

3D Parallel FEM (II)

C

Kengo Nakajima

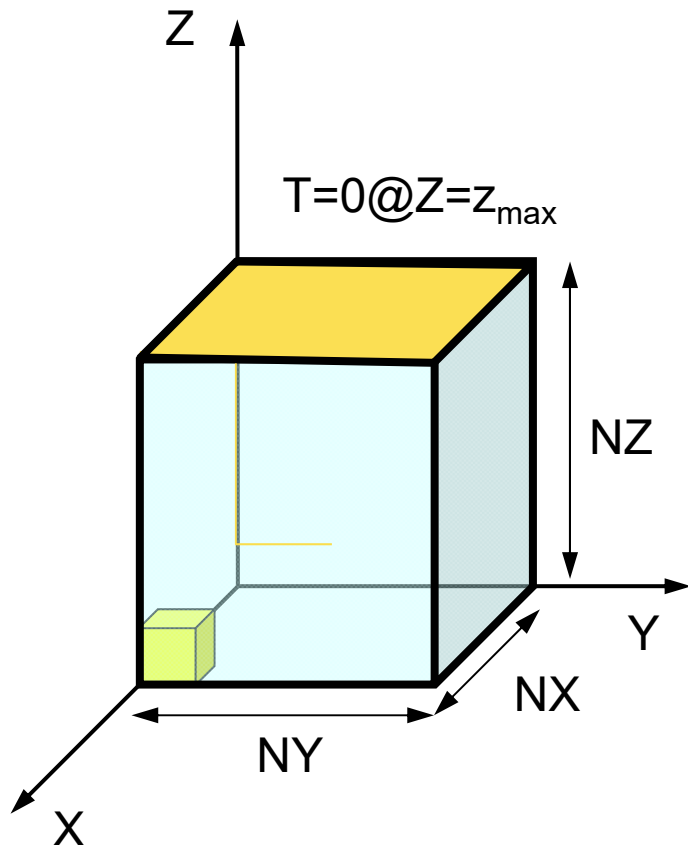
Technical & Scientific Computing II (4820-1028)

Seminar on Computer Science II (4810-1205)

Hybrid Distributed Parallel Computing (3747-111)

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

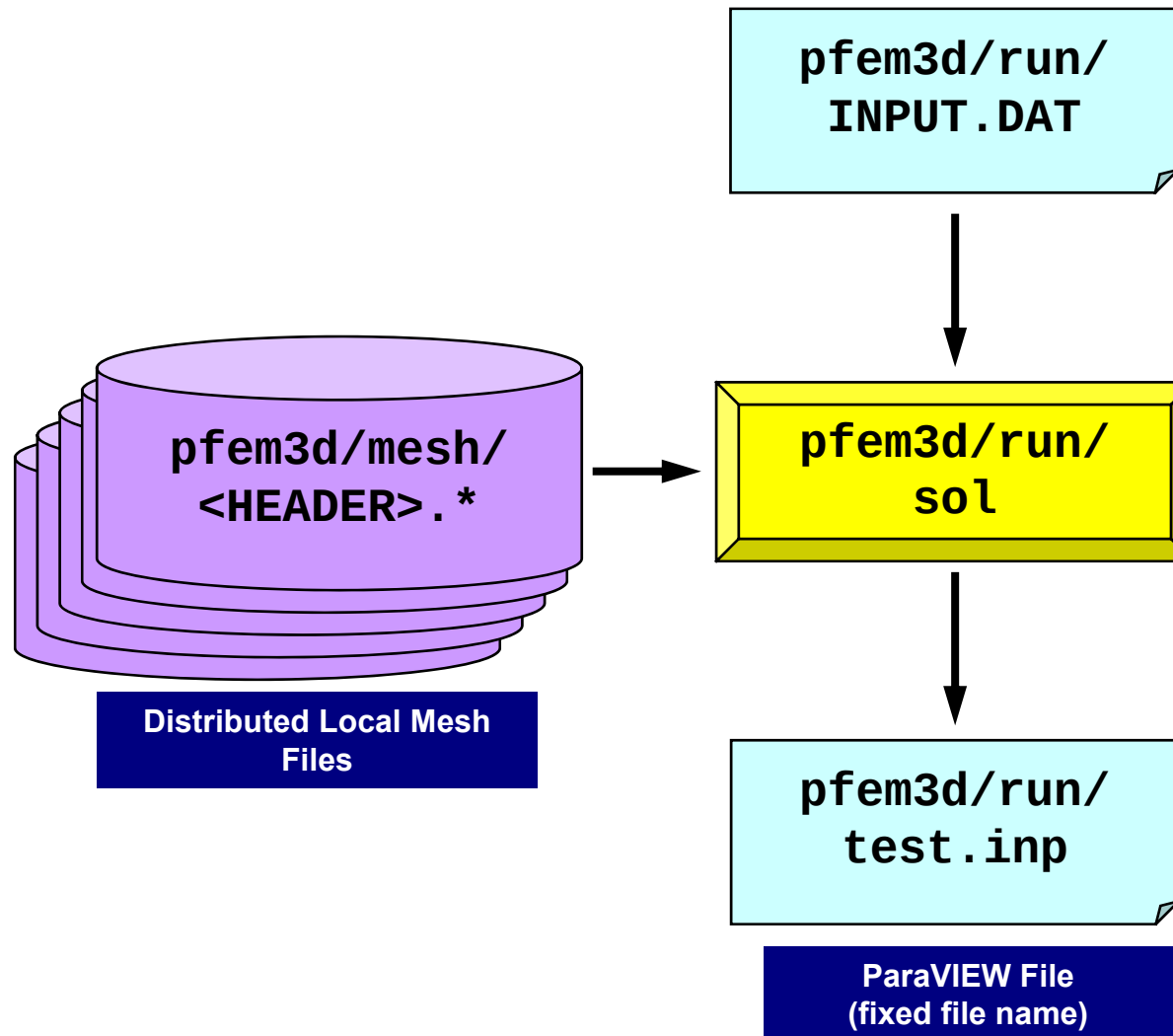
Finite-Element Procedures

- 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

more stable with “export I_MPI_PERHOST=32”

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

```
#PBS -q u-lecture
```

```
#PBS -N flat
```

```
#PBS -l select=4:mpiprocs=32
```

```
#PBS -Wgroup_list=gt16
```

```
#PBS -l walltime=00:05:00
```

```
#PBS -e err
```

```
#PBS -o test.lst
```

```
cd $PBS_O_WORKDIR
```

```
. /etc/profile.d/modules.sh
```

```
export I_MPI_PIN_DOMAIN=socket
```

```
export I_MPI_PERHOST=32
```

```
mpirun ./impimap.sh ./sol
```

Name of “QUEUE”

Job Name

node#, proc#/node

Group Name (Wallet)

Computation Time

Standard Error

Standard Outpt

go to current dir
(ESSENTIAL)

Execution on each socket

MPI proc#/node (=mpiprocs)

Exec's

```
#PBS -l select=1:mpiprocs=4
```

```
#PBS -l select=1:mpiprocs=16
```

```
#PBS -l select=1:mpiprocs=36
```

```
#PBS -l select=2:mpiprocs=32
```

```
#PBS -l select=4:mpiprocs=32
```

```
#PBS -l select=8:mpiprocs=36
```

1-node, 4-core's

1-node, 16-core's

1-node, 36-core's

2-nodes, 32x2=64-core's

4-nodes, 32x4=128-core's

8-nodes, 36x8=288-core's

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

more stable with “export I_MPI_PERHOST=32”

```
#!/bin/sh
#PBS -q u-lecture
#PBS -N flat
#PBS -l select=4:mpiprocs=32
#PBS -Wgroup_list=gt16
#PBS -l walltime=00:05:00
#PBS -e err
#PBS -o test.lst
```

Name of “QUEUE”
Job Name
node#, proc#/node
Group Name (Wallet)
Computation Time
Standard Error
Standard Outpt

```
cd $PBS_O_WORKDIR
. /etc/profile.d/modules.sh
```

go to current dir
(ESSENTIAL)

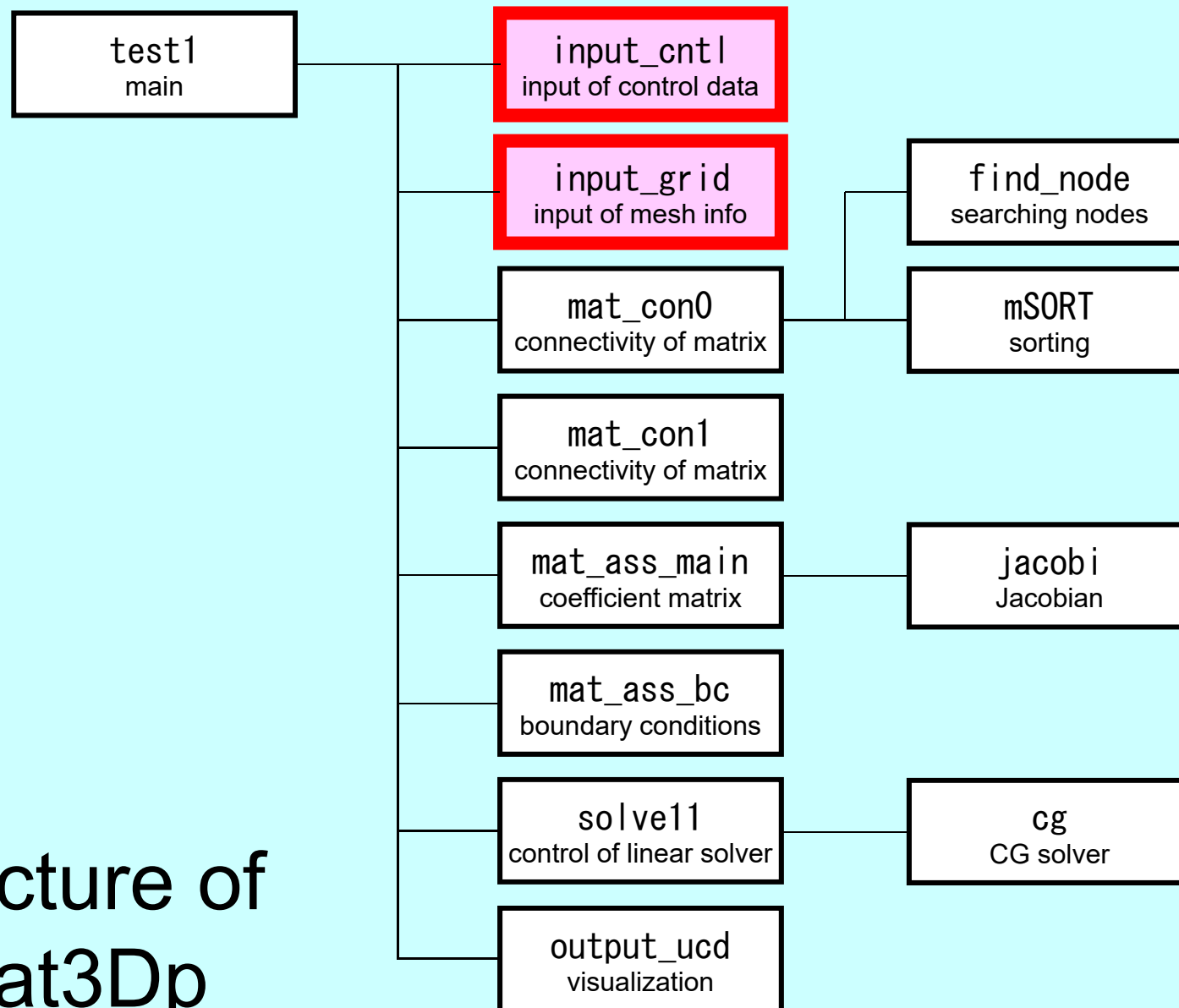
```
export I_MPI_PIN_DOMAIN=socket
export I_MPI_PERHOST=32
mpirun ./impimap.sh ./sol
```

Execution on each socket
MPI proc#/node (=mpiprocs)
Exec's

```
#PBS -l select=1:mpiprocs=4
#PBS -l select=1:mpiprocs=16
#PBS -l select=1:mpiprocs=36
#PBS -l select=2:mpiprocs=32
#PBS -l select=4:mpiprocs=32
#PBS -l select=8:mpiprocs=36
```

1-node, 4-core's
1-node, 16-core's
1-node, 36-core's
2-nodes, 32x2=64-core's
4-nodes, 32x4=128-core's
8-nodes, 36x8=288-core's

Structure of heat3Dp



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
fname	C	[80]	I	Name of mesh file
N, NP	I		I	# Node (N: Internal, NP: Internal + External)
ICELTOT	I		I	# Element
NODGRPtot	I		I	# Node Group
XYZ	R	[NP][3]	I	Node Coordinates
ICELNOD	I	[ICELTOT][8]	I	Element Connectivity
NODGRP_INDEX	I	[NODGRPtot+1]	I	# Node in each Node Group
NODGRP_ITEM	I	[NODGRP_INDEX[NODGRPtot+1]]	I	Node ID in each Node Group
NODGRP_NAME	C80	[NODGRP_INDEX[NODGRPtot+1]]	I	Name of NodeGroup
NLU	I		0	# Non-Zero Off-Diagonals at each node
NPLU	I		0	# Non-Zero Off-Diagonals
D	R	[NP]	0	Diagonal Block of Global Matrix
B, X	R	[NP]	0	RHS, Unknown Vector

Global Variables: pfem_util.h (2/4)

Name	Type	Size	I/O	Definition
AMAT	R	[NPLU]	0	Non-Zero Off-Diagonal Components of Global Matrix
indexLU	I	[NP+1]	0	# Non-Zero Off-Diagonal Components
itemLU	I	[NPLU]	0	Column ID of Non-Zero Off-Diagonal Components
INLU	I	[NP]	0	Number of Non-Zero Off-Diagonal Components at Each Node
IALU	I	[NP][NLU]	0	Column ID of Non-Zero Off-Diagonal Components at Each Node
IWKX	I	[NP][2]	0	Work Arrays
ITER, ITERactual	I		I	Number of CG Iterations (MAX, Actual)
RESID	R		I	Convergence Criteria (fixed as 1.e-8)
pfemIarray	I	[100]	0	Integer Parameter Array
pfemRarray	R	[100]	0	Real Parameter Array

Global Variables: pfem_util.h (3/4)

Name	Type	Size	I/O	Definition
o8th	R		I	= 0.125
PNQ, PNE, PNT	R	[2][2][8]	0	$\frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta} (i=1 \sim 8)$ at each Gaussian Quad. Point
POS, WEI	R	[2]	0	Coordinates, Weighting Factor at each Gaussian Quad. Point
NCOL1, NCOL2	I	[100]	0	Work arrays for sorting
SHAPE	R	[2][2][2][8]	0	$N_i (i=1 \sim 8)$ at each Gaussian Quad Point
PNX, PNY, PNZ	R	[2][2][2][8]	0	$\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z} (i=1 \sim 8)$ at each Gaussian Quad. Point
DETJ	R	[2][2][2]	0	Determinant of Jacobian Matrix at each Gaussian Quad. Point
COND, QVOL	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
PETOT	I		0	Number of PE's
my_rank	I		0	Process ID of MPI
errno	I		0	Error Flag
NEIBPETOT	I		I	Number of Neighbors
NEIBPE	I	[NEIBPETOT]	I	ID of Neighbor
IMPORT_INDEX EXPORT_INEDX	I	[NEIBPETOT+1]	I	Size of Import/Export Arrays for Communication Table
IMPORT_ITEM	I	[NPimport]	I	Receiving Table (External Points) NPimport=IMPORT_INDEX[NEIBPETOT+1])
EXPORT_ITEM	I	[NPexport]	I	Sending Table (Boundary Points) NPexport=EXPORT_INDEX[NEIBPETOT+1])
ICELTOT_INT	I		I	Number of Local Elements
intELEM_list	I	[ICELTOT_INT]	I	List of Local Elements

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

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",
            &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]);

```

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]);}
    }

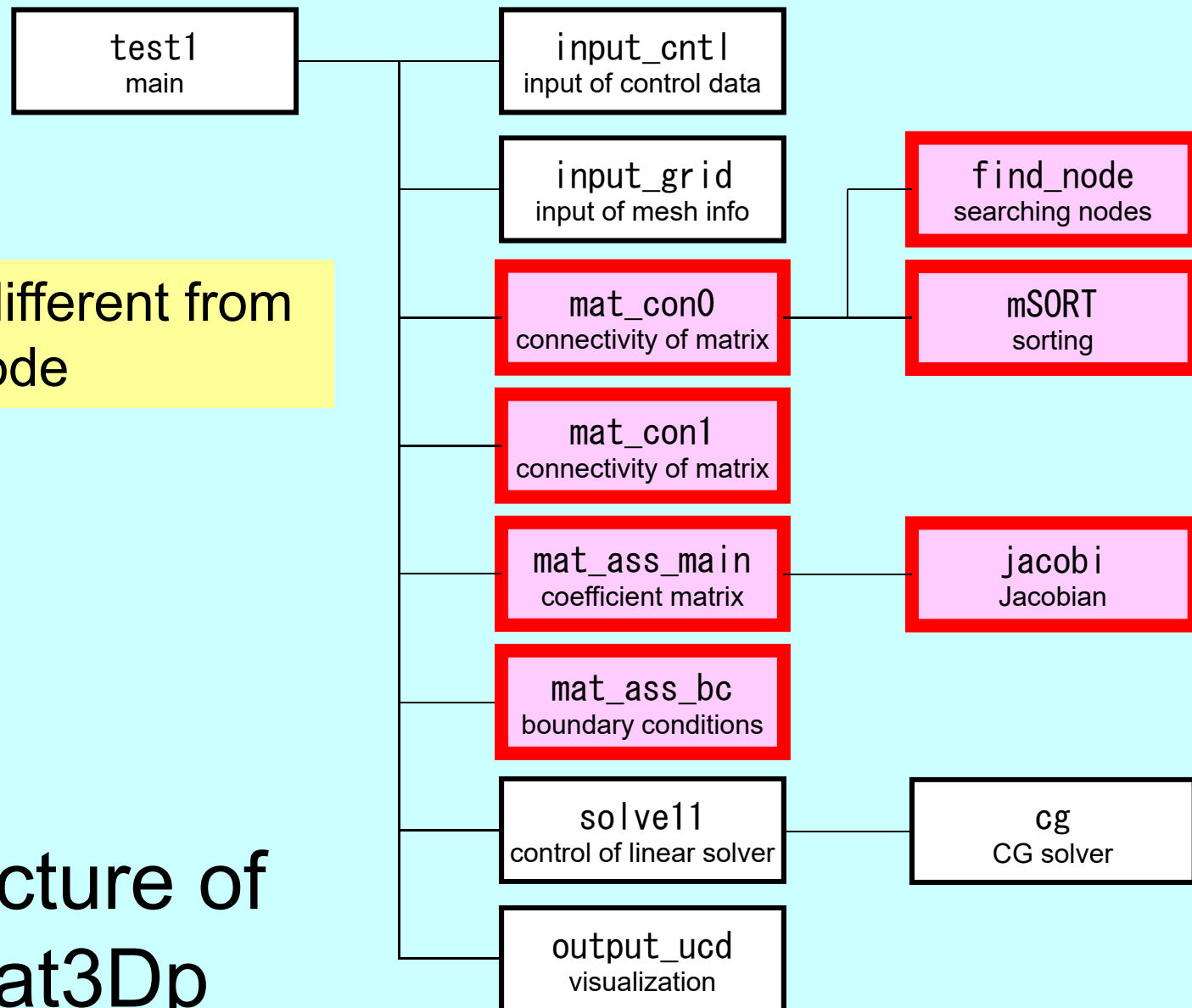
```

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



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_CON0();
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_CON0();
    MAT_CON1();

    MAT_ASS_MAIN();
    MAT_ASS_BC();

    SOLVE11();

    OUTPUT_UCD();
    PFEM_FINALIZE();
}

```

MAT_CON0: generates INU, IALU

MAT_CON1: generates index, item

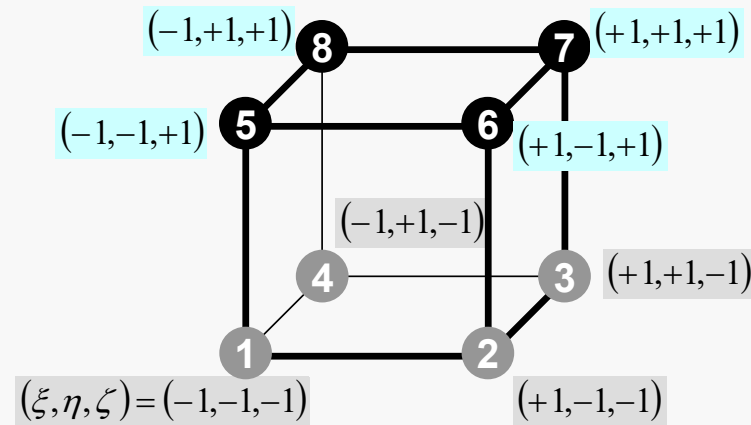
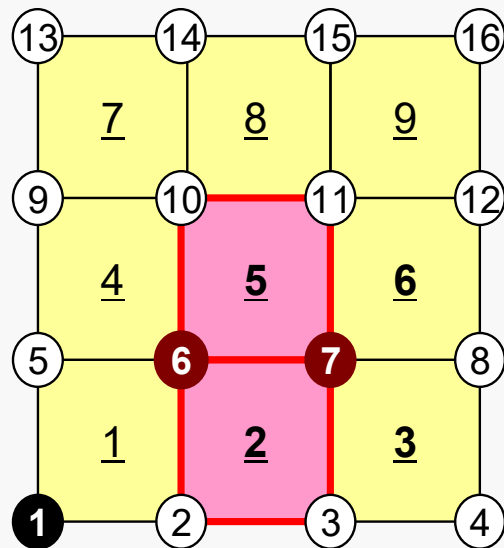
Node ID starting from “1”

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_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;

```

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

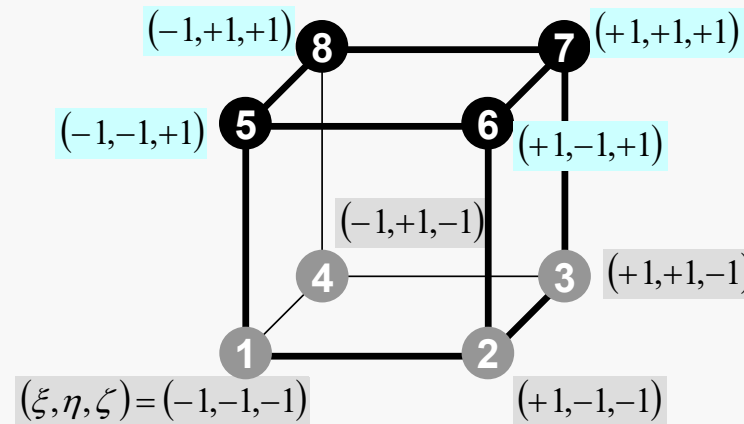
```
for( ice|=0; ice|< ICELTOT; ice|++ ) {
```

```
  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);
```



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;

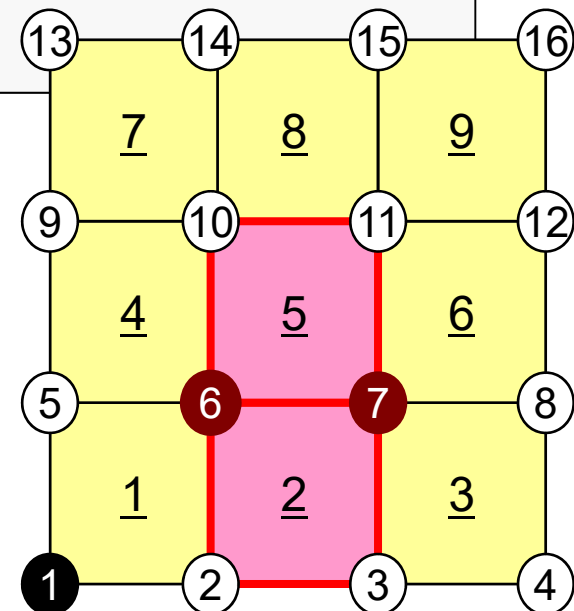
    for (kk=1; kk<=INLU[ip1-1]; kk++) {
        if (ip2 == IALU[ip1-1][kk-1]) return;
    }

    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;

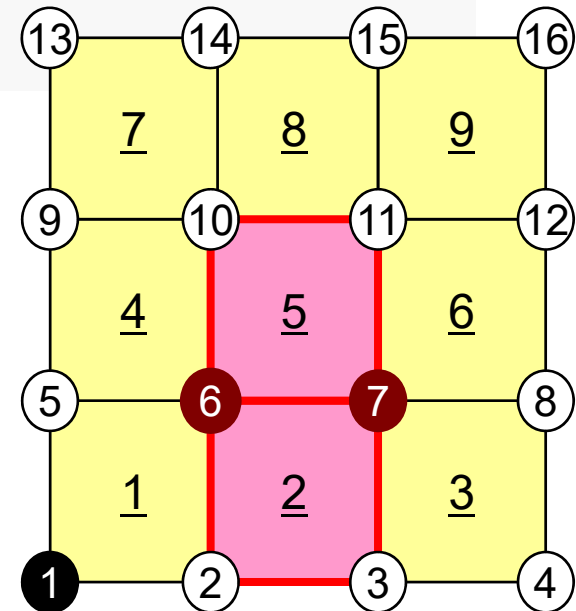
    for (kk=1;kk<=INLU[ip1-1];kk++) {
        if(ip2 == IALU[ip1-1][kk-1]) return;
    }

    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);

```

```

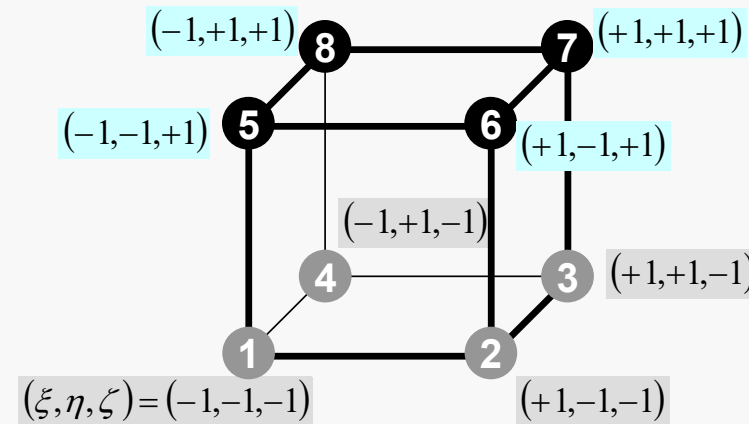
FIND_TS_NODE (in6, in1);
FIND_TS_NODE (in6, in2);
FIND_TS_NODE (in6, in3);
FIND_TS_NODE (in6, in4);
FIND_TS_NODE (in6, in5);
FIND_TS_NODE (in6, in7);
FIND_TS_NODE (in6, in8);

```

```

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: Overview

```

do kpn= 1, 2      Gaussian Quad. points in  $\zeta$ -direction
  do jpn= 1, 2    Gaussian Quad. points in  $\eta$ -direction
    do ipn= 1, 2  Gaussian Quad. Points in  $\xi$ -direction
      Define Shape Function at Gaussian Quad. Points (8-points)
      Its derivative on natural/local coordinate is also defined.
    enddo
  enddo
enddo

```

```

do icel= 1, ICELTOT  Loop for Element
  Jacobian and derivative on global coordinate of shape functions at
  Gaussian Quad. Points are defined according to coordinates of 8 nodes. (JACOBI)

```

```

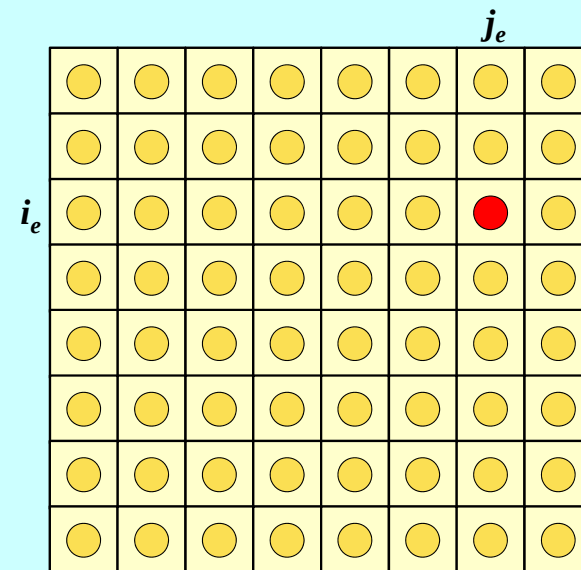
do ie= 1, 8          Local Node ID
  do je= 1, 8        Local Node ID
    Global Node ID: ip, jp
    Address of  $A_{ip, jp}$  in "itemLU" : kk

```

```

    do kpn= 1, 2      Gaussian Quad. points in  $\zeta$ -direction
      do jpn= 1, 2    Gaussian Quad. points in  $\eta$ -direction
        do ipn= 1, 2  Gaussian Quad. points in  $\xi$ -direction
          integration on each element
          coefficients of element matrices
          accumulation to global matrix
        enddo
      enddo
    enddo
  enddo
enddo
enddo

```



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;
```

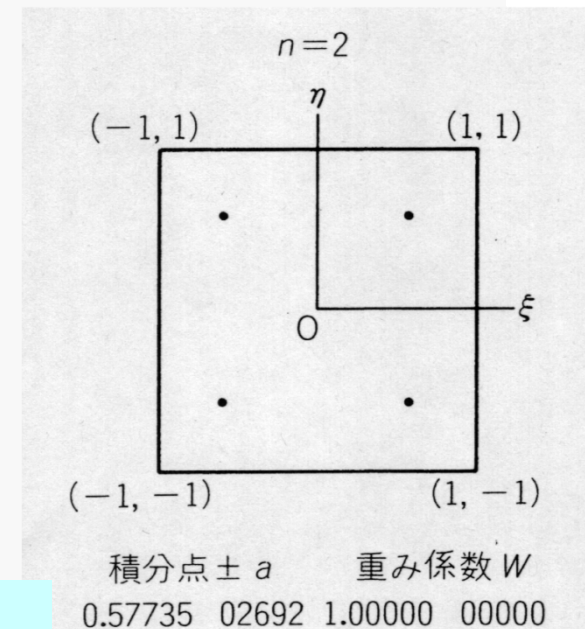
```
KINT nodLOCAL[8];
```

```
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++) {

```

```

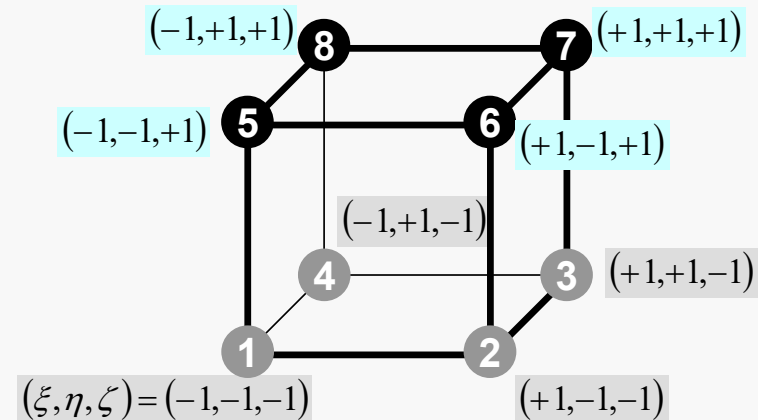
      QP1= 1. e0 + POS[ip];
      QM1= 1. e0 - POS[ip];
      EP1= 1. e0 + POS[jp];
      EM1= 1. e0 - POS[jp];
      TP1= 1. e0 + POS[kp];
      TM1= 1. e0 - POS[kp];

```

```

      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
(ξ_i, η_j, ζ_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;

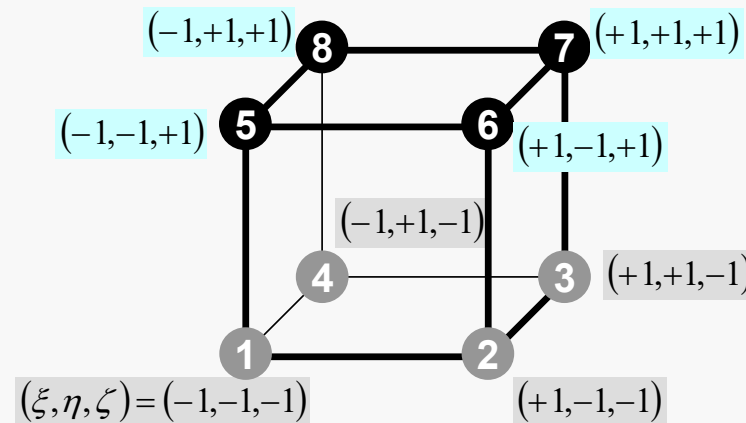
PNE[ip][kp][0] = - 08th * QM1 * TM1;
PNE[ip][kp][1] = - 08th * QP1 * TM1;
PNE[ip][kp][2] = + 08th * QP1 * TM1;
PNE[ip][kp][3] = + 08th * QM1 * TM1;
PNE[ip][kp][4] = - 08th * QM1 * TP1;
PNE[ip][kp][5] = - 08th * QP1 * TP1;
PNE[ip][kp][6] = + 08th * QP1 * TP1;
PNE[ip][kp][7] = + 08th * QM1 * TP1;

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];

```



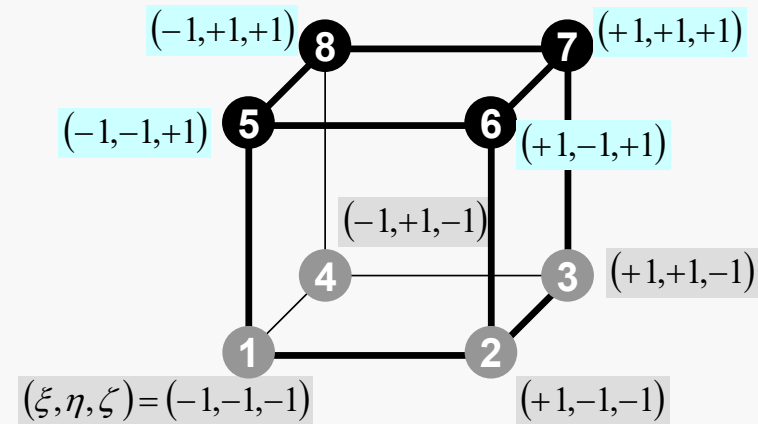
MAT_ASS_MAIN (4/6)

```

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

```

Node ID (Global)



```

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 (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];
```

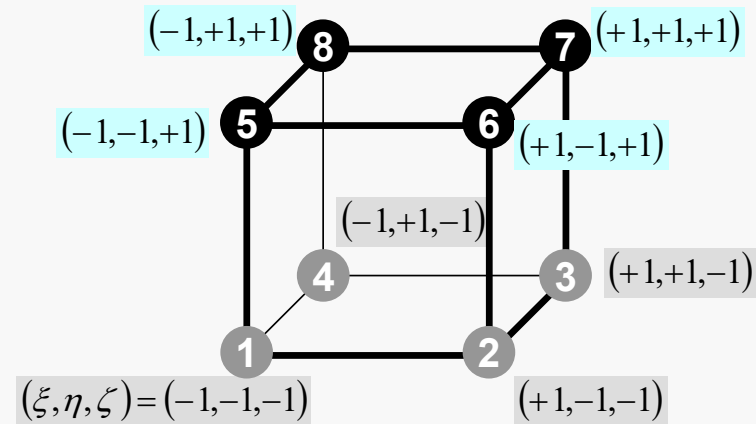
Y-Coordinates
of 8 nodes

```
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

```
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);
```



Coordinates:
Node ID - 1

MAT_ASS_MAIN (4/6)

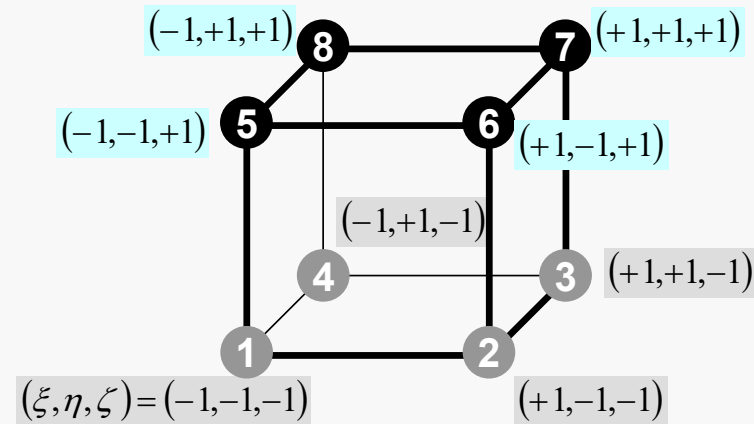
```
nodLOCAL[0]= in1;
nodLOCAL[1]= in2;
nodLOCAL[2]= in3;
nodLOCAL[3]= in4;
nodLOCAL[4]= in5;
nodLOCAL[5]= in6;
nodLOCAL[6]= in7;
nodLOCAL[7]= in8;
```

```
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



Coordinates:
Node ID - 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];
```

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

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

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

```
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);
```

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;
```

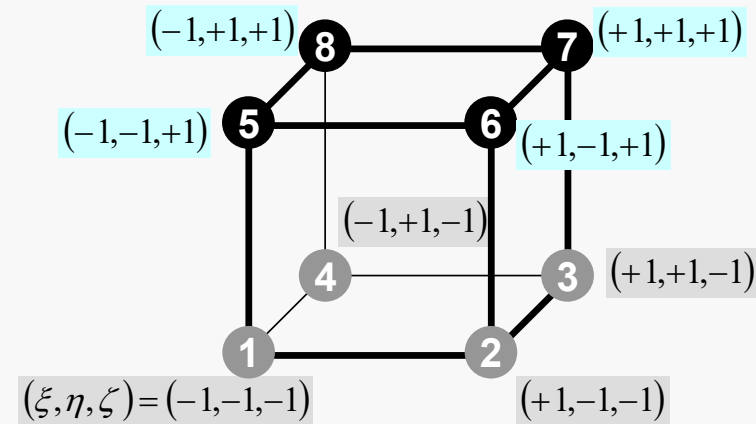
```
X1=XYZ[in1-1][0];
X2=XYZ[in2-1][0];
X3=XYZ[in3-1][0];
X4=XYZ[in4-1][0];
X5=XYZ[in5-1][0];
X6=XYZ[in6-1][0];
X7=XYZ[in7-1][0];
X8=XYZ[in8-1][0];
```

```
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, 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);
```

MAT_ASS_MAIN (5/6)

```

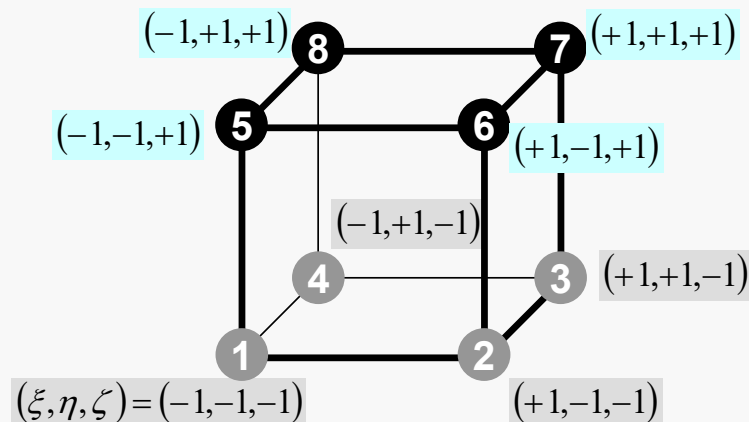
/**
CONSTRUCT the GLOBAL MATRIX
**/

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

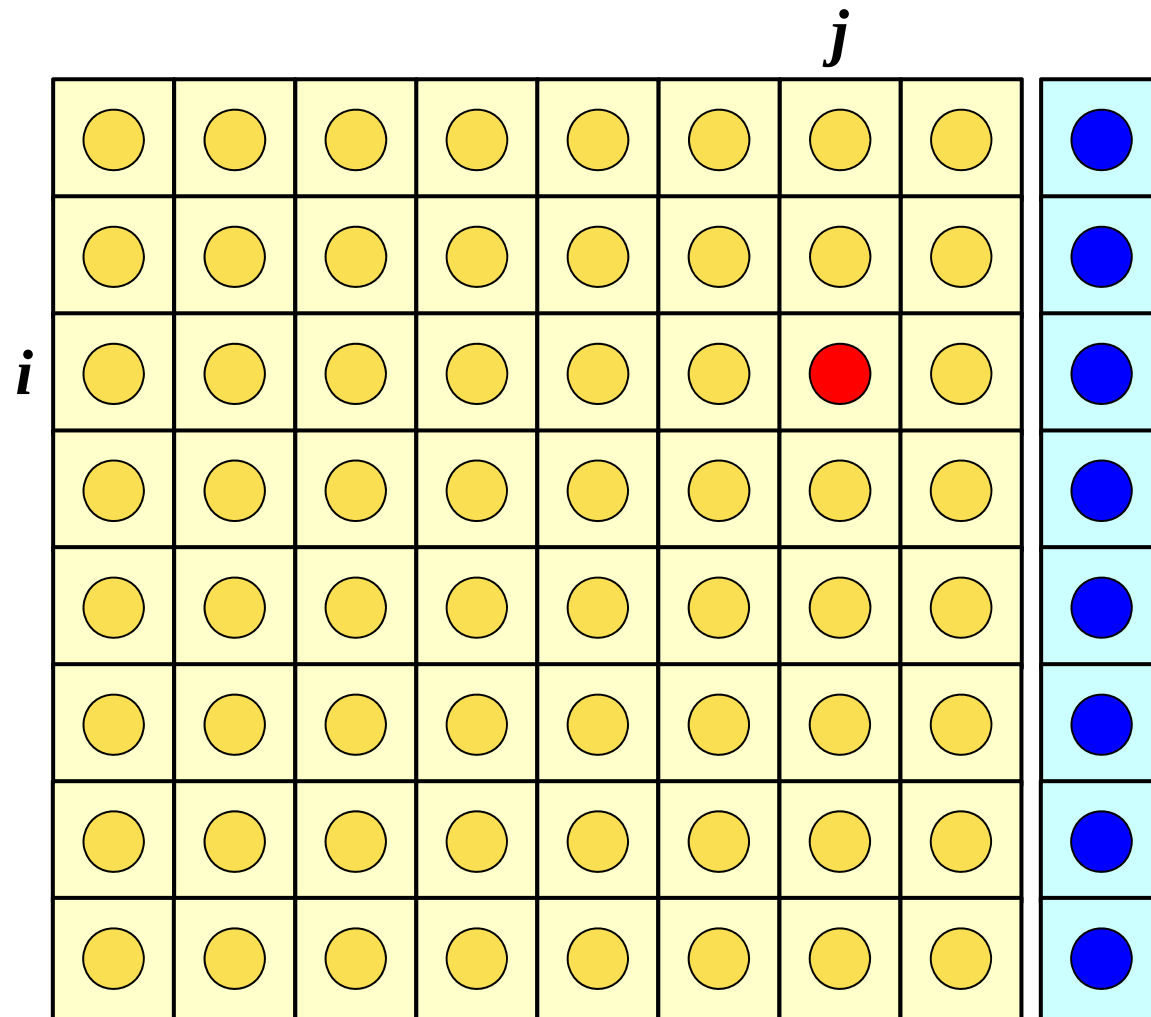
$$A_{ip, jp}$$

kk: address in “itemLU”

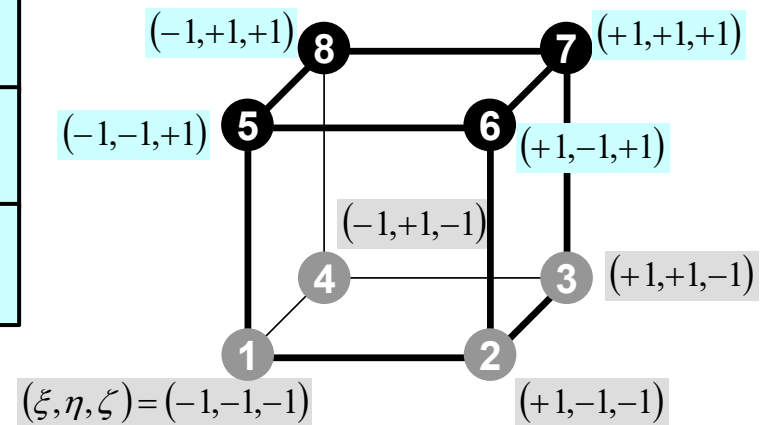
ip= nodLOCAL[ie]
jp= nodLOCAL[je]

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

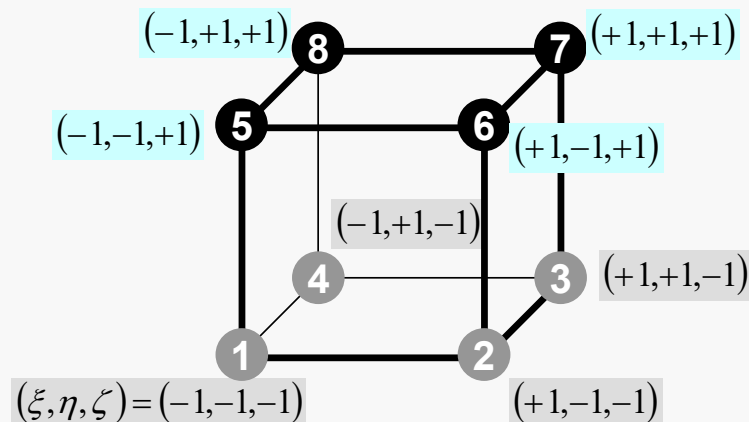
```

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*COND0*(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;
}
}
}
}
}

```

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

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

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

```

```

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

```

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

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

```

    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;
}

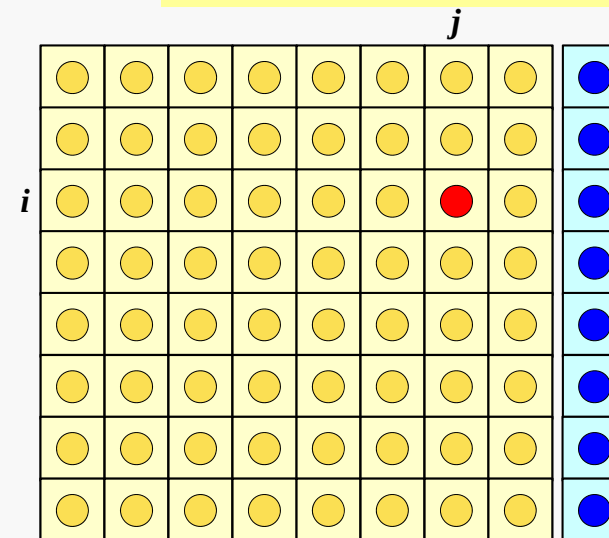
```

```

}
}
}
}

```

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



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;
}
}
}
}

```

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

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

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

$$QVC = |x_C + y_C|$$

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

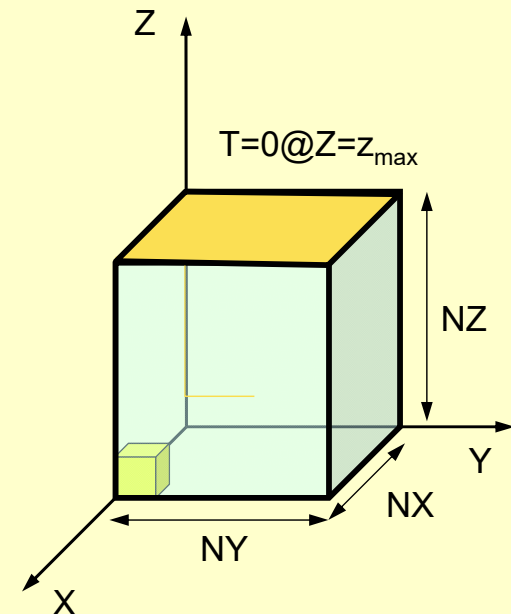
$$[f]^{(e)} = QV0 \cdot QVC$$

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(indexLU(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)

```

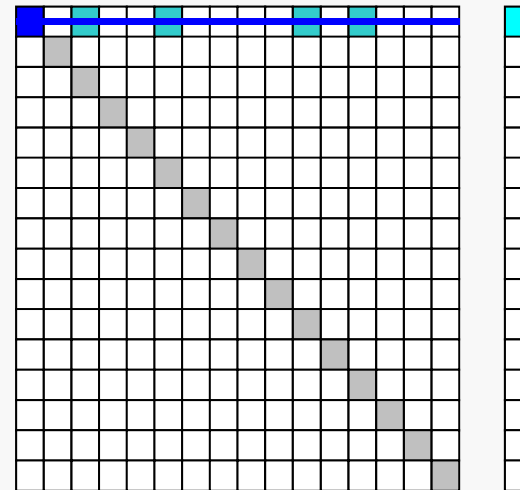
for(in=0; in<NP; in++) {
  if( IWKX[in][0] == 1 ) {
    B[in]= 0. e0;
    D[in]= 1. e0;
    for(k=indexLU[in]; k<indexLU[in+1]; k++) {
      AMAT[k]= 0. e0;
    }
  }
}

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

Erase !!



Same as 1CPU case

MAT_ASS_BC (2/2)

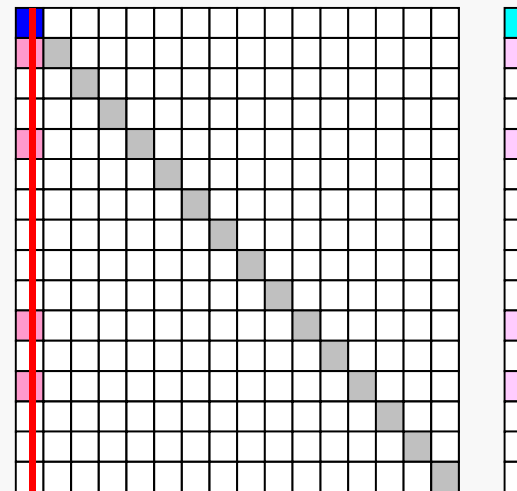
```

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



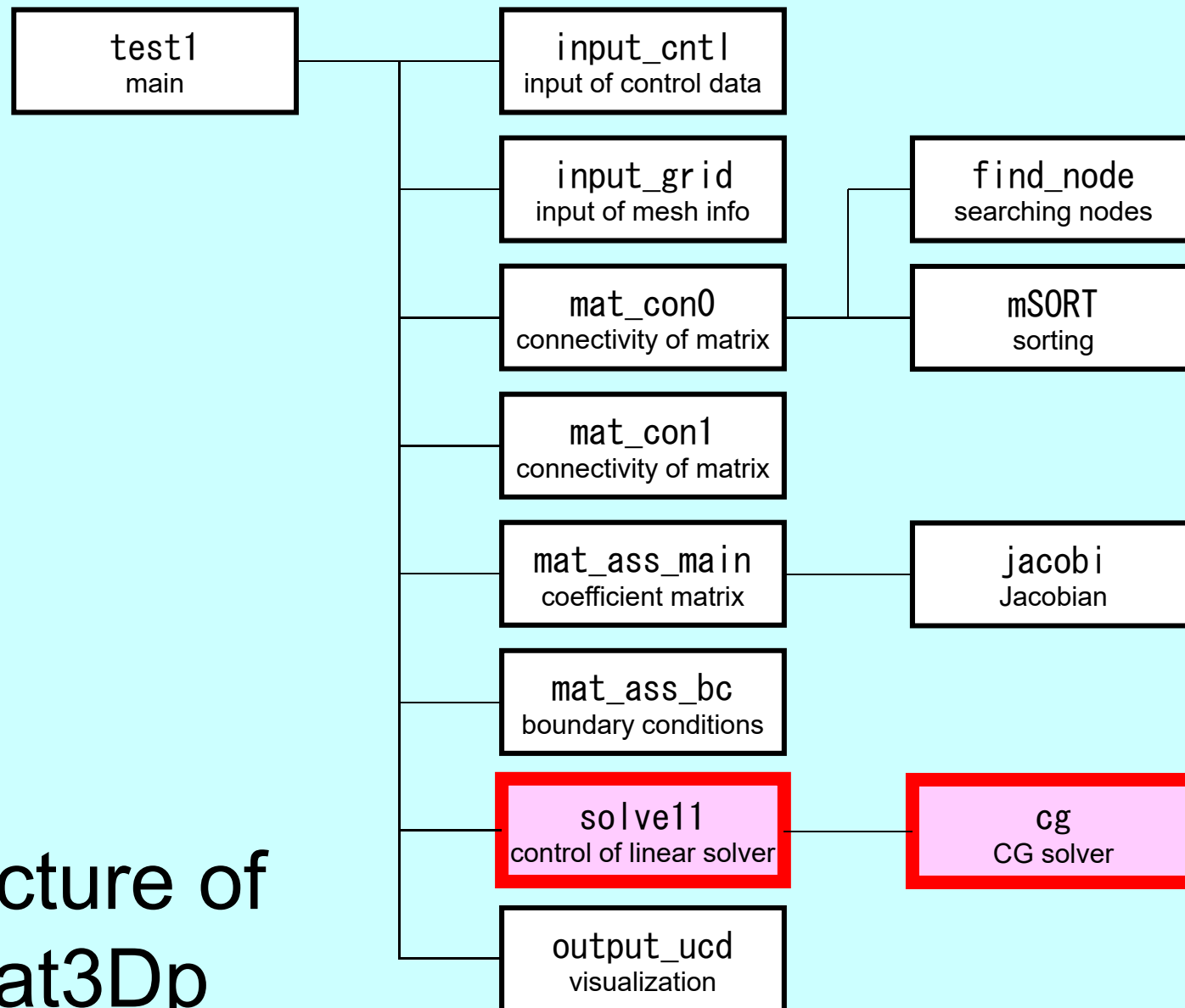
Elimination and Erase

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**

Structure of heat3Dp



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, ...
  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} |
  |-----|
  */
  RH00= 0. e0;
  for (i=0; i<N; i++) {
    RH00+= WW[R][i]*WW[Z][i];
  }
  MPI_Allreduce (&RH00, &RHO, 1, MPI_DOUBLE, MPI_SUM,
    MPI_COMM_WORLD);

  /**
  | {p} = {z} if      ITER=1
  | BETA= RHO / RH01 otherwise
  |-----|
  */
  if ( ITER == 1 ) {
    for (i=0; i<N; i++) {
      WW[P][i]=WW[Z][i];
    }
  } else {
    BETA= RHO / RH01;
    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 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;

RH01 = RHO ;
}

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

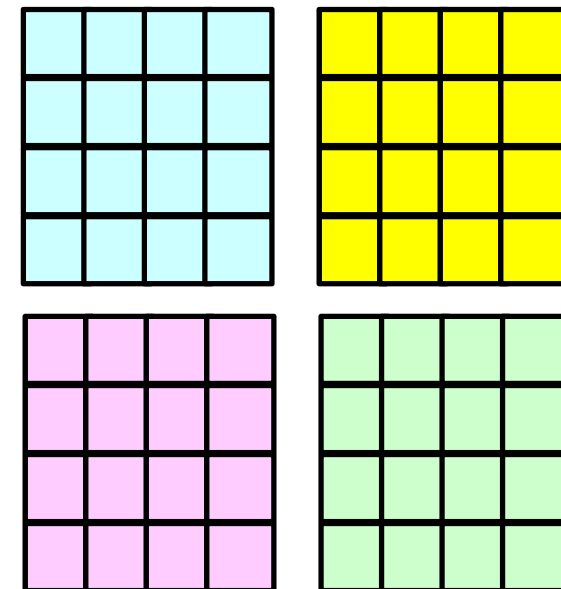
free ( (KREAL**)WW);
deallocate_vector ( (KREAL**)WR);
deallocate_vector ( (KREAL**)WS);
}

```

Updated temperature for external nodes

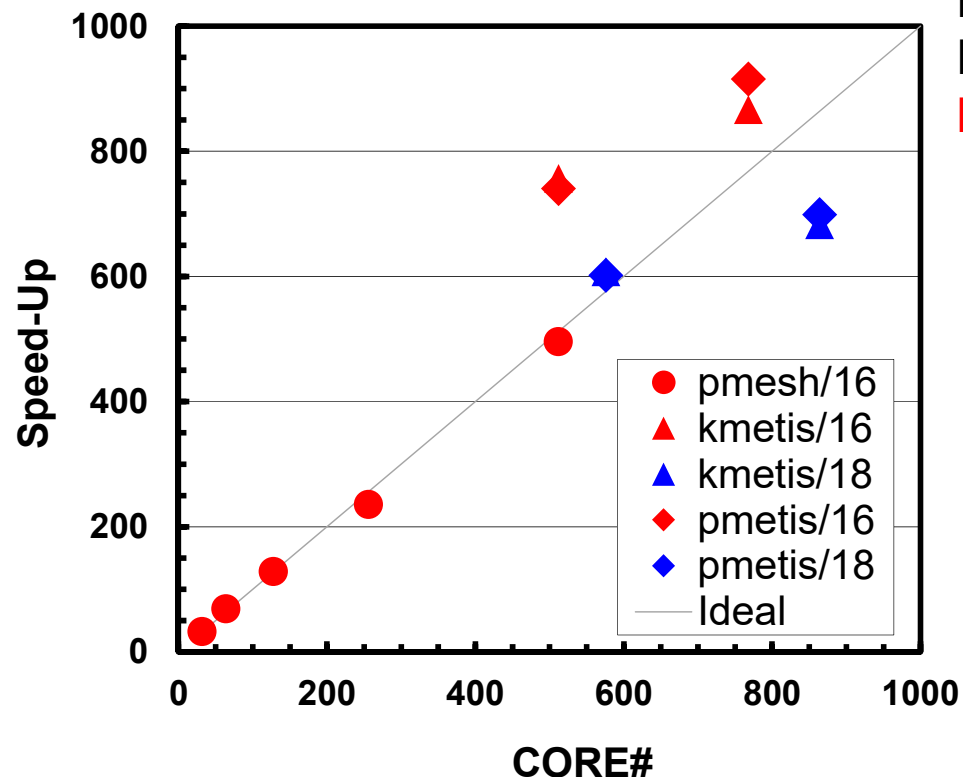
OUTPUT_UCD for Visualization

- Gather information of elements in “intELEM_list” on each process
- Gather the following information to process #0 using MPI_Allgatherv
 - 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: Strong Scaling: Fortran

- $256 \times 128 \times 128$ nodes
 - 4,194,304 nodes, 4,112,895 elements
- 32~864 cores
- Linear Solver



- pmesh/16: pmesh 16 cores/socket
- kmetis/16: kmetis 16 cores/socket
- kmetis/18: kmetis 18 cores/socket
- pmetis/16: pmetis 16 cores/sockets
- pmetis/18: pmetis 18 cores/sockets

Performance of pmesh/16 w/32 cores= 32.0

```
256 128 128
  8   2   2
pcube
```

```
select=1:
mpiprocs=32
```

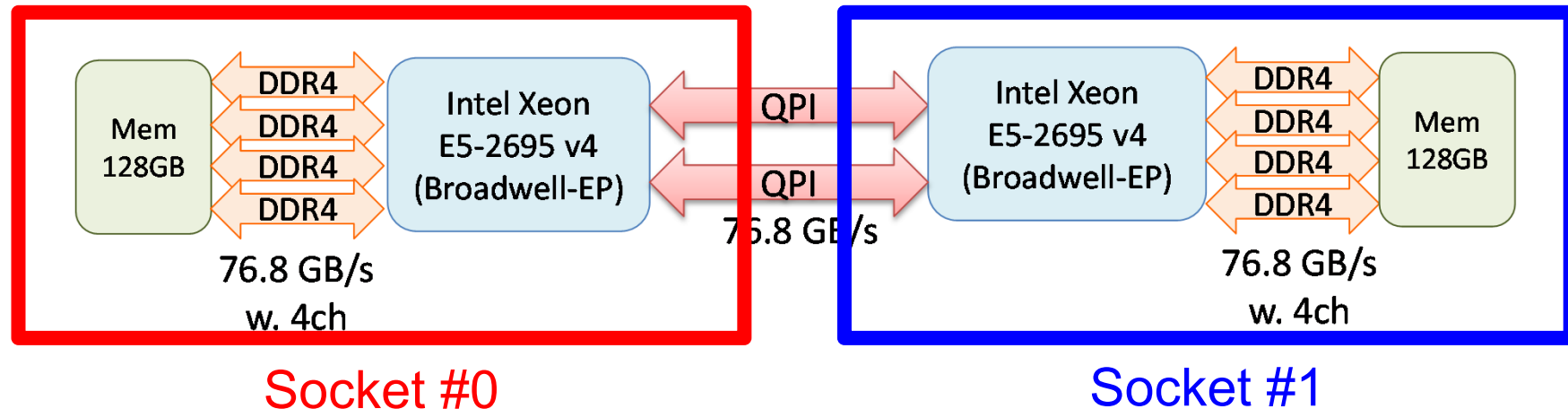
```
256 128 128
  8   4   2
pcube
```

```
select=2:
mpiprocs=32
```

```
256 128 128
 16   4   4
pcube
```

```
select=8:
mpiprocs=32
```

1 Node = 2 Sockets (CPU's)

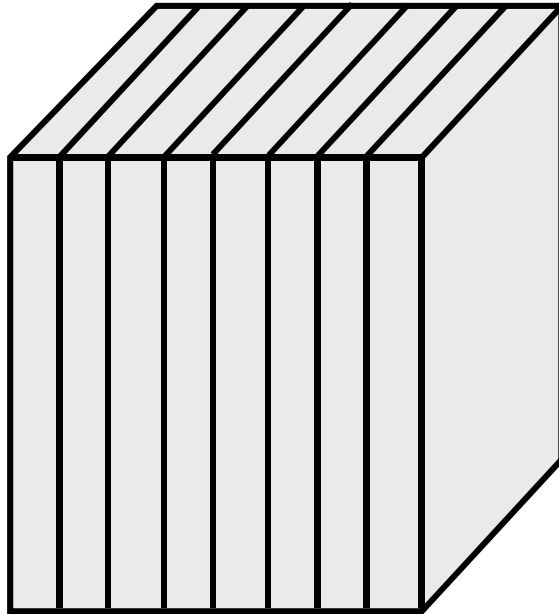


- Each Node of Reedbush-U
 - 2 Sockets (CPU's) of Intel Broadwell-EP
 - Each socket has 18 cores
- Each core of a socket can access to the memory on the other socket : NUMA (Non-Uniform Memory Access)
 - `I_MPI_PIN_DOMAIN=socket`, `impimap.sh`: local memory to be used

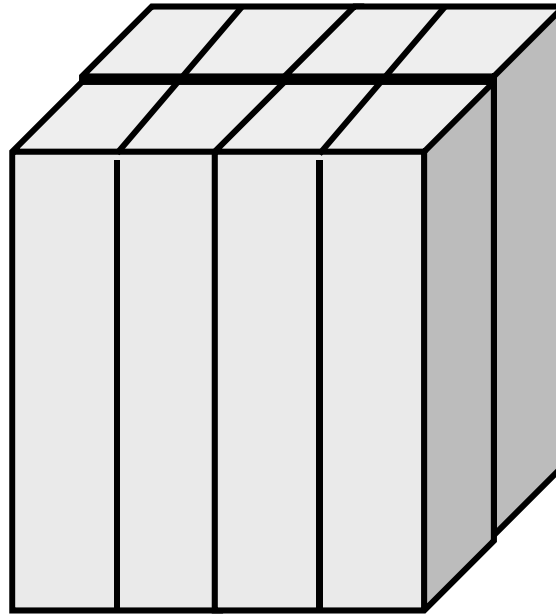
Exercise (1/2)

- 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 (1D-3D, kmetis, pmetis)
- “*.inp” may take long time.
 - delete “call OUTPUT_UCD”
 - src, part

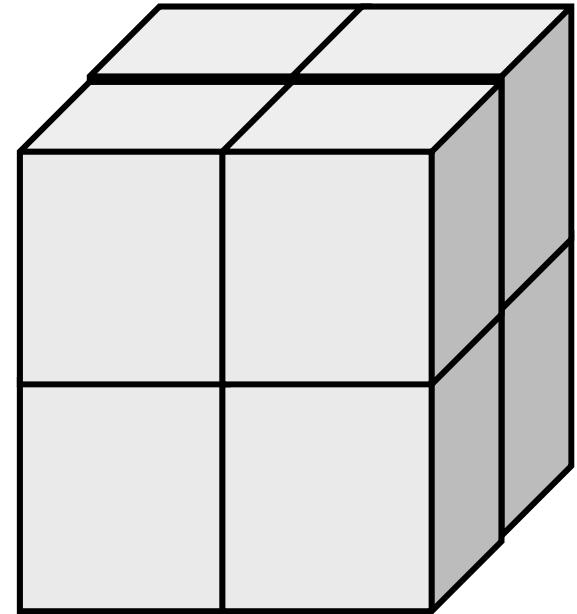
1D-3D Decomposition



1D



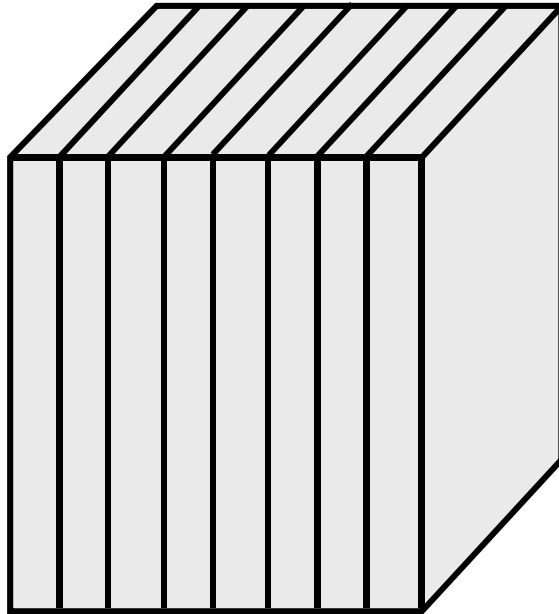
2D



3D

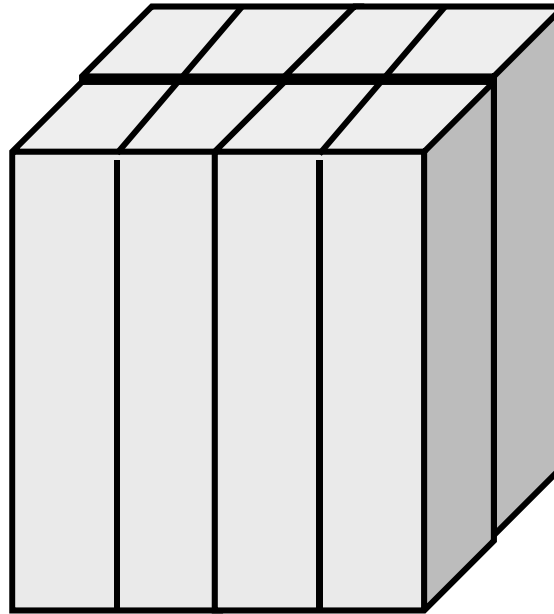
1D-3D Decomposition

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



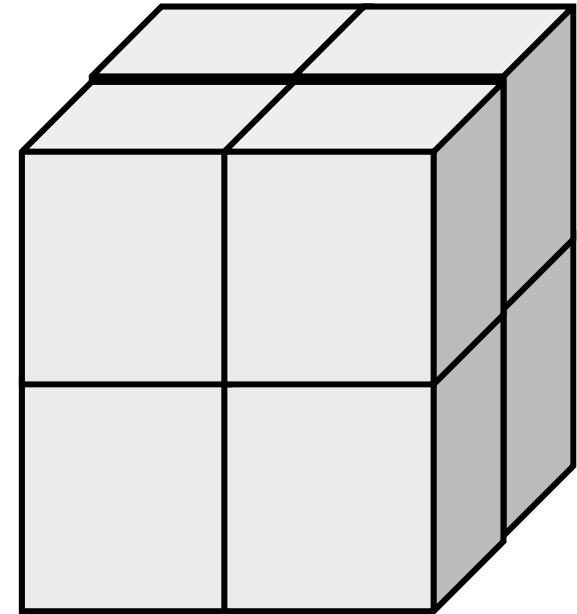
1D

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



2D

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

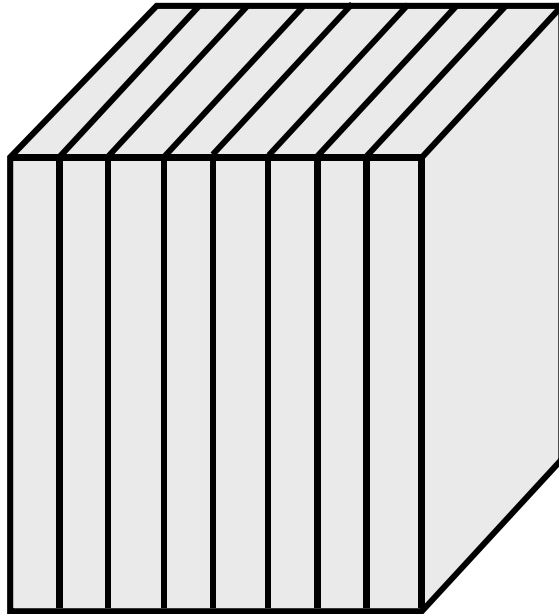


3D

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

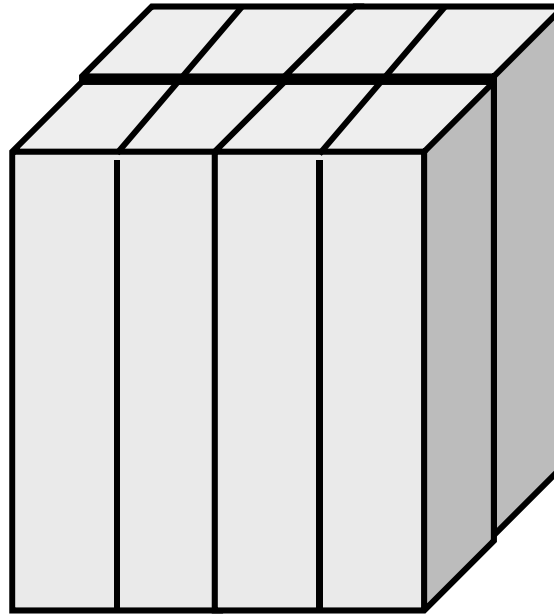
1D-3D Decomposition

mesh.inp



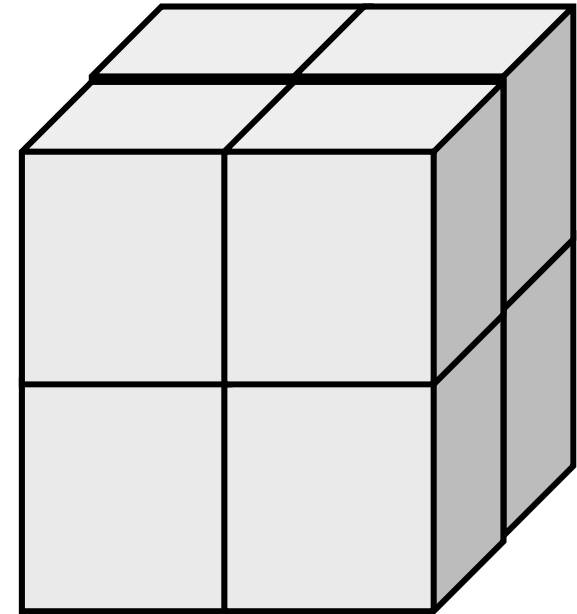
1D

64	64	64
8	1	1
pcube		



2D

64	64	64
4	2	1
pcube		



3D

64	64	64
2	2	2
pcube		

Exercise (2/2)

- 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];
    }
}

```

SEND/RECV (Original)

```

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) ;

```

```

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]);}

```

```

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);

```


If numbering of external nodes is continuous in each neighboring process ...

	84	81	85	82	83	86	88	87	
96	57	58	59	60	61	62	63	64	73
95	49	50	51	52	53	54	55	56	74
94	41	42	43	44	45	46	47	48	80
93	33	34	35	36	37	38	39	40	79
92	25	26	27	28	29	30	31	32	78
91	17	18	19	20	21	22	23	24	77
90	9	10	11	12	13	14	15	16	76
89	1	2	3	4	5	6	7	8	75
	65	66	67	68	69	70	71	72	

SEND/RECV (NEW:1)

```

sta1=(MPI_Status*) allocate_vector (sizeof (MPI_Status) , 2*NEIBPETOT) ;
req1=(MPI_Request*) allocate_vector (sizeof (MPI_Request) , 2*NEIBPETOT) ;

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]);

for ( neib=1;neib<=NEIBPETOT;neib++) {
    istart=IMPORT_INDEX[neib-1];
    inum  =IMPOERT_INDEX[neib] - istart;
    MPI_Irecv(&X[istart], inum, MPI_DOUBLE,
             NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req1[NEIBPETOT+neib-1]);}

MPI_Waitall (2*NEIBPETOT, req1, sta1);

```

SEND/RECV (NEW:2), N0: int. node

```
sta1=(MPI_Status*) allocate_vector (sizeof (MPI_Status), 2*NEIBPETOT);  
req1=(MPI_Request*) allocate_vector (sizeof (MPI_Request), 2*NEIBPETOT);
```

```
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];}
```

N0: 内点総数

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

```
for ( neib=1;neib<=NEIBPETOT;neib++) {  
    istart=IMPORT_INDEX[neib-1] + N0;  
    inum  =IMPORT_INDEX[neib] - IMPORT_INDEX[neib-1];  
    MPI_Irecv(&X[istart], inum, MPI_DOUBLE,  
             NEIBPE[neib-1], 0, MPI_COMM_WORLD, &req1[NEIBPETOT+neib-1]);}
```

```
MPI_Waitall (2*NEIBPETOT, req1, sta1);
```