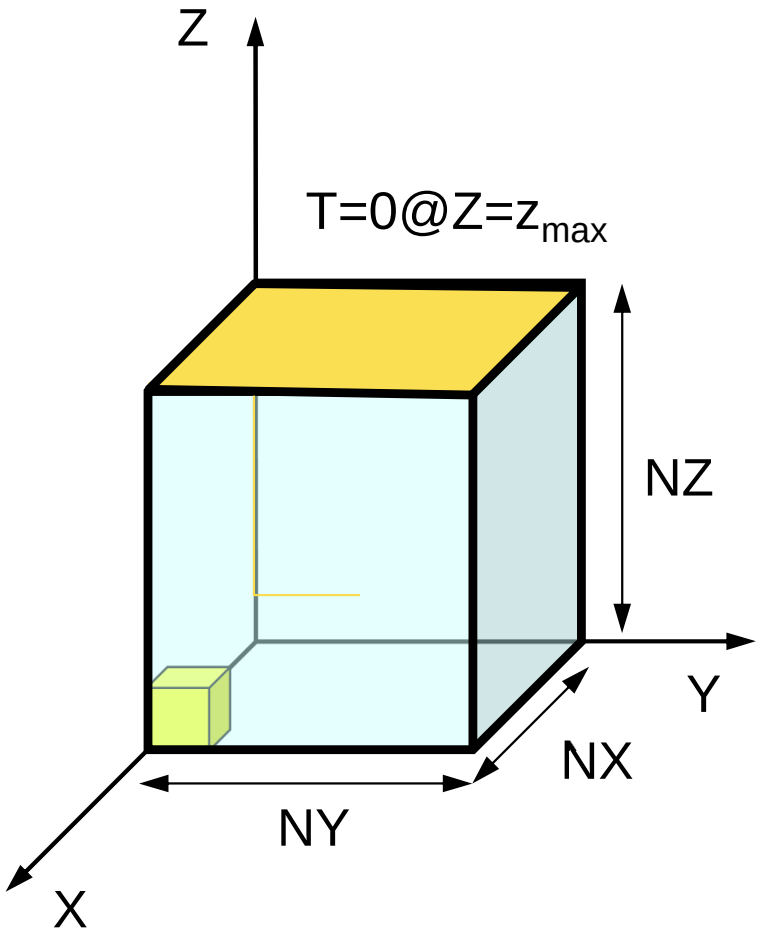
3D Parallel FEM (II)
Fortran

Kengo Nakajima
RIKEN R-CCS

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

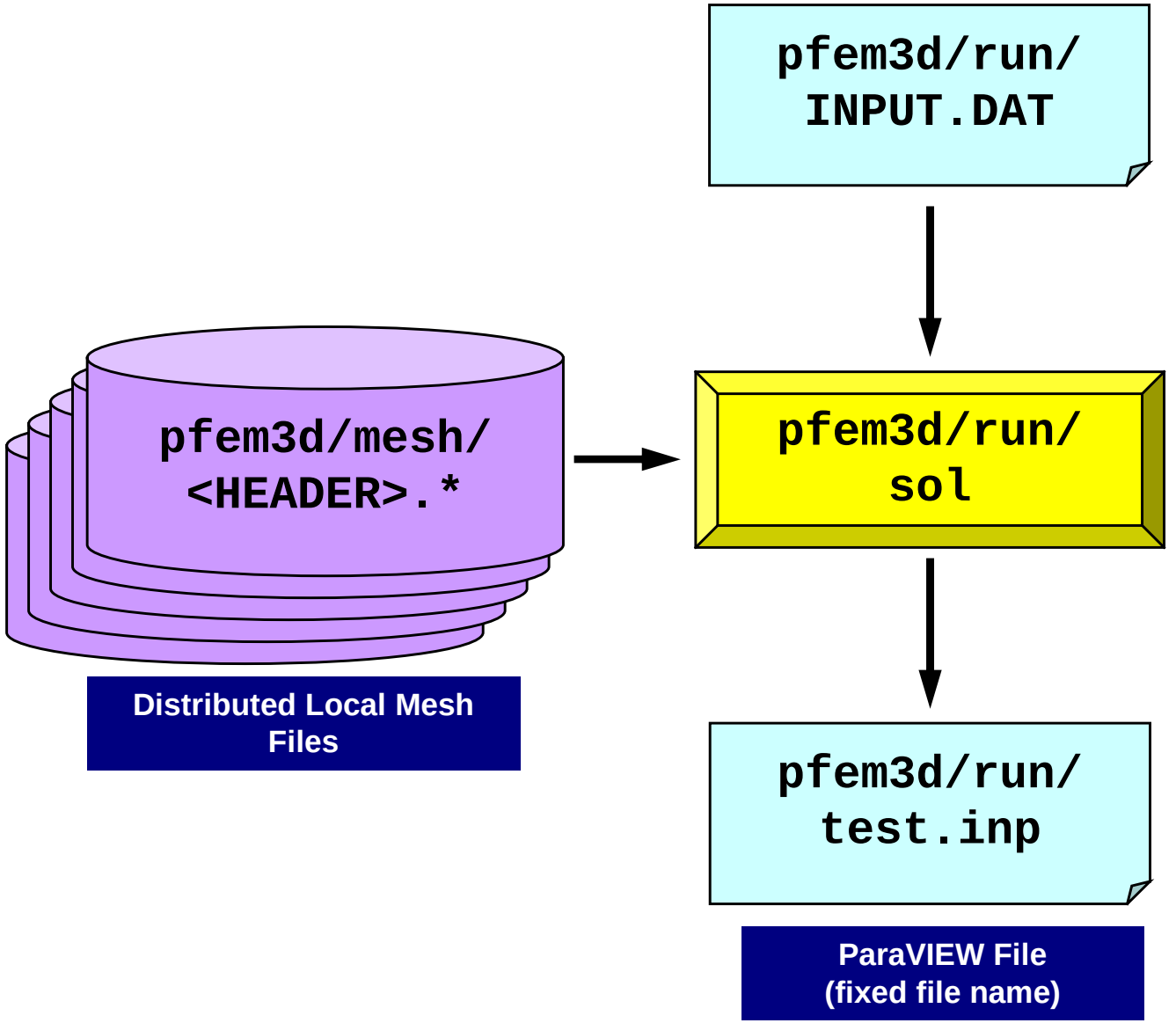
Finite-Element Procedures

- 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

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

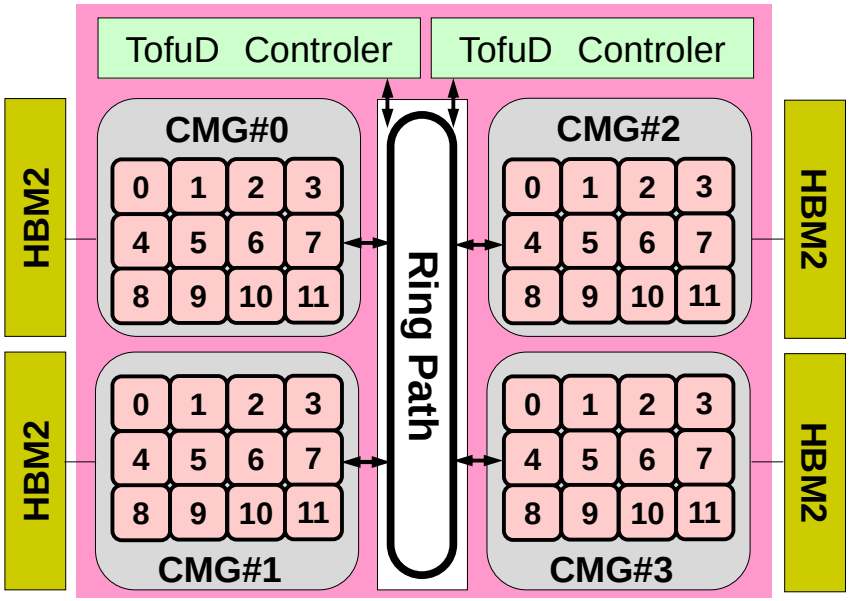
$$\dot{Q}(x,y,z)=QVOL\left|x_C+y_C\right|$$

pFEM/pfem3d/run/a08.sh

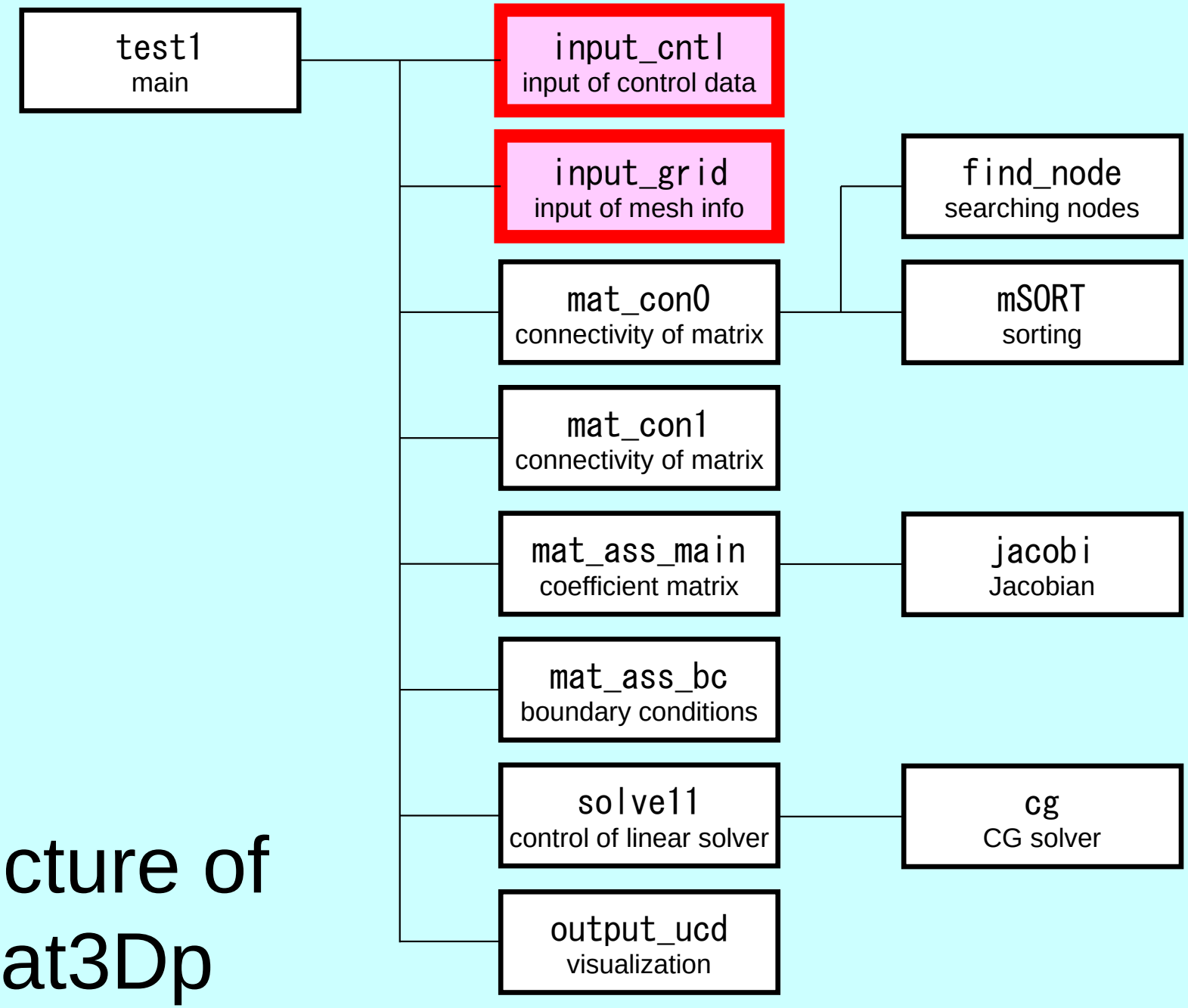
```
#!/bin/sh
#PJM -N “flat-08“                    Job Name
#PJM -L rscgrp=lecture6-o            Name of “Queue/Resource Group”
#PJM –L node=8                       Node #
#PJM –mpi proc=384                   Total MPI # (384/8= 48 per node)
#PJM –L elapse=00:15:00              Computation Time
#PJM –g gt36                         Group Name (Wallet)
#PJM -j
#PJM -e err                          Standard Error
#PJM -o a08.lst          Standard Output

module load fj
module load fjmpi

mpiexec ./sol
mpiexec numactl -l ./sol
```

Number of Processes

```
#PJM -L node=1;#PJM --mpi proc= 1    1-node, 1-proc, 1-proc/n
#PJM -L node=1;#PJM --mpi proc= 4    1-node, 4-proc, 4-proc/n
#PJM -L node=1;#PJM --mpi proc=12   1-node,12-proc,12-proc/n
#PJM -L node=1;#PJM --mpi proc=24   1-node,24-proc,24-proc/n
#PJM -L node=1;#PJM --mpi proc=48   1-node,48-proc,48-proc/n

#PJM -L node= 4;#PJM --mpi proc=192  4-node,192-proc,48-proc/n
#PJM -L node= 8;#PJM --mpi proc=384  8-node,384-proc,48-proc/n
#PJM -L node=12;#PJM --mpi proc=576 12-node,576-proc,48-proc/n
```



Structure of heat3Dp



Main Part

```
      program heat3Dp

      use solver11
      use pfem_util

      implicit REAL*8(A-H,O-Z)

      call PFEM_INIT

      call INPUT_CNTL
      call INPUT_GRID

      call MAT_CON0
      call MAT_CON1

      call MAT_ASS_MAIN
      call MAT_ASS_BC

      call SOLVE11

      call OUTPUT_UCD

      call PFEM_FINALIZE

      end program heat3Dp
```

Global Variables: pfem_util.f (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	(0:NODGRPtot)	I	# Node in each Node Group
NODGRP_ITEM	I	(NODGRP_INDEX(NODGRPTOT))	I	Node ID in each Node Group
NODGRP_NAME	C80	(NODGRP_INDEX(NODGRPTOT))	I	Name of NodeGroup
NLU	I		O	# Non-Zero Off-Diagonals at each node
NPLU	I		O	# Non-Zero Off-Diagonals
D	R	(NP)	O	Diagonal Block of Global Matrix
B,X	R	(NP)	O	RHS, Unknown Vector

Global Variables: pfem_util.f (2/4)

Name	Type	Size	I/O	Definition
AMAT	R	(NPLU)	O	Non-Zero Off-Diagonal Components of Global Matrix
index	I	(0:NP)	O	# Non-Zero Off-Diagonal Components
item	I	(NPLU)	O	Column ID of Non-Zero Off-Diagonal Components
INLU	I	(NP)	O	Number of Non-Zero Off-Diagonal Components at Each Node
IALU	I	(NP,NLU)	O	Column ID of Non-Zero Off-Diagonal Components at Each Node
IWKX	I	(NP,2)	O	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)	O	Integer Parameter Array
pfemRarray	R	(100)	O	Real Parameter Array

Global Variables: pfem_util.f (3/4)

Name	Type	Size	I/O	Definition
O8th	R		I	= 0.125
PNQ, PNE, PNT	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
POS, WEI	R	(2)	O	Coordinates, Weighting Factor at each Gaussian Quad. Point
NCOL1, NCOL2	I	(100)	O	Work arrays for sorting
SHAPE	R	(2,2,2,8)	O	N_i (i=1~8) at each Gaussian Quad Point
PNX, PNY, PNZ	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
DETJ	R	(2,2,2)	O	Determinant of Jacobian Matrix at each Gaussian Quad. Point
COND, QVOL	R		I	Thermal Conductivity, Heat Generation Rate

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

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

Global Variables: pfem_util.f (4/4)

Name	Type	Size	I/O	Definition
PETOT	I		O	Number of PE's
my_rank	I		O	Process ID of MPI
errno	I		O	Error Flag
NEIBPETOT	I		I	Number of Neighbors
NEIBPE	I	(NEIBPETOT)	I	ID of Neighbor
IMPORT_INDEX **EXPORT_INEDX**	I	(0:NEIBPETOT)	I	Size of Import/Export Arrays for Communication Table
IMPORT_ITEM	I	(Npimport)	I	Receiving Table (External Points) NPimport=IMPORT_INDEX(NEIBPETOT))
EXPORT_ITEM	I	(Npexport)	I	Sending Table (Boundary Points) NPexport=EXPORT_INDEX(NEIBPETOT))
ICELTOT_INT	**I**		**I**	**Number of Local Elements**
intELEM_list	**I**	**(ICELTOT_INT)**	**I**	**List of Local Elements**

Start/End: MPI_Init/Finalize

```
      subroutine PFEM_INIT
      use pfem_util
      implicit REAL*8 (A-H,O-Z)

      call MPI_INIT      (ierr)
      call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
      call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )

      pfemRarray= 0.d0
      pfemIarray= 0

      return
      end
```

```
      subroutine PFEM_FINALIZE
      use pfem_util
      implicit REAL*8 (A-H,O-Z)

      call MPI_FINALIZE (errno)
      if (my_rank.eq.0) stop ' * normal termination'

      return
      end
```

Reading Control File: INPUT_CNTL

```
      subroutine INPUT_CNTL
      use pfem_util

      implicit REAL*8 (A-H,O-Z)

      if (my_rank.eq.0) then
        open (11,file='INPUT.DAT', status='unknown')
        read (11,'(a80)') HEADER
        read (11,*) ITER
        read (11,*) COND, QVOL
        read (11,*) RESID
        close (11)
      endif

      call MPI_BCAST (HEADER, 80, MPI_CHARACTER, 0, MPI_COMM_WORLD,ierr)
      call MPI_BCAST (ITER  , 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (COND  , 1, MPI_DOUBLE_PRECISION, 0,
     &                                    MPI_COMM_WORLD, ierr)
      call MPI_BCAST (QVOL  , 1, MPI_DOUBLE_PRECISION, 0,
     &                                    MPI_COMM_WORLD, ierr)
      call MPI_BCAST (RESID , 1, MPI_DOUBLE_PRECISION, 0,
     &                                    MPI_COMM_WORLD, ierr)

      pfemRarray(1)= RESID
      pfemIarray(1)= ITER

      return
      end
```

Reading Meshes: INPUT_GRID (1/3)

```
      subroutine INPUT_GRID
      use pfem_util
      implicit REAL*8 (A-H,O-Z)

      call define_file_name (HEADER, fname, my_rank)
      open (11, file= fname, status= 'unknown', form= 'formatted')

!C
!C-- NEIB-PE
      read (11,'(10i10)') kkk
      read (11,'(10i10)') NEIBPETOT
      allocate (NEIBPE(NEIBPETOT))

      read (11,'(10i10)') (NEIBPE(i), i= 1, NEIBPETOT)

      do i= 1, NEIBPETOT
        if (NEIBPE(i).gt.PETOT-1) then
          call ERROR_EXIT (202, my_rank)
        endif
      enddo
```

Name of Distributed Local Mesh File: DEFINE_FILE_NAME HEADER + Rank ID

```
      subroutine DEFINE_FILE_NAME (HEADERo, filename, my_rank)

      character (len=80) ::  HEADERo, filename
      character (len=80) ::  HEADER
      character (len= 1) ::  SUBindex1
      character (len= 2) ::  SUBindex2
      character (len= 3) ::  SUBindex3
      integer:: LENGTH, ID

      HEADER= adjustL (HEADERo)
      LENGTH= len_trim(HEADER)

      if (my_rank.le.9) then
        ID= 1
        write(SUBindex1 ,'(i1.1)') my_rank
       else if (my_rank.le.99) then
        ID= 2
        write(SUBindex2 ,'(i2.2)') my_rank
       else if (my_rank.le.999) then
        ID= 3
        write(SUBindex3 ,'(i3.3)') my_rank
      endif

      if (ID.eq.1) filename= HEADER(1:LENGTH)//'.'//SUBindex1
      if (ID.eq.2) filename= HEADER(1:LENGTH)//'.'//SUBindex2
      if (ID.eq.3) filename= HEADER(1:LENGTH)//'.'//SUBindex3

      end subroutine define_file_name
```

allocate, deallocate for C

```
#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)

```
!C
!C-- NODE
      read (11,'(10i10)')  NP, N
      allocate (XYZ(NP,3), NODE_ID(NP,2))
      XYZ= 0.d0
        do i= 1, NP
          read (11,*) NODE_ID(i,1), NODE_ID(i,2), (XYZ(i,kk),kk=1,3)
        enddo

!C
!C-- ELEMENT
      read (11,*)  ICELTOT, ICELTOT_INT

      allocate (ICELNOD(ICELTOT,8), intELEM_list(ICELTOT))
      allocate (ELEM_ID(ICELTOT,2))
      read (11,'(10i10)') (NTYPE, i= 1, ICELTOT)
      do icel= 1, ICELTOT
        read (11,'(i10,2i5,8i10)') (ELEM_ID(icel,jj),jj=1,2),           &
     &                         IMAT, (ICELNOD(icel,k), k= 1, 8)
      enddo

      read (11,'(10i10)') (intELEM_list(ic0), ic0= 1, ICELTOT_INT)
```

Reading Meshes: INPUT_GRID (3/3)

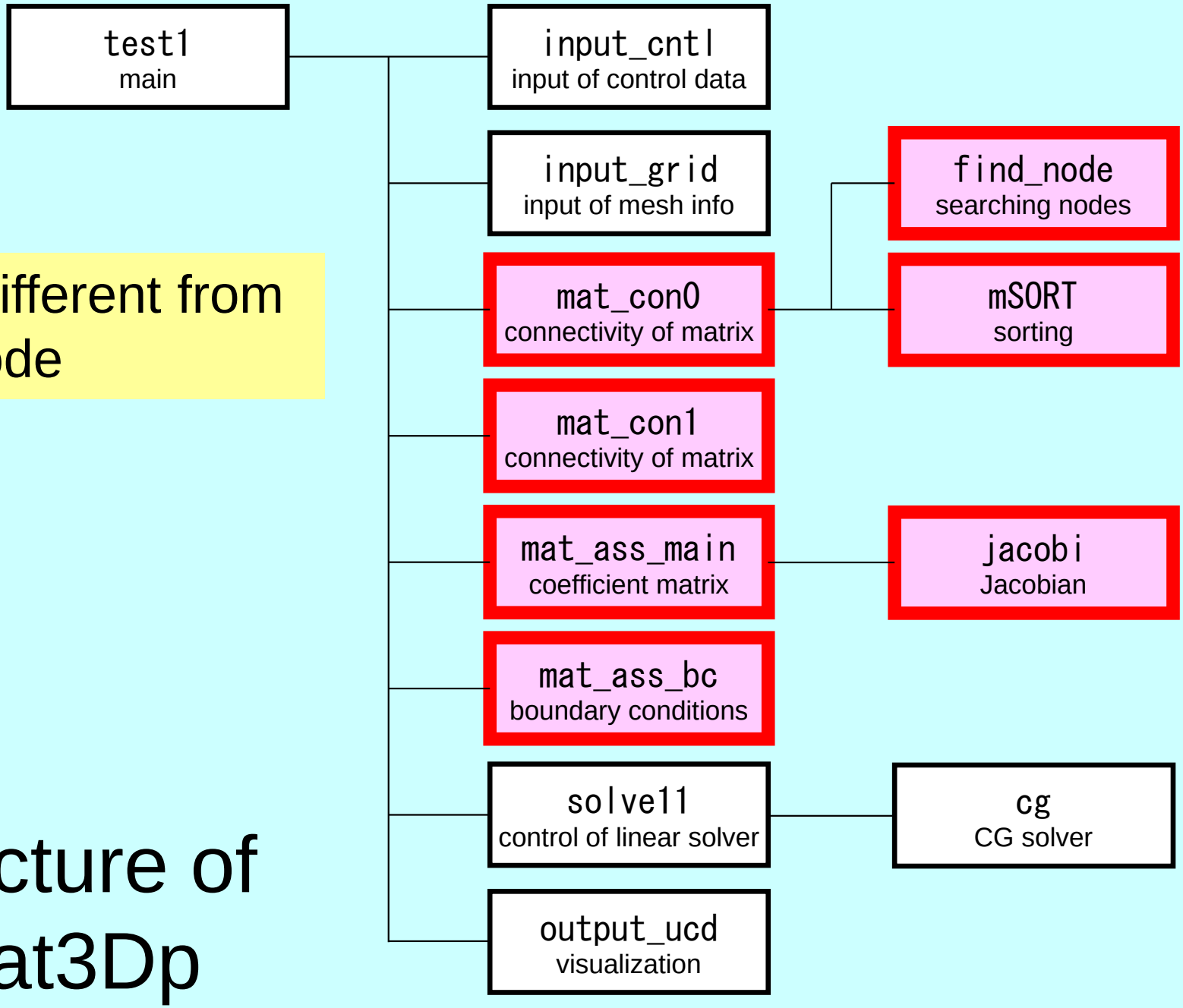
```
!C-- COMMUNICATION table
      allocate (IMPORT_INDEX(0:NEIBPETOT))
      allocate (EXPORT_INDEX(0:NEIBPETOT))

      IMPORT_INDEX= 0
      EXPORT_INDEX= 0

      if (PETOT.ne.1) then
      read (11,'(10i10)') (IMPORT_INDEX(i), i= 1, NEIBPETOT)
      nn= IMPORT_INDEX(NEIBPETOT)
      allocate (IMPORT_ITEM(nn))
      do i= 1, nn
        read (11,*) IMPORT_ITEM(i)
      enddo

      read (11,'(10i10)') (EXPORT_INDEX(i), i= 1, NEIBPETOT)
      nn= EXPORT_INDEX(NEIBPETOT)
      allocate (EXPORT_ITEM(nn))
      do i= 1, nn
        read (11,*) EXPORT_ITEM(i)
      enddo
      endif
!C-- NODE grp. info.
      read (11,'(10i10)') NODGRPtot
      allocate (NODGRP_INDEX(0:NODGRPtot),NODGRP_NAME(NODGRPtot))
      NODGRP_INDEX= 0

      read (11,'(10i10)') (NODGRP_INDEX(i), i= 1, NODGRPtot)
      nn= NODGRP_INDEX(NODGRPtot)
      allocate (NODGRP_ITEM(nn))

      do k= 1, NODGRPtot
        iS= NODGRP_INDEX(k-1) + 1
        iE= NODGRP_INDEX(k  )
        read (11,'(a80)') NODGRP_NAME(k)
        nn= iE - iS + 1
        if (nn.ne.0) then
          read (11,'(10i10)') (NODGRP_ITEM(kk),kk=iS, iE)
        endif
      enddo
```

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

```
      program heat3Dp

      use solver11
      use pfem_util

      implicit REAL*8(A-H,O-Z)

      call PFEM_INIT
      call INPUT_CNTL
      call INPUT_GRID

      call MAT_CON0
      call MAT_CON1

      call MAT_ASS_MAIN
      call MAT_ASS_BC

      call SOLVE11

      call OUTPUT_UCD

      call PFEM_FINALIZE

      end program heat3Dp
```

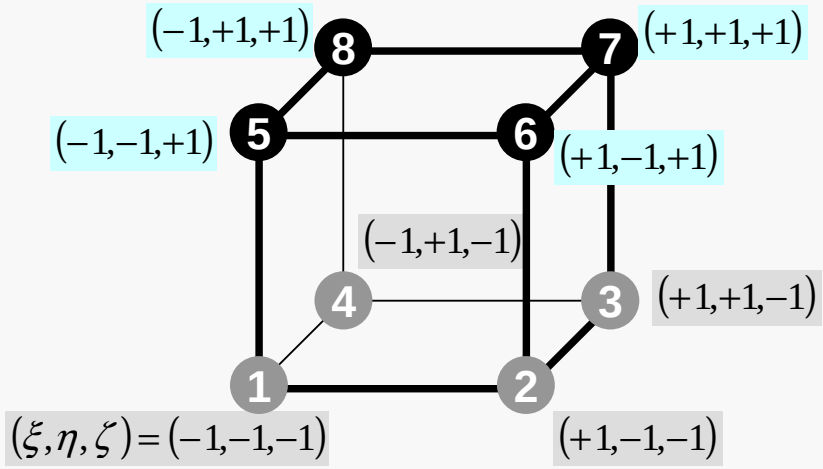
MAT_CON0: generates INU, IALU
MAT_CON1: generates index, item

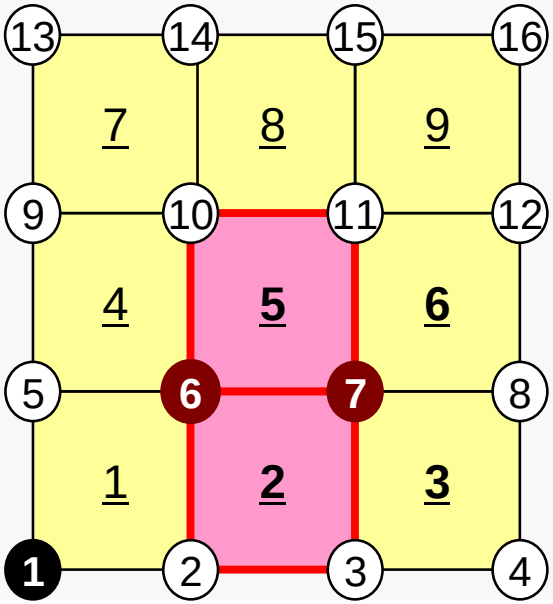
Please compare parallel/serial codes

```
$ cd /work/t87XYZ/pFEM/pfem3d/src

$ diff mat_con1.f ../../fem3d/src/mat_con0.f
$ diff mat_con0.f ../../fem3d/src/mat_con1.f
$ diff mat_ass_main.f ../../fem3d/src/mat_ass_main.f
$ diff mat_ass_bc.f ../../fem3d/src/mat_ass_bc.f
```

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)

```
!C
!C***
!C*** MAT_CON0
!C***
!C
      subroutine MAT_CON0
      use pfem_util
      implicit REAL*8 (A-H,O-Z)

      NLU= 26

      allocate (INLU(NP), IALU(NP,NLU))

      INLU= 0
      IALU= 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 NUL in this way.

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

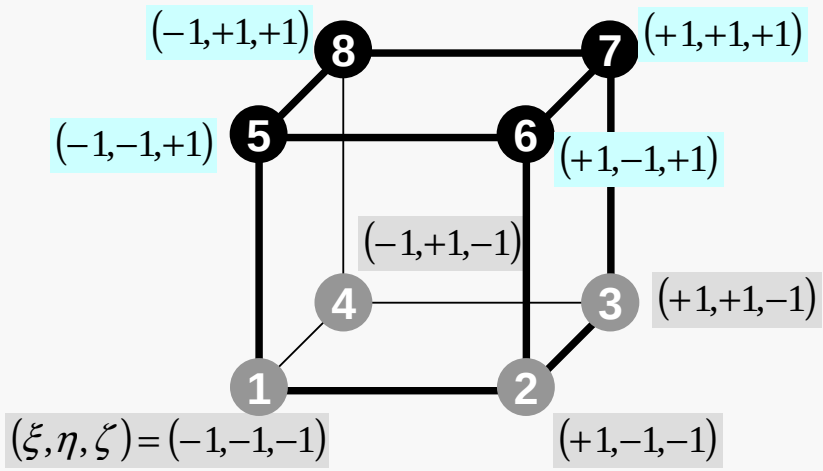
Generating Connectivity of Matrix MAT_CON0 (1/4)

```
!C
!C***
!C*** MAT_CON0
!C***
!C
      subroutine MAT_CON0
      use pfem_util
      implicit REAL*8 (A-H,O-Z)

      NLU= 26

      allocate (INLU(NP), IALU(NP,NLU))

      INLU= 0
      IALU= 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)

```
      do icel= 1, ICELTOT
        in1= ICELNOD(icel,1)
        in2= ICELNOD(icel,2)
        in3= ICELNOD(icel,3)
        in4= ICELNOD(icel,4)
        in5= ICELNOD(icel,5)
        in6= ICELNOD(icel,6)
        in7= ICELNOD(icel,7)
        in8= ICELNOD(icel,8)

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

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

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



FIND_TS_NODE: Search Connectivity

INLU,IALU: Automatic Search

```
!C
!C***
!C*** FIND_TS_NODE
!C***
!C
        subroutine FIND_TS_NODE (ip1,ip2)

          do kk= 1, INLU(ip1)
            if (ip2.eq.IALU(ip1,kk)) return
          enddo

          icou= INLU(ip1) + 1
          IALU(ip1,icou)= ip2
          INLU(ip1     )= icou

          return

        end subroutine FIND_TS_NODE
```

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

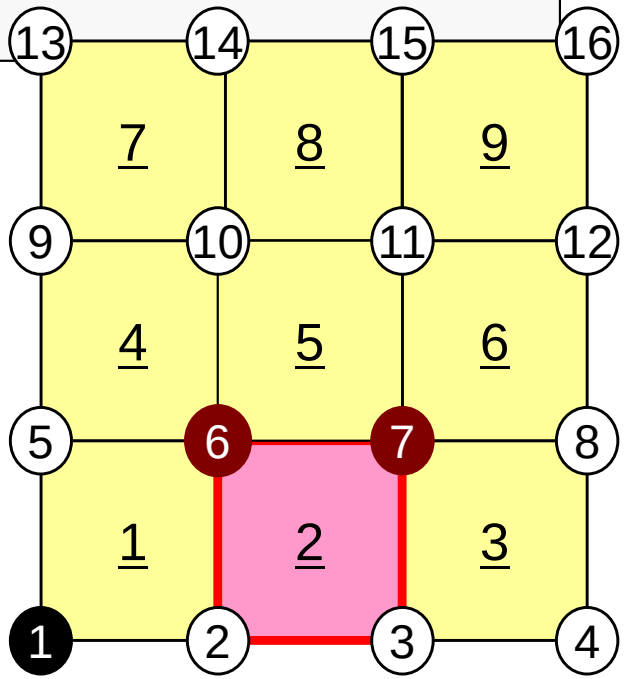
FIND_TS_NODE: Search Connectivity

INLU,IALU: Automatic Search **Element #2**

```
!C
!C***
!C*** FIND_TS_NODE
!C***
!C
        subroutine FIND_TS_NODE (ip1,ip2)

          do kk= 1, INLU(ip1)
            if (ip2.eq.IALU(ip1,kk)) return
          enddo

          icou= INLU(ip1) + 1
          IALU(ip1,icou)= ip2
          INLU(ip1     )= icou

          return

        end subroutine FIND_TS_NODE
```

Checking whether ip2 is included in IALU(ip1,kk), or not

ip1: No.6 node
ip2: No.7 node

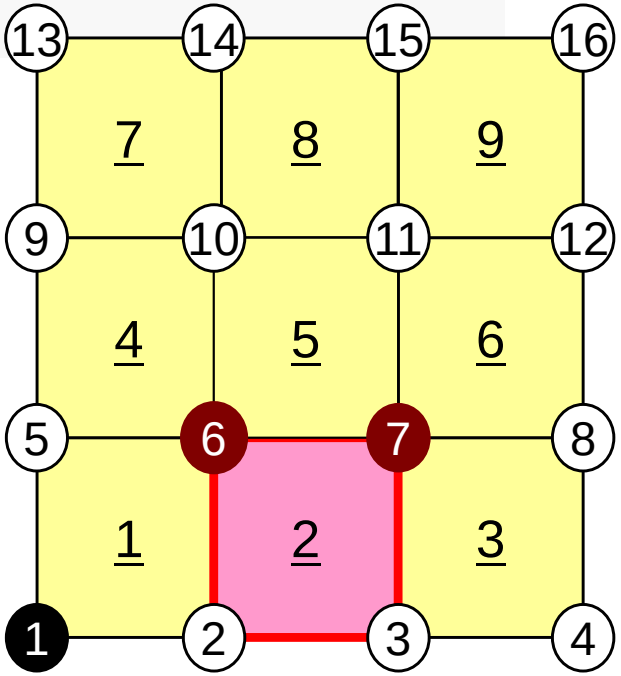


FIND_TS_NODE: Search Connectivity

INLU,IALU: Automatic Search **Element #2**

```
!C
!C***
!C*** FIND_TS_NODE
!C***
!C
        subroutine FIND_TS_NODE (ip1,ip2)

          do kk= 1, INLU(ip1)
            if (ip2.eq.IALU(ip1,kk)) return
          enddo

          icou= INLU(ip1) + 1
          IALU(ip1,icou)= ip2
          INLU(ip1     )= icou

          return

        end subroutine FIND_TS_NODE
```

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



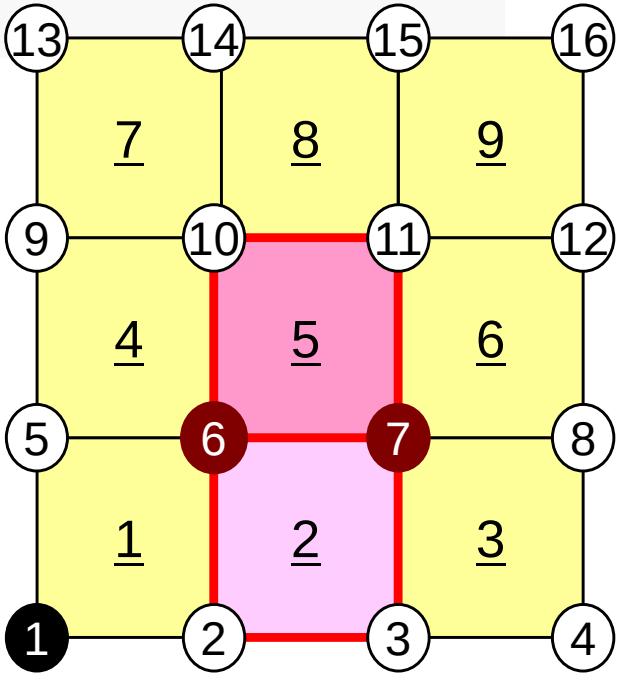
ip1: No.6 node
ip2: No.7 node

FIND_TS_NODE: Search Connectivity

INLU,IALU: Automatic Search **Element #5**

```
!C
!C***
!C*** FIND_TS_NODE
!C***
!C
        subroutine FIND_TS_NODE (ip1,ip2)

          do kk= 1, INLU(ip1)
            if (ip2.eq.IALU(ip1,kk)) return
          enddo

          icou= INLU(ip1) + 1
          IALU(ip1,icou)= ip2
          INLU(ip1     )= icou

          return

        end subroutine FIND_TS_NODE
```

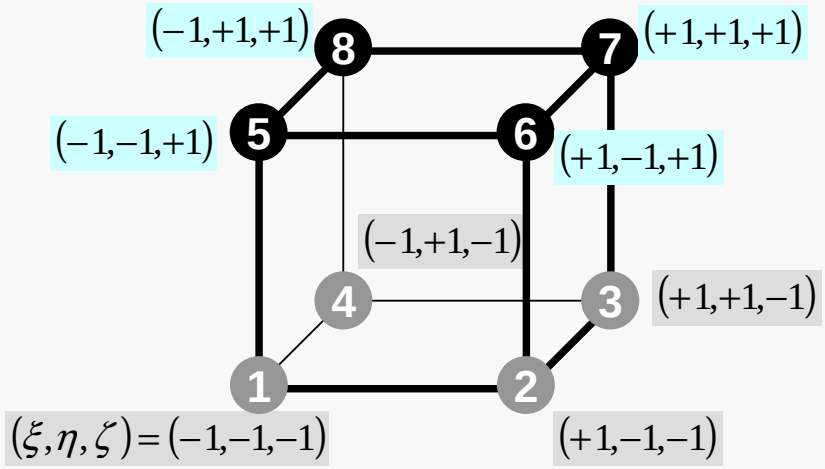
If the target node is already included in IALU, proceed to next pair of nodes



ip1: No.6 node
ip2: No.7 node

Generating Connectivity of Matrix MAT_CON0 (3/4)

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

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

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

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



Generating Connectivity of Matrix MAT_CON0 (4/4)

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

    do in= 1, N
      NN= INLU(in)
      do k= 1, NN
        NCOL1(k)= IALU(in,k)
      enddo
      call mSORT (NCOL1, NCOL2, NN)
      do k= NN, 1, -1
        IALU(in,NN-k+1)= NCOL1(NCOL2(k))
      enddo
    enddo
```

Sort IALU(i,k) in ascending order by “bubble” sorting for less than 100 components.

MAT_CON1: CRS format

```
!C
!C***
!C*** MAT_CON1
!C***
!C
      subroutine MAT_CON1
      use pfem_util
      implicit REAL*8 (A-H,O-Z)

      allocate (index(0:NP))
      index= 0

      do i= 1, NP
        index(i)= index(i-1) + INLU(i)
      enddo

      NPLU= index(NP)

      allocate (item(NPLU))

      do i= 1, NP
        do k= 1, INLU(i)
              kk = k + index(i-1)
          item(kk)=      IALU(i,k)
        enddo
      enddo

      deallocate (INLU, IALU)

      end subroutine MAT_CON1
```

C

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

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

FORTRAN

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

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

MAT_CON1: CRS format

```
!C
!C***
!C*** MAT_CON1
!C***
!C
      subroutine MAT_CON1
      use pfem_util
      implicit REAL*8 (A-H,O-Z)

      allocate (index(0:NP))
      index= 0

      do i= 1, NP
        index(i)= index(i-1) + INLU(i)
      enddo

      NPLU= index(NP)

      allocate (item(NPLU))

      do i= 1, NP
        do k= 1, INLU(i)
              kk = k + index(i-1)
          item(kk)=      IALU(i,k)
        enddo
      enddo

      deallocate (INLU, IALU)

      end subroutine MAT_CON1
```

NPLU=indexLU(NP)
Size of array: itemLU
Total number of non-zero off-diagonal blocks

MAT_CON1: CRS format

```
!C
!C***
!C*** MAT_CON1
!C***
!C
      subroutine MAT_CON1
      use pfem_util
      implicit REAL*8 (A-H,O-Z)

      allocate (index(0:NP))
      index= 0

      do i= 1, NP
        index(i)= index(i-1) + INLU(i)
      enddo

      NPLU= index(NP)

      allocate (item(NPLU))

      do i= 1, NP
        do k= 1, INLU(i)
              kk = k + index(i-1)
          item(kk)=      IALU(i,k)
        enddo
      enddo

      deallocate (INLU, IALU)

      end subroutine MAT_CON1
```

itemLU
store node ID starting from 1

MAT_CON1: CRS format

```
!C
!C***
!C*** MAT_CON1
!C***
!C
      subroutine MAT_CON1
      use pfem_util
      implicit REAL*8 (A-H,O-Z)

      allocate (index(0:NP))
      index= 0

      do i= 1, NP
        index(i)= index(i-1) + INLU(i)
      enddo

      NPLU= index(NP)

      allocate (item(NPLU))

      do i= 1, NP
        do k= 1, INLU(i)
              kk = k + index(i-1)
          item(kk)=      IALU(i,k)
        enddo
      enddo

      deallocate (INLU, IALU)

      end subroutine MAT_CON1
```

Not required any more

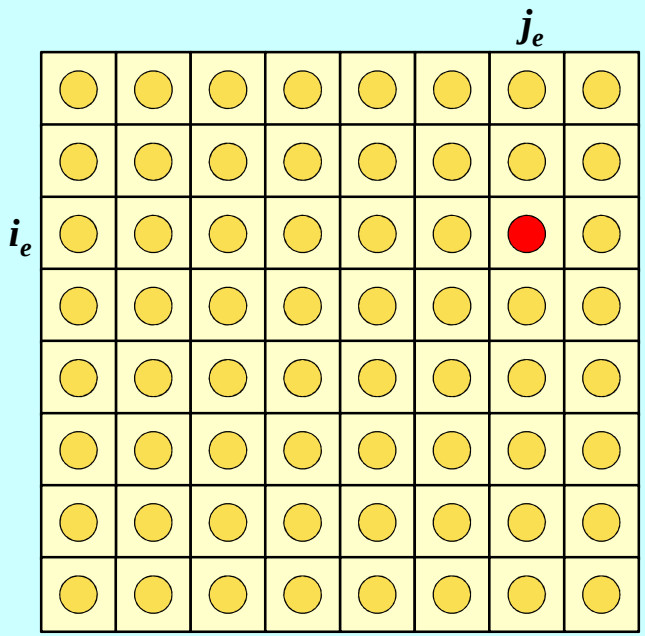
Main Part

```
      program heat3Dp

      use solver11
      use pfem_util

      implicit REAL*8(A-H,O-Z)

      call PFEM_INIT
      call INPUT_CNTL
      call INPUT_GRID

      call MAT_CON0
      call MAT_CON1

      call MAT_ASS_MAIN
      call MAT_ASS_BC

      call SOLVE11

      call OUTPUT_UCD

      call PFEM_FINALIZE

      end program heat3Dp
```

MAT_ASS_MAIN: Overview

```
do kpn= 1, 2          Gaussian Quad. points in ζ-direction
  do jpn= 1, 2        Gaussian Quad. points in η-direction
    do ipn= 1, 2      Gaussian Quad. Pointe in ξ-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 "item" : kk

      do kpn= 1, 2        Gaussian Quad. points in ζ-direction
        do jpn= 1, 2      Gaussian Quad. points in η-direction
          do ipn= 1, 2    Gaussian Quad. points in ξ-direction
            integration on each element
            coefficients of element matrices
            accumulation to global matrix
          enddo
        enddo
      enddo
    enddo
  enddo
enddo
```



MAT_ASS_MAIN (1/6)

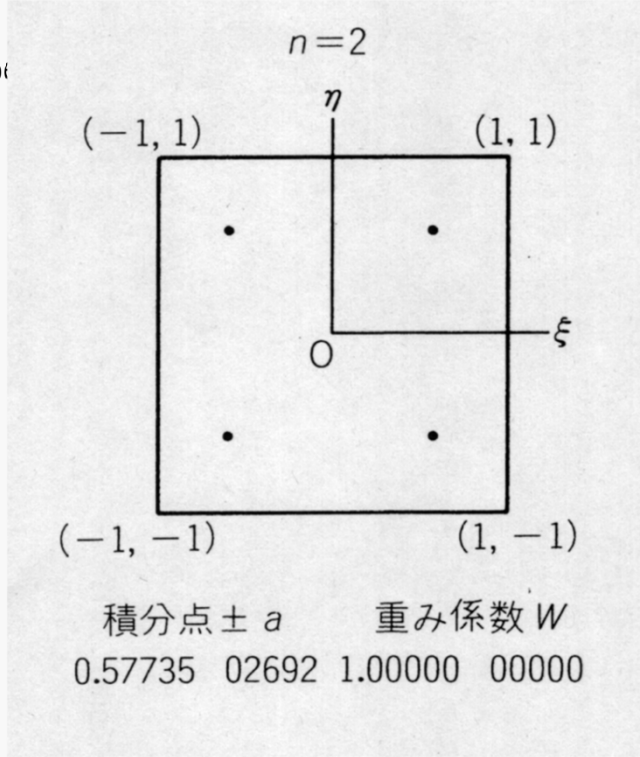
```
!C
!C***
!C*** MAT_ASS_MAIN
!C***
!C
      subroutine MAT_ASS_MAIN
      use pfem_util
      implicit REAL*8 (A-H,O-Z)
      integer(kind=kint), dimension(  8) :: nodLOCAL

      allocate (AMAT(NPLU))
      allocate (B(NP), D(NP), X(NP))

      AMAT= 0.d0                    Non-Zero Off-Diagonal components (coef. matrix)
         B= 0.d0                    RHS vector
         X= 0.d0                    Unknowns
         D= 0.d0                    Diagonal components (coef. matrix)

      WEI(1)= +1.0000000000D+00
      WEI(2)= +1.0000000000D+00

      POS(1)= -0.5773502692D+00
      POS(2)= +0.5773502692D+00
```

MAT_ASS_MAIN (1/6)

```
!C
!C***
!C*** MAT_ASS_MAIN
!C***
!C
      subroutine MAT_ASS_MAIN
      use pfem_util
      implicit REAL*8 (A-H,O-Z)
      integer(kind=kint), dimension(  8) :: nodLOCAL

      allocate (AMAT(NPLU))
      allocate (B(NP), D(NP), X(NP))

      AMAT= 0.d0                    Non-Zero Off-Diagonal components (coef. matrix)
         B= 0.d0                    RHS vector
         X= 0.d0                    Unknowns
         D= 0.d0                    Diagonal components (co

      WEI(1)= +1.0000000000D+00
      WEI(2)= +1.0000000000D+00

      POS(1)= -0.5773502692D+00
      POS(2)= +0.5773502692D+00
```

POS: Quad. Point

WEI: Weighting Factor



MAT_ASS_MAIN (2/6)

```
!C
!C-- INIT.
!C     PNQ   - 1st-order derivative of shape function by QSI
!C     PNE   - 1st-order derivative of shape function by ETA
!C     PNT   - 1st-order derivative of shape function by ZET
!C

      do kp= 1, 2
      do jp= 1, 2
      do ip= 1, 2

        QP1= 1.d0 + POS(ip)
        QM1= 1.d0 - POS(ip)
        EP1= 1.d0 + POS(jp)
        EM1= 1.d0 - POS(jp)
        TP1= 1.d0 + POS(kp)
        TM1= 1.d0 - POS(kp)

        SHAPE(ip,jp,kp,1)= 08th * QM1 * EM1 * TM1
        SHAPE(ip,jp,kp,2)= 08th * QP1 * EM1 * TM1
        SHAPE(ip,jp,kp,3)= 08th * QP1 * EP1 * TM1
        SHAPE(ip,jp,kp,4)= 08th * QM1 * EP1 * TM1
        SHAPE(ip,jp,kp,5)= 08th * QM1 * EM1 * TP1
        SHAPE(ip,jp,kp,6)= 08th * QP1 * EM1 * TP1
        SHAPE(ip,jp,kp,7)= 08th * QP1 * EP1 * TP1
        SHAPE(ip,jp,kp,8)= 08th * QP1 * EP1 * TP1
```

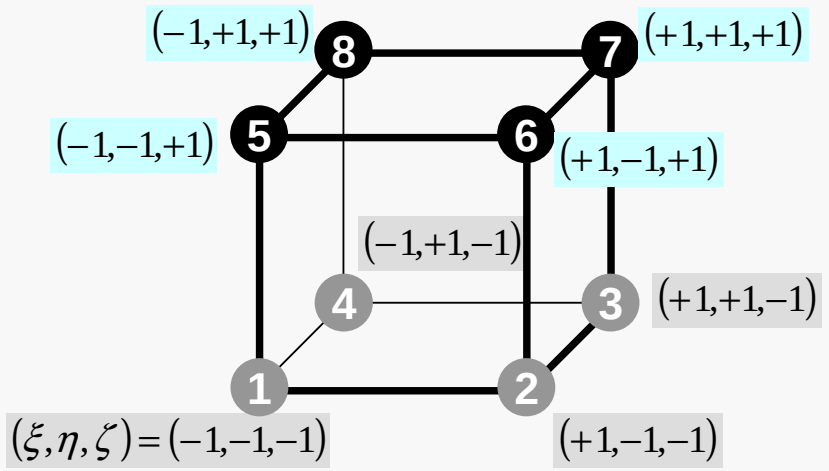
MAT_ASS_MAIN (2/6)

```
!C
!C-- INIT.
!C     PNQ   - 1st-order derivative of shape function by QSI
!C     PNE   - 1st-order derivative of shape function by ETA
!C     PNT   - 1st-order derivative of shape function by ZET
!C

      do kp= 1, 2
      do jp= 1, 2
      do ip= 1, 2

        QP1= 1.d0 + POS(ip)
        QM1= 1.d0 - POS(ip)
        EP1= 1.d0 + POS(jp)
        EM1= 1.d0 - POS(jp)
        TP1= 1.d0 + POS(kp)
        TM1= 1.d0 - POS(kp)

        SHAPE(ip,jp,kp,1)= 08th * QM1 * EM1 * TM1
        SHAPE(ip,jp,kp,2)= 08th * QP1 * EM1 * TM1
        SHAPE(ip,jp,kp,3)= 08th * QP1 * EP1 * TM1
        SHAPE(ip,jp,kp,4)= 08th * QM1 * EP1 * TM1
        SHAPE(ip,jp,kp,5)= 08th * QM1 * EM1 * TP1
        SHAPE(ip,jp,kp,6)= 08th * QP1 * EM1 * TP1
        SHAPE(ip,jp,kp,7)= 08th * QP1 * EP1 * TP1
        SHAPE(ip,jp,kp,8)= 08th * QP1 * EP1 * TP1
```

$$\mathrm{QP1(i)} = (1+\xi_i), \quad \mathrm{QM1(i)} = (1-\xi_i)$$
$$\mathrm{EP1(j)} = (1+\eta_j), \quad \mathrm{EM1(j)} = (1-\eta_i)$$
$$\mathrm{TP1(k)} = (1+\zeta_k), \quad \mathrm{TM1(k)} = (1-\zeta_k)$$

MAT_ASS_MAIN (2/6)

```
!C
!C-- INIT.
!C     PNQ   - 1st-order derivative of shape function by QSI
!C     PNE   - 1st-order derivative of shape function by ETA
!C     PNT   - 1st-order derivative of shape function by ZET
!C

      do kp= 1, 2
      do jp= 1, 2
      do ip= 1, 2

        QP1= 1.d0 + POS(ip)
        QM1= 1.d0 - POS(ip)
        EP1= 1.d0 + POS(jp)
        EM1= 1.d0 - POS(jp)
        TP1= 1.d0 + POS(kp)
        TM1= 1.d0 - POS(kp)

        SHAPE(ip,jp,kp,1)= 0.8d0 * QM1 * EM1 * TM1
        SHAPE(ip,jp,kp,2)= 0.8d0 * QP1 * EM1 * TM1
        SHAPE(ip,jp,kp,3)= 0.8d0 * QP1 * EP1 * TM1
        SHAPE(ip,jp,kp,4)= 0.8d0 * QM1 * EP1 * TM1
        SHAPE(ip,jp,kp,5)= 0.8d0 * QM1 * EM1 * TP1
        SHAPE(ip,jp,kp,6)= 0.8d0 * QP1 * EM1 * TP1
        SHAPE(ip,jp,kp,7)= 0.8d0 * QP1 * EP1 * TP1
        SHAPE(ip,jp,kp,8)= 0.8d0 * QP1 * EP1 * TP1
```



MAT_ASS_MAIN (2/6)

```
!C
!C-- INIT.
!C     PNQ   - 1st-order derivative of shape function by QSI
!C     PNE   - 1st-order derivative of shape function by ETA
!C     PNT   - 1st-order derivative of shape function by ZET
!C

      do kp= 1, 2
      do jp= 1, 2
      do ip= 1, 2

        QP1= 1.d0 + POS(ip)
        QM1= 1.d0 - POS(ip)
        EP1= 1.d0 + POS(jp)
        EM1= 1.d0 - POS(jp)
        TP1= 1.d0 + POS(kp)
        TM1= 1.d0 - POS(kp)

        SHAPE(ip,jp,kp,1)= 08th * QM1 * EM1 * TM1
        SHAPE(ip,jp,kp,2)= 08th * QP1 * EM1 * TM1
        SHAPE(ip,jp,kp,3)= 08th * QP1 * EP1 * TM1
        SHAPE(ip,jp,kp,4)= 08th * QM1 * EP1 * TM1
        SHAPE(ip,jp,kp,5)= 08th * QM1 * EM1 * TP1
        SHAPE(ip,jp,kp,6)= 08th * QP1 * EM1 * TP1
        SHAPE(ip,jp,kp,7)= 08th * QP1 * EP1 * TP1
        SHAPE(ip,jp,kp,8)= 08th * QP1 * 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 (3/6)

```
      PNQ(jp,kp,1)= - O8th * EM1 * TM1
      PNQ(jp,kp,2)= + O8th * EM1 * TM1
      PNQ(jp,kp,3)= + O8th * EP1 * TM1
      PNQ(jp,kp,4)= - O8th * EP1 * TM1
      PNQ(jp,kp,5)= - O8th * EM1 * TP1
      PNQ(jp,kp,6)= + O8th * EM1 * TP1
      PNQ(jp,kp,7)= + O8th * EP1 * TP1
      PNQ(jp,kp,8)= - O8th * EP1 * TP1
      PNE(ip,kp,1)= - O8th * QM1 * TM1
      PNE(ip,kp,2)= - O8th * QP1 * TM1
      PNE(ip,kp,3)= + O8th * QP1 * TM1
      PNE(ip,kp,4)= + O8th * QM1 * TM1
      PNE(ip,kp,5)= - O8th * QM1 * TP1
      PNE(ip,kp,6)= - O8th * QP1 * TP1
      PNE(ip,kp,7)= + O8th * QP1 * TP1
      PNE(ip,kp,8)= + O8th * QM1 * TP1
      PNT(ip,jp,1)= - O8th * QM1 * EM1
      PNT(ip,jp,2)= - O8th * QP1 * EM1
      PNT(ip,jp,3)= - O8th * QP1 * EP1
      PNT(ip,jp,4)= - O8th * QM1 * EP1
      PNT(ip,jp,5)= + O8th * QM1 * EM1
      PNT(ip,jp,6)= + O8th * QP1 * EM1
      PNT(ip,jp,7)= + O8th * QP1 * EP1
      PNT(ip,jp,8)= + O8th * QM1 * EP1
    enddo
  enddo
enddo

do icel= 1, ICELTOT
  CONDO= COND

  in1= ICELNOD(icel,1)
  in2= ICELNOD(icel,2)
  in3= ICELNOD(icel,3)
  in4= ICELNOD(icel,4)
  in5= ICELNOD(icel,5)
  in6= ICELNOD(icel,6)
  in7= ICELNOD(icel,7)
  in8= ICELNOD(icel,8)
```

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

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

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

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

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

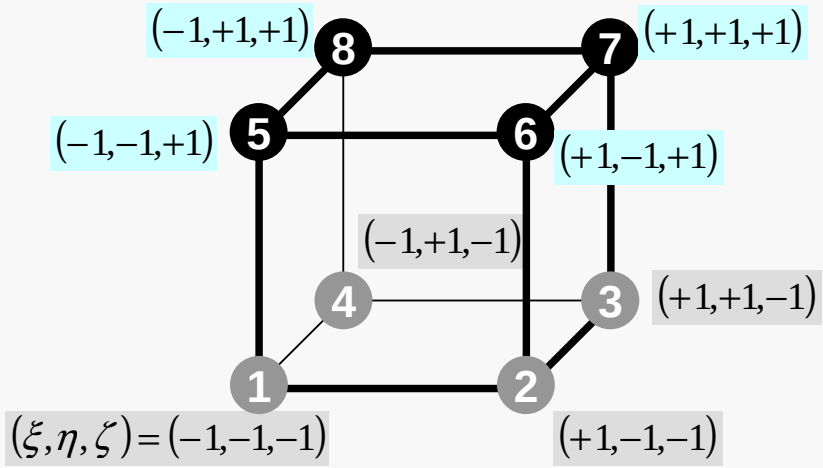
$$\frac{\partial N_3}{\partial \xi}(\xi_i,\eta_j,\zeta_k)=+\frac{1}{8}(1+\eta_j)(1-\zeta_k)$$

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

First Order Derivative of Shape Functions at (ξ_i,η_j,ζ_k)

MAT_ASS_MAIN (3/6)

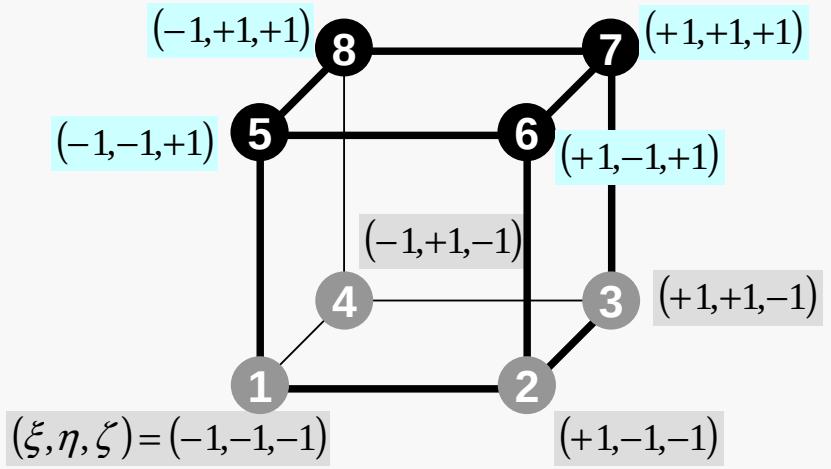
```
      PNQ(jp,kp,1)= - 08th * EM1 * TM1
      PNQ(jp,kp,2)= + 08th * EM1 * TM1
      PNQ(jp,kp,3)= + 08th * EP1 * TM1
      PNQ(jp,kp,4)= - 08th * EP1 * TM1
      PNQ(jp,kp,5)= - 08th * EM1 * TP1
      PNQ(jp,kp,6)= + 08th * EM1 * TP1
      PNQ(jp,kp,7)= + 08th * EP1 * TP1
      PNQ(jp,kp,8)= - 08th * EP1 * TP1
      PNE(ip,kp,1)= - 08th * QM1 * TM1
      PNE(ip,kp,2)= - 08th * QP1 * TM1
      PNE(ip,kp,3)= + 08th * QP1 * TM1
      PNE(ip,kp,4)= + 08th * QM1 * TM1
      PNE(ip,kp,5)= - 08th * QM1 * TP1
      PNE(ip,kp,6)= - 08th * QP1 * TP1
      PNE(ip,kp,7)= + 08th * QP1 * TP1
      PNE(ip,kp,8)= + 08th * QM1 * TP1
      PNT(ip,jp,1)= - 08th * QM1 * EM1
      PNT(ip,jp,2)= - 08th * QP1 * EM1
      PNT(ip,jp,3)= - 08th * QP1 * EP1
      PNT(ip,jp,4)= - 08th * QM1 * EP1
      PNT(ip,jp,5)= + 08th * QM1 * EM1
      PNT(ip,jp,6)= + 08th * QP1 * EM1
      PNT(ip,jp,7)= + 08th * QP1 * EP1
      PNT(ip,jp,8)= + 08th * QM1 * EP1
    enddo
  enddo
enddo

do icel= 1, ICELTOT
  CONDO= COND

  in1= ICELNOD(icel,1)
  in2= ICELNOD(icel,2)
  in3= ICELNOD(icel,3)
  in4= ICELNOD(icel,4)
  in5= ICELNOD(icel,5)
  in6= ICELNOD(icel,6)
  in7= ICELNOD(icel,7)
  in8= ICELNOD(icel,8)
```



MAT_ASS_MAIN (4/6))

```
      nodLOCAL(1)= in1
      nodLOCAL(2)= in2
      nodLOCAL(3)= in3
      nodLOCAL(4)= in4
      nodLOCAL(5)= in5
      nodLOCAL(6)= in6
      nodLOCAL(7)= in7
      nodLOCAL(8)= in8

      X1= XYZ(in1,1)
      X2= XYZ(in2,1)
      X3= XYZ(in3,1)
      X4= XYZ(in4,1)
      X5= XYZ(in5,1)
      X6= XYZ(in6,1)
      X7= XYZ(in7,1)
      X8= XYZ(in8,1)
      Y1= XYZ(in1,2)
      Y2= XYZ(in2,2)
      Y3= XYZ(in3,2)
      Y4= XYZ(in4,2)
      Y5= XYZ(in5,2)
      Y6= XYZ(in6,2)
      Y7= XYZ(in7,2)
      Y8= XYZ(in8,2)
      QVC= 08th * (X1+X2+X3+X4+X5+X6+X7+X8+
     &             Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8)
      Z1= XYZ(in1,3)
      Z2= XYZ(in2,3)
      Z3= XYZ(in3,3)
      Z4= XYZ(in4,3)
      Z5= XYZ(in5,3)
      Z6= XYZ(in6,3)
      Z7= XYZ(in7,3)
      Z8= XYZ(in8,3)

      call JACOBI (DETJ, PNQ, PNE, PNT, PNX, PNY, PZ,              &
     &             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)



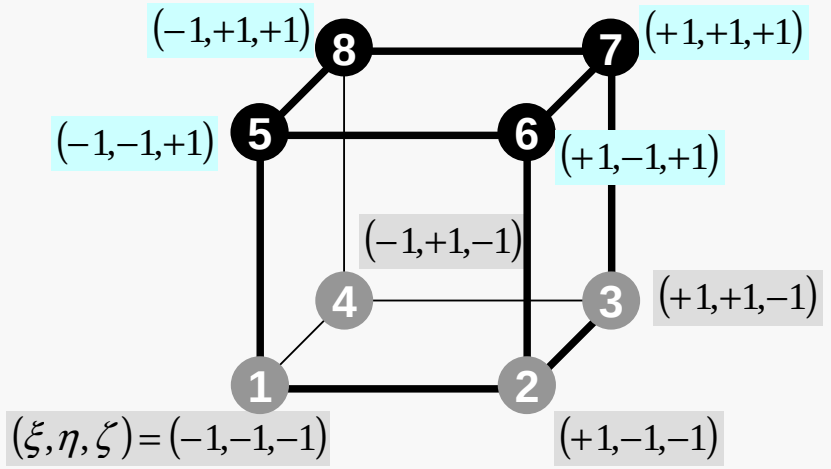
MAT_ASS_MAIN (4/6)

```
      nodLOCAL(1)= in1
      nodLOCAL(2)= in2
      nodLOCAL(3)= in3
      nodLOCAL(4)= in4
      nodLOCAL(5)= in5
      nodLOCAL(6)= in6
      nodLOCAL(7)= in7
      nodLOCAL(8)= in8

      X1= XYZ(in1,1)
      X2= XYZ(in2,1)
      X3= XYZ(in3,1)
      X4= XYZ(in4,1)
      X5= XYZ(in5,1)
      X6= XYZ(in6,1)
      X7= XYZ(in7,1)
      X8= XYZ(in8,1)
      Y1= XYZ(in1,2)
      Y2= XYZ(in2,2)
      Y3= XYZ(in3,2)
      Y4= XYZ(in4,2)
      Y5= XYZ(in5,2)
      Y6= XYZ(in6,2)
      Y7= XYZ(in7,2)
      Y8= XYZ(in8,2)
      QVC= 08th * (X1+X2+X3+X4+X5+X6+X7+X8+
     &             Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8)
      Z1= XYZ(in1,3)
      Z2= XYZ(in2,3)
      Z3= XYZ(in3,3)
      Z4= XYZ(in4,3)
      Z5= XYZ(in5,3)
      Z6= XYZ(in6,3)
      Z7= XYZ(in7,3)
      Z8= XYZ(in8,3)

      call 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 )
```

X-Coordinates of 8 nodes

Y-Coordinates of 8 nodes

Z-Coordinates of 8 nodes

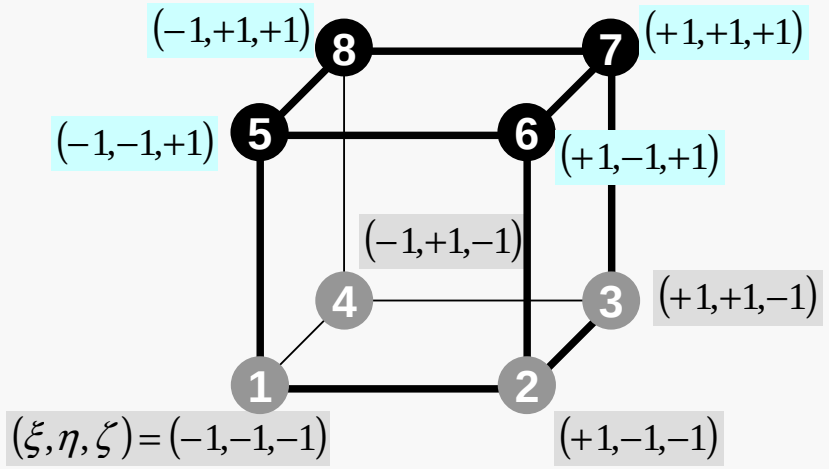


MAT_ASS_MAIN (4/6)

```
      nodLOCAL(1)= in1
      nodLOCAL(2)= in2
      nodLOCAL(3)= in3
      nodLOCAL(4)= in4
      nodLOCAL(5)= in5
      nodLOCAL(6)= in6
      nodLOCAL(7)= in7
      nodLOCAL(8)= in8

      X1= XYZ(in1,1)
      X2= XYZ(in2,1)
      X3= XYZ(in3,1)
      X4= XYZ(in4,1)
      X5= XYZ(in5,1)
      X6= XYZ(in6,1)
      X7= XYZ(in7,1)
      X8= XYZ(in8,1)
      Y1= XYZ(in1,2)
      Y2= XYZ(in2,2)
      Y3= XYZ(in3,2)
      Y4= XYZ(in4,2)
      Y5= XYZ(in5,2)
      Y6= XYZ(in6,2)
      Y7= XYZ(in7,2)
      Y8= XYZ(in8,2)
      QVC= 08th * (X1+X2+X3+X4+X5+X6+X7+X8+
     &             Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8)
      Z1= XYZ(in1,3)
      Z2= XYZ(in2,3)
      Z3= XYZ(in3,3)
      Z4= XYZ(in4,3)
      Z5= XYZ(in5,3)
      Z6= XYZ(in6,3)
      Z7= XYZ(in7,3)
      Z8= XYZ(in8,3)

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

X-Coordinates of 8 nodes

Y-Coordinates of 8 nodes



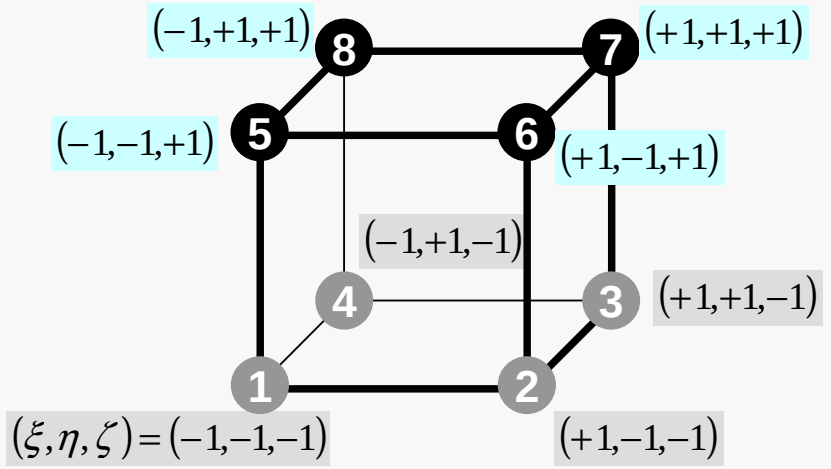
$$\frac{\partial}{\partial x}\left(\lambda\frac{\partial T}{\partial x}\right)+\frac{\partial}{\partial y}\left(\lambda\frac{\partial T}{\partial y}\right)+\frac{\partial}{\partial z}\left(\lambda\frac{\partial T}{\partial z}\right)+\dot{Q}(x,y,z)=0$$

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

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

MAT_ASS_MAIN (4/6)

```
      nodLOCAL(1)= in1
      nodLOCAL(2)= in2
      nodLOCAL(3)= in3
      nodLOCAL(4)= in4
      nodLOCAL(5)= in5
      nodLOCAL(6)= in6
      nodLOCAL(7)= in7
      nodLOCAL(8)= in8

      X1= XYZ(in1,1)
      X2= XYZ(in2,1)
      X3= XYZ(in3,1)
      X4= XYZ(in4,1)
      X5= XYZ(in5,1)
      X6= XYZ(in6,1)
      X7= XYZ(in7,1)
      X8= XYZ(in8,1)
      Y1= XYZ(in1,2)
      Y2= XYZ(in2,2)
      Y3= XYZ(in3,2)
      Y4= XYZ(in4,2)
      Y5= XYZ(in5,2)
      Y6= XYZ(in6,2)
      Y7= XYZ(in7,2)
      Y8= XYZ(in8,2)
      QVC= 08th * (X1+X2+X3+X4+X5+X6+X7+X8+
     &             Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8)
      Z1= XYZ(in1,3)
      Z2= XYZ(in2,3)
      Z3= XYZ(in3,3)
      Z4= XYZ(in4,3)
      Z5= XYZ(in5,3)
      Z6= XYZ(in6,3)
      Z7= XYZ(in7,3)
      Z8= XYZ(in8,3)

      call 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 )
```



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

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

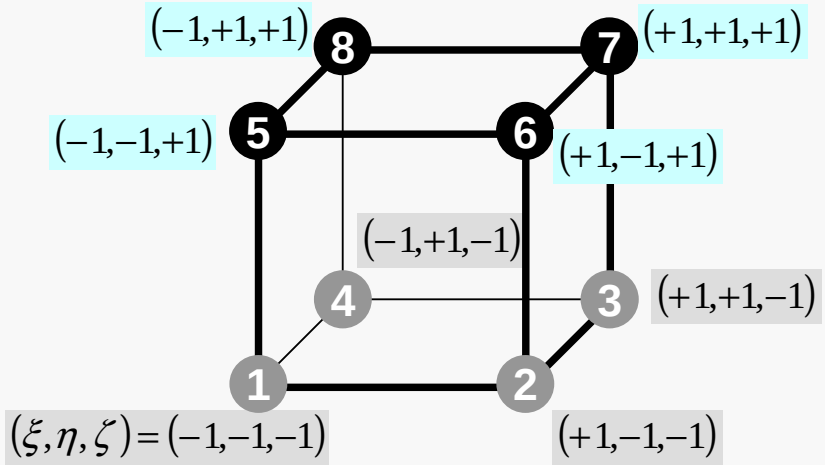
$$QVC=\left|x_C+y_C\right|$$

MAT_ASS_MAIN (4/6)

```
      nodLOCAL(1)= in1
      nodLOCAL(2)= in2
      nodLOCAL(3)= in3
      nodLOCAL(4)= in4
      nodLOCAL(5)= in5
      nodLOCAL(6)= in6
      nodLOCAL(7)= in7
      nodLOCAL(8)= in8

      X1= XYZ(in1,1)
      X2= XYZ(in2,1)
      X3= XYZ(in3,1)
      X4= XYZ(in4,1)
      X5= XYZ(in5,1)
      X6= XYZ(in6,1)
      X7= XYZ(in7,1)
      X8= XYZ(in8,1)
      Y1= XYZ(in1,2)
      Y2= XYZ(in2,2)
      Y3= XYZ(in3,2)
      Y4= XYZ(in4,2)
      Y5= XYZ(in5,2)
      Y6= XYZ(in6,2)
      Y7= XYZ(in7,2)
      Y8= XYZ(in8,2)
      QVC= 08th * (X1+X2+X3+X4+X5+X6+X7+X8+
     &             Y1+Y2+Y3+Y4+Y5+Y6+Y7+Y8)
      Z1= XYZ(in1,3)
      Z2= XYZ(in2,3)
      Z3= XYZ(in3,3)
      Z4= XYZ(in4,3)
      Z5= XYZ(in5,3)
      Z6= XYZ(in6,3)
      Z7= XYZ(in7,3)
      Z8= XYZ(in8,3)

      call 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)

```
!C
!C== CONSTRUCT the GLOBAL MATRIX

        do ie= 1, 8
          ip = nodLOCAL(ie)
        do je= 1, 8
          jp = nodLOCAL(je)

          kk= 0
          if (jp.ne.ip) then
            iiS= index(ip-1) + 1
            iiE= index(ip  )
            do k= iiS, iiE
              if ( item(k).eq.jp ) then
                kk= k
                exit
              endif
            enddo
          endif
```



Non-Zero Off-Diagonal Block in Global Matrix

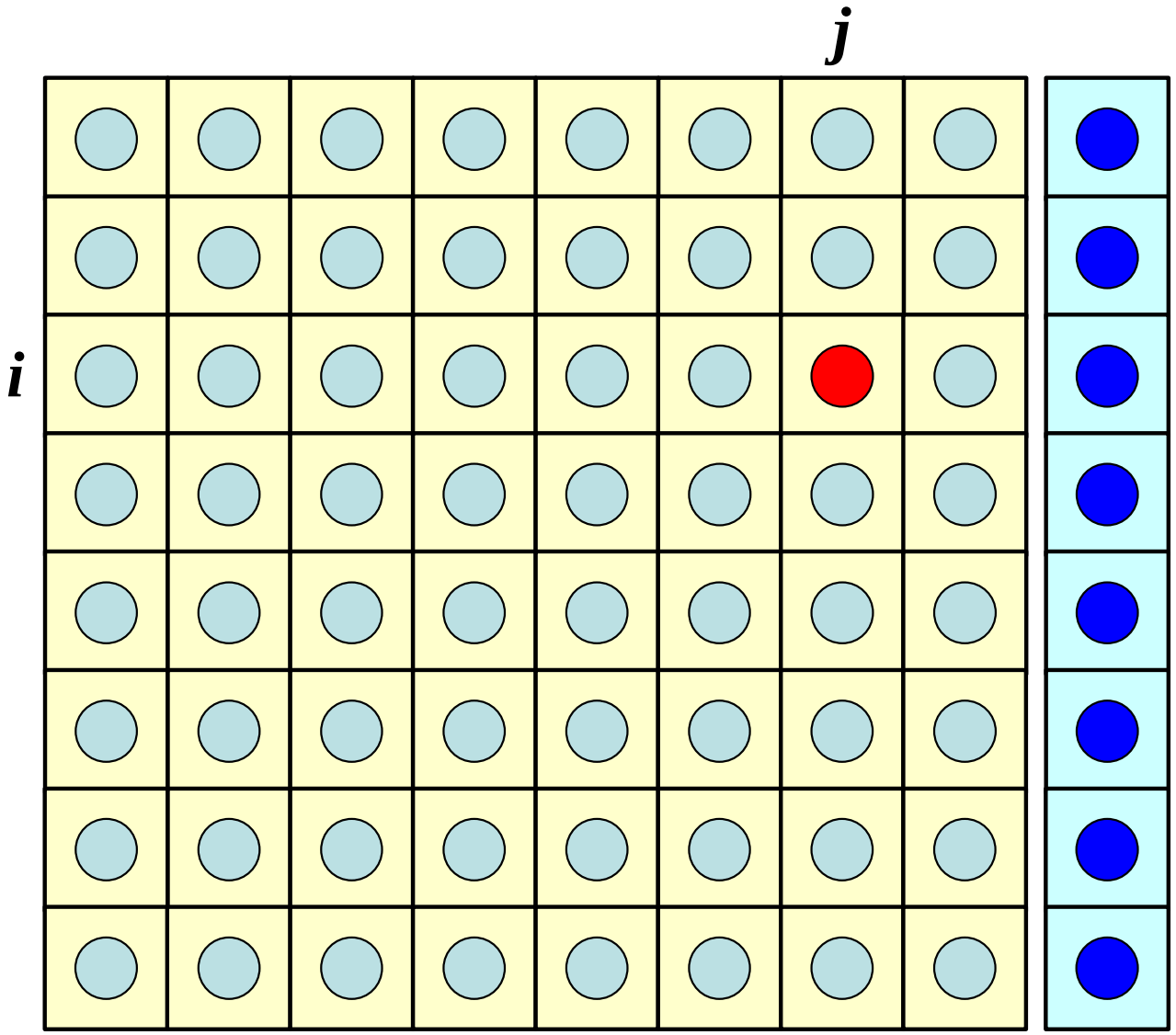
$A_{ip,jp}$

kk: address in “item”

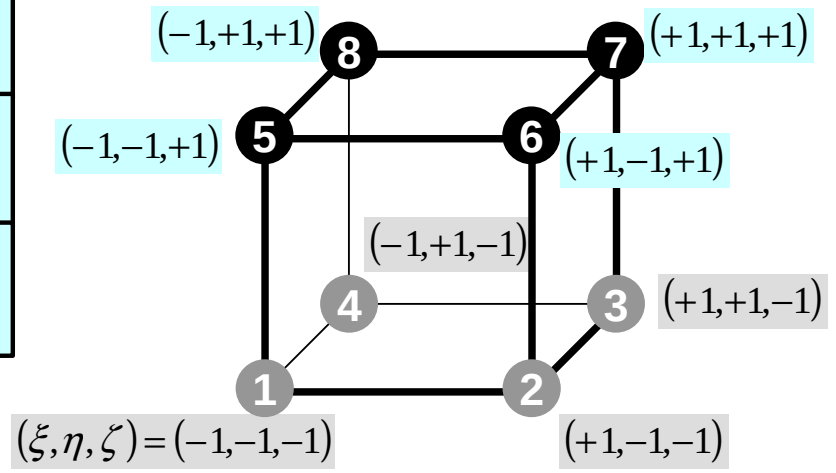
ip= nodLOCAL(ie)
jp= nodLOCAL(je)

Node ID (ip,jp) starting from 1

Element Matrix: 8x8



$$\lfloor k_{ij} \rfloor \quad (i, j = 1 \ldots 8)$$



MAT_ASS_MAIN (5/6)

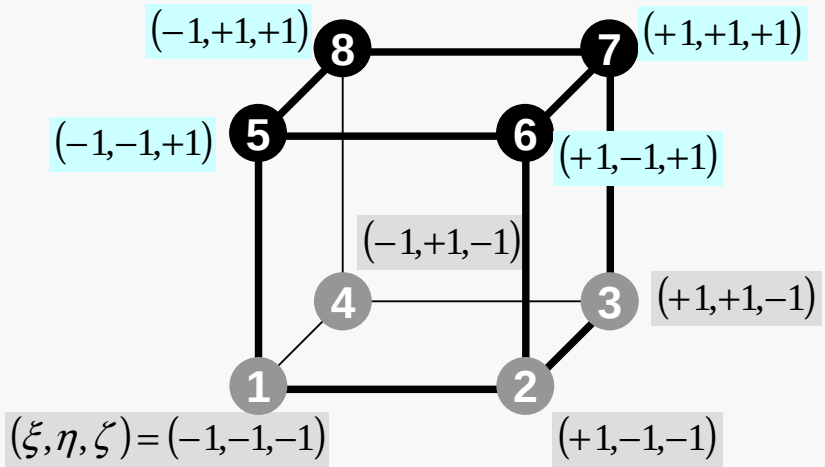
```
!C
!C== CONSTRUCT the GLOBAL MATRIX

        do ie= 1, 8
          ip = nodLOCAL(ie)
        do je= 1, 8
          jp = nodLOCAL(je)

          kk= 0
          if (jp.ne.ip) then
            iiS= index(ip-1) + 1
            iiE= index(ip  )
            do k= iiS, iiE
              if ( item(k).eq.jp ) then
                kk= k
                exit
              endif
            enddo
          endif
```

Element Matrix (i_e~j_e): Local ID
Global Matrix (i_p~j_p): Global ID

kk : address in “item” starting from “1”

k: starting from “1”

ip,jp: starting from “1”



MAT_ASS_MAIN (6/6)

```
        QV0   = 0.d0
        COEFij= 0.d0
        do kpn= 1, 2
        do jpn= 1, 2
        do ipn= 1, 2
          coef= 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= COEFij + coef * CONDO *
     &                     (PNXi*PNXj+PNYi*PNYj+PNZi*PNZj) *
     &                      DETJ(ipn,jpn,kpn)

          SHi= SHAPE(ipn,jpn,kpn,ie)
          QV0= QV0 + SHi * QVOL * coef * DETJ(ipn,jpn,kpn)
        enddo
        enddo
        enddo

        if (jp.eq.ip) then
          D(ip)= D(ip) + COEFij
          B(ip)= B(ip) + QV0*QVC
         else
          AMAT(kk)= AMAT(kk) + COEFij
        endif
      enddo
      enddo
    enddo

    return
    end
```

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

MAT_ASS_MAIN (6/6)

```
      QV0   = 0.d0
      COEFij= 0.d0
      do kpn= 1, 2
      do jpn= 1, 2
      do ipn= 1, 2
        coef= 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= COEFij + coef * CONDO *
     &                   (PNXi*PNXj+PNYi*PNYj+PNZi*PNZj) *
     &                    DETJ(ipn, jpn,kpn)

        SHi= SHAPE(ipn, jpn,kpn, ie)
        QV0= QV0 + SHi * QV0L * coef * DETJ(ipn, jpr
      enddo
      enddo
      enddo

      if (jp.eq.ip) then
        D(ip)= D(ip) + COEFij
        B(ip)= B(ip) + QV0*QVC
       else
        AMAT(kk)= AMAT(kk) + COEFij
      endif
    enddo
    enddo
  enddo

  return
  end
```

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

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

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

MAT_ASS_MAIN (6/6)

```
      do kpn= 1, 2
      do jpn= 1, 2
      do ipn= 1, 2
        coef= 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= COEFij + coef * CONDO * (PNXi*PNXj+PNYi*PNYj+PNZi*PNZj) *
     &                                  DETJ(ipn, jpn, kpn)

      enddo
      enddo
      enddo
```

$$\mathrm{coef} = W_i \cdot W_j \cdot W_k$$

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

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

MAT_ASS_MAIN (6/6)

```
      do kpn= 1, 2
      do jpn= 1, 2
      do ipn= 1, 2
        coef= 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= COEFij + coef * CONDO * (PNXi*PNXj+PNYi*PNYj+PNZi*PNZj) *
     &                                  DETJ(ipn, jpn, kpn)

      enddo
      enddo
      enddo
```

$$\mathrm{coef} = W_i \cdot W_j \cdot W_k$$

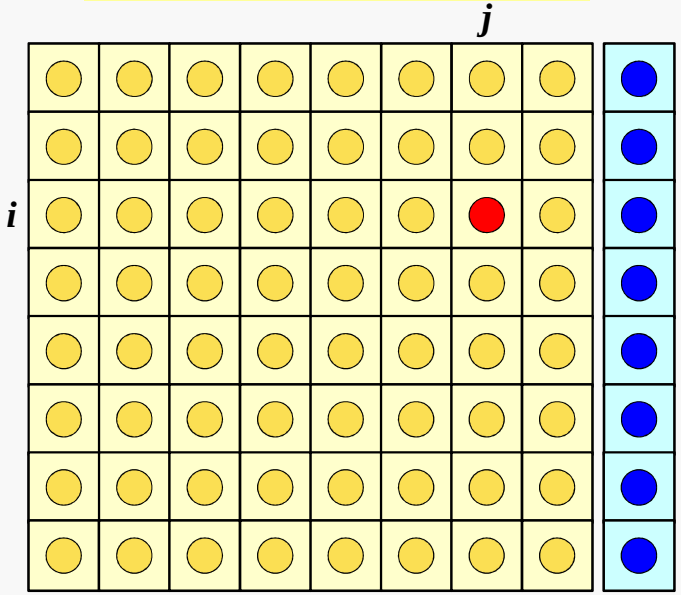
$$I = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} f(\xi,\eta,\zeta)\, d\xi d\eta d\zeta = \sum_{i=1}^{L}\sum_{j=1}^{M}\sum_{k=1}^{N} W_i \cdot W_j \cdot W_k \cdot f(\xi_i,\eta_j,\zeta_k)$$

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

MAT_ASS_MAIN (6/6)

```
      QV0   = 0.d0
      COEFij= 0.d0
      do kpn= 1, 2
      do jpn= 1, 2
      do ipn= 1, 2
        coef= 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= COEFij + coef * CONDO *
     &                   (PNXi*PNXj+PNYi*PNYj+PNZi*PNZj) *
     &                    DETJ(ipn,jpn,kpn)

        SHi= SHAPE(ipn,jpn,kpn,ie)
        QV0= QV0 + SHi * QVOL * coef * DETJ(ipn,jpn,kpn)
      enddo
      enddo
      enddo

      if (jp.eq.ip) then
        D(ip)= D(ip) + COEFij
        B(ip)= B(ip) + QV0*QVC
       else
        AMAT(kk)= AMAT(kk) + COEFij
      endif
    enddo
    enddo
  enddo

  return
  end
```

$$[k_{ij}] \quad (i,j=1\ldots8)$$



MAT_ASS_MAIN（6/6）

```
      QV0   = 0.d0
      COEFij= 0.d0
      do kpn= 1, 2
      do jpn= 1, 2
      do ipn= 1, 2
        coef= 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= COEFij + coef * CONDO *
     &                   (PNXi*PNXj+PNYi*PNYj+PNZi*PNZj) *
     &                    DETJ(ipn,jpn,kpn)

        SHi= SHAPE(ipn,jpn,kpn,ie)
        QV0= QV0 + SHi * QVOL * coef * DETJ(ipn,jpn,kpn)
      enddo
      enddo
      enddo

      if (jp.eq.ip) then
        D(ip)= D(ip) + COEFij
        B(ip)= B(ip) + QV0*QVC
       else
        AMAT(kk)= AMAT(kk) + COEFij
      endif
    enddo
    enddo
  enddo

  return
  end
```

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

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

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

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

MAT_ASS_MAIN (6/6)

```
do kpn= 1, 2
do jpn= 1, 2
do ipn= 1, 2
  coef= WEI(ipn)*WEI(jpn)*WEI(kpn)

  SHi= SHAPE(ipn,jpn,kpn,ie)
  QVO= QVO + SHi * QVOL * coef * DETJ(ipn,jpn,kpn)

enddo
enddo
enddo
```

$$\mathrm{coef} = W_i \cdot W_j \cdot W_k$$

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

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

$$\int_V QVOL[N]^T dV = \iiint QVOL[N] dxdydz = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} \{QVOL\, N_i\} \det|J|\, d\xi d\eta d\zeta$$

MAT_ASS_MAIN (6/6)

```
do kpn= 1, 2
do jpn= 1, 2
do ipn= 1, 2
  coef= WEI(ipn)*WEI(jpn)*WEI(kpn)

  SHi= SHAPE(ipn, jpn,kpn, ie)
  QVO= QVO + SHi * QVOL * coef * DETJ(ipn, jpn, kpn)

enddo
enddo
enddo
```

$$\mathrm{coef} = W_i \cdot W_j \cdot W_k$$

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

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

$$\int_V QVOL[N]^T dV = \iiint QVOL[N] dxdydz = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} \{QVOL\, N_i\} \det|J|\, d\xi d\eta d\zeta$$

MAT_ASS_MAIN（6/6）

```
      QV0   = 0.d0
      COEFij= 0.d0
      do kpn= 1, 2
      do jpn= 1, 2
      do ipn= 1, 2
        coef= 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= COEFij + coef * CONDO *
     &                   (PNXi*PNXj+PNYi*PNYj+PNZi*PNZj) *
     &                    DETJ(ipn,jpn,kpn)

        SHi= SHAPE(ipn,jpn,kpn,ie)
        QV0= QV0 + SHi * QVOL * coef * DETJ(ipn,jpn,kpn)
      enddo
      enddo
      enddo

      if (jp.eq.ip) then
        D(ip)= D(ip) + COEFij
        B(ip)= B(ip) + QV0*QVC
       else
        AMAT(kk)= AMAT(kk) + COEFij
      endif
    enddo
    enddo
  enddo

  return
  end
```

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

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

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

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

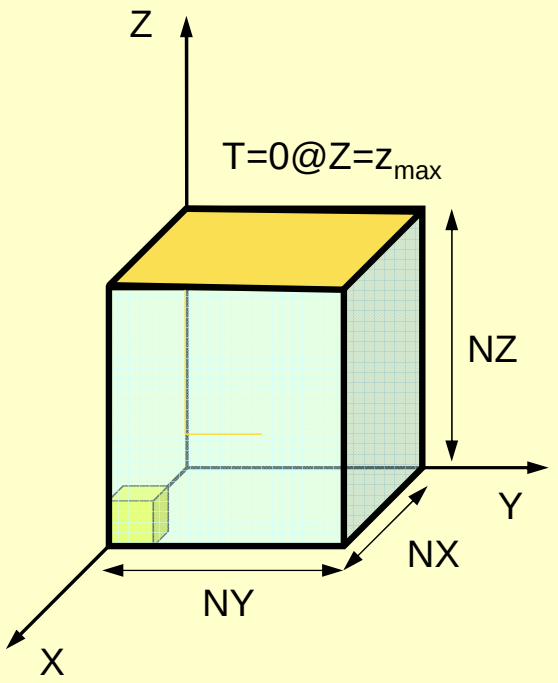
$$QVC = \left|x_C + y_C\right|$$

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

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= index(i-1)+1, index(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= index(i-1)+1, index(i)        Non-Zero Off-Diagonal Nodes
    if (IWKX(item(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)

```
      subroutine MAT_ASS_BC
      use pfem_util
      implicit REAL*8 (A-H,O-Z)

      allocate (IWKX(NP,2))
      IWKX= 0

!C
!C== Z=Zmax

      do in= 1, NP
        IWKX(in,1)= 0
      enddo

      ib0= -1
      do ib0= 1, NODGRPtot
        if (NODGRP_NAME(ib0).eq.'Zmax') exit
      enddo

      do ib= NODGRP_INDEX(ib0-1)+1, NODGRP_INDEX(ib0)
        in= NODGRP_ITEM(ib)
        IWKX(in,1)= 1
      enddo
```

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

IWKX(in,1)= 1

MAT_ASS_BC (2/2)

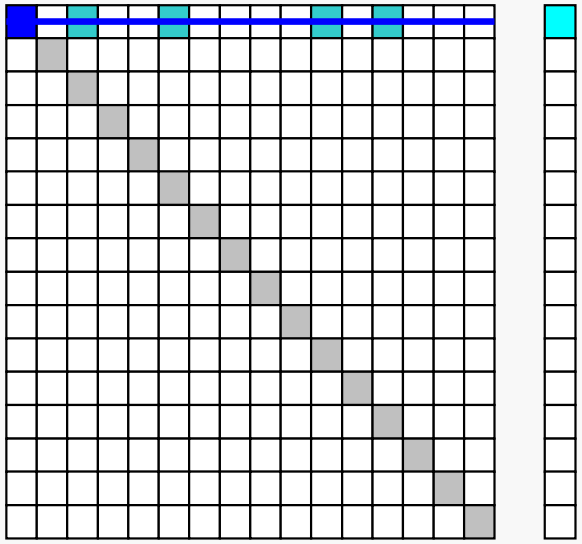
```
      do in= 1, NP
        if (IWKX(in,1).eq.1) then
          B(in)= 0.d0
          D(in)= 1.d0

          iS= index(in-1) + 1
          iE= index(in  )
          do k= iS, iE
            AMAT(k)= 0.d0
          enddo
        endif
      enddo

      do in= 1, NP
        iS= index(in-1) + 1
        iE= index(in  )
        do k= iS, iE
          if (IWKX(item(k),1).eq.1) then
            AMAT(k)= 0.d0
          endif
        enddo
      enddo
!C==
      return
      end
```

MAT_ASS_BC (2/2)

```
      do in= 1, NP
        if (IWKX(in,1).eq.1) then
          B(in)= 0.d0
          D(in)= 1.d0

          iS= index(in-1) + 1
          iE= index(in  )
          do k= iS, iE
            AMAT(k)= 0.d0
          enddo
        endif
      enddo

      do in= 1, NP
        iS= index(in-1) + 1
        iE= index(in  )
        do k= iS, iE
          if (IWKX(item(k),1).eq.1) then
            AMAT(k)= 0.d0
          endif
        enddo
      enddo
!C==
      return
      end
```

Boundary Nodes: IWKX(in,1)=1

Erase !!

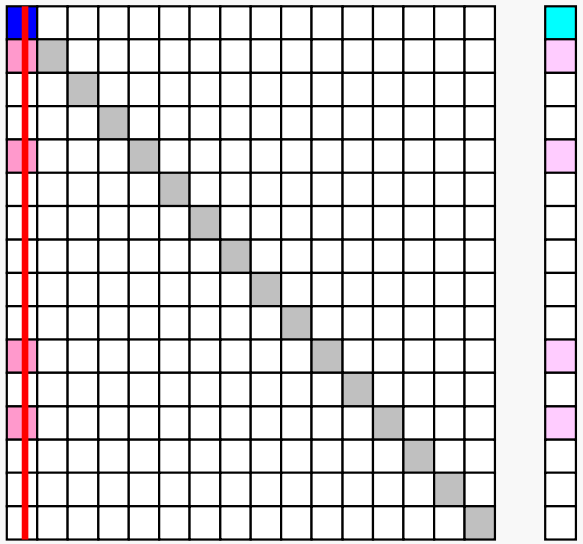


Same as 1CPU case

境界条件：MAT_ASS_BC（2/2）

```
      do in= 1, NP
        if (IWKX(in,1).eq.1) then
          B(in)= 0.d0
          D(in)= 1.d0

          iS= index(in-1) + 1
          iE= index(in  )
          do k= iS, iE
            AMAT(k)= 0.d0
          enddo
        endif
      enddo

      do in= 1, NP
        iS= index(in-1) + 1
        iE= index(in  )
        do k= iS, iE
          if (IWKX(item(k),1).eq.1) then
            AMAT(k)= 0.d0
          endif
        enddo
      enddo
!C==
      return
      end
```

Boundary Nodes: IWKX(in,1)=1



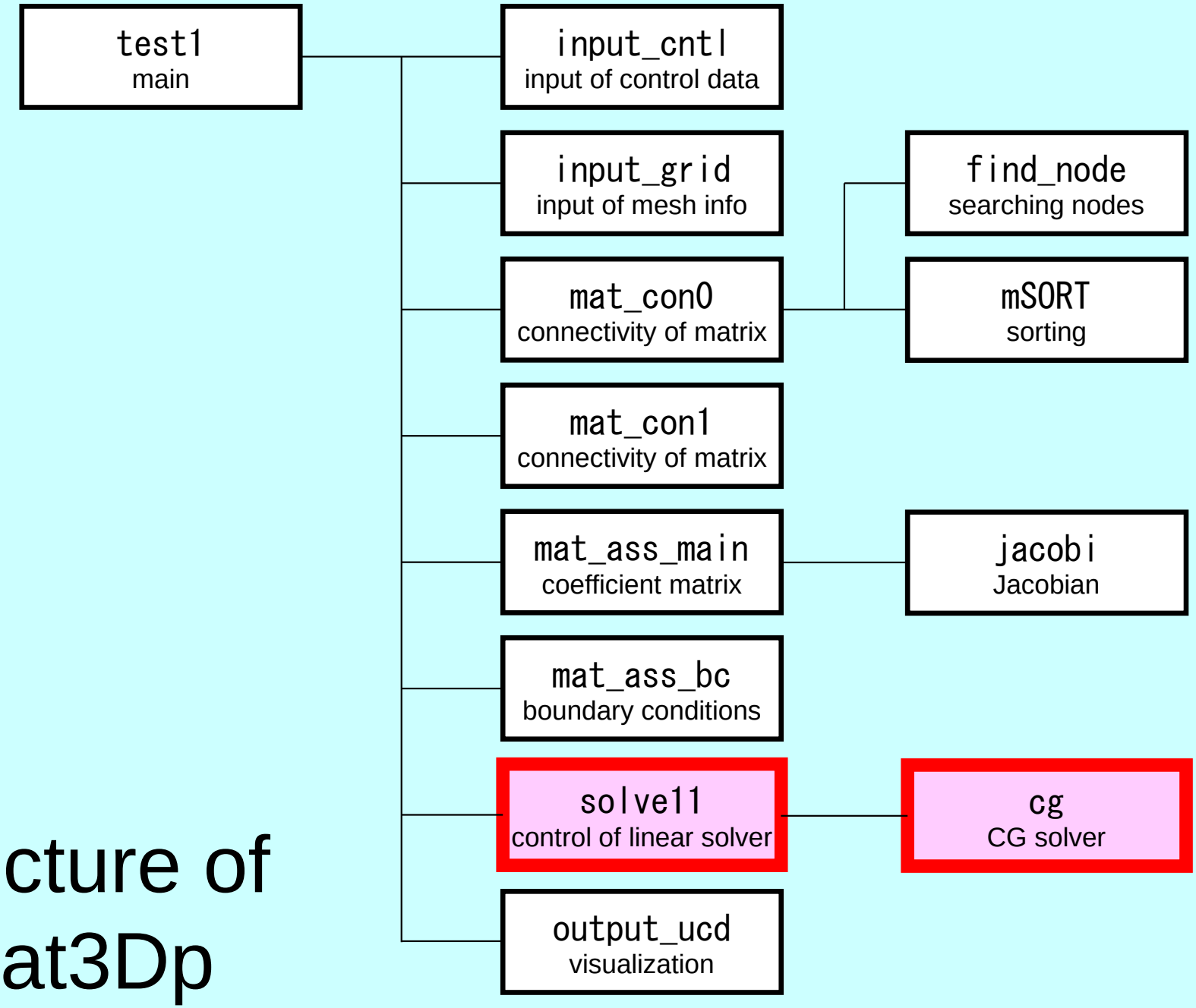
Elimination and Erase

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

```
      program heat3Dp

      use solver11
      use pfem_util

      implicit REAL*8(A-H,O-Z)

      call PFEM_INIT
      call INPUT_CNTL
      call INPUT_GRID

      call MAT_CON0
      call MAT_CON1

      call MAT_ASS_MAIN
      call MAT_ASS_BC

      call SOLVE11

      call OUTPUT_UCD

      call PFEM_FINALIZE

      end program heat3Dp
```

SOLVE11

```
      module SOLVER11

       contains
         subroutine SOLVE11

         use pfem_util
         use solver_CG

         implicit REAL*8 (A-H,O-Z)

         integer :: ERROR, ICFLAG
         character(len=char_length) :: BUF

         data ICFLAG/0/

!C
!C +------------+
!C | PARAMETERs |
!C +------------+
!C===
         ITER      = pfemIarray(1)              Max. Iterations for CG
         RESID     = pfemRarray(1)              Convergence Criteria for CG
!C===

!C
!C +-----------------+
!C | ITERATIVE solver |
!C +-----------------+
!C===
         call CG                                                          &
     &      ( N, NP, NPLU, D, AMAT, index, item, B, X,  RESID,            &
     &        ITER, ERROR, my_rank,                                       &
     &        NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_ITEM,               &
     &                           EXPORT_INDEX, EXPORT_ITEM)
       ITERactual= ITER
!C===

       end subroutine SOLVE11
       end module SOLVER11
```

Preconditioned CG Solver

Diagonal Scaling/Point Jacobi Preconditioning

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    solve [M]z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
     else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

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

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)

```
      subroutine CG                                                     &
     &   (N, NP, NPLU, D, AMAT, index, item, B, X, RESID,               &
     &    ITER, ERROR, my_rank,                                         &
     &    NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_ITEM,                 &
     &                       EXPORT_INDEX, EXPORT_ITEM)

      use  solver_SR

      implicit REAL*8(A-H,O-Z)
      include  'precision.inc'
      include  'mpif.h'

      integer(kind=kint ), intent(in):: N, NP, NPLU, my_rank
      integer(kind=kint ), intent(in):: NEIBPETOT
      integer(kind=kint ), intent(inout):: ITER, ERROR
      real   (kind=kreal), intent(inout):: RESID

      real(kind=kreal), dimension(NP)   , intent(inout):: B, X, D
      real(kind=kreal), dimension(NPLU), intent(inout):: AMAT

      integer(kind=kint ), dimension(0:NP),intent(in) :: index
      integer(kind=kint ), dimension(NPLU),intent(in) :: item

      integer(kind=kint ), pointer :: NEIBPE(:)
      integer(kind=kint ), pointer :: IMPORT_INDEX(:), IMPORT_ITEM(:)
      integer(kind=kint ), pointer :: EXPORT_INDEX(:), EXPORT_ITEM(:)

      real(kind=kreal), dimension(:),    allocatable:: WS, WR          Sending/Receiving Buffer
      real(kind=kreal), dimension(:,:),  allocatable:: WW

      integer(kind=kint), parameter ::  R= 1
      integer(kind=kint), parameter ::  Z= 2
      integer(kind=kint), parameter ::  Q= 2
      integer(kind=kint), parameter ::  P= 3
      integer(kind=kint), parameter :: DD= 4

      integer(kind=kint ) :: MAXIT
      real   (kind=kreal) :: TOL, W, SS
```

Variables/Arrays in CG Solver (1/2)

Name	Type	Size	I/O	Definition
N,NP	I		I	# Node (Internal, Internal+External)
NPLU	I		O	# Non-Zero Off-Diagonals
D	R	(NP)	O	Diagonal Block of Global Matrix
B,X	R	(NP)	O	RHS, Unknown Vector
AMAT	R	(NPLU)	O	Non-Zero Off-Diagonal Components of Global Matrix
index	I	(0:NP)	O	# Non-Zero Off-Diagonal Components
item	I	(NPLU)	O	Column ID of Non-Zero Off-Diagonal Components
ITER	I		I/O	Number of CG Iterations (MAX: In, Actual: Out)
RESID	R		I/O	Convergence Criteria (In), Final Residual Norm (Out)
MAXIT	I		-	Maximum Number of CG Iterations
TOL	R		-	Convergence Criteria
WW	R	(NP,4)	-	Work Arrays
P,Q,R,Z,DD	I		-	Vector ID for **WW** (1-4)

Variables/Arrays in CG Solver (2/2)

Name	Type	Size	I/O	Definition
PETOT	I		I	Number of PE's
my_rank	I		I	Process ID of MPI
NEIBPETOT	I		I	Number of Neighbors
NEIBPE	I	(NEIBPETOT)	I	ID of Neighbor
IMPORT_INDEX **EXPORT_INEDX**	I	(0:NEIBPETOT)	I	Size of Import/Export Arrays for Communication Table
IMPORT_ITEM	I	(NPimport)	I	Receiving Table (External Points) NPimport=IMPORT_INDEX(NEIBPETOT))
EXPORT_ITEM	I	(NPexport)	I	Sending Table (Boundary Points) NPexport=EXPORT_INDEX(NEIBPETOT))
WR, WS	R	(NP)		Receiving/Sending Buffer for Point-to-Point Communications

CG Solver (2/6)

```
 COMMtime= 0.d0
      COMPtime= 0.d0

      ERROR= 0

      allocate (WW(NP,4),WR(NP),WS(NP))

      MAXIT  = ITER
       TOL   = RESID

      X = 0.d0
      WS= 0.d0
      WR= 0.d0
!C
!C +---------------------+
!C | {r0}= {b} - [A]{xini} |
!C +---------------------+
!C===

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

      do j= 1, N
        WW(j,DD)= 1.d0/D(j)
        WVAL= B(j) - D(j)*X(j)
        do k= index(j-1)+1, index(j)
          i= item(k)
          WVAL= WVAL - AMAT(k)*X(i)
        enddo
        WW(j,R)= WVAL
      enddo

      BNRM20= 0.d0
      do i= 1, N
        BNRM20= BNRM20 + B(i)**2
      enddo
      call MPI_Allreduce (BNRM20, BNRM2, 1, MPI_DOUBLE_PRECISION,       &
     &                    MPI_SUM, MPI_COMM_WORLD, ierr)
```

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)

```
 subroutine  SOLVER_SEND_RECV                                          &
&                ( N, NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_ITEM,&
&                                        EXPORT_INDEX, EXPORT_ITEM,&
&                  WS, WR, X, my_rank)

 implicit REAL*8 (A-H,O-Z)
 include  'mpif.h'
 include  'precision.inc'
 integer(kind=kint )                 , intent(in)    ::  N
 integer(kind=kint )                 , intent(in)    ::  NEIBPETOT
 integer(kind=kint ), pointer :: NEIBPE      (:)
 integer(kind=kint ), pointer :: IMPORT_INDEX(:)
 integer(kind=kint ), pointer :: IMPORT_ITEM  (:)
 integer(kind=kint ), pointer :: EXPORT_INDEX(:)
 integer(kind=kint ), pointer :: EXPORT_ITEM  (:)
 real   (kind=kreal), dimension(N), intent(inout):: WS
 real   (kind=kreal), dimension(N), intent(inout):: WR
 real   (kind=kreal), dimension(N), intent(inout):: X
 integer                          , intent(in)    :: my_rank
 integer(kind=kint ), dimension(:,:), save, allocatable :: sta1, sta2, req1, req2
 integer(kind=kint ), save :: NFLAG
 data NFLAG/0/

 if (NFLAG.eq.0) then
   allocate (sta1(MPI_STATUS_SIZE,NEIBPETOT), sta2(MPI_STATUS_SIZE,NEIBPETOT))
   allocate (req1(NEIBPETOT), req2(NEIBPETOT))
   NFLAG= 1
 endif

 do neib= 1, NEIBPETOT
   istart= EXPORT_INDEX(neib-1)
   inum  = EXPORT_INDEX(neib  ) - istart
   do k= istart+1, istart+inum
      ii   = EXPORT_ITEM(k)
     WS(k)= X(ii)
   enddo

   call MPI_Isend (WS(istart+1), inum, MPI_DOUBLE_PRECISION,       &
&                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib),    &
&                  ierr)
 enddo
```

SOLVER_SEND_RECV (2/2)

```
      do neib= 1, NEIBPETOT
        istart= IMPORT_INDEX(neib-1)
        inum  = IMPORT_INDEX(neib  ) - istart
        call MPI_Irecv (WR(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req2(neib),    &
     &                  ierr)
      enddo

      call MPI_Waitall (NEIBPETOT, req2, sta2, ierr)

      do neib= 1, NEIBPETOT
        istart= IMPORT_INDEX(neib-1)
        inum  = IMPORT_INDEX(neib  ) - istart
        do k= istart+1, istart+inum
            ii = IMPORT_ITEM(k)
          X(ii)= WR(k)
        enddo
      enddo

      call MPI_Waitall (NEIBPETOT, req1, sta1, ierr)

      end subroutine solver_send_recv
      end module     solver_SR
```

CG Solver (3/6)

```
 do iter= 1, MAXIT

!C
!C +-----------------+
!C | {z}= [Minv]{r} |
!C +-----------------+
      do i= 1, N
        WW(i,Z)= WW(i,R) * WW(i,DD)
      enddo

!C
!C +---------------+
!C | {RHO}= {r}{z} |
!C +---------------+
      RHO0= 0.d0

      do i= 1, N
        RHO0= RHO0 + WW(i,R)*WW(i,Z)
      enddo

      call MPI_Allreduce (RHO0, RHO, 1, MPI_DOUBLE_PRECISION,
     &                    MPI_SUM, MPI_COMM_WORLD, ierr)

!C
!C +-----------------------------+
!C | {p} = {z} if      ITER=1    |
!C | BETA= RHO / RHO1  otherwise |
!C +-----------------------------+
      if ( ITER.eq.1 ) then
        do i= 1, N
          WW(i,P)= WW(i,Z)
        enddo
       else
        BETA= RHO / RHO1
        do i= 1, N
          WW(i,P)= WW(i,Z) + BETA*WW(i,P)
        enddo
      endif
```

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

CG Solver (4/6)

```
!C
!C +-------------+
!C | {q}= [A]{p} |
!C +-------------+
      call SOLVER_SEND_RECV
     &    ( NP, NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_ITEM,
     &      EXPORT_INDEX, EXPORT_ITEM, WS, WR, WW(1,P),
     &      my_rank)

      do j= 1, N
        WVAL= D(j)*WW(j,P)
        do k= index(j-1)+1, index(j)
           i= item(k)
           WVAL= WVAL + AMAT(k)*WW(i,P)
        enddo
        WW(j,Q)= WVAL
      enddo

!C
!C +--------------------+
!C | ALPHA= RHO / {p}{q} |
!C +--------------------+
      C10= 0.d0
      do i= 1, N
        C10= C10 + WW(i,P)*WW(i,Q)
      enddo
      call MPI_Allreduce (C10, C1, 1, MPI_DOUBLE_PRECISION,
     &                    MPI_SUM, MPI_COMM_WORLD, ierr)

      ALPHA= RHO / C1
```

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

CG Solver (5/6)

```
!C
!C +---------------------+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +---------------------+
      do i= 1, N
         X(i)  = X (i)    + ALPHA * WW(i,P)
        WW(i,R)= WW(i,R) - ALPHA * WW(i,Q)
      enddo

      DNRM20= 0.d0
      do i= 1, N
        DNRM20= DNRM20 + WW(i,R)**2
      enddo
      call MPI_Allreduce (DNRM20, DNRM2, 1,
     &                    MPI_DOUBLE_PRECISION,
     &                    MPI_SUM, MPI_COMM_WORLD, ierr)

      RESID= dsqrt(DNRM2/BNRM2)
        if ( RESID.le.TOL    ) exit
        if ( ITER .eq.MAXIT ) ERROR= -300

        RHO1 = RHO

      enddo
!C===

   30 continue

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

      deallocate (WW,WR,WS)

      end subroutine        CG
      end module     solver_CG
```

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|**

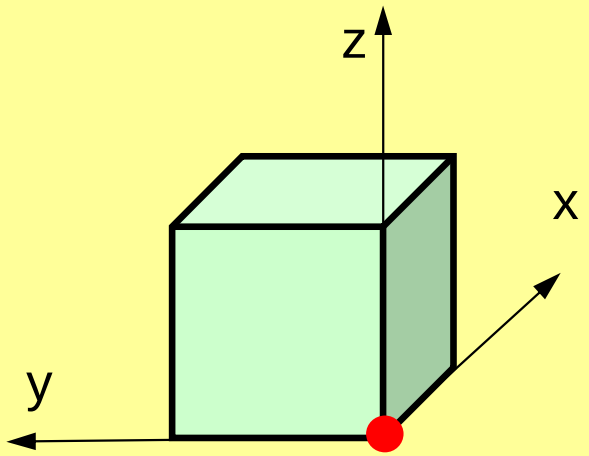
end

CG Solver (6/6)

```
!C
!C +---------------------+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +---------------------+
      do i= 1, N
         X(i)  = X (i)    + ALPHA * WW(i,P)
        WW(i,R)= WW(i,R) - ALPHA * WW(i,Q)
      enddo

      DNRM20= 0.d0
      do i= 1, N
        DNRM20= DNRM20 + WW(i,R)**2
      enddo
      call MPI_Allreduce (DNRM20, DNRM2, 1,
     &                    MPI_DOUBLE_PRECISION,
     &                    MPI_SUM, MPI_COMM_WORLD, ierr)

      RESID= dsqrt(DNRM2/BNRM2)
        if ( RESID.le.TOL    ) exit
        if ( ITER .eq.MAXIT ) ERROR= -300

        RHO1 = RHO

      enddo
!C===

   30 continue

      call SOLVER_SEND_RECV                                    & Updated temperature for external nodes

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

      deallocate (WW,WR,WS)

      end subroutine        CG
      end module     solver_CG
```

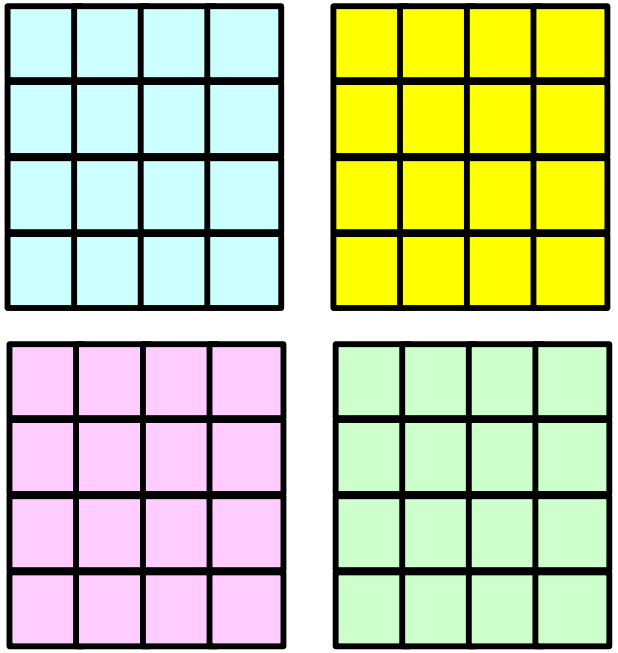
Final Output & Validation

```
 call SOLVE11
 call OUTPUT_UCD

 do i= 1, N
   if (XYZ(i,1).eq.0.d0.and.XYZ(i,2).eq.0.d0.
&                          and.XYZ(i,3).eq.0.d0) then
     write (*,'(2i8,1pe16.6)') my_rank, i, X(i)
   endif
 enddo

 call PFEM_FINALIZE

 end program heat3Dp
```



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



Time for Computation

```
!C
!C +----------------+
!C | MATRIX assemble |
!C +----------------+
!C===
      START_TIME= MPI_WTIME()

        call MAT_ASS_MAIN
        call MAT_ASS_BC

      END_TIME= MPI_WTIME()
      if (my_rank.eq.0) then
        write (*, '("*** matrix ass.  ", 1pe16.6, " sec.",/)')          &
     &         END_TIME-START_TIME
      endif
!C===

!C
!C +--------+
!C | SOLVER |
!C +--------+
!C===
      START_TIME= MPI_WTIME()
        call SOLVE11
      END_TIME= MPI_WTIME()

      if (my_rank.eq.0) then
        write (*, '("*** real  COMP.  ", 1pe16.6, " sec.",/)')          &
     &         END_TIME-START_TIME
      endif
!C===
```

Example

- (256x256x192) nodes = 12,582,912
- (255x255x191) elements = 12,419,775
- 8 nodes, 48-processes/node, 384-processes

k-MeTis (1/2): 48x8= 384 part's

```
>$ cd /work/gt36/t36XXX/pFEM/pfem3d/mesh

<modify inp_mg, mg.sh, inp_kmetis>
<modify part_kmetis.sh>

>$ pjsub mg.sh
>$ pjsub part_kmetis.sh
```

inp_mg

```
255 255 191
```

inp_kmetis

```
cube.0
2
384
aaa
```

255x255x191 elements
256x256x192 nodes
48x8= 384 partitions

k-MeTis (2/2): 48x8= 384 part's

```
>$ cd ../run
<modify INPUT.DAT, a08k.sh>

>$ pjsub a08k.sh
```

INPUT.DAT

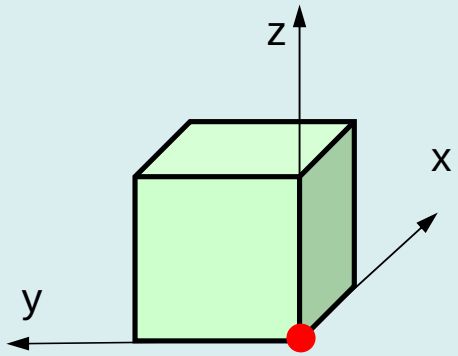
```
../mesh/aaa
2000
1.0 1.0
1.0e-08
```

a08k.sh

```
#!/bin/sh
#PJM -N "flat-08"
#PJM -L rscgrp=lecture6-o
#PJM -L node=8
#PJM --mpi proc=384
#PJM -L elapse=00:15:00
#PJM -g gt36
#PJM -j
#PJM -e err
#PJM -o a08k.lst

module load fj
module load fjmpi

mpiexec ./sol
mpiexec numactl -l ./sol
```

a08k.lst

```
*** matrix conn.      3.516576E-02 sec.
*** matrix ass.       4.487921E-02 sec.

    1    1.370663E+01
    2    1.356864E+01
    3    1.343335E+01
    4    1.330067E+01
    5    1.317050E+01
(…)
  668    1.366236E-08
  669    1.264988E-08
  670    1.162203E-08
  671    1.062147E-08
  672    9.649164E-09
*** real  COMP.       2.035819E+00 sec.

     265        1     3.526204E+06
 * normal termination
```



pmesh (1/2): 48x8= 384 part's

```
>$ cd /work/gt36/t36XXX/pFEM/pfem3d/pmesh

<modify mesh.inp, mg.sh>

>$ pjsub mg.sh
```

mesh.inp

```
256 256 192
  8   8   6
pcube
```

255x255x191 elements
256x256x192 nodes
48x8= 384 partitions

mg.sh

```
#!/bin/sh
#PJM -N "pmg"
#PJM -L rscgrp=lecture6-o
#PJM -L node=8
#PJM --mpi proc=384
#PJM -L elapse=00:10:00
#PJM -g gt36
#PJM -j
#PJM -e err
#PJM -o pmg.lst

module load fj
module load fjmpi

mpiexec ./pmesh
rm wk.*
```

pmesh (2/2): 48x8= 384 part's

```
>$ cd ../run
<modify INPUT.DAT, a08.sh>

>$ pjsub a08.sh
```

INPUT.DAT

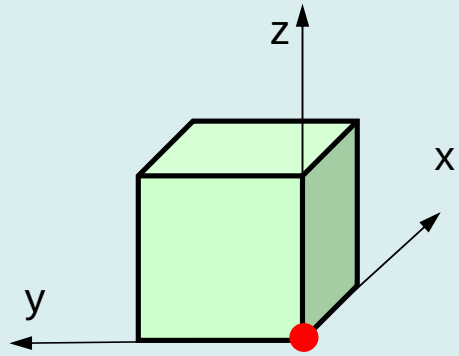
```
../pmesh/pcube
2000
1.0 1.0
1.0e-08
```

a08.sh

```
#!/bin/sh
#PJM -N "flat-08"
#PJM -L rscgrp=lecture6-o
#PJM -L node=8
#PJM --mpi proc=384
#PJM -L elapse=00:15:00
#PJM -g gt36
#PJM -j
#PJM -e err
#PJM -o a08.lst

module load fj
module load fjmpi

mpiexec ./sol
mpiexec numactl -l ./sol
```

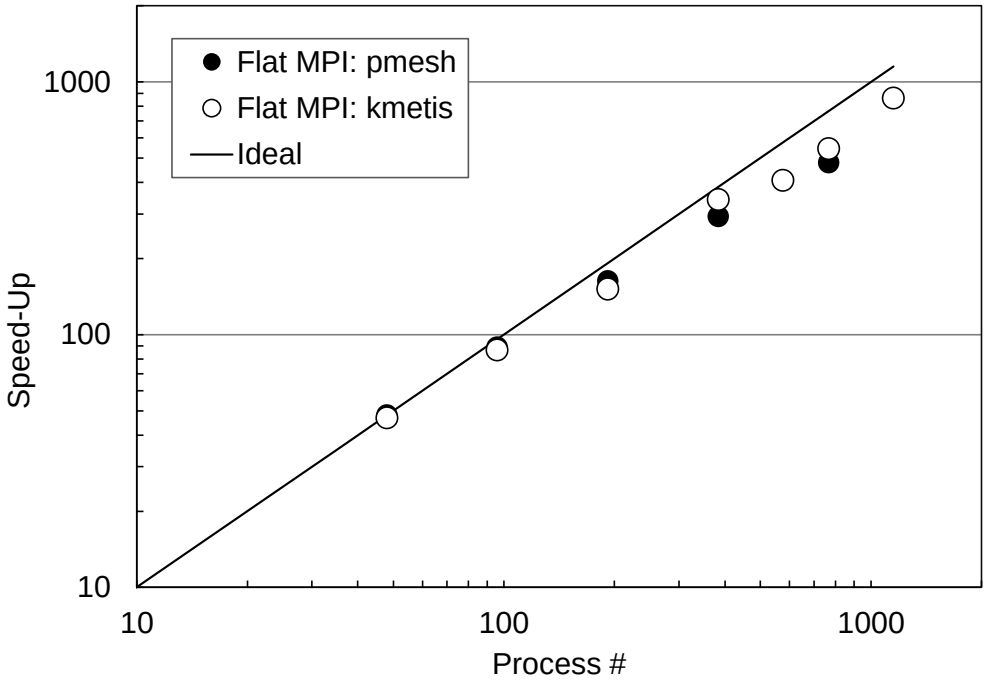
a08.lst

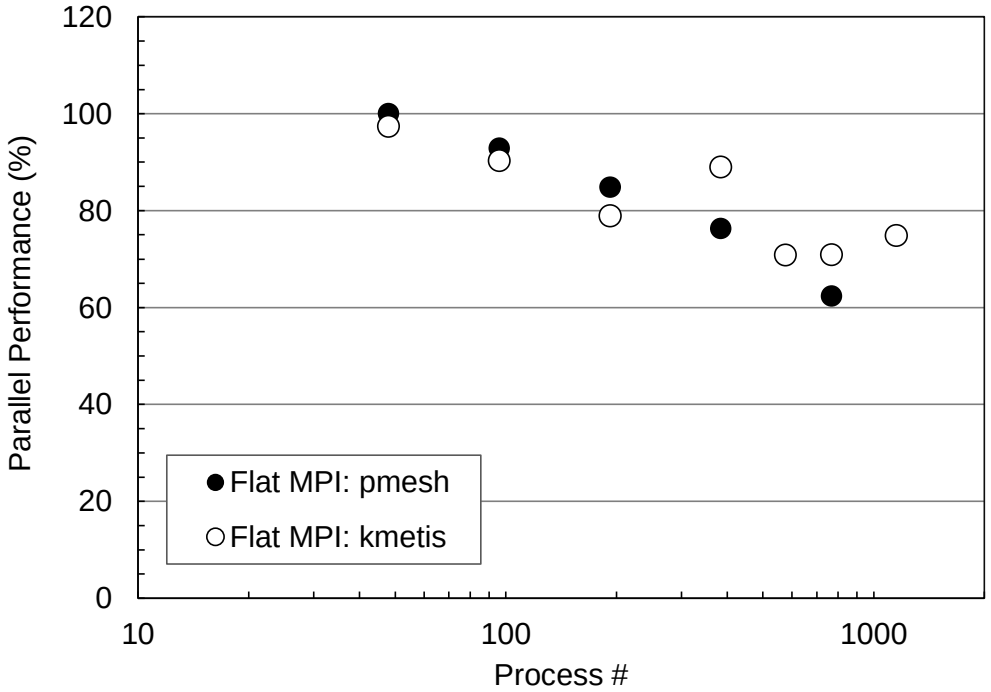
```
*** matrix conn.      3.516576E-02 sec.
*** matrix ass.       4.487921E-02 sec.

    1    1.370663E+01
    2    1.356864E+01
    3    1.343335E+01
    4    1.330067E+01
    5    1.317050E+01
(…)
  668    1.366236E-08
  669    1.264988E-08
  670    1.162203E-08
  671    1.062147E-08
  672    9.649164E-09
*** real  COMP.        2.058311E+00 sec.

        0       1     3.526204E+06
 * normal termination
```



Example: Strong Scaling: C

- 256×256×192 nodes, 12,582,912 DOF
- 48～1,152 cores (1~24 nodes), Flat MPI
- Linear Solver

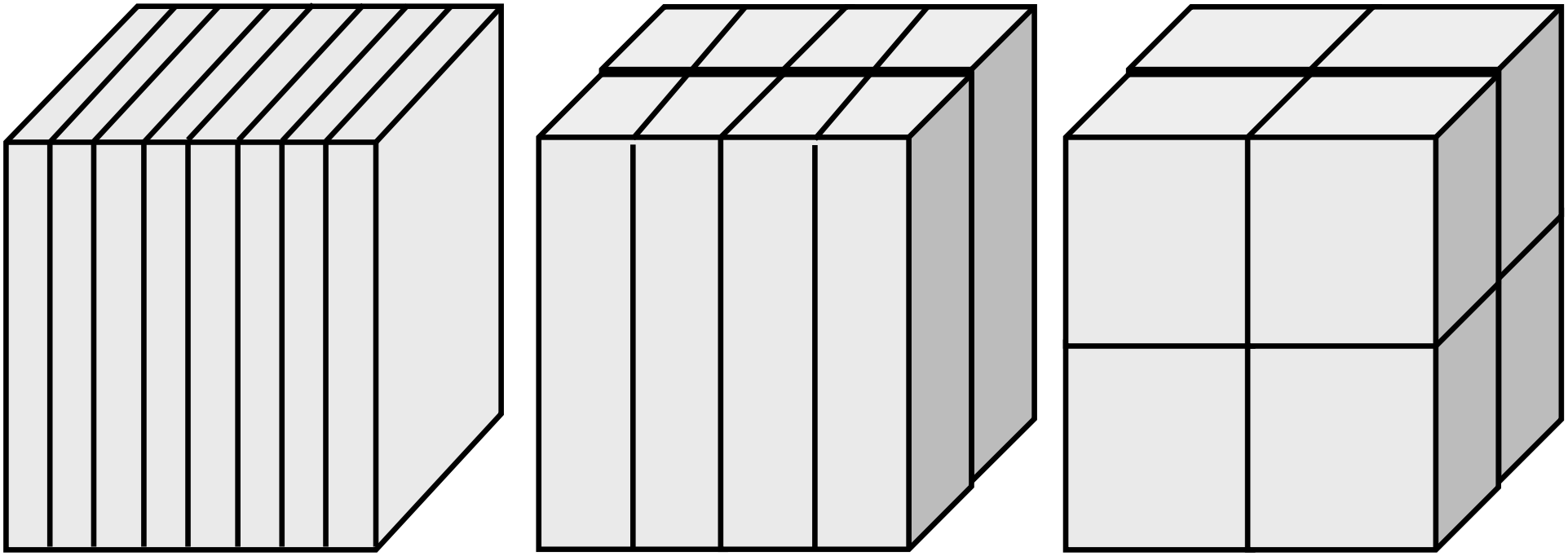




Exercise (1/2)

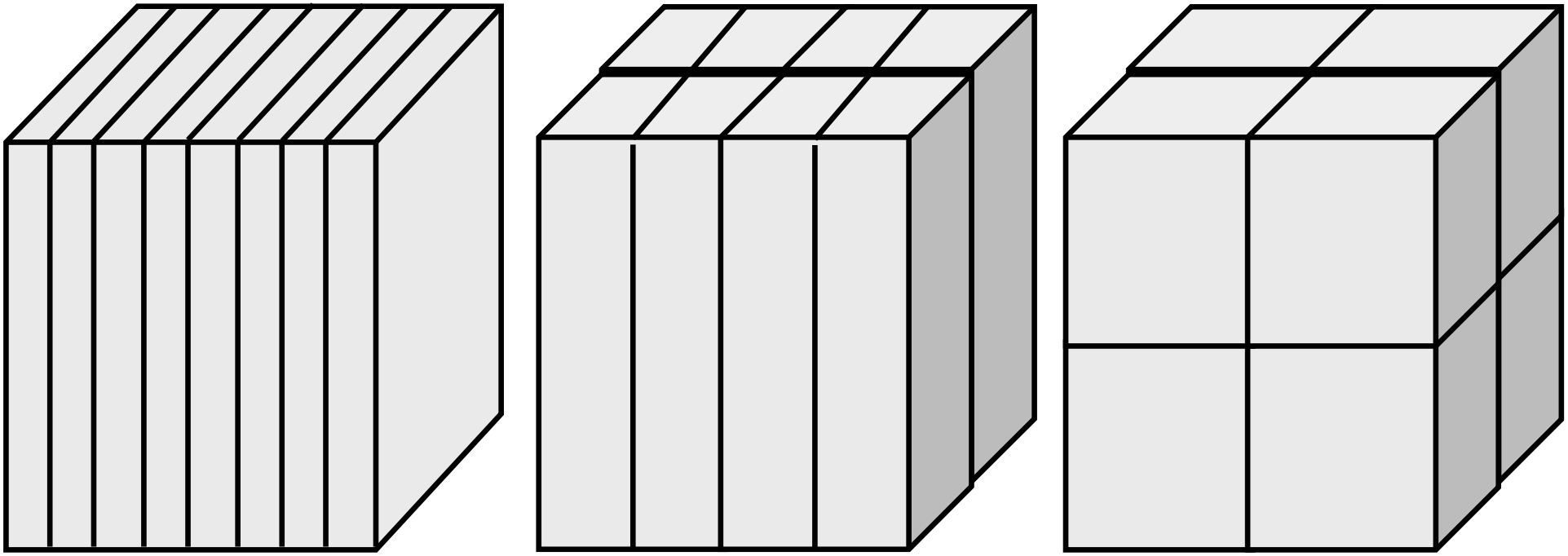
- Evaluation behavior and performance of “sol”
- Example
  - Strong Scaling
    - Fixed entire problem size
  - Weak Scaling
    - Fixed problem size/core, time for 1 iterations
  - Parameters
    - Problem size
    - Domain decomposition (1D-3D, kmetis, pmetis)
- “*.inp” may take long time.
  - delete “call OUTPUT_UCD”
  - src, part

1D-3D Decomposition



1D-3D Decomposition

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



1D 2D 3D

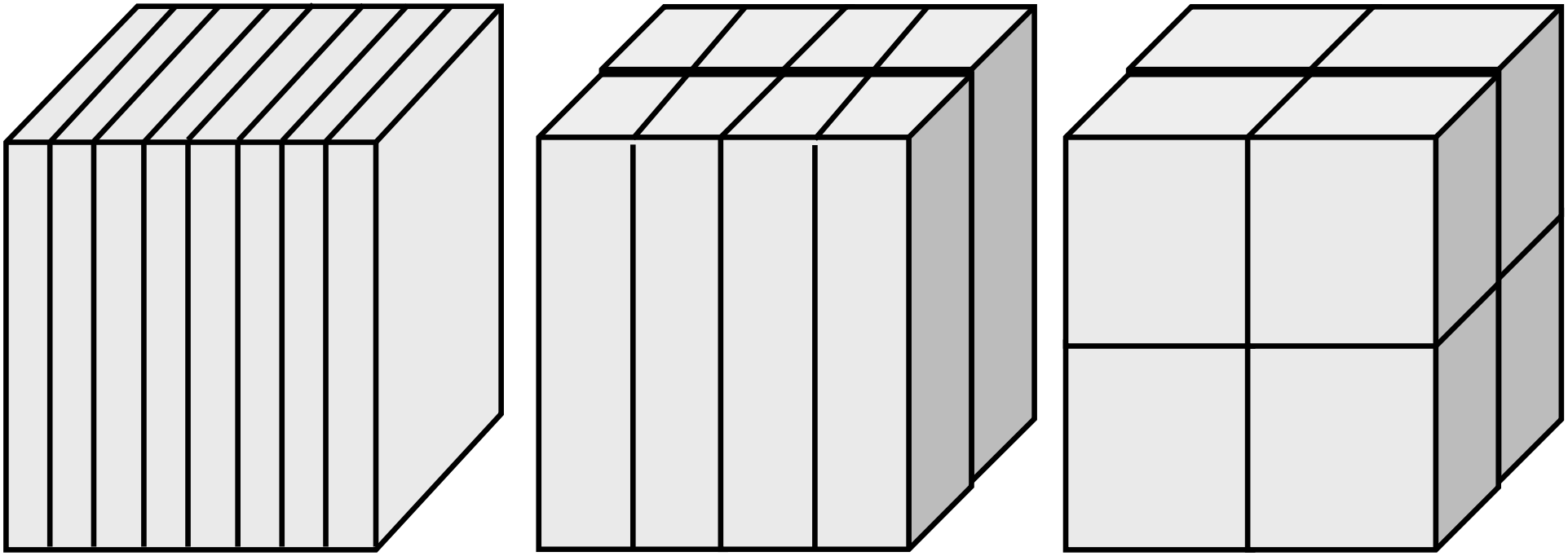
16 N^2 x 7 = 112 N^2

16 N^2 x 4 = 64 N^2

16 N^2 x 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)

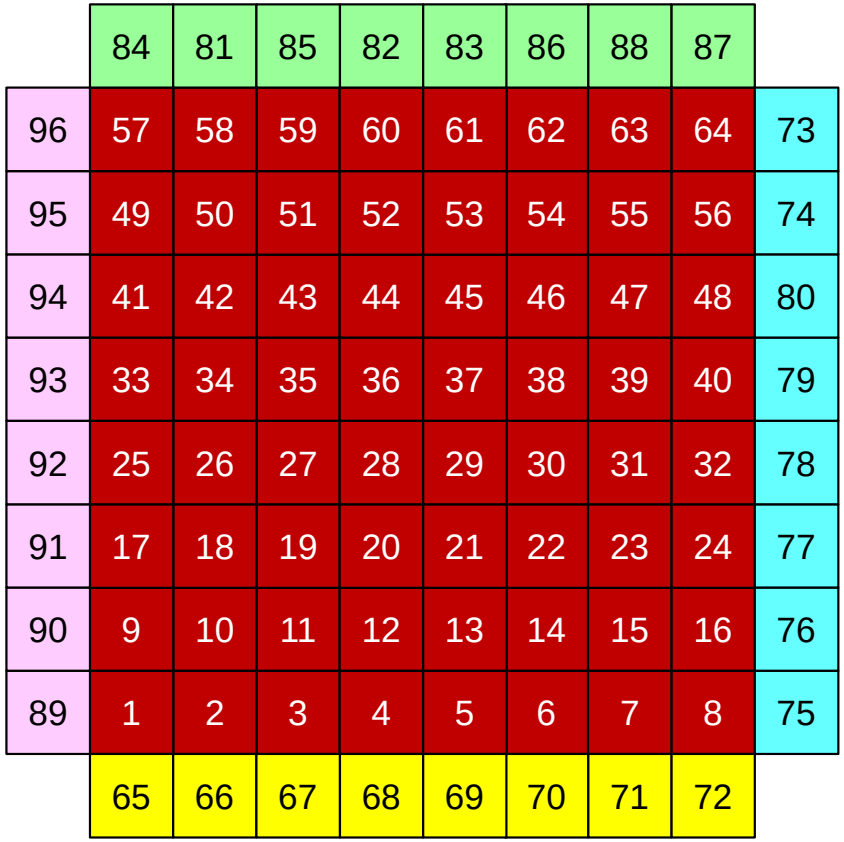
- Improve PE-to-PE communication part (solver_SR)
  - Copying to receiving buffer, Combining MPI_Wait_all
- Actually, numbering of external nodes in each neighboring domain is continuous
- You can also apply this to 1D case

```
      do neib= 1, NEIBPETOT
        istart= IMPORT_INDEX(neib-1)
        inum  = IMPORT_INDEX(neib  ) - istart
        call MPI_Irecv (WR(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req2(neib),   &
     &                  ierr)
      enddo

      call MPI_Waitall (NEIBPETOT, req2, sta2, ierr)

      do neib= 1, NEIBPETOT
        istart= IMPORT_INDEX(neib-1)
        inum  = IMPORT_INDEX(neib  ) - istart
        do k= istart+1, istart+inum
            ii = IMPORT_ITEM(k)
          X(ii)= WR(k)
        enddo
      enddo
```

SEND/RECV (Original)

```
!C
!C-- INIT.
      allocate (sta1(MPI_STATUS_SIZE,NEIBPETOT), sta2(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (req1(NEIBPETOT), req2(NEIBPETOT))

!C
!C-- SEND
      do neib= 1, NEIBPETOT
        istart= EXPORT_INDEX(neib-1)
        inum  = EXPORT_INDEX(neib  ) - istart
        do k= istart+1, istart+inum
          WS(k)= X(EXPORT_ITEM(k))
        enddo
        call MPI_ISEND (WS(istart+1), inum, MPI_DOUBLE_PRECISION,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib), ierr)
      enddo

!C
!C-- RECEIVE
      do neib= 1, NEIBPETOT
        istart= IMPORT_INDEX(neib-1)
        inum  = IMPORT_INDEX(neib  ) - istart
        call MPI_IRECV (WR(istart+1), inum, MPI_DOUBLE_PRECISION,           &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req2(neib), ierr)
      enddo
      call MPI_WAITALL (NEIBPETOT, req2, sta2, ierr)

      do neib= 1, NEIBPETOT
        istart= IMPORT_INDEX(neib-1)
        inum  = IMPORT_INDEX(neib  ) - istart
        do k= istart+1, istart+inum
          X(IMPORT_ITEM(k))= WR(k)
        enddo
      enddo
      call MPI_WAITALL (NEIBPETOT, req1, sta1, ierr)
```

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



SEND/RECV (NEW:1)

```
!C
!C-- INIT.
      allocate (sta1(MPI_STATUS_SIZE,2*NEIBPETOT))
      allocate (req1(2*NEIBPETOT))

!C
!C-- SEND
      do neib= 1, NEIBPETOT
        istart= EXPORT_INDEX(neib-1)
        inum  = EXPORT_INDEX(neib  ) - istart
        do k= istart+1, istart+inum
          WS(k)= X(EXPORT_ITEM(k))
        enddo
      enddo

      do neib= 1, NEIBPETOT
        istart= EXPORT_INDEX(neib-1)
        inum  = EXPORT_INDEX(neib  ) - istart
        call MPI_ISEND (WS(istart+1), inum, MPI_DOUBLE_PRECISION,                    &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib), ierr)
      enddo

!C
!C-- RECEIVE
      do neib= 1, NEIBPETOT
        inum  = IMPORT_INDEX(neib) - STACK_IMPORT(neib-1)
        istart= IMPORT_ITEM(IMPORT_INDEX(neib-1)+1)

        call MPI_IRECV (X(istart), inum, MPI_DOUBLE_PRECISION,                       &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(NEIBPETOT+neib), ierr)
      enddo

      call MPI_WAITALL (2*NEIBPETOT, req1, sta1, ierr)
```

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

```
!C
!C-- INIT.
      allocate (sta1(MPI_STATUS_SIZE,2*NEIBPETOT))
      allocate (req1(2*NEIBPETOT))

!C
!C-- SEND
      do neib= 1, NEIBPETOT
        istart= EXPORT_INDEX(neib-1)
        inum  = EXPORT_INDEX(neib  ) - istart
        do k= istart+1, istart+inum
          WS(k)= X(EXPORT_ITEM(k))
        enddo
      enddo

      do neib= 1, NEIBPETOT
        istart= EXPORT_INDEX(neib-1)
        inum  = EXPORT_INDEX(neib  ) - istart
        call MPI_ISEND (WS(istart+1), inum, MPI_DOUBLE_PRECISION,                  &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib), ierr)
      enddo

!C
!C-- RECEIVE
      do neib= 1, NEIBPETOT
        inum  = IMPORT_INDEX(neib) - IMPORT_INDEX(neib-1)
        istart= IMPORT_INDEX(neib-1) + N0 + 1

        call MPI_IRECV (X(istart), inum, MPI_DOUBLE_PRECISION,                     &
     &                  NEIBPE(neib), 0, MPI_COMM_WORLD, req1(NEIBPETOT+neib), ierr)
      enddo

      call MPI_WAITALL (2*NEIBPETOT, req1, sta1, ierr)
```

N0: Total Number of Internal Nodes