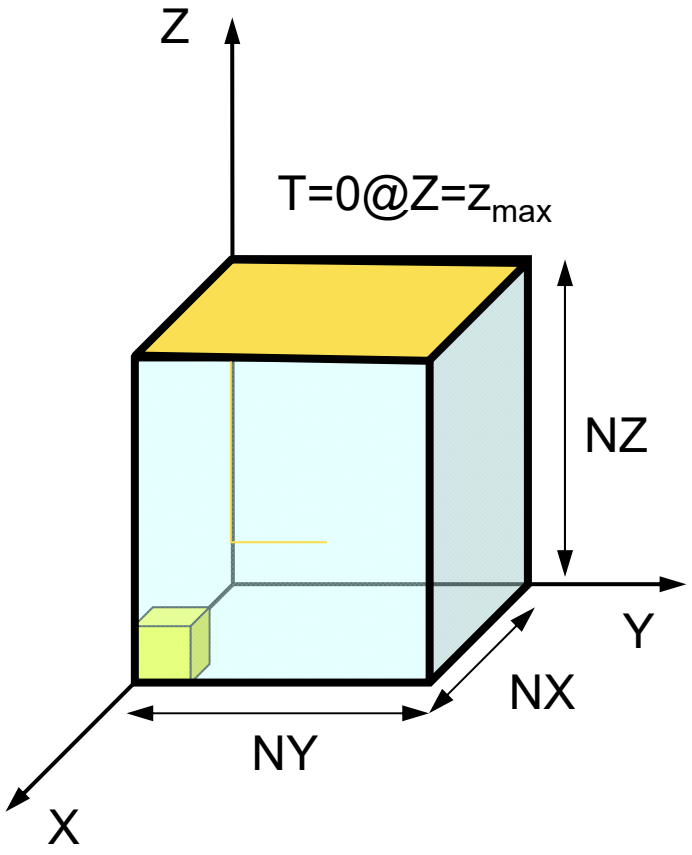
並列有限要素法による
三次元定常熱伝導解析プログラム
演習編

中島 研吾
東京大学情報基盤センター

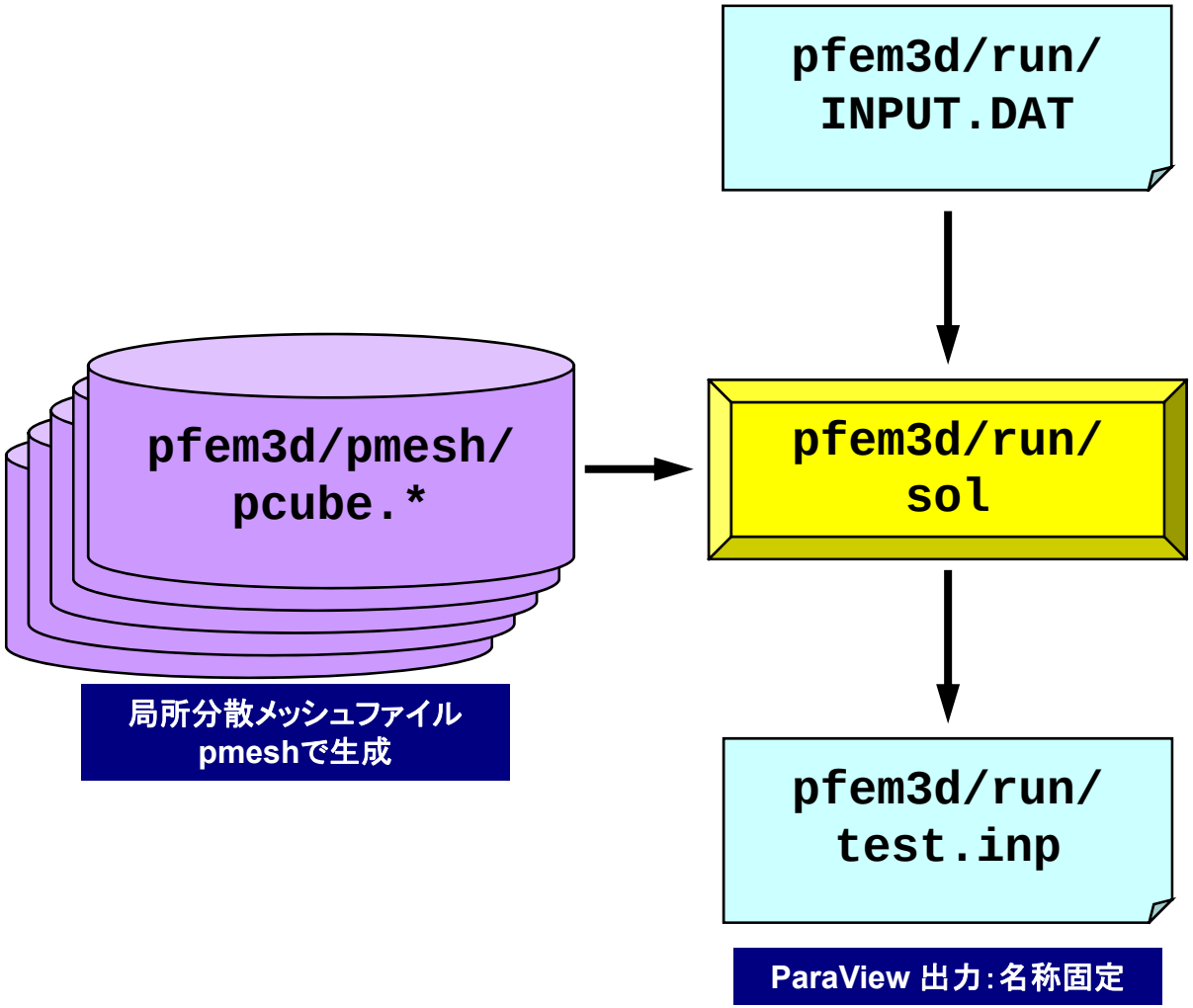
対象とする問題：三次元定常熱伝導

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



- 定常熱伝導＋発熱
- 一様な熱伝導率 λ
- 直方体
  - 一辺長さ1の立方体（六面体）要素
  - 各方向にNX・NY・NZ個
- 境界条件
  - T=0@Z=z_{max}
- 体積当たり発熱量は位置（メッシュの中心の座標 x_c,y_c）に依存
  - $\dot{Q}(x,y,z)=QVOL|x_C+y_C|$

並列有限要素法の手順（並列計算実行）

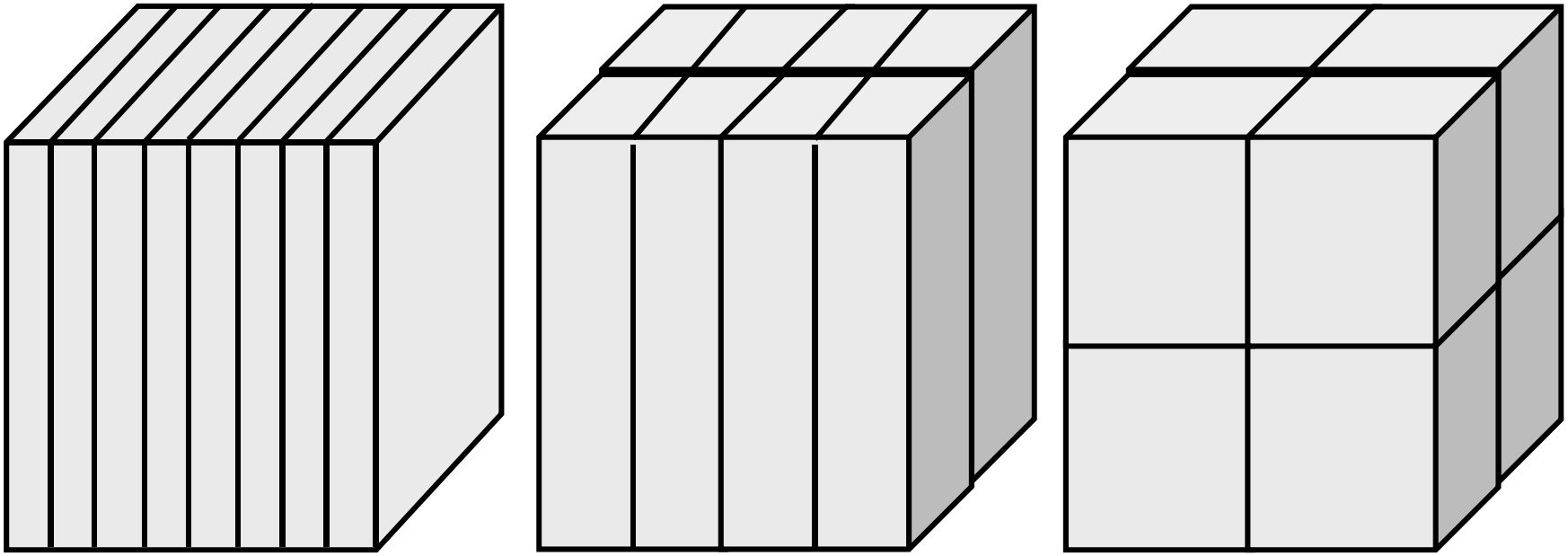


課題(1/2)

- 自分で問題設定を行い，「sol」の挙動を分析してみよ
- 例
  - 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
    - 領域分割手法(RCB，K-METIS，P-METIS，1D～3D)の影響。
  - メッシュ生成，領域分割におけるFX10の性能低い
    - 128^3くらいが限界(領域分割に15分以上かかる)
- 「*.inp」の出力に時間がかかる場合がある。
  - OUTPUT_UCDの呼び出しのコメントアウト
  - src，part

1D～3D分割

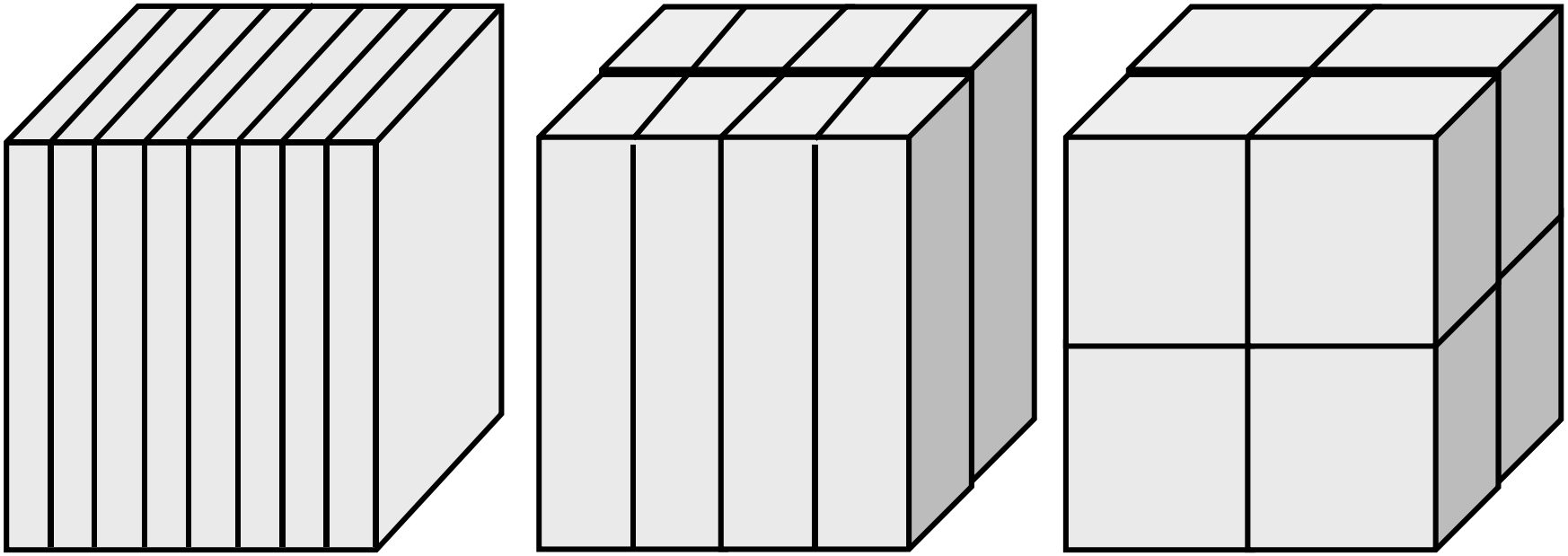
どの方法が良いか考えて見よ



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

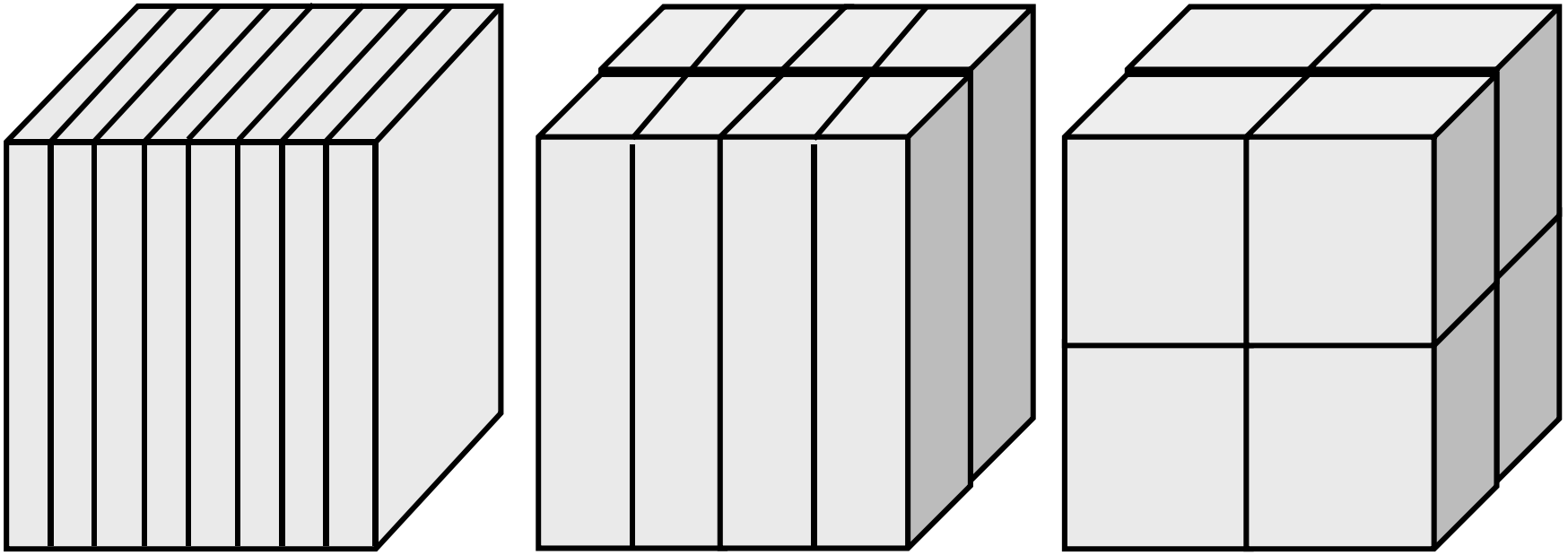
16 N^2 x 7 = 112 N^2

16 N^2 x 4 = 64 N^2

16 N^2 x 3 = 48 N^2

1D～3D分割

mesh.inp



1D型

```
64  64  64
 8   1   1
pcube
```

2D型

```
64  64  64
 4   2   1
pcube
```

3D型

```
64  64  64
 2   2   2
pcube
```

課題(2/2)

- 領域間通信(solver_SR)の性能改善ができないかどうか考えてみよ。
  - Recv. Bufferへのコピーを効率的に実施できないか?

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

      call MPI_Waitall (NEIBPETOT, req2, sta2, ierr)

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

SEND/RECV (Original)

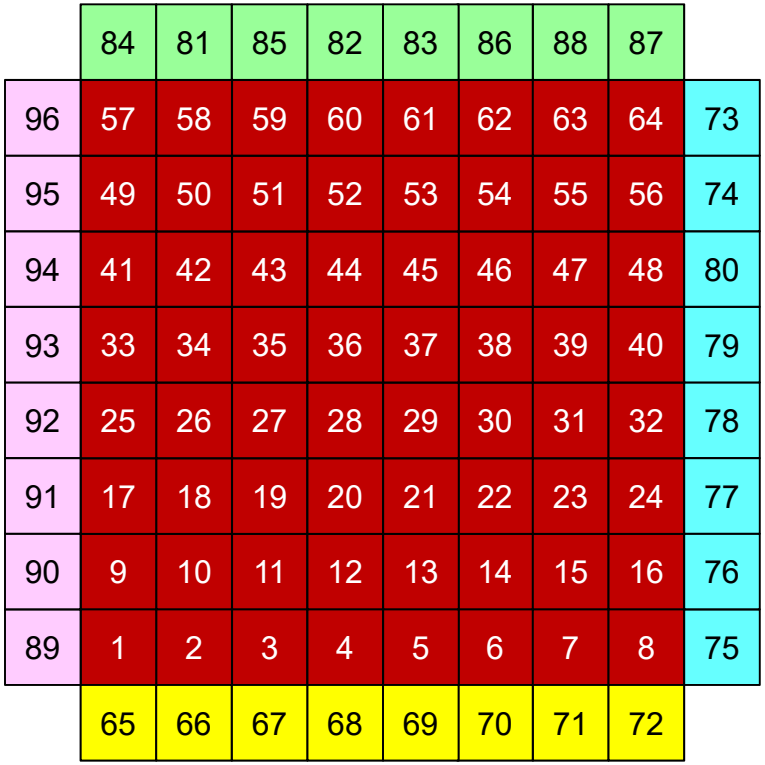
```
!C
!C-- INIT.
      allocate (sta1(MPI_STATUS_SIZE,NEIBPETOT), sta2(MPI_STATUS_SIZE,NEIBPETOT))
      allocate (req1(NEIBPETOT), req2(NEIBPETOT))

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

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

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

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



SEND/RECV (NEW:1)

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

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

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

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

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

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

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

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

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

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

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

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

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

N0: 内点総数

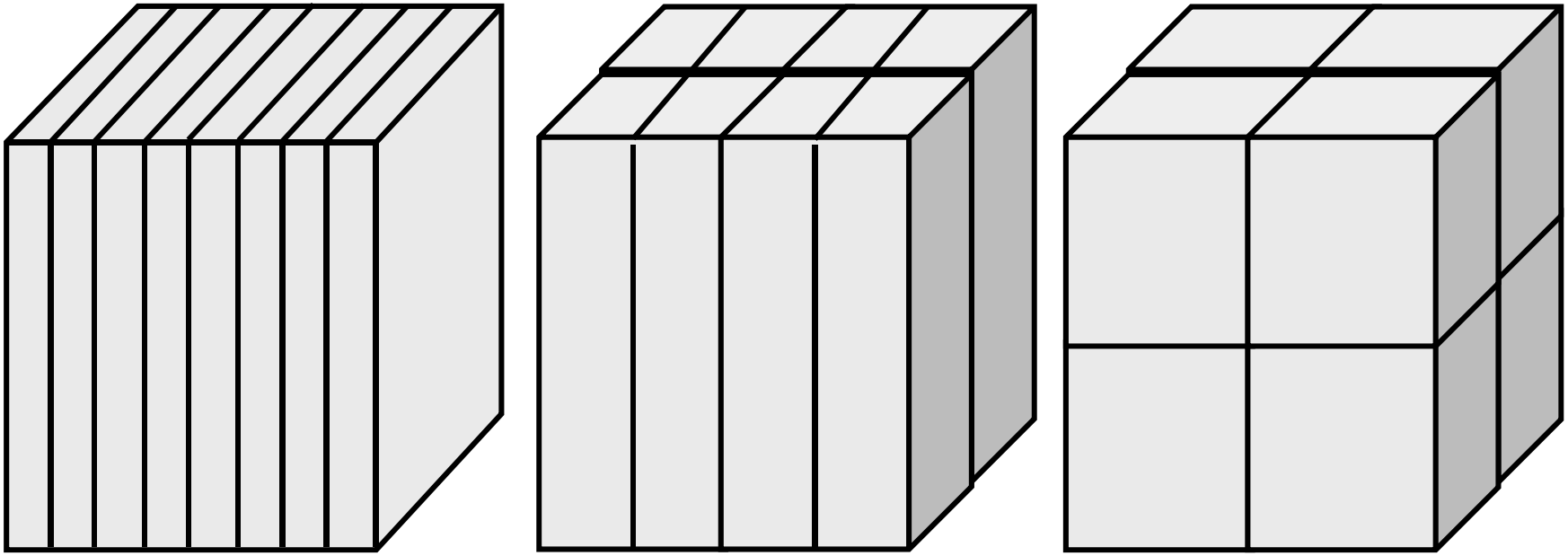
計算例(通信最適化済)@東大 Strong Scaling

- 192×192×128節点(4,718,592節点，4,633,087要素)
- 16～192コア
- Solver計算時間

core #	sec. (speed-up)
16（4x2x2）	16.6（16.0）
32（4x4x2）	8.69（30.5）
64（4x4x4）	4.55（59.6）
96（6x4x4）	3.54（74.7）
128（8x4x4）	2.69（98.4）
192（8x6x4）	1.84（144.）

例題：1D～3D分割

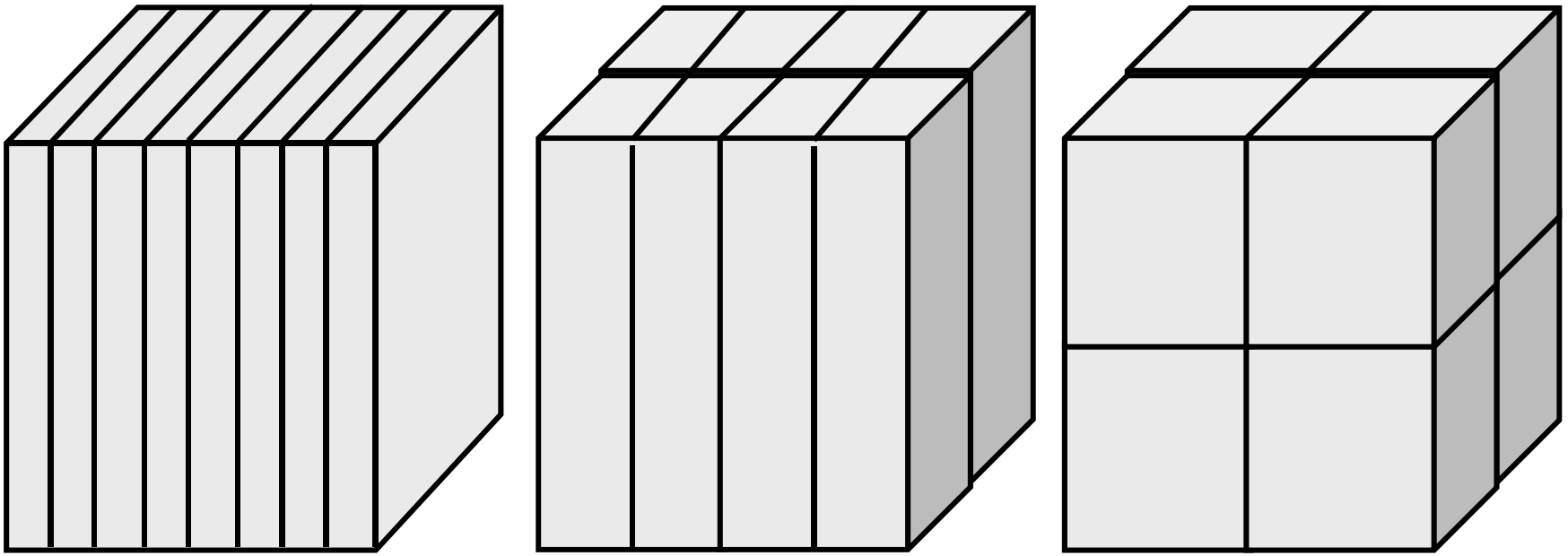
どの方法が良いか考えて見よ



1D型　2D型　3D型

手順(1/4)

```
>$ cd ~/pFEM/pfem3d/pmesh
(1.”mesh.inp”を変更)
```



1D型

```
 80  80  80
  8   1   1
pcube
```

2D型

```
 80  80  80
  4   2   1
pcube
```

3D型

```
 80  80  80
  2   2   2
pcube
```

手順(2/4)

(2.”mg.sh”を変更)

```
#!/bin/sh
#PJM -L “node=1“                ノード数（≦12）
#PJM -L “elapse=00:10:00“       実行時間（≦15分）
#PJM -L “rscgrp=small“          実行キュー名
#PJM -
#PJM -o “mg.lst“                標準出力
#PJM --mpi “proc=8“             MPIプロセス数（≦192）

mpiexec ./pmesh
```

手順(3/4)

```
>$ cd ../run
(3.”go0.sh”を修正)

>$ pjsub go0.sh
```

```
#!/bin/sh
#PJM -L “node=1“                  ノード数（≦12）
#PJM -L “elapse=00:10:00“         実行時間（≦15分）
#PJM -L “rscgrp=small“            実行キュー名
#PJM -
#PJM -o “mg.lst“                  標準出力
#PJM --mpi “proc=8“               MPIプロセス数（≦192）

mpiexec ./sol0
```

制御ファイル：INPUT.DAT

```
../pmesh/pcube HEADER
2000           ITER
1.0 1.0        COND, QVOL
1.0e-08        RESID
```

- HEADER：　局所分散ファイルヘッダ名
  pcube.my_rank
- ITER：　反復回数上限
- COND：　熱伝導率
- QVOL：　体積当たり発熱量係数
- RESID：　反復法の収束判定値

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

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

手順(4/4):PCへファイルを移動

(4.手元のPCでcygwinもしくはterminalを立ち上げ
　適当なフォルダへ移動)

```
>$ scp sus15XX@pi.ircpi.kobe-u.ac.jp:~/pFEM/run/test.inp .
```

（5.Paraviewを立ち上げて，上記”test.inp”を開く）

計算例(通信最適化済)@東大 Strong Scaling

- 192×192×128節点(4,718,592節点，4,633,087要素)

core #	sec. (speed-up)
16 (4x2x2)	16.6 (16.0)
32 (4x4x2)	8.69 (30.5)
64 (4x4x4)	4.55 (59.6)
96 (6x4x4)	3.54 (74.7)
128 (8x4x4)	2.69 (98.4)
192 (8x6x4)	1.84 (144.)

```
192 192 128
  4   2   2
pcube
```

```
#!/bin/sh
#PJM -L “node=1”
#PJM -L “elapse=00:10:00“
#PJM -L “rscgrp=small“
#PJM -
#PJM -o “mg.lst“
#PJM --mpi “proc=16“
mpiexec ./pmesh
```

```
#!/bin/sh
#PJM -L “node=1”
#PJM -L “elapse=00:10:00“
#PJM -L “rscgrp=small“
#PJM -
#PJM -o “test.lst“
#PJM --mpi “proc=16“
mpiexec ./sol0
```

計算例(通信最適化済)@東大 Strong Scaling

- 192×192×128節点(4,718,592節点，4,633,087要素)

core #	sec. (speed-up)
16 (4x2x2)	16.6 (16.0)
32 (4x4x2)	8.69 (30.5)
64 (4x4x4)	4.55 (59.6)
96 (6x4x4)	3.54 (74.7)
128 (8x4x4)	2.69 (98.4)
192 (8x6x4)	1.84 (144.)

```
192 192 128
  4   4   2
pcube
```

```
#!/bin/sh
#PJM -L “node=2”
#PJM -L “elapse=00:10:00“
#PJM -L “rscgrp=small“
#PJM -
#PJM -o “mg.lst“
#PJM --mpi “proc=32“
mpiexec ./pmesh
```

```
#!/bin/sh
#PJM -L “node=2”
#PJM -L “elapse=00:10:00“
#PJM -L “rscgrp=small“
#PJM -
#PJM -o “test.lst“
#PJM --mpi “proc=32“
mpiexec ./sol0
```

計算例(通信最適化済)@東大 Strong Scaling

- 192×192×128節点(4,718,592節点，4,633,087要素)

core #	sec. (speed-up)
16 (4x2x2)	16.6 (16.0)
32 (4x4x2)	8.69 (30.5)
64 (4x4x4)	4.55 (59.6)
96 (6x4x4)	3.54 (74.7)
128 (8x4x4)	2.69 (98.4)
192 (8x6x4)	1.84 (144.)

```
192 192 128
  8   6   4
pcube
```

```
#!/bin/sh
#PJM -L “node=12”
#PJM -L “elapse=00:10:00“
#PJM -L “rscgrp=small“
#PJM -
#PJM -o “mg.lst“
#PJM --mpi “proc=192“
mpiexec ./pmesh
```

```
#!/bin/sh
#PJM -L “node=12”
#PJM -L “elapse=00:10:00“
#PJM -L “rscgrp=small“
#PJM -
#PJM -o “test.lst“
#PJM --mpi “proc=192“
mpiexec ./sol0
```