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
2019年5月9日@東京大学情報基盤センター遠隔会議室

OpenFOAM入門

今野　雅
(株式会社OCAEL・東京大学客員研究員)

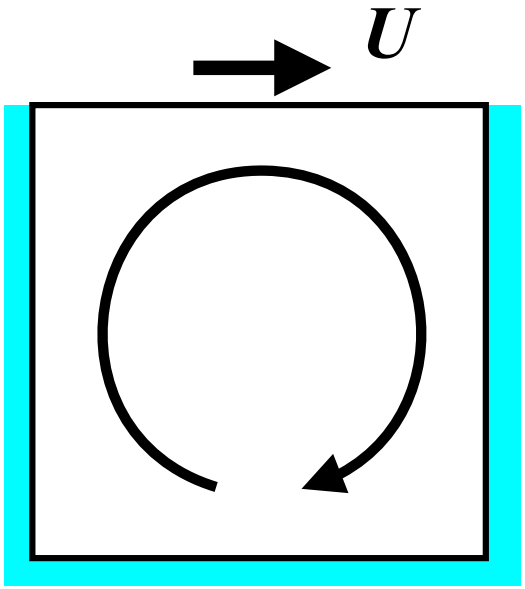
講習会プログラム(予定)

- 10:00-10:40 **Oakforest-PACSへのログイン**
- 10:40-11:00 **Oakforest-PACS概要**
- 11:00-11:30 **OpenFOAM概要**
- 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
- 14:15-15:45 **キャビティ流れ演習II**
  - ✓ ParaViewによる解析結果可視化
  - ✓ postProcess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基礎
  - ✓ snappyHexMeshによる格子生成基礎
  - ✓ その他チュートリアルの実行

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

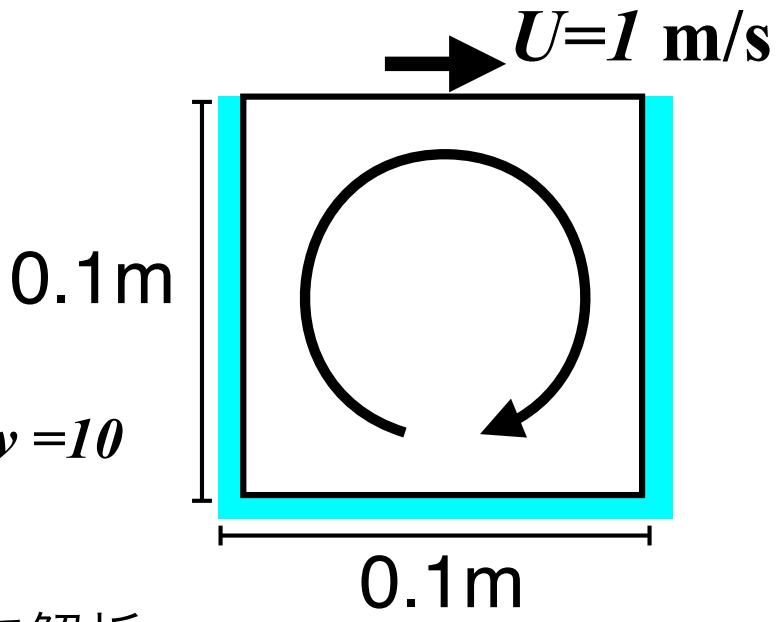


[Ghia 1982] U Ghia, K.N Ghia, C.T Shinl: High-Re solution for incompressible flow using the Navier-Stokes equations and the multigrid method. J. Comput. Phys., 48:387–411, 1982. URL

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)：$\nu=0.01$ m²/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

  ✓ 非定常非圧縮性層流解析ソルバicoFoamで解析



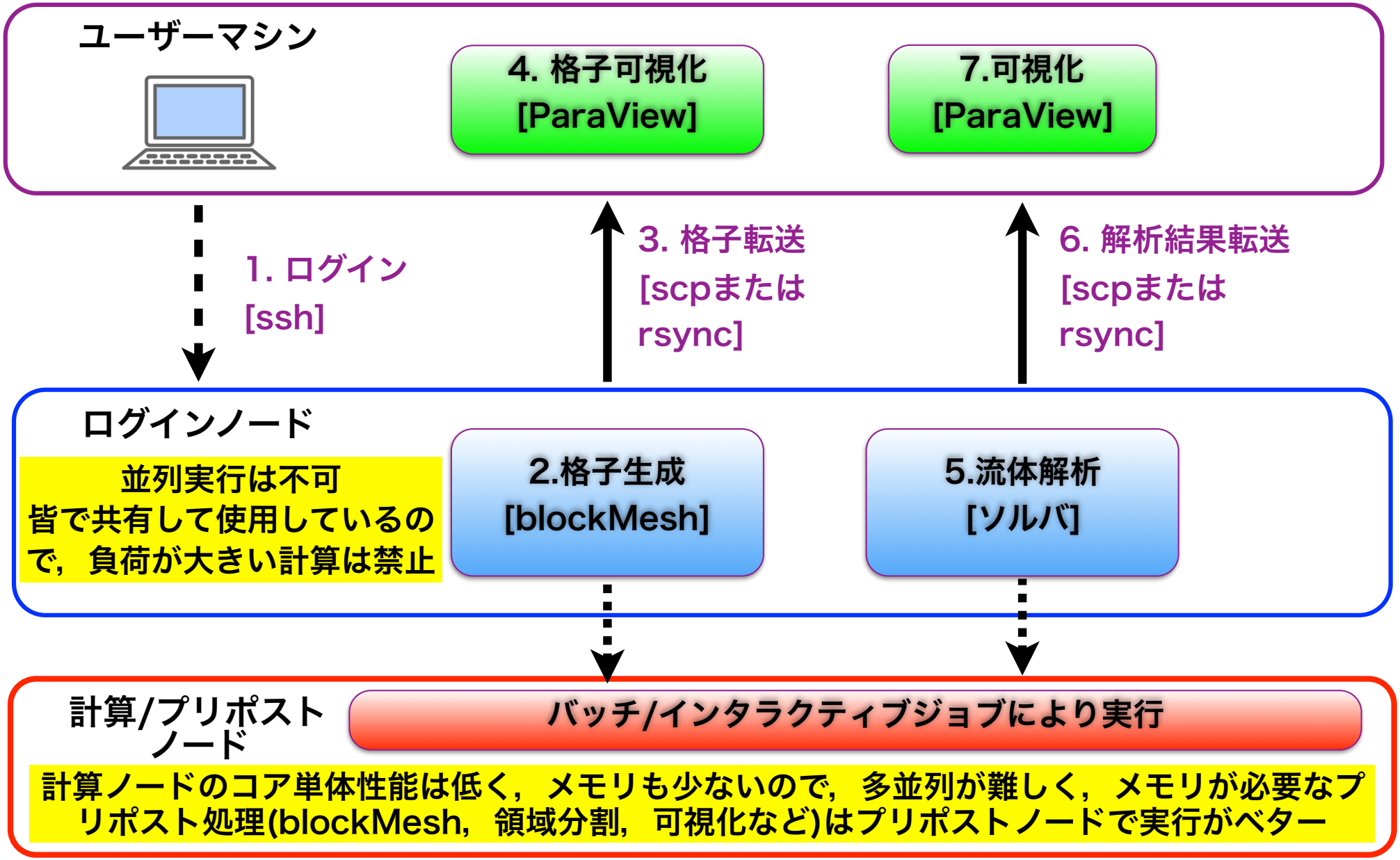
非定常非圧縮層流解析ソルバicoFoamの基礎方程式

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

スパコンでのOpenFOAMの代表的な解析手順



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

OFPへのログイン (X転送する-Yオプションを付ける)

```
ssh -Y txxxxx@ofp.jcahpc.jp # txxxxxは利用番号
xev                         # X転送が可能か調べる．Ctrl-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

```
cd /work/gt00/$USER
```

講習会用ファイルの展開

```
cp -a /work/gt00/share/lecture-OpenFOAM-materials/cavity/lec-cavity ./
```

講習会用ディレクトリの参照を容易にするためにシンボリックリンクを貼る

```
ln -s /work/gt00/$USER/lec-cavity ~/
```

講習会用ディレクトリへの移動と中身確認

```
cd ~/lec-cavity
ls
```

```
bin foamRunTutorials.sh share
```

OpenFOAM関連のmodule

標準でloadされているmoduleの表示

```
module list
```

```
Currently Loaded Modulefiles:
  1) impi/2018.1.163    2) intel/2018.1.163
```

利用可能なmoduleの表示

```
module avail -l #-lオプションで更新日などの詳細が表示される
```

```
- Package ----------------------------+- Versions -+- Last mod. ------
:  # OFのmoduleはgcc版なので表示されない
```

全moduleのunloadと利用可能なmoduleの表示

```
module purge
module avail -l
```

```
gcc/4.8.5                                          default      2017/02/22  5:03:57
gcc/4.9.4                                                       2017/04/28  9:50:51
```

gcc moduleのload

```
module load gcc #/バージョンを省略すると，defaultのバージョンが指定される
#gcc/4.8.5をloadするとIntel MPIのmodule impi/2017.3.196が自動的にload
```

OpenFOAM関連のmodule(続き)

利用可能なmoduleの表示

```
module avail -l
```

```
- Package ----------------------------+- Versions -+- Last mod. ------
/home/opt/local/modulefiles/L/mpi/gcc/4.8.5/impi/2017.3.196:
openfoam/3.0.1                                default     2017/02/23  5:30:01
openfoam/4.1                                              2017/07/26  8:43:28
```

OpenFOAMは3.0.1(デフォルト)と4.1．ビルド環境：Gcc-4.8.5，Intel MPI 2017.3，整数ラベル・実数変数サイズ64bit．最適化フラグが-O3なので，ログイン/プリポスト/計算ノードで動作する

```
/home/opt/local/modulefiles/L/compiler/gcc/4.8.5:
impi/2017.1.132                                           2017/02/23  1:54:40
:
impi/2017.3.196                               default     2017/06/30  4:26:35
impi/5.1.3.258
```

MPIライブラリはIntel MPI 2017.3がデフォルトで使用される

```
/home/opt/local/modulefiles/L/core:
vtune/2016.4.0.470476                                     2016/11/11 11:34:22
:
vtune/2018.1.0.535340                         default     2017/12/20  8:29:14
```

性能プロファイラ(Intel VTune)のmoduleも使用できる

module版OpenFOAM 4.1の環境設定

module openfoam/4.1のhelp表示

```
module help openfoam/4.1
```

```
(2) Make a job script
    [username@ofp01 ~]$ vi run.sh
    #!/bin/sh
バッチジョブ独自の設定
    #PJM -L rscgrp=regular
    #PJM -L node=1
    #PJM -L elapse=1:00:00
    #PJM -j
module版OF-4.1の環境設定部分
    module purge                          標準で有効なmoduleをpurgeで全てunload
    module load gcc/4.8.5                 Gcc-4.8.5のmoduleをload
    module load openfoam/4.1              OF-4.1のmoduleをload
    source $WM_PROJECT_DIR/etc/bashrc     OpenFOAMの環境設定
```

module版はKNL独自のバイナリではなく，Intel Xeon機のログイン/プリポストノードでも動作するので，module helpでのjob scriptにおける環境設定部分を実行すれば，ログイン/プリポストノードでもOpenFOAMのコマンドが実行可能(ただし，ログインノードでは並列計算は禁止)

module版OpenFOAMのエイリアス設定

module版OpenFOAMのエイリアス設定

```
more ~/lec-cavity/etc/bashrc
```

```
alias OF41='\
module purge;\
module load gcc/4.8.5;\
module load openfoam/4.1;\
source $WM_PROJECT_DIR/etc/bashrc'
alias OF301='\
module purge;\
module load gcc/4.8.5;\
module load openfoam/3.0.1;\
source $WM_PROJECT_DIR/etc/bashrc'
```

emacs，nano，geditなどの使い慣れたエディタを使用し赤字部分を追加する．なお，X転送せずに端末内でemacsを起動するには emacs -nw

ついでに，OpenFOAM 3.0.1用のエイリアスも定義する．様々なバージョンやビルド環境のOF設定の切替にも便利

上記を~/.bashrc の末尾に足して.bashrc を実行し，エイリアスを有効にする

```
cat ~/lec-cavity/etc/bashrc >> ~/.bashrc
source ~/.bashrc
```

module版OpenFOAM 4.1の環境設定

```
OF41
```

module版OpenFOAMの特徴

- 現在用意されているのはOpenFOAM Foundation版の3.0.1と4.1のみ
  - OpenFOAM Foundation版は，MPIのメモリ使用量最適化がなされておらず，概ね2,000並列以上ではメモリ使用量が莫大に増加するので，大規模並列計算ではv1812等のPlus版を使用する必要がある
- 整数ラベルbit数が64でビルドされている
  - 64bit整数の範囲の格子・界面数が扱えるが，32bit に比べメモリ使用量が増え，計算時間も僅かに増加する
- コンパイラはGcc-4.8.5
  - Intelコンパイラを用いてビルドしたほうが僅かに速い場合がある
  - Intelコンパイラを用いた場合，KNL用最適化フラグ(-xmic-avx512等)を付加してビルドが可能であり，僅かに速度が向上する．ただし，この場合，ログインノードやプリポストノードでは動作しない
- moduleが用意されていないOpenFOAMのバージョンやコンパイラ，最適化フラグ，整数ラベルbit数，MPIライブラリ等の異なる設定については，OpenFOAM自動ビルドスクリプト installOpenFOAM 等を使用して，ソースからビルドする必要がある

OpenFOAMの環境設定によるエイリアス

alias

```
alias app='cd $FOAM_APP’
alias foam='cd $WM_PROJECT_DIR’
alias lib='cd $FOAM_LIBBIN'
alias run='cd $FOAM_run'
alias sol='cd $FOAM_SOLVERS'
alias src='cd $FOAM_SRC'
alias tut='cd $FOAM_TUTORIALS'
alias util='cd $FOAM_UTILITIES’
alias wmSet='. $WM_PROJECT_DIR/etc/bashrc'
alias wmUnset='. $WM_PROJECT_DIR/etc/config.sh/unset’
alias wmDP='wmSet WM_PRECISION_OPTION=DP'
alias wmSP='wmSet WM_PRECISION_OPTION=SP'
alias wm32='wmSet WM_ARCH_OPTION=32'
alias wm64='wmSet WM_ARCH_OPTION=64'
alias wmSchedOn='export WM_SCHEDULER=$WM_PROJECT_DIR/wmake/
wmakeScheduler’
alias wmSchedOff='unset WM_SCHEDULER'
```

アプリケーションのソースに移動
インストールディレクリに移動
ライブラリのディレクリに移動
ユーザの実行用ディレクトリに移動
ソルバのソースに移動
ソース・ディレクトリに移動
チュートリアルディレクトリに移動
ユーティリティのソースに移動

環境設定更新
環境設定初期化

実数倍精度環境へ変更
実数単精度環境へ変更
32bit環境へ変更
64bit環境へ変更

wmake(コンパイルスクリプト)の並列
ビルドスケジューラの設定On/Off

cavityチュートリアルケースのコピー

cavityチュートリアルケースのコピー

```
cd ~/lec-cavity
cp -r $FOAM_TUTORIALS/incompressible/icoFoam/cavity/cavity ./
# $FOAM_TUTORIALS はチュートリアルのディレクトリを示す環境変数
cd cavity
```

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コマンド：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が，OFPにはインストール済み)

```
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

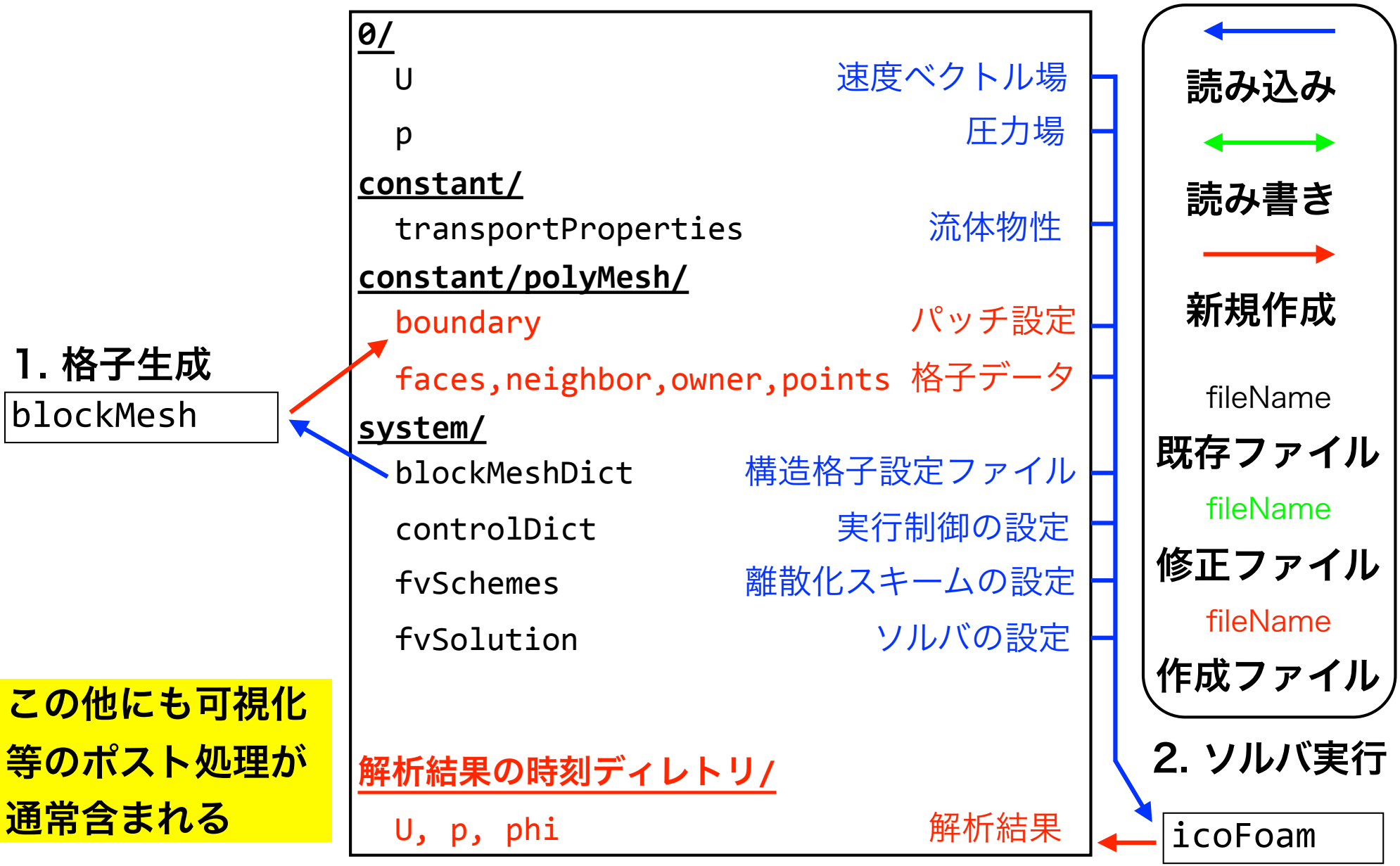
ジョブスクリプトなどの講習用ファイルのコピー

```
cp -a ../share/* ./
```

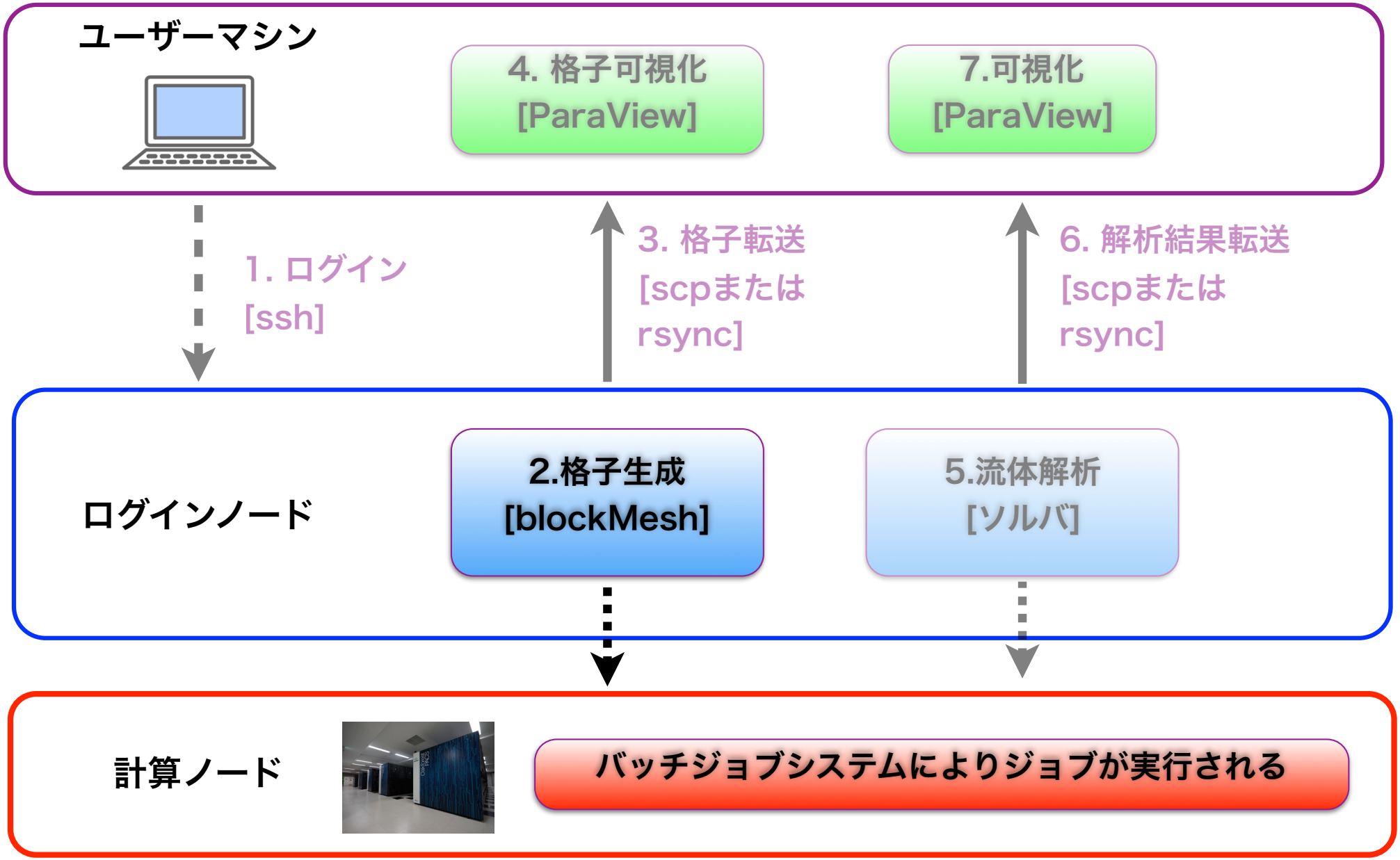
キャビティケースのディレクトリ構成

```
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
```

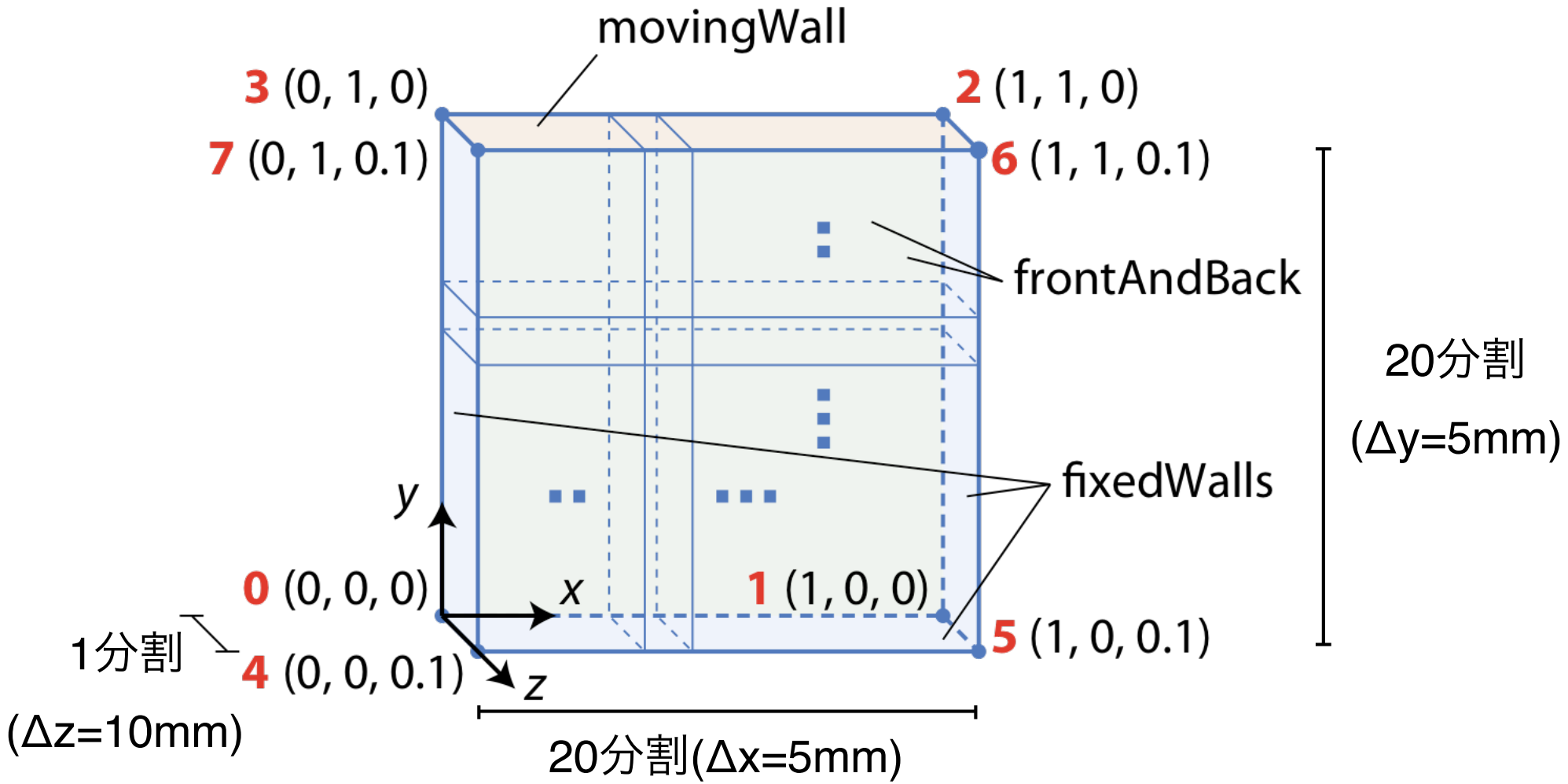
キャビティケースの解析手順



blockMeshによる格子生成



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:  4.x                                   |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     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%)
```

エディタでは間違って修正を行なう可能性があるので，ファイルの修正を必要としない場合，moreなど閲覧専用コマンド(ページャー)で中身を確認するのが望ましい．
moreを機能拡張したlessや，エディタviを閲覧専用にしたviewなど，様々なコマンドがある

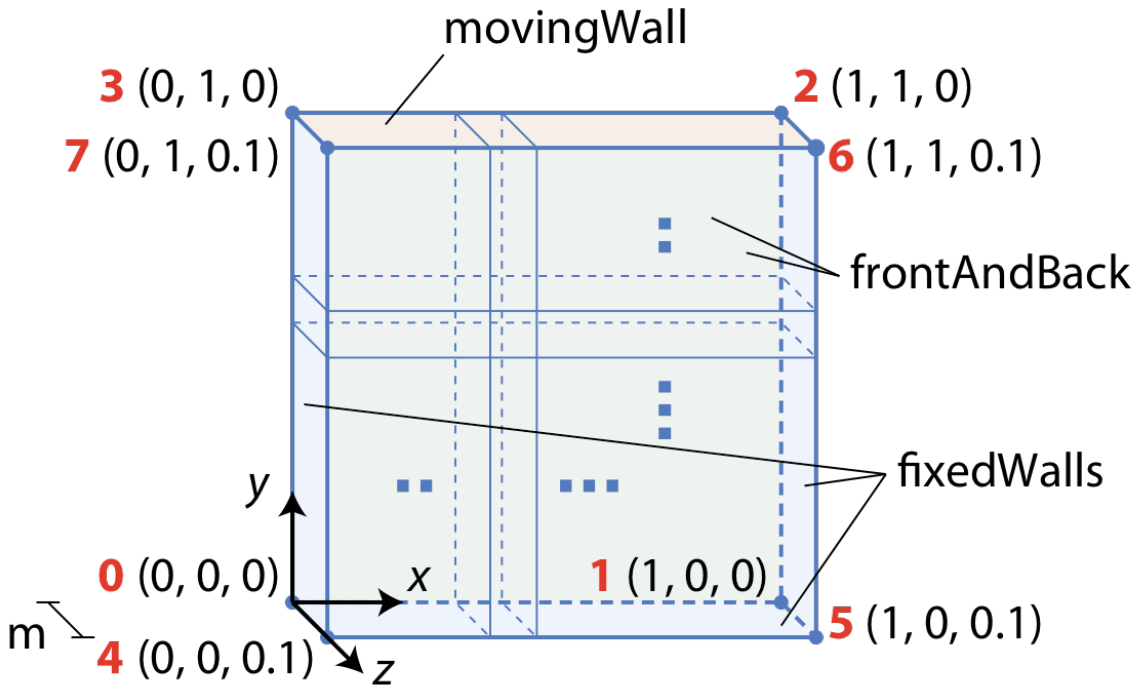
moreコマンドは続ける(英語版ではmore)と表示してキー入力待ちとなる．

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

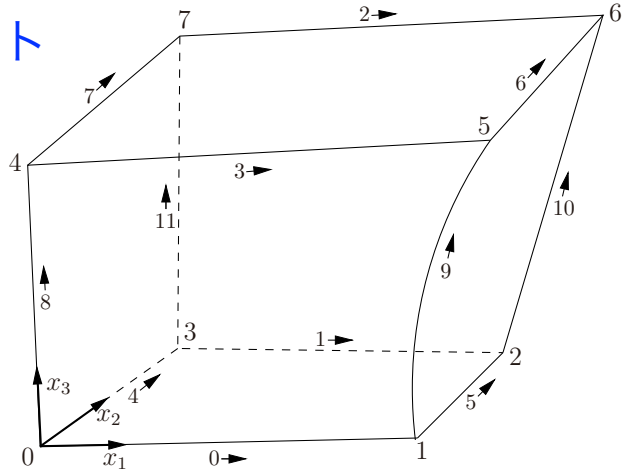
```
convertToMeters 0.1; //メートル単位への変換係数

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
);
```

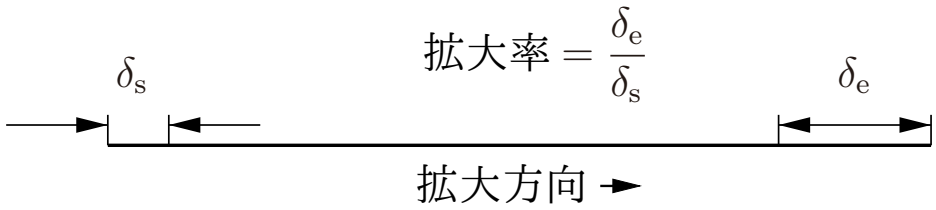


設定ファイルblockMeshDict (続き)

system/blockMeshDict

```
blocks //格子ブロックの定義
(
  hex //形状: hex=六面体 (各ブロックは構造格子なので, 常にhex)
  (0 1 2 3 4 5 6 7) //verticesで定義した頂点番号のリスト
  //頂点番号はどの点から開始しても良い
  //分割数や格子拡大率の方向がこれで定まる
  //x1方向:0→1, x2方向:1→2, x3方向:0→4
  (20 20 1) //各方向の分割数
  simpleGrading //拡大率が各方向毎に一定
  (1 1 1) //各方向の拡大率(最初と最後の格子幅の比. 1=等間隔)
  //edgeGrading (1 1 1 1 2 2 2 2 3 3 3 3) のように各辺毎に拡大率の定義も可能
);

edges//辺が曲線の場合に指定
(
);
```



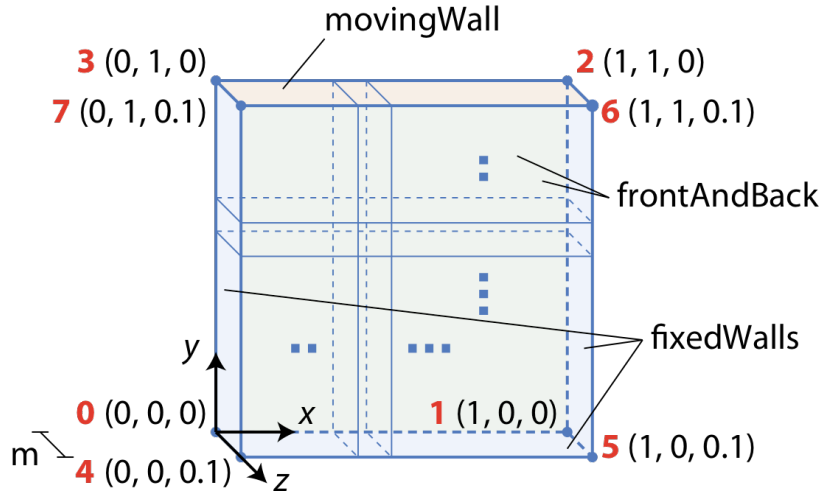


(図引用元: OpenFOAMユーザガイド)和訳

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

more ofp0mesh.sh #ジョブスクリプトの名前は任意

```
#!/bin/bash
#PJM -L rscgrp=lecture-flat
#PJM -L node=1
#PJM -L elapse=0:15:00
#PJM -g gt00
#PJM -S
module purge                              # 標準で有効なmoduleをpurgeで全てunload
module load gcc/4.8.5                     # Gcc-4.8.5のmoduleをload
module load openfoam/4.1                  # OpenFOAM-4.1のmoduleをload
source $WM_PROJECT_DIR/etc/bashrc # OpenFOAMの環境設定
blockMesh >& log.blockMesh                # 格子生成実行
```

-L rscgrp=lecture-flat ： リソースグループ名(必須)(講習中は tutorial-flat のほうが優先度が高いので，lecture-flatが混雑時には変更．ジョブ待ち確認: pjstat --rsc -b

-L node=1 ： ノード数指定(必須)

-L elapse=0:15:00 ： 経過時間制限値(15分=講習用リソースグループの最大値)

-g gt00 ： ジョブ実行時にトークンを消費するプロジェクト名(必須)(gt00=講習用)

-S ： ジョブ統計情報とノードごとの詳細情報をファイルに出力

blockMeshのジョブ投入

ジョブの投入

```
pjsub ofp0mesh.sh
```

```
[INFO] PJM 0000 pjsub Job JOB_ID submitted. #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)

```
pjstat  # 10文字を超えるジョブ名を全て表示させるには pjstat -l
```

→ ジョブ実行前の場合

```
Oakforest-PACS scheduled stop time: 2018/04/27(Fri) 09:00:00 (Remain:  1days 2:26:39)

JOB_ID       JOB_NAME   STATUS  PROJECT  RSCGROUP             START_DATE        ELAPSE           TOKEN NODE
JOB_ID       blockMesh. QUEUED  gt00     lecture-flat         --/-- --:--:--    00:00:00             -    1
```

→ ジョブ実行中の場合

```
JOB_ID       blockMesh. RUNNING gt00     lecture-flat         04/26 06:33:15<   00:00:03             -    1
```

→ 全ジョブ終了の場合

```
No unfinished job found.
```

ジョブの削除(今回は行わない)

```
pjdel JOB_ID
```

生成されたファイルの確認

```
tree
```

```
├── 0
│   ├── U
│   └── p
├── constant
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── faces
│   │   ├── neighbour
│   │   ├── owner
│   │   └── points
│   └── transportProperties
├── log.blockMesh
├── ofp0mesh.sh
├── ofp0mesh.sh.[eio]1234567
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── fvSchemes
    └── fvSolution
```

生成された格子ファイル

log.blockMesh: blockMeshのログ

1234567: ジョブID

*.e1234567 : ジョブの標準エラー出力

*.i1234567 : ジョブの統計情報

*.o1234567 : ジョブの標準出力

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 ofp0mesh.sh.e*
```

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

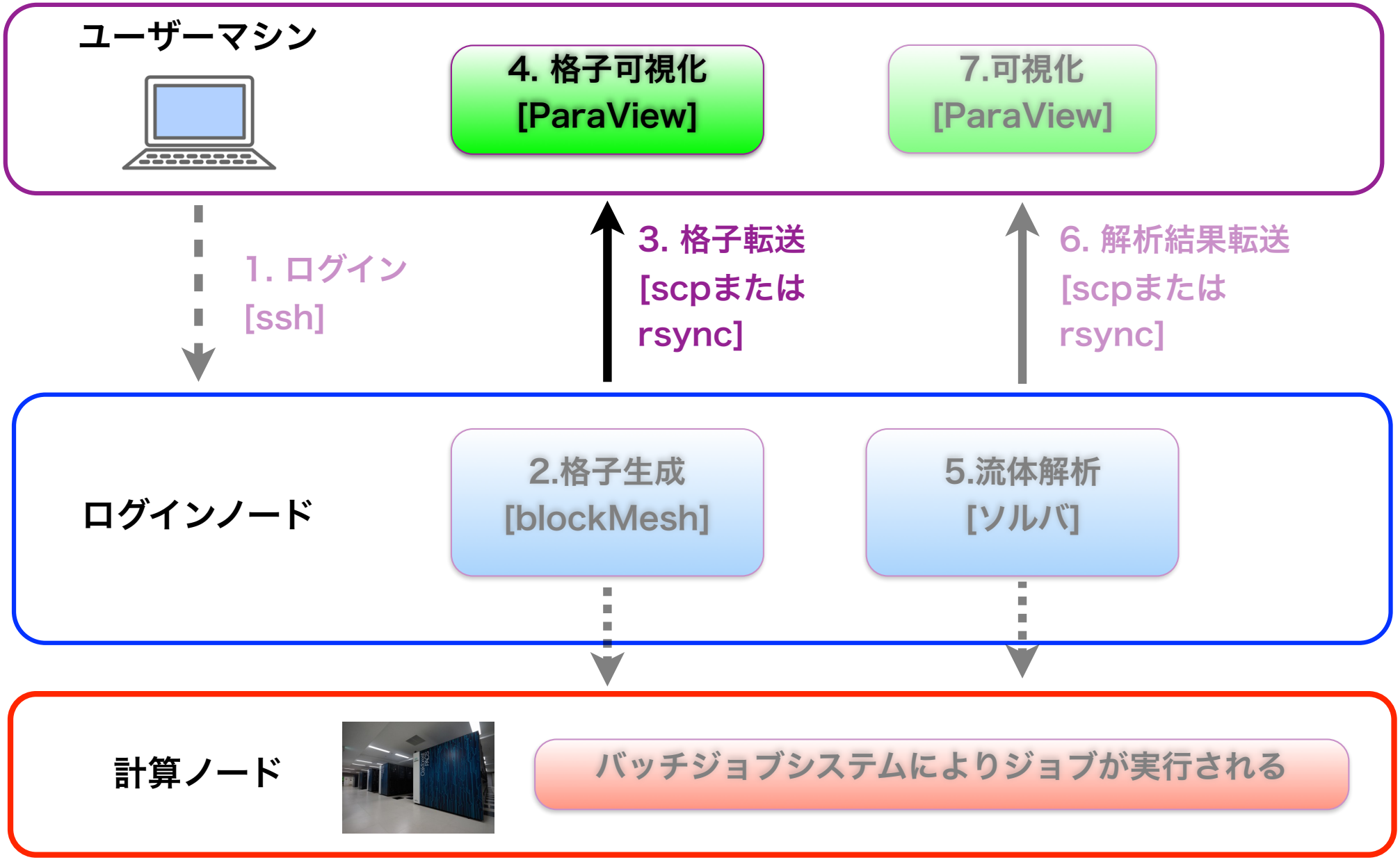
格子数

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



講習用ファイル一式と格子データの転送

ユーザーマシン(別端末)　　: ~/lec-cavity

ファイル一式を転送 [rsync]

ログインノード(ofp.jcahpc.jp)　　: ~/lec-cavity

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,

講習用ファイル一式と作成した格子データの転送 (以降, 赤の実線枠は別端末で実行)

```
cd #ホームディレクトリに移動
mkdir lec-cavity #講習用ディレクトリを作成
rsync -auv txxxxx@ofp.jcahpc.jp:lec-cavity/ ~/lec-cavity/ #転送
#転送元と転送先どちらもに/(スラッシュ)を付ける
#txxxxxは利用者番号. passphraseを聞かれた場合には, 登録したものを入力する
#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)
```

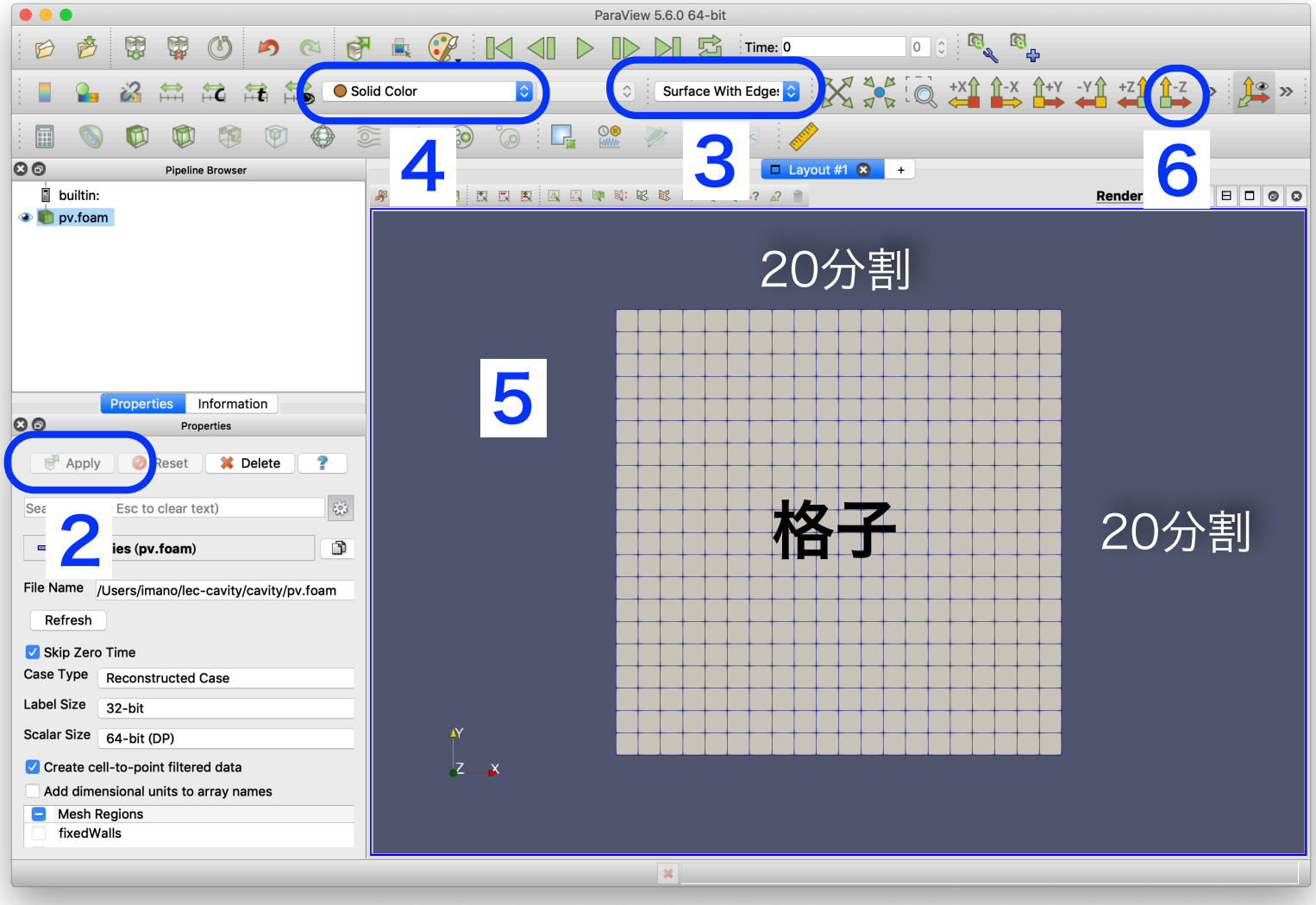
キャビティケースに移動(別端末で実行)

```
cd ~/lec-cavity/cavity/
```

ParaViewによる格子の可視化

ParaView用ダミーファイル(.foam)の作成(別端末で実行)

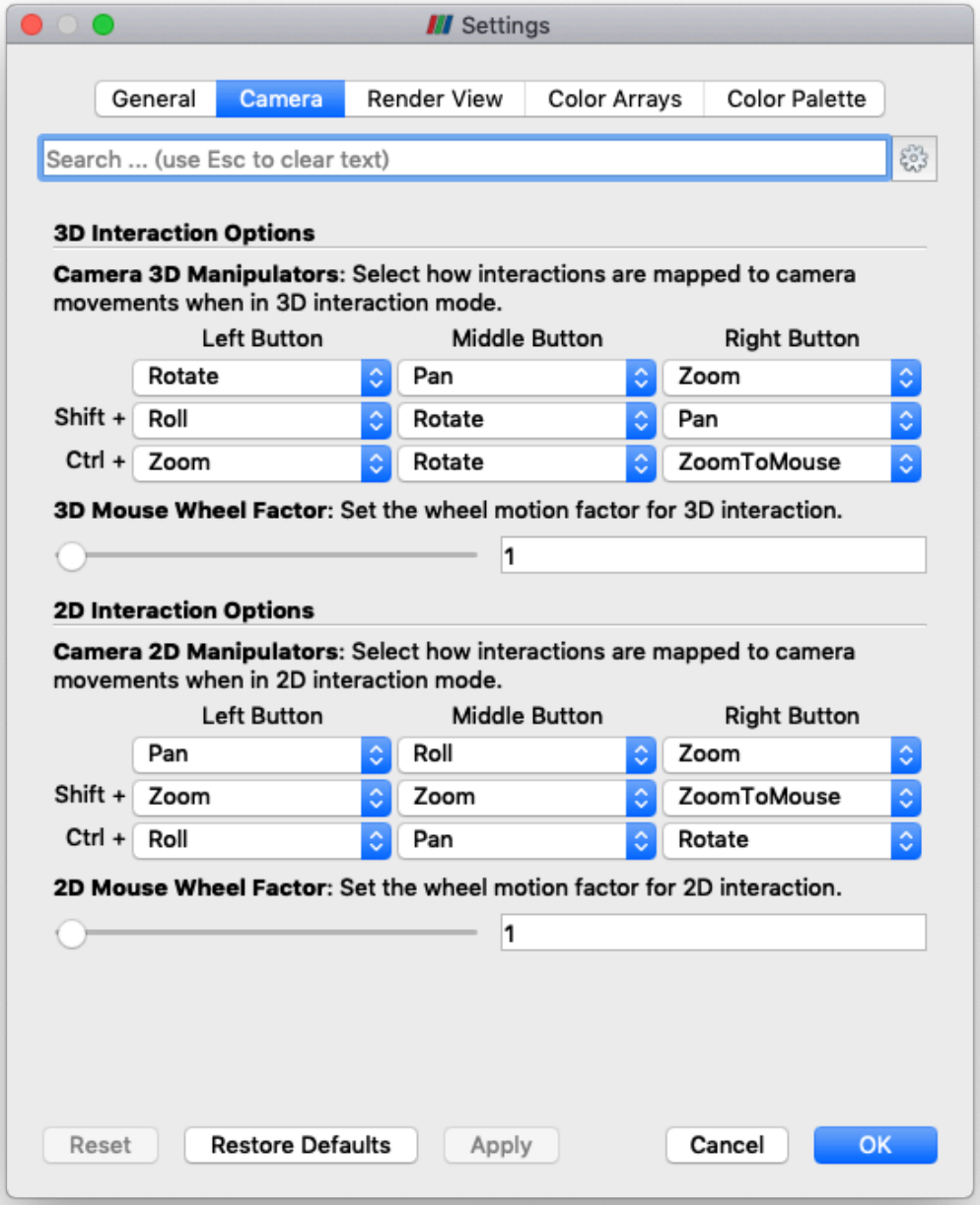
```
touch pv.foam
```



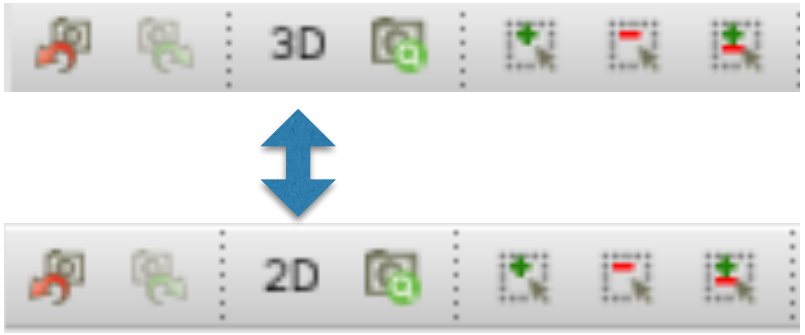
ParaViewで以下の操作を行う

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(選択済の場合もある)
5. マウスで動す
6. Z軸を画面の法線方向にする

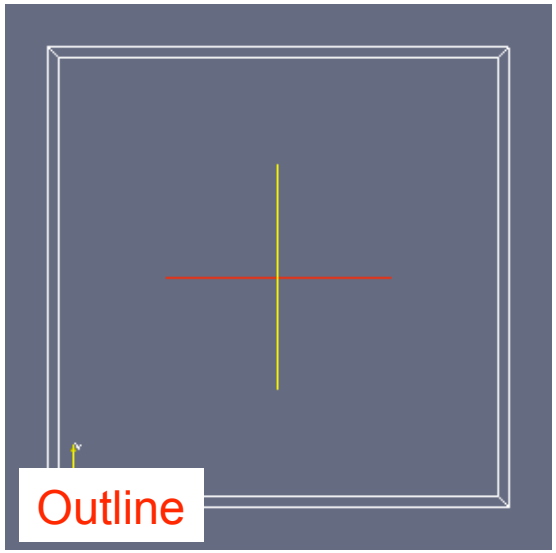
ParaViewのカメラ3D・2D操作確認・設定

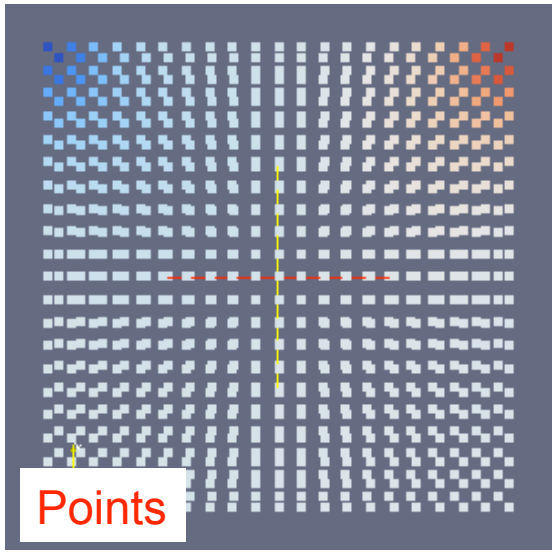


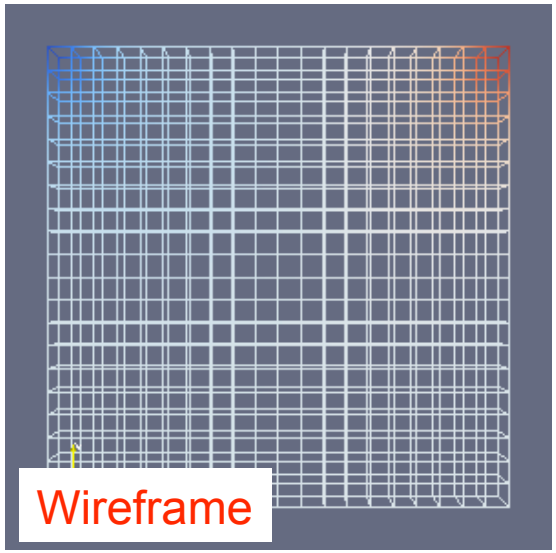
- RenderView画面における，カメラの3D操作や2D操作の設定を確認
  - ✓ ParaViewメニュー/Preferences/Cameraタブ
  - ✓ 設定を変更する事も可能
  - ✓ Restore Defaultsで元に戻せる
- 3D操作と2D操作の切り替え
  - ✓ RenderView画面の左上の3Dまたは2Dボタンを押す.

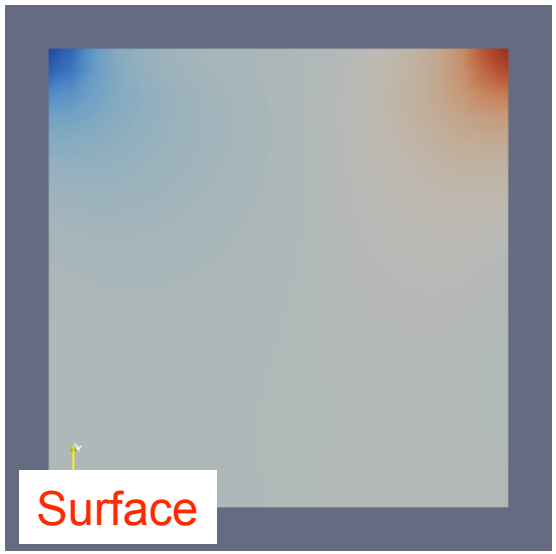


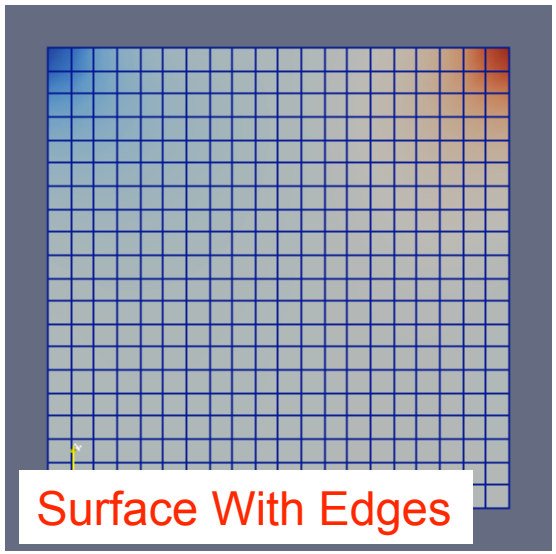
ParaViewの表示方法(Representation)

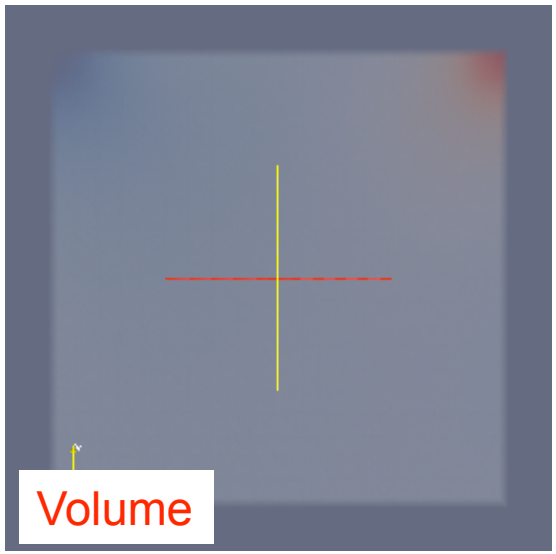






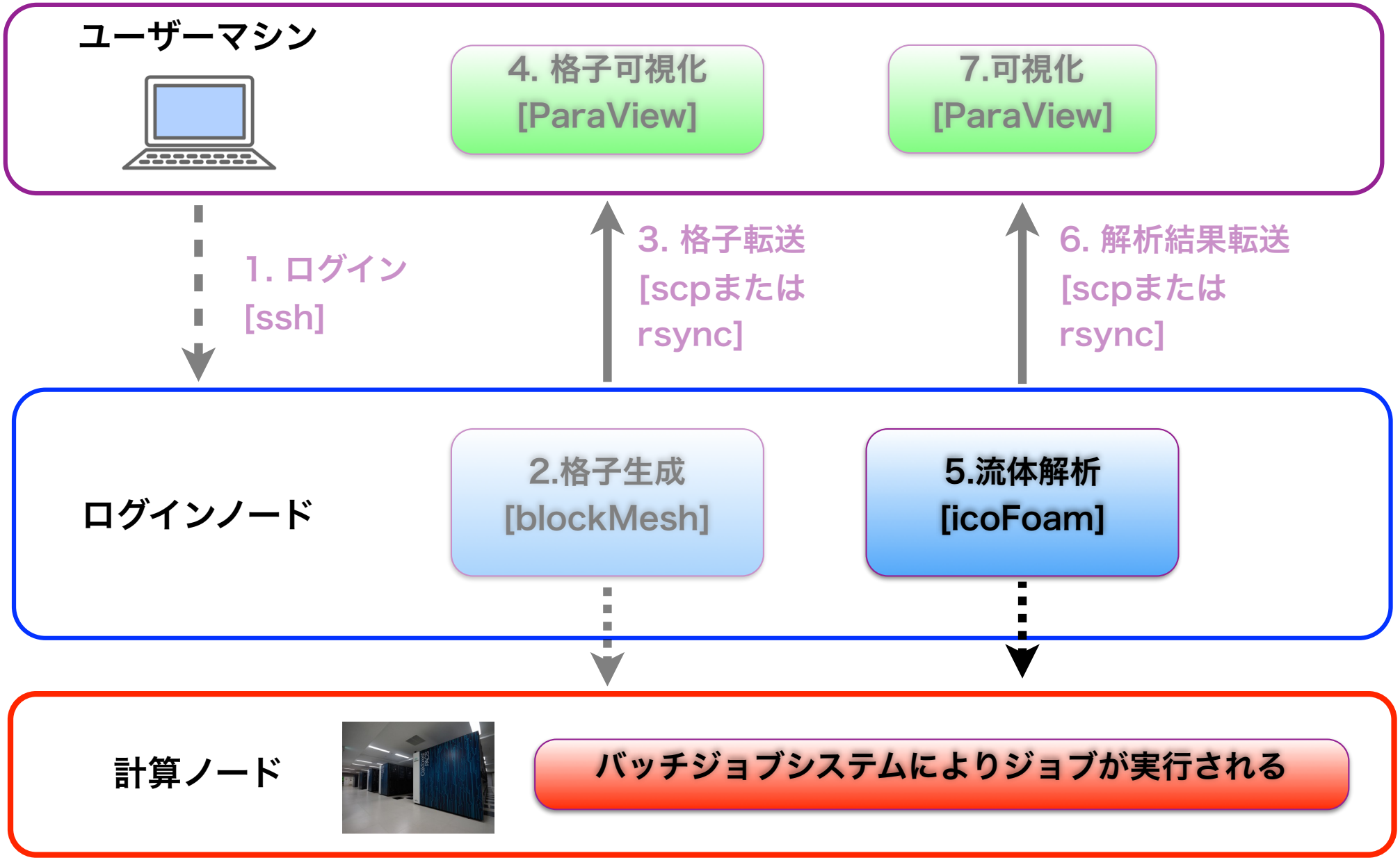






(図引用元: 大嶋 拓也 (新潟大学)「キャビティ流れの解析、paraFoamの実習」 第一回OpenFOAM講習会)

icoFoamによる流れ解析



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)
```

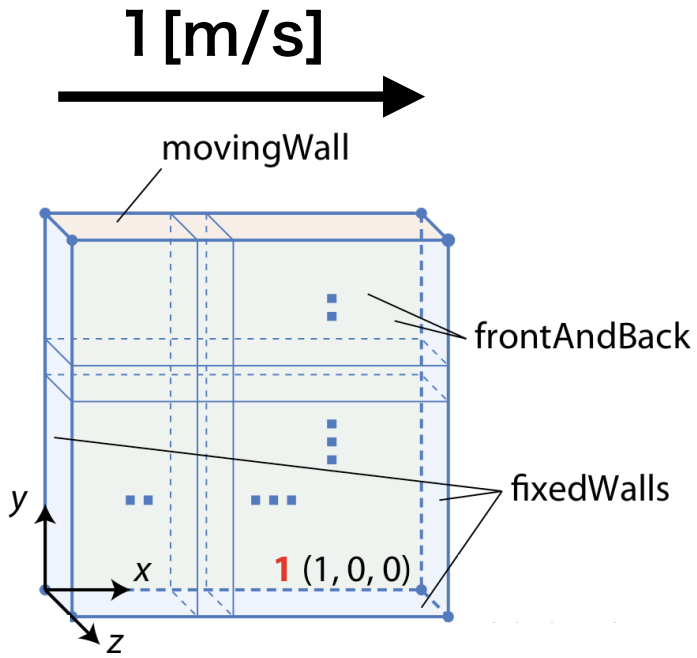
各演算では左辺・右辺での単位一致の検査がなされる

→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
```

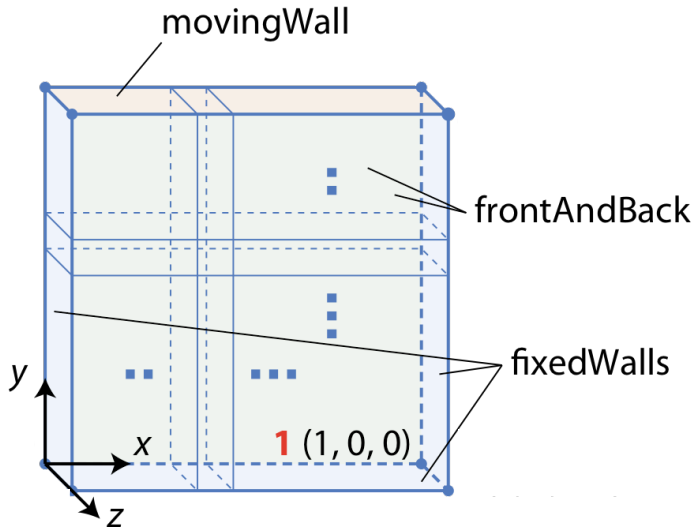
0/p

```
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)-1・(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          単位の次元[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
    relTol 0.05; //収束判定閾値(初期残差との比)
  }
  pFinal          //PISO法最終反復での圧力
  {
    $p;
    relTol           0;
  }
}
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
run TimeModifiable 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
```

```
FoamFile
{
:
}

functions
{
#includeFunc residuals
}
```

emacs，nano，geditなどの使い慣れたエディタを使用し赤字部分を追加
なお，X転送せずに端末内でemacsを起動するには emacs -nw

FoamFile{..}の後であれば，挿入位置はどこでも良い．

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
```

逐次実行ジョブスクリプトとジョブ投入

```
more ofp1sol.sh
```

ofp1sol.sh（逐次実行ジョブ用スクリプト）

```
(ofp0mesh.shと同様なので略)
source $WM_PROJECT_DIR/etc/bashrc # OpenFOAMの環境設定
# ソルバ実行
# numactl -p1 : MC-DRAMに優先的にメモリを割当てる
# icoFoam : ソルバ
# $PJM_JOBNAME : バッチジョブ名
# ${PJM_SUBJOBID:-$PJM_JOBID} : サブジョブID(ステップジョブ時)またはジョブID
# ソルバは，同じディレクトリ上で，異なるバッチジョブから実行したり，リスタート時に
# 複数実行する場合があるので，ログ名にバッチジョブ名とジョブIDを付加している
numactl -p1 \
icoFoam >& \
log.icoFoam.$PJM_JOBNAME.${PJM_SUBJOBID:-$PJM_JOBID}
```

ジョブの投入

```
pjsub ofp1sol.sh
```

生成ファイルの確認

ファイルの確認(pjstatでジョブ完了を確認後)

```
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
│   ├── 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                    解析結果の時刻ディレクトリ(中身は0.1と同様)
├── 0.3                    :
├── 0.4                    :
├── 0.5                    :
├── log.icoFoam.*          ソルバのログ
├── ofp1sol.sh.e1234567    ジョブの標準エラーファイル
├── ofp1sol.sh.i1234567    ジョブの統計情報ファイル
└── ofp1sol.sh.o1234567    ジョブの標準出力ファイル
```

流体解析のログ

ログの確認

```
more log.icoFoam.*
```

```
Build  : 4.1
Exec   : icoFoam
Date   : May 1 2019
Time   : 00:00:00
Host   : “c1234.ofp"
PID    : 123456
Case   : /work/0/gt00/txxxxx/lec-cavity/cavity
nProcs : 1
sigFpe : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Allowing user-supplied system call operations
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
Create time

Create mesh for time = 0

PISO: Operating solver in PISO mode
```

ビルドバージョン(実行環境依存)
実行コマンド
開始日時(実行環境依存)
開始時刻(実行環境依存)
ホスト名(実行環境依存)
プロセスID(実行環境依存)
ケースディレクトリ(実行環境依存)
使用プロセッサ数(今回非並列計算なので1)

オプションフラグ

ソルバー自体のログ

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 = 0.590864, Final residual =
2.65225e-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)

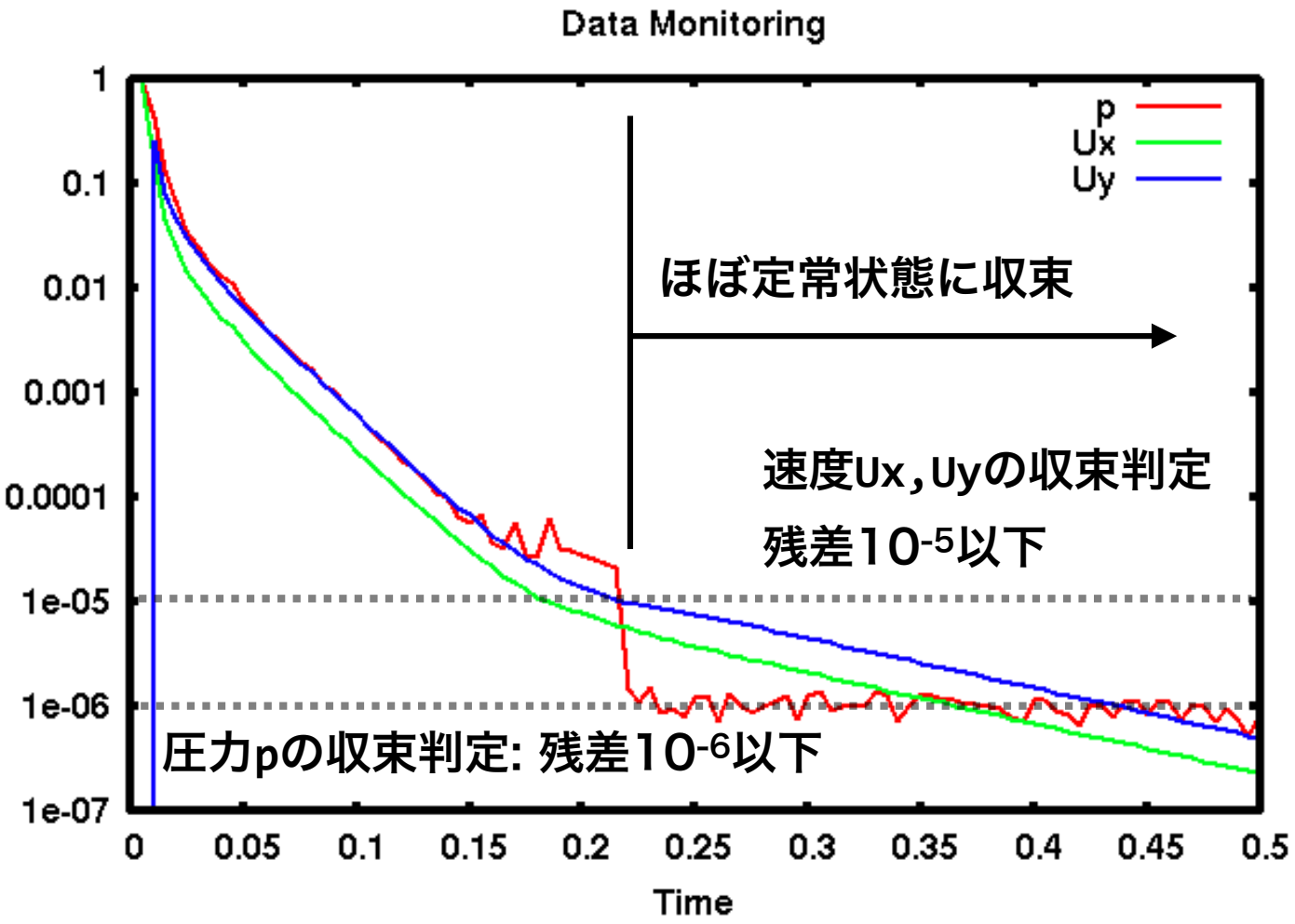
log.icoFoam.*

```
time step continuity errors : sum local = 2.74685e-09, global =
-2.6445e-19, cumulative = -4.44444e-19
#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

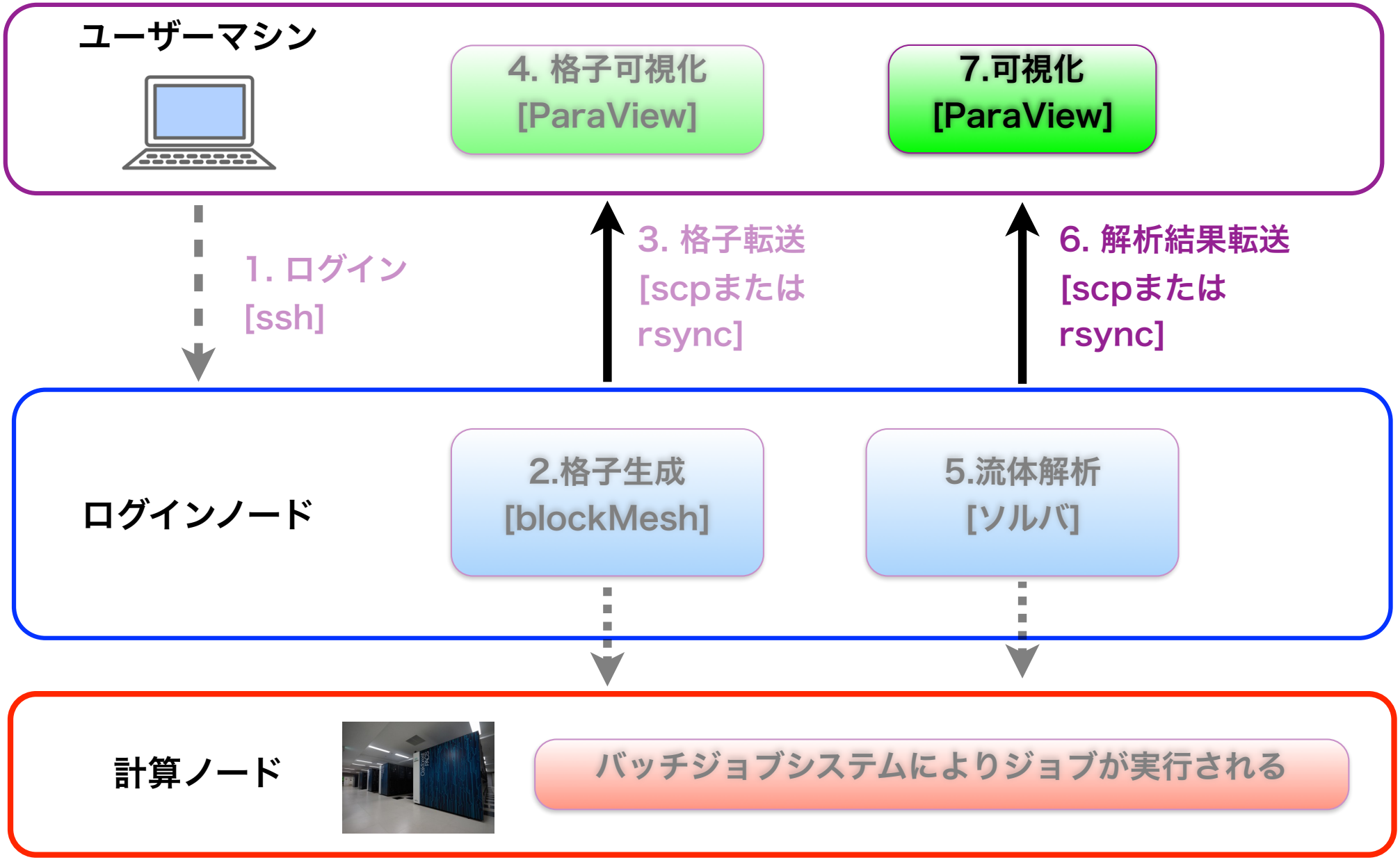
初期残差のプロット (-r 1で1秒間隔で更新. 動作しない→ OF41 を実行)

```
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



解析結果の転送

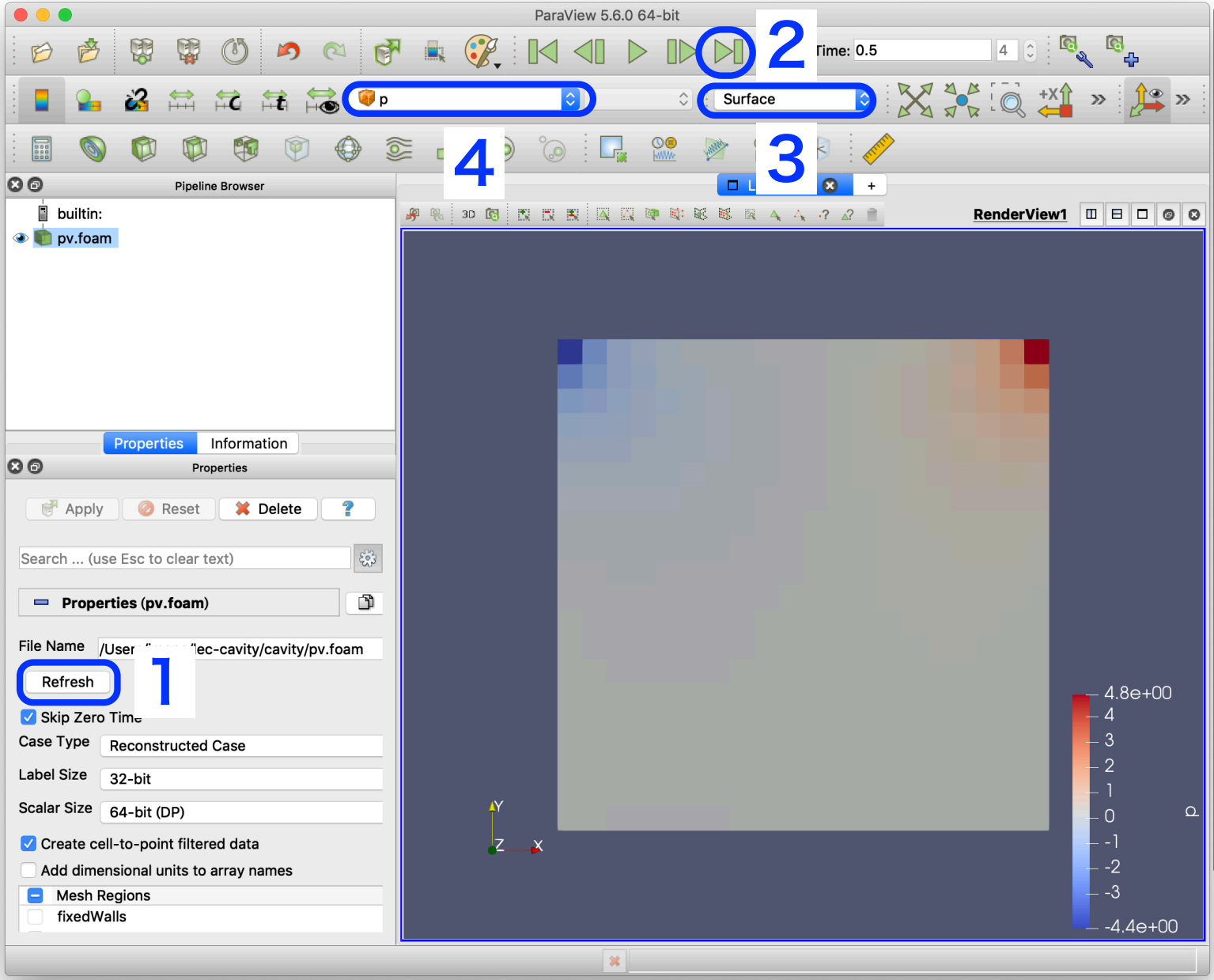
解析結果の転送(別端末で実行)

```
#↑(カーソル上)を押して前のコマンドを呼び出す
rsync -auv txxxxx@ofp.jcahpc.jp:lec-cavity/ ~/lec-cavity/
```

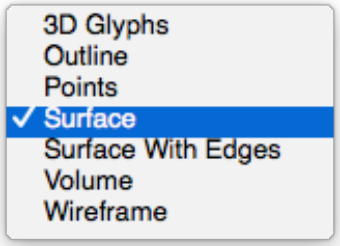
```
receiving file list ... done
cavity/
cavity//log.icoFoam.ofp1sol.sh.l1234567
cavity/ofp1sol.sh.e1234567
cavity/ofp1sol.sh.i1234567
cavity/ofp1sol.sh.o1234567
cavity/0.1/
cavity/0.1/U
:
```

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)

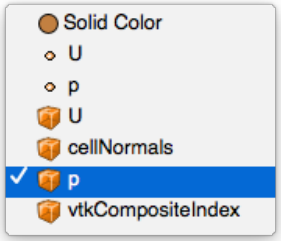
ParaViewによる圧力の可視化



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

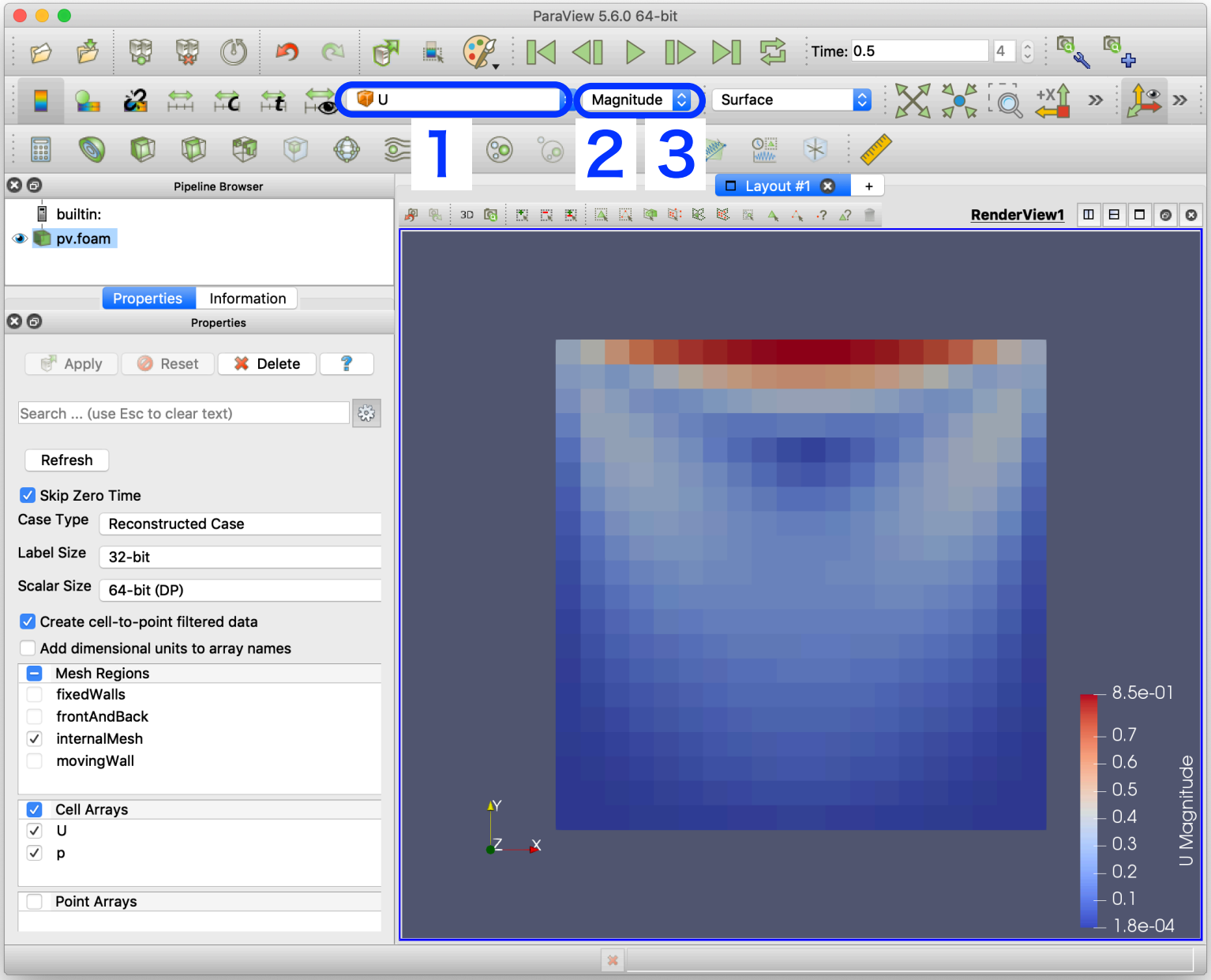


4. Coloring/□p

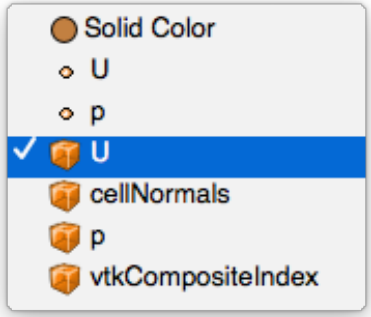


(□は格子の値そのままで補間無し、・は補間有り=スムージング)

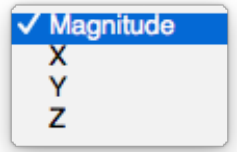
ParaViewによる風速の可視化



1. Coloring/□U

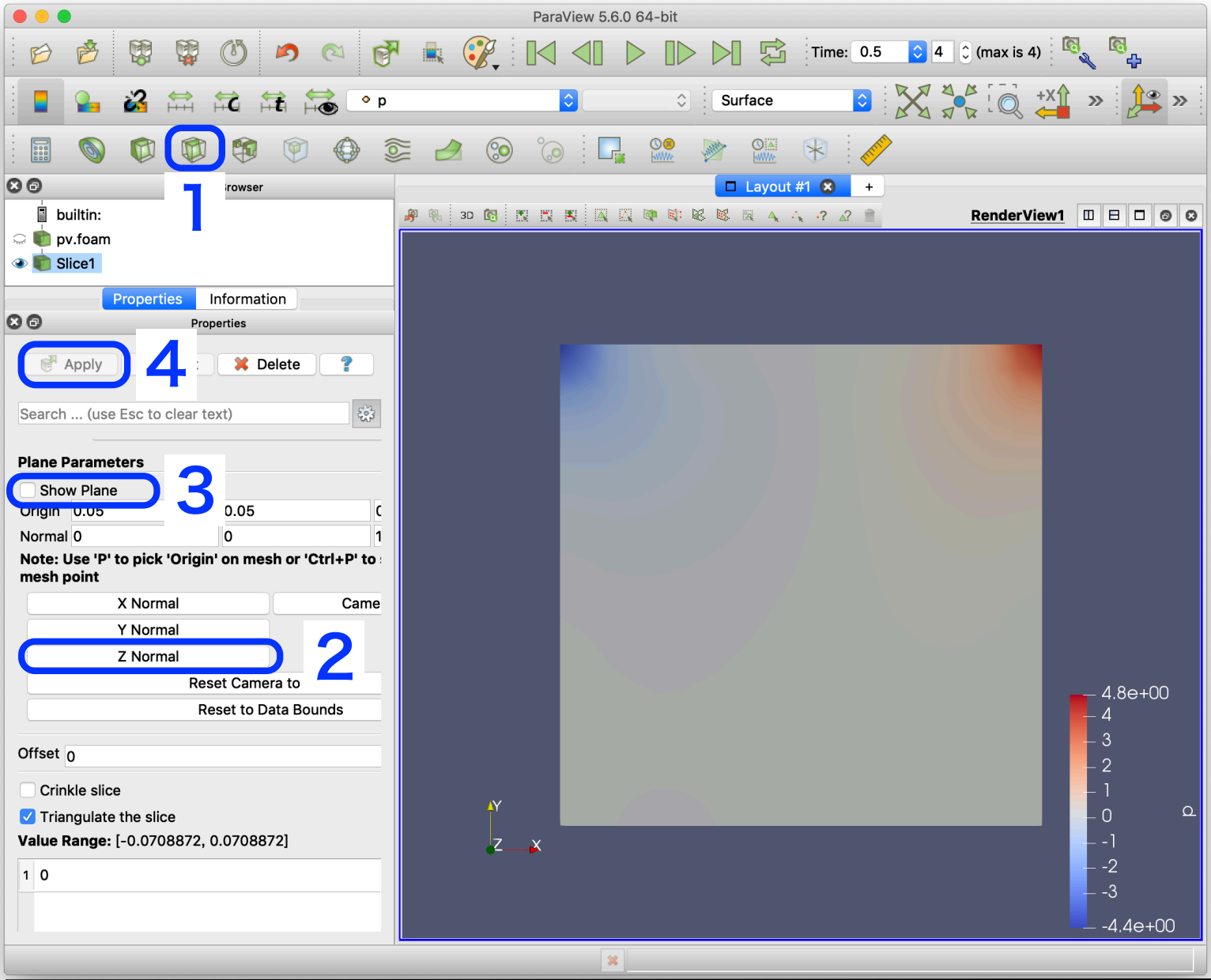


2. MagnitudeをX，Y，Zに変更することで，各風速成分が可視化できる．



3. Magnitudeに戻す．

ParaViewによる風速ベクトルの可視化

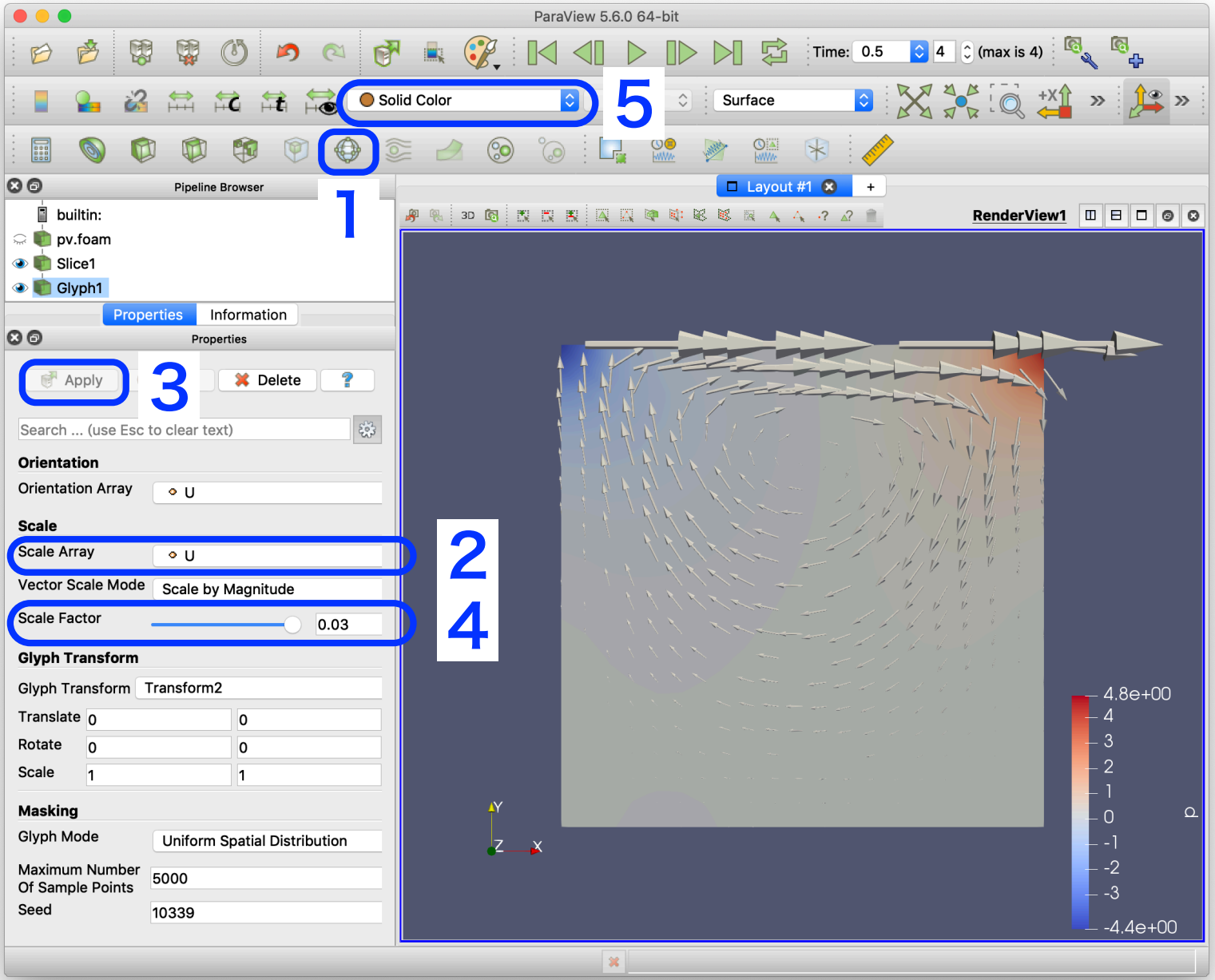


1. Sliceフィルタ
2. Z Normal
3. Show Planeを非選択
4. Apply

(注)ParaViewは通常，格子の節点にベクトルを描くので，Slice フィルタで断面を切らないと，表面と裏面で2重に表示される．

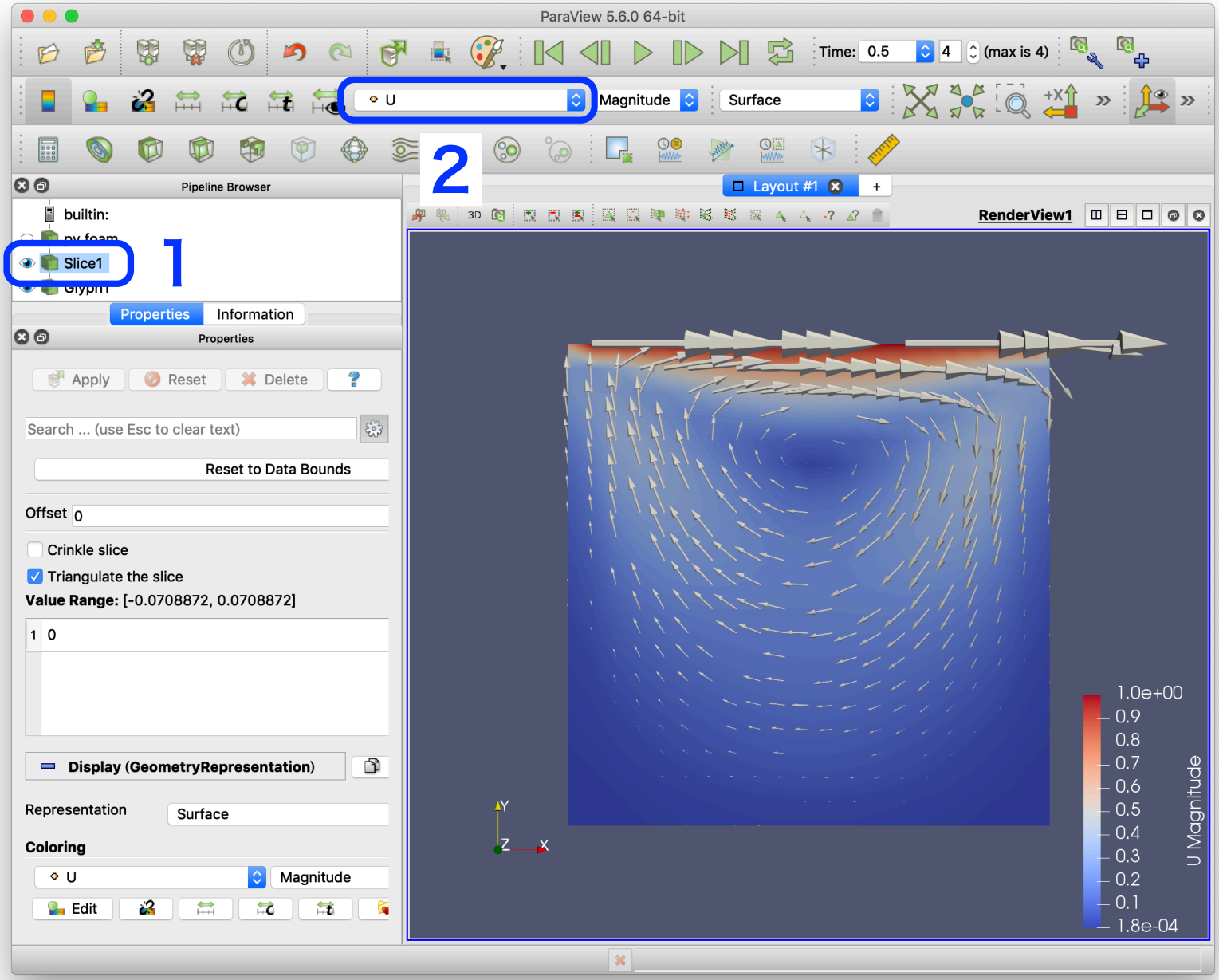
なお，CellCenterフィルタにより格子中心の値を取り出してから描いても良い．

ParaViewによる風速ベクトルの可視化(続き)



1. Glyphフィルタ
2. Scale Arrayに
   ・Uを選択
3. Apply
4. Scale Factor:
   0.03 (任意)
5. Coloring/Solid Color を選択

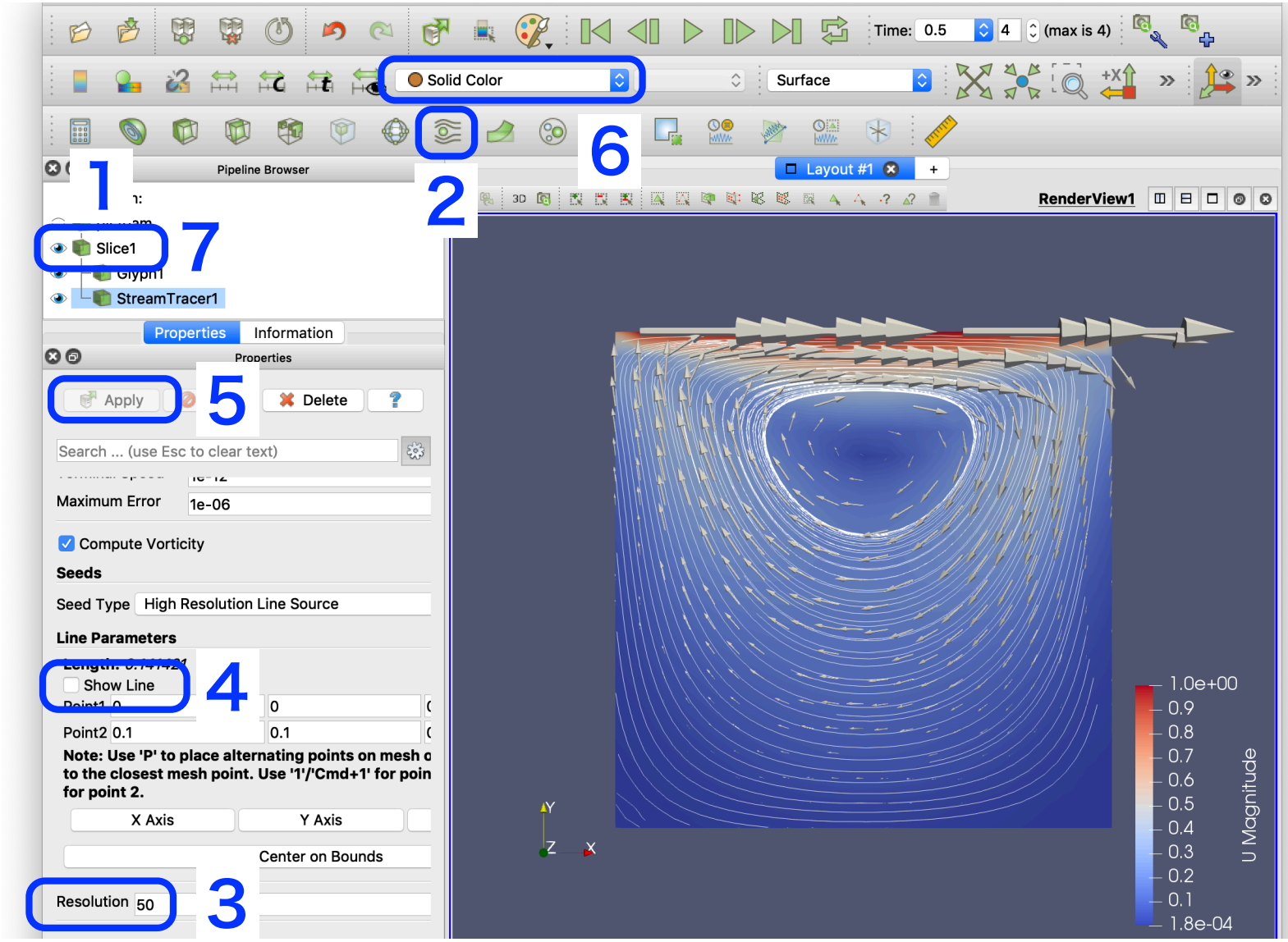
ParaViewによる風速ベクトルの可視化(続き)



1. Pipeline BrowserでSliceフィルタを選択. もし断面が非表示になっていた場合は，表示(目のマーク)をチェック(注)
2. Coloring/・Uを選択
3. 可視化が終了したら，Fileメニュー/Disconnect

(注) ParaViewのバージョンによって，挙動が異なる.

ParaViewによる流線の可視化



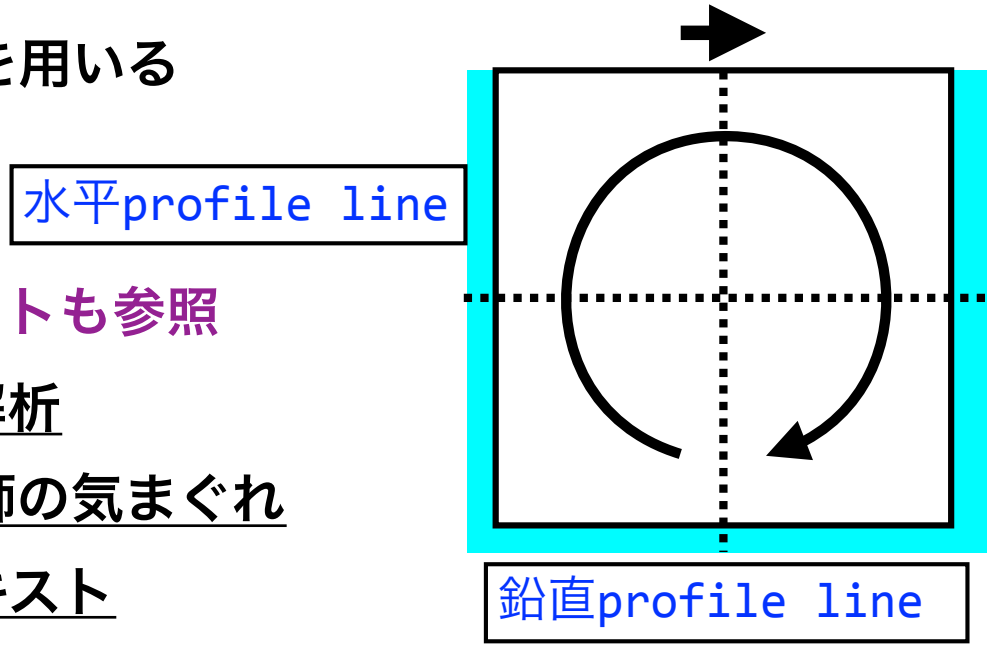
1. Pipeline BrowserでSliceフィルタ選択
2. Stream Tracerフィルタ
3. Resolutionを50
4. Show Line非選択
5. Apply
6. Coloring/Solid Color
7. Sliceフィルタ表示

注) Ghiaらの論文での流線と厳密に比較するには，以下のように流れ関数を算出後，等値面を描く．

```
streamFunction #解析結果(流束phi)から流れ関数streamFunctionを算出
```

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Oakforest-PACS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

ケースの複製(Re数=100, 辺の分割数=20, ノード数=1，並列数=2x2)

```
cd ~/lec-cavity
mkdir cavity2 #新ケースのディレクトリ作成(名前は任意だが，このまま打つ)
foamCopySettings cavity cavity2 #設定をコピー
cd cavity2
foamCleanTutorials #実行結果を消去
```

Ghiaらの解析に合わせた設定変更

格子生成設定ファイルの編集

```
vi system/blockMeshDict
```

```
convertToMeters 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         15;      //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   1000;      //書き出す間隔(1000time step=5s毎)
```

終了時刻を15s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，概ね定常になるまでに多くの時間積分が必要であり，完全には定常にならない．

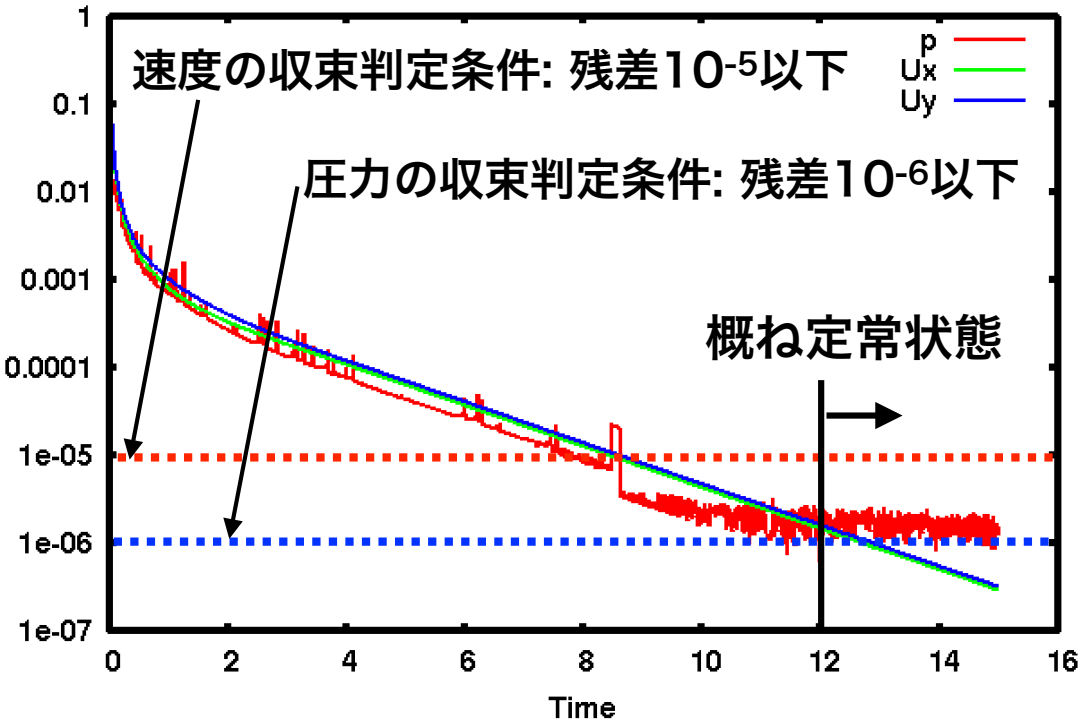
ジョブ投入・残差モニター・強制終了

ジョブ投入

```
pjsub ofp0mesh.sh
pjstat                    #ジョブの終了を確認してから次に進む
pjsub ofp1sol.sh
```

ofp1sol.shがジョブ実行中になったら残差モニターとログ確認

```
foamMonitor -r 1 -l postProcessing/residuals/0/residuals.dat &
tail -f log.icoFoam.*
```



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

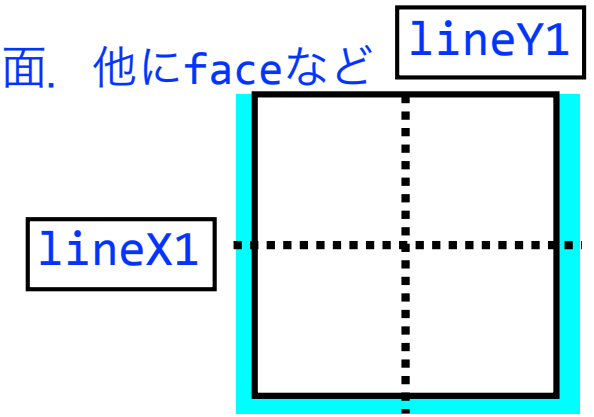
```
pjdel JOB_ID
```

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
```

```
libs ("libsampling.so"); //本関数のライブラリ(動的にリンクされる)
type sets; //集合サンプリング．他にもsurfaces(面サンプリング)等がある
setFormat raw; //出力フォーマット，raw:テキスト形式．他にvtk,csv,gnuplot等がある
interpolationScheme cellPointFace; //補間方法(格子中心，節点，界面の値から補間)
//cell:格子中心のみ(格子内一定)，cellPoint:格子中心，節点から補間
fields ( U ); //サンプリングする場のリスト
sets //集合サンプリングの定義
(
  lineX1 //サンプリング名
  {
    type  midPointAndFace; //サンプリング型，格子中心と界面．他にfaceなど
    axis  x; //出力する座標軸，x/y/z/xyz(全座標)
    start (-0.001 0.5 0.05); //サンプリング開始点
    end   ( 1.001 0.5 0.05); //サンプリング終了点
  }
  lineY1 //サンプリング名(以下，lineX1と同様なので略)
```



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
```

本来は，このようなプリポスト処理は，プリポストノードで実行するのが望ましいが，演習ではプリポストノードが使用できず，高負荷ではないのでログインノードで実行する

サンプリング結果確認

```
more postProcessing/sample/15/lineX1_U.xy
```

postProcessing/sample/15/lineX1_U.xy

```
x    Ux    Uy    Uz
```

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

```
0    0    0    0
0.025    -0.00210791    0.0432661 0
:
0.975    -0.00499169    -0.0506717 0
1    0    0    0
```

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)

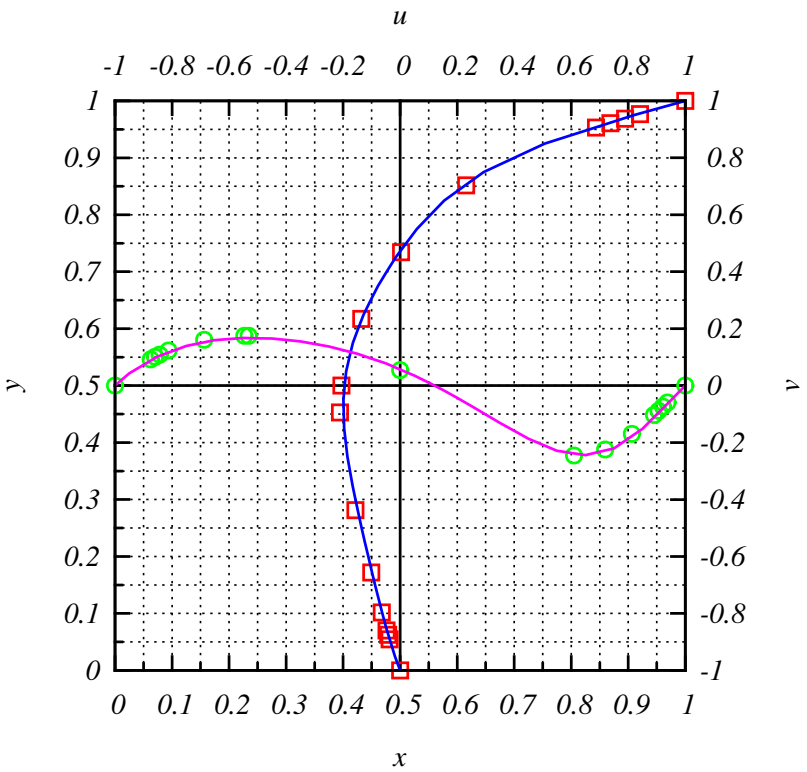
gnuplot実行

```
gnuplot profiles.gp
```

プロットファイル表示(表示できない場合には，PC側に同期して表示する)

```
evince profiles.pdf &
```

プロット結果と自習演習



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

自習課題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.

自習課題2: 自習課題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15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

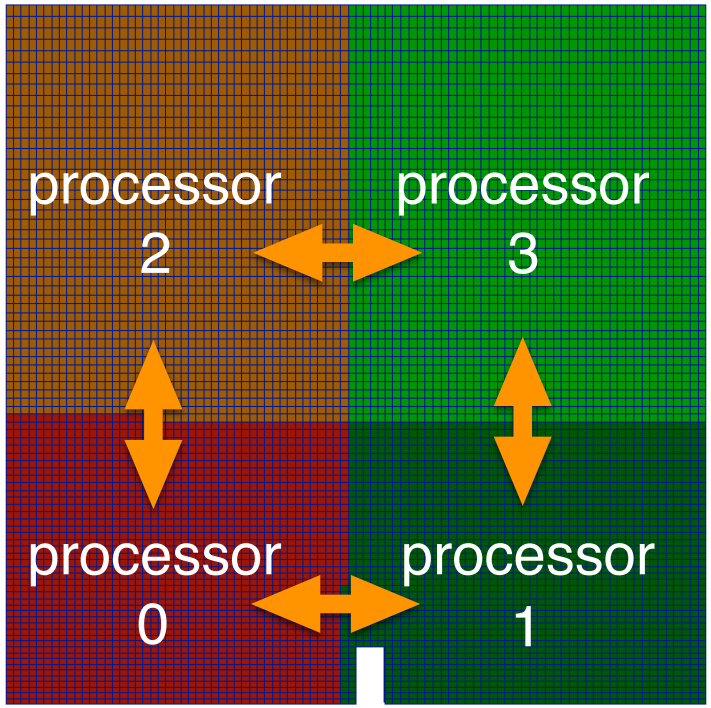
1. 格子生成
2. 領域分割 (decomposePar)
3. MPI並列でソルバを実行

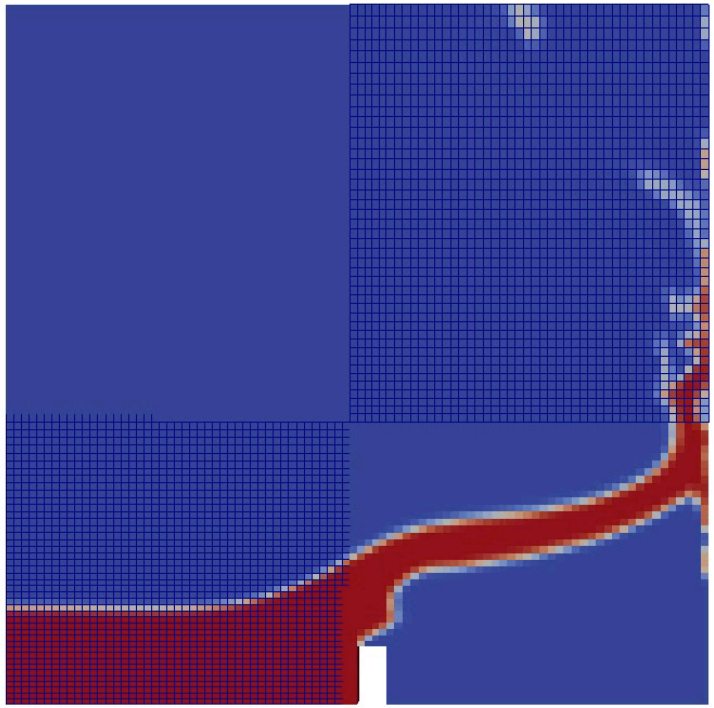
(MPI+OpenMPのハイブリット並列は標準では未実装．研究例有り)

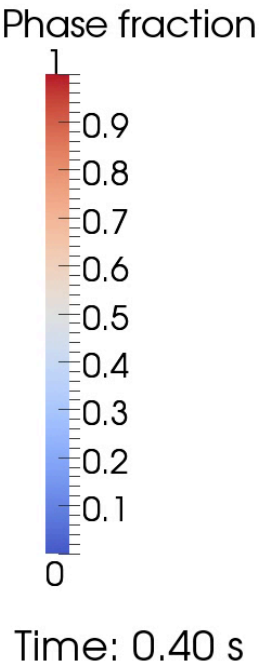
4. 領域毎の解析結果を再構築 (reconstructPar)

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

領域分割用ジョブスクリプト

```
more ofp2de.sh
```

ofp2de.sh

```
#!/bin/bash
#PJM -L rscgrp=lecture-flat
#PJM -L node=1
#PJM -L elapse=0:15:00
#PJM -g gt00
#PJM -S
module purge                          # 標準で有効なmoduleをpurgeで全てunload
module load gcc/4.8.5                 # Gcc-4.8.5のmoduleをload
module load openfoam/4.1              # OpenFOAM-4.1のmoduleをload
source $WM_PROJECT_DIR/etc/bashrc # OpenFOAMの環境設定
decomposePar -cellDist >& log.decomposePar # 領域分割
# -cellDist : 格子の領域番号を場cellDistに出力(可視化用．必須ではない)
```

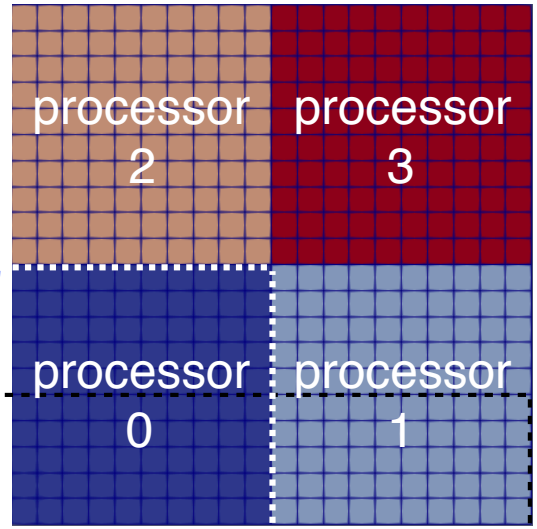
領域分割ジョブの投入

```
pjsub ofp2de.sh
```

領域分割ジョブ投入とログの確認

領域分割のログの確認(ジョブ完了後)

```
more log.decomposePar
```



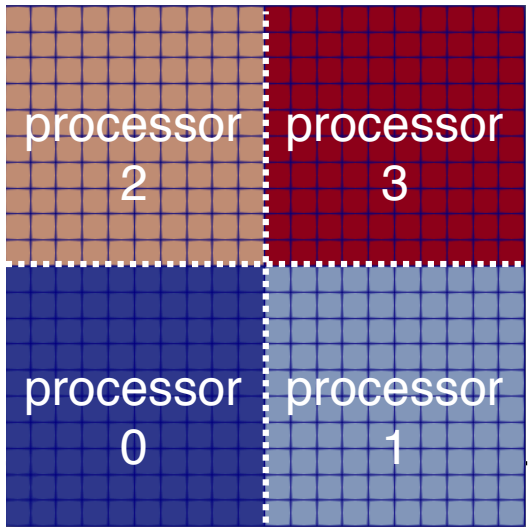
log.decomposePar

```
Processor 0 #プロセッサ0の担当分割領域
    Number of cells = 100 #格子数
    Number of faces shared with processor 1 = 10
    #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 = 10
    #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 = 20 #上記の共有界面数の合計(=10+10)
    Number of boundary faces = 220 #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 = 100 #格子数
```

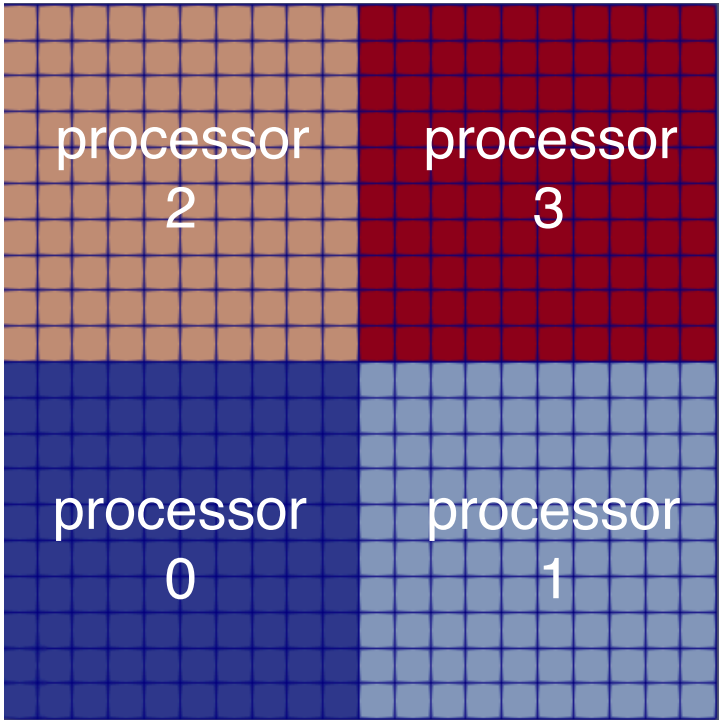
領域分割のログの確認(続き)

log.decomposePar

```
Number of processor faces = 40 #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 = 100 (0% above average 100)
Max number of processor patches = 2 (0% above average 2)
Max number of faces between processors = 20 (0% above average 20)

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



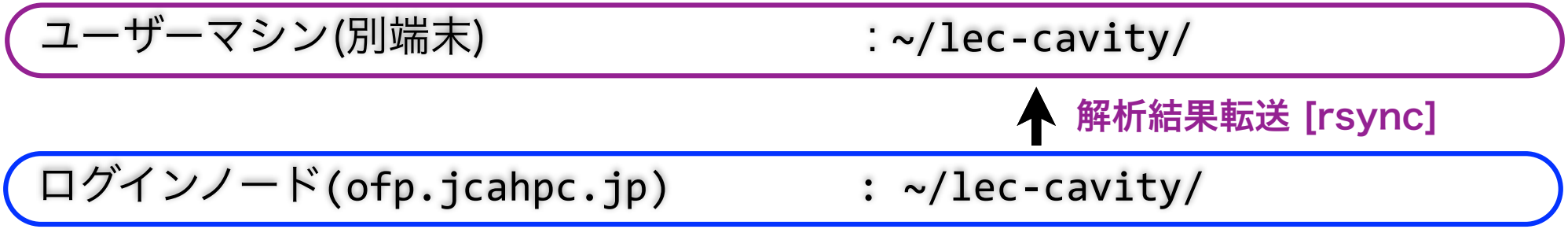

```
constant/
  polyMesh/     #格子データ
0/              #時刻ディレクトリ
  U, p          #場データ
```

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

解析結果の転送



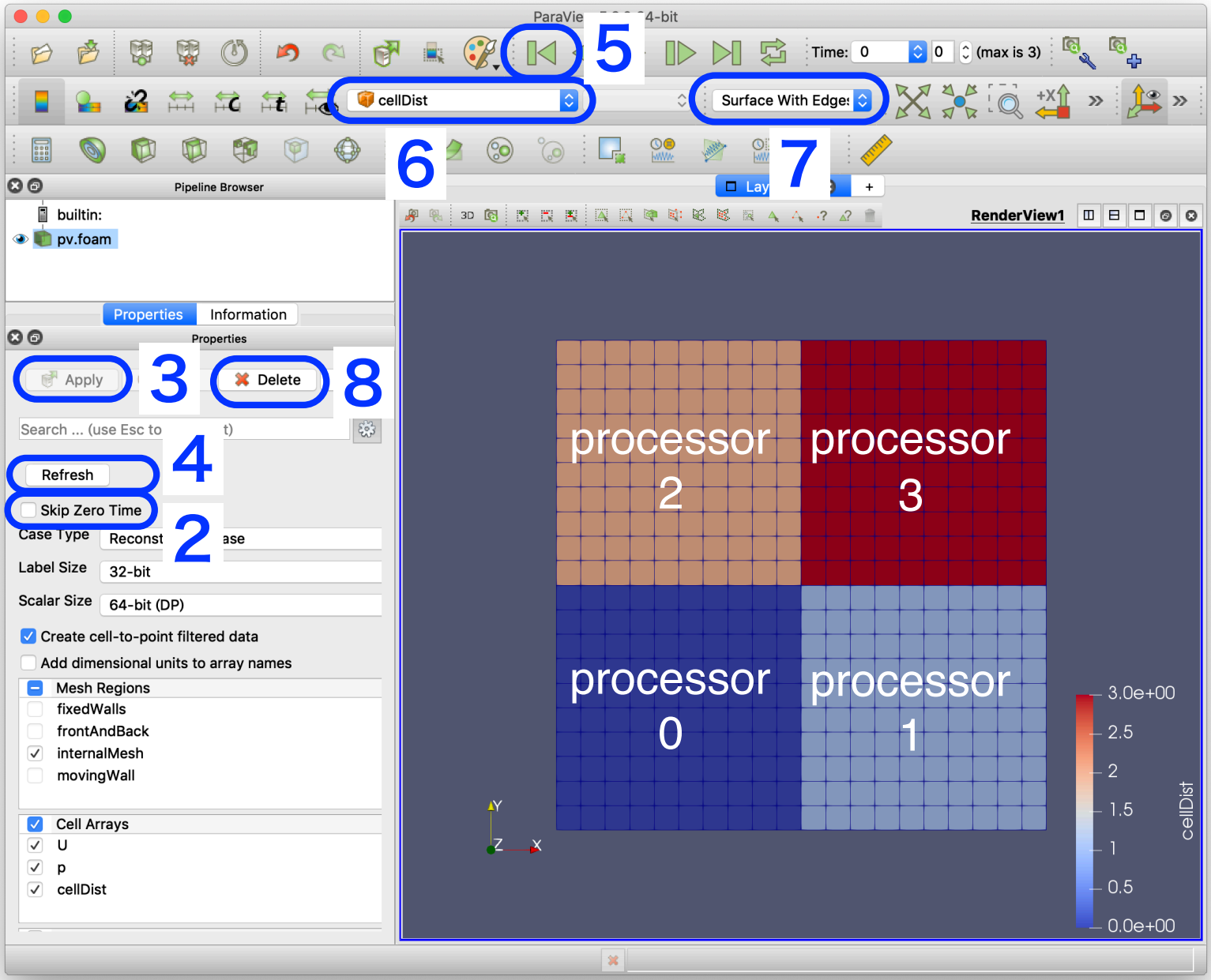
解析結果の転送(別端末で実行)

```
# ↑(カーソル上)を押して前のコマンドを呼び出す
rsync -auv txxxxx@ofp.jcahpc.jp:lec-cavity/ ~/lec-cavity/
```

新ケースディレクトリへの移動および可視化用ファイル作成 (別端末で実行)

```
cd ~/lec-cavity/cavity2
touch pv.foam
```

ParaViewによる分割領域の可視化



1. Fileメニュー/Openでcaivty2のpv.foamを読む
2. Skip Zero Timeをアンチェック(注)
3. Apply
4. Refresh
5. First Frame
6. Coloring/□cellDist 選択
7. Representation/Surface With Edges 選択
8. pv.foamをDelete

(注) 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

並列計算実行用ジョブスクリプト

ofp3sol-par.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
#PJM -L rscgrp=lecture-flat
#PJM -L node=1
#PJM --mpi proc=4
(略)
source /usr/local/bin/mpi_core_setting.sh # MPIプロセスのピニング
export I_MPI_DEBUG=5                      # Intel MPIのDEBUG情報レベル
# 並列ソルバ実行
# mpiexec.hydra : Intel MPIのMPI並列実行コマンド
# -n $PJM_MPI_PROC : プロセス数指定(PJMによりPJM_MPI_PROCに設定される)
# numactl -p1 : MC-DRAMに優先的にメモリを割当てる
# icoFoam : ソルバ
# -parallel : 並列実行の指定(OpenFOAMのコマンド共通のオプション)
mpiexec.hydra -n $PJM_MPI_PROC \
numactl -p1 \
icoFoam -parallel >& \
log.icoFoam.$PJM_JOBNAME.${PJM_SUBJOBID:-$PJM_JOBID}
```

MPIプロセス数

並列計算とログ確認

```
pjsub ofp3sol-par.sh
more log.icoFoam.ofp3sol-par.*      #ジョブ開始後
```

```
[0] MPI startup(): Multi-threaded optimized library
[0] MPI startup(): shm data transfer mode
[1] MPI startup(): shm data transfer mode
[2] MPI startup(): shm data transfer mode
[3] MPI startup(): shm data transfer mode
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       143922   c3847.ofp  {2,70,138,206}
[0] MPI startup(): 1       143923   c3847.ofp  {4,72,140,208}
[0] MPI startup(): 2       143924   c3847.ofp  {6,74,142,210}
[0] MPI startup(): 3       143925   c3847.ofp  {8,76,144,212}
[0] MPI startup(): I_MPI_DEBUG=5
[0] MPI startup(): I_MPI_FABRICS_LIST=tmi
[0] MPI startup(): I_MPI_FALLBACK=0
:
[0] MPI startup(): I_MPI_JOB_FAST_STARTUP=1
[0] MPI startup(): I_MPI_PIN_MAPPING=4:0 2,1 4,2 6,3 8

nProcs : 4
Slaves  :
3
(
"c3847.ofp.143923"
```

使用されるファブリック．Omni-Pathのデフォルトでは，ノード内はshm(共有メモリ)，ノード間はtmi(Tag Matching Interface)

MPIプロセスのピニング情報

使用するファブリックリストの指定(*)

指定したファブリックを必ず使用する(*)

高速プロセス起動アルゴリズムon(*)

ピニング設定(*)

計算で使用されているプロセス数

Slaveプロセスの"計算ノードのホスト名.PID"

*) mpi_core_setting.shで設定されている環境変数

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)

MPIプロセスのピニング

ピニング・スクリプト(/usr/local/bin/mpi_core_setting.sh)によるMPIプロセスのピニング

物理コア0は，CPUスケージューリングクロック割り込み抑止設定(動的Tickless)がされていないので，なるべく使用しない．さらに，同じタイルにある物理コア1も，なるべく使用しない．

図出典：[OFP]

一般的にL1専有よりL2専有のほうがピーク性能が高い(速い)．
ただし，34プロセス以上のL1専有のほうが，費用(トークン)対性能が高い事が多い．

1ノード内プロセス数が1〜33
L2キャシュを1プロセスが専有

1ノード内プロセス数が34〜66
L1キャシュを1プロセスが専有

ベンチマーク例: 今野: OpenFOAMによる流体解析ベンチマークテスト FOCUS・クラウド・スパコンでのチャネルおよびボックスファン流れ解析, 2017

並列計算結果の再構築用ジョブスクリプト

```
more ofp4re.sh
```

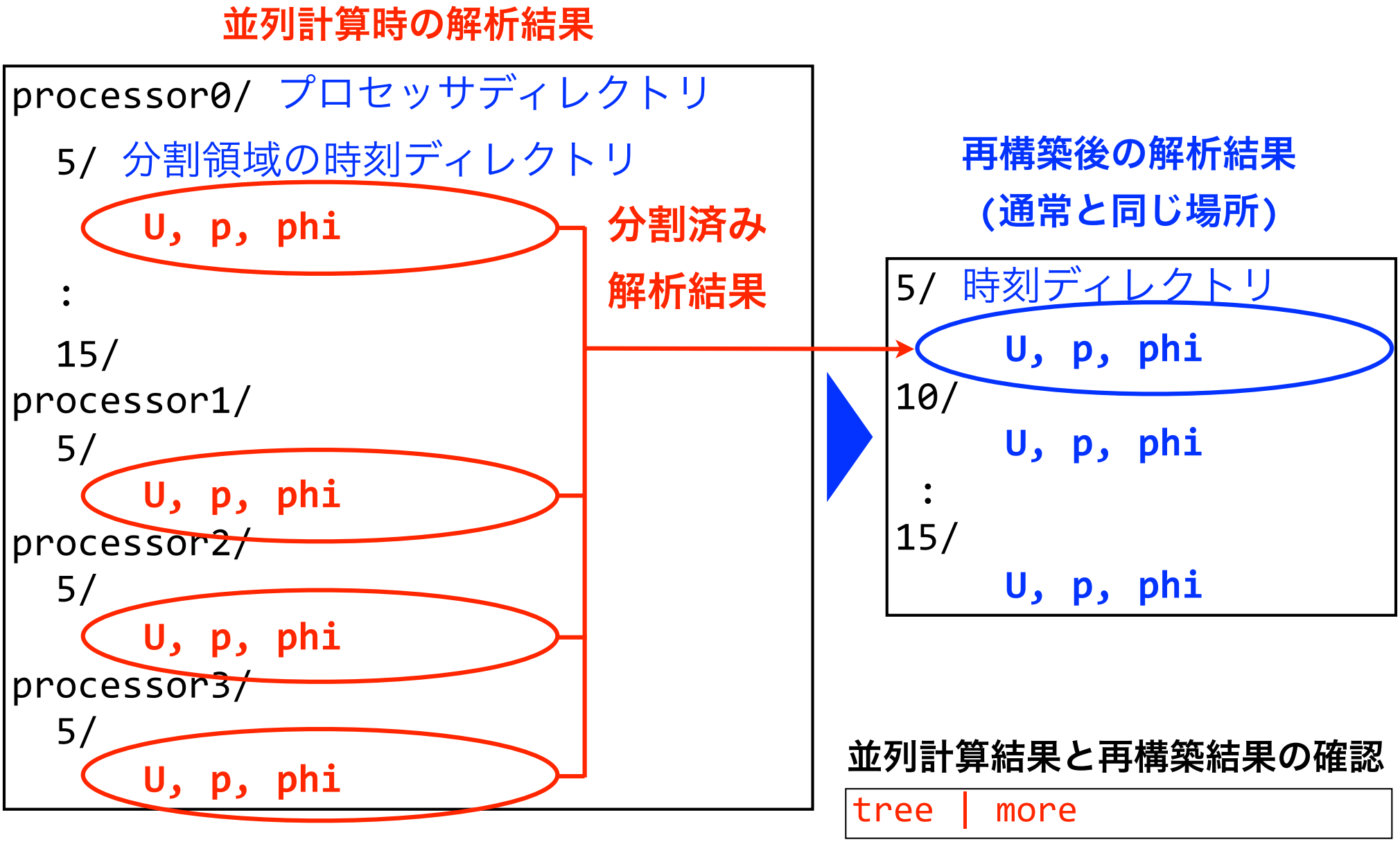
ofp4re.sh

```
#!/bin/bash
#PJM -L rscgrp=lecture-flat
#PJM -L node=1
#PJM -L elapse=0:15:00
#PJM -g gt00
#PJM -S
module purge                          # 標準で有効なmoduleをpurgeで全てunload
module load gcc/4.8.5                 # Gcc-4.8.5のmoduleをload
module load openfoam/4.1              # OpenFOAM-4.1のmoduleをload
source $WM_PROJECT_DIR/etc/bashrc # OpenFOAMの環境設定
reconstructPar >& log.reconstructPar # 領域再構築
```

領域分割ジョブの投入

```
pjsub ofp4re.sh
```

並列計算結果の再構築



並列計算結果と再構築結果の確認

```
tree | more
```

演習課題

課題1 Re=100，20分割格子，1ノードP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：n並列時の最初と最後の時間ステップのExecutionTimeの差をt(n)として，以下で求める(初期ステップ完了にかかる時間は除外して並列化効率を算出)

- スピードアップ率：$S_P = T_S / T_P =$ t(1)/t(P)
- 並列化効率 ：$E_P = S_P / P$ = (t(1)/t(P))/P

ソルバーのログからt(n)を算出するスクリプトを使用して算出する場合(赤字がt(n))

```
../bin/averageExecutionTime.sh
```

```
#Filename,TimeSteps[-],InitTime[s],LastTime[s],Time[s],AveTime[s]
log.icoFoam.ofp1sol.sh.2508789,3000,0.13,17.27,17.14,0.00571524
log.icoFoam.ofp3sol-n4.sh.2508866,3000,0.13,24.47,24.34,0.00811604
```

pythonなどでスピードアップ率等を計算

```
python -c "print(17.14/24.34)"
```

```
0.704190713698 #逆に遅くなる
```

今回は格子数が20×20=400と非常に少なく，プロセスあたりの格子数10×10=100に対して，MPI通信が必要な共有界面数が10×2=20と相対的に非常に多いので，並列計算の効率が非常に悪い.

演習課題(続き)

課題2 128または256分割格子で1ノードおよび複数ノード(最大16)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．また，MPIプロセスのピニングを実行しないケースも調べてみる．

ケースの複製例(Re=100, 辺の分割数128, ノード数1，並列数=8x8)

```
cd ~/lec-cavity
mkdir cavity3 #新ケースのディレクトリ作成(名前は任意)
foamCopySettings cavity2 cavity3 #設定をコピー
cd cavity3
foamCleanTutorials #実行結果を消去
```

領域分割設定ファイルの編集例

```
vi system/decomposeParDict
```

```
numberOfSubdomains 64;       //領域分割数
method              simple;
// methodを scotch に変更しても良い．この場合，simpleCoeffs 内の変更は不要
simpleCoeffs //単純に軸方向に分割
{
    n       ( 8 8 1 ); //分割数
```

演習課題(続き)

格子生成設定ファイルの編集例

```
vi system/blockMeshDict
```

```
blocks
(
    hex (0 1 2 3 4 5 6 7) (128 128 1) simpleGrading (1 1 1)
);
```

実行制御の設定ファイルの編集例

```
vi system/controlDict
```

```
endTime         0.505;      //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   1000;       //書き出す間隔(1000time step=5s毎，出力しない)
```

定常までの解析では時間がかかるので，ベンチマークテストでは1回目の時間ステップから101回目の時間ステップ(0.505[s])までの100時間ステップでの実行時間を比較する．また，結果ファイルの書き出しも行わない．

演習課題(続き)

MPI並列ジョブファイルの編集例

```
vi ofp3sol-par.sh
```

```
#PJM -L node=1
#PJM --mpi proc=64
```

ノード数(最大16)とMPIプロセス数を適宜変更する

手動での解析ジョブ実行 (pjstatでジョブの完了確認後，次の行に進む→面倒)

```
pjsub ofp0mesh.sh
pjsub ofp1sol.sh
pjsub ofp2de.sh
pjsub ofp3sol-n64.sh
```

今回は実行しない

長いジョブを続けて実行するには以下のように，ステップジョブ実行するほうが便利

```
pjsub --step ofp0mesh.sh #出力: [INFO] PJM 0000 pjsub Job 1234567_0 submitted.
pjsub --step --sparam jid=1234567 ofp1sol.sh #jid=ジョブID．_0(サブジョブID)は不要
pjsub --step --sparam jid=1234567 ofp2de.sh
pjsub --step --sparam jid=1234567 ofp3sol-n64.sh
```

今回は実行しない

上記を自動的に実行するスクリプトAllrun.batchで実行

```
./Allrun.batch ofp[0-3]*.sh     #引数に実行したいジョブ名を順番に並べる
pjstat -E                       #-Eオプションステップジョブ内のサブジョブを展開
../bin/averageExecutionTime.sh #完了後．実行時間を算出し，スピードアップ率を求める
```

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(ホットスポット)の計算時間の割合など，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

```
more ofp5sol-vtune.sh
```

ofp5sol-vtune.sh（逐次実行ジョブ用スクリプト，赤字がVTune用追加分）

```
module load vtune                    # VTuneのmoduleをload
# ソルバ実行のプロファイリング
# amplxe-cl : CUI版のVTuneコマンド
# -c advanced-hotspots : ホットスポットの情報を収集
# -r 保存ディレクトリ(既に存在しているとエラー)
amplxe-cl -c advanced-hotspots \
-r vtune.$PJM_JOBNAME.${PJM_SUBJOBID:-$PJM_JOBID} \
numactl -p1 \
icoFoam >& \
log.icoFoam.$PJM_JOBNAME.${PJM_SUBJOBID:-$PJM_JOBID}
```

VTuneのデータ保存ディレクトリ名は任意だが，既に存在しているとVTuneの起動時にエラーとなるので，ユニークになるように，バッチジョブ名とジョブIDを付加している

プロファイラVTuneのMPI並列実行

```
more ofp6sol-par-vtune.sh
```

ofp6sol-par-vtune.sh（並列実行ジョブ用スクリプト，赤字がVTune追加分）

```
:
#PJM -L node=1
#PJM --mpi proc=64
:
module load vtune                       # VTuneのmoduleをload
source /usr/local/bin/mpi_core_setting.sh # MPIプロセスのピニング
export I_MPI_DEBUG=5                    # Intel MPIのDEBUG情報レベル
# 並列ソルバ実行のプロファイリング
# amplxe-cl : CUI版のVTuneコマンド
# -c advanced-hotspots : ホットスポットの情報を収集
# -r 保存ディレクトリ:収集MPIランク範囲
mpiexec.hydra -n $PJM_MPI_PROC \
-gtool "amplxe-cl -c advanced-hotspots \
-r vtune.$PJM_JOBNAME.${PJM_SUBJOBID:-$PJM_JOBID}:0" \
numactl -p1 \
icoFoam -parallel >& \
log.icoFoam.$PJM_JOBNAME.${PJM_SUBJOBID:-$PJM_JOBID}
```

プロファイラVTune付き実行

プロファイラ付きでジョブ実行

```
pjsub ofp5sol-vtune.sh
tail -f log.icoFoam.ofp5sol-vtune.sh.* #ジョブ実行後にソルバのログを追跡
```

```
Elapsed Time:        52.835
Paused Time:         0.0
CPU Time:            52.120
Average CPU Usage:   0.968
CPI Rate:            1.101
```

ソルバのログの末尾にVTuneのログが出力される

平均CPU使用率

Cycles Per Instructions(命令あたりのサイクル):小さいほど良い

コマンドラインでテキスト形式に変換（GUIの場合 amplxe-gui を実行）

```
module load vtune
export INTEL_LICENSE_FILE=/home/opt/local/cores/intel/licenses
amplxe-cl -R hotspots -r vtune.ofp5sol-vtune.sh.* > vtune.txt
more vtune.txt
```

```
Function                                          CPU Time
Foam::DICPreconditioner::precondition             18.25s
Foam::lduMatrix::Amul                             10.710s
```

線形ソルバPCGのDIC前処理

行列ベクトル積

並列実行の結果(ofp6sol-par-vtune.shについて上記を実行)

```
I_MPI_memcpy                                      0.760s
native_queued_spin_lock_slowpath                  0.380s
Foam::DICPreconditioner::precondition             0.350s
```

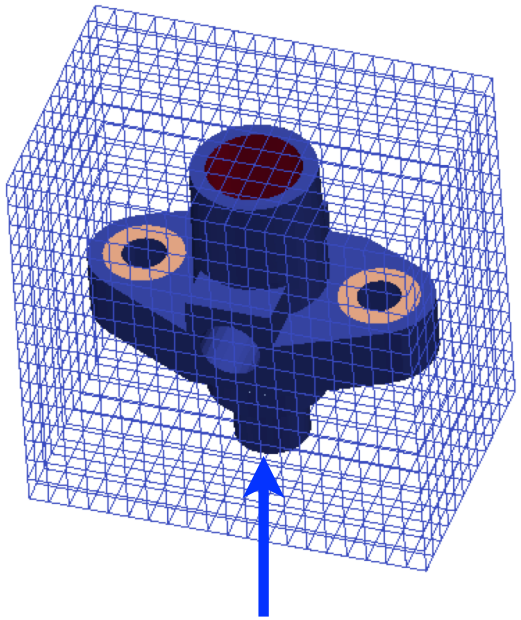
MPI関連(並列数が増えると支配的となる)

線形ソルバPCGのDIC前処理

snappyHexMeshによる格子生成基礎

blockMesh
構造格子メッシャー

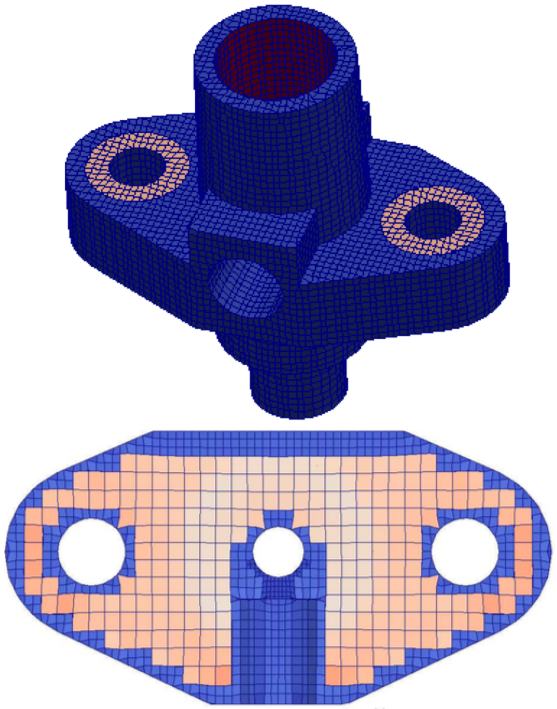
解析領域を含む構造格子のベース格子を生成



解析領域

snappyHexMesh
構造格子メッシャー

ベース格子を細分割して，形状に適合したメッシュを自動的に生成



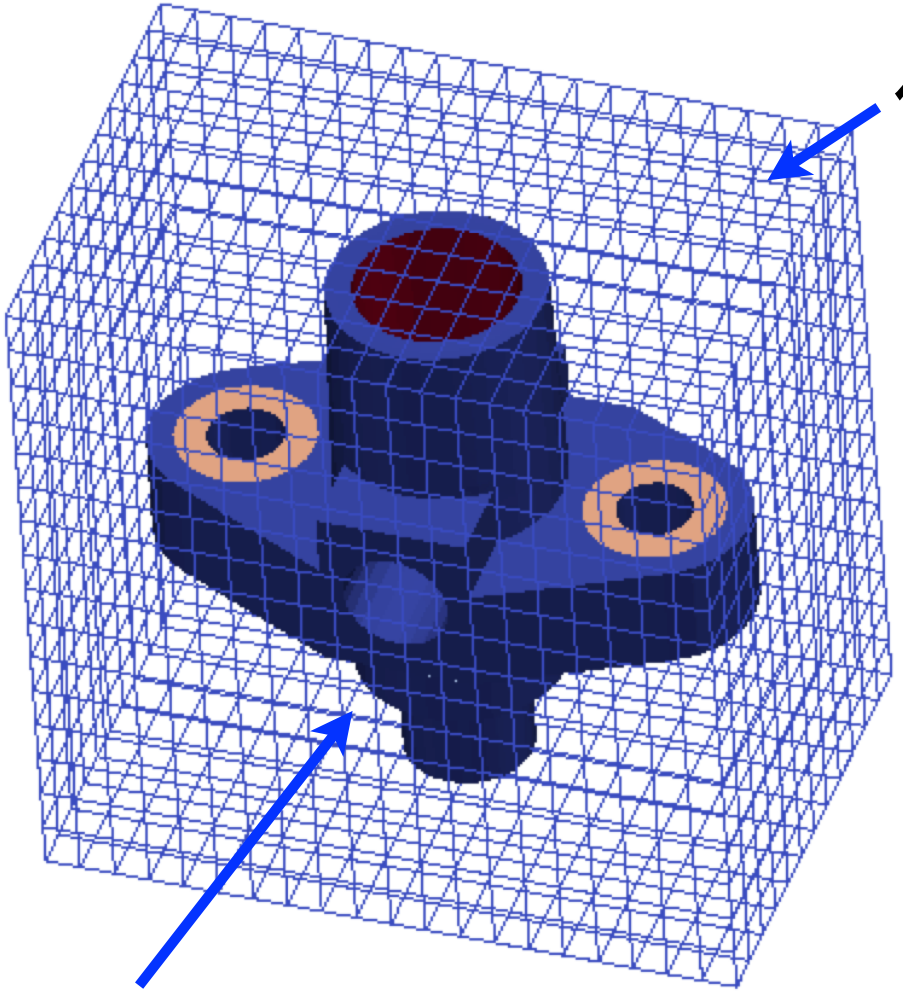
○ ほぼ六面体の格子が自動的に生成可能

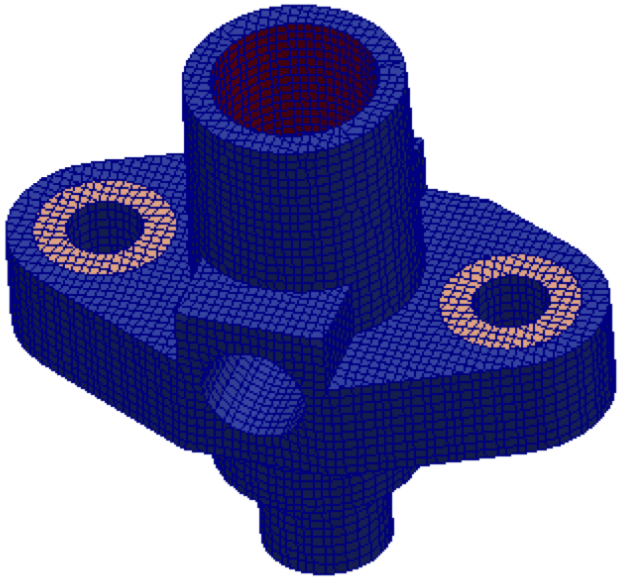
× 任意間隔のベース格子が作成が難しい

× 生成格子がベース格子の形状に大きく依存

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





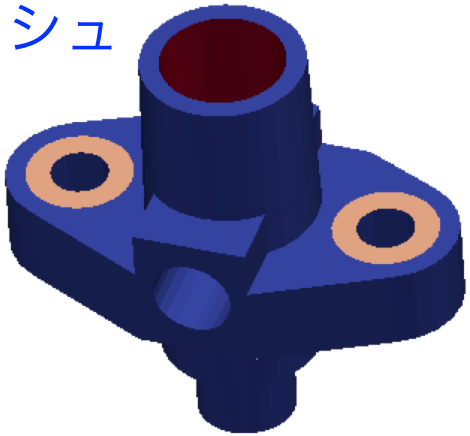
生成された格子(パッチ別に色分け)

CADデータ (STL形式、solidブロック別に色分け)

snappyHexMeshDict概要

system/snappyHexMeshDict

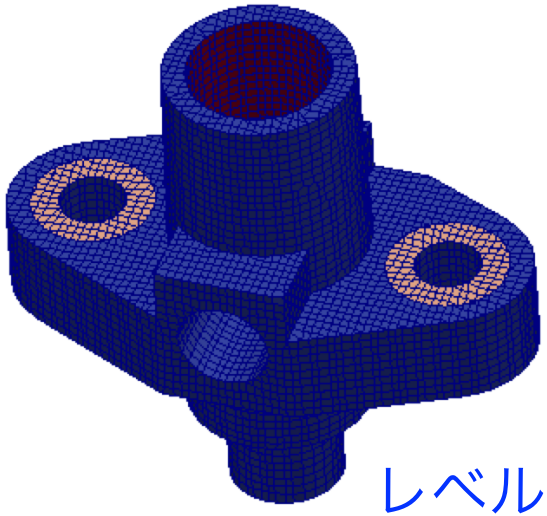
```
castellatedMesh true; //階段状格子
snap            true; //境界適合する
addLayers       false; //レイヤ付加しない

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)
};
```



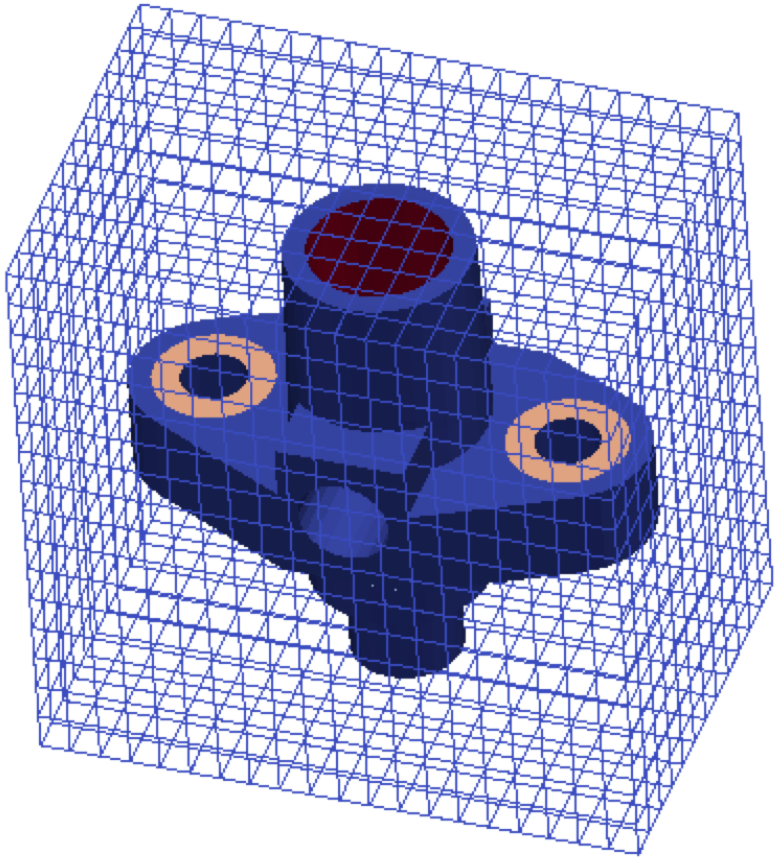
snappyHexMeshDict概要(続き)

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)

チュートリアル実行用ジョブスクリプト

foamRunTutorials.sh

```
#!/bin/bash
#PJM -L rscgrp=lecture-flat
#PJM -L node=1
#PJM --mpi proc=33
#PJM -L elapse=0:15:00
#PJM -g gt00
#PJM -S
module purge                          # 標準で有効なmoduleをpurgeで全てunload
module load gcc/4.8.5                 # Gcc-4.8.5のmoduleをload
module load openfoam/4.1              # OpenFOAM-4.1のmoduleをload
source $WM_PROJECT_DIR/etc/bashrc # OpenFOAMの環境設定
source /usr/local/bin/mpi_core_setting.sh # MPIプロセスのピニング
export I_MPI_DEBUG=5                  # Intel MPIのDEBUG情報レベル
# tutorialケースの実行
foamRunTutorials >& log.$PJM_JOBNAME.${PJM_SUBJOBID:-$PJM_JOBID}
```

チュートリアルケースの並列数は最大で20程度なので，L2キャッシュを専有できるノードあたりの最大プロセス数33を設定.
チュートリアルケースによって，並列数は異なり，非並列のケースも多いが，上記で確保したプロセス数以下の並列数ならば動作する

foamRunTutorialsコマンドでチュートリアルを実行.

foamRunTutorialsは，チュートリアルのディレクトリにAllrunスクリプトがあれば，それを実行し，なければ，blockMeshとsystem/controlDictで定義されたapplicationを実行する

flangeチュートリアルの実行と格子可視化

```
cd ~/lec-cavity
cp -r $FOAM_TUTORIALS/mesh/snappyHexMesh/flange .
cd flange
cp ../foamRunTutorials.sh ./
pjsub foamRunTutorials.sh
tail -f log.*
```

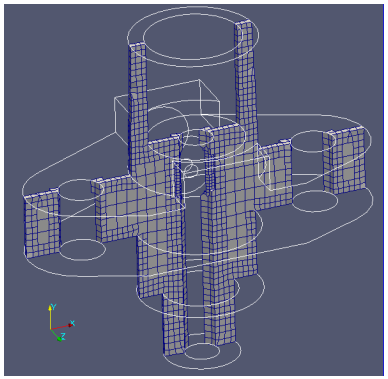
ジョブが開始されたら，チュートリアルのログ(通常log.アプリケーション名)を追跡する

解析結果の転送(別端末で実行)

```
rsync -auv txxxxx@ofp.jcahpc.jp:lec-cavity/ ~/lec-cavity/
cd ~/lec-cavity/flange/
touch pv.foam
```

ParaViewで可視化

1. File/Open *flange/pv.foam*→OK
2. Mesh Regions *select*→Apply
3. Filters/Extract Block/Block Indices/ patches *select*→Apply　(全FileterはAlphabeticalにある)
4. Filters/Feature Edges→Apply→Coloring/Solid Color
5. Pipeline browser/ExtractBlock1 *hidden*, *pv*.foam *select*
6. Filters/Slice/Z Normal→Show Plane,Triangulate the slice *uncheck*, Crinkle slice *check*→Apply
7. Representation/Surface With Edges→ Coloring/Solid Color
8. File/Disconnect



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

その他のチュートリアルの実行

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
finiteArea	有限面積法
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．でのカテゴリ，ケース名，以下のサイトの情報等を参考に，実行したいケースを選ぶ．

[XSIM] XSim OpenFOAM 付属チュートリアル一覧 (https://www.xsim.info/articles/OpenFOAM/Tutorials.html)

[OFT] オープンCAE勉強会@関西OpenFOAMチュートリアルドキュメント作成プロジェクト(https://sites.google.com/site/freshtamanegi/)

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

```
cd ~/lec-cavity
cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/motorBike ./
```

motorBikeケースに移動

```
cd motorBike
```

チュートリアル実行用ジョブスクリプトのコピーと設定変更

```
cp ../foamRunTutorials.sh ./
#並列数を変更するには，例えばsystem/decomposeParDictで並列数を32，methodをscotchに
```

ジョブの投入・ジョブ確認

```
pjsub foamRunTutorials.sh
pjstat
```

ログ確認 (ジョブの実行開始後に行う)

```
tail -f log.*
Ctrl-C
tail -f log.*
```

Endが出てログの更新が止まったら，Ctrl-Cを押して，またtailを実行.

ソルバのログ log.simpleFoam が出てくるまで何度か繰り返す

再実行する場合には以下のように初期化してから再実行を行う(今回は実行しない)

```
foamCleanTutorials
```

チュートリアルの実行例(続き)

チュートリアルの実行結果の転送(別端末で実行)

```
# ↑(カーソル上)を押して前のコマンドを呼び出し，並列計算結果のprocessor*ディレクトリを除外するため，--exclude=processor*を追加して実行
rsync -auv txxxxx@ofp.jcahpc.jp:lecture/ ~/lec-cavity/ --exclude=processor*
```

解析実行時に生成された可視化データを表示する(この場合，pv.foam の作成は不要)

1. File/Open → *motorbike/postProcessing/cuttingPlane/500/U_yNormal.vtk* → OK → Apply → カメラ操作
2. Coloring/・U
3. File/Open → *../../sets/streamLines/500/track0_U.vtk* → OK → Apply
4. Coloring/・U
5. File/Open → *../..//sets/streamLines/500/track0_U.vtk* → OK → Apply
6. File/Open → *postProcessing/sets/wallBoundedStreamLines/500/track0_U_UNear.vtk* → OK → Apply