Introduction to Parallel Programming for Multicore/Manycore Clusters

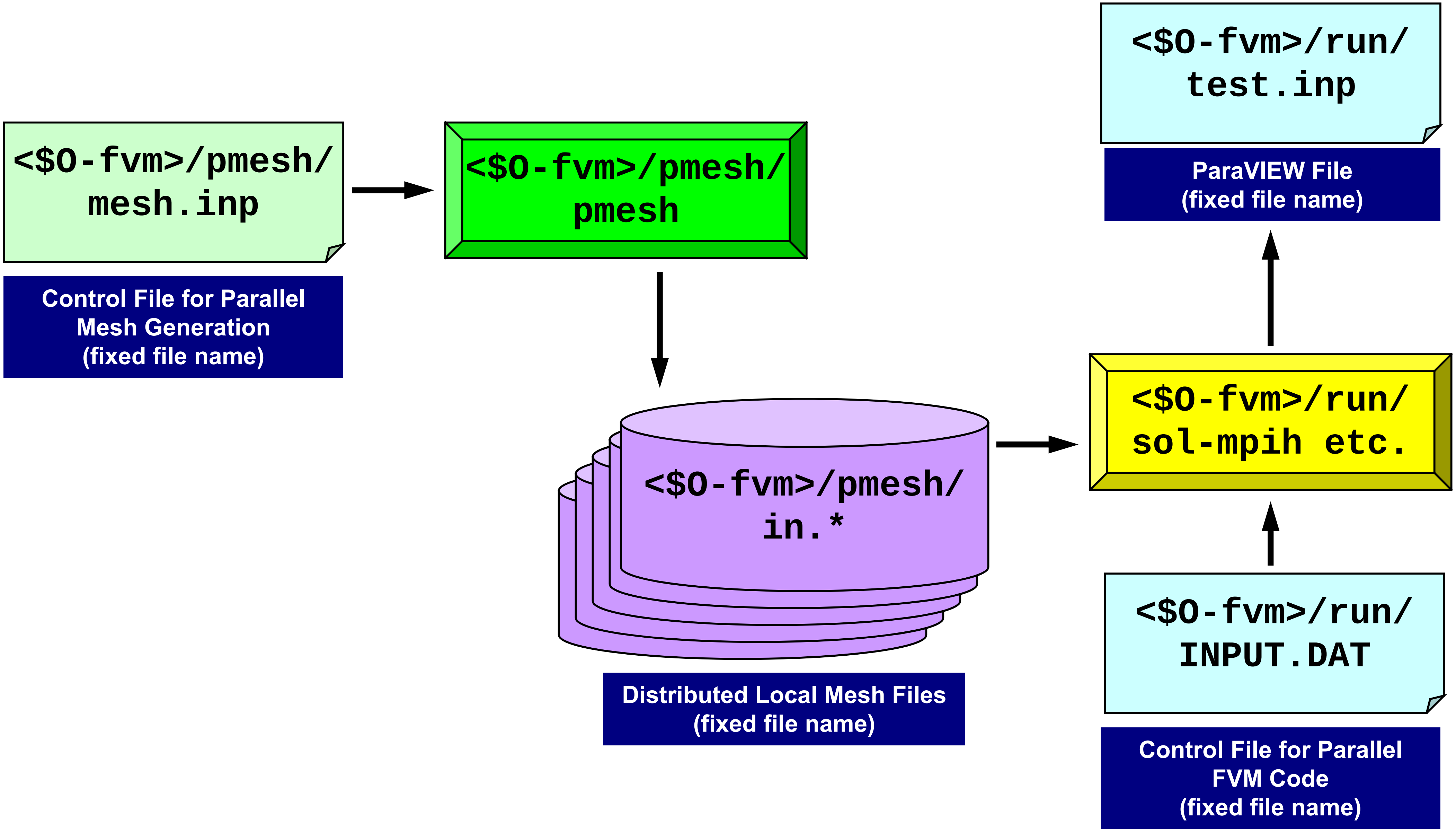
Part II-4: OpenMP/MPI Hybrid

Kengo Nakajima
Information Technology Center
The University of Tokyo

Overview

- Parallel Distributed Data Structure
- Parallel FVM Code
  - Parallel Visuzlization
- Parallel Performance
  - Super-Linear in Strong Scaling

Procedures for Parallel FVM



<$O-fvm>/pmesh/mesh.inp

Control File for Parallel Mesh Generation (fixed file name)

<$O-fvm>/pmesh/pmesh

<$O-fvm>/pmesh/in.*

Distributed Local Mesh Files (fixed file name)

<$O-fvm>/run/sol-mpih etc.

<$O-fvm>/run/test.inp

ParaVIEW File (fixed file name)

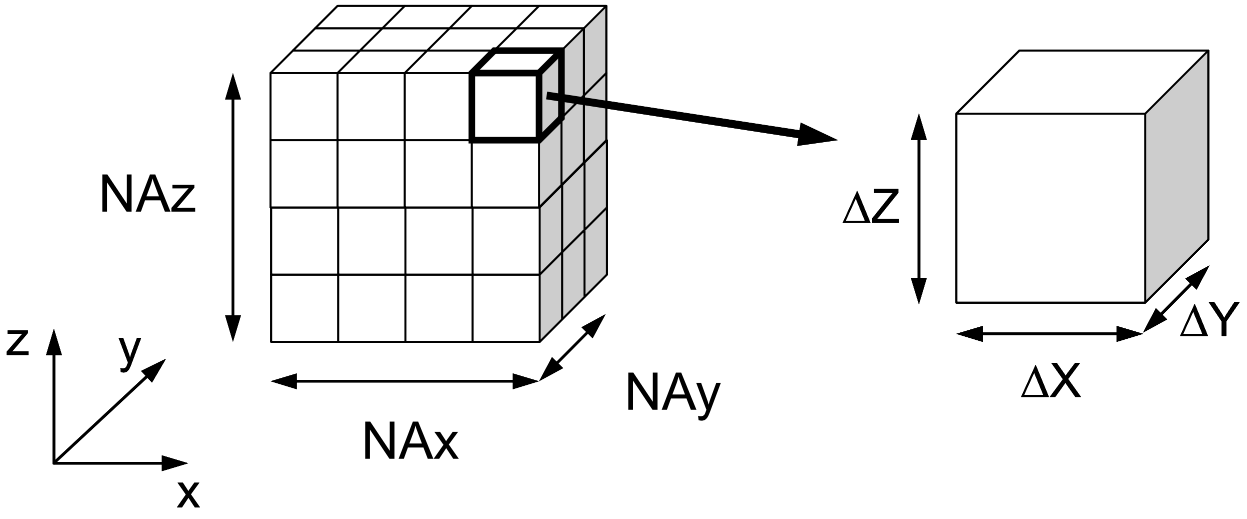
<$O-fvm>/run/INPUT.DAT

Control File for Parallel FVM Code (fixed file name)

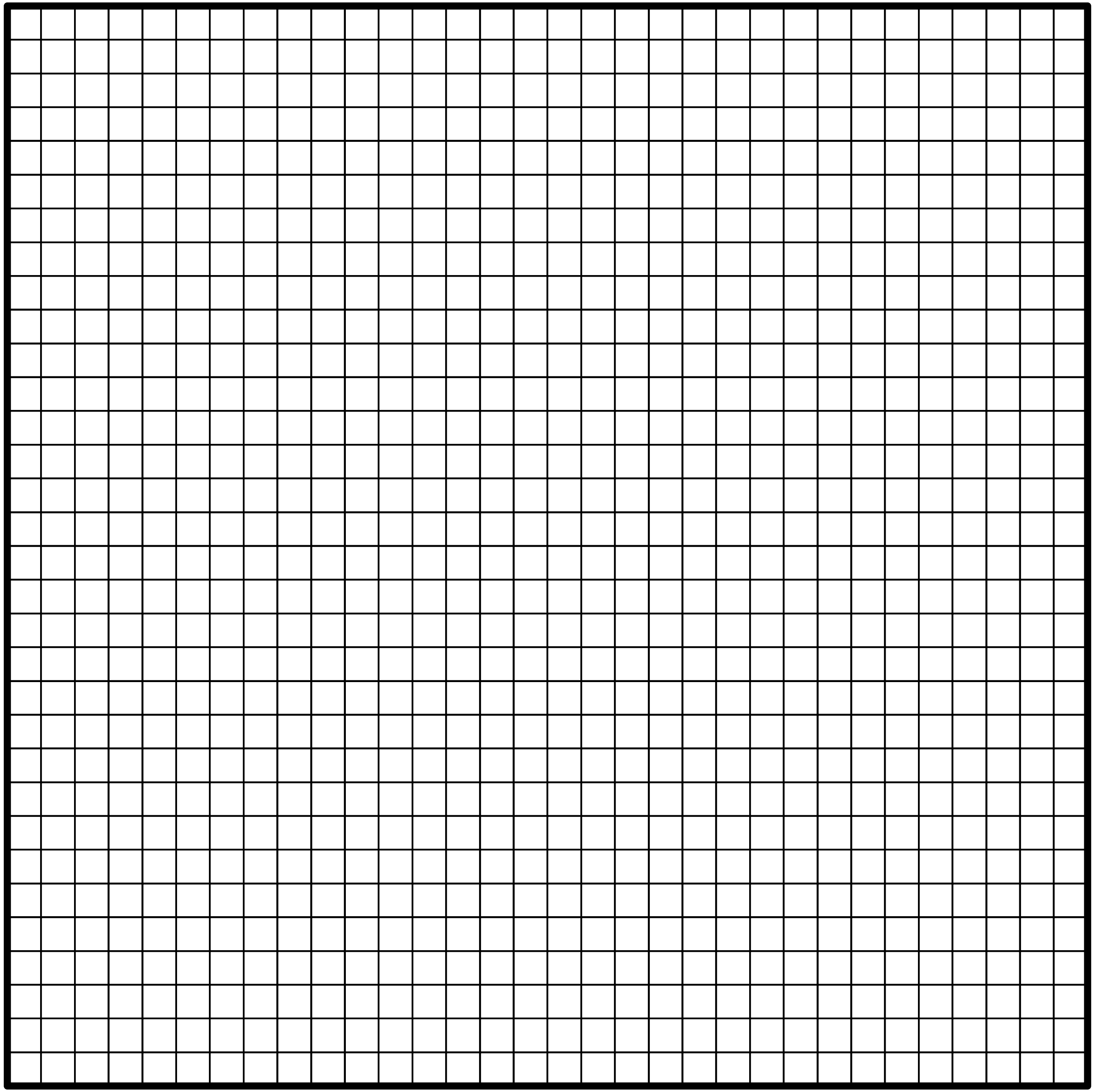
“mesh.inp”: parallel mesh generation

```
(values)  (variables) (descriptions)

 32  32  1  NAx,NAy,NAz   Total number of meshes in X-, Y-, and Z-directions
  4  4   1  npx,npy,npz   Partition # in each direction (X,Y,Z)
```

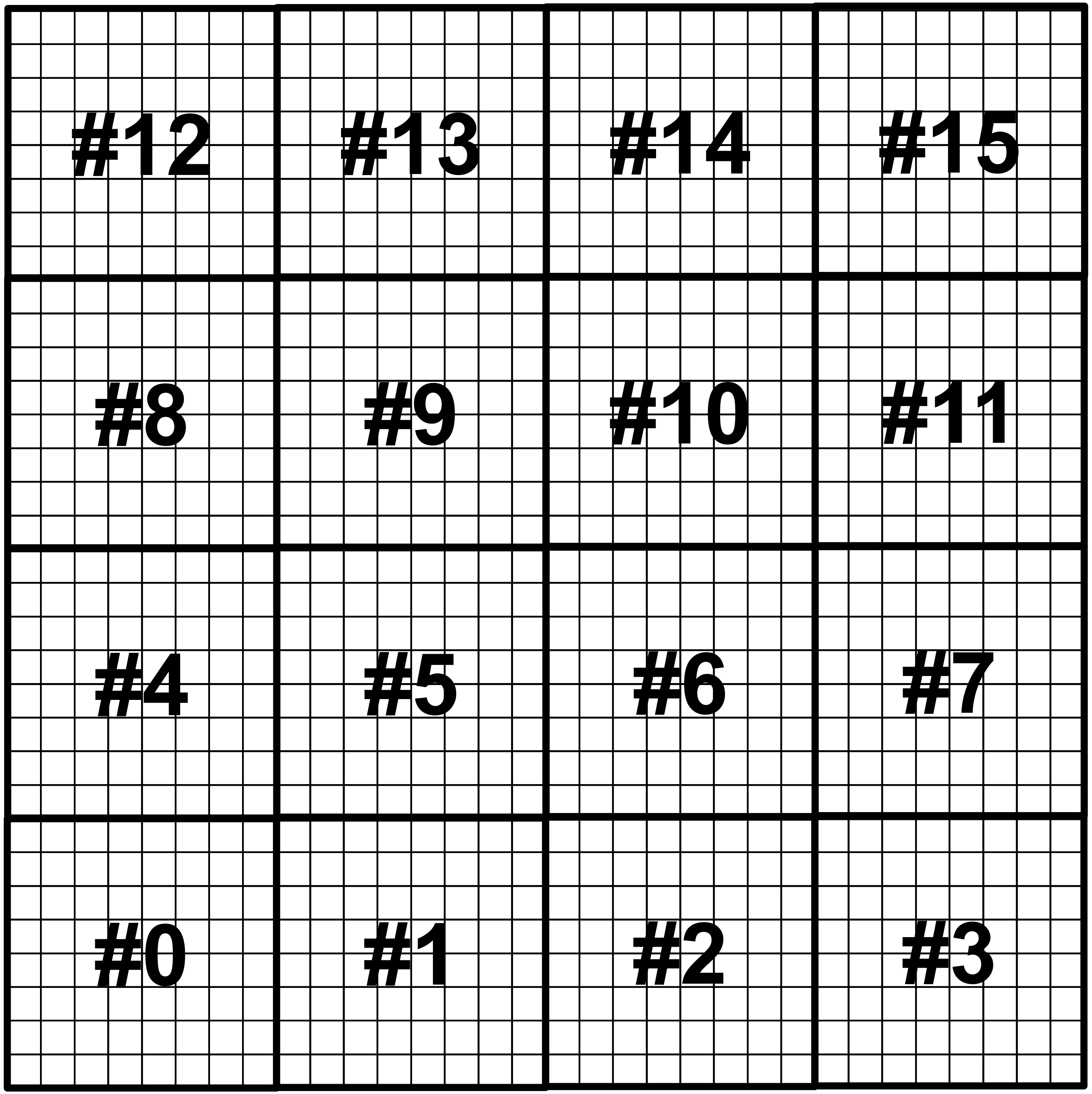
- Each of “NAx,NAy,NAz” must be “divisible（割り切れる）” by each of “npx,npy,npz”
- MPI process # = npx×npy×npz
  - In this case
  - 32x32x1 meshes,
  - 4x4x1= 16 partitions
  - 8x8x1= 64 meshes for each partition



Entire Mesh: 32x32x1



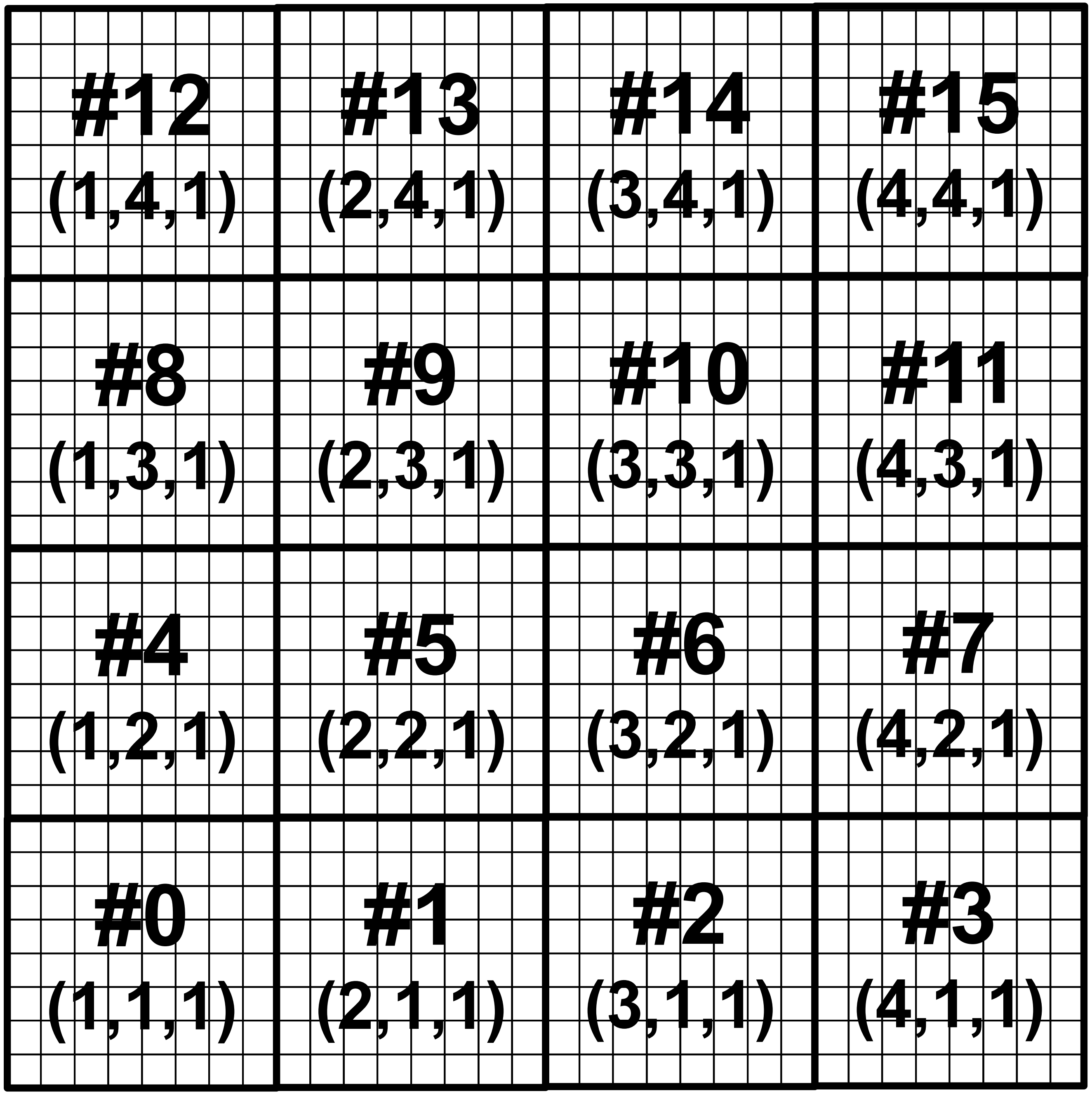
Divided into: 4x4x1 MPI Processes



<u>mesh.inp</u>

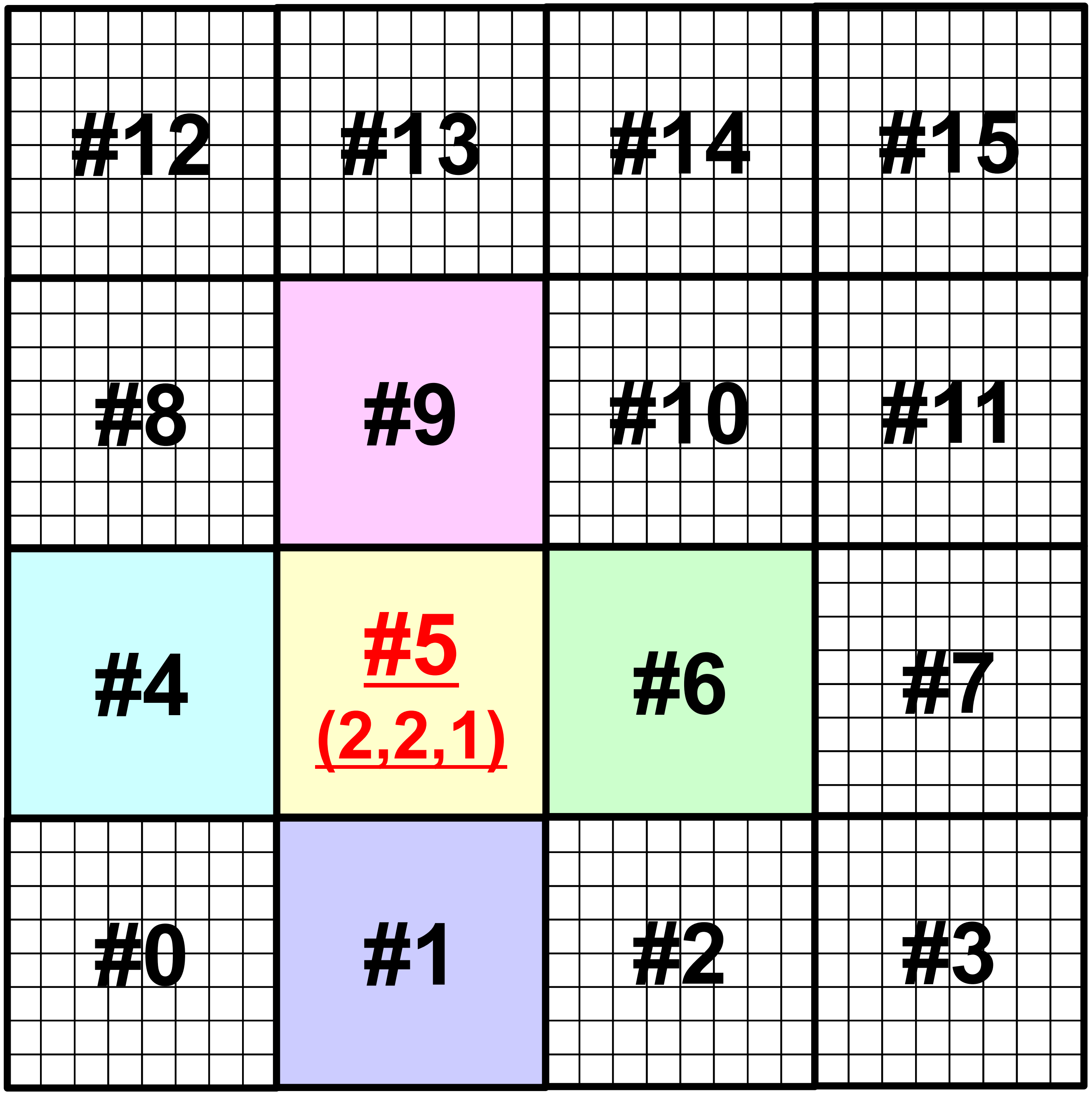
```
32  32   1
 4   4   1
```

Divided into: 4x4x1 MPI "Global Location" of Processes (NPi,NPj,NPk)

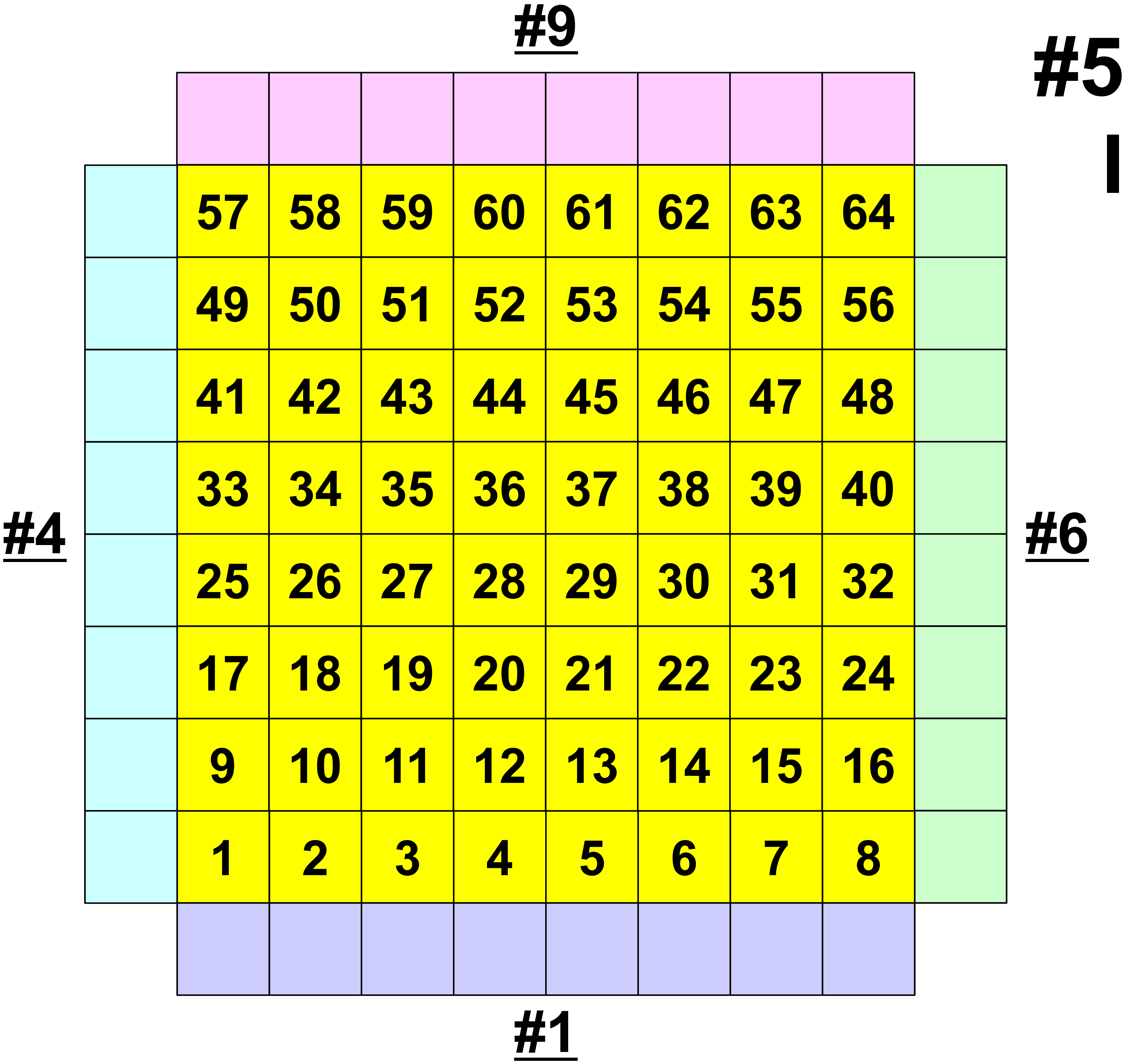


#12 (1,4,1)	#13 (2,4,1)	#14 (3,4,1)	#15 (4,4,1)
#8 (1,3,1)	#9 (2,3,1)	#10 (3,3,1)	#11 (4,3,1)
#4 (1,2,1)	#5 (2,2,1)	#6 (3,2,1)	#7 (4,2,1)
#0 (1,1,1)	#1 (2,1,1)	#2 (3,1,1)	#3 (4,1,1)

Divided into: 4x4x1 MPI Processes



#5 Process Internal Nodes



#9

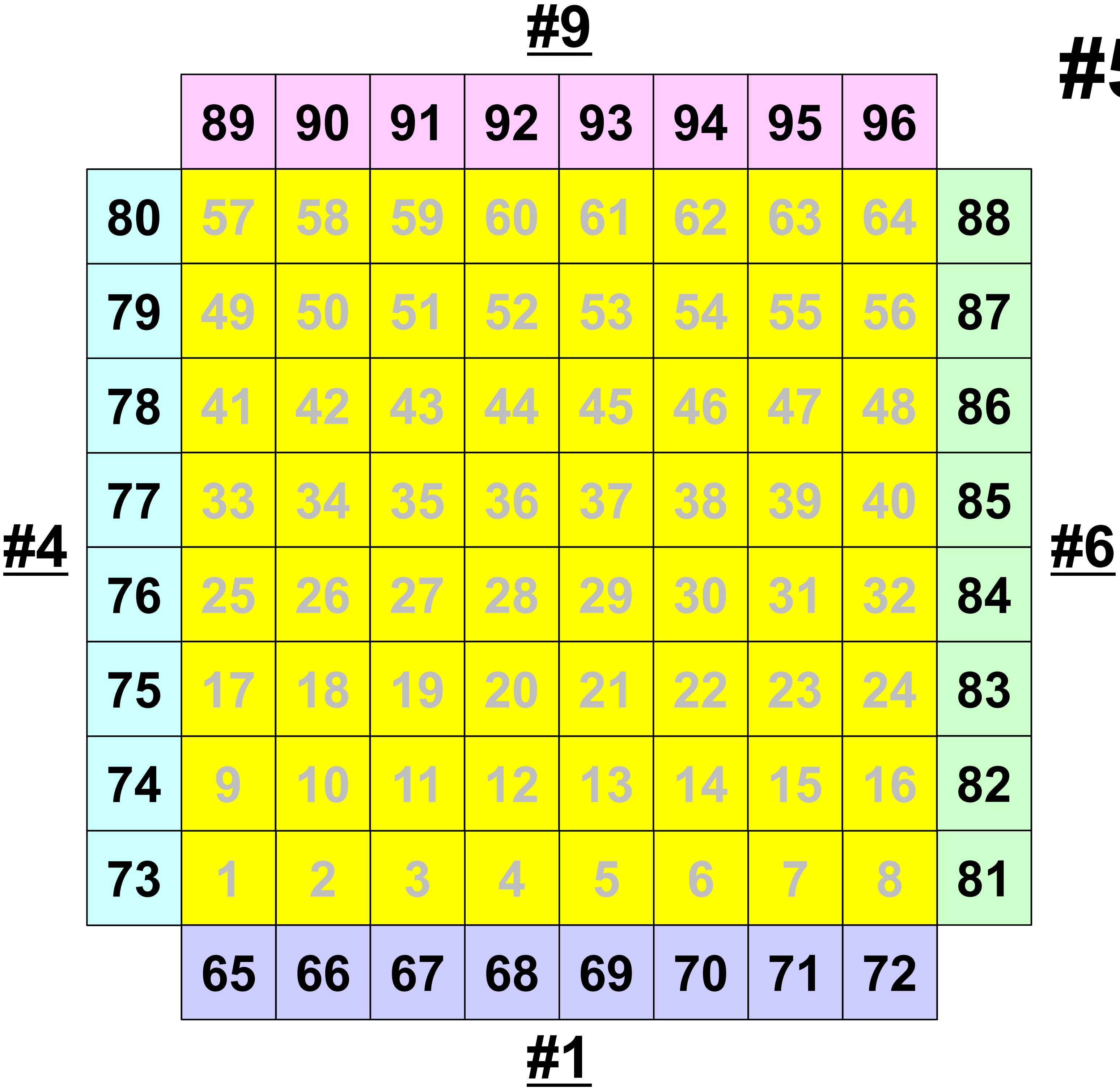
#4

#6

#1

57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

#5 Process External Nodes



#9

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

#4

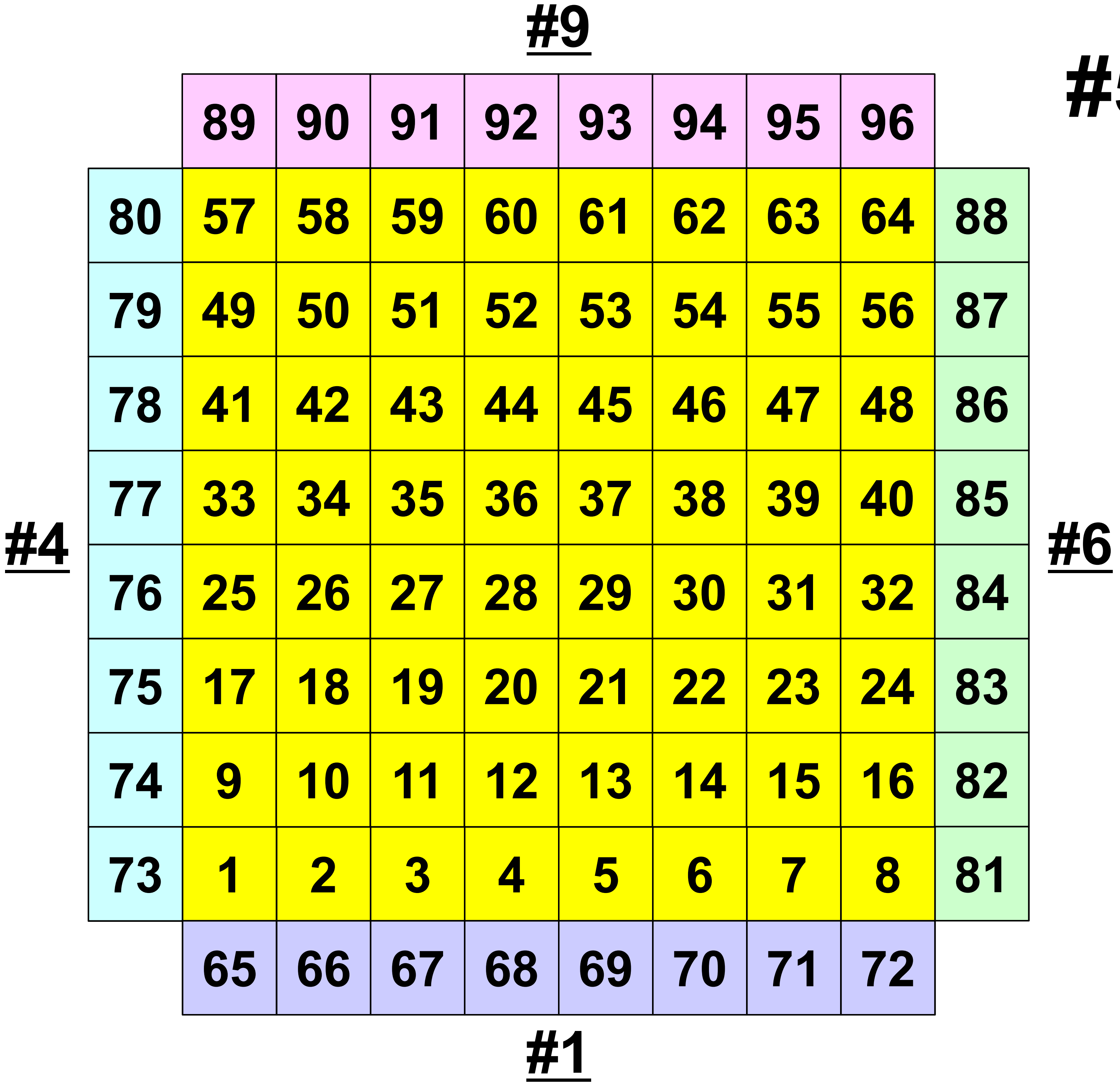
#6

#1

#5 Process All Nodes

64: Internal

32: External



#9

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

#4

#6

#1

Let’s do parallel mesh generation !!

Oakleaf-FX

```
>$ cd
>$ cd hybrid/fvm (<$O-fvm>)
>$ cd pmesh
>$ mpifrtpx –Kfast pmesh.f –o pmesh
(modify mesh.inp, mg.sh)
>$ pjsub mg.sh
```

mg.sh: parallel mesh generation

”proc” must be equal to (npx × npy × npz)

Each MPI process generates each local mesh file

mg.sh

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:05:00"
#PJM -L "rscgrp=lecture7"
#PJM -g "gt17"
#PJM -j
#PJM -o "mg.lst"
#PJM --mpi "proc=16"

mpiexec ./pmesh
rm wk.*
```

mesh.inp

```
32  32   1
 4   4   1
```

```
8 processes
“node=1“
“proc=8”
```

```
64 processes
“node=4“
“proc=64”
```

```
16 processes
“node=1“
“proc=16”
```

```
192 processes
“node=12“
“proc=192”
```

```
32 processes
“node=2“
“proc=32”
```

main program

```
      program MAIN

      use STRUCT
      use PCG
      use solver_PCG

      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 )

      call INPUT
      call POINTER_INIT
      call BOUNDARY_CELL
      call CELL_METRICS
      call POI_GEN

      PHI= 0.d0
        call solve_PCG                                                  &
     &      ( ICELTOT, NP, NPLU, indexLU, itemLU, D, BFORCE,            &
     &        PHI, AMAT, NEIBPETOT, NEIBPE,                             &
     &        IMPORT_INDEX, IMPORT_ITEM, EXPORT_INDEX, EXPORT_ITEM,     &
     &        EPSICCG, ITR, IER, my_rank)

      call ParallelVIS

      call MPI_FINALIZE (ierr)

      end
```

input: reading “INPUT.DAT”

```
!C
!C***
!C*** INPUT
!C***
!C
!C    INPUT CONTROL DATA
!C
      subroutine INPUT
      use STRUCT
      use PCG

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

      character*80 CNTFIL

!C
!C-- CNTL. file
      open  (11, file='INPUT.DAT', status='unknown')
        read (11,'(a80)') HEADER
        read (11,*) METHOD
        read (11,*) DX, DY, DZ
        read (11,*) OMEGA, EPSICCG
        read (11,*) VISceltot
      close (11)
!C===

      return
      end
```

In this case a single file (INPUT.DAT) is read from all MPI processes.

This is not good.
If you have O(10^6) processes, file-system may fail.

If you have a time, please re-write the program using MPI_Bcast, as shown in the next page.

using MPI_Bcast

```
     if (my_rank.eq.0) then
       open  (11, file='INPUT.DAT', status='unknown')
         read (11,'(a80)') HEADER
         read (11,*) METHOD
         read (11,*) DX, DY, DZ
         read (11,*) OMEGA, EPSICCG
         read (11,*) VISceltot
        close (11)
      endif

      call MPI_BCAST (HEADER,    80, MPI_CHARACTER, 0, MPI_COMM_WORLD,ierr)
      call MPI_BCAST (METHOD,     1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (VISceltot,  1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (DX, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (DY, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (DZ, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (OMGA, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)
      call MPI_BCAST (EPSOCCG, 1, MPI_DOUBLE_PRECISION, 0, MPI_COMM_WORLD, ierr)

      return
      end
```

"INPUT.DAT" is read at rank#0 process, and information is sent to all processes by MPI_Bcast.

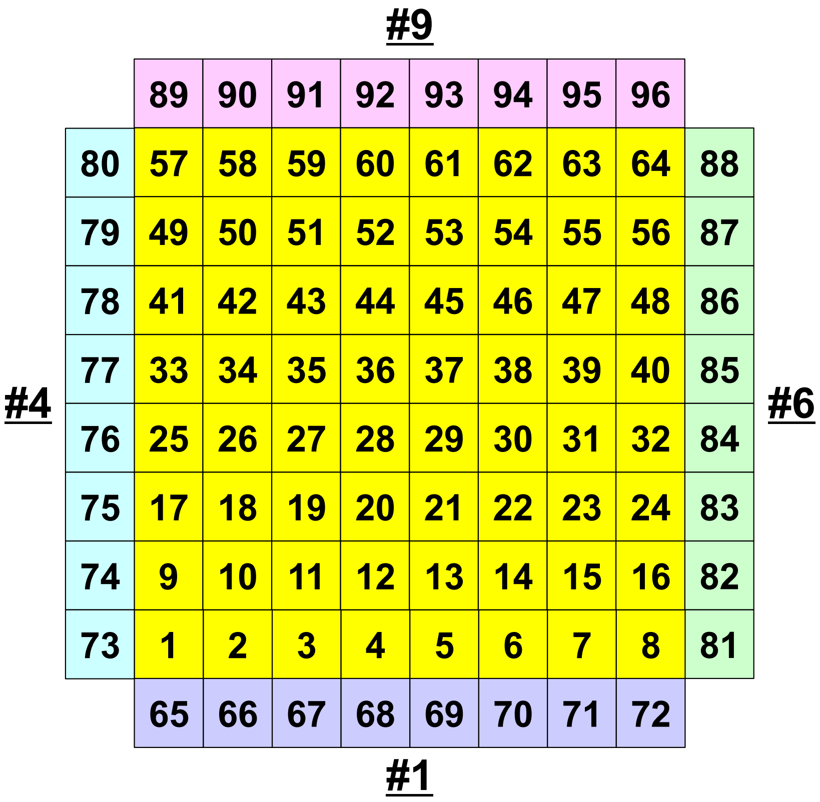
Control Data: INPUT.DAT

```
../pmesh/in                                HEADER
3                                          MEHOD 1:2:3
1.00e-00 1.00e-00 1.00e-00                 DX/DY/DZ
0.10 1.0e-08                               OMEGA, EPSICCG
1000                                       VISceltot
```

- **`HEADER`:** Header of Distributed Local Files: Fixed
  `(../pmesh/in.0, ../pmesh/in.1 ...)`
- **`METHOD`:** Preconditioning Method, fixed as 3 (Point Jacobi)
- **`DX,DY,DZ`:** Mesh Size
- **`OMEGA`:** (not in use)
- **`EPSICCG`:** Convergence Criteria for CG Method
- **`VISceltot`:**
  Approximate number of meshes for visualization

pointer_init (1/12) “in.5”

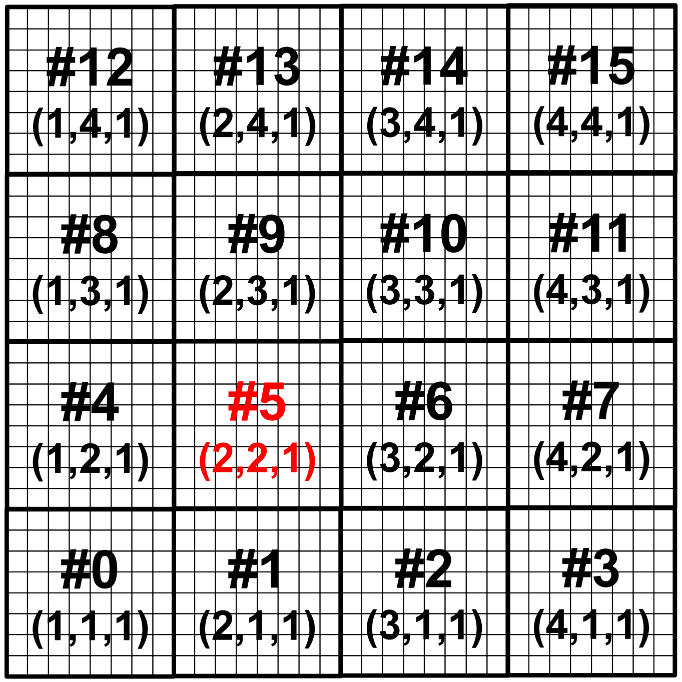
```
read (21,'(10i10)') NPX, NPY, NPZ
read (21,'(10i10)') NX, NY, NZ
read (21,'(10i10)') NPi, NPj, NPk

read (21,'(10i10)') NEIBPETOT
read (21,'(10i10)') (NEIBPE(ip),ip=1,NEIBPETOT)

read (21,'(10i10)') NP, ICELTOT
```



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

```
     4     4     1  npx, npy, npz
     8     8     1  NX, NY, NZ (= NAx/npx etc.)
     2     2     1  NPi, NPi, NPk (Global Loc. of Proc’ s)
     4                   # of Neighbors(NEIBPETOT)
     1     4     6     9 ID of Neighbor (NEIBPE)
    96    64             NP, ICELTOT
     1     0     0     1
     2     0     0     1
     3     0     0     1
     4     0     0     1
(...)
    93     0     0     1
    94     0     0     1
    95     0     0     1
    96     0     0     1
    96
     1    73     2    65     9     0     0
     2     1     3    66    10     0     0
     3     2     4    67    11     0     0
     4     3     5    68    12     0     0
(...)
    27    26    28    19    35     0     0
(...)
    64    63    88    56    96     0     0
(...)
    93    92    94    61     0     0     0
    94    93    95    62     0     0     0
    95    94    96    63     0     0     0
    96    95     0    64     0     0     0
     8    16    24    32
    65    66    67    68    69    70    71    72    73    74
    75    76    77    78    79    80    81    82    83    84
    85    86    87    88    89    90    91    92    93    94
    95    96
     8    16    24    32
     1     2     3     4     5     6     7     8     1     9
    17    25    33    41    49    57     8    16    24    32
    40    48    56    64    57    58    59    60    61    62
    63    64
```

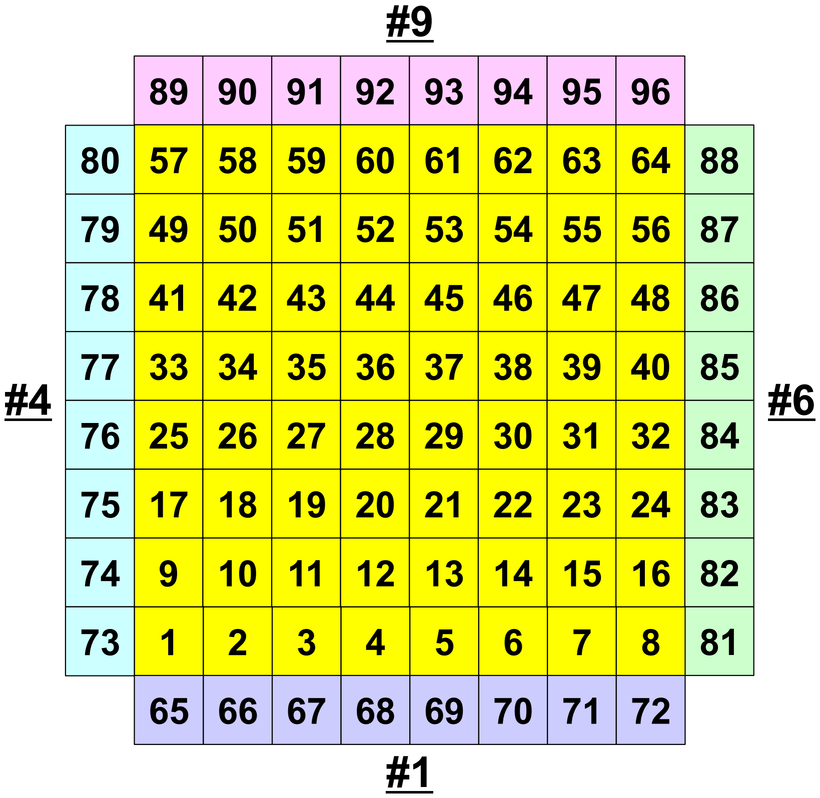
Name	Type	Content
`NAx, NAy, NAz`	`I`	Number of Entire Meshes in X-/Y-/Z-direction
`NPX, NPY, NPZ`	`I`	Number of Divisions in X-/Y-/Z-direction
`NPi, NPj, NPk`	`I`	Global Location of the Processes
`NX, NY, NZ`	`I`	Number of Local Meshes in X-/Y-/Z-direction `NX=NAx/NPX, NY=NAy/NPY`, `NZ=NAz/NPZ`
`NEIBPETOT`	`I`	Number of Neighbors
`NEIBPE(NEIBPETOT)`	`I`	ID of Neighbor
`ICELTOT`	`I`	Number of Internal Meshes (=`N`)
`NP`	`I`	Number of (Internal + External) Meshes



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

pointer_init (2/12) "in.5"

```
do i= 1, NP
  read (21,'(4i10)') ii,(BOUNDARY(i,k), k=1,3)
enddo

read (21,'(10i10)') ii
do i= 1, NP
  read (21,'(7i10)') ii,(NEIBcell(i,k), k=1,6)
enddo
```



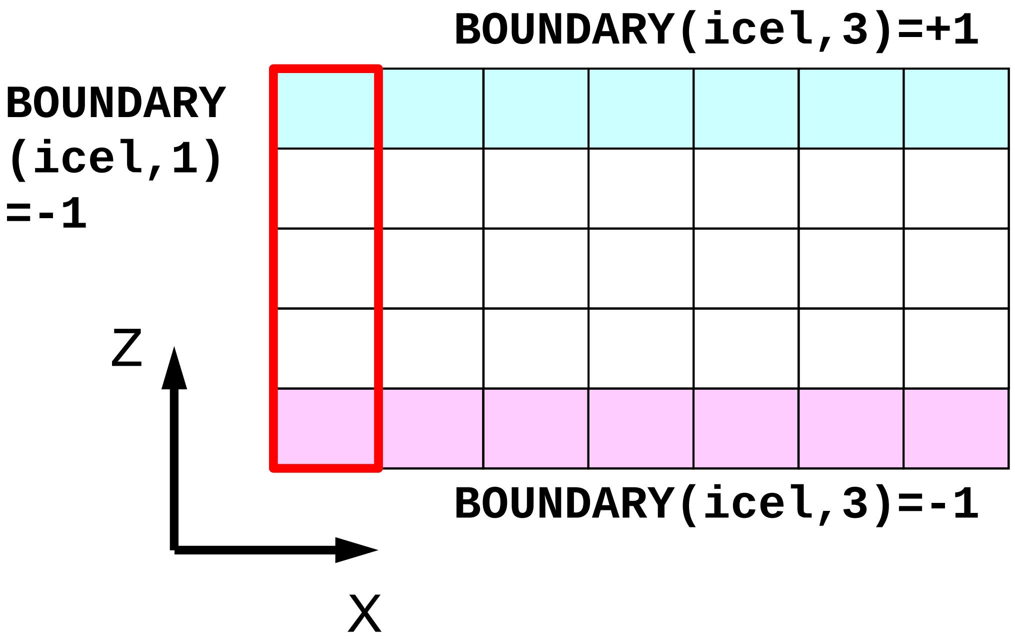
	89	90	91	92	93	94	95	96	
80	57	58	59	60	61	62	63	64	88
79	49	50	51	52	53	54	55	56	87
78	41	42	43	44	45	46	47	48	86
77	33	34	35	36	37	38	39	40	85
76	25	26	27	28	29	30	31	32	84
75	17	18	19	20	21	22	23	24	83
74	9	10	11	12	13	14	15	16	82
73	1	2	3	4	5	6	7	8	81
	65	66	67	68	69	70	71	72	

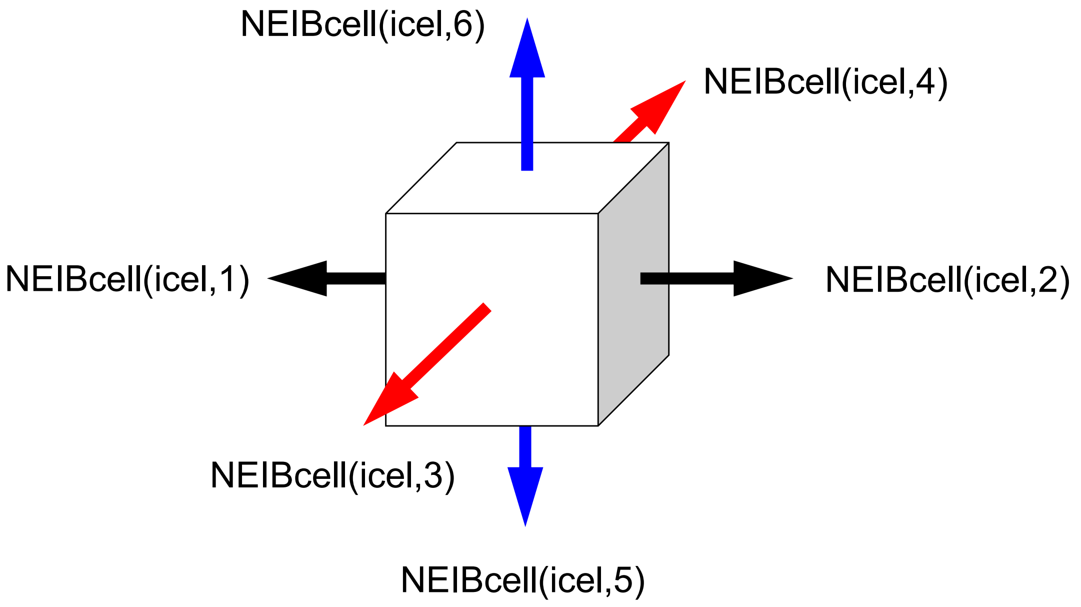
```
   4       4       1
   8       8       1
   2       2       1
   4
   1       4       6       9
  96      64
   1       0       0       1             ii, BOUNDARY(icel,k)
   2       0       0       1                 k=1-3, Zmax
   3       0       0       1
   4       0       0       1
(...)
  93       0       0       1
  94       0       0       1
  95       0       0       1
  96       0       0       1
  96
   1      73       2      65       9       0       0 ii, NEIBcell(i,k)
   2       1       3      66      10       0       0     k=1-6
   3       2       4      67      11       0       0
   4       3       5      68      12       0       0
(...)
  27      26      28      19      35       0       0
(...)
  64      63      88      56      96       0       0
(...)
  93      92      94      61       0       0       0
  94      93      95      62       0       0       0
  95      94      96      63       0       0       0
  96      95       0      64       0       0       0
   8      16      24      32
  65      66      67      68      69      70      71      72      73      74
  75      76      77      78      79      80      81      82      83      84
  85      86      87      88      89      90      91      92      93      94
  95      96
   8      16      24      32
   1       2       3       4       5       6       7       8       1       9
  17      25      33      41      49      57       8      16      24      32
  40      48      56      64      57      58      59      60      61      62
  63      64
```

Name	Type	Content

`BOUNDARY(NP,3)`	**`I`**	Boundary Information
		`BOUNDARY(icel,1)= -1: Xmin, +1: Xmax`
		`BOUNDARY(icel,2)= -1: Ymin, +1: Ymax`
		`BOUNDARY(icel,3)= -1: Zmin, +1: Zmax`

`NEIBcell(NP,6)`	**`I`**	ID's of Neighboring Meshes





NEIBcell(icel,1)= icel – 1
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,6)= icel + NX*NY

pointer_init (3/12) “in.5”

```
icou= 0
do k= 1, NZ
do j= 1, NY
do i= 1, NX
  icou= icou + 1
  XYZ(icou,1)= i + (NPi-1)*NX
  XYZ(icou,2)= j + (NPj-1)*NY
  XYZ(icou,3)= k + (NPk-1)*NZ
enddo
enddo
enddo
```

Global ID

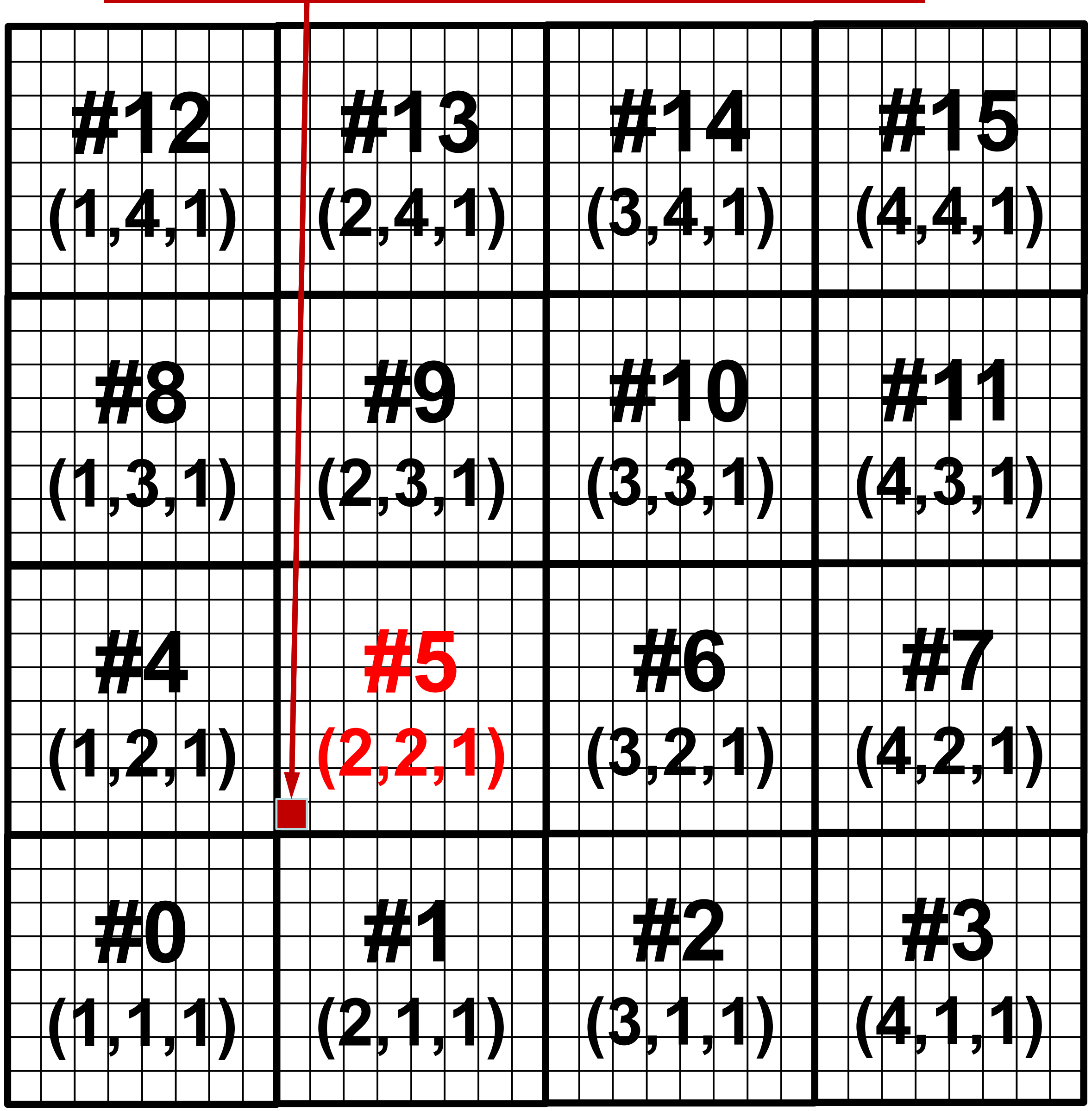
```
XYZ(icel,1)= 1 + (NPi-1)*NX
XYZ(icel,2)= 1 + (NPj-1)*NY
XYZ(icel,3)= 1 + (NPk-1)*NZ
icel= 1
```

Local Location

```
i= XYZ(icel,1)
j= XYZ(icel,2)
k= XYZ(icel,3)
icel= (k-1)*NX*NY+(j-1)*NX+i
```

Global Location

```
XYZ(icel,1)= i + (NPi-1)*NX
XYZ(icel,2)= j + (NPj-1)*NY
XYZ(icel,3)= k + (NPk-1)*NZ
```



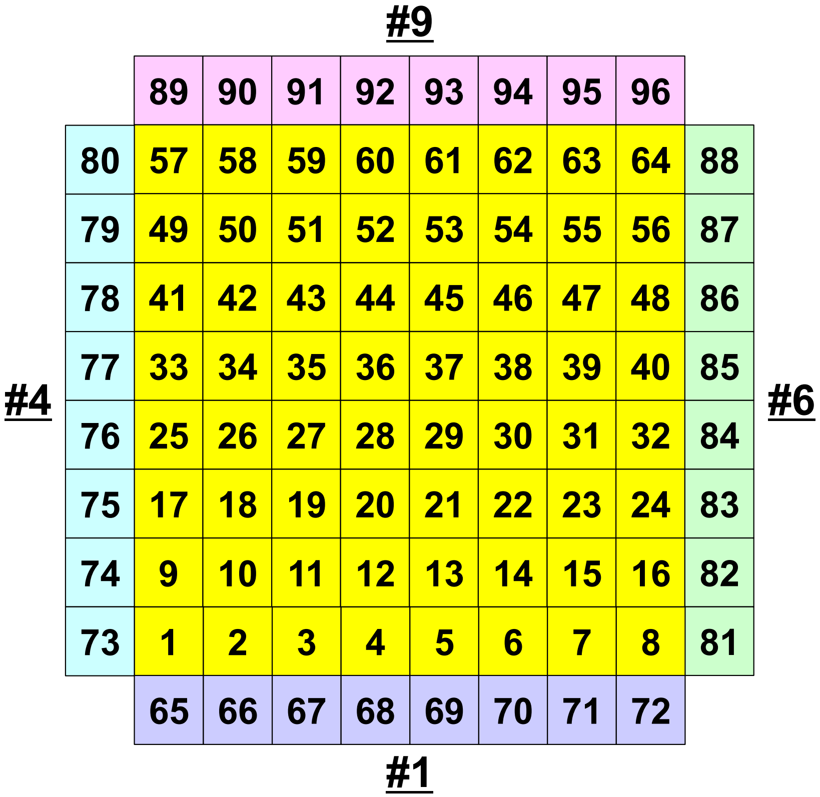
#12 (1,4,1)	#13 (2,4,1)	#14 (3,4,1)	#15 (4,4,1)
#8 (1,3,1)	#9 (2,3,1)	#10 (3,3,1)	#11 (4,3,1)
#4 (1,2,1)	#5 (2,2,1)	#6 (3,2,1)	#7 (4,2,1)
#0 (1,1,1)	#1 (2,1,1)	#2 (3,1,1)	#3 (4,1,1)

pointer_init (4/12) “in.5”

```
   4       4       1
   8       8       1
   2       2       1
   4                                 # of Neighbors(NEIBPETOT)
   1       4       6       9         ID of Neighbor (NEIBPE)
  96      64                         NP, ICELTOT
   1       0       0       1
                                   9       0       0
                                  10       0       0
   3       2       4      67      11       0       0
   4       3       5      68      12       0       0
 (...)
  27      26      28      19      35       0       0
 (...)
  64      63      88      56      96       0       0
 (...)
  93      92      94      61       0       0       0
  94      93      95      62       0       0       0
  95      94      96      63       0       0       0
  96      95       0      64       0       0       0
   8      16      24      32
  65      66      67      68      69      70      71      72      73      74
  75      76      77      78      79      80      81      82      83      84
  85      86      87      88      89      90      91      92      93      94
  95      96
   8      16      24      32
   1       2       3       4       5       6       7       8       1       9
  17      25      33      41      49      57       8      16      24      32
  40      48      56      64      57      58      59      60      61      62
  63      64
```

External Nodes

```
allocate (IMPORT_INDEX(0:NEIBPETOT), EXPORT_INDEX(0:NEIBPETOT))
read (21,'(10i10)') (IMPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= IMPORT_INDEX(NEIBPETOT)
allocate (IMPORT_ITEM(nn))
read (21,'(10i10)') (IMPORT_ITEM(k), k= 1, nn)

read (21,'(10i10)') (EXPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= EXPORT_INDEX(NEIBPETOT)
allocate (EXPORT_ITEM(nn))
read (21,'(10i10)') (EXPORT_ITEM(k), k= 1, nn)
```



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

Generalized Communication Table

Name	Type	Definition
`PETOT`	I	Number of PE's
`my_rank`	I	Process ID of MPI
`NEIBPETOT`	I	Number of Neighbors
`NEIBPE(NEIBPETOT)`	I	ID of Neighbor

`IMPORT_INDEX(0:NEIBPETOT)`
`EXPORT_INEDX(0:NEIBPETOT)`
	I	Size of Import/Export Arrays for Communication Table

`IMPORT_ITEM(nni)`	I	Receiving Table (External Points) **`nni=IMPORT_INDEX(NEIBPETOT)`**
`EXPORT_ITEM(nne)`	I	Sending Table (Boundary Points) **`nne=EXPORT_INDEX(NEIBPETOT)`**

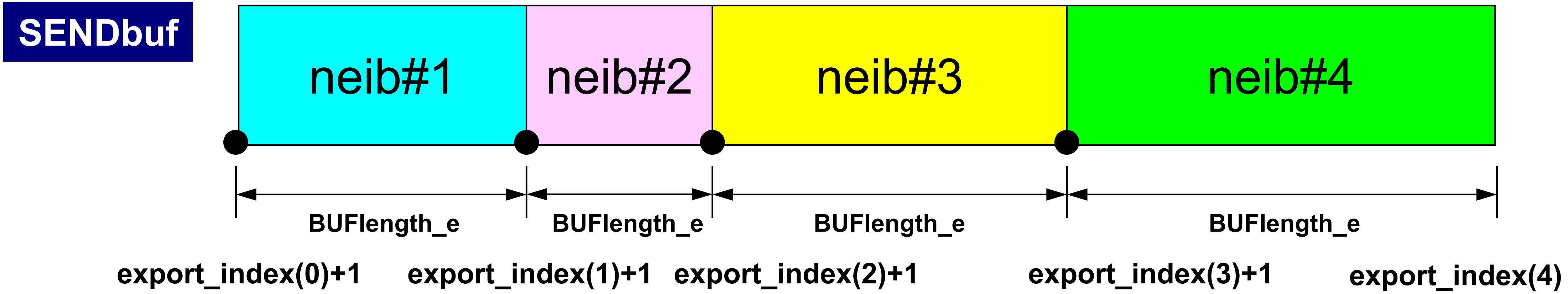
Generalized Communication Table: Send

- Neighbors
  - – NEIBPETOT, NEIBPE(neib)
- Message size for each neighbor
  - – export_index(neib), neib= 0, NEIBPETOT
- ID of **boundary** nodes
  - – export_item(k), k= 1, export_index(NEIBPETOT)
- Messages to each neighbor
  - – SENDbuf(k), k= 1, export_index(NEIBPETOT)

Fortran

SEND: MPI_Isend/Irecv/Waitall

Fortran



```
 do neib= 1, NEIBPETOT
   do k= export_index(neib-1)+1, export_index(neib)
     kk= export_item(k)
     SENDbuf(k)= VAL(kk)
   enddo
 enddo

 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_e= iE_e + 1 - iS_e

   call MPI_ISEND                                                   &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, request_send(neib), ierr)
 enddo

 call MPI_WAITALL (NEIBPETOT, request_send, stat_recv, ierr)
```

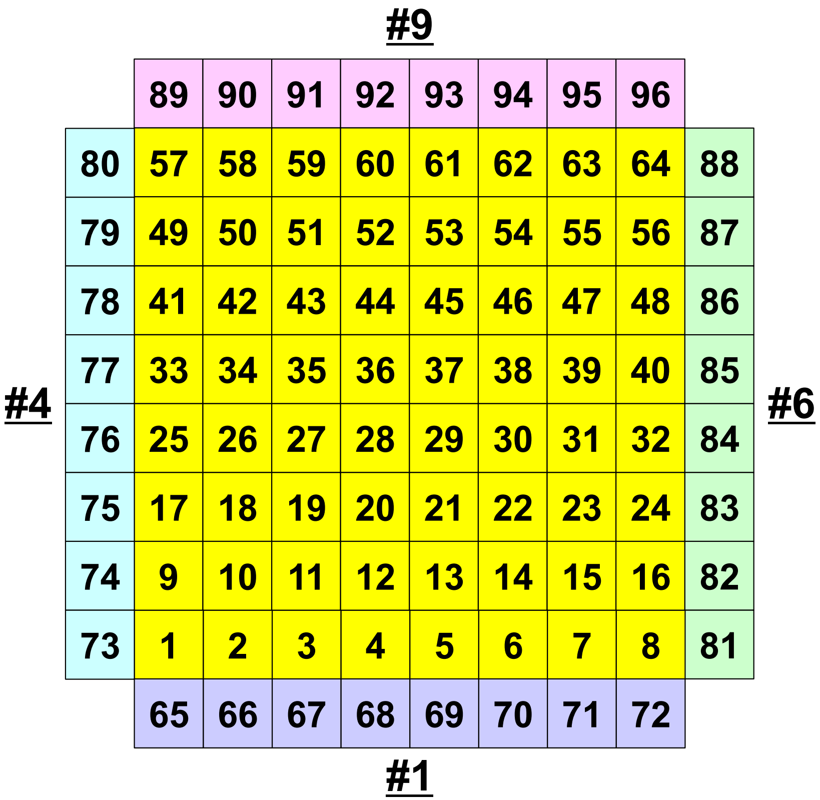
Copied to sending buffers

pointer_init (5/12) “in.5”

```
allocate (IMPORT_INDEX(0:NEIBPETOT), EXPORT_INDEX(0:NEIBPETOT))
read (21,'(10i10)') (IMPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= IMPORT_INDEX(NEIBPETOT)
allocate (IMPORT_ITEM(nn))
read (21,'(10i10)') (IMPORT_ITEM(k), k= 1, nn)

read (21,'(10i10)') (EXPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= EXPORT_INDEX(NEIBPETOT)
allocate (EXPORT_ITEM(nn))
read (21,'(10i10)') (EXPORT_ITEM(k), k= 1, nn)
```

External Nodes

```
 4     4     1
 8     8     1
 2     2     1
 4                          # of Neighbors(NEIBPETOT)
 1     4     6     9        ID of Neighbor (NEIBPE)
96    64                    NP, ICELTOT
 1     0     0     1
                             9     0     0
                            10     0     0
 3     2     4    67        11     0     0
 4     3     5    68        12     0     0
(...)
27    26    28    19        35     0     0
(...)
64    63    88    56        96     0     0
(...)
93    92    94    61         0     0     0
94    93    95    62         0     0     0
95    94    96    63         0     0     0
96    95     0    64         0     0     0
 8    16    24    32
65    66    67    68        69    70    71    72    73    74
75    76    77    78        79    80    81    82    83    84
85    86    87    88        89    90    91    92    93    94
95    96
 8    16    24    32
 1     2     3     4         5     6     7     8     1     9
17    25    33    41        49    57     8    16    24    32
40    48    56    64        57    58    59    60    61    62
63    64
```



#9

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

#4 #6

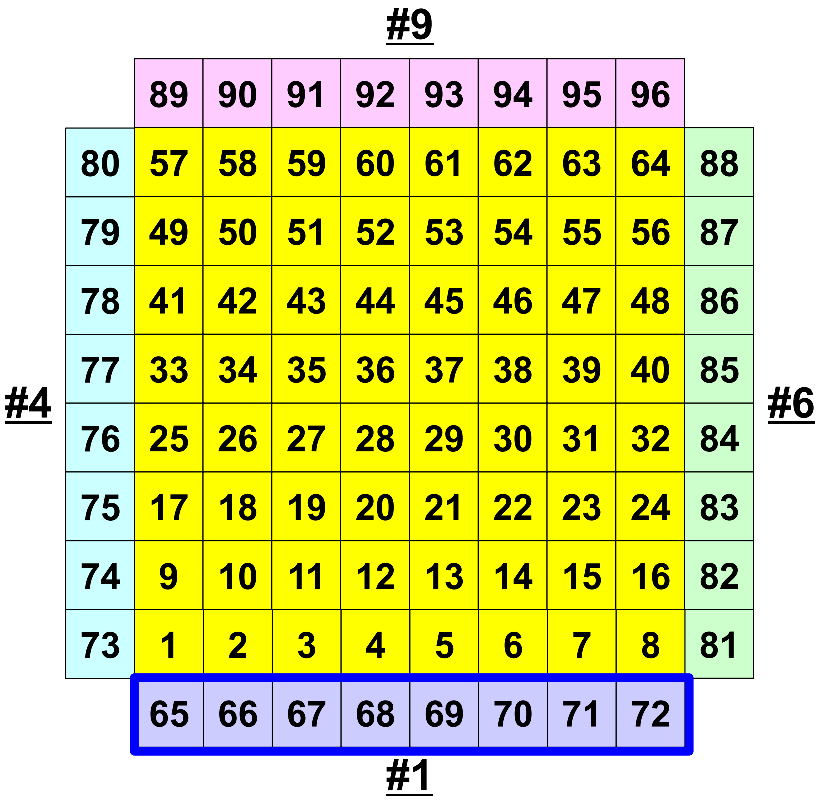
#1

pointer_init (6/12) “in.5”

```
allocate (IMPORT_INDEX(0:NEIBPETOT), EXPORT_INDEX(0:NEIBPETOT))
read (21,'(10i10)') (IMPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= IMPORT_INDEX(NEIBPETOT)
allocate (IMPORT_ITEM(nn))
read (21,'(10i10)') (IMPORT_ITEM(k), k= 1, nn)

read (21,'(10i10)') (EXPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= EXPORT_INDEX(NEIBPETOT)
allocate (EXPORT_ITEM(nn))
read (21,'(10i10)') (EXPORT_ITEM(k), k= 1, nn)
```

```
   4      4      1
   8      8      1
   2      2      1
   4                               # of Neighbors(NEIBPETOT)
   1      4      6      9          ID of Neighbor (NEIBPE)
  96     64                        NP, ICELTOT
   1      0      0      1
                               9      0      0
                              10      0      0
   3      2      4     67     11      0      0
   4      3      5     68     12      0      0
 (...)
  27     26     28     19     35      0      0
 (...)
  64     63     88     56     96      0      0
 (...)
  93     92     94     61      0      0      0
  94     93     95     62      0      0      0
  95     94     96     63      0      0      0
  96     95      0     64      0      0      0
   8     16     24     32
  65     66     67     68     69     70     71     72     73     74
  75     76     77     78     79     80     81     82     83     84
  85     86     87     88     89     90     91     92     93     94
  95     96
   8     16     24     32
   1      2      3      4      5      6      7      8      1      9
  17     25     33     41     49     57      8     16     24     32
  40     48     56     64     57     58     59     60     61     62
  63     64
```

External Nodes



#9

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

#4 #6 #1

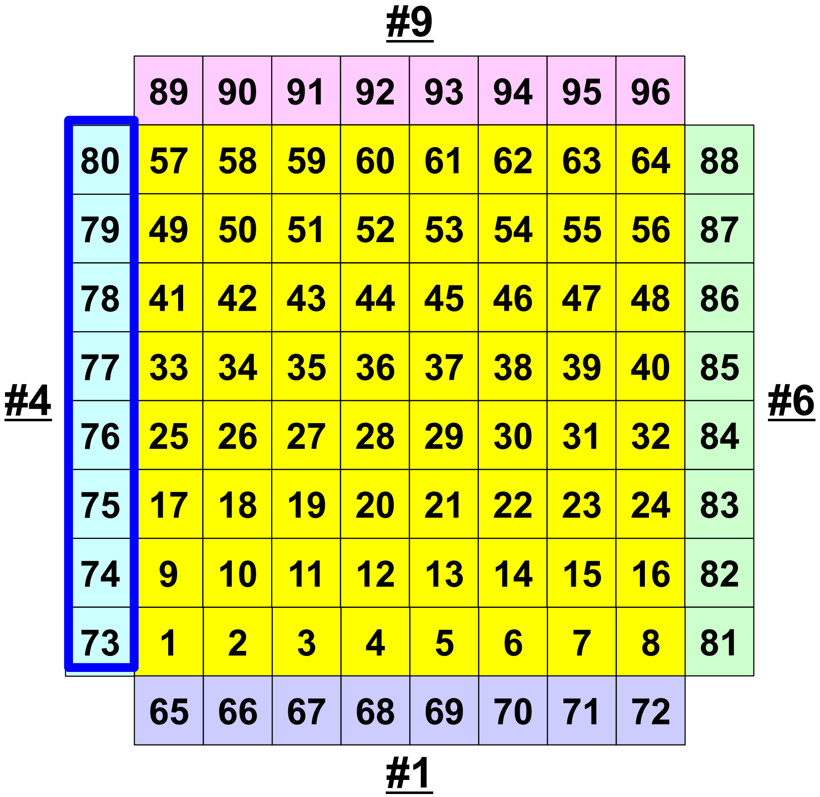
pointer_init (7/12) “in.5”

```
4         4         1
8         8         1
2         2         1
4                                   # of Neighbors(NEIBPETOT)
1         4         6         9     ID of Neighbor (NEIBPE)
96        64                        NP, ICELTOT
1         0         0         1
```

```
allocate (IMPORT_INDEX(0:NEIBPETOT), EXPORT_INDEX(0:NEIBPETOT))
read (21,'(10i10)') (IMPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= IMPORT_INDEX(NEIBPETOT)
allocate (IMPORT_ITEM(nn))
read (21,'(10i10)') (IMPORT_ITEM(k), k= 1, nn)

read (21,'(10i10)') (EXPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= EXPORT_INDEX(NEIBPETOT)
allocate (EXPORT_ITEM(nn))
read (21,'(10i10)') (EXPORT_ITEM(k), k= 1, nn)
```

External Nodes

```
                                        9         0         0
                                        10        0         0
3         2         4         67        11        0         0
4         3         5         68        12        0         0
(...)
27        26        28        19        35        0         0
(...)
64        63        88        56        96        0         0
(...)
93        92        94        61        0         0         0
94        93        95        62        0         0         0
95        94        96        63        0         0         0
96        95        0         64        0         0         0
8         16        24        32
65        66        67        68        69        70        71        72        73        74
75        76        77        78        79        80        81        82        83        84
85        86        87        88        89        90        91        92        93        94
95        96
8         16        24        32
1         2         3         4         5         6         7         8         1         9
17        25        33        41        49        57        8         16        24        32
40        48        56        64        57        58        59        60        61        62
63        64
```



#9

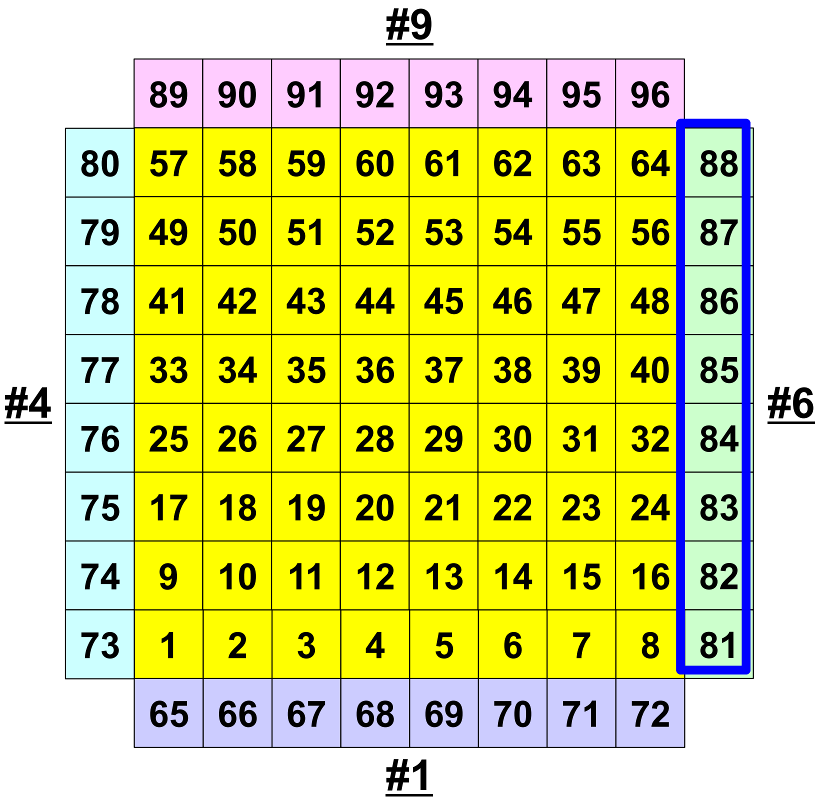
	89	90	91	92	93	94	95	96	
80	57	58	59	60	61	62	63	64	88
79	49	50	51	52	53	54	55	56	87
78	41	42	43	44	45	46	47	48	86
77	33	34	35	36	37	38	39	40	85
76	25	26	27	28	29	30	31	32	84
75	17	18	19	20	21	22	23	24	83
74	9	10	11	12	13	14	15	16	82
73	1	2	3	4	5	6	7	8	81
	65	66	67	68	69	70	71	72	

pointer_init (8/12) “in.5”

```
allocate (IMPORT_INDEX(0:NEIBPETOT), EXPORT_INDEX(0:NEIBPETOT))
read (21,'(10i10)') (IMPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= IMPORT_INDEX(NEIBPETOT)
allocate (IMPORT_ITEM(nn))
read (21,'(10i10)') (IMPORT_ITEM(k), k= 1, nn)

read (21,'(10i10)') (EXPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= EXPORT_INDEX(NEIBPETOT)
allocate (EXPORT_ITEM(nn))
read (21,'(10i10)') (EXPORT_ITEM(k), k= 1, nn)
```

External Nodes

```
 4       4       1
 8       8       1
 2       2       1
 4                                               # of Neighbors(NEIBPETOT)
 1       4       6       9                       ID of Neighbor (NEIBPE)
96      64                                       NP, ICELTOT
 1       0       0       1
                                 9       0       0
                                10       0       0
 3       2       4      67      11       0       0
 4       3       5      68      12       0       0
(...)
27      26      28      19      35       0       0
(...)
64      63      88      56      96       0       0
(...)
93      92      94      61       0       0       0
94      93      95      62       0       0       0
95      94      96      63       0       0       0
96      95       0      64       0       0       0
 8      16      24      32
65      66      67      68      69      70      71      72      73      74
75      76      77      78      79      80      81      82      83      84
85      86      87      88      89      90      91      92      93      94
95      96
 8      16      24      32
 1       2       3       4       5       6       7       8       1       9
17      25      33      41      49      57       8      16      24      32
40      48      56      64      57      58      59      60      61      62
63      64
```



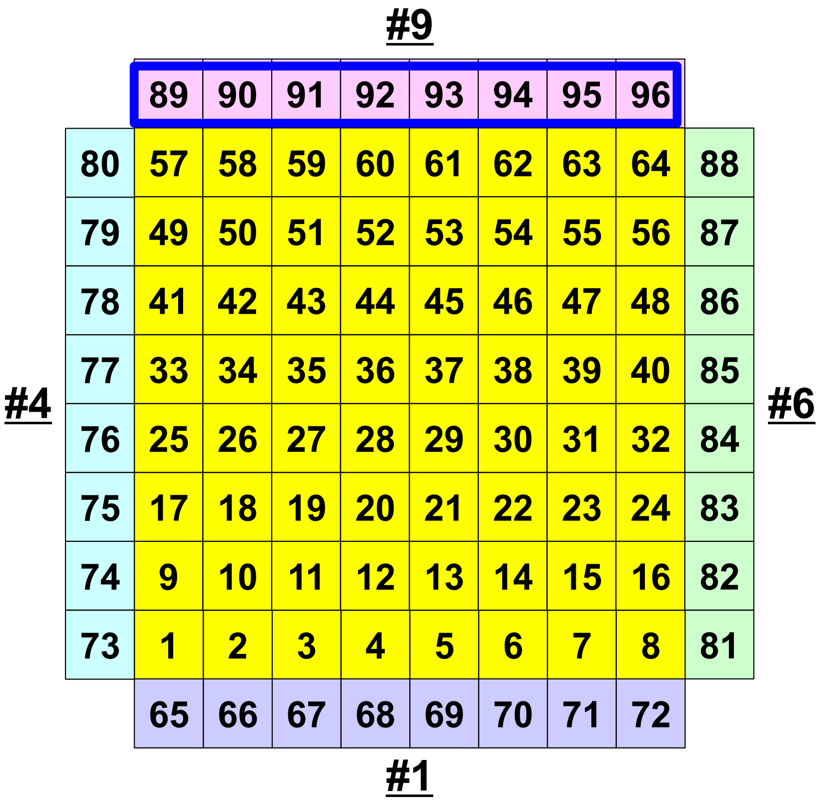
	89	90	91	92	93	94	95	96	
80	57	58	59	60	61	62	63	64	88
79	49	50	51	52	53	54	55	56	87
78	41	42	43	44	45	46	47	48	86
77	33	34	35	36	37	38	39	40	85
76	25	26	27	28	29	30	31	32	84
75	17	18	19	20	21	22	23	24	83
74	9	10	11	12	13	14	15	16	82
73	1	2	3	4	5	6	7	8	81
	65	66	67	68	69	70	71	72	

pointer_init (9/12)
“in.5”

```
allocate (IMPORT_INDEX(0:NEIBPETOT), EXPORT_INDEX(0:NEIBPETOT))
read (21,'(10i10)') (IMPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= IMPORT_INDEX(NEIBPETOT)
allocate (IMPORT_ITEM(nn))
read (21,'(10i10)') (IMPORT_ITEM(k), k= 1, nn)

read (21,'(10i10)') (EXPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= EXPORT_INDEX(NEIBPETOT)
allocate (EXPORT_ITEM(nn))
read (21,'(10i10)') (EXPORT_ITEM(k), k= 1, nn)
```

```
 4     4     1
 8     8     1
 2     2     1
 4                       # of Neighbors(NEIBPETOT)
 1     4     6     9     ID of Neighbor (NEIBPE)
96    64                 NP, ICELTOT
 1     0     0     1
                         9     0     0
                        10     0     0
 3     2     4    67    11     0     0
 4     3     5    68    12     0     0
(...)
27    26    28    19    35     0     0
(...)
64    63    88    56    96     0     0
(...)
93    92    94    61     0     0     0
94    93    95    62     0     0     0
95    94    96    63     0     0     0
96    95     0    64     0     0     0
 8    16    24    32
65    66    67    68    69    70    71    72    73    74
75    76    77    78    79    80    81    82    83    84
85    86    87    88    89    90    91    92    93    94
95    96
 8    16    24    32
 1     2     3     4     5     6     7     8     1     9
17    25    33    41    49    57     8    16    24    32
40    48    56    64    57    58    59    60    61    62
63    64
```

External Nodes



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

pointer_init (10/12)

```
allocate (IMPORT_INDEX(0:NEIBPETOT), EXPORT_INDEX(0:NEIBPETOT))
read (21,'(10i10)') (IMPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= IMPORT_INDEX(NEIBPETOT)
allocate (IMPORT_ITEM(nn))
read (21,'(10i10)') (IMPORT_ITEM(k), k= 1, nn)

read (21,'(10i10)') (EXPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= EXPORT_INDEX(NEIBPETOT)
allocate (EXPORT_ITEM(nn))
read (21,'(10i10)') (EXPORT_ITEM(k), k= 1, nn)
```

```
   4       4       1
   8       8       1
   2       2       1
   4                                # of Neighbors(NEIBPETOT)
   1       4       6       9        ID of Neighbor (NEIBPE)
  96      64                        NP, ICELTOT
   1       0       0       1
```

```
                                   9       0       0
                                  10       0       0
                                  11       0       0
```

External Nodes

Numbering of External Nodes is Continuous

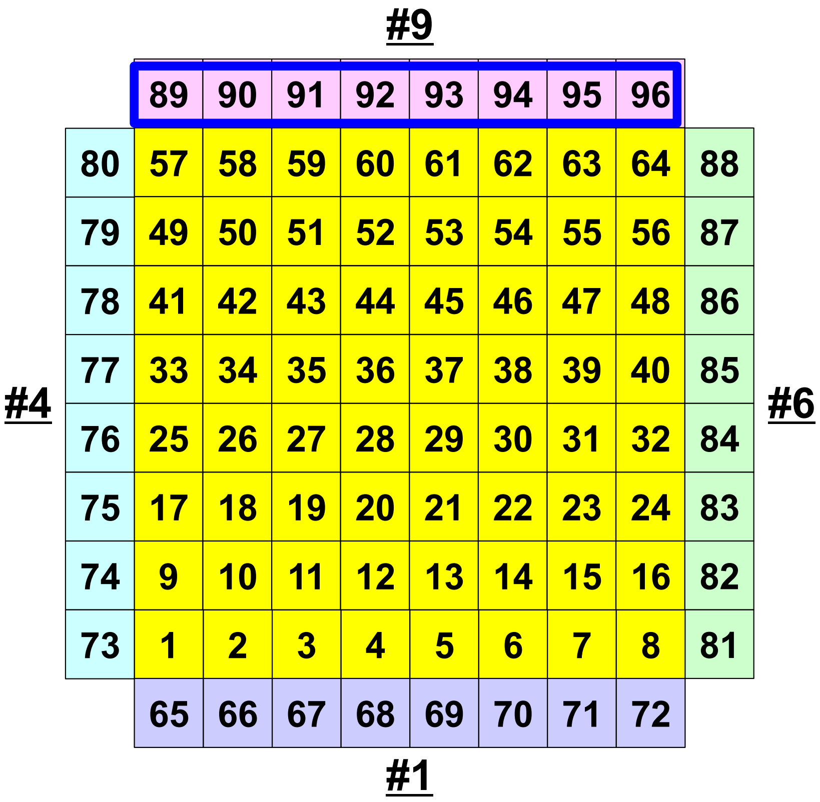
#1: 65-72

#4: 73-80

#6: 81-88

#9: 89-96

```
  96      95       0      64       0       0       0
   8      16      24      32
  65      66      67      68      69      70      71      72      73      74
  75      76      77      78      79      80      81      82      83      84
  85      86      87      88      89      90      91      92      93      94
  95      96
   8      16      24      32
   1       2       3       4       5       6       7       8       1       9
  17      25      33      41      49      57       8      16      24      32
  40      48      56      64      57      58      59      60      61      62
  63      64
```



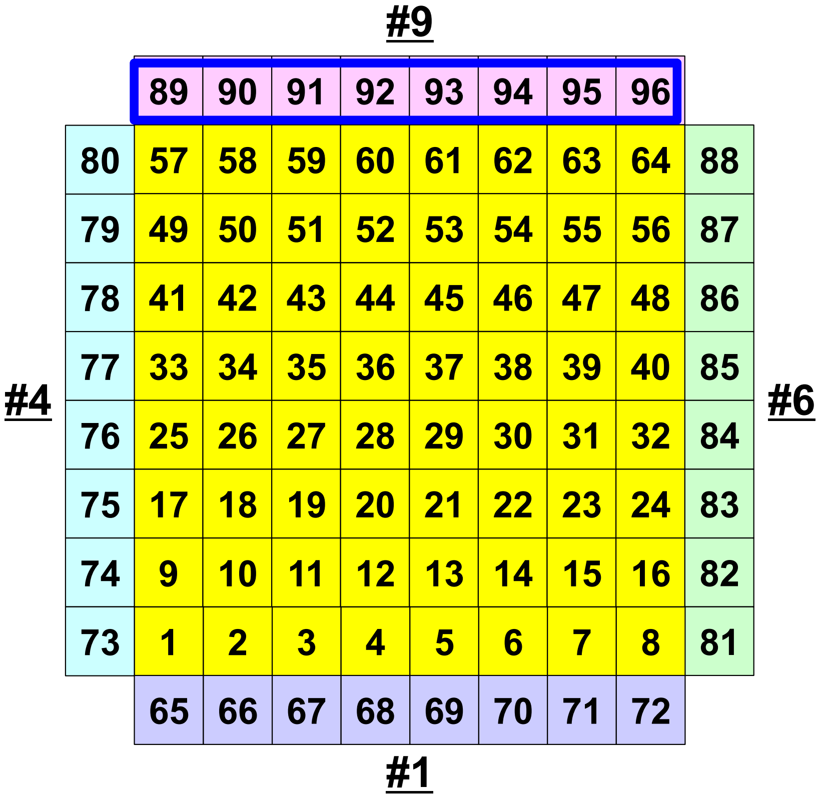
	89	90	91	92	93	94	95	96	
80	57	58	59	60	61	62	63	64	88
79	49	50	51	52	53	54	55	56	87
78	41	42	43	44	45	46	47	48	86
77	33	34	35	36	37	38	39	40	85
76	25	26	27	28	29	30	31	32	84
75	17	18	19	20	21	22	23	24	83
74	9	10	11	12	13	14	15	16	82
73	1	2	3	4	5	6	7	8	81
	65	66	67	68	69	70	71	72	

pointer_init (11/12)

```
allocate (IMPORT_INDEX(0:NEIBPETOT), EXPORT_INDEX(0:NEIBPETOT))
read (21,'(10i10)') (IMPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= IMPORT_INDEX(NEIBPETOT)
allocate (IMPORT_ITEM(nn))
read (21,'(10i10)') (IMPORT_ITEM(k), k= 1, nn)

read (21,'(10i10)') (EXPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= EXPORT_INDEX(NEIBPETOT)
allocate (EXPORT_ITEM(nn))
read (21,'(10i10)') (EXPORT_ITEM(k), k= 1, nn)
```

Boundary Nodes

```
 4    4    1
 8    8    1
 2    2    1
 4                             # of Neighbors(NEIBPETOT)
 1    4    6    9              ID of Neighbor (NEIBPE)
96   64                        NP, ICELTOT
 1    0    0    1

                    9    0    0
                   10    0    0
 3    2    4   67  11    0    0
 4    3    5   68  12    0    0
(...)
27   26   28   19  35    0    0
(...)
64   63   88   56  96    0    0
(...)
93   92   94   61   0    0    0
94   93   95   62   0    0    0
95   94   96   63   0    0    0
96   95    0   64   0    0    0
 8   16   24   32
65   66   67   68  69   70   71   72   73   74
75   76   77   78  79   80   81   82   83   84
85   86   87   88  89   90   91   92   93   94
95   96
 8   16   24   32
 1    2    3    4   5    6    7    8    1    9
17   25   33   41  49   57    8   16   24   32
40   48   56   64  57   58   59   60   61   62
63   64
```



#9

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

#4

#6

#1

Generalized Communication Table

Name	Type	Definition
PETOT	I	Number of PE’s
my_rank	I	Process ID of MPI
NEIBPETOT	I	Number of Neighbors
NEIBPE(NEIBPETOT)	I	ID of Neighbor

IMPORT_INDEX(0:NEIBPETOT)
EXPORT_INEDX(0:NEIBPETOT)
	I	Size of Import/Export Arrays for Communication Table

IMPORT_ITEM(nni)	I	Receiving Table (External Points) **nni=IMPORT_INDEX(NEIBPETOT))**
EXPORT_ITEM(nne)	I	Sending Table (Boundary Points) **nne=EXPORT_INDEX(NEIBPETOT))**

Generalized Communication Table: Receive

- Neighbors
  - – NEIBPETOT, NEIBPE(neib)
- Message size for each neighbor
  - – import_index(neib), neib= 0, NEIBPETOT
- ID of **external** nodes
  - – import_item(k), k= 1, import_index(NEIBPETOT)
- Messages from each neighbor
  - – RECVbuf(k), k= 1, import_index(NEIBPETOT)

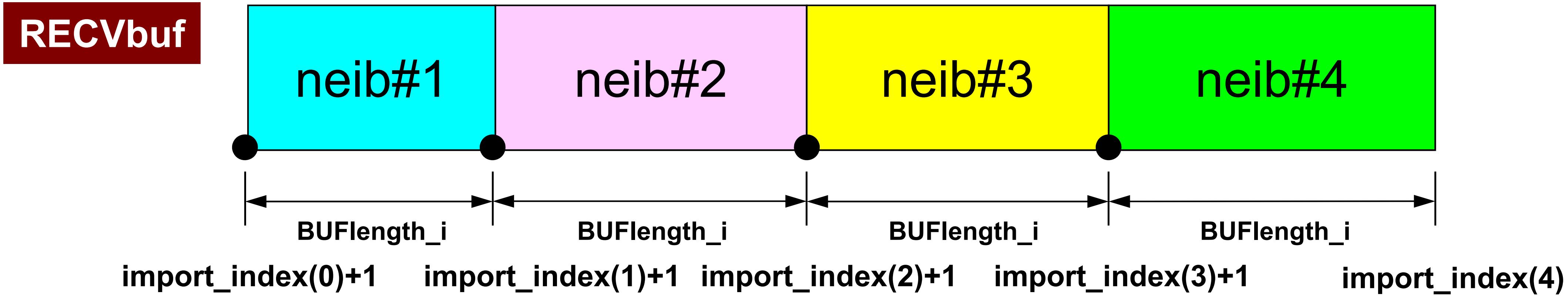
Fortran

RECV: MPI_Isend/Irecv/Waitall

Fortran

```
  do neib= 1, NEIBPETOT
    iS_i= import_index(neib-1) + 1
    iE_i= import_index(neib  )
    BUFlength_i= iE_i + 1 - iS_i

    call MPI_IRECV                                                    &
&          (RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&           MPI_COMM_WORLD, request_recv(neib), ierr)
   enddo

   call MPI_WAITALL (NEIBPETOT, request_recv, stat_recv, ierr)

   do neib= 1, NEIBPETOT
      do k= import_index(neib-1)+1, import_index(neib)
        kk= import_item(k)
        VAL(kk)= RECVbuf(k)
      enddo
   enddo
```

Copied from receiving buffer



RECVbuf

neib#1

neib#2

neib#3

neib#4

BUFlength_i

BUFlength_i

BUFlength_i

BUFlength_i

import_index(0)+1

import_index(1)+1

import_index(2)+1

import_index(3)+1

import_index(4)

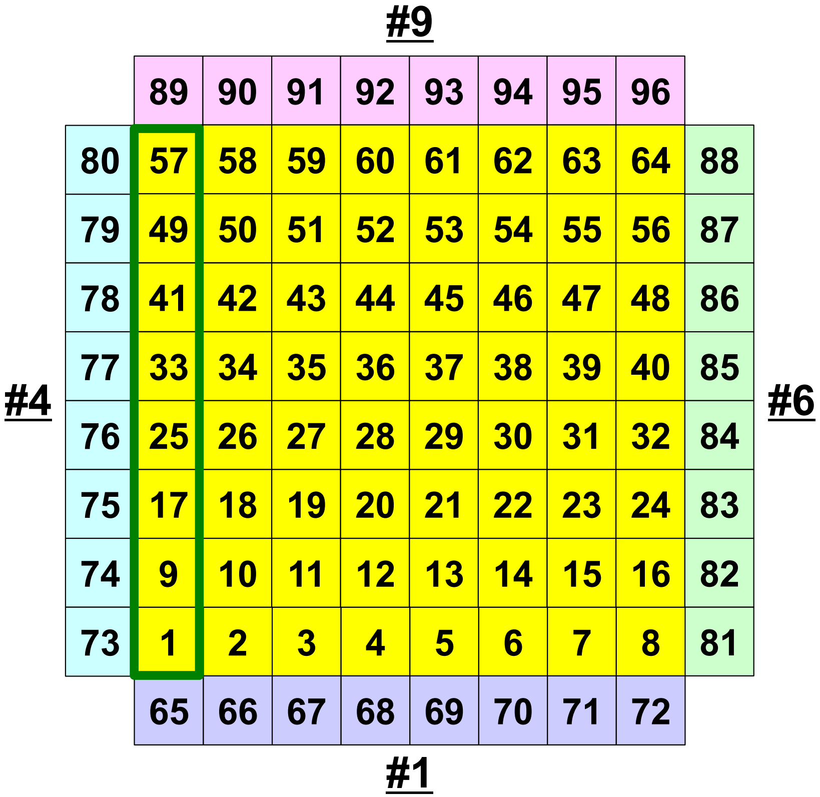
pointer_init (12/12)

```
allocate (IMPORT_INDEX(0:NEIBPETOT), EXPORT_INDEX(0:NEIBPETOT))
read (21,'(10i10)') (IMPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= IMPORT_INDEX(NEIBPETOT)
allocate (IMPORT_ITEM(nn))
read (21,'(10i10)') (IMPORT_ITEM(k), k= 1, nn)

read (21,'(10i10)') (EXPORT_INDEX(ip), ip= 1, NEIBPETOT)
nn= EXPORT_INDEX(NEIBPETOT)
allocate (EXPORT_ITEM(nn))
read (21,'(10i10)') (EXPORT_ITEM(k), k= 1, nn)
```

Boundary Nodes

```
   4       4       1
   8       8       1
   2       2       1
   4                                 # of Neighbors(NEIBPETOT)
   1       4       6       9         ID of Neighbor (NEIBPE)
  96      64                         NP, ICELTOT
   1       0       0       1
```

```
                                9       0       0
                               10       0       0
   3       2       4      67   11       0       0
   4       3       5      68   12       0       0
(...)
  27      26      28      19   35       0       0
(...)
  64      63      88      56   96       0       0
(...)
  93      92      94      61    0       0       0
  94      93      95      62    0       0       0
  95      94      96      63    0       0       0
  96      95       0      64    0       0       0
   8      16      24      32
  65      66      67      68   69      70      71      72      73      74
  75      76      77      78   79      80      81      82      83      84
  85      86      87      88   89      90      91      92      93      94
  95      96
   8      16      24      32
   1       2       3       4    5       6       7       8       1       9
  17      25      33      41   49      57       8      16      24      32
  40      48      56      64   57      58      59      60      61      62
  63      64
```



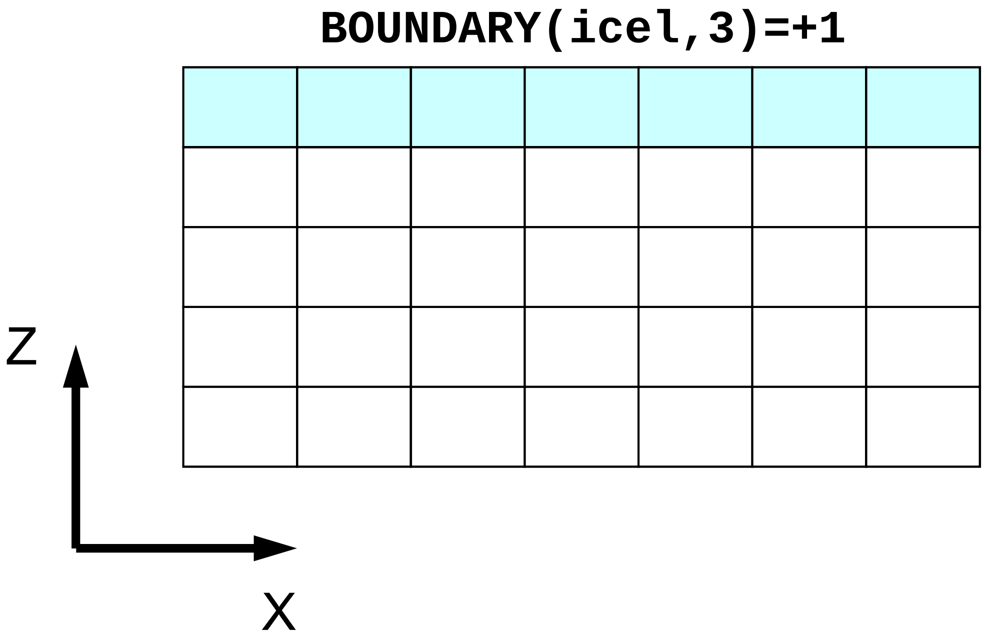
	89	90	91	92	93	94	95	96	
80	57	58	59	60	61	62	63	64	88
79	49	50	51	52	53	54	55	56	87
78	41	42	43	44	45	46	47	48	86
77	33	34	35	36	37	38	39	40	85
76	25	26	27	28	29	30	31	32	84
75	17	18	19	20	21	22	23	24	83
74	9	10	11	12	13	14	15	16	82
73	1	2	3	4	5	6	7	8	81
	65	66	67	68	69	70	71	72	

boundary_cell

```
!C
!C***
!C*** BOUNDARY_CELL
!C***
!C
      subroutine BOUNDARY_CELL
      use STRUCT

      implicit REAL*8 (A-H,O-Z)
!C
!C +-------+
!C | Zmax |
!C +-------+
!C===
      icou= 0
      do i= 1, ICELTOT
        if (BOUNDARY(i,3).eq.1) icou= icou + 1
      enddo

      ZmaxCELtot= icou
      allocate (ZmaxCEL(ZmaxCELtot))

      icou= 0
      do i= 1, ICELTOT
        if (BOUNDARY(i,3).eq.1) then
          icou= icou + 1
          ZmaxCEL(icou)= i
        endif
      enddo
!C===
      return
      end
```

Meshes @ Z=Z_{max}

Number:	**ZmaxCELtot**
Mesh ID:	**ZmaxCEL(:)**



- Parallel Distributed Data Structure
- **Parallel FVM Code**
  - Parallel Visualization
- Parallel Performance
  - Super-Linear in Strong Scaling

main program

```
      program MAIN

      use STRUCT
      use PCG
      use solver_PCG

      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 )

      call INPUT
      call POINTER_INIT
      call BOUNDARY_CELL
      call CELL_METRICS
      call POI_GEN

      PHI= 0.d0
        call solve_PCG                                                 &
     &      ( ICELTOT, NP, NPLU, indexLU, itemLU, D, BFORCE,           &
     &        PHI, AMAT, NEIBPETOT, NEIBPE,                            &
     &        IMPORT_INDEX, IMPORT_ITEM, EXPORT_INDEX, EXPORT_ITEM,    &
     &        EPSICCG, ITR, IER, my_rank)

      call ParallelVIS

      call MPI_FINALIZE (ierr)

      end
```

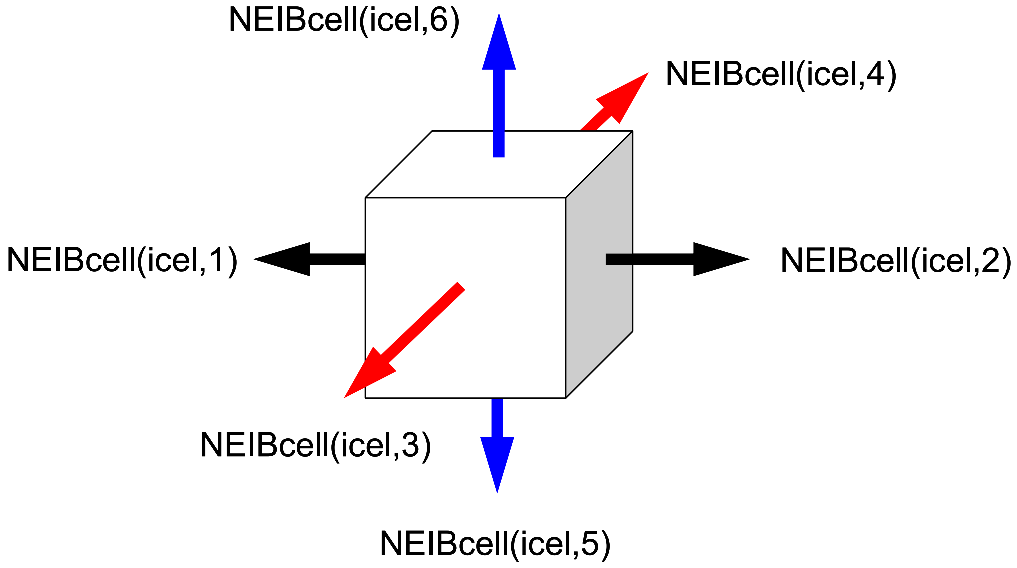
Variables/Arrays for Matrix

Name	Type	Content
ICELTOT	I	Number of Internal Meshes (=N)
NP	I	Number of (Internal + External) Meshes
D(NP)	R	Diagonal components of the matrix (N= ICELTOT)
BFORCE(NP)	R	RHS vector
PHI(NP)	R	Unknown vector
indexLU(0:NP)	I	# of L/U non-zero off-diag. comp. (CRS)
NPLU	I	Total # of L/U non-zero off-diag. comp. (CRS)
itemLU(NPLU)	I	Column ID of L/U non-zero off-diag. comp. (CRS)
ALU(NPLU)	R	L/U non-zero off-diag. comp. (CRS)

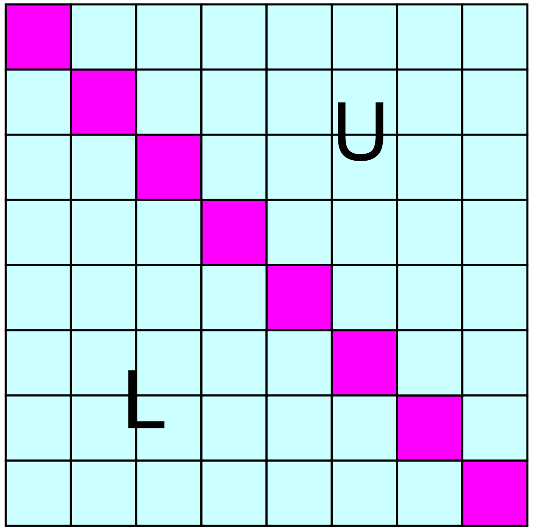
Name	Type	Content
NLU	I	MAX. # of L/U non-zero off-diag. comp. for each mesh (=6)
INLU(NP)	I	# of L/U non-zero off-diag. comp.
IALU(NLU,NP)	I	Column ID of L/U non-zero off-diag. comp.

poi_gen (1/3)

```
do icel= 1, ICELTOT
  icN1= NEIBcell(icel,1)
  icN2= NEIBcell(icel,2)
  icN3= NEIBcell(icel,3)
  icN4= NEIBcell(icel,4)
  icN5= NEIBcell(icel,5)
  icN6= NEIBcell(icel,6)

  if (icN5.ne.0) then
    icou= INLU(icel) + 1
    IALU(icou,icel)= icN5
    INLU(     icel)= icou
  endif

(...)

  if (icN6.ne.0) then
    icou= INLU(icel) + 1
    IALU(icou,icel)= icN6
    INLU(     icel)= icou
  endif
enddo

allocate (indexLU(0:NP))
indexLU(0)= 0
do icel= 1, NP
  indexLU(icel)= INLU(icel) + indexLU(icel-1)
enddo

NPLU= indexLU(NP)
allocate (itemLU(NPLU), AMAT(NPLU))
```



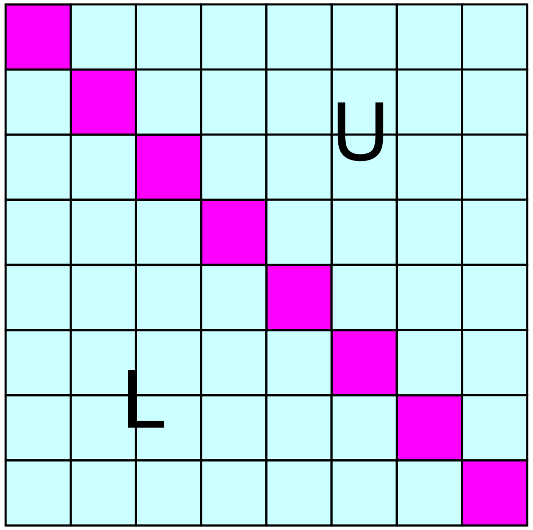
This part is almost same as that of the code for single CPU, thanks to the local data structure



```
!$omp parallel do private (icel,icou,icN1,icN2,icN3,icN4,icN5,icN6) &
!$omp&            private (coef,j,ii,jj,kk,VOL0)

      do icel= 1,  ICELTOT
        icN1= NEIBcell(icel,1)
        icN2= NEIBcell(icel,2)
        icN3= NEIBcell(icel,3)
        icN4= NEIBcell(icel,4)
        icN5= NEIBcell(icel,5)
        icN6= NEIBcell(icel,6)

        VOL0= VOLCEL(icel)

        icou= indexLU(icel-1)
        if (icN5.ne.0) then
          coef    =RDZ * ZAREA
          D(icel)= D(icel) - coef
                   icou = icou + 1
              AMAT(icou)= coef
            itemLU(icou)= icN5
        endif
(...)
        if (icN6.ne.0) then
          coef    = RDZ * ZAREA
          D(icel)= D(icel) - coef
                   icou = icou + 1
              AMAT(icou)= coef
            itemLU(icou)= icN6
        endif

        ii= XYZ(icel,1)
        jj= XYZ(icel,2)
        kk= XYZ(icel,3)

        BFORCE(icel)= -dfloat(ii+jj+kk) * VOL0

      enddo
```

poi_gen (2/3)

loop is up to “IECLTOT”, not “NP”



```
!$omp parallel do private (icel,icou,icN1,icN2,icN3,icN4,icN5,icN6) &
!$omp&            private (coef,j,ii,jj,kk,VOL0)

      do icel= 1,  ICELTOT
        icN1= NEIBcell(icel,1)
        icN2= NEIBcell(icel,2)
        icN3= NEIBcell(icel,3)
        icN4= NEIBcell(icel,4)
        icN5= NEIBcell(icel,5)
        icN6= NEIBcell(icel,6)

        VOL0= VOLCEL(icel)

        icou= indexLU(icel-1)
        if  (icN5.ne.0) then
          coef   =RDZ * ZAREA
          D(icel)= D(icel) - coef
                   icou = icou + 1
              AMAT(icou)= coef
            itemLU(icou)= icN5
        endif
(...)
        ii= XYZ(icel,1)
        jj= XYZ(icel,2)
        kk= XYZ(icel,3)

        BFORCE(icel)= -dfloat(ii+jj+kk) * VOL0
      enddo
```

```
!C
!C +--------------------------+
!C | DIRICHLET BOUNDARY CELLs |
!C +--------------------------+
!C   TOP SURFACE
!C===
      do ib= 1, ZmaxCELtot
        icel= ZmaxCEL(ib)
        coef= 2.d0 * RDZ * ZAREA
        D(icel)= D(icel) - coef
      enddo
!C===
      return
      end
```

poi_gen (3/3)

Solver_PCG

Preconditioning
Dot Products
DAXPY

```
!C
!C +----------------+
!C | {z}= [Minv]{r} |
!C +----------------+
!C===

!$omp parallel do
      do i= 1, N
        W(i,Z)= W(i,R)*W(i,DD)
      enddo
!C===

!C
!C +-------------+
!C | RHO= {r}{z} |
!C +-------------+
!C===
      RHO0= 0.d0
!$omp parallel do private(i) reduction(+:RHO0)
      do i= 1, N
        RHO0= RHO0 + W(i,R)*W(i,Z)
      enddo

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

!C
!C +----------------------+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +----------------------+
!C===

!$omp parallel do
      do i= 1, N
        X(i)  = X(i)   + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo
```

N=ICELTOT

Loops are “1~N (not, 1~NP)”

Precond, DAXPY:
OpenMP directives are just inserted.

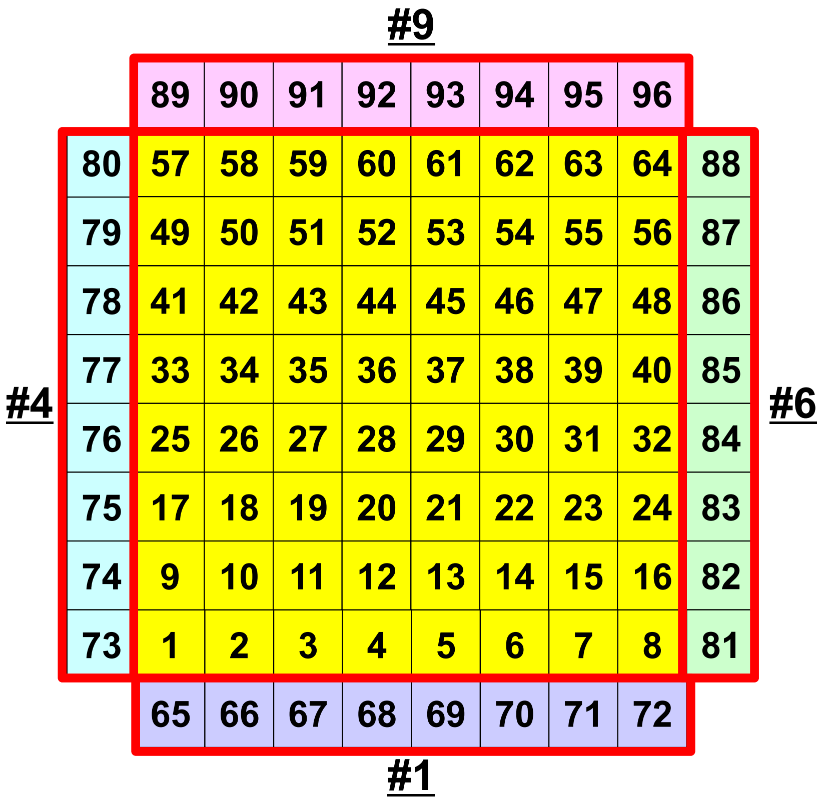
Dot Products:
OpenMP + MPI_Allreduce

Value is calculated on each MPI process via OpenMP, and global-sum is taken by MPI_Allreduce

Solver_PCG

Matrix Vector Products

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

!$omp parallel do private (i,k,VAL)
      do i= 1, N
        VAL= D(i)*W(i,P)
        do k= indexLU(i-1)+1, indexLU(i)
          VAL= VAL + AMAT(k)*W(itemLU(k),P)
        enddo
        W(i,Q)= VAL
      enddo
!C===
```



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

Values on external nodes are “imported” by “SOLVER_SEND_RECV”, then Mat-Vec Products are calculated at each MPI process (in purely local manner).

SEND/RECV (Initial: see Part II-3)

```
!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= STACK_EXPORT(neib-1)
        inum  = STACK_EXPORT(neib  ) - istart
!$omp parallel do private (ii)
        do k= istart+1, istart+inum
           ii    = NOD_EXPORT(k)
           WS(k)= X(ii)
        enddo
        call MPI_ISEND (WS(istart+1), inum, MPI_DOUBLE_PRECISION,          &
                        NEIBPE(neib), 0, MPI_COMM_WORLD, req1(neib), ierr)
      enddo
!C
!C-- RECEIVE
      do neib= 1, NEIBPETOT
        istart= STACK_IMPORT(neib-1)
        inum  = STACK_IMPORT(neib  ) - istart
        call MPI_IRECV (WR(istart+1), inum, MPI_DOUBLE_PRECISION,                    &
                        NEIBPE(neib), 0, MPI_COMM_WORLD, req2(neib), ierr)
      enddo
      call MPI_WAITALL (NEIBPETOT, req2, sta2, ierr)
      do neib= 1, NEIBPETOT
        istart= STACK_IMPORT(neib-1)
        inum  = STACK_IMPORT(neib  ) - istart
!$omp parallel do private (ii)
        do k= istart+1, istart+inum
            ii = NOD_IMPORT(k)
          X(ii)= WR(k)
        enddo
      enddo
      call MPI_WAITALL (NEIBPETOT, req1, sta1, ierr)
```

MPI_Waitall for Receiving
Value of receiving buffer (WR)
can be used (copied to X)

MPI_Waitall for Sending

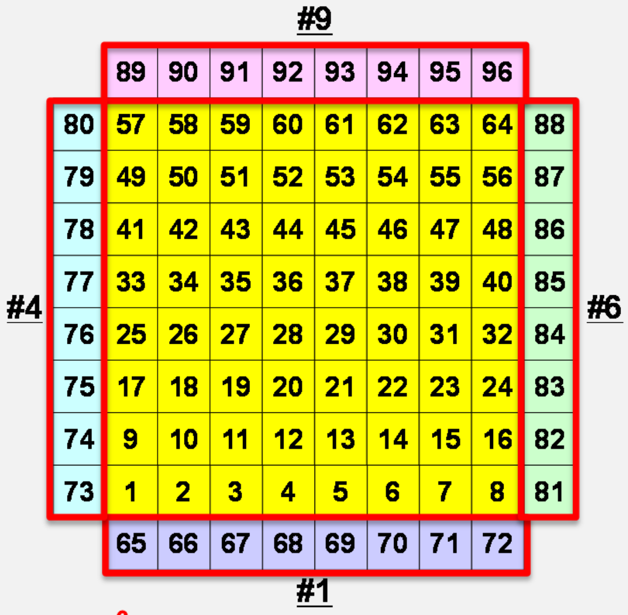
SEND/RECV (<$O-fvm/src>)

```
!C
!C-- INIT.
      allocate (sta1(MPI_STATUS_SIZE,2*NEIBPETOT))
      allocate (req1(2*NEIBPETOT))
!C
!C-- SEND
      do neib= 1, NEIBPETOT
        istart= STACK_EXPORT(neib-1)
        inum  = STACK_EXPORT(neib  ) - istart
!$omp parallel do private (ii)
        do k= istart+1, istart+inum
           ii    = NOD_EXPORT(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
!C
!C-- RECEIVE
      do neib= 1, NEIBPETOT
        istart= STACK_IMPORT(neib-1)
        inum  = STACK_IMPORT(neib  ) - istart
        call MPI_IRECV (WR(istart+1), inum, MPI_DOUBLE_PRECISION,                    &
                        NEIBPE(neib), 0, MPI_COMM_WORLD, req1(NEIBPETOT+neib), ierr)
      enddo

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

      do neib= 1, NEIBPETOT
        istart= STACK_IMPORT(neib-1)
        inum  = STACK_IMPORT(neib  ) - istart
!$omp parallel do private (ii)
        do k= istart+1, istart+inum
            ii = NOD_IMPORT(k)
          X(ii)= WR(k)
        enddo
      enddo
```

“Combined“ MPI_Waitall for Receiving/Sending
Lower overhead compared to separated MPI_Waitall’s

SEND/RECV (<$O-fvm/src0>)

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

!C
!C-- SEND
      do neib= 1, NEIBPETOT
        istart= STACK_EXPORT(neib-1)
        inum  = STACK_EXPORT(neib  ) - istart
!$omp parallel do private (ii)
        do k= istart+1, istart+inum
           ii   = NOD_EXPORT(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


!C
!C-- RECEIVE
      do neib= 1, NEIBPETOT
        istart= STACK_IMPORT(neib-1)
        inum  = STACK_IMPORT(neib  ) - istart
        ii    = NOD_IMPORT(istart+1)
        call MPI_IRECV (X(ii), inum, MPI_DOUBLE_PRECISION,                          &
                        NEIBPE(neib), 0, MPI_COMM_WORLD, req1(NEIBPETOT+neib), ierr)
      enddo

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

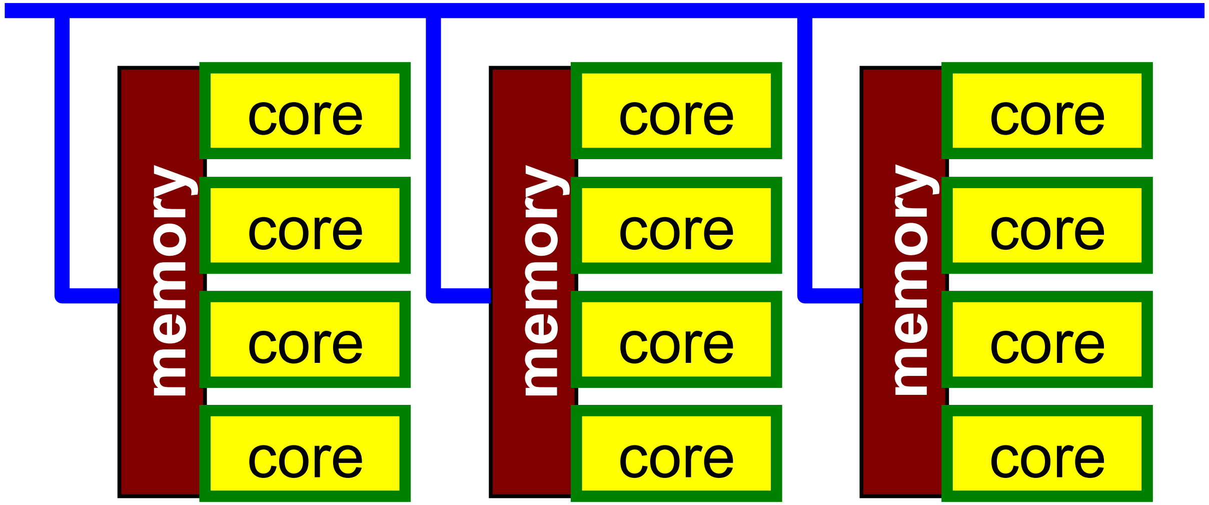


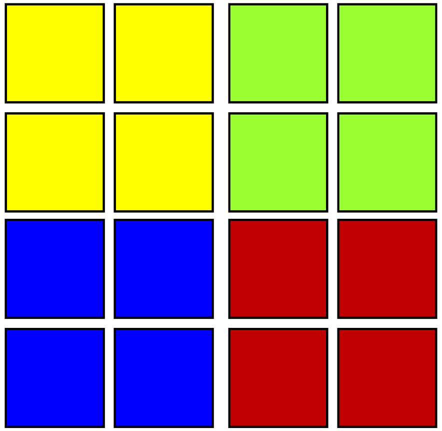
	89	90	91	92	93	94	95	96	
80	57	58	59	60	61	62	63	64	88
79	49	50	51	52	53	54	55	56	87
78	41	42	43	44	45	46	47	48	86
77	33	34	35	36	37	38	39	40	85
76	25	26	27	28	29	30	31	32	84
75	17	18	19	20	21	22	23	24	83
74	9	10	11	12	13	14	15	16	82
73	1	2	3	4	5	6	7	8	81
	65	66	67	68	69	70	71	72	

Because numbering of external nodes is continuous at each neighbor, values at external nodes can be received at X(:) without copying via WS(:) . Could be faster than <$src>.

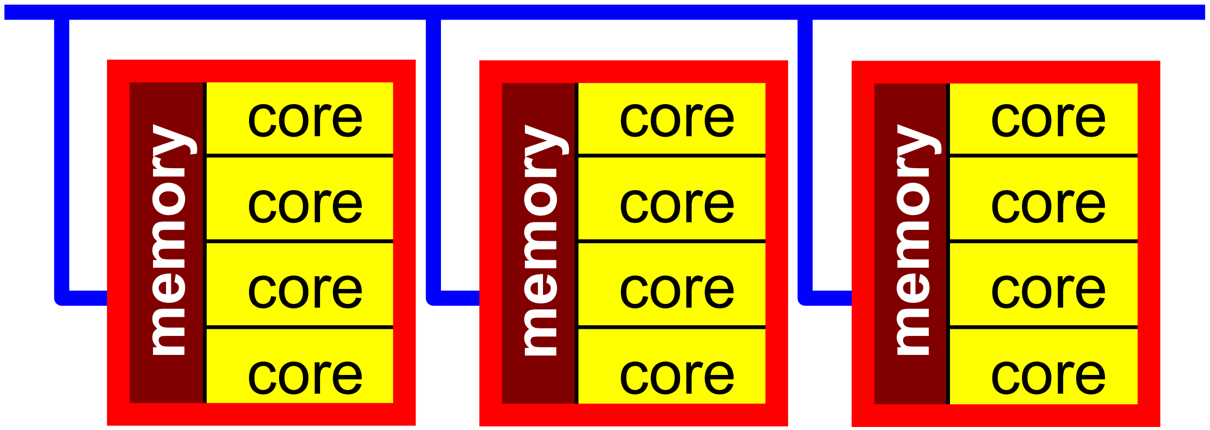
Flat MPI vs. Hybrid

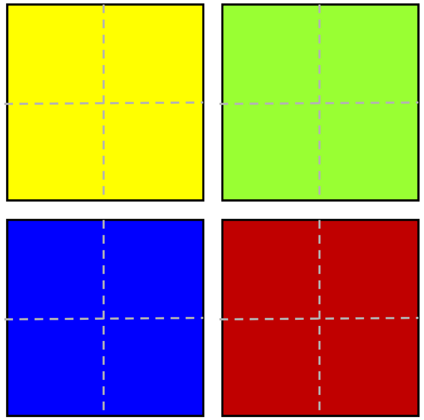
Flat-MPI：Each Core -> Independent





Hybrid：Hierarchal Structure





Hybrid Parallel Programming Model

- Message Passing (e.g. MPI) + Multi Threading (e.g. OpenMP, CUDA, OpenCL, OpenACC etc.)
- In K computer and FX10, hybrid parallel programming is recommended
  - MPI + Automatic Parallelization by Fujitsu's Compiler
  - Personally, I do not like to call this "hybrid" !!!
- Expectations for Hybrid
  - Number of MPI processes (and sub-domains) to be reduced
  - O(10^8-10^9)-way MPI might not scale in Exascale Systems
  - Easily extended to Heterogeneous Architectures
    - CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    - MPI+X: OpenMP, OpenACC, CUDA, OpenCL
- Different exec. files are used for Hybrid and Flat MPI

Compile & Run (<$O-fvm/src>)

Hybrid

```
>$ cd
>$ cd hybrid/fvm/src
>$ make clean
>$ make
>$ ls ../run/sol-mpih
   sol-mpih
>$ cd ../run
(modify goh.sh, INPUT.DAT)
>$ pjsub goh.sh (or mpiexec ...)
```

Flat MPI

```
>$ cd
>$ cd hybrid/fvm/src
>$ make –f make-f clean
>$ make –f make-f
>$ ls ../run/sol-mpif
   sol-mpif
>$ cd ../run
(modify gof.sh, INPUT.DAT)
>$ pjsub gof.sh (or mpiexec ...)
```

Compile & Run (<$O-fvm/src0>)

Hybrid

```
>$ cd
>$ cd hybrid/fvm/src0
>$ make clean
>$ make
>$ ls ../run/sol0-mpih
   sol0-mpih
>$ cd ../run
(modify go0h.sh, INPUT.DAT)
>$ pjsub go0h.sh (or mpiexec ...)
```

Flat MPI

```
>$ cd
>$ cd hybrid/fvm/src0
>$ make –f make-f clean
>$ make –f make-f
>$ ls ../run/sol0-mpif
   sol0-mpif
>$ cd ../run
(modify go0f.sh, INPUT.DAT)
>$ pjsub go0f.sh (or mpiexec ...)
```

Makefile’s

Makefile (Hybrid)

```
F90        = mpifrtpx
F90OPTFLAGS= -Kfast,openmp

F90FLAGS =$(F90OPTFLAGS)

.SUFFIXES:
.SUFFIXES: .o .f .f90 .c
#
.f90.o:; $(F90) -c $(F90FLAGS)  $(F90OPTFLAG) $<
.f.o:; $(F90) -c $(F90FLAGS)  $(F90OPTFLAG) $<
#
OBJS = ¥
solver_SR.o solver_PCG.o struct.o pcg.o ¥
boundary_cell.o cell_metrics.o ¥
input.o main.o poi_gen.o pointer_init.o vis.o

TARGET = ../run/sol-mpih

...
```

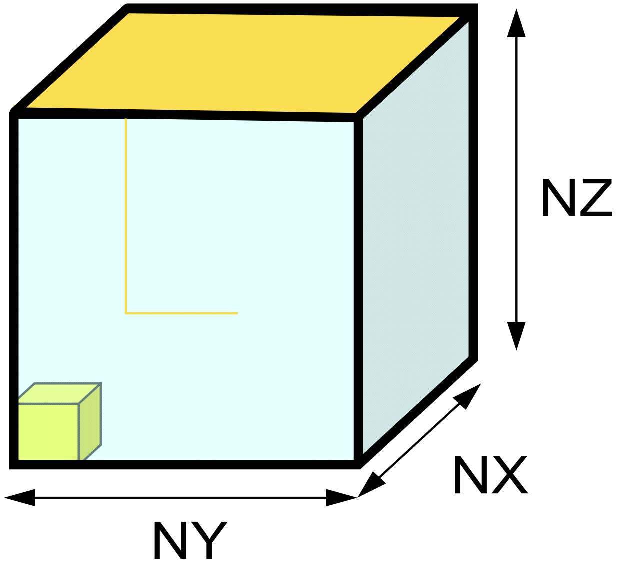
Makefile (Flat MPI)

```
F90        = mpifrtpx
F90OPTFLAGS= -Kfast

F90FLAGS =$(F90OPTFLAGS)

.SUFFIXES:
.SUFFIXES: .o .f .f90 .c
#
.f90.o:; $(F90) -c $(F90FLAGS)  $(F90OPTFLAG) $<
.f.o:; $(F90) -c $(F90FLAGS)  $(F90OPTFLAG) $<
#
OBJS = ¥
solver_SR.o solver_PCG.o struct.o pcg.o ¥
boundary_cell.o cell_metrics.o ¥
input.o main.o poi_gen.o pointer_init.o vis.o

TARGET = ../run/sol-mpif

...
```

- Parallel Distributed Data Structure
- **Parallel FVM Code**
  - **Parallel Visualization**
- Parallel Performance
  - Super-Linear in Strong Scaling

Simplified Parallel Visualization

- Only applicable to mesh files generated by “pmesh”
  - regular structure
- Number of meshes for ParaView has to be specified in the control file of parallel FVM.
- Number of meshes for visualization at each MPI process is defined according to local gradient of ϕ at each MPI process.
  - Octree-based meshes for visualization
  - More meshes are assigned to MPI processes with larger local gradient of temperature
    - Sophisticated rules must be introduced
  - Finally, local meshes are merged to a single file



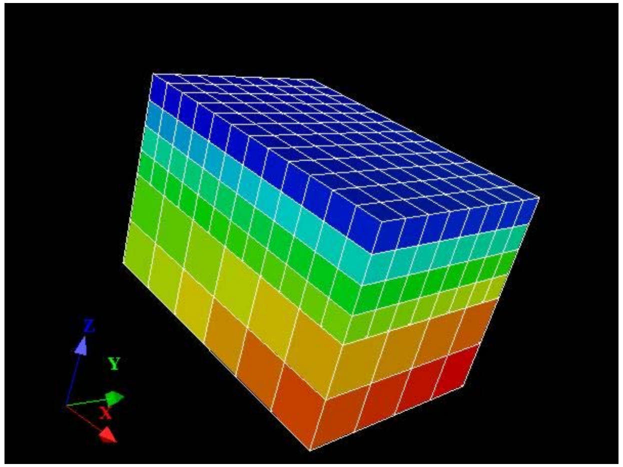
Control Data: INPUT.DAT

```
../pmesh/in                              HEADER
3                                        MEHOD 1:2:3
1.00e-00 1.00e-00 1.00e-00               DX/DY/DZ
0.10 1.0e-08                             OMEGA, EPSICCG
1000                                     VISceltot
```

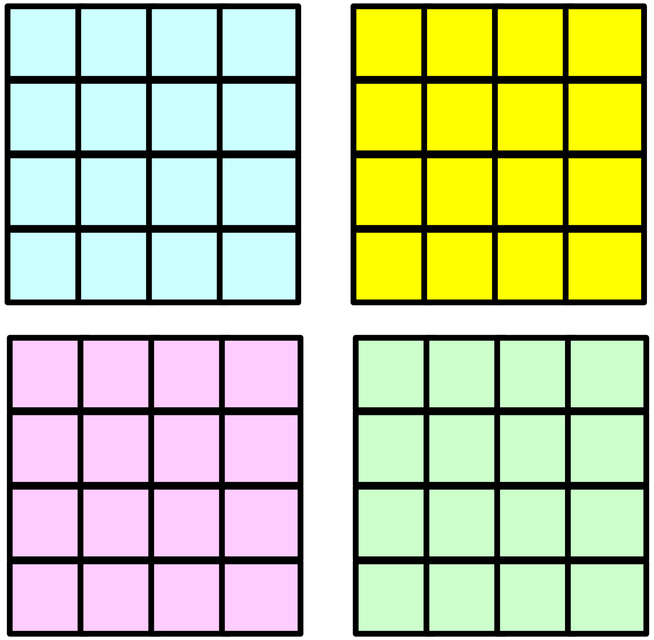
- **HEADER:** Header of Distributed Local Files: Fixed **(../pmesh/in.0, ../pmesh/in.1 ...)**
- **METHOD:** Preconditioning Method, fixed as 3 (Point Jacobi)
- **DX,DY,DZ:** Mesh Size
- **OMEGA:** (not in use)
- **EPSICCG:** Convergence Criteria for CG Method
- **VISceltot:** Approximate number of meshes for visualization

Example: Visualization

- 558 × 372 × 372 (=77,218,272) meshes
- 12 nodes, 192 threads (16 threads/node)
- Visualization
  - 912 nodes, 432 elem’s



- (Reduced) Mesh files for visualization are generated at each MPI process, and then merged to a single file.
  - At process boundaries, nodes are generated in redundant manner. Therefore, we have more redundant nodes if we have more MPI processes.
    - under development



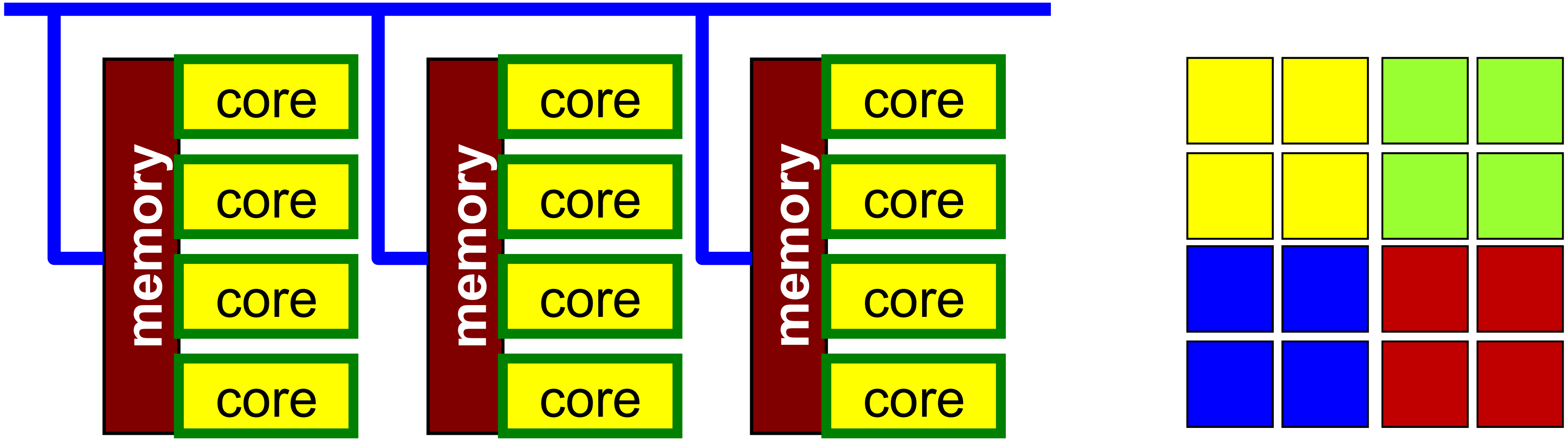
- Parallel Distributed Data Structure
- Parallel FVM Code
  - Parallel Visualization
- **Parallel Performance**
  - Super-Linear in Strong Scaling

Hybrid Parallel Programming Model

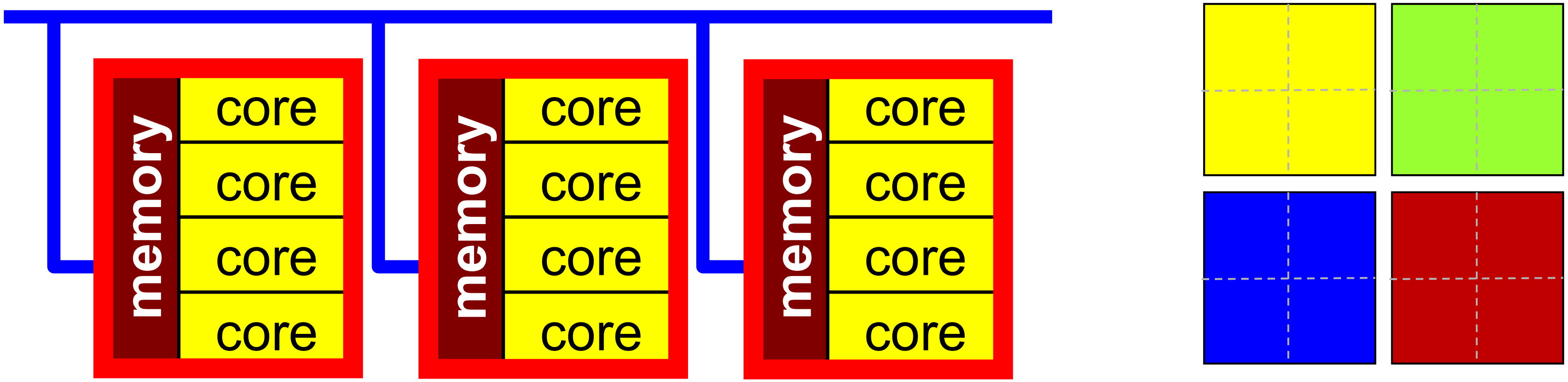
- Message Passing (e.g. MPI) + Multi Threading (e.g. OpenMP, CUDA, OpenCL, OpenACC etc.)
- In K computer and FX10, hybrid parallel programming is recommended
  - MPI + Automatic Parallelization by Fujitsu’s Compiler
  - Personally, I do not like to call this “hybrid” !!!
- Expectations for Hybrid
  - Number of MPI processes (and sub-domains) to be reduced
  - O(10^8-10^9)-way MPI might not scale in Exascale Systems
  - Easily extended to Heterogeneous Architectures
    - CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    - MPI+X: OpenMP, OpenACC, CUDA, OpenCL
- Different exec. files are used for Hybrid and Flat MPI

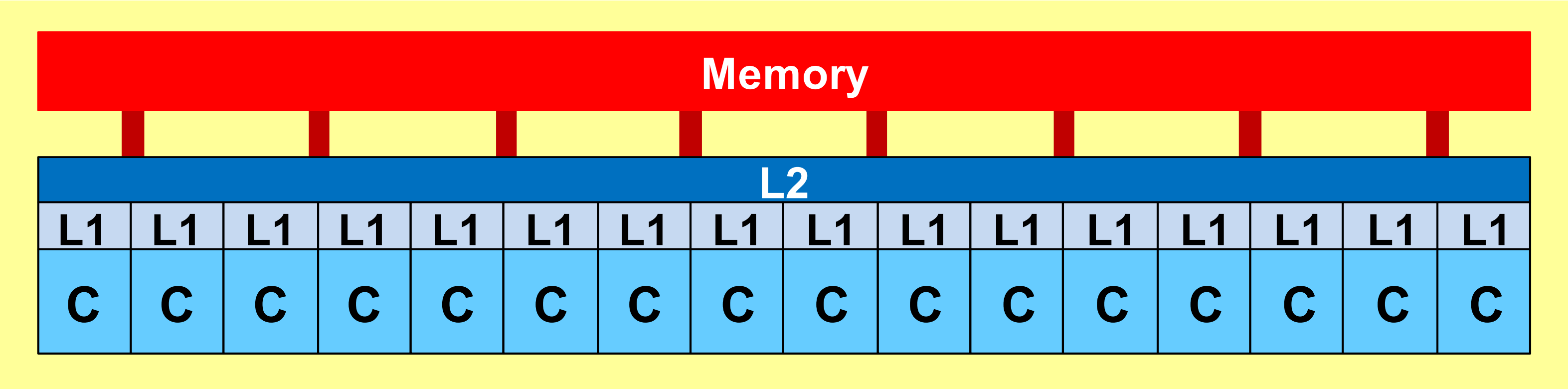
Flat MPI vs. Hybrid

Flat-MPI：Each Core -> Independent



Hybrid：Hierarchal Structure

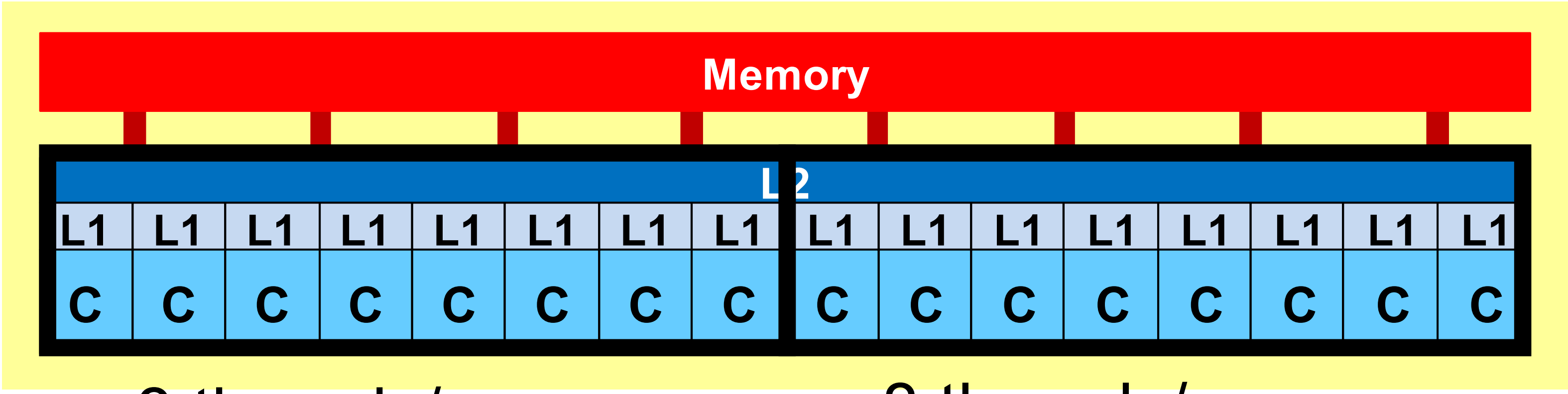



Memory
L2
L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1
C C C C C C C C C C C C C C C C

HB M x N

Number of OpenMP threads per a single MPI process

Number of MPI process per a single node


Memory
L2
L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1
C C C C C C C C C C C C C C C C

8 threads/process

8 threads/process

HB 8 x 2

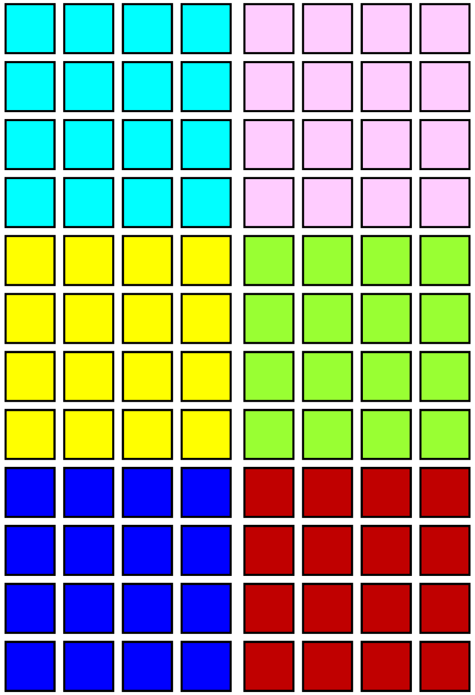
Number of OpenMP threads per a single MPI process

Number of MPI process per a single node

Size (and number) of local data changes according to parallel programming model

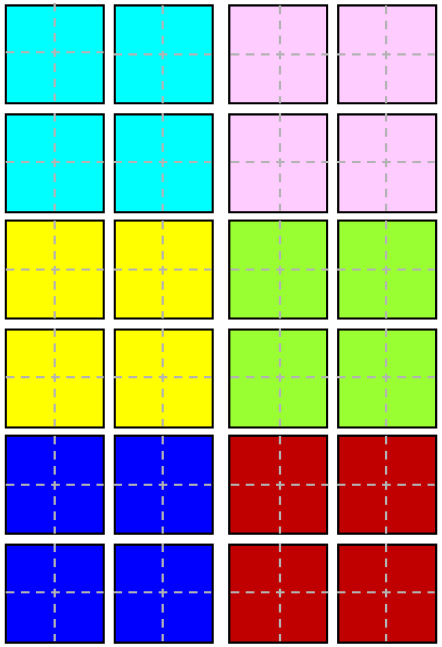
example: 6 nodes, 96 cores

```
NAx NAy NAz
npx npy npz
```



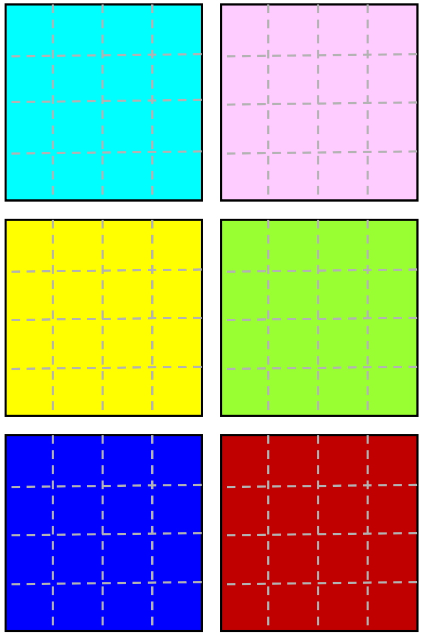
Flat MPI

```
128 192 64
  8  12  1
```



HB 4x4

```
128 192 64
  4   6  1
```



HB 16x1

```
128 192 64
  2   3  1
```

Batch Script (1/2) Env. Var.: OMP_NUM_THREADS

Flat MPI

```
#!/bin/sh
#PJM -L "node=6"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=96"

mpiexec ./sol-mpif
```

Hybrid 16×1

```
#!/bin/sh
#PJM -L "node=6"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=6"

export OMP_NUM_THREADS=16
mpiexec ./sol-mpih
```

Batch Script (2/2)
Env. Var.: OMP_NUM_THREADS

Hybrid 4×4

```
#!/bin/sh
#PJM -L "node=6"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=24"

export OMP_NUM_THREADS=4
mpiexec ./sol-mpih
```

Hybrid 8×2

```
#!/bin/sh
#PJM -L "node=6"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi “proc=12"

export OMP_NUM_THREADS=8
mpiexec ./sol-mpih
```

Time for CG Solver (<$src>)

12 nodes, 300x200x200 meshes

1744 iterations

	sec.
Flat MPI	9.09
HB 4x4	8.74
HB 8x2	8.70
HB 16x1	**8.67**

mesh.inp

```
300   200   200
  3     2     2
```

mg.sh

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM --mpi "proc=12"

mpiexec ./pmesh
```

goh.sh

```
#!/bin/sh
#PJM -L "node=12"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=12"

export OMP_NUM_THREADS=16
mpiexec ./sol-mpih
```

Time for CG Solver (<$src>)

12 nodes, 300x200x200 meshes

1744 iterations

	sec.
Flat MPI	9.09
HB 4x4	8.74
HB 8x2	**8.70**
HB 16x1	8.67

mesh.inp

```
300   200   200
  6     2     2
```

mg.sh

```
#!/bin/sh
#PJM -L "node=2"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=24"

mpiexec ./pmesh
```

goh.sh

```
#!/bin/sh
#PJM -L "node=12"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=24"

export OMP_NUM_THREADS=8
mpiexec ./sol-mpih
```

Time for CG Solver (<$src>)

12 nodes, 300x200x200 meshes
1744 iterations

	sec.
Flat MPI	9.09
HB 4x4	**8.74**
HB 8x2	8.70
HB 16x1	8.67

mesh.inp

```
300  200  200
  6    4    2
```

mg.sh

```
#!/bin/sh
#PJM -L "node=3"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=48"

mpiexec ./pmesh
```

goh.sh

```
#!/bin/sh
#PJM -L "node=12"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=48"

export OMP_NUM_THREADS=4
mpiexec ./sol-mpih
```

Time for CG Solver (<$src>)

12 nodes, 300x200x200 meshes

1744 iterations

	sec.
Flat MPI	**9.09**
HB 4x4	8.74
HB 8x2	8.70
HB 16x1	8.67

mesh.inp

```
300  200  200
 12    4    4
```

mg.sh

```
#!/bin/sh
#PJM -L "node=12"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=192"

mpiexec ./pmesh
```

gof.sh

```
#!/bin/sh
#PJM -L "node=12"
#PJM -L "elapse=00:05:00"
#PJM -j
#PJM -L "rscgrp=lecture7“
#PJM -g “gt17“
#PJM -o "test.lst"
#PJM --mpi "proc=192"

mpiexec ./sol-mpif
```

Which is better ? Flat MPI or Hybrid ...

- Depends on:
  - – Hardware Environment
  - – Number of MPI Processes
  - – Types of Application
  - – Problem Size
- Generally speaking, *hybrid* is better than *flat MPI*, if number of nodes is larger.

Example: FEM Applications on FX10

In this case HB 16x1 is the worst

	ndx,ndy,ndz （#MPI proc.）	Iter's	sec.	
Flat MPI	8 6 4 (192)	1240	73.9	
HB 1×16	8 6 4 (192)	1240	73.6	-Kopenmp OMP_NUM_THREADS=1
HB 2×8	4 6 4 (96)	1240	78.8	
HB 4×4	4 3 4 (48)	1240	80.3	
HB 8×2	4 3 2 (24)	1240	81.1	
HB 16×1	2 3 2 (12)	1240	81.9	

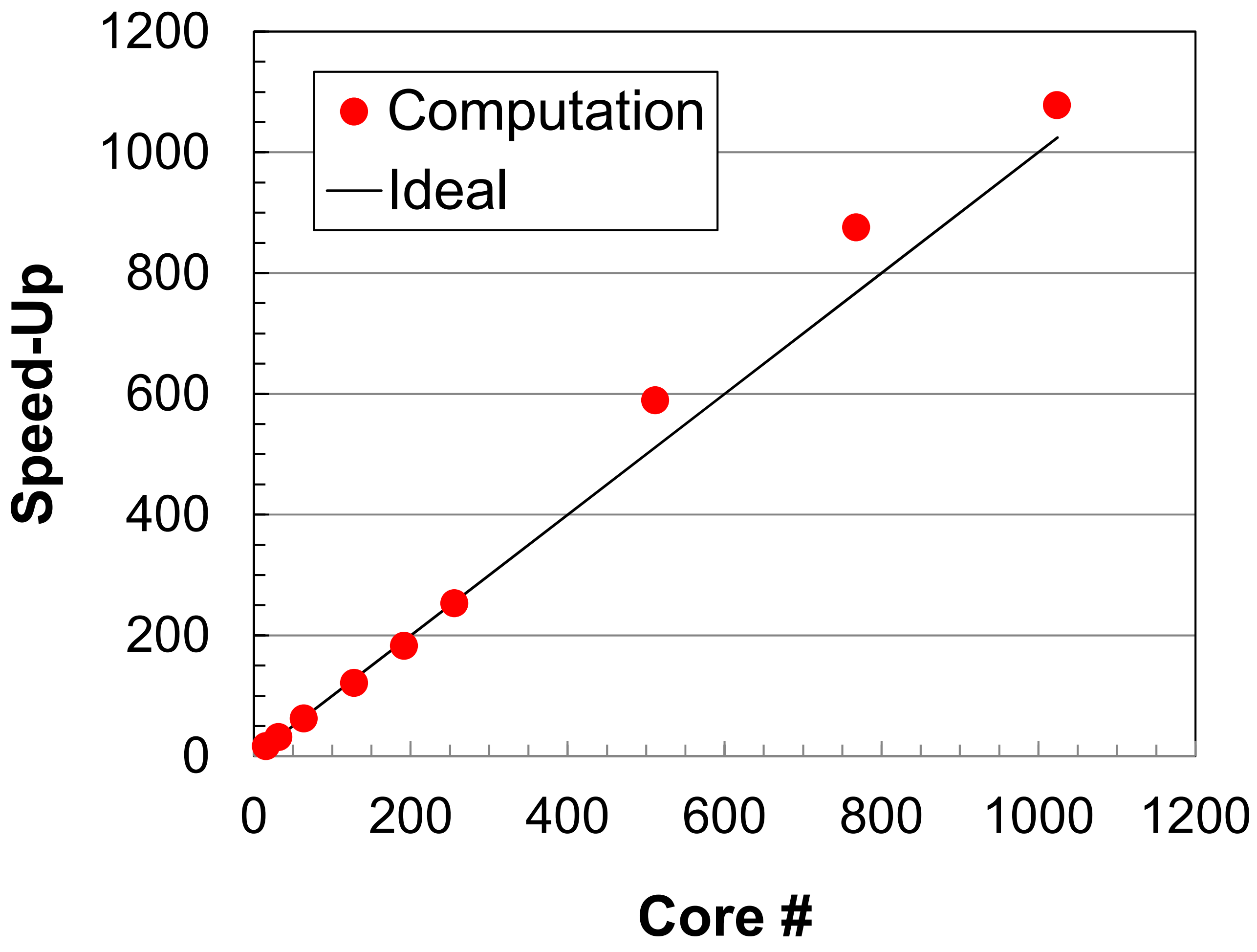
Performance of Parallel Computing Scalability

- Weak Scaling（弱）
  - Solving N^x sized problem using N^x computational resources during same computation time
    - for large-scale problems
    - e.g. CG solver: more iterations needed for larger problems

- Strong Scaling（強）
  - Solving a problem using N^x computational resources during 1/N computation time
    - for faster computation

Strong Scaling HB 16x1

from 1-64 nodes (16-1,024 cores)
100x200x300=6,000,000 meshes, <$src>



- Parallel Distributed Data Structure
- Parallel FVM Code
  - Parallel Visualization
- **Parallel Performance**
  - **Super-Linear in Strong Scaling**

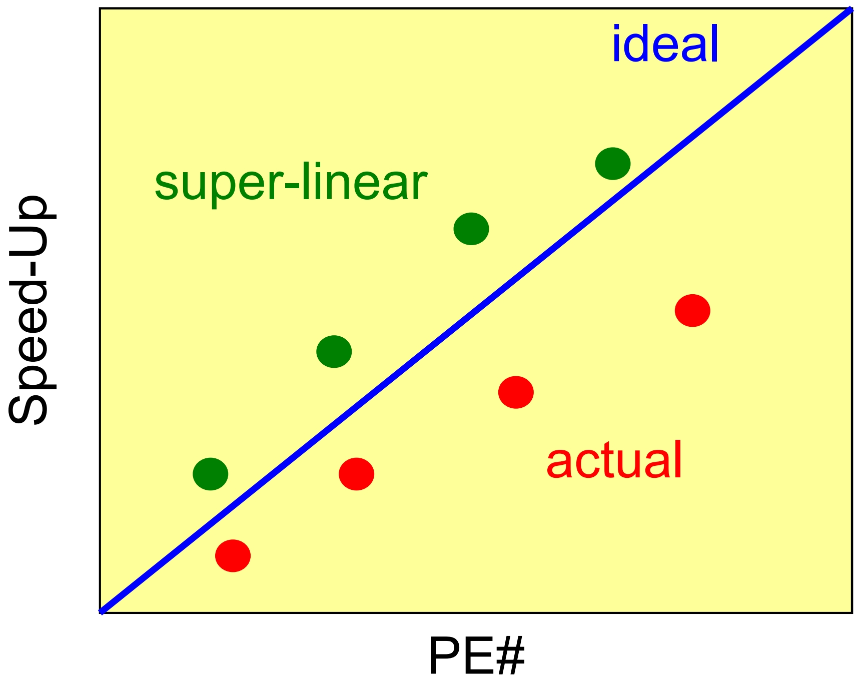
Generally, Performance is lower than ideal one

- Time for MPI communication
  - Time for sending data
  - Communication bandwidth between nodes
  - Time is proportional to size of sending/receiving buffers
- Time for starting MPI
  - latency
  - does not depend on size of buffers
    - depends on number of calling, increases according to process #
  - $O(10^0)$-$O(10^1)$ μsec.

Generally, Performance is lower than ideal one (cont.)

- Synchronization of MPI
  - Increases according to number of processes
- If computation time is relatively small, these effects are not negligible.
  - If the size of messages is small, effect of “latency” is significant.

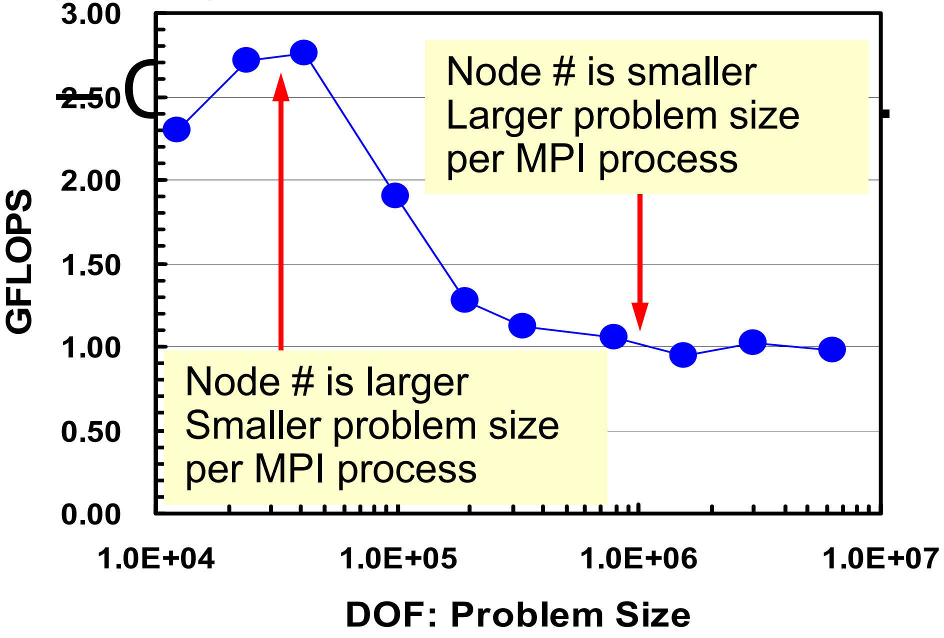
Super-Linear in Strong Scaling



- In strong scaling case where entire problem size is fixed, performance is generally lower than the ideal one due to communication overhead.
- But sometime, actual performance may be better than the ideal one. This is called “super-linear”
  - – only for scalar processors
  - – does not happen in vector processors

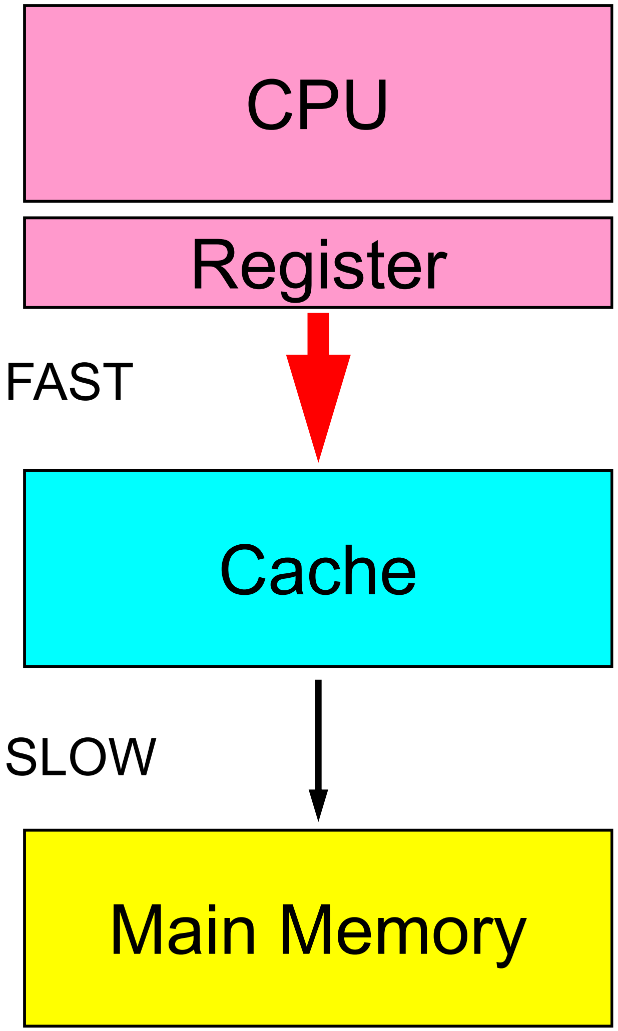
Why does “Super-Linear” happen ?

- Effect of Cache
- In scalar processors, performance for smaller problem is generally better.



Node # is smaller
Larger problem size
per MPI process

Node # is larger
Smaller problem size
per MPI process



Distributed Local Data Structure for Parallel Computation

- Distributed local data structure for domain-to-doain communications has been introduced, which is appropriate for such applications with sparse coefficient matrices (e.g. FDM, FEM, FVM etc.).
  - SPMD
  - Local Numbering: Internal pts to External pts
  - Generalized communication table
- Everything is easy, if proper data structure is defined:
  - Values at boundary pts are copied into sending buffers
  - Send/Recv
  - Values at external pts are updated through receiving buffers