東京大学情報基盤センター
お試しアカウント付き並列プログラミング講習会「OpenFOAM入門」
2017年5月12日@東京大学情報基盤センター遠隔会議室

OpenFOAM講習

今野　雅
(株式会社OCAEL・東京大学客員研究員)

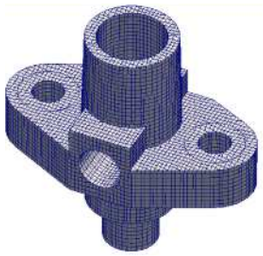
講習会プログラム(予定)

- 12:30-14:00 キャビティ流れ演習I
  - ✓ blockMeshによる格子生成
  - ✓ ParaViewによる格子可視化
  - ✓ icoFoamによる流れ解析
  - ✓ ParaViewによる解析結果可視化
- 14:15-15:45 キャビティ流れ演習II
  - ✓ sampleによる解析結果サンプリング
  - ✓ gnuplotによる解析結果プロット
- 16:00-18:00 キャビティ流れ演習III
  - ✓ 並列計算
  - ✓ プロファイラーの使い方
  - ✓ その他チュートリアルの実行
  - ✓ snappyHexMeshによる格子生成
  - ✓ 演習・質疑

OpenFOAMでの代表的な解析手順

前処理(格子生成など)

格子生成
[blockMesh,
snappyHexMeshなど]



または

格子生成(サードパーティ)
[cfMesh,Salome,gmsh
商用メッシャー等]

必要あれば格子変換
[gmshToFoam等]

解析

初期設定
[setFields等]

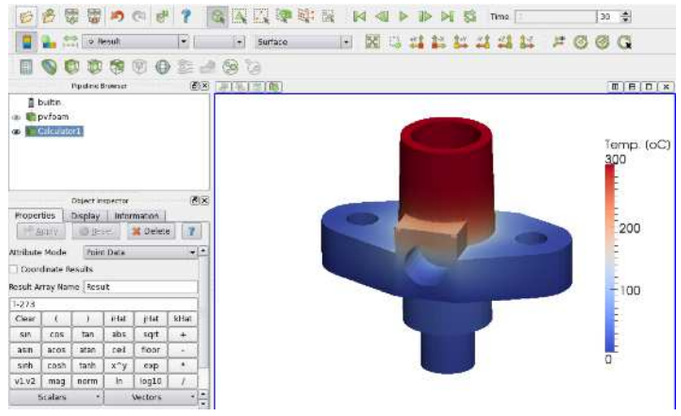
領域分割(並列計算時)
[decomposePar]

解析ソルバ
[icoFoam等]

領域統合(並列計算時)
[reconstructPar]

後処理(可視化など)

可視化(サードパーティ)
[ParaView, Visit,
商用可視化ツール等]



様々な結果後処理
[sample等]

スパコンでのOpenFOAMの代表的な解析手順

ユーザーマシン

4. 格子可視化
[ParaView]

7.可視化
[ParaView]

1. ログイン
[ssh]

3. 格子転送
[scpまたは
rsync]

6. 解析結果転送
[scpまたは
rsync]

ログインノード

2.格子生成
[blockMesh]

5.流体解析
[ソルバ]

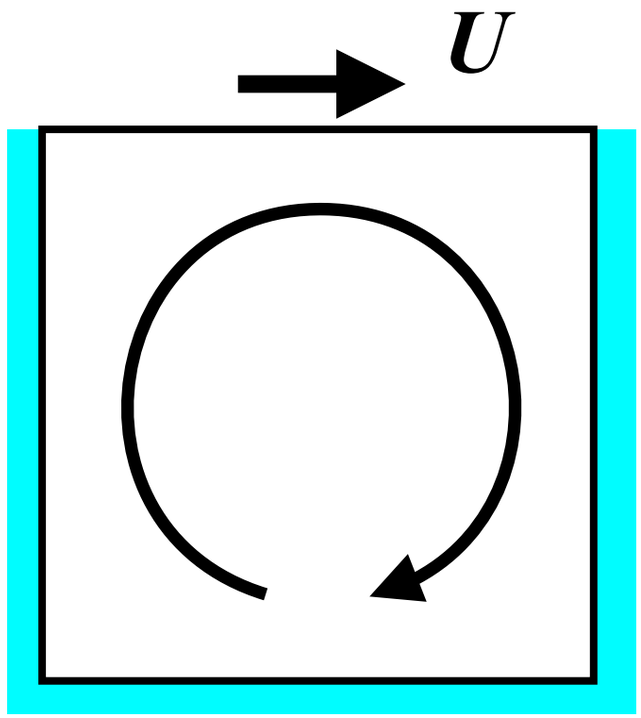
計算ノード

バッチジョブシステムによりジョブが実行される

キャビティ流れ演習I

キャビティ流れとは

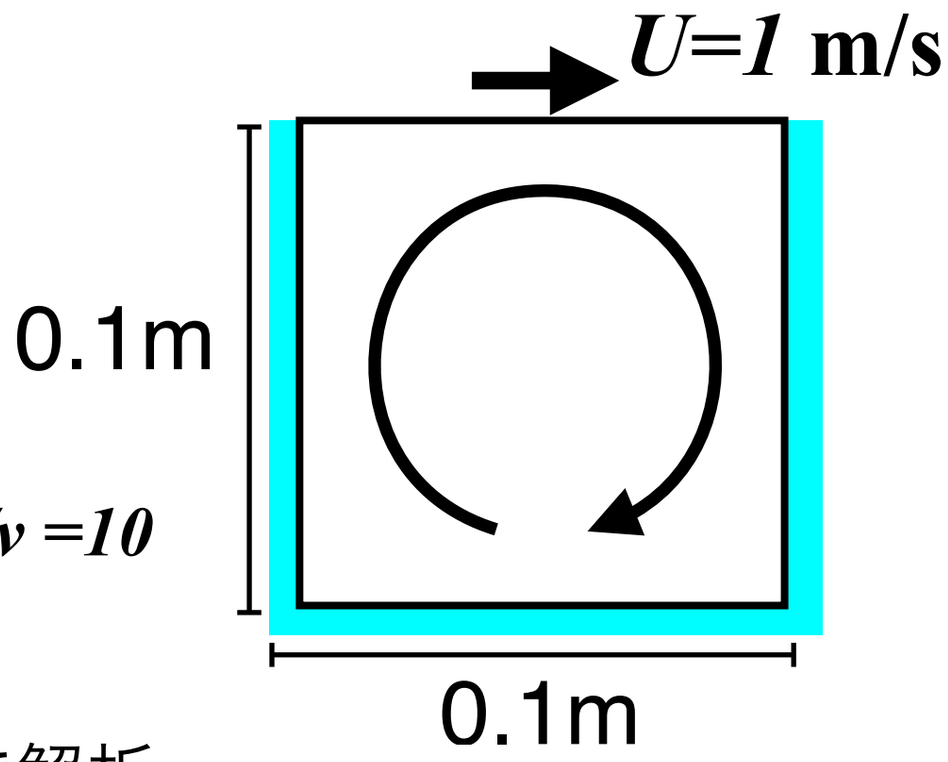
- キャビティ(空洞)の上壁が動き、その摩擦で空洞内の流体が動く流れ場
- 単純な形状と境界条件でCFDでの設定が容易
- レイノルズ数の増加に伴ない，１次循環渦の大きさや中心位置，２次以上の循環渦の有無や大きさなどの様相が変化する
- CFDソフトウェアの基礎的な検証例(ベンチマークテスト)として良く用いられる
- Ghiaらの計算結果との比較が多い



[Ghia 1982] U Ghia, K.N Ghia, C.T Shinl: High-Re solution for incompressible flow using the Navier-Stokes equations and the multigrid method. J. Comput. Phys., 48:387–411, 1982.

キャビティ流れのチュートリアル

- 代表長さ(辺長)：$d=0.1$ m
- 代表速度(上壁の移動速度)：$U=1$ m/s
- 動粘性係数 (=粘性係数/密度)：$\nu=0.01$ m^2/s
- レイノルズ数(慣性力と粘性力の比)：$Re= dU/\nu =10$
- 粘性が強く，乱れがほとんど無い流れ
  - ✓ 非圧縮性非定常流れ解析ソルバicoFoamで解析



非圧縮層流解析ソルバicoFoamの基礎方程式

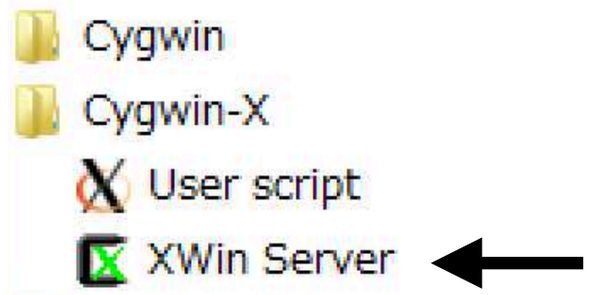
質量保存式　：$\nabla \cdot \boldsymbol{U} = 0$

運動量保存式：$\dfrac{\partial \boldsymbol{U}}{\partial t} + \nabla \cdot (\boldsymbol{U}\boldsymbol{U}) - \nabla \cdot \nu \nabla \boldsymbol{U} = -\nabla p$

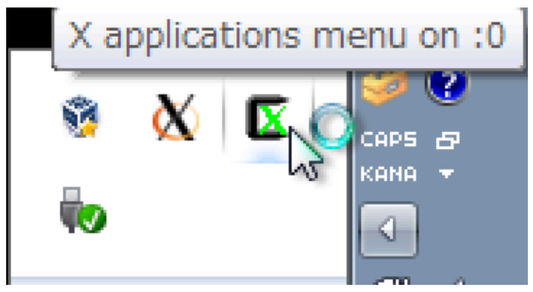
ここで，$\boldsymbol{U}$: 速度ベクトル，p ：流体の密度で割られた圧力，ν： 動粘性係数

X端末xtermの起動(Cygwinのみ)

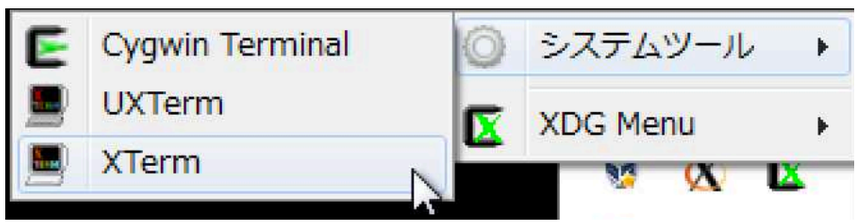
1. スタートメニューからXWin Serverを起動

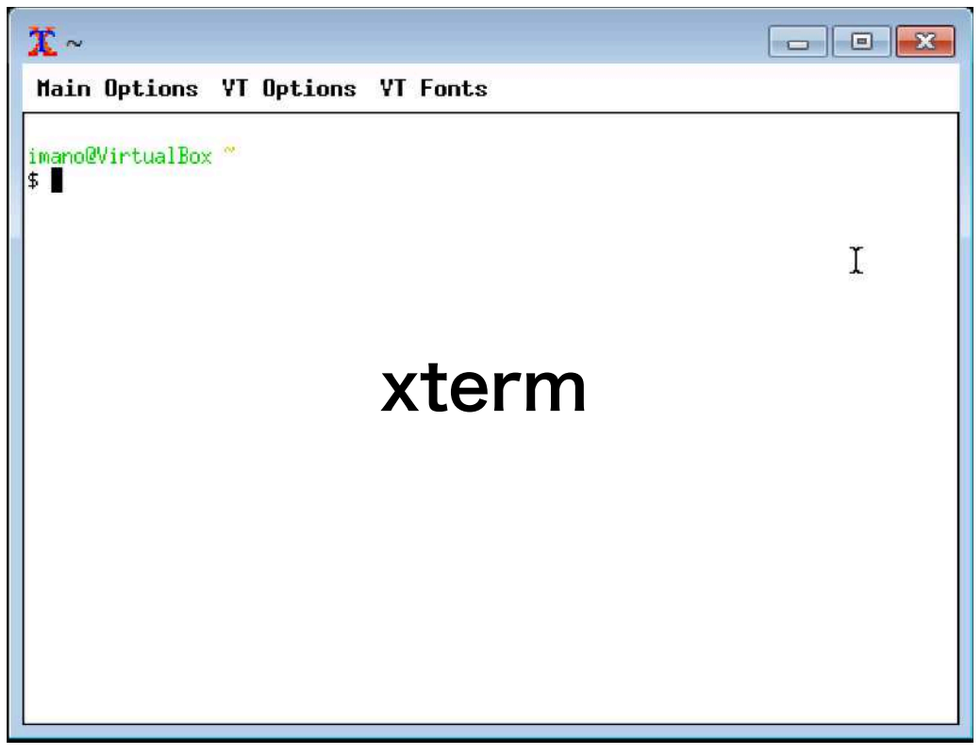


2. ツールバーの「X application menu」を押す



3. システムメニューからXtermを起動する





xterm

Windows 7の場合
他のバージョンでは
異なる可能性があります

ログイン・講習会用ファイルの展開

以降，実線枠の赤字は実行コマンド，点線枠の黒字は実行結果，青字はコメント

Reedbushへのログイン (GUI用にX転送する-Yオプションを付ける)

```
ssh reedbush.cc.u-tokyo.ac.jp -l txxxxx -Y
```

txxxxxは利用番号

```
xev
```

X転送が可能か調べる．Cntr-C(コントロールキーとCで停止)

計算用ディレクトリへの移動

```
cdw
```

講習会用ファイルの展開

```
cp /lustre/gt00/zyyyyy/lecture.tar ./
tar xf lecture.tar
```

zyyyyyは講師の利用番号
講習会以外ではWEBページからコピー

lectureをホームディレクトリへのリンク

```
ln -s $PWD/lecture ~/
```

~/lectureで/lustre/gt00/txxxxx/lecture が指定可能(ファイル同期に便利)

Reedbush-UのOpenFOAM関連module

利用可能なmoduleの表示

```
module avail -l
```

```
- Package ----------------------------+- Versions -+- Last mod. ------
/lustre/app/modulefiles/compiler:
intel-mpi/2017.2.174                        default     2017/03/30 Intelコンパイラ

/lustre/app/modulefiles/mpi:
intel-mpi/2017.2.174                        default     2017/03/30 Intel MPI
mpt/2.14                                                2016/06/28 SGI MPT

/lustre/app/modulefiles/application:
openfoam/1612-mpt                                       2017/01/27 SGI MPT版
openfoam/3.0.1                                          2016/08/31 Intel MPI版
openfoam/3.0.1-mpt                                      2016/08/31 SGI MPT版

/lustre/app/modulefiles/tools:
intel-vtune/17.2.0.499904                               2017/05/08 プロファイラ
```

Intel MPI版のOpenFOAMは通信エラーになる場合有り．今回は1612-mpt(v1612+版)を使用

OpenFOAM v1612+の環境設定

OpenFOAM v1612+ SGI MPT版用のmodule環境の設定

```
module unload intel
```
標準でloadされているintelコンパイラをunload

```
module load intel/17.0.2.174
```
intelコンパイラ17.0.2をload

```
module unload intel-mpi
```
intel-mpiは不要なのでunload

```
module load mpt/2.14
```
mpt/2.14をload

```
module load openfoam/1612-mpt
```
OpenFOAM v1612+をload

OpenFOAMの環境設定

```
. $WM_PROJECT_DIR/etc/bashrc
```

module環境の設定によりOpenFOAMの実行が可能になるが，コマンドラインでの操作に便利なエイリアスやOpenFOAM用の環境変数を定義するためには，上記も必要
($WM_PROJECT_DIR はOpenFOAMのインストール先を示す環境変数)

OpenFOAM環境設定用のエイリアス

参考) OpenFOAMの主なエイリアス (`alias`で確認可能． cdw等も表示される)

- ✓`run` ： ユーザの実行用ディレクトリに行く
- ✓`src` ： ライブラリのソースがあるディレクトリに行く
- ✓`app` ： 標準アプリケーションのソースがあるディレクトリに行く
- ✓`util` ： 標準ユーティリティのソースがあるディレクトリに行く
- ✓`sol` ： 標準ソルバーのソースがあるディレクトリに行く
- ✓`tut` ： チュートリアルディレクトリに行く

OpenFOAM環境設定用エイリアス

```
alias OF1612mpt='\
module unload intel intel-mpi mpt openfoam;\
module load intel/17.0.2.174;\
module unload intel-mpi;\
module load mpt/2.14;\
unset WM_PROJECT_DIR;\
module load openfoam/1612-mpt;\
. $WM_PROJECT_DIR/etc/bashrc\
'
```

左の内容を~/.bashrc に加え，
再ログイン，または以下を実行
. ~/.bashrc

エイリアスによるOpenFOAMの環境設定

```
OF1612mpt
```

キャビティケースのコピー

cavityのケースのコピー

```
cdw
cd lecture/
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity/cavity ./
cd cavity
```

cdではなく cdw であることに注意

$FOAM_TUTORIALS はチュートリアルのディレクトリを示す環境変数

ケースディレクトリのディレクトリ構成を表示(次のどちらも試してみる)

```
tree
```

```
.
├── 0
│   ├── U
│   └── p
├── constant
│   └── transportProperties
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── fvSchemes
    └── fvSolution
```

tree：ディレクトリ構造を樹木状に表示(標準コマンドではない)

```
find | sort
```

```
.
./0
./0/U
./0/p
./constant
./constant/transportProperties
./system
./system/blockMeshDict
./system/controlDict
./system/fvSchemes
./system/fvSolution
```

findとsortは標準コマンド

キャビティケースのディレクトリ構成

```
0/                          //初期条件・境界条件ディレクトリ
   U                        //速度ベクトル
   p                        //圧力

constant/                   //不変な格子・定数・条件を格納するディレクトリ
   transportProperties      //流体物性(物性モデル，動粘性係数，密度など)

constant/polyMesh/          //格子データのディレクトリ

system/                     //解析条件を設定するディレクトリ
   blockMeshDict            //構造格子設定ファイル
   controlDict              //実行制御の設定
   fvSchemes                //離散化スキームの設定
   fvSolution               //時間解法やマトリックスソルバの設定
```

キャビティケースの解析手順

1. 格子生成

`blockMesh`

0/
- U — 速度ベクトル場
- p — 圧力場

constant/
- transportProperties — 流体物性

constant/polyMesh/
- boundary — パッチ設定
- faces,neighbor,owner,points — 格子データ

system/
- blockMeshDict — 構造格子設定ファイル
- controlDict — 実行制御の設定
- fvSchemes — 離散化スキームの設定
- fvSolution — ソルバの設定

解析結果の時刻ディレトリ/
- U, p, phi — 解析結果

この他にも可視化等のポスト処理が通常含まれる

2. ソルバ実行

`icoFoam`

読み込み

読み書き

新規作成

fileName

既存ファイル

fileName

修正ファイル

fileName

作成ファイル

blockMeshによる格子生成

ユーザーマシン

4. 格子可視化
[ParaView]

7.可視化
[ParaView]

1. ログイン
[ssh]

3. 格子転送
[scpまたは
rsync]

6. 解析結果転送
[scpまたは
rsync]

ログインノード

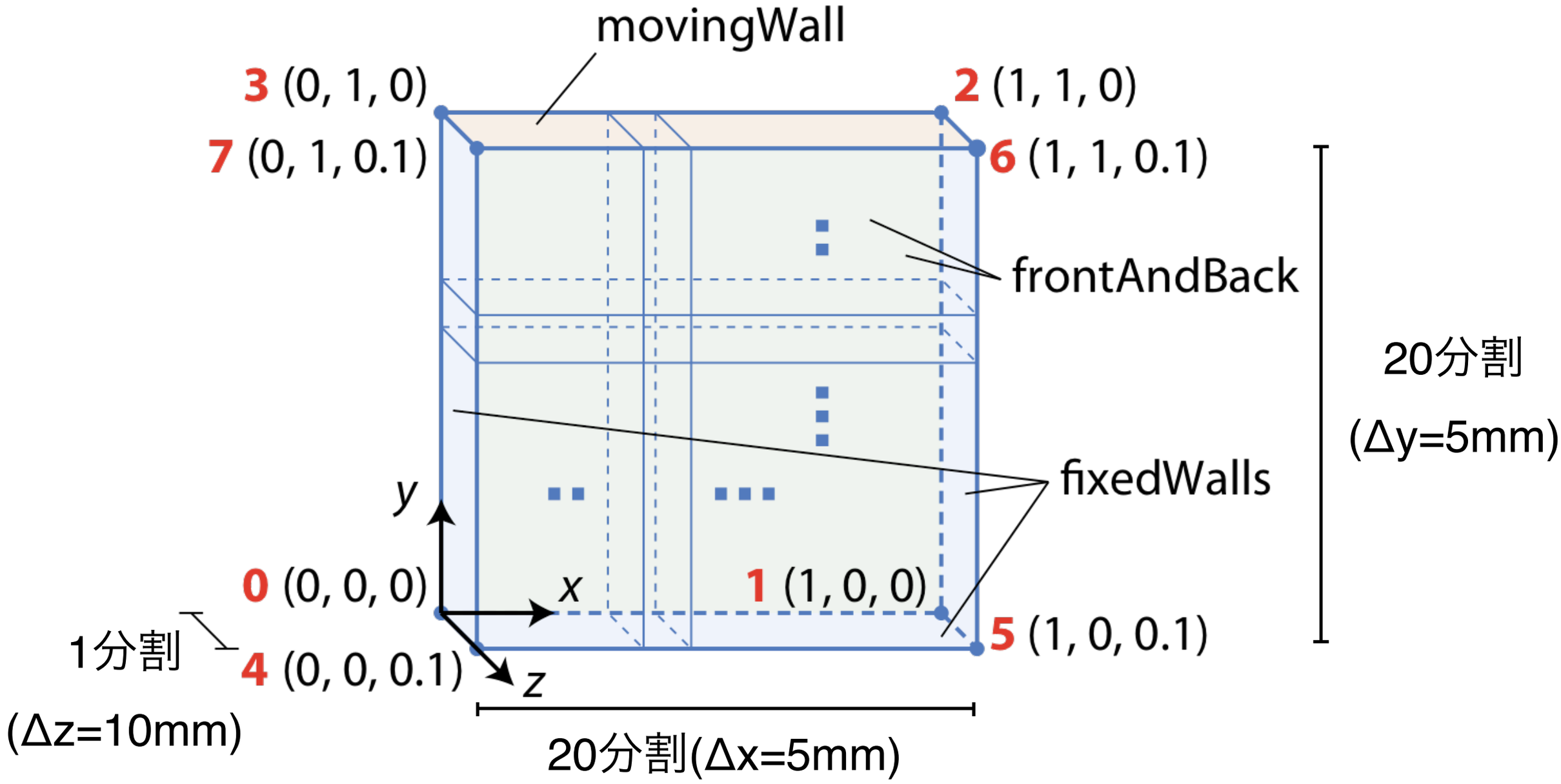
2.格子生成
[blockMesh]

5.流体解析
[ソルバ]

計算ノード

バッチジョブシステムによりジョブが実行される

キャビティケースの格子分割



注)2次元なので、z方向は1分割(幅は任意)

図出典: 大嶋 拓也 (新潟大学)「キャビティ流れの解析、paraFoamの実習」 第一回OpenFOAM講習会

blockMeshDictの確認

```
more system/blockMeshDict
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  plus                                  |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

convertToMeters 0.1;

vertices
(
    (0 0 0)
    (1 0 0)
    (1 1 0)
--More--(55%)
```

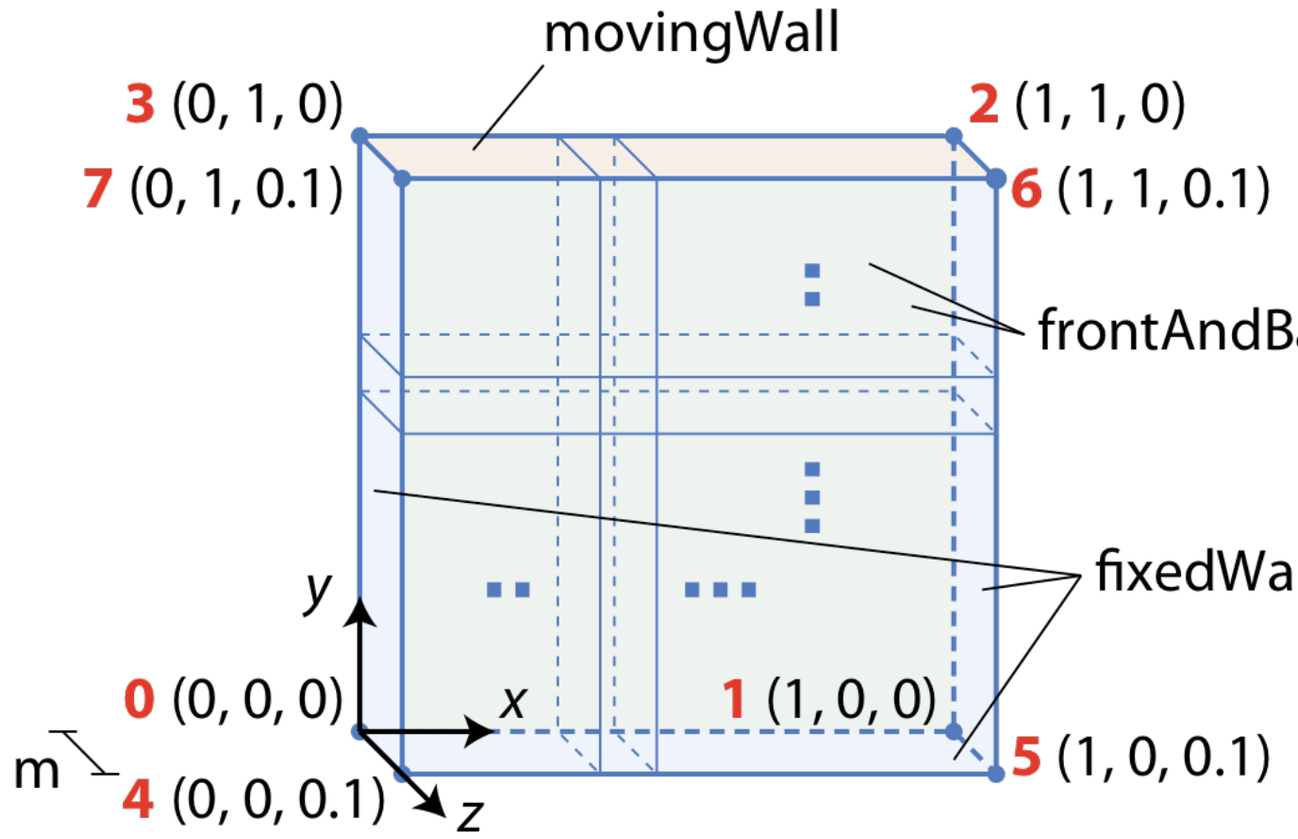
moreコマンドは続ける(英語版ではmore)と表示してキー入力待ちとなる.

主な操作キー　SPC：前，　b：後，　/文字：文字を検索，　.：繰り返し，　h：ヘルプ，　q：終了

設定ファイルblockMeshDict

system/blockMeshDict

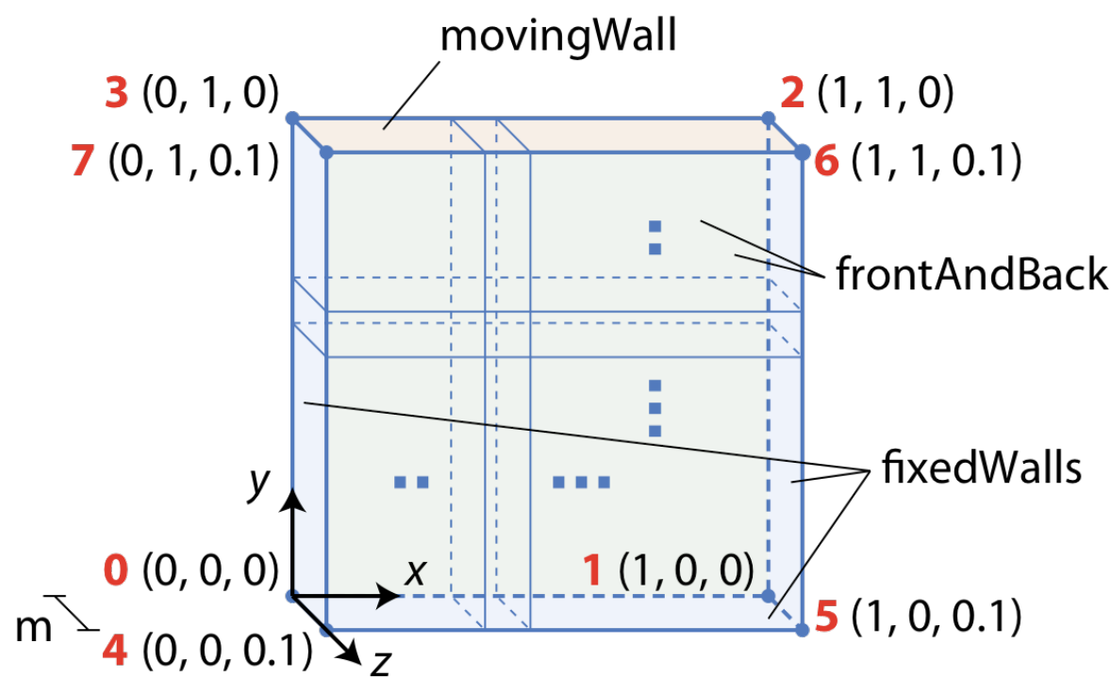
```
convertToMeters 0.1; //メートル単位への変換係数

vertices        //頂点の座標リスト
(
    (0 0 0)    //頂点0
    (1 0 0)    //頂点1
    (1 1 0)    //頂点2
    (0 1 0)    //頂点3
    (0 0 0.1) //頂点4
    (1 0 0.1) //頂点5
    (1 1 0.1) //頂点6
    (0 1 0.1) //頂点7
);
```



設定ファイルblockMeshDict (続き)

system/blockMeshDict

```
blocks   //格子ブロックの定義
(
   hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading  (1 1 1)
//形状  頂点番号リスト        分割数      格子幅は等比級数  格子幅比
//格子幅比は最初と最後の格子の比.  1=等間隔.
);

edges  //辺が曲線の場合に指定
(
);
```

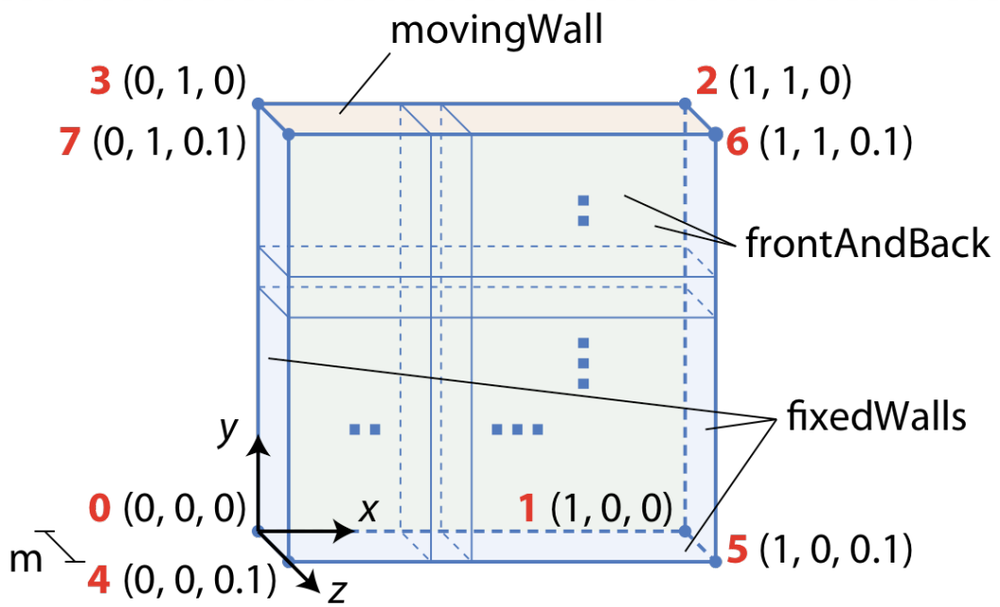


設定ファイルblockMeshDict (続き)

system/blockMeshDict

```
boundary                    //境界の設定
(
    movingWall              //境界の名前
    {
        type wall;          //種別：壁面
        faces               //界面リスト
        (
            (3 7 6 2) //頂点リスト
//内部から見て時計回り.
//ただし，開始頂点は任意.
        );
    }

    fixedWalls              //境界の名前
//中略
```

```
    frontAndBack    //境界の名前
    {
        type empty; //空の境界(2次元)
        faces
        (
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```



blockMesh実行用ジョブスクリプト

```
cp -a ../share/* ./
more blockMesh.sh
```

← ジョブスクリプト，プロットスクリプト，Ghiaらの解析データ，サンプリング設定等の講習用ファイルのコピー

blockMesh.sh

```
#!/bin/bash
#PBS -q u-lecture
#PBS -W group_list=gt00
#PBS -l select=1:ncpus=1:mpiprocs=1:ompthreads=1
#PBS -l walltime=0:10:00

cd $PBS_O_WORKDIR
export MPI_BUFFER_SIZE=20000000
. /etc/profile.d/modules.sh
module purge
module load intel/17.0.2.174
module load mpt/2.14
module load openfoam/1612-mpt

blockMesh >& log.blockMesh
```

キュー名(**演習中はu-tutorialのほうが優先度が高い**)

グループ名

最大実行時間

ジョブディレクトリに移動

select(ノード数)=1
ncpus(ノード当りコア数)=1
mpiprocs(ノード当りMPI数)=1
ompthreads(プロセス当りスレッド数)=1 **フラットMPIでは通常1**

標準で有効なmoduleに依存しないよう，
module purgeで初期化後に必要分をload

blockMeshの実行

blockMeshのジョブ投入

ジョブの投入

```
qsub blockMesh.sh
```

```
JOB_ID.reedbush-pbsadmin0 #JOB_IDは各自異なる
```

ジョブ状態確認 (以降の演習ではジョブ状態の確認を適宜行ってください)

```
rbstat
```

→ ジョブ実行前の場合

```
JOB_ID     JOB_NAME   STATUS  PROJECT  QUEUE          START_DATE       ELAPSE   TOKEN    NODE
JOB_ID     blockMesh. QUEUED  gt00     u-lecture      -                -        -        1
```

→ ジョブ実行中の場合

```
JOB_ID     JOB_NAME   STATUS  PROJECT  QUEUE          START_DATE       ELAPSE   TOKEN    NODE
JOB_ID      blockMesh. RUNNING gt00      u-lecture       -                -        -        1
```

→ 全ジョブ終了の場合

```
JOB_ID     JOB_NAME   STATUS  PROJECT  QUEUE          START_DATE       ELAPSE   TOKEN    NODE
```

ジョブの削除(今回は行わない)

```
qdel JOB_ID
```

生成されたファイルの確認

```
tree
```

```
├── 0
│   ├── U
│   └── p
├── blockMesh.sh
├── blockMesh.sh.e91438
├── blockMesh.sh.o91438
├── constant
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── faces
│   │   ├── neighbour
│   │   ├── owner
│   │   └── points
│   └── transportProperties
├── icoFoam.sh
├── log.blockMesh
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── fvSchemes
    └── fvSolution
```

ジョブの標準出力と標準エラー出力．数字はJOB_ID

生成された格子ファイル

blockMeshのログ

他にも../shareからコピーされたファイルがあるが省略

ジョブの出力ファイルを確認(エラーが出ていないことを確認)

```
more blockMesh.sh.*
```

blockMeshのログ確認

```
more log.blockMesh
```

log.blockMesh

```
Mesh Information
----------------
  boundingBox: (0 0 0) (0.1 0.1 0.01)
  nPoints: 882
  nCells: 400
  nFaces: 1640
  nInternalFaces: 760
----------------
Patches
----------------
  patch 0 (start: 760 size: 20) name: movingWall
  patch 1 (start: 780 size: 60) name: fixedWalls
  patch 2 (start: 840 size: 800) name: frontAndBack

End
```

解析領域範囲

節点数

格子数

界面(フェース)数

内部界面(フェース)数

パッチ(境界面)情報

ParaViewによる格子可視化

ユーザーマシン

4. 格子可視化
[ParaView]

7.可視化
[ParaView]

1. ログイン
[ssh]

3. 格子転送
[scpまたは
rsync]

6. 解析結果転送
[scpまたは
rsync]

ログインノード

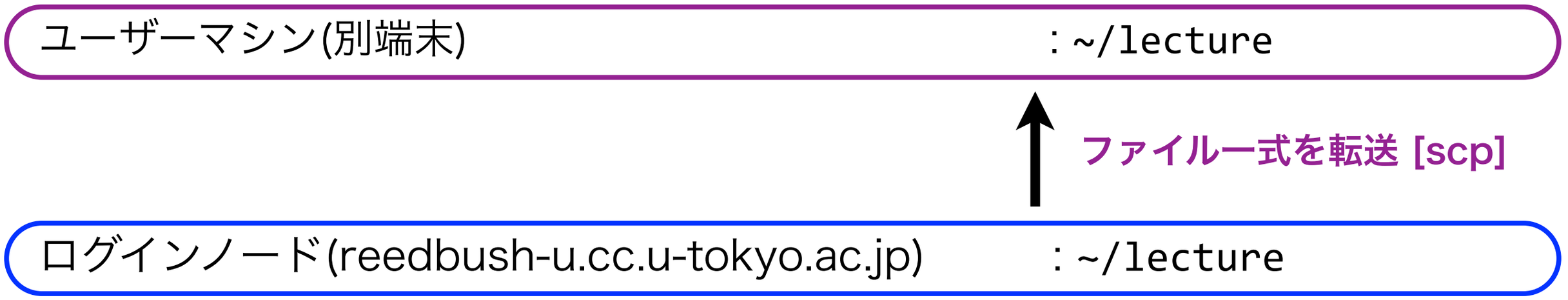
2.格子生成
[blockMesh]

5.流体解析
[ソルバ]

計算ノード

バッチジョブシステムによりジョブが実行される

講習用ファイル一式と格子データの転送



データ転送用にログインしている端末と別の端末を立ちあげる

講習用ファイル一式と作成した格子データの転送(別端末で実行)

```
cd
ls
scp -r txxxxx@reedbush-u.cc.u-tokyo.ac.jp:~/lecture ./
```

もし，既に lecture がある場合には， mv lecture lecture.orig などで移動

txxxxxは利用者番号(xxxxxは数字).

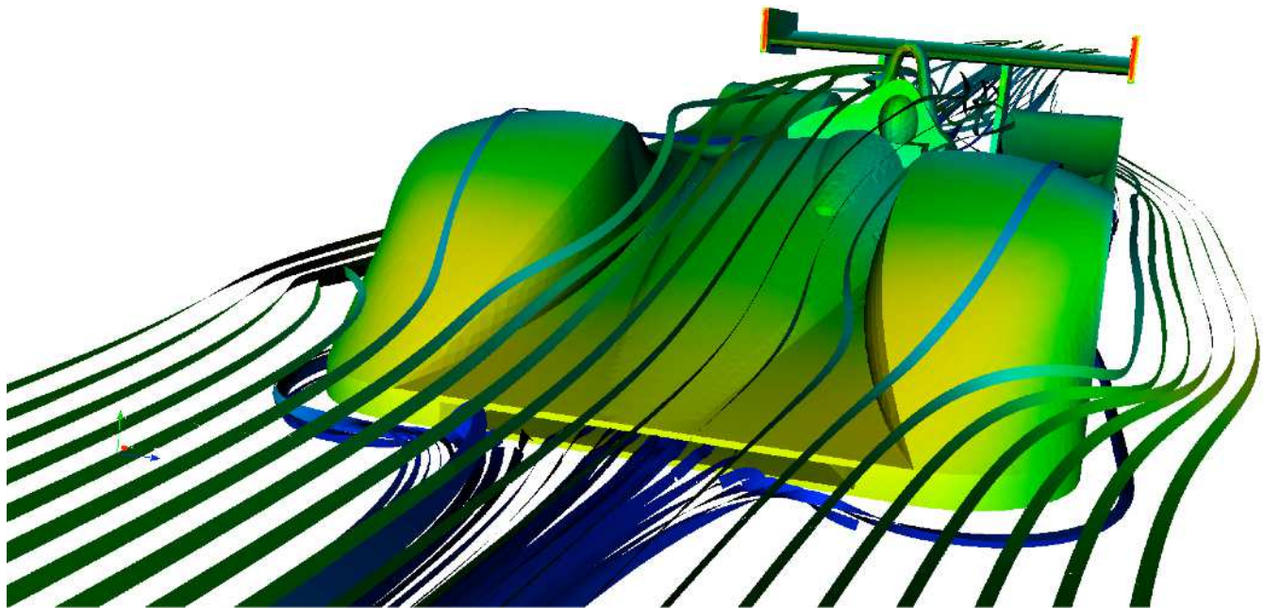
passphraseを聞かれた場合には，登録したものを入力する

キャビティケースに移動(別端末で実行)

```
cd lecture/cavity/
```

ParaViewとは?

- オープンソース、スケーラブル、かつマルチプラットフォームな可視化アプリケーション
- 大容量データセットを処理するための分散型計算手法のサポート
- オープン、柔軟かつ直感的なユーザインターフェイス
- オープンな規格に基づいた拡張性の高いモジュール化構造
- 柔軟な3条項BSDライセンス
- 有償の保守およびサポート



ル・マンのレースカー周りの気流

(ブラジル リオ・デ・ジャネイロ NACAD/COPPE/UFRJ Renato N. Elias)

図出典：Kenneth Moreland et.al : Large Scale Visualization with ParaView, Supercomputing 2014 Tutorial, November 16, 2014

ParaViewによるOpenFOAMデータ可視化

✓ **OpenFOAMの格子や解析結果はParaViewで可視化やデータ解析が可能**

✓ **OpenFOAMデータ読み込み方法**

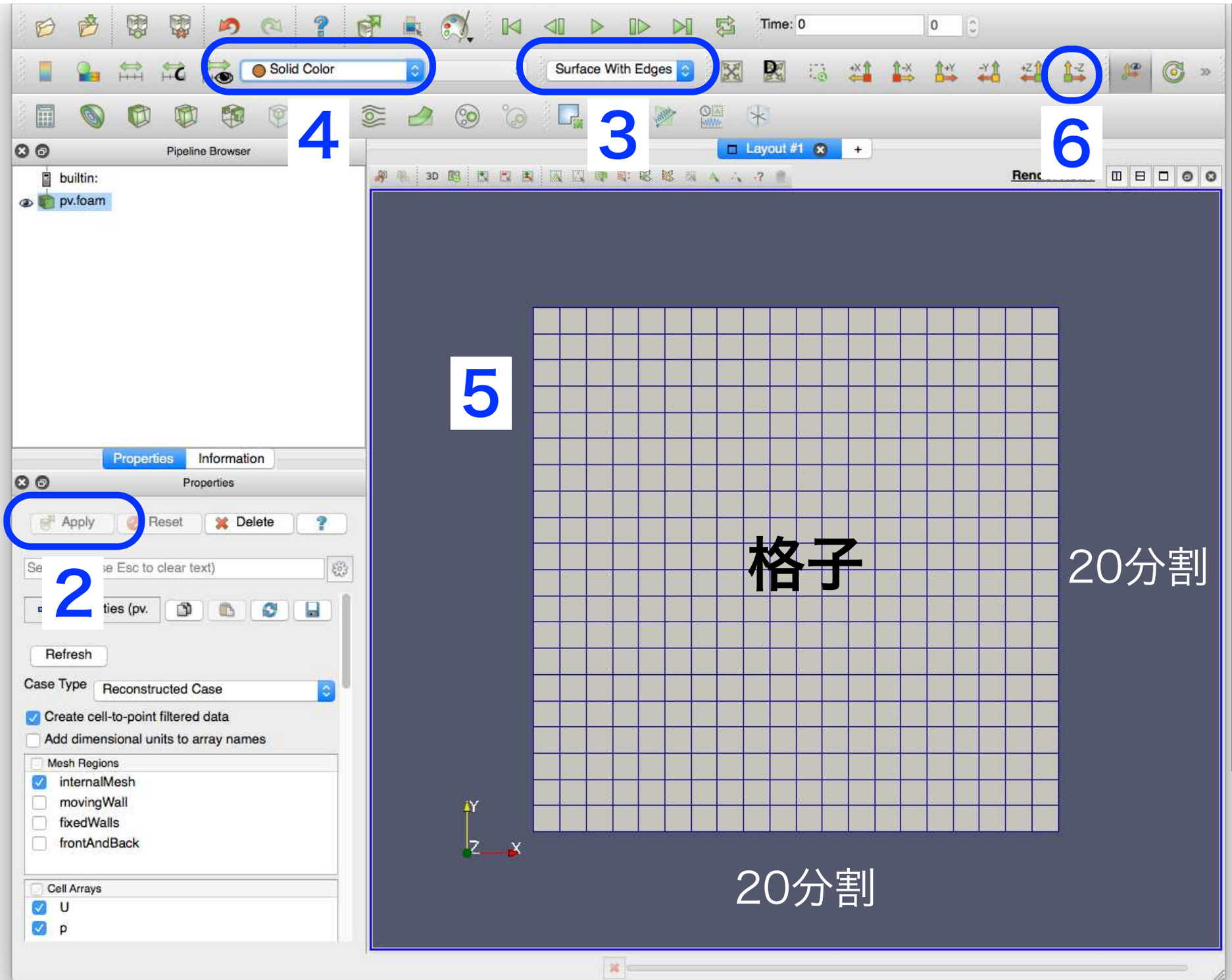
- **OpenFOAM附属のparaFoamコマンド使用 (非推奨だが，後で一度試す)**
  - ParaViewを起動するクライアント側にOpenFOAMの環境が必要
  - ログインノード上で可視化する場合はこの方法
  - ログインノードでの起動は負荷が掛かり，かつX転送は遅いため，非推奨
- **通常のParaView(Ver.3.8以降)を使用　(今回はこの方法)**
  - ParaViewを起動するクライアント側にOpenFOAMの環境は不要
  - スパコンの格子・解析結果をPCに転送して，PC上のParaViewで可視化
  - OpenFOAMのデータを読むには拡張子が.foamのダミーファイルが必要

ParaView用ダミーファイル(.foam)の作成(別端末で実行)

```
touch pv.foam
```

touchで空ファイルが作成される．拡張子以外の名前は任意

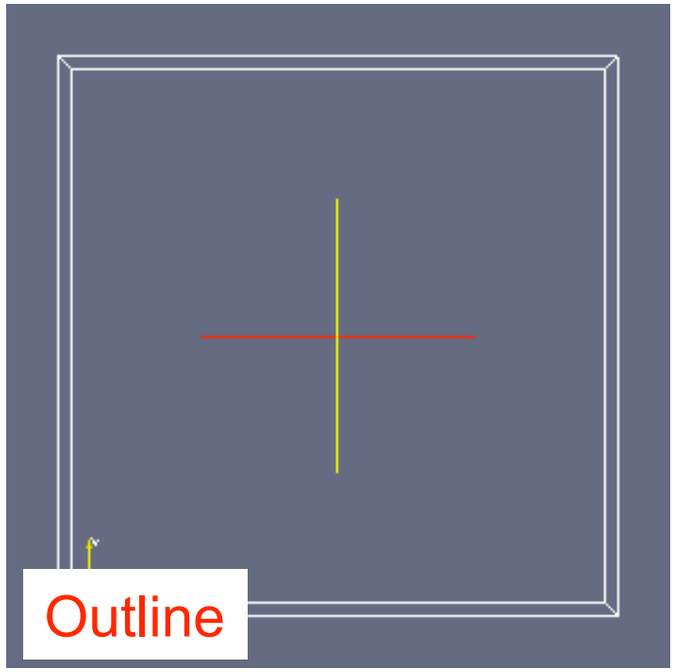
ParaViewによる格子の可視化

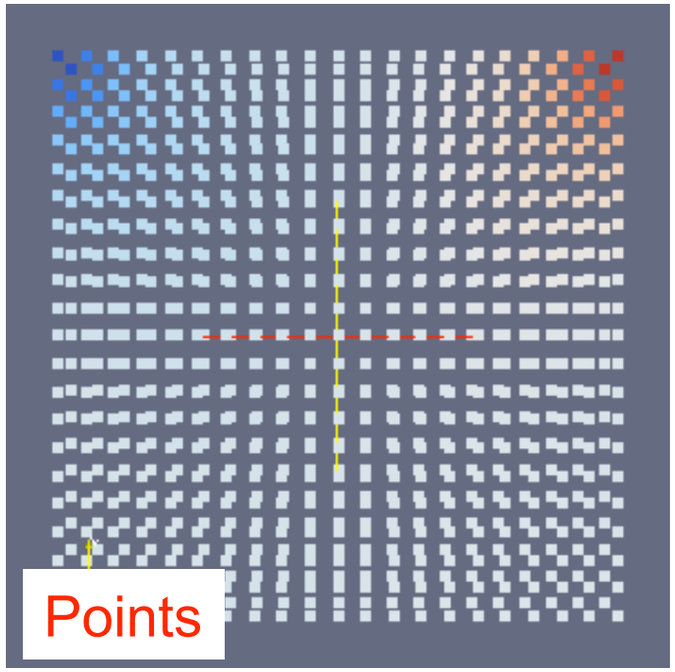


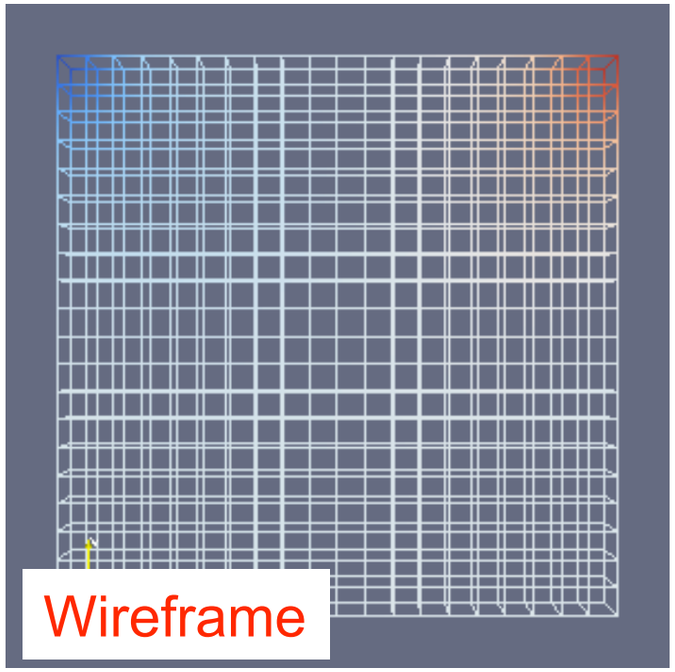
1. Fileメニュー/Open/cavityのディレクトリのpv.foamを開く
2. Apply
3. Representation/Surface With Edges 選択
4. Coloring/Solid Color 選択
5. マウスで適当に動す
6. 法線を”-Z”方向にして，リセット

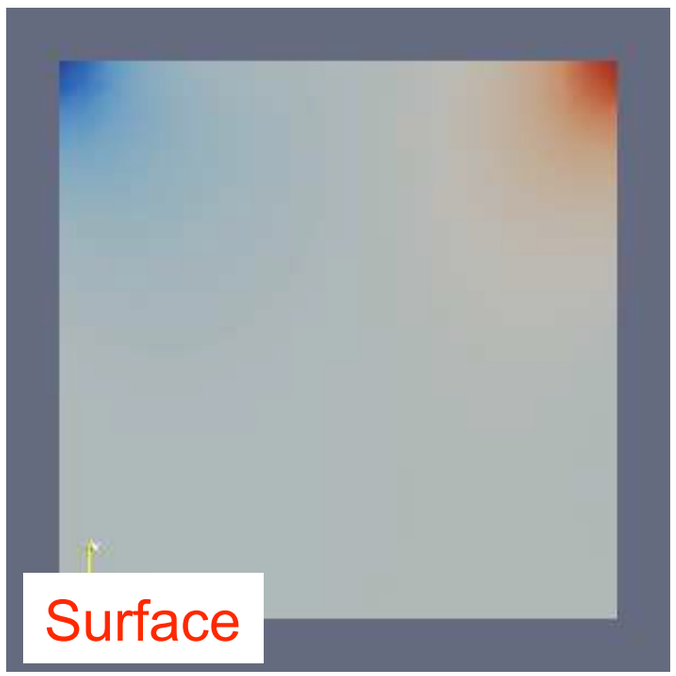
ログインノードで
paraFoamを実行し，
2以下を試してみる

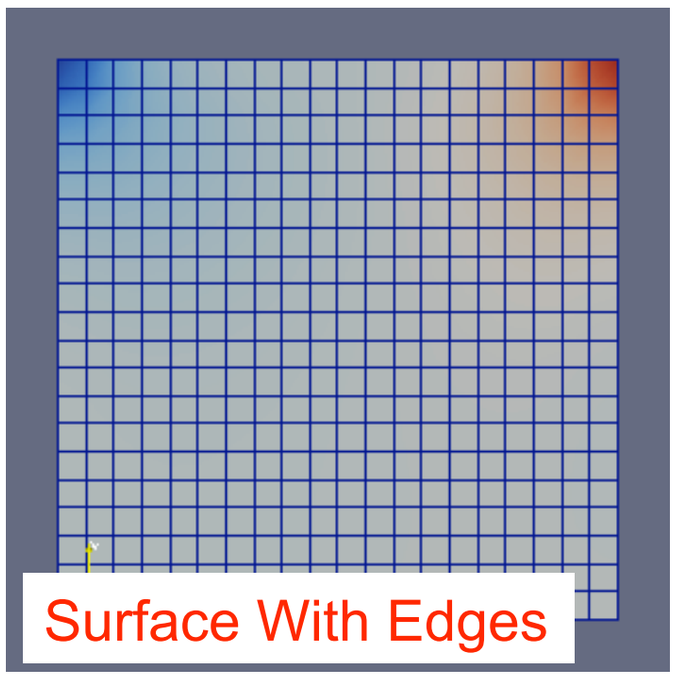
ParaViewの表示方法(Representation)

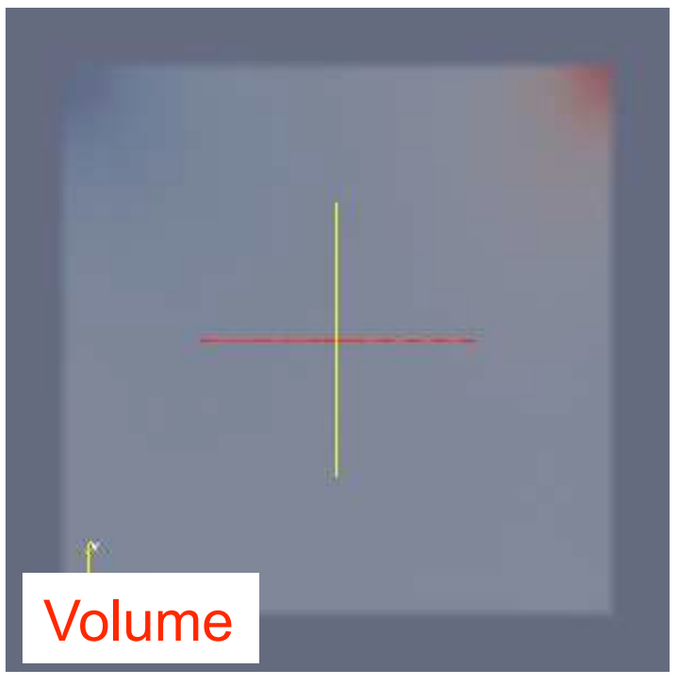












(図引用元: 大嶋 拓也 (新潟大学)「キャビティ流れの解析、paraFoamの実習」 第一回OpenFOAM講習会)

icoFoamによる流れ解析

ユーザーマシン

4. 格子可視化
[ParaView]

7.可視化
[ParaView]

1. ログイン
[ssh]

3. 格子転送
[scpまたは
rsync]

6. 解析結果転送
[scpまたは
rsync]

ログインノード

2.格子生成
[blockMesh]

5.流体解析
[icoFoam]

計算ノード

バッチジョブシステムによりジョブが実行される

初期条件・境界条件設定(速度)

0/ 初期条件・境界条件ディレクトリ

U 速度ベクトル

ファイルの確認

```
more 0/U
```

0/U

```
dimensions  [ 0    1   -1   0    0        0    0   ];
#単位の次元      質量  長さ  時間  温度  物質量     電流  光度
#SI単位での例   [kg] [m] [s]  [K]  [kgmol] [A]  [cd]
#(長さ[m])・(時間[s])-1 → 速度[m/s]

internalField    uniform (0 0 0);
#内部の場          一様分布   0ベクトル(=速度0)
```

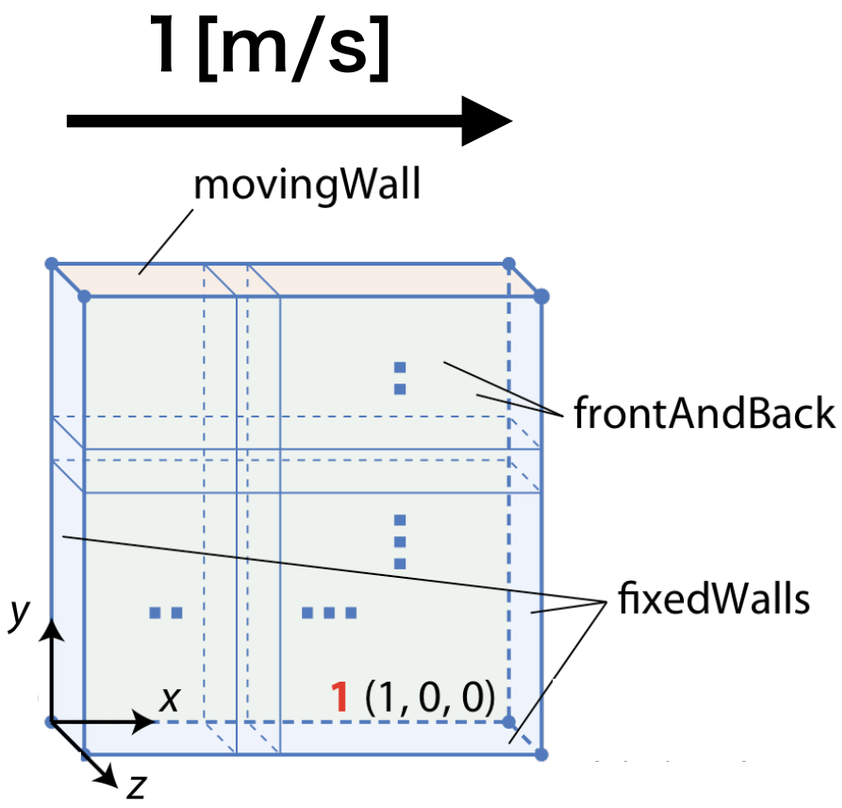
各演算では左辺・右辺での単位一致の検査がなされる

→カスタマイズ時に非物理的な演算の実装をチェックできる

初期条件・境界条件設定(速度)

0/U

```
boundaryField                    #境界条件
{
  movingWall                     #移動壁
  {
    type   fixedValue;           #値固定
    value uniform (1 0 0); #(1,0,0)で一様(移動壁)
  }
  fixedWalls                     #固定壁
  {
    type   noslip;               #すべりなし壁
 }
  frontAndBack
  {
    type   empty;                #2次元なので空
  }
}
```



初期条件・境界条件設定(圧力)

0/ 初期条件・境界条件ディレクトリ

p 圧力

ファイルの確認

```
more 0/p
```

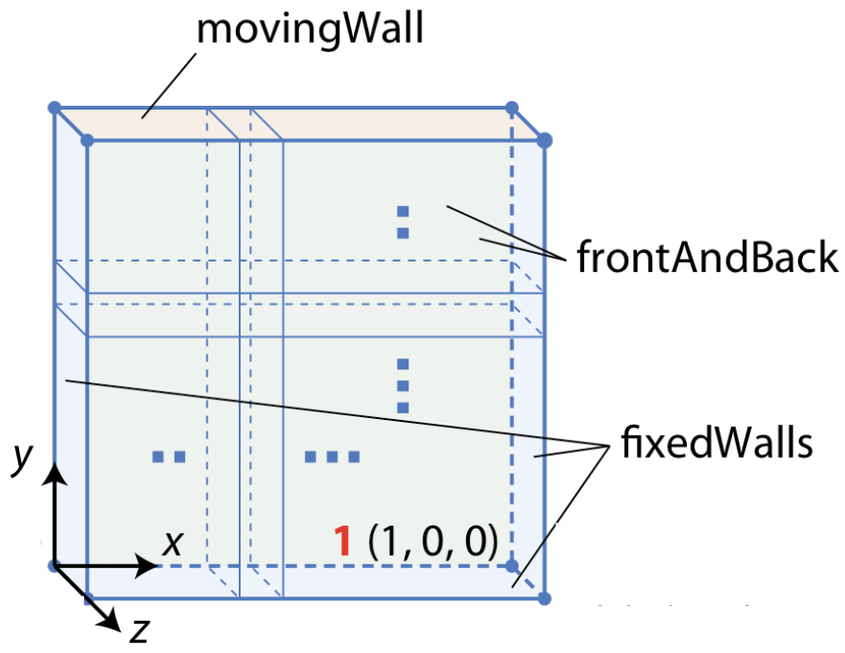
0/p

```
dimensions   [ 0    2   -2   0    0        0    0   ];
#単位の次元      質量  長さ  時間  温度  物質量     電流  光度
#SI単位での例   [kg] [m] [s]  [K]  [kgmol] [A]  [cd]
#OpenFOAMの非圧縮性ソルバでは、圧力は密度で割っている
#(質量)・(長さ)・(時間)-2 / ((質量)・(長さ)-3) = (長さ)2 ・(時間)-2
#        圧力の次元          密度の次元              pの次元

internalField    uniform  0;
#内部の場          一様分布   相対圧力0(非圧縮性流体では相対圧を解く)
```

初期条件・境界条件設定

0/p

```
boundaryField                #境界条件
{
  movingWall                 #移動壁
  {
    type zeroGradient; #非圧縮性浮力無し流れでの壁では，境界の法線方向勾配0
  }

  fixedWalls                 #固定壁
  {
    type zeroGradient;
  }

  frontAndBack
  {
    type empty;              #2次元なので空
  }
}
```



流体物性の設定

constant/ **不変な格子・定数・条件を格納するディレクトリ**

transportProperties 流体物性(物性モデル，動粘性係数，密度など)

ファイルの確認

```
more constant/transportProperties
```

constant/transportProperties

```
nu              [ 0 2 -1 0 0 0 0 ] 0.01; //動粘性係数nu
//変数名            単位[m²/s]          値
```

実行条件等の設定

```
system/                    //解析条件を設定するディレクトリ
  controlDict              //実行制御の設定
  fvSchemes                //離散化スキームの設定
  fvSolution               //時間解法やマトリックスソルバの設定
```

system/fvSchemes（主な行のみ）

```
ddtSchemes //時間項離散化スキーム
{
 default Euler; //Euler法
//default: 明示的な指定が無い場合の設定
}
gradSchemes //勾配項離散化スキーム
{
  default Gauss linear; //ガウス積分・線形
}
divSchemes //発散項・移流項離散化スキーム
{
  div(phi,U) Gauss linear;
//div(phi,U): 速度Uの移流項(phiは流束)
}
```

system/fvSolution（主な行のみ）

```
solvers                          //線形ソルバ
{
  p                              //圧力
  {
    solver           PCG;    //ソルバ
    preconditioner DIC;    //前処理
    tolerance        1e-06; //収束判定閾値
  }
}
PISO //PISO法(圧力・速度連成手法の一種)の設定
{
  nCorrectors     2;       //PISO反復回数
}
```

実行条件等の設定 (続き)

system/controlDict

```
application         icoFoam;    //ソルバー名
startFrom           startTime;  //解析開始の設定法(他にlatestTime等)
startTime           0;          //解析の開始時刻 [s]
stopAt              endTime;    //解析終了の設定法(他にnextWrite等)
endTime             0.5;        //解析の終了時刻 [s]
deltaT              0.005;      //時間刻み [s]
writeControl        timeStep;   //解析結果書き出しの決定法
writeInterval       20;         //書き出す間隔(20time step=0.1s毎)
writeFormat         ascii;      //データファイルのフォーマット(ascii, binary)
writePrecision      6;          //データファイルの有効桁(上記がasciiの場合)
writeCompression    off;        //データファイルの圧縮(off, on)
timeFormat          general;    //時刻ディレクトリのフォーマット
timePrecision       6;          //時刻ディレクトリのフォーマット有効桁
runTimeModifiable true;         //各時間ステップで設定ファイルを再読み込みするか
```

残差プロット用の設定変更

実行制御の設定ファイルの編集

```
vi system/controlDict
```

emacs，nano，geditなどの使い慣れたエディタを使用して編集する

system/controlDict

```
functions
{
#includeFunc residuals
}
```

Fが大文字である事に注意

上記の内容を加えることにより，ソルバ実行時に，線形ソルバーで解かれる変数(ここでは，Ux，Uy:速度，p:圧力)の線形一次方程式の初期残差が，各時刻ステップ毎(定常計算の場合は反復毎)に出力されるので，プロット可能となる

postProcessing/residuals/開始時刻ステップ/residuals.dat

```
# Residuals
# Time                Ux                  Uy                  p
0.005                 1.000000e+00        0.000000e+00        1.000000e+00
0.01                  1.606860e-01        2.608280e-01        4.289250e-01
```

icoFoam用ジョブスクリプトとジョブ投入

icoFoam.sh (逐次ジョブ用スクリプト)

```
#!/bin/bash
#PBS -q u-lecture
#PBS -W group_list=gt00
#PBS -l select=1:ncpus=1:mpiprocs=1:ompthreads=1
#PBS -l walltime=0:10:00

cd $PBS_O_WORKDIR
export MPI_BUFFER_SIZE=20000000
. /etc/profile.d/modules.sh
module purge
module load intel/17.0.2.174
module load mpt/2.14
module load openfoam/1612-mot

icoFoam >& log.icoFoam
```

キュー名(演習中はu-tutorialのほうが優先度が高い)

icoFoamを実行(blockMesh.shとの変更点はここのみ)

ジョブの投入

```
qsub icoFoam.sh
```

生成ファイルの確認

ファイルの確認

```
tree
```

新規に生成されたもの

```
├── 0.1    解析結果の時刻ディレクトリ(0.1, 0.2, 0.3. 0.4, 0.5)
│   ├── U    速度ベクトル
│   ├── p    圧力
│   ├── phi    流束(フラックス)
│   └── uniform    時刻ディレクトリの情報ファイルを収めたディレクトリ
│       └── time    時刻や時間刻み等の情報
├── 0.2    以降の時刻ディレクトリは省略
├── icoFoam.sh.e91439    ジョブの標準エラーファイル
├── icoFoam.sh.o91439    ジョブの標準出力ファイル
└── log.icoFoam    ソルバのログ
```

流体解析のログ

ログの確認

```
more log.icoFoam
```

log.icoFoam

```
Build  : v1612+                 #ビルドバージョン(実行環境依存)
Exec   : icoFoam                #実行コマンド
Date   : Jan 1 1970             #開始日時(実行環境依存)
Time   : 00:00:00               #開始時刻(実行環境依存)
Host   : “nxxx"                 #ホスト名(実行環境依存)
PID    : xxxxx                  #プロセスID(実行環境依存)
Case   : /xx/lecture/cavity     #ケースディレクトリ(実行環境依存)
nProcs : 1                      #使用プロセッサ数(今回非並列計算なので1)
#以下は実行時環境設定(実行環境依存)
fileModificationChecking : Monitoring run-time modified files using
timeStampMaster (fileModificationSkew 10)
allowSystemOperations : Allowing user-supplied system call operations
```

流体解析のログ(続き)

log.icoFoam

```
Starting time loop                    #時間(反復)ループの開始

Time = 0.005                          #時刻

Courant Number mean: 0 max: 0 #クーラン数空間平均値，最大値(1を超えないようにする)
smoothSolver:  Solving for Ux, Initial residual = 1, Final residual =
8.90511e-06, No Iterations 19
#Ux(速度のx方向成分)の離散方程式についての線形ソルバのログ(Uyについても同様)
#smoothSolver: 線型ソルバ(Gauss-Seidel法)
#Initial residual: 初期残差
#Final residual: 最終残差
#No Iterations: 反復回数

DICPCG:  Solving for p, Initial residual = 1, Final residual = 7.55423e-07, No
Iterations 35
#p(圧力)の離散方程式についての線形ソルバのログ
#DICPCG: 線型ソルバ．PCG(前処理付き共役勾配法)+前処理DIC
```

流体解析のログ(続き)

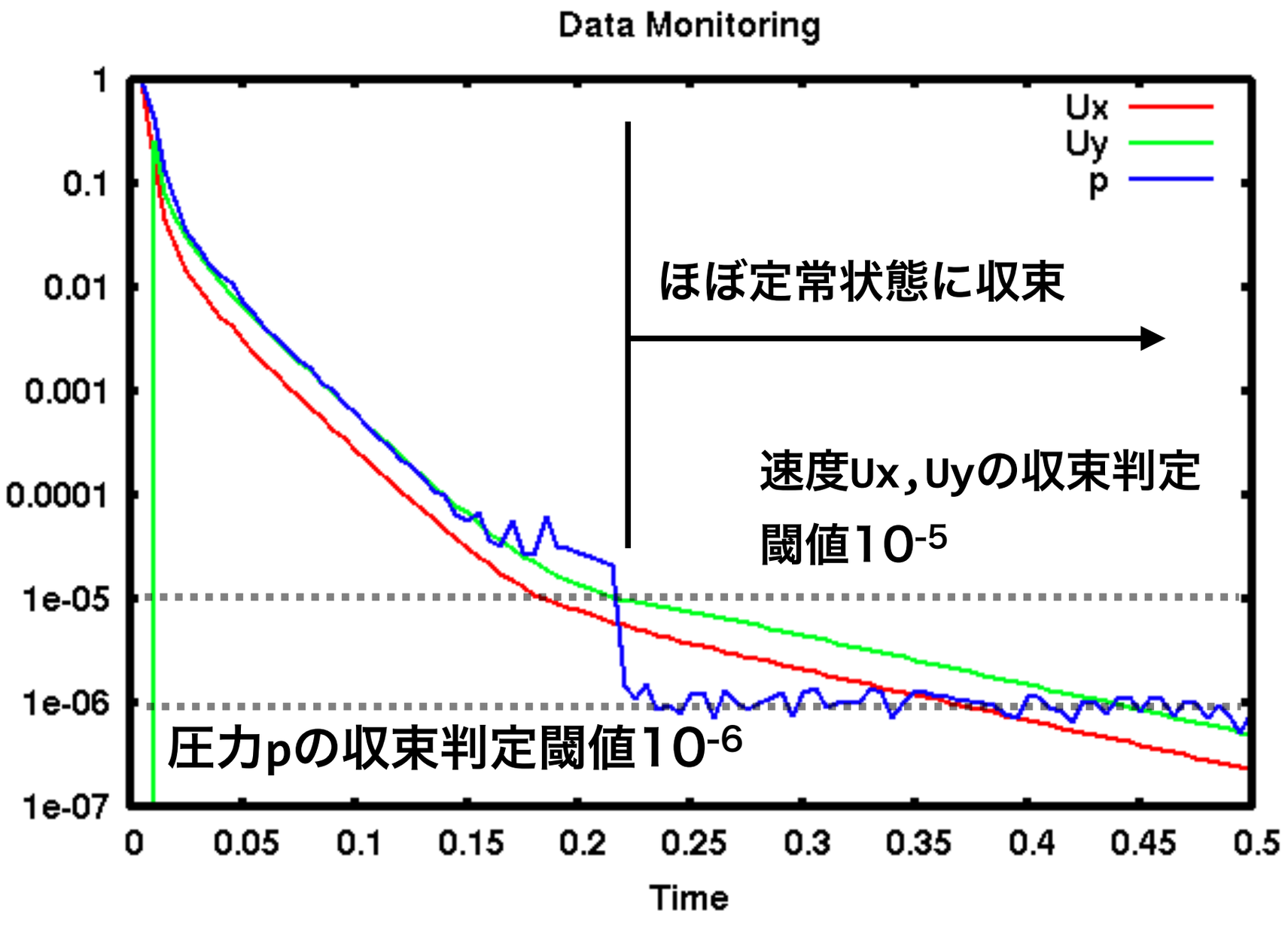
log.icoFoam

```
time step continuity errors : sum local = 5.03808e-09, global =
-7.94093e-21, cumulative = -7.94093e-21
#連続の式の誤差
#sum local   : 誤差絶対値の格子体積重み付け平均
#global      : 誤差(符号あり)の格子体積重み付け平均
#cumulative : globalの累積

ExecutionTime = 0.22 s  ClockTime = 4 s
#ExecutionTime: 計算のみに要した時間(0.01秒単位)
#ClockTime: ファイルI/Oなどシステム時間も含めた実際の経過時間(秒単位)

Time = 0.01 #時刻。以下同様
#中略

End
#OpenFOAMのアプリケーションは，正常終了の場合，通常最後にEndを出力する.
```

初期残差のプロット

初期残差のプロット (-r 1で1秒間隔で更新)

```
foamMonitor -r 1 -l postProcessing/residuals/0/residuals.dat &
```



- ソルバー実行中は自動的にグラフが更新される.
- 残差の時系列データ(postProcessing/residuals/0/residuals.dat)が60秒間変更無い場合, 自動的に終了する.
- または, Ctrl+Cで終了できる.

ParaViewによる解析結果可視化

ユーザーマシン

4. 格子可視化
[ParaView]

7.可視化
[ParaView]

1. ログイン
[ssh]

3. 格子転送
[scpまたは
rsync]

6. 解析結果転送
[scpまたは
rsync]

ログインノード

2.格子生成
[blockMesh]

5.流体解析
[ソルバ]

計算ノード

バッチジョブシステムによりジョブが実行される

解析結果の転送

ユーザーマシン(別端末) : ~/lecture/cavity/

解析結果転送 [rsync]

ログインノード(reedbush-u.cc.u-tokyo.ac.jp) : ~/lecture/cavity/

解析結果の転送(別端末で実行)

```
rsync -auv txxxxx@reedbush-u.cc.u-tokyo.ac.jp:~/lecture/cavity/ ./
```

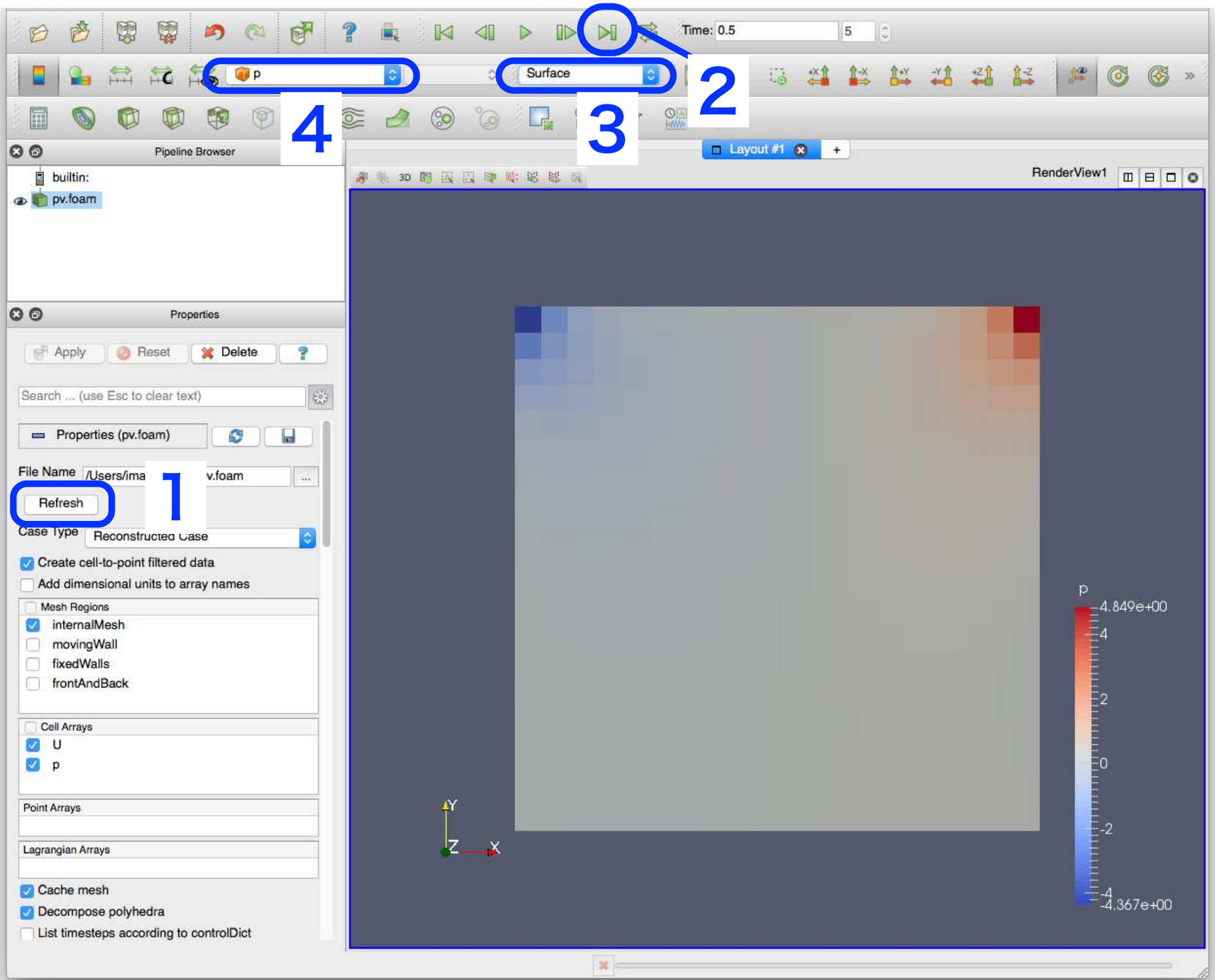
転送元と転送先どちらもに/(スラッシュ)を付ける ↑ ↑

a=archive(ディレクトリを再帰的かつ，ファイル情報を保持したまま転送),
u=update(新規・更新されたもののみ転送)，v=verbose(転送情報を表示)

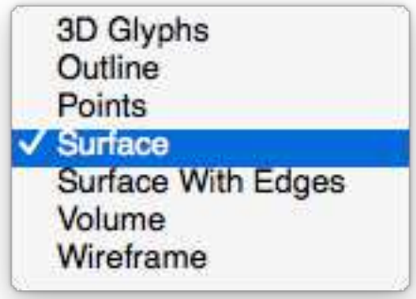
```
receiving file list ... done
./
icoFoam.sh.eJOB_ID
icoFoam.sh.oJOB_ID
log.icoFoam
:
```

新規作成・更新されたicoFoamの解析結果やログファイルのみ転送されるので転送量が少ない(rsyncを使う理由)

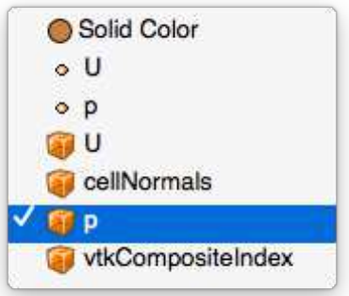
ParaViewによる圧力の可視化



1. Refresh(更新)
2. Last Frame
3. Representation /Surface

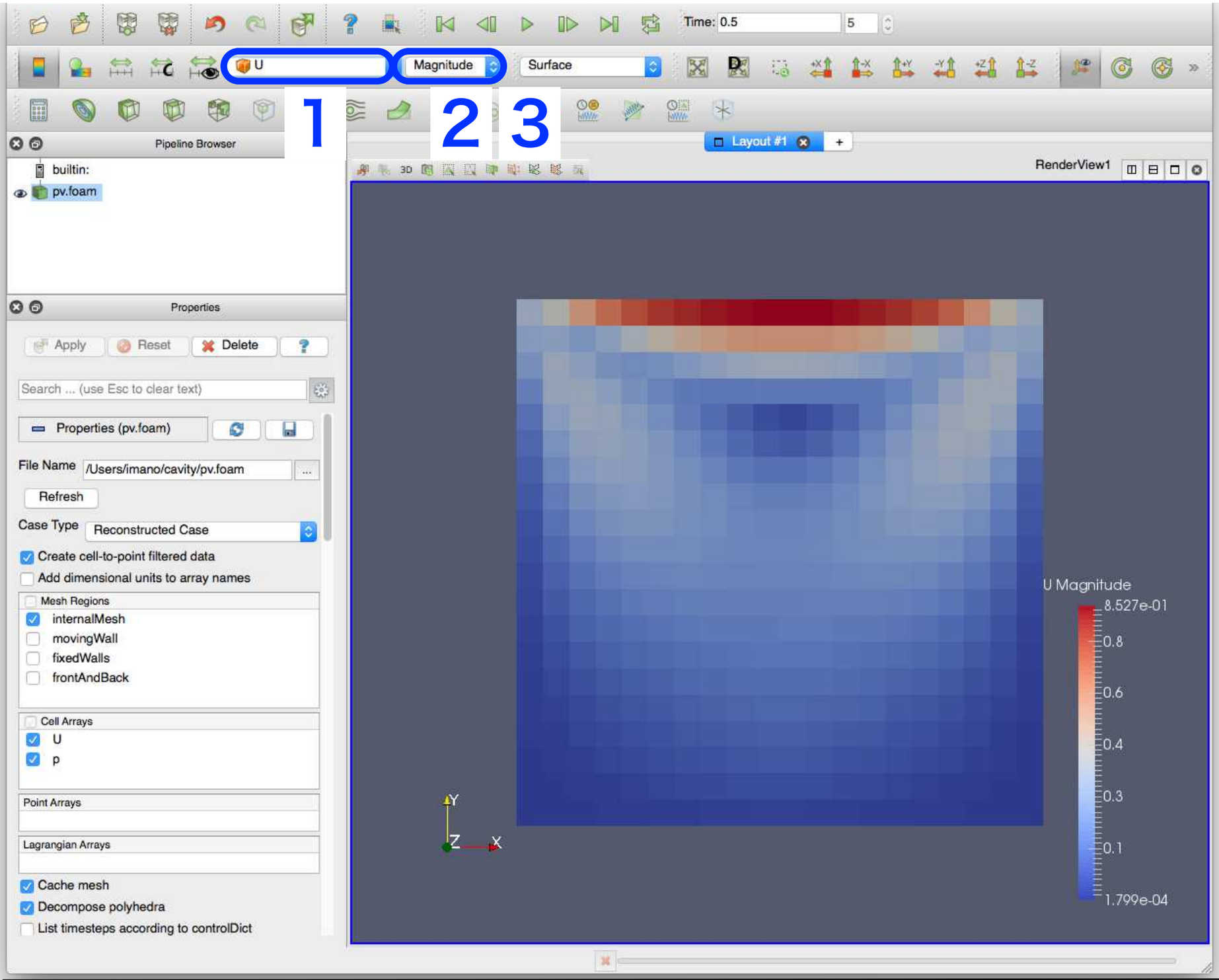


4. Coloring/□p

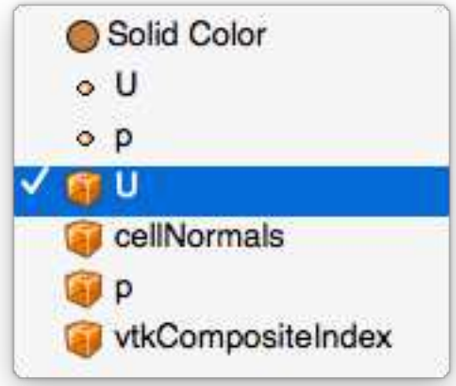


(□は格子の値そのままで補間無し、○は補間有り=スムージング)

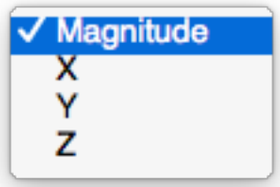
ParaViewによる風速の可視化



1. Coloring/□U

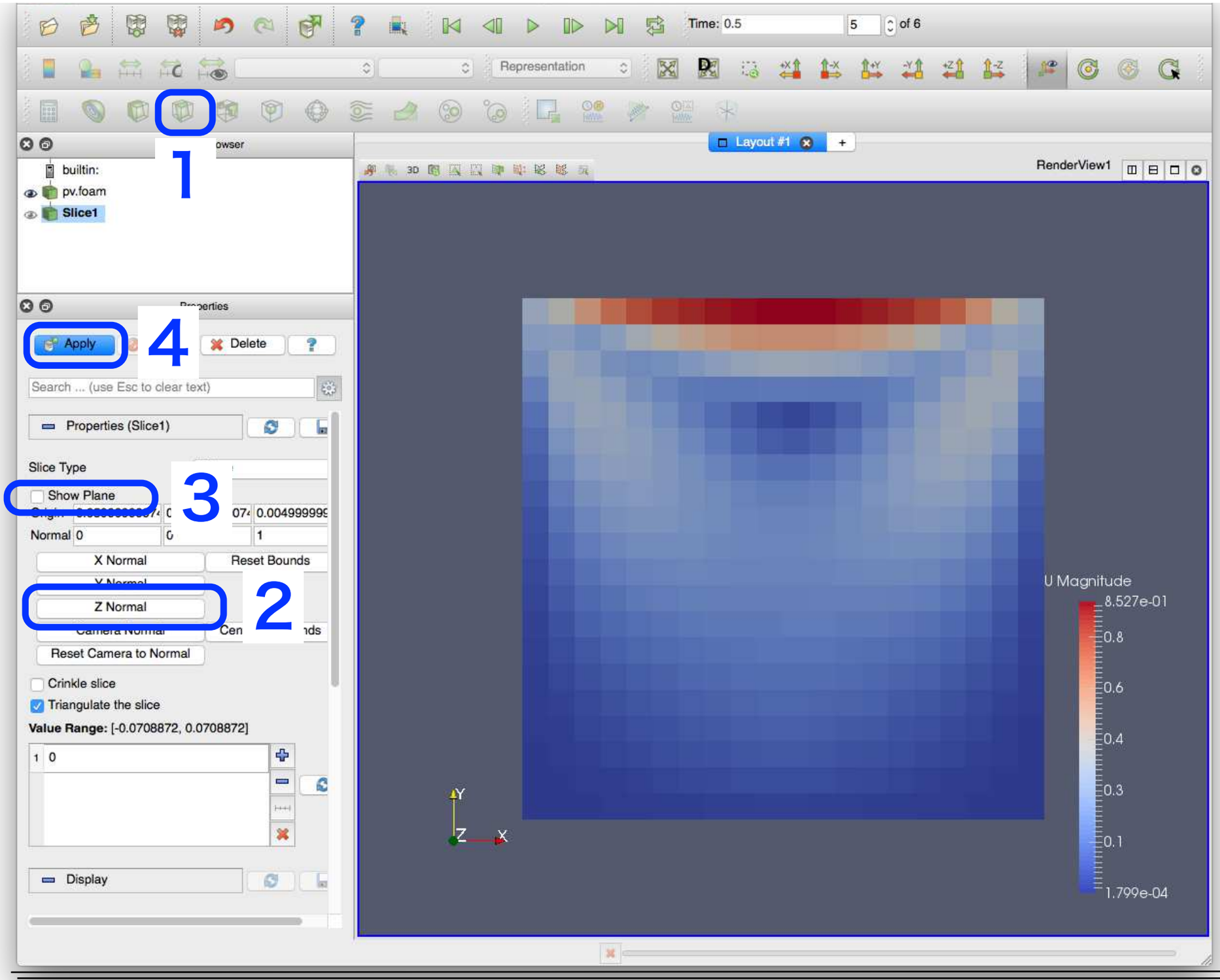


2. Magnitudeを X，Y，Zに変更することで，各風速成分が可視化できる.



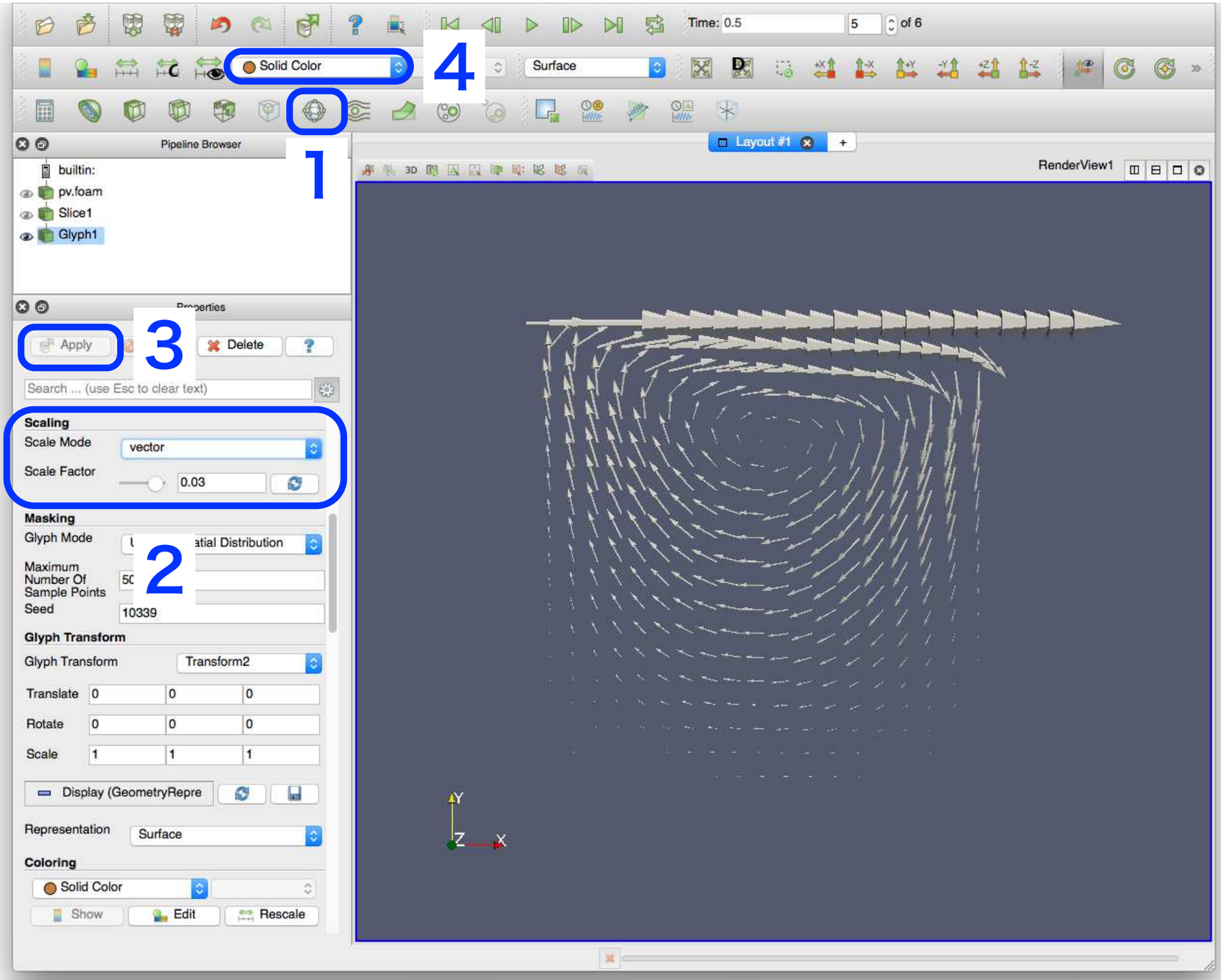
3. Magnitudeに戻す.

ParaViewによる風速ベクトルの可視化



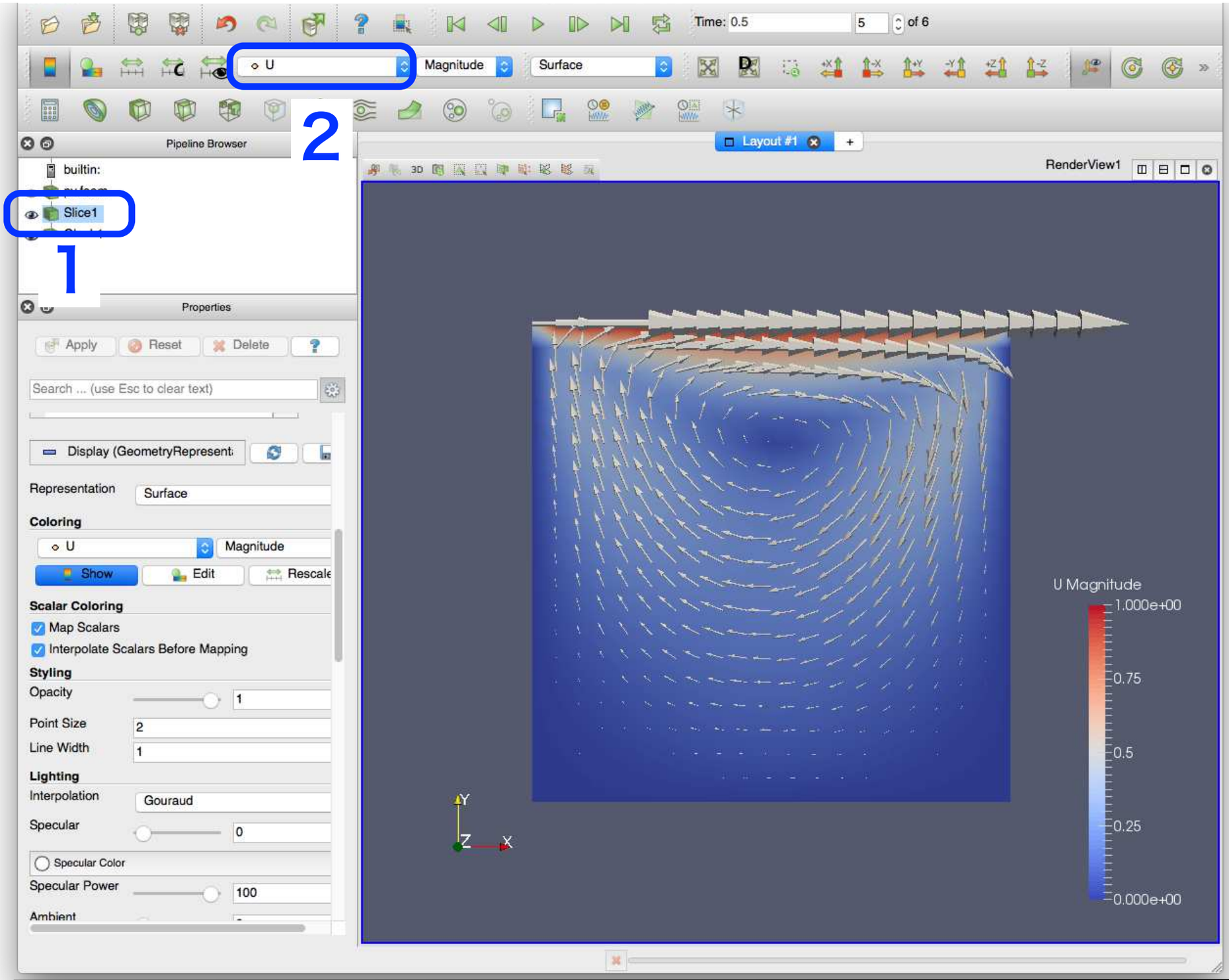
1. Sliceフィルタ
2. Z Normal
3. Show Planeを非選択
4. Apply

ParaViewによる風速ベクトルの可視化



1. Glyphフィルタ
2. Scaling/Scale Mode/vectorを選択．Scale Factor: 0.03
3. Apply
4. Coloring/Solid Color を選択

ParaViewによる風速ベクトルの可視化

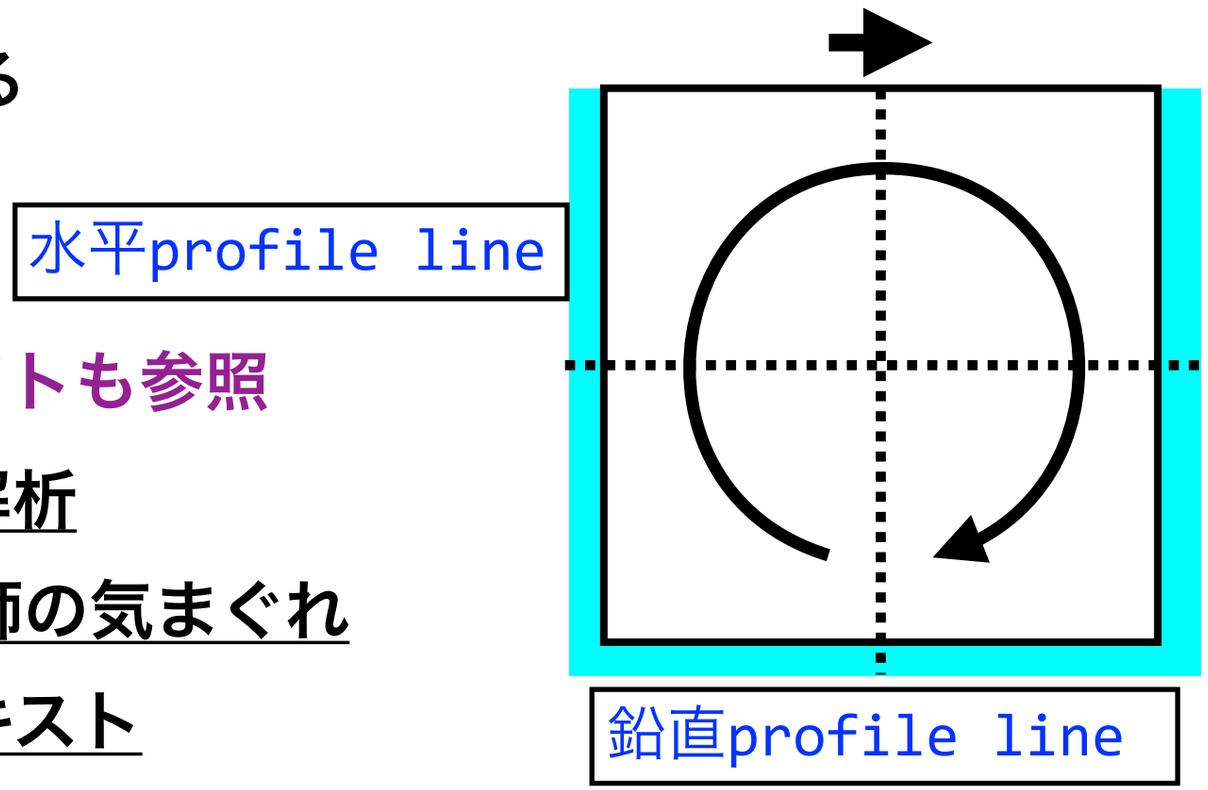


1. Sliceフィルタがもし非表示になっていた場合(注)は表示(目のマークをチェック)
2. Coloring/・Uを選択
3. Quitメニュー/Quit ParaViewでParaView終了

(注) ParaViewのバージョンによって，挙動が異なる.

キャビティ流れ演習II

- OpenFOAMの解析の検証(Validation)を行うため，[Ghia 1982]によるキャビティ中心のprofile lineでの速度の計算結果との比較プロットを作成する
- 比較プロットを行う場合，計算結果のサンプリングが必要
- サンプリングは `postProcess`ユーティリティで行う(OpenFOAM-4.0, v1612+より前のバージョンでは`sample`ユーティリティ)
- プロットは各自手慣れなツールを用いれば良いが，ここではReedbushにインストール済みである`gnuplot`を用いる



OpenFOAMでの検証では以下のサイトも参照

- PENGUINITIS - キャビティ流れ解析
- オープンCAE勉強会@関西 - 「講師の気まぐれOpenFOAMもくもく講習会」テキスト

Ghiaらによるcavity解析の再現

- Ghiaらによる解析では，以下のようにOpenFOAMのチュートリアルのcavityケースと設定が異なるため，cavityケースをコピー後修正する
  - ✓ キャビティの辺長：0.1 → 1
  - ✓ 辺の分割数：20 → 128 or 256 (ただし，最初は20のままにする)
  - ✓ Re数：10 → 100, 400, 1000, 3200, 5000, 7500, 10000

cavityケースの複製 (Re=100)

```
cdw
cd lecture/
cp -a cavity cavityRe100
cd cavityRe100/
foamCleanTutorials
echo *.sh.*
rm *.sh.*
```

ログファイルやpostProcessingなどを消去

バッチジョブの出力ファイルを表示

echoをrmに変更して実行．*と.の間に空白を入れ，通常ファイルを全消去するミスを防ぐ．少ファイルにはrm -iで確認しながら消去も有用

Ghiaらの解析に合わせた設定変更(続き)

格子生成設定ファイルの編集

```
vi system/blockMeshDict
```

```
convertToMeters ~~0.1~~1; //メートル単位への変換係数

vertices        //頂点の座標リスト
(
    (0 0 0)    //頂点0
    (1 0 0)    //頂点1 (変換係数を1にしたので, 辺長が1に修正される)
```

実行制御の設定ファイルの編集

```
vi system/controlDict
```

```
endTime         ~~0.5~~30;      //解析の終了時刻 [s]
deltaT          0.005;          //時間刻み [s]
writeControl    timeStep;       //解析結果書き出しの決定法
writeInterval   ~~20~~1000;     //書き出す間隔(1000time step=5s毎)
```

Re数の増加により定常になるまでに必要な時間が長くなる. 終了時刻を30sにし, 結果を書き出す間隔を5s毎にする. ただし, 高Re数では多くの積分時間が必要.

格子生成と流体解析のジョブ投入

serial.sh (逐次ジョブ用スクリプト)

```
#!/bin/bash
#PBS -q u-lecture
#PBS -W group_list=gt00
#PBS -l select=1:ncpus=1:mpiprocs=1:ompthreads=1
#PBS -l walltime=0:10:00

cd $PBS_O_WORKDIR
export MPI_BUFFER_SIZE=20000000
. /etc/profile.d/modules.sh
module purge
module load intel/17.0.2.174
module load mpt/2.14
module load openfoam/1612-mot

blockMesh >& log.blockMesh
icoFoam >& log.icoFoam
```

キュー名(演習中はu-tutorialのほうが優先度が高い)

blockMeshとicoFoamを続けて実行

ジョブの投入

```
qsub serial.sh
```

ジョブの確認・残差モニター・強制終了

ジョブ状態確認

```
rbstat
```

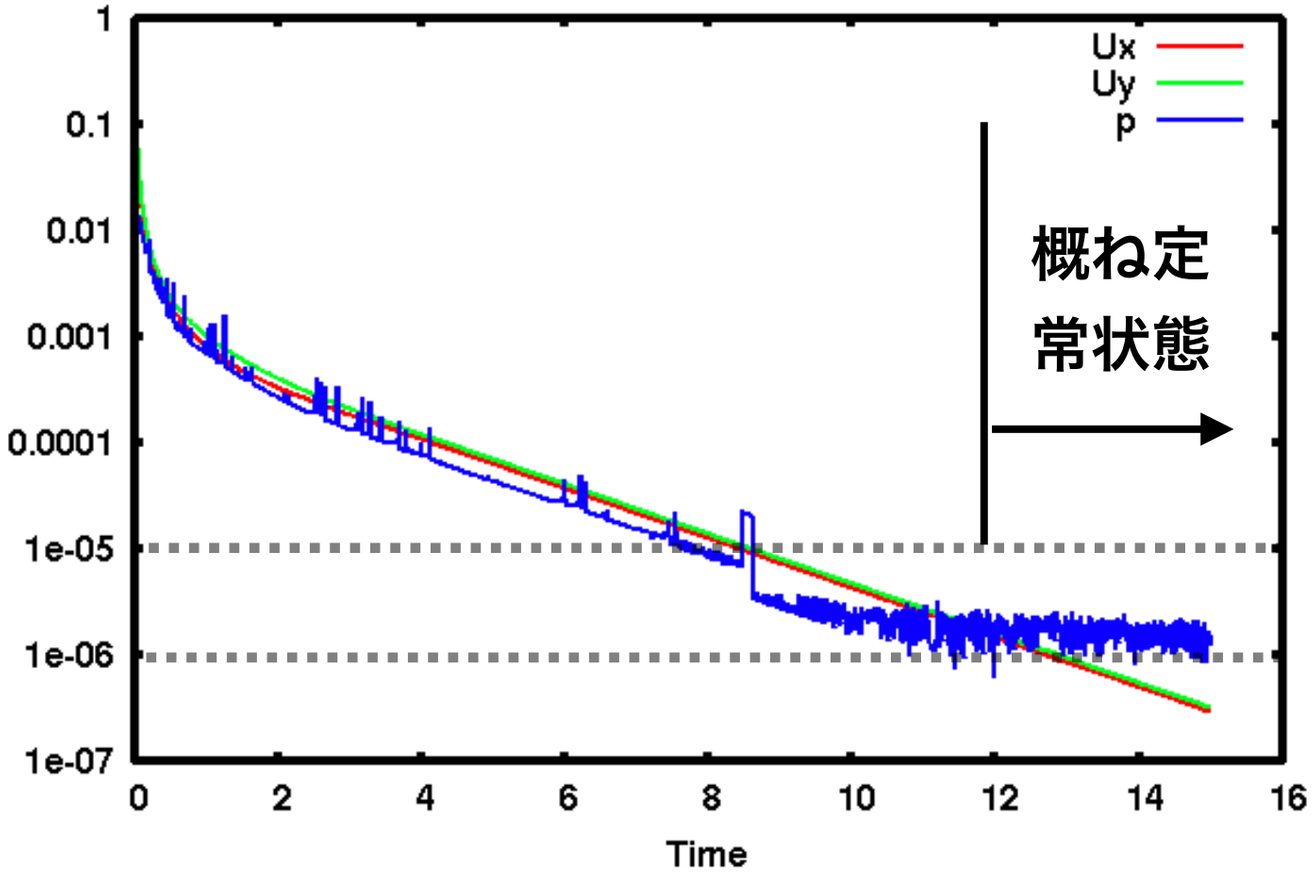
ジョブ実行中になったら初期残差をモニター

```
foamMonitor -r 1 -l postProcessing/residuals/0/residuals.dat &
```

X転送が不可な場合には，ログにおける線形ソルバーの反復回数をモニター

```
tail -f log.icoFoam
```

tailの終了はCtrl-C



速度と圧力の残差が収束判定以下になったり，線形ソルバーの反復回数(No Iteration)がほぼ0になったら概ね定常.
終了時刻まで長く待てない場合には，ジョブを削除して途中で終了しても良いし，途中でサンプリングとプロットをしても良い

```
qdel JOB_ID
```

計算結果のサンプリング

サンプリング設定ファイルの確認(既にcavityケースでコピーしている)

```
more system/sample
```

system/sample

```
type sets;
setFormat raw;
interpolationScheme cellPointFace;
sets
(
  lineX1
  {
    type  midPointAndFace;
    axis  x;
    start (-0.001 0.5 0.05);
    end   ( 1.001 0.5 0.05);
  }
  lineY1
  lineX1と同様なので略
);
fields( U );
```

出力マット，raw:テキスト形式．他にvtk,csv,gnuplot等

補間方法，cellPointFace: 格子中心，節点，界面での値から補間
cell：格子中心のみ(格子内一定)
cellPoint:格子中心，節点から補間

集合サンプリングの定義

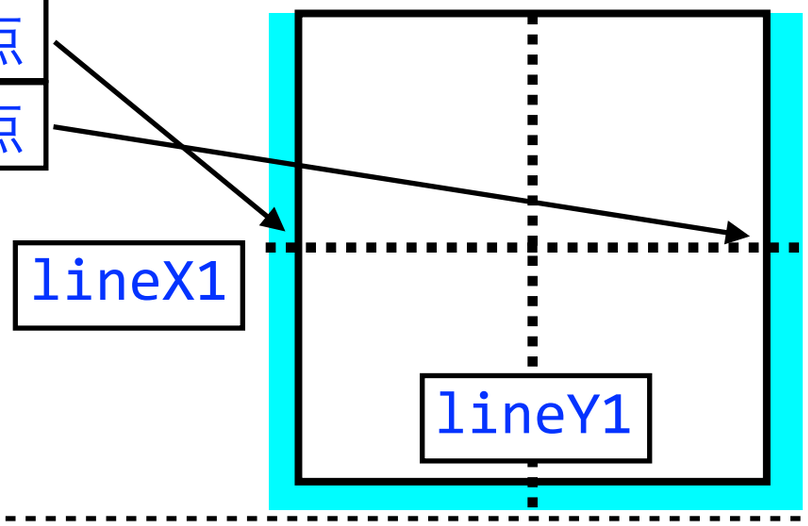
サンプリング名

サンプリング型，格子中心と界面．他にfaceなど

出力する座標軸，x/y/z/xyz(全座標)

サンプリング開始点

サンプリング終了点



サンプリングする場の リスト

サンプリングの実行

サンプリング実行

```
postProcess -func sample -latestTime
```

-latestTimeは最終時刻のみ実行とするオプション.
オプション無しの場合，出力された時刻全てに対して実行される

サンプリング結果確認

```
more postProcessing/sample/30/lineX1_U.xy
```

postProcessing/sample/30/lineX1_U.xy

```
x    Ux    Uy    Uz
0    0    0    0
0.025    -0.00210791    0.0432661 0
:
0.975    -0.00499169    -0.0506717 0
1    0    0    0
```

sampleにおけるAxisの指定がxなので，x座標が出力されている

プロット

gnuplotの入力ファイル確認

```
more profiles.gp
```

profiles.gp(一部のみ表示)

```
plot \
 'u-vel.dat' using 3:2 axes x2y1 title 'Ghia et al., u' with point pt 4\
,'v-vel.dat' using 2:3 axes x1y2 title 'Ghia et al., v' with point pt 6\
,'< cat postProcessing/sample/*/lineY1_U.xy’\
 using 2:1 axes x2y1 title 'case 0, u' \
,'< cat postProcessing/sample/*/lineX1_U.xy' \
 using 1:3 axes x1y2 title 'case 0, v'
```

u-vel.dat，v-vel.datがGhiaらの結果(出典：オープンCAE勉強会@関西 - 「講師の気まぐれOpenFOAMもくもく講習会」テキスト)．Re数に応じて赤字のカラム番号を要変更
Re=100(3カラム)，400(4)，1000(5)，3200(6)，5000(7)，7500(8),10000(9)

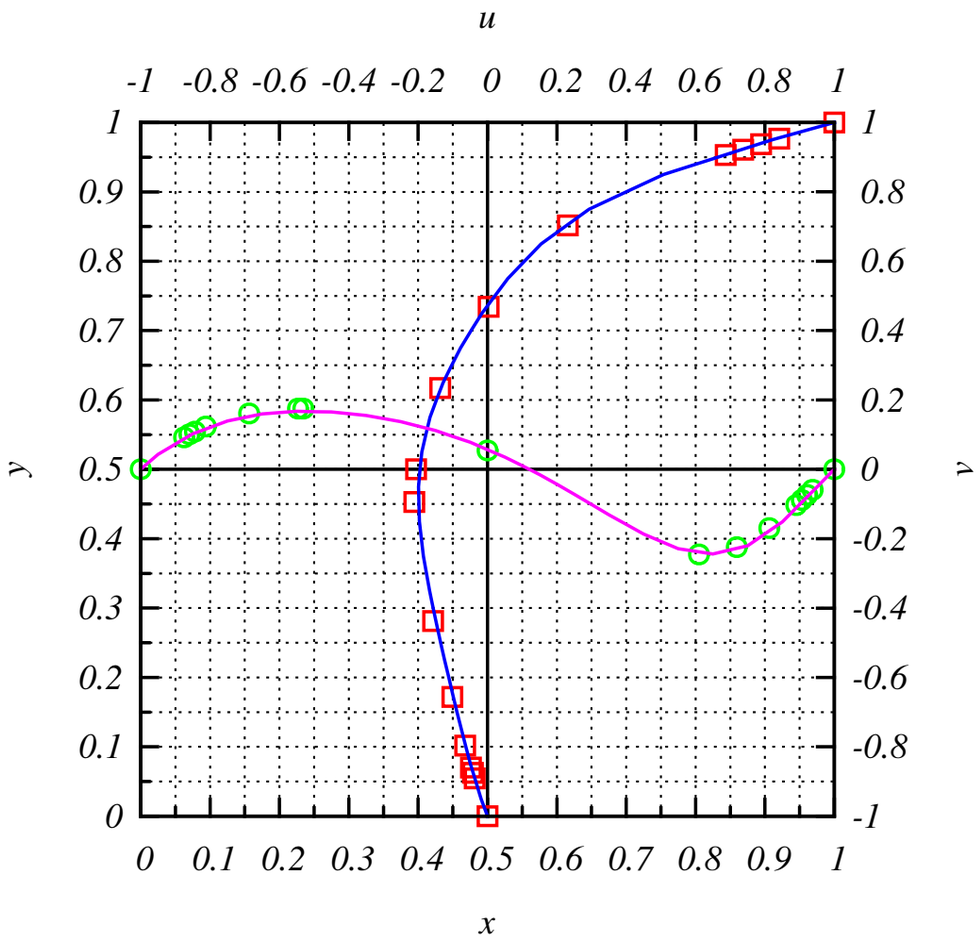
gnuplot実行

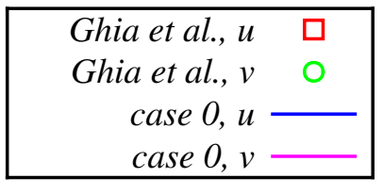
```
gnuplot profiles.gp
```

プロットファイル表示(X転送不可の時は，端末のアスキープロットを参照)

```
evince profiles.pdf
```

プロット結果と演習





Re=100の場合には，粗い20分割で，
Ghiaらによる128分割の計算結果とほぼ一致

Ghiaらの検討Re数：
100, 400, 1000, 3200, 5000, 7500, 10000

課題1: Re数を上げていき，定常状態に近い計算結果を取得しプロットする.

ヒント：Re数を変更するには，動粘性係数nuを変更する．profiles.gpも変更する.

課題2: 課題1でGhiaらの結果と大きく異なる場合，1辺の分割を128に変更して，Ghiaらと一致するか確かめる．なお，高Re数で積分時間を増した場合には，最大実行時間10分以内に収束しない場合もある.

ヒント：格子の分割数を変更するには，blockMeshの設定を変更する.

キャビティ流れ演習III

OpenFOAMの並列計算手法

1. 格子生成
2. 領域分割 (decomposePar)
3. MPI並列でソルバを実行

(MPI+OpenMPのハイブリット並列は標準では未実装．研究例有り[1][2])

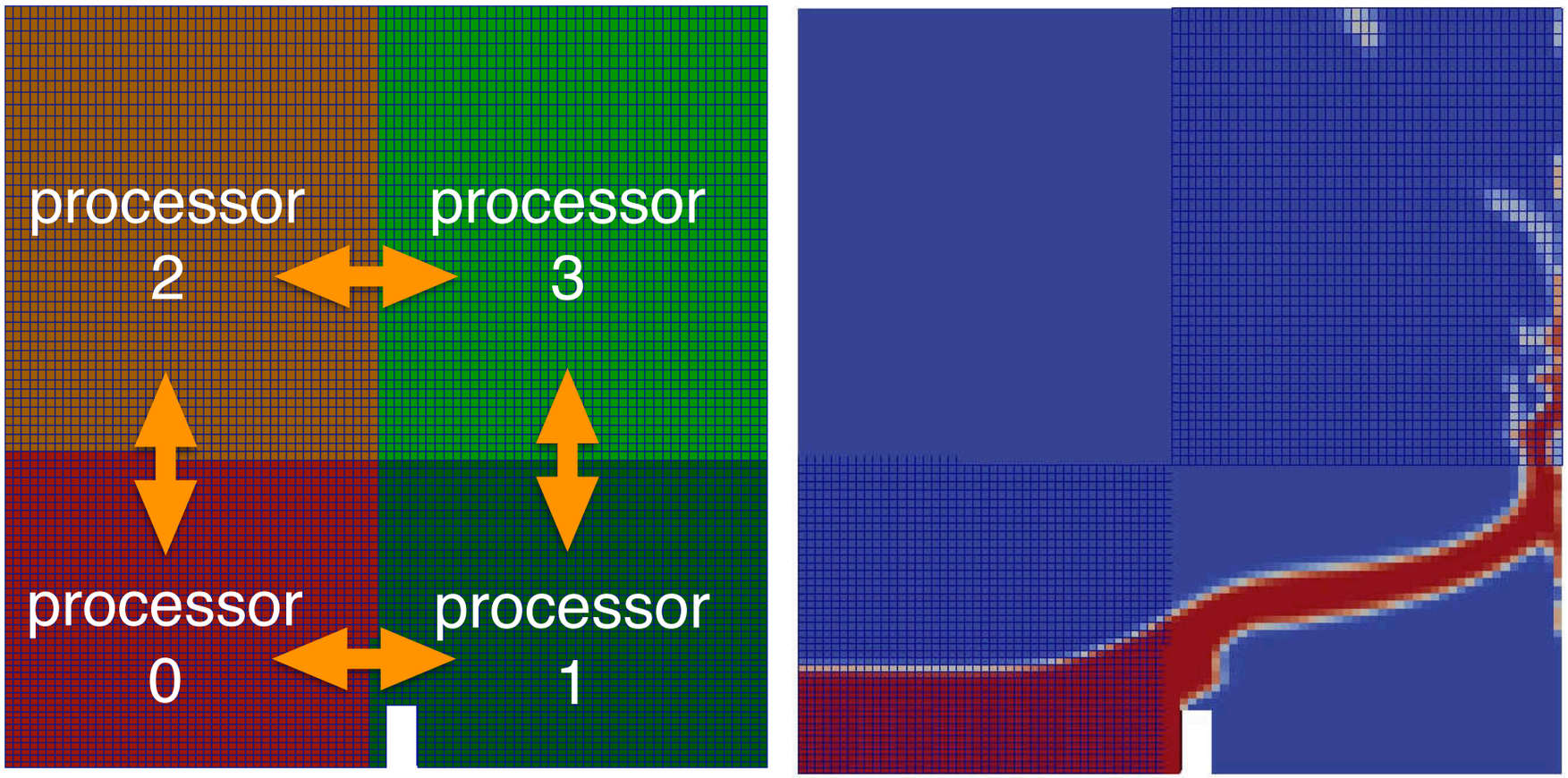
4. 領域毎の解析結果を再構築 (reconstructPar)

[1] 櫻井隆雄，片桐孝洋，大島聡史，猪貝光祥，黒田久泰: OpenFOAMへの疎行列計算ライブラリXabclibの適用と評価，オープンCAEシンポジウム2014

[2] 内山学，ファム・バン・フック，千葉修一，井上義昭，浅見暁:OpenFOAMによる流体コードのHybrid並列化の評価 第151回HPC研究発表会

プロセッサ間MPI通信 (フラットMPI通信)

プロセッサ間通信は，計算効率を低下させる

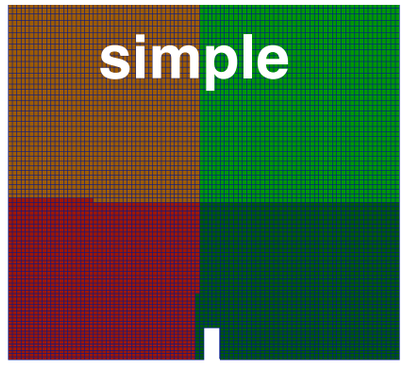


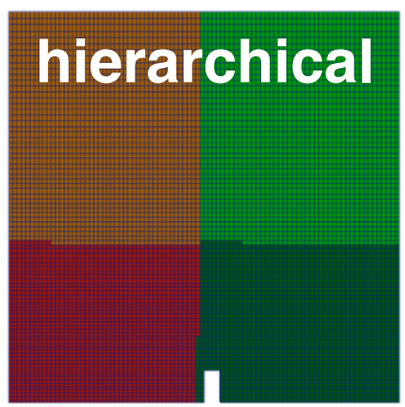
damBreakFineチュートリアル

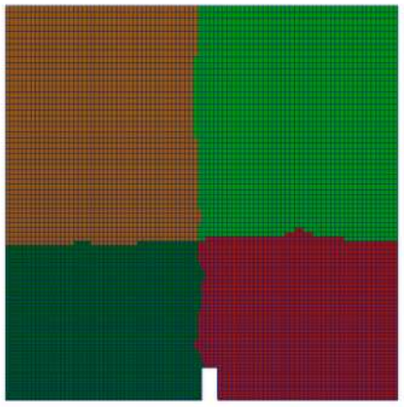
領域分割の設定

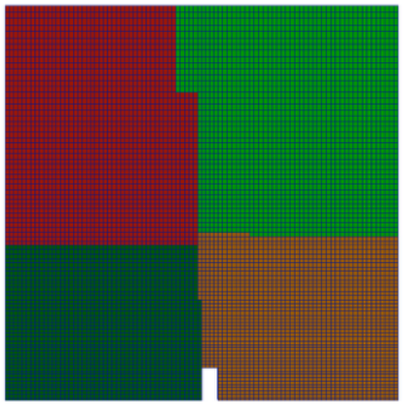
system/decomposeParDict

```
numberOfSubdomains 4;          //領域分割数

method                  simple; //領域分割方法

simpleCoeffs //単純に軸方向に分割
{
    n          ( 2 2 1 ); //分割数
    delta      0.001;
}

hierarchicalCoeffs //分割方向の順番を指定
{
    n          ( 2 2 1 );
    order      xyz; //分割方向の順番
    delta      0.001;
}
```









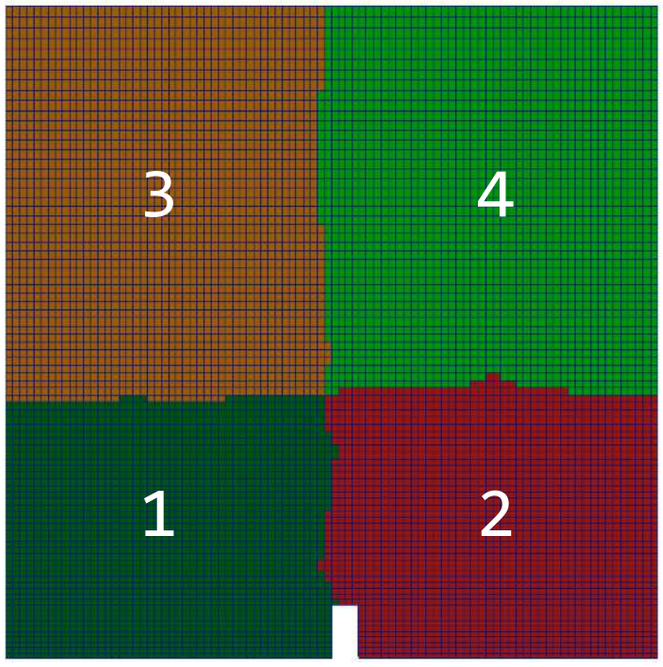
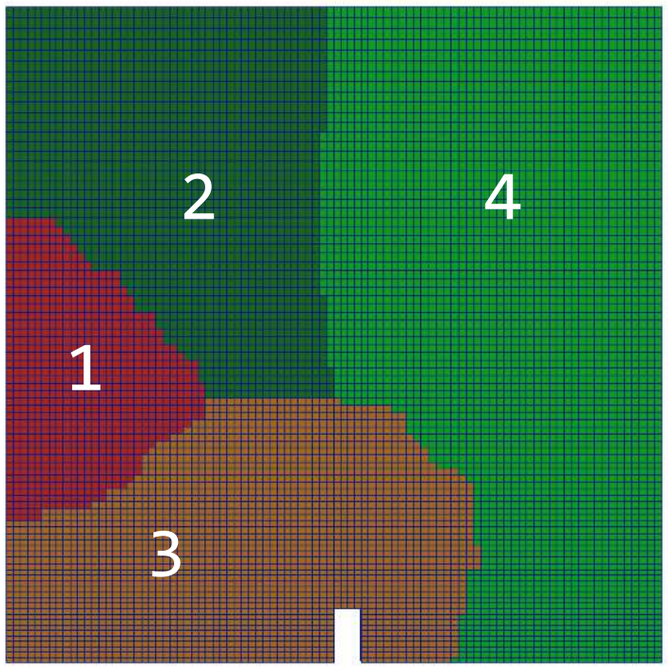
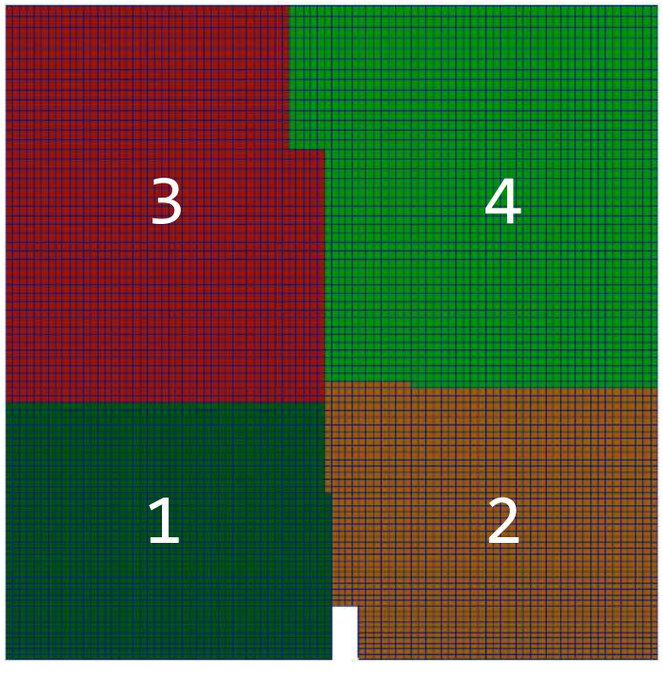
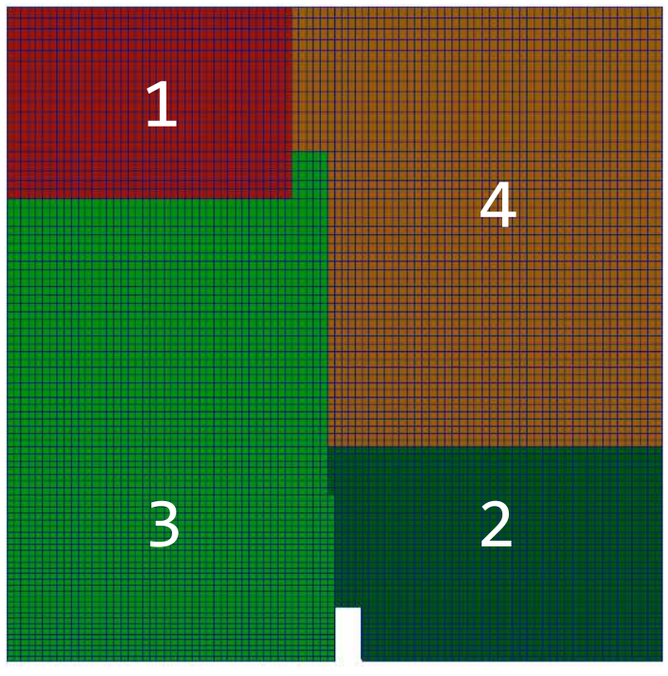
metis

metis：Metisライブラリを使用。プロセッサ間の通信量に大きく影響する分割領域間の界面数を最小化。ライセンスにより商用利用や再配布が自由ではない

scotch：Scotchライブラリを使用。フリーソフトライセンスでMetisと互換APIを持つ

manual：格子を割り充てるプロセッサを手動で指定

重み付き領域分割例

	重み無し	重み付き
metis		
scotch		

system/decomposeParDict

```
scotchCoeffs
または
metisCoeffs
{
   processorWeights
   ( 1 2 3 4 );
//プロセッサ毎の格子数の重み係数
//省略時は重み無し
}
```

重み付けは，ノード間で性能が異なる場合に用いる場合があるが，通常は用いない

並列計算実行用ジョブスクリプト

parallel.sh（フラットMPI並列ジョブ用スクリプト，赤字が並列用追加分）

```
#!/bin/bash
#PBS -q u-lecture
#PBS -W group_list=gt00
#PBS -l select=1:ncpus=4:mpiprocs=4:ompthreads=1
#PBS -l walltime=0:10:00

cd $PBS_O_WORKDIR
export MPI_BUFFER_SIZE=20000000
. /etc/profile.d/modules.sh
module purge
module load intel/17.0.2.174
module load mpt/2.14
module load openfoam/1612-mpt

blockMesh >& log.blockMesh
decomposePar -cellDist >& log.decomposePar
mpiexec_mpt icoFoam -parallel >& log.icoFoam
reconstructPar >& log.reconstructPar
```

キュー名(演習中はu-tutorialのほうが優先度が高い)

select(ノード数)=1
ncpus(ノードあたりのコア数)=4
mpiprocs(ノードあたりのMPI数)=4
ompthreads(プロセスあたりスレッド数)=1

領域分割(-cellDistオプションは分割領域可視化用．必須ではない)

並列でのicoFoam実行

SGI MPT版 mpiexec

並列化オプション

再構築

ジョブ投入と領域分割のログの確認

ジョブの投入(並列計算用にケースをコピーしてから投入)

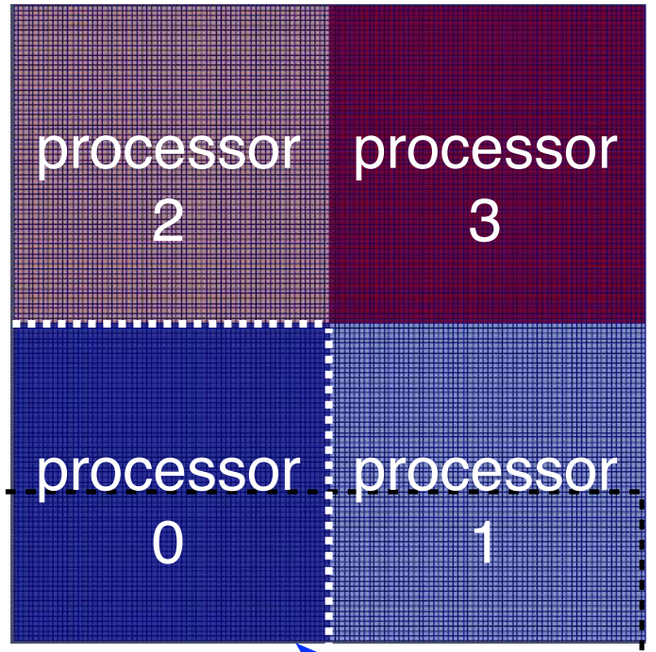
```
qsub parallel.sh
```

領域分割のログの確認(ジョブ開始後に実行)

```
more log.decomposePar
```

log.decomposePar

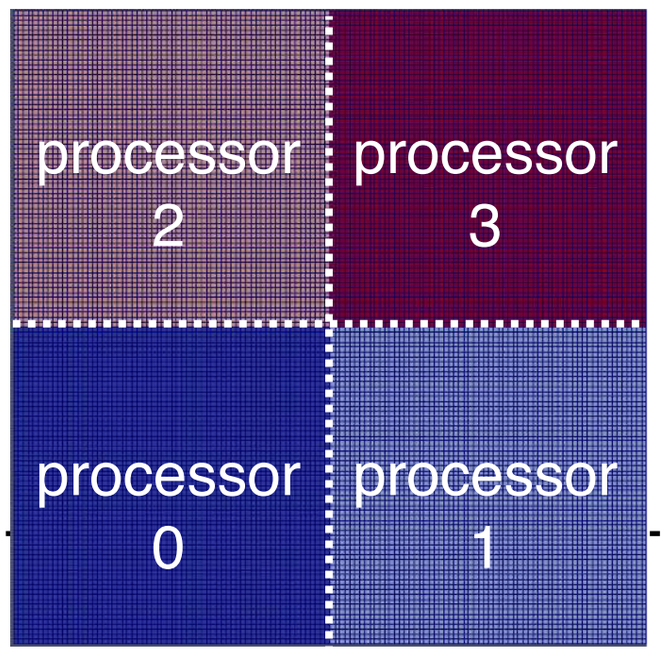
```
Processor 0 #プロセッサ0の担当分割領域
    Number of cells = 4096 #格子数
    Number of faces shared with processor 1 = 64
    #プロセッサ番号1の担当分割領域との共有界面数
    Number of faces shared with processor 2 = 64
    #プロセッサ番号2の担当分割領域との共有界面数
    Number of processor patches = 2 #上記の界面を共有するプロセッサ数
    Number of processor faces = 128 #上記の共有界面数の合計(=64+64)
    Number of boundary faces = 8320 #この領域における境界面の界面の合計

Processor 1  #プロセッサ1の担当分割領域
    Number of cells = 4096 #格子数
```



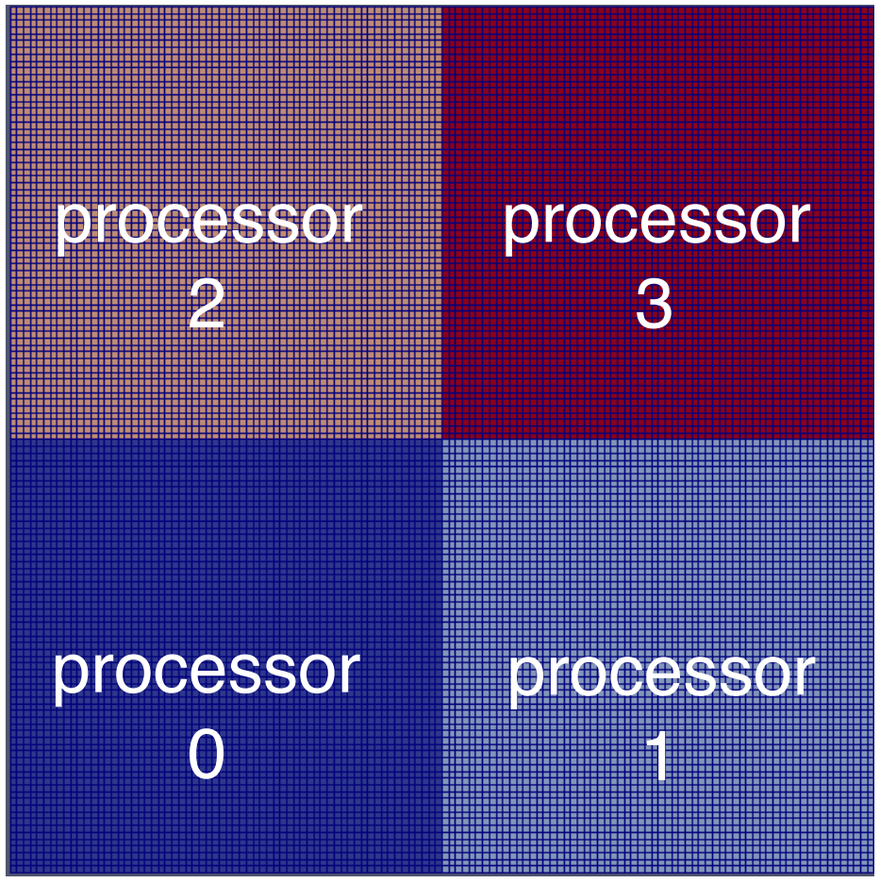
領域分割のログの確認(続き)

log.decomposePar

```
Number of processor faces = 256 #共有界面数の総数(小さいほうが良い)
#以下、全プロセッサ担当分割領域における各種統計値
#プロセッサの計算能力が同等な場合、以下の量はバラツキが無いほうが良い
Max number of cells = 4096 (0% above average 4096)
Max number of processor patches = 2 (0% above average 2)
Max number of faces between processors = 128 (0% above average 128)

Wrote decomposition as volScalarField to cellDist for use in
postprocessing.
Time = 0

Processor 0: field transfer
#プロセッサ0のディレクトリに場データを出力(以下同様)
Processor 1: field transfer
Processor 2: field transfer
Processor 3: field transfer
```



領域分割結果




```
constant/
  polyMesh/     #格子データ
0/              #時刻ディレクトリ
  U, p          #場データ
```

領域分割前のデータ

```
processor0/     #プロセッサディレクトリ
  constant
    polyMesh/   #分割領域の格子データ
      0/        #時刻ディレクトリ
        U,p     #分割領域の場データ

processor1/     #以下processor0と同様
processor2/
processor3/
```

領域分割により生成されたデータ

並列計算時のログ

ログ中のプロセッサ数や通信オプションを確認

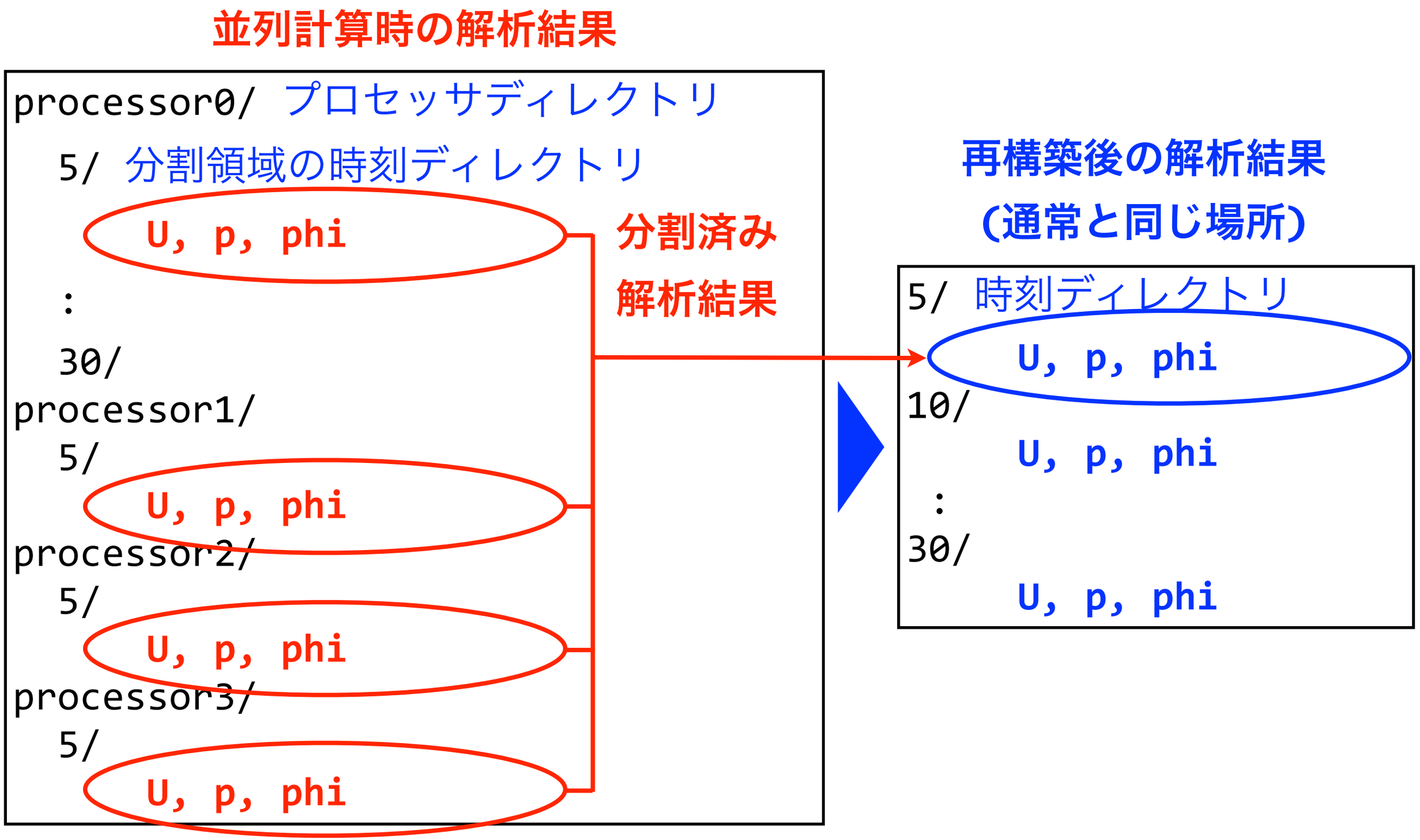
```
more log.icoFoam
```

log.icoFoam

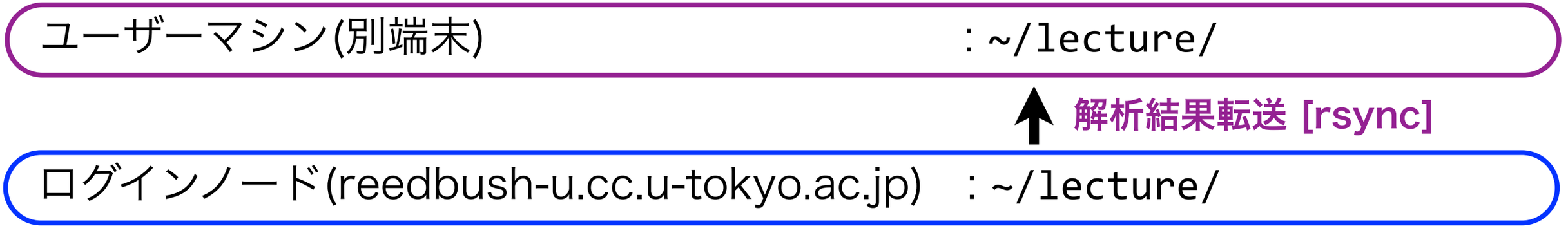
```
nProcs : 4              #計算で使用されている総プロセッサ数
Slaves :
3                       #マスタープロセスに従属するスレーブプロセス数(nProcs-1)
(
"n343.10463"            #ホスト名.PID(プロセスID) (ホスト名やPIDは実行環境依存)
"n343.10464"            #計算で使用されているノード数=ユニークなホスト名の数
"n343.10465"
)
Pstream initialized with: #通信条件(以下がデフォルト．詳細は省略)
    floatTransfer      : 0
    nProcsSimpleSum    : 0
    commsType          : nonBlocking
    polling iterations : 0
```

意図したノード・プロセッサ数で動いているか確認(失敗→ジョブファイルの指定確認)

並列計算結果の再構築


並列計算時の解析結果
processor0/ プロセッサディレクトリ
5/ 分割領域の時刻ディレクトリ
U, p, phi
:
30/
processor1/
5/
U, p, phi
processor2/
5/
U, p, phi
processor3/
5/
U, p, phi
分割済み
解析結果
再構築後の解析結果
(通常と同じ場所)
5/ 時刻ディレクトリ
U, p, phi
10/
U, p, phi
:
30/
U, p, phi

解析結果の転送



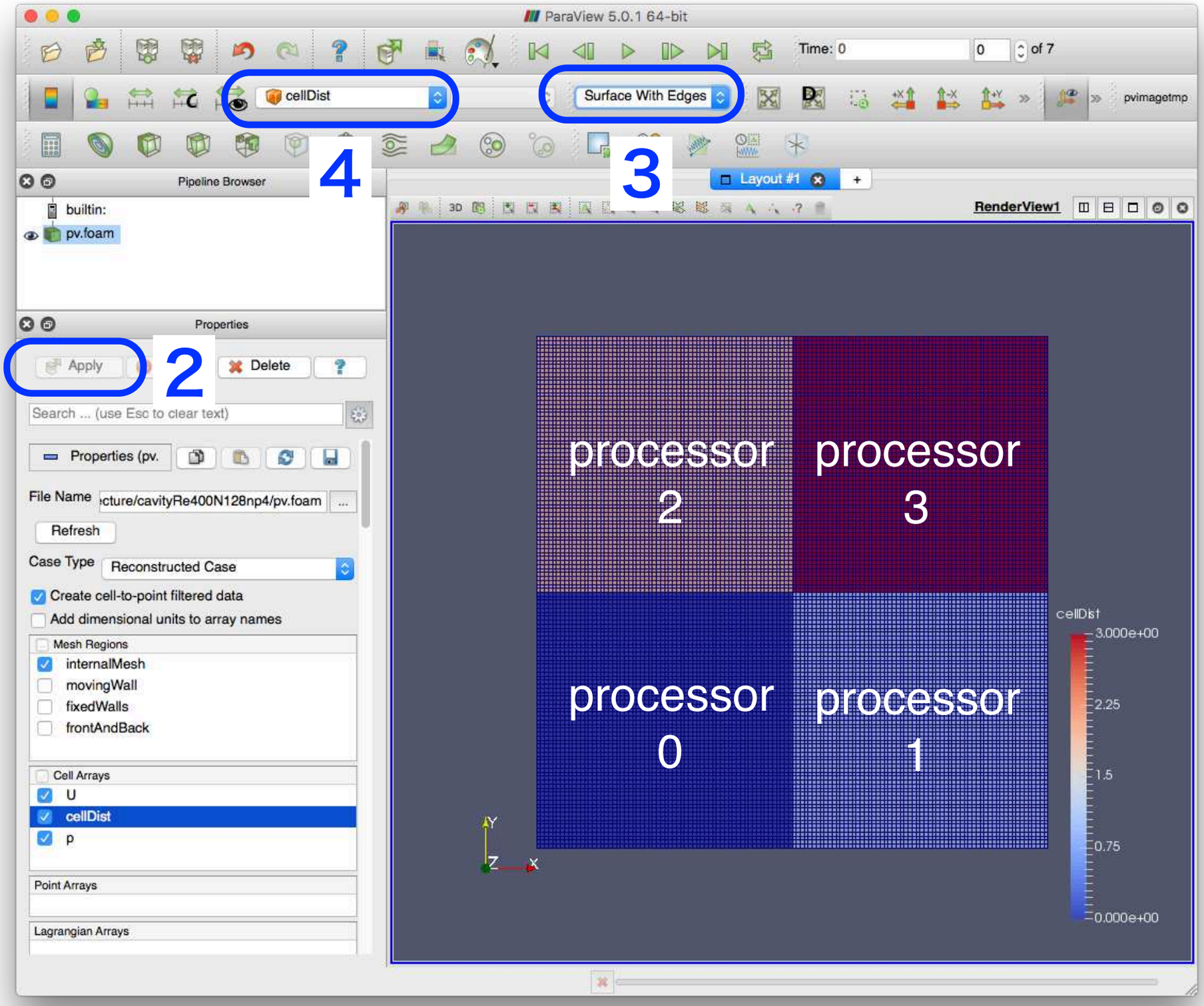
解析結果の転送(別端末で実行)

```
cd ~/lecture
rsync -auv txxxxx@reedbush-u.cc.u-tokyo.ac.jp:~/lecture/ ./
```

↑(カーソル上)を押して前のコマンドを呼び出して修正するほうが速い

```
cd 並列計算を行なったケースのディレクトリ
touch pv.foam
```

ParaViewによる分割領域の可視化



1. Fileメニュー/Open/cavityのディレクトリのpv.foamを開く
2. Apply
3. Representation/Surface With Edges 選択
4. Coloring/□cellDist 選択

decomposeParのcellDistオプションにより，0ディレクトリに分割領域のプロセッサ番号の場であるcellDistが出力されるので，それを可視化

台数効果と並列化効率

- 並列計算による台数効果 (スピードアップ) S_P

$$S_P = T_S / T_P$$

ここで

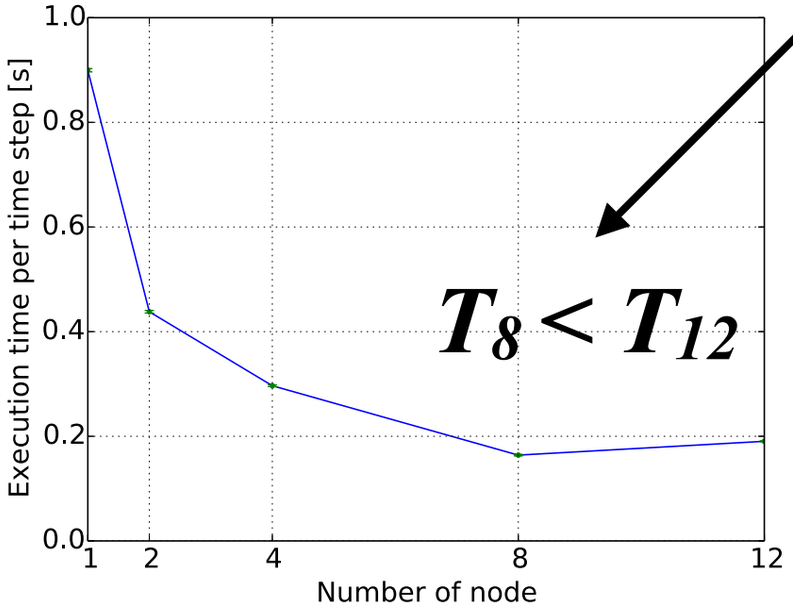
T_S ： 1コア(または1ノード)での実行時間

T_P ： Pコア(またはPノード)での実行時間

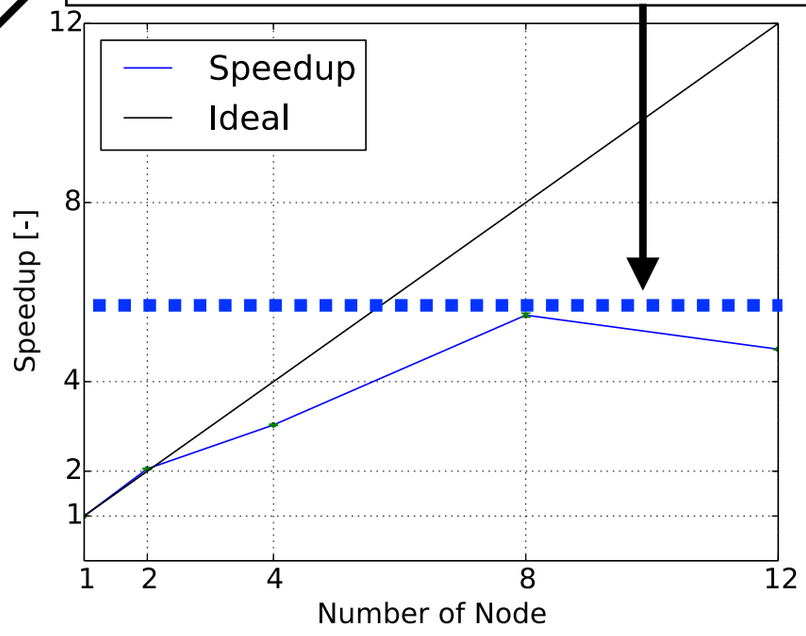
- 並列化効率 E_P

$$E_P = S_P / P \times 100\ [\%]$$

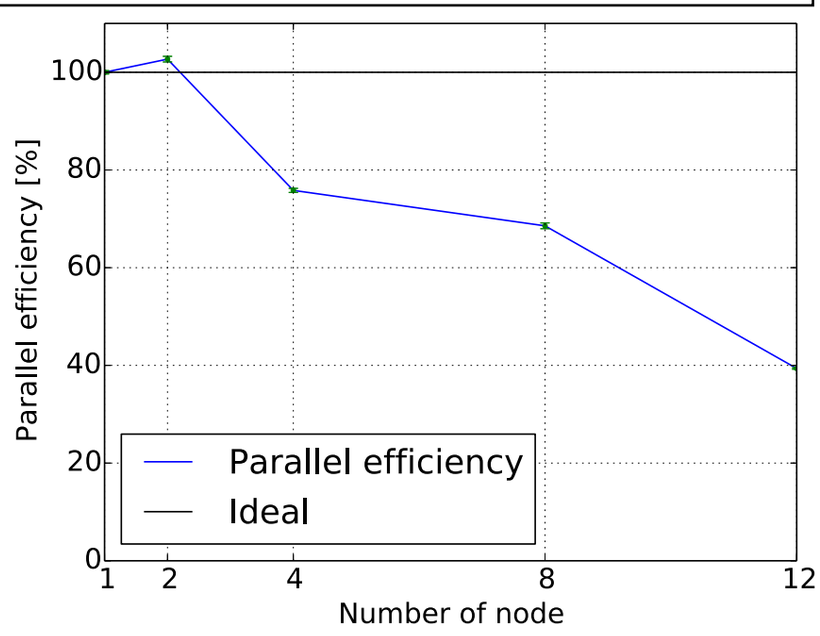
このケースでは12ノード以上使用しても非効率
→台数効果の飽和(Saturation)をベンチマークテストで事前に把握し，非効率な計算を行わないことが重要



実行時間



台数効果(スピードアップ)



並列化効率

ベンチマークテスト

- **ベンチマークテスト：プログラム実行時間やFLOPS値などの性能指標の計測**
- **並列計算機でのベンチマークテストは重要**
  - ✓ 並列数を変更させて，台数効果(スピードアップ)や並列化効率を調べ，効率の良い並列数を決定できる
  - ✓ 時間ステップ数や反復数が小さい予備計算で検討するのが効率的
- **OpenFOAM等のCFDコードでは圧力線型ソルバのベンチマークテストも重要**
  - ✓ 実行時間は圧力線型ソルバの種類や前処理方法に強く依存
  - ✓ 線型ソルバーの速さは並列数にも依存
    - 並列数小→AMG(代数マルチグリッド) > PCG(前処理付き共役勾配法)
    - 並列数大→PCG > AMG

演習課題

課題1 Re=400，20 or 128分割格子，1ノード4並列における並列計算のスピードアップ率および並列化効率(Strong scaling)を求める.

方法：非並列計算ケースのログlog.icoFoamおよび並列計算ケースのログlog.icoFoamにおける，最後の時間ステップのClockTimeをそれぞれt1,t4として以下で求める

スピードアップ率=t1/t4

並列化効率=(t1/t4)/4

課題2 1ノードおよび複数ノードでの異なる並列数の計算を行い，スピードアップ率および並列化効率(Strong scaling)を求める.

課題3 256分割格子で，課題1，2を行う

プロファイラVTuneの基礎的使い方

- プロファイラにより，計算負荷が高い部分(ボトルネック)等，計算効率改善のためのおおまかなデータを，ソースの改変無しに取得可能
- ソースレベルの詳細プロファイリングには，再ビルドやソース改変が必要
- intelのプロファイラVTuneの基礎的かつCLIでの使い方を示す(GUIでの使い方はReedbush利用支援ポータル内のマニュアルを参照)

intel VTuneのmoduleをロード

```
module load intel-vtune/17.0.2.478468
```

プロファイラ実行用にcavityケースを複製

```
cdw
cd lecture/
cp -r cavityRe100 cavityRe100-p
cd cavityRe100-p
foamCleanTutorials
```

格子再生成 (短時間の実行なので今回はログインノードで実行)

```
blockMesh
```

プロファイラVTuneでの実行

ホットスポットの情報収集用にintel VTuneを用いてicoFoamを実行

```
amplxe-cl -collect hotspots -r vtune icoFoam > log.icoFoam
tail log.icoFoam
```

```
Elapsed Time:         8.271
CPU Time:             6.650
Average CPU Usage:    0.798
```

CPU使用率 79.8%

プロファイラデータをテキスト形式に変換し，表示

```
amplxe-cl -R hotspots -r vtune > vtune.txt
more vtune.txt
```

```
Function                                              CPU Time
:
std::ostream::flush                                   1.511s
:
Foam::DICPreconditioner::precondition  0.150s
:
Foam::lduMatrix::Amul                                 0.140s
:
```

解析本体ではDICの前処理や行列ベクトル積のCPU時間が多いことがわかる

その他のチュートリアルの実行(自習課題)

DNS	直接数値シミュレーション
basic	基礎的なCFDコード
combustion	燃焼
compressible	圧縮性流れ
discreteMethods	離散要素法
electromagnetics	電磁流体
financial	金融工学
heatTransfer	熱輸送
incompressible	非圧縮性流れ
mesh	格子生成
multiphase	多層流
lagrangian	ラグランジアン粒子追跡
resources	形状データ等の共用リソース置き場
stressAnalysis	固体応力解析

カテゴリ別に多数のチュートリアルがある.
OpenFOAMチュートリアルドキュメント作成プロジェクト[OFT]やカテゴリ，ケース名等を参考に，実行したいケースを選ぶ.

チュートリアルの実行例

チュートリアルケースのコピー(最初に一回のみ行う)

```
cdw
cd lecture/
cp -r $FOAM_TUTORIALS .
```

motorBikeケースに移動

```
cd tutorials/incompressible/simpleFoam/motorBike/
```

逐次計算用ジョブスクリプトのコピー

```
cp ~/lecture/foamRunTutorials.sh ./
```

ジョブの投入・ジョブ確認

```
qsub foamRunTutorials.sh
rbstat
```

ログ確認 (ジョブの実行開始後に行う．途中で^Cで終了して良い)

```
tail -f log.foamRunTutorials
```

チュートリアルのログが残っていると，foamRunTutorialsによるチュートアリアルの再実行ができないので，再実行する場合には以下のように初期化する．

```
foamCleanTutorials                    #今回は実行しない
```

チュートリアルの実行例(続き)

チュートリアルの転送(別端末で実行)

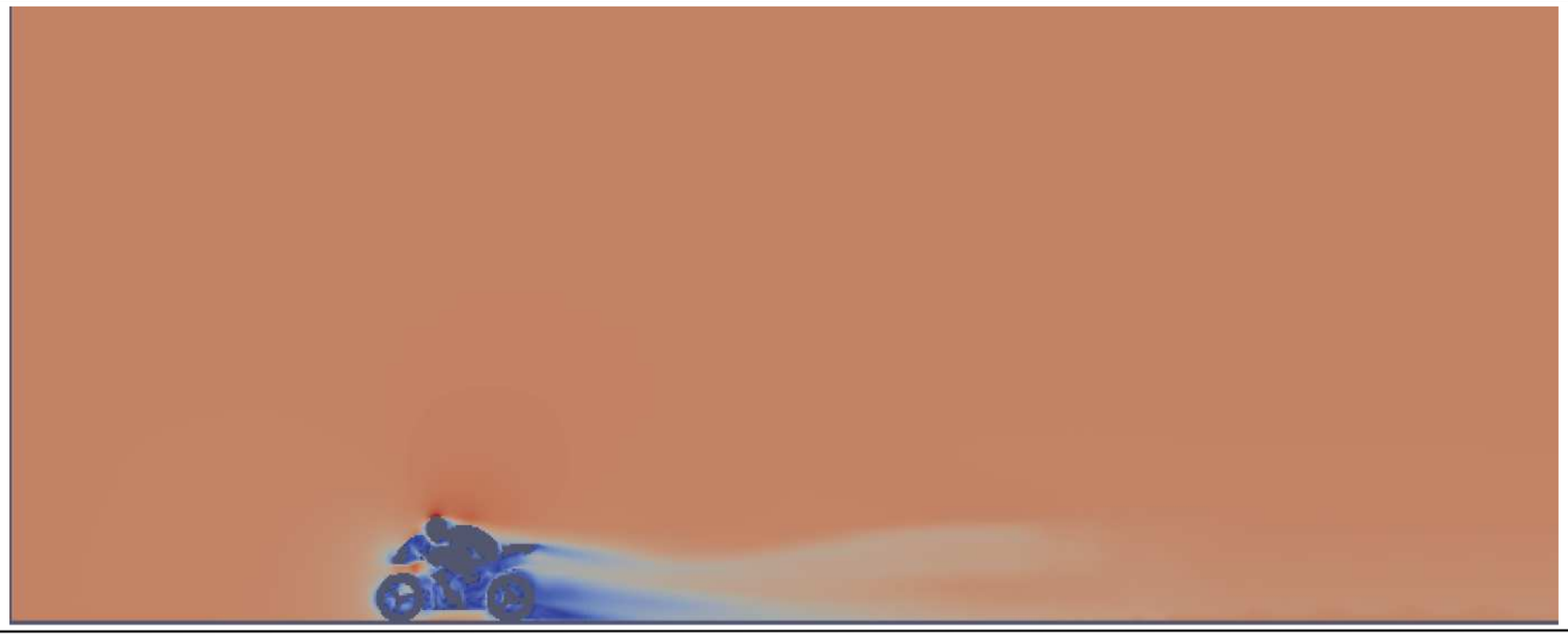
```
cd ~/lecture/
mkdir tutorials
cd tutorials/
rsync -auv txxxxx@reedbush-u.cc.u-tokyo.ac.jp:~/lecture/tutorials/ ./ --
exclude=processor*
```

実際は1行

excludeオプションで可視化に不要なディレクトリprocessor*を指定して除外する

ParaViewによる可視化

```
cd incompressible/simpleFoam/motorBike/
touch pv.foam
```

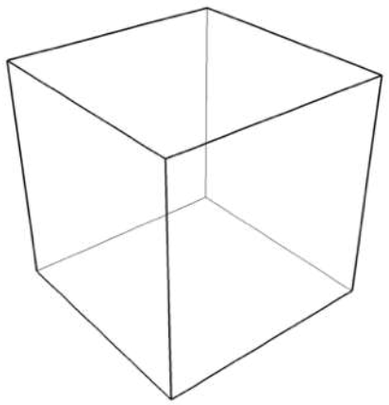


1. Fileメニュー/Open/motorBikeのディレクトリのpv.foamを開く
2. Fileメニュー/Open/postProcessing/cuttingPlane/500/U_yNormal.vtk
3. Coloring/・U

snappyHexMeshによる格子生成演習

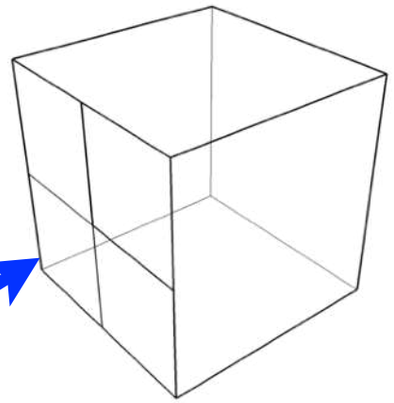
snappyHexMeshの特徴

- 六面体格子(hex)または”面が分割された六面体的状格子”(split hex)からなる格子を自動生成する。



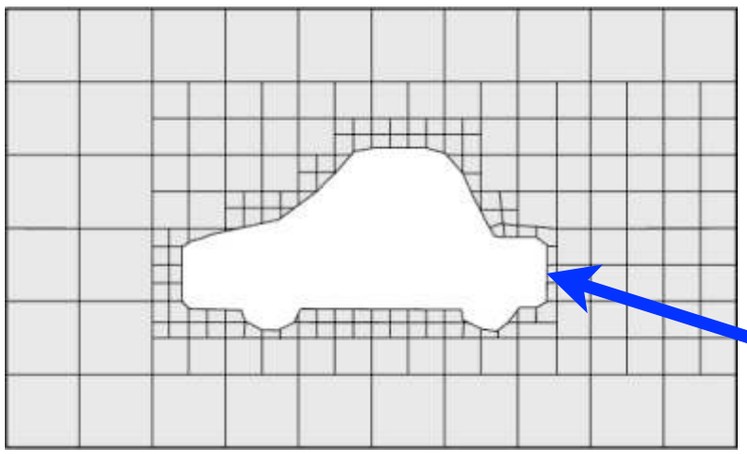
hex

- STL形式等の三角形分割曲面に適合した格子が生成できる(辞書で定義される直方体や球面等の基礎形状も利用可)



split面

split hex

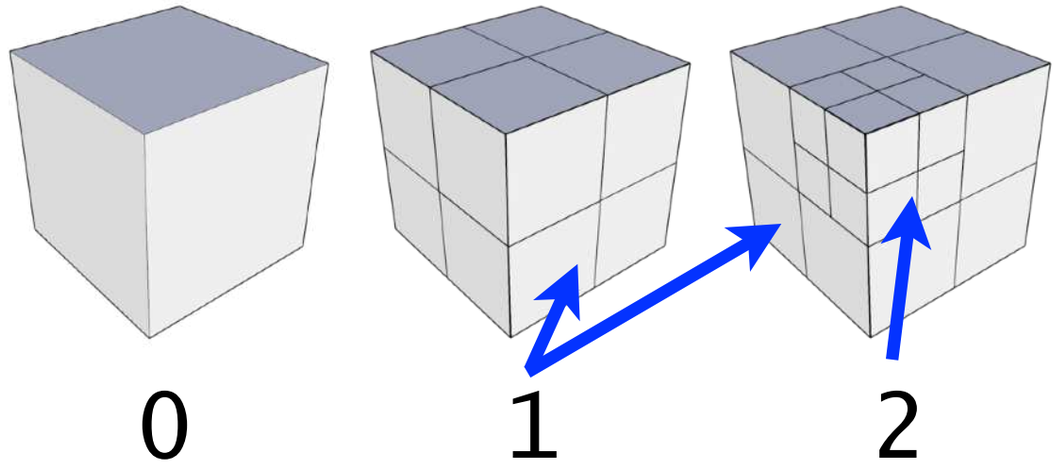


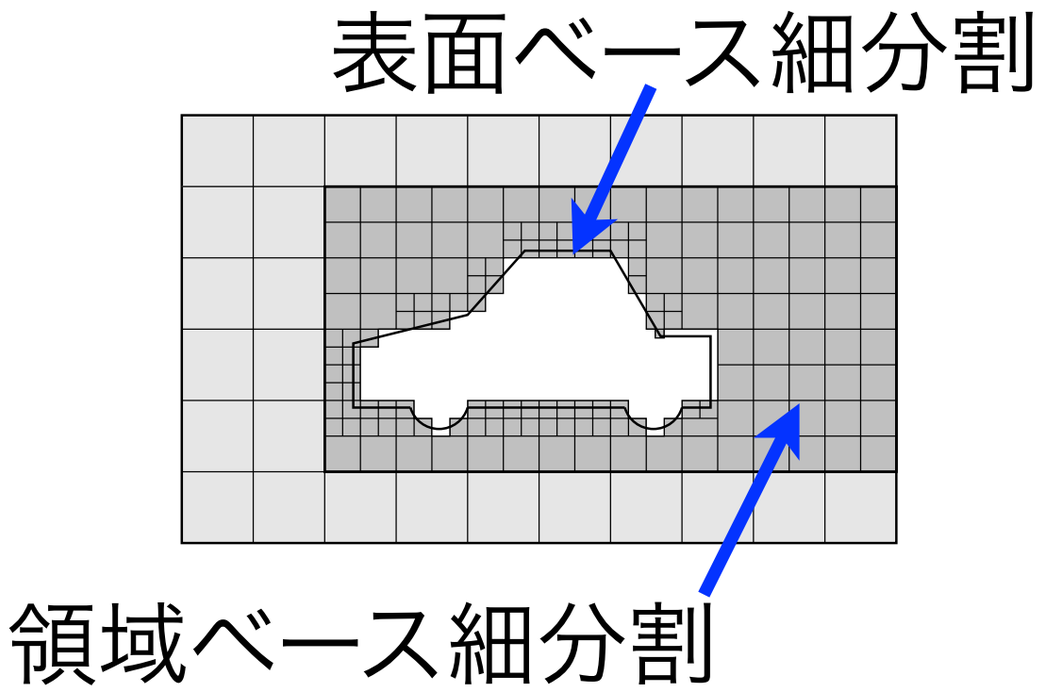
三角形分割曲面

(図引用元: OpenFOAM User Guide)

snappyHexMeshの特徴

- 格子の細分割は8分木(2x2x2)の分割を再帰的に行う
- 三角形分割曲面や基礎形状の表面の細分割レベルを指定できる(表面ベースの細分割)
- 細分割領域を三角形分割曲面や基礎形状を用いて別途定義できる (領域ベースの再分割)

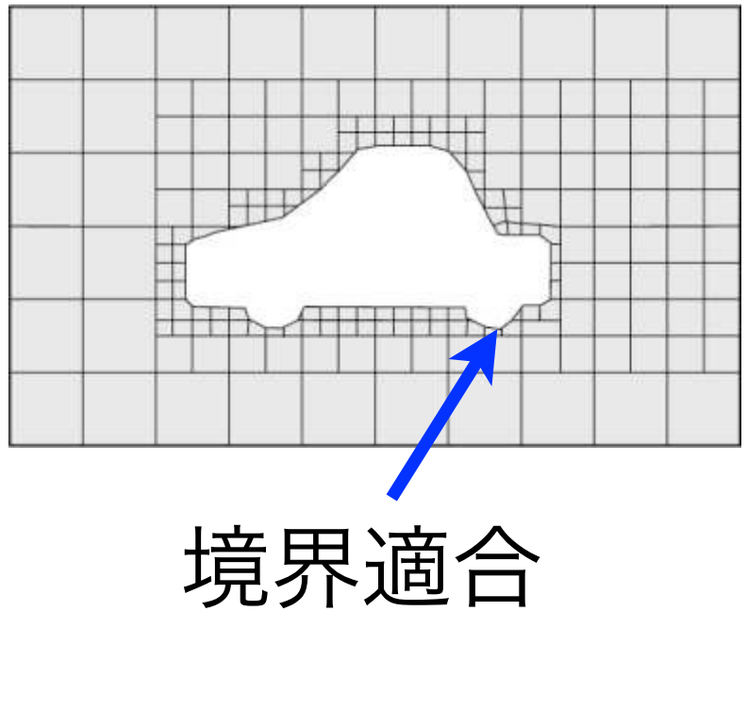




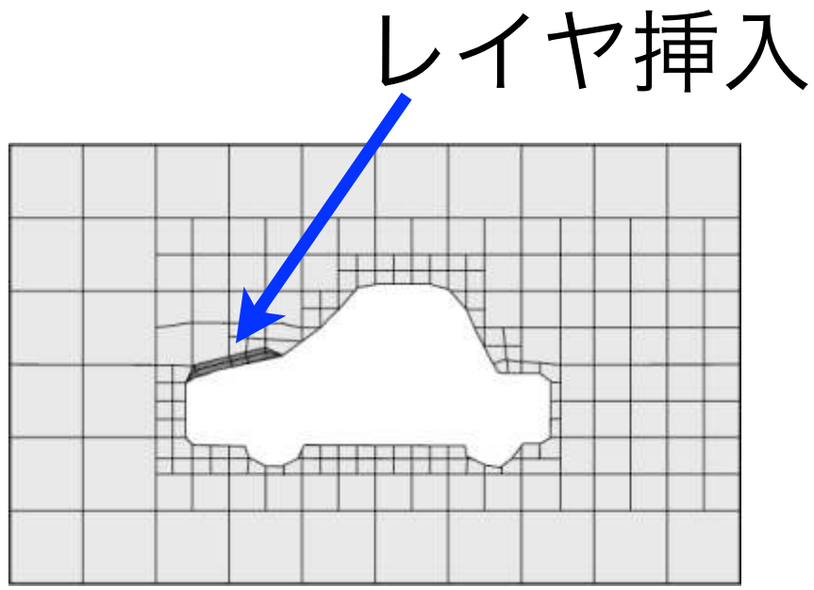
(図引用元: OpenFOAM User Guide)

snappyHexMeshの特徴(続き)

- 三角形分割曲面や基礎形状の表面に境界適合するように格子を滑らかに移動させることができる



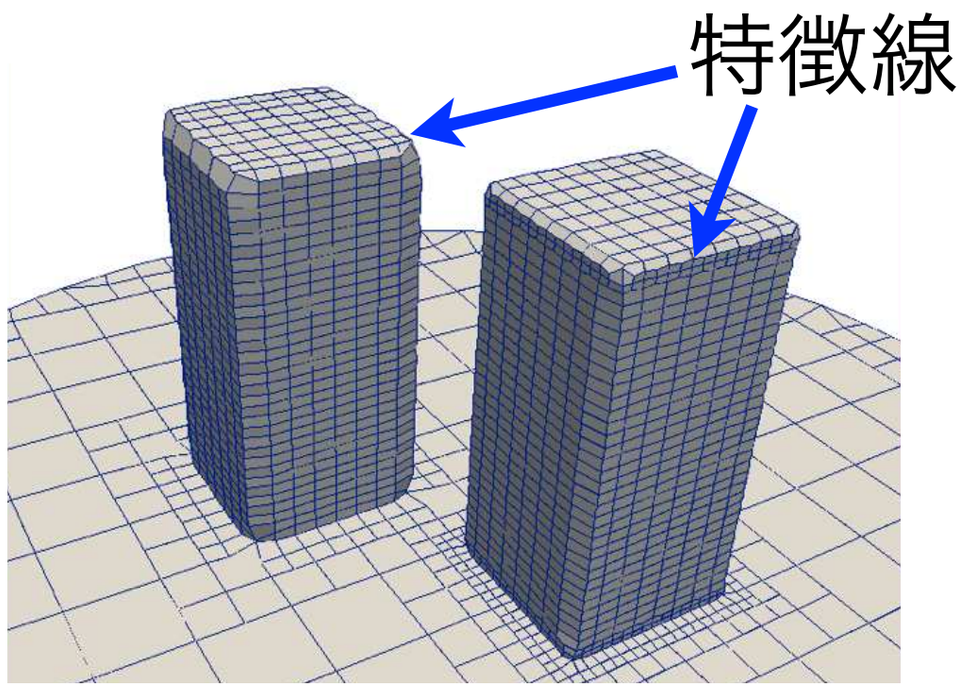
- 三角形分割曲面や基礎形状の表面にレイヤを滑らかに挿入できる

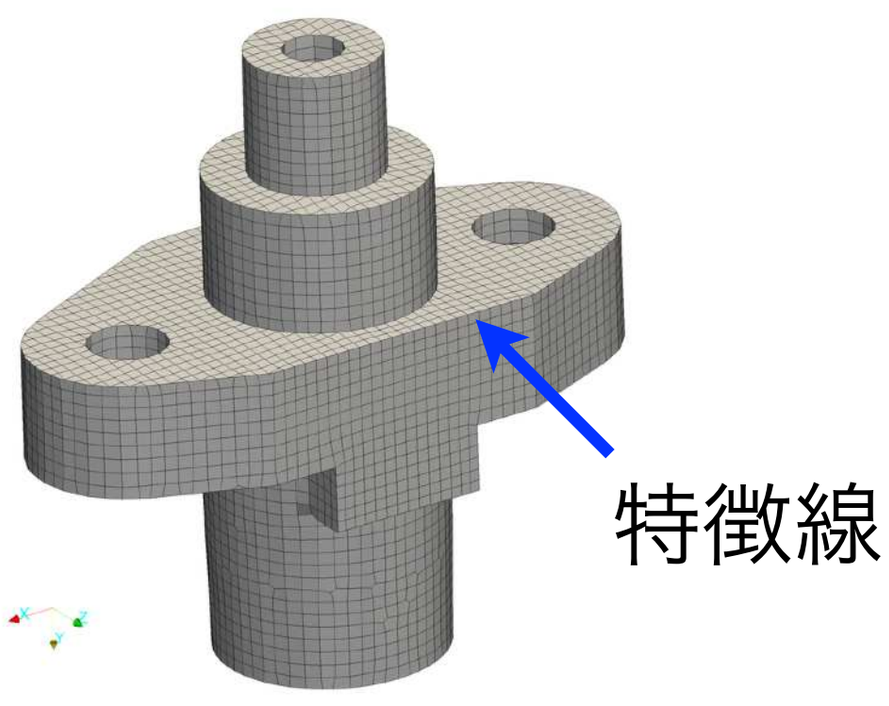


(図引用元: OpenFOAM User Guide)

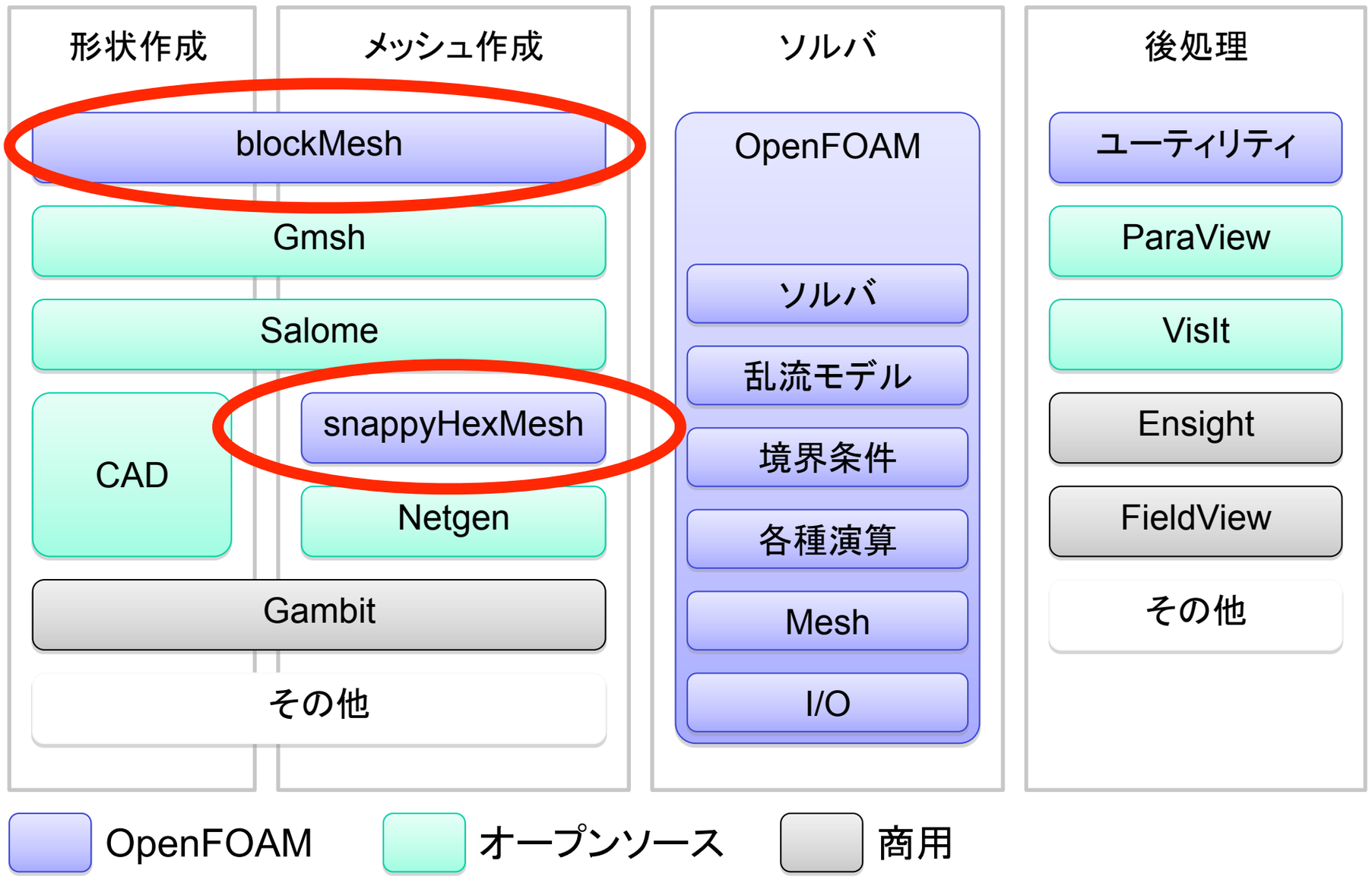
snappyHexMeshの特徴(続き)

- 分割領域毎の格子が揃うように(ロードバランシング)並列に格子生成ができる
- 境界適合における特徴線の再現機能はver-1.7までは実装されておらず、特徴線は丸くなった
- Ver-2.0から特徴線再現機能が実装された





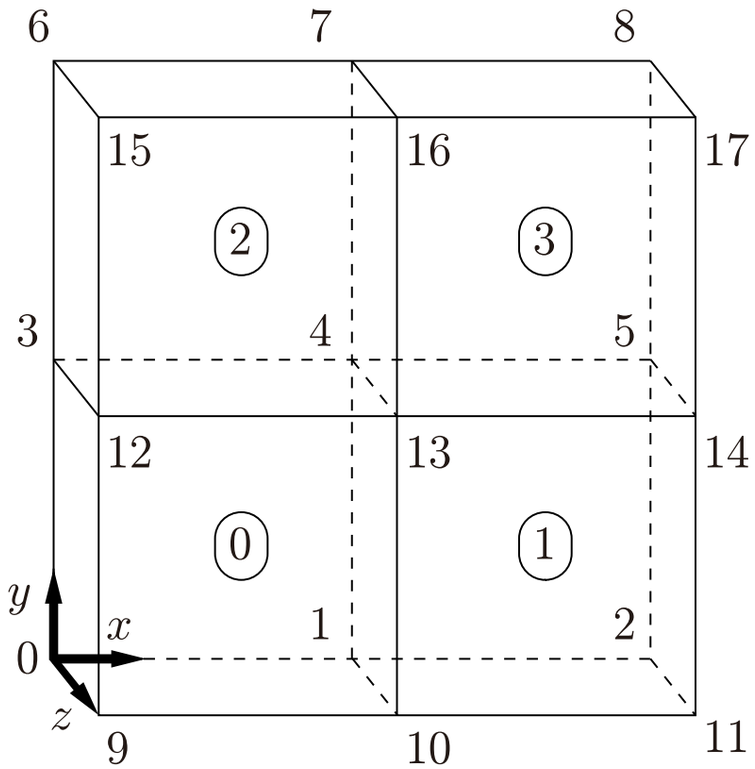
OpenFOAMの格子生成ユーティリティ



(図引用元: 大嶋 拓也 (新潟大学)「イントロダクション: OpenFOAM概要」
第一回OpenFOAM講習会)

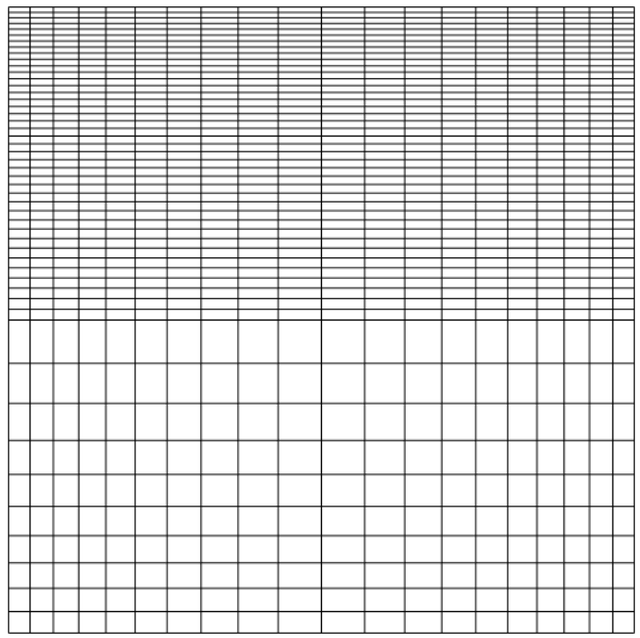
blockMesh

節点座標や分割方法を
`blockMeshDict`で定義



`blockMesh`

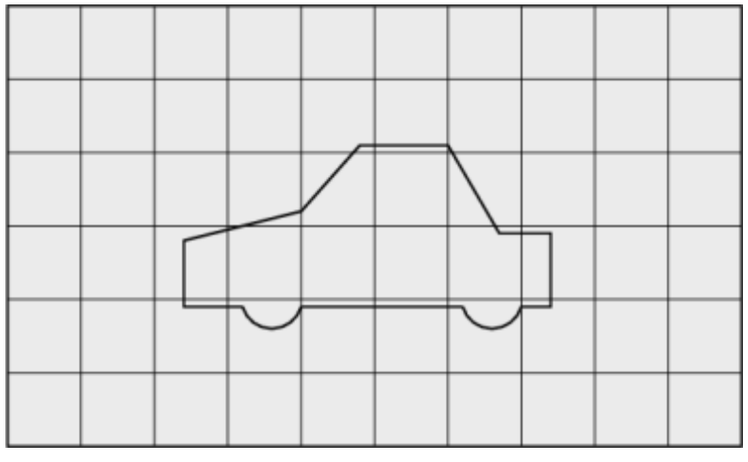
生成格子



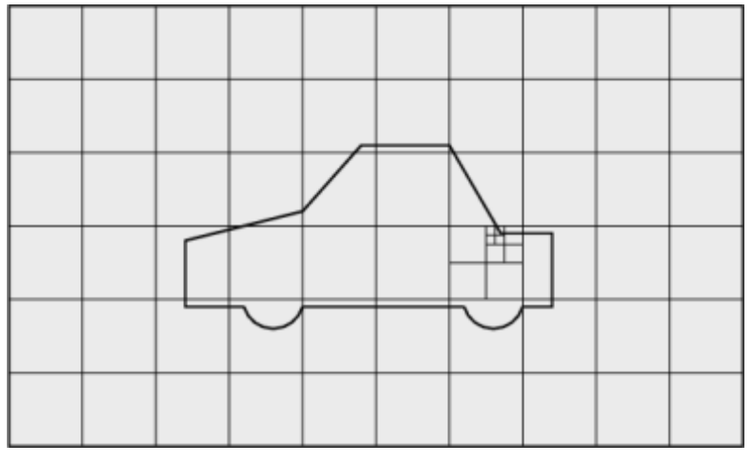
基礎的な六面体構造格子を生成する

(図引用元: OpenFOAM User Guide)

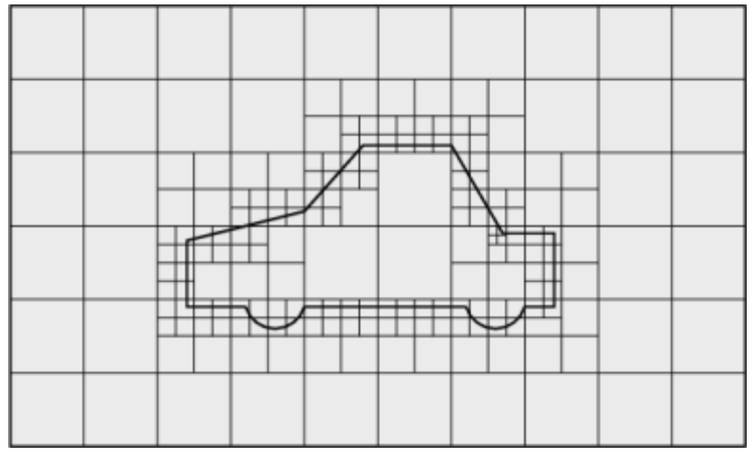
snappyHexMesh



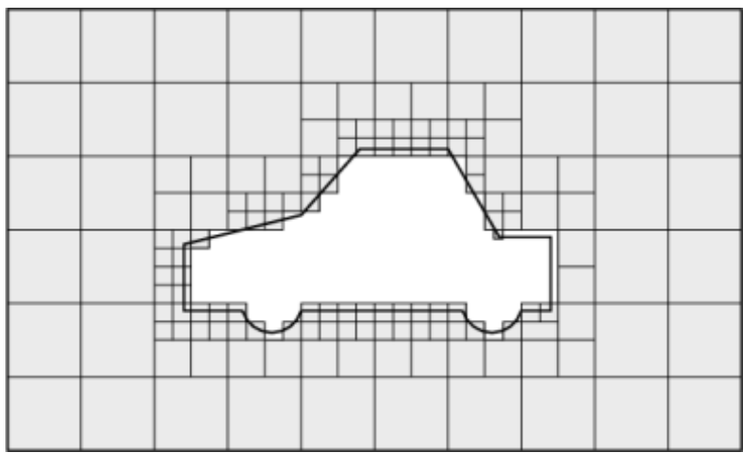
1. ベース格子



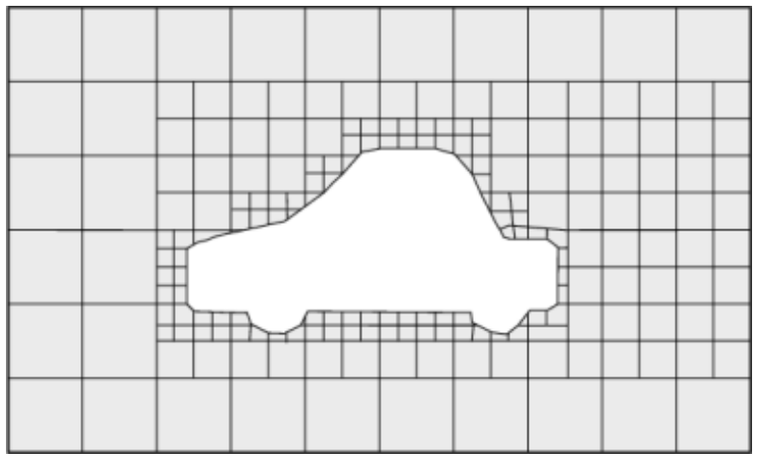
2. 物体表面の細分割



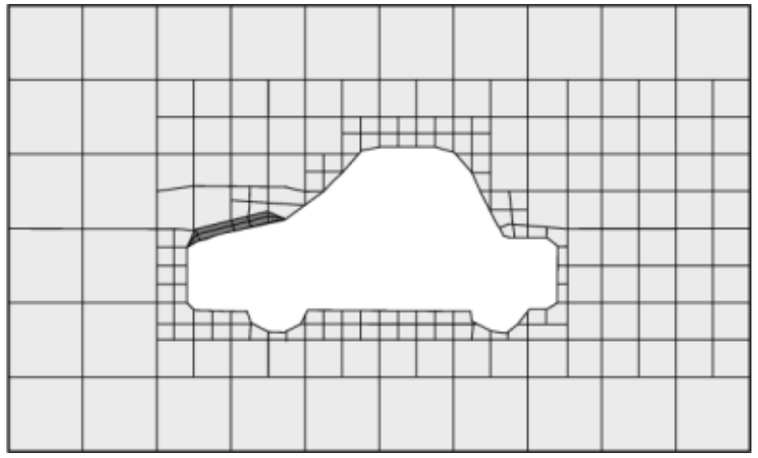
3. スムージング



4. 階段状格子



5. 境界適合(snap)

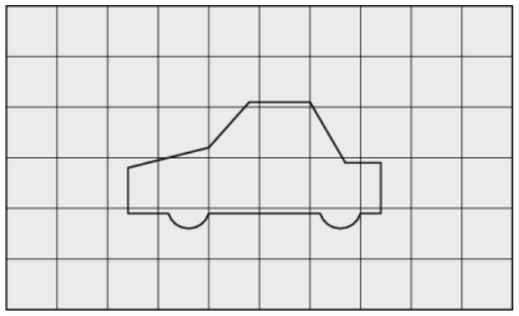


6. レイヤ付加

ベース六面体を元に複雑格子を生成

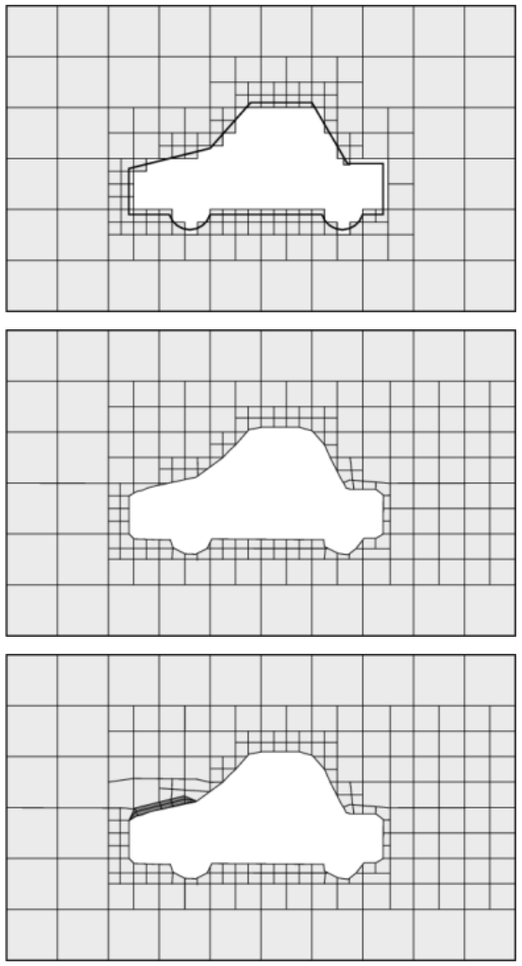
(図引用元: OpenFOAM User Guide)

格子の生成プロセス



ベースの
六面体格子

格子生成場所：
`constant/polyMesh`



階段状格子
`1/polyMesh`

境界適合
`2/polyMesh`

レイヤ付加
`3/polyMesh`

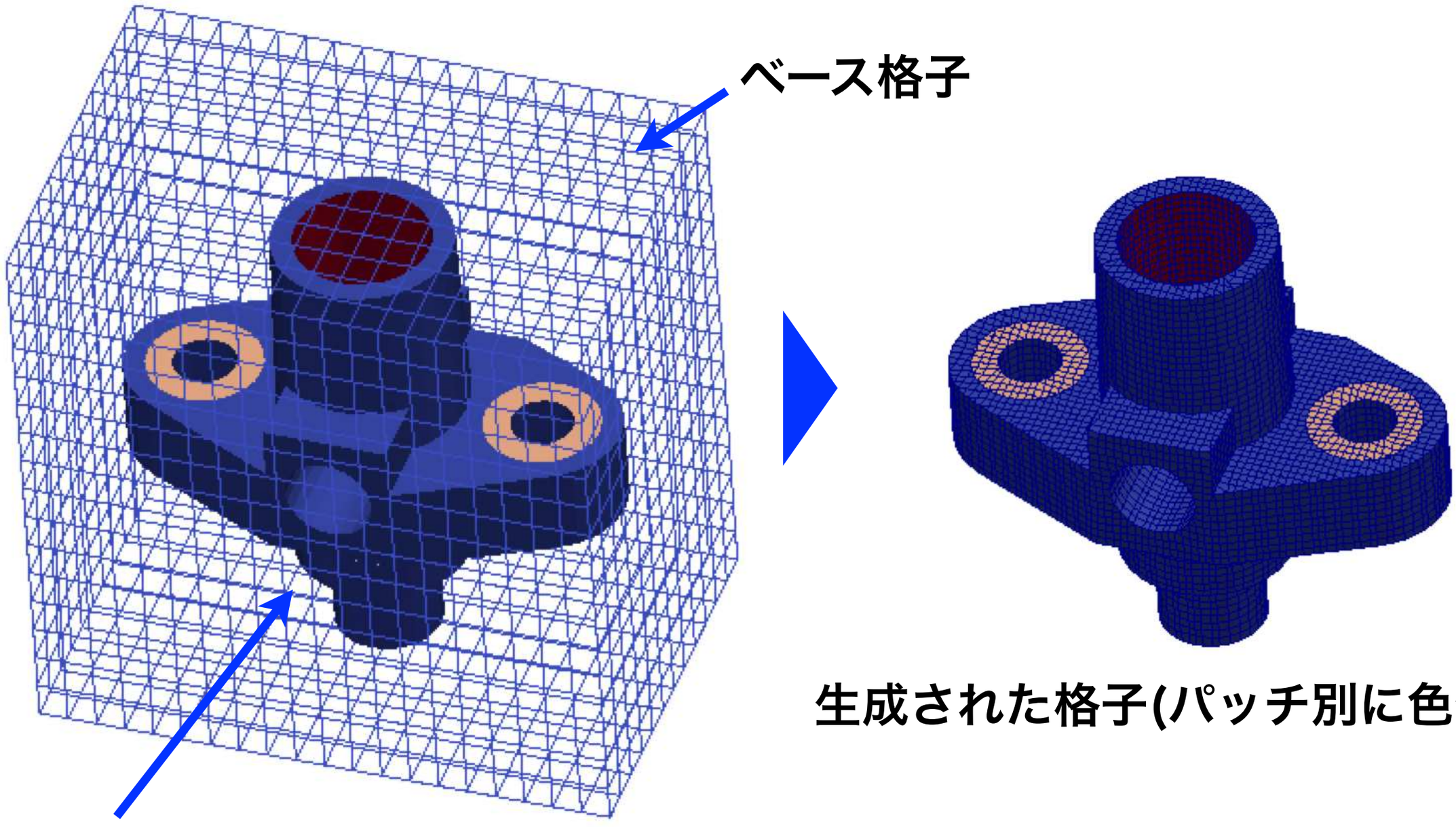
注)`overwrite`オプションを付けると
`constant/polyMesh`に上書き

`blockMesh` → `snappyHexMesh`

(図引用元: OpenFOAM User Guide)

snappyHexMeshによるflange格子生成

snappyHexMeshを用いて、フランジのCADデータ(STL形式)からフランジ内部の熱伝導解析用格子を作成するチュートリアル



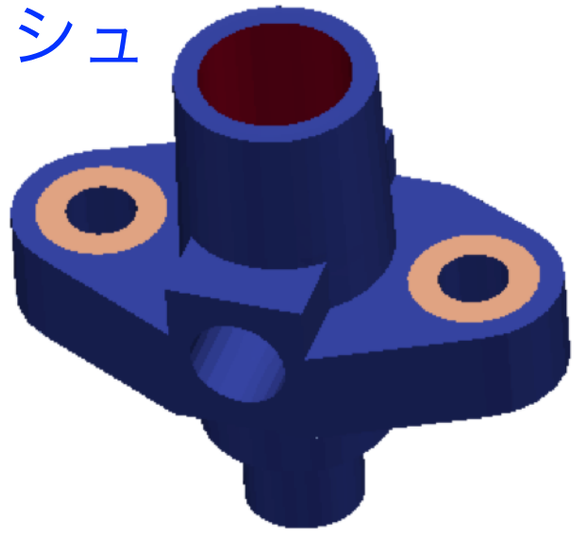
CADデータ (STL形式、solidブロック別に色分け)

生成された格子(パッチ別に色分け)

snappyHexMeshDict概要

system/snappyHexMeshDict

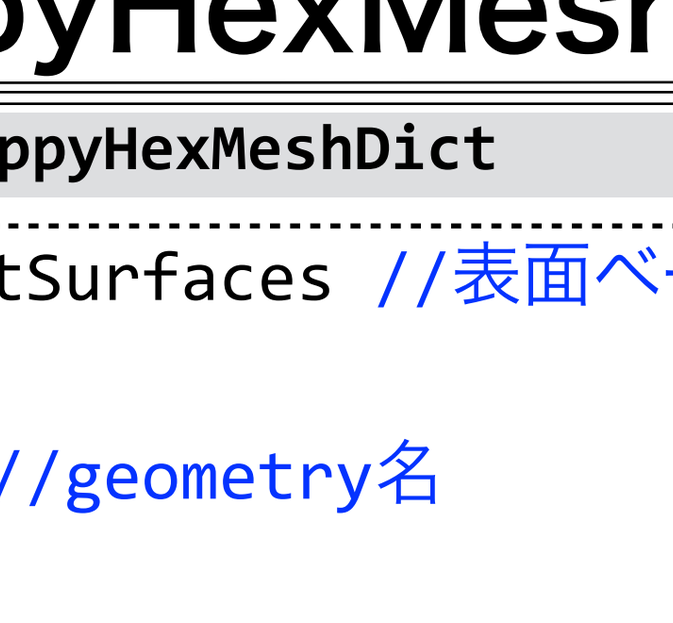
```
castellatedMesh true; //階段状格子
snap            true; //境界適合する
addLayers       false; //レイヤ付加しない

geometry
{
    flange.stl //フランジCADファイル（constant/triSurface下）
    {
        type triSurfaceMesh; //三角分割表面メッシュ
        name flange; //geometry名前
    }

    (略)
};
```



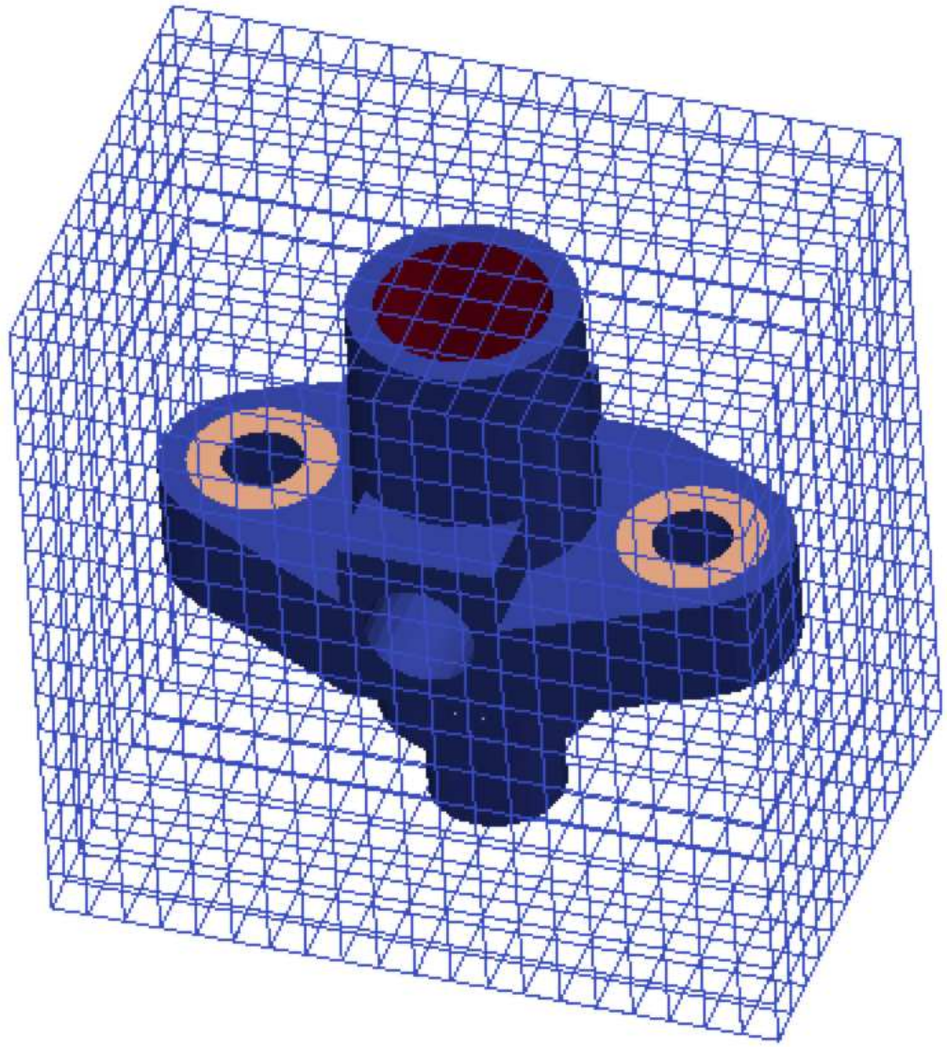
snappyHexMeshDict概要

system/snappyHexMeshDict

```
refinementSurfaces //表面ベースの細分割
{
  flange //geometry名
  {
    //(最小レベル、最大レベル)
    level (2 2);
  }
}
```



レベル2(表面)



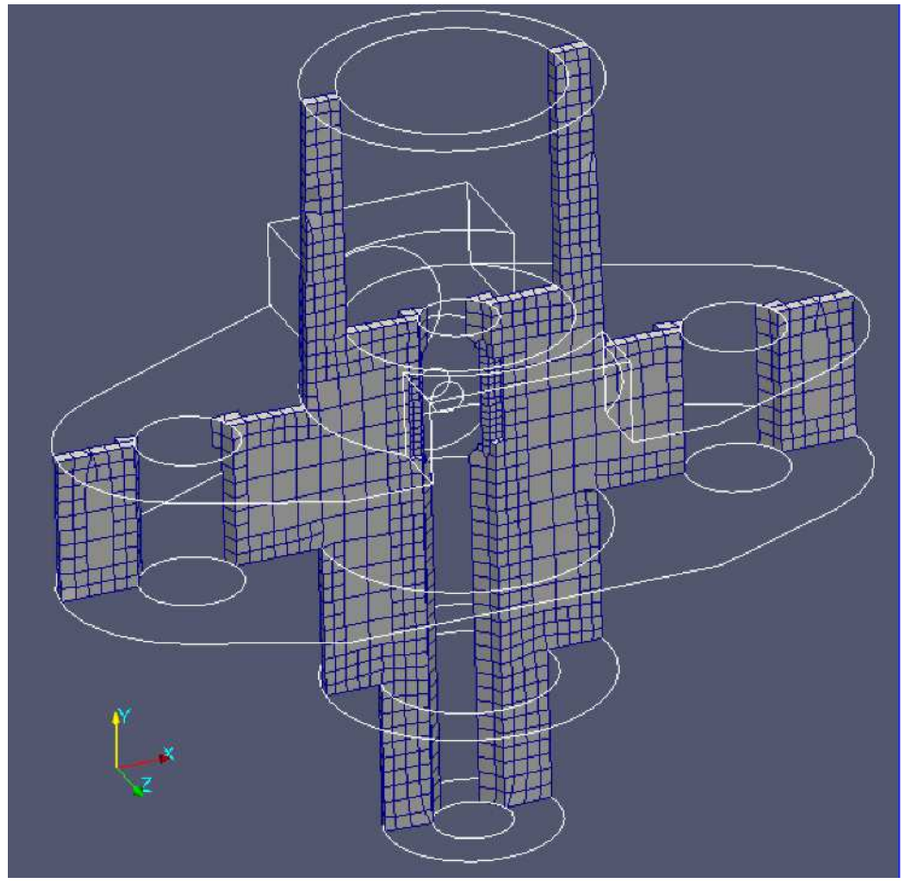
レベル0(ベース格子)

flangeチュートリアルの実行と格子可視化

```
cdw
cd lecture/tutorials/mesh/snappyHexMesh/flange/
cp ~/lecture/foamRunTutorials.sh ./
qsub foamRunTutorials.sh
```

ParaViewで可視化(適宜データを転送してから実行)

1. Use VTKPolyhedron *check*→Mesh Parts *select all*→Apply
2.Filters/Extract Block/Block Indices/ patches *select*→Apply
3. Filters/Feature Edges→Apply→Coloring/Solid Color
4. Pipeline browser/ExtractBlock1 *hidden*, flange.OpenFOAM *select*
5. Filters/Slice/Z Normal→Show Plane,Triangulate the slice *uncheck*, Crinkle slice *check*→Apply
6. Representation/Surface With Edges→ Coloring/Solid Color



特徴線(Feature edegs)フィルターは内部(internalMesh)に適用できないので，Extract Blockフィルターで internalMesh以外のpachesのみ抽出する

module版OpenFOAM 3.0.1の利用

~/.bashrc（下記内容を加え，. ~/.bashrc を実行）

```
alias OF301mpt='\
module unload intel intel-mpi mpt openfoam;\
module load intel/16.0.3.210;\
module load mpt/2.14;\
unset WM_PROJECT_DIR;\
module load openfoam/3.0.1-mpt;\
. $WM_PROJECT_DIR/etc/bashrc\
'
alias OF301impi='\
module unload intel intel-mpi mpt openfoam;\
module unload openfoam;\
module load intel/16.0.3.210 ;\
module load intel-mpi/5.1.3.210;\
unset WM_PROJECT_DIR;\
module load openfoam/3.0.1;\
. $WM_PROJECT_DIR/etc/bashrc\
'
```

SGI MPT版 OpenFOAM 3.0.1 の環境設定

```
OF301mpt
```

INTEL MPI版 OpenFOAM 3.0.1 の環境設定

```
OF301impt
```

INTEL MPI版OpenFOAM 4.0, v1606+の利用

~/.bashrc（下記内容を加え，. ~/.bashrc を実行）

```
alias OF40impi='\
module unload intel intel-mpi mpt openfoam;\
module load intel/16.0.3.210;\
module load intel-mpi/5.1.3.210;\
. /lustre/app/openfoam/4.0/OpenFOAM-4.0/etc/bashrc\
'
alias OF1606impi='\
module unload intel intel-mpi mpt openfoam;\
module load intel/16.0.3.210;\
module load intel-mpi/5.1.3.210;\
. /lustre/app/openfoam/1606+/OpenFOAM-v1606+/etc/bashrc\
'
```

INTEL MPI版 OpenFOAM 4.0の環境設定

```
OF40impi
```

INTEL MPI版 OpenFOAM v1606+の環境設定

```
OF1606impi
```

OpenFOAM独自ビルドの準備

OpenFOAMディレクトリの作成とホームディレクトリからのリンク

```
cdw
mkdir OpenFOAM
cd OpenFOAM
ln -s $PWD ~/
```

計算ディレクトリやホームディレクトリに元々OpenFOAMディレクトリがある場合には，それらを名称変更または移動してから実行

独自ビルド用にシステムにインストール済のOpenFOAMをコピー(時間がかかります)

```
cp -a /lustre/app/openfoam/4.0/OpenFOAM-4.0 ./ &
cp -a /lustre/app/openfoam/4.0/ThirdParty-4.0 ./ &
cp -a /lustre/app/openfoam/1606+/OpenFOAM-v1606+ ./ &
cp -a /lustre/app/openfoam/1606+/ThirdParty-v1606+ ./ &
cp -a /lustre/app/openfoam/1612-mpt/OpenFOAM-v1612+ ./ &
cp -a /lustre/app/openfoam/1612-mpt/ThirdParty-v1612+ ./ &
```

独自ビルド用のOpenFOAM環境設定

~/.bashrc（下記内容を加え，. ~/.bashrc を実行）

```
alias OF40mpt='\
module unload intel intel-mpi mpt openfoam;\
module load intel/16.0.3.210;\
module load mpt/2.14;\
. ${HOME/home/lustre}/OpenFOAM/OpenFOAM-4.0/etc/bashrc WM_MPLIB=SGIMPI\
'
alias OF1606mpt='\
module unload intel intel-mpi mpt openfoam;\
module load intel/16.0.3.210;\
module load mpt/2.14;\
. ${HOME/home/lustre}/OpenFOAM/OpenFOAM-v1606+/etc/bashrc WM_MPLIB=SGIMPI\
'
alias OF1612impi='\
module unload intel intel-mpi mpt openfoam;\
module load intel/17.0.2.174;\
module load intel-mpi/2017.2.174;\
export MPI_ROOT=$I_MPI_ROOT;\
. ${HOME/home/lustre}/OpenFOAM/OpenFOAM-v1612+/etc/bashrc WM_MPLIB=INTELMPI\
'
```

SGI MPT版OpenFOAM-4.0のビルド

独自ビルド用OpenFOAM-4.0の設定変更

```
vi ~/OpenFOAM/OpenFOAM-4.0/etc/bashrc
```

```
#export FOAM_INST_DIR=${BASH_SOURCE%/*/*/*} || \
#export FOAM_INST_DIR=$HOME/$WM_PROJECT
export FOAM_INST_DIR=/lustre/app/openfoam/4.0
```

旧

```
export FOAM_INST_DIR=${BASH_SOURCE%/*/*/*} || \
export FOAM_INST_DIR=$HOME/$WM_PROJECT
~~export FOAM_INST_DIR=/lustre/app/openfoam/4.0~~
```

新(元に戻す)

MPIライブラリ依存のライブラリをビルド

```
OF40mpt
cd $WM_THIRD_PARTY_DIR
./Allwmake
src
cd Pstream/
./Allwmake
cd ../parallel/
./Allwmake
```

MPIライブラリ依存のライブラリをビルド

OpenFOAM-v1606+のSGI MPT依存のライブラリをビルド

```
OF1606mpt
cd $WM_THIRD_PARTY_DIR
./Allwmake
src
cd Pstream/
./Allwmake
cd ../parallel/
./Allwmake
```

boost, CGALのビルドに失敗するが無視する

OpenFOAM-v1612+のINTEL MPI依存のライブラリをビルド

```
OF1612impi
cd $WM_THIRD_PARTY_DIR
./Allwmake
src
cd Pstream/
./Allwmake
cd ../parallel/
./Allwmake
```

boost, CGALのビルドに失敗するが無視する