並列アプリケーション開発法入門（III）

2007年7月11日

中島研吾

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

予 定

- 7月11日（水）
  - 「eps_fvm」の並列化
- 7月18日（水）
  - 密行列を使用したアプリケーション

内 容

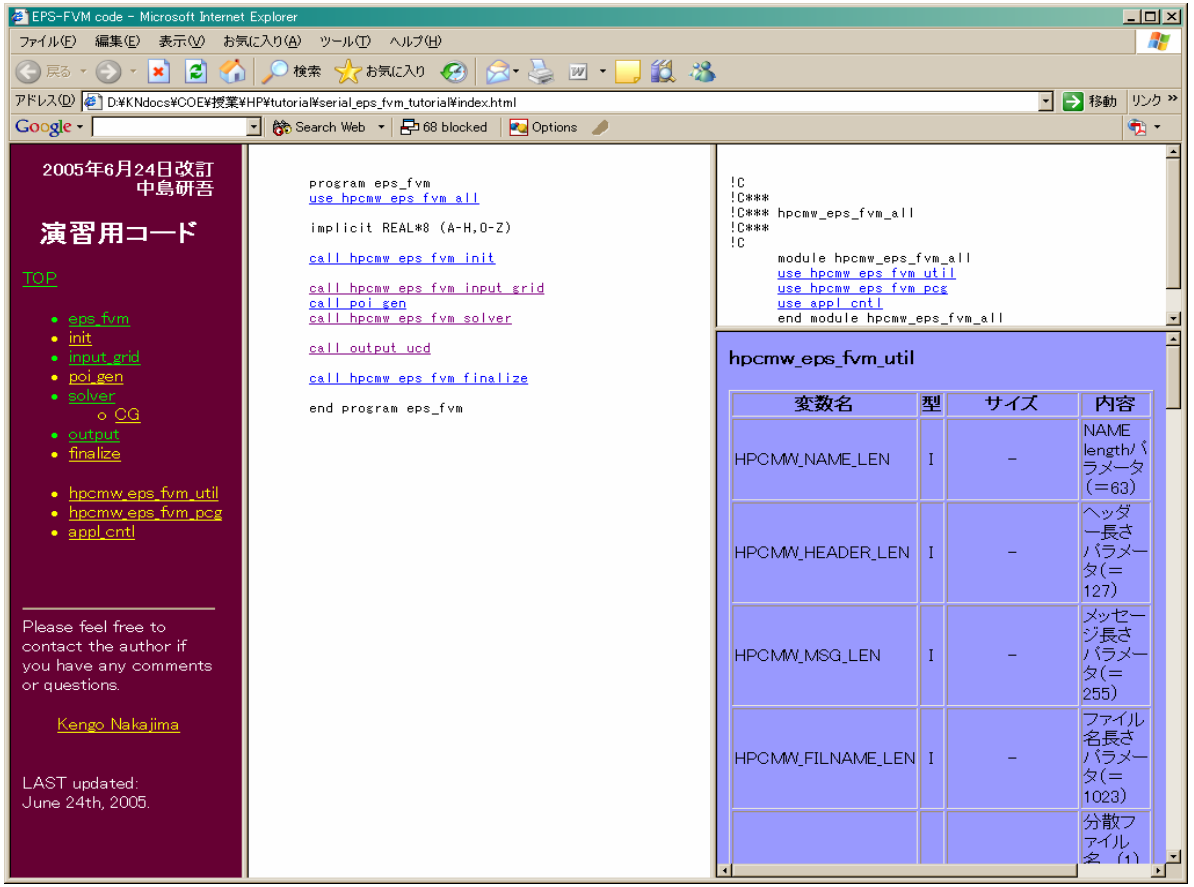
- アプリケーションにおける処理:「eps_fvm」の復習
- 並列化
  - 局所分散データ
  - 共役勾配法
  - 可視化
- 課題P1

プログラム内容に関するチュートリアル FORTRAN版

http://www-solid.eps.s.u-tokyo.ac.jp/~nakajima/07s/CS08/index.html

http://www-solid.eps.s.u-tokyo.ac.jp/~nakajima/tutorial/serial_eps_fvm_tutorial/index.html

MACへ持ってきて展開してもできる



プログラムの概要

$> cd <$P1>/serial
$> cat eps_fvm.f メインプログラム

```
      program eps_fvm

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

      call hpcmw_eps_fvm_init
      call hpcmw_eps_fvm_input_grid

      call poi_gen
      call hpcmw_eps_fvm_solve
      call output_ucd

      call hpcmw_eps_fvm_finalize
      end program eps_fvm
```

「eps_fvm」並列化に向けて

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

「eps_fvm」並列化に向けて

- hpcmw_eps_fvm_input_grid
  - 並列分散メッシュデータ
  - 並列分散通信データ
- poi_gen
  - この部分は（多分）変更不要
- hpcmw_eps_fvm_solver
  - 課題S3と同様の方法
- output_ucd
  - 並列分散処理について考慮する必要あり

「eps_fvm」処理：メイン（1/3）
hpcmw_eps_fvm_init/finalize

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

これらは実際はMPIのサブルーチン（MPI_INIT, MPI_FINALIZE）を呼んで初期化，後処理を実施している。

「eps_fvm」はシリアル処理であるが，「MPI_WTIME」等を利用するために，MPIのライブラリをリンクしている。

「eps_fvm」処理：メイン（2/3）
hpcmw_eps_fvm_all

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

変数の内容について記載したモジュールブロック（コモンブロックのようなもの）。

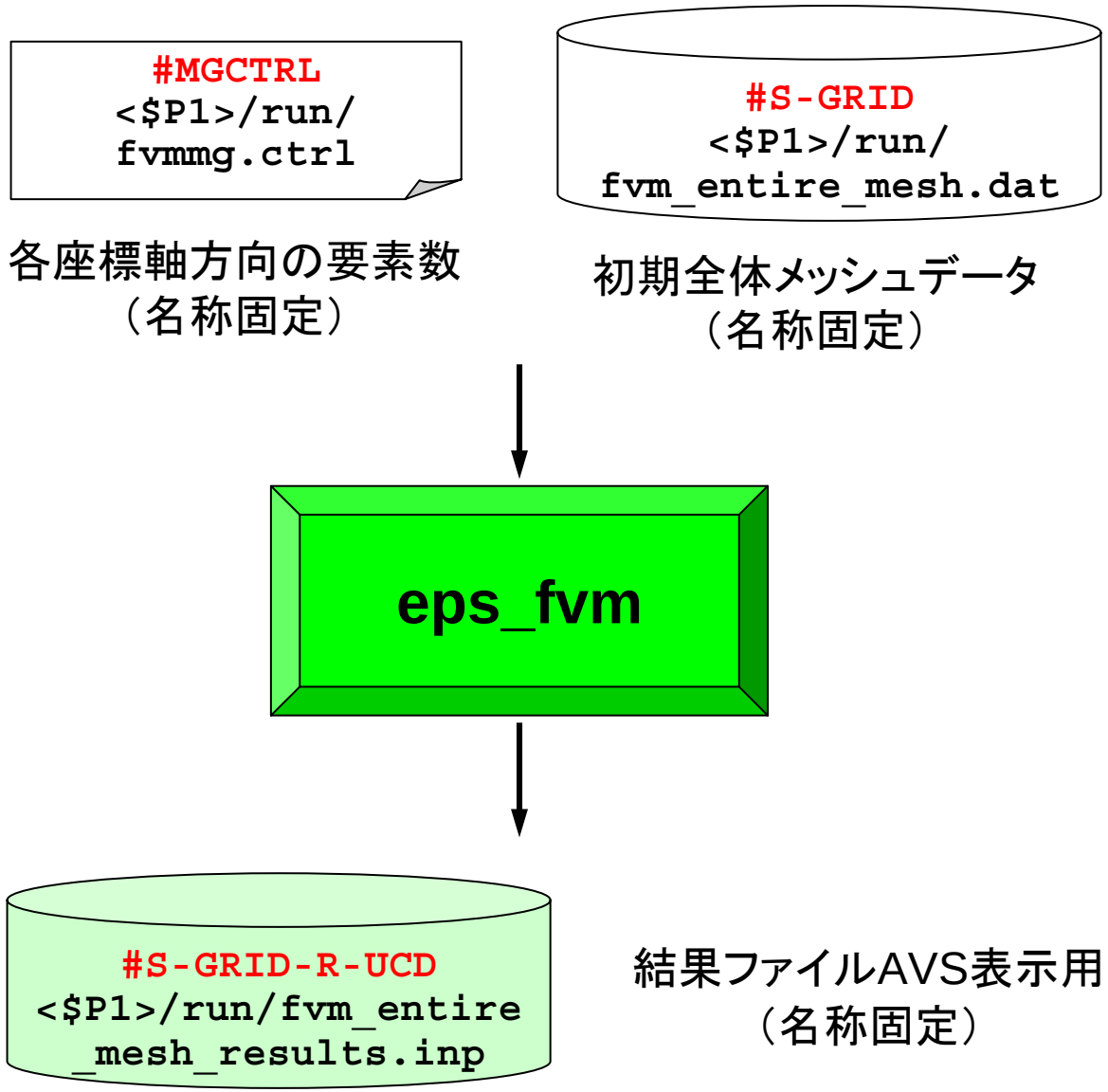
詳細はチュートリアル参照

http://www-solid.eps.s.u-tokyo.ac.jp/~nakajima/07s/CS08/index.html

http://www-solid.eps.s.u-tokyo.ac.jp/~nakajima/tutorial/serial_eps_fvm_tutorial/index.html

MACへ持ってきて展開してもできる

eps_fvm の処理


#MGCTRL
<$P1>/run/
fvmmg.ctrl
#S-GRID
<$P1>/run/
fvm_entire_mesh.dat
各座標軸方向の要素数
（名称固定）
初期全体メッシュデータ
（名称固定）
eps_fvm
#S-GRID-R-UCD
<$P1>/run/fvm_entire
_mesh_results.inp
結果ファイルAVS表示用
（名称固定）

「eps_fvm」処理：メイン（3/3）
その他

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid     メッシュ読み込み(#S-GRID)
call poi_gen                      マトリクス生成
call hpcmw_eps_fvm_solver         線形ソルバー
call output_ucd                   AVS用結果ファイル書き出し(#S-GRID-R-UCD)

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

メッシュ読み込み(#S-GRID)
マトリクス生成
線形ソルバー
AVS用結果ファイル書き出し(#S-GRID-R-UCD)

「eps_fvm」処理：メッシュ読み込み（1/5）

```
      subroutine hpcmw_eps_fvm_input_grid
      use hpcmw_eps_fvm_all
      implicit REAL*8 (A-H,O-Z)

      character(len=HPCMW_NAME_LEN) :: member
      character(len=80            ) :: LINE

!C
!C +------------+
!C | MESH INPUT |
!C +------------+
!C===
      open (22, file='fvmmg.ctrl', status='unknown')
        read (22,*) NX, NY, NZ
      close (22)

      IUNIT= 11
      open (IUNIT,file= 'fvm_entire_mesh.dat', status='unknown')

!C
!C-- NODE
      read (IUNIT, '(10i10)') NODE_tot

      if (NODE_tot.ne.NX*NY*NZ) then
        write (*,'(a)') "incosistent test.grid and test.mesh !!!"
        call hpcmw_eps_fvm_abort
      endif

      allocate (NODE_VOL(NODE_tot), NODE_COND(NODE_tot),                &
     &          NODE_XYZ(3*NODE_tot))

      do i= 1, NODE_tot
        read (IUNIT,'(i10,5e16.6)') ii, NODE_VOL(i), NODE_COND(i),      &
     &                              (NODE_XYZ(3*i-3+k), k=1, 3)
      enddo
```

各座標軸方向の要素数，AVS出力用に必要

各要素情報の読み込み

NODE_tot：
総要素数

「eps_fvm」処理：メッシュ読み込み（1/5）

```
!C
!C-- NODE
      read (IUNIT, '(10i10)') NODE_tot

      if (NODE_tot.ne.NX*NY*NZ) then
        write (*,'(a)') "incosistent test.grid and test.mesh !!!"
        call hpcmw_eps_fvm_abort
      endif

      allocate (NODE_VOL(NODE_tot), NODE_COND(NODE_tot),                &
     &          NODE_XYZ(3*NODE_tot))

      do i= 1, NODE_tot
        read (IUNIT,'(i10,5e16.6)') ii, NODE_VOL(i), NODE_COND(i),      &
     &                              (NODE_XYZ(3*i-3+k), k=1, 3)
      enddo
```

```
12  要素数:NODE_tot
 1       1.000000E+00    1.000000E+00    5.000000E-01    5.000000E-01    5.000000E-01
 2       1.000000E+00    1.000000E+00    1.500000E+00    5.000000E-01    5.000000E-01
 3       1.000000E+00    1.000000E+00    5.000000E-01    1.500000E+00    5.000000E-01
 4       1.000000E+00    1.000000E+00    1.500000E+00    1.500000E+00    5.000000E-01
 5       1.000000E+00    1.000000E+00    5.000000E-01    5.000000E-01    1.500000E+00
 6       1.000000E+00    1.000000E+00    1.500000E+00    5.000000E-01    1.500000E+00
 7       1.000000E+00    1.000000E+00    5.000000E-01    1.500000E+00    1.500000E+00
 8       1.000000E+00    1.000000E+00    1.500000E+00    1.500000E+00    1.500000E+00
 9       1.000000E+00    1.000000E+00    5.000000E-01    5.000000E-01    2.500000E+00
10       1.000000E+00    1.000000E+00    1.500000E+00    5.000000E-01    2.500000E+00
11       1.000000E+00    1.000000E+00    5.000000E-01    1.500000E+00    2.500000E+00
12       1.000000E+00    1.000000E+00    1.500000E+00    1.500000E+00    2.500000E+00
要素番号 要素体積        要素熱伝導率     要素中心x座標    要素中心y座標    要素中心z座標
         NODE_VOL(i)     NODE_COND(i)    NODE_XYZ(3*i-2) NODE_XYZ(3*i-1) NODE_XYZ(3*i)
```

「eps_fvm」処理：メッシュ読み込み（2/5）

コネクティビティ情報

```
!C
!C-- CONNECTION
      read (IUNIT,'(10i10)') CONN_tot
      allocate (CONN_NODE(2*CONN_tot), CONN_COEF(CONN_tot))
      do i= 1, CONN_tot
        read (IUNIT,'( 2i10, 3e16.6)') in1, in2, AREA, D1, D2
        CONN_NODE(2*i-1)= in1
        CONN_NODE(2*i  )= in2
        C1 = NODE_COND(in1)
        C2 = NODE_COND(in2)
        CONN_COEF(i)= AREA / ( D1/C1 + D2/C2 )
      enddo
```

```
20  コネクティビティ総数: CONN_tot
 1          2    1.000000E+00    5.000000E-01    5.000000E-01
 1          3    1.000000E+00    5.000000E-01    5.000000E-01
 1          5    1.000000E+00    5.000000E-01    5.000000E-01
 2          4    1.000000E+00    5.000000E-01    5.000000E-01
…
 6         10    1.000000E+00    5.000000E-01    5.000000E-01
 7          8    1.000000E+00    5.000000E-01    5.000000E-01
 7         11    1.000000E+00    5.000000E-01    5.000000E-01
 8         12    1.000000E+00    5.000000E-01    5.000000E-01
 9         10    1.000000E+00    5.000000E-01    5.000000E-01
 9         11    1.000000E+00    5.000000E-01    5.000000E-01
10         12    1.000000E+00    5.000000E-01    5.000000E-01
11         12    1.000000E+00    5.000000E-01    5.000000E-01
E1         E2    S:要素境界面積    d1:E1重心～        d2:E2重心～
                                    境界面距離          境界面距離

E1= CONN_NODE(2*ic-1)
E2= CONN_NODE(2*ic)
```

CONN_tot:
総コネクティビティ数

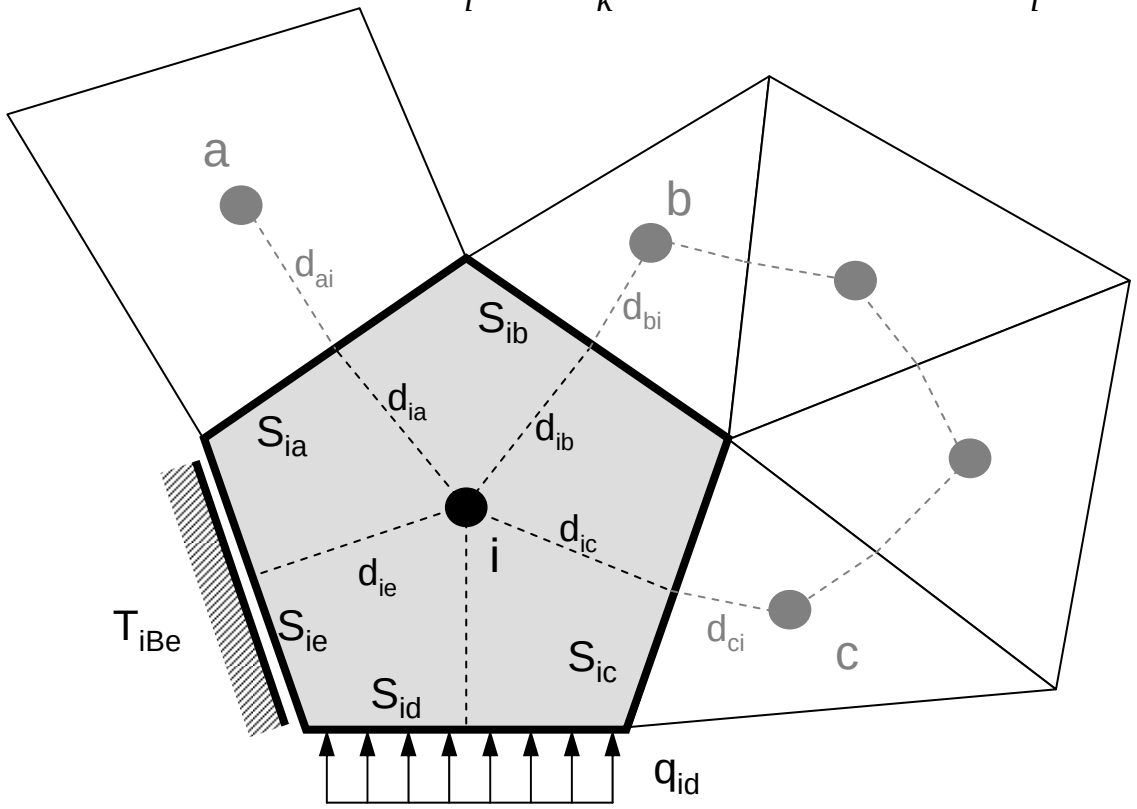
有限体積法による空間離散化

熱流束に関するつりあい式

隣接要素との熱伝導　　温度固定境界

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right)+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right)+\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

要素境界面通過熱流束　　体積発熱

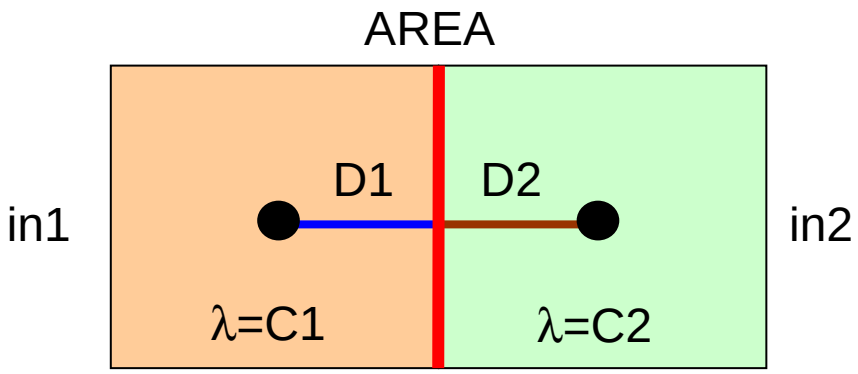


λ　：熱伝導率
V_i　：要素体積
S　：表面面積
d_{ij}　：要素中心から表面までの距離
q　：表面フラックス
Q　：体積発熱
T_{iB}　：境界温度

「eps_fvm」処理：メッシュ読み込み（2/5）

コネクティビティ情報

```
!C
!C-- CONNECTION
      read (IUNIT,'(10i10)') CONN_tot
      allocate (CONN_NODE(2*CONN_tot), CONN_COEF(CONN_tot))
      do i= 1, CONN_tot
        read (IUNIT,'( 2i10, 3e16.6)') in1, in2, AREA, D1, D2
        CONN_NODE(2*i-1)= in1
        CONN_NODE(2*i  )= in2
        C1 = NODE_COND(in1)
        C2 = NODE_COND(in2)
        CONN_COEF(i)= AREA / ( D1/C1 + D2/C2 )
      enddo
```

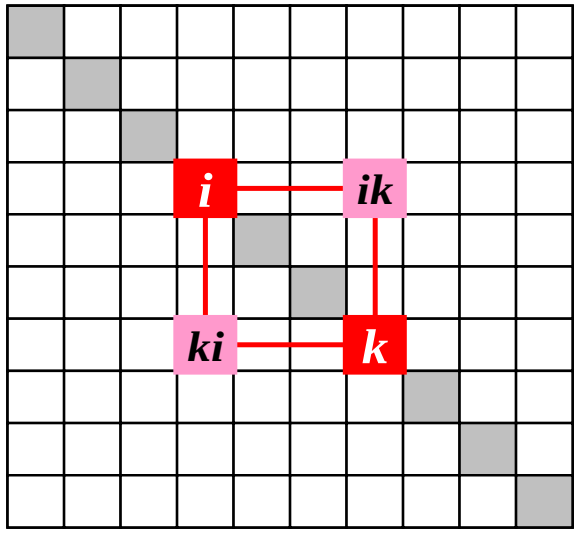


$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

CONN_COEF

CONN_tot:
総コネクティビティ数
（これを2倍すると非対角成分総数）

非対角成分の計算を実施している

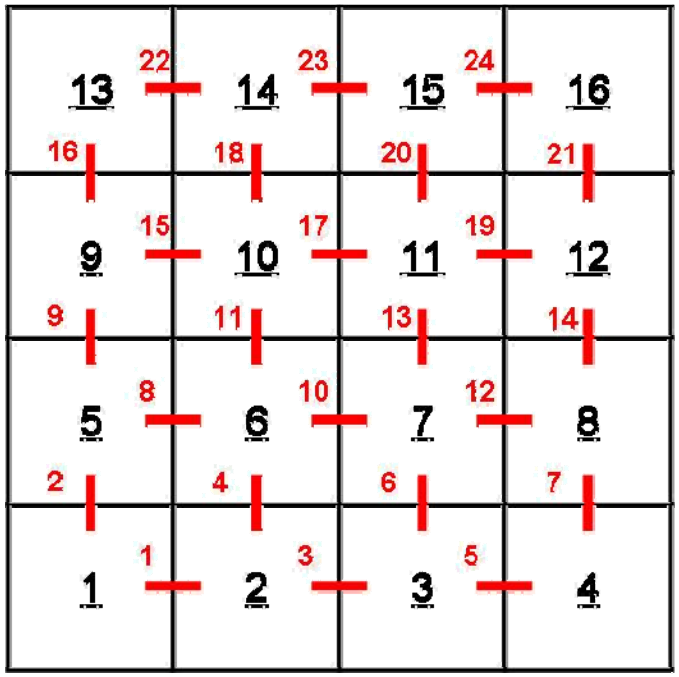


係数行列の非対角成分

要素間コネクティビティの例

```
16
 1    1.000000E+00    1.000000E+00    5.000000E-01    5.000000E-01    5.000000E-01
 2    1.000000E+00    1.000000E+00    1.500000E+00    5.000000E-01    5.000000E-01
 3    1.000000E+00    1.000000E+00    2.500000E+00    5.000000E-01    5.000000E-01
 4    1.000000E+00    1.000000E+00    3.500000E+00    5.000000E-01    5.000000E-01
 5    1.000000E+00    1.000000E+00    5.000000E-01    1.500000E+00    5.000000E-01
 6    1.000000E+00    1.000000E+00    1.500000E+00    1.500000E+00    5.000000E-01
 7    1.000000E+00    1.000000E+00    2.500000E+00    1.500000E+00    5.000000E-01
 8    1.000000E+00    1.000000E+00    3.500000E+00    1.500000E+00    5.000000E-01
 9    1.000000E+00    1.000000E+00    5.000000E-01    2.500000E+00    5.000000E-01
10    1.000000E+00    1.000000E+00    1.500000E+00    2.500000E+00    5.000000E-01
11    1.000000E+00    1.000000E+00    2.500000E+00    2.500000E+00    5.000000E-01
12    1.000000E+00    1.000000E+00    3.500000E+00    2.500000E+00    5.000000E-01
13    1.000000E+00    1.000000E+00    5.000000E-01    3.500000E+00    5.000000E-01
14    1.000000E+00    1.000000E+00    1.500000E+00    3.500000E+00    5.000000E-01
15    1.000000E+00    1.000000E+00    2.500000E+00    3.500000E+00    5.000000E-01
16    1.000000E+00    1.000000E+00    3.500000E+00    3.500000E+00    5.000000E-01
24
 1          2    1.000000E+00    5.000000E-01    5.000000E-01
 1          5    1.000000E+00    5.000000E-01    5.000000E-01
 2          3    1.000000E+00    5.000000E-01    5.000000E-01
 2          6    1.000000E+00    5.000000E-01    5.000000E-01
 3          4    1.000000E+00    5.000000E-01    5.000000E-01
 3          7    1.000000E+00    5.000000E-01    5.000000E-01
 4          8    1.000000E+00    5.000000E-01    5.000000E-01
 5          6    1.000000E+00    5.000000E-01    5.000000E-01
 5          9    1.000000E+00    5.000000E-01    5.000000E-01
 6          7    1.000000E+00    5.000000E-01    5.000000E-01
 6         10    1.000000E+00    5.000000E-01    5.000000E-01
 7          8    1.000000E+00    5.000000E-01    5.000000E-01
 7         11    1.000000E+00    5.000000E-01    5.000000E-01
 8         12    1.000000E+00    5.000000E-01    5.000000E-01
 9         10    1.000000E+00    5.000000E-01    5.000000E-01
 9         13    1.000000E+00    5.000000E-01    5.000000E-01
10         11    1.000000E+00    5.000000E-01    5.000000E-01
10         14    1.000000E+00    5.000000E-01    5.000000E-01
11         12    1.000000E+00    5.000000E-01    5.000000E-01
11         15    1.000000E+00    5.000000E-01    5.000000E-01
12         16    1.000000E+00    5.000000E-01    5.000000E-01
13         14    1.000000E+00    5.000000E-01    5.000000E-01
14         15    1.000000E+00    5.000000E-01    5.000000E-01
15         16    1.000000E+00    5.000000E-01    5.000000E-01
 4
 4    1.000000E+00    5.000000E-01    0.000000E+00
 8    1.000000E+00    5.000000E-01    0.000000E+00
12    1.000000E+00    5.000000E-01    0.000000E+00
16    1.000000E+00    5.000000E-01    0.000000E+00
```

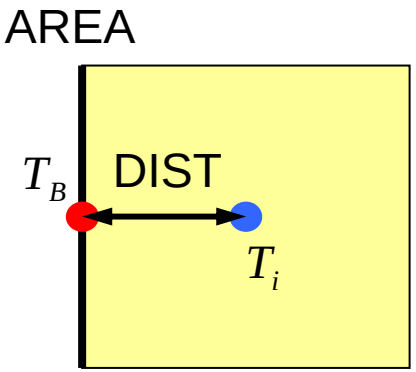
```
4
 1    1.000000E+00    1.000000E+00
 5    1.000000E+00    1.000000E+00
 9    1.000000E+00    1.000000E+00
13    1.000000E+00    1.000000E+00
 4
 6    1.000000E+00
 7    1.000000E+00
10    1.000000E+00
11    1.000000E+00
```



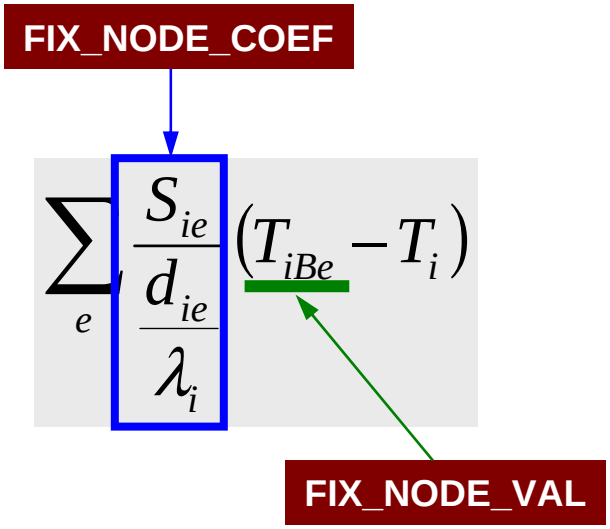
「eps_fvm」処理：メッシュ読み込み（3/5）

ディリクレ境界条件

```
!C
!C-- DIRICHLET
      read (IUNIT,'(10i10)') FIX_NODE_tot
      allocate (FIX_NODE_ID(FIX_NODE_tot), FIX_NODE_COEF(FIX_NODE_tot))
      allocate (FIX_NODE_VAL(FIX_NODE_tot))

      do i= 1, FIX_NODE_tot
        read (IUNIT, '(i10, 3e16.6)')                                    &
     &        FIX_NODE_ID(i), AREA, DIST, FIX_NODE_VAL(i): Tk
        icel= FIX_NODE_ID(i)
        COND= NODE_COND(icel)
        FIX_NODE_COEF(i)= AREA / (DIST/COND)
      enddo
```




```
 6   ディリクレ境界条件を与える要素数: FIX_NODE_tot
 2       1.000000E+00    5.000000E-01    0.000000E+00
 4       1.000000E+00    5.000000E-01    0.000000E+00
 6       1.000000E+00    5.000000E-01    0.000000E+00
 8       1.000000E+00    5.000000E-01    0.000000E+00
10       1.000000E+00    5.000000E-01    0.000000E+00
12       1.000000E+00    5.000000E-01    0.000000E+00
要素番号   境界面面積S      境界面と要素重心距離d   境界値TB
FIX_NODE_ID(ib)                          FIX_NODE_VAL(ib)
```



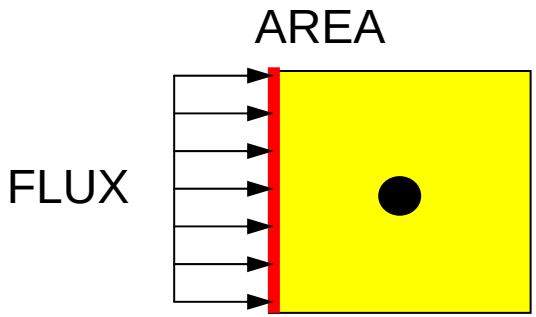
$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}(T_k - T_i) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}(T_{iBe} - T_i) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

「eps_fvm」処理：メッシュ読み込み（4/5）

ノイマン境界条件

```
!C
!C-- NEUMANN
      read (IUNIT,'(10i10)') SURF_NODE_tot
      allocate                                                          &
     &   (SURF_NODE_ID  (SURF_NODE_tot), SURF_NODE_FLUX(SURF_NODE_tot))

      do i= 1, SURF_NODE_tot
        read (IUNIT, '(i10, 3e16.6)') SURF_NODE_ID(i), AREA, FLUX
        SURF_NODE_FLUX(i)= AREA*FLUX
      enddo
```

```
 6   ノイマン境界条件を与える要素数: SURF_NODE_tot
 1       1.000000E+00    1.000000E+00
 3       1.000000E+00    1.000000E+00
 5       1.000000E+00    1.000000E+00
 7       1.000000E+00    1.000000E+00
 9       1.000000E+00    1.000000E+00
11       1.000000E+00    1.000000E+00
要素番号   境界表面積S       表面フラックスQS
SURF_NODE_ID(ib)
```



$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right)+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right)+\sum_d S_{id}\dot{q}_{id}+V_i\dot{Q}_i = 0$$

SURF_NODE_FLUX

「eps_fvm」処理：メッシュ読み込み（5/5）

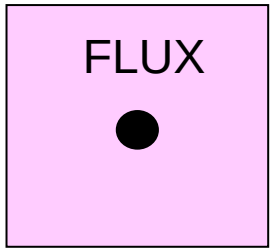
体積発熱

```
!C
!C-- BODY FLUX
      read (IUNIT,'(10i10)') BODY_NODE_tot

      allocate (BODY_NODE_FLUX(NODE_tot))
      do i= 1, BODY_NODE_tot
        read (IUNIT, '(i10, 3e16.6)') icel, FLUX
        BODY_NODE_FLUX(icel)= FLUX * NODE_VOL(icel)
      enddo
```

```
4    体積発熱を与える要素数
5        1.000000E+00
6        1.000000E+00
7        1.000000E+00
8        1.000000E+00
要素番号   体積フラックスQV
```

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

BODY_NODE_FLUX



メッシュ関連変数（1/2）

変数名	型	配列サイズ	内容
NODE_tot	I	-	内点数＋外点数
intNODE_tot	I	-	内点数
NODE_GLOBAL(:)	I	NODE_tot	グローバル要素番号
NODE_VOL(:)	R	NODE_tot	要素体積
NODE_COND(:)	R	NODE_tot	要素熱伝導率
NODE_XYZ(:)	R	3*NODE_tot	要素重心座標（3次元）
CONN_tot	I	-	コネクティビティ総数
CONN_node(:)	I	2*CONN_tot	コネクティビティ構成要素
CONN_COEF(:)	R	CONN_tot	コネクティビティ係数
FIX_NODE_tot	I	-	ディリクレ境界条件適用要素数
FIX_NODE_ID(:)	I	FIX_NODE_tot	ディリクレ境界条件適用要素番号
FIX_NODE_COEF(:)	R	FIX_NODE_tot	ディリクレ境界条件係数
FIX_NODE_VAL(:)	R	FIX_NODE_tot	ディリクレ境界条件値
SURF_NODE_tot	I	-	ノイマン境界条件適用要素数
SURF_NODE_ID(:)	I	SURF_NODE_tot	ノイマン境界条件適用要素番号
SURF_NODE_FLUX(:)	R	SURF_NODE_tot	ノイマン境界条件フラックス
BODY_NODE_tot	I	-	体積発熱境界条件適用要素数
BODY_NODE_ID(:)	I	BODY_NODE_tot	体積発熱境界条件適用要素番号
BODY_NODE_FLUX(:)	R	BODY_NODE_tot	体積発熱境界条件フラックス

シリアル計算では
`NODE_tot= intNODE_tot`

メッシュ関連変数（2/2）

変数名	型	配列サイズ	内容
PETOT	I	-	プロセッサ数
errno	I	-	エラーもどり値
my_rank	I	-	ランク番号
n_neighbor_pe	I	-	隣接領域数
neighbor_pe(:)	I	n_neighbor_pe	隣接領域ID
import_index(:)	I	0:n_neighbor_pe	受信テーブル用インデックス
import_item(:)	I	import_index(n_neighbor_pe)	受信テーブル
export_index(:)	I	0:n_neighbor_pe	送信テーブル用インデックス
export_item(:)	I	export_index(n_neighbor_pe)	送信テーブル

これは主として並列計算用

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

メッシュ読み込み(#S-GRID)
マトリクス生成
線形ソルバー
AVS用結果ファイル書き出し(#S-GRID-R-UCD)

マトリクス関連変数表

変数名	型	サイズ	内容
NPLU	I	-	連立一次方程式係数マトリクス非対角成分総数
D(:)	R	NODE_tot	連立一次方程式係数マトリクス対角成分
PHI(:)	R	NODE_tot	連立一次方程式未知数ベクトル
BFORCE(:)	R	NODE_tot	連立一次方程式右辺ベクトル
index(:)	I	0:NODE_tot	係数マトリクス非対角成分要素番号用一次元圧縮配列（非対角成分数）
item(:)	I	NPLU	係数マトリクス非対角成分要素番号用一次元圧縮配列（非対角成分要素番号）
AMAT(:)	R	NPLU	係数マトリクス非対角成分要素番号用一次元圧縮配列（非対角成分）

```
do i= 1, N
  q(i)= D(i)*p(i)
  do k= index(i-1)+1, index(i)
    q(i)= q(i) + AMAT(k)*p(item(k))
  enddo
enddo
```

行列ベクトル積への適用

非ゼロ成分のみを格納，疎行列向け方法

D (i)	対角成分(実数，i=1,N)
INDEX(i)	非対角成分数に関する一次元配列 (整数，i=0,N)
ITEM(k)	非対角成分の要素番号 (整数，k=1， INDEX(N))
AMAT(k)	非対角成分 (実数，k=1， INDEX(N))

```
{Y}= [A]{X}

do i= 1, N
  Y(i)= D(i)*X(i)
  do k= INDEX(i-1)+1, INDEX(i)
    Y(i)= Y(i) + AMAT(k)*X(ITEM(k))
  enddo
enddo
```

「eps_fvm」処理：マトリクス生成（1/5）

```
!C
!C***
!C*** POI_GEN
!C***
!C
!C    generate COEF. MATRIX for POISSON equations
!C
      subroutine POI_GEN

      use hpcmw_eps_fvm_all
      implicit REAL*8 (A-H,O-Z)
      integer, pointer :: IWKX(:,:)
!C
!C +-------+
!C | INIT. |
!C +-------+
!C===

!C
!C-- MATRIX
      nn = intNODE_tot
      nnp=    NODE_tot

      allocate (BFORCE(nnp), D(nn), PHI(nnp))
      allocate (index(0:nn))

      BFORCE= 0.d0
         PHI= 0.d0
           D= 0.d0

      index= 0
```

「eps_fvm」処理：マトリクス生成（2/5）

```
!C
!C-- ETC.
      allocate (IWKX(intNODE_tot,6))
      IWKX= 0

      do ic= 1, CONN_tot
        in1= CONN_NODE(2*ic-1)
        in2= CONN_NODE(2*ic  )

        ik1= index(in1) + 1
        IWKX (in1,ik1)= ic
        index(in1    )= ik1

        ik2= index(in2) + 1
        IWKX (in2,ik2)= ic
        index(in2    )= ik2
      enddo

      do i= 1, nn
        index(i)= index(i-1) + index(i)
      enddo

      NPLU= index(nn)

      allocate (item(NPLU), AMAT(NPLU))
```

非対角成分数算出
非対角成分抽出

index(in)：
各要素の非対角成分数
（のインデックス）

IWKX(in,1-6)：
各要素の非対角成分の
コネクティビティID

非対角要素数を増やした場合は
IWKXのサイズを増やせばよい
（現在は隣接要素数が6までに限定されている）

「eps_fvm」処理：マトリクス生成（2/5）

```
!C
!C-- ETC.
      allocate (IWKX(intNODE_tot,7))
      IWKX= 0

      do ic= 1, CONN_tot
        in1= CONN_NODE(2*ic-1)
        in2= CONN_NODE(2*ic  )

        ik1= index(in1) + 1
        IWKX (in1,ik1)= ic
        index(in1    )= ik1

        ik2= index(in2) + 1
        IWKX (in2,ik2)= ic
        index(in2    )= ik2
      enddo

      do i= 1, nn
        index(i)= index(i-1) + index(i)
      enddo

      NPLU= index(nn)

      allocate (item(NPLU), AMAT(NPLU))
```

index(in)：
各要素の非対角成分数のインデックス

NPLU：
非対角成分の総数

「eps_fvm」処理：マトリクス生成（3/5）

```
      do i= 1, intNODE_tot
        do j= 1, index(i)-index(i-1)
          k= index(i-1) + j

          ic = IWKX(i,j)
          in1= CONN_NODE(2*ic-1)
          in2= CONN_NODE(2*ic  )

          if (in1.eq.i) then
            item(k)= in2
           else
            item(k)= in1
          endif
        enddo
      enddo
!C===

!C
!C +----------------------------+
!C | INTERIOR NODEs + BODY FLUX |
!C +----------------------------+
!C===
      do icel= 1, intNODE_tot
        BFORCE(icel)= BFORCE(icel) + BODY_NODE_FLUX(icel)
      enddo

      do i= 1, intNODE_tot
        do j= index(i-1)+1, index(i)
          icon= IWKX(i,j-index(i-1))
          AMAT(j)= -CONN_COEF(icon)
          D   (i)= D(i) + CONN_COEF(icon)
        enddo
      enddo
      deallocate (IWKX)
!C===
```

item(k)：
非対角成分

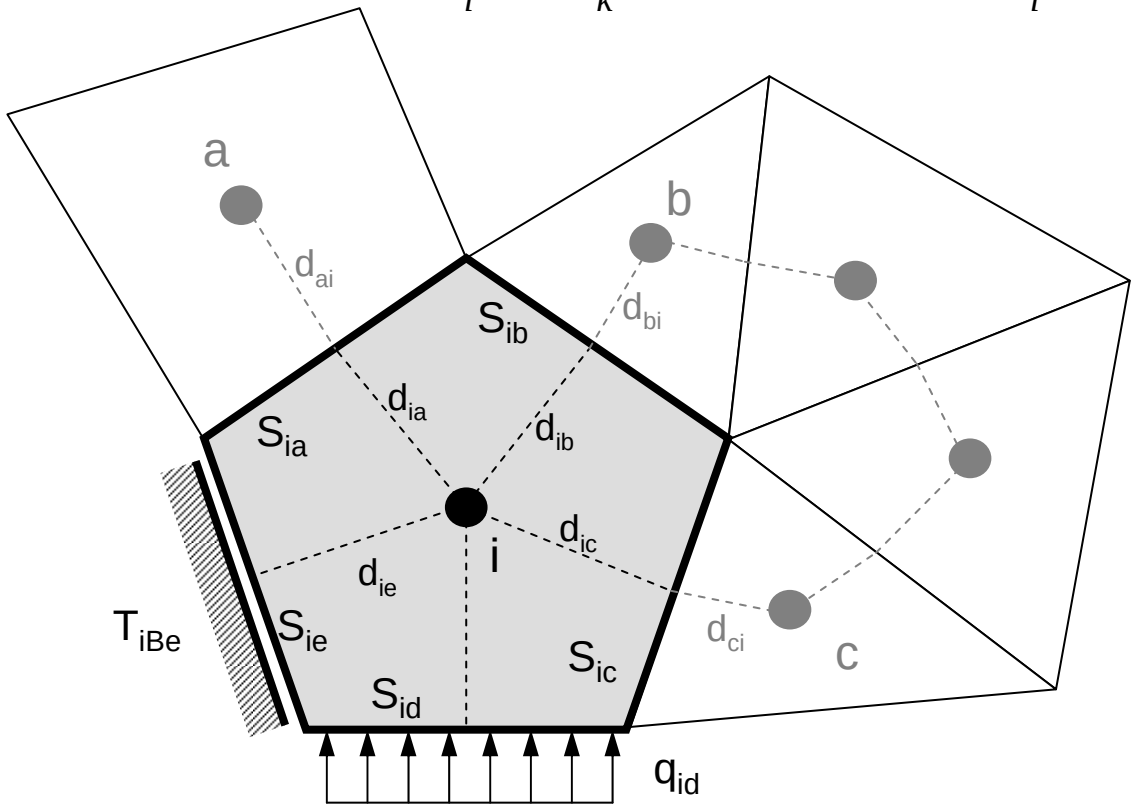
有限体積法による空間離散化

熱流束に関するつりあい式

隣接要素との熱伝導　　温度固定境界

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right)+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right)+\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

要素境界面通過熱流束　　体積発熱



- λ : 熱伝導率
- V_i : 要素体積
- S : 表面面積
- d_{ij} : 要素中心から表面までの距離
- q : 表面フラックス
- Q : 体積発熱
- T_{iB} : 境界温度

全体マトリクスの生成

要素iに関する釣り合い

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

隣接要素との熱伝導

温度固定境界

要素境界面通過熱流束

体積発熱

全体マトリクスの生成

要素iに関する釣り合い

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right)+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right)+\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

$$-\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_k+\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_i-\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_{iBe}+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_i=\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i$$

定数項：右辺へ移項

全体マトリクスの生成

要素iに関する釣り合い

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right)+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right)+\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

$$-\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_k+\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_i-\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_{iBe}+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_i=\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i$$

定数項：右辺へ移項

$$\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\right]T_i-\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_k\right]=\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_{iBe}$$

D（対角成分） **AMAT（非対角成分）** **BFORCE（右辺）**

「eps_fvm」処理：マトリクス生成（3/5）

体積発熱

```
      do i= 1, intNODE_tot
        do j= 1, index(i)-index(i-1)
          k= index(i-1) + j

          ic = IWKX(i,j)
          in1= CONN_NODE(2*ic-1)
          in2= CONN_NODE(2*ic  )

          if (in1.eq.i) then
            item(k)= in2
           else
            item(k)= in1
          endif
        enddo
      enddo
!C===

!C
!C +----------------------------+
!C | INTERIOR NODEs + BODY FLUX |
!C +----------------------------+
!C===
      do icel= 1, intNODE_tot
        BFORCE(icel)= BFORCE(icel) + BODY_NODE_FLUX(icel)
      enddo

      do i= 1, intNODE_tot
        do j= index(i-1)+1, index(i)
          icon= IWKX(i,j-index(i-1))
          AMAT(j)= -CONN_COEF(icon)
          D   (i)= D(i) + CONN_COEF(icon)
        enddo
      enddo
      deallocate (IWKX)
!C===
```

$$\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\right]T_i-\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_k\right]$$

$$=\sum_d S_{id}\dot{q}_{id}+V_i\dot{Q}_i+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_{iBe}$$

BFORCE（右辺）

BODY_NODE_FLUX

「eps_fvm」処理：マトリクス生成（3/5）

内部熱伝導

```
      do i= 1, intNODE_tot
        do j= 1, index(i)-index(i-1)
          k= index(i-1) + j

          ic = IWKX(i,j)
          in1= CONN_NODE(2*ic-1)
          in2= CONN_NODE(2*ic  )

          if (in1.eq.i) then
            item(k)= in2
           else
            item(k)= in1
          endif
        enddo
      enddo
!C===

!C
!C +----------------------------+
!C | INTERIOR NODEs + BODY FLUX |
!C +----------------------------+
!C===
      do icel= 1, intNODE_tot
        BFORCE(icel)= BFORCE(icel) + BODY_NODE_FLUX(icel)
      enddo

      do i= 1, intNODE_tot
        do j= index(i-1)+1, index(i)
          icon= IWKX(i,j-index(i-1))
          AMAT(j)= -CONN_COEF(icon)
          D   (i)= D(i) + CONN_COEF(icon)
        enddo
      enddo
      deallocate (IWKX)
!C===
```

D（対角成分） AMAT（非対角成分）

$$\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\right]T_i-\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_k\right]$$

CONN_COEF CONN_COEF

$$=\sum_d S_{id}\dot{q}_{id}+V_i\dot{Q}_i+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_{iBe}$$

「eps_fvm」処理：マトリクス生成（4/5）

ディリクレ境界条件（表面温度固定）

```
!C
!C +-----------+
!C | DIRICHLET |
!C +-----------+
!C===
      do i= 1, FIX_NODE_tot
        icel= FIX_NODE_ID(i)
        D     (icel)= D     (icel) + FIX_NODE_COEF(i)
        BFORCE(icel)= BFORCE(icel) + FIX_NODE_COEF(i)*FIX_NODE_VAL(i)
      enddo
!C===
```

D（対角成分）

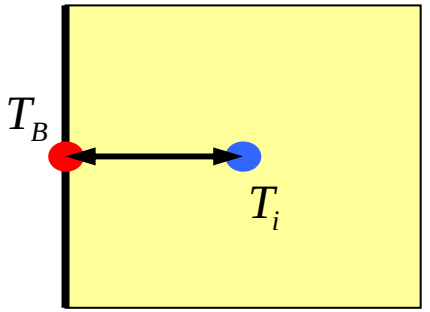
$$\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}} + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\right] T_i - \left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}} T_k\right]$$

FIX_NODE_COEF

$$= \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}} T_{iBe}$$

BFORCE（右辺）

FIX_NODE_COEF * FIX_NODE_VAL



「eps_fvm」処理：マトリクス生成（5/5）

ノイマン境界条件（表面熱流束）

```
!C
!C +---------------+
!C | SURFACE FLUX |
!C +---------------+
!C===
      do i= 1, SURF_NODE_tot
        icel= SURF_NODE_ID(i)
        BFORCE(icel)= BFORCE(icel) + SURF_NODE_FLUX(i)
      enddo
!C===
      return
      end
```

$$\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\right]T_i-\left[\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}T_k\right]$$

$$=\sum_d S_{id}\dot{q}_{id}+V_i\dot{Q}_i+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}T_{iBe}$$

BFORCE（右辺）

SURF_NODE_FLUX

- アプリケーションにおける処理:「eps_fvm」の復習
- 並列化
  - 局所分散データ
  - 共役勾配法
  - 可視化
- 課題P1

「eps_fvm」の並列化

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

どのようなことが言えるか?

- プログラムの変更がある部分
- プログラムの変更がない部分
- 通信が必要な部分，必要ない部分

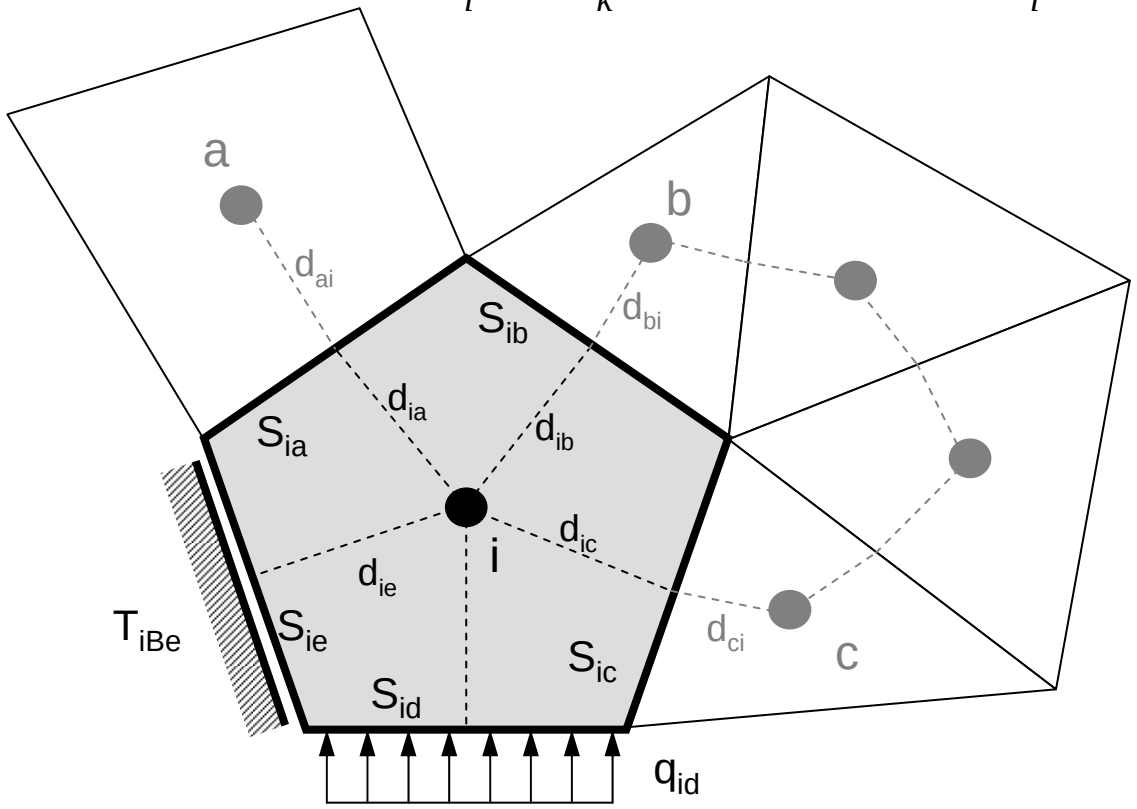
有限体積法による空間離散化

熱流束に関するつりあい式：ディリクレ境界条件考慮

隣接要素との熱伝導　　温度固定境界

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right)+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right)+\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

要素境界面通過熱流束　　体積発熱



- λ ：熱伝導率
- V_i ：要素体積
- S ：表面面積
- d_{ij} ：要素中心から表面までの距離
- q ：表面フラックス
- Q ：体積発熱
- T_{iB} ：境界温度

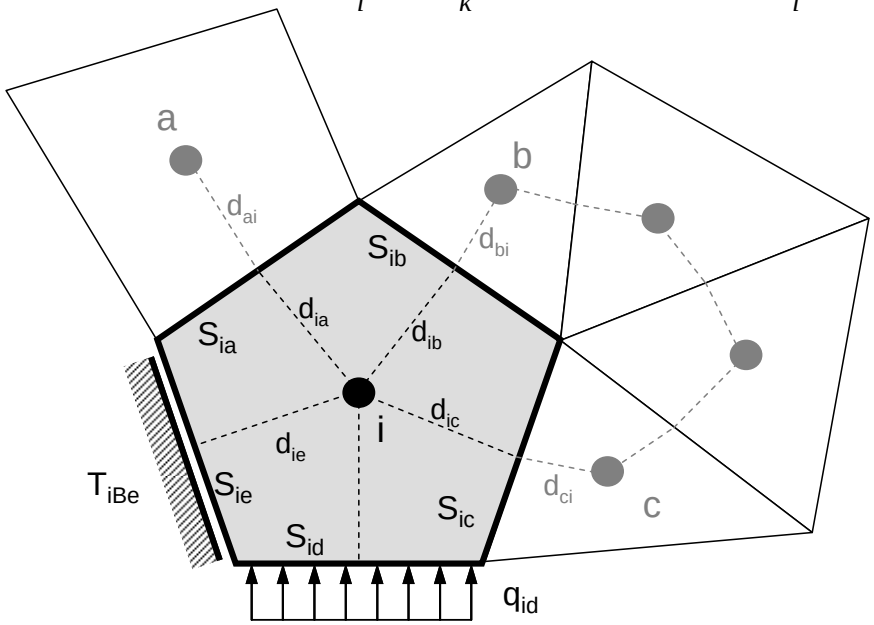
有限体積法（FVM）における処理の特徴

- 計算にあたっては，隣接した要素の情報のみが必要。
- 局所的な処理が可能。
- したがって並列計算には適した手法である。

隣接要素との熱伝導　　温度固定境界

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

要素境界面通過熱流束　　体積発熱



λ　：熱伝導率
V_i　：要素体積
S　：表面面積
d_{ij}　：要素中心から表面までの距離
q　：表面フラックス
Q　：体積発熱
T_{iB}　：境界温度

並列計算を実施する場合には?

21	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

並列計算を実施する場合には?

21	22	23	24	25
16	17	18	19	20
11	12	13	14	15
6	7	8	9	10
1	2	3	4	5

3領域に分割

#PE2

21	22	23
16	17	18
11	12	13

#PE1

24	25
19	20
14	15
9	10

#PE0

6	7	8		
1	2	3	4	5

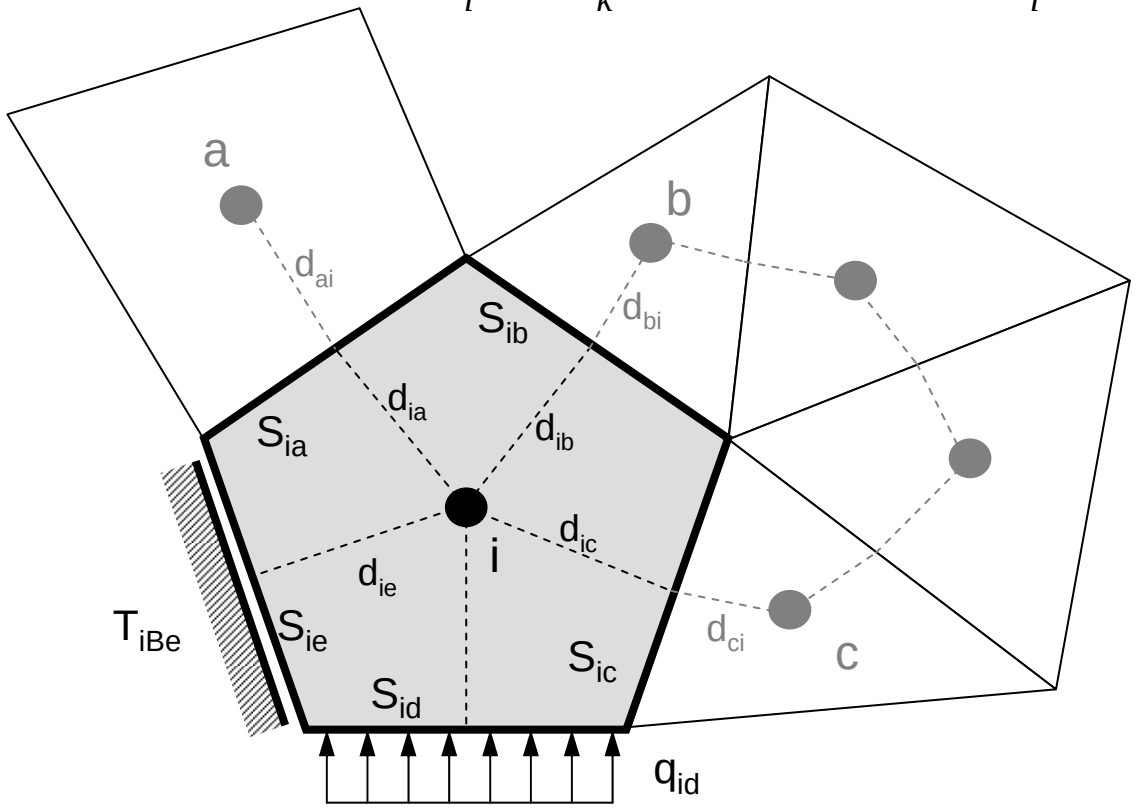
有限体積法：隣接メッシュの情報が必要

熱流束に関するつりあい式

隣接要素との熱伝導　　温度固定境界

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right)+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right)+\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

要素境界面通過熱流束　　体積発熱



- λ ：熱伝導率
- V_i ：要素体積
- S ：表面面積
- d_{ij} ：要素中心から表面までの距離
- q ：表面フラックス
- Q ：体積発熱
- T_{iB} ：境界温度

オーバーラップ領域のデータ必要

#PE2

21	22	23	24
16	17	18	19
11	12	13	14
6	7	8	

#PE1

23	24	25
18	19	20
13	14	15
8	9	10
	4	5

#PE0

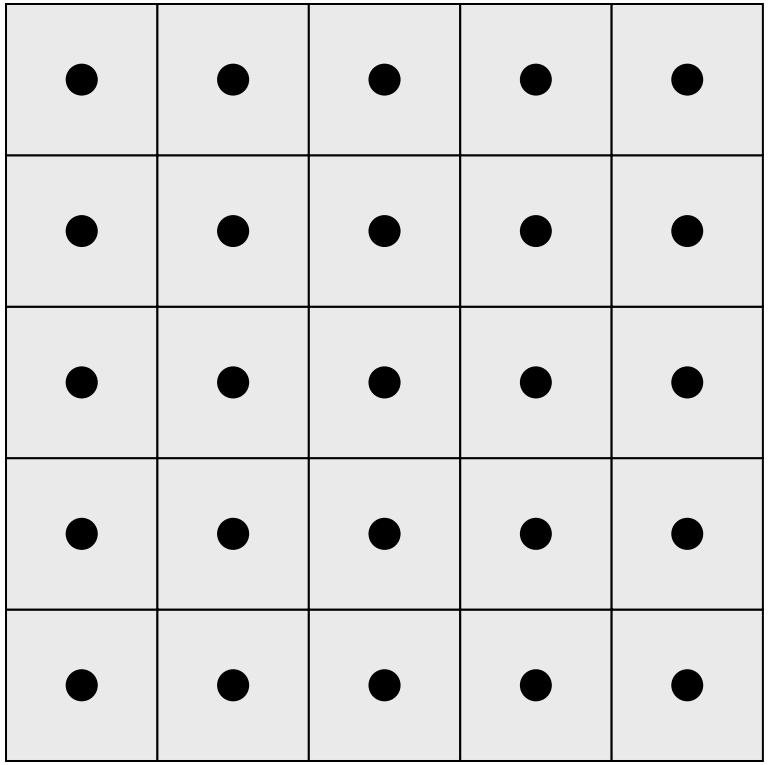
11	12	13		
6	7	8	9	10
1	2	3	4	5

有限体積法

各領域に必要な情報(1/4)

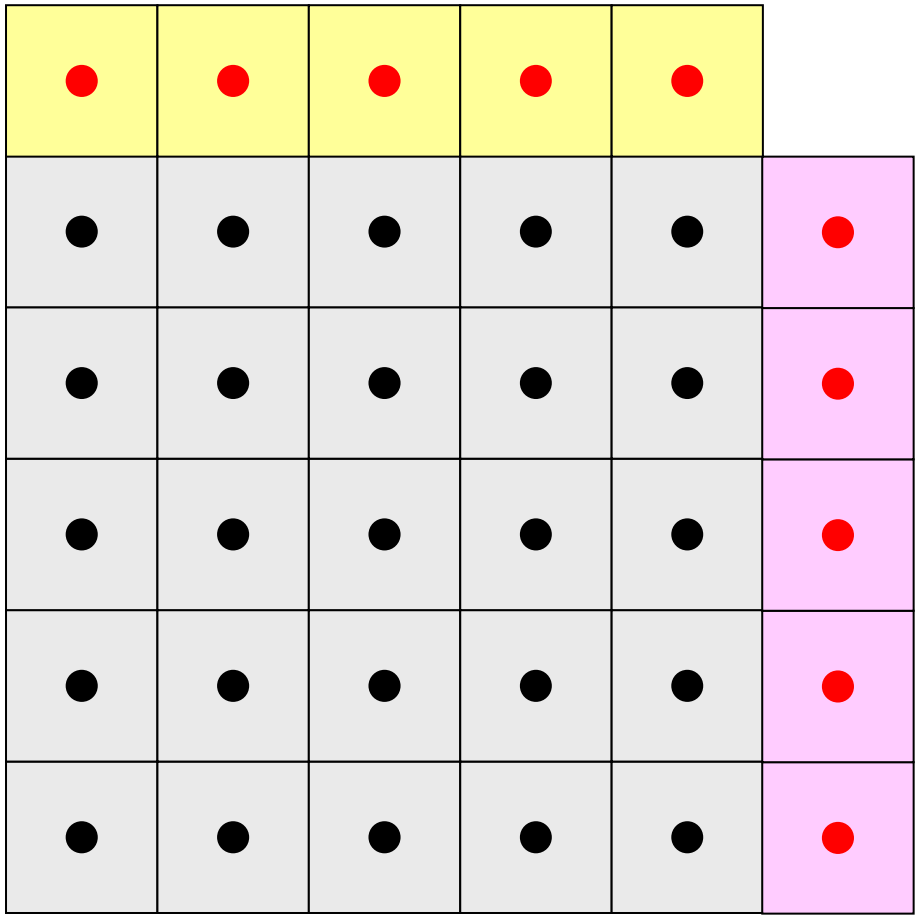
内点(Internal Points)

その領域にアサインされた要素



有限体積法

各領域に必要な情報(2/4)



内点(Internal Points)

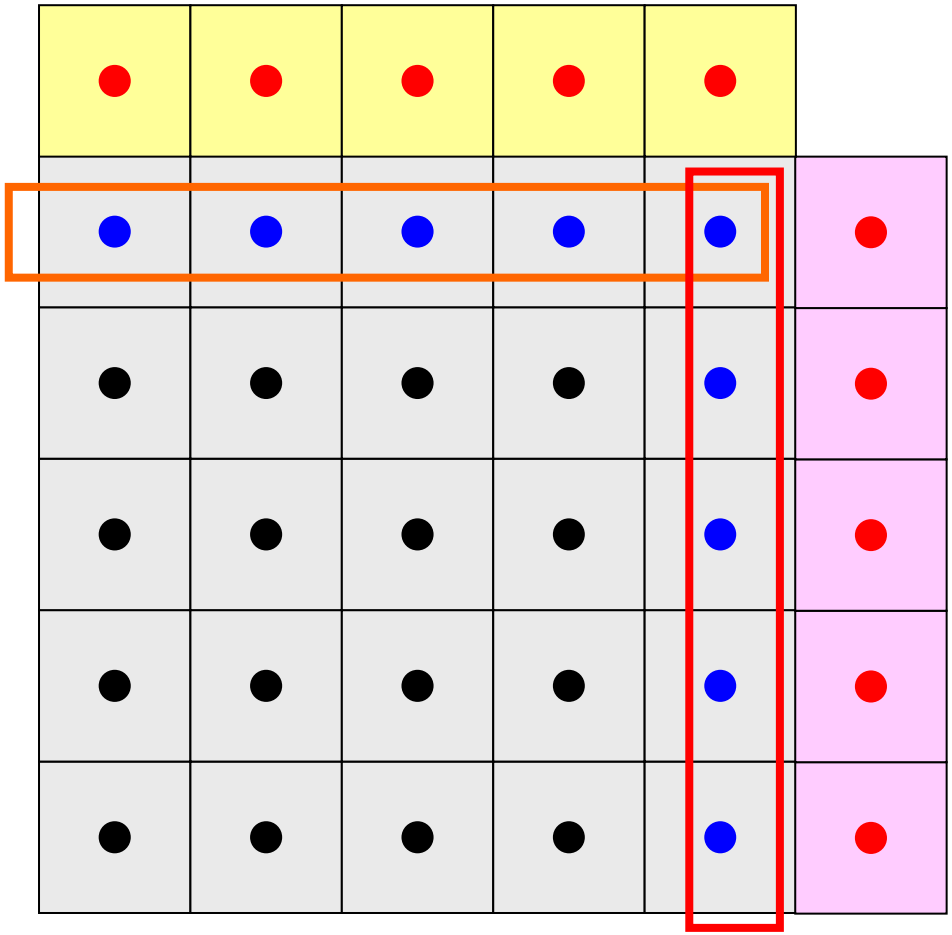
その領域にアサインされた要素

外点(External Points)

他の領域にアサインされた要素であるがその領域の計算を実施するのに必要な要素
(オーバーラップ領域の要素)

有限体積法

各領域に必要な情報(4/4)



内点(Internal Points)

その領域にアサインされた要素

外点(External Points)

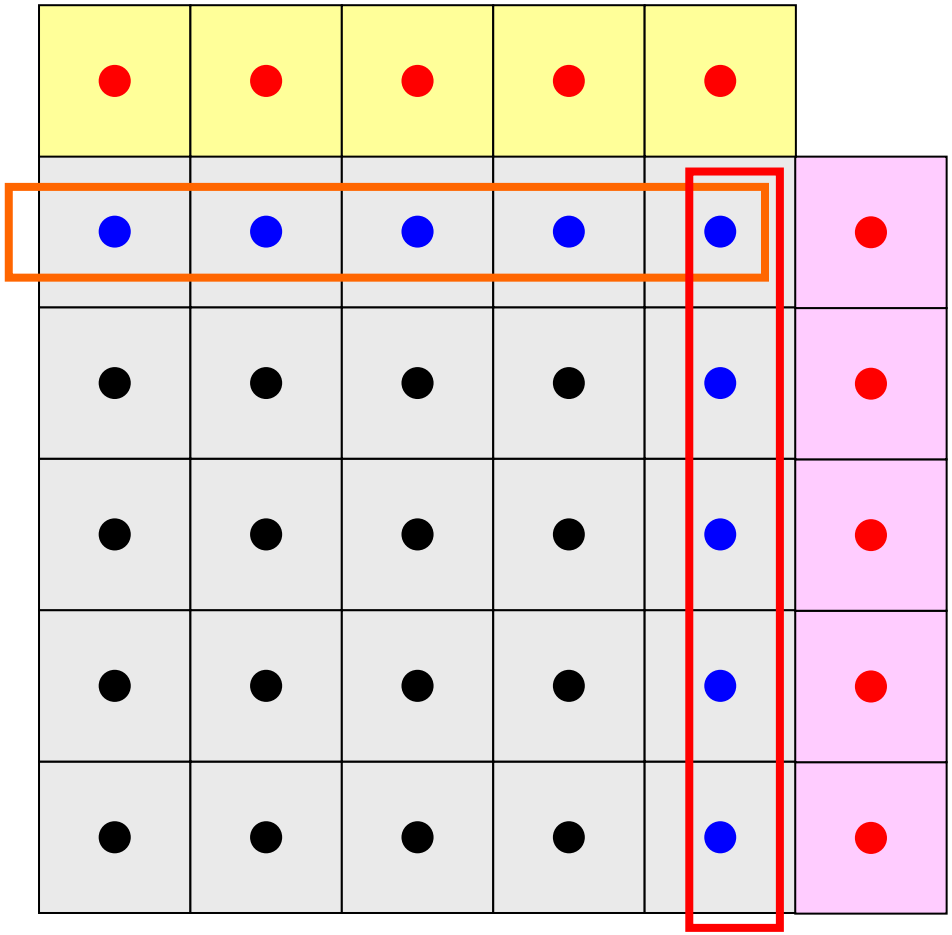
他の領域にアサインされた要素であるがその領域の計算を実施するのに必要な要素
(オーバーラップ領域の要素)

境界点(Boundary Points)

内点のうち，他の領域の外点となっている要素
他の領域の計算に使用される要素

有限体積法

各領域に必要な情報(4/4)



内点(Internal Points)

その領域にアサインされた要素

外点(External Points)

他の領域にアサインされた要素であるがその領域の計算を実施するのに必要な要素
(オーバーラップ領域の要素)

境界点(Boundary Points)

内点のうち，他の領域の外点となっている要素
他の領域の計算に使用される要素

領域間相互の関係

通信テーブル:外点，境界点の関係
隣接領域

各領域データ(局所データ)仕様

「一般化された通信テーブル」に対応

21	22	23	24	
13	14	15	16	20
9	10	11	12	19
5	6	7	8	18
1	2	3	4	17

- 内点，外点
  - 内点～外点となるように局所番号をつける
- 隣接領域情報
  - オーバーラップ要素を共有する領域
  - 隣接領域数，番号
- 外点情報
  - どの領域から，何個の，どの外点の情報を「受信：import」するか
- 境界点情報
  - 何個の，どの境界点の情報を，どの領域に「送信：export」するか

領域間通信
一般化された通信テーブル

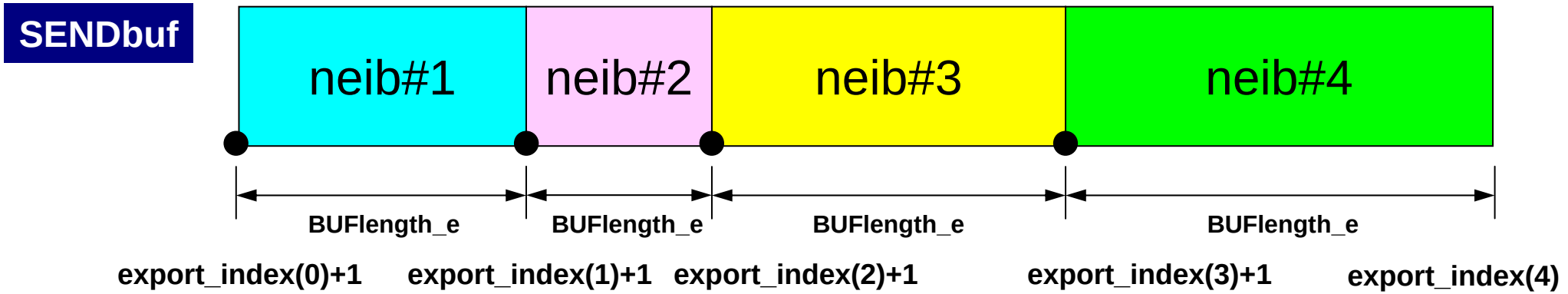
- 「通信」とは「外点」の情報を，その「外点」が本来属している領域から得ることである。

- 「通信テーブル」とは領域間の外点の関係の情報を記述したもの。
  - 「送信テーブル（export）」，「受信テーブル（import）」がある。

一般化された通信テーブル：送信

- 送信相手
  - NEIBPETOT，NEIB(neib)
- それぞれの送信相手に送るメッセージサイズ
  - export_index(neib), neib= 1, NEIBPETOT
- 「境界点」番号
  - export_item(k), k= 1, export_index(NEIBPETOT)
- それぞれの送信相手に送るメッセージ
  - SENDbuf(k), k= 1, export_index(NEIBPETOT)

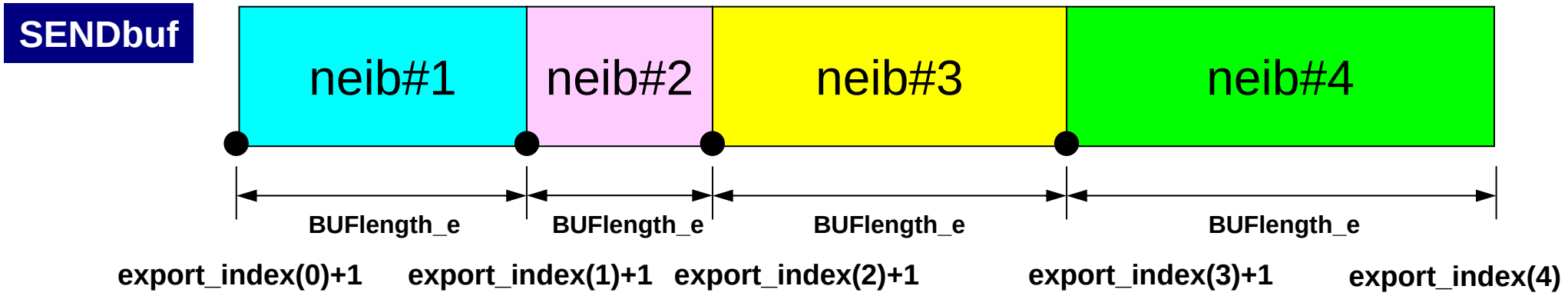
送信（MPI_Isend/Irecv/Waitall）




```
 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)
```

送信バッファへの代入
温度などの変数を直接送信，受信に使うのではなく，このようなバッファへ一回代入して計算することを勧める。

送信（MPI_Sendrecv）




```
 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_SENDRECV                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo
```

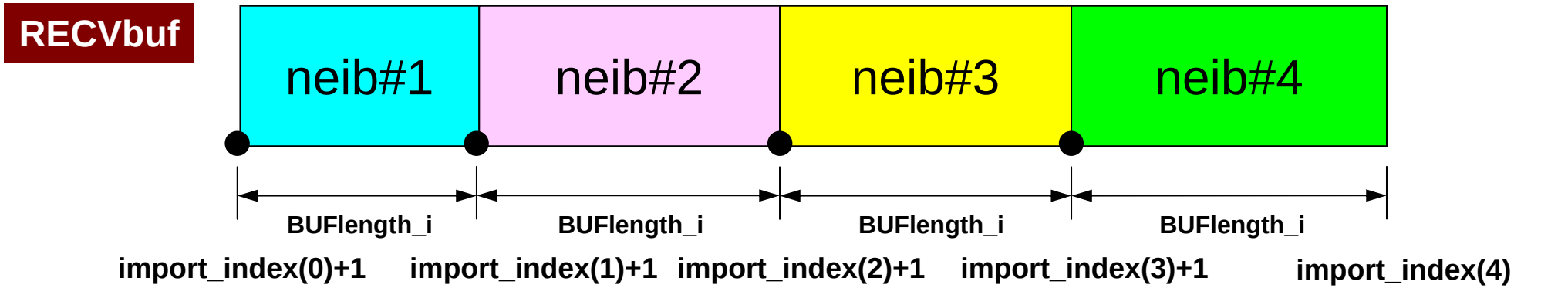
送信バッファへの代入

一般化された通信テーブル：受信

- 受信相手
  - NEIBPETOT，NEIB(neib)
- それぞれの受信相手から受け取るメッセージサイズ
  - import_index(neib)
- 「外点」番号
  - import_item(k), k= 1, import_index(NEIBPETOT)
- それぞれの受信相手から受け取るメッセージ
  - RECVbuf(k), k= 1, import_index(NEIBPETOT)

受信（MPI_Isend/Irecv/Waitall）

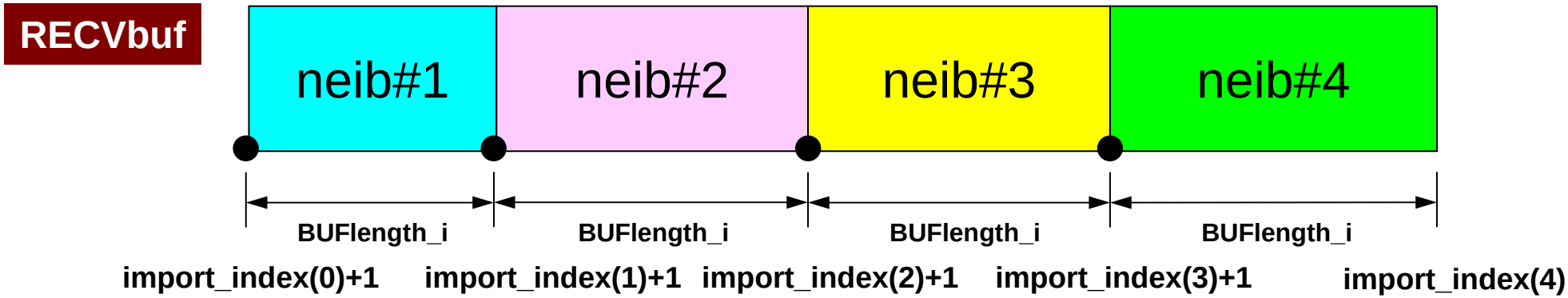
```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   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)                                  受信バッファから代入
     enddo
  enddo
```



受信（MPI_Sendrecv）

```
 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_SENDRECV                                                  &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo

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

受信バッファからの代入



行列ベクトル積における例（課題S3）

```
        do neib= 1, NEIBPETOT
          do k= export_index(neib-1)+1, export_index(neib)
            kk= export_item(k)
            SENDbuf(k)= W(kk,P)
          enddo
        enddo
!C
!C-- SEND & RECV.
        do neib= 1,  NEIBPETOT
          is= export_index(neib-1) + 1
          ir= import_index(neib-1) + 1
          len_s= export_index(neib) - export_index(neib-1)
          len_r= import_index(neib) - import_index(neib-1)
          call MPI_SENDRECV
     &         (SENDbuf(is), len_s, MPI_DOUBLE_PRECISION,NEIBPE(neib),0,&
     &          RECVbuf(ir), len_r, MPI_DOUBLE_PRECISION,NEIBPE(neib),0,&
     &          MPI_COMM_WORLD, stat1, ierr)
        enddo
!C-    update
        do neib= 1, NEIBPETOT
          do k= import_index(neib-1)+1, import_index(neib)
            kk= import_item(k)
            W(kk,P)= RECVbuf(k)
          enddo
        enddo

        do i= 1, N
          W(i,Q) = DIAG(i)*W(i,P)
          do j= INDEX(i-1)+1, INDEX(i)
            W(i,Q) = W(i,Q) + AMAT(j)*W(ITEM(j),P)
          enddo
        enddo
```

境界値の送信
バッファへのコピー

送受信

受信バッファの内容
を外点の値として
代入

FVMにおける領域間通信

- 差分法等，係数が疎行列のアプリケーションについては領域間通信はこの方法で実施可能。

- 適切なデータ構造が定められれば，処理は非常に簡単。
  - 送信バッファに「境界点」の値を代入
  - 送信，受信
  - 受信バッファの値を「外点」の値として更新

「eps_fvm」の並列化

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

どのようなことが言えるか?

・プログラムの変更がある部分
・プログラムの変更がない部分
・通信が必要な部分，必要ない部分

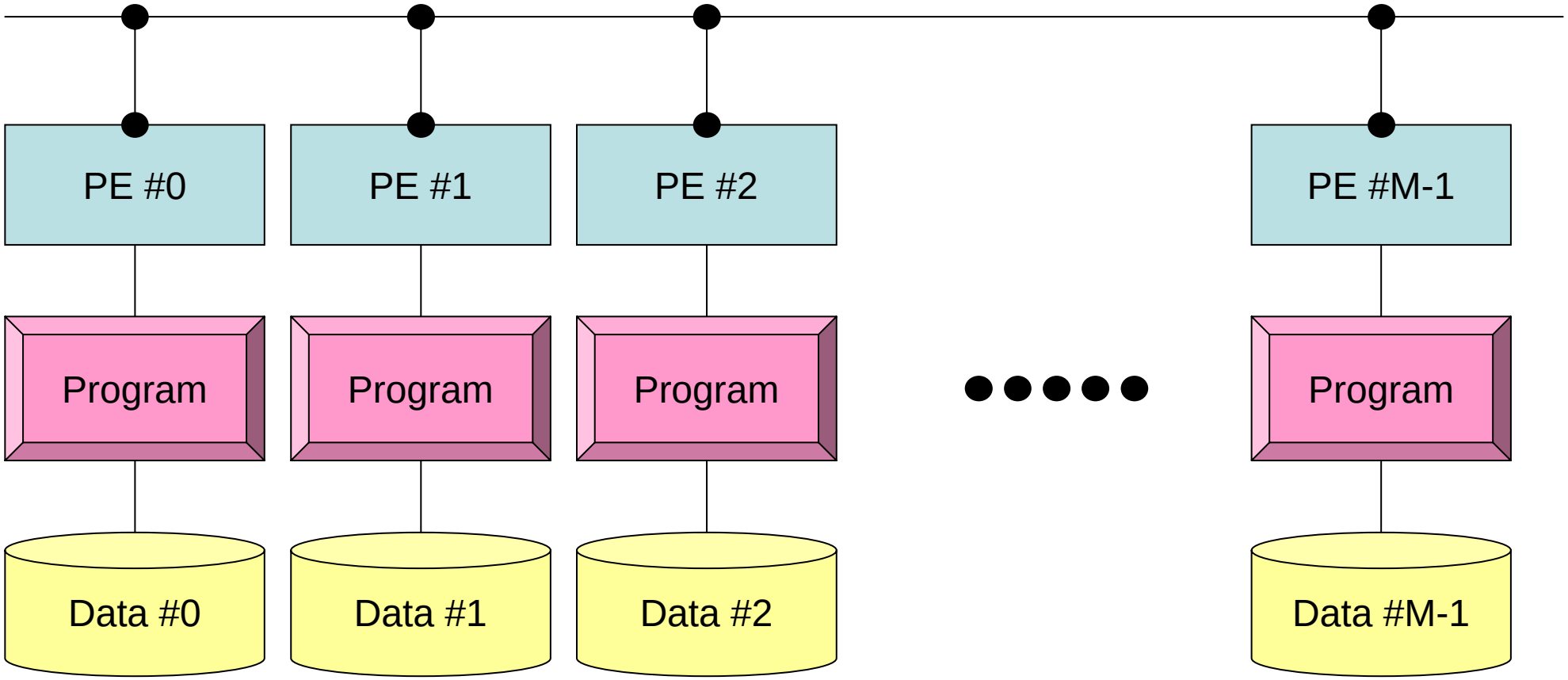
「eps_fvm」並列化に向けて

- hpcmw_eps_fvm_input_grid
  - 並列分散メッシュデータ
  - 並列分散通信データ
- poi_gen
  - この部分は（多分）変更不要
- hpcmw_eps_fvm_solver
  - 課題S3と同様の方法
- output_ucd
  - 並列分散処理について考慮する必要あり

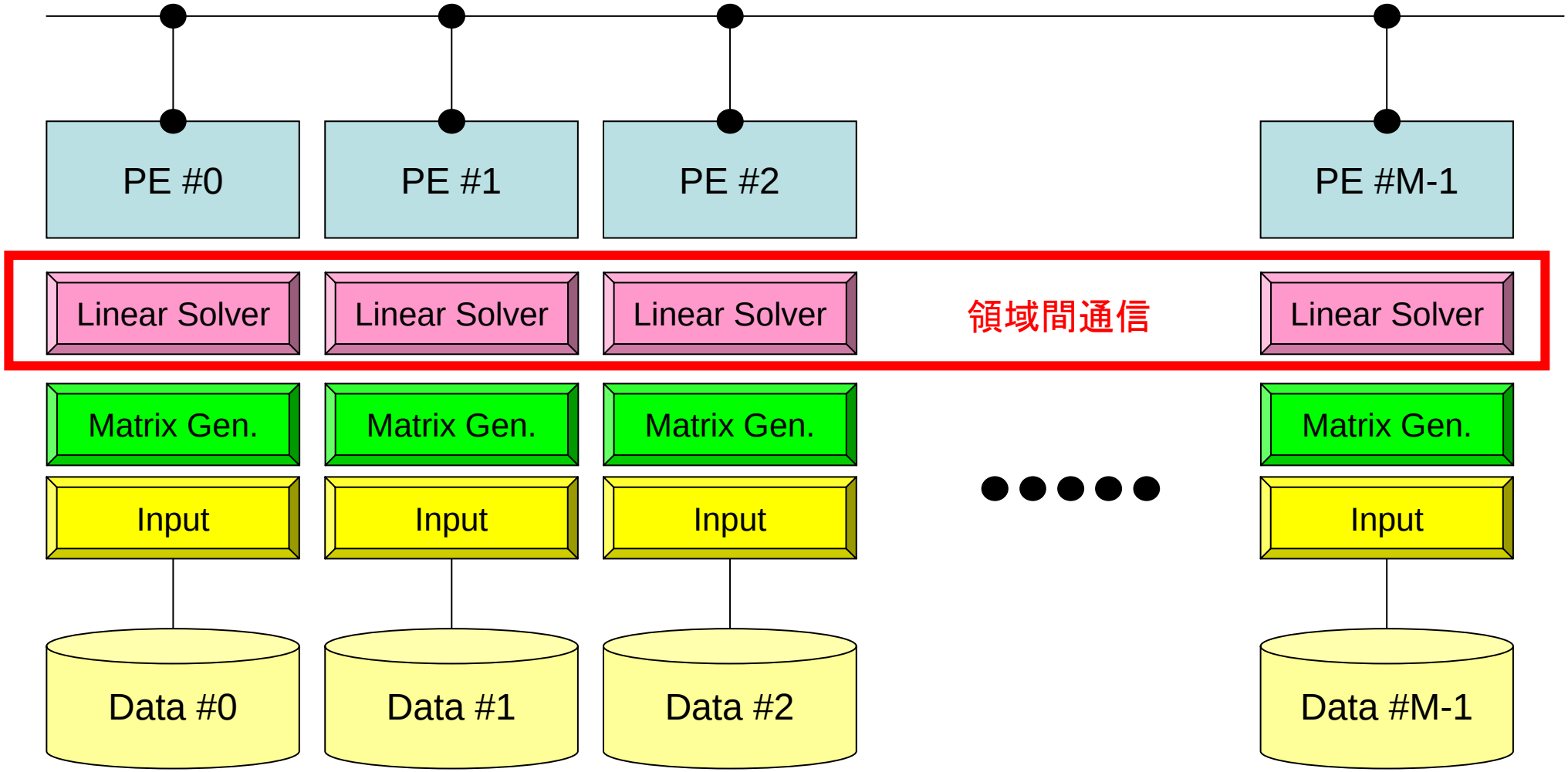
「eps_fvm」の「並列化」

- メッシュデータ入力
  - 通信用データを入力する必要があるが，基本的に，serialのままでよい。プロセッサ間の通信は不要。プロセッサごとに異なったデータを読む必要がある。
- 係数マトリクス生成部
  - オーバーラップ領域（外点）の，幾何学的データ，物性値があるので，これもserialのプログラムのまま，プロセッサ間の通信は不要。
- 線形ソルバー
  - 行列ベクトル積，内積の部分で通信が生じる。
- 今回は線形計算であるので，問題は無いが，非線形計算の場合，残差の最大最小値等についてはグローバルな最大最小値が必要となる。

再びSPMD


PE #0
PE #1
PE #2
PE #M-1
Program
Program
Program
Program
Data #0
Data #1
Data #2
Data #M-1

「eps_fvm」ではこうなる



ファイルのコピー，インストール

```
$> cd <$S07>

FORTRAN
$> cp /home/nakajima/class/2007summer/F/p1c-f.tar .
$> tar xvf p1c-f.tar
C
$> cp /home/nakajima/class/2007summer/C/p1c-c.tar .
$> tar xvf p1c-c.tar

$> cd <$P1>
$> ls

   CS10/  local/  mesh/  run/  serial/  src/
```

プログラム類

- ソースコード
  - **<$P1>/src**
  - ここで「**make**」すると「**<$P1>/run**」 に「**sol**」という実行形式ができる。以下これを使用する。

```
$> cd <$P1>/src
$> make
$> ls -l ../run/sol
     sol
```

- チュートリアル

http://www-solid.eps.s.u-tokyo.ac.jp/~nakajima/tutorial/parallel_lib_tutorial/

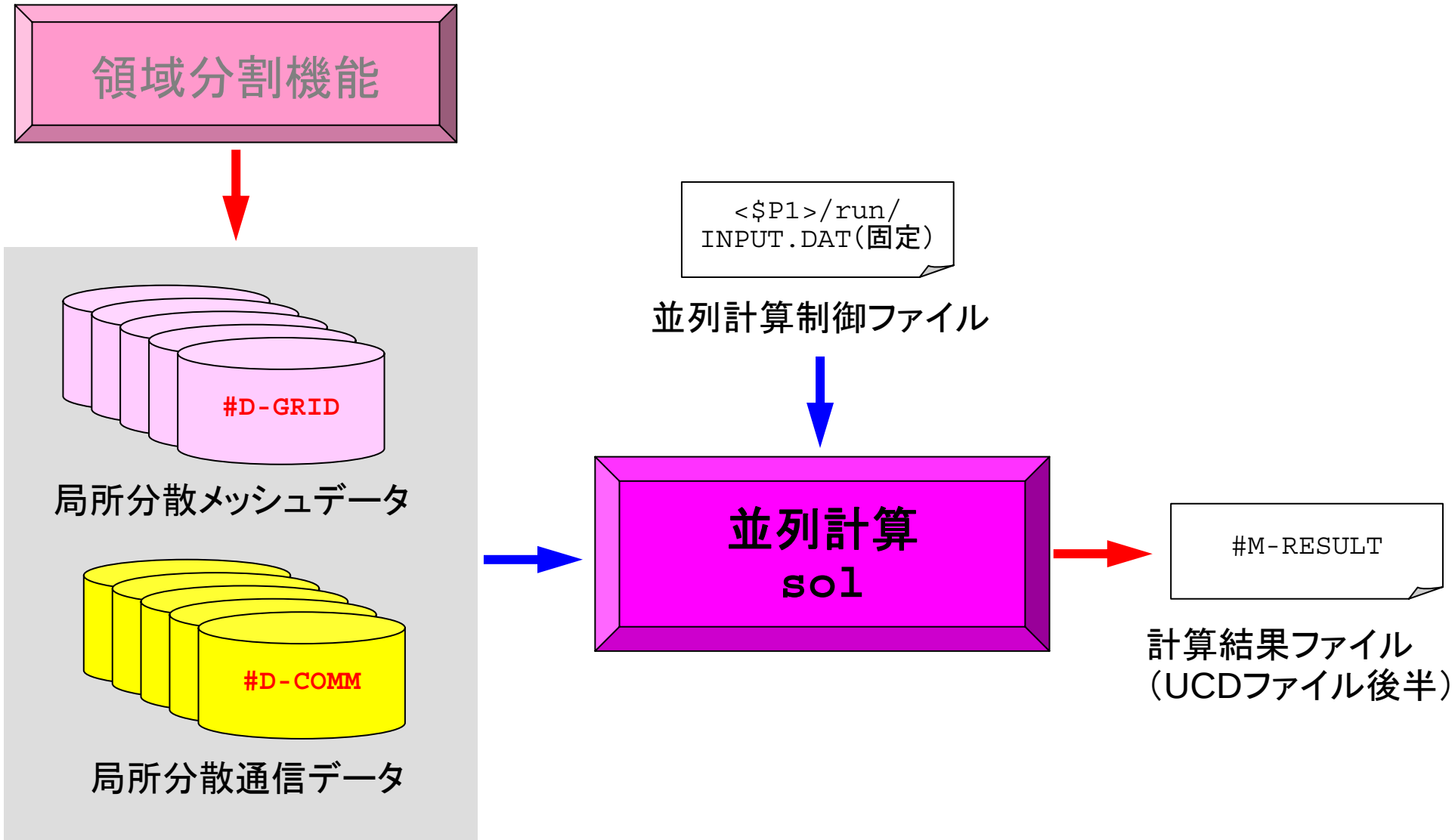
http://www-solid.eps.s.u-tokyo.ac.jp/~nakajima/tutorial/parallel_lib_tutorial.tar

ファイル準備（CS10）

```
$> cd <$P1>/CS10
$> cat fvmmg.ctrl
   32  32  32
$> cat fvmpart.ctrl
   !INITIAL FILE
   fvm_entire_mesh.dat

   !METHOD
   RCB
   X,Y,Z

   !REGION NUMBER
   8

   !MESH FILE
   mesh.rcb

   !COMMUNICATION FILE
   comm.rcb

   !UCD
   32-32-32-rcb-8.inp
$> eps_fvm_mg
$> eps_fvm_part
```

```
$> ls -l mesh.rcb.*
   mesh.rcb.0 … mesh.rcb.7
$> ls -l comm.rcb.*
   comm.rcb.0 … comm.rcb.7
```

並列シミュレーションにおけるI/O



並列計算制御ファイル

- **INPUT.DAT**（名称固定）
- 実行形式「**sol**」と同じディレクトリになければならない（この場合は**<$P1>/run**）。
- 全ての項目は省略不可。

```
../CS10/mesh.rcb      局所分散メッシュファイルのヘッダ名
../CS10/comm.rcb      局所分散通信ファイルのヘッダ名
../CS10/result        可視化用出力ファイル名（後述）
1                     可視化用出力の有無（=1のとき出力）
```

計算実行

```
$> cd run
$> cat INPUT.DAT
  ../CS10/mesh.rcb
  ../CS10/comm.rcb
  ../CS10/result
  1
```

「go.sh」を下記のように書き換える

```
  #!/bin/sh
  #PBS -N test
  #PBS -o test.lst
  #PBS -e err
  #PBS -l nodes=4:ppn=2
  #PBS -l walltime=10:00
  cd $PBS_O_WORKDIR
  NPROCS=`wc -l < $PBS_NODEFILE`
  mpirun -v -machinefile $PBS_NODEFILE -np $NPROCS sol

$> qsub go.sh
$> ls -l ../CS10/result
  result
```

「eps_fvm」の並列化

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

どのようなことが言えるか?

- プログラムの変更がある部分
- プログラムの変更がない部分
- 通信が必要な部分，必要ない部分

例えば，poi_genは全く変わらない

```
$> cd <$P1>/src
$> diff poi_gen.f ../serial/poi_gen.f
```

データ入力部分(1/5)

```
      subroutine hpcmw_eps_fvm_input_grid
      use hpcmw_eps_fvm_all
      implicit REAL*8 (A-H,O-Z)

      character(len=HPCMW_NAME_LEN) :: member
      character(len=80            ) :: LINE

!C
!C +-------+
!C | FILES |
!C +-------+
!C===
      open (11, file='INPUT.DAT', status='unknown')
        read (11,'(a127)') HEADERgrid
        read (11,'(a127)') HEADERcomm
        read (11,`(a127)') AVSfile
        read (11,*)        PVISFLAG
      close (11)

      allocate (hpcmw_eps_fvm_files(4))

      member= 'gridfile'
      call hpcmw_eps_fvm_define_file_name (member, HEADERgrid)

      member= 'commfile'
      call hpcmw_eps_fvm_define_file_name (member, HEADERcomm)
!C===
```

制御ファイル「INPUT.DAT」

ヘッダーを与えて分散ファイル名を生成する。

分散ファイル名生成(1/2)

```
!C
!C***
!C*** HPCMW_EPS_FVM_DEFINE_FILE_NAME
!C***
!C
      subroutine HPCMW_EPS_FVM_DEFINE_FILE_NAME (member, HEADERo)

      character (len=HPCMW_HEADER_LEN) ::  HEADERo, FILENAME
      character (len=HPCMW_NAME_LEN)     ::  member
      character (len=HPCMW_NAME_LEN)     ::  HEADER
      character (len= 1) ::  SUBindex1
      character (len= 2) ::  SUBindex2
      character (len= 3) ::  SUBindex3
      character (len= 4) ::  SUBindex4
      character (len= 5) ::  SUBindex5
      character (len= 6) ::  SUBindex6
      integer:: LENGTH, ID

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

分散ファイル名生成(2/2)

```
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
 else if (my_rank.le.9999) then
  ID= 4
  write(SUBindex5 ,'(i4.4)') my_rank
 else if (my_rank.le.99999) then
  ID= 5
  write(SUBindex6 ,'(i5.5)') my_rank
 else if (my_rank.le.999999) then
  ID= 6
  write(SUBindex4 ,'(i6.6)') 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
if (ID.eq.4) filename= HEADER(1:LENGTH)//'.'//SUBindex4
if (ID.eq.5) filename= HEADER(1:LENGTH)//'.'//SUBindex5
if (ID.eq.6) filename= HEADER(1:LENGTH)//'.'//SUBindex6

if (member.eq. 'gridfile') hpcmw_eps_fvm_files(1)= filename
if (member.eq. 'commfile') hpcmw_eps_fvm_files(4)= filename

end subroutine hpcmw_eps_fvm_define_file_name
```

領域数1,000,000まで対応可能

データ入力部分(2/5)

ここらへんはserialと同じ
各PEでファイル名を書き出して見よ

```
!C
!C +------------+
!C | MESH INPUT |
!C +------------+
!C===
      IUNIT= 11
      open (IUNIT,file= hpcmw_eps_fvm_files(1), status='unknown')

!C
!C-- NODE
      read (IUNIT, '(10i10)') NODE_tot

      allocate (NODE_VOL(NODE_tot), NODE_COND(NODE_tot),                &
     &          NODE_XYZ(3*NODE_tot))

      do i= 1, NODE_tot
        read (IUNIT,'(i10,5e16.6)') ii, NODE_VOL(i), NODE_COND(i),      &
     &                              (NODE_XYZ(3*i-3+k), k=1, 3)
      enddo

!C
!C-- CONNECTION
      read (IUNIT,'(10i10)') CONN_tot
      allocate (CONN_NODE(2*CONN_tot), CONN_COEF(CONN_tot))
      do i= 1, CONN_tot
        read (IUNIT,'( 2i10, 3e16.6)') (CONN_NODE(2*i-2+k), k= 1, 2),   &
     &                                  AREA, D1, D2
        in1= CONN_NODE(2*i-1)
        in2= CONN_NODE(2*i  )
        C1 = NODE_COND(in1)
        C2 = NODE_COND(in2)
        CONN_COEF(i)= AREA / ( D1/C1 + D2/C2 )
      enddo
```

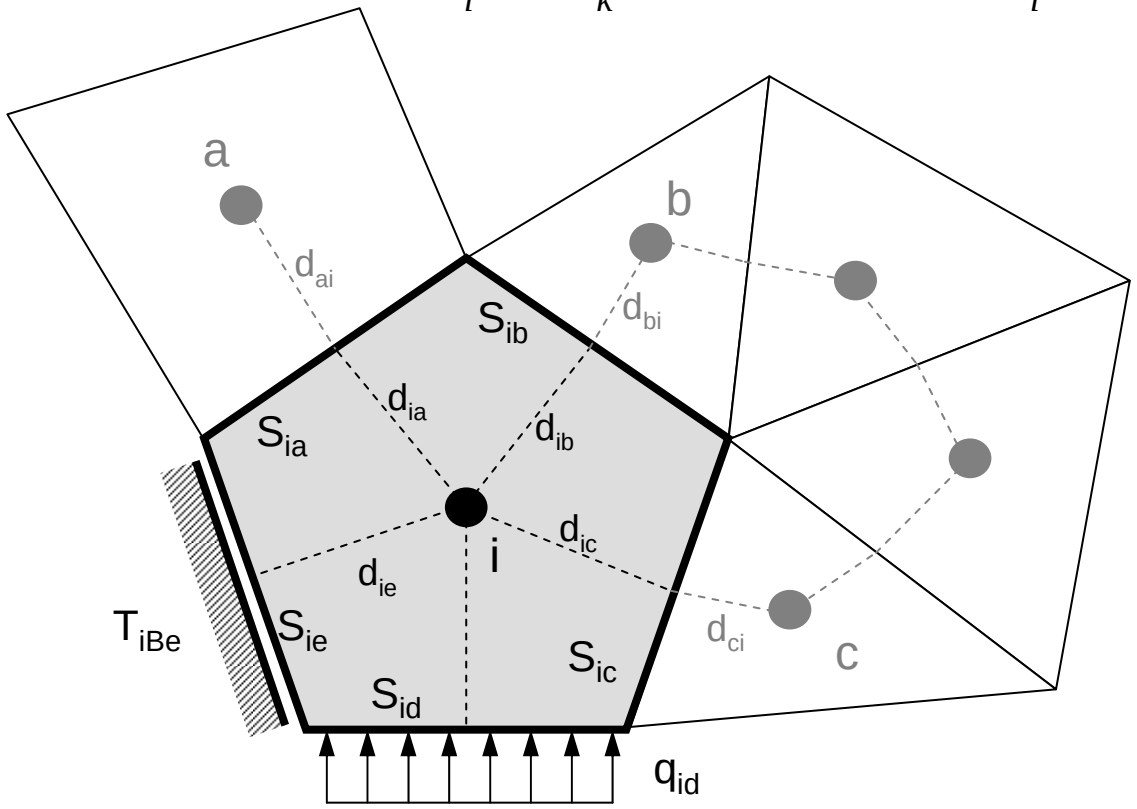
有限体積法：隣接メッシュの情報が必要

熱流束に関するつりあい式

隣接要素との熱伝導　　温度固定境界

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right)+\sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right)+\sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

要素境界面通過熱流束　　体積発熱



- λ ：熱伝導率
- V_i ：要素体積
- S ：表面面積
- d_{ij} ：要素中心から表面までの距離
- q ：表面フラックス
- Q ：体積発熱
- T_{iB} ：境界温度

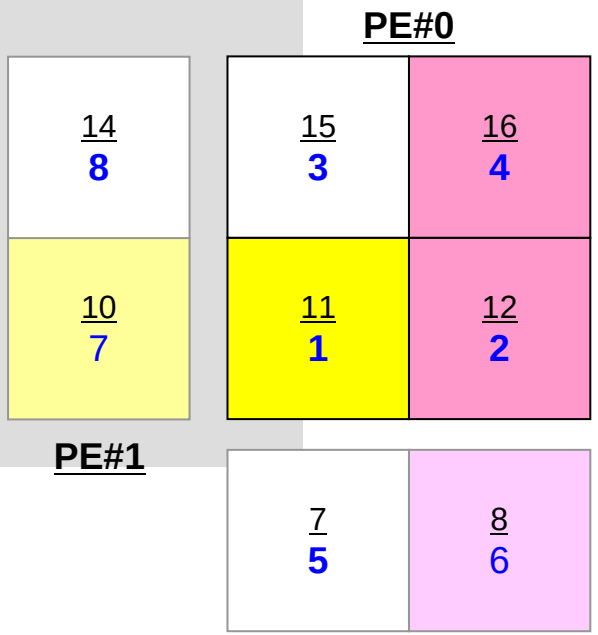
データ入力部分(3/5)

```
!C
!C-- DIRICHLET
      read (IUNIT,'(10i10)') FIX_NODE_tot
      allocate (FIX_NODE_ID(FIX_NODE_tot), FIX_NODE_COEF(FIX_NODE_tot))
      allocate (FIX_NODE_VAL(FIX_NODE_tot))

      do i= 1, FIX_NODE_tot
        read (IUNIT, '(i10, 3e16.6)')                                    &
     &        FIX_NODE_ID(i), AREA, DIST, FIX_NODE_VAL(i)
        icel= FIX_NODE_ID(i)
        COND= NODE_COND(icel)
        FIX_NODE_COEF(i)= AREA / (DIST/COND)
      enddo
!C
!C-- NEUMANN
      read (IUNIT,'(10i10)') SURF_NODE_tot
      allocate                                                           &
     &  (SURF_NODE_ID  (SURF_NODE_tot), SURF_NODE_FLUX(SURF_NODE_tot))

      do i= 1, SURF_NODE_tot
        read (IUNIT, '(i10, 3e16.6)') SURF_NODE_ID(i), AREA, FLUX
        SURF_NODE_FLUX(i)= AREA*FLUX
      enddo
!C
!C-- BODY FLUX
      read (IUNIT,'(10i10)') BODY_NODE_tot

      allocate (BODY_NODE_FLUX(NODE_tot))
      do i= 1, BODY_NODE_tot
        read (IUNIT, '(i10, 3e16.6)') icel, FLUX
        BODY_NODE_FLUX(icel)= FLUX * NODE_VOL(icel)
      enddo
      close (IUNIT)
```

局所分散メッシュファイル（mesh.0）

要素 / コネクティビティ / 境界条件

```
8
1    1.000000E+00    1.000000E+00    2.500000E+00    2.500000E+00    5.000000E-01
2    1.000000E+00    1.000000E+00    3.500000E+00    2.500000E+00    5.000000E-01
3    1.000000E+00    1.000000E+00    2.500000E+00    3.500000E+00    5.000000E-01
4    1.000000E+00    1.000000E+00    3.500000E+00    3.500000E+00    5.000000E-01
5    1.000000E+00    1.000000E+00    2.500000E+00    1.500000E+00    5.000000E-01
6    1.000000E+00    1.000000E+00    3.500000E+00    1.500000E+00    5.000000E-01
7    1.000000E+00    1.000000E+00    1.500000E+00    2.500000E+00    5.000000E-01
8    1.000000E+00    1.000000E+00    1.500000E+00    3.500000E+00    5.000000E-01
8
5          1    1.000000E+00    5.000000E-01    5.000000E-01
6          2    1.000000E+00    5.000000E-01    5.000000E-01
7          1    1.000000E+00    5.000000E-01    5.000000E-01
1          2    1.000000E+00    5.000000E-01    5.000000E-01
1          3    1.000000E+00    5.000000E-01    5.000000E-01
2          4    1.000000E+00    5.000000E-01    5.000000E-01
8          3    1.000000E+00    5.000000E-01    5.000000E-01
3          4    1.000000E+00    5.000000E-01    5.000000E-01
2
2    1.000000E+00    5.000000E-01    0.000000E+00
4    1.000000E+00    5.000000E-01    0.000000E+00
0
1
1    1.000000E+00
```



基本的に初期全体メッシュファイルと同じ

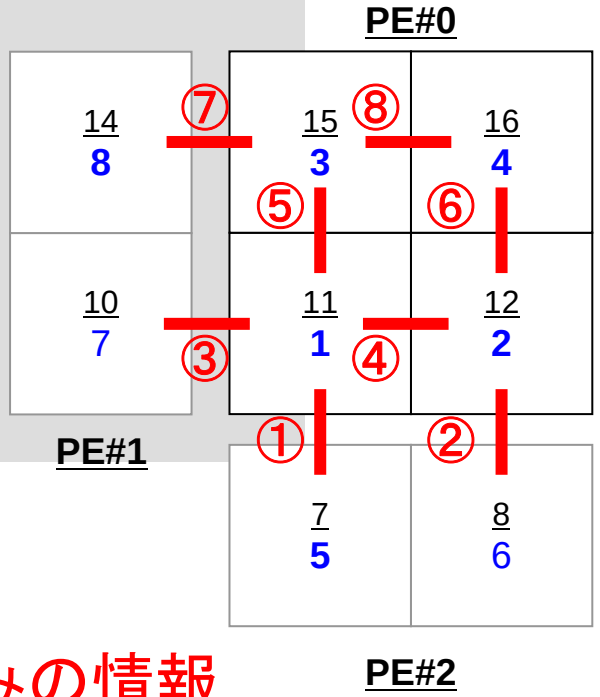
局所要素番号による記述

境界条件（ディリクレ，ノイマン，体積発熱）:「内点」のみの情報

局所分散メッシュファイル（mesh.0）

要素 / コネクティビティ / 境界条件

```
8
1    1.000000E+00    1.000000E+00    2.500000E+00    2.500000E+00    5.000000E-01
2    1.000000E+00    1.000000E+00    3.500000E+00    2.500000E+00    5.000000E-01
3    1.000000E+00    1.000000E+00    2.500000E+00    3.500000E+00    5.000000E-01
4    1.000000E+00    1.000000E+00    3.500000E+00    3.500000E+00    5.000000E-01
5    1.000000E+00    1.000000E+00    2.500000E+00    1.500000E+00    5.000000E-01
6    1.000000E+00    1.000000E+00    3.500000E+00    1.500000E+00    5.000000E-01
7    1.000000E+00    1.000000E+00    1.500000E+00    2.500000E+00    5.000000E-01
8    1.000000E+00    1.000000E+00    1.500000E+00    3.500000E+00    5.000000E-01
8
5          1    1.000000E+00    5.000000E-01    5.000000E-01  ①
6          2    1.000000E+00    5.000000E-01    5.000000E-01  ②
7          1    1.000000E+00    5.000000E-01    5.000000E-01  ③
1          2    1.000000E+00    5.000000E-01    5.000000E-01  ④
1          3    1.000000E+00    5.000000E-01    5.000000E-01  ⑤
2          4    1.000000E+00    5.000000E-01    5.000000E-01  ⑥
8          3    1.000000E+00    5.000000E-01    5.000000E-01  ⑦
3          4    1.000000E+00    5.000000E-01    5.000000E-01  ⑧
2
2    1.000000E+00    5.000000E-01    0.000000E+00
4    1.000000E+00    5.000000E-01    0.000000E+00
0
1
1    1.000000E+00
```



基本的に初期全体メッシュファイルと同じ
局所要素番号による記述
コネクティビティ:「内点～内点」,「内点～外点」のみの情報

データ入力部分(4/5)

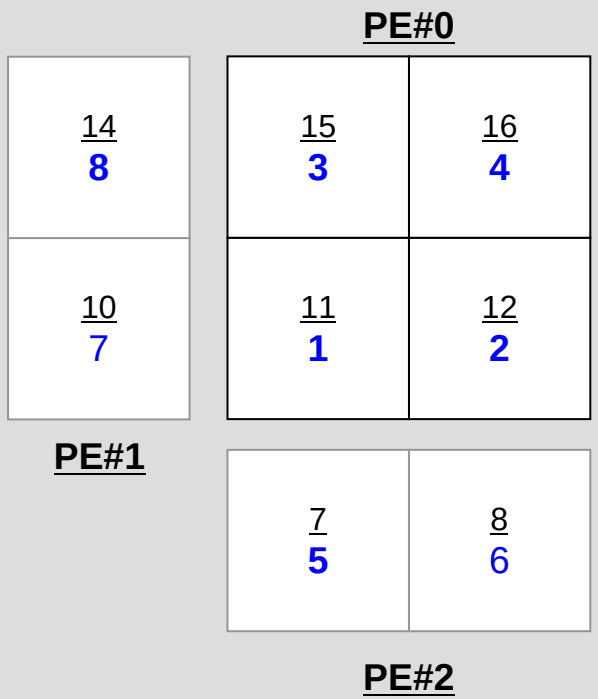
分散通信データ

```
!C
!C +-------------+
!C | COMM INPUT |
!C +-------------+
!C===
      IUNIT= 12
      open (IUNIT,file= hpcmw_eps_fvm_files(4), status='unknown')

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') n_neighbor_pe

      allocate (neighbor_pe(n_neighbor_pe))
      allocate (import_index(0:n_neighbor_pe))
      allocate (export_index(0:n_neighbor_pe))

      import_index= 0
      export_index= 0

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (neighbor_pe(k), k= 1, n_neighbor_pe)


      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (import_index(k), k= 1, n_neighbor_pe)
      nn= import_index(n_neighbor_pe)
      allocate (import_item(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (import_item(k), k= 1, nn)
```

局所分散通信ファイル（comm.0）

隣接領域

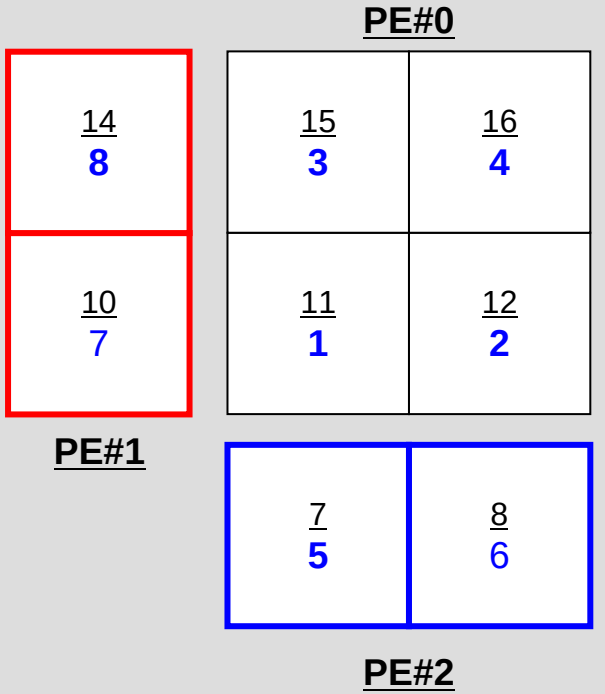
```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```



局所分散通信ファイル（comm.0）

受信テーブル，外点情報

```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```



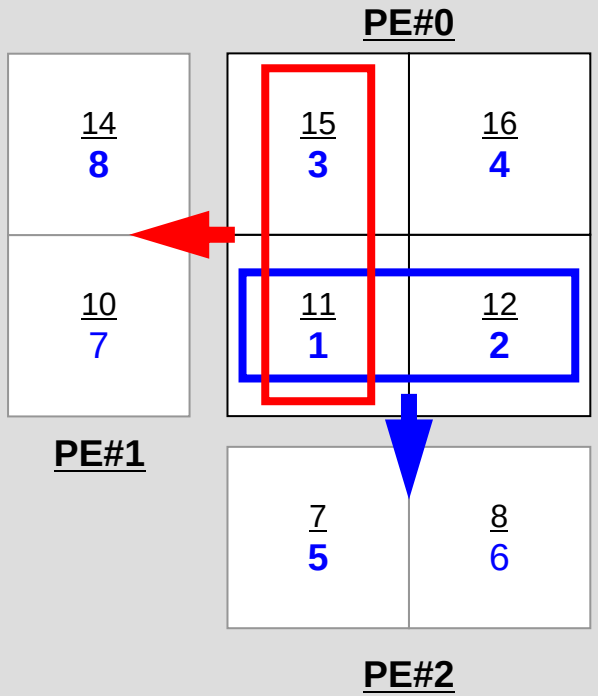
データ入力部分(5/5)

```
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_index(k), k= 1, n_neighbor_pe)
      nn= export_index(n_neighbor_pe)
      allocate (export_item(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_item(k), k= 1, nn)

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') intNODE_tot

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') nn

      allocate (NODE_GLOBAL(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (NODE_GLOBAL(k), k= 1, nn)

      close (IUNIT)
!C===

      end subroutine hpcmw_eps_fvm_input_grid
```

局所分散通信ファイル（comm.0）

送信テーブル，境界点情報

```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```



データ入力部分(5/5)

```
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_index(k), k= 1, n_neighbor_pe)
      nn= export_index(n_neighbor_pe)
      allocate (export_item(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_item(k), k= 1, nn)

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') intNODE_tot

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') nn

      allocate (NODE_GLOBAL(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (NODE_GLOBAL(k), k= 1, nn)

      close (IUNIT)
!C===

      end subroutine hpcmw_eps_fvm_input_grid
```

内点数

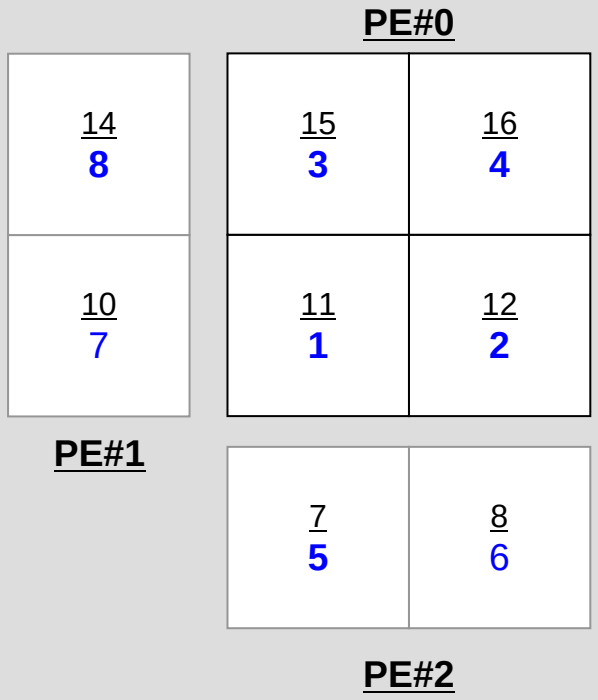
内点＋外点数(NODE_tot)

局所分散通信ファイル(comm.0)

内点数，総要素数(内点＋外点)，全体要素番号

```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```

全体要素番号(局所番号順)



- アプリケーションにおける処理:「eps_fvm」の復習
- 並列化
  - 局所分散データ
  - **共役勾配法**
  - 可視化
- 課題P1

「eps_fvm」の並列化

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

どのようなことが言えるか?

- プログラムの変更がある部分
- プログラムの変更がない部分
- 通信が必要な部分，必要ない部分

例えば，poi_genは全く変わらない

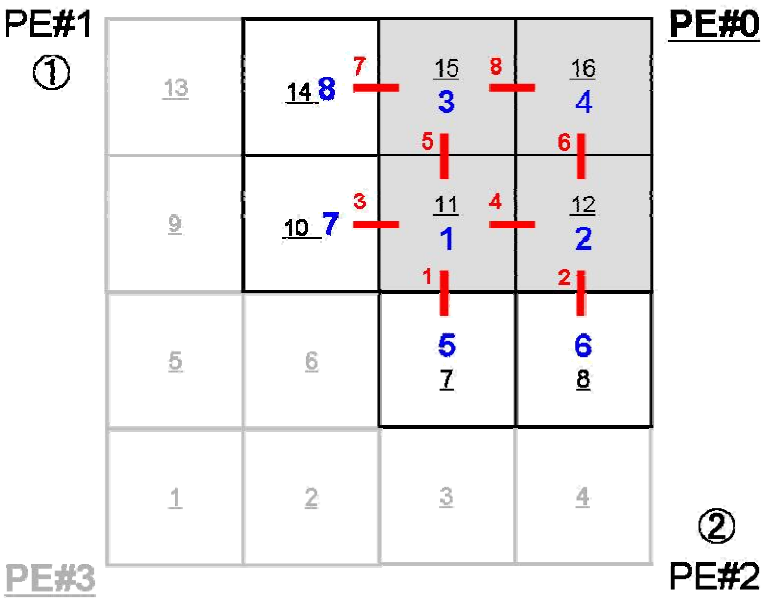
```
$> cd <$P1>/src
$> diff poi_gen.f ../serial/poi_gen.f
```

poi_gen（1/2）

- 係数行列生成部は1PEの場合と全く同じ。
- 理由
  - 係数行列生成に必要な情報が全て局所分散メッシュファイルに含まれている。

局所分散メッシュファイル

```
8
1    1.00    1.00    2.50E+00    2.50E+00    5.00E-01
2    1.00    1.00    3.50E+00    2.50E+00    5.00E-01
3    1.00    1.00    2.50E+00    3.50E+00    5.00E-01
4    1.00    1.00    3.50E+00    3.50E+00    5.00E-01
5    1.00    1.00    2.50E+00    1.50E+00    5.00E-01
6    1.00    1.00    3.50E+00    1.50E+00    5.00E-01
7    1.00    1.00    1.50E+00    2.50E+00    5.00E-01
8    1.00    1.00    1.50E+00    3.50E+00    5.00E-01
8
5          1    1.000000E+00    5.000000E-01    5.000000E-01
6          2    1.000000E+00    5.000000E-01    5.000000E-01
7          1    1.000000E+00    5.000000E-01    5.000000E-01
1          2    1.000000E+00    5.000000E-01    5.000000E-01
1          3    1.000000E+00    5.000000E-01    5.000000E-01
2          4    1.000000E+00    5.000000E-01    5.000000E-01
8          3    1.000000E+00    5.000000E-01    5.000000E-01
3          4    1.000000E+00    5.000000E-01    5.000000E-01
2
2    1.000000E+00    5.000000E-01    0.000000E+00
4    1.000000E+00    5.000000E-01    0.000000E+00
0
1
1    1.000000E+00
```

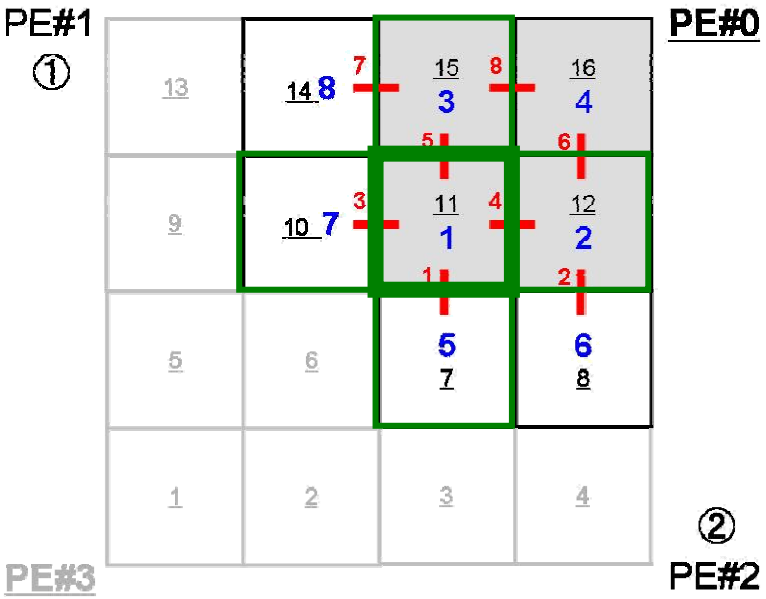


poi_gen（2/2）

- 例えば1番の要素（全体番号11番）における係数行列生成に必要な隣接要素の情報（2,3,5,7番（全体番号：12,15,7,10））は全て局所分散メッシュデータに全て含まれている。
  - うち，5番，7番は外点

局所分散メッシュファイル

```
8
1    1.00    1.00    2.50E+00    2.50E+00    5.00E-01
2    1.00    1.00    3.50E+00    2.50E+00    5.00E-01
3    1.00    1.00    2.50E+00    3.50E+00    5.00E-01
4    1.00    1.00    3.50E+00    3.50E+00    5.00E-01
5    1.00    1.00    2.50E+00    1.50E+00    5.00E-01
6    1.00    1.00    3.50E+00    1.50E+00    5.00E-01
7    1.00    1.00    1.50E+00    2.50E+00    5.00E-01
8    1.00    1.00    1.50E+00    3.50E+00    5.00E-01
8
5         1    1.000000E+00    5.000000E-01    5.000000E-01
6         2    1.000000E+00    5.000000E-01    5.000000E-01
7         1    1.000000E+00    5.000000E-01    5.000000E-01
1         2    1.000000E+00    5.000000E-01    5.000000E-01
1         3    1.000000E+00    5.000000E-01    5.000000E-01
2         4    1.000000E+00    5.000000E-01    5.000000E-01
8         3    1.000000E+00    5.000000E-01    5.000000E-01
3         4    1.000000E+00    5.000000E-01    5.000000E-01
2
2    1.000000E+00    5.000000E-01    0.000000E+00
4    1.000000E+00    5.000000E-01    0.000000E+00
0
1
1    1.000000E+00
```



前処理付き共役勾配法の並列化
Preconditioned Conjugate Gradient Method（CG）

Compute $\mathbf{r}^{(0)}= \mathbf{b}-[A]\mathbf{x}^{(0)}$
for i= 1, 2, …
  solve $[M]\mathbf{z}^{(i-1)}= \mathbf{r}^{(i-1)}$
  $\rho_{i-1}= \mathbf{r}^{(i-1)}\ \mathbf{z}^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  endif
  $\mathbf{q}^{(i)}= [A]\mathbf{p}^{(i)}$
  $\alpha_i = \rho_{i-1}/\mathbf{p}^{(i)}\mathbf{q}^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence $|\mathbf{r}|$
end

並列計算，領域間通信が必要な部分

- 行列ベクトル積
- 内積

hpcmw_eps_fvm_comm.f

- MPIに関連した共通サブルーチン群
- MPIサブルーチンの代替
- 「eps_fvm」における1対1通信
  - 一般化された通信テーブル
- 利点
  - MPIサブルーチンを直接呼ぶよりも引数を減らすことができる。
  - 1対1通信においては「送信バッファへの代入，送信，受信，受信バッファからの読み出し」という操作をそのたびに記述する必要が無くなる。

- HPC-MW（HPC-Middleware）
  - 隠蔽

hpcmw_eps_fvm_commの内容

http://www-solid.eps.s.u-tokyo.ac.jp/~nakajima/tutorial/parallel_lib_tutorial/

MPI_BARRIER

```
 subroutine hpcmw_eps_fvm_barrier
```

MPI_ALLREDUCE(スカラー)

```
 subroutine hpcmw_eps_fvm_allreduce_R ( VAL, ntag)
 subroutine hpcmw_eps_fvm_allreduce_I ( VAL, ntag)
```

MPI_BCAST(スカラー)

```
 subroutine hpcmw_eps_fvm_bcast_R ( VAL, nbase)
 subroutine hpcmw_eps_fvm_bcast_I ( VAL, nbase)
 subroutine hpcmw_eps_fvm_bcast_C ( VAL, n, nbase)
```

MPI_ALLREDUCE(ベクトル)

```
 subroutine hpcmw_eps_fvm_allreduce_RV ( VAL, n, ntag)
 subroutine hpcmw_eps_fvm_allreduce_IV ( VAL, n, ntag)
```

MPI_BCAST(ベクトル)

```
 subroutine hpcmw_eps_fvm_bcast_RV ( VAL, n, nbase)
 subroutine hpcmw_eps_fvm_bcast_IV ( VAL, n, nbase)
 subroutine hpcmw_eps_fvm_bcast_CV ( VAL, n, nn, nbase)
```

一対一通信用(ベクトル)

```
 subroutine hpcmw_eps_fvm_update_1_R (X, n)
```

hpcmw_eps_fvm_allreduce_R

```
!C
!C***
!C*** hpcmw_eps_fvm_allREDUCE_R
!C***
!C
      subroutine hpcmw_eps_fvm_allreduce_R ( VAL, ntag)
      use hpcmw_eps_fvm_util
      implicit REAL*8 (A-H,O-Z)
      integer :: ntag, ierr
      real(kind=kreal) :: VAL, VALM

      if (ntag .eq. hpcmw_sum) then
        call MPI_allREDUCE                                          &
     &       (VAL, VALM, 1, MPI_DOUBLE_PRECISION, MPI_SUM,          &
     &        MPI_COMM_WORLD, ierr)
      endif

      if (ntag .eq. hpcmw_max) then
        call MPI_allREDUCE                                          &
     &       (VAL, VALM, 1, MPI_DOUBLE_PRECISION, MPI_MAX,          &
     &        MPI_COMM_WORLD, ierr)
      endif

      if (ntag .eq. hpcmw_min) then
        call MPI_allREDUCE                                          &
     &       (VAL, VALM, 1, MPI_DOUBLE_PRECISION, MPI_MIN,          &
     &        MPI_COMM_WORLD, ierr)
      endif

      VAL= VALM

      end subroutine hpcmw_eps_fvm_allreduce_R
```

hpcmw_eps_fvm_allreduce_RV

```
!C
!C***
!C*** hpcmw_eps_fvm_allREDUCE_RV
!C***
!C
      subroutine hpcmw_eps_fvm_allreduce_RV ( VAL, n, ntag)
      use hpcmw_eps_fvm_util
      implicit REAL*8 (A-H,O-Z)
      integer :: n, ntag, ierr
      real(kind=kreal), dimension(n) :: VAL
      real(kind=kreal), dimension(:), allocatable :: VALM

      allocate (VALM(n))
      if (ntag .eq. hpcmw_sum) then
        call MPI_allREDUCE                                          &
     &       (VAL, VALM, n, MPI_DOUBLE_PRECISION, MPI_SUM,          &
     &        MPI_COMM_WORLD, ierr)
      endif

      if (ntag .eq. hpcmw_max) then
        call MPI_allREDUCE                                          &
     &       (VAL, VALM, n, MPI_DOUBLE_PRECISION, MPI_MAX,          &
     &        MPI_COMM_WORLD, ierr)
      endif

      if (ntag .eq. hpcmw_min) then
        call MPI_allREDUCE                                          &
     &       (VAL, VALM, n, MPI_DOUBLE_PRECISION, MPI_MIN,          &
     &        MPI_COMM_WORLD, ierr)
      endif

      VAL= VALM
      deallocate (VALM)

      end subroutine hpcmw_eps_fvm_allreduce_RV
```

hpcmw_eps_fvm_update_1_R（1/2）

```
      subroutine hpcmw_eps_fvm_update_1_R (X, n)
      use hpcmw_eps_fvm_util

      implicit REAL*8 (A-H,O-Z)
      integer :: n, nn, ierr
      real(kind=kreal), dimension(n) :: X
      real(kind=kreal), dimension(:), allocatable :: WS, WR

      integer(kind=kint ), dimension(:,:), allocatable :: sta1
      integer(kind=kint ), dimension(:,:), allocatable :: sta2
      integer(kind=kint ), dimension(:  ), allocatable :: req1
      integer(kind=kint ), dimension(:  ), allocatable :: req2

      nn= max (n, import_index(n_neighbor_pe),                          &
     &            export_index(n_neighbor_pe))

      allocate (WS(nn), WR(nn))
!C
!C-- INIT.
      allocate (sta1(MPI_STATUS_SIZE,n_neighbor_pe))
      allocate (sta2(MPI_STATUS_SIZE,n_neighbor_pe))
      allocate (req1(n_neighbor_pe))
      allocate (req2(n_neighbor_pe))
```

送信，受信バッファの定義

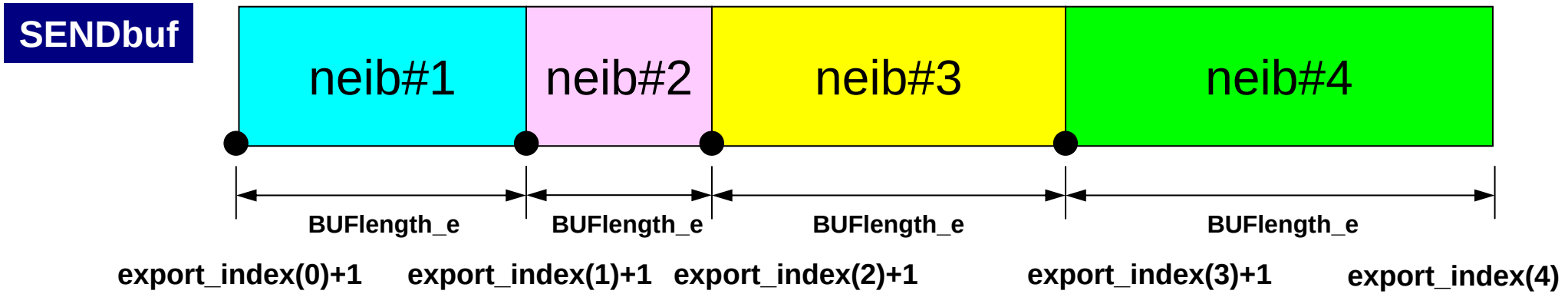
hpcmw_eps_fvm_update_1_R（2/2）

```
!C
!C-- SEND
      do neib= 1, n_neighbor_pe
        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,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req1(neib), ierr)
      enddo
!C
!C-- RECEIVE
      do neib= 1, n_neighbor_pe
        istart= import_index(neib-1)
        inum  = import_index(neib  ) - istart
        call MPI_IRECV (WR(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req2(neib), ierr)
      enddo

      call MPI_WAITALL (n_neighbor_pe, req2, sta2, ierr)

      do neib= 1, n_neighbor_pe
        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 (n_neighbor_pe, req1, sta1, ierr)

      deallocate (sta1, sta2, req1, req2, WS, WR)]
      end subroutine hpcmw_eps_fvm_update_1_R
```

送信バッファへの代入

送信

送信（MPI_Isend/Irecv/Waitall）




```
 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)
```

送信バッファへの代入
温度などの変数を直接送信，受信に使うのではなく，このようなバッファへ一回代入して計算することを勧める。

hpcmw_eps_fvm_update_1_R（2/2）

```
!C
!C-- SEND
      do neib= 1, n_neighbor_pe
        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,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req1(neib), ierr)
      enddo
!C
!C-- RECEIVE
      do neib= 1, n_neighbor_pe
        istart= import_index(neib-1)
        inum  = import_index(neib  ) - istart
        call MPI_IRECV (WR(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req2(neib), ierr)
      enddo

      call MPI_WAITALL (n_neighbor_pe, req2, sta2, ierr)

      do neib= 1, n_neighbor_pe
        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 (n_neighbor_pe, req1, sta1, ierr)

      deallocate (sta1, sta2, req1, req2, WS, WR)]
      end subroutine hpcmw_eps_fvm_update_1_R
```

受信

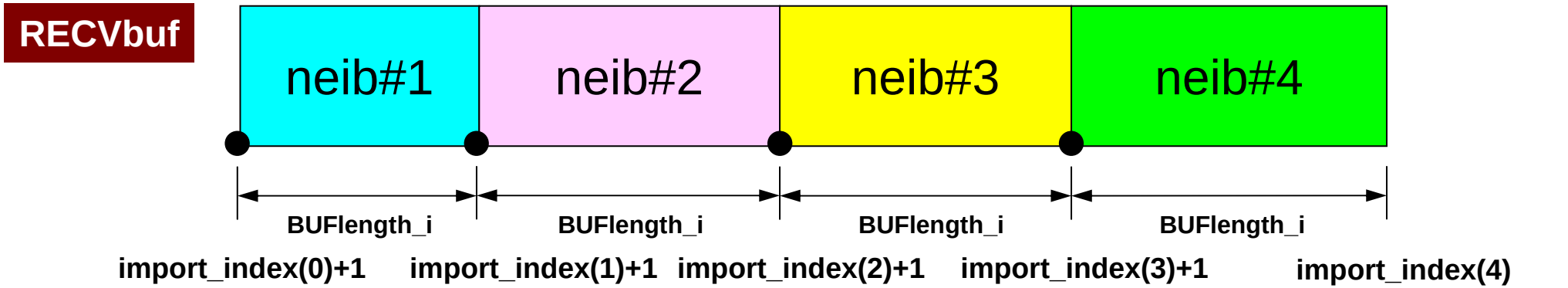
hpcmw_eps_fvm_update_1_R（2/2）

```
!C
!C-- SEND
      do neib= 1, n_neighbor_pe
        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,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req1(neib), ierr)
      enddo
!C
!C-- RECEIVE
      do neib= 1, n_neighbor_pe
        istart= import_index(neib-1)
        inum  = import_index(neib  ) - istart
        call MPI_IRECV (WR(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req2(neib), ierr)
      enddo

      call MPI_WAITALL (n_neighbor_pe, req2, sta2, ierr)

      do neib= 1, n_neighbor_pe
        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 (n_neighbor_pe, req1, sta1, ierr)

      deallocate (sta1, sta2, req1, req2, WS, WR)]
      end subroutine hpcmw_eps_fvm_update_1_R
```

受信バッファからのデータ取り出し

受信（MPI_Isend/Irecv/Waitall）

```
 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)                                  受信バッファから代入
     enddo
  enddo
```



共役勾配法

```
!C
!C***
!C*** CG
!C***
!C
      subroutine hpcmw_eps_fvm_solver_CG                                &
     &          ( N, NP, NPLU, D, B, X, EPS, ITR, IER,                  &
     &            index, item, COEF, Tcomm)

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

      real(kind=kreal), dimension(N ) :: D
      real(kind=kreal), dimension(NP) :: B
      real(kind=kreal), dimension(NP) :: X

      integer             , dimension(0:N) :: index
      integer             , dimension(NPLU):: item
      real   (kind=kreal), dimension(NPLU):: COEF

      real(kind=kreal) :: EPS, Tcomm

      integer :: ITR, IER
      integer :: P, Q, R, Z, DD

      real(kind=kreal), dimension(:,:), allocatable, save :: W
```

N : intNODE_tot
NP: NODE_tot

hpcmw_solver.f

```
      subroutine hpcmw_eps_fvm_solver

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

      EPS = 1.d-8
      ITR = NODE_tot

      call hpcmw_eps_fvm_solver_CG                                    &
     &    ( intNODE_tot, NODE_tot, NPLU, D, BFORCE, DELPHI, EPS,      &
     &      ITR,  IER, index, item, AMAT, COMMtime)

      open (11, file='fvmmg.ctrl', status='unknown')
        read (11,*) NX, NY, NZ
      close (11)

      iS= NX*NY*NZ/2 + NX*NY/2
      do i= iS+1, iS+NX
        write (*,'(i8,3(1pe16.6))') i, DELPHI(i)
      enddo

      end subroutine hpcmw_eps_fvm_solver
```

共役勾配法の「並列化」(1/3)

- 行列ベクトル積

```
!C
!C +-------------+
!C | {q}= [A]{p} |
!C +-------------+
!C===
      exchange W(i,P)

      do i= 1, N
        W(i,Q) = D(i) * W(i,P)
        do j= index(i-1)+1, index(i)
          W(i,Q) = W(i,Q) + COEF(j) * W(item(j),P)
        enddo
      enddo
!C===
```

共役勾配法の「並列化」(1/3)

- 行列ベクトル積

```
!C
!C +-------------+
!C | {q}= [A]{p} |
!C +-------------+
!C===
      S_TIME= MPI_WTIME()
      call hpcmw_eps_fvm_update_1_R (W(1,P), NP)
      E_TIME= MPI_WTIME()
      Tcomm= Tcomm + E_TIME - S_TIME


      do i= 1, N
        W(i,Q) = D(i) * W(i,P)
        do j= index(i-1)+1, index(i)
          W(i,Q) = W(i,Q) + COEF(j) * W(item(j),P)
        enddo
      enddo
!C===
```

共役勾配法の「並列化」(2/3)

- 内積
  - MPI_ALLREDUCE

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

      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

      allreduce RHO
!C===
```

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

      do i= 1, NP
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

      allreduce RHO
!C===
```

こうしてはいけない・・・何故か?
(N:内点数, NP:内点＋外点)

```
N : intNODE_tot
NP: NODE_tot
```

共役勾配法の「並列化」(2/3)

- 内積
  - MPI_ALLREDUCE

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

      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo


      S_TIME= MPI_WTIME()
      call hpcmw_eps_fvm_allreduce_R (RHO, hpcmw_sum)
      E_TIME= MPI_WTIME()
      Tcomm= Tcomm + E_TIME - S_TIME
!C===
```

共役勾配法の「並列化」(3/3)

- N(**intNODE_tot**)とNP(**NODE_tot**)の違いに注意。
  - 基本的に計算はN個分やればよい(ループは`do i=1,N`)。
  - 外点の値を変えるような(左辺にくる)計算はしない。
    - 外点の値が必要なときは行列ベクトル積のみ。
    - そのときに通信して外点の値をもらってくれば良い。

- 最後に，従属変数(X)の外点における最新値をもらっておくことを忘れないように。

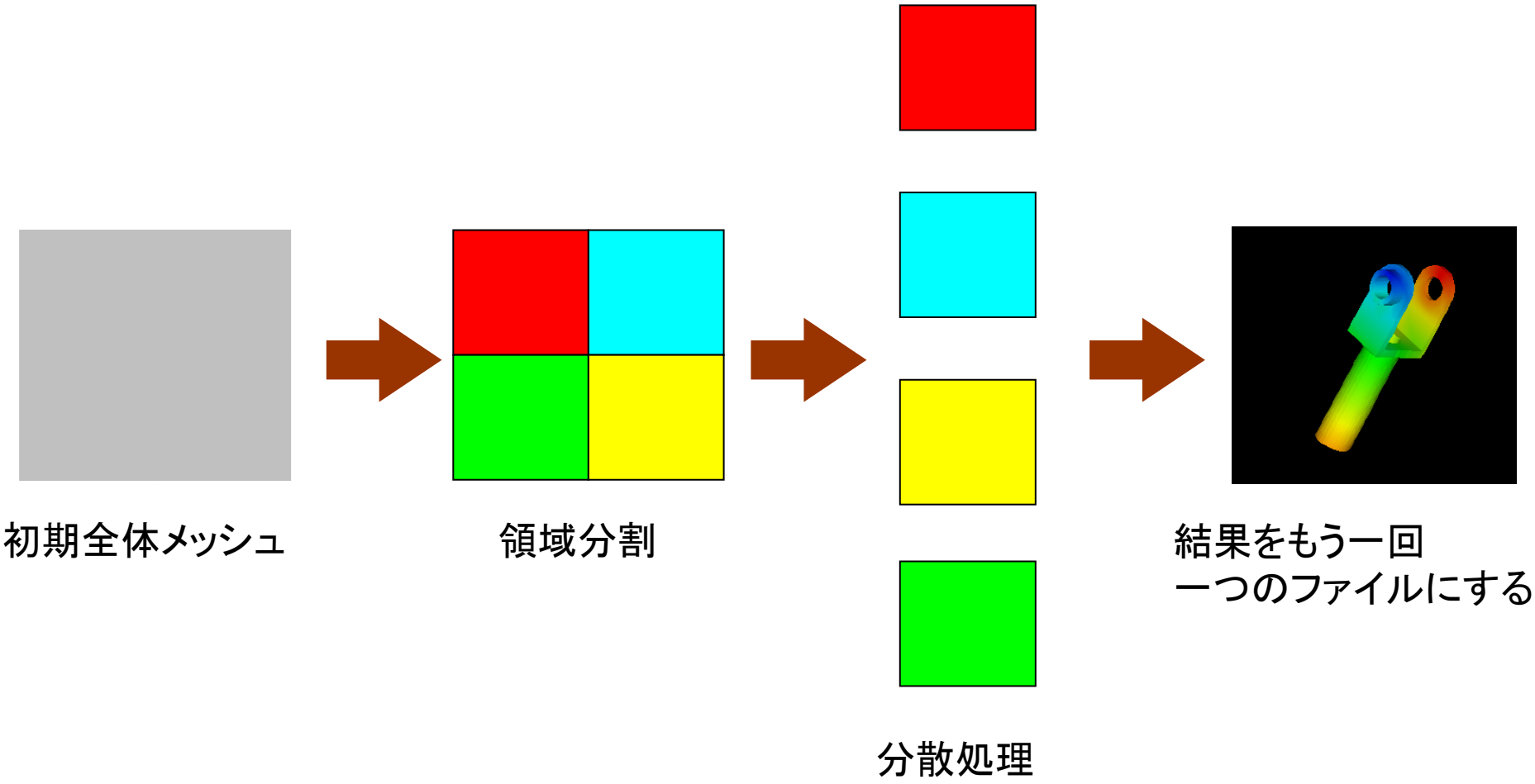
- アプリケーションにおける処理:「eps_fvm」の復習
- 並列化
  - 局所分散データ
  - 共役勾配法
  - 可視化
- 課題P1

可視化

- いわゆる，「並列可視化」をやってもらうつもりだったのであるが，プログラムが完全構造格子用に作られたものであるため，「eps_fvm」のような非構造格子的なものでは対応が難しい。

- 幸い，MicroAVSが各自で使えるので，少し違ったアプローチを試みることにした。

やるべきこと



可視化

- いわゆる，「並列可視化」をやってもらうつもりだったのであるが，プログラムが完全構造格子用に作られたものであるため，「eps_fvm」のような非構造格子的なものでは対応が難しい。

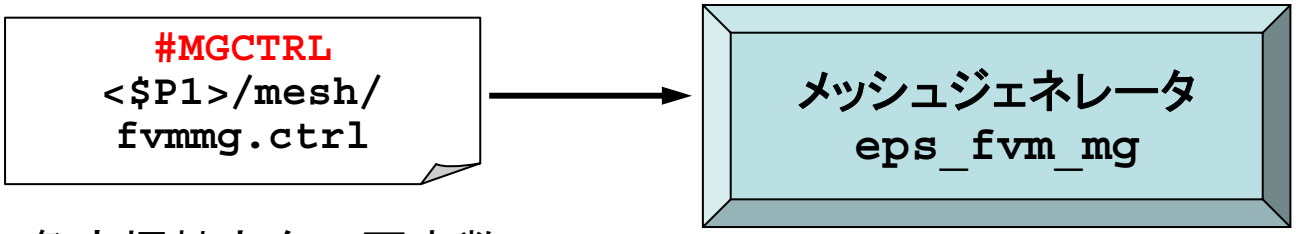
- 幸い，MicroAVSが各自で使えるので，少し違ったアプローチを試みることにした。

- 本当の「並列可視化」とは言えないが，課題S1の成果等を利用する機会にもなるので，このようなやりかたを試みる（**MPI_ALLGATHERv**）。

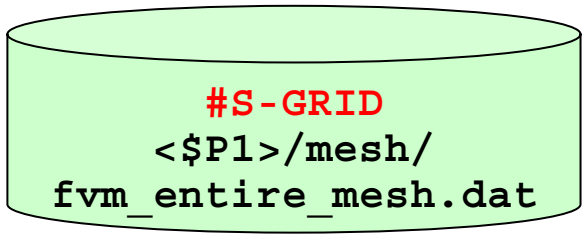
AVS UCDファイル

- 最終出力はUCD（Unstructured Cell Data）フォーマットとして，これをMicroAVSで読み込む。
- ファイル名は「**`*.inp`**」でなければならない。
- UCDファイルは以下の2部分から構成されているのであるが，今回は「形状」については，初期全体メッシュジェネレータで自動的に生成されている（`fvm_entire_mesh.inp-geo`）。
  - 形状（節点，要素）
  - 結果
- 結果のみについてアペンドすればよい。

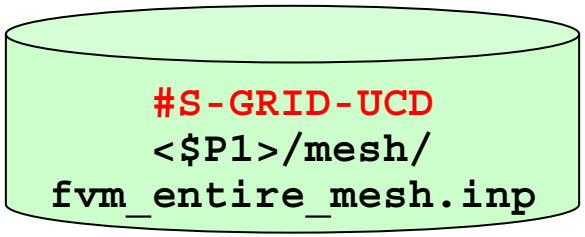
初期全体メッシュ生成



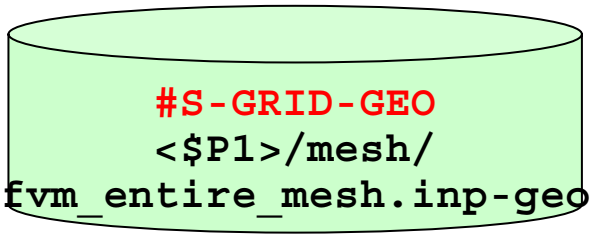
各座標軸方向の要素数
（名称固定）



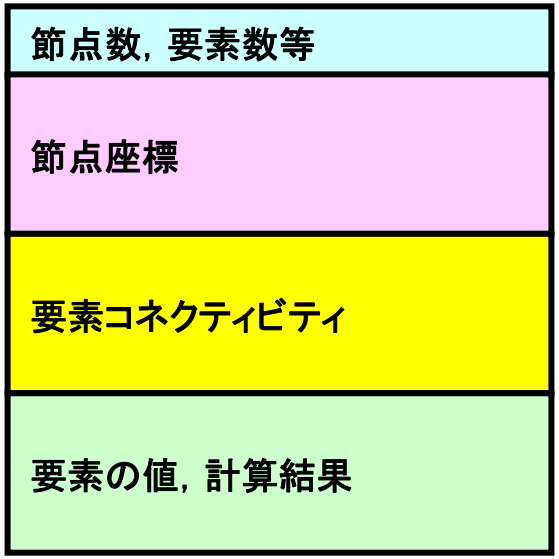
初期全体メッシュデータ
（名称固定）

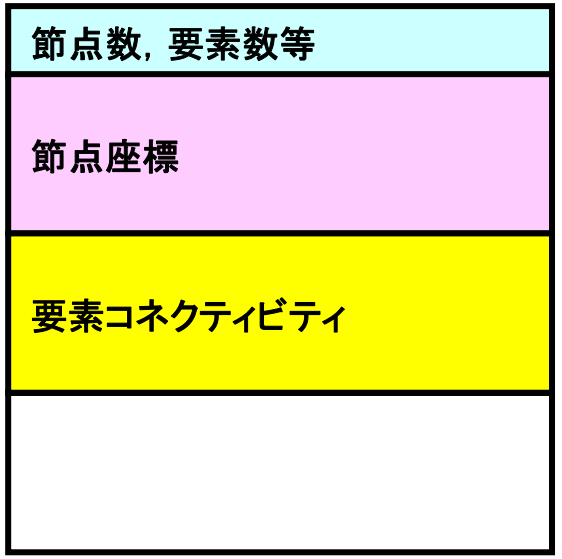


初期全体メッシュデータ
UCDファイル（AVS用）
（名称固定）



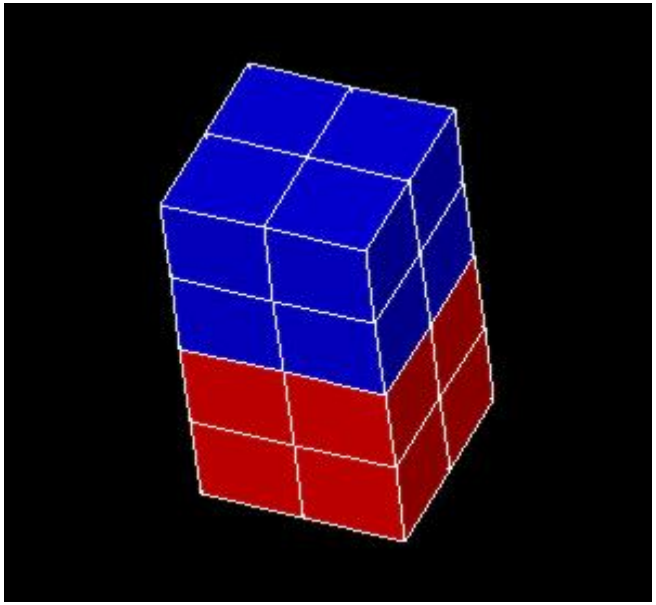
初期全体メッシュデータ
UCDファイル形状部
（AVS用）（名称固定）





AVS UCDファイルの例（1/2）
45節点，16要素の六面体

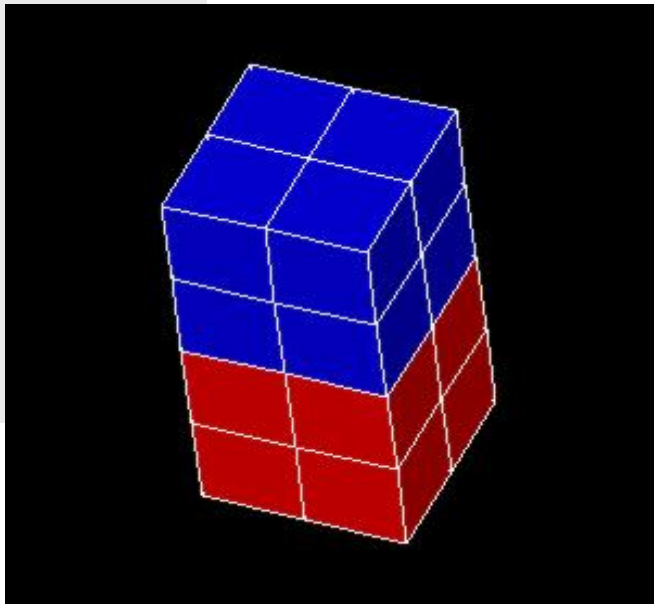
```
45        16         0         1         0
 1    0.000000E+00    0.000000E+00    0.000000E+00
 2    1.000000E+00    0.000000E+00    0.000000E+00
 3    2.000000E+00    0.000000E+00    0.000000E+00
 4    0.000000E+00    1.000000E+00    0.000000E+00
 5    1.000000E+00    1.000000E+00    0.000000E+00
 6    2.000000E+00    1.000000E+00    0.000000E+00
 7    0.000000E+00    2.000000E+00    0.000000E+00
 8    1.000000E+00    2.000000E+00    0.000000E+00
 9    2.000000E+00    2.000000E+00    0.000000E+00
10    0.000000E+00    0.000000E+00    1.000000E+00
11    1.000000E+00    0.000000E+00    1.000000E+00
12    2.000000E+00    0.000000E+00    1.000000E+00
13    0.000000E+00    1.000000E+00    1.000000E+00
14    1.000000E+00    1.000000E+00    1.000000E+00
15    2.000000E+00    1.000000E+00    1.000000E+00
16    0.000000E+00    2.000000E+00    1.000000E+00
17    1.000000E+00    2.000000E+00    1.000000E+00
18    2.000000E+00    2.000000E+00    1.000000E+00
19    0.000000E+00    0.000000E+00    2.000000E+00
20    1.000000E+00    0.000000E+00    2.000000E+00
21    2.000000E+00    0.000000E+00    2.000000E+00
22    0.000000E+00    1.000000E+00    2.000000E+00
23    1.000000E+00    1.000000E+00    2.000000E+00
24    2.000000E+00    1.000000E+00    2.000000E+00
25    0.000000E+00    2.000000E+00    2.000000E+00
26    1.000000E+00    2.000000E+00    2.000000E+00
27    2.000000E+00    2.000000E+00    2.000000E+00
28    0.000000E+00    0.000000E+00    3.000000E+00
29    1.000000E+00    0.000000E+00    3.000000E+00
30    2.000000E+00    0.000000E+00    3.000000E+00
31    0.000000E+00    1.000000E+00    3.000000E+00
32    1.000000E+00    1.000000E+00    3.000000E+00
33    2.000000E+00    1.000000E+00    3.000000E+00
34    0.000000E+00    2.000000E+00    3.000000E+00
35    1.000000E+00    2.000000E+00    3.000000E+00
36    2.000000E+00    2.000000E+00    3.000000E+00
37    0.000000E+00    0.000000E+00    4.000000E+00
38    1.000000E+00    0.000000E+00    4.000000E+00
39    2.000000E+00    0.000000E+00    4.000000E+00
40    0.000000E+00    1.000000E+00    4.000000E+00
41    1.000000E+00    1.000000E+00    4.000000E+00
42    2.000000E+00    1.000000E+00    4.000000E+00
43    0.000000E+00    2.000000E+00    4.000000E+00
44    1.000000E+00    2.000000E+00    4.000000E+00
45    2.000000E+00    2.000000E+00    4.000000E+00
```



AVS UCDファイルの例（2/2）
45節点，16要素の六面体

```
          1   1 hex          1     2     5     4    10    11    14    13
          2   1 hex          2     3     6     5    11    12    15    14
          3   1 hex          4     5     8     7    13    14    17    16
          4   1 hex          5     6     9     8    14    15    18    17
          5   1 hex         10    11    14    13    19    20    23    22
          6   1 hex         11    12    15    14    20    21    24    23
          7   1 hex         13    14    17    16    22    23    26    25
          8   1 hex         14    15    18    17    23    24    27    26
          9   1 hex         19    20    23    22    28    29    32    31
         10   1 hex         20    21    24    23    29    30    33    32
         11   1 hex         22    23    26    25    31    32    35    34
         12   1 hex         23    24    27    26    32    33    36    35
         13   1 hex         28    29    32    31    37    38    41    40
         14   1 hex         29    30    33    32    38    39    42    41
         15   1 hex         31    32    35    34    40    41    44    43
         16   1 hex         32    33    36    35    41    42    45    44
  1  1
COLOR, color
          1         2.00e+00
          2         2.00e+00
          3         2.00e+00
          4         2.00e+00
          5         2.00e+00
          6         2.00e+00
          7         2.00e+00
          8         2.00e+00
          9         1.00e+00
         10         1.00e+00
         11         1.00e+00
         12         1.00e+00
         13         1.00e+00
         14         1.00e+00
         15         1.00e+00
         16         1.00e+00
```

この部分のみ自分で作成すれば良い



手 順

- 各プロセッサにおける計算結果（**PHI(:)**に格納される）を，あるプロセッサに集める。
  - 「内点」の結果のみで良い。
- そのプロセッサから，結果を以下の書式で，グローバル番号順に書き出す（「,」は入れない）：

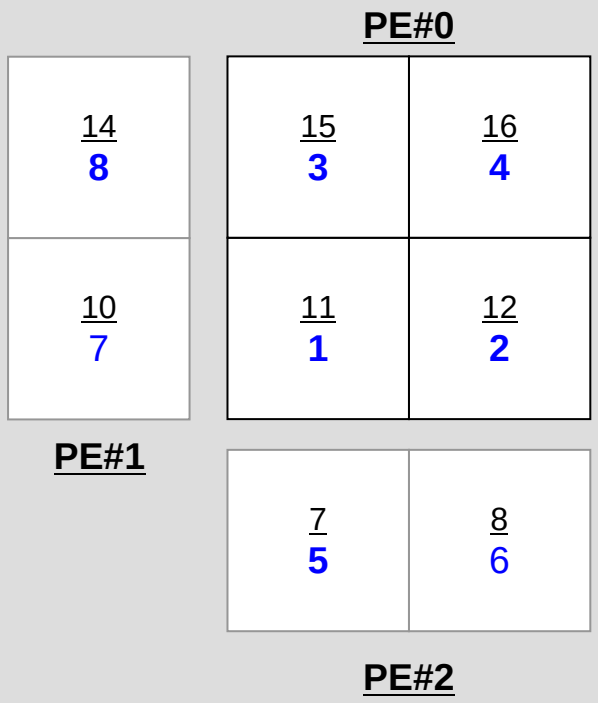
  <グローバル要素番号><計算結果>
- ファイルを「**fvm_entire_mesh.inp-geo**」にアペンドしてUCDファイルを生成する。
- ヒント
  - MPI_ALLGATHERVを使う
  - グローバルIDは次ページの情報から取得

局所分散通信ファイル(comm.0)

内点数，総要素数(内点＋外点)，全体要素番号

```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```

全体要素番号(局所番号順)

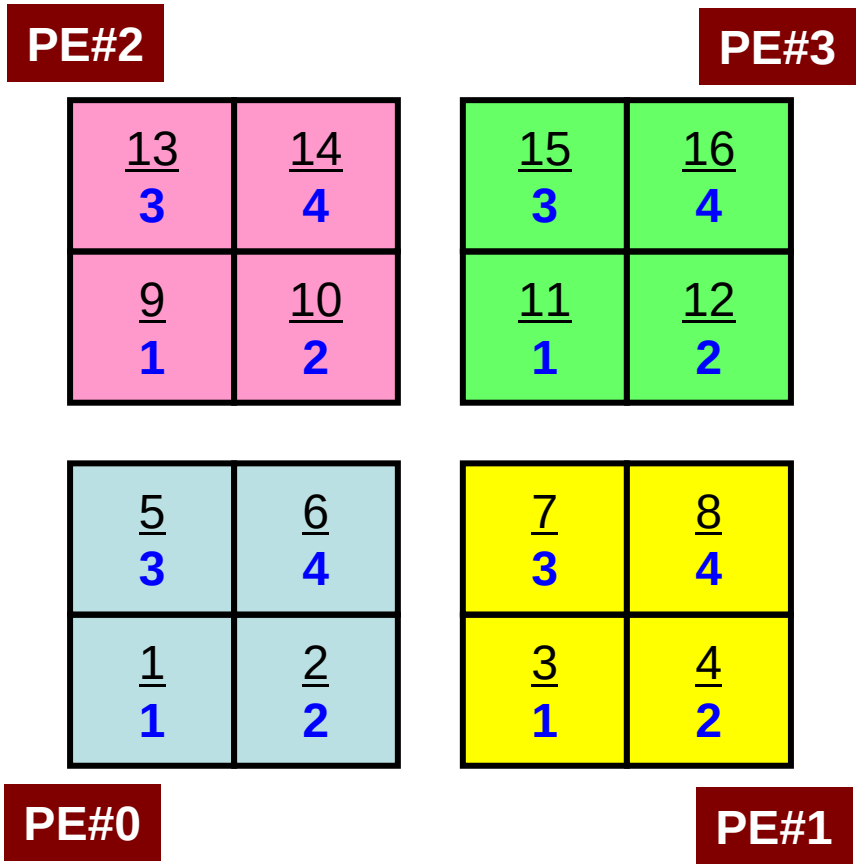


要素のグローバルID
hpcmw_eps_fvm_input_grid

```
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_index(k), k= 1, n_neighbor_pe)
      nn= export_index(n_neighbor_pe)
      allocate (export_item(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_item(k), k= 1, nn)

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') intNODE_tot

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') nn

      allocate (NODE_GLOBAL(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (NODE_GLOBAL(k), k= 1, nn)

      close (IUNIT)
!C===

      end subroutine hpcmw_eps_fvm_input_grid
```

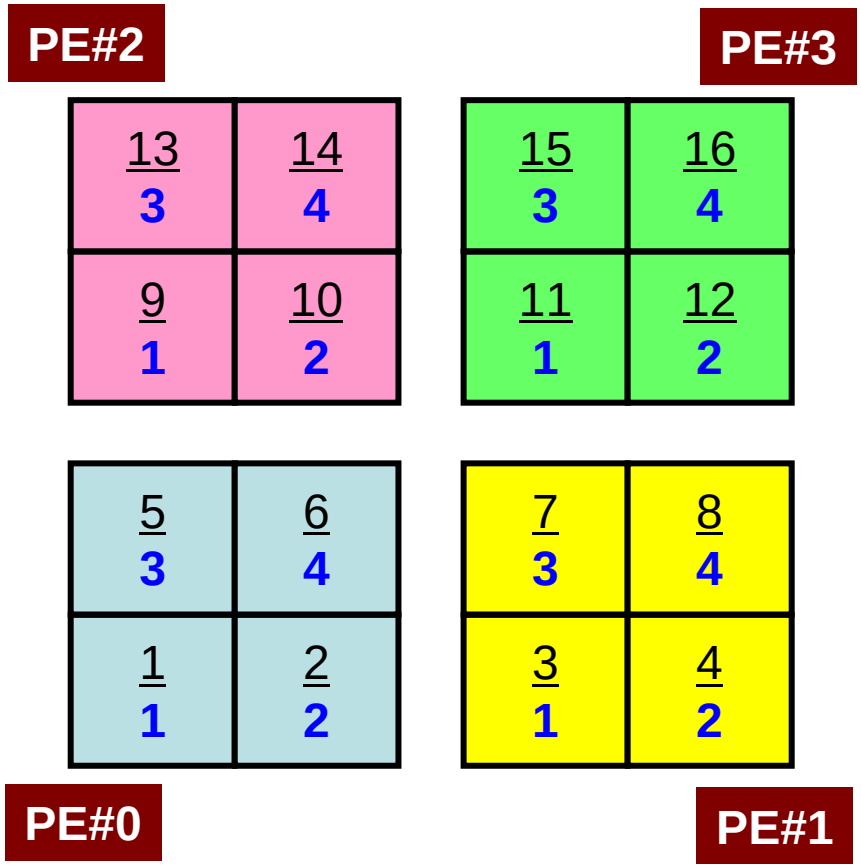
例題

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



内点

例題：計算結果例：PHIの中身



PE#	要素番号（ローカル）	要素番号（グローバル）	PHI(i)
0	1	1	1.150000E+01
0	2	2	9.500000E+00
0	3	5	1.150000E+01
0	4	6	9.500000E+00
2	1	3	6.500000E+00
2	2	4	2.500000E+00
2	3	7	6.500000E+00
2	4	8	2.500000E+00
1	1	9	1.150000E+01
1	2	10	9.500000E+00
1	3	13	1.150000E+01
1	4	14	9.500000E+00
3	1	11	6.500000E+00
3	2	12	2.500000E+00
3	3	15	6.500000E+00
3	4	16	2.500000E+00

hpcmw_eps_fvm_global_output（1/2）

```
      subroutine hpcmw_eps_fvm_global_output
      use hpcmw_eps_fvm_all
      implicit REAL*8 (A-H,O-Z)

      integer, dimension(:), allocatable :: rcounts, displs
      integer, dimension(:), allocatable :: NODE_ID_G
      integer, dimension(:), allocatable :: NEWtoOLD
      real(kind=kreal), dimension(:), allocatable :: VAL
!C
!C-- INIT.
      allocate (rcounts(PETOT), displs(0:PETOT))
      rcounts= 0
      displs = 0

      call MPI_allGATHER                                              &
     &    (intNODE_tot, 1, MPI_INTEGER, rcounts, 1, MPI_INTEGER,      &
     &     MPI_COMM_WORLD, ierr)

      do ip= 1, PETOT
        displs(ip)= displs(ip-1) + rcounts(ip)
      enddo

      NODE_tot_G= displs(PETOT)

      allocate (NODE_ID_G(NODE_tot_G), NEWtoOLD(NODE_tot_G))
      allocate (VAL(NODE_tot_G))
```

AllGATHERvの準備

領域全体の要素数
（内点数の和）

hpcmw_eps_fvm_global_output（2/2）

```
!C
!C-- GLOBAL ARRAY
      call MPI_allGATHERv                                                &
     &    (NODE_GLOBAL, intNODE_tot,         MPI_INTEGER,               &
     &     NODE_ID_G  , rcounts, displs(0), MPI_INTEGER,                 &
     &     MPI_COMM_WORLD, ierr)

      call MPI_allGATHERv                                                &
     &    (PHI, intNODE_tot,         MPI_DOUBLE_PRECISION,               &
     &     VAL   , rcounts, displs(0), MPI_DOUBLE_PRECISION,             &
     &     MPI_COMM_WORLD, ierr)

      if (my_rank.eq.0) then
        do i= 1, NODE_tot_G
          j= NODE_ID_G(i)
          NEWtoOLD(j)= i
        enddo

        IUNIT= 12
        open (IUNIT, file= AVSfile, status='unknown', position='append')
          do j= 1, NODE_tot_G
            ii= NEWtoOLD(j)
            write (IUNIT,'(i8,1pe16.6)') j, VAL(ii)
          enddo
        close (IUNIT)
      endif

      end subroutine hpcmw_eps_fvm_global_output
```

グローバル要素番号 NODE_ID_G

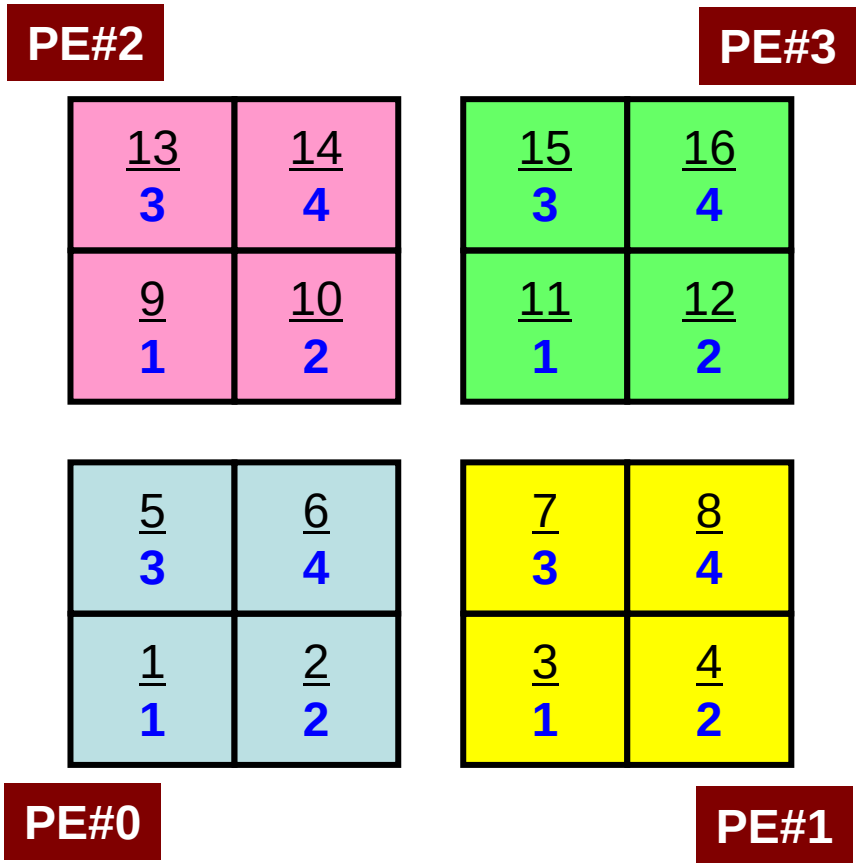
hpcmw_eps_fvm_global_output（2/2）

```
!C
!C-- GLOBAL ARRAY
      call MPI_allGATHERv                                                &
     &    (NODE_GLOBAL, intNODE_tot,         MPI_INTEGER,               &
     &     NODE_ID_G  , rcounts, displs(0), MPI_INTEGER,                 &
     &     MPI_COMM_WORLD, ierr)

      call MPI_allGATHERv                                                &
     &    (PHI, intNODE_tot,         MPI_DOUBLE_PRECISION,               &
     &     VAL   , rcounts, displs(0), MPI_DOUBLE_PRECISION,             &
     &     MPI_COMM_WORLD, ierr)

      if (my_rank.eq.0) then
        do i= 1, NODE_tot_G
          j= NODE_ID_G(i)
          NEWtoOLD(j)= i
        enddo

        IUNIT= 12
        open (IUNIT, file= AVSfile, status='unknown', position='append')
          do j= 1, NODE_tot_G
            ii= NEWtoOLD(j)
            write (IUNIT,'(i8,1pe16.6)') j, VAL(ii)
          enddo
        close (IUNIT)
      endif

      end subroutine hpcmw_eps_fvm_global_output
```

計算結果ベクトル VAL

この状態のVALの中身

PE番号順に格納されており，
グローバル要素番号順になっていない



PE#	i	要素番号（グローバル） NODE_ID_G(i)	VAL(i)
0	1	1	1.150000E+01
0	2	2	9.500000E+00
0	3	5	1.150000E+01
0	4	6	9.500000E+00
1	5	3	6.500000E+00
1	6	4	2.500000E+00
1	7	7	6.500000E+00
1	8	8	2.500000E+00
2	9	9	1.150000E+01
2	10	10	9.500000E+00
2	11	13	1.150000E+01
2	12	14	9.500000E+00
3	13	11	6.500000E+00
3	14	12	2.500000E+00
3	15	15	6.500000E+00
3	16	16	2.500000E+00

hpcmw_eps_fvm_global_output（2/2）

```
!C
!C-- GLOBAL ARRAY
      call MPI_allGATHERv                                                &
     &    (NODE_GLOBAL, intNODE_tot,         MPI_INTEGER,               &
     &     NODE_ID_G  , rcounts, displs(0), MPI_INTEGER,                 &
     &     MPI_COMM_WORLD, ierr)

      call MPI_allGATHERv                                                &
     &    (DELPHI, intNODE_tot,         MPI_DOUBLE_PRECISION,            &
     &     VAL   , rcounts, displs(0), MPI_DOUBLE_PRECISION,             &
     &     MPI_COMM_WORLD, ierr)

      if (my_rank.eq.0) then
        do i= 1, NODE_tot_G
          j= NODE_ID_G(i)
          NEWtoOLD(j)= i
        enddo

        IUNIT= 12
        open (IUNIT, file= AVSfile, status='unknown', position='append')
          do j= 1, NODE_tot_G
            ii= NEWtoOLD(j)
            write (IUNIT,'(i8,1pe16.6)') j, VAL(ii)
          enddo
        close (IUNIT)
      endif

      end subroutine hpcmw_eps_fvm_global_output
```

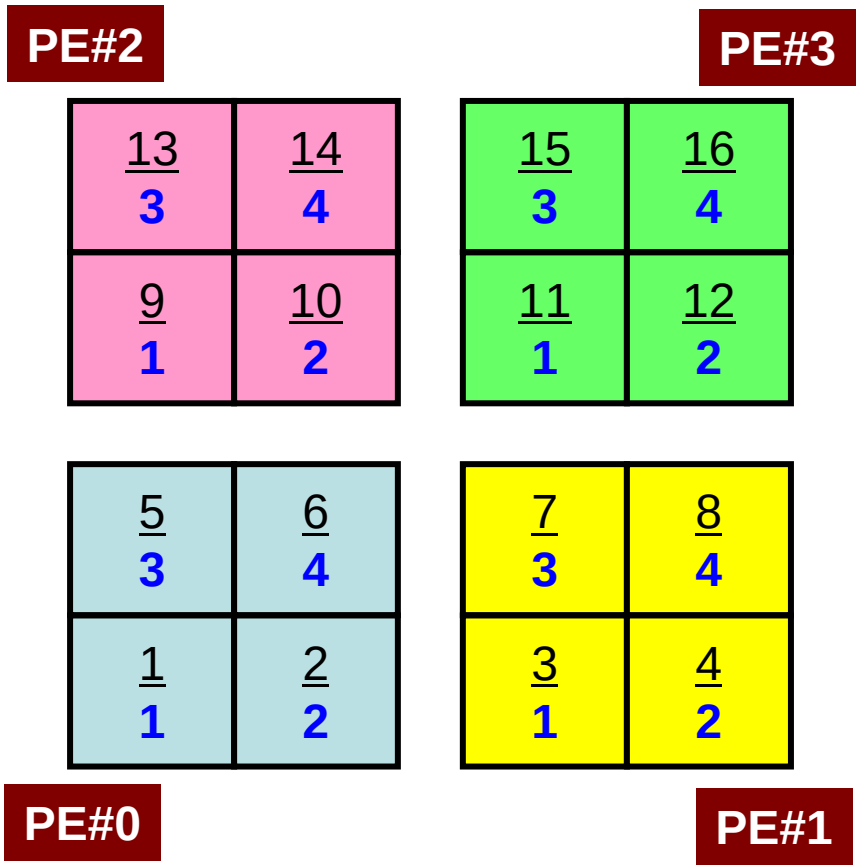
Global番号順に
並べ替え（my_rank=0のみ）

hpcmw_eps_fvm_global_output（2/2）

```
!C
!C-- GLOBAL ARRAY
      call MPI_allGATHERv                                                &
     &    (NODE_GLOBAL, intNODE_tot,         MPI_INTEGER,               &
     &     NODE_ID_G  , rcounts, displs(0), MPI_INTEGER,                 &
     &     MPI_COMM_WORLD, ierr)

      call MPI_allGATHERv                                                &
     &    (DELPHI, intNODE_tot,         MPI_DOUBLE_PRECISION,            &
     &     VAL   , rcounts, displs(0), MPI_DOUBLE_PRECISION,             &
     &     MPI_COMM_WORLD, ierr)

      if (my_rank.eq.0) then
        do i= 1, NODE_tot_G
          j= NODE_ID_G(i)
          NEWtoOLD(j)= i
        enddo

        IUNIT= 12
        open (IUNIT, file= AVSfile, status='unknown')
          do j= 1, NODE_tot_G
            ii= NEWtoOLD(j)
            write (IUNIT,'(i8,1pe16.6)') j, VAL(ii)
          enddo
        close (IUNIT)
      endif

      end subroutine hpcmw_eps_fvm_global_output
```

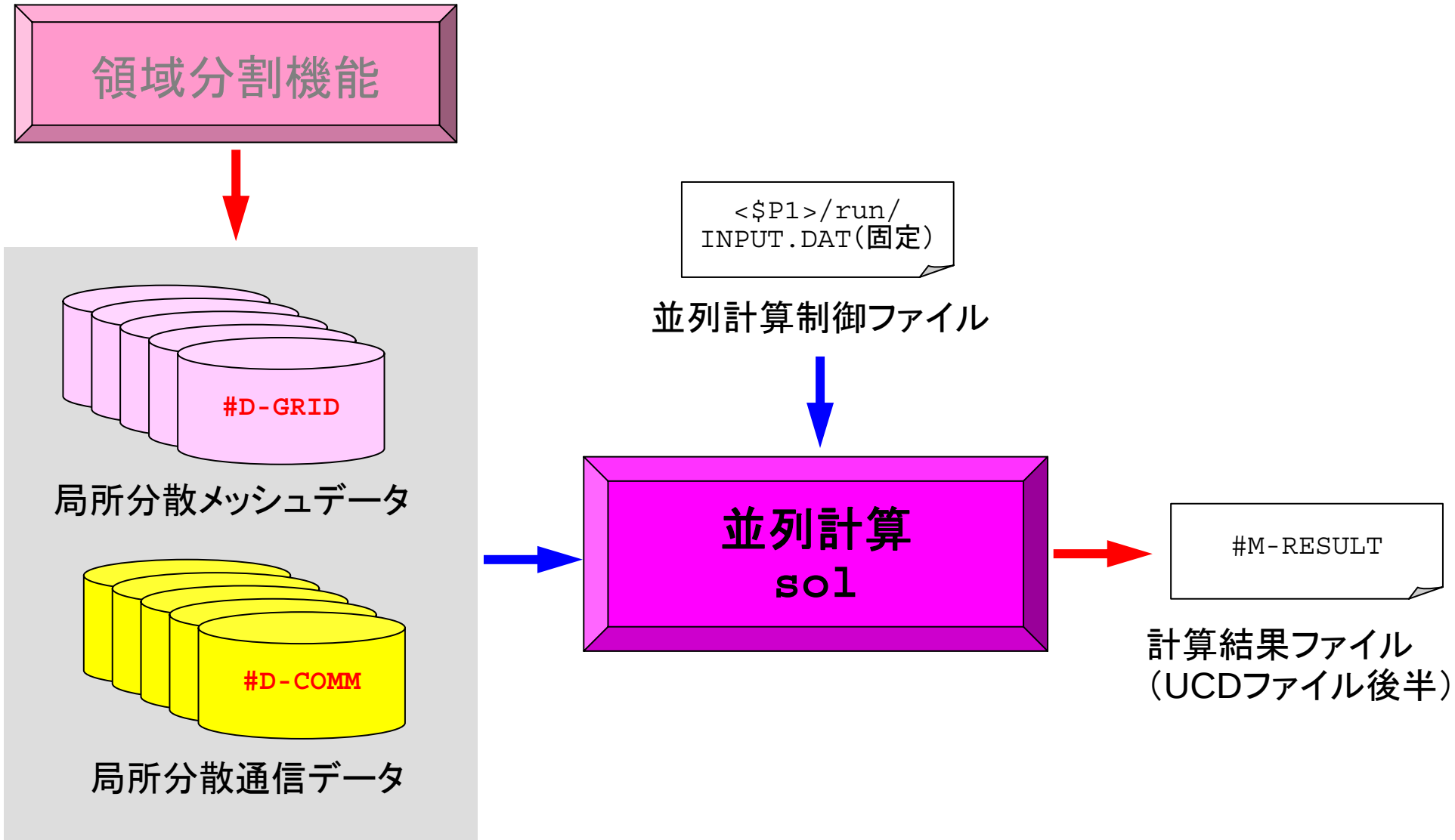
書き出し（my_rank=0のみ）

並び替えた後のVALの中身



PE#	i	要素番号（グローバル）	VAL(i)
0	1	1	1.150000E+01
0	2	2	9.500000E+00
1	3	3	6.500000E+00
1	4	4	2.500000E+00
0	5	5	1.150000E+01
0	6	6	9.500000E+00
1	7	7	6.500000E+00
1	8	8	2.500000E+00
2	9	9	1.150000E+01
2	10	10	9.500000E+00
3	11	11	6.500000E+00
3	12	12	2.500000E+00
2	13	13	1.150000E+01
2	14	14	9.500000E+00
3	15	15	6.500000E+00
3	16	16	2.500000E+00

並列シミュレーションにおけるI/O



並列計算制御ファイル

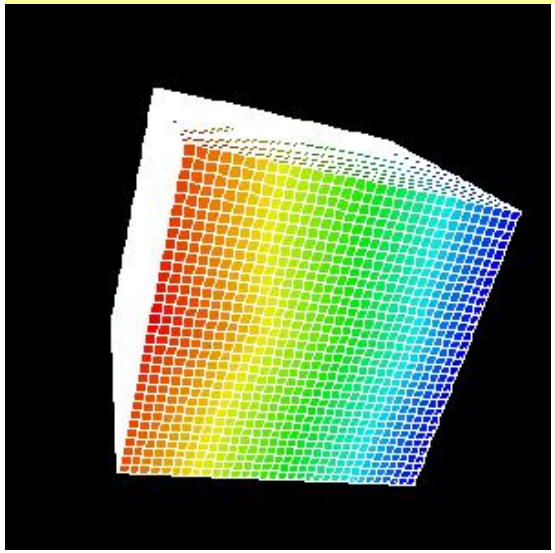
- **INPUT.DAT**（名称固定）
- 実行形式「**sol**」と同じディレクトリになければならない（この場合は**<$P1>/run**）。
- 全ての項目は省略不可。

```
../CS10/mesh.rcb      局所分散メッシュファイルのヘッダ名
../CS10/comm.rcb      局所分散通信ファイルのヘッダ名
../CS10/result        可視化用出力ファイル名（後述）
1                     可視化用出力の有無（=1のとき出力）
```

計算実行，ファイル生成

```
$> cd run
$> cat INPUT.DAT
  ../CS10/mesh.rcb
  ../CS10/comm.rcb
  ../CS10/result
  1

$> qsub go.sh
$> ls -l ../CS11/result
  result

$> cd ../CS11
$> cat fvm_entire_mesh.inp-geo result > test.inp
```

これが書きだされた結果ファイル



形状部分に結果ファイルを
アペンドして，AVS出力用の
ファイルを生成する。

- アプリケーションにおける処理:「eps_fvm」の復習
- 並列化
  - 局所分散データ
  - 共役勾配法
  - 可視化
- **課題P1**

「eps_fvm」の並列化

```
program eps_fvm
use hpcmw_eps_fvm_all

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

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call hpcmw_eps_fvm_global_output

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

可視化処理の部分を除くと，並列計算特有の処理は，共役勾配法の内積，行列ベクトル積程度。それも，hpcmw_eps_fvm_comm.fのようなサブルーチン群を使用すれば，一行の挿入で済んでしまう。

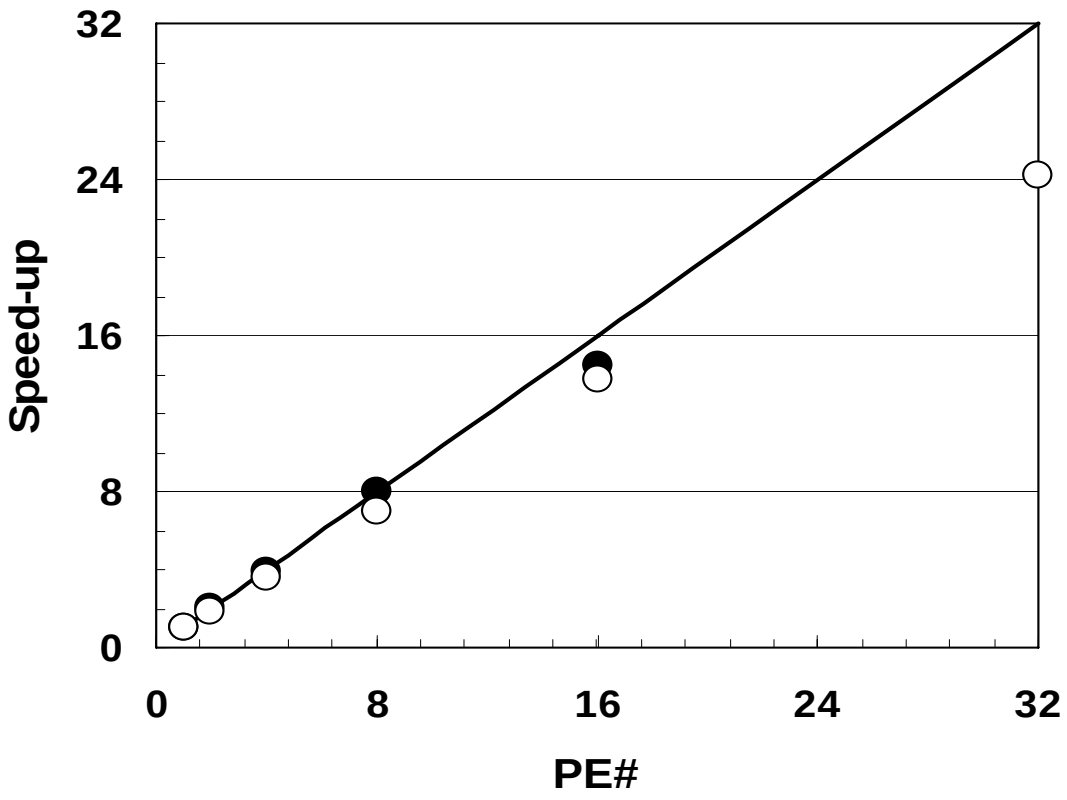
並列計算の「隠蔽」

並列分散データ構造の適切な設計が全て：一般化された通信テーブル

課題P1

- Strong Scaling
  - 問題サイズを固定，PE数を変化させて時間（全体，各部分）を測定。
- Weak Scaling
  - PEあたりの問題サイズを固定，1反復あたりの計算時間を求める。
- 考慮すべき項目
  - 問題サイズ
  - 1PE/ノード，2PE/ノード
  - 領域分割手法（RCB，METIS，1D～3D（次ページ））の影響。
- 注意
  - 夜間，休日等であれば8PEより多く使用しても良い（利用状況による）。
  - 通信時間を計測する工夫をして見よ。

計算例：Strong Scaling

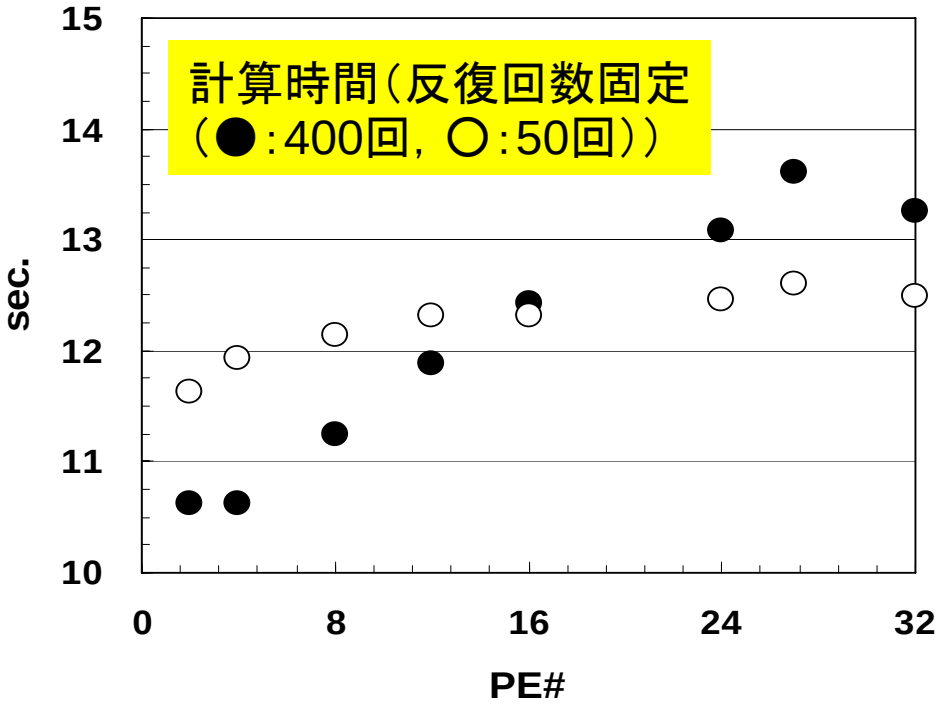


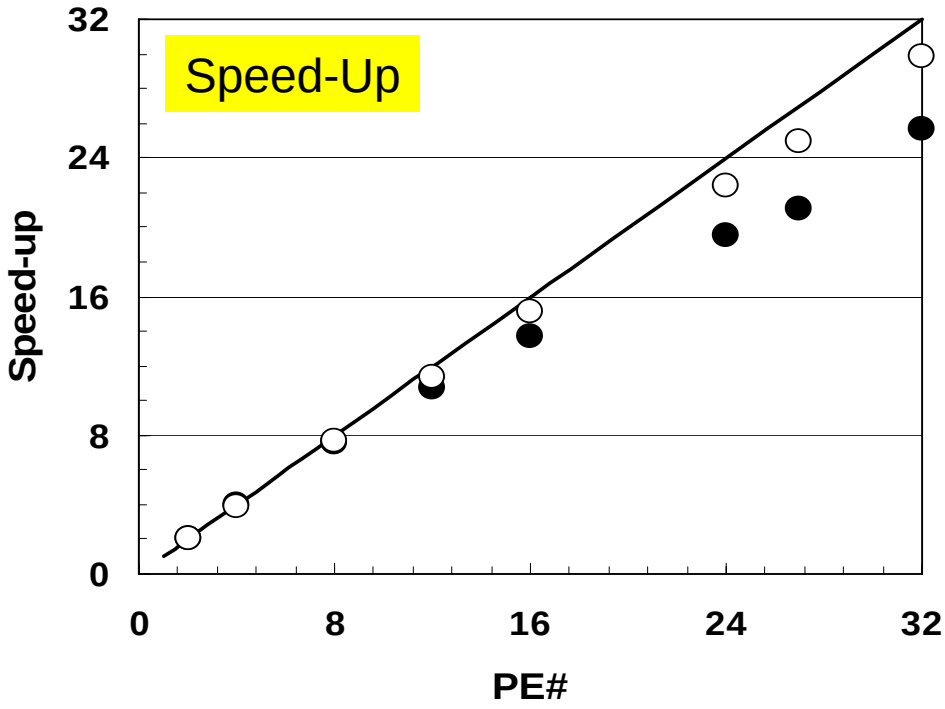
○：2PE/node, ●：1PE/node

- 問題規模を128^3に固定した場合
  - Strong Scaling
  - 1プロセッサの場合の結果を基準

計算例(別の問題)

PE当たり問題規模を固定:Weak Scaling





○:3×32^3DOF/PE, ●: 3×16^3DOF/PE

- 通信が無い場合，計算時間のプロットは平坦になるはずであるが，PE数が増加すると，通信の影響で増加する。

反復回数を固定したい場合

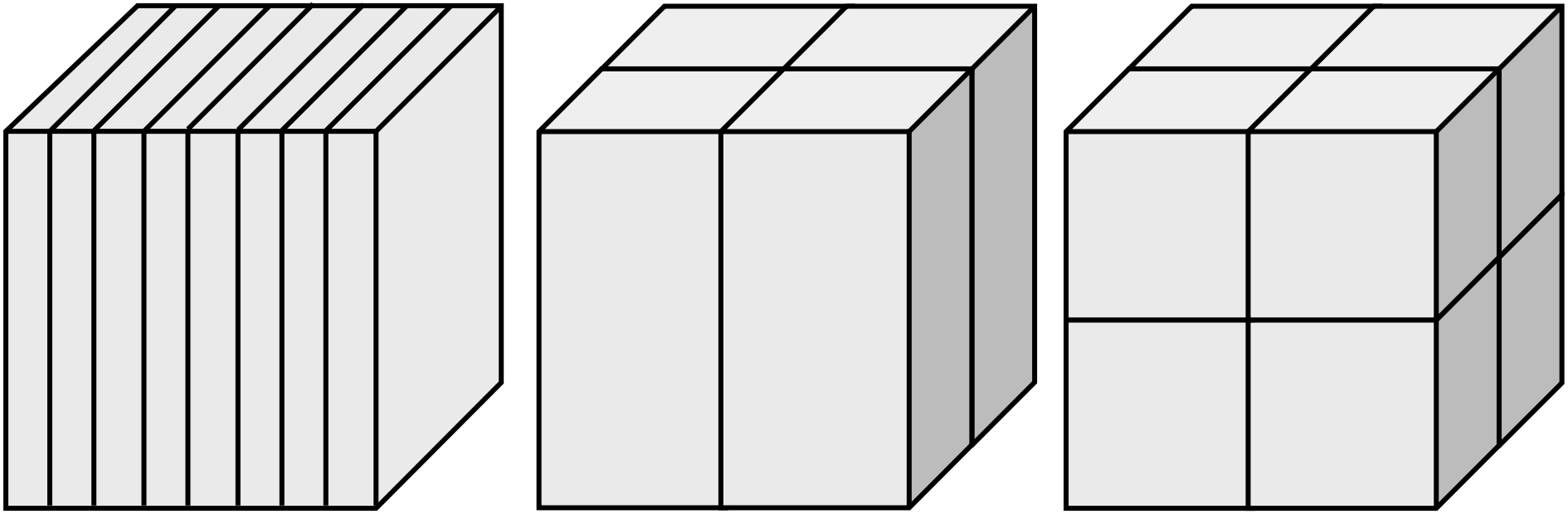
- hpcmw_eps_fvm_solver

```
!C
!C***
!C*** hpcmw_solver
!C***
!C
      subroutine hpcmw_eps_fvm_solver

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

      ISET= 0
      EPS = 1.d-8
      ITR = intNODE_tot
      call hpcmw_eps_fvm_allreduce_R (ITR, hpcmw_max)
      ITR = PETOT * ITR

      S_TIME= MPI_WTIME()
      call hpcmw_eps_fvm_solver_CG                                    &
     &    ( intNODE_tot, NODE_tot, N2, D, BFORCE, DELPHI, EPS,        &
     &      ITR, IER, WAindex, WAcoef, COMMtime)
      E_TIME= MPI_WTIME()
      ISET= ISET + 1
```

この部分を例えば「ITR=50」のようにする。

1D～3D分割

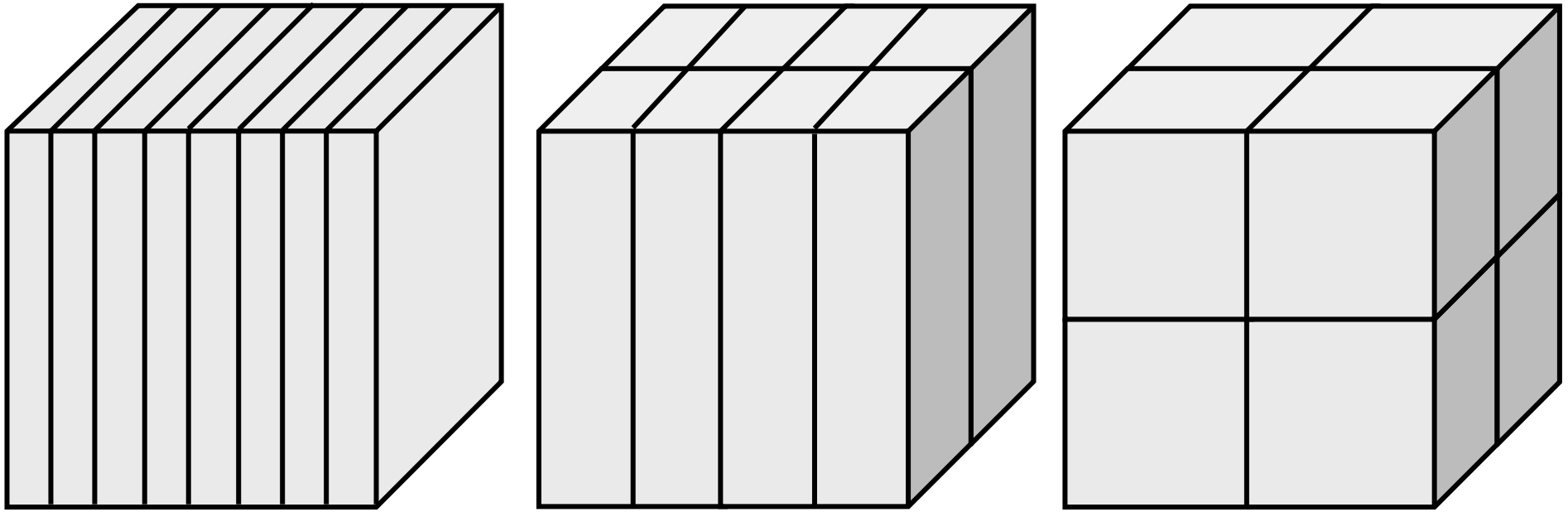
どの方法が良いか考えて見よ



1D型　2D型　3D型

1D～3D分割

通信量の総和（各辺4N，8領域とする）

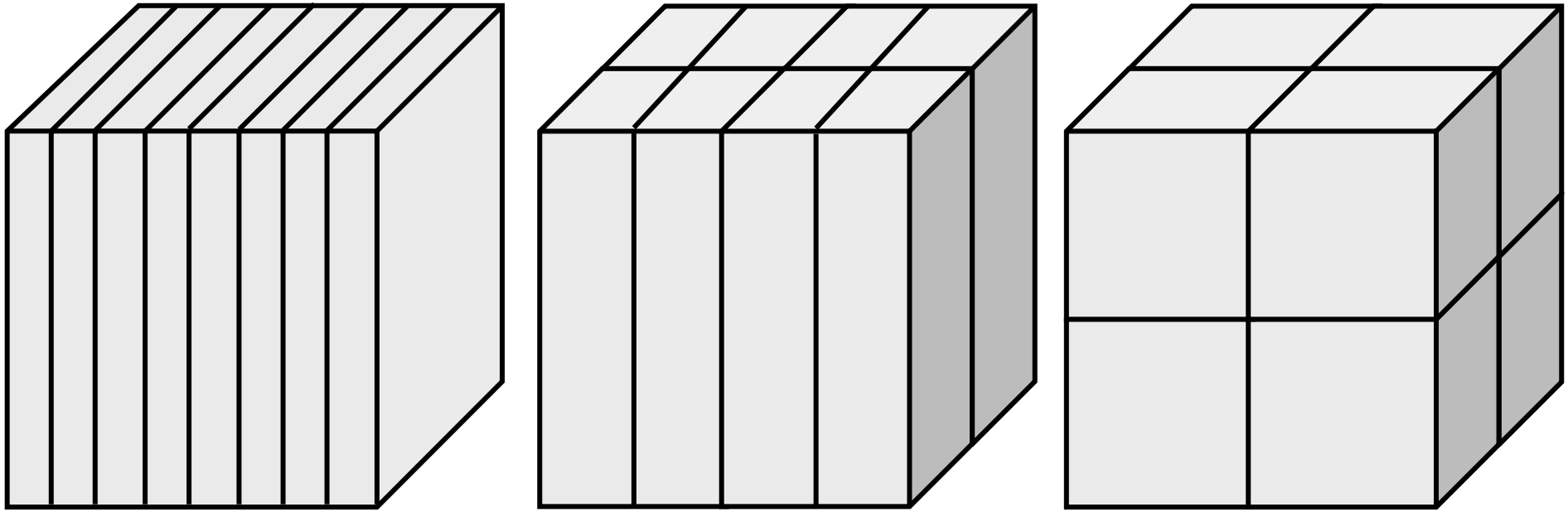


1D型　　2D型　　3D型

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分割

80x80x80要素，8領域の計算例



1D型　　2D型　　3D型

16 N^2 x 7 = 112 N^2

16 N^2 x 4 = 64 N^2

16 N^2 x 3 = 48 N^2

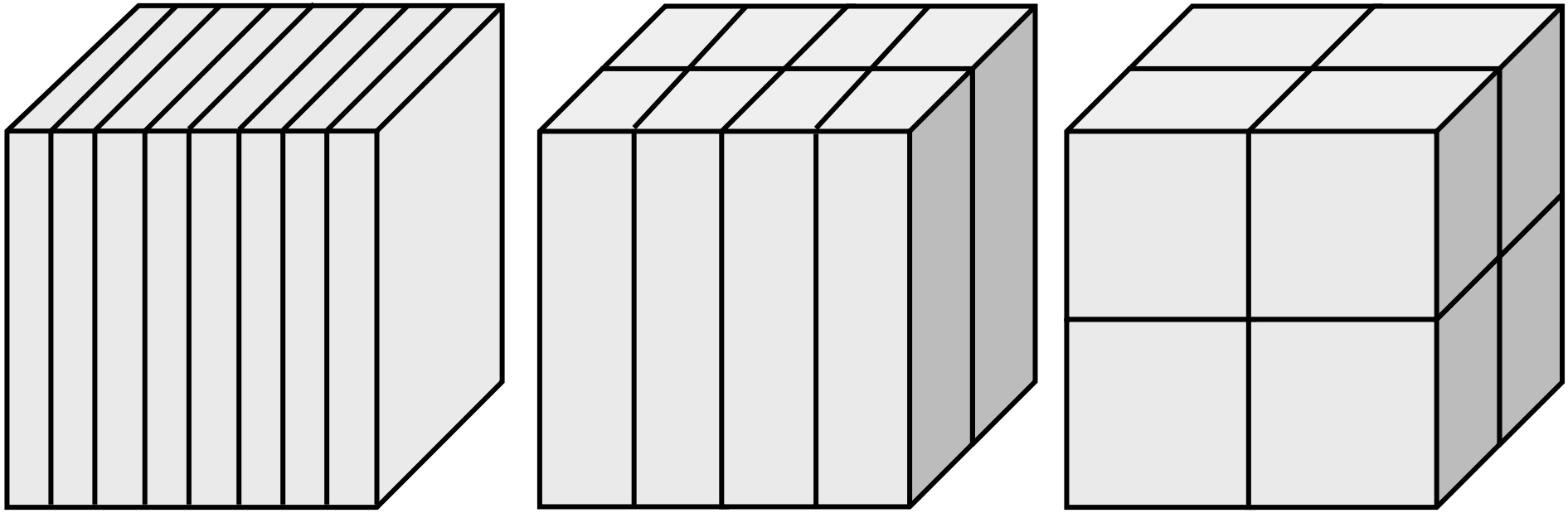
```
### solver time     8.867188E+00
### comm.  time     3.488281E+00
### work ratio      6.066079E+01
```

```
### solver time     6.574219E+00
### comm.  time     1.285156E+00
### work ratio      8.045157E+01
```

```
### solver time     6.230469E+00
### comm.  time     1.031250E+00
### work ratio      8.344828E+01
```

1D～3D分割

fvmpart.ctrl



1D型　2D型　3D型

```
!METHOD
RCB
X,X,X

!REGION NUMBER
8
```

```
!METHOD
RCB
X,Y,X

!REGION NUMBER
8
```

```
!METHOD
RCB
X,Y,Z

!REGION NUMBER
8
```