);
FIND_TS_NODE (in4, in6);
FIND_TS_NODE (in4, in7);
FIND_TS_NODE (in4, in8);
```

```
FIND_TS_NODE (in5, in1);
FIND_TS_NODE (in5, in2);
FIND_TS_NODE (in5, in3);
FIND_TS_NODE (in5, in4);
FIND_TS_NODE (in5, in6);
FIND_TS_NODE (in5, in7);
FIND_TS_NODE (in5, in8);
```

```
FIND_TS_NODE (in6, in1);
FIND_TS_NODE (in6, in2);
FIND_TS_NODE (in6, in3);
FIND_TS_NODE (in6, in4);
FIND_TS_NODE (in6, in5);
FIND_TS_NODE (in6, in7);
FIND_TS_NODE (in6, in8);
```

```
FIND_TS_NODE (in7, in1);
FIND_TS_NODE (in7, in2);
FIND_TS_NODE (in7, in3);
FIND_TS_NODE (in7, in4);
FIND_TS_NODE (in7, in5);
FIND_TS_NODE (in7, in6);
FIND_TS_NODE (in7, in8);
```



行列コネクティビティ生成： MAT_CON0 (4/4)

```
FIND_TS_NODE (in8, in1);  
FIND_TS_NODE (in8, in2);  
FIND_TS_NODE (in8, in3);  
FIND_TS_NODE (in8, in4);  
FIND_TS_NODE (in8, in5);  
FIND_TS_NODE (in8, in6);  
FIND_TS_NODE (in8, in7);  
}  
  
for (in=0; in<N; in++) {  
    NN=INLU[in];  
    for (k=0; k<NN; k++) {  
        NCOL1[k]=IALU[in][k];  
    }  
  
    mSORT (NCOL1, NCOL2, NN);  
  
    for (k=NN; k>0; k--) {  
        IALU[in][NN-k]= NCOL1 [NCOL2[k-1]-1];  
    }  
}
```

各節点において, IALU[i][k]が
小さい番号から大きい番号に
並ぶようにソート(単純なバブルソート)
せいぜい100程度のものをソートする

CRS形式への変換 : MAT_CON1

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1()
{
    int i, k, kk;

    indexLU= (KINT*) allocate_vector (sizeof (KINT), N+1);
    for (i=0; i<N+1; i++) indexLU[i]=0;

    for (i=0; i<NP; i++) {
        indexLU[i+1]=indexLU[i]+INLU[i];
    }

    NPLU=indexLU[NP];

    itemLU= (KINT*) allocate_vector (sizeof (KINT), NPLU);

    for (i=0; i<NP; i++) {
        for (k=0; k<INLU[i]; k++) {
            kk=k+indexLU[i];
            itemLU[kk]=IALU[i][k]-1;
        }
    }

    deallocate_vector (INLU);
    deallocate_vector (IALU);
}
```

C

$$\text{index}[i+1] = \sum_{k=0}^i \text{INLU}[k]$$

$$\text{index}[0] = 0$$

FORTRAN

$$\text{index}(i) = \sum_{k=1}^i \text{INLU}(k)$$

$$\text{index}(0) = 0$$

CRS形式への変換 : MAT_CON1

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1()
{
    int i, k, kk;

    indexLU= (KINT*) allocate_vector(sizeof(KINT), N+1);
    for (i=0; i<N+1; i++) indexLU[i]=0;

    for (i=0; i<NP; i++) {
        indexLU[i+1]=indexLU[i]+INLU[i];
    }

    NPLU=indexLU[NP];

    itemLU= (KINT*) allocate_vector(sizeof(KINT), NPLU);

    for (i=0; i<NP; i++) {
        for (k=0; k<INLU[i]; k++) {
            kk=k+indexLU[i];
            itemLU[kk]=IALU[i][k]-1;
        }
    }

    deallocate_vector(INLU);
    deallocate_vector(IALU);
}
```

NPLU=index[NP]
itemのサイズ
非ゼロ非対角成分総数

CRS形式への変換 : MAT_CON1

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1 ()
{
    int i, k, kk;

    indexLU= (KINT*) allocate_vector (sizeof (KINT), N+1);
    for (i=0; i<N+1; i++) indexLU[i]=0;

    for (i=0; i<NP; i++) {
        indexLU[i+1]=indexLU[i]+INLU[i];
    }

    NPLU=indexLU[NP];

    itemLU= (KINT*) allocate_vector (sizeof (KINT), NPLU);

    for (i=0; i<NP; i++) {
        for (k=0; k<INLU[i]; k++) {
            kk=k+indexLU[i];
            itemLU[kk]=IALU[i][k]-1;
        }
    }

    deallocate_vector (INLU);
    deallocate_vector (IALU);
}
```

itemに1から始まる
節点番号を記憶

CRS形式への変換 : MAT_CON1

```
#include <stdio.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE* fp_log;
void MAT_CON1 ()
{
    int i, k, kk;

    indexLU= (KINT*) allocate_vector (sizeof (KINT), N+1);
    for (i=0; i<N+1; i++) indexLU[i]=0;

    for (i=0; i<NP; i++) {
        indexLU[i+1]=indexLU[i]+INLU[i];
    }

    NPLU=indexLU[NP];

    itemLU= (KINT*) allocate_vector (sizeof (KINT), NPLU);

    for (i=0; i<NP; i++) {
        for (k=0; k<INLU[i]; k++) {
            kk=k+indexLU[i];
            itemLU[kk]=IALU[i][k]-1;
        }
    }

    deallocate_vector (INLU);
    deallocate_vector (IALU);
}
```

これらはもはや不要

全体処理

```
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"
extern void PFEM_INIT(int, char**);
extern void INPUT_CNTL();
extern void INPUT_GRID();
extern void MAT_CONO();
extern void MAT_CON1();
extern void MAT_ASS_MAIN();
extern void MAT_ASS_BC();
extern void SOLVE11();
extern void OUTPUT_UCD();
extern void PFEM_FINALIZE();
int main(int argc, char* argv[])
{
    double START_TIME, END_TIME;

    PFEM_INIT(argc, argv);

    INPUT_CNTL();
    INPUT_GRID();

    MAT_CONO();
    MAT_CON1();

    MAT_ASS_MAIN();
    MAT_ASS_BC();

    SOLVE11();

    OUTPUT_UCD();
    PFEM_FINALIZE();
}
```

MAT_ASS_MAIN : 全体構成

```

do kpn= 1, 2      ガウス積分点番号 (ζ方向)
  do jpn= 1, 2    ガウス積分点番号 (η方向)
    do ipn= 1, 2  ガウス積分点番号 (ξ方向)
      ガウス積分点 (8個) における形状関数,
      およびその「自然座標系」における微分の算出
    enddo
  enddo
enddo

```

```

do icel= 1, ICELTOT 要素ループ
  8節点の座標から, ガウス積分点における, 形状関数の「全体座標系」における微分,
  およびヤコビアンを算出 (JACOBI)

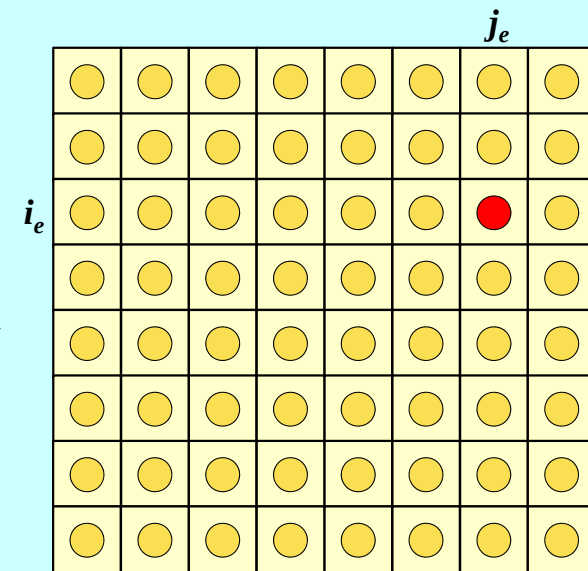
```

```

do ie= 1, 8      局所節点番号
  do je= 1, 8    局所節点番号
    全体節点番号 : ip, jp
    Aip, jp の itemLUI におけるアドレス : kk

    do kpn= 1, 2  ガウス積分点番号 (ζ方向)
      do jpn= 1, 2  ガウス積分点番号 (η方向)
        do ipn= 1, 2  ガウス積分点番号 (ξ方向)
          要素積分⇒要素行列成分計算, 全体行列への足しこみ
        enddo
      enddo
    enddo
  enddo
enddo
enddo

```



係数行列 : MAT_ASS_MAIN (1/6)

```
#include <stdio.h>
#include <math.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
extern void JACOBI();
void MAT_ASS_MAIN()
{
    int i, k, kk;
    int ip, jp, kp;
    int ipn, jpn, kpn;
    int icel;
    int ie, je;
    int iiS, iiE;
    int in1, in2, in3, in4, in5, in6, in7, in8;
    double SHi;
    double QP1, QM1, EP1, EM1, TP1, TM1;
    double X1, X2, X3, X4, X5, X6, X7, X8;
    double Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8;
    double Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8;
    double PNXi, PNYi, PNZi, PNXj, PNYj, PNZj;
    double CONDO, QVO, QVC, COEFij;
    double coef;
```

```
KINT nodLOCAL[8];
```

```
AMAT=(KREAL*) allocate_vector(sizeof(KREAL), NPLU);
B =(KREAL*) allocate_vector(sizeof(KREAL), NP );
D =(KREAL*) allocate_vector(sizeof(KREAL), NP);
X =(KREAL*) allocate_vector(sizeof(KREAL), NP);
```

係数行列 (非零非対角成分)
 右辺ベクトル
 解ベクトル
 係数行列 (対角成分)

```
for (i=0; i<NPLU; i++) AMAT[i]=0.0;
for (i=0; i<NP ; i++) B[i]=0.0;
for (i=0; i<NP ; i++) D[i]=0.0;
for (i=0; i<NP ; i++) X[i]=0.0;
```

```
WEI[0]= 1.0000000000e0;
WEI[1]= 1.0000000000e0;
POS[0]= -0.5773502692e0;
POS[1]= 0.5773502692e0;
```

係数行列 : MAT_ASS_MAIN (1/6)

```
#include <stdio.h>
#include <math.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
extern void JACOBI();
void MAT_ASS_MAIN()
{
    int i, k, kk;
    int ip, jp, kp;
    int ipn, jpn, kpn;
    int icel;
    int ie, je;
    int iiS, iiE;
    int in1, in2, in3, in4, in5, in6, in7, in8;
    double SHi;
    double QP1, QM1, EP1, EM1, TP1, TM1;
    double X1, X2, X3, X4, X5, X6, X7, X8;
    double Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8;
    double Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8;
    double PNXi, PNYi, PNZi, PNXj, PNYj, PNZj;
    double CONDO, QVO, QVC, COEFij;
    double coef;
```

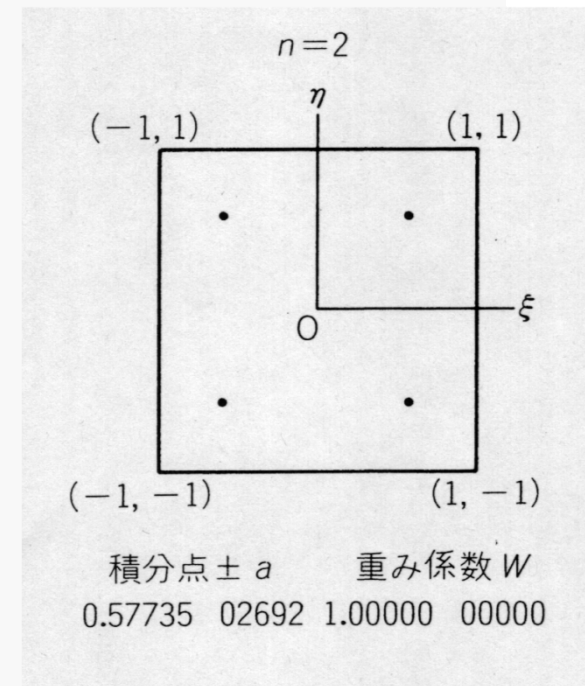
```
KINT nodLOCAL[8];
```

```
AMAT=(KREAL*) allocate_vector(sizeof(KREAL), NPLU);
B=(KREAL*) allocate_vector(sizeof(KREAL), NP);
D=(KREAL*) allocate_vector(sizeof(KREAL), NP);
X=(KREAL*) allocate_vector(sizeof(KREAL), NP);
```

```
for(i=0; i<NPLU; i++) AMAT[i]=0.0;
for(i=0; i<NP; i++) B[i]=0.0;
for(i=0; i<NP; i++) D[i]=0.0;
for(i=0; i<NP; i++) X[i]=0.0;
```

```
WEI[0]= 1.000000000e0;
WEI[1]= 1.000000000e0;
POS[0]= -0.5773502692e0;
POS[1]= 0.5773502692e0;
```

POS: 積分点座標
WEI: 重み係数



系数行列：MAT_ASS_MAIN (2/6)

```

/**
  INIT.
  PNQ - 1st-order derivative of shape function by QSI
  PNE - 1st-order derivative of shape function by ETA
  PNT - 1st-order derivative of shape function by ZET
***/

for(ip=0; ip<2; ip++) {
  for(jp=0; jp<2; jp++) {
    for(kp=0; kp<2; kp++) {

      QP1= 1. e0 + POS[ip];
      QM1= 1. e0 - POS[ip];
      EP1= 1. e0 + POS[jp];
      EM1= 1. e0 - POS[jp];
      TP1= 1. e0 + POS[kp];
      TM1= 1. e0 - POS[kp];

      SHAPE[ip][jp][kp][0]= 08th * QM1 * EM1 * TM1;
      SHAPE[ip][jp][kp][1]= 08th * QP1 * EM1 * TM1;
      SHAPE[ip][jp][kp][2]= 08th * QP1 * EP1 * TM1;
      SHAPE[ip][jp][kp][3]= 08th * QM1 * EP1 * TM1;
      SHAPE[ip][jp][kp][4]= 08th * QM1 * EM1 * TP1;
      SHAPE[ip][jp][kp][5]= 08th * QP1 * EM1 * TP1;
      SHAPE[ip][jp][kp][6]= 08th * QP1 * EP1 * TP1;
      SHAPE[ip][jp][kp][7]= 08th * QM1 * EP1 * TP1;
    }
  }
}

```

系数行列：MAT_ASS_MAIN (2/6)

```

/**
  INIT.
  PNQ - 1st-order derivative of shape function by QSI
  PNE - 1st-order derivative of shape function by ETA
  PNT - 1st-order derivative of shape function by ZET
***/

```

```

for(ip=0; ip<2; ip++) {
  for(jp=0; jp<2; jp++) {
    for(kp=0; kp<2; kp++) {

```

```

      QP1= 1. e0 + POS[ip];
      QM1= 1. e0 - POS[ip];
      EP1= 1. e0 + POS[jp];
      EM1= 1. e0 - POS[jp];
      TP1= 1. e0 + POS[kp];
      TM1= 1. e0 - POS[kp];

```

```

      SHAPE[ip][jp][kp][0]= 08th * QM1 * EM1 * TM1;
      SHAPE[ip][jp][kp][1]= 08th * QP1 * EM1 * TM1;
      SHAPE[ip][jp][kp][2]= 08th * QP1 * EP1 * TM1;
      SHAPE[ip][jp][kp][3]= 08th * QM1 * EP1 * TM1;
      SHAPE[ip][jp][kp][4]= 08th * QM1 * EM1 * TP1;
      SHAPE[ip][jp][kp][5]= 08th * QP1 * EM1 * TP1;
      SHAPE[ip][jp][kp][6]= 08th * QP1 * EP1 * TP1;
      SHAPE[ip][jp][kp][7]= 08th * QM1 * EP1 * TP1;

```

$$QP1(i) = (1 + \xi_i), \quad QM1(i) = (1 - \xi_i)$$

$$EP1(j) = (1 + \eta_j), \quad EM1(j) = (1 - \eta_j)$$

$$TP1(k) = (1 + \zeta_k), \quad TM1(k) = (1 - \zeta_k)$$

系数行列：MAT_ASS_MAIN (2/6)

```

/**
  INIT.
  PNQ - 1st-order derivative of shape function by QSI
  PNE - 1st-order derivative of shape function by ETA
  PNT - 1st-order derivative of shape function by ZET
***/

```

```

for(ip=0; ip<2; ip++) {
  for(jp=0; jp<2; jp++) {
    for(kp=0; kp<2; kp++) {

```

```

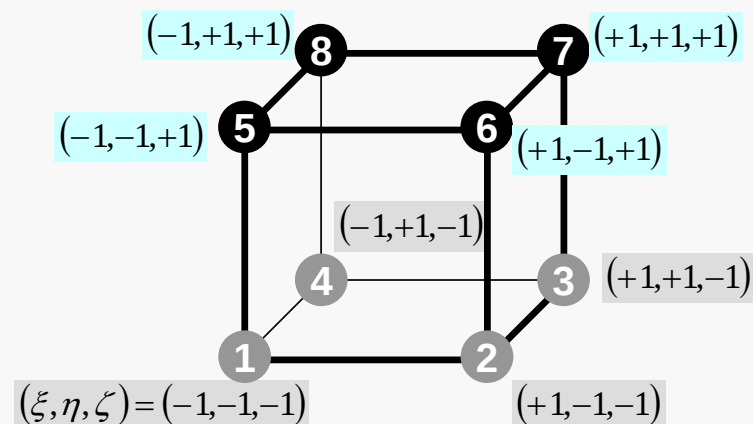
      QP1= 1. e0 + POS[ip];
      QM1= 1. e0 - POS[ip];
      EP1= 1. e0 + POS[jp];
      EM1= 1. e0 - POS[jp];
      TP1= 1. e0 + POS[kp];
      TM1= 1. e0 - POS[kp];

```

```

      SHAPE[ip][jp][kp][0]= 08th * QM1 * EM1 * TM1;
      SHAPE[ip][jp][kp][1]= 08th * QP1 * EM1 * TM1;
      SHAPE[ip][jp][kp][2]= 08th * QP1 * EP1 * TM1;
      SHAPE[ip][jp][kp][3]= 08th * QM1 * EP1 * TM1;
      SHAPE[ip][jp][kp][4]= 08th * QM1 * EM1 * TP1;
      SHAPE[ip][jp][kp][5]= 08th * QP1 * EM1 * TP1;
      SHAPE[ip][jp][kp][6]= 08th * QP1 * EP1 * TP1;
      SHAPE[ip][jp][kp][7]= 08th * QM1 * EP1 * TP1;

```



系数行列：MAT_ASS_MAIN (2/6)

```

/**
  INIT.
  PNQ - 1st-order derivative of shape function by QSI
  PNE - 1st-order derivative of shape function by ETA
  PNT - 1st-order derivative of shape function by ZET
***/

for(ip=0; ip<2; ip++) {
  for(jp=0; jp<2; jp++) {
    for(kp=0; kp<2; kp++) {

      QP1= 1. e0 + POS[ip];
      QM1= 1. e0 - POS[ip];
      EP1= 1. e0 + POS[jp];
      EM1= 1. e0 - POS[jp];
      TP1= 1. e0 + POS[kp];
      TM1= 1. e0 - POS[kp];

      SHAPE[ip][jp][kp][0]= 08th * QM1 * EM1 * TM1;
      SHAPE[ip][jp][kp][1]= 08th * QP1 * EM1 * TM1;
      SHAPE[ip][jp][kp][2]= 08th * QP1 * EP1 * TM1;
      SHAPE[ip][jp][kp][3]= 08th * QM1 * EP1 * TM1;
      SHAPE[ip][jp][kp][4]= 08th * QM1 * EM1 * TP1;
      SHAPE[ip][jp][kp][5]= 08th * QP1 * EM1 * TP1;
      SHAPE[ip][jp][kp][6]= 08th * QP1 * EP1 * TP1;
      SHAPE[ip][jp][kp][7]= 08th * QM1 * EP1 * TP1;
    }
  }
}

```

$$N_1(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1-\eta)(1-\zeta)$$

$$N_2(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1-\eta)(1-\zeta)$$

$$N_3(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1+\eta)(1-\zeta)$$

$$N_4(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1+\eta)(1-\zeta)$$

$$N_5(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1-\eta)(1+\zeta)$$

$$N_6(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1-\eta)(1+\zeta)$$

$$N_7(\xi, \eta, \zeta) = \frac{1}{8}(1+\xi)(1+\eta)(1+\zeta)$$

$$N_8(\xi, \eta, \zeta) = \frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)$$

係数行列 : MAT_ASS_MAIN (3/6)

```

PNQ[jp][kp][0] = - 08th * EM1 * TM1;
PNQ[jp][kp][1] = + 08th * EM1 * TM1;
PNQ[jp][kp][2] = + 08th * EP1 * TM1;
PNQ[jp][kp][3] = - 08th * EP1 * TM1;
PNQ[jp][kp][4] = - 08th * EM1 * TP1;
PNQ[jp][kp][5] = + 08th * EM1 * TP1;
PNQ[jp][kp][6] = + 08th * EP1 * TP1;
PNQ[jp][kp][7] = - 08th * EP1 * TP1;

```

```

PNE[ip][kp][0] = - 08th * QM1 * TM1;
PNE[ip][kp][1] = - 08th * QP1 * TM1;
PNE[ip][kp][2] = + 08th * QP1 * TM1;
PNE[ip][kp][3] = + 08th * QM1 * TM1;
PNE[ip][kp][4] = - 08th * QM1 * TP1;
PNE[ip][kp][5] = - 08th * QP1 * TP1;
PNE[ip][kp][6] = + 08th * QP1 * TP1;
PNE[ip][kp][7] = + 08th * QM1 * TP1;

```

```

PNT[ip][jp][0] = - 08th * QM1 * EM1;
PNT[ip][jp][1] = - 08th * QP1 * EM1;
PNT[ip][jp][2] = - 08th * QP1 * EP1;
PNT[ip][jp][3] = - 08th * QM1 * EP1;
PNT[ip][jp][4] = + 08th * QM1 * EM1;
PNT[ip][jp][5] = + 08th * QP1 * EM1;
PNT[ip][jp][6] = + 08th * QP1 * EP1;
PNT[ip][jp][7] = + 08th * QM1 * EP1;

```

```

    }
  }
  for( icel=0; icel< ICELTOT; icel++) {
    CONDO= COND;

```

```

    in1=ICELNOD[icel][0];
    in2=ICELNOD[icel][1];
    in3=ICELNOD[icel][2];
    in4=ICELNOD[icel][3];
    in5=ICELNOD[icel][4];
    in6=ICELNOD[icel][5];
    in7=ICELNOD[icel][6];
    in8=ICELNOD[icel][7];

```

$$PNQ(j,k) = \frac{\partial N_l}{\partial \xi} (\xi = \xi_i, \eta = \eta_j, \zeta = \zeta_k)$$

$$PNE(i,k) = \frac{\partial N_l}{\partial \eta} (\xi = \xi_i, \eta = \eta_j, \zeta = \zeta_k)$$

$$PNT(i,j) = \frac{\partial N_l}{\partial \zeta} (\xi = \xi_i, \eta = \eta_j, \zeta = \zeta_k)$$

$$\frac{\partial N_1}{\partial \xi} (\xi_i, \eta_j, \zeta_k) = -\frac{1}{8} (1 - \eta_j) (1 - \zeta_k)$$

$$\frac{\partial N_2}{\partial \xi} (\xi_i, \eta_j, \zeta_k) = +\frac{1}{8} (1 - \eta_j) (1 - \zeta_k)$$

$$\frac{\partial N_3}{\partial \xi} (\xi_i, \eta_j, \zeta_k) = +\frac{1}{8} (1 + \eta_j) (1 - \zeta_k)$$

$$\frac{\partial N_3}{\partial \xi} (\xi_i, \eta_j, \zeta_k) = -\frac{1}{8} (1 + \eta_j) (1 - \zeta_k)$$

(ξ_i, η_j, ζ_k) における形状関数の一階微分

系数行列：MAT_ASS_MAIN (3/6)

```

PNQ[ip][kp][0] = - 08th * EM1 * TM1;
PNQ[ip][kp][1] = + 08th * EM1 * TM1;
PNQ[ip][kp][2] = + 08th * EP1 * TM1;
PNQ[ip][kp][3] = - 08th * EP1 * TM1;
PNQ[ip][kp][4] = - 08th * EM1 * TP1;
PNQ[ip][kp][5] = + 08th * EM1 * TP1;
PNQ[ip][kp][6] = + 08th * EP1 * TP1;
PNQ[ip][kp][7] = - 08th * EP1 * TP1;

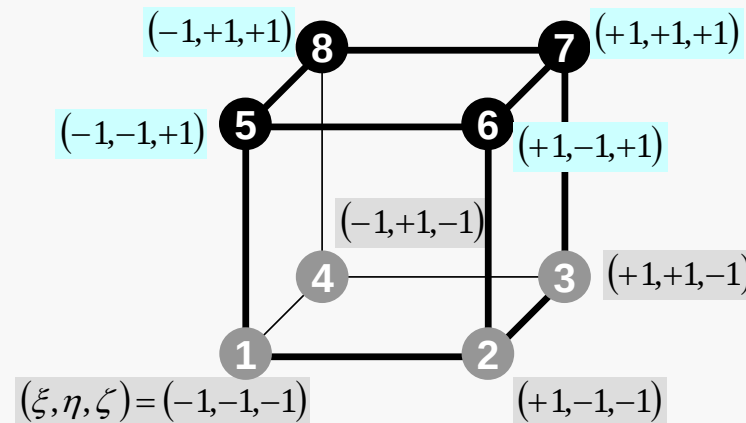
PNE[ip][kp][0] = - 08th * QM1 * TM1;
PNE[ip][kp][1] = - 08th * QP1 * TM1;
PNE[ip][kp][2] = + 08th * QP1 * TM1;
PNE[ip][kp][3] = + 08th * QM1 * TM1;
PNE[ip][kp][4] = - 08th * QM1 * TP1;
PNE[ip][kp][5] = - 08th * QP1 * TP1;
PNE[ip][kp][6] = + 08th * QP1 * TP1;
PNE[ip][kp][7] = + 08th * QM1 * TP1;

PNT[ip][jp][0] = - 08th * QM1 * EM1;
PNT[ip][jp][1] = - 08th * QP1 * EM1;
PNT[ip][jp][2] = - 08th * QP1 * EP1;
PNT[ip][jp][3] = - 08th * QM1 * EP1;
PNT[ip][jp][4] = + 08th * QM1 * EM1;
PNT[ip][jp][5] = + 08th * QP1 * EM1;
PNT[ip][jp][6] = + 08th * QP1 * EP1;
PNT[ip][jp][7] = + 08th * QM1 * EP1;
    }
}

for( icel=0; icel< ICELTOT; icel++) {
    CONDO= COND;

    in1=ICELNOD[icel][0];
    in2=ICELNOD[icel][1];
    in3=ICELNOD[icel][2];
    in4=ICELNOD[icel][3];
    in5=ICELNOD[icel][4];
    in6=ICELNOD[icel][5];
    in7=ICELNOD[icel][6];
    in8=ICELNOD[icel][7];

```



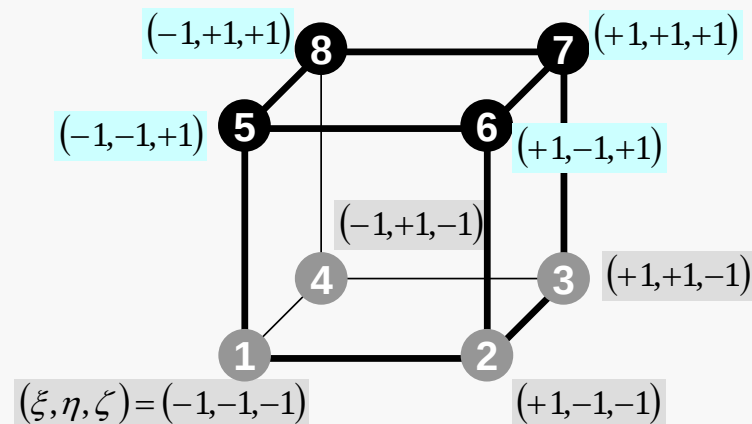
係数行列 : MAT_ASS_MAIN (4/6)

```

nodLOCAL[0]= in1;
nodLOCAL[1]= in2;
nodLOCAL[2]= in3;
nodLOCAL[3]= in4;
nodLOCAL[4]= in5;
nodLOCAL[5]= in6;
nodLOCAL[6]= in7;
nodLOCAL[7]= in8;

```

8節点の節点番号



```

X1=XYZ[in1-1][0];
X2=XYZ[in2-1][0];
X3=XYZ[in3-1][0];
X4=XYZ[in4-1][0];
X5=XYZ[in5-1][0];
X6=XYZ[in6-1][0];
X7=XYZ[in7-1][0];
X8=XYZ[in8-1][0];

```

```

Y1=XYZ[in1-1][1];
Y2=XYZ[in2-1][1];
Y3=XYZ[in3-1][1];
Y4=XYZ[in4-1][1];
Y5=XYZ[in5-1][1];
Y6=XYZ[in6-1][1];
Y7=XYZ[in7-1][1];
Y8=XYZ[in8-1][1];

```

```
QVC= 08th*(X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8);
```

```

Z1=XYZ[in1-1][2];
Z2=XYZ[in2-1][2];
Z3=XYZ[in3-1][2];
Z4=XYZ[in4-1][2];
Z5=XYZ[in5-1][2];
Z6=XYZ[in6-1][2];
Z7=XYZ[in7-1][2];
Z8=XYZ[in8-1][2];

```

```

JACOBI (DETJ, PNQ, PNE, PNT, PNX, PNY, PNZ,
X1, X2, X3, X4, X5, X6, X7, X8,
Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8);

```

係数行列 : MAT_ASS_MAIN (4/6)

```
nodLOCAL[0]= in1;
nodLOCAL[1]= in2;
nodLOCAL[2]= in3;
nodLOCAL[3]= in4;
nodLOCAL[4]= in5;
nodLOCAL[5]= in6;
nodLOCAL[6]= in7;
nodLOCAL[7]= in8;
```

```
X1=XYZ[in1-1][0];
X2=XYZ[in2-1][0];
X3=XYZ[in3-1][0];
X4=XYZ[in4-1][0];
X5=XYZ[in5-1][0];
X6=XYZ[in6-1][0];
X7=XYZ[in7-1][0];
X8=XYZ[in8-1][0];
```

8節点のX座標

```
Y1=XYZ[in1-1][1];
Y2=XYZ[in2-1][1];
Y3=XYZ[in3-1][1];
Y4=XYZ[in4-1][1];
Y5=XYZ[in5-1][1];
Y6=XYZ[in6-1][1];
Y7=XYZ[in7-1][1];
Y8=XYZ[in8-1][1];
```

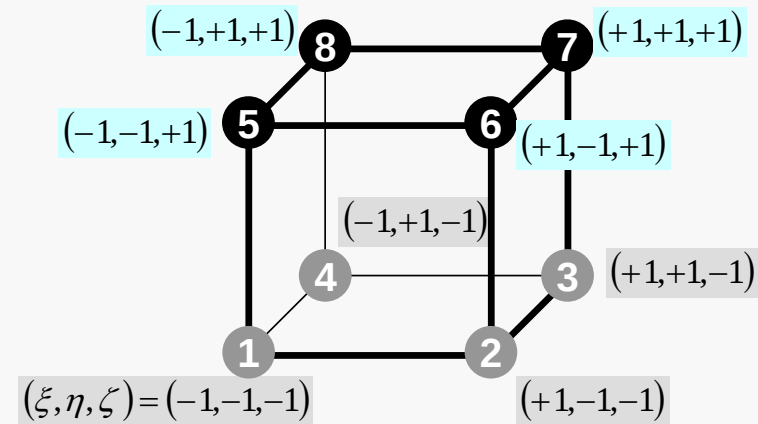
8節点のY座標

QVC= 08th*(X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8);

```
Z1=XYZ[in1-1][2];
Z2=XYZ[in2-1][2];
Z3=XYZ[in3-1][2];
Z4=XYZ[in4-1][2];
Z5=XYZ[in5-1][2];
Z6=XYZ[in6-1][2];
Z7=XYZ[in7-1][2];
Z8=XYZ[in8-1][2];
```

8節点のZ座標

JACOBI (DETJ, PNQ, PNE, PNT, PNQ, PNY, PNZ,
X1, X2, X3, X4, X5, X6, X7, X8,
Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8);



座標値:
節点番号から1引く

係数行列 : MAT_ASS_MAIN (4/6)

```
nodLOCAL[0]= in1;
nodLOCAL[1]= in2;
nodLOCAL[2]= in3;
nodLOCAL[3]= in4;
nodLOCAL[4]= in5;
nodLOCAL[5]= in6;
nodLOCAL[6]= in7;
nodLOCAL[7]= in8;
```

```
X1=XYZ[in1-1][0];
X2=XYZ[in2-1][0];
X3=XYZ[in3-1][0];
X4=XYZ[in4-1][0];
X5=XYZ[in5-1][0];
X6=XYZ[in6-1][0];
X7=XYZ[in7-1][0];
X8=XYZ[in8-1][0];
```

8節点のX座標

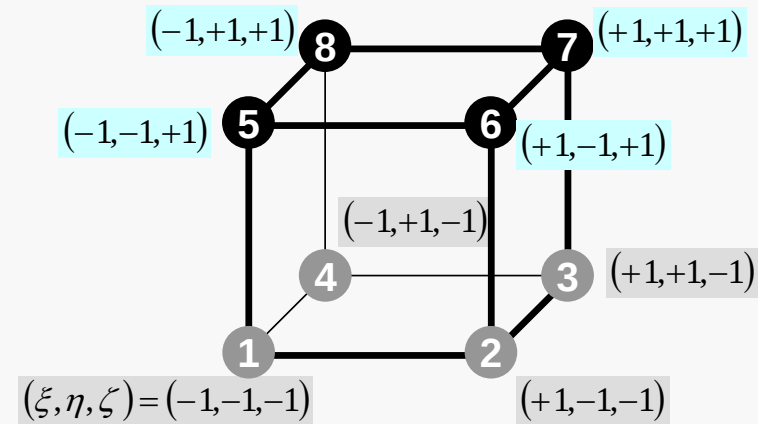
```
Y1=XYZ[in1-1][1];
Y2=XYZ[in2-1][1];
Y3=XYZ[in3-1][1];
Y4=XYZ[in4-1][1];
Y5=XYZ[in5-1][1];
Y6=XYZ[in6-1][1];
Y7=XYZ[in7-1][1];
Y8=XYZ[in8-1][1];
```

8節点のY座標

$QVC = 08th * (X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8);$

```
Z1=XYZ[in1-1][2];
Z2=XYZ[in2-1][2];
Z3=XYZ[in3-1][2];
Z4=XYZ[in4-1][2];
Z5=XYZ[in5-1][2];
Z6=XYZ[in6-1][2];
Z7=XYZ[in7-1][2];
Z8=XYZ[in8-1][2];
```

JACOBI (DETJ, PNQ, PNE, PNT, PNQ, PNY, PNZ,
X1, X2, X3, X4, X5, X6, X7, X8,
Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8);



座標値:
節点番号から1引く

$$\frac{\partial}{\partial x} \left(\lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(\lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left(\lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$

$$\dot{Q}(x, y, z) = QVOL |x_c + y_c|$$

体積当たり発熱量は位置（メッシュの中心の座標 x_c, y_c ）に依存

係数行列 : MAT_ASS_MAIN (4/6)

```
nodLOCAL[0]= in1;
nodLOCAL[1]= in2;
nodLOCAL[2]= in3;
nodLOCAL[3]= in4;
nodLOCAL[4]= in5;
nodLOCAL[5]= in6;
nodLOCAL[6]= in7;
nodLOCAL[7]= in8;
```

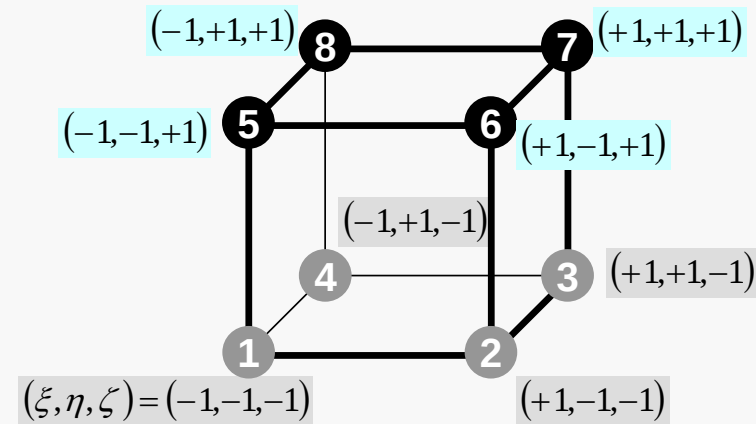
```
X1=XYZ[in1-1][0];
X2=XYZ[in2-1][0];
X3=XYZ[in3-1][0];
X4=XYZ[in4-1][0];
X5=XYZ[in5-1][0];
X6=XYZ[in6-1][0];
X7=XYZ[in7-1][0];
X8=XYZ[in8-1][0];
```

```
Y1=XYZ[in1-1][1];
Y2=XYZ[in2-1][1];
Y3=XYZ[in3-1][1];
Y4=XYZ[in4-1][1];
Y5=XYZ[in5-1][1];
Y6=XYZ[in6-1][1];
Y7=XYZ[in7-1][1];
Y8=XYZ[in8-1][1];
```

QVC= 08th*(X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8);

```
Z1=XYZ[in1-1][2];
Z2=XYZ[in2-1][2];
Z3=XYZ[in3-1][2];
Z4=XYZ[in4-1][2];
Z5=XYZ[in5-1][2];
Z6=XYZ[in6-1][2];
Z7=XYZ[in7-1][2];
Z8=XYZ[in8-1][2];
```

JACOBI (DETJ, PNQ, PNE, PNT, PNX, PNY, PNZ,
X1, X2, X3, X4, X5, X6, X7, X8,
Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8);



座標値:
節点番号から1引く

$$\frac{\partial}{\partial x} \left(\lambda \frac{\partial T}{\partial x} \right) + \frac{\partial}{\partial y} \left(\lambda \frac{\partial T}{\partial y} \right) + \frac{\partial}{\partial z} \left(\lambda \frac{\partial T}{\partial z} \right) + \dot{Q}(x, y, z) = 0$$

$$\dot{Q}(x, y, z) = QVOL |x_c + y_c|$$

$$QVC = |x_c + y_c|$$

系数行列：MAT_ASS_MAIN (4/6)

```
nodLOCAL[0]= in1;
nodLOCAL[1]= in2;
nodLOCAL[2]= in3;
nodLOCAL[3]= in4;
nodLOCAL[4]= in5;
nodLOCAL[5]= in6;
nodLOCAL[6]= in7;
nodLOCAL[7]= in8;
```

```
X1=XYZ[in1-1][0];
X2=XYZ[in2-1][0];
X3=XYZ[in3-1][0];
X4=XYZ[in4-1][0];
X5=XYZ[in5-1][0];
X6=XYZ[in6-1][0];
X7=XYZ[in7-1][0];
X8=XYZ[in8-1][0];
```

```
Y1=XYZ[in1-1][1];
Y2=XYZ[in2-1][1];
Y3=XYZ[in3-1][1];
Y4=XYZ[in4-1][1];
Y5=XYZ[in5-1][1];
Y6=XYZ[in6-1][1];
Y7=XYZ[in7-1][1];
Y8=XYZ[in8-1][1];
```

```
QVC= 08th*(X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8);
```

```
Z1=XYZ[in1-1][2];
Z2=XYZ[in2-1][2];
Z3=XYZ[in3-1][2];
Z4=XYZ[in4-1][2];
Z5=XYZ[in5-1][2];
Z6=XYZ[in6-1][2];
Z7=XYZ[in7-1][2];
Z8=XYZ[in8-1][2];
```

```
JACOBI (DETJ, PNO, PNE, PNT, PNX, PNY, PNZ,
X1, X2, X3, X4, X5, X6, X7, X8,
Y1, Y2, Y3, Y4, Y5, Y6, Y7, Y8, Z1, Z2, Z3, Z4, Z5, Z6, Z7, Z8);
```

係数行列 : MAT_ASS_MAIN (5/6)

```

/**
  CONSTRUCT the GLOBAL MATRIX
  */
for (ie=0; ie<8; ie++) {
  ip=nodLOCAL[ie];

  for (je=0; je<8; je++) {
    jp=nodLOCAL[je];

    kk=-1;
    if ( jp != ip ) {
      iiS=indexLU[ip-1];
      iiE=indexLU[ip];
      for ( k=iiS; k<iiE; k++) {
        if ( itemLU[k] == jp-1 ) {
          kk=k;
          break;
        }
      }
    }
  }
}

```

全体行列の非対角成分

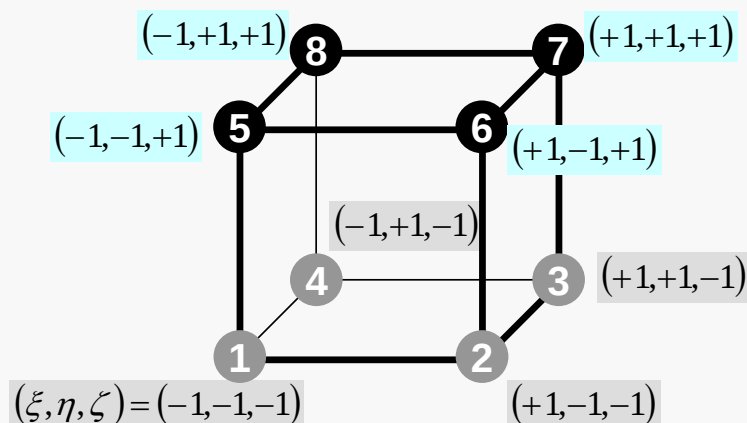
$$A_{ip, jp}$$

kk: におけるアドレス

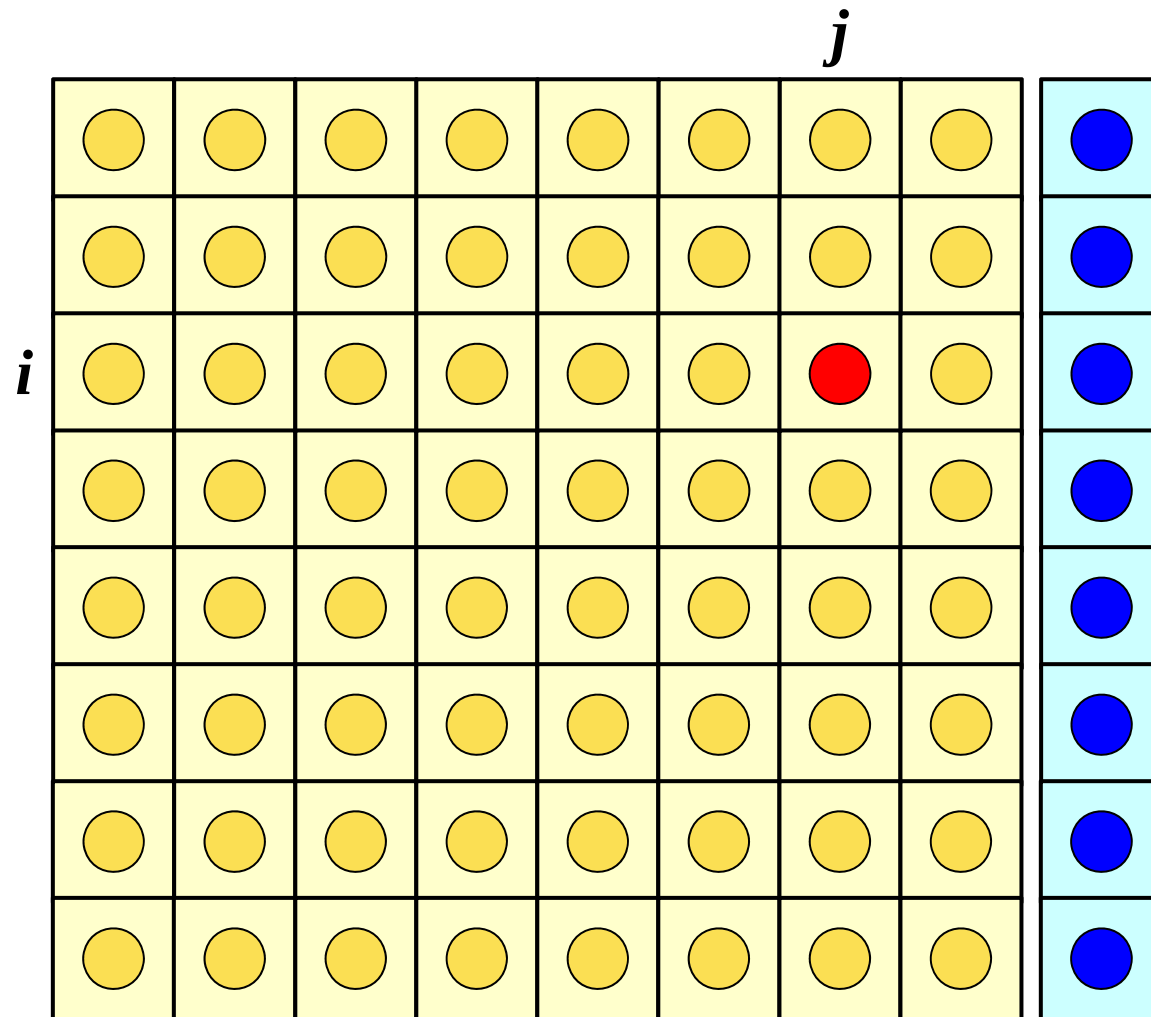
ip= nodLOCAL[ie]

jp= nodLOCAL[je]

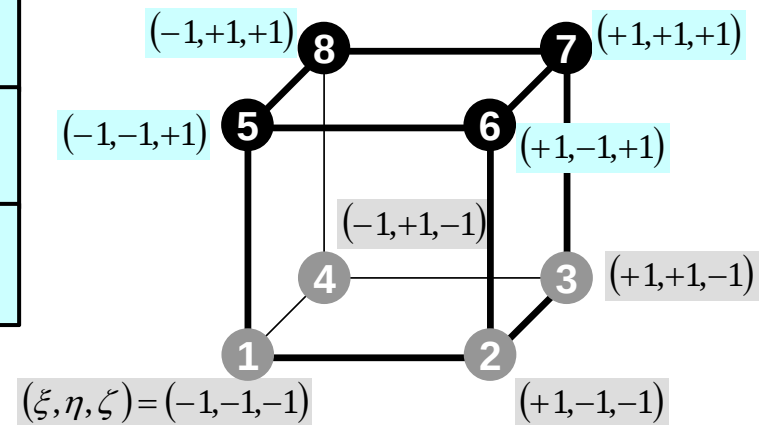
1から始まる節点番号



要素マトリクス : 8×8 行列



$$[k_{ij}] \quad (i, j = 1 \dots 8)$$



係数行列 : MAT_ASS_MAIN (5/6)

```

/**
CONSTRUCT the GLOBAL MATRIX
**/

for (ie=0; ie<8; ie++) {
    ip=nodLOCAL[ie];

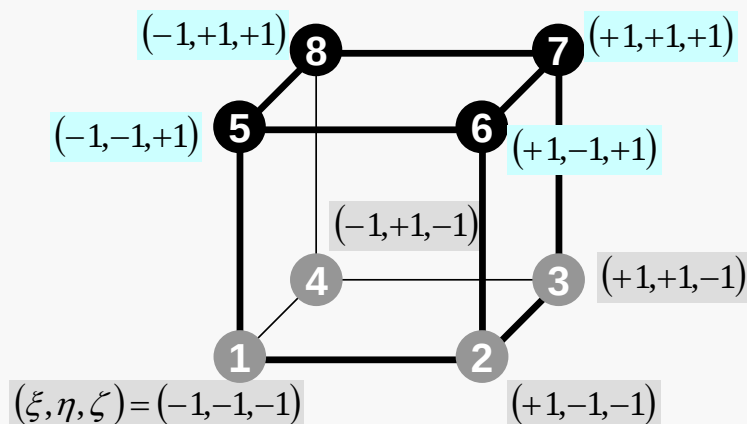
    for (je=0; je<8; je++) {
        jp=nodLOCAL[je];

        kk=-1;
        if ( jp != ip ) {
            iiS=indexLU[ip-1];
            iiE=indexLU[ip];
            for ( k=iiS; k<iiE; k++) {
                if ( itemLU[k] == jp-1 ) {
                    kk=k;
                    break;
                }
            }
        }
    }
}

```

要素マトリクス ($i_e \sim j_e$)
 全体マトリクス ($i_p \sim j_p$) の関係

kk: itemLUにおけるアドレス



系数行列：MAT_ASS_MAIN (6/6)

```

QV0= 0. e0;
COEFij= 0. e0;

for (kpn=0;kpn<2;kpn++) {
  for (jpn=0;jpn<2;jpn++) {
    for (ipn=0;ipn<2;ipn++) {
      coef= fabs(DETJ[ipn][jpn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];

```

```

      PNXi= PNX[ipn][jpn][kpn][ie];
      PNYi= PNY[ipn][jpn][kpn][ie];
      PNZi= PNZ[ipn][jpn][kpn][ie];

```

```

      PNXj= PNX[ipn][jpn][kpn][je];
      PNYj= PNY[ipn][jpn][kpn][je];
      PNZj= PNZ[ipn][jpn][kpn][je];

```

```

      COEFij+= coef*CONDO*(PNXi*PNXj+PNYi*PNYj+PNZi*PNZj);

```

```

      SHi= SHAPE[ipn][jpn][kpn][ie];
      QV0+= SHi * QVOL * coef;

```

```

    }
  }
}

```

```

if (jp==ip) {
  D[ip-1]+= COEFij;
  B[ip-1]+= QV0*QVC;
}

```

```

if (jp != ip) {
  AMAT[kk]+= COEFij;
}
}
}
}
}

```

$$- \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} \det|J| d\xi d\eta d\zeta$$

系数行列：MAT_ASS_MAIN (6/6)

```

QV0= 0. e0;
COEFij= 0. e0;

for (kpn=0;kpn<2;kpn++) {
  for (jpn=0;jpn<2;jpn++) {
    for (ipn=0;ipn<2;ipn++) {
      coef= fabs(DETJ[ipn][jpn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];

```

```

      PNXi= PNX[ipn][jpn][kpn][ie];
      PNYi= PNY[ipn][jpn][kpn][ie];
      PNZi= PNZ[ipn][jpn][kpn][ie];

```

```

      PNXj= PNX[ipn][jpn][kpn][je];
      PNYj= PNY[ipn][jpn][kpn][je];
      PNZj= PNZ[ipn][jpn][kpn][je];

```

```

      COEFij+= coef*CONDO*(PNXi*PNXj+PNYi*PNYj+PNZi*PNZj);

```

```

      SHi= SHAPE[ipn][jpn][kpn][ie];
      QV0+= SHi * QVOL * coef;

```

```

    }
  }
}

if (jp==ip) {
  D[ip-1]+= COEFij;
  B[ip-1]+= QV0*QVC;
}

```

```

if (jp != ip) {
  AMAT[kk]+= COEFij;
}
}
}
}
}

```

$$I = \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta, \zeta) d\xi d\eta d\zeta$$

$$= \sum_{i=1}^L \sum_{j=1}^M \sum_{k=1}^N \boxed{W_i \cdot W_j \cdot W_k} \cdot \boxed{f(\xi_i, \eta_j, \zeta_k)}$$

$$- \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} \det|J| d\xi d\eta d\zeta$$

系数行列：MAT_ASS_MAIN (6/6)

```

QV0= 0. e0;
COEFij= 0. e0;

for (kpn=0;kpn<2;kpn++) {
  for (jpn=0;jpn<2;jpn++) {
    for (ipn=0;ipn<2;ipn++) {
      coef= fabs(DETJ[ipn][jpn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];

```

```

      PNXi= PNX[ipn][jpn][kpn][ie];
      PNYi= PNY[ipn][jpn][kpn][ie];
      PNZi= PNZ[ipn][jpn][kpn][ie];

```

$$\text{coef} = W_i \cdot W_j \cdot W_k \cdot \det|J(\xi_i, \eta_j, \zeta_k)|$$

```

      PNXj= PNX[ipn][jpn][kpn][je];
      PNYj= PNY[ipn][jpn][kpn][je];
      PNZj= PNZ[ipn][jpn][kpn][je];

```

```

      COEFij+= coef*COND0*(PNXi*PNXj+PNYi*PNYj+PNZi*PNZj);

```

```

      SHi= SHAPE[ipn][jpn][kpn][ie];
      QV0+= SHi * QVOL * coef;

```

$$\begin{aligned}
 I &= \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} f(\xi, \eta, \zeta) d\xi d\eta d\zeta \\
 &= \sum_{i=1}^L \sum_{j=1}^M \sum_{k=1}^N W_i \cdot W_j \cdot W_k \cdot f(\xi_i, \eta_j, \zeta_k)
 \end{aligned}$$

```

    }
  }
}

if (jp==ip) {
  D[ip-1]+= COEFij;
  B[ip-1]+= QV0*QVC;
}

if (jp != ip) {
  AMAT[kk]+= COEFij;
}
}
}
}
}

```

$$- \int_{-1}^{+1} \int_{-1}^{+1} \int_{-1}^{+1} \left\{ \lambda \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + \lambda \frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \lambda \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right\} \det|J| d\xi d\eta d\zeta$$

系数行列：MAT_ASS_MAIN (6/6)

```

QV0= 0. e0;
COEFij= 0. e0;

for (kpn=0;kpn<2;kpn++) {
  for (jpn=0;jpn<2;jpn++) {
    for (ipn=0;ipn<2;ipn++) {
      coef= fabs(DETJ[ipn][jpn][kpn])*WEI[ipn]*WEI[jpn]*WEI[kpn];

```

```

      PNXi= PNX[ipn][jpn][kpn][ie];
      PNYi= PNY[ipn][jpn][kpn][ie];
      PNZi= PNZ[ipn][jpn][kpn][ie];

```

```

      PNXj= PNX[ipn][jpn][kpn][je];
      PNYj= PNY[ipn][jpn][kpn][je];
      PNZj= PNZ[ipn][jpn][kpn][je];

```

```

      COEFij+= coef*CONDO*(PNXi*PNXj+PNYi*PNYj+PNZi*PNZj);

```

```

      SHi= SHAPE[ipn][jpn][kpn][ie];
      QV0+= SHi * QVOL * coef;

```

```

    }
  }
}

```

```

if (jp==ip) {
  D[ip-1]+= COEFij;
  B[ip-1]+= QV0*QVC;
}

```

```

if (jp != ip) {
  AMAT[kk]+= COEFij;
}
}
}
}

```

$$[k]^{(e)} \{\phi\}^{(e)} = \{f\}^{(e)}$$

$$[f]^{(e)} = \int_V \dot{Q}[N]^T dV$$

$$\dot{Q}(x, y, z) = QVOL |x_C + y_C|$$

$$QVC = |x_C + y_C|$$

$$QV0 = \int_V QVOL [N]^T dV$$

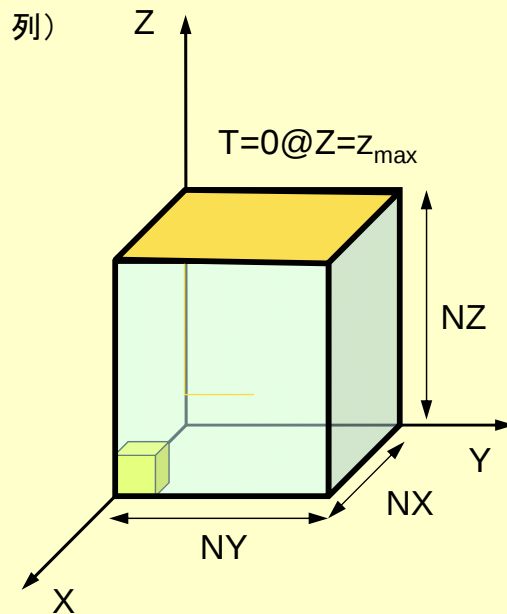
$$[f]^{(e)} = QV0 \cdot QVC$$

MAT_ASS_BC : 全体構成

```
do i= 1, NP    節点ループ
  (ディリクレ) 境界条件を設定する節点をマーク (IWKX)
enddo
```

```
do i= 1, NP    節点ループ
  if (IWKX(i,1).eq.1) then  マークされた節点だったら
    対応する右辺ベクトル (B) の成分, 対角成分 (D) の成分の修正 (行・列)
    do k= index(i-1)+1, index(i)
      対応する非零非対角成分 (AMAT) の成分の修正 (行)
    enddo
  endif
enddo
```

```
do i= 1, NP  節点ループ
  do k= index (i-1)+1, index (i)
    if (IWKX(item (k),1).eq.1) then  対応する非零非対角成分の
                                      節点がマークされていたら
      対応する右辺ベクトル, 非零非対角成分 (AMAT) の成分の修正 (列)
    endif
  enddo
enddo
```



境界条件 : MAT_ASS_BC (1/2)

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
void MAT_ASS_BC()
{
    int i, j, k, in, ib, ib0, icel;
    int in1, in2, in3, in4, in5, in6, in7, in8;
    int iq1, iq2, iq3, iq4, iq5, iq6, iq7, iq8;
    int iS, iE;
    double STRESS, VAL;

    IWKX=(KINT**) allocate_matrix(sizeof(KINT), N, 2);
    for(i=0; i<N; i++) for(j=0; j<2; j++) IWKX[i][j]=0;

    /**
     *Z=Zmax
     */
    for(in=0; in<NP; in++) IWKX[in][0]=0;

    ib0=-1;

    for( ib0=0; ib0<NODGRPtot; ib0++) {
        if( strcmp(NODGRP_NAME[ib0].name, "Zmax") == 0 ) break;
    }

    for( ib=NODGRP_INDEX[ib0]; ib<NODGRP_INDEX[ib0+1]; ib++) {
        in=NODGRP_ITEM[ib];
        IWKX[in-1][0]=1;
    }
}

```

節点グループ名が「Zmax」である
節点in(1から始まる)において:

$$IWKX[in-1][0] = 1$$

とする

境界条件 : MAT_ASS_BC (2/2)

```
for(in=0; in<NP; in++){
  if( IWKX[in][0] == 1 ){
    B[in]= 0. e0;
    D[in]= 1. e0;
    for(k=indexLU[in]; k<indexLU[in+1]; k++){
      AMAT[k]= 0. e0;
    }
  }
}

for(in=0; in<NP; in++){
  for(k=indexLU[in]; k<indexLU[in+1]; k++){
    if (IWKX[itemLU[k]][0] == 1 ) {
      AMAT[k]= 0. e0;
    }
  }
}
}
```

境界条件 : MAT_ASS_BC (2/2)

```

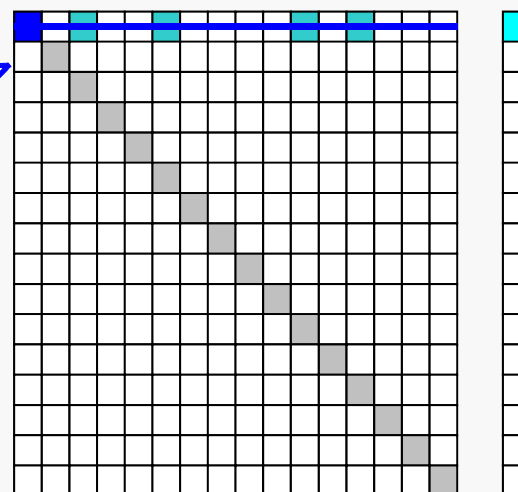
for(in=0; in<NP; in++){
  if( IWKX[in][0] == 1 ){
    B[in]= 0. e0;
    D[in]= 1. e0;
    for(k=indexLU[in]; k<indexLU[in+1]; k++){
      AMAT[k]= 0. e0;
    }
  }
}

for(in=0; in<NP; in++){
  for(k=indexLU[in]; k<indexLU[in+1]; k++){
    if (IWKX[itemLU[k]][0] == 1 ) {
      AMAT[k]= 0. e0;
    }
  }
}

```

IWKX[in-1][0]=1となる節点に対して
対角成分=1, 右辺=0, 非零対角成分=0

ゼロクリア



ここでやっていることも1CPUの時と
全く変わらない

境界条件 : MAT_ASS_BC (2/2)

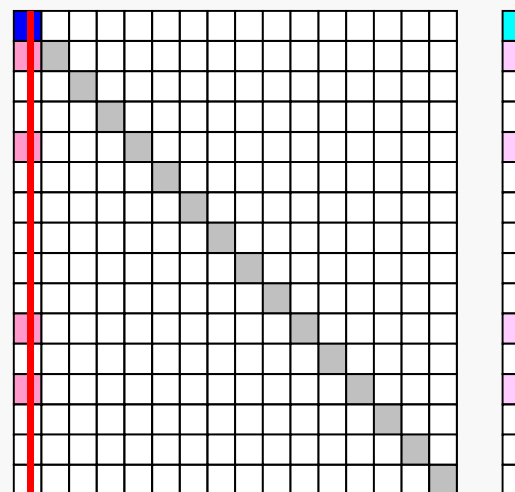
```

for(in=0; in<NP; in++){
  if( IWKX[in][0] == 1 ){
    B[in]= 0. e0;
    D[in]= 1. e0;
    for(k=indexLU[in]; k<indexLU[in+1]; k++){
      AMAT[k]= 0. e0;
    }
  }
}

for(in=0; in<NP; in++){
  for(k=indexLU[in]; k<indexLU[in+1]; k++){
    if( IWKX[itemLU[k]][0] == 1 ){
      AMAT[k]= 0. e0;
    }
  }
}

```

IWKX[in-1][0]=1となる節点を非零非対角成分として有している節点に対して、右辺へ移項、当該非零非対角成分=0



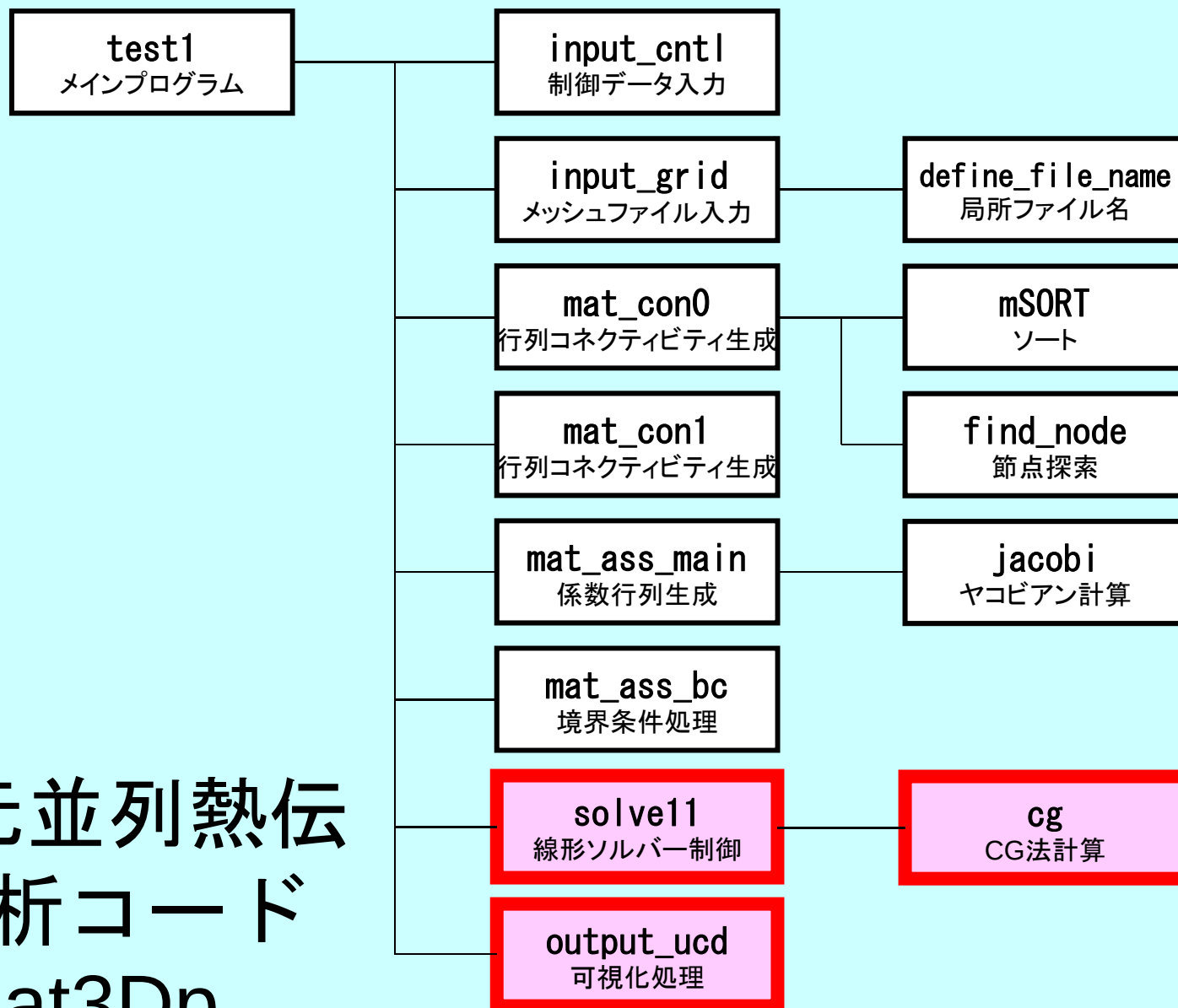
消去, ゼロクリア

ここでやっていることも1CPUの時と全く変わらない

並列有限要素法の処理：プログラム

- 初期化
 - 制御変数読み込み
 - 座標読み込み⇒要素生成 (N:節点数, NE: 要素数)
 - 配列初期化 (全体マトリクス, 要素マトリクス)
 - 要素⇒全体マトリクスマッピング (Index, Item)
- マトリクス生成
 - 要素単位の処理 (do icel= 1, NE)
 - 要素マトリクス計算
 - 全体マトリクスへの重ね合わせ
 - 境界条件の処理
- 連立一次方程式
 - 共役勾配法 (CG)

三次元並列熱伝 導解析コード heat3Dp



全体処理

```
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"
extern void PFEM_INIT(int, char**);
extern void INPUT_CNTL();
extern void INPUT_GRID();
extern void MAT_CONO();
extern void MAT_CON1();
extern void MAT_ASS_MAIN();
extern void MAT_ASS_BC();
extern void SOLVE11();
extern void OUTPUT_UCD();
extern void PFEM_FINALIZE();
int main(int argc, char* argv[])
{
    double START_TIME, END_TIME;

    PFEM_INIT(argc, argv);

    INPUT_CNTL();
    INPUT_GRID();

    MAT_CONO();
    MAT_CON1();

    MAT_ASS_MAIN();
    MAT_ASS_BC();

    SOLVE11();

    OUTPUT_UCD();
    PFEM_FINALIZE();
}
```

SOLVE11

```

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "pfem_util.h"
#include "allocate.h"
extern FILE *fp_log;
extern void CG();
void SOLVE11()
{
    int i, j, k, ii, L;

    int ERROR, ICFLAG=0;
    CHAR_LENGTH BUF;

    /**
    +-----+
    | PARAMETERS |
    +-----+
    **/
    ITER      = pfemIarray[0];      CG法の最大反復回数
    RESID     = pfemRarray[0];      CG法の収束判定値

    /**
    +-----+
    | ITERATIVE solver |
    +-----+
    **/
    CG ( N, NP, NPLU, D, AMAT, indexLU, itemLU,
        B, X, RESID, ITER, &ERROR, my_rank,
        NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_ITEM,
        EXPORT_INDEX, EXPORT_ITEM);}

```

前処理付き共役勾配法

Preconditioned Conjugate Gradient Method (CG)

```

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

```

前処理: 対角スケーリング

対角スケーリング, 点ヤコビ前処理

- 前処理行列として, もとの行列の対角成分のみを取り出した行列を前処理行列 $[M]$ とする。
 - 対角スケーリング, 点ヤコビ (point-Jacobi) 前処理

$$[M] = \begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

- $\text{solve } [M]z^{(i-1)} = r^{(i-1)}$ という場合に逆行列を簡単に求めることができる。

CG法 (1/6)

```

#include <stdio.h>
#include <math.h>
#include "mpi.h"
#include "precision.h"
#include "allocate.h"

extern FILE *fp_log;

extern void SOLVER_SEND_RECV ();

/**
CG solves the linear system  $Ax = b$  using the Conjugate Gradient
iterative method with the following preconditioners
***/

void CG (
    KINT N, KINT NP, KINT NPLU, KREAL D[],
    KREAL AMAT[], KINT indexLU[], KINT itemLU[],
    KREAL B[], KREAL X[], KREAL RESID, KINT ITER, KINT *ERROR, int my_rank,
    int NEIBPETOT, int NEIBPE[],
    int IMPORT_INDEX[], int IMPORT_ITEM[],
    int EXPORT_INDEX[], int EXPORT_ITEM[])
{
    int i, j, k;
    int ieL, isL, ieU, isU;
    double WVAL;
    double BNRM20, BNRM2, DNRM20, DNRM2;
    double S1_TIME, E1_TIME;
    double ALPHA, BETA;
    double C1, C10, RHO, RH00, RH01;
    int iterPRE;
    KREAL *WS, *WR;          送信バッファ, 受信バッファ
    KREAL **WW;

    KINT R=0, Z=1, Q=1, P=2, DD=3;
    KINT MAXIT;
    KREAL TOL;

    double COMptime, COMMtime, R1;
    double START_TIME, END_TIME;

```

CG法 (2/6)

```

ERROR= 0;

WW=(KREAL**) allocate_matrix(sizeof(KREAL), 4, NP);
WS=(KREAL* ) allocate_vector(sizeof(KREAL),  NP);
WR=(KREAL* ) allocate_vector(sizeof(KREAL),  NP);

MAXIT  = ITER;
TOL    = RESID;

for (i=0; i<NP; i++) X [i]=0.0;
for (i=0; i<NP; i++) for (j=0; j<4; j++) WW[j][i]=0.0;
for (i=0; i<NP; i++) WS[i]=0.0;
for (i=0; i<NP; i++) WR[i]=0.0;

/**
+-----+
| {r0}= {b} - [A] {xini} |
+-----+
**/

SOLVER_SEND_RECV
( NP, NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_ITEM,
  EXPORT_INDEX, EXPORT_ITEM, WS, WR, X , my_rank);

for (j=0; j<N; j++) {
  WW[DD][j]= 1.0/D[j];
  WVAL= B[j] - D[j]*X[j];

  for ( k=indexLU[j];k<indexLU[j+1];k++) {
    i= itemLU[k];
    WVAL+= -AMAT[k]*X[i];
  }
  WW[R][j]= WVAL;
}

BNRM20= 0. e0;
for (i=0; i<N; i++) {
  BNRM20+= B[i]*B[i];}

MPI_Allreduce (&BNRM20, &BNRM2, 1, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

```

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

SOLVER_SEND_RECV (1/2)

```

#include <stdio.h>
#include <math.h>
#include "mpi.h"
#include "precision.h"
#include "allocate.h"
static MPI_Status *sta1,*sta2;
static MPI_Request *req1,*req2;
static KINT NFLAG=0;
extern FILE *fp_log;
void SOLVER_SEND_RECV( int N, int NEIBPETOT,
    int NEIBPE[], int IMPORT_INDEX[], int IMPORT_ITEM[],
    int EXPORT_INDEX[], int EXPORT_ITEM[],
    KREAL WS[], KREAL WR[], KREAL X[], int my_rank)
{
    int ii,k,neib,istart,inum;
/**
    INIT.
***/
    if( NFLAG == 0 ){
        sta1=(MPI_Status*)allocate_vector(sizeof(MPI_Status),NEIBPETOT);
        sta2=(MPI_Status*)allocate_vector(sizeof(MPI_Status),NEIBPETOT);
        req1=(MPI_Request*)allocate_vector(sizeof(MPI_Request),NEIBPETOT);
        req2=(MPI_Request*)allocate_vector(sizeof(MPI_Request),NEIBPETOT);
        NFLAG=1;}

/**
    SEND
***/
    for( neib=1;neib<=NEIBPETOT;neib++){
        istart=EXPORT_INDEX[neib-1];
        inum =EXPORT_INDEX[neib]-istart;
        for( k=istart;k<istart+inum;k++){
            ii= EXPORT_ITEM[k];
            WS[k]= X[ii-1];
        }

        MPI_Isend(&WS[istart],inum,MPI_DOUBLE,
            NEIBPE[neib-1],0,MPI_COMM_WORLD,&req1[neib-1]);
    }

```

SOLVER_SEND_RECV (2/2)

```
/**
  RECEIVE
  */
for( neib=1;neib<=NEIBPETOT;neib++) {
  istart=IMPORT_INDEX[neib-1];
  inum =IMPORT_INDEX[neib]-istart;
  MPI_Irecv (&WR[istart], inum, MPI_DOUBLE,
             NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req2[neib-1]);
}

MPI_Waitall (NEIBPETOT, req2, sta2);

for( neib=1;neib<=NEIBPETOT;neib++) {
  istart=IMPORT_INDEX[neib-1];
  inum =IMPORT_INDEX[neib]-istart;
  for( k=istart;k<istart+inum;k++) {
    ii = IMPORT_ITEM[k];
    X[ii-1]= WR[k];
  }
}

MPI_Waitall (NEIBPETOT, req1, sta1);
}
```

CG法 (3/6)

```

for( ITER=1; ITER<= MAXIT; ITER++) {

  /**
  | {z}= [Minv] {r} |
  |-----|
  */
  for(i=0; i<N; i++) {
    WW[Z][i]= WW[DD][i]*WW[R][i];
  }

  /**
  | {RHO}= {r} {z} |
  |-----|
  */
  RHOO= 0. e0;
  for(i=0; i<N; i++) {
    RHOO+= WW[R][i]*WW[Z][i];
  }
  MPI_Allreduce (&RHOO, &RHO, 1, MPI_DOUBLE, MPI_SUM,
    MPI_COMM_WORLD);

  /**
  | {p} = {z} if      ITER=1
  | BETA= RHO / RHO1 otherwise
  |-----|
  */
  if( ITER == 1 ) {
    for(i=0; i<N; i++) {
      WW[P][i]=WW[Z][i];
    }
  } else {
    BETA= RHO / RHO1;
    for(i=0; i<N; i++) {
      WW[P][i]=WW[Z][i] + BETA*WW[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

CG法 (4/6)

```

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

SOLVER_SEND_RECV
( NP, NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_ITEM,
  EXPORT_INDEX, EXPORT_ITEM, WS, WR, WW[P], my_rank);

for( j=0; j<N; j++) {
  WVAL= D[j] * WW[P][j];
  for(k=indexLU[j]; k<indexLU[j+1]; k++) {
    i=itemLU[k];
    WVAL+= AMAT[k] * WW[P][i];
  }
  WW[Q][j]=WVAL;
}

/**
| ALPHA= RHO / {p} {q} |
**/

C10= 0. e0;
for(i=0; i<N; i++) {
  C10+=WW[P][i]*WW[Q][i];
}

MPI_Allreduce (&C10, &C1, 1, MPI_DOUBLE, MPI_SUM,
               MPI_COMM_WORLD);

ALPHA= RHO / C1;

```

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

CG法 (5/6)

```

/**
+-----+
+ {x}= {x} + ALPHA*{p}
+ {r}= {r} - ALPHA*{q}
+-----+
**/
for(i=0;i<N;i++){
  X[i] += ALPHA *WW[P][i];
  WW[R][i] += -ALPHA *WW[Q][i];
}

DNRM2= 0. e0;
for(i=0;i<N;i++){
  DNRM2+=WW[R][i]*WW[R][i];
}
MPI_Allreduce (&DNRM2,&DNRM2, 1, MPI_DOUBLE,MPI_SUM,
               MPI_COMM_WORLD);

RESID= sqrt(DNRM2/BNRM2);

if ( RESID <= TOL ) break;
if ( ITER == MAXIT ) *ERROR= -300;

RHO1 = RHO ;
}

SOLVER_SEND_RECV
( NP, NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_ITEM,
  EXPORT_INDEX, EXPORT_ITEM, WS, WR, X, my_rank);

free ( (KREAL**)WW);
deallocate_vector ( (KREAL**)WR);
deallocate_vector ( (KREAL**)WS);
}

```

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

CG法 (6/6)

```

/**
+-----+
| {x} = {x} + ALPHA*{p} |
| {r} = {r} - ALPHA*{q} |
+-----+
**/
for (i=0; i<N; i++) {
    X [i] += ALPHA *WW[P][i];
    WW[R][i] += -ALPHA *WW[Q][i];
}

DNRM2= 0. e0;
for (i=0; i<N; i++) {
    DNRM2+=WW[R][i]*WW[R][i];
}
MPI_Allreduce (&DNRM2, &DNRM2, 1, MPI_DOUBLE, MPI_SUM,
               MPI_COMM_WORLD);

RESID= sqrt(DNRM2/BNRM2);

if ( RESID <= TOL ) break;
if ( ITER == MAXIT ) *ERROR= -300;

RHO1 = RHO ;
}

SOLVER_SEND_RECV
( NP, NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_ITEM,
  EXPORT_INDEX, EXPORT_ITEM, WS, WR, X, my_rank);

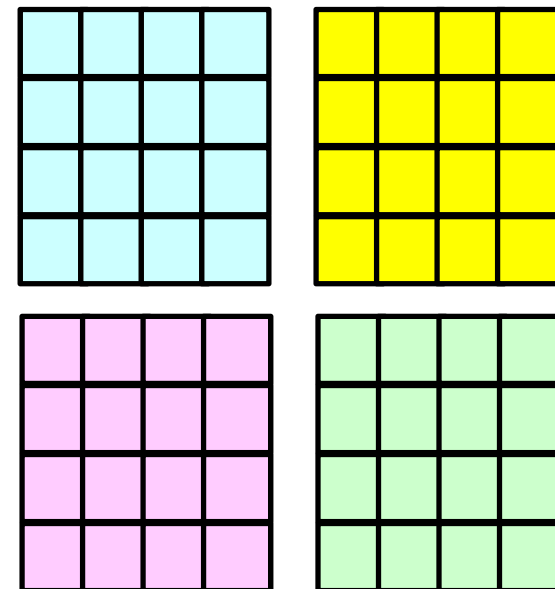
free ( (KREAL**)WW);
deallocate_vector ( (KREAL**)WR);
deallocate_vector ( (KREAL**)WS);
}

```

外点に最新の値（温度）を入れる

OUTPUT_UCD

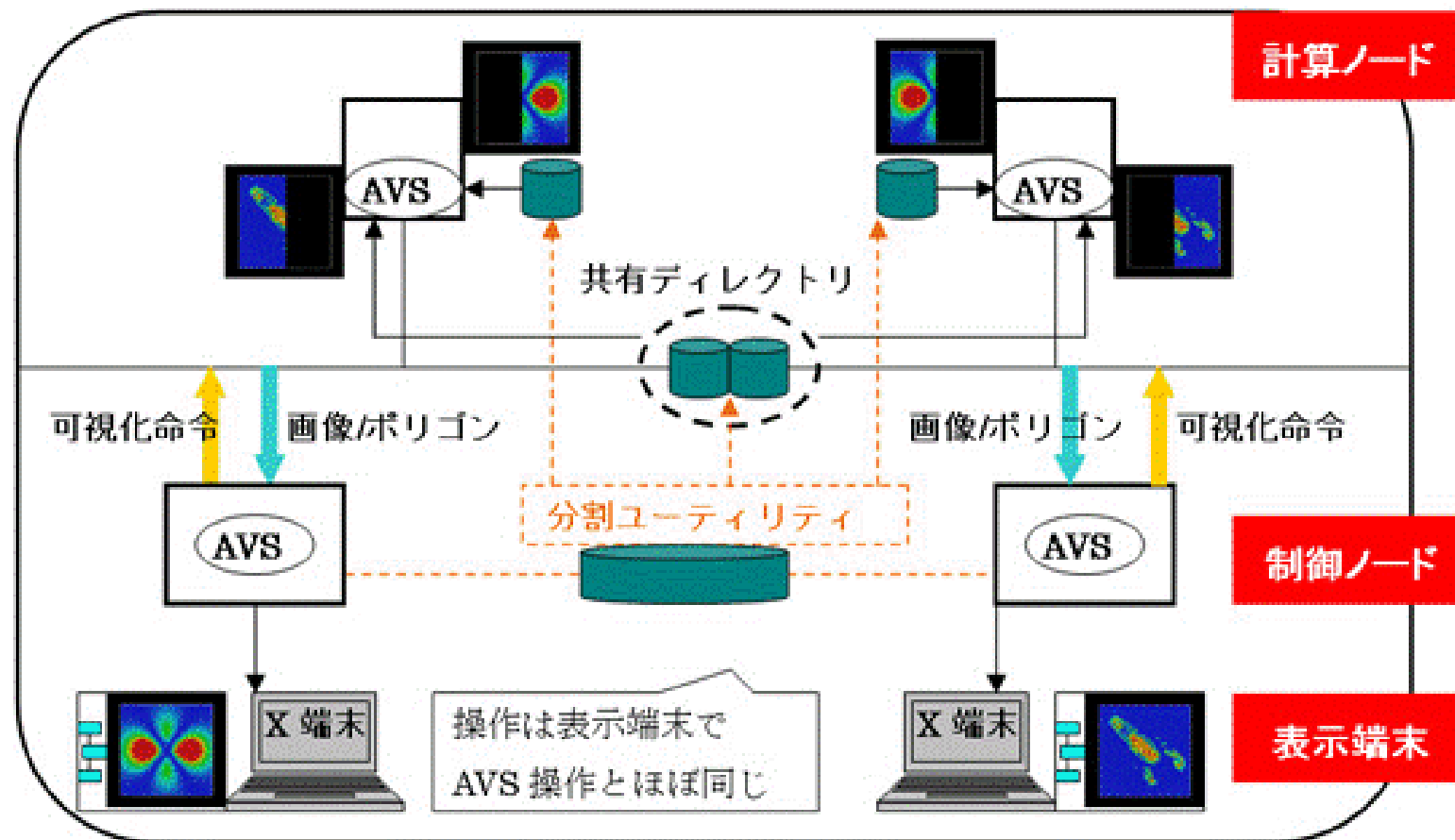
- 各領域からintELEM_listに所属する要素の情報を集める
 - intELEM_list: 各節点の属する領域番号のうち最も若い番号の領域に所属する・・・と見なす
- MPI_Allgathervを使って以下の情報を一箇所に集める
 - 節点: 節点座標, 温度
 - 要素: 要素コネクティビティ(要素を構成する節点)
- 節点の情報は一部重複
- 問題規模が大きくなると困難
 - あまり賢いやり方ではない
 - 並列可視化



AVS/Express PCE Parallel Cluster Edition

- <http://www.cybernet.co.jp/avs/products/pce/>
- AVS/Express PCEでは、クラスタ化された複数のLinuxマシンで、各計算ノードが持つ部分領域のみを可視化し、最終的な可視化結果のみ制御ノード上で表示するという構成になっている。
- 並列計算の結果、出力される大規模データを可視化する場合でも、高い精度を保ったまま、可視化処理を実現することが可能。
- 並列計算機上で対話処理可能
 - 最新版ではWindowsより制御可能
 - T2K東大で利用可能

AVS/Express PCE Parallel Cluster Edition



計算例

- $128 \times 128 \times 128$ メッシュ
- 16～128コア
 - RCB, pMETIS(96コアの場合のみ)
- Solver計算時間

core #	Fortran sec. (speed-up)	C sec. (speed-up)
16	10.8 (16.0)	11.3 (16.0)
32	4.93 (35.0)	7.59 (23.8)
64	2.52 (68.5)	2.31 (78.3)
96	1.44 (120.)	1.35 (134.)
128	1.24 (139.)	1.21 (149.)

課題P1(1/3)

- 自分で問題設定を行い, 「sol」の挙動を分析してみよ
- 例
 - Strong Scaling
 - 問題サイズを固定, PE数を変化させて時間(全体, 各部分)を測定。
 - Weak Scaling
 - PEあたりの問題サイズを固定, 1反復あたりの計算時間を求める。
 - 考慮すべき項目
 - 問題サイズ
 - 領域分割手法(RCB, K-METIS, P-METIS, 1D~3D)の影響。
 - メッシュ生成, 領域分割におけるFX10の性能低い
 - 128³くらいが限界(領域分割に15分以上かかる)
- 「*.inp」の出力に時間がかかる場合がある。
 - OUTPUT_UCDの呼び出しのコメントアウト
 - src, part

課題P1(2/3)

- 領域間通信(solver_SR)の性能改善ができないかどうか考えてみよ。
 - Recv. Bufferへのコピーを効率的に実施できないか？

```
for (neib=0; neib<NeibPETot; neib++){
    tag= 0;
    iS_i= import_index[neib];
    iE_i= import_index[neib+1];
    BUFlength_i= iE_i - iS_i

    ierr= MPI_Irecv
        (&RecvBuf[iS_i], BUFlength_i, MPI_DOUBLE, NeibPE[neib], 0,
         MPI_COMM_WORLD, &ReqRecv[neib])
}

MPI_Waitall(NeibPETot, ReqRecv, StatRecv);

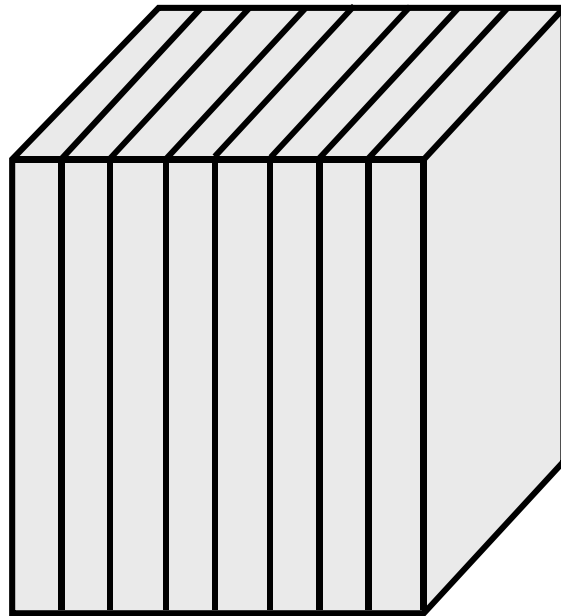
for (neib=0; neib<NeibPETot;neib++){
    for (k=import_index[neib];k<import_index[neib+1];k++){
        kk= import_item[k];
        VAL[kk]= RecvBuf[k];
    }
}
```

課題P1(3/3)

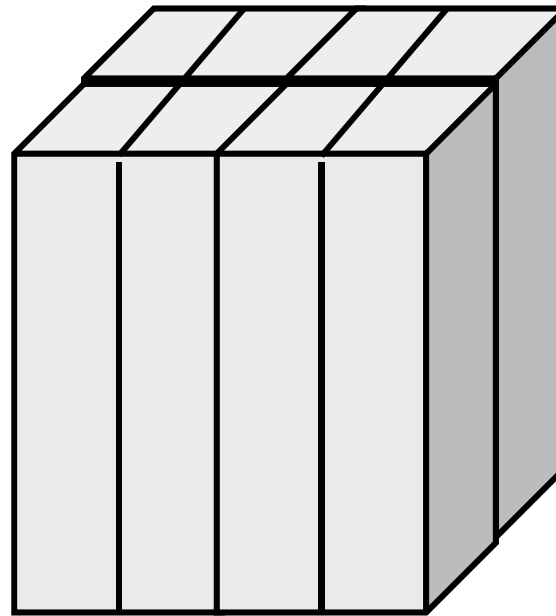
- 提出期限: 2012年10月12日(金)17:00(メール添付可)
- 提出物(レポート): 最高級仕様
 - 表紙: 氏名, 学籍番号, 課題番号を明記
 - A4 10枚以内(図表含む)でまとめること
 - プログラムリスト
 - 結果出力リスト(最小限にとどめること)

1D～3D分割

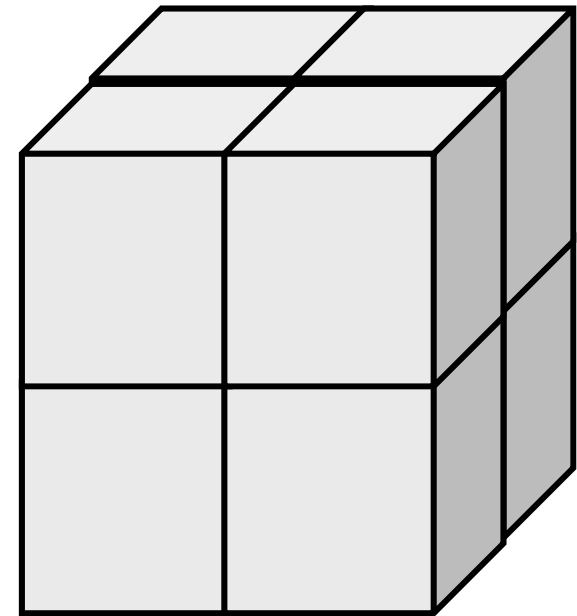
どの方法が良いか考えて見よ



1D型



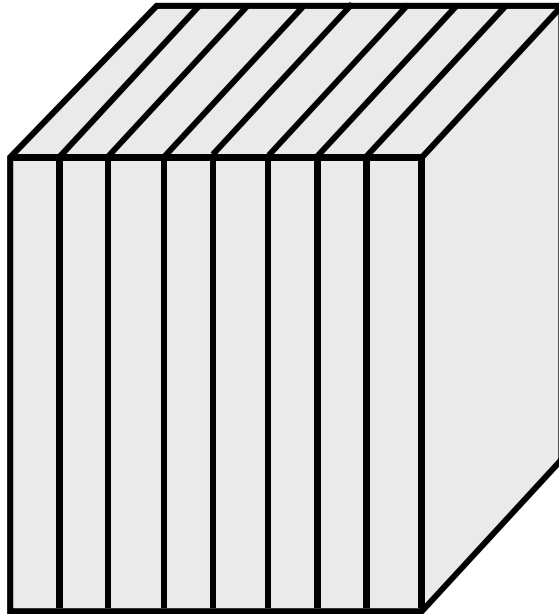
2D型



3D型

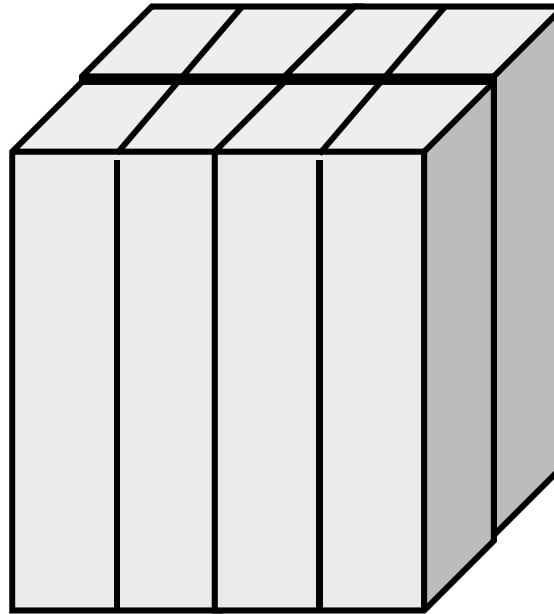
1D～3D分割

通信量の総和(各辺4N, 8領域とする)



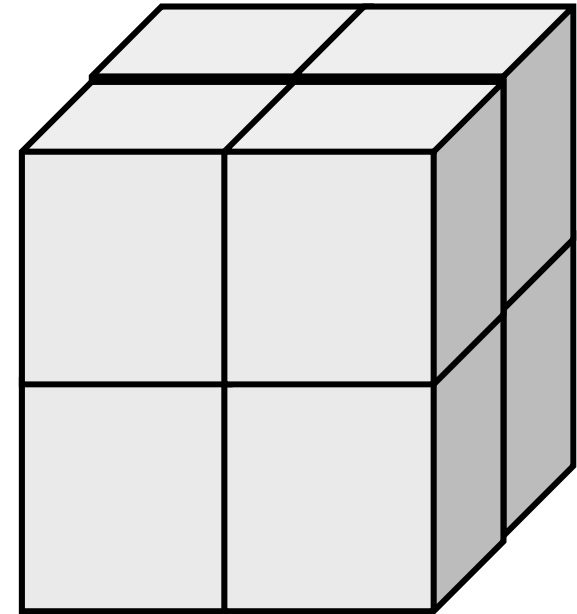
1D型

$$16 N^2 \times 7 = 112 N^2$$



2D型

$$16 N^2 \times 4 = 64 N^2$$

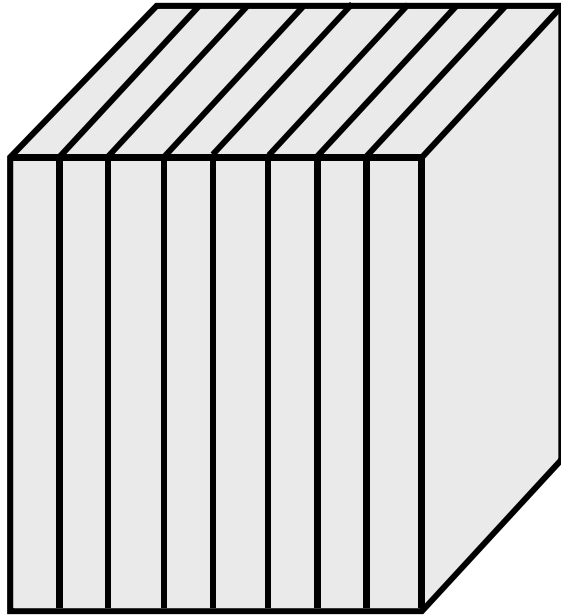


3D型

$$16 N^2 \times 3 = 48 N^2$$

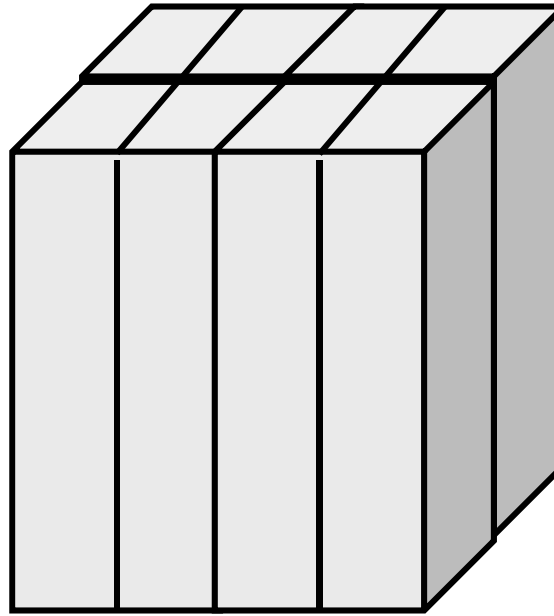
1D~3D分割

inp_rcb



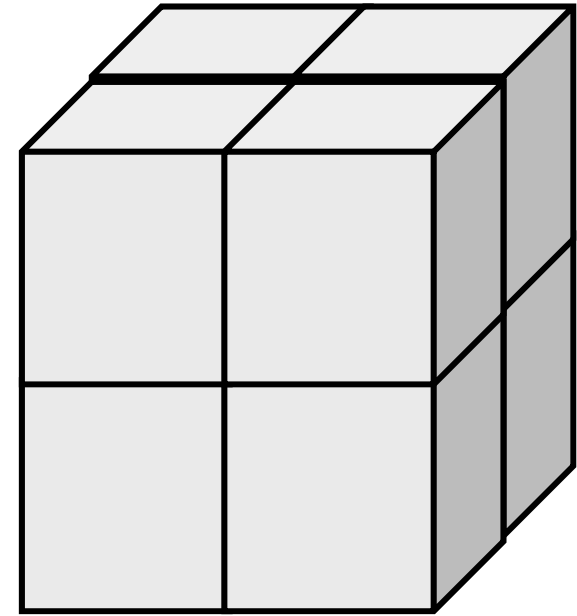
1D型

```
cube.0  
1  
3  
aaa  
1  
1  
1
```



2D型

```
cube.0  
1  
3  
aaa  
1  
2  
1
```



3D型

```
cube.0  
1  
3  
aaa  
1  
2  
3
```