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
2018年5月8日@東京大学情報基盤センター遠隔会議室

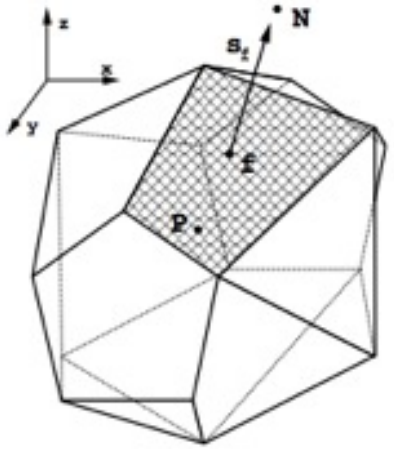
OpenFOAM入門

今野　雅
(株式会社OCAEL・東京大学客員研究員)

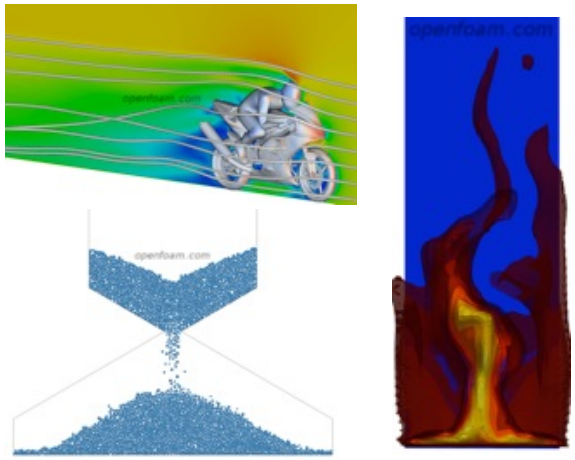
講習会プログラム(予定)

- 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
- 14:15-15:45 キャビティ流れ演習II
  - ✓ icoFoamによる流れ解析
  - ✓ ParaViewによる解析結果可視化
  - ✓ 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
  - ✓ snappyHexMeshによる格子生成
  - ✓ その他チュートリアルの実行

OpenFOAMの概要



有限体積法
ポリヘドラル

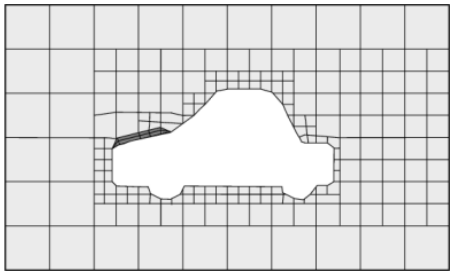


マルチフィジックス

$$\frac{\partial T}{\partial t} + \nabla \cdot (\boldsymbol{U}T) + \nabla \cdot (\alpha \nabla T) = S_T$$

```
solve(fvm::ddt(T)
    + fvm::div(phi,T)
    - fvm::laplacian(DT,T)
   == fvOptions(T));
```

C++



境界適合Hex
メッシャー

乱流モデル:
RAS, LES, DES, …
線型ソルバー :
AMG, PCG, PBiCG, …
離散化スキーム: …

多数のモデル実装済

GPL
Open Source

カスタマイズ可能
低コストな超並列計算

図出典： [OFF] The OpenFOAM Foundation (http://www.openfoam.org/)

OpenFOAMの歴史

- 1989年ー2000年：**研究室のFORTRANコード時代**、開発元：英Imperial CollegeのGosman研(Star-CDの開発元)の Henry Weller, Charlie Hill
- 1993年夏：**事故により全コード消失。**C++で書き直し(FOAM)
- 1999年ー2004年：**商用コード期 (FOAM)** Field Operation And Manipulationの略、開発元：∇Nabla(Henry, Hrvoje Jasak, Mattijs Janssensら)、代理店：CAEソリューションズ(元フルイドテクノロジー)
- 2004年12月：**オープンソース化 (現在のOpenFOAMに名称変更)**、開発元：OpenCFD(Henry, Mattijs, Chris Greenshields)
- 2011年8月15日：**SGIによる買収**、GPL下のソースの管理や配布は、同時に設立されたThe OpenFOAM® Foundationが運用
- 2012年9月12日：**ESIによる買収**、Foundationによる運用は継続
- 2014年：**Henryらが独立(CFD Direct社)．ソースはFoundationが管理**

標準ソルバー・チュートリアルのカテゴリ

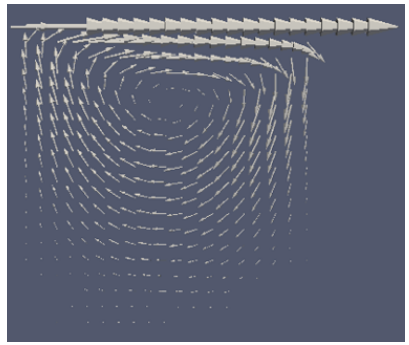
カテゴリ名	カテゴリ内容	カテゴリ名	カテゴリ名
DNS	直接数値解析	finiteArea	有限面積法
basic	基礎的なCFDコード	heatTransfer	熱輸送
combustion	燃焼	stressAnalysis	固体応力解析
compressible	圧縮性流れ	incompressible	非圧縮性流れ
discreteMethods	離散要素法	lagrangian	ラグランジアン粒子追跡
electromagnetics	電磁流体	multiphase	多層流
financial	金融工学		

リリース年	2016				2017		
リリース月	6	10	6	12	6	7	12
バージョン	4.0	4.1	v1606+	v1612+	v1706	5.0	v1712
カテゴリ数	12	12	12	12	12	12	13(+finiteArea)
ソルバ数	82	82	86	86	95	86	101
チュートリアル	207	207	226	253	284	229	298

注) 青色バージョン：OpenFOAM Foundation系, 赤色バージョン：Plus(ESI)系

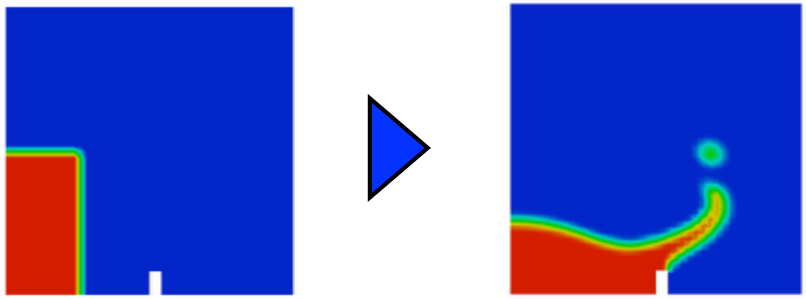
チュートリアルとは

- チュートリアルとは
  - ✓ OpenFOAM標準ソルバーの実行例
  - ✓ `foamRunTutorials` コマンドにより自動的に解析が実行できる
- ユーザガイド第2章で扱っているチュートリアル
  - ✓ cavity：天井駆動のキャビティ流れ

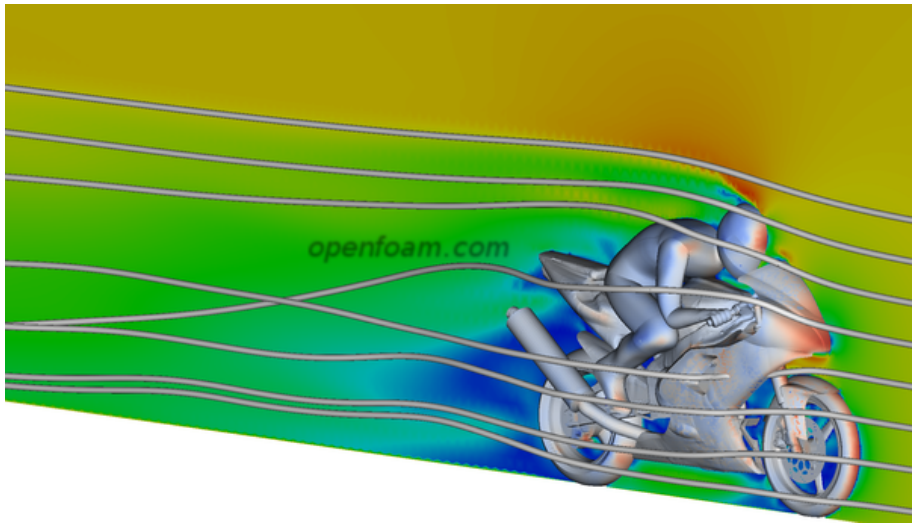


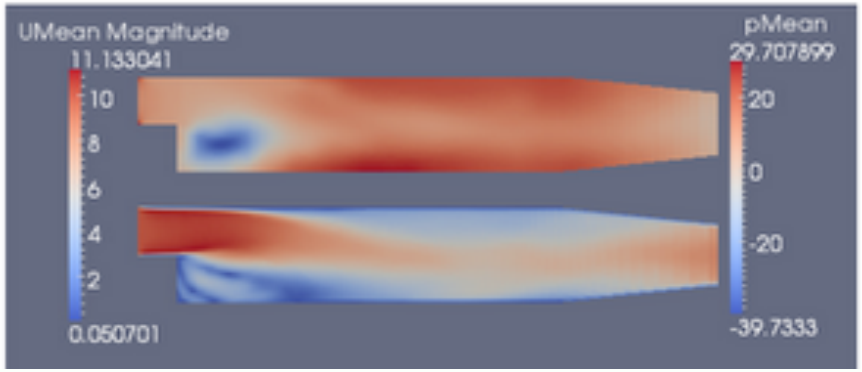
  - ✓ plateHole：穴あき板の応力解析

  - ✓ damBreak：ダムの決壊



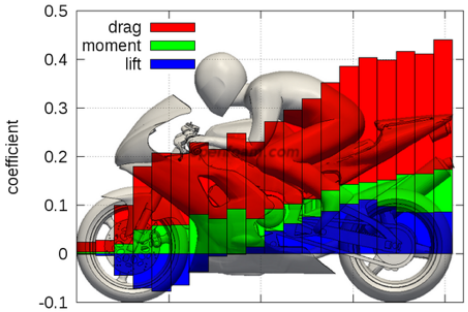
単相等温流れのチュートリアル例





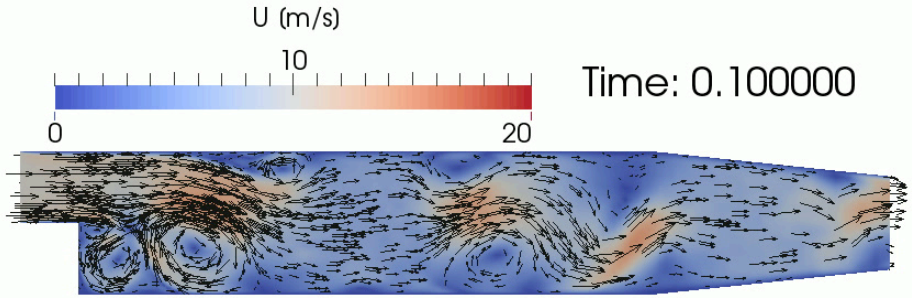
時間平均
圧力(p)

時間平均
速度(U)



バイク周りの流れ
moterBike

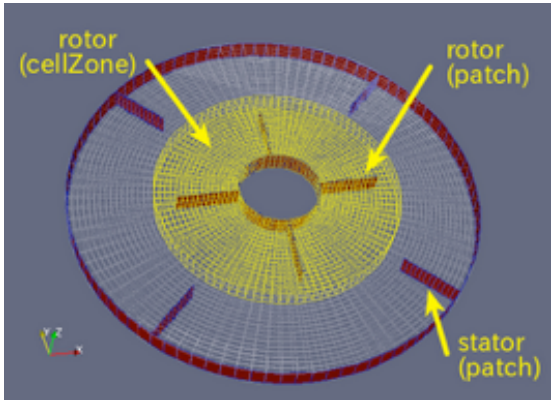
図出典： [OFF]

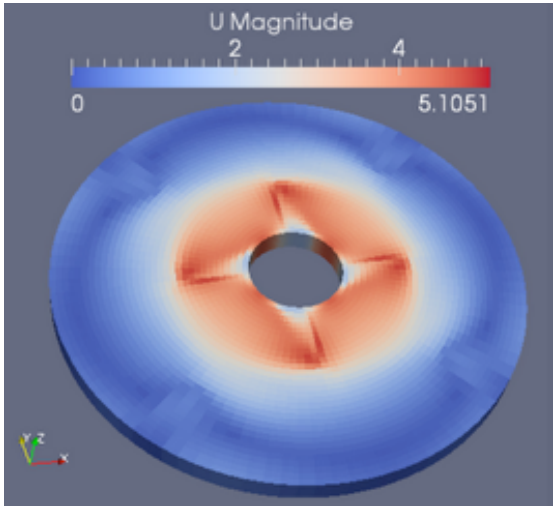


バックステップ流れ(LES)
pitzDaily

[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/)

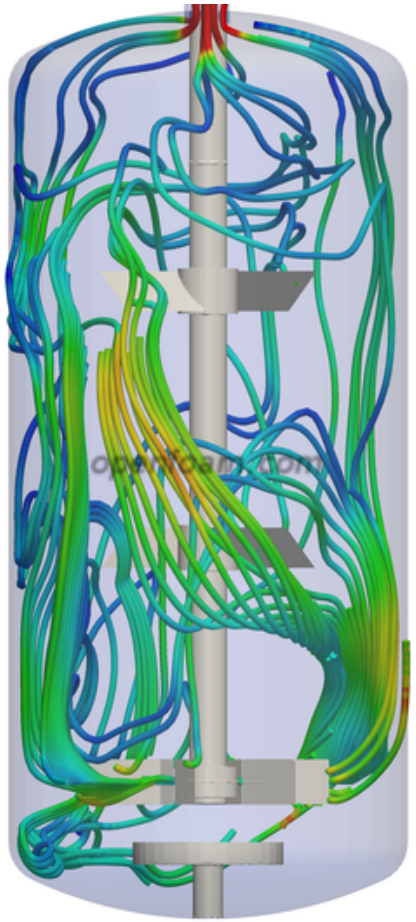
回転撹拌槽のチュートリアル例





回転攪拌槽の流れ(MRF)
mixerVessel2D

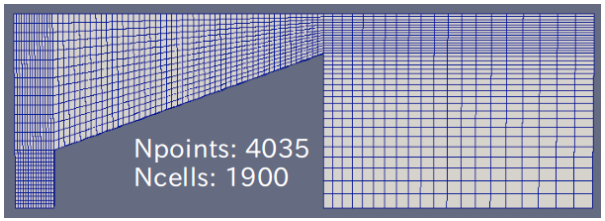
図出典
[OFT]

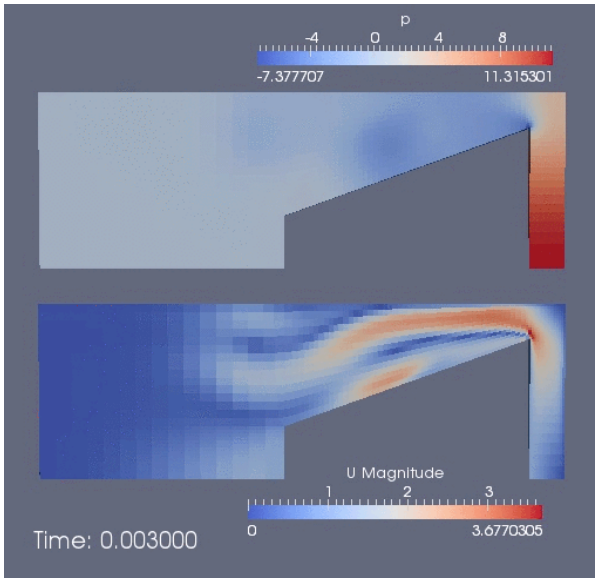


回転攪拌撹拌槽内の流れ
mixerVesselAMI

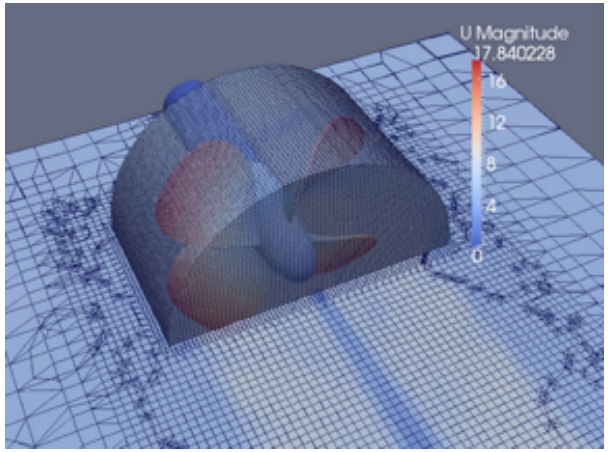
図出典
[OFF]

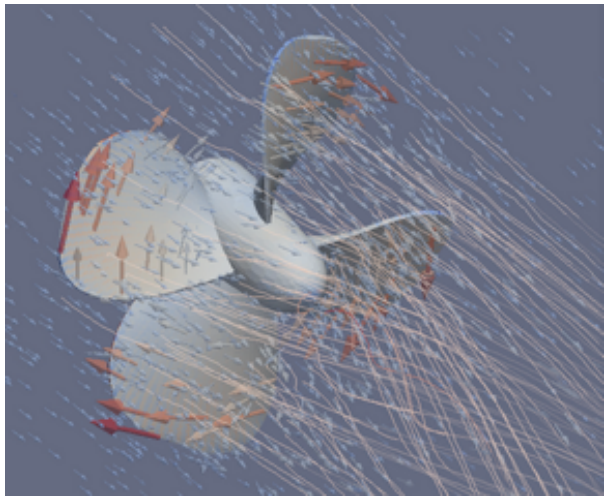
移動格子のチュートリアル例





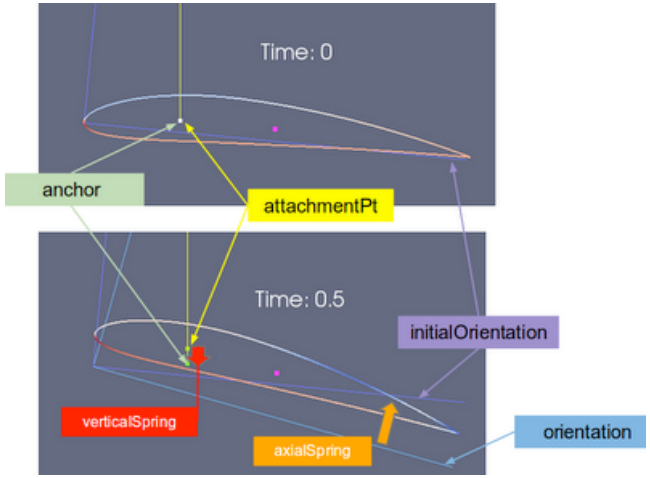
ピストン押し込み流れ
movingCone

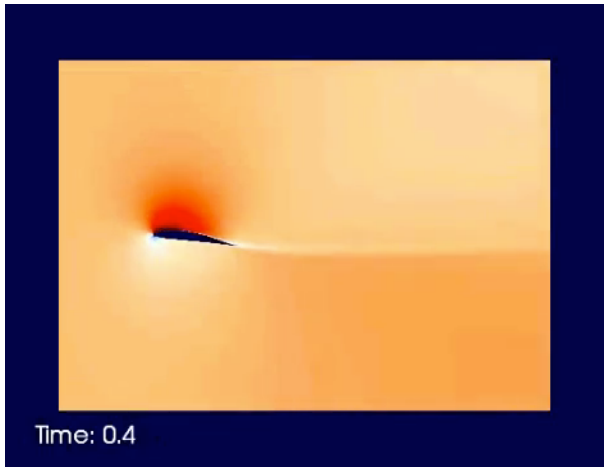




スクリューの回転流れ場
propeller

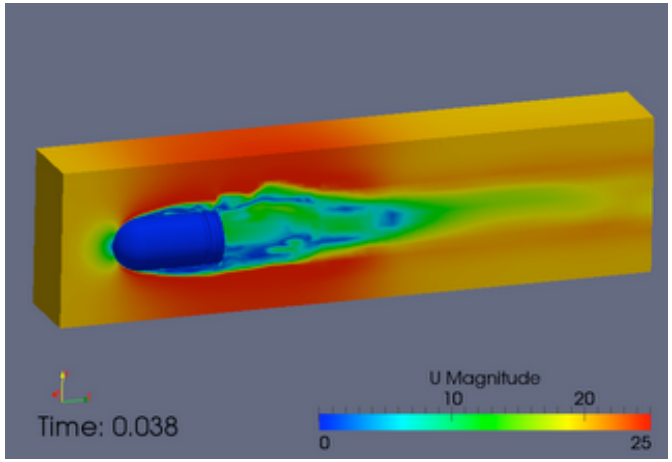
図出典 [OFT]



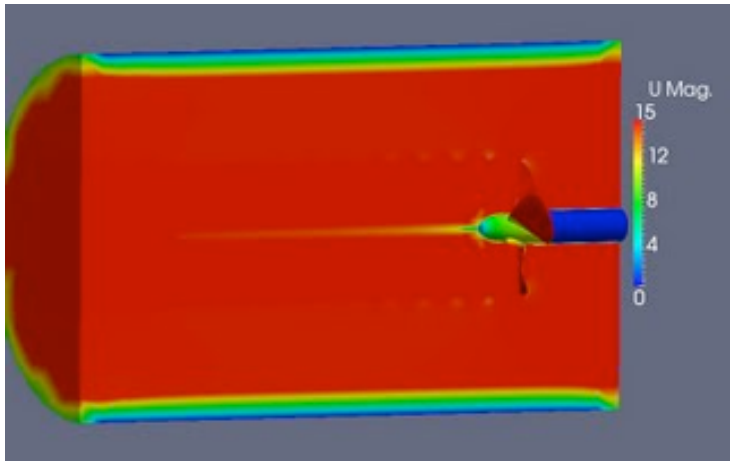


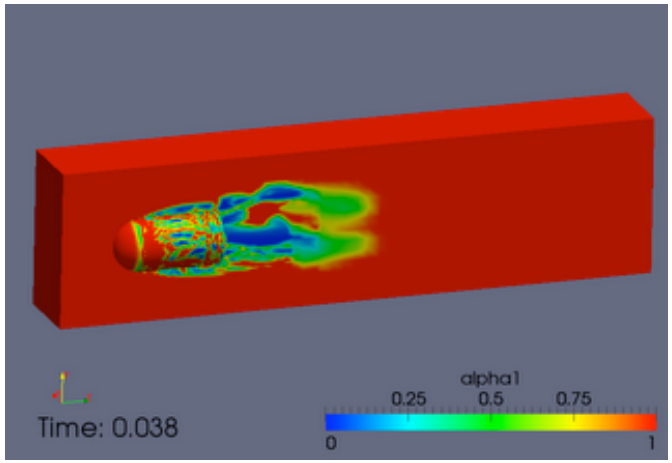
翼型の6自由度剛体運動
wingMotion

相変化のチュートリアル例

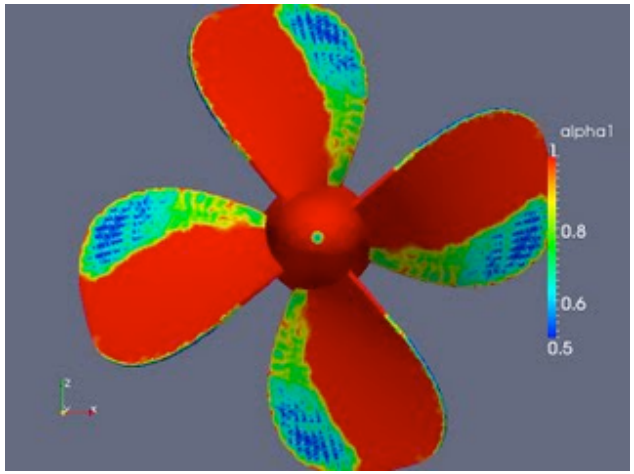


速度





相率



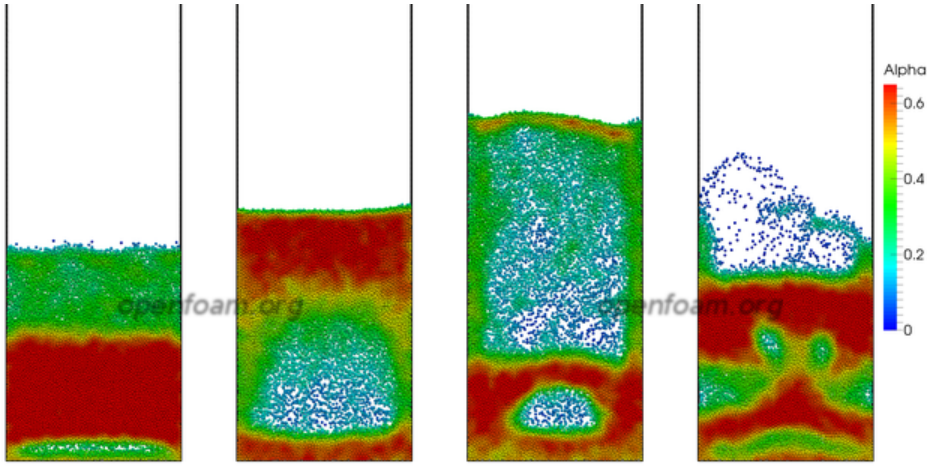
弾丸周りのキャビテーション
cavitatingBullet

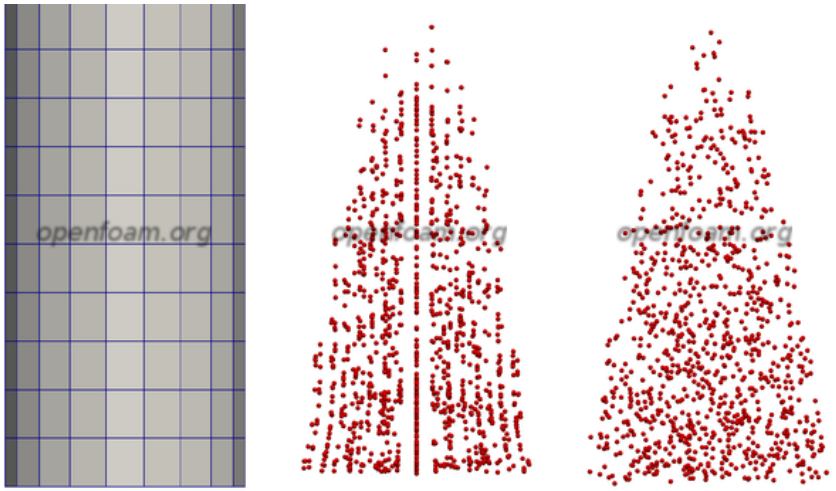
プロペラ周りのキャビテーション
propeller

図出典 [OFT]

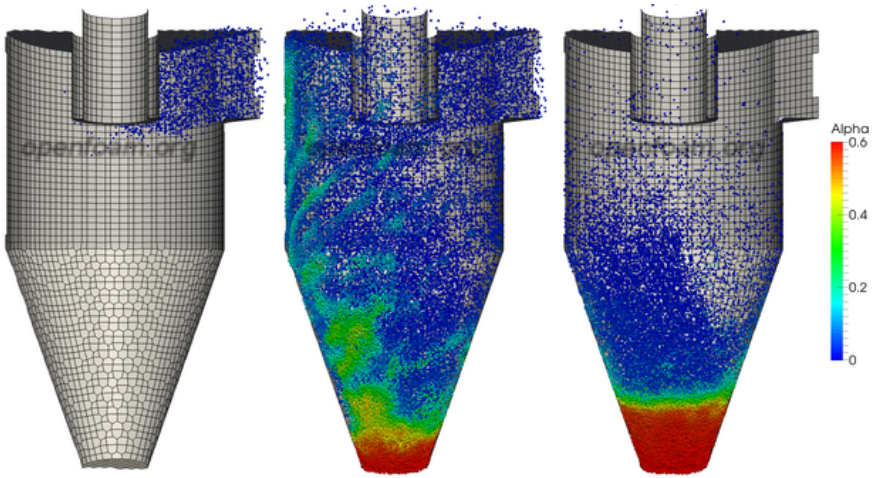
粒子計算のチュートリアル例

図出典： [OFF]

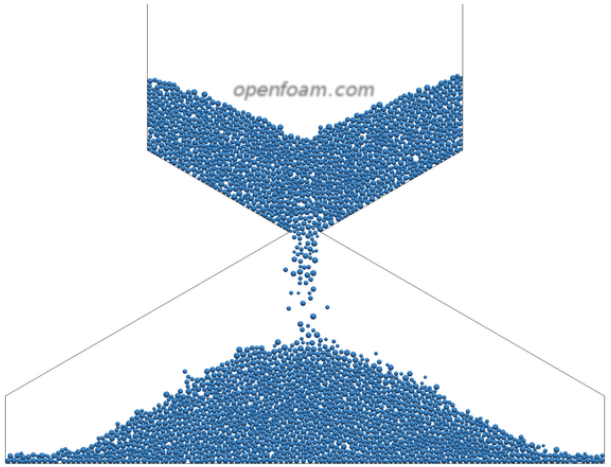




Particle Tracking

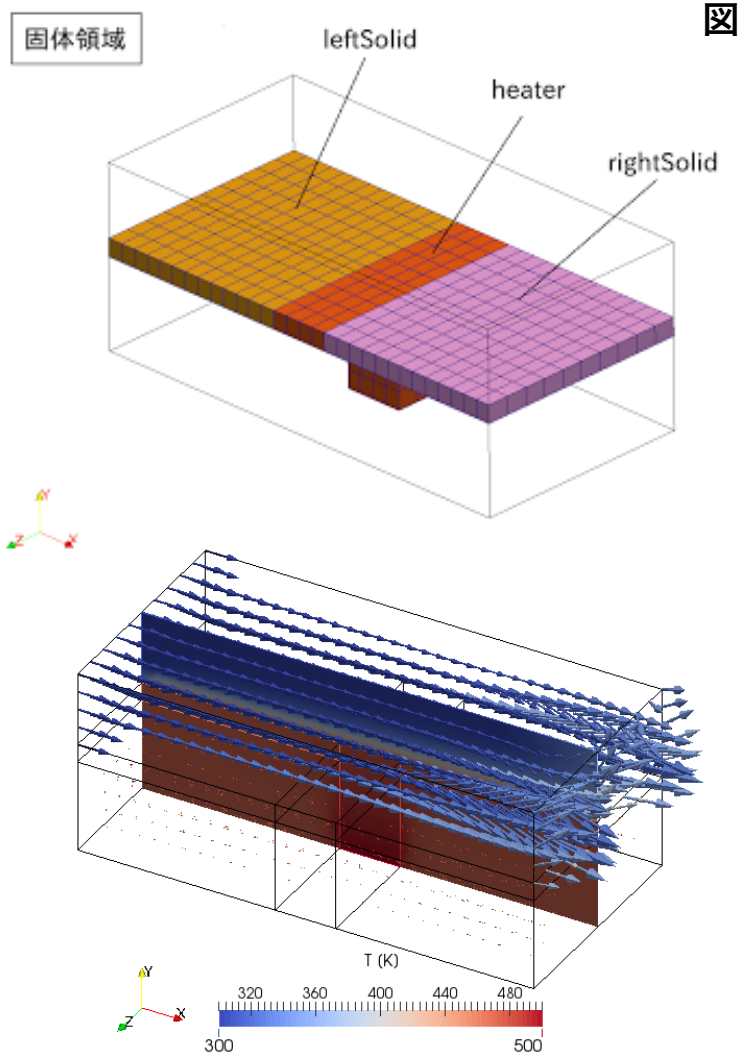


Multiphase Particle-in-Cell
(MP-PIC)



Discrete Element Modeling
(DEM)

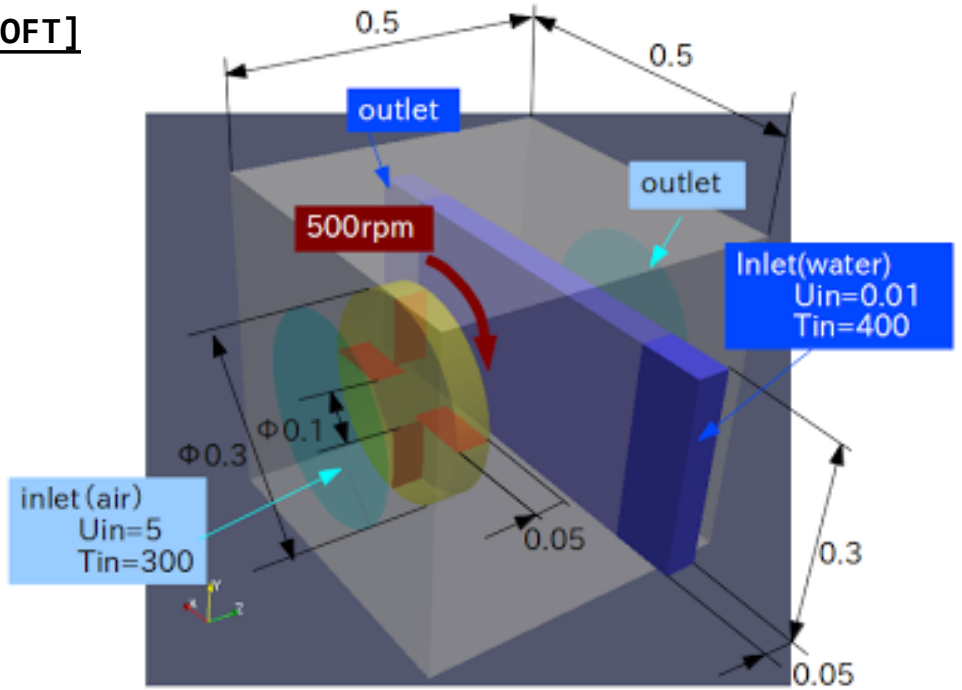
連成熱伝達解析(CHT)のチュートリアル例



連成熱伝達解析

multiRegionHeaterRadiation

図出典 [OFT]



回転する熱交換機解析(MRF)

heatExchanger

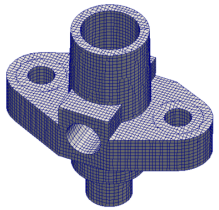
OpenFOAMのCHT解析における欠点

- エネルギー保存式を全領域で連成せず，領域毎に解くので収束が遅い
- 形態係数を用いた放射解析の精度が悪い
- 熱収支計算が容易ではない(熱解析共通)

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]

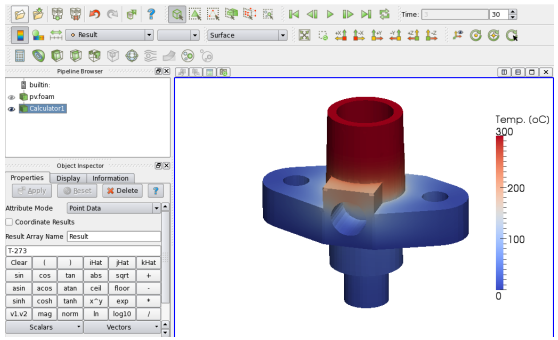
領域分割(並列計算時)
[decomposePar]

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]

OpenFOAMの稼働環境

Linux, Mac, Windows機で動作

ITO (Xeon機)
FOCUS (Xeon機)
FX10, FX100 (SPARC64機)
東工大 TSUBAME (GPU機あり)
東大 Reedbush-U,H (Xeon機+GPU機)
JCAHPC Oakforest-PACS (Xeon Phi機)
Etc.

Laptop　PC　クラスタ　クラウド　スーパーコンピュータ　フラッグシップ
～100万格子　～1000万格子　～10億格子　～1000億格子

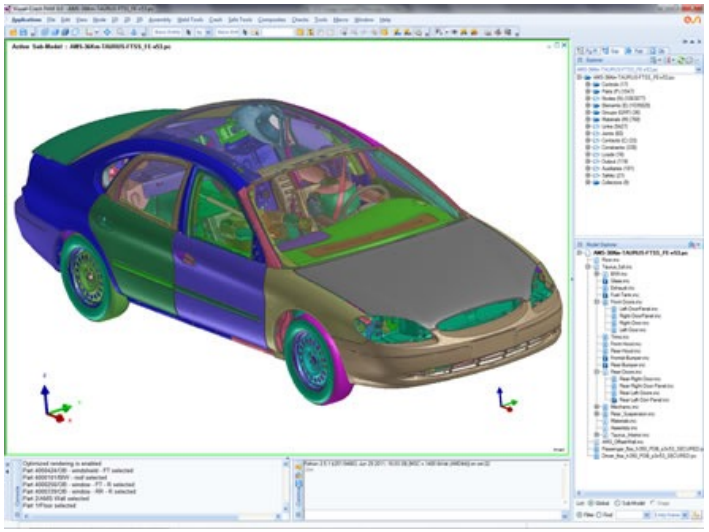
Amazon EC2(GPU機あり)
Microsoft Azure(GPU機あり)
富士通TCクラウド
Etc.

京 (SPARC64機)
RISTがHPCI課題の京ユーザ向けに最適化を支援

OpenFOAMの主な派生(Fork)版

- **商用版**

図出典：ESI (https://www.esi.co.jp/news/2014/PressRelease_0128.html)



Visual-CFD(ESI)

✓ **HELYX(Engys):** OF拡張版+GUI

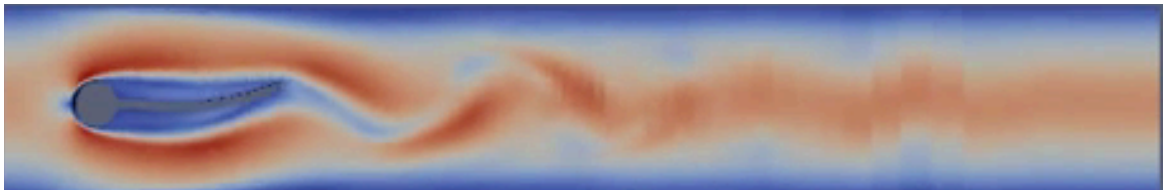
✓ **iconCFD(IDAJ, ICON):** OF拡張版+GUI

✓ **Visual-CFD(ESI):** GUI

- **オープンソース版**

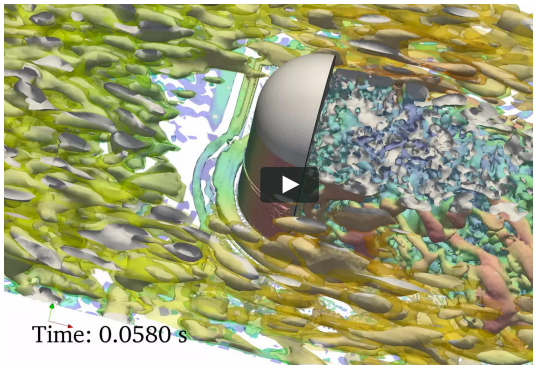
✓ **HELYX-OS(Engys):** GUI

✓ **foam:** Hrvoje Jasak(クロアチア ザグレブ大学教授，Wikki社 代表)が主導するコミュニティベース版．FSIやBlock coupledソルバ等の公式版に無い機能を実装



foamの 流体・構造連成(FSI)

✓ **OpenFOAM+(ESI):** 安定化と機能拡張

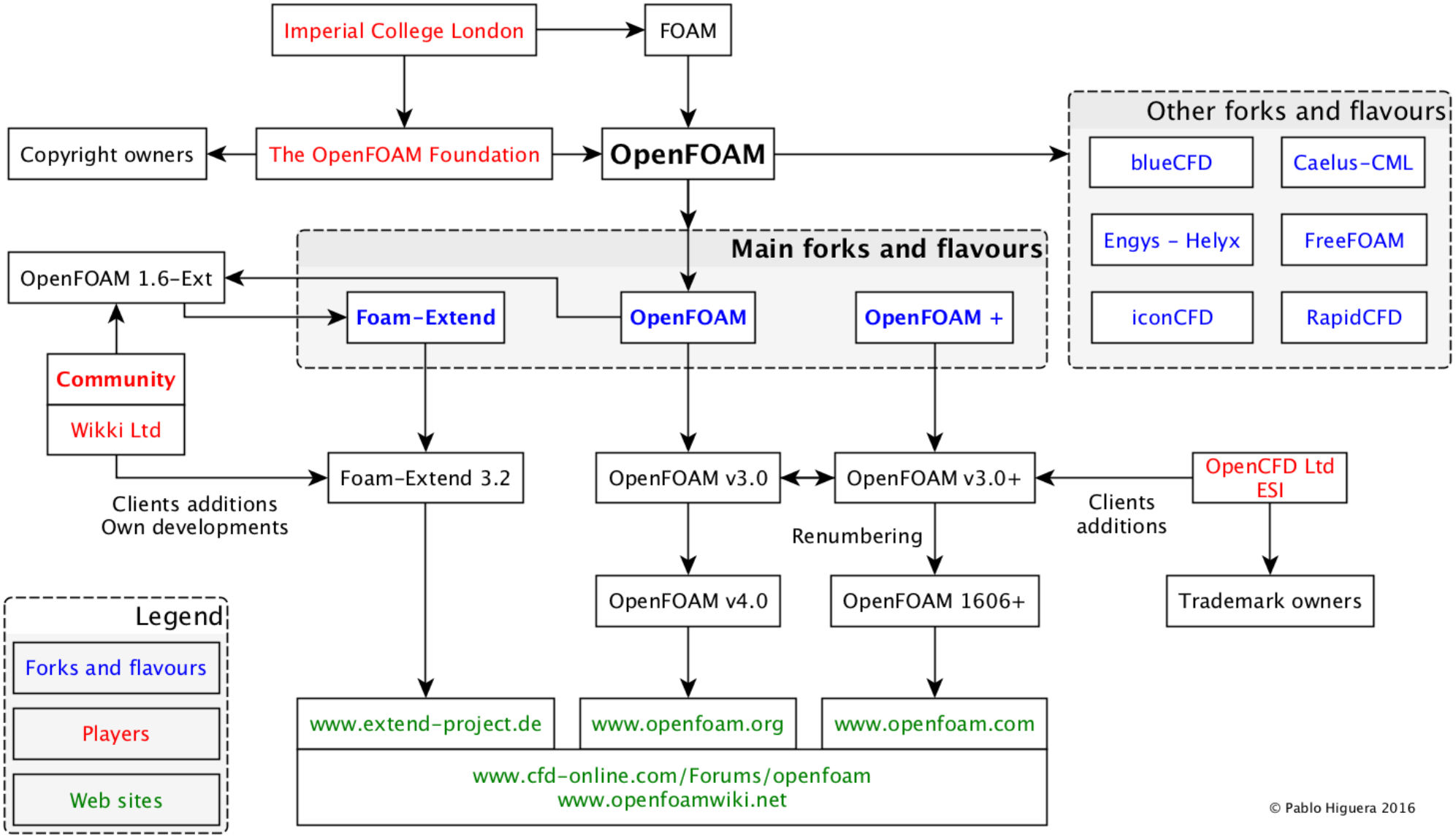


OpenFOAM+の変動風生成

図出典：www.openfoam.com

OpenFOAMの派生図

図出典：olaFoam (https://sites.google.com/site/olafoamcfd/)

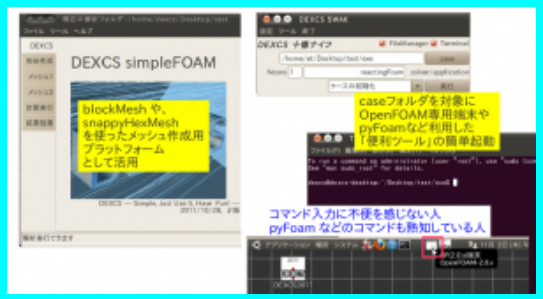


OpenFOAMの課題

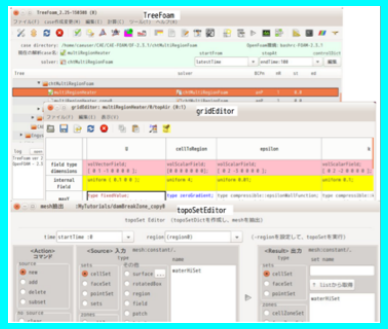
- 設定用GUIが無く，全てテキストファイルで設定する必要があるが，詳細な公式マニュアルがほとんど無い → ソースコードを読まないと詳細な設定方法がわからないので，初心者には設定が困難．解析条件に応じた推奨設定も不明

- GUIに関しては商用(iconCFD, Visual-CFD)，オープンソース(HelyxOS, DEXCS, TreeFoam)などが続々登場してきた

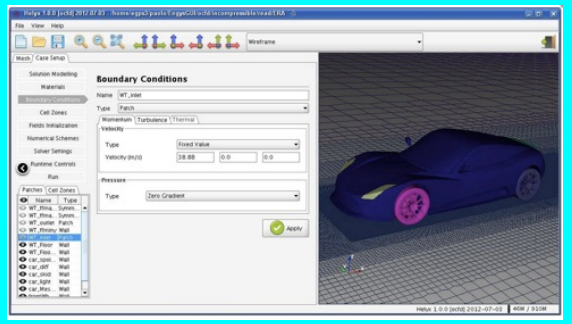
オープンソースGUI例



DEXCS (野村氏作)



TreeFoam(藤井氏作)



HelyxOS(engys社)

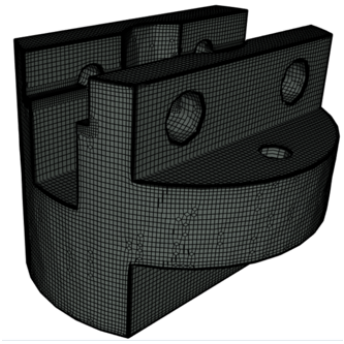
OpenFOAMの課題

- メッシャー(blockMesh, snappyHexMesh)で格子を生成するのが遅く，質の良いレイヤを貼るのは困難 → 初心者は格子生成で断念

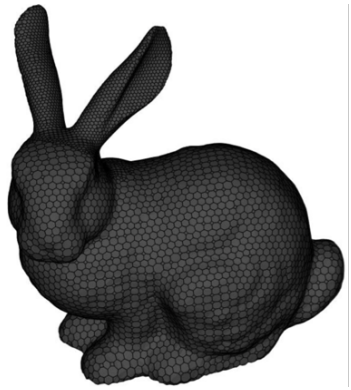
- Pointwise，HEXPRESSなどの商用メッシャーはOpenFOAMの格子を出力できるようになっている.
- Helyx, iconCFDなどの商用ForkではsnappyHexMeshの機能を改善
- オープンソースでハイブリッド並列，レイヤ付加性能に優れたOpenFOAM用メッシャーcfMeshも登場(v1712に取り込まれた)

cfMesh

cfMeshによるレイヤ付き六面体格子

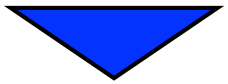


cfMeshによるポリヘドラル格子



OpenFOAMの課題

- ハイブリッド並列の非対応 → GPGPUやXeon Phi等のメニーコア機で非効率．ハイブリッド並列よりMPIプロセス数が多くなるので，MPIプロセス間の通信コストがかかる．



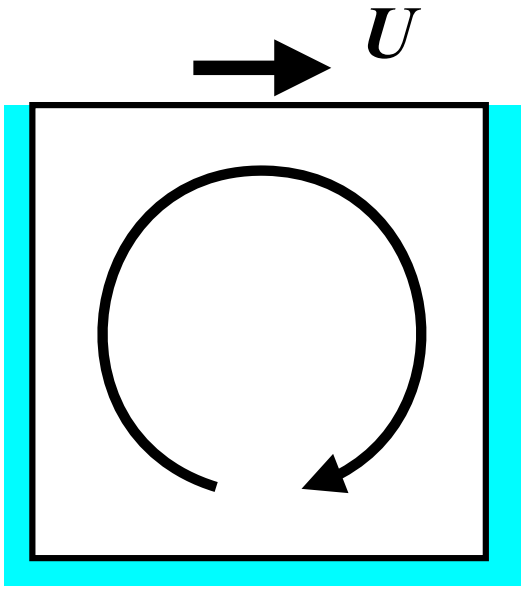
以下のような様々な研究や実装が行われている

- Amani AlOnazi: Design and Optimization of OpenFOAM-based CFD Applications for Modern Hybrid and Heterogeneous HPC Platforms, Master Thesis, King Abdullah University of Science and Technology, 2014, URL
- 櫻井，片桐ら: 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, URL
- 内山，フックら: OpenFOAMによる流体コードのHybrid並列化の評価，情報処理学会, 2015, URL
- 山岸，井上ら: OpenFOAMのメニーコア・GPUへの対応に向けた取り組みの紹介，オープンCAEシンポジウム, 2017, URL
- 富岡，吉藤ら: OpenFOAMスレッド並列化のための基礎検討, オープンCAEシンポジウム, 2017, URL
- 今野: OpenFOAMにおけるCommunication-Avoiding CG法の実装と性能評価, オープンCAEシンポジウム, 2017, URL
- simFlow社: RapidCFD(NVIDIA CUDA用フルGPU版OpenFOAM), オープンソース, URL

キャビティ流れ演習I

キャビティ流れとは

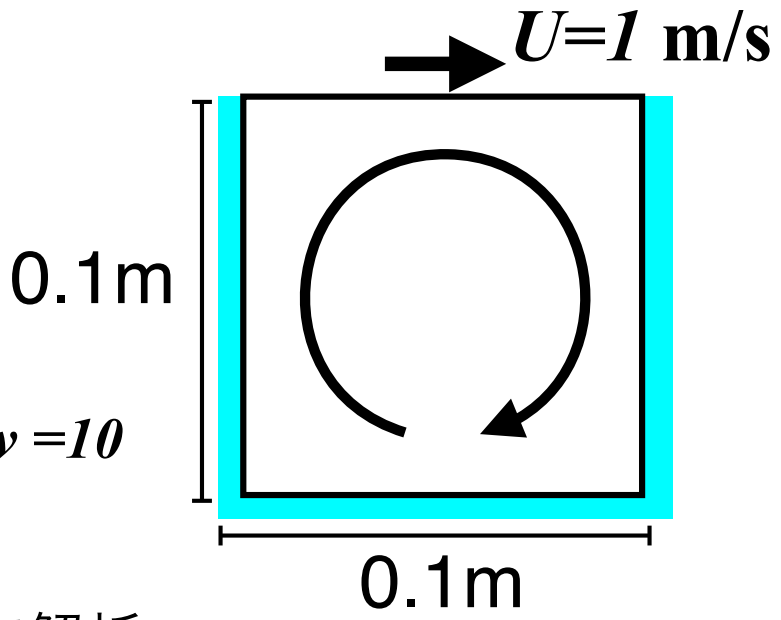
- キャビティ(空洞)の上壁が動き、その摩擦で空洞内の流体が動く流れ場
- 単純な形状と境界条件でCFDでの設定が容易
- レイノルズ数の増加に伴ない，１次循環渦の大きさや中心位置，２次以上の循環渦の有無や大きさなどの様相が変化する
- CFDソフトウェアの基礎的な検証例(ベンチマークテスト)として良く用いられる
- Ghiaらの計算結果との比較が多い



[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)：***v=0.01*** **m²/s**
- レイノルズ数(慣性力と粘性力の比)：***Re= dU/v =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



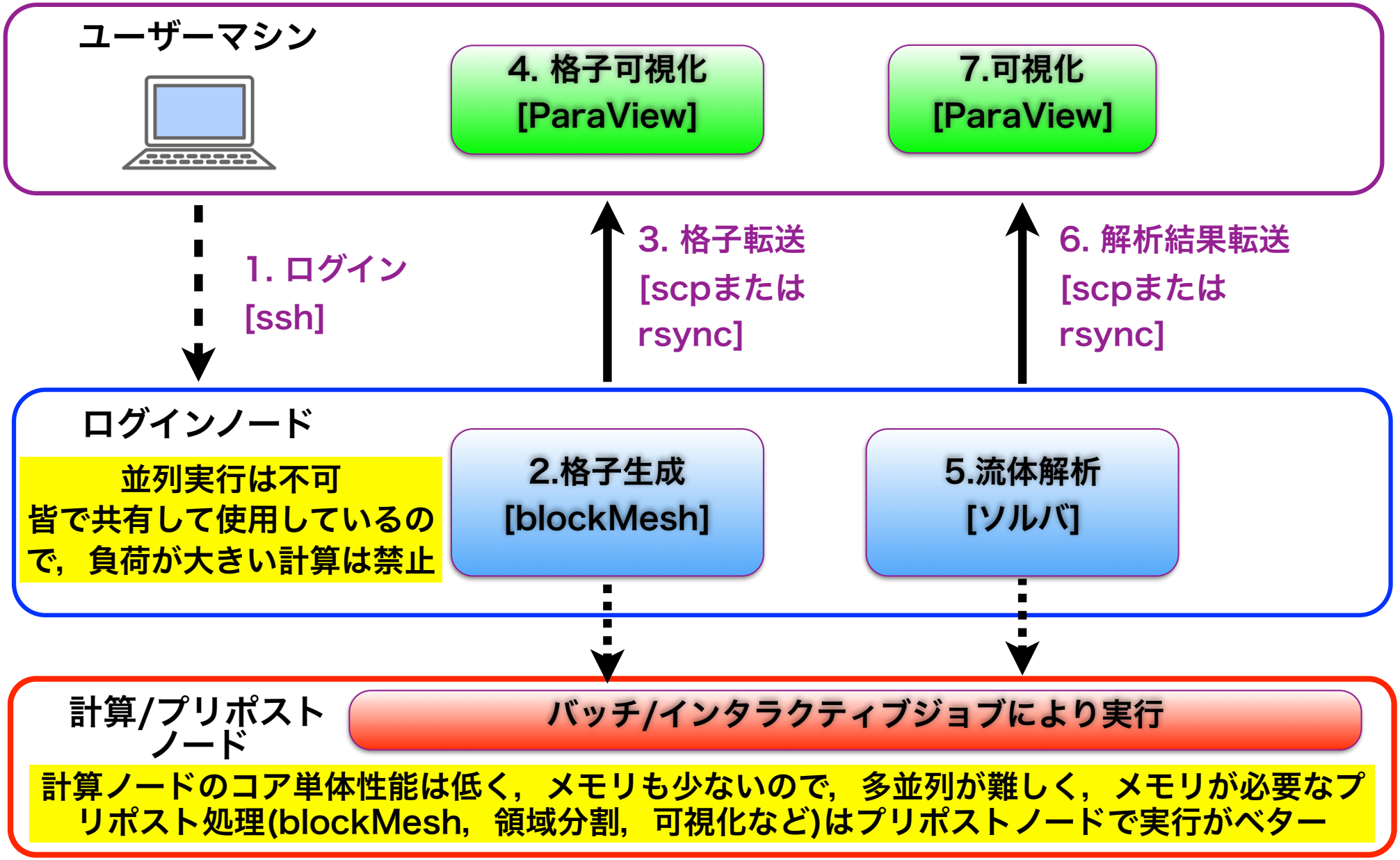
非定常非圧縮層流解析ソルバicoFoamの基礎方程式

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 /work/gt00/share/lecture20180508.tar.xz ./
tar xf lecture20180508.tar.xz
```

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/lecture ~/lecture
```

講習会用ディレクトリへの移動と中身確認

```
cd ~/lecture
ls
```

```
bin etc 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
```

-1オプションは必須ではないが，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

OpenFOAM関連のmodule

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
```

```
----------- Module Specific Help for 'openfoam/4.1' --------------
:
(2) Make a job script
    [username@ofp01 ~]$ vi run.sh
    #!/bin/sh

    #PJM -L rscgrp=regular
    #PJM -L node=1
    #PJM -L elapse=1:00:00
    #PJM -j

    module purge
    module load gcc/4.8.5
    module load openfoam/4.1
    source $WM_PROJECT_DIR/etc/bashrc
```

バッチジョブの設定(後述)

module版OF-4.1の環境設定部分

標準で有効なmoduleをpurgeで全てunload

Gcc-4.8.5のmoduleをload

OF-4.1のmoduleをload

OpenFOAMの環境設定

gcc/4.8.5をloadするとIntel MPIのmodule impi/2017.3.196が自動的にloadされる

module版OpenFOAM 4.1の環境設定

module版OpenFOAM 4.1の環境設定

```
module purge
module load gcc/4.8.5
module load openfoam/4.1
source $WM_PROJECT_DIR/etc/bashrc
```

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，これらのノードでもmodule helpでのjob scriptにおける環境設定部分を実行すれば良い

OpenFOAM環境設定用エイリアス

```
more ~/lecture/etc/bashrc
```

```
# OpenFOAM-4.1(gcc/4.8.5, impi/2017.3.196)
alias OF41='\
module purge;\
module load gcc/4.8.5;\
module load openfoam/4.1;\
source $WM_PROJECT_DIR/etc/bashrc\
'
# OpenFOAM-3.0.1(gcc/4.8.5, impi/2017.3.196)
:
```

上記を1コマンドで実行できるようエイリアス(別名)を定義する．様々なバージョンやビルド環境(コンパイラ，オプション，MPIライブラリ，整数・実数変数bitサイズ)のOF設定の切替にも便利

ログイン時にエイリアスが自動的に有効になるように~/.bashrcに設定

```
echo 'source $HOME/lecture/etc/bashrc' >> ~/.bashrc
source ~/.bashrc
```

エイリアスによるOpenFOAM環境設定

エイリアス (OFバージョン)	module	コンパイラ	最適化フラグ (動作しないノード)	整数ラベルbit数	MPIライブラリ
OF301	○	Gcc 4.8.5	-O3(なし)	64	Intel MPI 2017.3 Intel MPI 2018.0
OF41	○	Gcc 4.8.5	-O3(なし)	64	Intel MPI 2017.3 Intel MPI 2018.0
OF1712Icc	×	Intel 2018.1	-O3 (なし)	32	Intel MPI 2018.1
OF1712IccKNL	×	Intel 2018.1	-O3 -xmic-avx512 (ログイン/プリポスト)	32	Intel MPI 2018.1

- OpenFOAM Foundation版(3.0.1, 4.1等)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Plus版(v1712等)を使用する必要がある
- 整数ラベルbit数が64の場合，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
- 本演習ではOpenFOAM自動ビルドスクリプトinstallOpenFOAMでビルドしたv1712を使用
  OFではKNL用最適化フラグ(-xmic-avx512)による速度向上は僅かなので，今回は使用しない

ログインノードで動作するIcc版OF-1712の環境設定

```
OF1712Icc
```

OpenFOAMのエイリアス

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 wmDP='wmSet WM_PRECISION_OPTION=DP'
alias wmSP='wmSet WM_PRECISION_OPTION=SP'
alias wmInt32='wmSet WM_LABEL_SIZE=32'
alias wmInt64='wmSet WM_LABEL_SIZE=64'
alias wmSet='. $WM_PROJECT_DIR/etc/bashrc'
alias wmUnset='. $WM_PROJECT_DIR/etc/config.sh/unset’
alias wmSchedOn='export WM_SCHEDULER=$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
ビルド済実数倍精度環境へ変更
ビルド済実数倍精度環境へ変更
ビルド済32bitラベル環境へ変更
ビルド済64bitラベル環境へ変更
環境設定更新
環境設定初期化
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

キャビティケースのコピー

cavityのケースのコピー

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

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

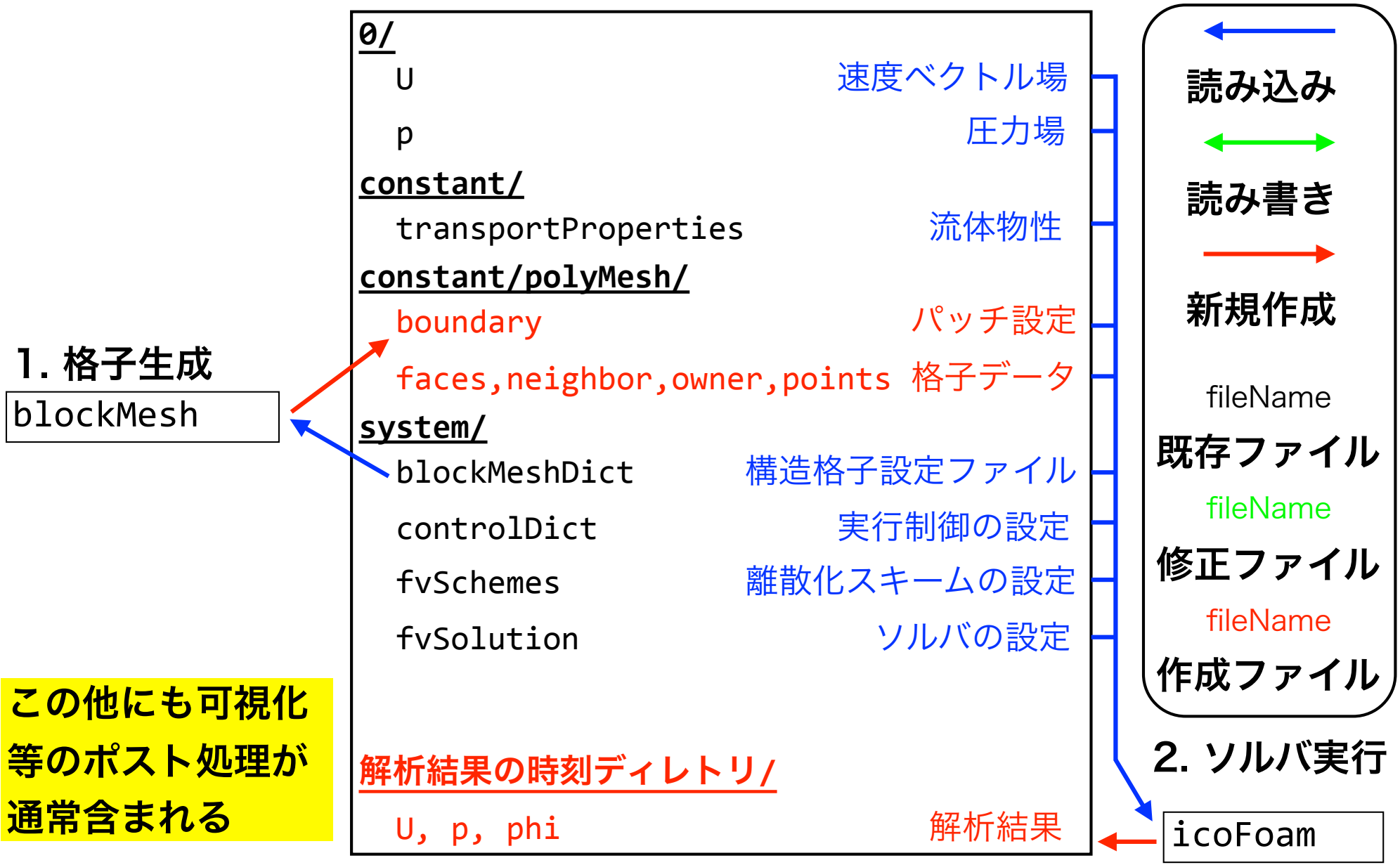
ジョブスクリプトなどの講習用ファイルのコピー

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

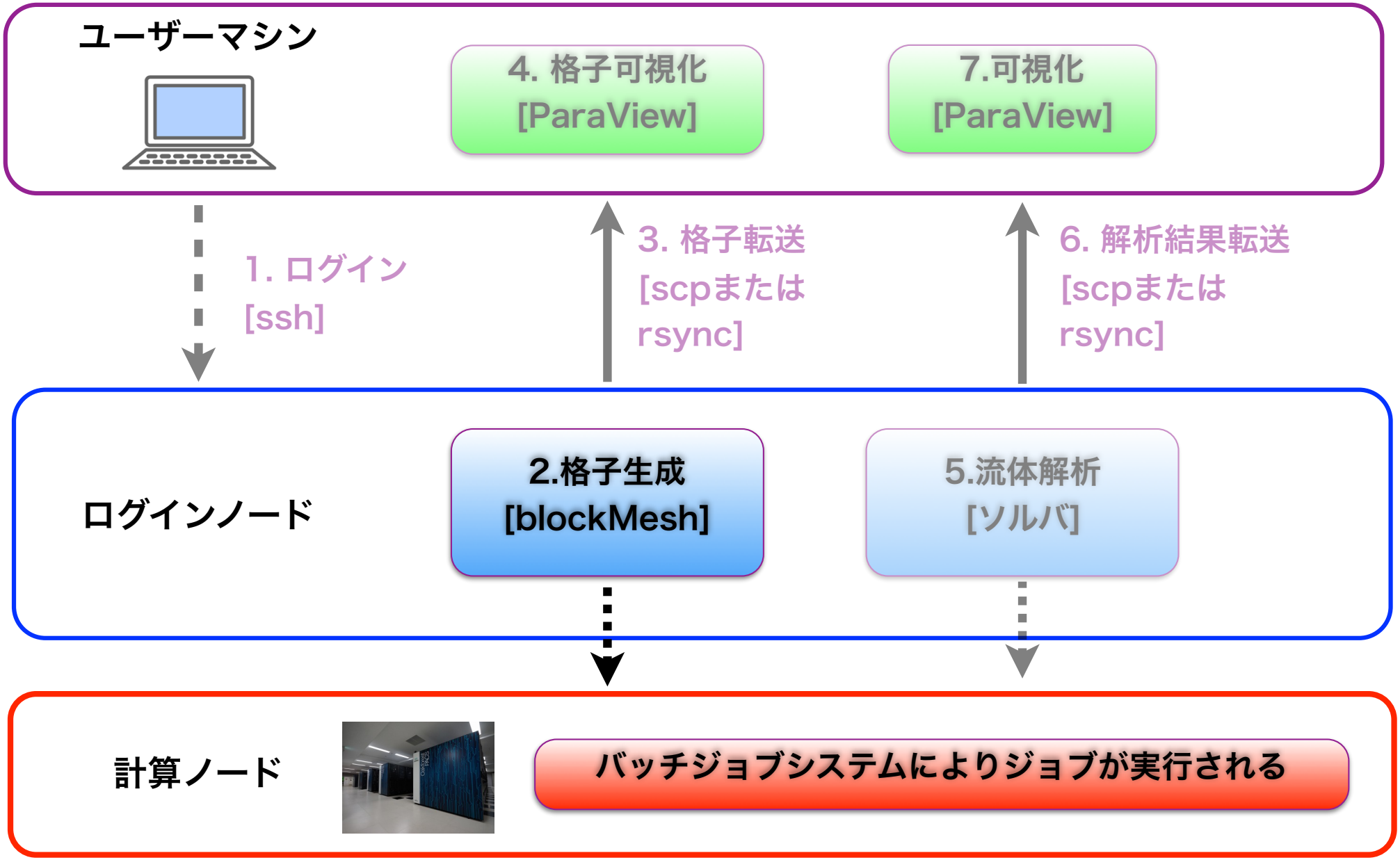
キャビティケースのディレクトリ構成

```
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
```

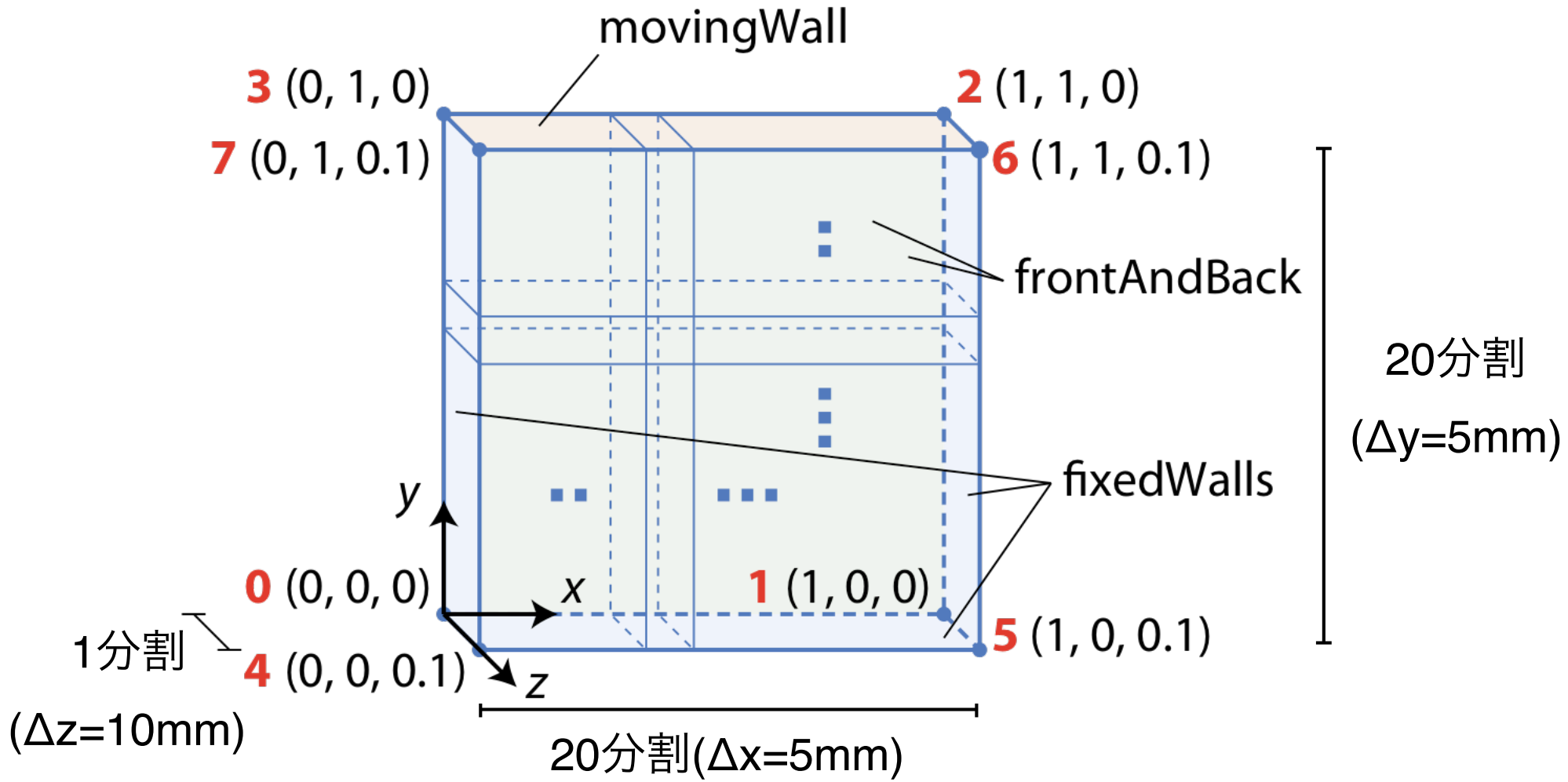
キャビティケースの解析手順



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:  plus                                  |
|   \\  /    A nd           | Web:      www.OpenFOAM.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

scale 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

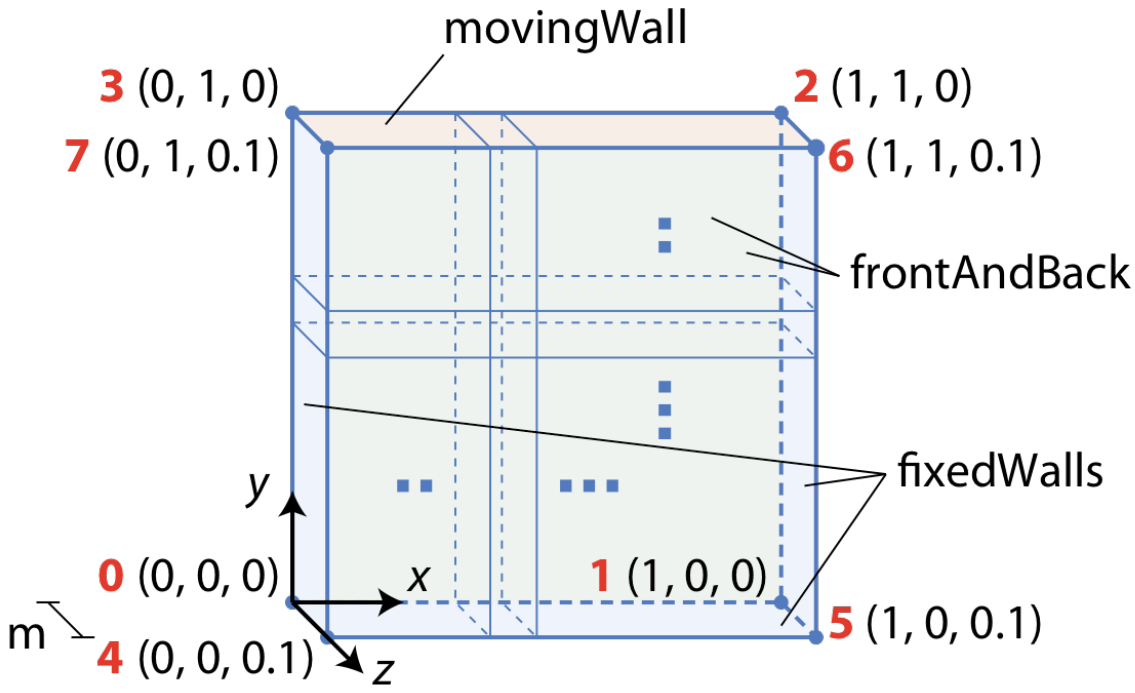
moreコマンドは続ける(英語版ではmore)と表示してキー入力待ちとなる．

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

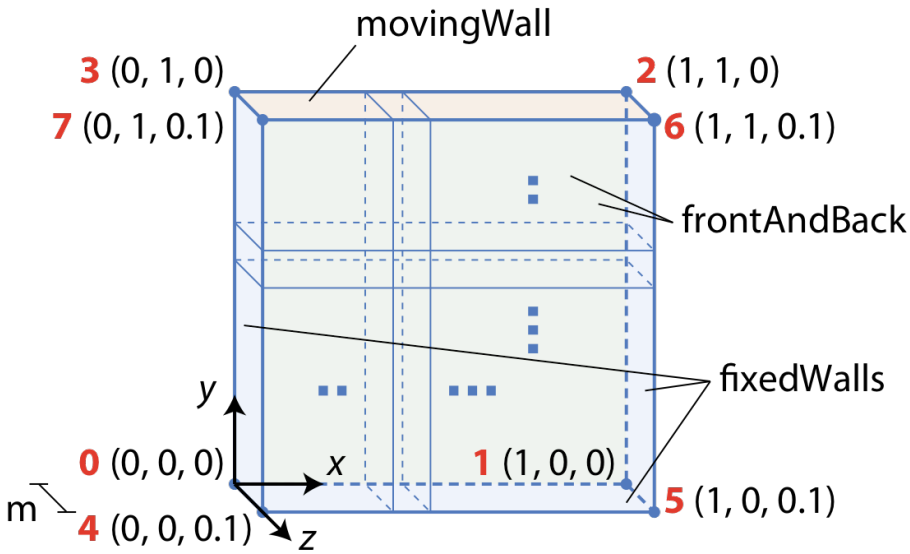
```
scale 0.1; //メートル単位への変換係数

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

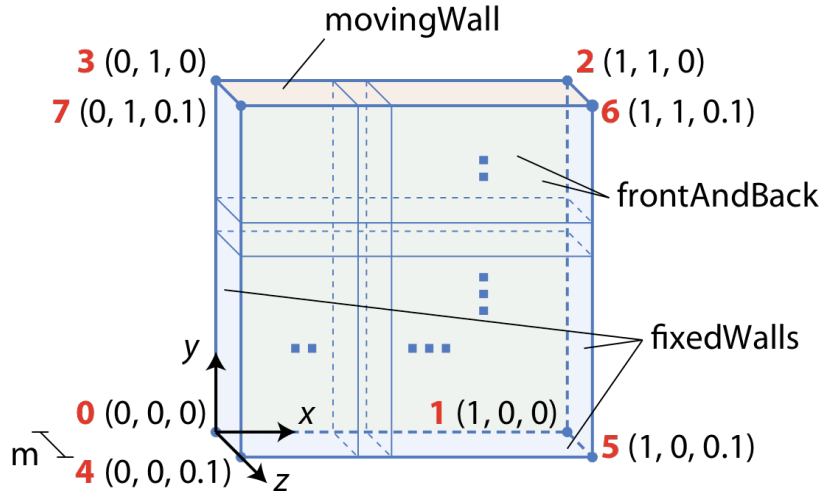
```
blocks   //格子ブロックの定義
(
   hex (0 1 2 3 4 5 6 7) (20 20 1) simpleGrading   (1 1 1)
//形状  頂点番号リスト        分割数      格子幅は等比級数  格子幅比
//格子幅比は最初と最後の格子の比.  1=等間隔.
);

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

more blockMesh.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
module load intel/2018.1.163
export HOME=/work/gt00/$USER
. /work/gt00/share/OpenFOAM/OpenFOAM-v1712/etc/bashrc \
WM_COMPILER=Icc2018_1_163 WM_MPLIB=INTELMPI2018_1_163
env
blockMesh >& $PJM_JOBNAME.l$PJM_JOBID
```

リソースグループ名(必須)

ノード数(必須)

経過時間制限値([[hour:]minite:]second)(注)

ジョブ実行時にトークンを消費するプロジェクト(必須)

ノードごとを含むジョブ統計情報をファイルに出力

標準で有効なmoduleに依存しないよう，module purgeで全moduleをunload後，必要分をload

$HOMEを参照するOFのスクリプト用に設定

OF-v1712の環境設定

WM_COMPILER(コンパイラ)，WM_MPLIB(MPI)などの変数を引数にして環境を切替え可能

念のため実行時の環境変数を記録しておく

blockMeshの実行．ログファイル名は任意だが，ここでは，"ジョブ名.1ジョブID"とした

注)空いている隙間リソースに割りあてるバックフィルスケジューリング機能により実行開始が早まる可能性が高くなるので，経過時間を見積って適切な経過時間制限値を設定する

blockMeshのジョブ投入

ジョブの投入

```
pjsub blockMesh.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
```

→ 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
├── blockMesh.sh
├── blockMesh.sh.[eilo]1234567
├── constant
│   ├── polyMesh
│   │   ├── boundary
│   │   ├── faces
│   │   ├── neighbour
│   │   ├── owner
│   │   └── points
│   └── transportProperties
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── fvSchemes
    └── fvSolution
```

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

*.l1234567 : blockMeshのログ

*.o1234567 : ジョブの標準出力

生成された格子ファイル

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.e*
```

blockMeshのログ確認

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

blockMesh.sh.lジョブID

```
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

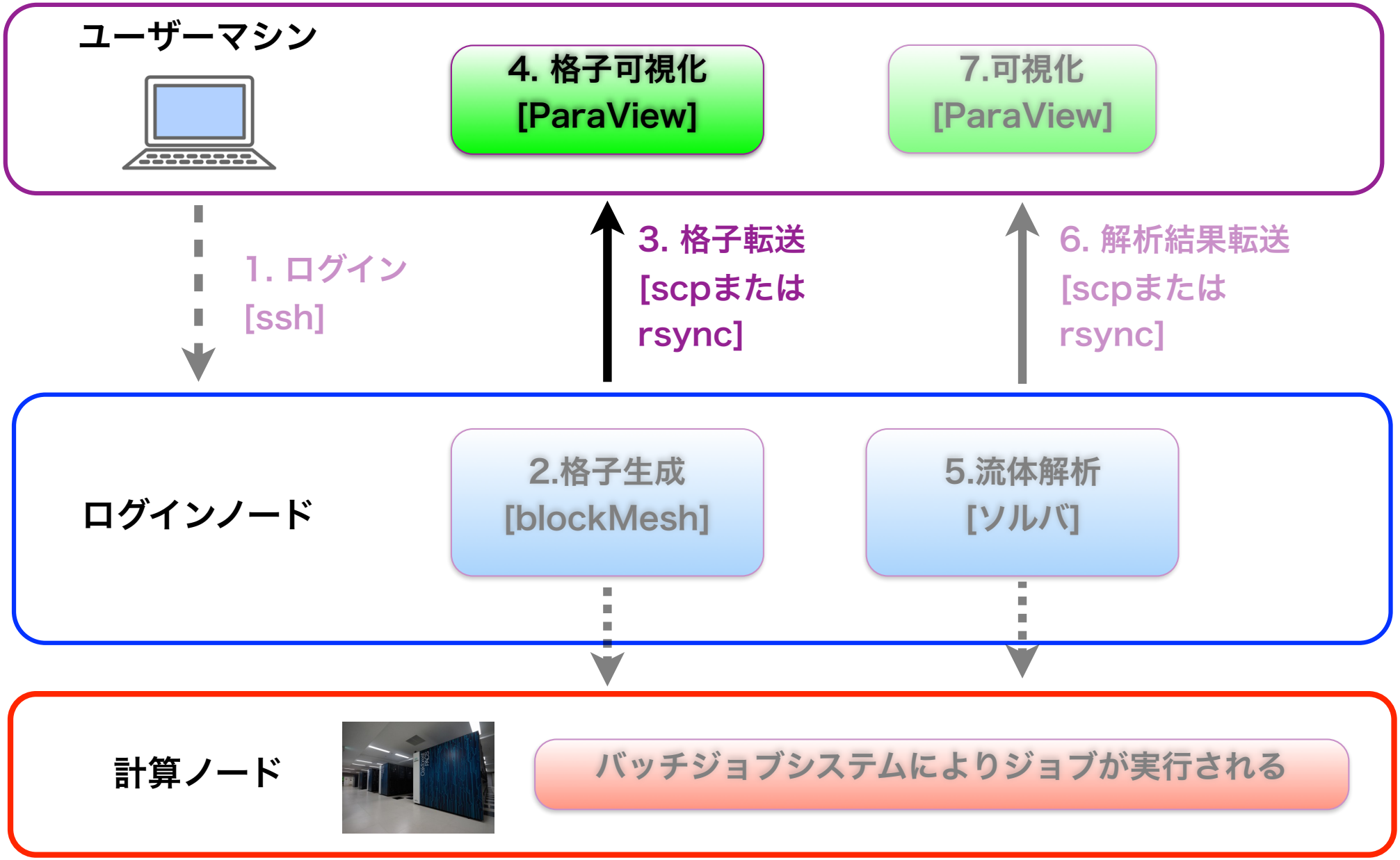
格子数

界面(フェース)数

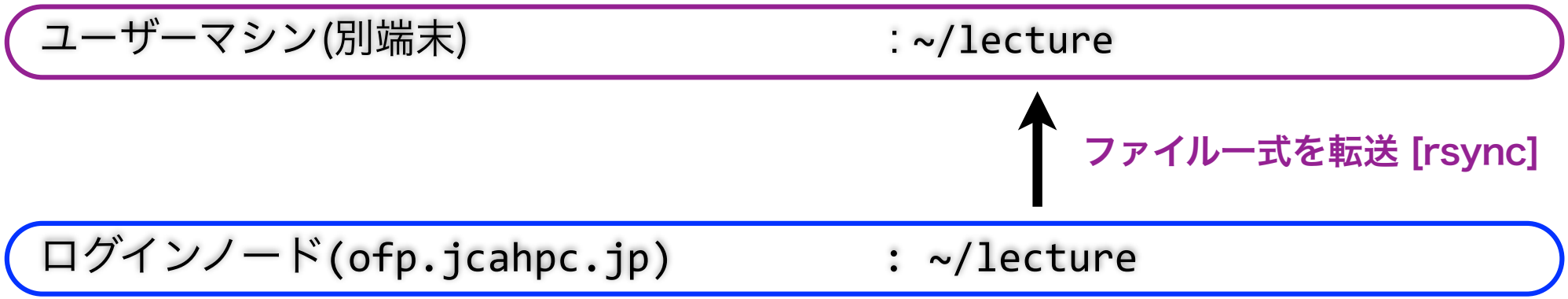
内部界面(フェース)数

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



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
mkdir lecture
rsync -auv txxxxx@ofp.jcahpc.jp:lecture/ ~/lecture/
```

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

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),

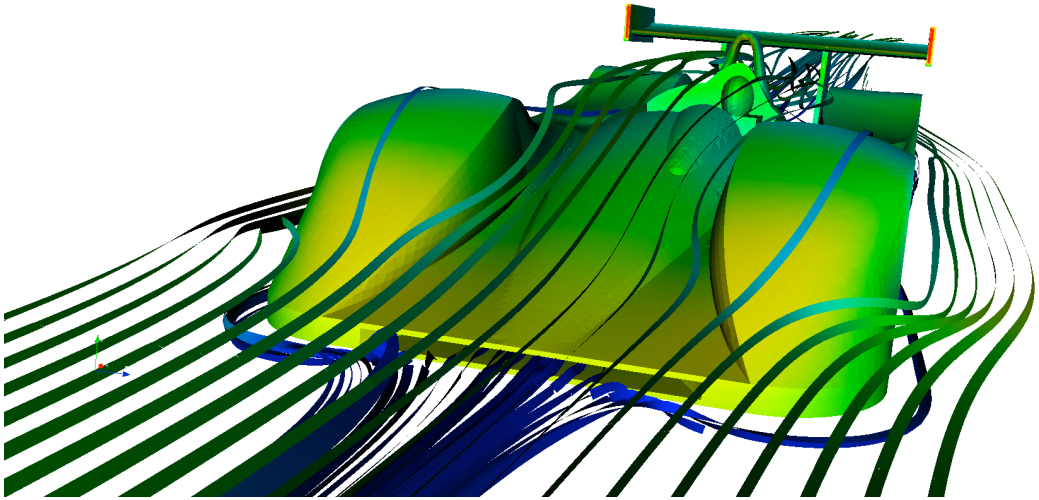
u=update(新規・更新されたもののみ転送)，v=verbose(転送情報を表示)

キャビティケースに移動(別端末で実行)

```
cd ~/lecture/cavity/
```

ParaViewとは?

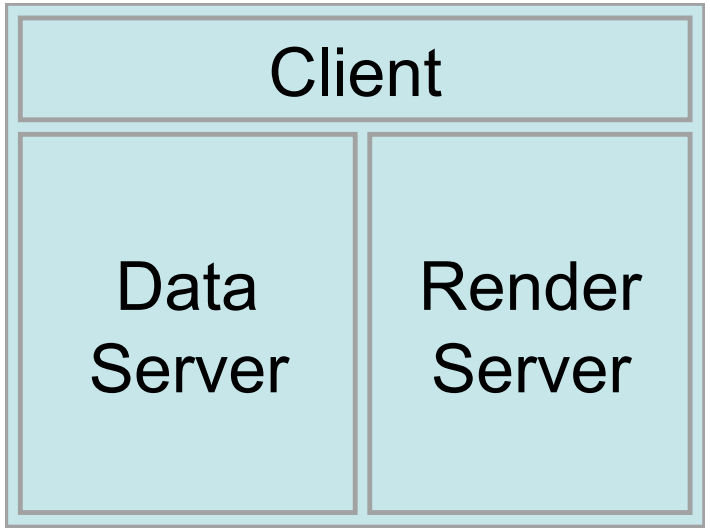
- オープンソース、スケーラブル、かつマルチプラットフォームな可視化アプリケーション
- 大容量データセットを処理するための分散型計算手法のサポート
- オープン、柔軟かつ直感的なユーザインターフェイス
- オープンな規格に基づいた拡張性の高いモジュール化構造
- 柔軟な3条項BSDライセンス
- 有償の保守およびサポート

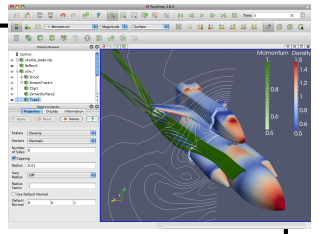


ル・マンのレースカー周りの気流
(ブラジル リオ・デ・ジャネイロ NACAD/COPPE/UFRJ Renato N. Elias)

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

ParaViewの3層構造





- クライアント
  - ✓ 可視化の作成を担当(GUI)
  - ✓ オブジェクトの作成、実行、削除を制御するが実際のデータは全く保持しない
  - ✓ 常にシリアル実行

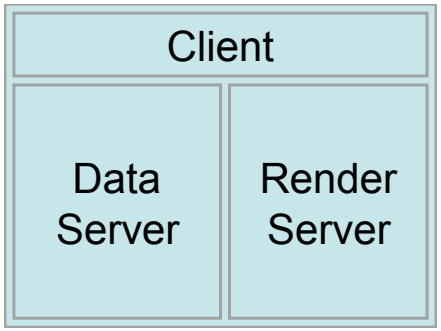
- データ・サーバ
  - ✓ データの読込み、フィルタリング、書出しを担当
  - ✓ 全てのパイプライン・オブジェクトはデータ・サーバが保持
  - ✓ 並列実行可能

- レンダー・サーバ
  - ✓ レンダリングを担当を担当
  - ✓ 並列実行可能 (内蔵の並列レンダリング機能が有効化)

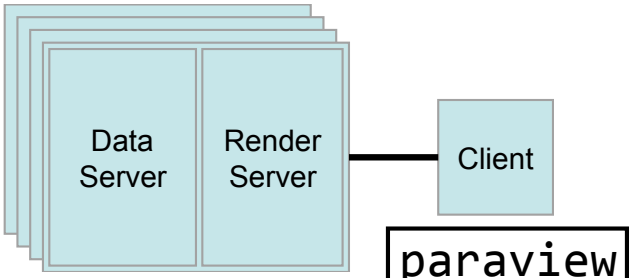
図出典：The ParaView Tutorial for Version 4.4
https://www.paraview.org/Wiki/The_ParaView_Tutorial

ParaViewの3層構造

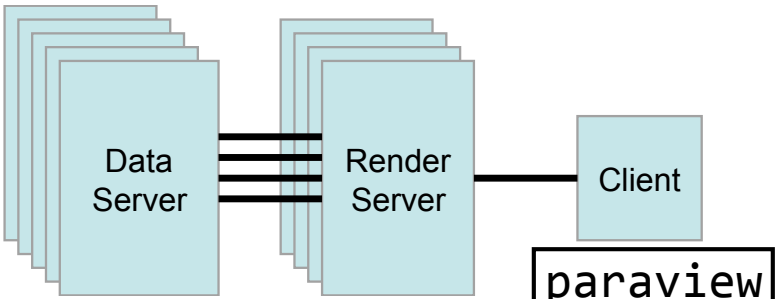
コマンド名



paraview



pvserver　paraview



pvdataserver　pvrenderserver　paraview

スタンドアローン・モード

- 全てが統合してシリアル動作

クライアント–サーバ・モード

- 並列可視化可能
- リモート可視化も可能

クライアント–レンダー・サーバー–データ・サーバ・モード

- サーバ間の通信量が多く，3つが独立動作する利点が少ないので，非推奨

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**

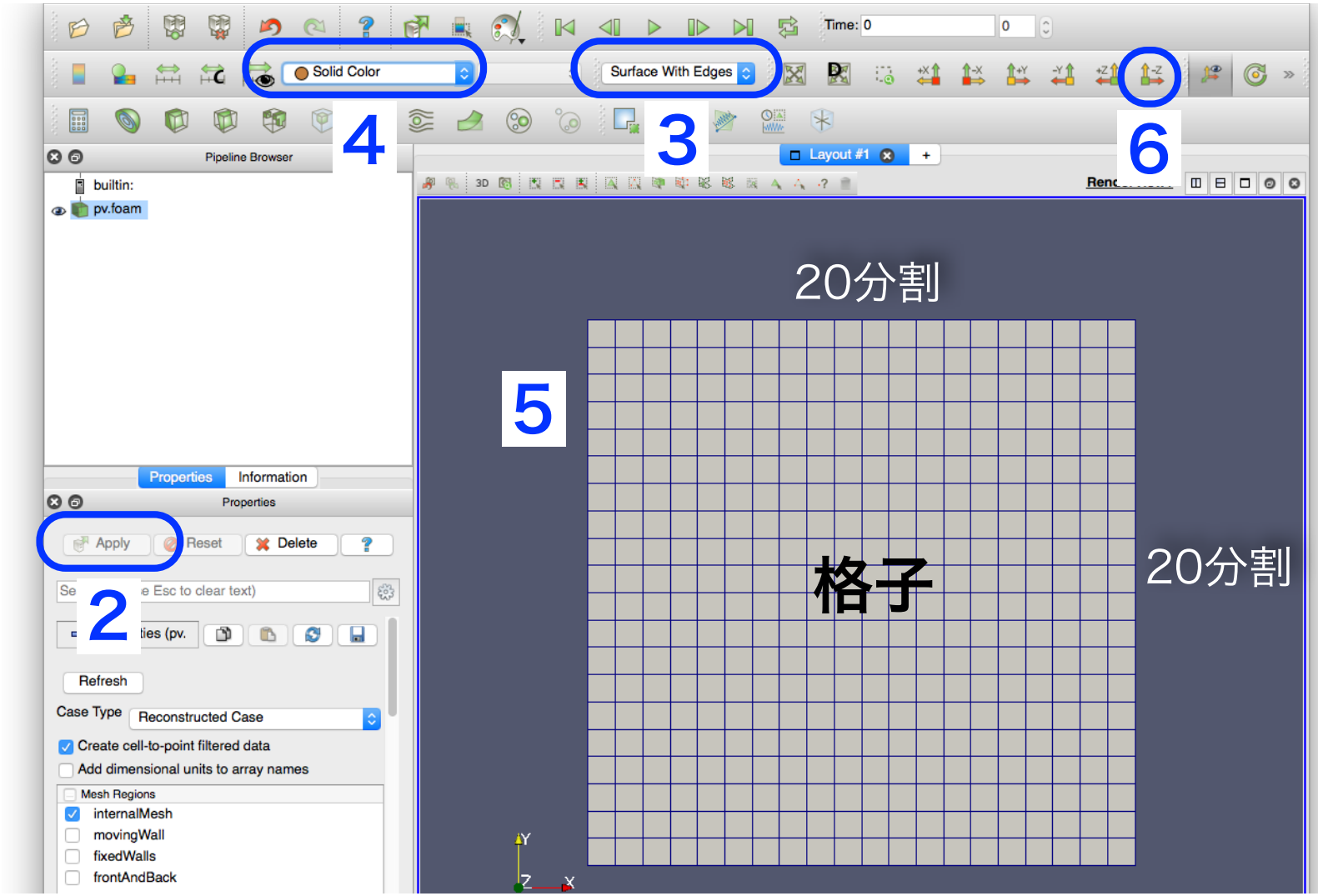
✓ **OpenFOAMデータ読み込み方法**

- **OpenFOAM附属のparaFoamコマンド使用 (非推奨)**
  - 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通常この方法
  - 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**
  - ParaViewを起動するクライアント側にOpenFOAMの環境は不要
  - 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
    - **スタンドアローン・モード：**スパコン側の格子・解析結果をクライアント側に転送して，クライアント側のParaViewで可視化 **(今回はこの方法)**
    - **クライアント–サーバ・モード：**スパコン側で起動したpvserverと，クライアント側で起動したParaViewが通信して可視化

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)

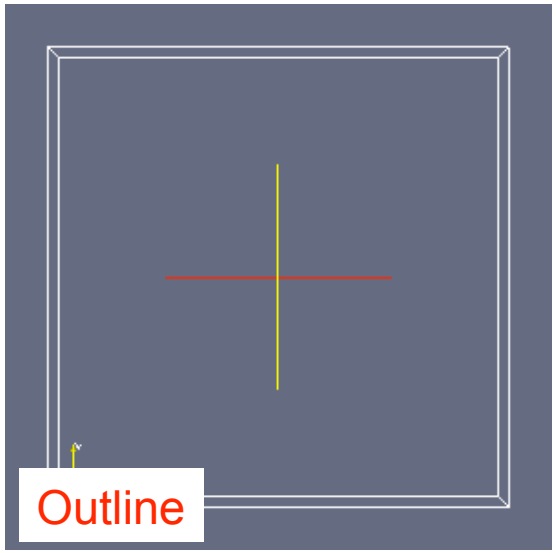
```
touch pv.foam
```

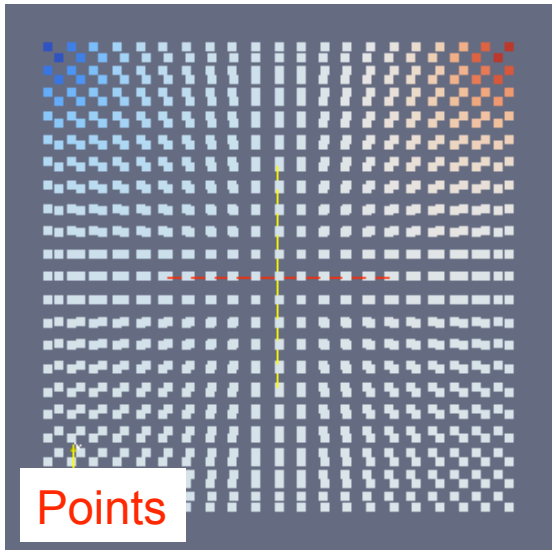


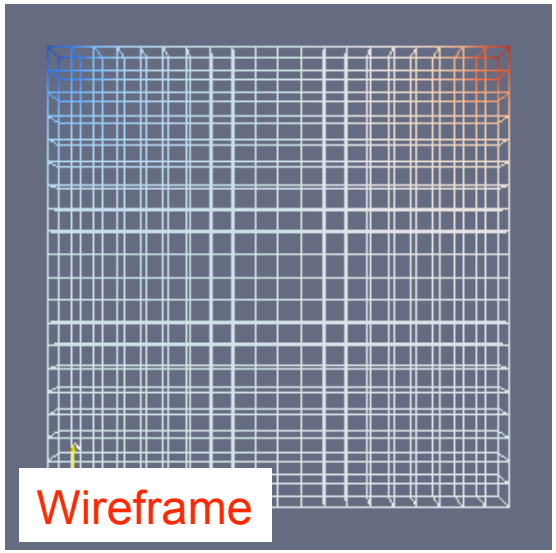
ParaViewで以下の操作を行う

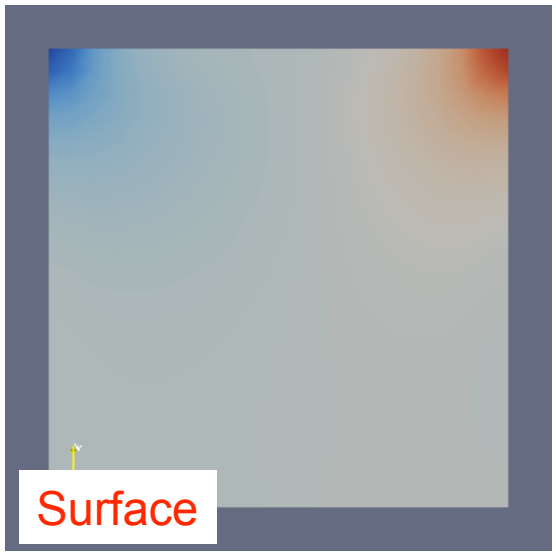
1. Fileメニュー/Open/cavityのディレクトリのpv.foamを開く
2. Apply
3. Representation/Surface With Edges 選択
4. Coloring/Solid Color 選択
5. マウスで動す

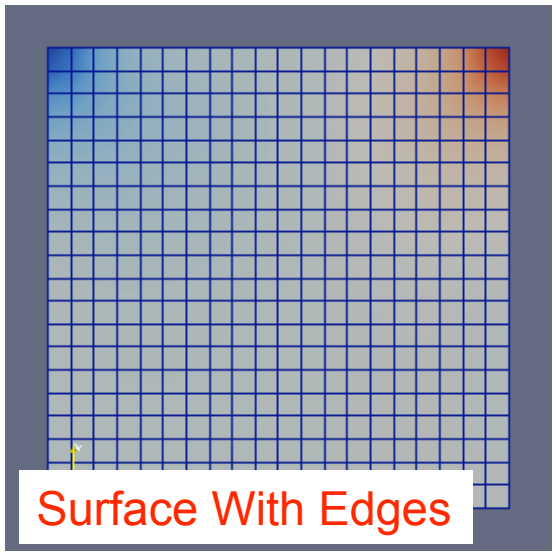
ParaViewの表示方法(Representation)

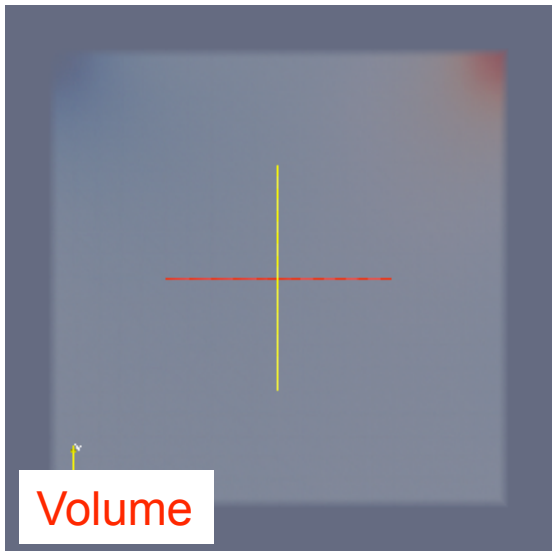






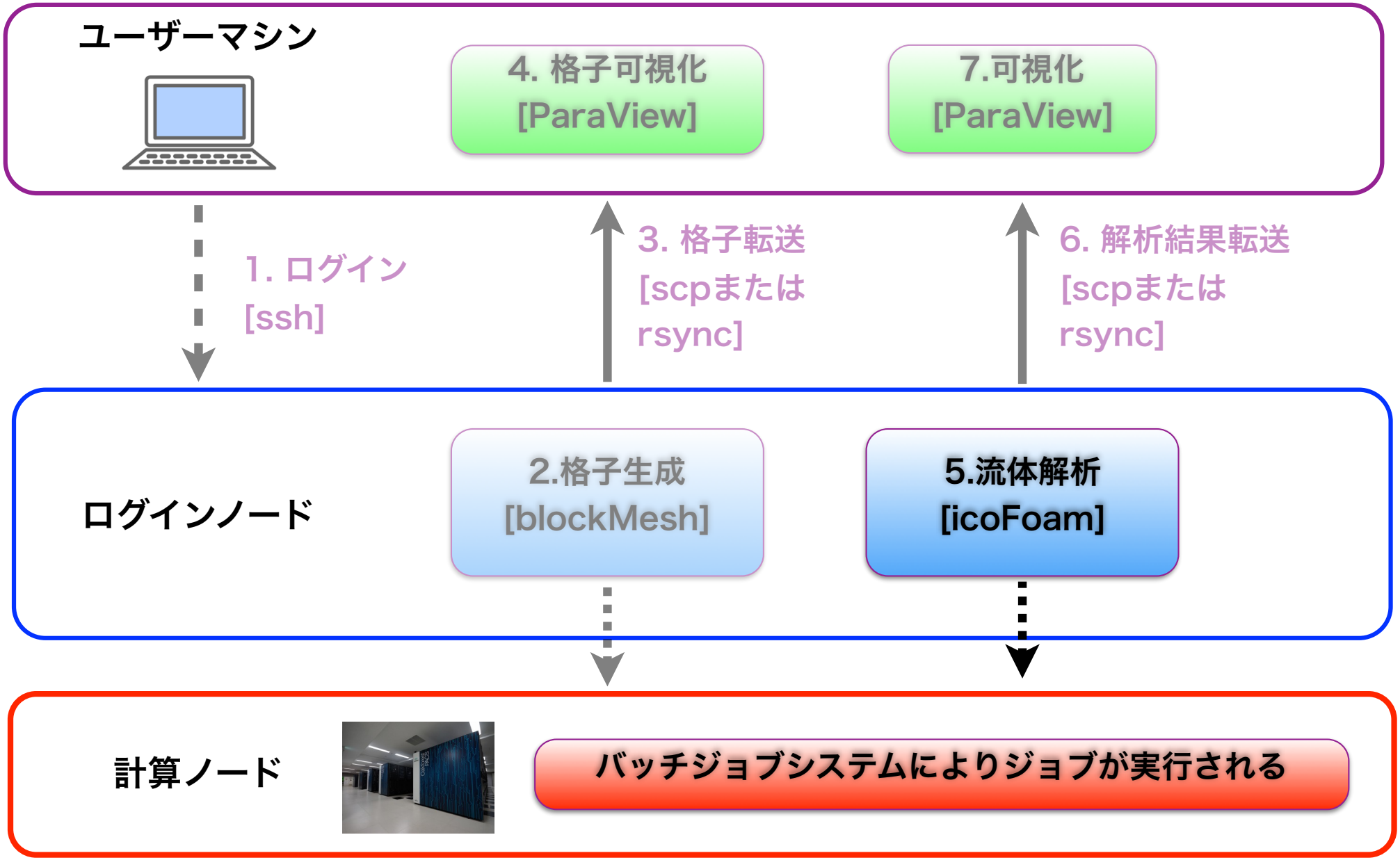






(図引用元: 大嶋 拓也 (新潟大学)「キャビティ流れの解析、paraFoamの実習」 第一回OpenFOAM講習会)

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)
```

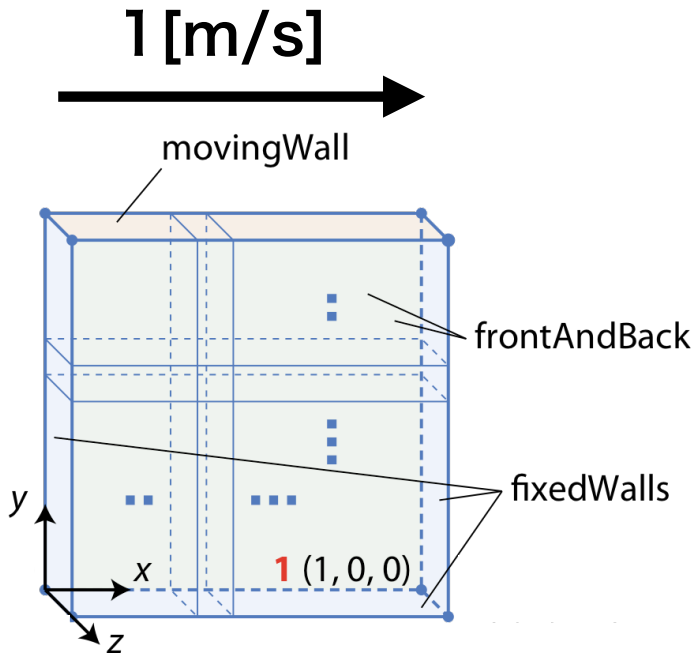
各演算では左辺・右辺での単位一致の検査がなされる

→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
```

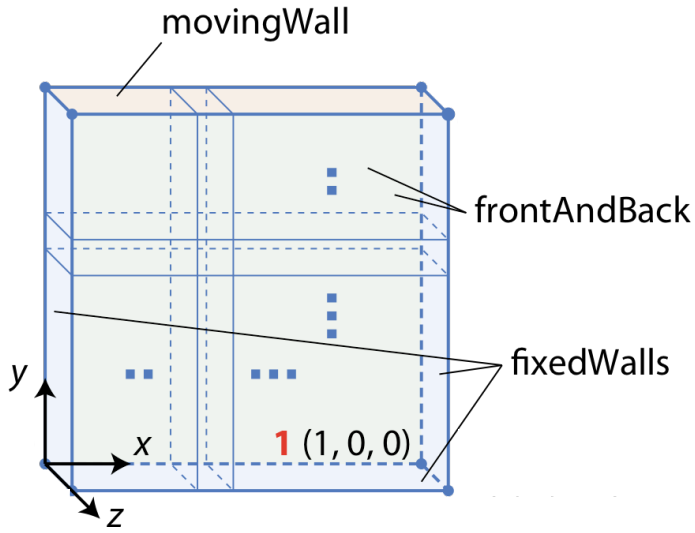
0/p

```
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              0.01; //動粘性係数nu
//変数名          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
```

```
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 seq.sh
```

seq.sh（逐次実行ジョブ用スクリプト）

```
#!/bin/bash
#PJM -L rscgrp=lecture-flat
#PJM -L node=1
#PJM -L elapse=0:15:00
#PJM -g gt00
#PJM -S

module purge
module load intel/2018.1.163
export HOME=/work/gt00/$USER
. /work/gt00/share/OpenFOAM/OpenFOAM-v1712/etc/bashrc \
WM_COMPILER=Icc18_1_163 WM_MPLIB=INTELMPI2018_1_163

env
icoFoam >& $PJM_JOBNAME.l$PJM_JOBID
```

icoFoamを実行(blockMesh.shとの変更点はここのみ)

ジョブの投入

```
pjsub seq.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                    :
├── seq.sh.e1234567        ジョブの標準エラーファイル
├── seq.sh.i1234567        ジョブの統計情報ファイル
├── seq.sh.l1234567        ソルバのログ
└── seq.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 seq.sh.l*
```

```
Build   : v1712
Arch    : “LSB;label=32;scalar=64”
Exec    : icoFoam
Date    : Jan 1 1970
Time    : 00:00:00
Host    : “xxxx"
PID     : xxxxx
I/O     : uncollated
Case    : /xx/lecture/cavity
nProcs  : 1
```

ビルドバージョン(実行環境依存)
バイト順;ラベルbit数;実数変数bit数
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)
領域分割ファイルを1ファイルにまとめない
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)

流体解析のログ(続き)

seq.sh.1ジョブID

```
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)

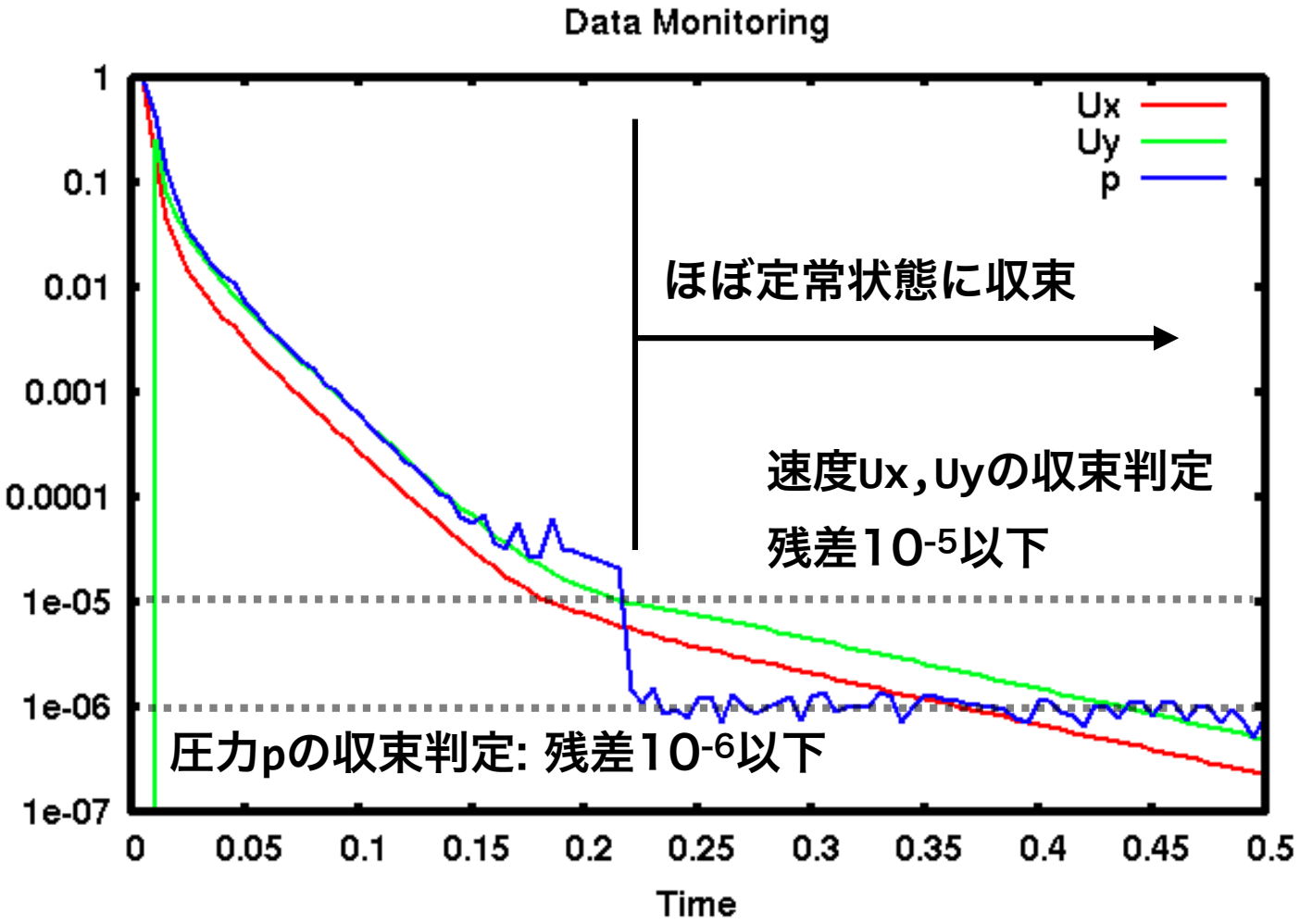
seq.sh.1ジョブID

```
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

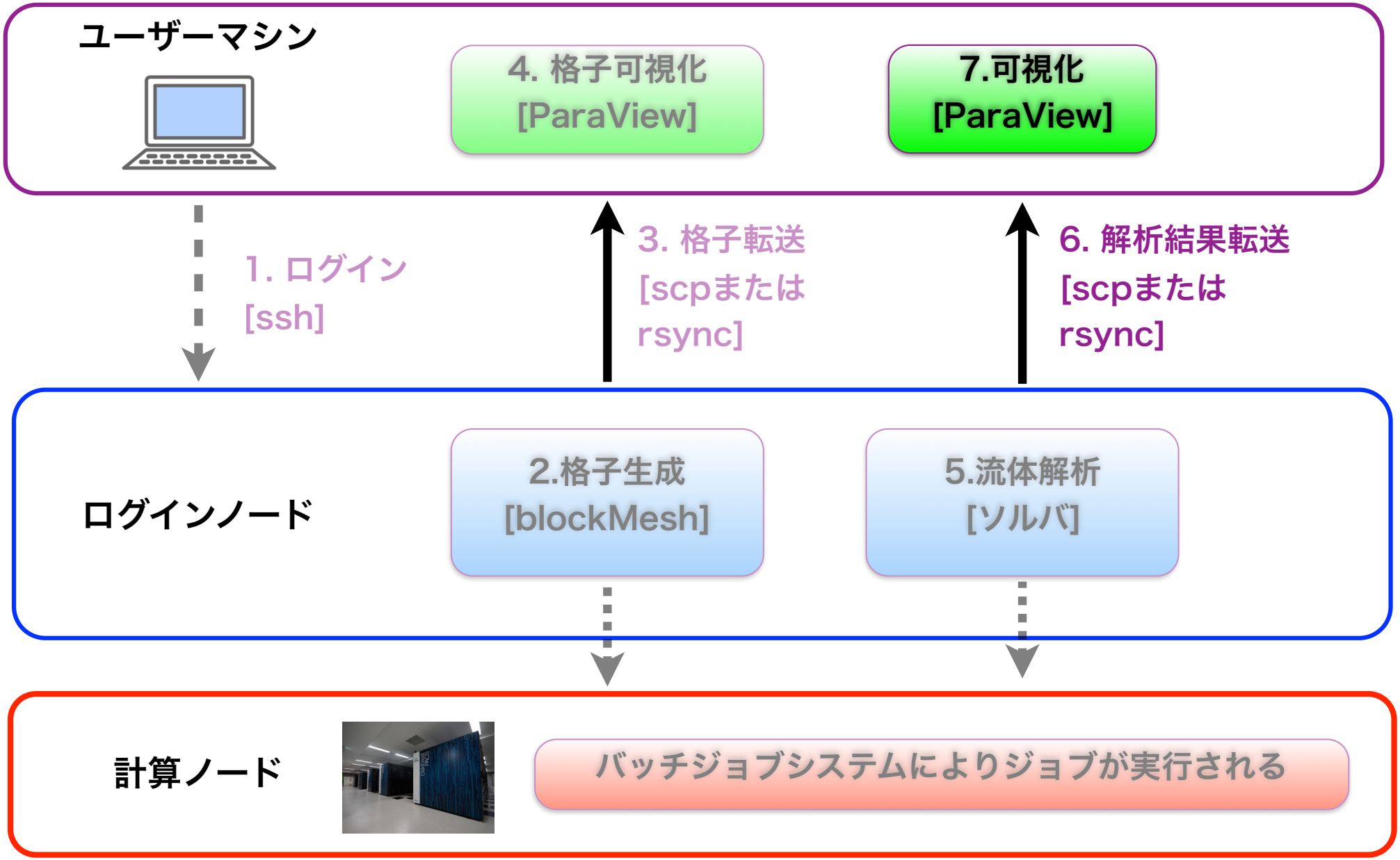
初期残差のプロット (-r 1で1秒間隔で更新)

```
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



解析結果の転送

解析結果の転送(別端末で実行)

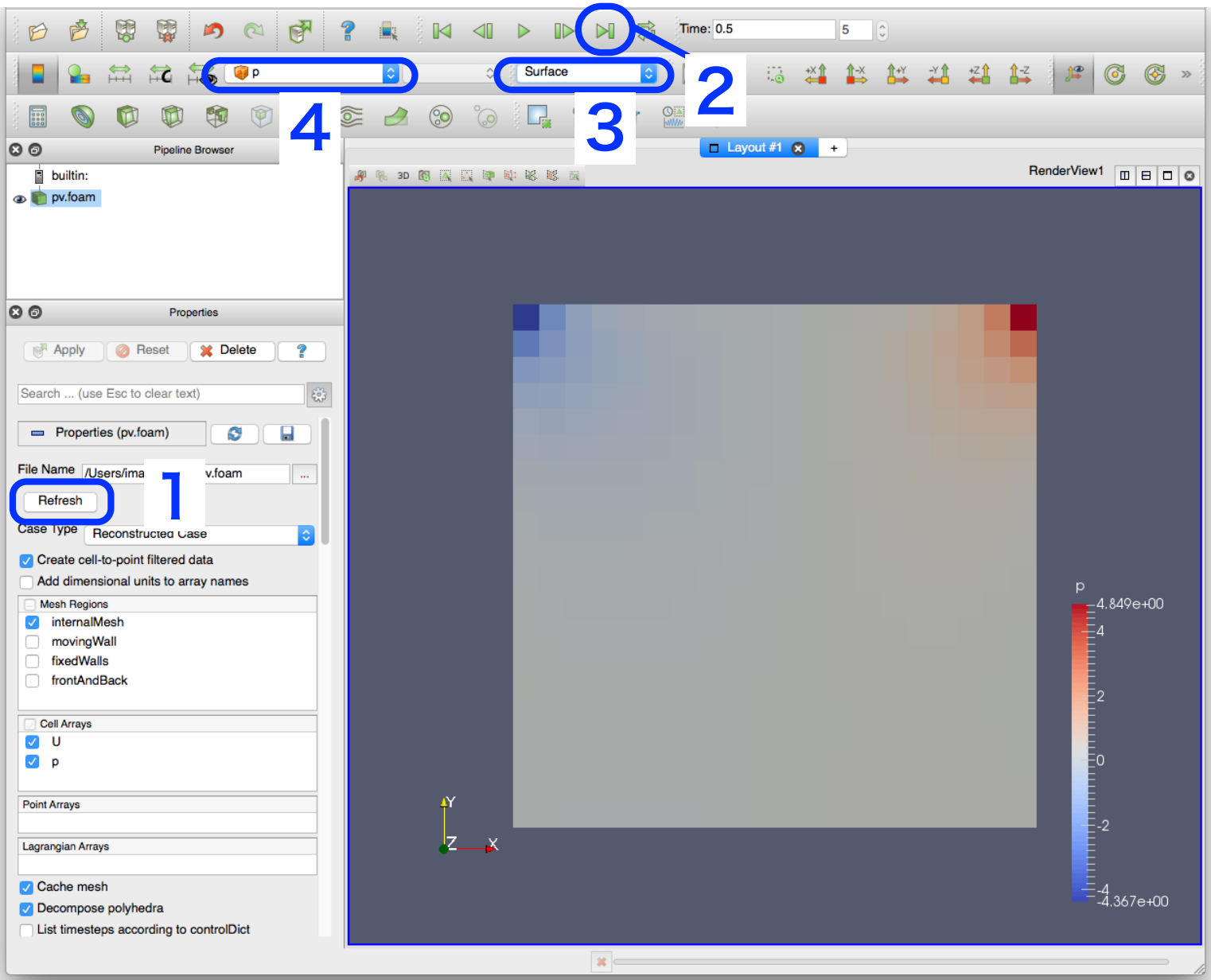
↑(カーソル上)を押して前のコマンドを呼び出す

```
rsync -auv txxxxx@ofp.jcahpc.jp:lecture/ ~/lecture/
```

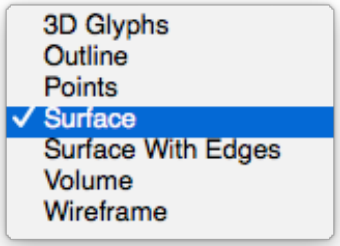
```
receiving file list ... done
cavity/
cavity/seq.sh.e1234567
cavity/seq.sh.i1234567
cavity/seq.sh.l1234567
cavity/seq.sh.o1234567
cavity/0.1/
cavity/0.1/U
:
```

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)

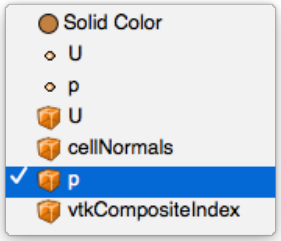
ParaViewによる圧力の可視化



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

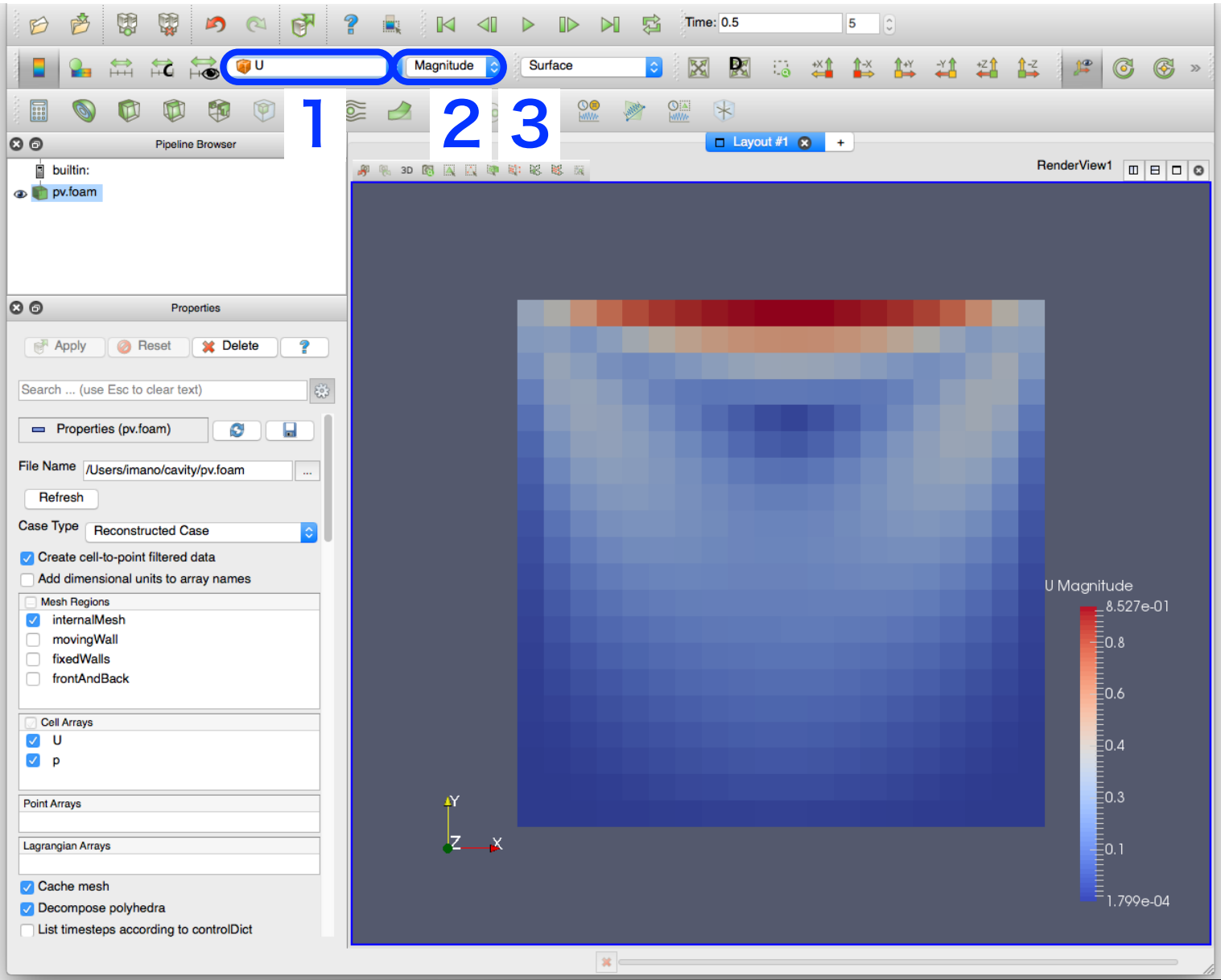


4. Coloring/□p

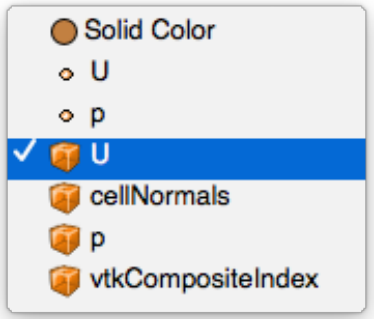


(□は格子の値そのままで補間無し、○は補間有り=スムージング)

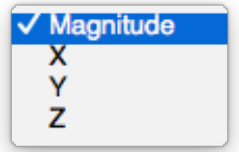
ParaViewによる風速の可視化



1. Coloring/□U

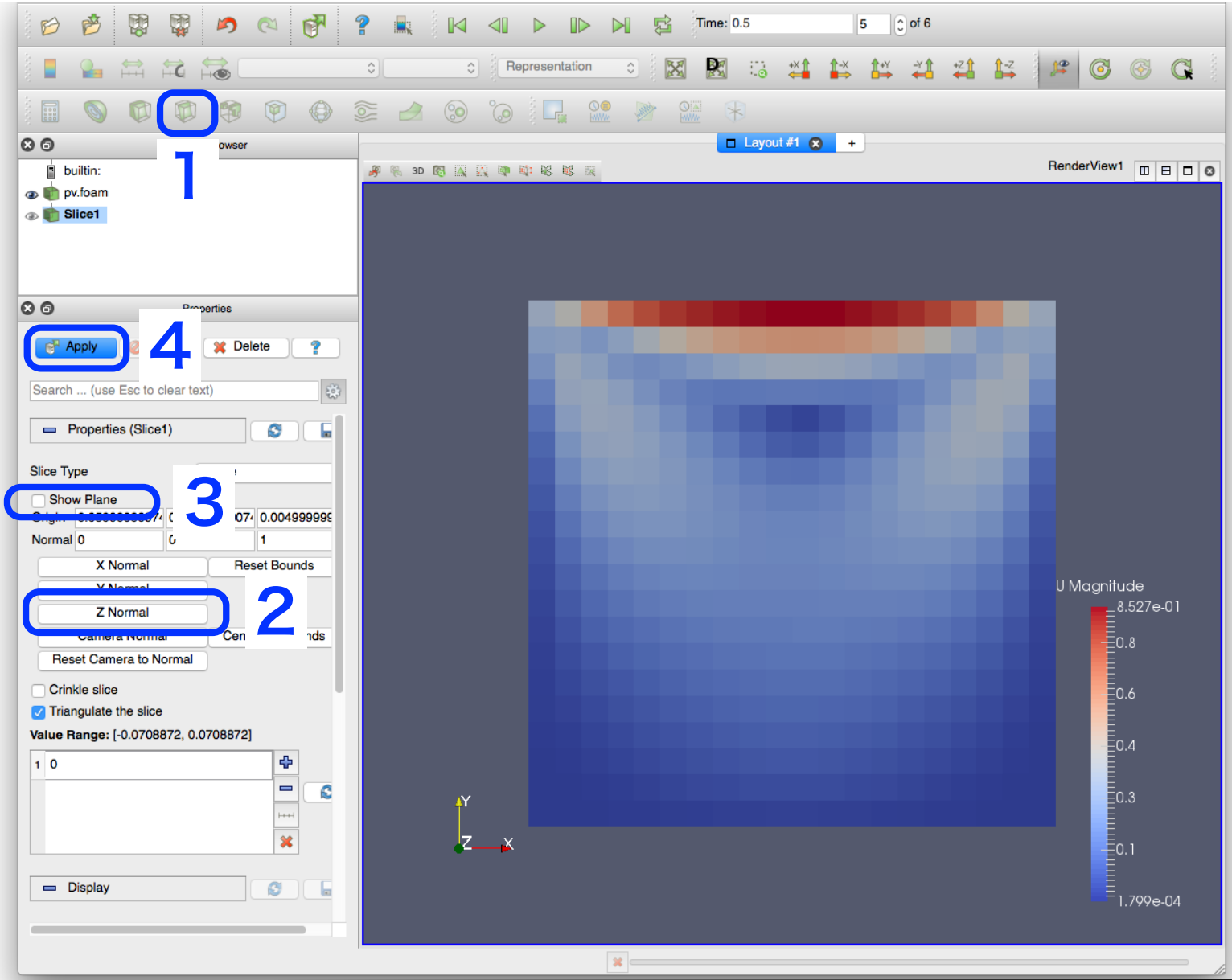


2. Magnitudeを X，Y，Zに変更することで，各風速成分が可視化できる．



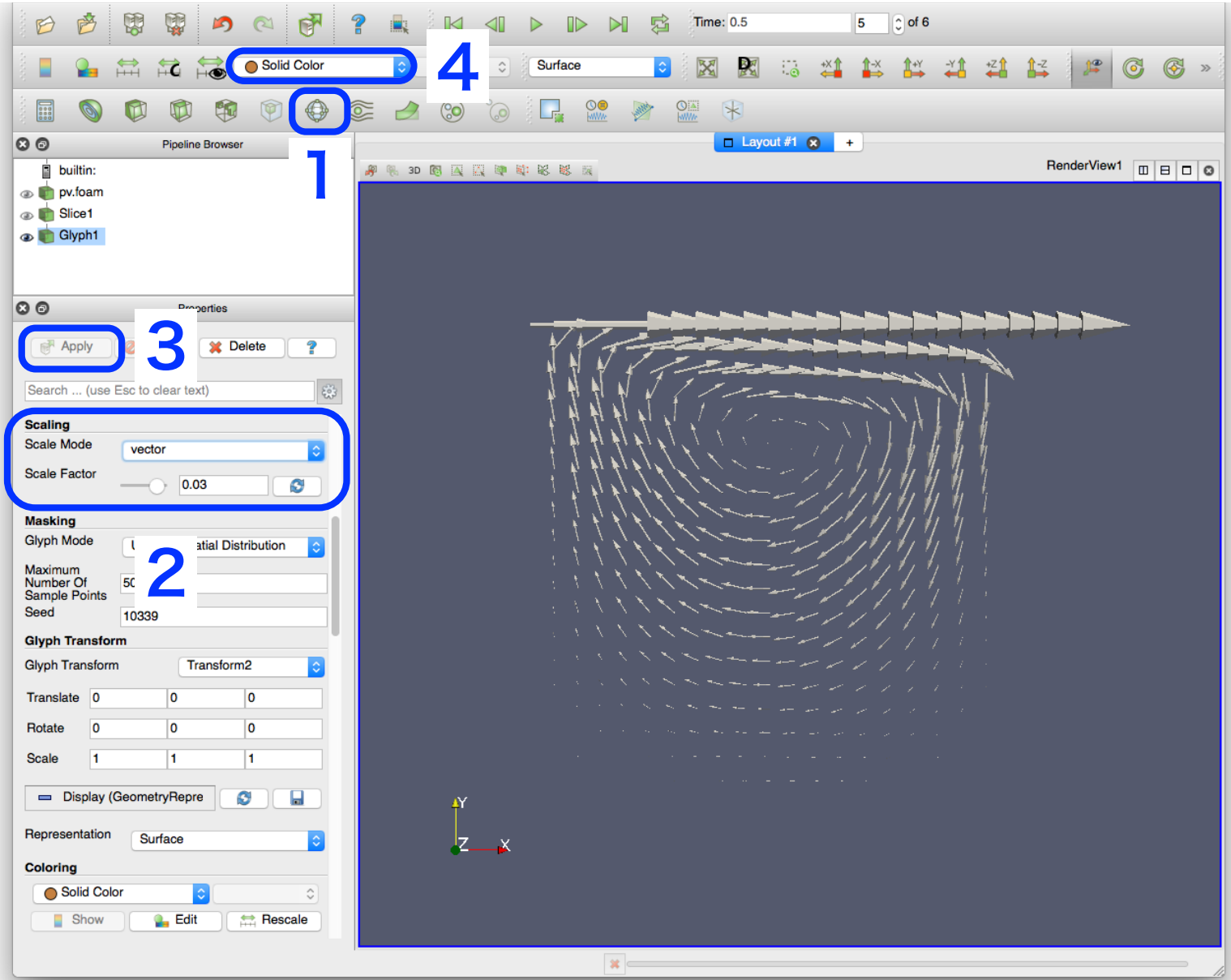
3. Magnitudeに戻す．

ParaViewによる風速ベクトルの可視化



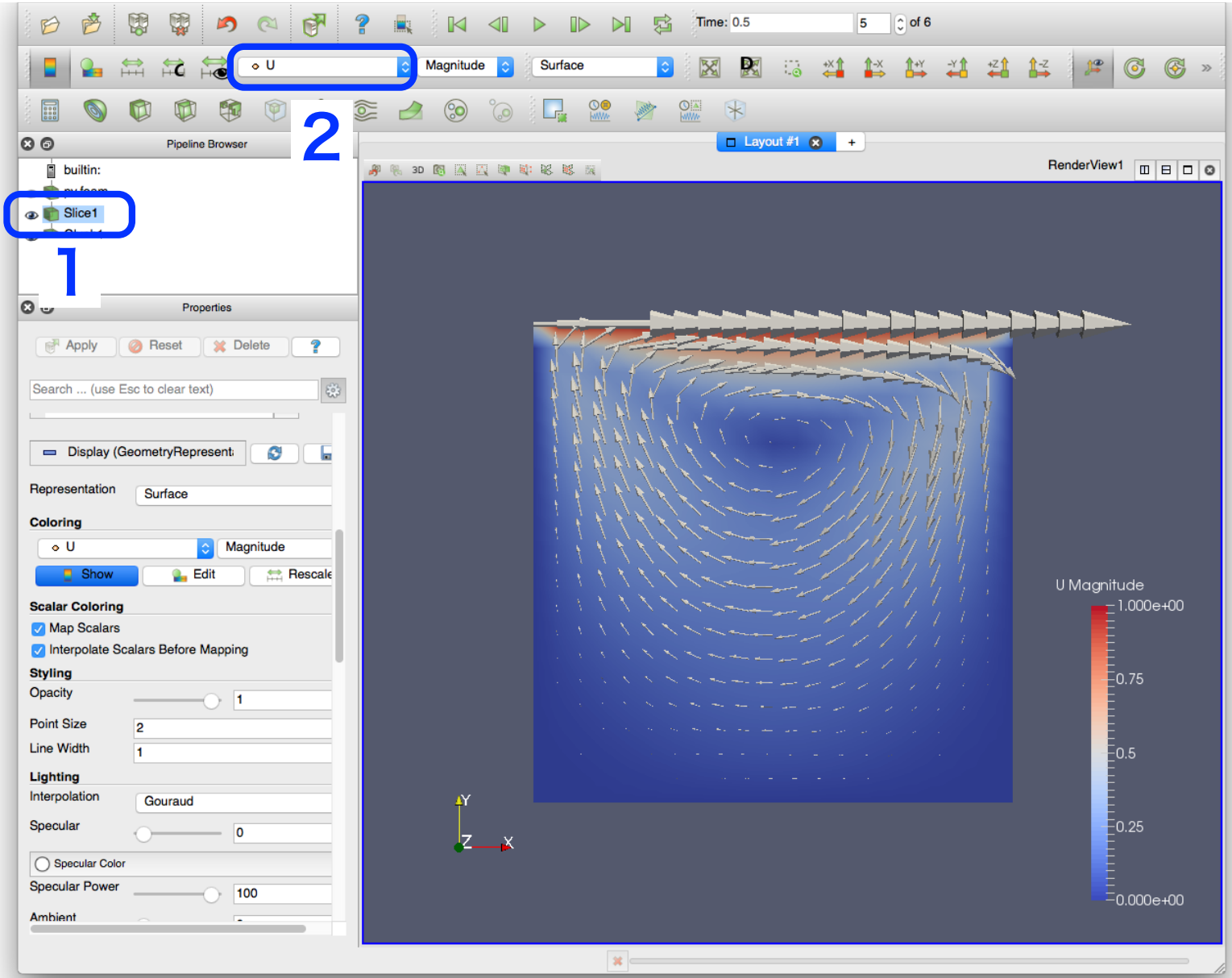
1. Sliceフィルタ
2. Z Normal
3. Show Planeを非選択
4. Apply

ParaViewによる風速ベクトルの可視化(続き)



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(続き)

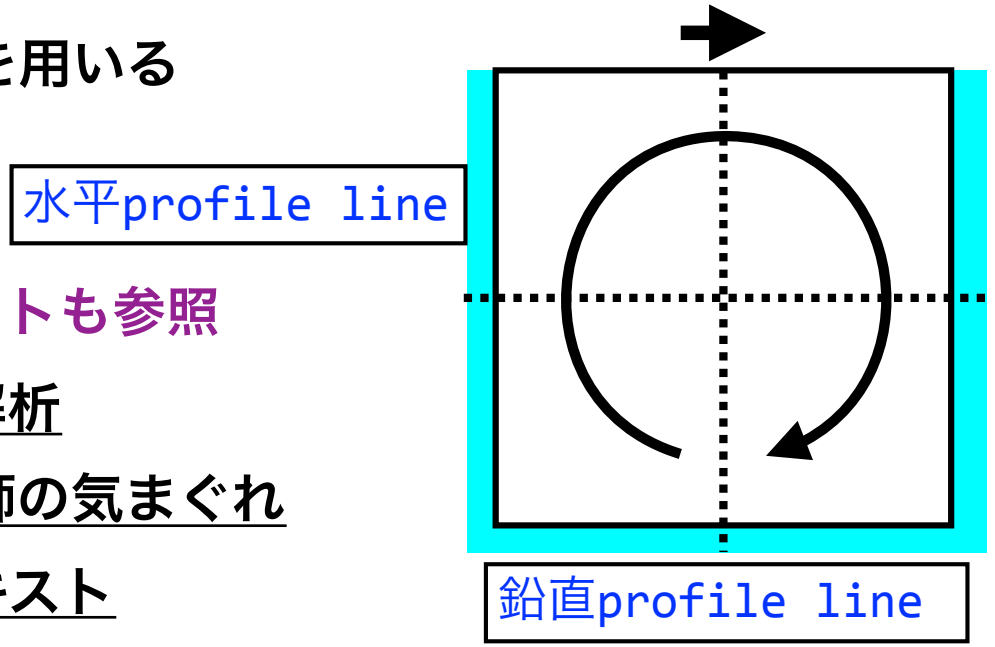


1. Sliceフィルタがもし非表示になっていた場合(注)は表示(目のマークをチェック)
2. Coloring/・Uを選択
3. Quitメニュー/Quit ParaViewでParaView終了

(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 ~/lecture
mkdir cavity2
foamCopySettings cavity cavity2
cd cavity2
foamCleanTutorials
ls -l
rm *.sh.*
ls -l
```

新ケースのディレクトリ作成(名前は任意)

設定をコピー

postProcessingなどの実行結果を消去

バッチジョブの出力ファイルを消去

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
```

```
scale 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，定常になるまでに多くの積分時間が必要．もしくは，そもそも定常にならない．

ジョブの確認・残差モニター・強制終了

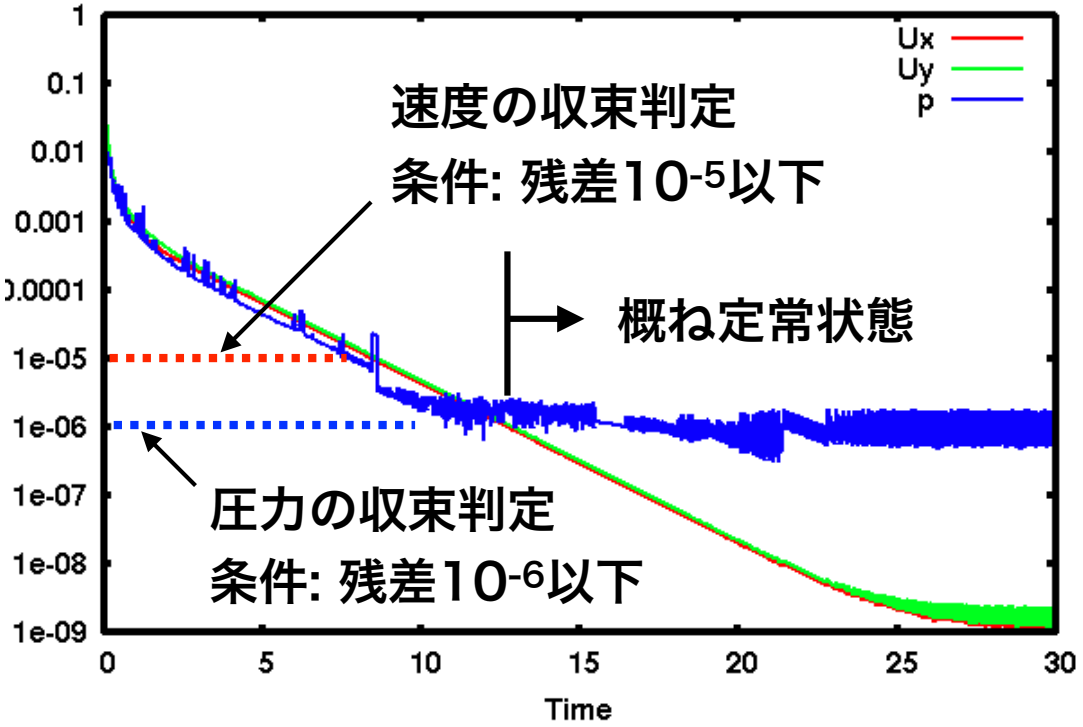
ジョブ状態確認

```
pjsub blockMesh.sh
pjstat
pjsub seq.sh
pjstat
```

ジョブの終了を確認してから次に進む．tail -f *.sh.l*でログ追跡も有用

seq.shが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 &
```



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
```

```
libs            ("libsampling.so");
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
);
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

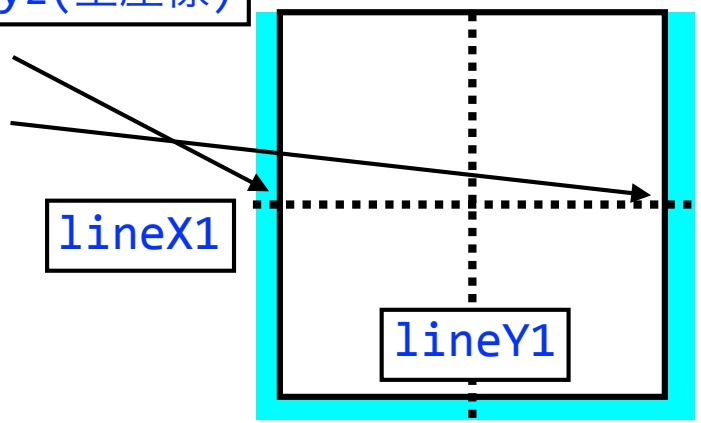
出力する座標軸，x/y/z/xyz(全座標)

サンプリング開始点

サンプリング終了点

lineX1と同様なので略

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

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
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)

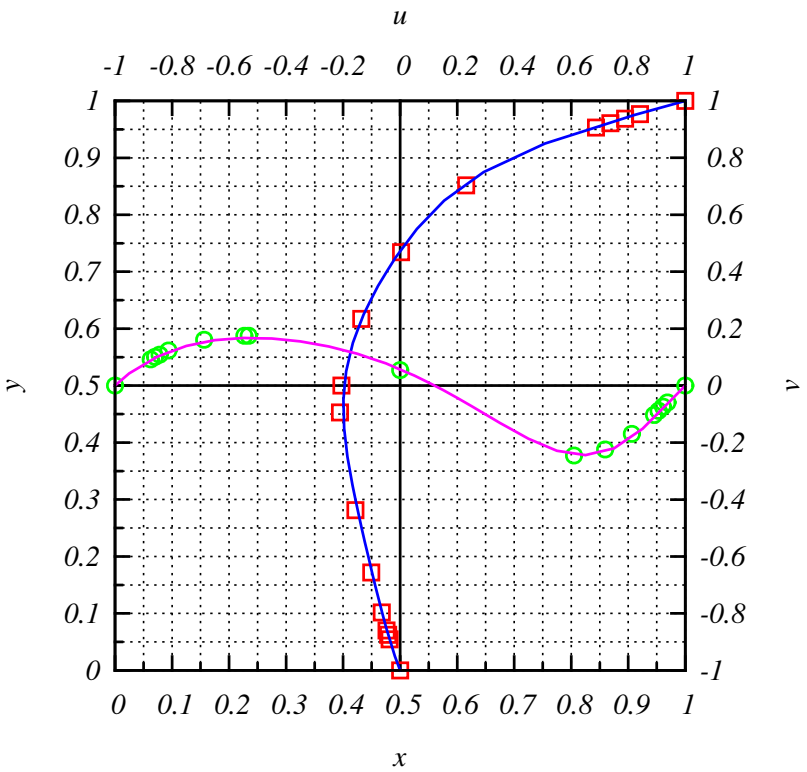
gnuplot実行

```
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

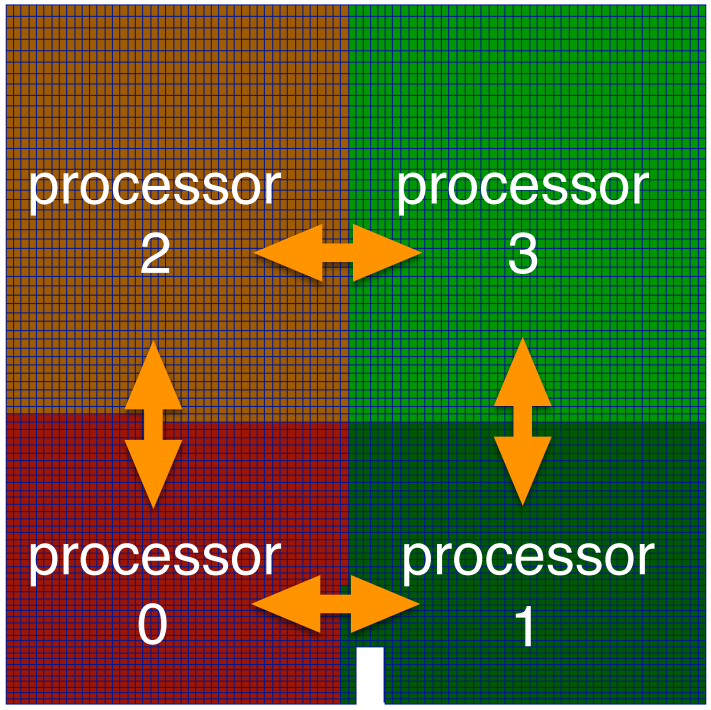
1. 格子生成
2. 領域分割 (decomposePar)
3. MPI並列でソルバを実行

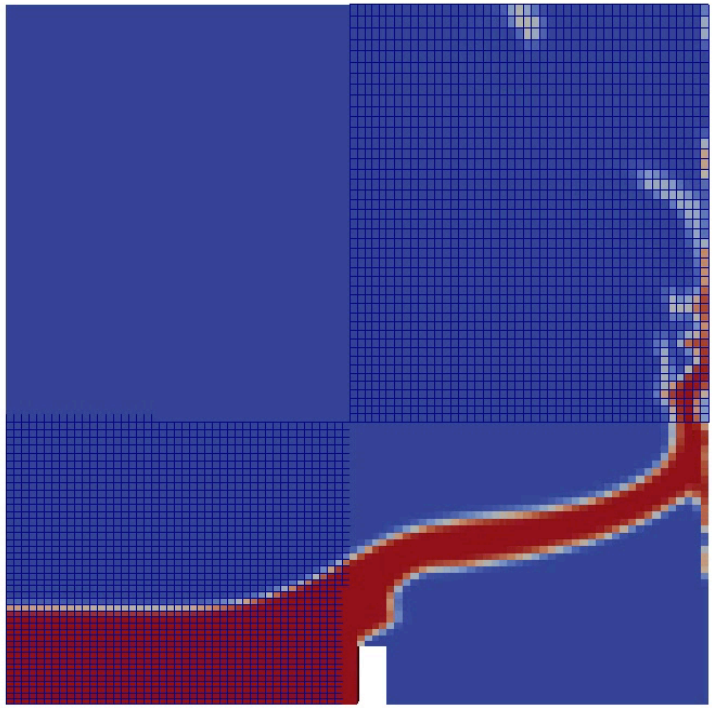
(MPI+OpenMPのハイブリット並列は標準では未実装．研究例有り)

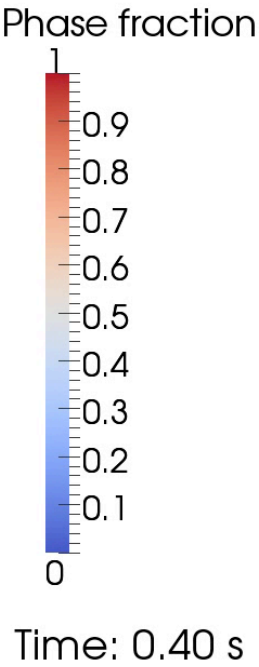
4. 領域毎の解析結果を再構築 (reconstructPar)

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





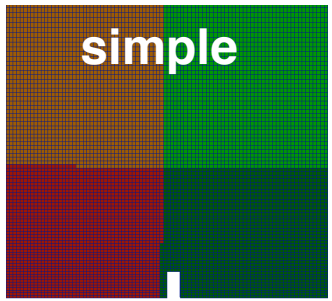


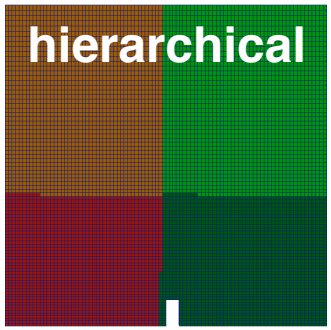
damBreakFineチュートリアル

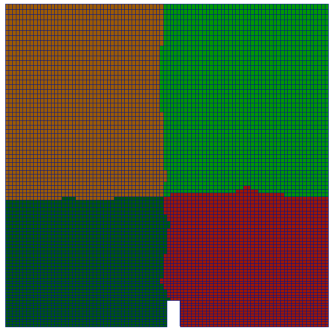
領域分割の設定

system/decomposeParDict

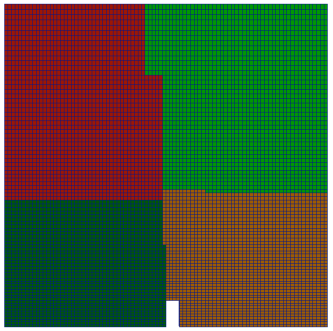
```
numberOfSubdomains 4;        //領域分割数

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



sctoch

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

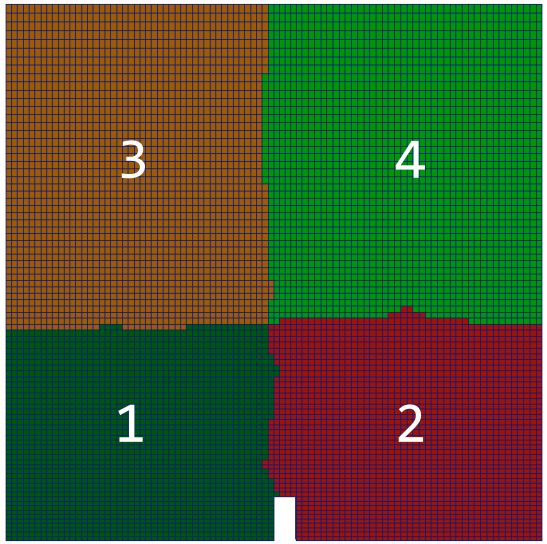
manual：格子を割り充てるプロセッサを手動で指定

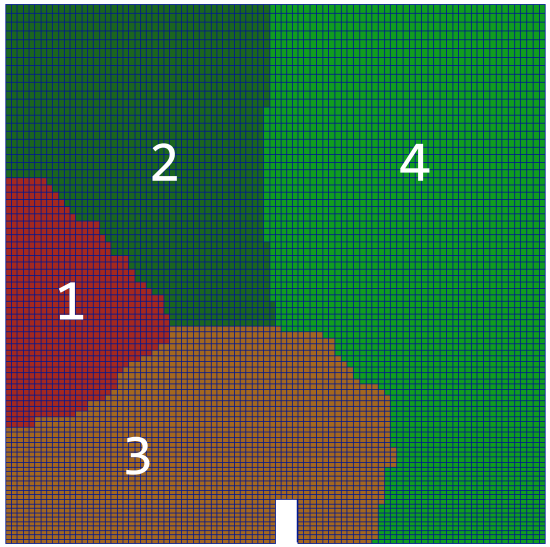
重み付き領域分割例

重み無し

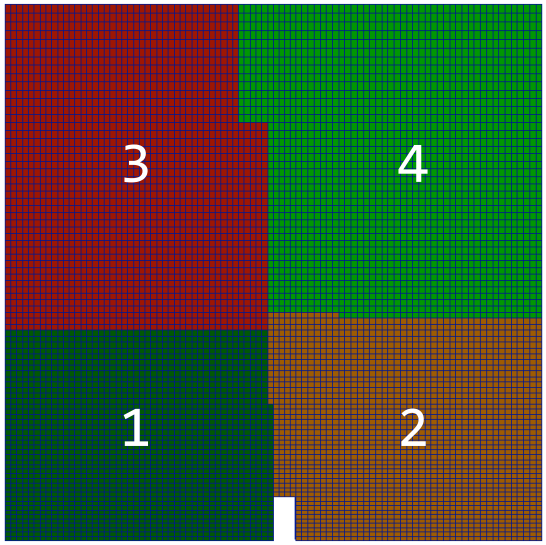
重み付き

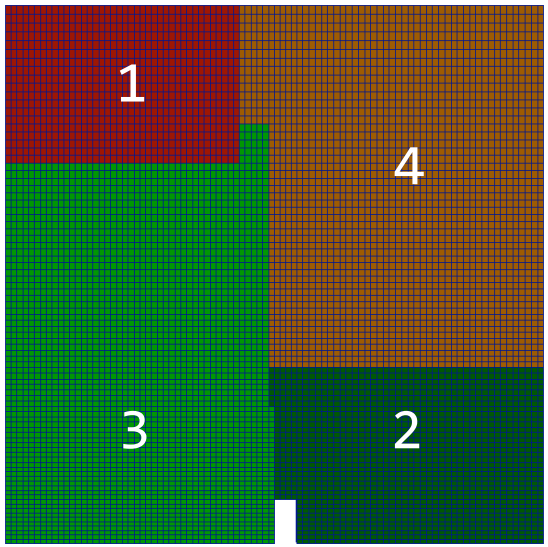
metis





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

領域分割用ジョブスクリプト

decomposePar.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
module load intel/2018.1.163
export HOME=/work/gt00/$USER
. /work/gt00/share/OpenFOAM/OpenFOAM-v1712/etc/bashrc \
WM_COMPILER=Icc2018_1_163 WM_MPLIB=INTELMPI2018_1_163

decomposePar -cellDist >& $PJM_JOBNAME.l$PJM_JOBID
```

領域分割

-cellDistオプションは分割領域可視化用なので，省略可

領域分割ジョブ投入とログの確認

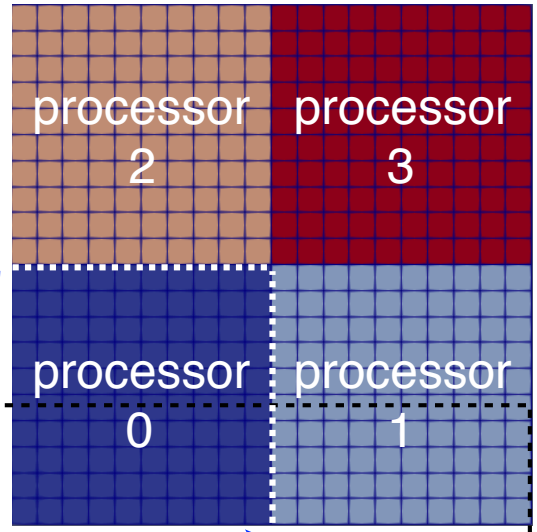
領域分割ジョブの投入

```
pjsub decomposePar.sh
```

領域分割のログの確認(ジョブ完了後)

```
more decomposePar.sh.l*
```

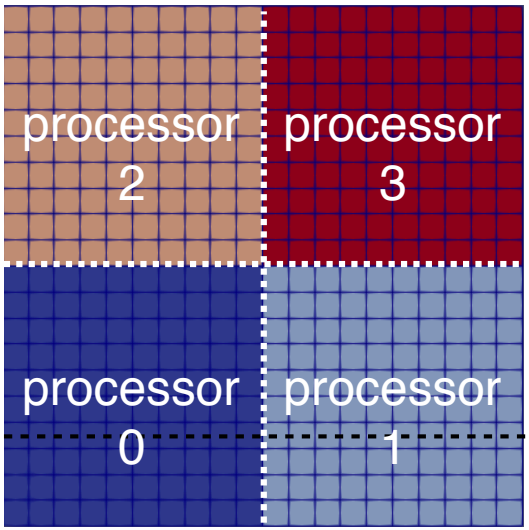
```
Processor 0 #プロセッサ0の担当分割領域
    Number of cells = 100 #格子数
    Number of faces shared with processor 1 = 10
    #プロセッサ番号1の担当分割領域との共有界面数
    Number of faces shared with processor 2 = 10
    #プロセッサ番号2の担当分割領域との共有界面数
    Number of processor patches = 2 #上記の界面を共有するプロセッサ数
    Number of processor faces = 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
```



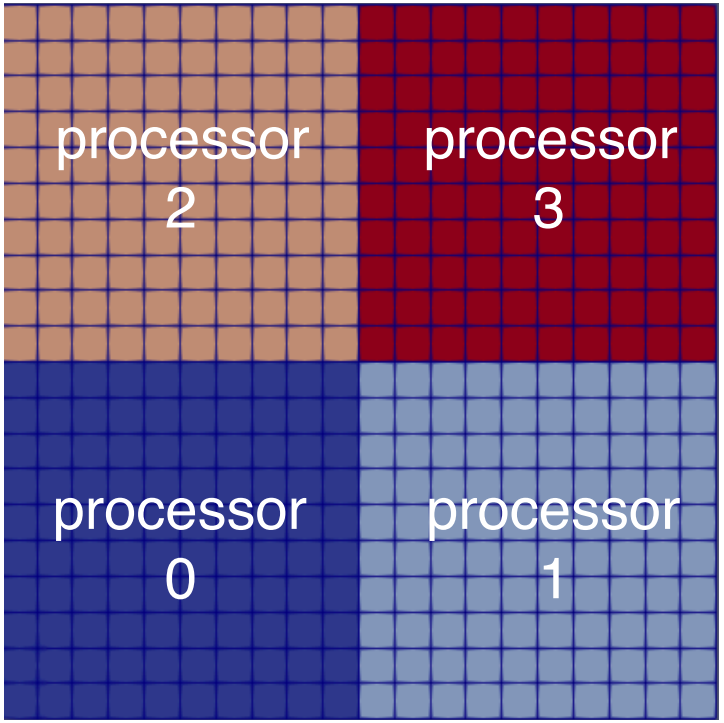
領域分割のログの確認(続き)

decomposePar.sh.1ジョブID

```
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



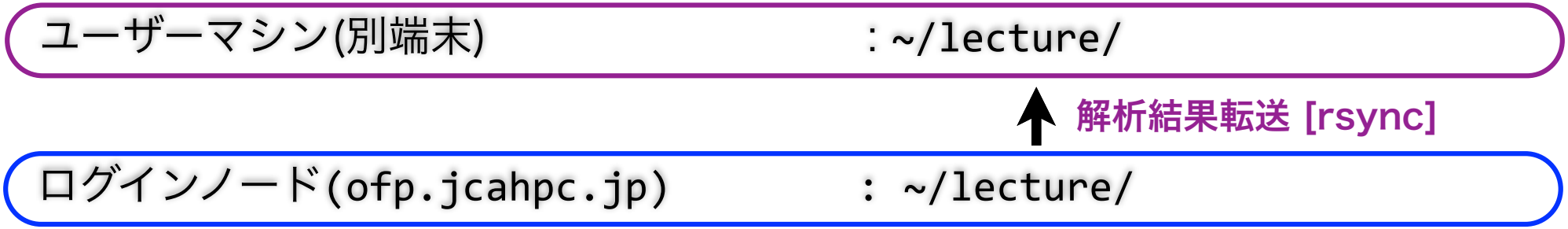

```
constant/
  polyMesh/     #格子データ
0/              #時刻ディレクトリ
  U, p          #場データ
```

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

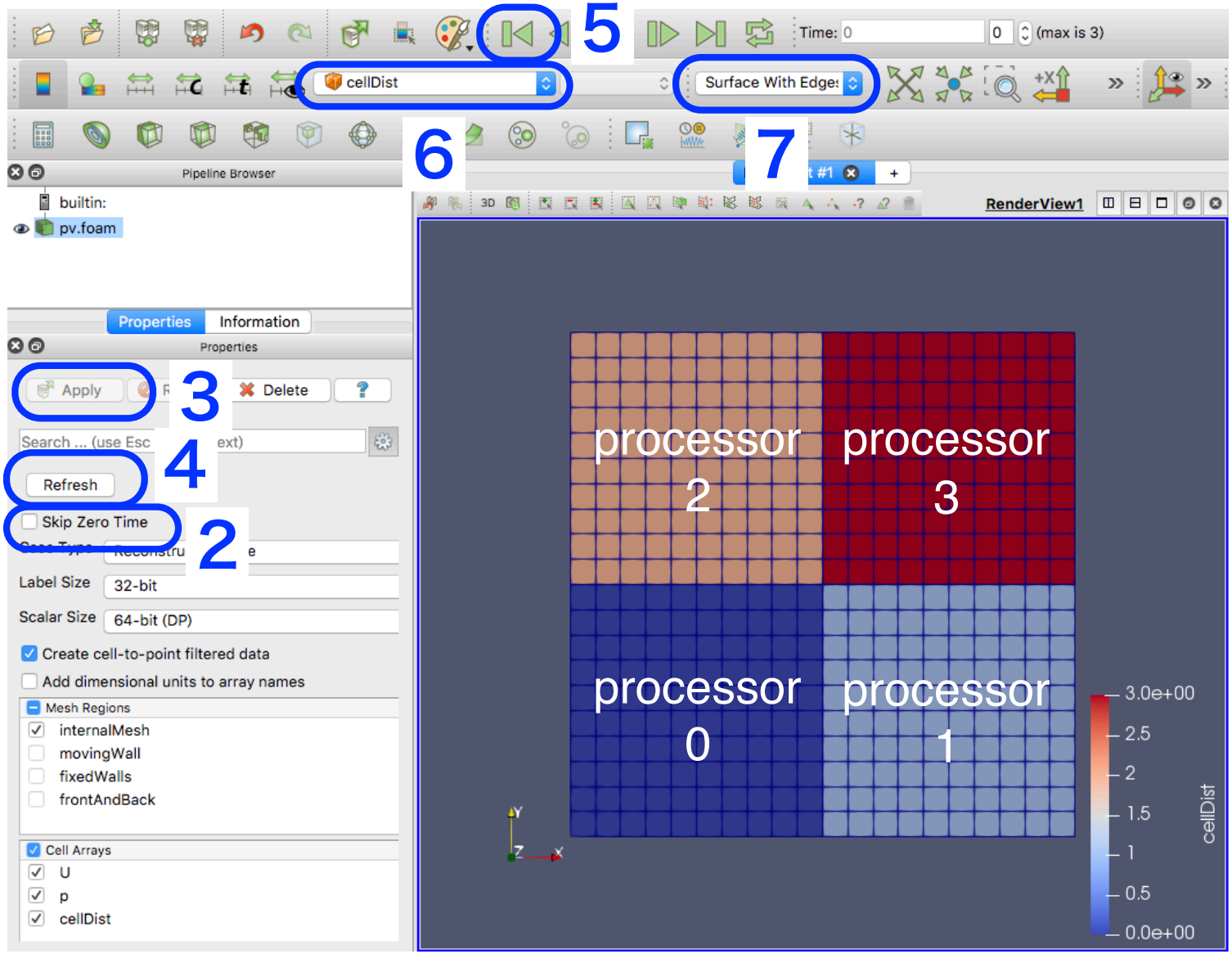


解析結果の転送(別端末で実行)

↑(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:lecture/ ~/lecture/
cd ~/lecture/cavity2
touch pv.foam
```

ParaViewによる分割領域の可視化



1. Fileメニュー/Open/pv.foam
2. Skip Zero Time をアンチェック
3. Apply
4. Refresh
5. First Frame
6. Coloring/□cellDist 選択
7. Representation/Surface With Edges 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

並列計算実行用ジョブスクリプト

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
#PJM -L elapse=0:15:00
#PJM -g gt00
#PJM -S
module purge
module load intel/2018.1.163
export HOME=/work/gt00/$USER
. /work/gt00/share/OpenFOAM/OpenFOAM-v1712/etc/bashrc \
WM_COMPILER=Icc2018_1_163 WM_MPLIB=INTELMPI2018_1_163

. /usr/local/bin/mpi_core_setting.sh
export I_MPI_DEBUG=5

env
mpiexec.hydra -n $PJM_MPI_PROC  \
numactl -p1 \
icoFoam -parallel >& $PJM_JOBNAME.l$PJM_JOBID
```

MPIプロセス数

MPIプロセスのピニング(コアへの固定化)

MPIプロセスのピリング等のデバッグ情報取得

プロセス数=$PJM_MPI_PROCMPI=4で並列実行

MCDRAMをできるだけ使用

並列計算とログ確認

```
pjsub par.sh
more par.sh.l*
```

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
[0] MPI startup(): 1       143923   c3847.ofp  4
[0] MPI startup(): 2       143924   c3847.ofp  6
[0] MPI startup(): 3       143925   c3847.ofp  8
[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_INFO_NUMA_NODE_MAP=hfi1_0:0
[0] MPI startup(): I_MPI_INFO_NUMA_NODE_NUM=2
[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
Hosts  :
(
    (c3847.ofp 4)
```

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
Rank0 → Pin2, Rank1 → Pin4, etc.
以下のI_MPI_PIN_MAPPING 参照

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

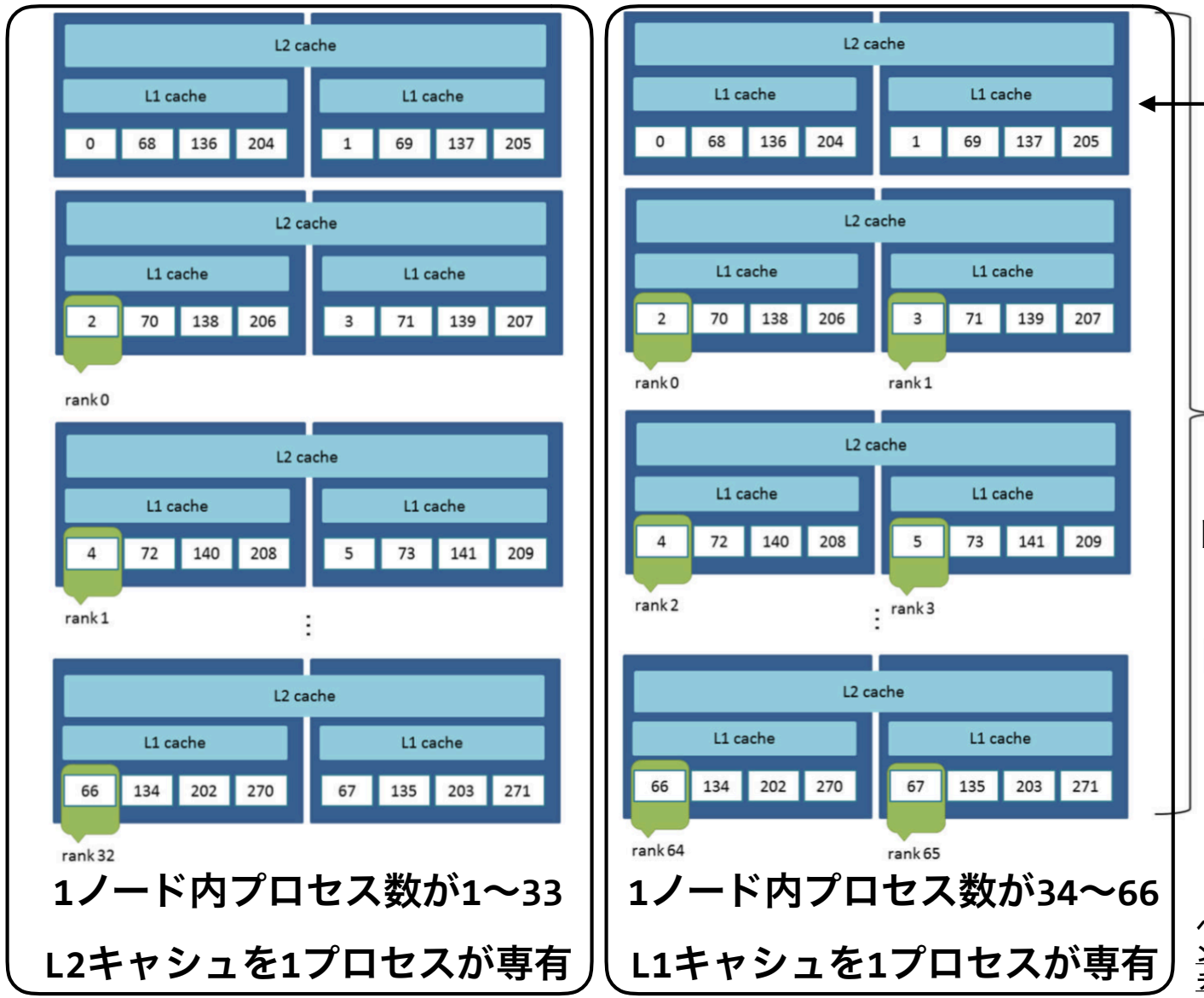
(計算ノードのホスト名 MPIプロセス数)

*) 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



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

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，未使用

```
export
I_MPI_PIN_PROCESSOR_EXCLUDE_LIST
=0,1,68,136,137,204,205 でも可能
```

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が，OFではそれほど変わないので，L1専有のほうが，費用(トークン)対性能が高い事が多い

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

reconstructPar.sh

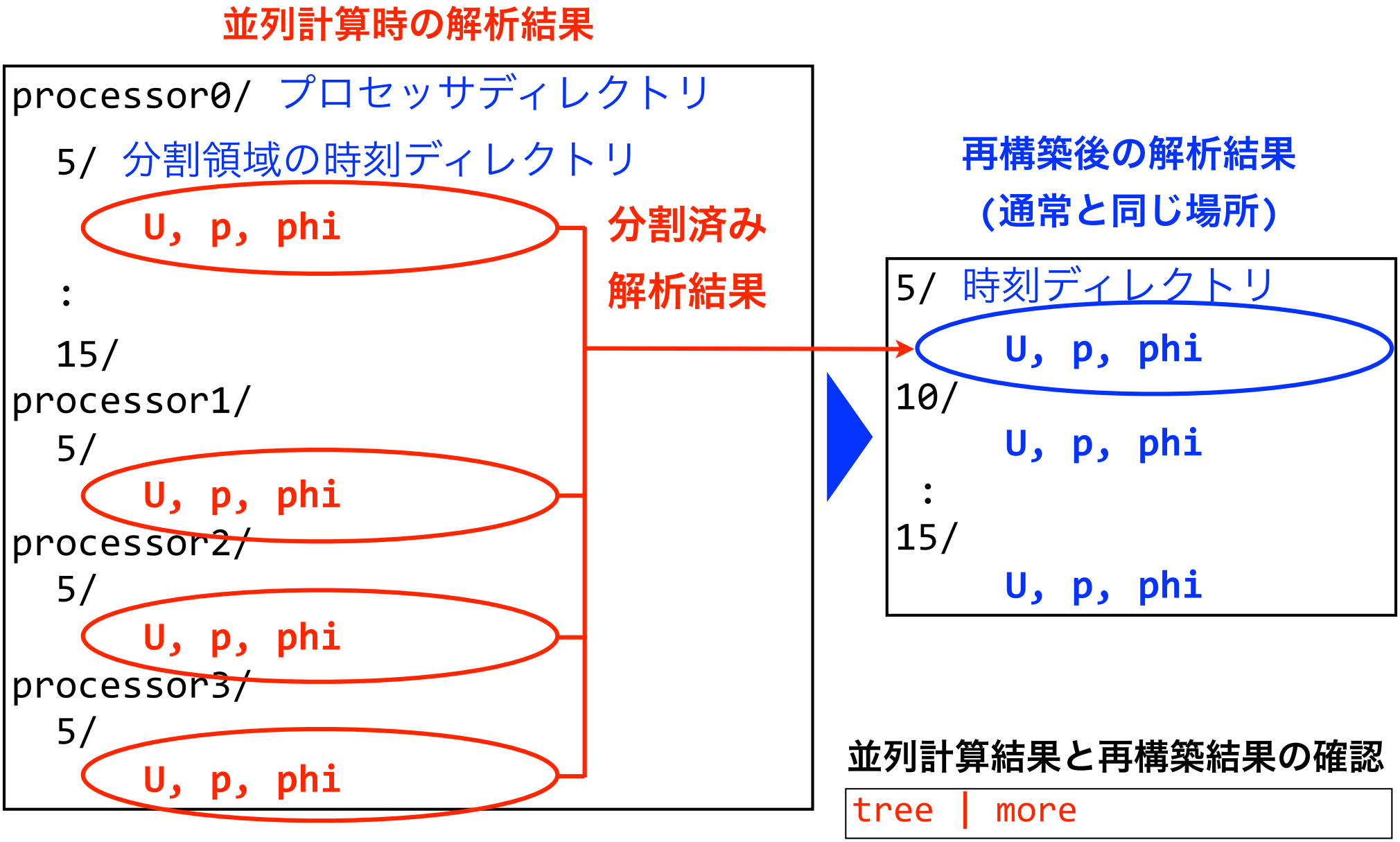
```
#!/bin/bash
#PJM -L rscgrp=lecture-flat
#PJM -L node=1
#PJM -L elapse=0:05:00
#PJM -g gt00
#PJM -S

module purge
module load intel/2018.1.163
export HOME=/work/gt00/$USER
. /work/gt00/share/OpenFOAM/OpenFOAM-v1712/etc/bashrc \
WM_COMPILER=Icc2018_1_163 WM_MPLIB=INTELMPI2018_1_163

env
reconstructPar >& $PJM_JOBNAME.l$PJM_JOBID
```

並列計算結果の再構築

領域分割ジョブの投入

```
pjsub reconstructPar.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
```

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

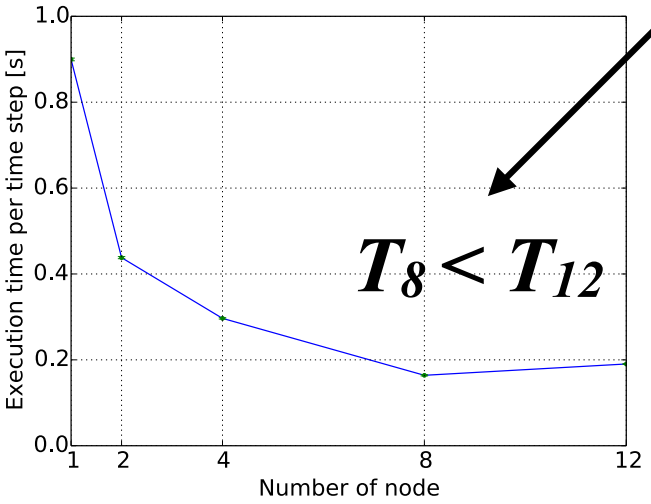
T_S ： ベースとなるプロセス数(1プロセス，1ノード等)での実行時間

T_P ： ベースとなるプロセス数×Pでの実行時間

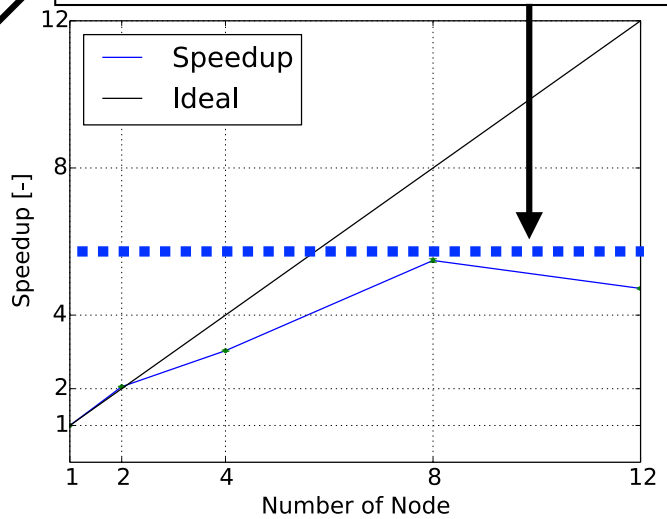
- 並列化効率 E_P

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

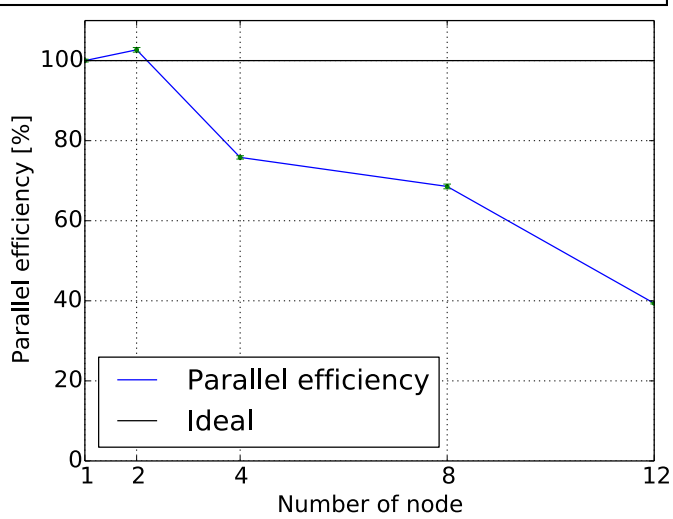
このケースでは12ノード以上使用しても非効率
→台数効果の飽和(Saturation)をベンチマークテストで事前に把握し，非効率な計算を行わないことが重要



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=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 [%]：$E_P = S_P / P \times 100 =$ (t(1)/t(P))/P×100

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

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

```
#Filename,TimeSteps[-],InitTime[s],LastTime[s],Time[s],AveTime[s]
par.sh.l1234567,3000,0.55,27.57,27.02,0.00900967
seq.sh.l1234567,3000,0.5,18.95,18.45,0.00615205
```

pythonやbcなどで計算

```
python -c "print 18.45/27.02" # bcの場合: echo 18.45/27.02 | bc -l
```

```
0.68282753515
```

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と相対的に多いので，並列計算の効率が悪い.

演習課題(続き)

課題2 128分割格子で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.

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 ~/lecture
mkdir cavity3
foamCopySettings cavity2 cavity3
cd cavity3
foamCleanTutorials
rm *.sh.*
```

新ケース用ディレクトリ作成(名前は任意)

設定をコピー

postProcessingなどの実行結果を消去

バッチジョブの出力ファイルを消去

領域分割設定ファイルの編集例

```
vi system/decomposeParDict
```

```
numberOfSubdomains 64;       //領域分割数
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 par.sh
```

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

ノード数と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)

```
pjsub blockMesh.sh
pjsub seq.sh
pjsub decomposePar.sh
pjsub par.sh
```

今回は実行しない

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 blockMesh.sh #出力: [INFO] PJM 0000 pjsub Job 1234567_0 submitted.
pjsub --step --sparam jid=1234567 seq.sh  #jid=ジョブID. _0(サブジョブID)は不要
pjsub --step --sparam jid=1234567 decomposePar.sh
pjsub --step --sparam jid=1234567 par.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を実行(more Allrun.seq_par で中身確認後)

```
./Allrun.seq_par
pjstat -E
../bin/averageExecutionTime.sh
```

ステップジョブ内のサブジョブを確認するには-Eオプションを付ける

ジョブが完了したら実行時間から並列化効率等を求める

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)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

seq-vtune.sh (赤字がvtune用追加分)

```
module purge
module load intel/2018.1.163
module load vtune/2018.1.0.535340
export HOME=/work/gt00/$USER
. /work/gt00/share/OpenFOAM/OpenFOAM-v1712/etc/bashrc \
WM_COMPILER=Icc2018_1_163 WM_MPLIB=INTELMPI2018_1_163

env
amplxe-cl -c advanced-hotspots -r $PJM_JOBNAME.v$PJM_JOBID \
icoFoam >& $PJM_JOBNAME.l$PJM_JOBID
```

vtuneのmoduleをload

amplxe-cl: VTuneコマンド
-c advanced-hotspots
(高度ホットスポット解析.
他にもhotspotsなどがある)
-r 保存ディレクトリ(実行時に存在しているとエラー)

par-vtune.sh (赤字が並列計算用追加分)

```
mpiexec.hydra -n $PJM_MPI_PROC  \
-gtool "amplxe-cl -collect hotspots -r $PJM_JOBNAME.v$PJM_JOBID:0" \
numactl -p1 \
icoFoam -parallel >& $PJM_JOBNAME.l$PJM_JOBID
```

-gtoolの引数にVTuneの実行コマンドを記述
-r 保存ディレクトリ:解析MPIプロセス範囲

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 seq-vtune.sh
tail -f seq-vtune.sh.l*
```

ソルバのログを追跡

```
Elapsed Time:          45.183
Paused Time:           0.0
CPU Time:              44.560
Average CPU Usage:     0.993
CPI Rate:              1.121
```

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 99.3%

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
amplxe-cl -R hotspots -r seq-vtune.sh.v* > seq-vtune.txt
more seq-vtune.txt
```

```
Function                                          CPU Time
Foam::DICPreconditioner::precondition             15.210s
Foam::lduMatrix::Amul                             10.960s
```

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(seq-vtune.shの代りにpar-vtune.shについて上記を実行)

```
native_queued_spin_lock_slowpath          2.940s
I_MPI_memcpy_min0_max64_sse               0.730s
Foam::DICPreconditioner::precondition     0.290s
```

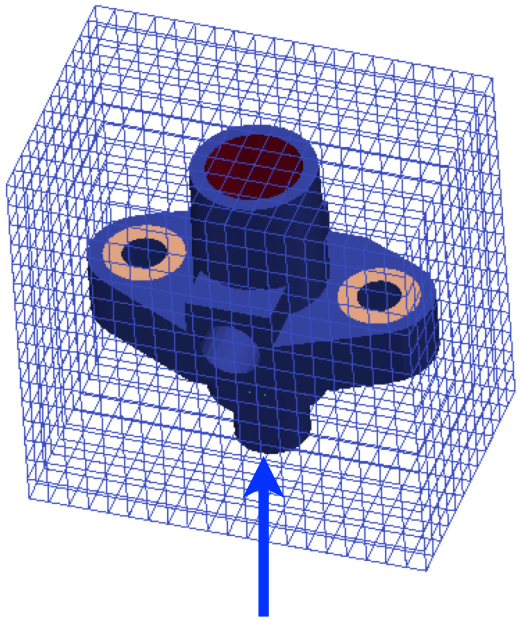
MPI関連(並列数が増えると支配的となる)

線形ソルバPCGのDIC前処理

snappyHexMeshによる格子生成演習

blockMesh
構造格子メッシャー

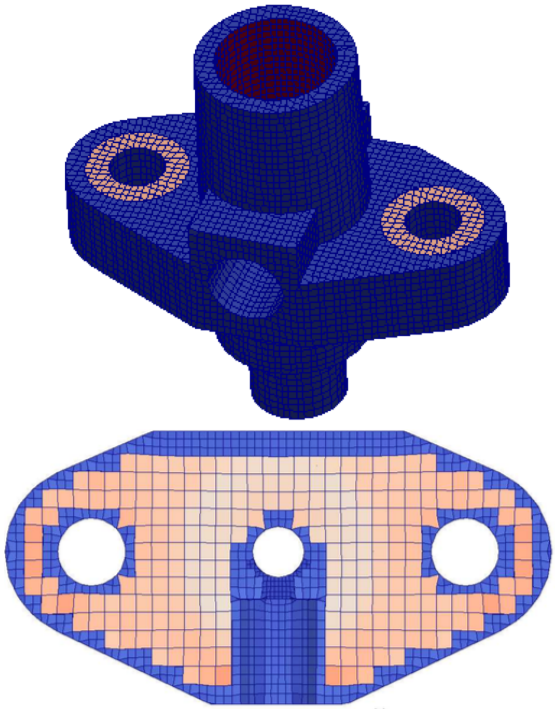
解析領域を含む構造格子のベース格子を生成



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



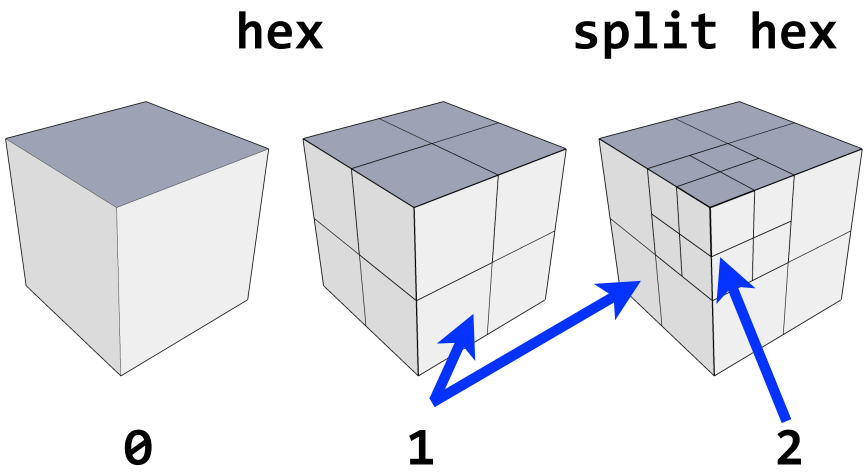
○ ほぼ六面体の格子が自動的に生成可能

× 任意間隔のベース格子が作成が難しい

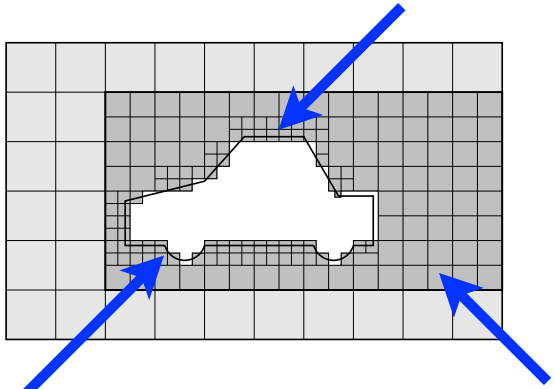
× 生成格子がベース格子の形状に大きく依存

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
- STL形式等の三角形分割曲面形状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)
- 格子の細分割は8分木(2x2x2の分割を再帰的に行う)
- 曲面形状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)
- 細分割領域を曲面形状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)

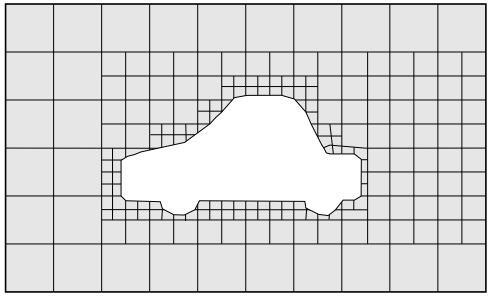


三角形分割曲面形状
(STL, wavefront OBJなど)

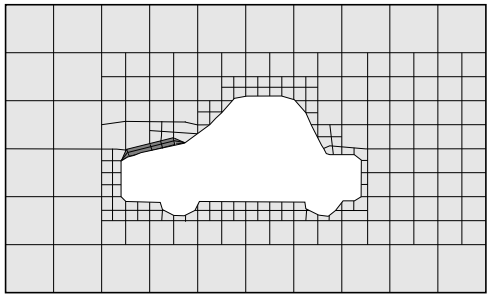


snappyHexMeshの特徴(続き)

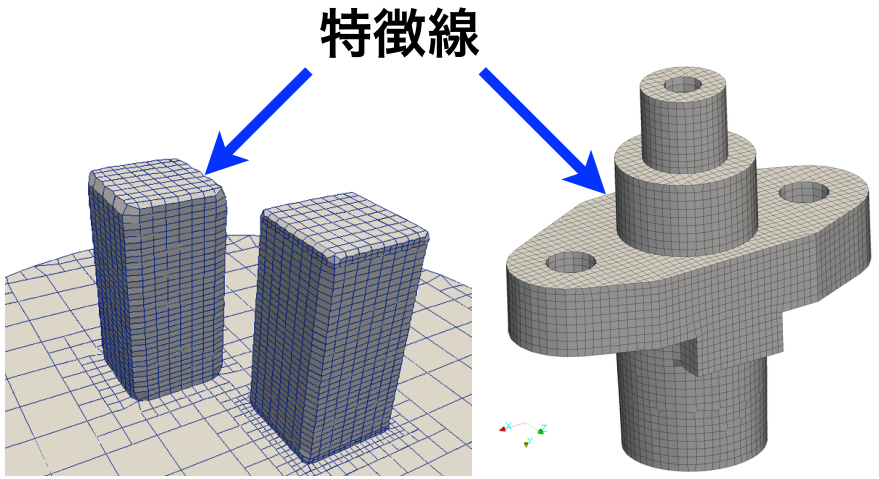
- STL表面や基礎形状の表面に境界適合するように格子を滑らかに移動させることができる
- STL表面や基礎形状の表面にレイヤを滑らかに挿入できる
- 分割領域毎の格子数をロードバランシングしながら，並列に格子生成ができる
- 境界適合における特徴辺の再現はVer. 2までは実装されておらず、特徴辺は丸くなったが，Ver. 2から特徴辺再現機能が実装され，形状再現性が高まった.
- 自動格子生成が可能だが，設定が繁雑で，特にレイヤ挿入の設定・制御は困難.



境界適合
(snapping



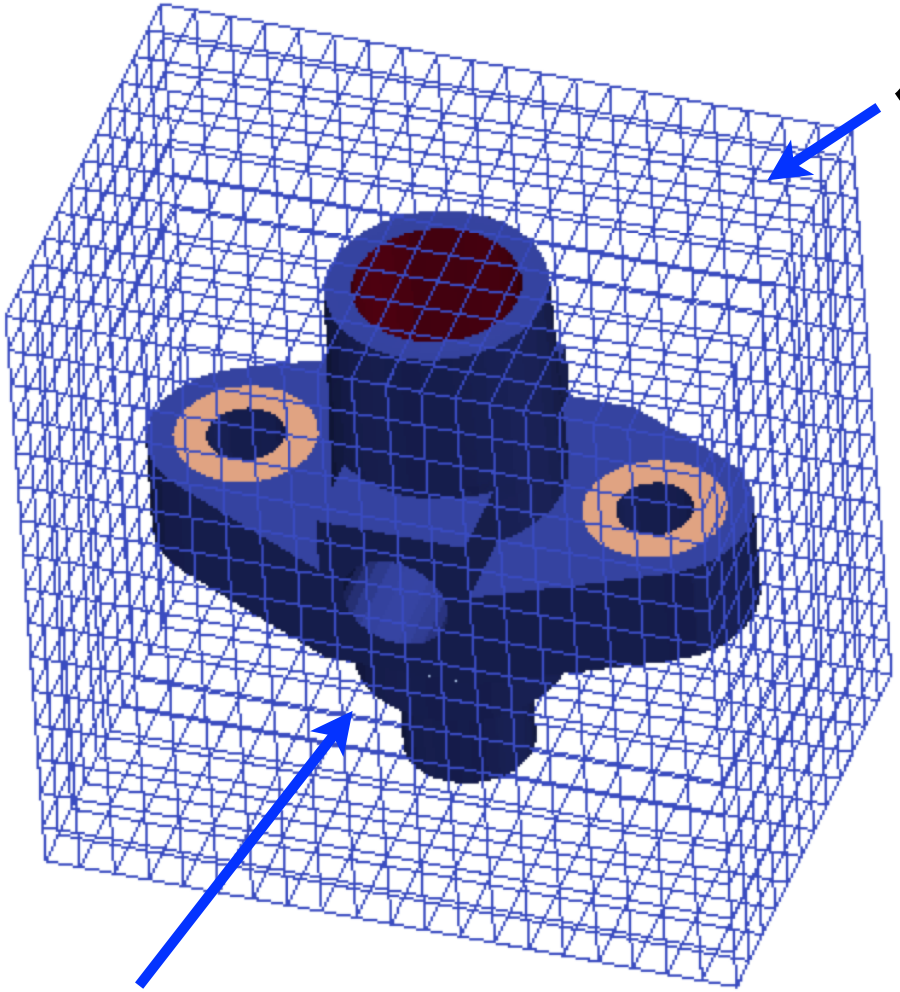
レイヤ挿入



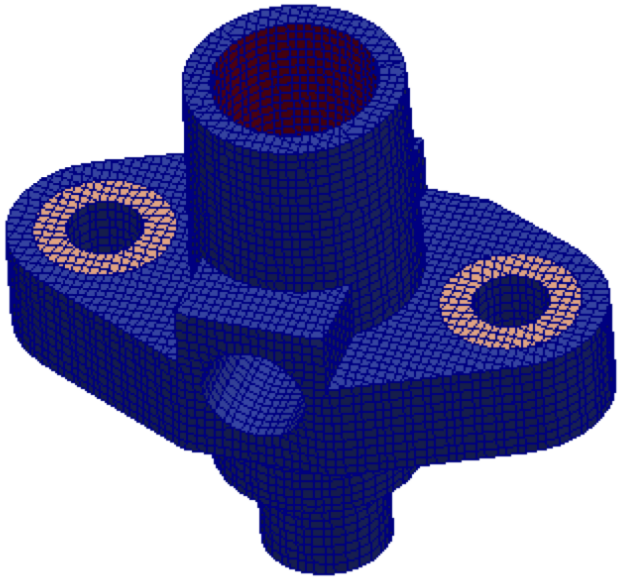
Ver. 2未満　　Ver. 2以降

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)

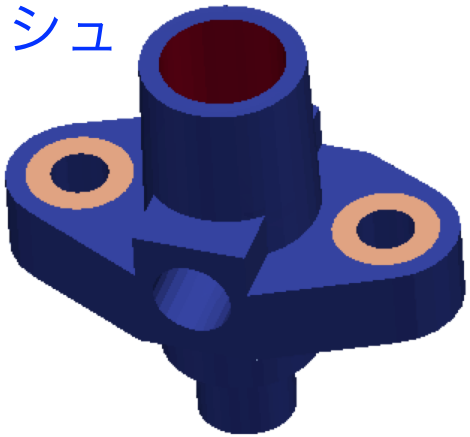


生成された格子(パッチ別に色分け)

snappyHexMeshDict概要

system/snappyHexMeshDict

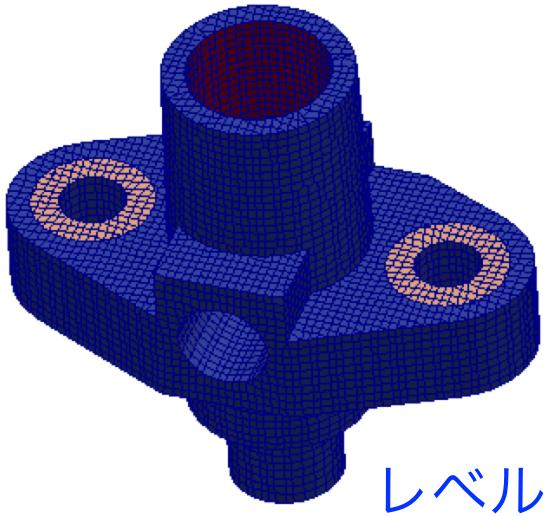
```
castellatedMesh true; //階段状格子
snap            true; //境界適合する
addLayers       false; //レイヤ付加しない

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)
};
```



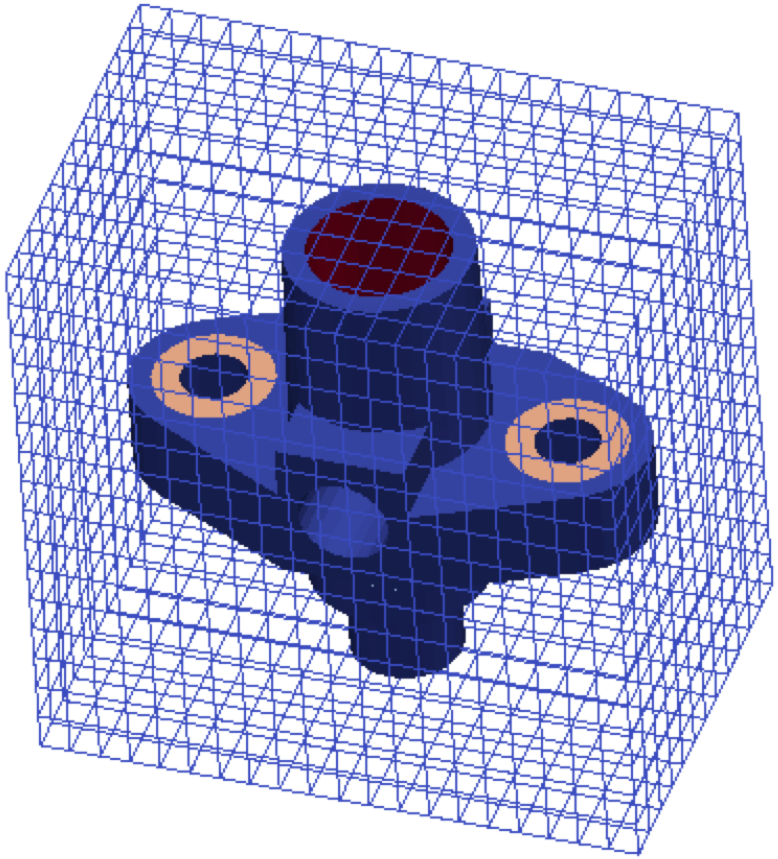
snappyHexMeshDict概要(続き)

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
module load intel/2018.1.163
export HOME=/work/gt00/$USER
. /work/gt00/share/OpenFOAM/OpenFOAM-v1712/etc/bashrc \
WM_COMPILER=Icc2018_1_163 WM_MPLIB=INTELMPI2018_1_163

. /usr/local/bin/mpi_core_setting.sh
export I_MPI_DEBUG=5

env
foamRunTutorials >& $PJM_JOBNAME.l$PJM_JOBID
```

OF1712では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

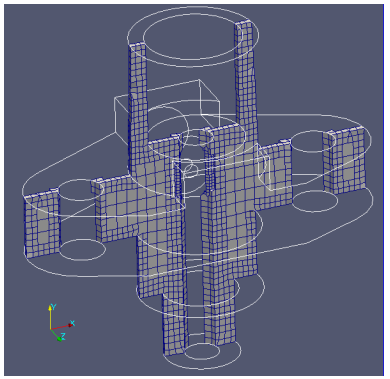
flangeチュートリアルの実行と格子可視化

```
cd ~/lecture
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:lecture/ ~/lecture/
cd ~/lecture/flange/
touch pv.foam
```



ParaViewで可視化(適宜データを転送してから実行)

1. File/Open *pv.foam*→OK
2. Decompose Polyhedra *uncheck*→Mesh Regions *select*→Apply
3. Filters/Extract Block/Block Indices/ patches *select*→Apply
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

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.
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

```
cd ~/lecture
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

```
cp ../foamRunTutorials.sh ./
```

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

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)

チュートリアルの実行結果の転送(別端末で実行)

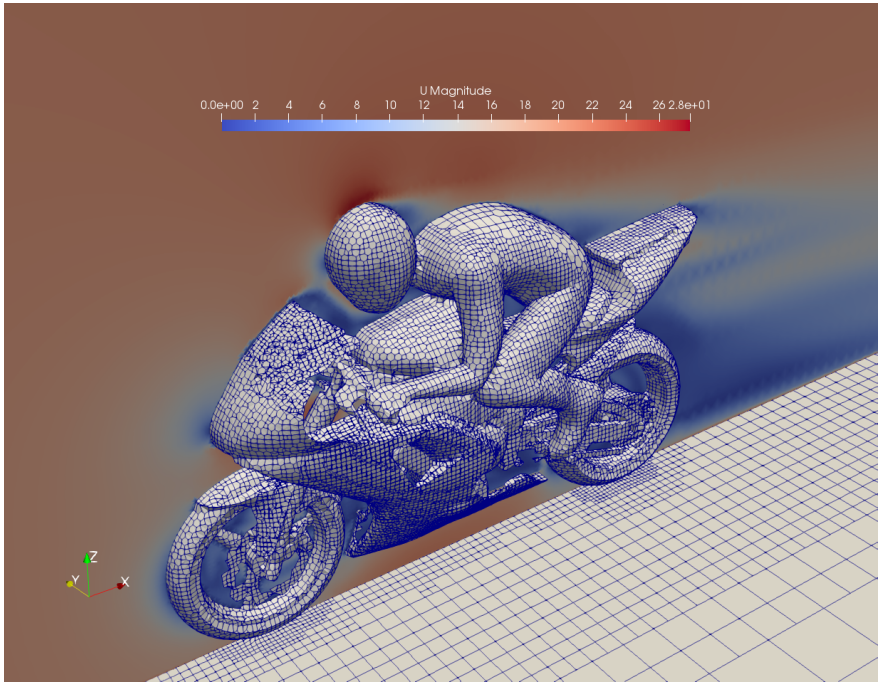
```
rsync -auv txxxxx@ofp.jcahpc.jp:lecture/ ~/lecture/
cd ~/lecture/motorBike/
touch pv.foam
```

または，カーソル↑で呼出す

ParaViewによる可視化(領域分割されたデータを可視化する例)

1. File/Open *pv.foam*→OK
2. Case Type *Decomposed Case* → Mesh Regions *unselect* → Apply
3. Mesh Regions *select* → InternalMesh,frontAndBack,inlet,outlet,upperWall *unselect* → Apply
4. Representation/Surface With Edges
5. File/Open → postProcessing/cuttingPlane/最終時刻/U_yNormal.vtk → OK → Apply
6. Coloring/・U

postProcessing以下は実行時作成・可視化されたファイル

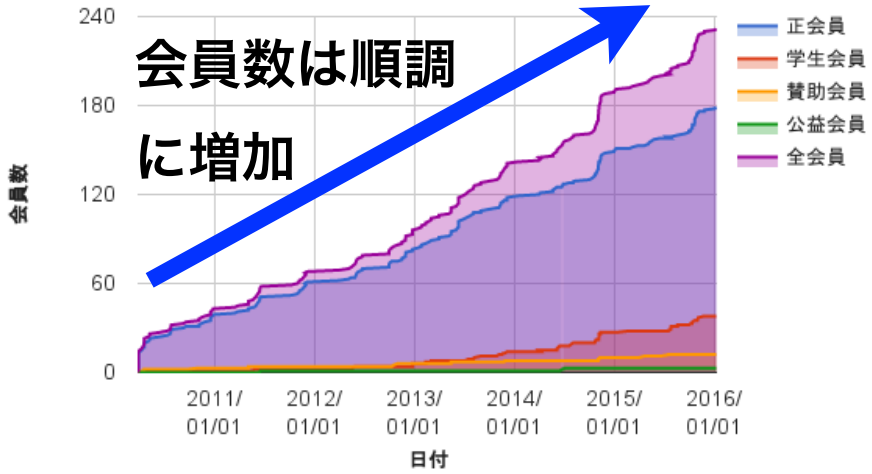


15分間の制限時間では，最終時刻の500まで解析できておらず，並列計算結果の再構築(reconstructPar)が実行されないので，ここでは並列計算結果をそのまま可視化する

関連情報：オープンCAE学会

- ✓ **設立**：2009年11月
- ✓ **会長**： 中川 慎二(富山県立大学)
- ✓ **会員数**：222 (2017年3月31日現在)
- ✓ 正：175，学生32，賛助12，公益6
- ✓ **委員会**：シンポジウム，講習会，資料翻訳，出版・編集，コミュニティ，V&V，国際化推進，広報・賛助，Web編集，モデルベースデザイン，表彰

OpenCAE
www.opencae.or.jp



主な年間行事

時期	名称	開催地
6月頃	総会・講習(1日)	東京
8月頃	サマースクール(2泊3日)	日本各地
11月頃	シンポジウム(3日程度) 見学会・講習会・講演会・ツアー等	日本各地 (隔年で東京開催)
3月頃	春季講習会(1日)	東京以外

講習会委員会: オープンソースCAE講習会運営

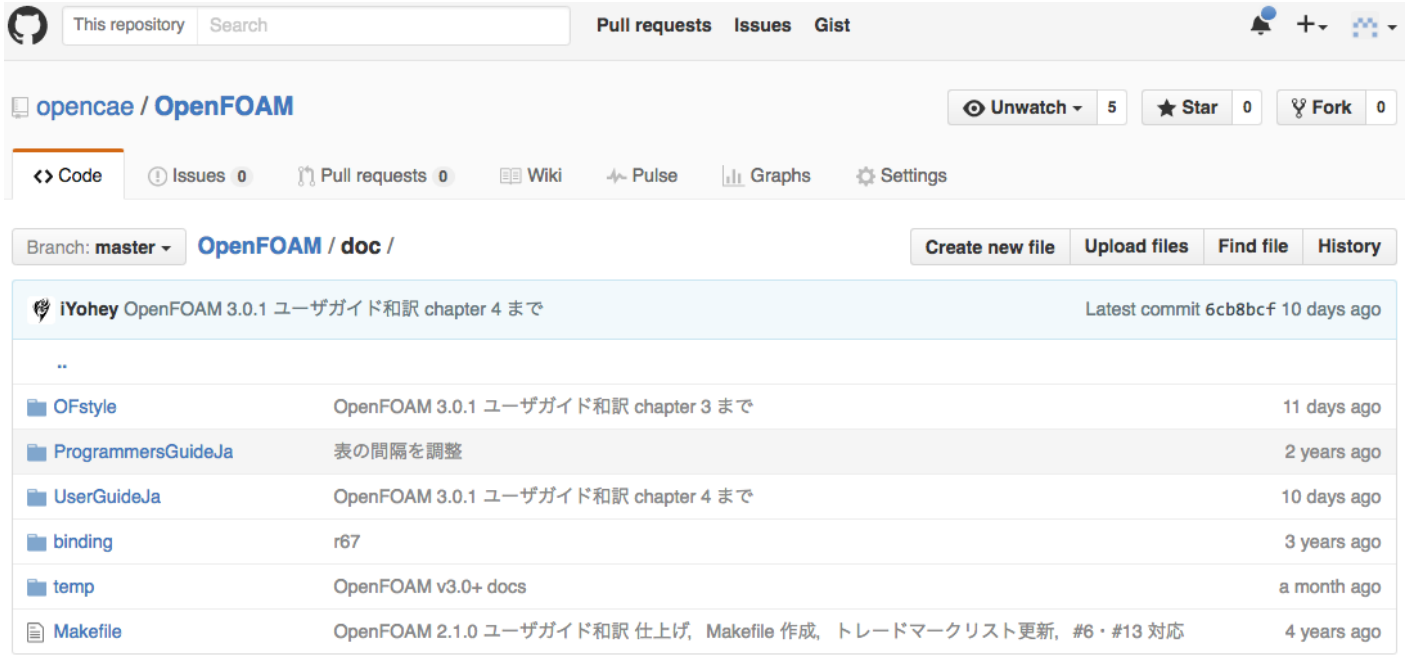
- **総会付帯講習会(2018年6月22日@市ケ谷)**
  - ✓ 90分×2並列×3コマ程度
- **サマースクール(2018年8月30日〜9月2日@箱根)**
  - ✓ 学生・若手(概ね40歳未満)対象
  - ✓ 合宿形式(2泊3日)
  - ✓ 講習会・懇親会・ディスカッション
- **シンポジウム(2018年12月頃@関東)**
  - ✓ 90分×3並列×4コマ程度
- **春季講習会(2018年3月頃@未定)**

詳細はオープンCAE学会のWebページ参照(http://www.opencae.or.jp/)

資料翻訳委員会: OpenFOAMユーザーガイド和訳

- LaTeXソースや図など全てレポジトリで公開
- OpenFOAM FoundationのWebサイトにも，学会の和訳が置かれている
- 現在，3.0.1版まで公開．学会でplus版の和訳も計画中

https://github.com/opencae/OpenFOAM

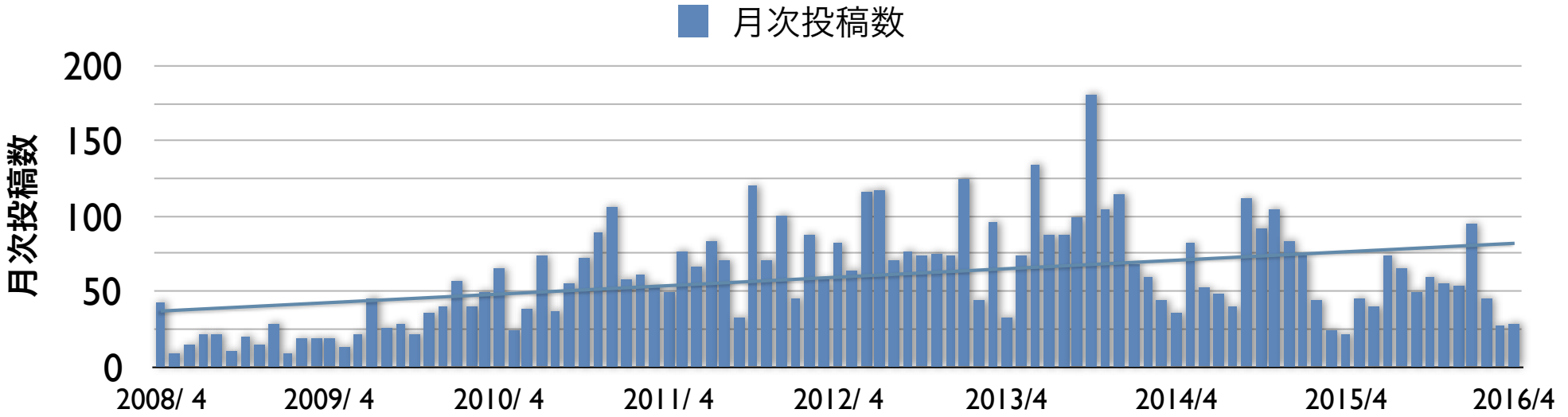


コミュニティ委員会: OpenFOAM掲示板運営

OpenFOAM Googleグループ

- 設立：2008年3月
- 登録者：919名 (2018/1/14時点)
- 質問，情報交換，イベント告知
- 匿名で質問可能
- 初心者の駆け込み寺

第４７回オープンＣＡＥ勉強会@岐阜のご案内 （1）	1 件の投稿	5月9日
OpenFOAM-v3.0+のバイナリインストールについて （4）	4	5月6日
チュートリアルcircuitBoardCoolingの設定について （3）	3	5月6日
オープンCAE講習会のご案内 （1）	1	5月5日
移動メッシュの破たんについて （3）	3	5月3日
【関西】(4/24) 第48回オープンCAE勉強会@関西 開催のご案内 …	3	4月23日
interDymFoamでの質問 （3）	3	4月23日
convectiveOutletの対流速度につきまして （2）	2	4月20日
第30回オープンCAE勉強会@広島 4月16日開催 （2）	2	4月10日
メッシュの結合について（enGridとblockMesh） （12）	12	4月9日
【勉強会@富山】4月16日(土)オープンCEA勉強会@富山(第42回)開…	1	4月8日



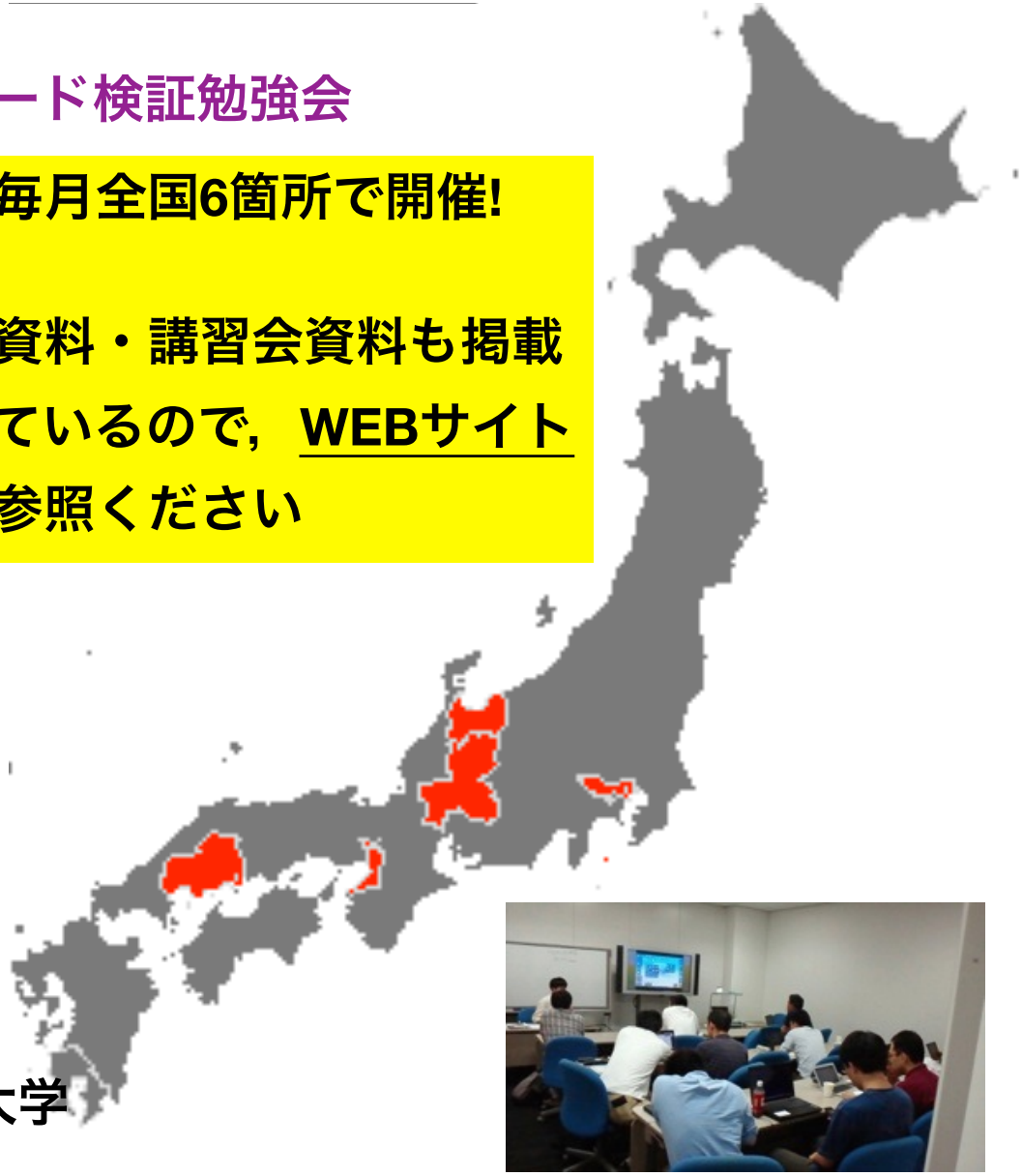
https://groups.google.com/forum/#!forum/openfoam

コミュニティ委員会: オープンCAE勉強会後援

1. @関東(流体など)(2010年6月～)
   Ustream配信(初期), OpenFOAMコード検証勉強会
2. @関西(2010年12月～)
   自作風洞実験, ブックレビュー
3. @岐阜(2011年1月～)
   初心者のみ質問会・夏合宿
4. @富山(2012年5月～)
   ミニ講習会・鱒寿司制覇懇親会
5. @広島(2012年7月～)
   ミニ講習会
6. @関東(構造など)(2014年10月～)
   構造解析に特化した勉強会

ほぼ毎月全国6箇所で開催!

発表資料・講習会資料も掲載されているので, WEBサイトをご参照ください

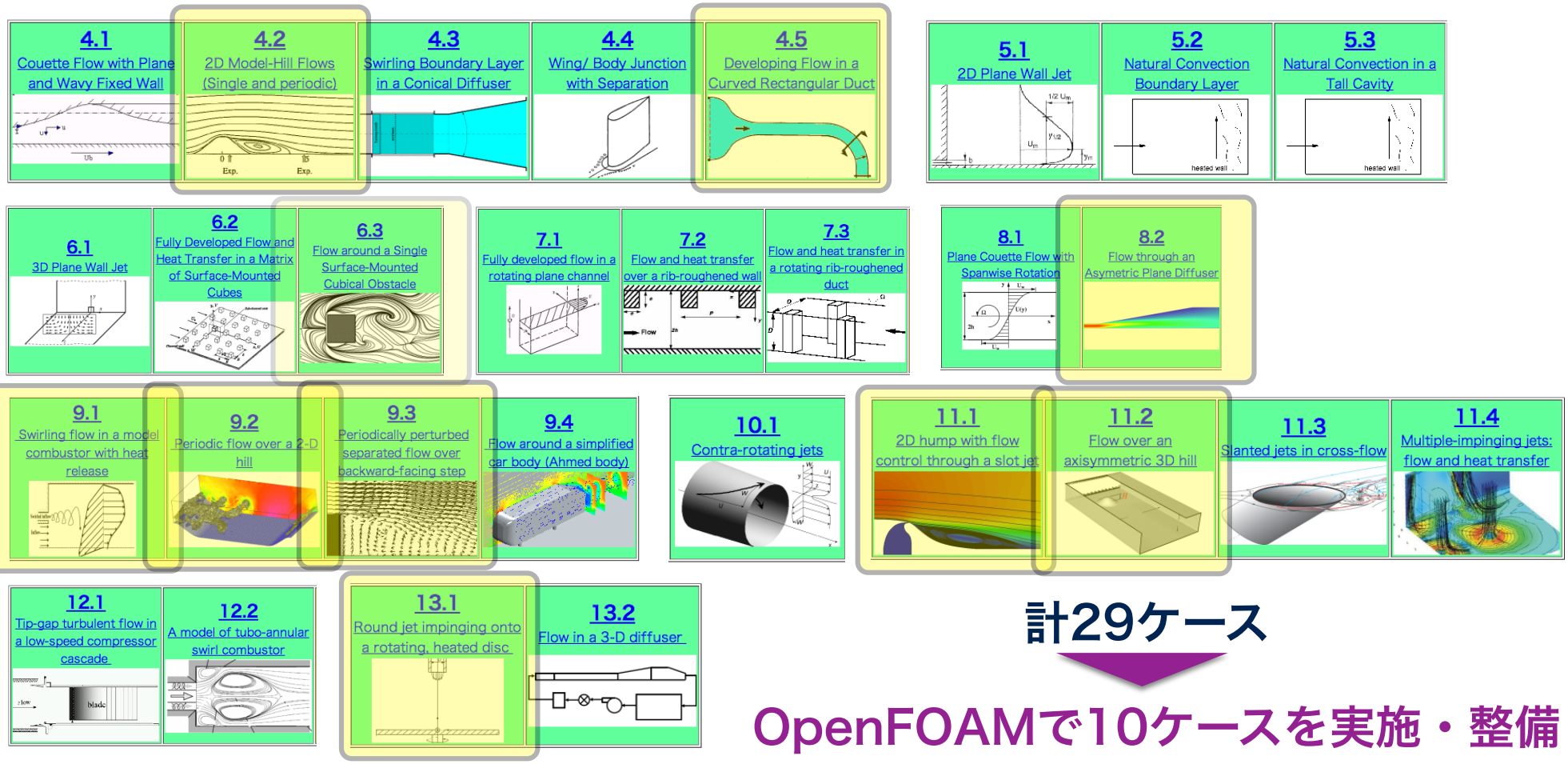


合同勉強会(2018年6月23日)@東洋大学

V&V委員会: ERCOFTAC SIG15ベンチマーク

学会のレポジトリで実験値との検証ケースを公開．実験値との比較プロットも自動で可能

ERCOFTAC(European Research Community on Flow, Turbulence And Combustion)内の乱流モデリンググループのワークショップ('95-'01)で実施したベンチマーク



計29ケース

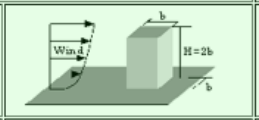
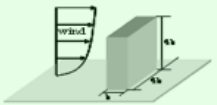
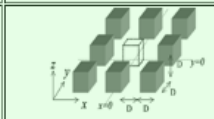
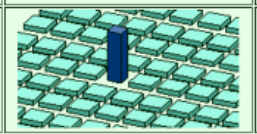
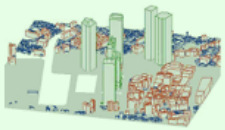
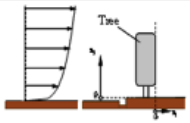
OpenFOAMで10ケースを実施・整備

V&V委員会: 市街地風環境ベンチマーク

日本建築学会 流体数値計算による風環境評価ガイドライン作成WG が整備

書籍

WEBサイト(実験データや標準計算条件)

	test case		dataset	Ref.
A	2:1:1 shape building model		Data file :CaseA(1_1_2).xls	[1]
B	4:4:1 shape building model		Data file : CaseB(4_4_1).xls	[2][3]
C	Simple Building blocks		Data file : CaseC(City_blocks).xls	-
D	A high-rise building in city blocks		Data file : CaseD(Highrise+Blocks).xls CAD File(DXF) : CaseD_dxf.zip CAD File(MCD) : CaseD_mcd.zip	[5]
E	Building complexes with simple building shape in actual urban area (Niigata)		Data file : CaseE(Niigata).xls CAD File(DXF) :CaseE_dxf.zip CAD File(MCD) : CaseE_mcd.zip	[6]
F	Building complexes with complicated building shape in actual urban area (Shinjuku)		Data file : CaseF(Shinjuku).xls CAD File(DXF) :CaseF_dxf.zip CAD File(MCD) : CaseF_mcd.zip CAD File(STL) : CaseF_stl.zip	[6]
G	Two-dimensional pine tree		Data file : CaseG(Tree).xls	[7]

OpenFOAMで6ケースを実施・整備

V&V委員会: 工学ナビでも公開

東京大学生産技術研究所・革新的シミュレーション研究センター(センター長：加藤千幸先生)が制作運営しているWEBサイト

- 複合材料強度信頼性評価 (10)
  - FrontCOMP (10)
- 音響解析 (2)
  - FFB-Acoustics (2)

ナノテクノロジー (65)

- 第一原理電子状態解析 (65)
  - PHASE (65)

ライフサイエンス (36)

- フラグメント分子軌道法 (16)
  - ABINIT-MP (16)
- 標準密度汎関数法 (20)
  - ProteinDF (20)

- OpenFOAM (16)
  - OpenFOAM Version 2.2.2 (16)
- 最適化設計 (3)
  - CHEETAH (3)
- 構造解析 (37)
  - ADVENTURECluster Solver (4)
  - Front ISTR (31)
- 流体構造連成解析 (1)
- 流体解析 (54)
  - FFR (4)
  - FFV (16)
  - FrontFlow/blue (28)
  - GKV (2)
  - GT5D (2)
  - LANS3D (3)

解析格子を以下に示す．計算格子生成にはOpenFOAM付属の自動格子生成ユーティリティsnappyHexMeshを使用し，学会提供のCADデータをSTL形式に変換したデータを使用して約530万要素の格子を自動生成させた．

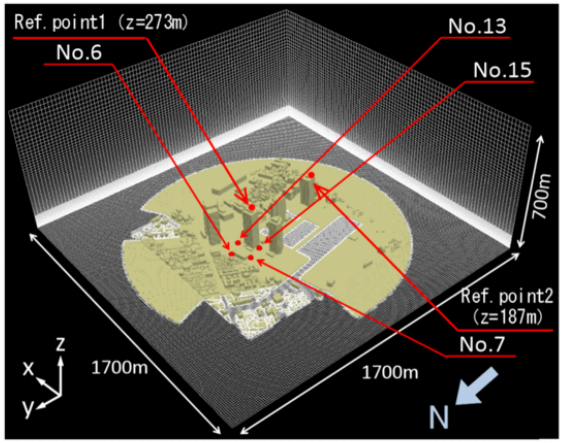
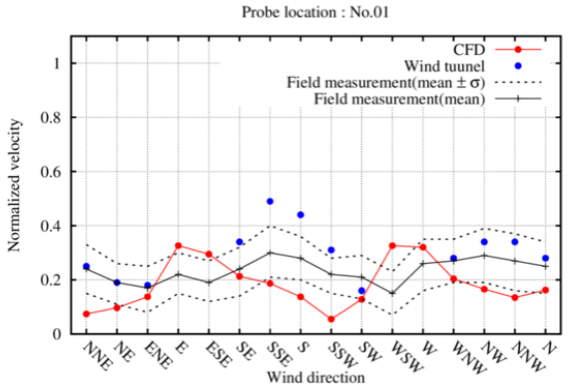


Fig. Calculation mesh (参考文献[Imano2010]から引用)

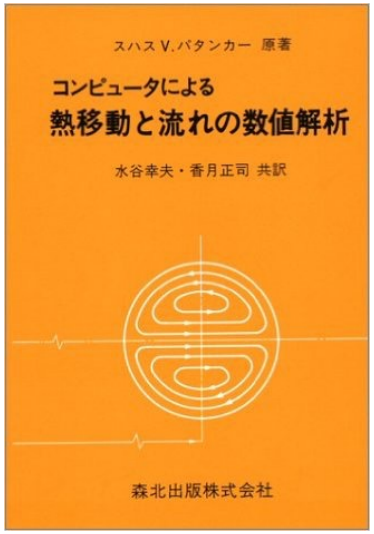
計算結果

各計測点における風向別の風速比を以下の図に示す．風洞実験値及び実測値[AIJ]も合わせて示す．何れの結果も，風向NE～N～NWでは参照点1(D)，それ以外の風向は参照点2(C)の風速で基準化してある．



例：新宿副都心高層ビル群の風環境

出版・編集委員会: OpenFOAM書籍編集



- 発売日：2016年6月17日
- 価格：3,456円(税込)
- 編集：オープンCAE学会
- 出版社：森北出版
- 日本初のOpenFOAM本!
- 書籍名はスハス V.パタンカー(森北出版)の「コンピュータによる熱移動と流れの数値解析」のオマージュ
- Amazonの機械工学カテゴリで1位達成!

OpenFOAMの主な国内会議

- **オープンCAEシンポジウム(2010年～，参加約80～150名)**
  - ✓ 主催: オープンCAE学会
  - ✓ 2～3並列×3～4コマ程度のトレーニング(構造解析・可視化・1DCAEも含む)
  - ✓ OF以外の流体解析ツールや構造解析・可視化関連の発表も有る．資料WEB公開
  - ✓ **現状OpenFOAM関連の発表件数は日本最大規模．年々増加**
- **OpenFOAM・CAEワークショップ (2013年～，参加約50～100名[参加費無料])**
  - ✓ 主催: 高度情報科学技術研究機構(RIST)
  - ✓ HPCI課題による「京」でのOFの事例．RISTによるOFのチューニング．富士通によるコンパイラの最適化などHPC向けの発表．資料WEB公開
- **ソフトウェアベンダーのユーザ会でのOpenFOAMセッション**
- **オープンCAE学会以外の学会でのOpenFOAMに関する研究発表**

学会，ユーザ会，HPC系WSなどで，年々OpenFOAMの発表が増加している

OpenFOAMの主な国際会議

- **OpenFOAM Workshop(2006年～，参加約140～400名)**
  - ✓ 主催: OpenFOAM Workshop committee
  - ✓ **世界最大規模(ドイツ開催時は参加者約400人)**
  - ✓ 3並列×4コマ程度のトレーニング(参加費に含まれる)
  - ✓ Hrvoje Jasakによる基調講演(開発方針など)
  - ✓ OFのカスタマイズ例・適用例など発表多数．講習会資料も公開
- **OpenFOAM User Conference (2013年～，初回参加222名)**
  - ✓ 主催: ESI社
  - ✓ 基調講演: OF開発にファンドしているVolkswagen等
  - ✓ 開発メンバーによる今後の開発方針，ユーザ適用事例等

多くの発表資料や講習会資料が公開されているので，
興味がある分野の資料を検索してみてください

関連Webサイト

[OFP] Oakforest-PACS スーパーコンピュータシステム(https://www.cc.u-tokyo.ac.jp/supercomputer/ofp/service/) **利用支援ポータル内の「Oakforest-PACS システム利用手引書」は必ず目を通す．なお，長期間の休止明けに文章更新や設定の変更が多いので，その際に更新情報を調べる．特に，デフォルトの設定が変更された場合，ジョブスクリプトでその設定を明示的に行なっていないと，計算速度や場合によっては計算結果が変わる可能性があるので注意する．**

[OFF] The OpenFOAM Foundation (http://www.openfoam.org/)

[OFC] OpenCFD Ltd (ESI Group)(http://www.openfoam.com/)

[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/)

[OCSJ] オープンCAE学会(http://www.opencae.or.jp/) OpenFOAMユーザガイド和訳，プログラマズガイド和訳，The ParaView Tutorial和訳，過去のシンポジウム・ワークショップ・講習会資料

[PENGUINITIS] (http://www.geocities.jp/penguinitis2002/) 圧倒的な情報量

[FN365] (http://caefn.com/) 多くのOpenFOAMに関するスライド(日本語・英語)も公開

[OFW] OpenFOAM Workshop (http://www.openfoamworkshop.org/) 世界最大規模のOpenFOAMの国際会議，カスタマイズ例・適用例など発表多数．要旨・スライド・トレーニング内容をWEB公開

[installOpenFOAM] OpenFOAM自動ビルドスクリプト(https://qiita.com/masazzz/items/857390c15a639deba4bb) Oakforest-PACSなどのスパコンでOpenFOAMを自動でビルドする