

# **3D Parallel FEM (II)**

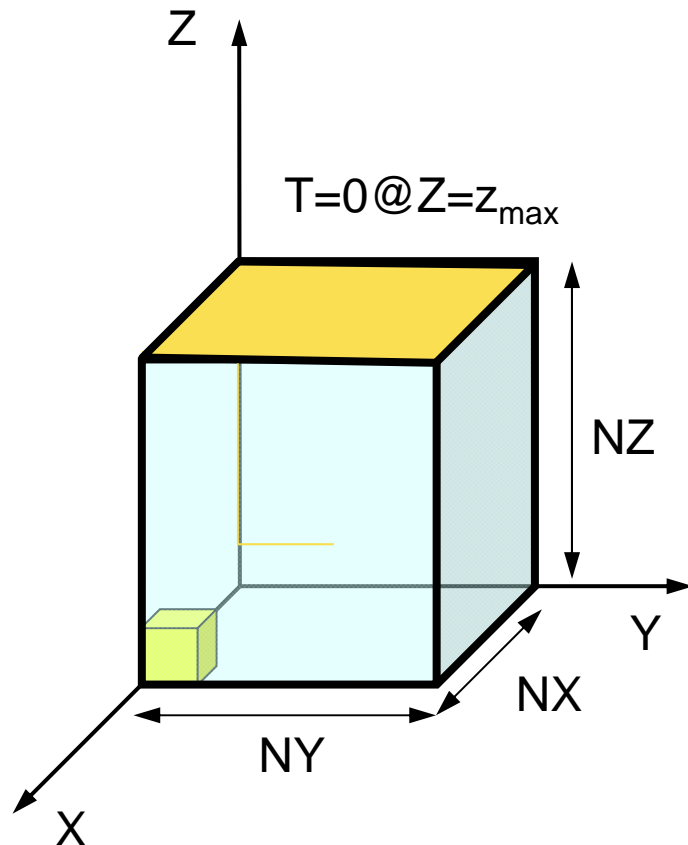
## **C**

Kengo Nakajima

Programming for Parallel Computing (616-2057)  
Seminar on Advanced Computing (616-4009)

# 3D Steady-State Heat Conduction

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



- Heat Generation
- Uniform thermal conductivity  $\lambda$
- HEX meshes
  - 1x1x1 cubes
  - NX, NY, NZ cubes in each direction
- Boundary Conditions
  - $T=0@Z=z_{\max}$
- Heat Gen. Rate is a function of location (cell center:  $x_c, y_c$ )
  - $\dot{Q}(x, y, z) = QVOL|x_c + y_c|$

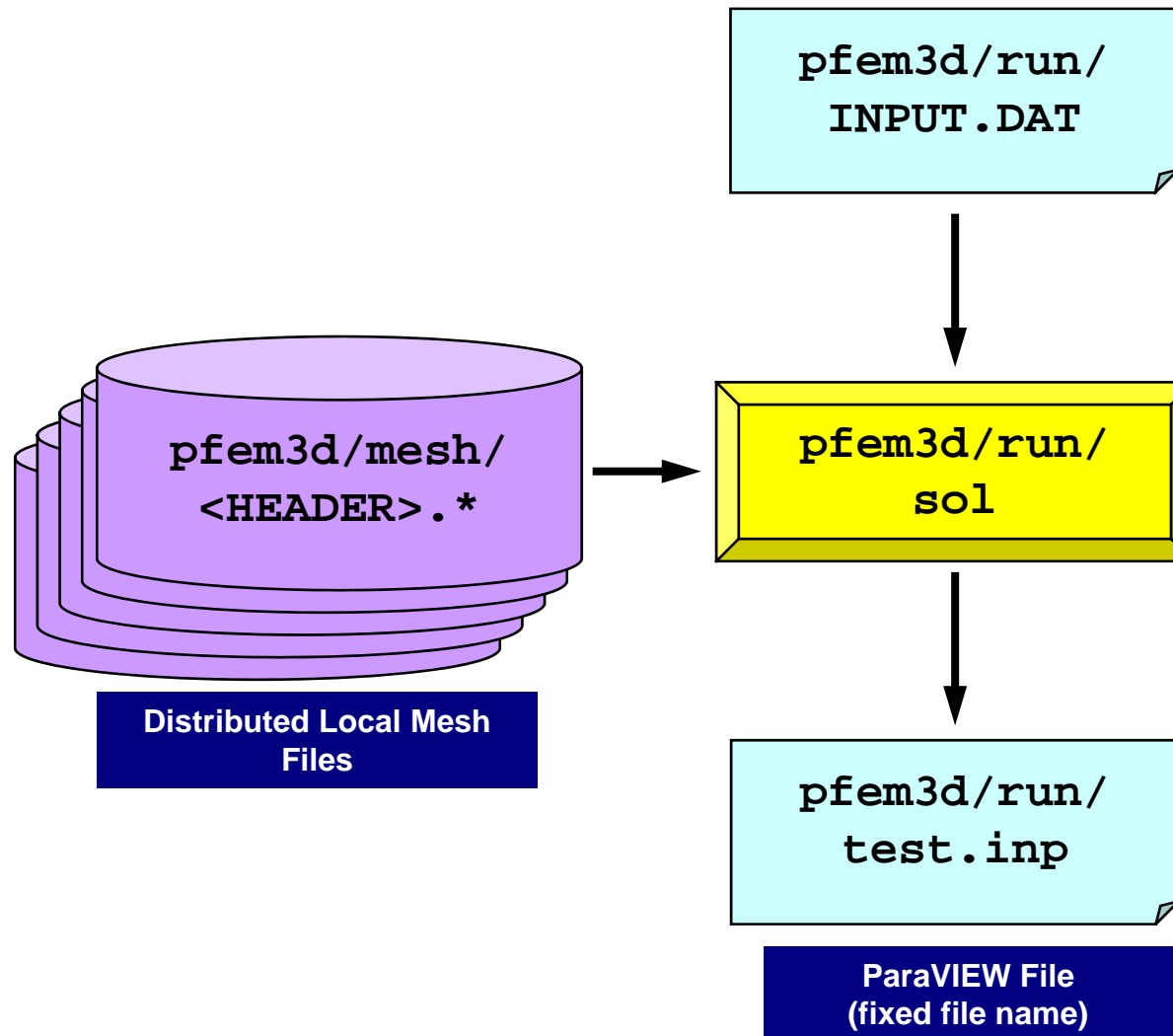
# Finite-Element Procedures

- Governing Equations
- Galerkin Method: Weak Form
- Element-by-Element Integration
  - Element Matrix
- Global Matrix
- Boundary Conditions
- Linear Solver

# FEM Procedures: Program

- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation to global matrix
  - Boundary Conditions
- Linear Solver
  - Conjugate Gradient Method

# Procedures for Parallel FEM



# Control File: INPUT.DAT

## INPUT.DAT

```

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

```

- **HEADER :** HEADER of distributed mesh files "HEADER".my\_rank
- **ITER :** Max. Iterations for CG
- **COND :** Thermal Conductivity
- **QVOL :** Heat Generation Rate
- **RESID :** Criteria for Convergence of CG

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

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

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

```
#!/bin/sh
#PJM -L "node=1"           Number of Nodes
#PJM -L "elapse=00:10:00"  Computation Time
#PJM -L "rscgrp=lecture"   Name of "QUEUE"
#PJM -g "gt71"            Group Name (Wallet)
#PJM -j
#PJM -o "hello.lst"       Standard Output
#PJM --mpi "proc=4"       MPI Process #

mpiexec ./sol             Execs
```

8分割

"node=1"

"proc=8"

16分割

"node=1"

"proc=16"

32分割

"node=2"

"proc=32"

64分割

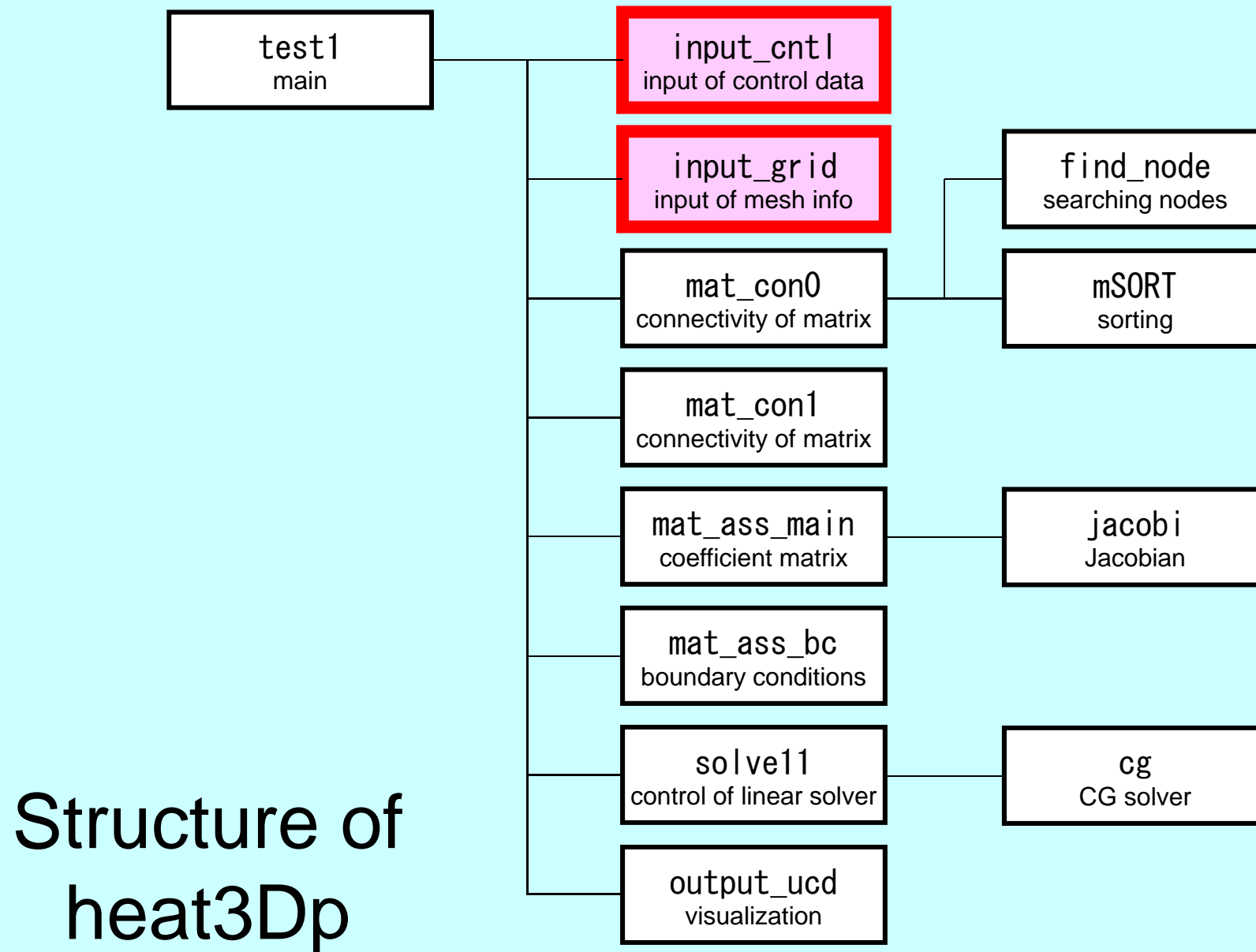
"node=4"

"proc=64"

192分割

"node=12"

"proc=192"





# Main Part

```
#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 Variables: pfem\_util.h (1/4)

Name	Type	Size	I/O	Definition
<b>fname</b>	C	[80]	I	Name of mesh file
<b>N, NP</b>	I		I	# Node (N: Internal, NP: Internal + External)
<b>ICELTOT</b>	I		I	# Element
<b>NODGRPtot</b>	I		I	# Node Group
<b>XYZ</b>	R	[NP][3]	I	Node Coordinates
<b>ICELNOD</b>	I	[ICELTOT][8]	I	Element Connectivity
<b>NODGRP_INDEX</b>	I	[NODGRPtot+1]	I	# Node in each Node Group
<b>NODGRP_ITEM</b>	I	[NODGRP_INDEX[NODGRPtot+1]]	I	Node ID in each Node Group
<b>NODGRP_NAME</b>	C80	[NODGRP_INDEX[NODGRPtot+1]]	I	Name of NodeGroup
<b>NLU</b>	I		O	# Non-Zero Off-Diagonals at each node
<b>NPLU</b>	I		O	# Non-Zero Off-Diagonals
<b>D</b>	R	[NP]	O	Diagonal Block of Global Matrix
<b>B, X</b>	R	[NP]	O	RHS, Unknown Vector

# Global Variables: pfem\_util.h (2/4)

Name	Type	Size	I/O	Definition
<b>AMAT</b>	R	[NPLU]	O	Non-Zero Off-Diagonal Components of Global Matrix
<b>indexLU</b>	I	[NP+1]	O	# Non-Zero Off-Diagonal Components
<b>itemLU</b>	I	[NPLU]	O	Column ID of Non-Zero Off-Diagonal Components
<b>INLU</b>	I	[NP]	O	Number of Non-Zero Off-Diagonal Components at Each Node
<b>IALU</b>	I	[NP][NLU]	O	Column ID of Non-Zero Off-Diagonal Components at Each Node
<b>IWKX</b>	I	[NP][2]	O	Work Arrays
<b>ITER, ITERactual</b>	I		I	Number of CG Iterations (MAX, Actual)
<b>RESID</b>	R		I	Convergence Criteria (fixed as 1.e-8)
<b>pfemIarray</b>	I	[100]	O	Integer Parameter Array
<b>pfemRarray</b>	R	[100]	O	Real Parameter Array

# Global Variables: pfem\_util.h (3/4)

Name	Type	Size	I/O	Definition
<b>o8th</b>	R		I	= 0.125
<b>PNQ, PNE, PNT</b>	R	[2][2][8]	O	$\frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta} (i=1 \sim 8)$ at each Gaussian Quad. Point
<b>POS, WEI</b>	R	[2]	O	Coordinates, Weighting Factor at each Gaussian Quad. Point
<b>NCOL1, NCOL2</b>	I	[100]	O	Work arrays for sorting
<b>SHAPE</b>	R	[2][2][2][8]	O	$N_i (i=1 \sim 8)$ at each Gaussian Quad Point
<b>PNX, PNY, PNZ</b>	R	[2][2][2][8]	O	$\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z} (i=1 \sim 8)$ at each Gaussian Quad. Point
<b>DETJ</b>	R	[2][2][2]	O	Determinant of Jacobian Matrix at each Gaussian Quad. Point
<b>COND, QVOL</b>	R		I	Thermal Conductivity, Heat Generation Rate

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

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

# Global Variables: pfem\_util.h (4/4)

Name	Type	Size	I/O	Definition
<b>PETOT</b>	I		O	Number of PE's
<b>my_rank</b>	I		O	Process ID of MPI
<b>errno</b>	I		O	Error Flag
<b>NEIBPETOT</b>	I		I	Number of Neighbors
<b>NEIBPE</b>	I	[NEIBPETOT]	I	ID of Neighbor
<b>IMPORT_INDEX</b> <b>EXPORT_INEDX</b>	I	[NEIBPETOT+1]	I	Size of Import/Export Arrays for Communication Table
<b>IMPORT_ITEM</b>	I	[NPimport]	I	Receiving Table (External Points) NPimport=IMPORT_INDEX[NEIBPETOT+1])
<b>EXPORT_ITEM</b>	I	[NPexport]	I	Sending Table (Boundary Points) NPexport=EXPORT_INDEX[NEIBPETOT+1])
<b>ICELTOT_INT</b>	<b>I</b>		<b>I</b>	<b>Number of Local Elements</b>
<b>intELEM_list</b>	<b>I</b>	<b>[ICELTOT_INT]</b>	<b>I</b>	<b>List of Local Elements</b>

# Start/End: 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);
    }
}
```

# Reading Control File: 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;
}
```

# Reading Meshes: 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);}
    }
}

```



# Name of Distributed Local Mesh File: DEFINE\_FILE\_NAME HEADER + Rank ID

```
#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

```

#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 );
}

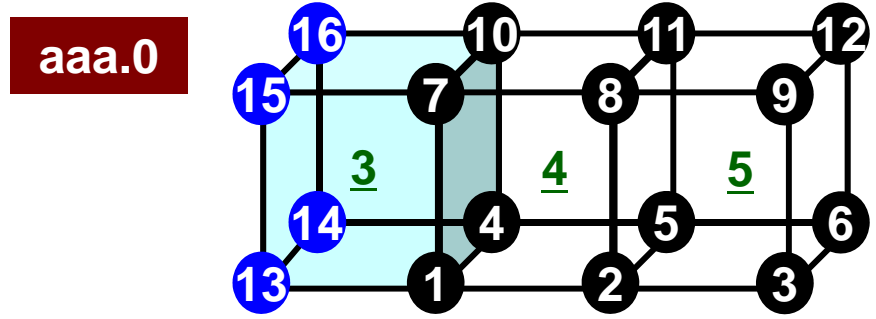
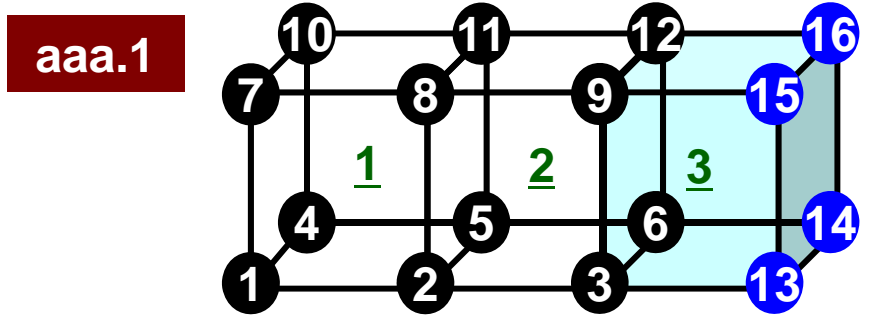
void** allocate_matrix(int size, int m, int n)
{
    void **aa;
    int i;
    if ( ( aa=(void **)malloc( m * sizeof(void*) ) ) == NULL ) {
        fprintf(stdout, "Error:Memory does not enough! aa in matrix %n");
        exit(1);
    }
    if ( ( aa[0]=(void *)malloc( m * n * size ) ) == NULL ) {
 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

```

Same interface with FORTRAN

# Local Numbering: Neighbors/Nodes



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	ID of PE	NEIBPETOT: # neighbors	NEIBPE(neib): ID of neighbors
1			
1			
1			
16			
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

# Reading Meshes: 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 %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]);
}

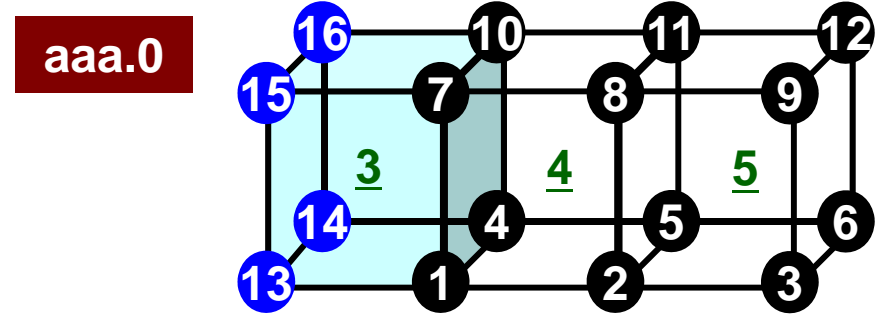
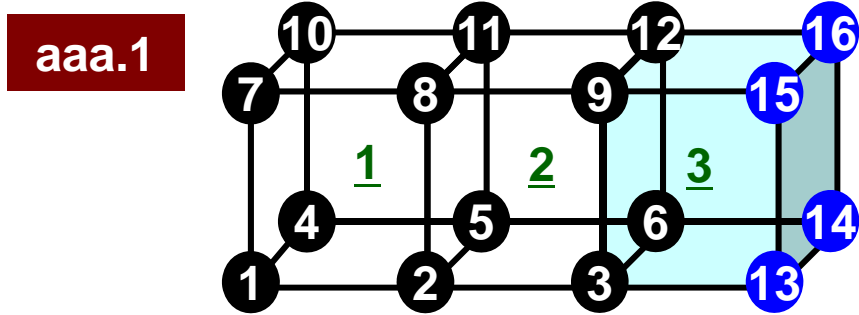
for (ic0=0; ic0<ICELTOT_INT; ic0++) fscanf(fp, "%d", &intELEM_list[ic0]);

```

# Local Numbering: Nodes

- Local node ID starts from “1” in each PE
  - Same program for 1-CPU can be used: SPMD
  - Local element ID also starts from “1”
- Numbering: Internal -> External Points
- Double Numbering
  - Local node ID at its “home” PE
  - ID of “home” PE

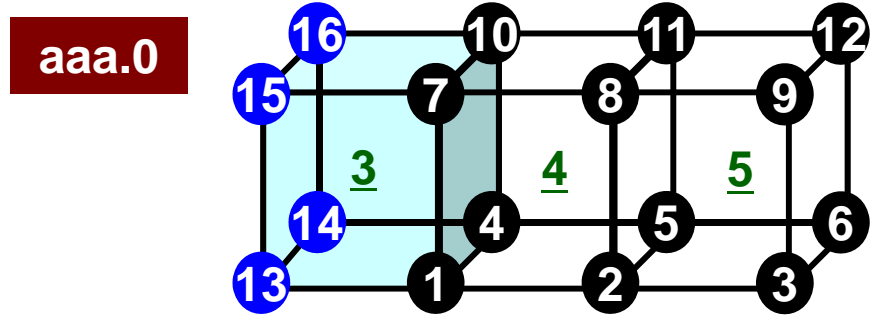
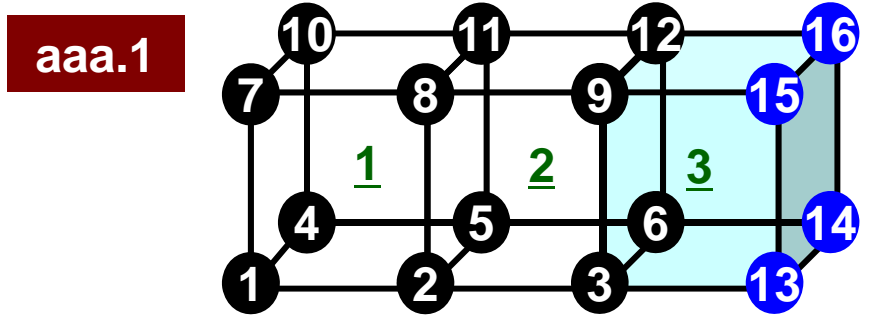
# Local Numbering: Nodes



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	(# Nodes, Internal Pts.)		
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

# Local Numbering: Nodes



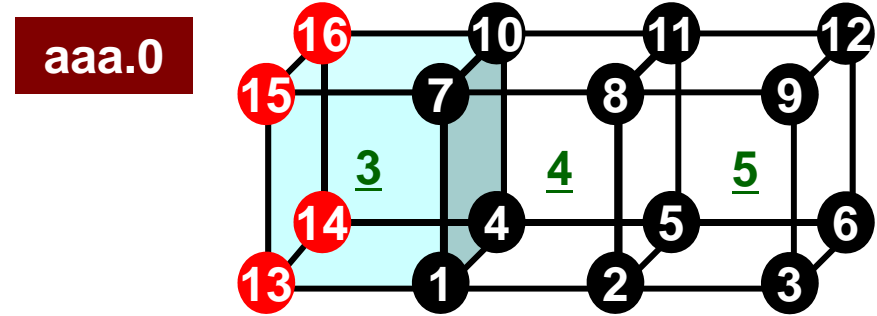
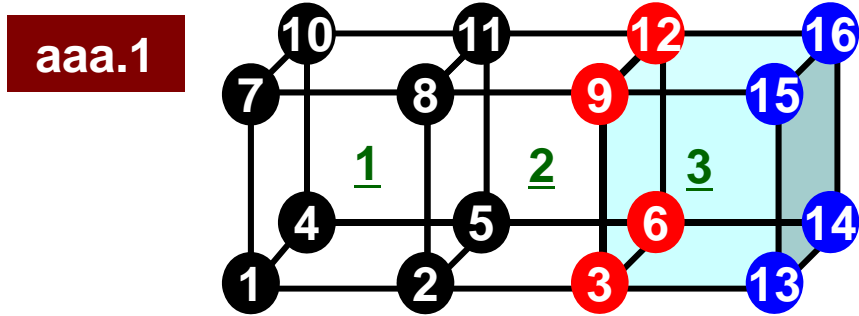
1					
1					
0					
16		12			
1	1	0.00	0.00	0.00	①
2	1	1.00	0.00	0.00	②
3	1	2.00	0.00	0.00	③
4	1	0.00	1.00	0.00	④
5	1	1.00	1.00	0.00	⑤
6	1	2.00	1.00	0.00	⑥
7	1	0.00	0.00	1.00	⑦
8	1	1.00	0.00	1.00	⑧
9	1	2.00	0.00	1.00	⑨
10	1	0.00	1.00	1.00	⑩
11	1	1.00	1.00	1.00	⑪
12	1	2.00	1.00	1.00	⑫
1	0	3.00	0.00	0.00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

"Home" PE, Local ID                      Coordinates

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

"Home" PE, Local ID                      Coordinates

# Local Numbering: Nodes



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

"Home" PE, Local ID                      Coordinates

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

"Home" PE, Local ID                      Coordinates



# Reading Meshes: 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);

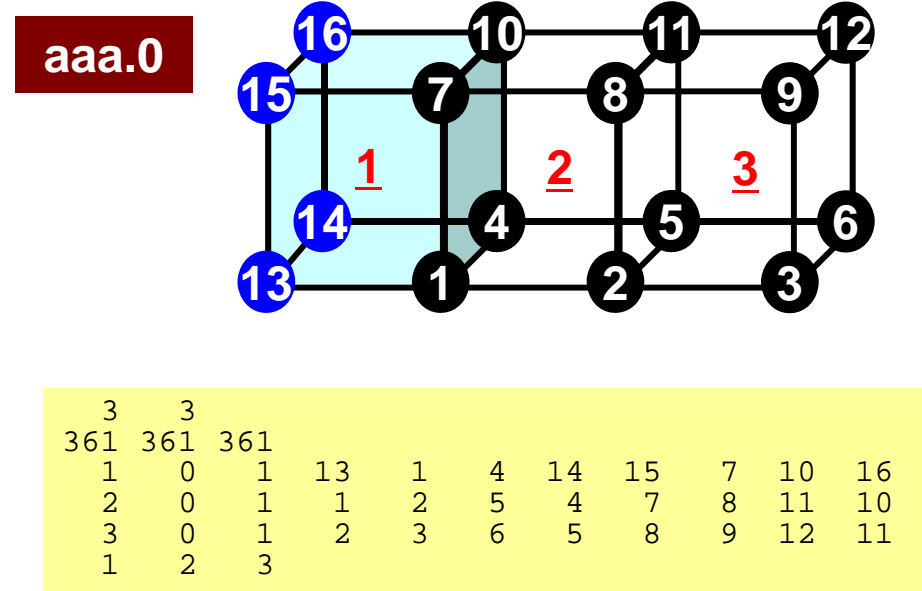
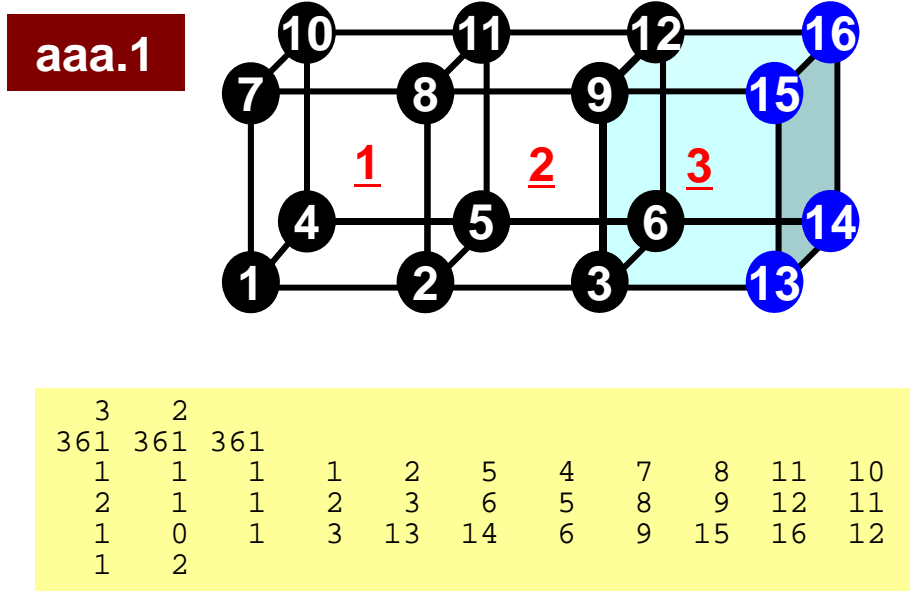
for (i=0; i<ICELTOT; i++) fscanf(fp, "%d", &NTYPE);

for (icel=0; icel<ICELTOT; icel++) {
    fscanf(fp, "%d %d %d %d %d %d %d %d %d %d %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]);
}

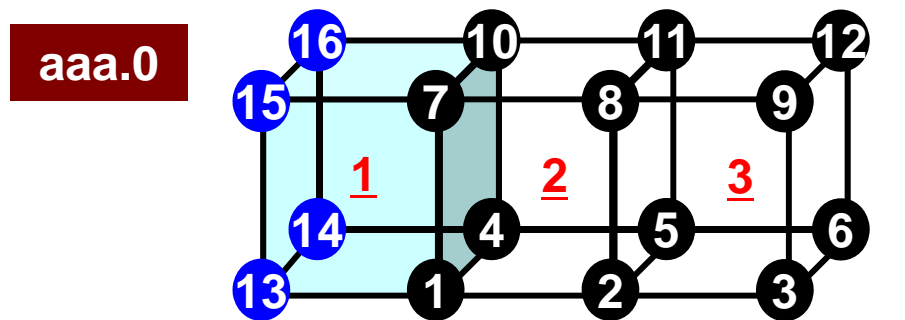
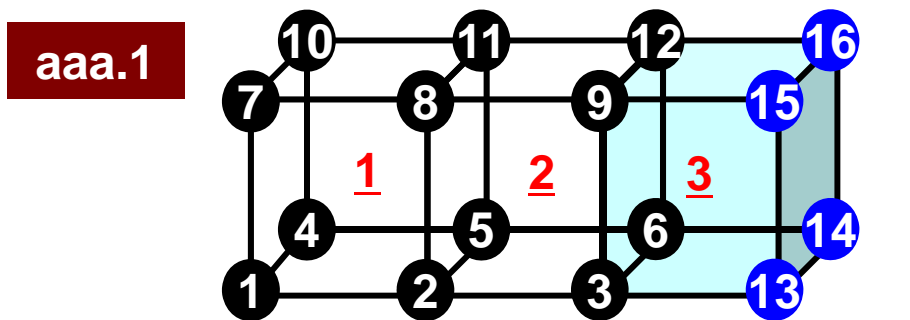
for (ic0=0; ic0<ICELTOT_INT; ic0++) fscanf(fp, "%d", &intELEM_list[ic0]);

```

# Local Numbering: Elements



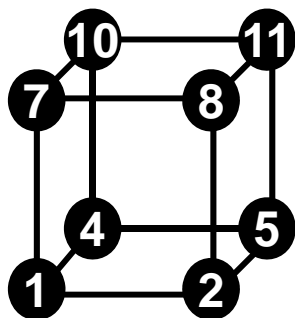
# Local Numbering: Elements



3			2									
361	361	361	1	2	5	4	7	8	11	10		
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	13	1	4	14	15	7	10	16
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									

- “Home” PE of Element

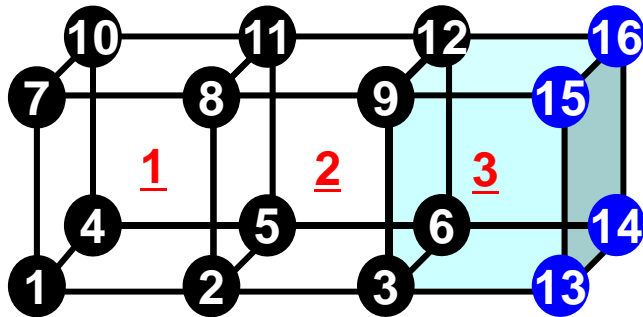


- Defined by “home” of 8 nodes
- If all of 8 nodes are internal pts., “home” of the element is that of 8 nodes.
- If external nodes are included, the smallest number of ID of “home” of the nodes is selected.
- In this case, “home” PE’s of elements in overlapped region are all “0”.
- used in OUTPUT\_UCD

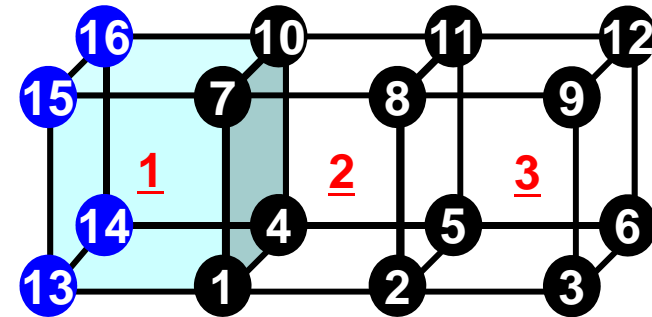


# Local Numbering: Elements

aaa.1



aaa.0



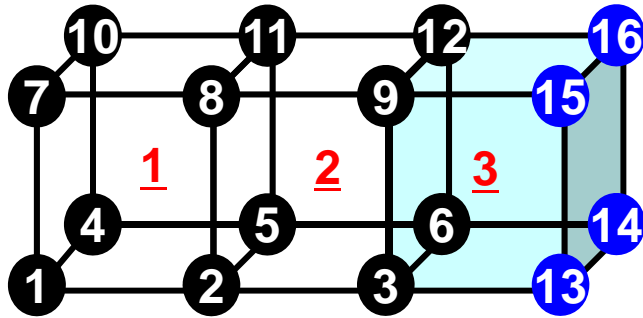
<u>3</u>	2										
361	361	361									
1	1	1	1	2	5	4	7	8	11	10	<u>1</u>
2	1	1	2	3	6	5	8	9	12	11	<u>2</u>
1	0	1	3	13	14	6	9	15	16	12	<u>3</u>
1	2										

<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									

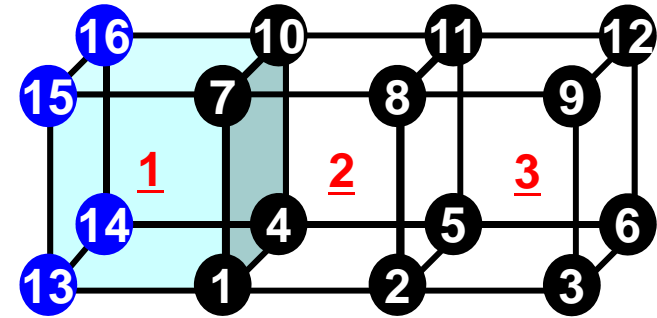
- Double Numbering for Element
  - Local ID at “home” PE
  - ID of “home” PE
- Material ID
- 8 Nodes
- Underlined local ID is used in the program

# Local Numbering: Elements

aaa.1



aaa.0



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

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

- aaa.1
  - 1, 2 are “Local Elements” (“Home Elements”)
- aaa.0
  - 1, 2, 3 are “Local Elements”

# Reading Meshes: 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]);}
}

```

# PE-to-PE Communication

## Generalized Communication Tables

- “Communication” in parallel FEM means obtaining information of “external points” from their “home” PE’s
- “Communication Tables” describe relationship of “external points” among PE’s
  - Send/Export, Recv/Import
- Sending information of “boundary points”
- Receiving information of “external points”

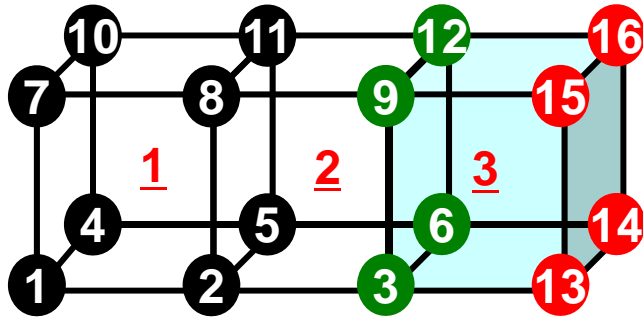


# Generalized Comm. Table: Send

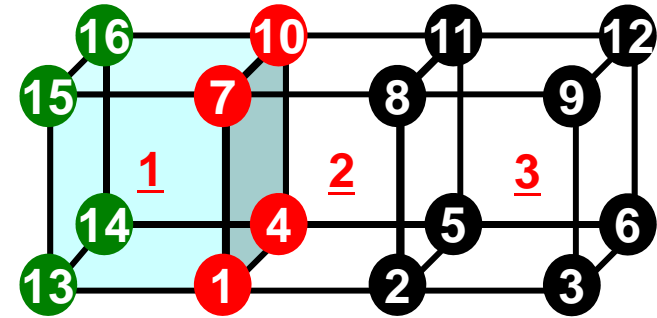
- Neighbors
  - NeibPETot, NeibPE[neib]
- Message size for each neighbor
  - export\_index[neib], neib= 0, NeibPETot-1
- ID of **boundary** points
  - export\_item[k], k= 0, export\_index[NeibPETot]-1
- Messages to each neighbor
  - SendBuf[k], k= 0, export\_index[NeibPETot]-1

# Communication Table (Send/Export)

aaa.1



aaa.0



```

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

```

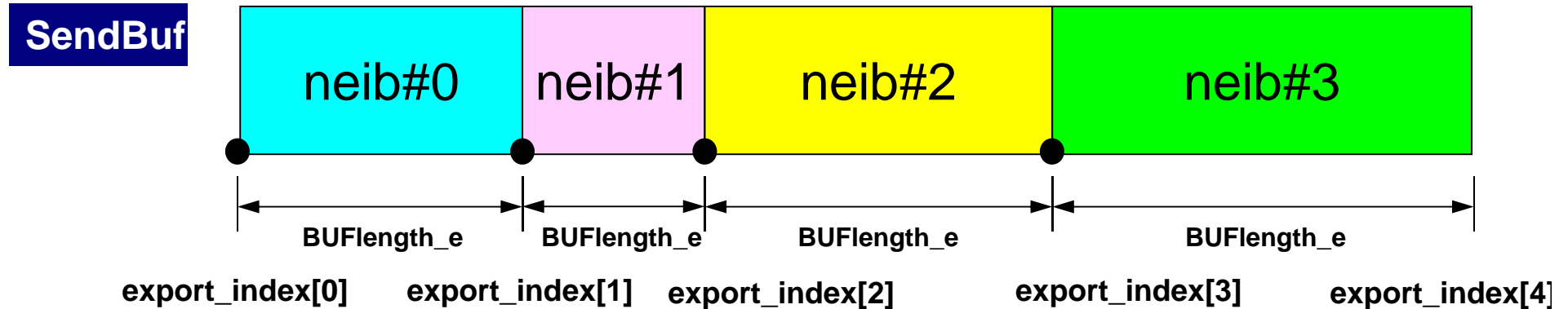
```

4
13      1
14      1
15      1
16      1
4      export_index(neib)
3      export_item
6
9
12

```

- `export_index` Size of Messages sent to Each Neighbor
  - # Neighbors= 1 in this case
- `export_item` Local ID of boundary points

# SEND: MPI\_Isend/Irecv/Waitall



`export_item (export_index[neib]:export_index[neib+1]-1)` are sent to neib-th neighbor

```
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];
  }
}
```

Copies to sending buffers

```
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

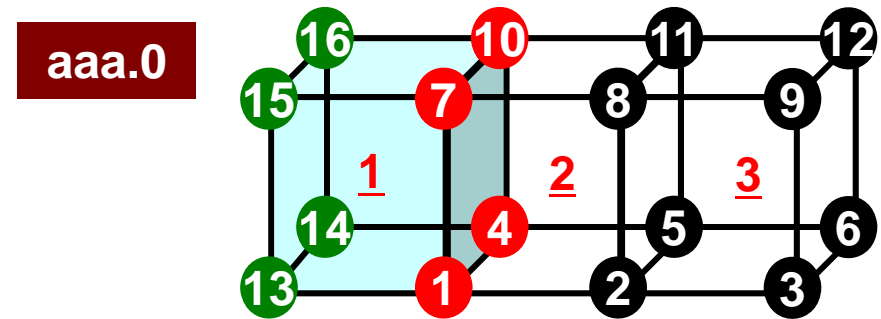
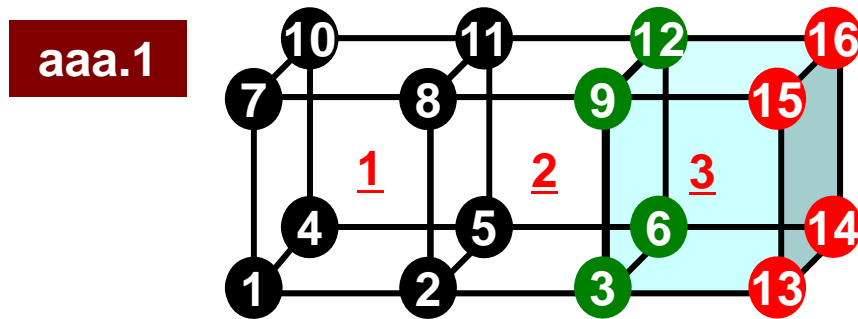
  ierr= MPI_Isend
    (&SendBuf[iS_e], BUFlength_e, MPI_DOUBLE, NeibPE[neib], 0,
     MPI_COMM_WORLD, &ReqSend[neib])
}
```

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

# Generalized Comm. Table: Receive

- Neighbors
  - NeibPETot , NeibPE[neib]
- Message size for each neighbor
  - import\_index[neib], neib= 0, NeibPETot-1
- ID of **external** points
  - import\_item[k], k= 0, import\_index[NeibPETot]-1
- Messages from each neighbor
  - RecvBuf[k], k= 0, import\_index[NeibPETot]-1

# Communication Table (Recv/Import)



```

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

```

```

4      import_index(neib)
13     1 import_item, "home" PE
14     1
15     1
16     1
4      export_index(neib)
1      export_item
4
7
10

```

- `import_index`    Size of Messages recv. from Each Neighbor
  - # Neighbors= 1 in this case
- `import_item`    Local ID of external points, and their “home”

# RECV: MPI\_Irecv/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])
}

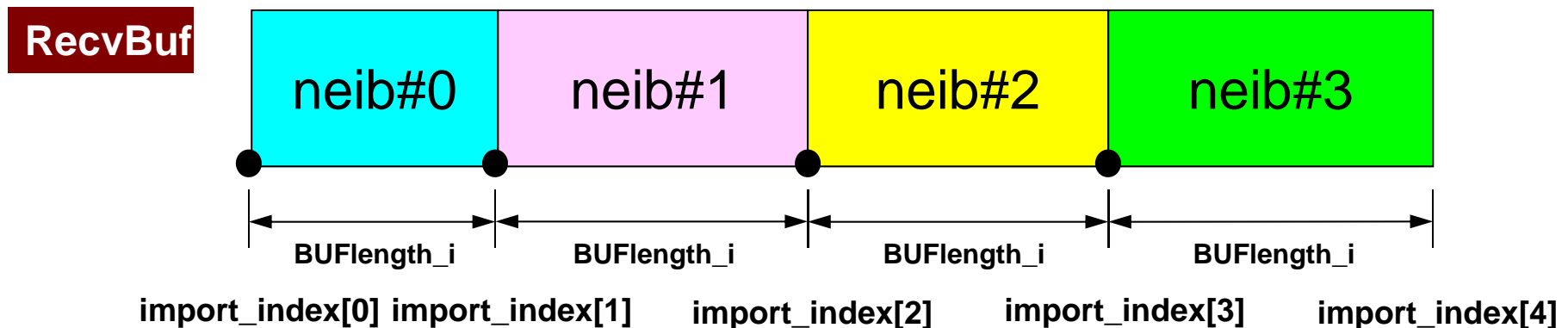
MPI_Waitall(NeibPETot, ReqRecv, StatRecv);

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];
    }
}

```

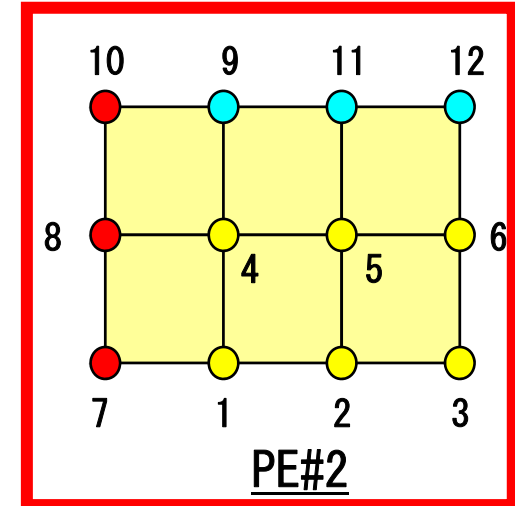
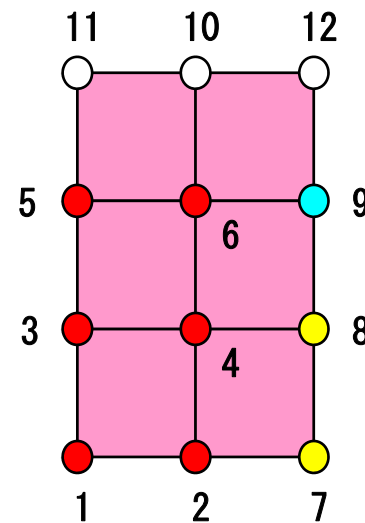
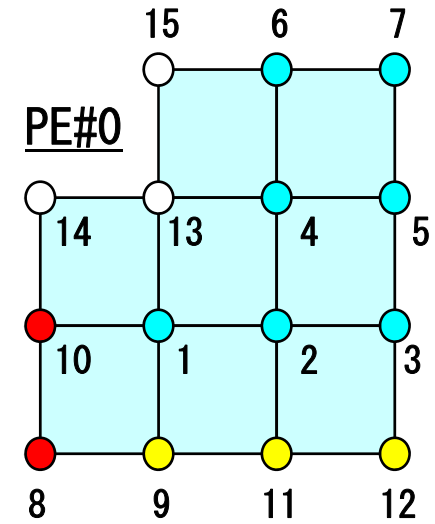
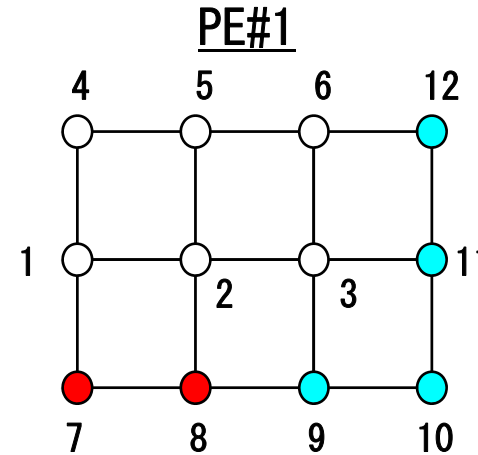
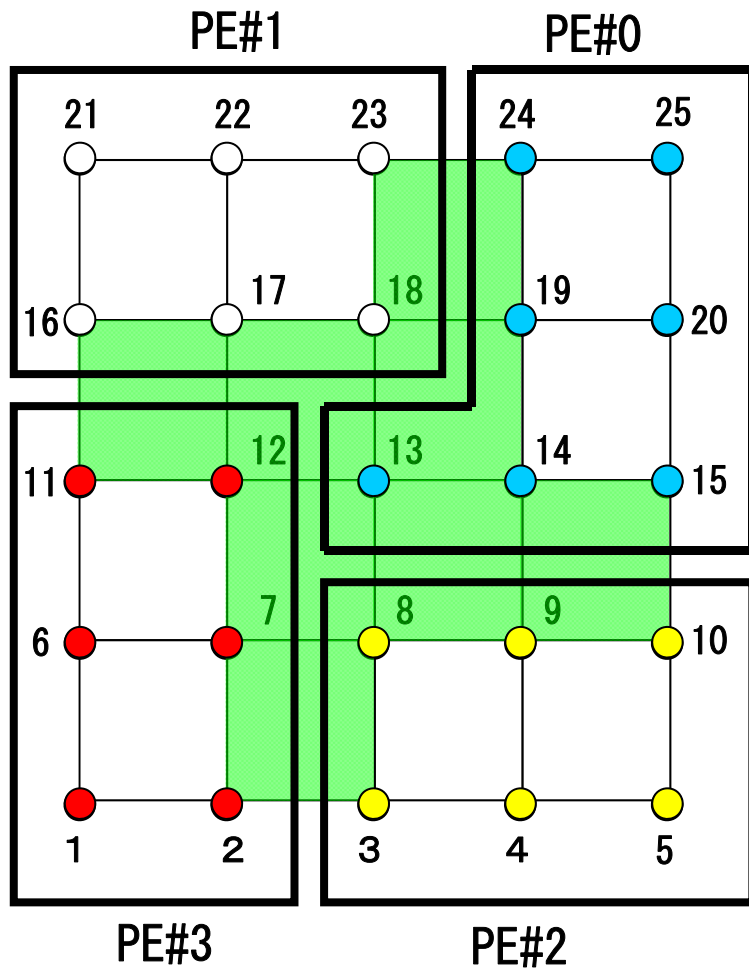
Copies from receiving buffers

`import_item (import_index[neib]:import_index[neib+1]-1)` are received from `neib`-th neighbor



# Node-based Partitioning

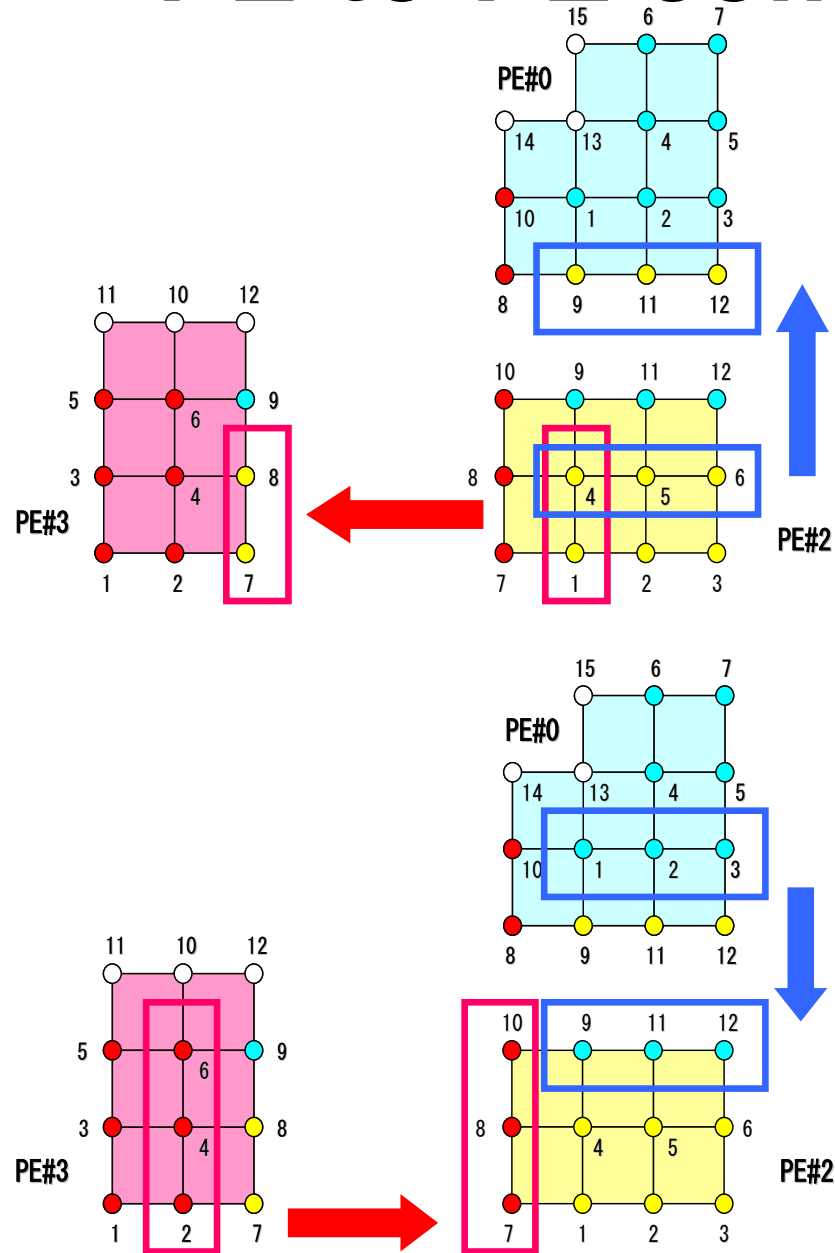
internal nodes - elements - external nodes



PE#3

PE#2

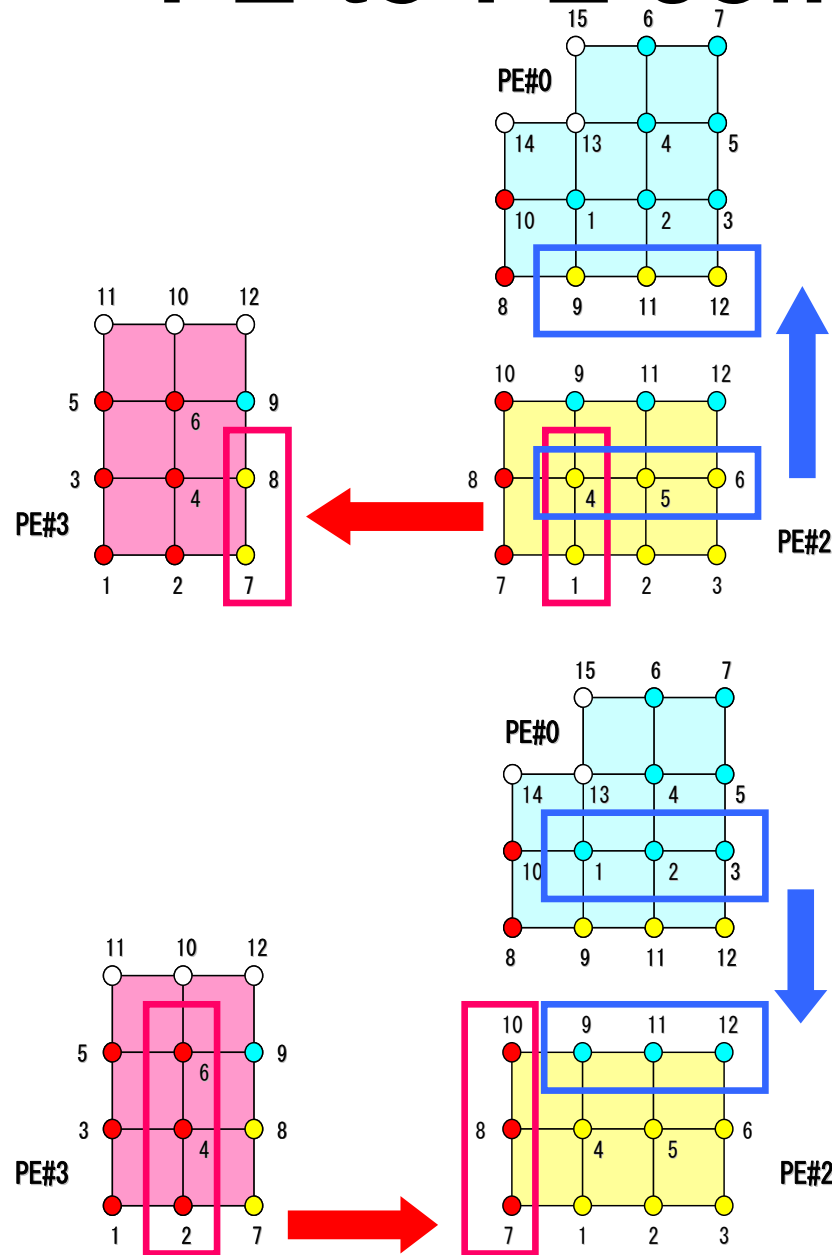
# PE-to-PE comm. : Local Data



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



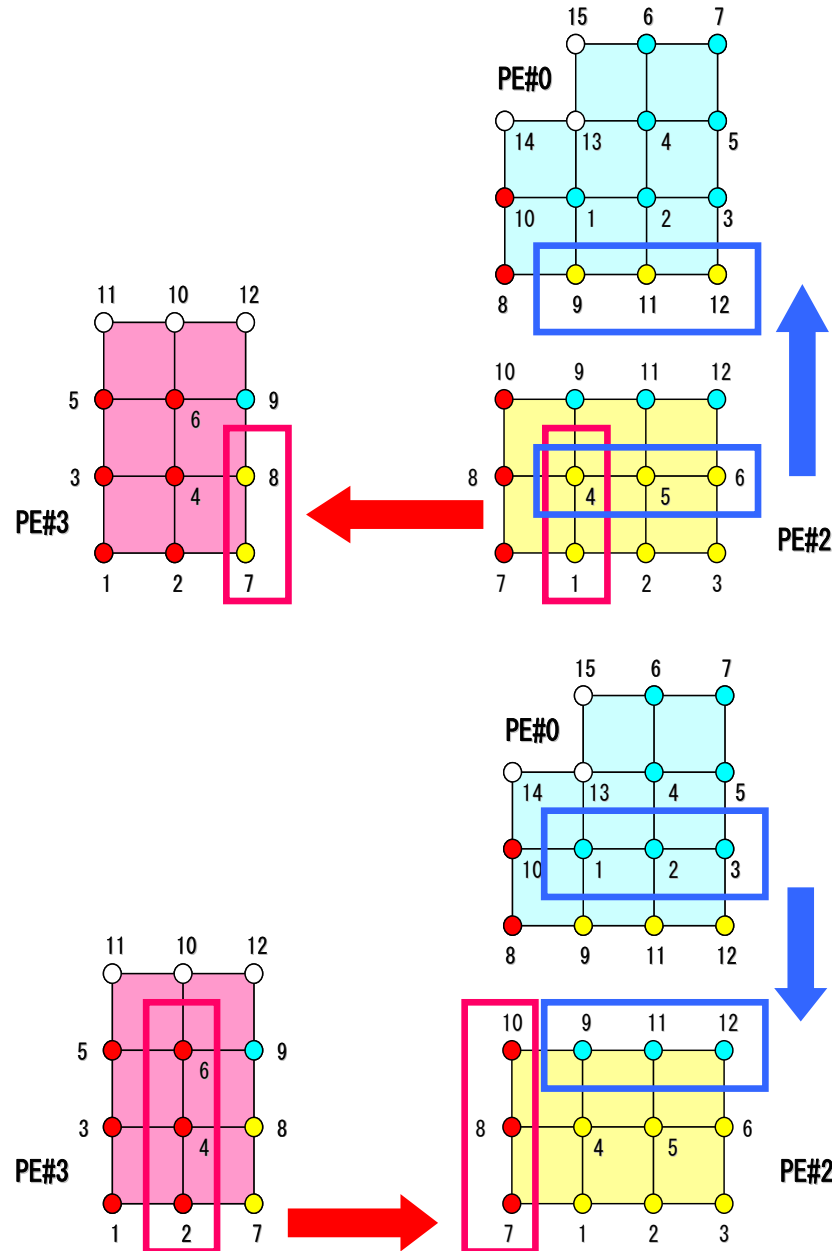
# PE-to-PE comm. : Local Data



	2	ID of the PE
	2	# Neighbors
	3	0
(...)		
	3	6
	7	3
	8	3
10	3	3
9	0	0
11	0	0
12	0	0
2	5	5
1		
4		
4		
5		
6		

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

# PE-to-PE comm. : SEND



	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**

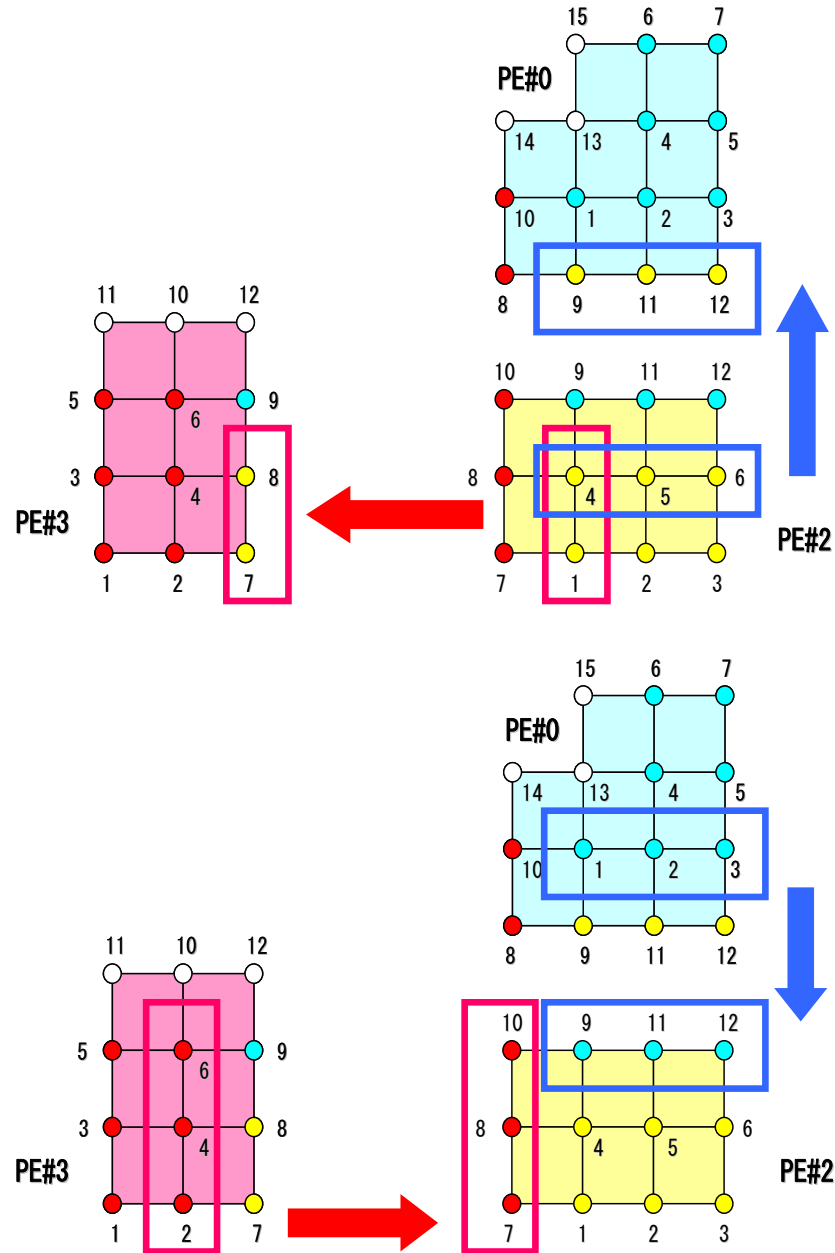
```

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

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

4th node is sent to two PE's
    
```

# PE-to-PE comm. : RECV



```
(...)
2
2
3
3
0
6 import_index
7
8
10
9
11
12
2
1
4
4
5
6
```

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

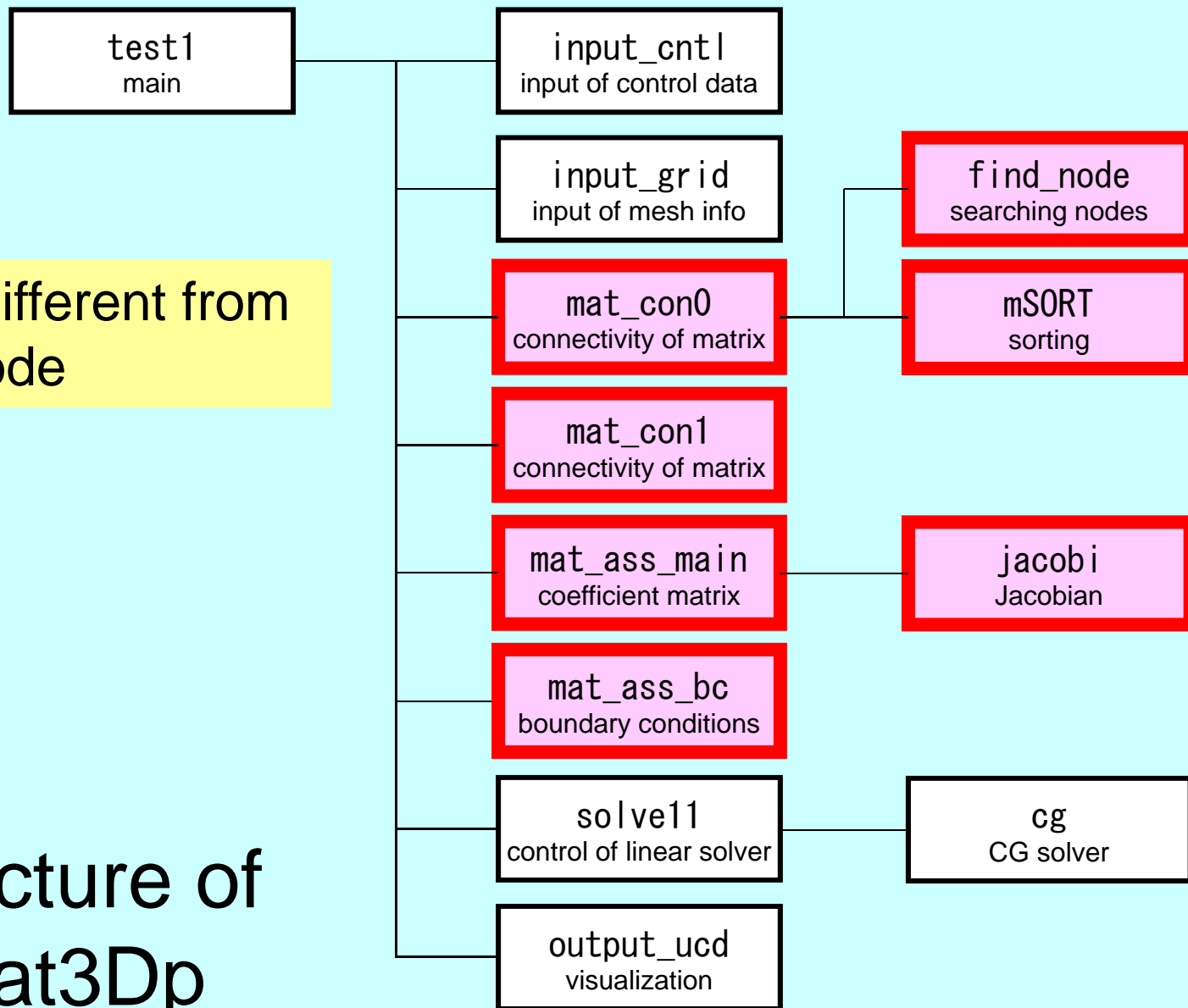
import_item[0-5]=7,8,10,9,11,12
```

# Parallel FEM Procedures: Program

- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - **Initialization of Arrays (Global/Element Matrices)**
  - **Element-Global Matrix Mapping (Index, Item)**
- **Generation of Matrix**
  - **Element-by-Element Operations (do icel= 1, NE)**
    - Element matrices
    - Accumulation to global matrix
  - **Boundary Conditions**
- Linear Solver
  - Conjugate Gradient Method

NOT so different from  
1-CPU code

## Structure of heat3Dp



# Towards Matrix Assembling

- In 1D, it was easy to obtain information related to index and item.
  - 2 non-zero off-diagonals for each node
  - ID of non-zero off-diagonal :  $i+1$ ,  $i-1$ , where “i” is node ID
- In 3D, situation is more complicated:
  - Number of non-zero off-diagonal components is between 7 and 26 for the current target problem
  - More complicated for real problems.
  - Generally, there are no information related to number of non-zero off-diagonal components beforehand.

# Towards Matrix Assembling

- In 1D, it was easy to obtain information related to index and item.
  - 2 non-zero off-diagonals for each node
  - ID of non-zero off-diagonal :  $i+1$ ,  $i-1$ , where “i” is node ID
- In 3D, situation is more complicated:
  - Number of non-zero off-diagonal components is between 7 and 26 for the current target problem
  - More complicated for real problems.
  - Generally, there are no information related to number of non-zero off-diagonal components beforehand.
- **Count number of non-zero off-diagonals using arrays: INLU[N], IALU[N][NLU]**

# Main Part

```

#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:** generates INU, IALU  
**MAT\_CON1:** generates index, item

**Node ID starting from "1"**

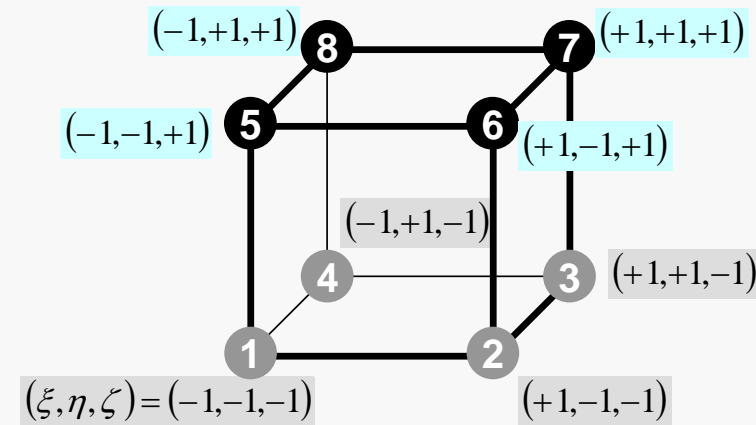
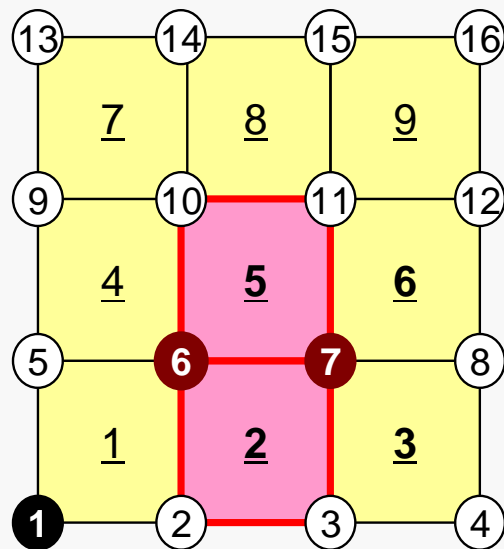


# MAT\_CON0: Overview

```

do icel= 1, ICELTOT
  generate INLU, IALU
  according to 8 nodes of hex. elements
  (FIND_NODE)
enddo

```



# Generating Connectivity of Matrix 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:

Number of maximum number of connected nodes to each node (number of upper/lower non-zero off-diagonal blocks)

In the current problem, geometry is rather simple. Therefore we can specify NLU in this way.

If it's not clear ->  
Try more flexible implementation

# Generating Connectivity of Matrix MAT\_CON0 (1/4)

```

/**
** MAT_CON0
**/

#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_CON0 ()
{
  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;

```

Array	Size	Description
INLU	[NP]	Number of connected nodes to each node (lower/upper)
IALU	[NP] [NLU]	Corresponding connected node ID (column ID)

# Generating Connectivity of Matrix MAT\_CON0 (2/4): Starting from 1

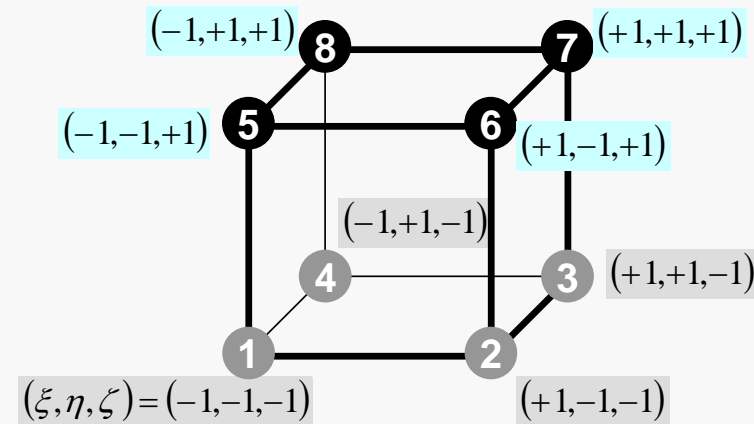
```
for( icel=0; icel < ICELTOT; icel++) {
```

```
  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];
```

```
  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);
```



# FIND\_TS\_NODE: Search Connectivity

## INLU,IALU: Automatic Search

```

/**
*** 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;
}

```

Array	Size	Description
INLU	[NP]	Number of connected nodes to each node (lower/upper)
IALU	[NP] [NLU]	Corresponding connected node ID (column ID)

# FIND\_TS\_NODE: Search Connectivity

## INLU,IALU: Automatic Search

```

/**
 *** FIND_TS_NODE
 **/

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

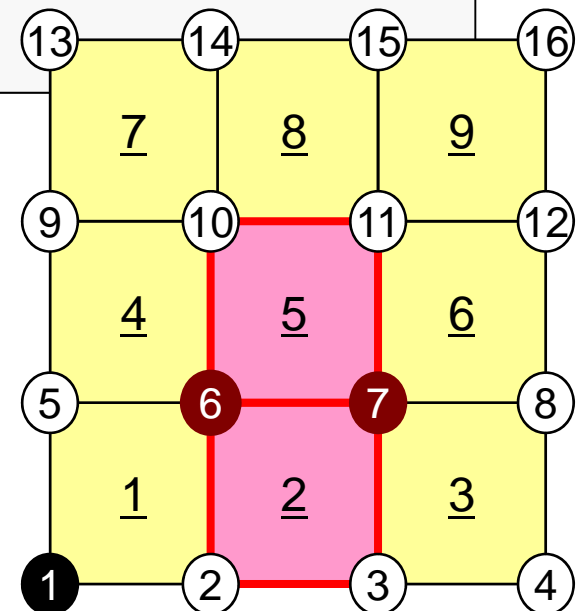
  for (kk=1;kk<=INLU[ip1-1];kk++) {
    if (ip2 == IALU[ip1-1][kk-1]) return;
  }

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

  return;
}

```

If the target node is already included in IALU, proceed to next pair of nodes



# FIND\_TS\_NODE: Search Connectivity

## INLU,IALU: Automatic Search

```

/**
 *** FIND_TS_NODE
 **/

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

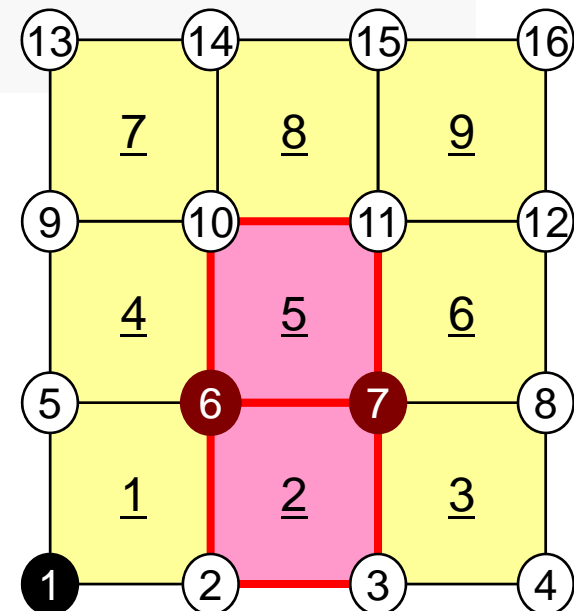
  for (kk=1;kk<=INLU[ip1-1];kk++) {
    if (ip2 == IALU[ip1-1][kk-1]) return;
  }

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

  return;
}

```

If the target node is NOT included in IALU, store the node in IALU, and add 1 to INLU.



# Generating Connectivity of Matrix 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);

```

```

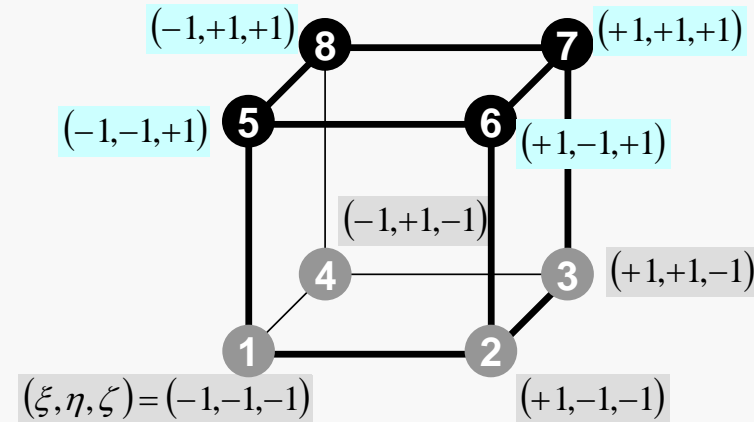
FIND_TS_NODE (in6, in1);
FIND_TS_NODE (in6, in2);
FIND_TS_NODE (in6, in3);
FIND_TS_NODE (in6, in4);
FIND_TS_NODE (in6, in5);
FIND_TS_NODE (in6, in7);
FIND_TS_NODE (in6, in8);

```

```

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);

```





# Generating Connectivity of Matrix 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];  
    }  
}
```

Sort IALU[i][k] in ascending order by  
“bubble” sorting for less than 100  
components.

# MAT\_CON1: CRS format

```

#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), NP+1);
    for (i=0; i<NP+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

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

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

# MAT\_CON1: CRS format

```

#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=indexLU[NP]**  
**Size of array: itemLU**  
**Total number of non-zero off-diagonal blocks**

# MAT\_CON1: CRS format

```

#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), NP+1);
    for (i=0; i<NP+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);
}

```

itemLU

store node ID starting from 0

# MAT\_CON1: CRS format

```

#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), NP+1);
    for (i=0; i<NP+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);
}

```

Not required any more

# Main Part

```
#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 (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);
```

```
Non-Zero Off-Diagonal components (coef. matrix)
RHS vector
Diagonal components (coef. matrix)
Unknowns
```

```
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;
```



# 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;

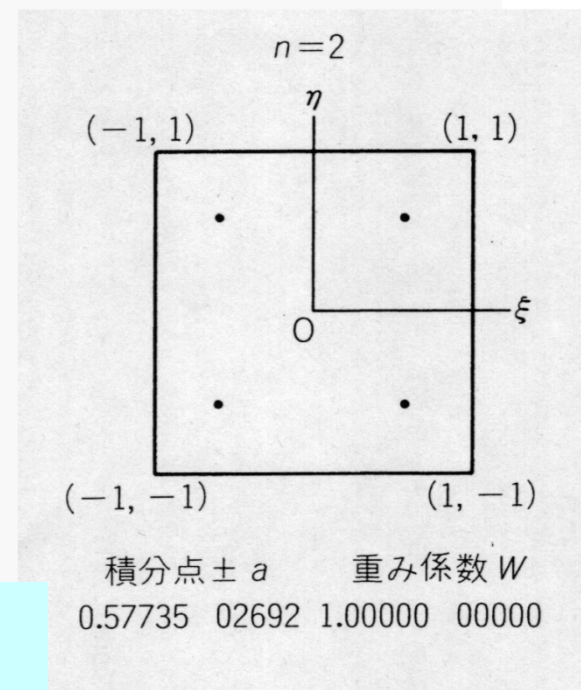
```

```

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

```

**POS:** Quad. Point  
**WEI:** Weighting Factor



# 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;
```

# 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;

```

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

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

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

# 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++) {

```

```

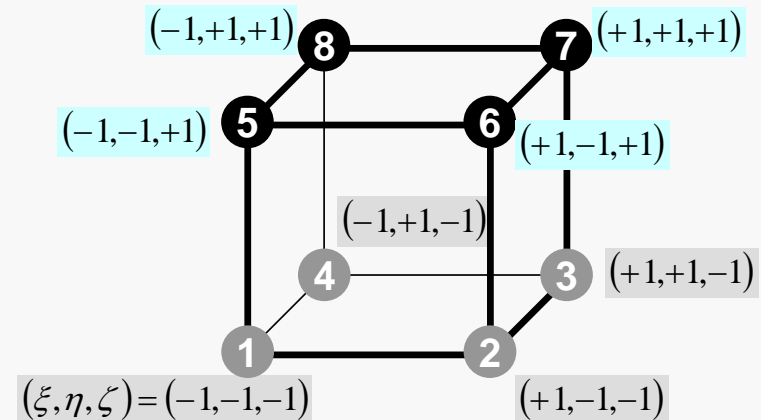
      QP1= 1. e0 + POS[ip]:
      QM1= 1. e0 - POS[ip]:
      EP1= 1. e0 + POS[jp]:
      EM1= 1. e0 - POS[jp]:
      TP1= 1. e0 + POS[kp]:
      TM1= 1. e0 - POS[kp]:

```

```

      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;

```



# 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;
    }
  }
}

```

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

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

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

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

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

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

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

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

# MAT\_ASS\_MAIN (2/6)

```

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] ;

```

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

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

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

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

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

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

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

First Order Derivative  
of Shape Functions at  
 $(\xi_i, \eta_j, \zeta_k)$

# 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 ;

```

```

}
}
}

```

```

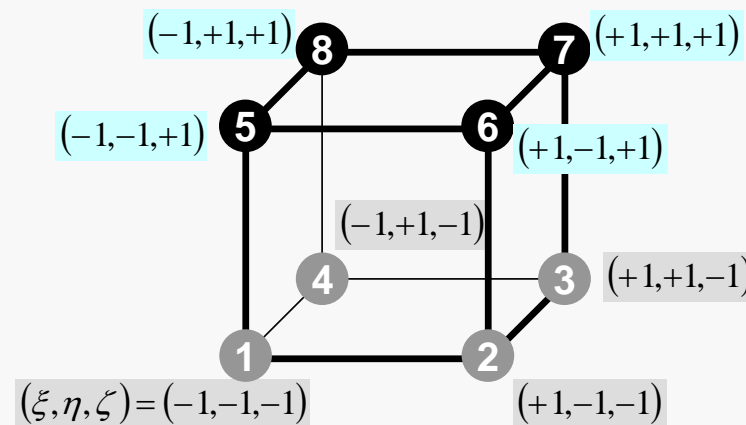
for ( icel=0; icel < ICELTOT; icel++) {
  CONDO= COND;

```

```

  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 ] ;

```



# 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] ;

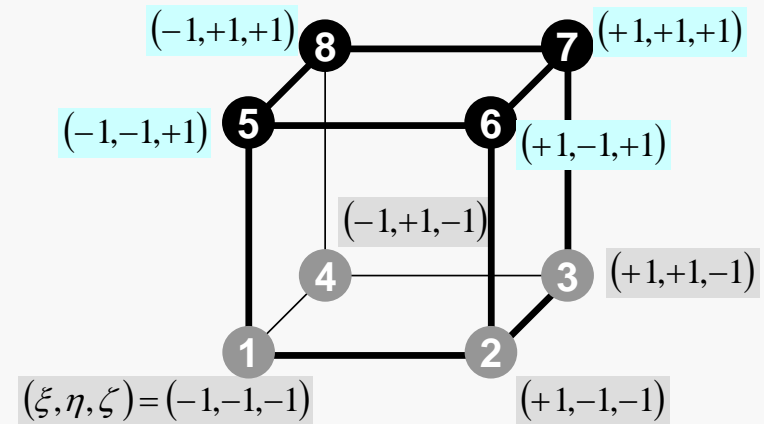
```

```

JACOBI (DETJ, PNQ, PNE, PNT, 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) ;

```

Node ID (Global)





# 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] ;

```

X-Coordinates  
of 8 nodes

```

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] ;

```

Y-Coordinates  
of 8 nodes

```

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

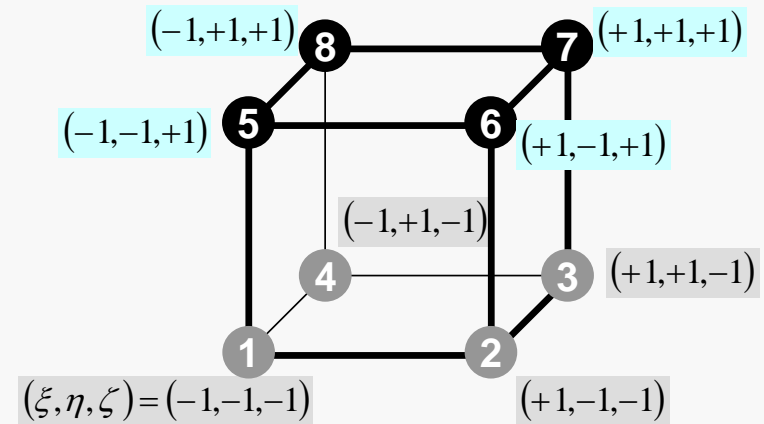
```

```

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] ;

```

Z-Coordinates  
of 8 nodes



Coordinates:  
Node ID - 1

```

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) ;

```

# 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] ;
```

X-Coordinates  
of 8 nodes

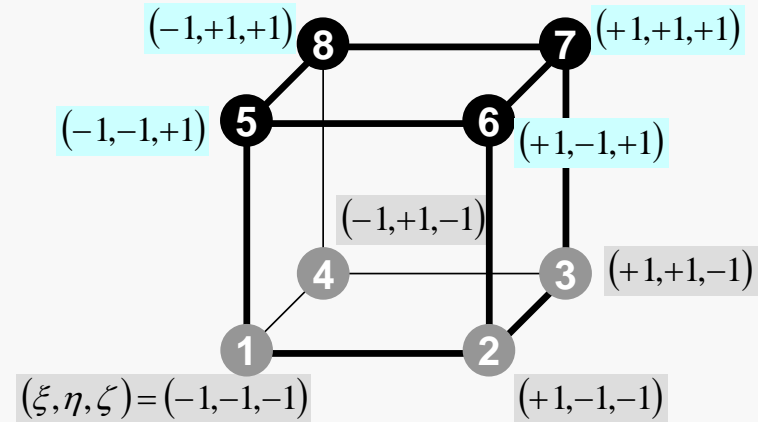
```
Y1=XYZ [in1-1] [1] ;
Y2=XYZ [in2-1] [1] ;
Y3=XYZ [in3-1] [1] ;
Y4=XYZ [in4-1] [1] ;
Y5=XYZ [in5-1] [1] ;
Y6=XYZ [in6-1] [1] ;
Y7=XYZ [in7-1] [1] ;
Y8=XYZ [in8-1] [1] ;
```

Y-Coordinates  
of 8 nodes

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) ;
```



Coordinates:  
Node ID - 1

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

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

Heat Gen. Rate is a function of location  
(cell center:  $x_c, y_c$ )

# 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];

```

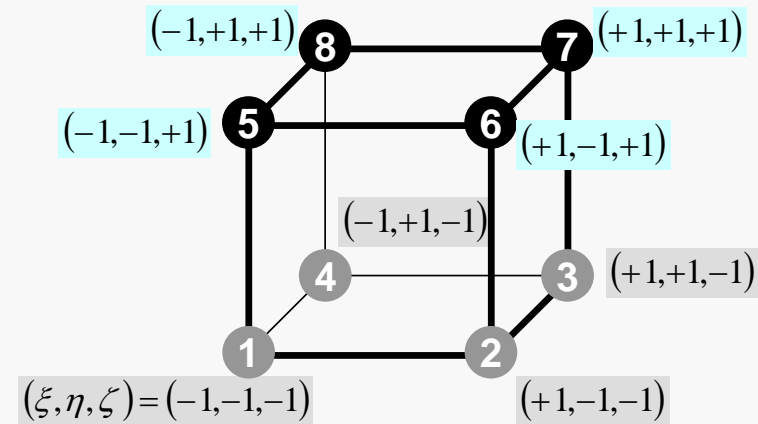
**QVC= 08th\*(X1+X2+X3+X4+X5+X6+X7+X8+Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8);**

```

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, 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);



**Coordinates:  
Node ID - 1**

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

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

$$QVC = |x_c + y_c|$$

# 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) ;

```

# MAT\_ASS\_MAIN (5/6)

```

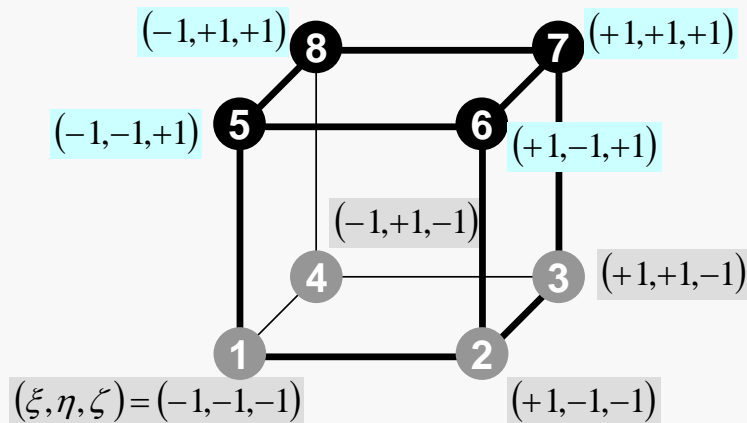
/**
CONSTRUCT the GLOBAL MATRIX
**/

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;
        }
      }
    }
  }
}

```



Non-Zero Off-Diagonal Block  
in Global Matrix

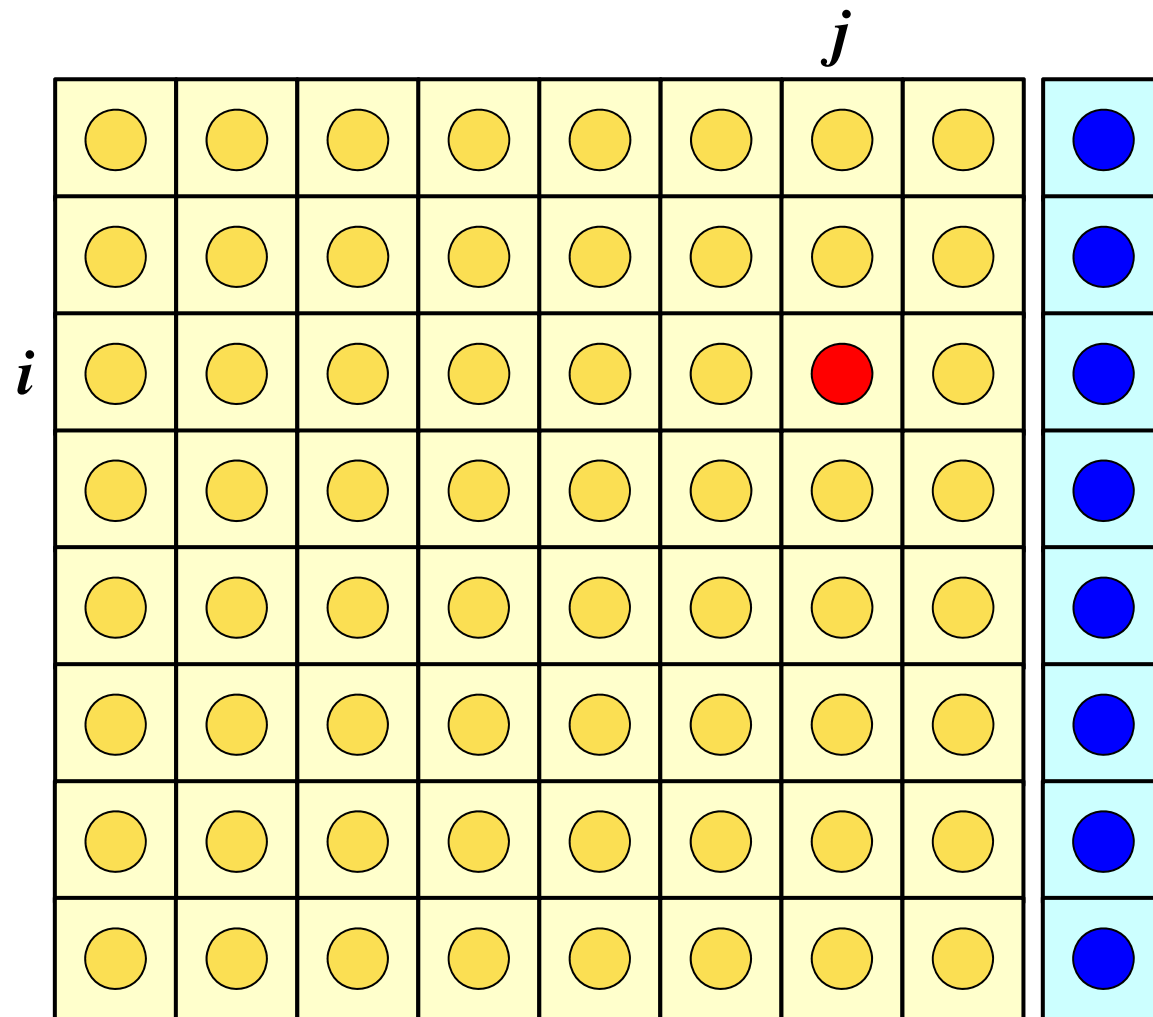
$$A_{ip, jp}$$

kk: address in “itemLU”

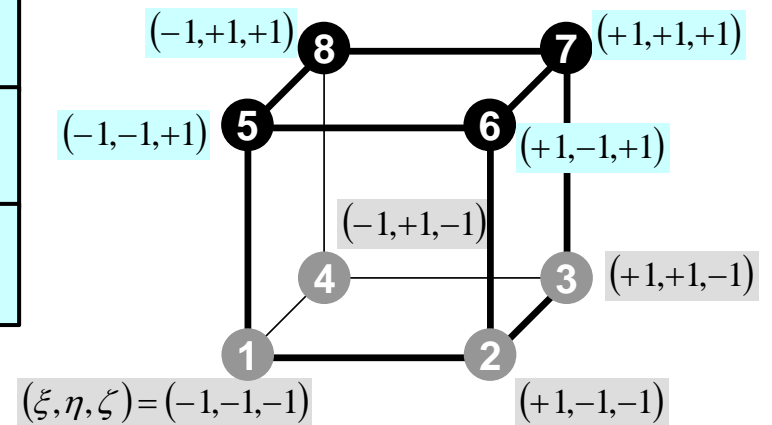
ip= nodLOCAL[ie]  
jp= nodLOCAL[je]

Node ID (ip,jp)  
starting from 1

# Element Matrix: 8x8



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



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

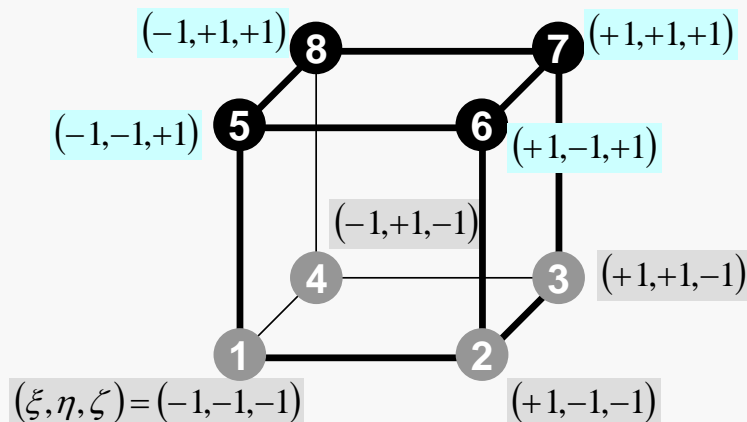
```

Element Matrix ( $i_e \sim j_e$ ): Local ID  
 Global Matrix ( $i_p \sim j_p$ ): Global ID

kk: address in "itemLU" starting from "0"

k: starting from "0"

ip,jp: starting from "1"



# 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;
}
}
}
}

```

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



# 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;
}
}
}
}

```

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

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

# 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*CONDO*(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;
}
}
}
}
}

```

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



# 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;
}
}
}
}

```

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

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

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

$$QVC = |x_c + y_c|$$

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

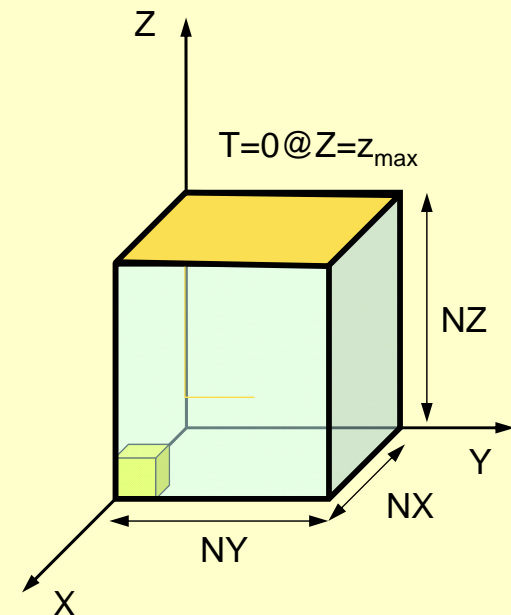
$$[f]^{(e)} = QV0 \cdot QVC$$

# MAT\_ASS\_BC: Overview

```
do i= 1, NP   Loop for Nodes
  "Mark" nodes where Dirichlet B.C. are applied (IWKX)
enddo
```

```
do i= 1, NP Loop for Nodes
  if (IWKX(i,1).eq.1) then if "marked" nodes
    corresponding components of RHS (B),
    Diagonal (D) are corrected
    do k= indexLU(i-1)+1, indexLU(i) Non-Zero Off-Diagonal Nodes
      corresponding comp. of non-zero off-diagonal
      components (AMAT) are corrected
    enddo
  endif
enddo
```

```
do i= 1, NP Loop for Nodes
  do k= indexLU(i-1)+1, indexLU(i) Non-Zero Off-Diagonal Nodes
    if (IWKX(itemLU(k),1).eq.1) then if corresponding non-zero
      off-diagonal node is "marked"
      corresponding components of RHS and AMAT are corrected (col.)
    endif
  enddo
enddo
```



# 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;
    }
}

```

If the node “in” is included in the node group “Zmax”

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

# 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;
    }
  }
}
}
```

# 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;
    }
  }
}

```

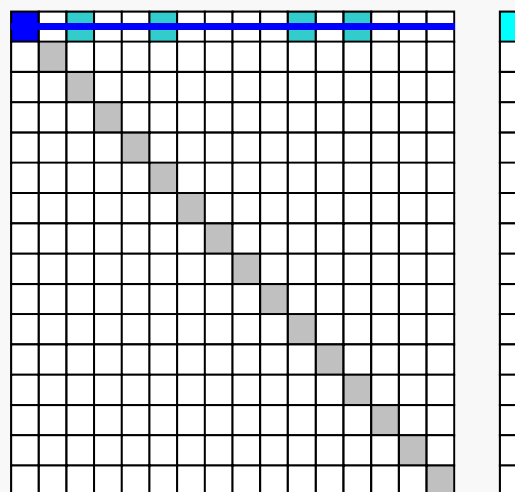
Boundary Nodes: IWKX[in-1][0]=1

```

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;
    }
  }
}
}

```

Erase !!



Same as 1CPU case



# MAT\_ASS\_BC (2/2)

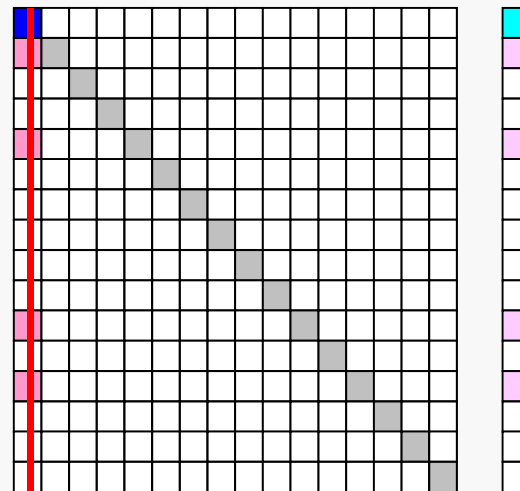
```

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;
    }
  }
}

```

Boundary Nodes:  $IWKX[in-1][0]=1$

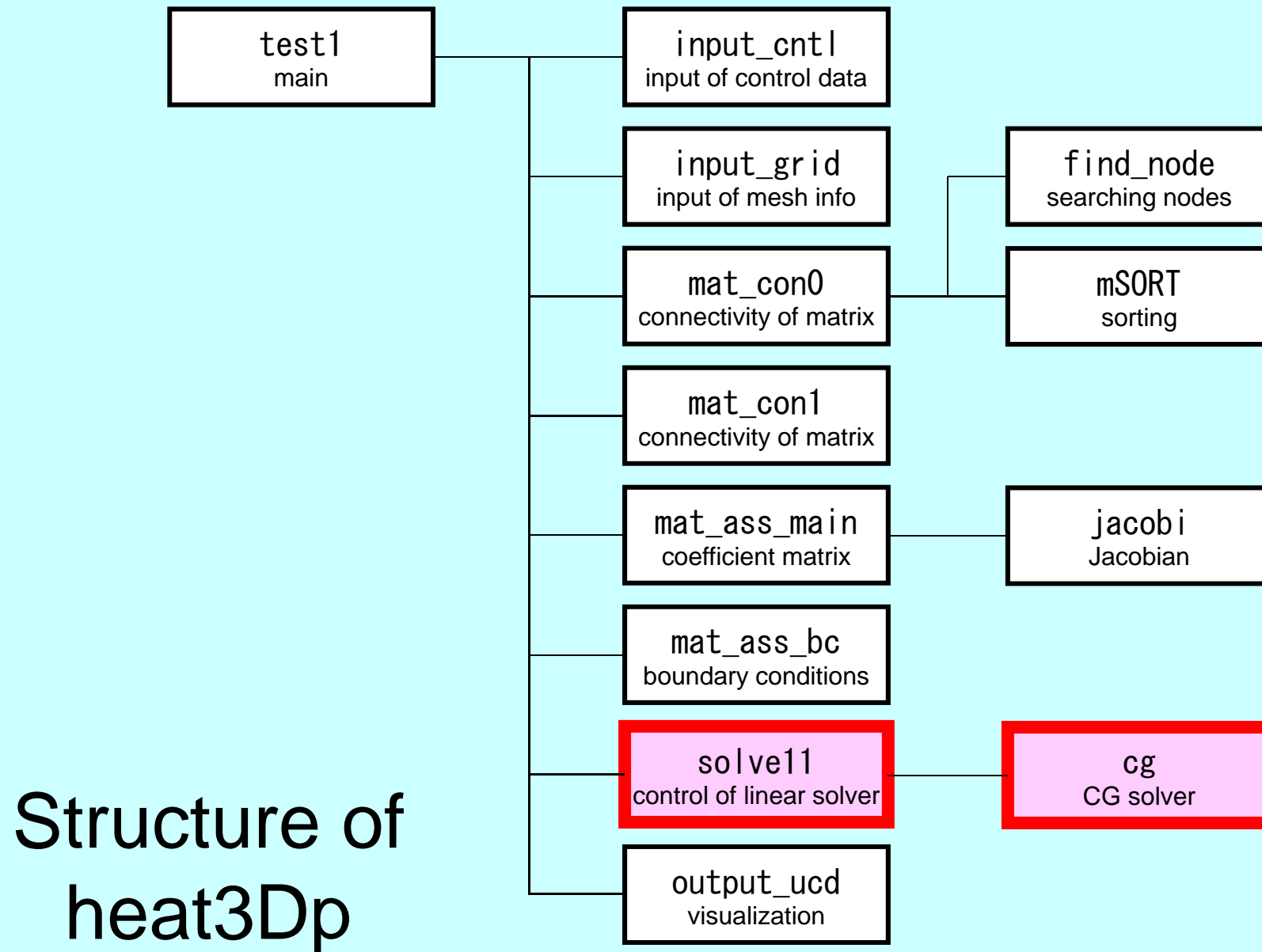


Elimination and Erase

Same as 1CPU case

# Parallel FEM Procedures: Program

- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation to global matrix
  - Boundary Conditions
- **Linear Solver**
  - **Conjugate Gradient Method**



# Main Part

```
#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];      Number of Iterations for CG
    RESID     = pfemRarray[0];      Convergence Criteria for CG
/**
+-----+
| 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);}

```

# Preconditioned CG Solver

## Diagonal Scaling/Point Jacobi Preconditioning

```

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

```

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

# Diagonal Scaling, Point-Jacobi

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

- solve  $[M]z^{(i-1)} = r^{(i-1)}$  is very easy.
- Provides fast convergence for simple problems.

# CG Solver (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;           Sending/Receiving Buffer
    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 Solver (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 Solver (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} |
  |-----|
  **/
  RHO0 = 0. e0;
  for (i=0; i<N; i++) {
    RHO0 += WW[R][i] * WW[Z][i];
  }
  MPI_Allreduce (&RHO0, &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[P][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 Solver (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 Solver (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 Solver (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);

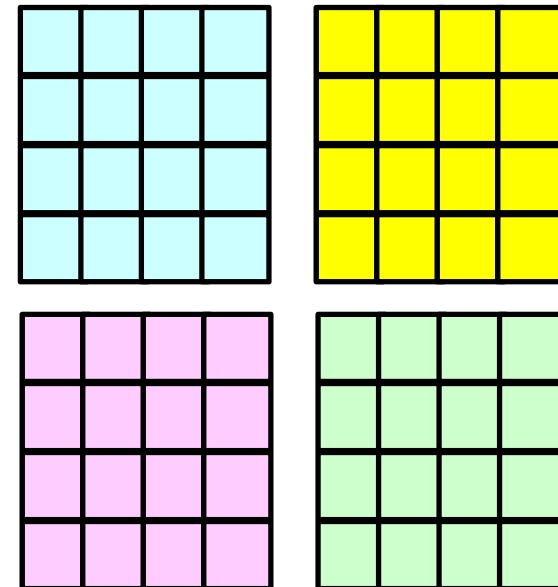
Updated temperature for external nodes

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

```

# OUTPUT\_UCD for Visualization

- Gather information of elements in “intELEM\_list” on each process
- Gather the following information to process #0 using MPI\_Allgather
  - Nodes: Coordinates, Displacement
  - Element: Connectivity
- Some overlapping in part of node information
- Not good for large-scale problems
  - Entire model on a single process
  - parallel visualization





# Example

- $128 \times 128 \times 128$  meshes
- 16~128 cores
  - RCB, pMETIS (for 96 cores)
- Linear Solver

<b>core #</b>	<b>Fortran sec. (speed-up)</b>	<b>C sec. (speed-up)</b>
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.)

# Report S3 (1/3)

- Evaluation behavior and performance of “sol”
- Example
  - Strong Scaling
    - Fixed entire problem size
  - Weak Scaling
    - Fixed problem size/core, time for 1 iterations
  - Parameters
    - Problem size
    - Domain decomposition (RCB, K-METIS, P-METIS, 1D-3D)
  - Mesh generation, partitioning: FX10 is slow
    - $\sim 128^3$
- “\*.inp” may take long time.
  - delete “call OUTPUT\_UCD”
  - src, part

# Report S3 (2/3)

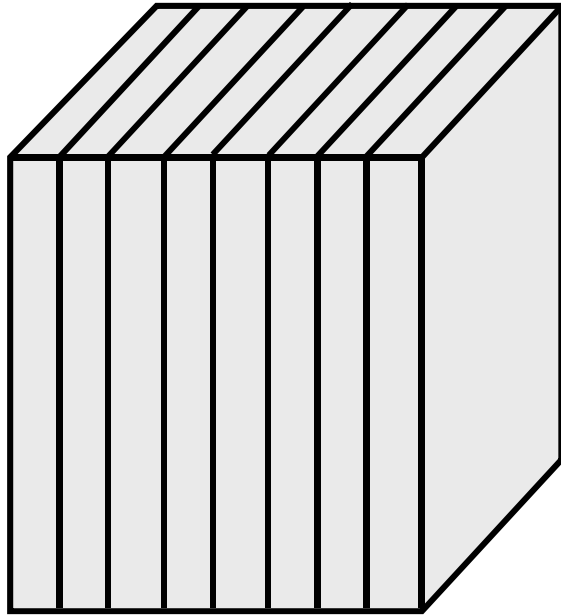
- Improve PE-to-PE communication part (solver\_SR)
  - Copying to receiving buffer, Combining MPI\_Wait\_all
- Actually, numbering of external nodes in each neighboring domain is continuous

```
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];  
    }  
}
```

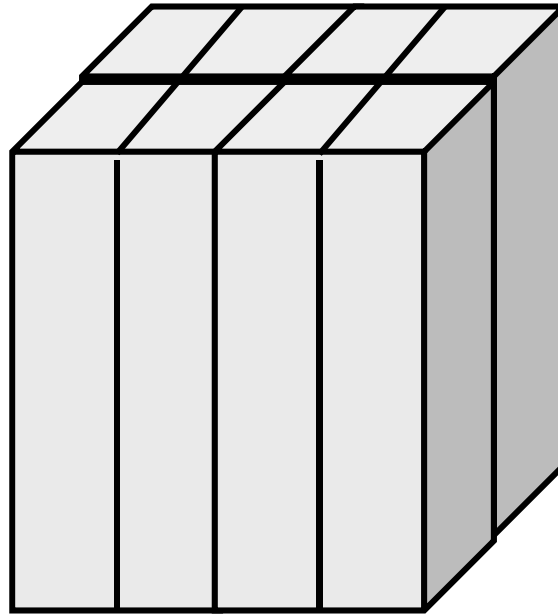
# Report S3 (3/3)

- Deadline: 17:00 October 12<sup>th</sup> (Sat), 2013.
  - Send files via e-mail at `nakajima(at)cc.u-tokyo.ac.jp`
- Report
  - Cover Page: Name, ID, and Problem ID (S3) must be written.
  - Less than ten pages including figures and tables (A4).
    - Strategy
    - Structure of the Program
    - Numerical Experiments, Performance Analysis
    - Remarks
  - Output list (as small as possible)

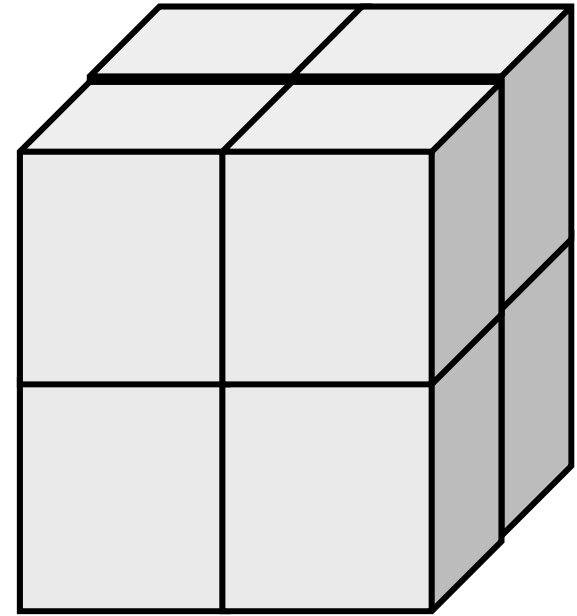
# 1D-3D Decomposition



1D



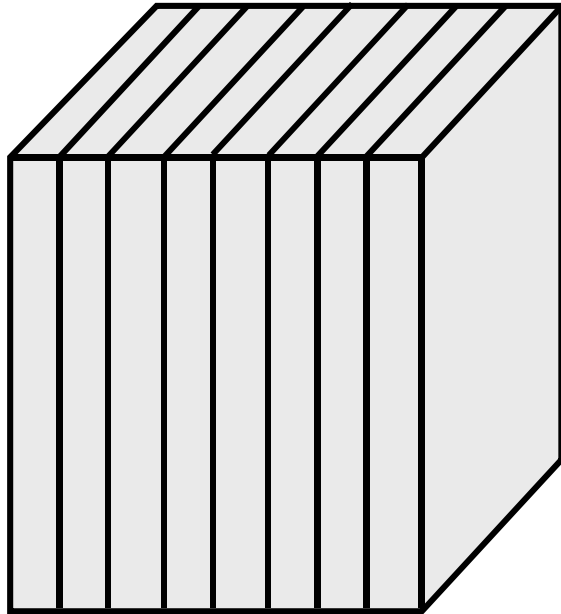
2D



3D

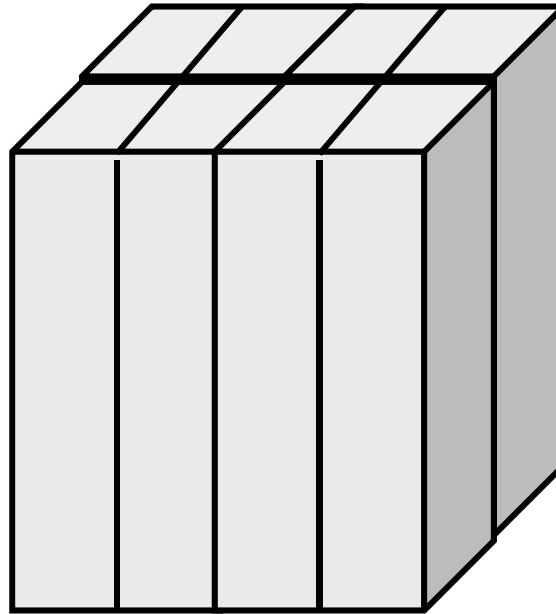
# 1D-3D Decomposition

Amount of comm.: each edge has  $4N$  points, 8 domains



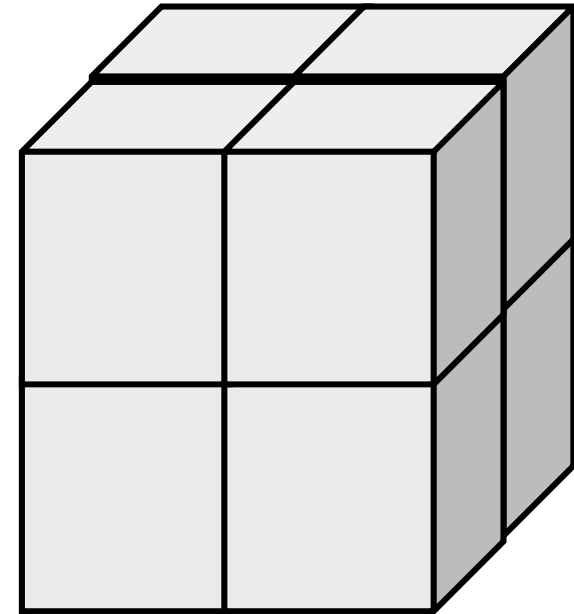
1D

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



2D

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

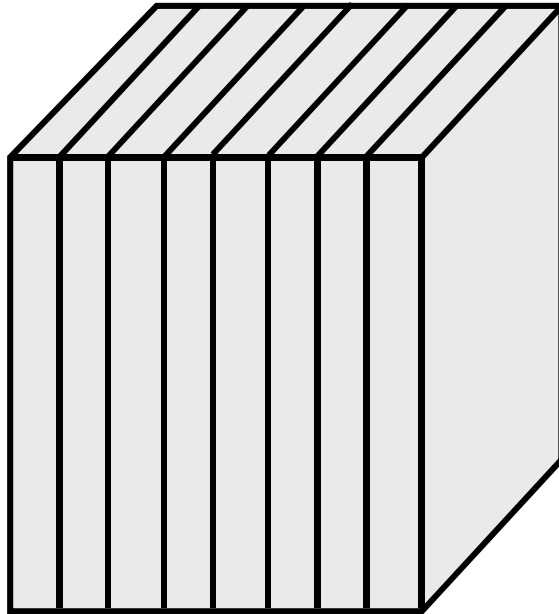


3D

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

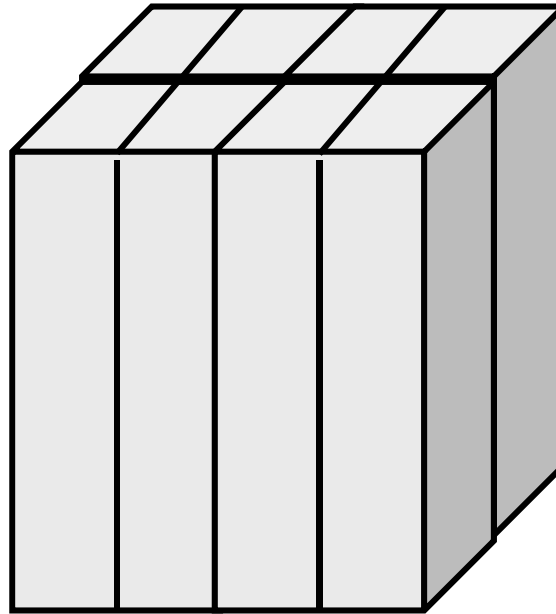
# 1D-3D Decomposition

inp\_rcb



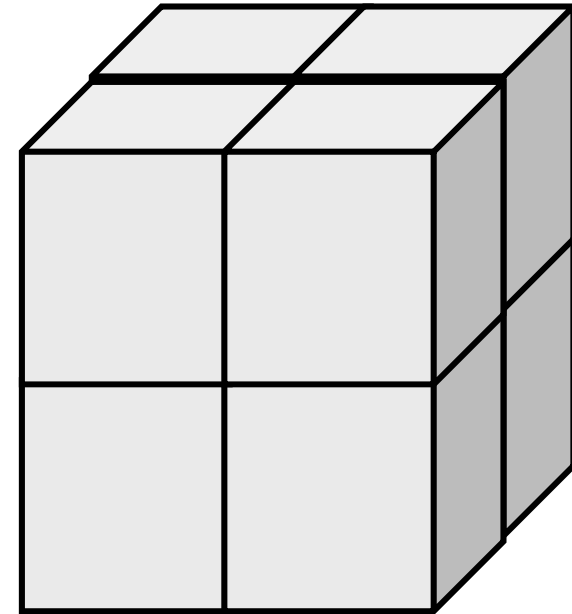
1D

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



2D

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



3D

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