2018年7月27日@大阪大学サイバーメディアセンター

OpenFOAMハンズオンセミナー

今野　雅

(株式会社OCAEL・東京大学情報基盤センター客員研究員)

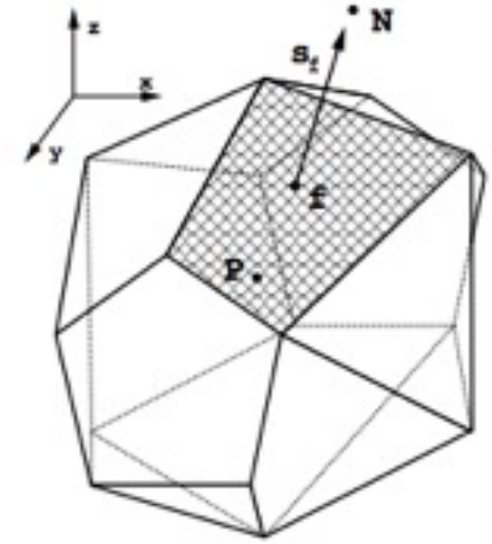
講習会プログラム

- 13:40-14:05 **OpenFOAM概要**
- 14:05-14:15 **OCTOPUSへのログイン**
- 14:15-14:30 **休憩**
- 14:30-16:00 **キャビティ流れ演習I**
  - ✓ blockMeshによる格子生成
  - ✓ ParaViewによる格子可視化
  - ✓ icoFoamによる流れ解析
  - ✓ ParaViewによる解析結果可視化
- 16:00-16:15 **休憩**
- 16:15-17:30 **キャビティ流れ演習II**
  - ✓ 解析結果サンプリング
  - ✓ 解析結果プロット
  - ✓ 並列計算
- 17:30-18:00 **その他チュートリアルの実行・質疑**

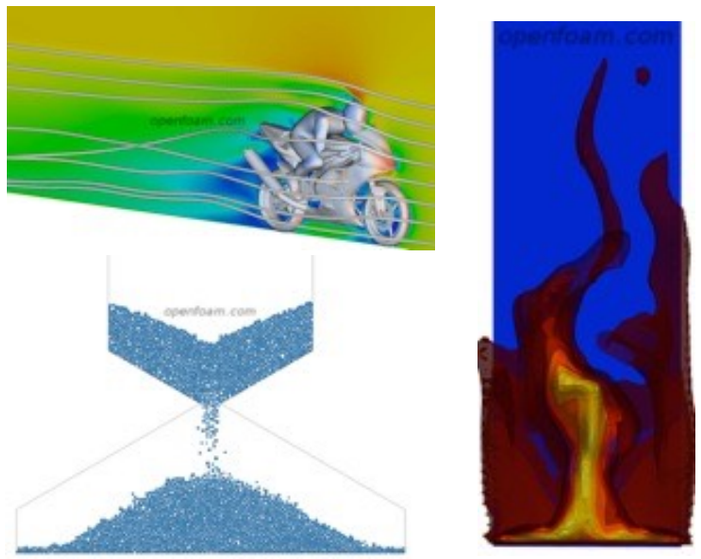
附録

- プロファイラVTuneの基礎的使い方
- Intel MPI使用時の注意点
- OCTOPUSへのOpenFOAMのインストール
- OCTUPUSでのOpenFOAMベンチマークテスト
- オープンCAE学会の紹介
- 関連WEBサイト

OpenFOAM概要



有限体積法
ポリヘドラル

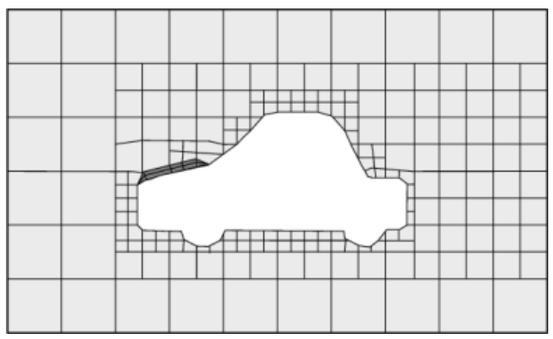


マルチフィジックス

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

```
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		

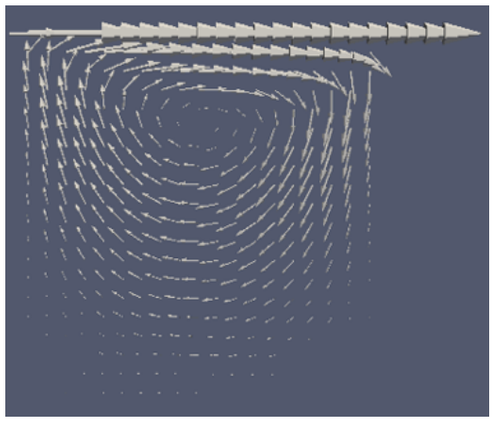
リリース年	2016				2017			2018	
リリース月	Jun	Oct	Jul	Dec	Jun	Jul	Dec	Jun	Jul
バージョン	4.0	4.1	v1606	v1612	v1706	5.0	v1712	v1806	6
カテゴリ数	12	12	12	12	12	12	13	13	12
ソルバ数	82	82	86	86	95	86	101	98	80
チュートリアル数	207	207	226	253	284	229	298	322	240

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

Foundation系はv1606+以降に採用されたMPIのメモリ使用量最適化がなされておらず，概ね2,000並列以上で使用量が莫大に増加するので，大規模並列ではPlus系を使用する必要がある

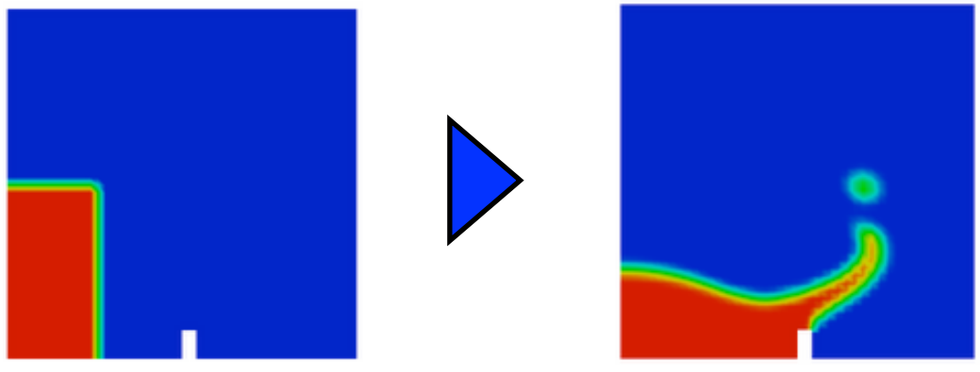
チュートリアルとは

- 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

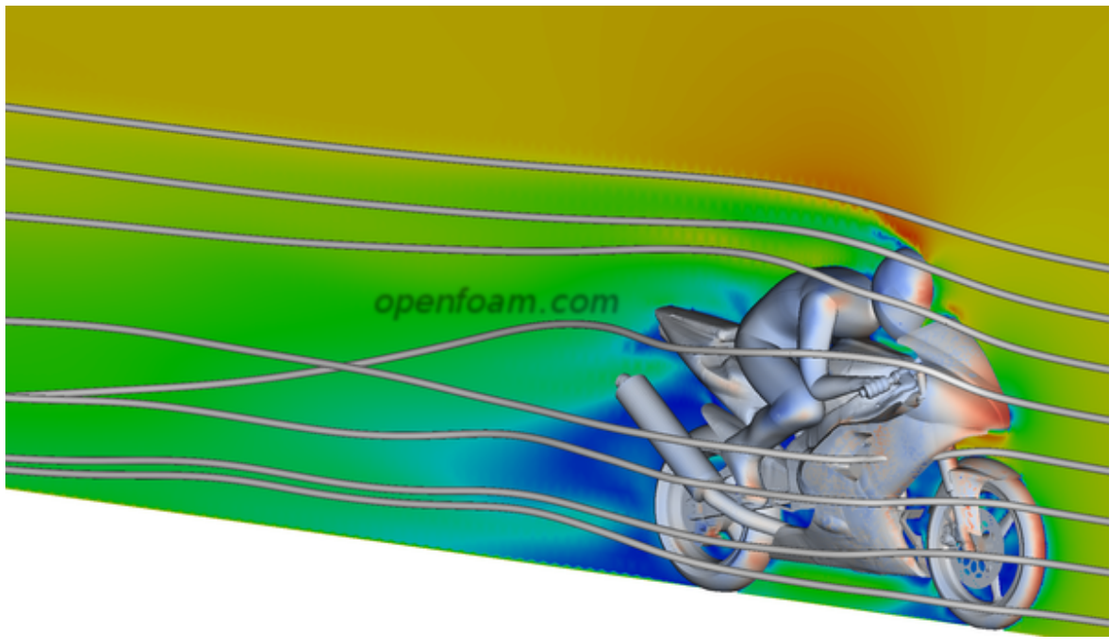


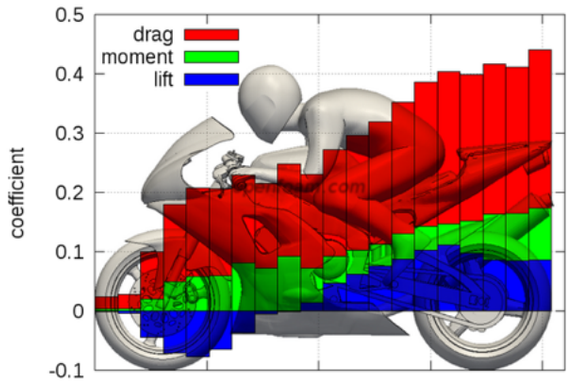
  - ✓ plateHole：穴あき板の応力解析

  - ✓ damBreak：ダムの決壊



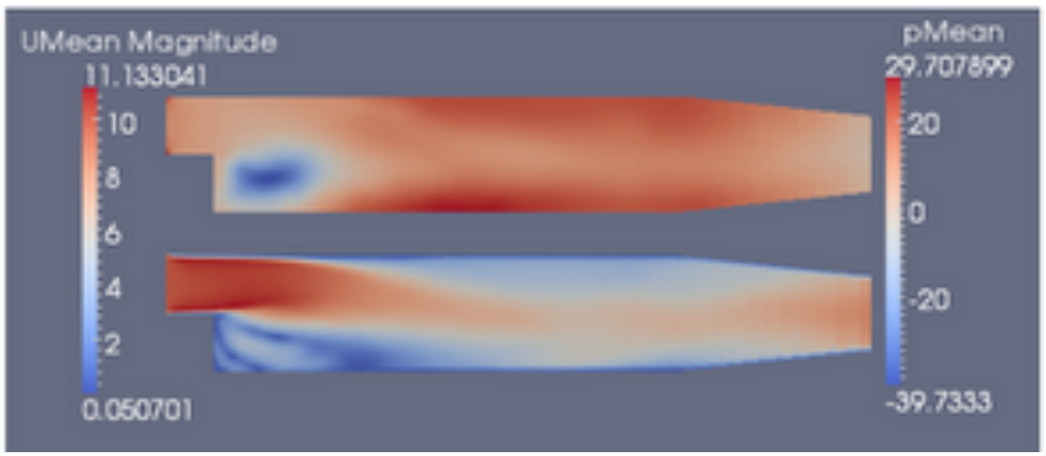
単相等温流れのチュートリアル例





バイク周りの流れ
moterBike

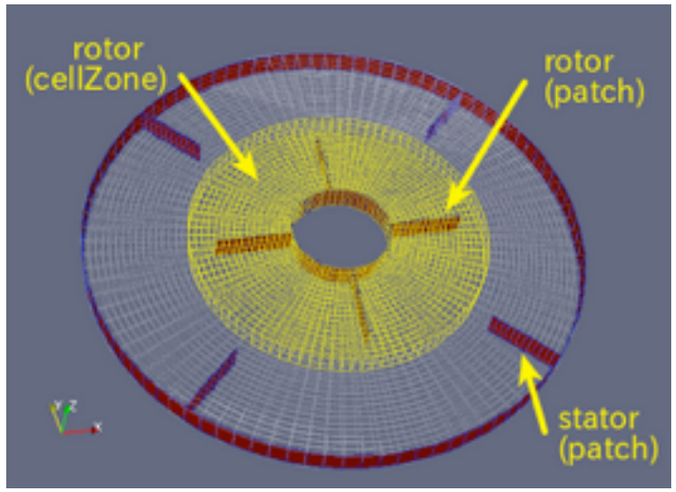
図出典：[OFF]

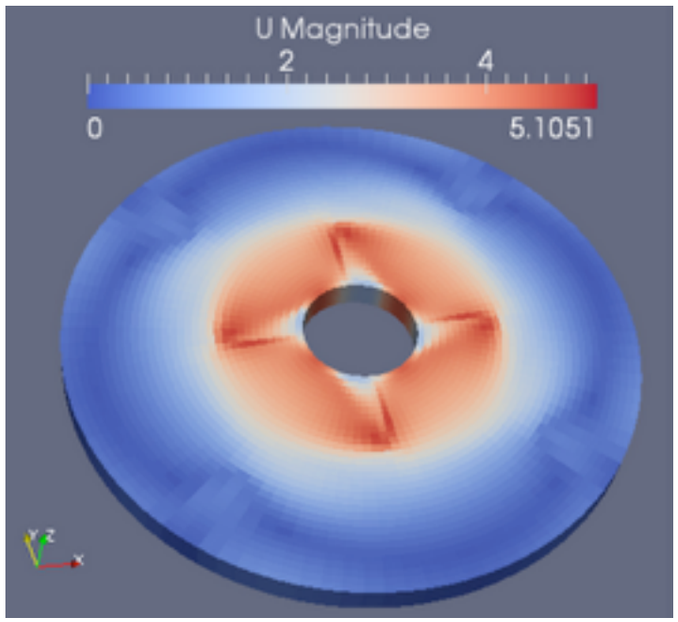


バックステップ流れ(LES)
pitzDaily

[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/)

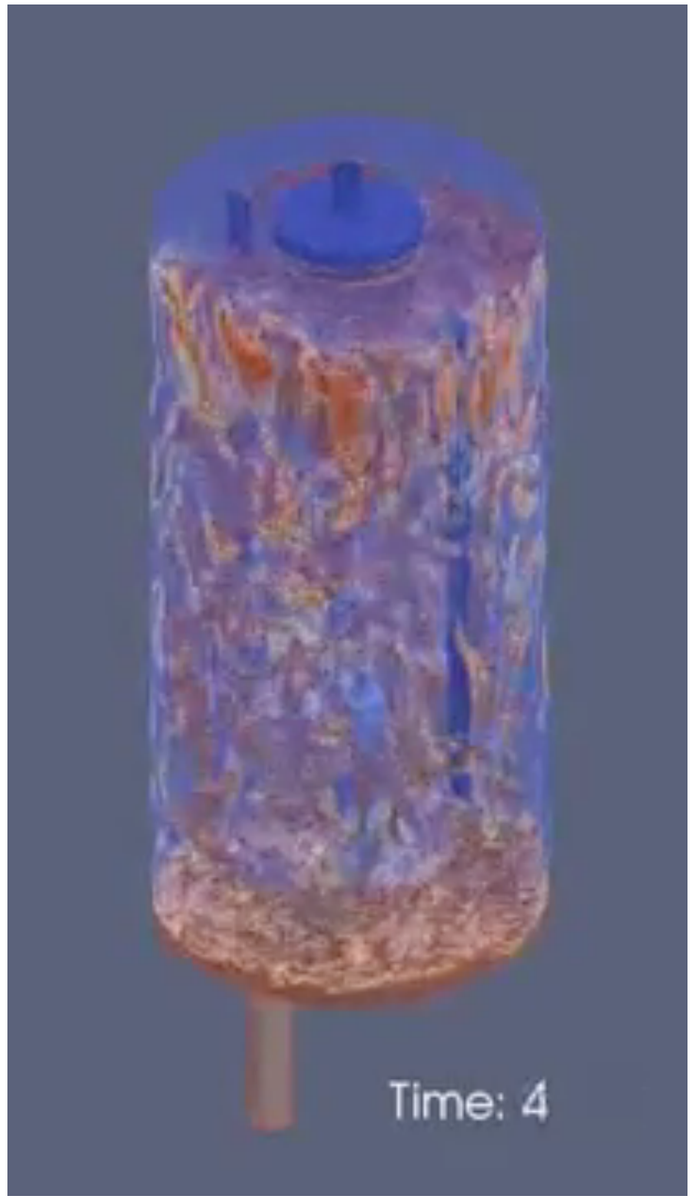
回転撹拌槽のチュートリアル例



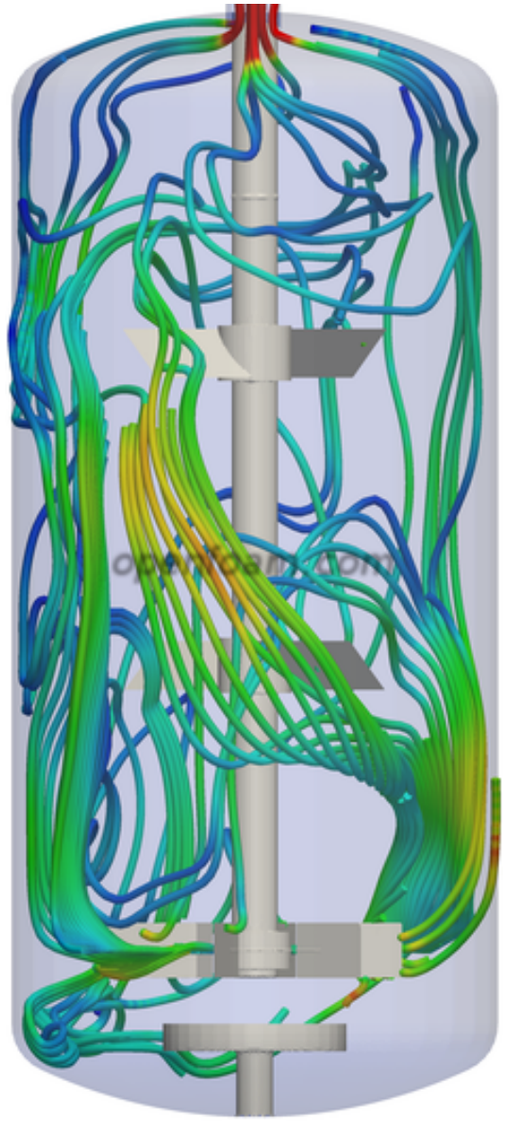


回転攪拌槽の流れ(MRF)
mixerVessel2D

図出典
[OFT]

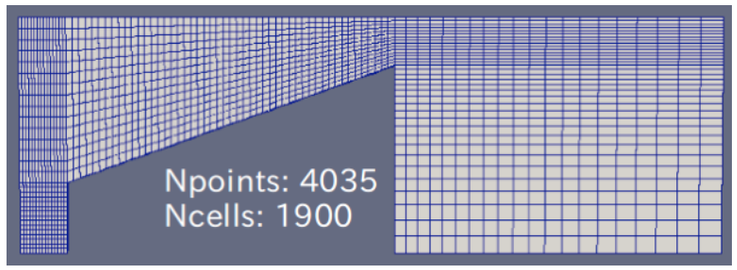


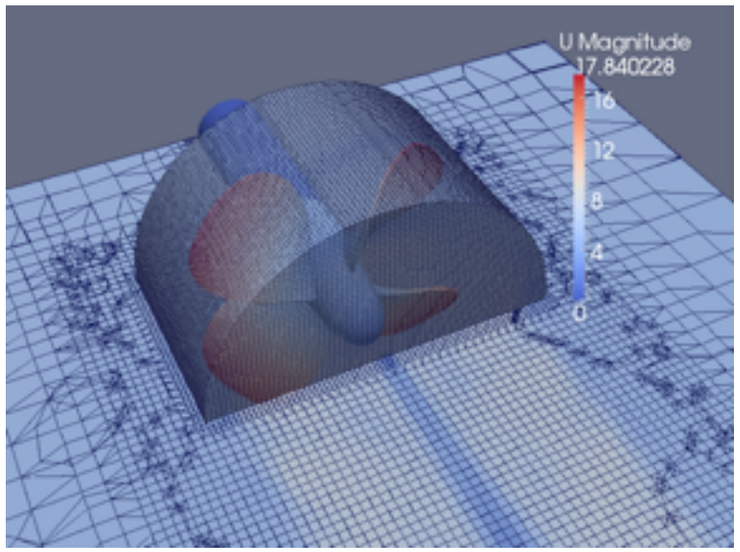
回転攪拌撹拌槽内の流れ
mixerVesselAMI

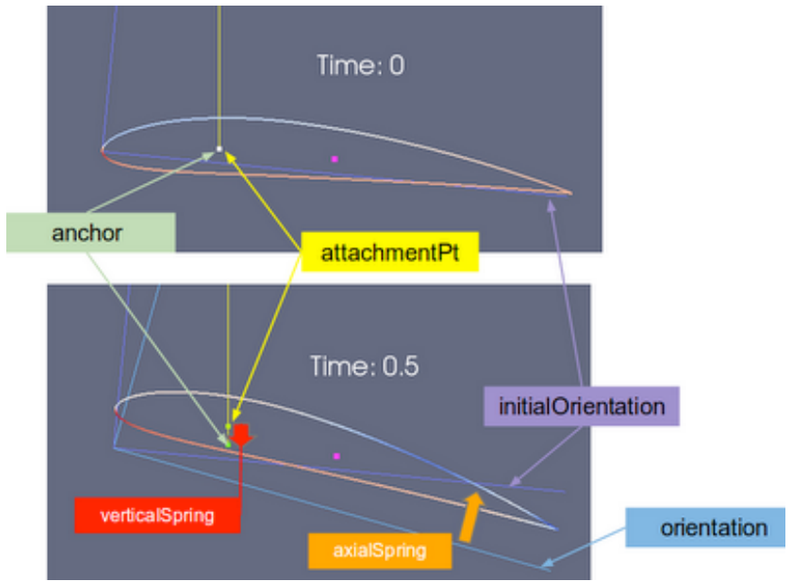


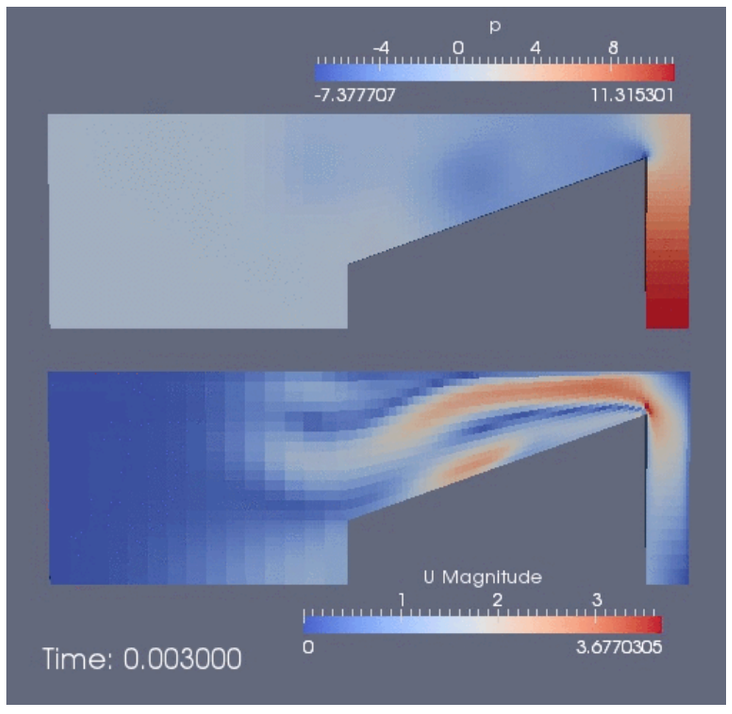
図出典
[OFF]

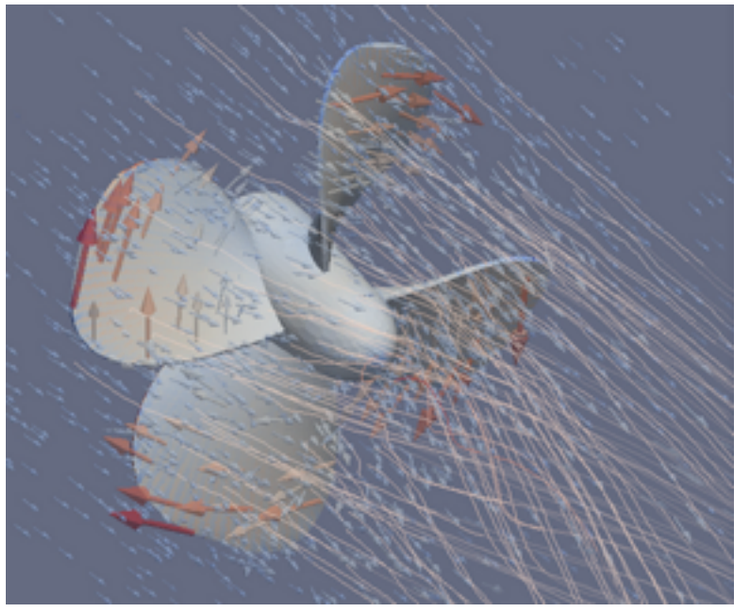
移動格子のチュートリアル例

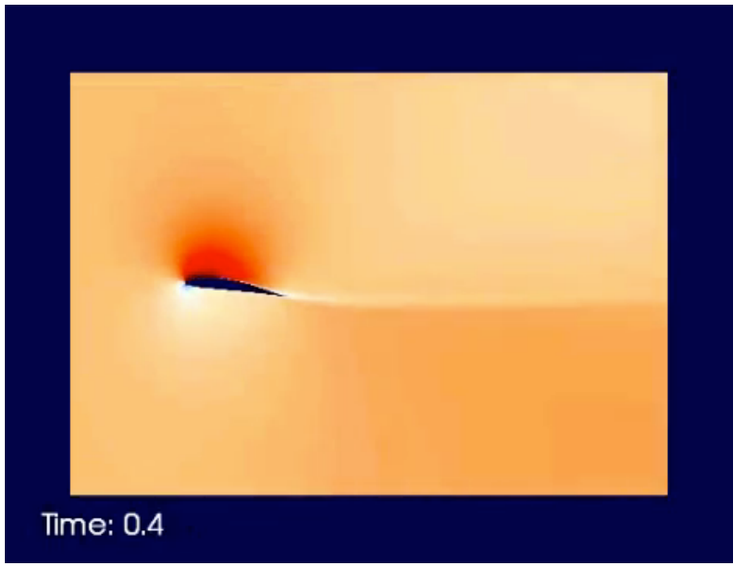












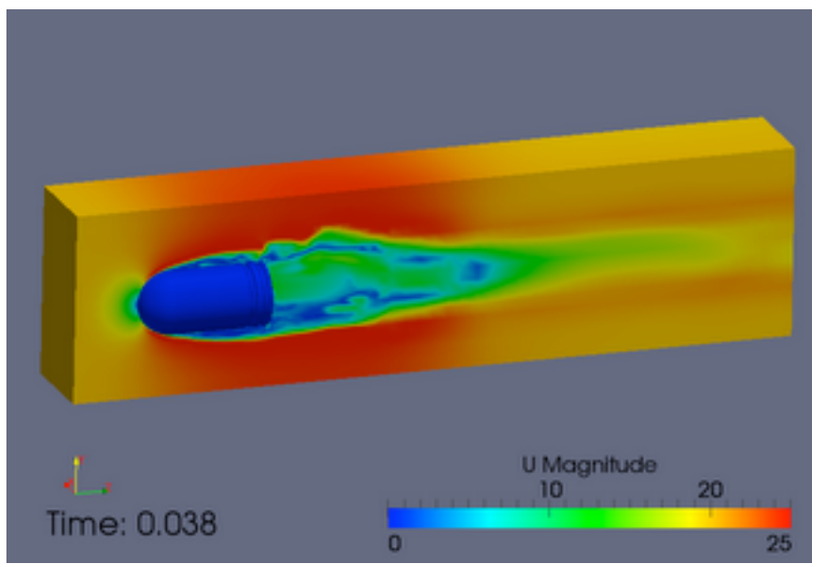
ピストン押し込み流れ
movingCone

スクリューの回転流れ場
propeller

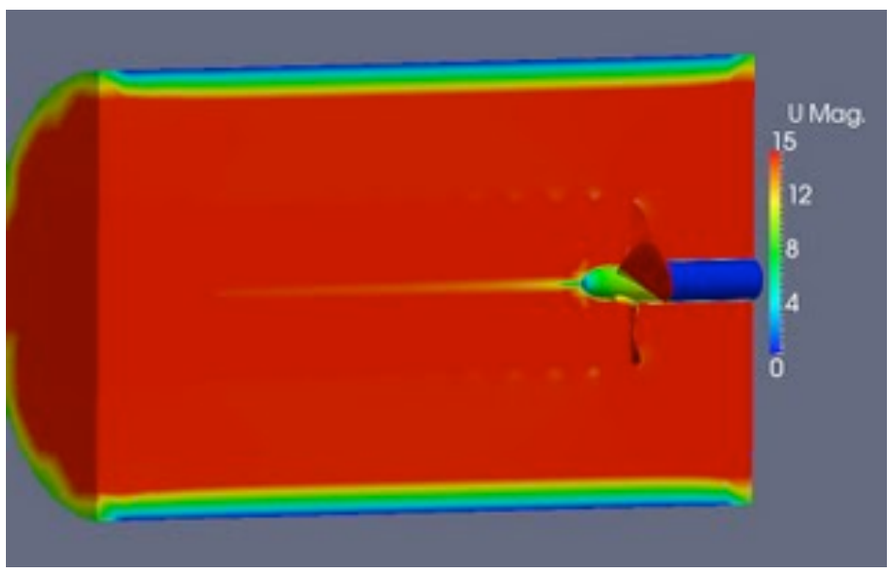
翼型の6自由度剛体運動
wingMotion

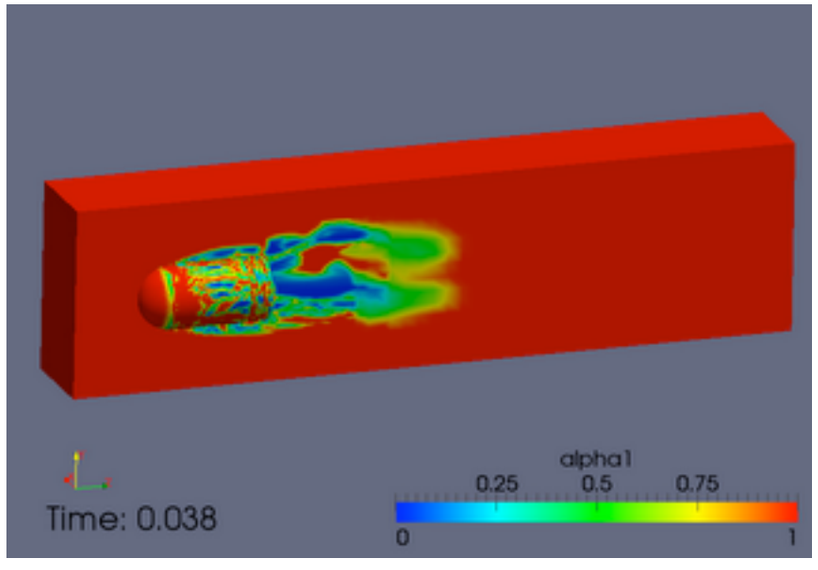
図出典 [OFT]

相変化のチュートリアル例

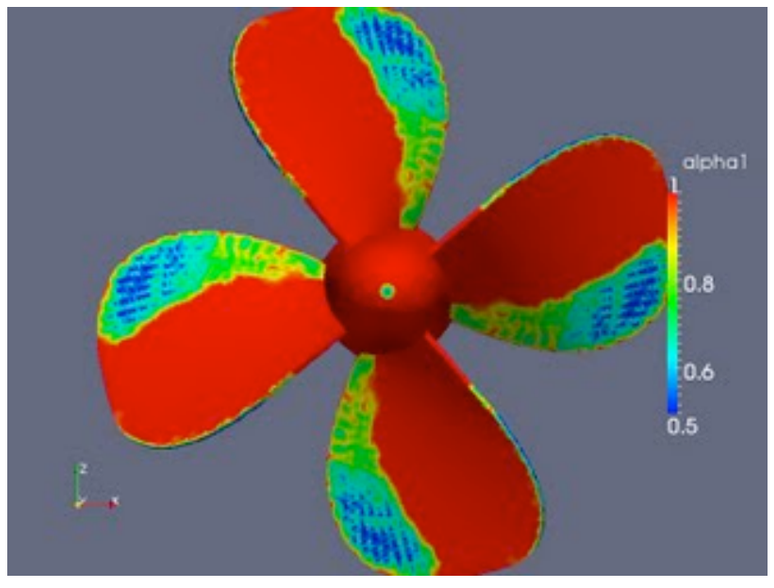


速度





相率



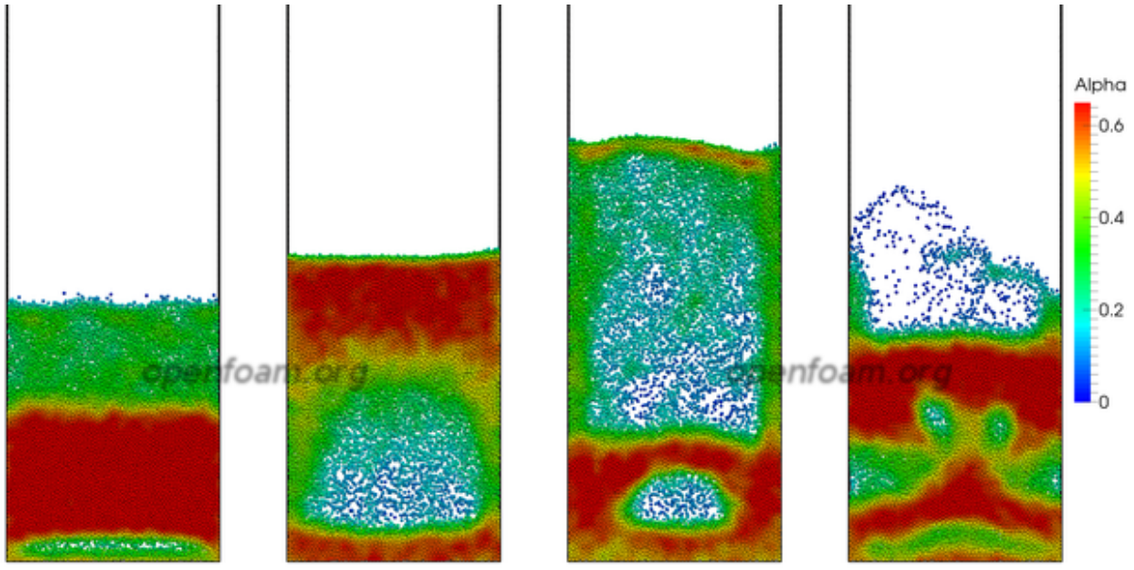
弾丸周りのキャビテーション
cavitatingBullet

プロペラ周りのキャビテーション
propeller

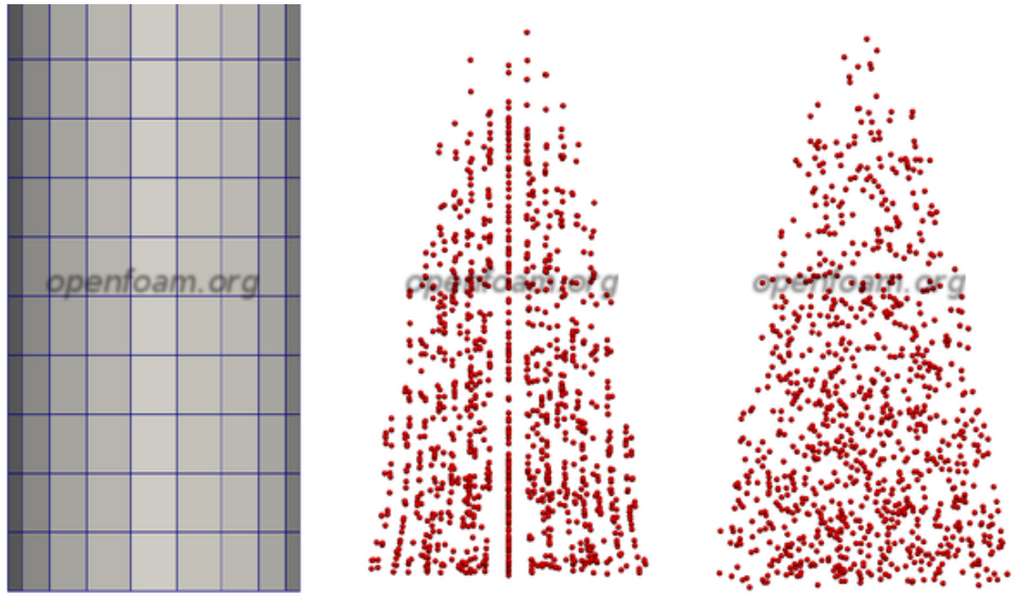
図出典 [OFT]

粒子計算のチュートリアル例

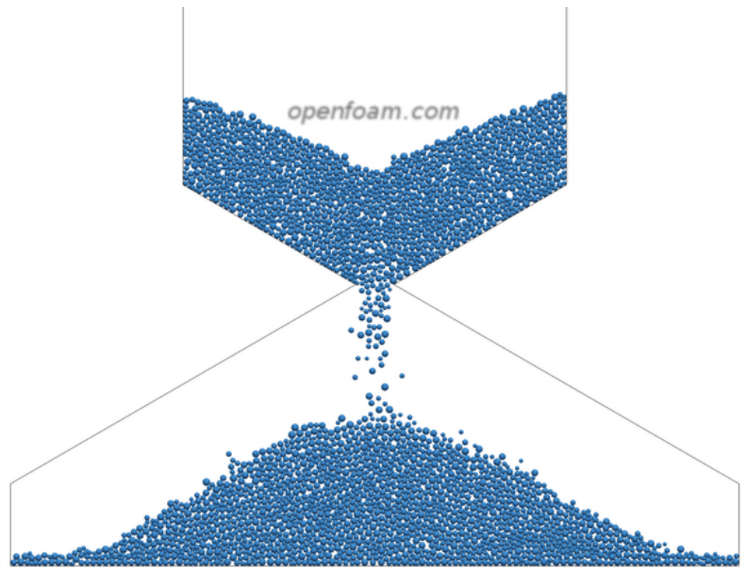
図出典： [OFF]



Multiphase Particle-in-Cell
(MP-PIC)

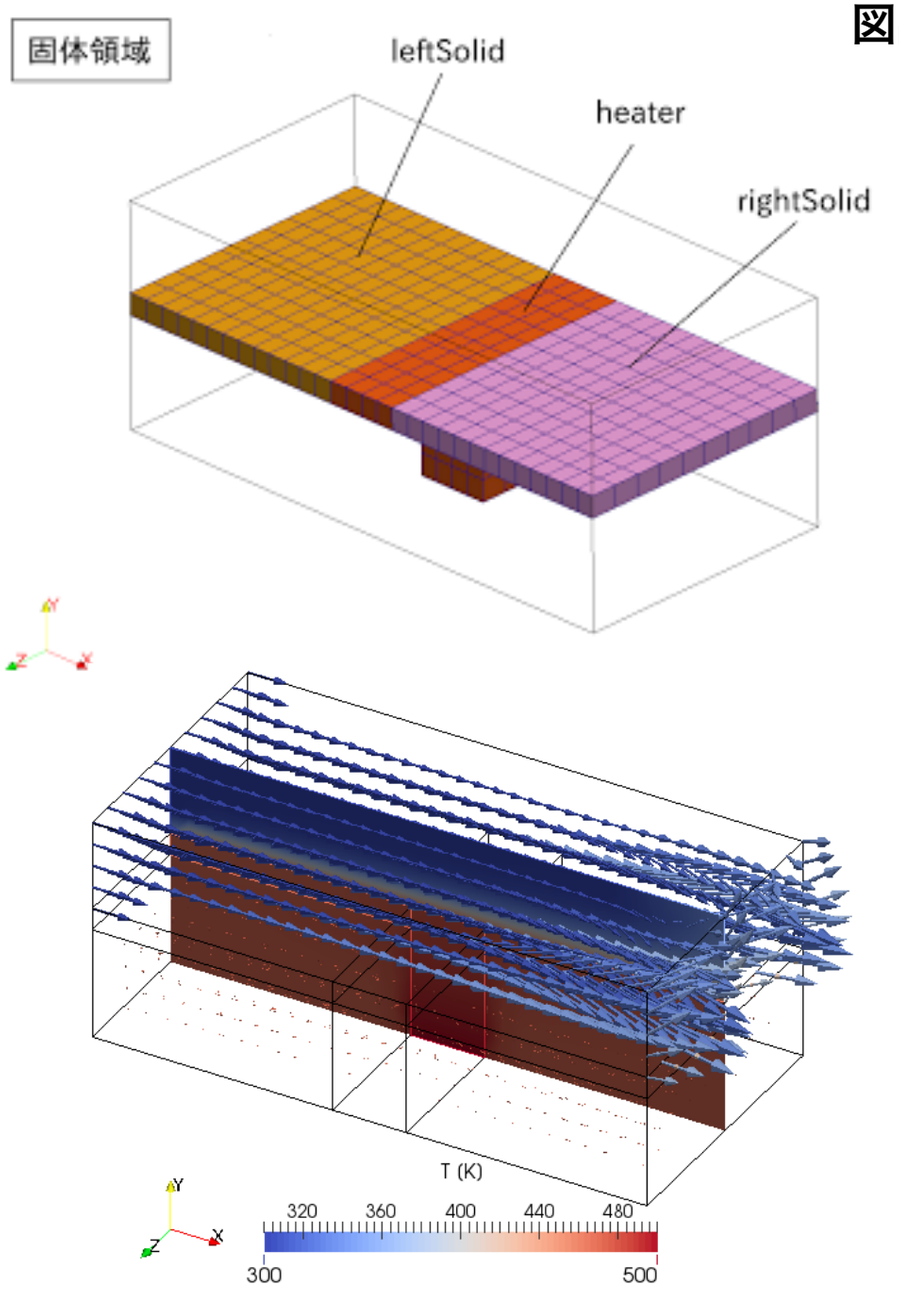


Particle Tracking



Discrete Element Modeling
(DEM)

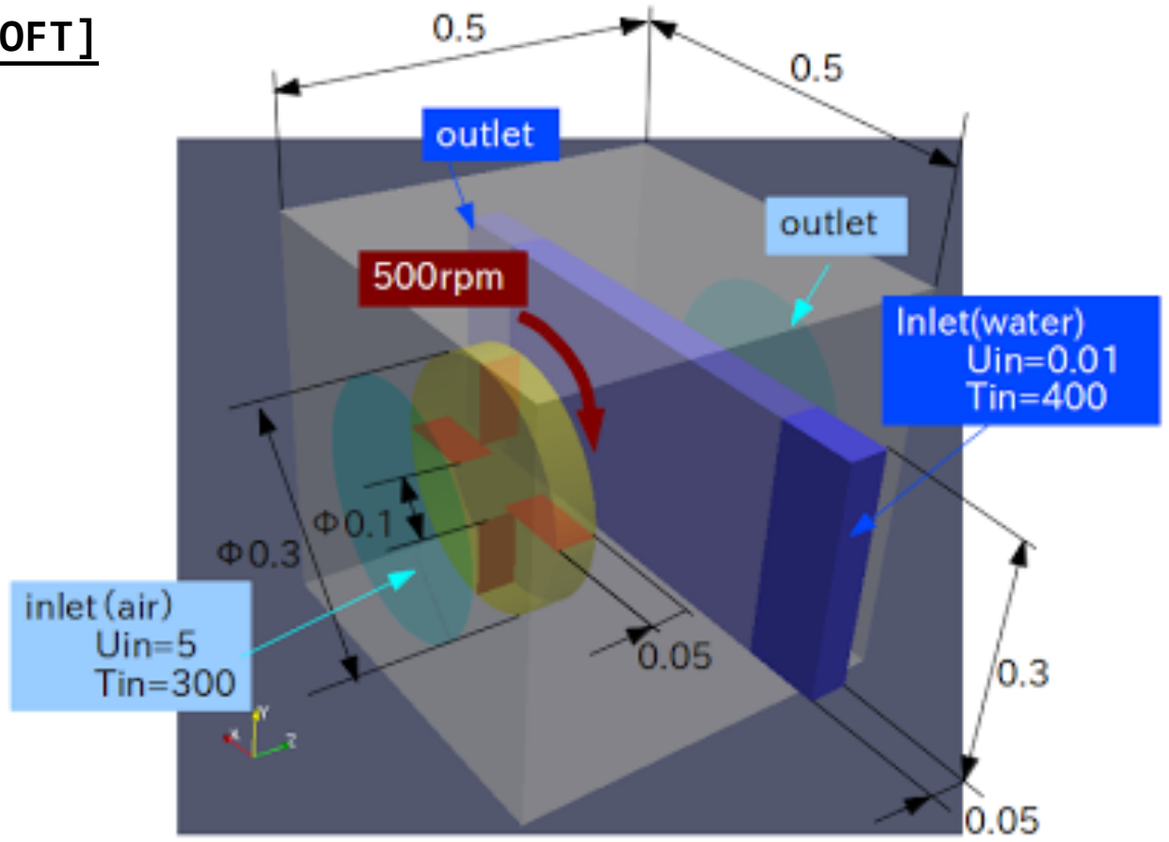
連成熱伝達解析(CHT)のチュートリアル例



連成熱伝達解析

multiRegionHeaterRadiation

図出典 [OFT]



回転する熱交換機解析(MRF)

heatExchanger

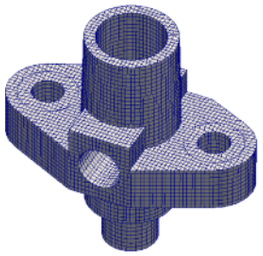
OpenFOAMのCHT解析における欠点

- 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]

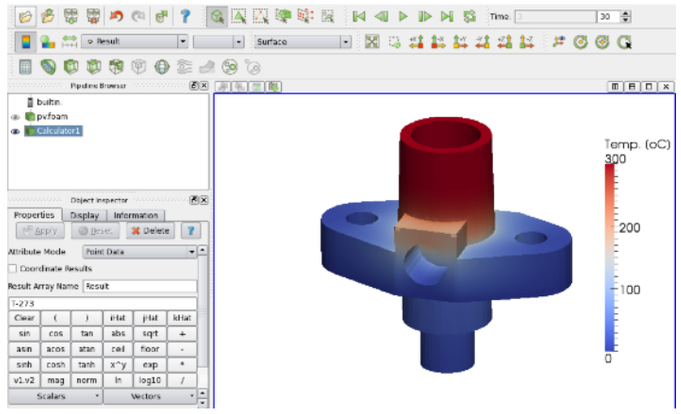
領域分割(並列計算時)
[decomposePar]

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

FOCUS
東工大 TSUBAME
東大 Reedbush-U, H, L
名古屋大学FX100
JCAHPC Oakforest-PACS
九州大学ITO
大阪大学 OCTOPUS，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**

  ✓ **HELYX(Engys):** OF拡張版+GUI

  ✓ **iconCFD(IDAJ, ICON):** OF拡張版+GUI

  ✓ **Visual-CFD(ESI):** GUI

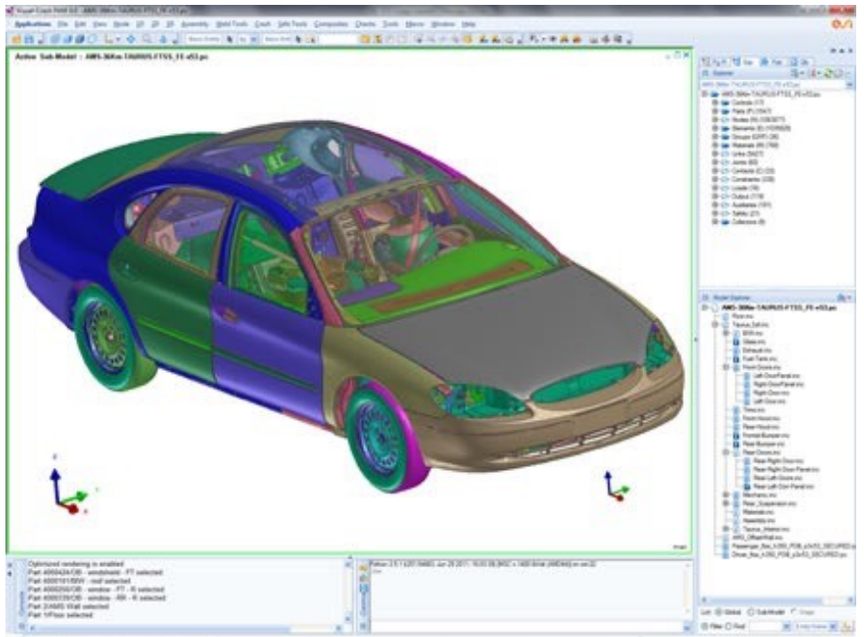
- **オープンソース版**

  ✓ **HELYX-OS(Engys):** GUI

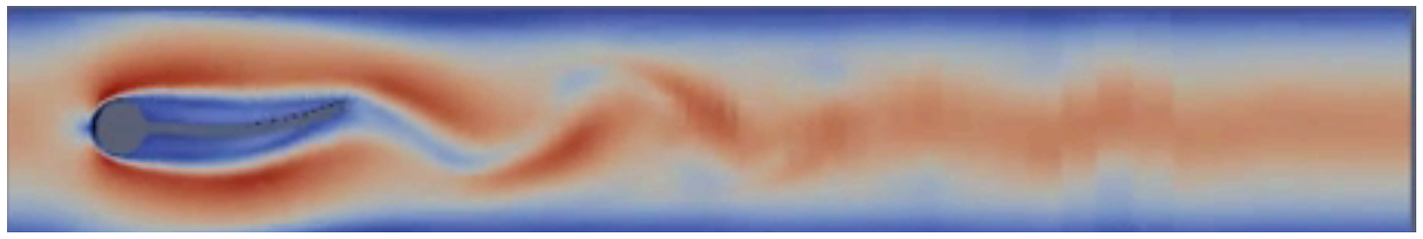
  ✓ **foam:** Hrvoje Jasak(クロアチア ザグレブ大学教授, Wikki社 代表)が主導するコミュニティベース版. FSIやBlock coupledソルバ等の公式版に無い機能を実装

  ✓ **OpenFOAM+(ESI):** 安定化と機能拡張

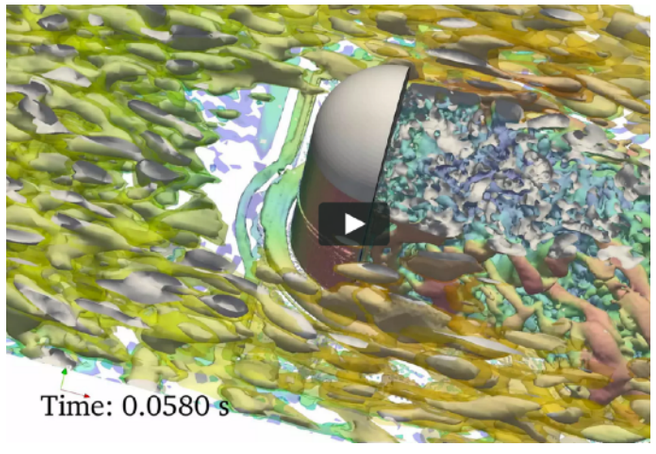
図出典：ESI (https://www.esi.co.jp/news/2014/PressRelease_0128.html)



Visual-CFD(ESI)



foamの 流体・構造連成(FSI)

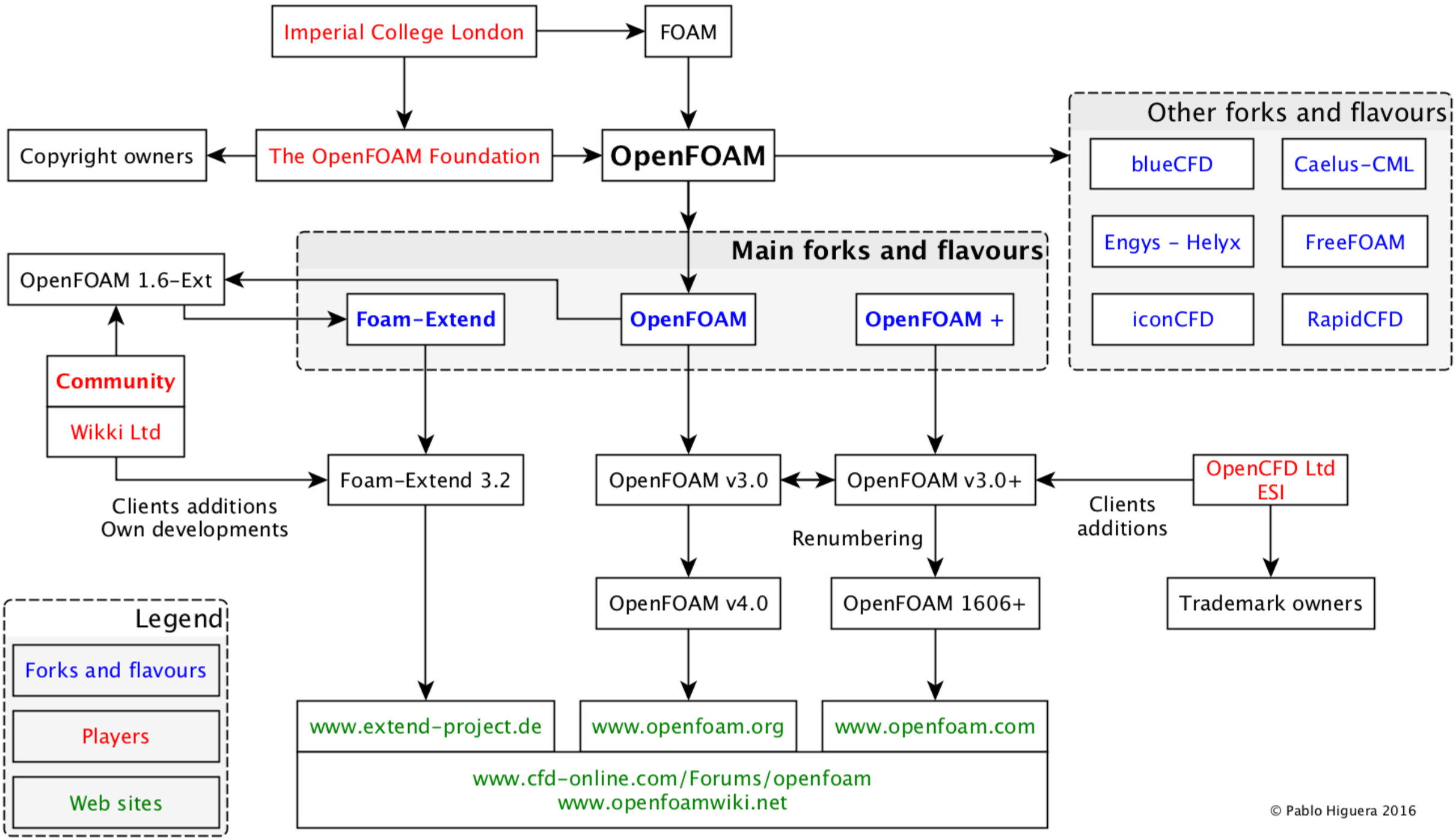


OpenFOAM+の変動風生成

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

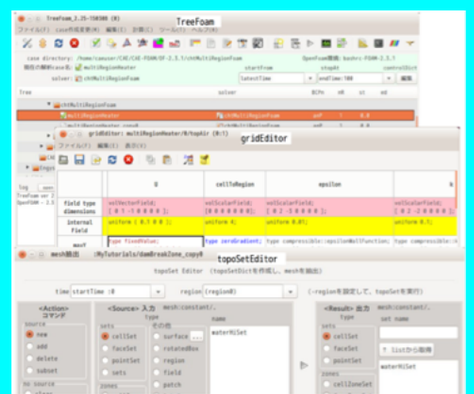
- 設定用GUIが無く，全てテキストファイルで設定する必要があるが，詳細な公式マニュアルがほとんど無い → ソースコードを読まないと詳細な設定方法がわからないので，初心者には設定が困難．解析条件に応じた推奨設定も不明

▼

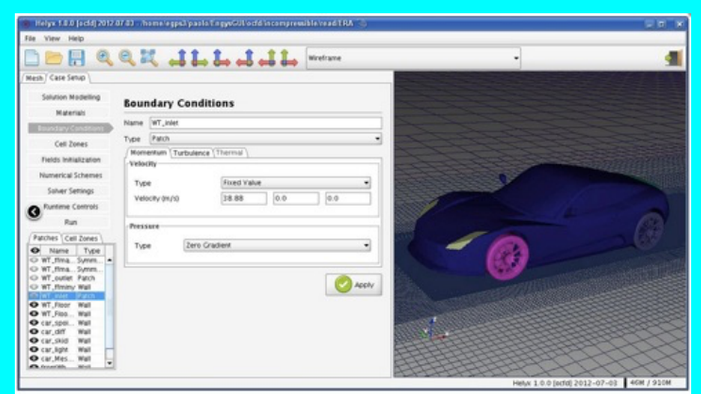
- GUIに関しては商用(iconCFD, Visual-CFD)，オープンソース(HelyxOS, DEXCS, TreeFoam)などが続々登場してきた

オープンソースGUI例

DEXCS (野村氏作)



TreeFoam(藤井氏作)



HelyxOS(engys社)

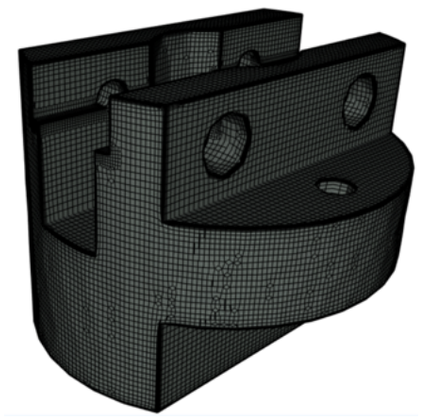
OpenFOAMの課題

- 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

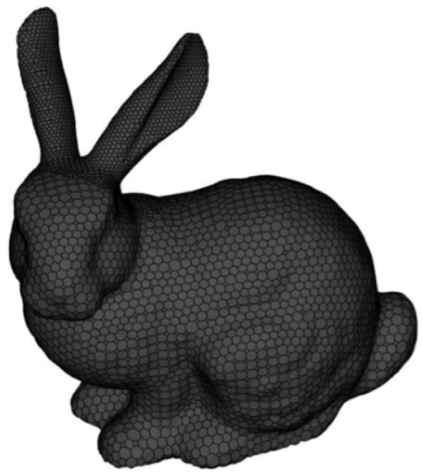
- Pointwise，HEXPRESSなどの商用メッシャーはOpenFOAMの格子を出力できるようになっている.
- Helyx, iconCFDなどの商用ForkではsnappyHexMeshの機能を改善
- オープンソースでハイブリッド並列，レイヤ付加性能に優れたOpenFOAM用メッシャーcfMeshも登場(v1712に取り込まれた)

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．

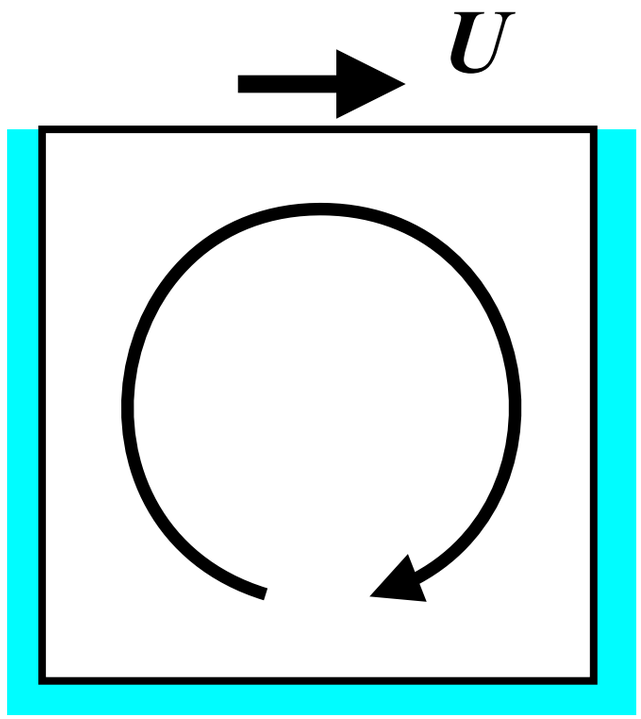
以下のような様々な研究や実装が行われている

- 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

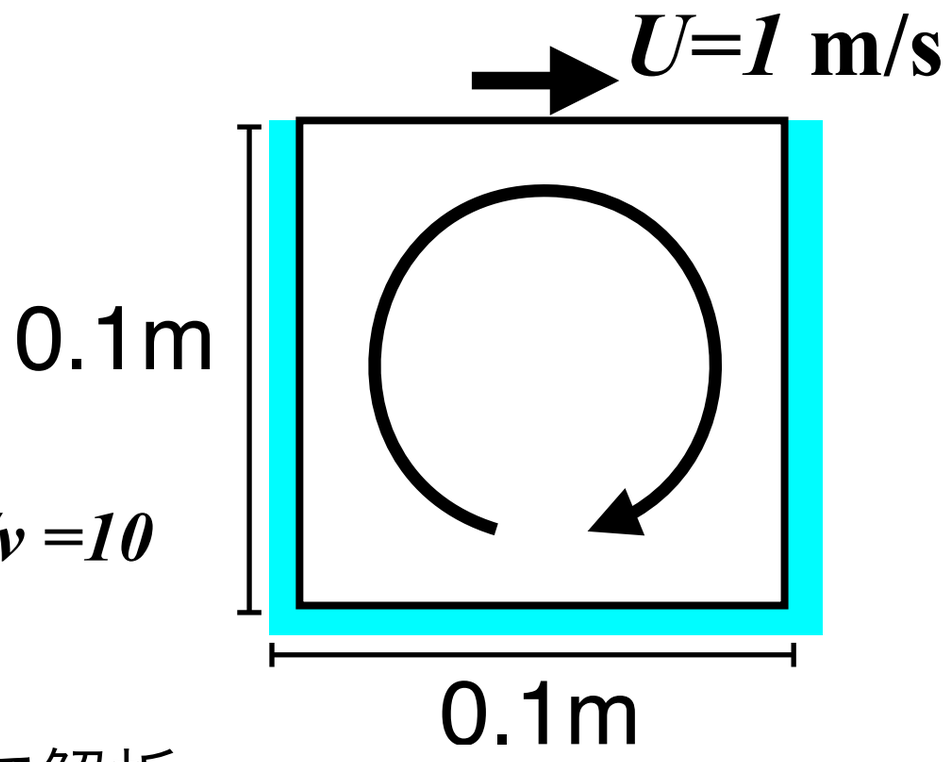


[Ghia 1982] U Ghia, K.N Ghia, C.T Shinl: High-Re solution for incompressible flow using the Navier-Stokes equations and the multigrid method. J. Comput. Phys., 48:387–411, 1982. URL

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

ユーザーマシン

4. 格子可視化
[ParaView]

7.可視化
[ParaView]

1. ログイン
[ssh]

3. 格子転送
[scpまたは
rsync]

6. 解析結果転送
[scpまたは
rsync]

ログインノード

皆で共有して使用しているので，通常，並列実行や負荷が大きい計算は不可

2.格子生成
[blockMesh]

5.流体解析
[ソルバ]

計算ノード

バッチジョブシステムによりジョブが実行される

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

OCTOPUSのログインノードへのログイン (X転送する-Yオプションを付ける)

```
ssh -Y xxxxxx@octopus.hpc.cmc.osaka-u.ac.jp  #xxxxxはユーザ名
```

X転送が有効になっておりgnuplotによるプロットが可能か調べる

```
gnuplot -p -e 'plot sin(x)' #別画面が出現すれば成功．別画面は消して良い
```

計算用ディレクトリへの移動

```
cd /octfs/work/$(id -gn)/$(id -un) #$(id -gn):グループ名，$(id -un):ユーザ名
```

講習会用ファイルのコピー

```
cp -a /octfs/apl/kosyu/OpenFOAM20180727/lecture ./
```

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 /octfs/work/$(id -gn)/$(id -un)/lecture ~/lecture
```

ログインノードでのOpenFOAM-4.1の環境設定(ログインやシェルを起動する度に実行)

```
source /octfs/apl/OpenFOAM/4.1/OpenFOAM-4.1/etc/bashrc
```

キャビティケースのコピー

cavityのケースのコピー

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

ケースディレクトリのディレクトリ構成を表示

```
find | sort
```

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

キャビティケースのディレクトリ構成

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

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

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

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

キャビティケースの解析手順

1. 格子生成

`blockMesh`

0/
- U — 速度ベクトル場
- p — 圧力場

constant/
- transportProperties — 流体物性

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

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

解析結果の時刻ディレトリ/
- U, p, phi — 解析結果

2. ソルバ実行

`icoFoam`

この他にも可視化等のポスト処理が通常含まれる

読み込み

読み書き

新規作成

fileName
既存ファイル

fileName
修正ファイル

fileName
作成ファイル

blockMeshによる格子生成

ユーザーマシン

4. 格子可視化
[ParaView]

7.可視化
[ParaView]

1. ログイン
[ssh]

3. 格子転送
[scpまたは
rsync]

6. 解析結果転送
[scpまたは
rsync]

ログインノード

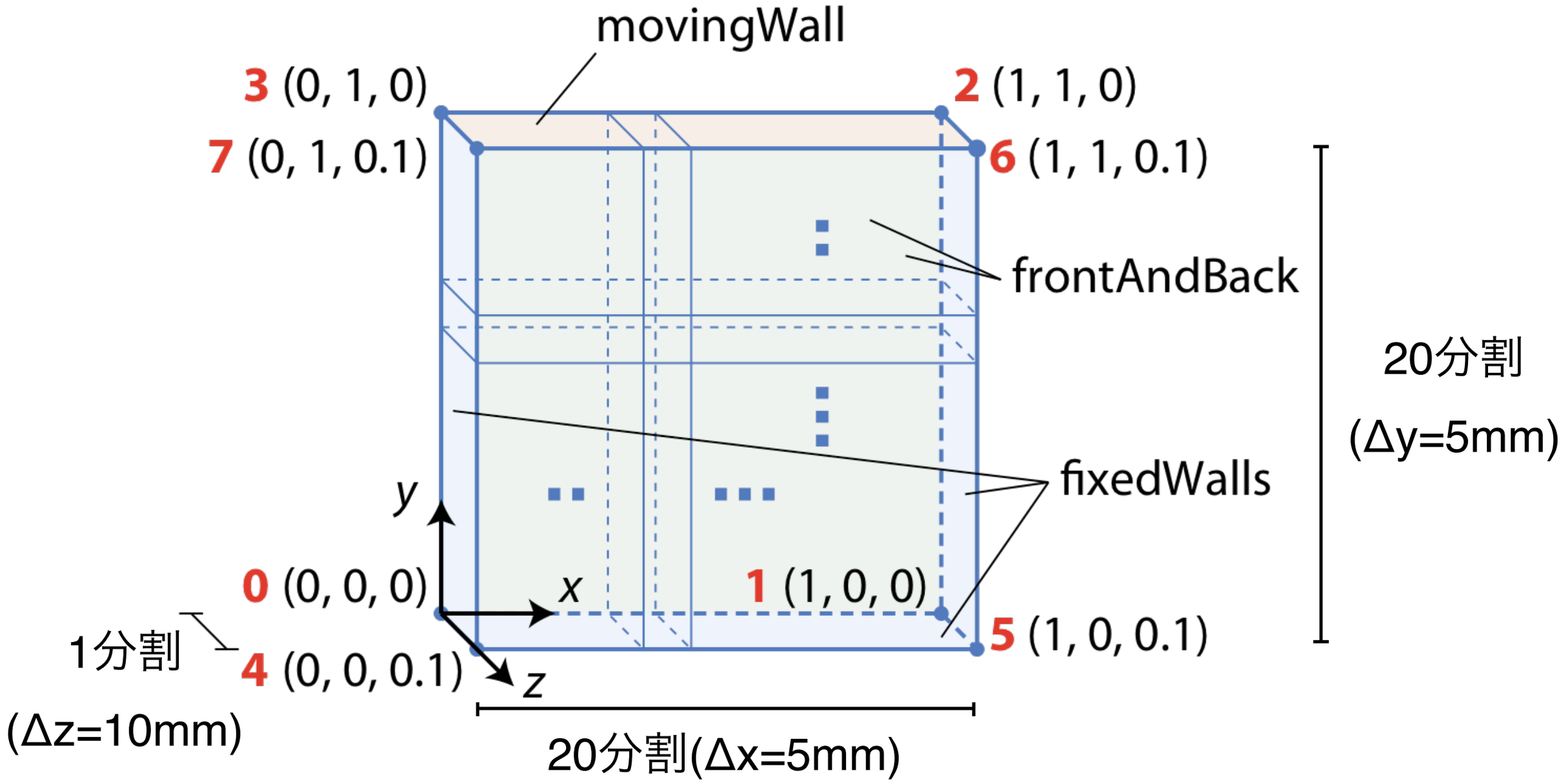
2.格子生成
[blockMesh]

5.流体解析
[ソルバ]

計算ノード

バッチジョブシステムによりジョブが実行される

キャビティケースの格子分割



注)2次元なので、z方向は1分割(幅は任意)

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

blockMeshDictの確認

```
more system/blockMeshDict
```

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  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

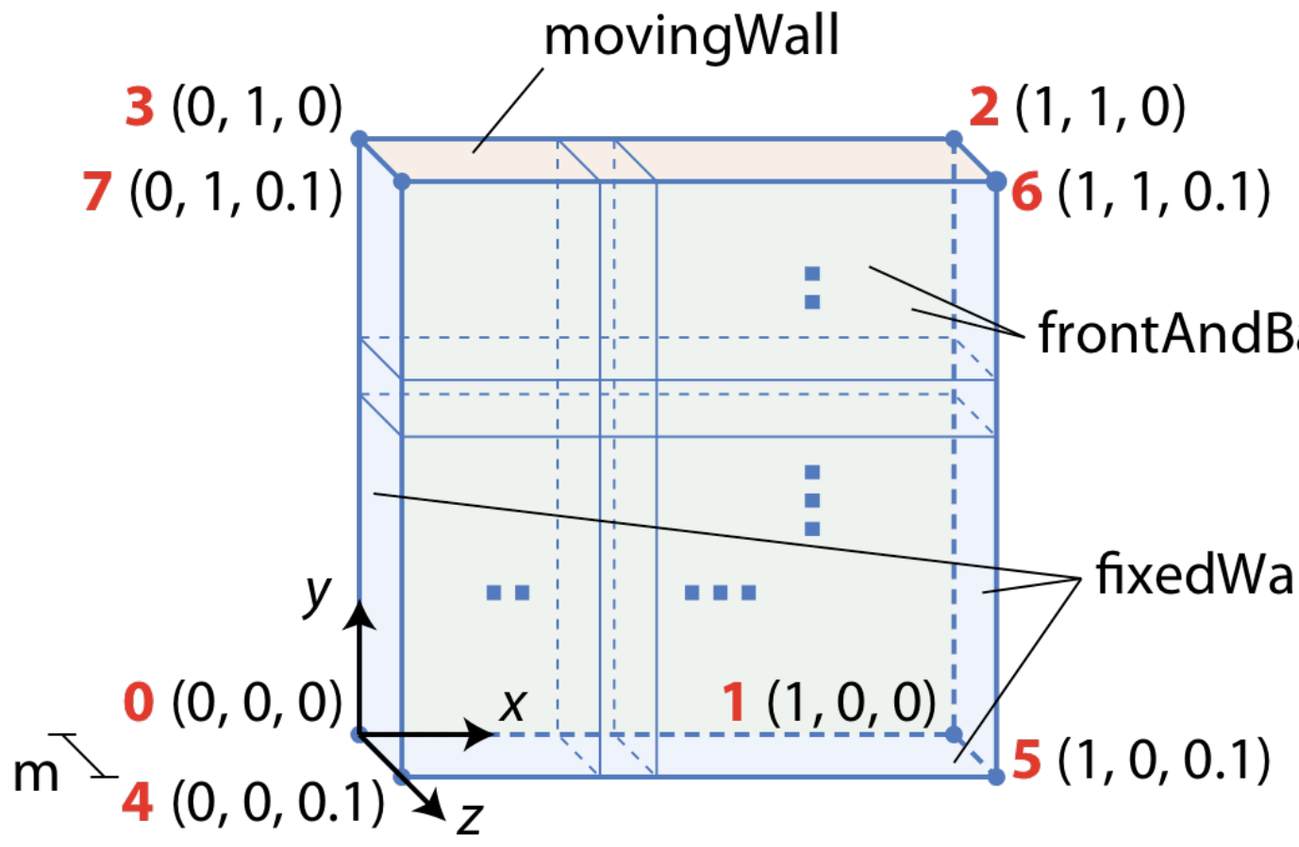
moreコマンドは続ける(英語版ではmore)と表示してキー入力待ちとなる．

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

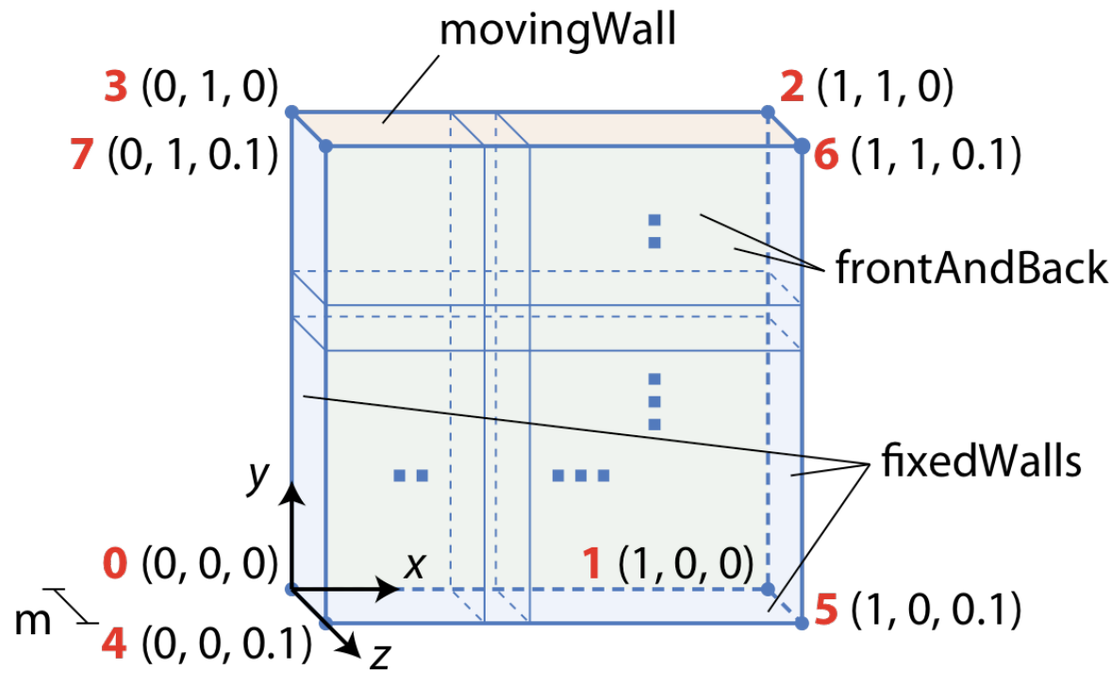
```
convertToMeters 0.1; //メートル単位への変換係数

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 (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
(
);
```

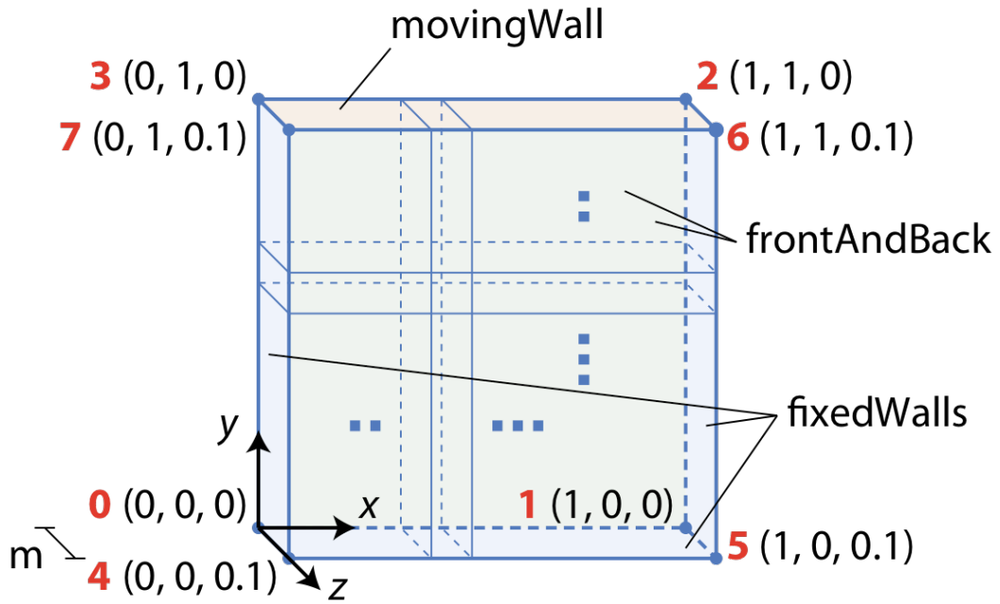


設定ファイルblockMeshDict (続き)

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

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/* ./
```

```
more blockMesh.sh
```

```
#!/bin/bash
#PBS -q OCTPHI
#PBS -l elapstim_req=0:10:00
#PBS -l cpunum_job=1
#PBS -b 1

# ジョブ投入時のディレクトリに移動
cd $PBS_O_WORKDIR
# OpenFOAM-4.1の環境設定
source /octfs/apl/OpenFOAM/4.1/OpenFOAM-4.1/etc/bashrc
# 念のため実行時の環境変数を記録
env
# blockMeshの実行
blockMesh >& $PBS_JOBNAME.l${PBS_JOBID#*:}
# end
```

ジョブクラス(キュー)名 **(講習会中は専用キューのLECTURE)**
経過時間制限([[hour:]minite:]second[.fraction])**(注)**
ノード毎のジョブ数(フラットMPIの場合，MPIプロセス数)
ノード数．非並列計算なので，MPIプロセス数とノード数は1

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，適切な経過時間制限値を設定するほうが良い

ジョブ実行時の標準/エラー出力のファイルはジョブ完了後に作成され，100MBの容量制限があるので，重要コマンドの出力はログファイル(名前は任意)にリダイレクトすると良い

ログファイル名=ジョブ名.lリクエストID
($PBS_JOBID=ジョブ番号:リクエストID.ホスト名)

blockMeshのジョブ投入

ジョブの投入

```
qsub blockMesh.sh
```

```
Request RequestID.oct submitted to queue: XXXXX. #RequestID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う)

```
sstat #qstatというコマンドもあるが，sstatは開始予定時間がわかるので便利
```

→ 実行待ち状態(STT=QUE)

```
RequestID       ReqName  UserName Queue                  Pri STT PlannedStartTime
--------------- -------- -------- -------- ---------------- --- -----------------
RequestID       blockMes xxxxxx   OP1C       0.5002/  0.5002 QUE -
```

→ リソースが割り当てられてた状態(STT=ASG，開始予定時間が表示される)

```
RequestID       blockMes xxxxxx   OP1C       0.5002/  0.5002 ASG 1970-01-01 00:00:00
```

→ 実行中(STT=RUN)

```
RequestID       blockMes xxxxxx   OP1C       0.5002/  0.5002 RUN Already Running...
```

ジョブの削除(今回は行わない)

```
qdel RequestID
```

生成されたファイルの確認

```
find | sort | more
```

```
.
./0
./0/U
./0/p
./blockMesh.sh.[elo]123456*
./constant
./constant/polyMesh
./constant/polyMesh/boundary
./constant/polyMesh/faces
./constant/polyMesh/neighbour
./constant/polyMesh/owner
./constant/polyMesh/points
./constant/transportProperties
./system
./system/blockMeshDict
./system/controlDict
./system/fvSchemes
./system/fvSolution
```

123456: リクエストID
*.e123456　　: ジョブの標準エラー出力
*.l123456.oct : blockMeshのログ
*.o123456　　: ジョブの標準出力

constant/polyMeshに格子データが生成される

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 2 (start: 780 size: 60) name: fixedWalls
  patch 3 (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

OpenFOAMのアプリケーションは通常終了時Endを出力する(例外もある)

ParaViewによる格子可視化

ユーザーマシン

4. 格子可視化
[ParaView]

7.可視化
[ParaView]

1. ログイン
[ssh]

3. 格子転送
[scpまたは
rsync]

6. 解析結果転送
[scpまたは
rsync]

ログインノード

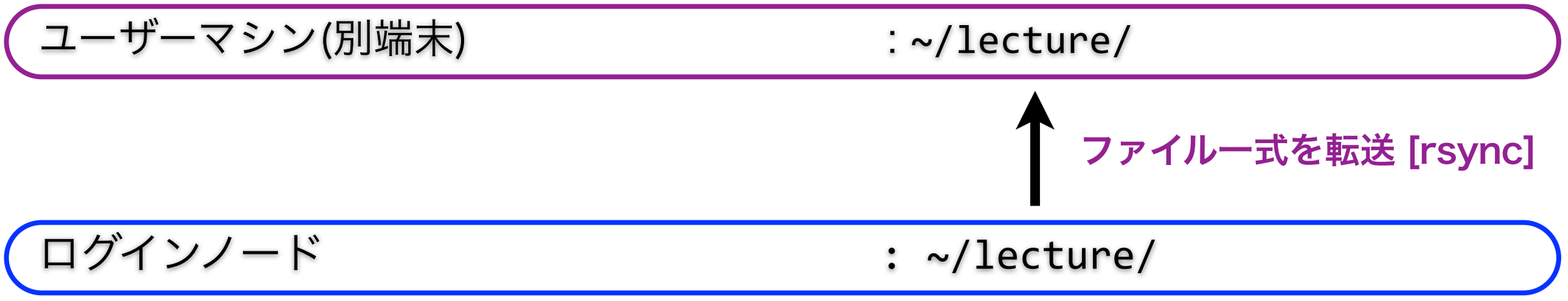
2.格子生成
[blockMesh]

5.流体解析
[ソルバ]

計算ノード

バッチジョブシステムによりジョブが実行される

講習用ファイル一式と格子データの転送



データ転送用にログインしている端末と別の端末を立ちあげる

講習用ファイル一式と作成した格子データの転送(別端末で実行)

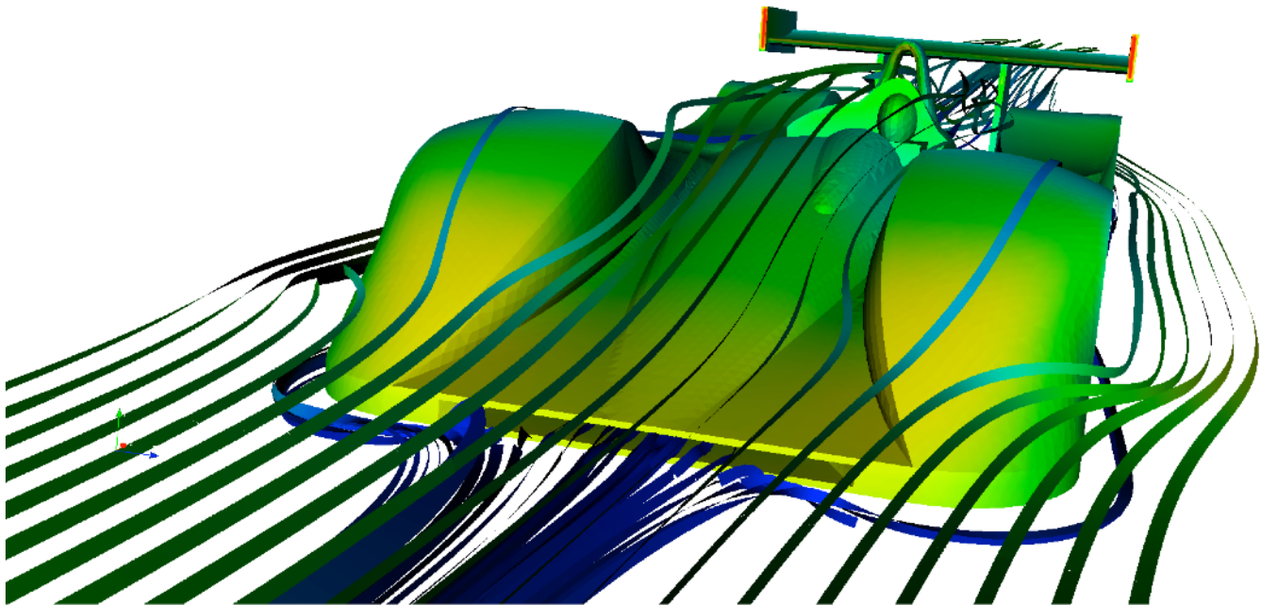
```
ls ~/
# 既に ~/lecture がある場合には，mv ~/lecture ~/lecture.orig などと別名にする
mkdir ~/lecture
rsync -auv xxxxxx@octopus.hpc.cmc.osaka-u.ac.jp:lecture/ ~/lecture/
# 転送元と転送先どちらもに/(スラッシュ)を付ける                ↑          ↑
# xxxxxxは利用者番号．パスワードを聞かれた場合には，登録したものを入力する
# a=archive(ディレクトリを再帰的かつ，ファイル情報を保持したまま転送),
u=update(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

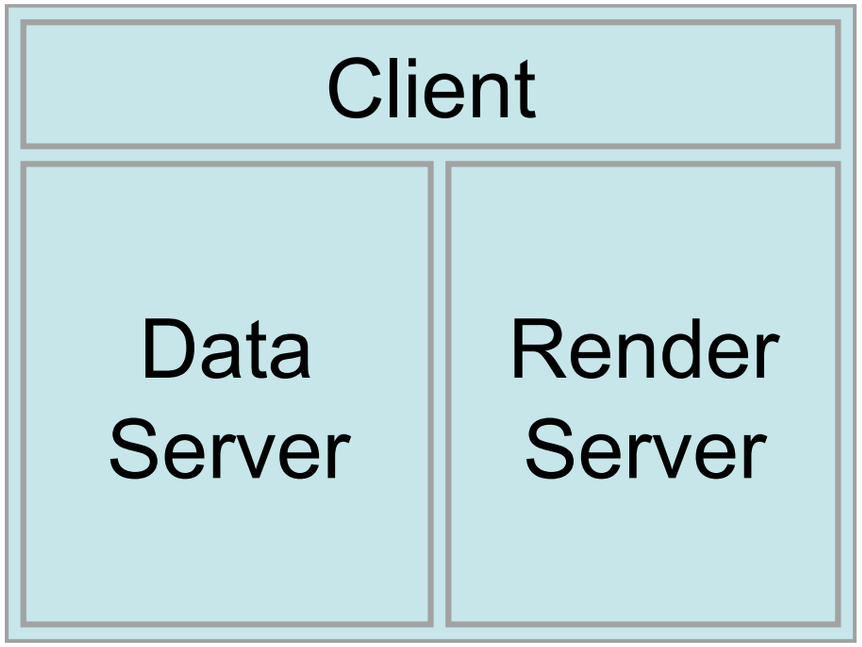


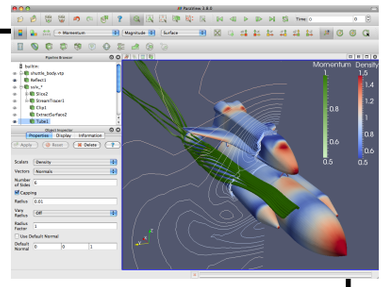
ル・マンのレースカー周りの気流

(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

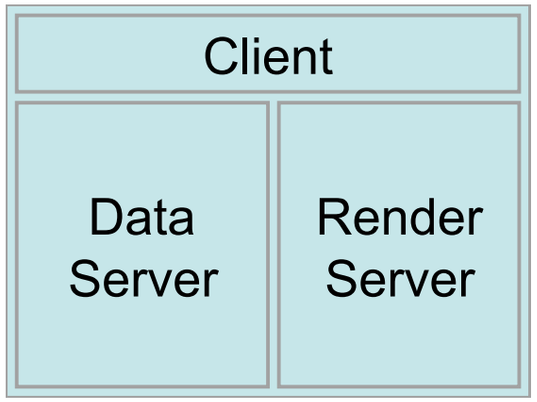
- データ・サーバ
  - ✓ データの読込み、フィルタリング、書出しを担当
  - ✓ 全てのパイプライン・オブジェクトはデータ・サーバが保持
  - ✓ 並列実行可能

- 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

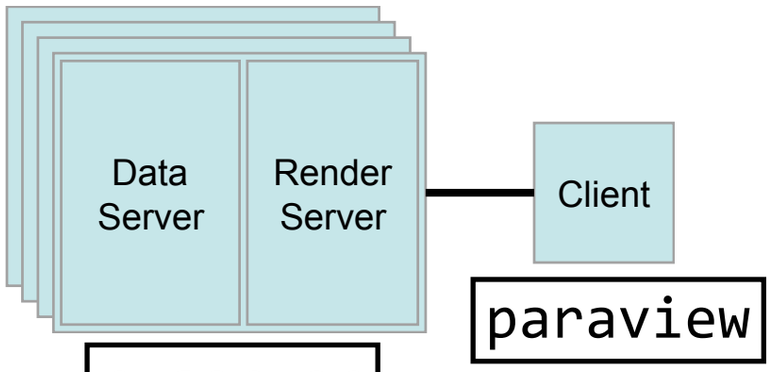
コマンド名



paraview

スタンドアローン・モード

- 全てが統合してシリアル動作

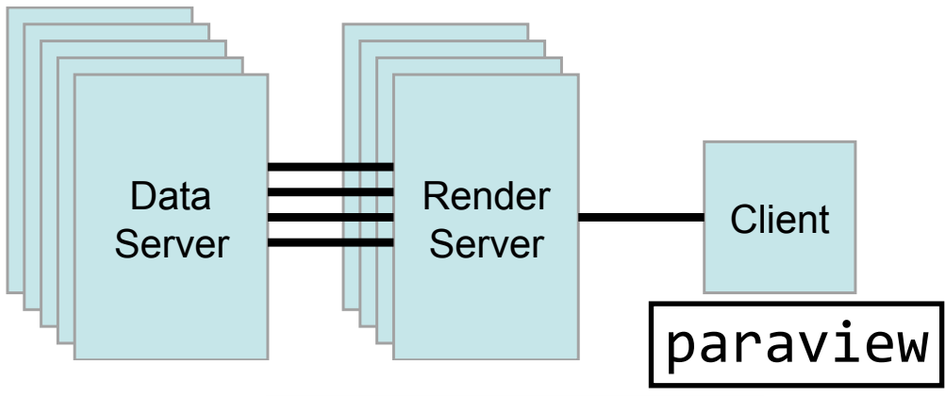


paraview

pvserver

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



paraview

pvdataserver　pvrenderserver

クライアント–レンダー・サーバ–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**

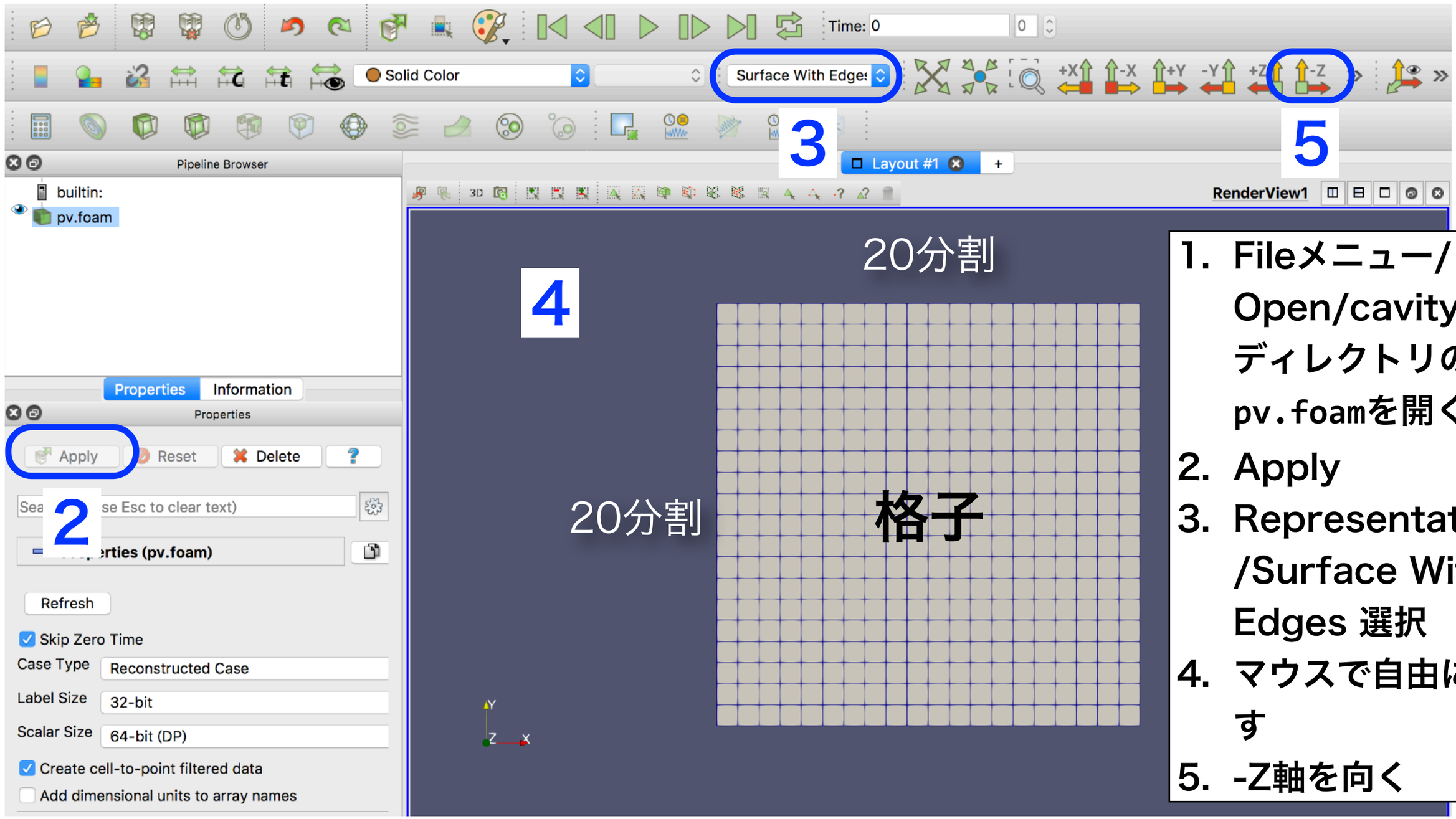
✓ **OpenFOAMデータ読み込み方法**

- **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

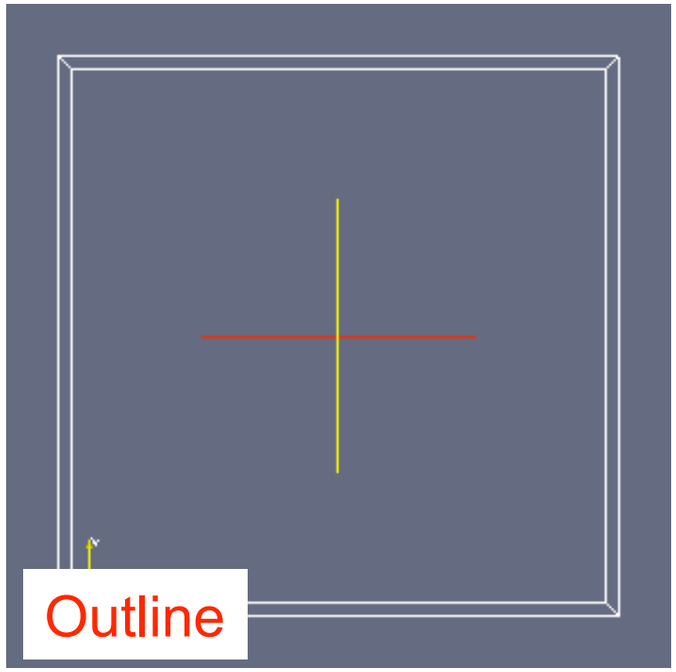
ParaView用ダミーファイル(.foam)の作成(別端末で実行)

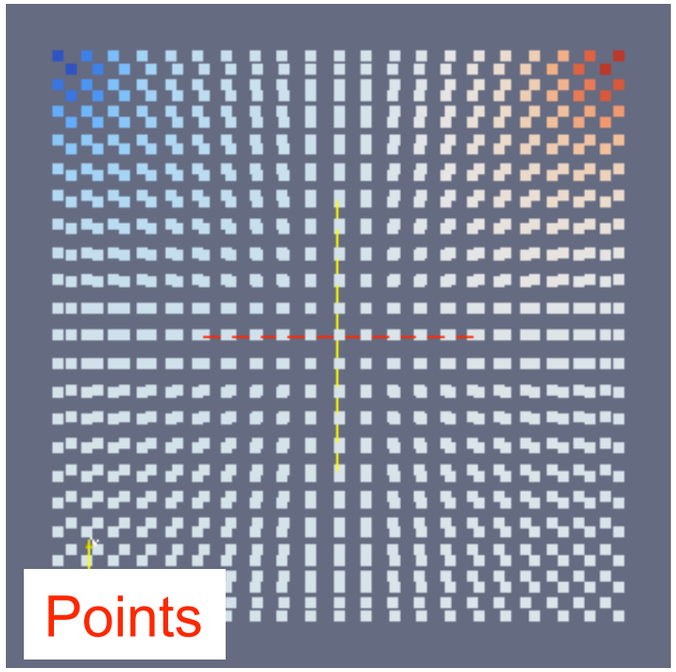
```
touch pv.foam
```

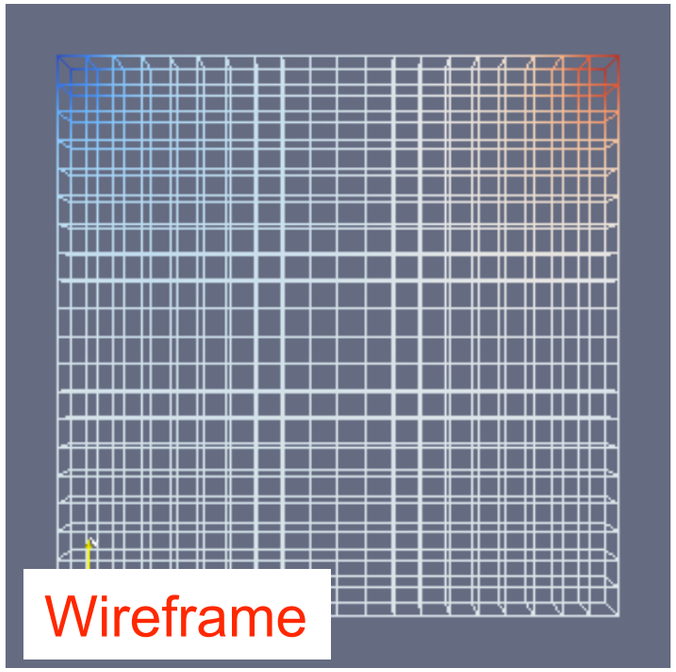


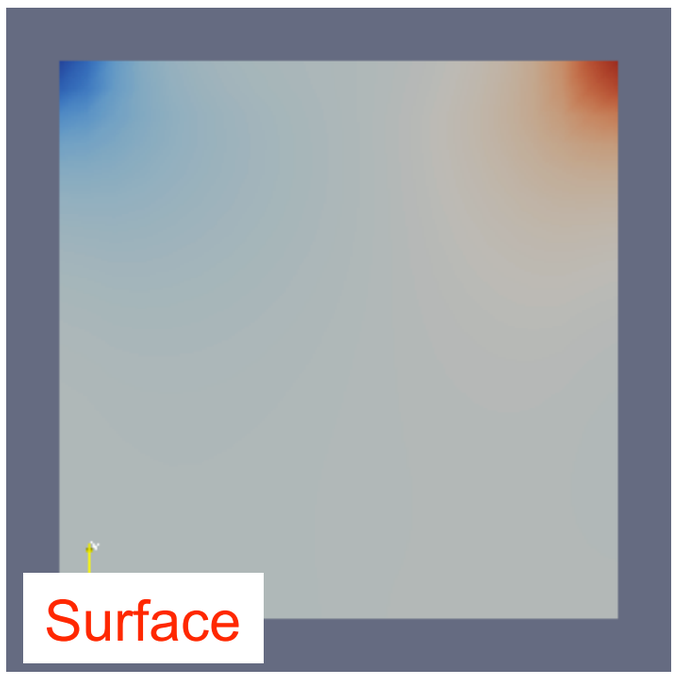
1. Fileメニュー/Open/cavityのディレクトリのpv.foamを開く
2. Apply
3. Representation/Surface With Edges 選択
4. マウスで自由に動す
5. -Z軸を向く

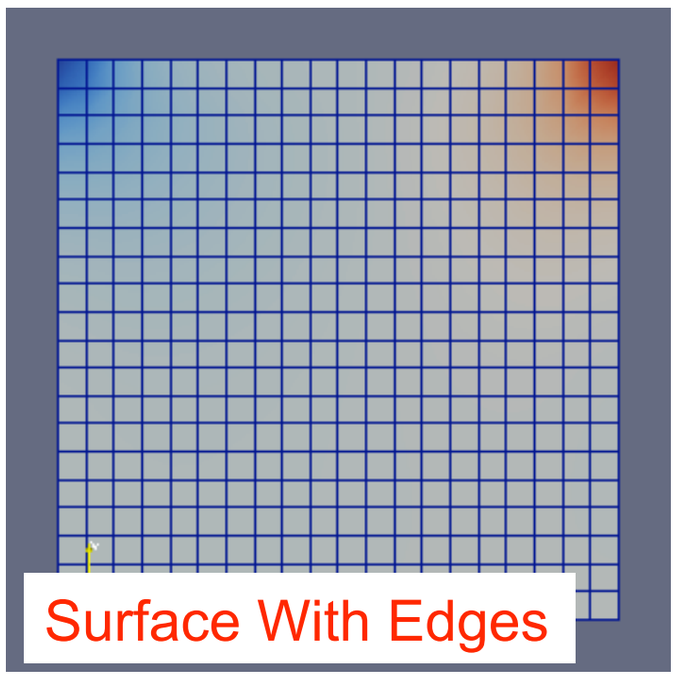
ParaViewの表示方法(Representation)

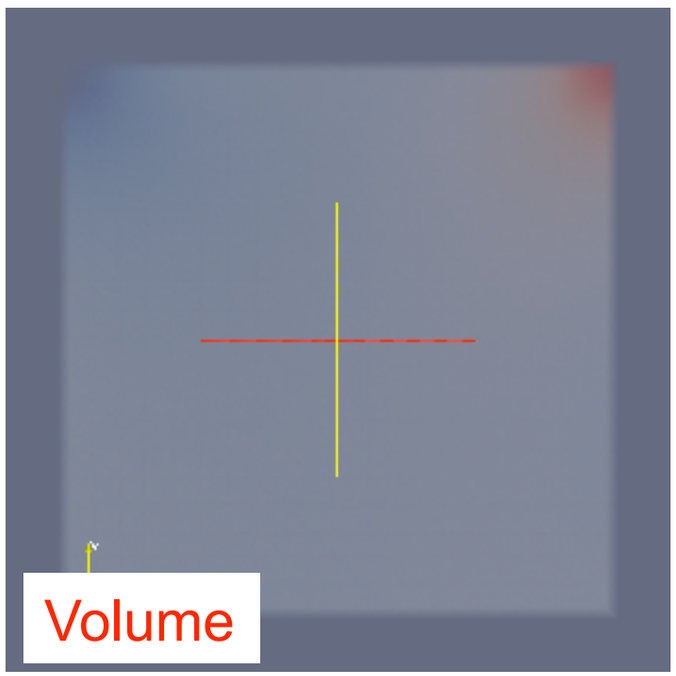












(図引用元: 大嶋 拓也 (新潟大学)「キャビティ流れの解析、paraFoamの実習」 第一回OpenFOAM講習会)

icoFoamによる流れ解析

ユーザーマシン

4. 格子可視化
[ParaView]

7.可視化
[ParaView]

1. ログイン
[ssh]

3. 格子転送
[scpまたは
rsync]

6. 解析結果転送
[scpまたは
rsync]

ログインノード

2.格子生成
[blockMesh]

5.流体解析
[icoFoam]

計算ノード

バッチジョブシステムによりジョブが実行される

初期条件・境界条件設定(速度)

0/ 初期条件・境界条件ディレクトリ

U 速度ベクトル

ファイルの確認

```
more 0/U
```

0/U

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

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

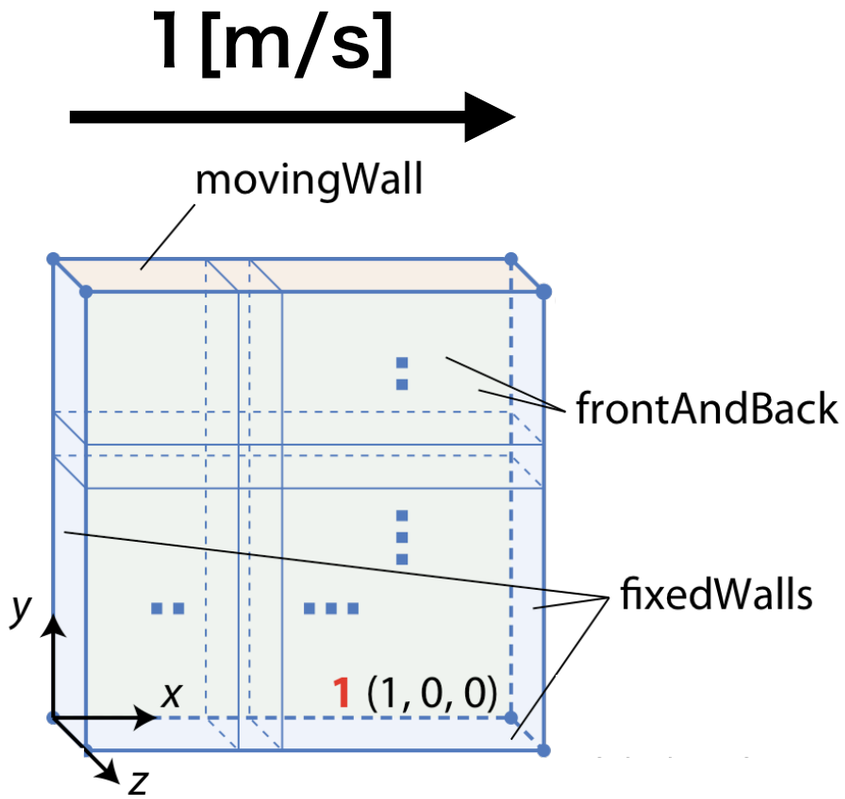
各演算では左辺・右辺での単位一致の検査がなされる

→カスタマイズ時に非物理的な演算の実装をチェックできる

初期条件・境界条件設定(速度)

0/U

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



初期条件・境界条件設定(圧力)

0/ 初期条件・境界条件ディレクトリ

p 圧力

ファイルの確認

```
more 0/p
```

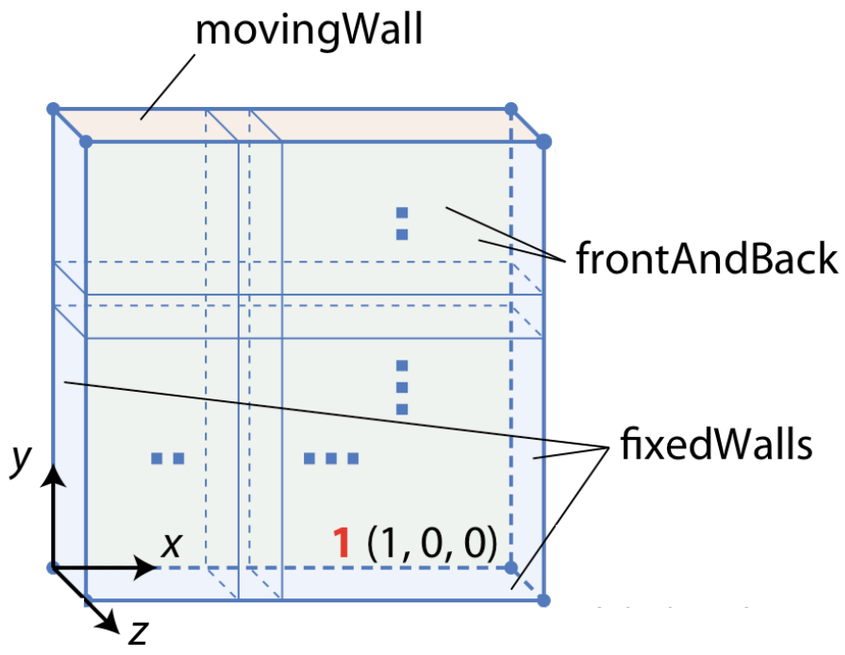
0/p

```
dimensions  [ 0    2   -2   0    0        0    0    ];
#単位の次元      質量 長さ 時間 温度 物質量     電流 光度
#SI単位での例  [kg] [m] [s]  [K]  [kgmol] [A]  [cd]
#OpenFOAMの非圧縮性ソルバでは、流体の密度で割った圧力を解いている
#(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
```

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

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

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
```

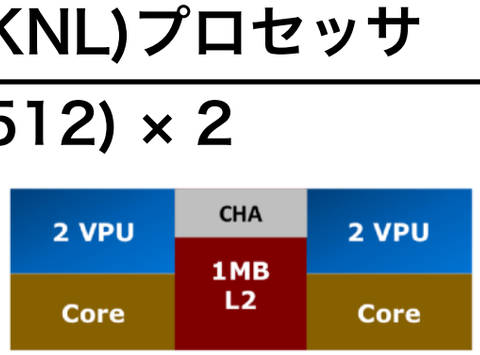
seq.sh（逐次実行ジョブ用スクリプト．blockMesh.shと異なる行のみ表示）

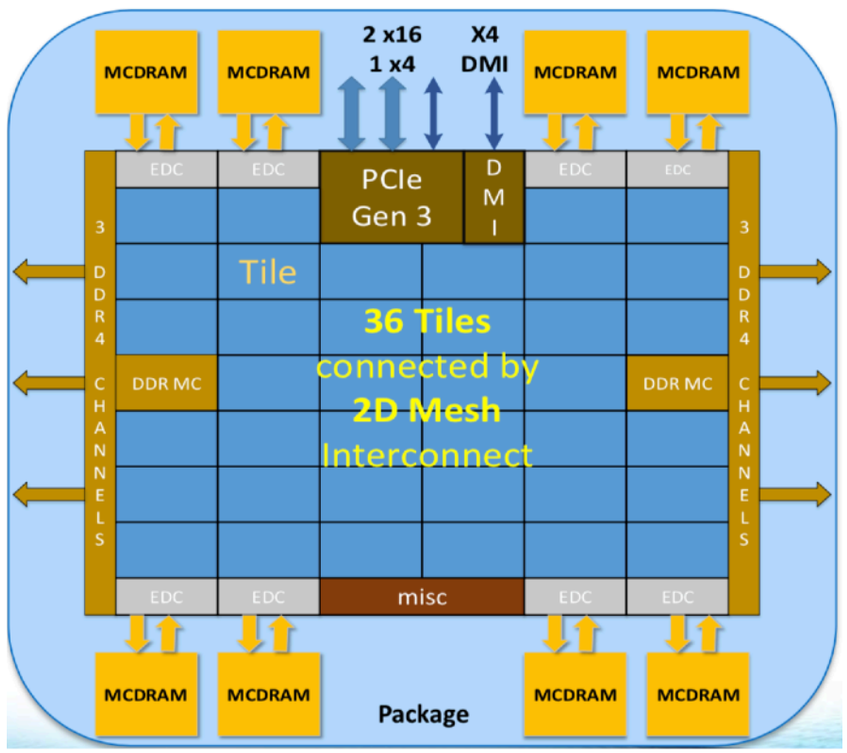
```
# icoFoamの実行
# numactl -p 1 : MCDRAMを優先的に使用する(Xeon Phiのみ必要)
# (numactl -m 1 : MCDRAMのみ使用．メモリ不足でエラーとなる)
numactl -p 1 \
icoFoam >& $PBS_JOBNAME.l${PBS_JOBID#*:}
```

Xeon Phi 7210(Knights Landing, KNL)プロセッサ

- コア：Atom(1.3GHz) + VPU(AVX512) × 2
- タイル：コア×2 + 共有L2キャッシュ
- プロセッサ：32タイル(64コア)
- メモリ
  - ✓ MCDRAM：16GB，400～GB/s(注，高バンド幅)
  - ✓ DDR4：192GB，90～GB/s(注，大容量)

図出典：Hotchips 27 (注) Stream Triadベンチマーク





ジョブの投入

```
qsub seq.sh
```

生成ファイルの確認

ファイルの確認(sstatでジョブ完了を確認後)

```
find | sort | more
```

新規に生成されたもの

./0.1	解析結果の時刻ディレクトリ
./0.1/U	速度ベクトル
./0.1/p	圧力
./0.1/phi	流束(フラックス)
./0.1/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
./0.1/uniform/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.e314672	ジョブの標準エラーファイル
./seq.sh.l314672.oct	ソルバのログ
./seq.sh.o314672	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  : 4.1
Exec   : icoFoam
Date   : Jan 01 1970
Time   : 00:00:00
Host   : "oct-xxxx"
PID    : xxxx
Case   : /octfs/work/x/x/lecture/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
```

ビルドバージョン
実行コマンド
開始日時
開始時刻
ホスト名
プロセスID
ケースディレクトリ
使用プロセッサ数

流体解析のログ(続き)

seq.sh.lRequestID

```
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)

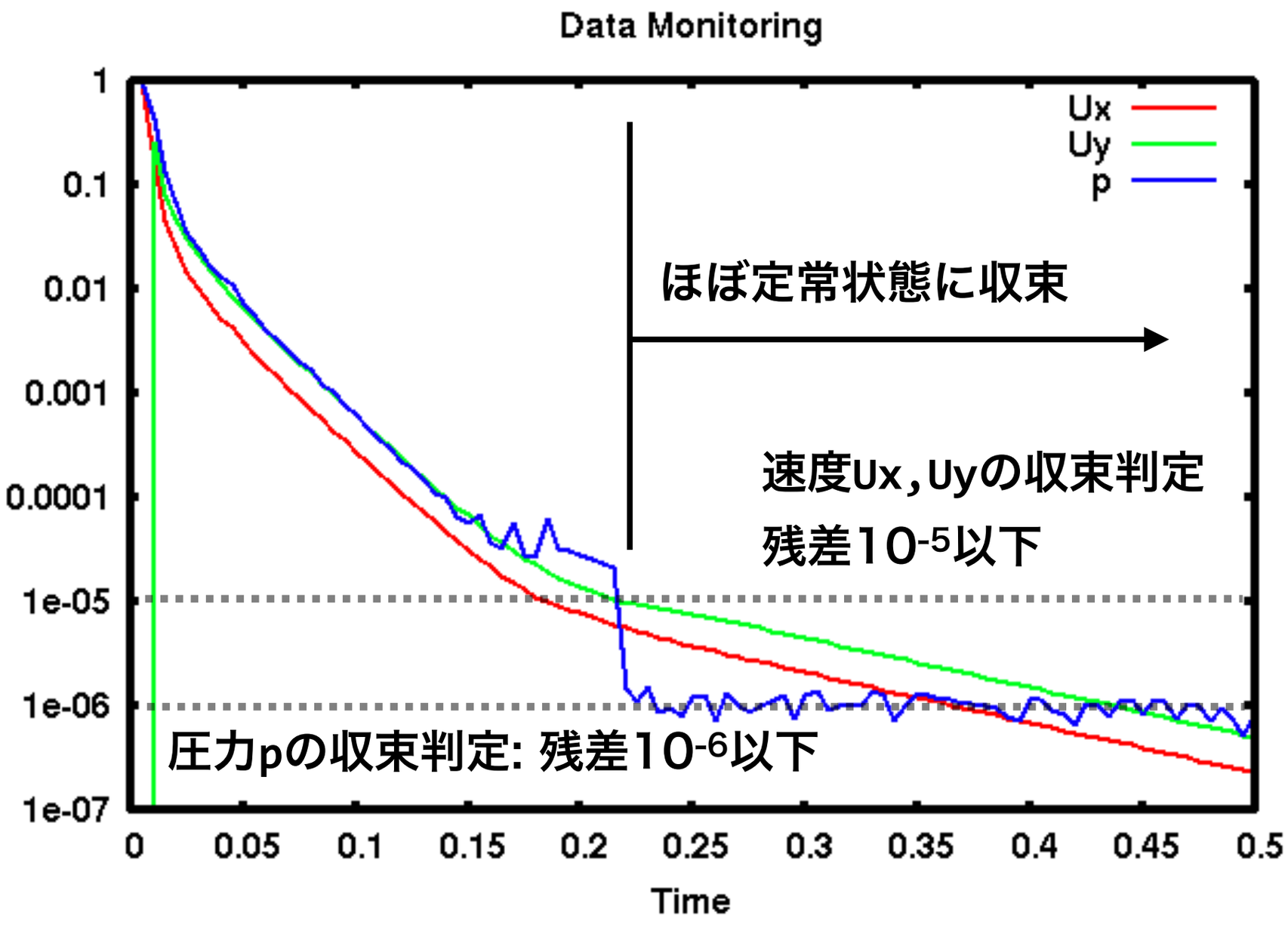
seq.sh.lRequestID

```
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のオプションは1秒間隔で更新. 必須ではない)

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



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

ParaViewによる解析結果可視化

ユーザーマシン

4. 格子可視化
[ParaView]

7.可視化
[ParaView]

1. ログイン
[ssh]

3. 格子転送
[scpまたは
rsync]

6. 解析結果転送
[scpまたは
rsync]

ログインノード

2.格子生成
[blockMesh]

5.流体解析
[ソルバ]

計算ノード

バッチジョブシステムによりジョブが実行される

解析結果の転送

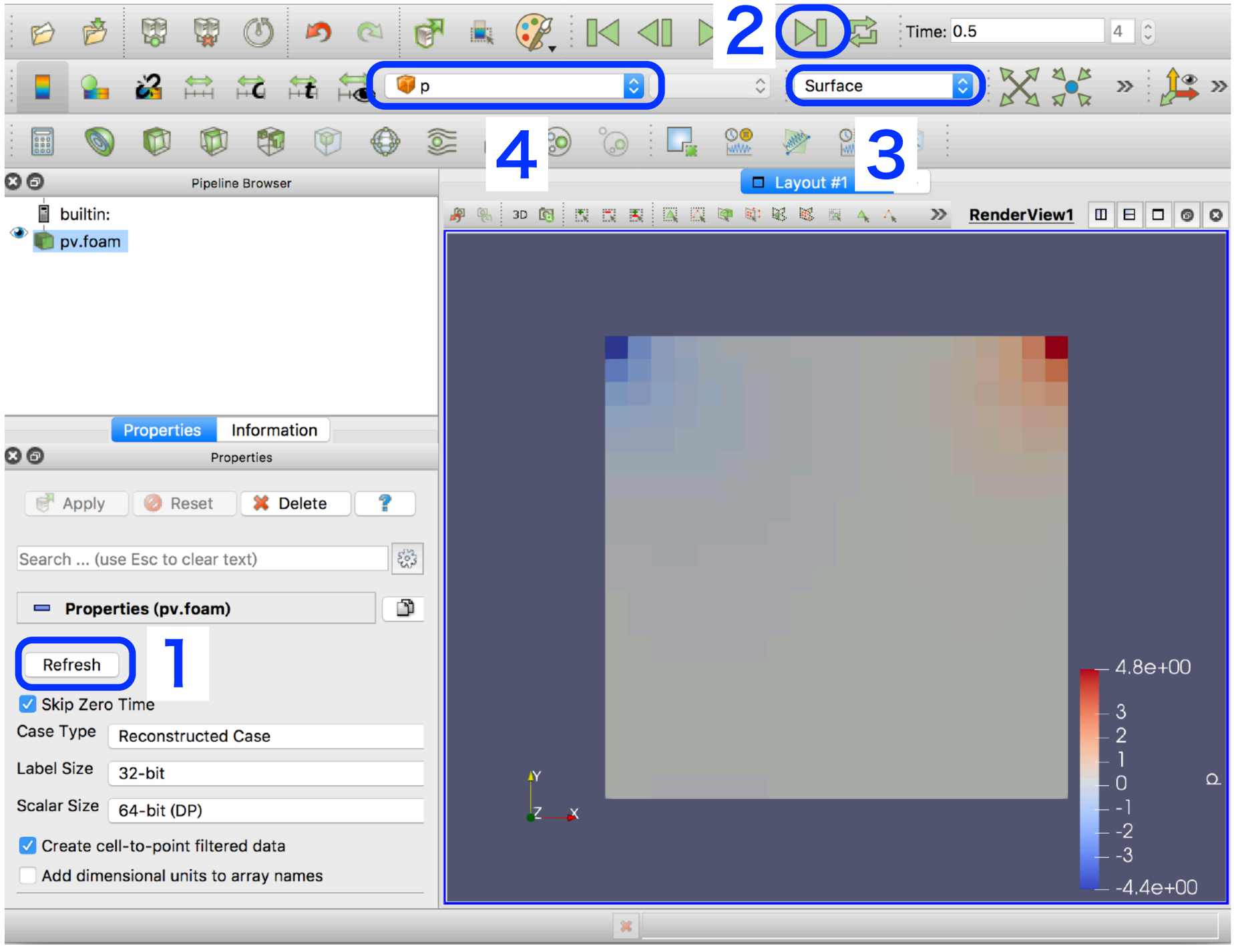
解析結果の転送(別端末で実行)

```
# カーソル上"↑"で前のコマンドが出る(ヒストリー機能). カーソル上下で履歴を辿れる
rsync -auv xxxxxx@octopus.hpc.cmc.osaka-u.ac.jp:lecture/ ~/lecture/
```

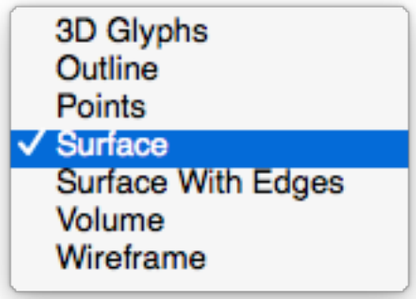
```
receiving file list ... done
cavity/
cavity/seq.sh.e1234567
cavity/seq.sh.0:l1234567.oct
cavity/seq.sh.o1234567
cavity/0.1/
cavity/0.1/U
:
```

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)

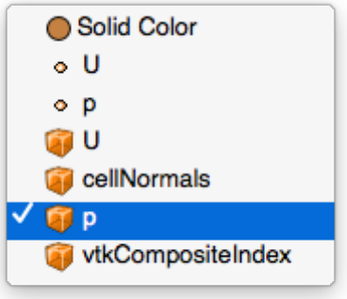
ParaViewによる圧力の可視化



1. Refresh(更新)
2. Last Frame (Timeが0.5に更新される)
3. Representation /Surface

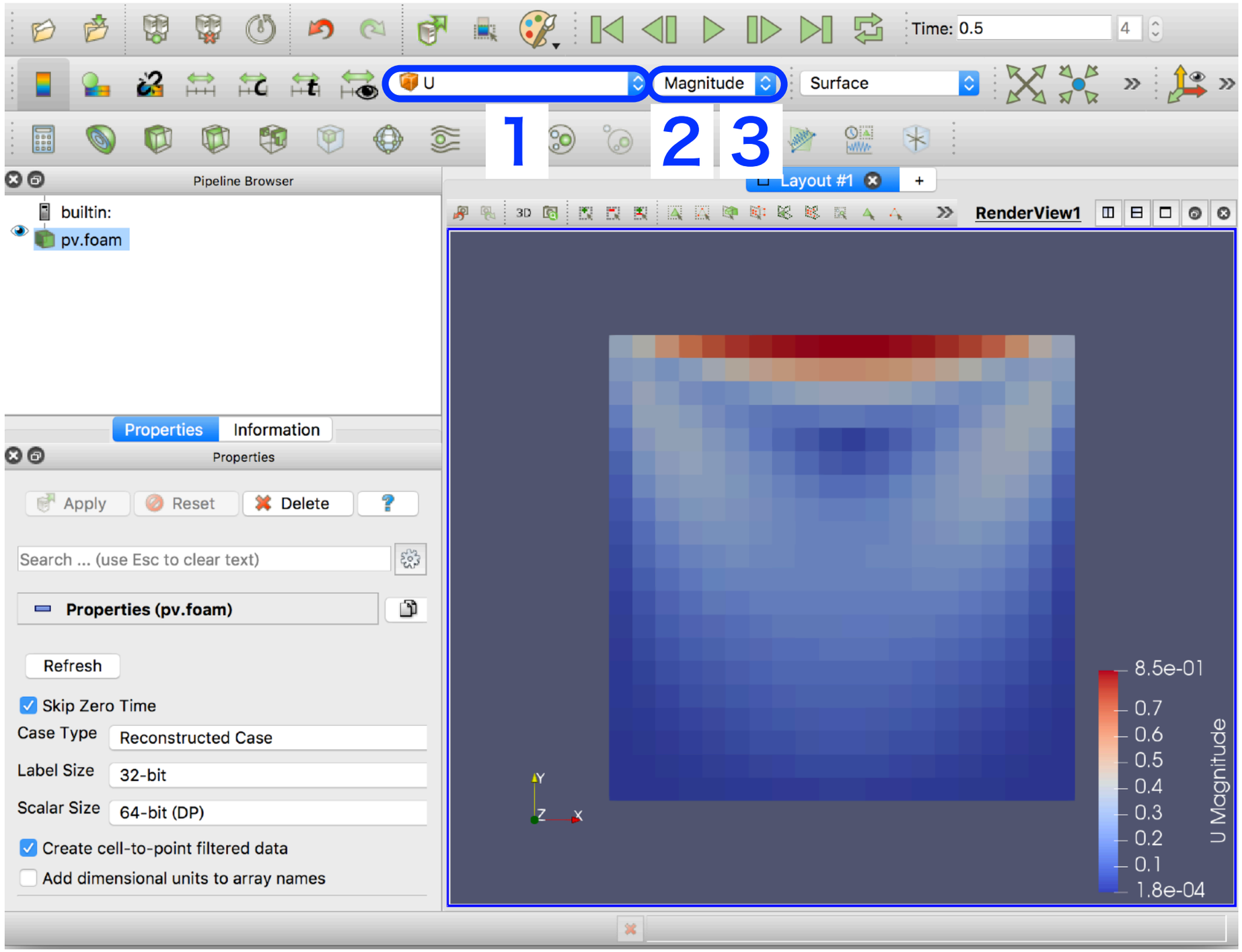


4. Coloring/□p

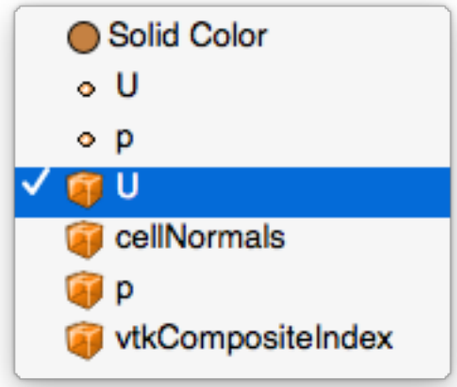


(□は格子の値そのままで補間無し、○は補間有り=スムージング)

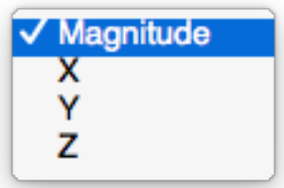
ParaViewによる風速の可視化



1. Coloring/□U

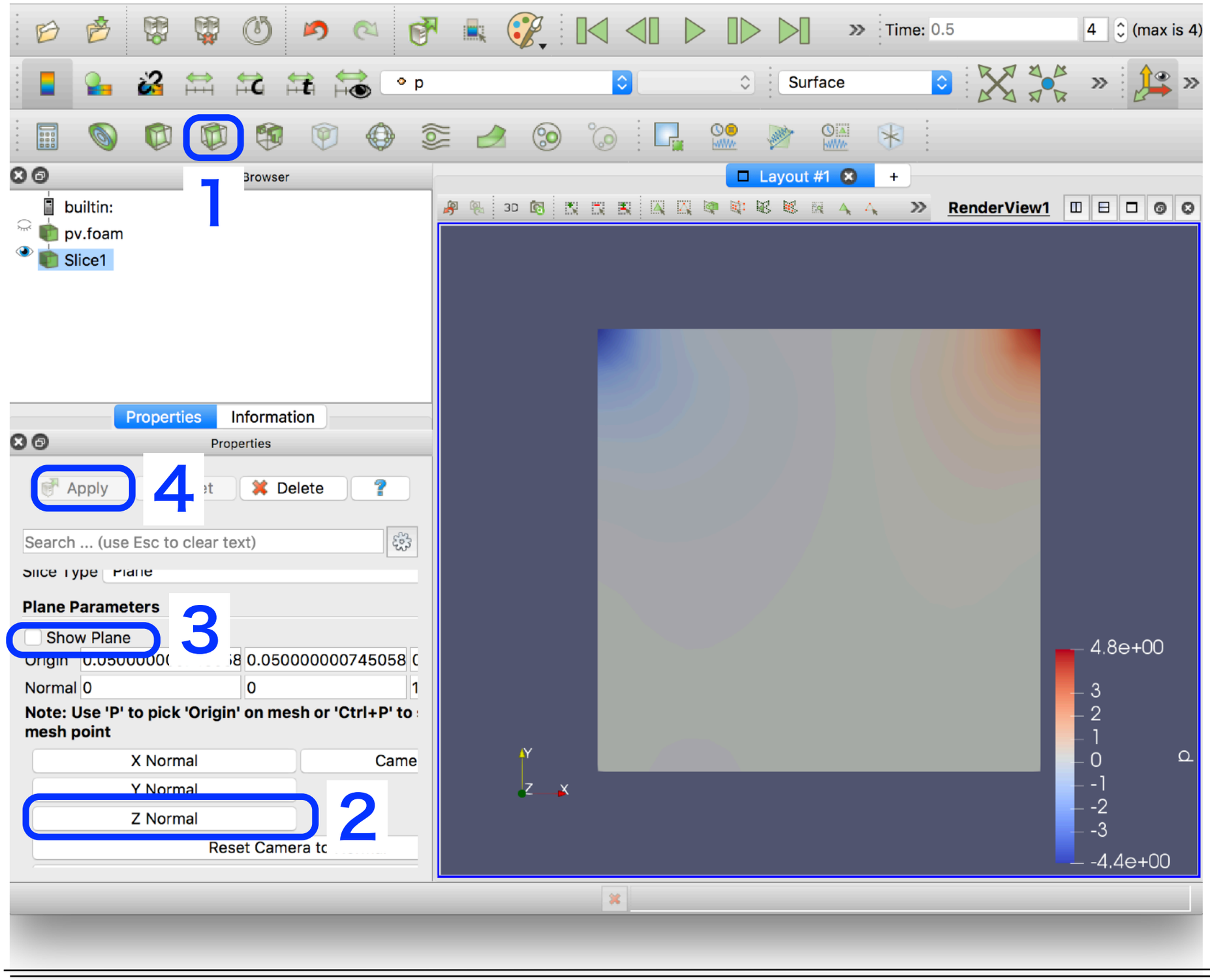


2. Magnitudeを X，Y，Zに変更することで，各風速成分が可視化できる.



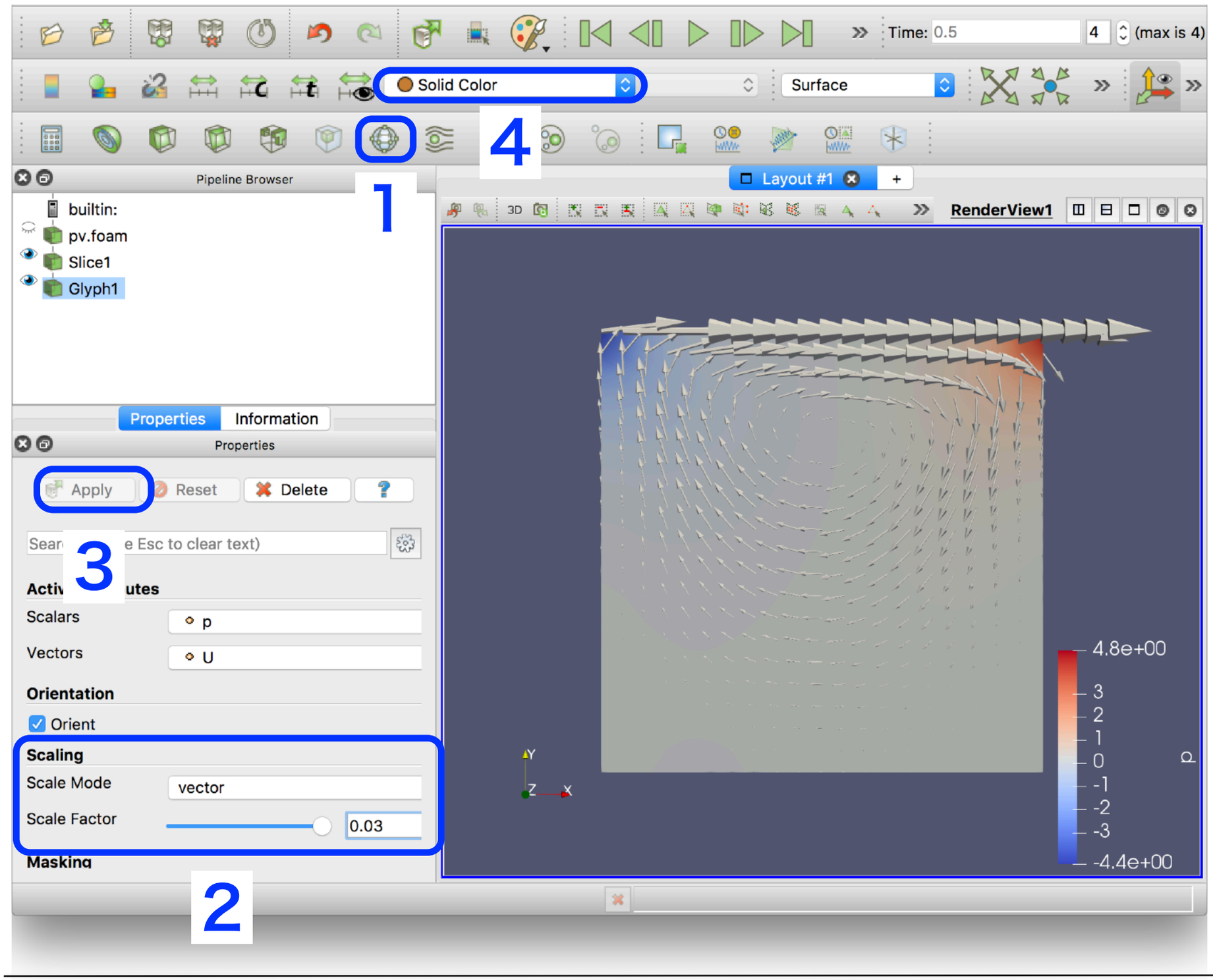
3. Magnitudeに戻す.

ParaViewによる風速ベクトルの可視化



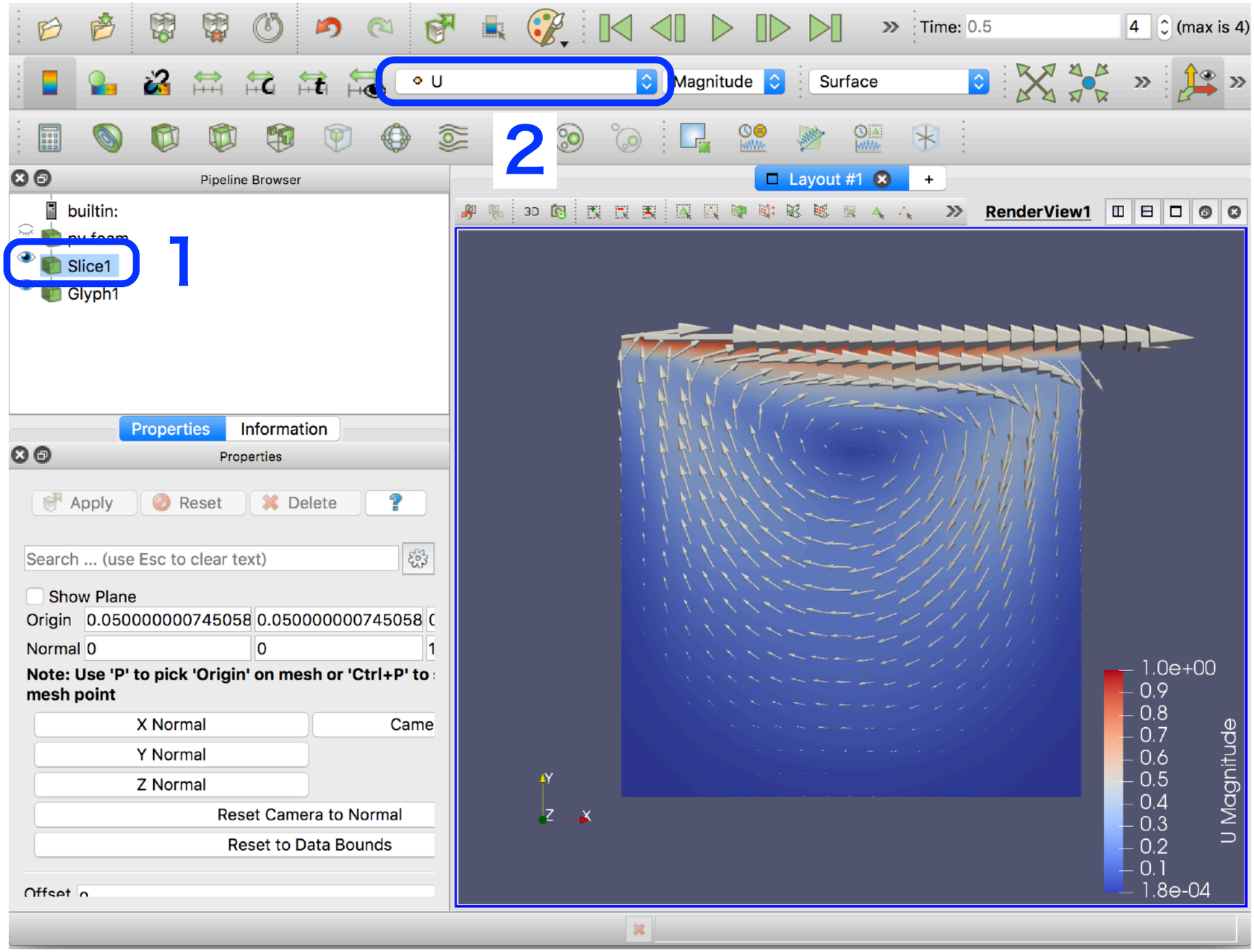
1. Sliceフィルタ
2. Z Normal
3. Show Planeを非選択
4. Apply

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)

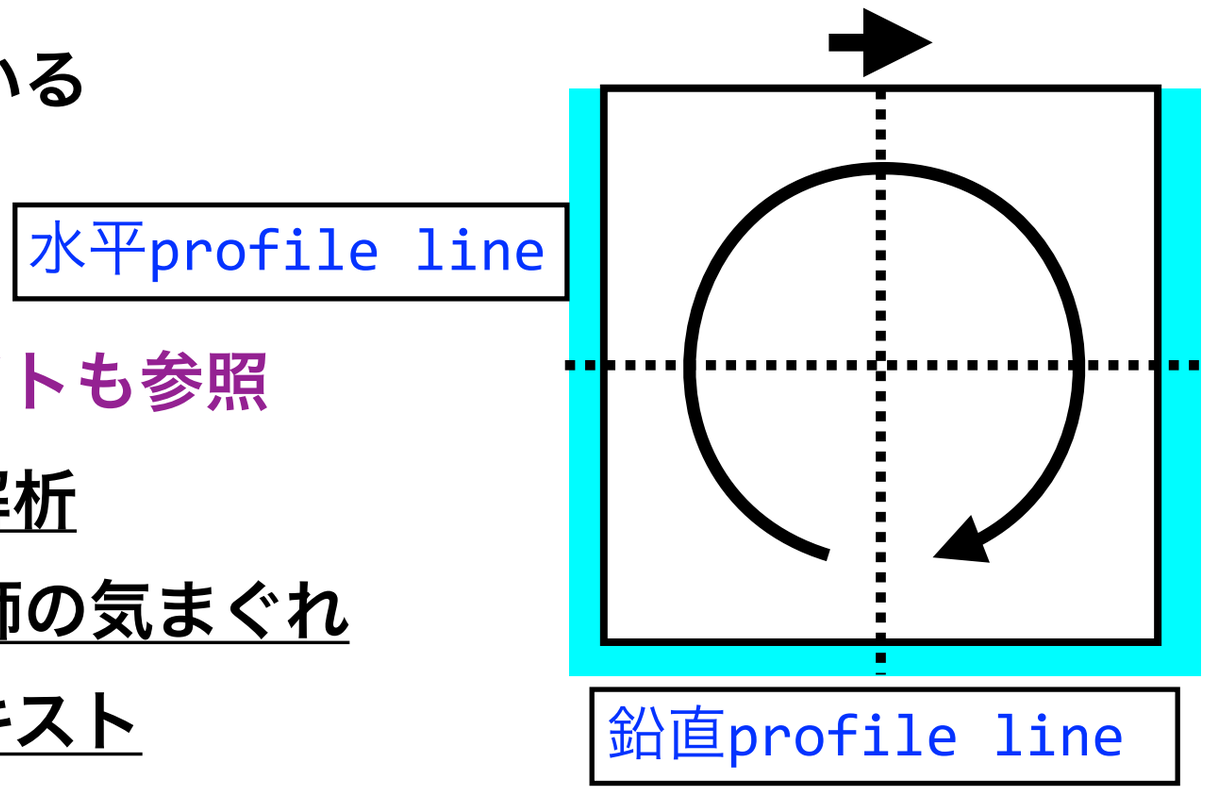


1. Sliceフィルタを選択(注)
2. Coloring/・Uを選択
3. Quitメニュー/Quit ParaView, または，ウィンドウを閉じる

(注) もし非表示になっていた場合は目のマークの表示をチェックする．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，ここではログインノード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 #新ケースのディレクトリ作成(名前は任意)
foamCopySettings cavity cavity2 #メッシュや格子以外の設定をコピー
cd cavity2
foamCleanTutorials #postProcessingなどの実行結果を消去
echo rm *.sh.* #バッチジョブの出力ファイルを消去(確認用に対象を表示)
rm *.sh.*      #バッチジョブの出力ファイルを消去(echoを取り実際に消去)
```

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，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．

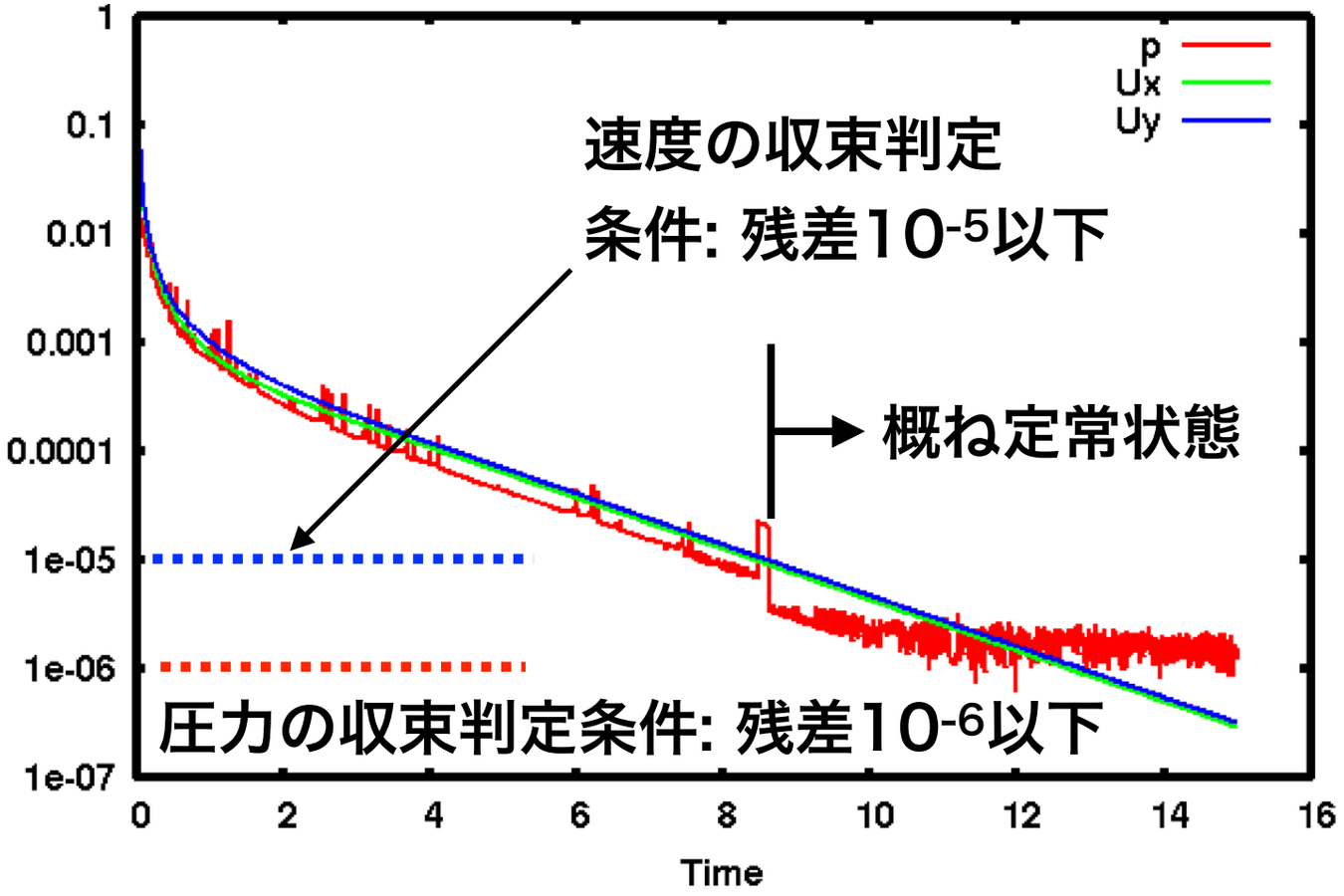
ジョブの確認・残差モニター・強制終了

ジョブ状態確認

```
qsub blockMesh.sh
sstat #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
qsub seq.sh
sstat
```

seq.shがジョブ実行中になったら初期残差をモニター(カーソル↑で呼びだし可)

```
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

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 );
```

サンプリングのライブラリ(動的リンク)

出力形式，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

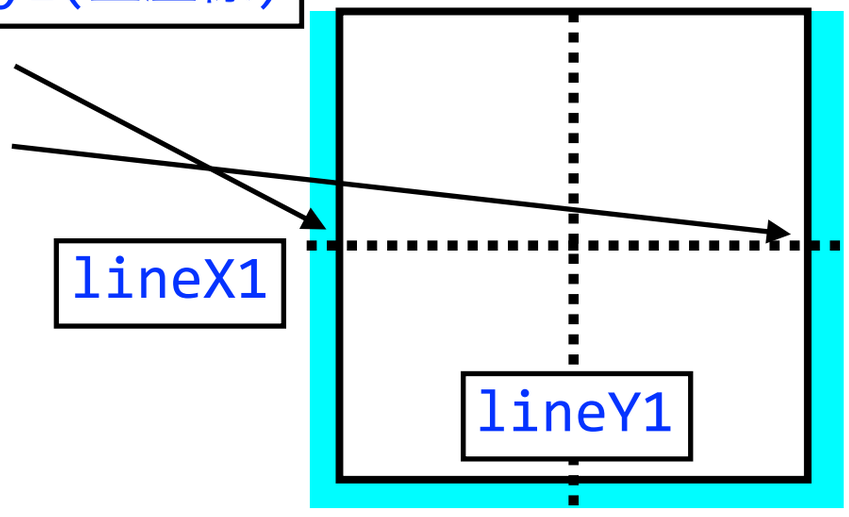
出力する座標軸，x/y/z/xyz(全座標)

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

このポスト処理は高負荷ではないのでログインノードで実行する．高負荷もしくは並列計算するプリポスト処理は計算ノードで実行する

サンプリング結果確認

```
more postProcessing/sample/*/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)

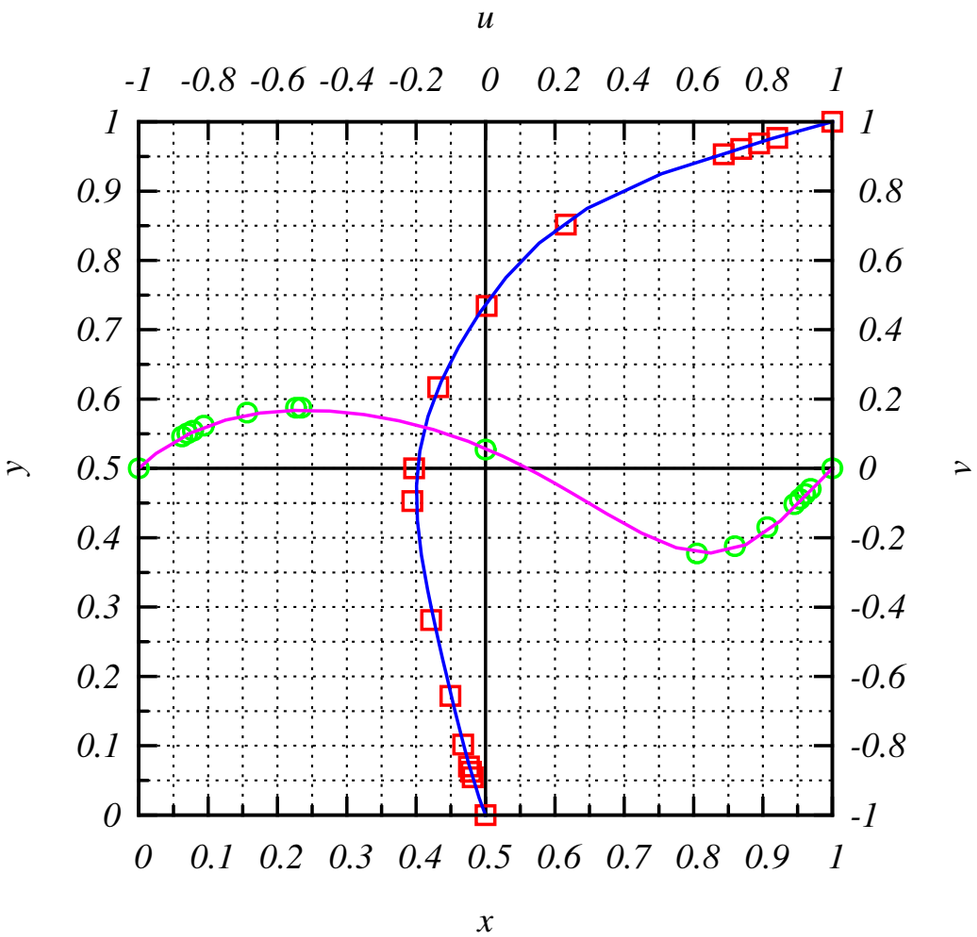
gnuplot実行

```
gnuplot profiles.gp
```

プロットファイル表示(Ctrl-Cで終了する)

```
gs -r150 profiles.pdf # -r<res> pixels/inch resolution
```

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1/Reに変更する．また，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．.

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.

並列計算

OpenFOAMの並列計算手法

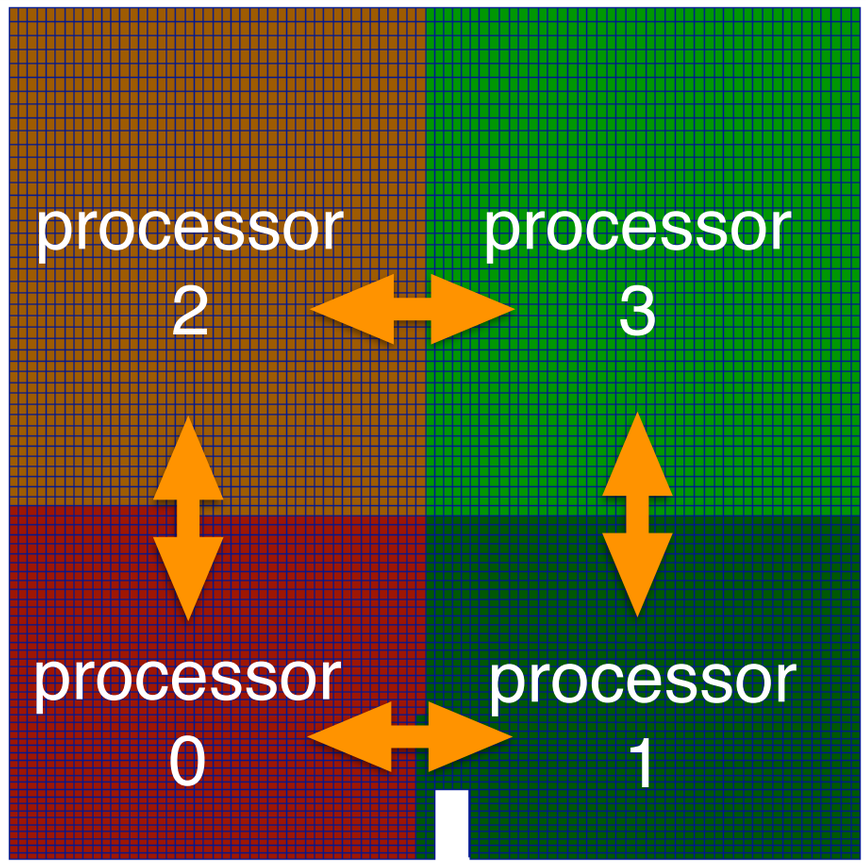
1. 格子生成
2. 領域分割 (decomposePar)
3. MPI並列でソルバを実行

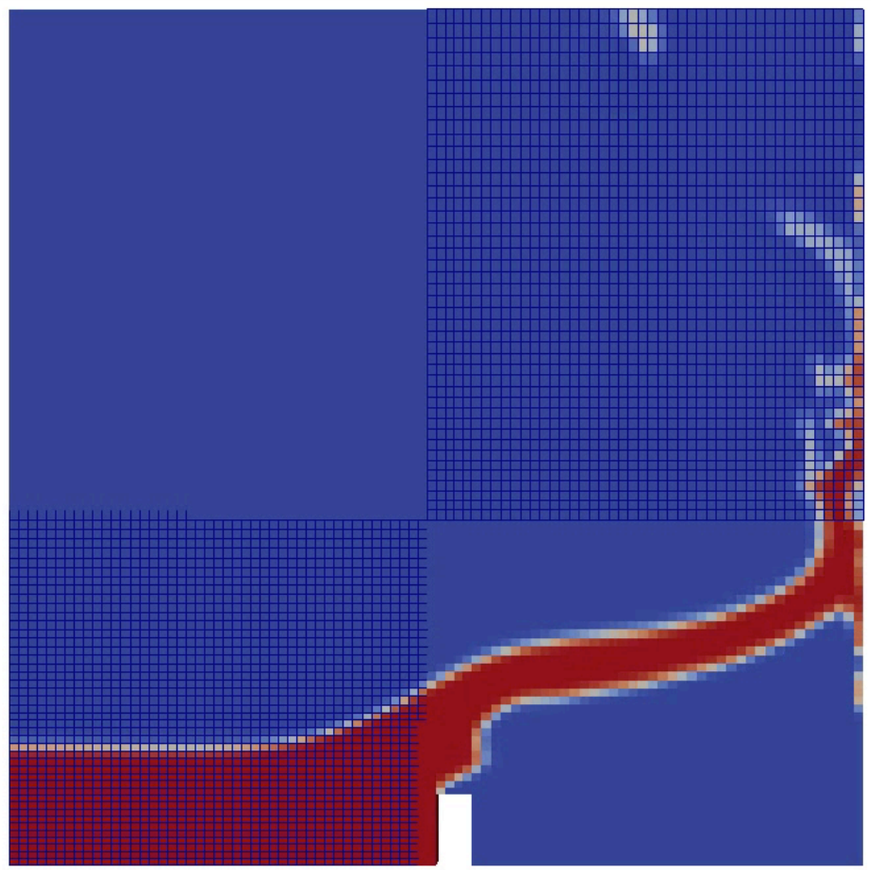
(MPI+OpenMPのハイブリット並列は標準では未実装．研究例有り)

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



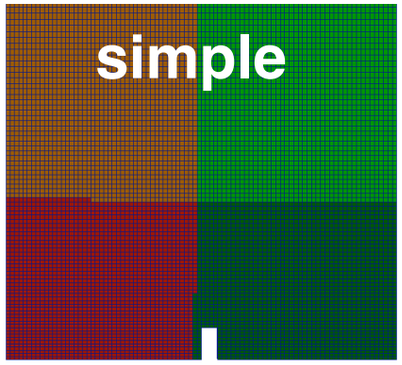


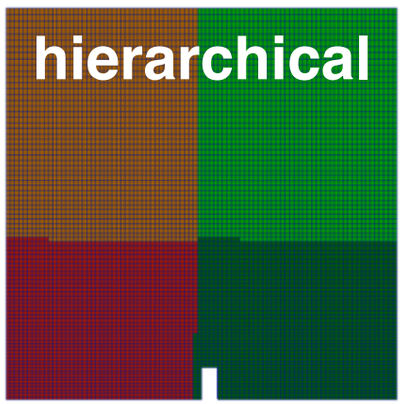
damBreakFineチュートリアル

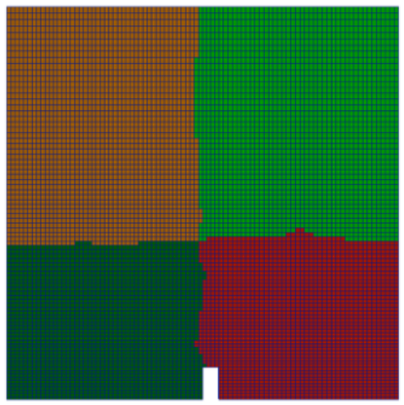
領域分割の設定

system/decomposeParDict

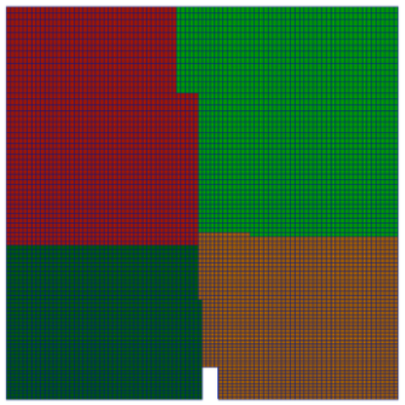
```
numberOfSubdomains 4;          //領域分割数

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



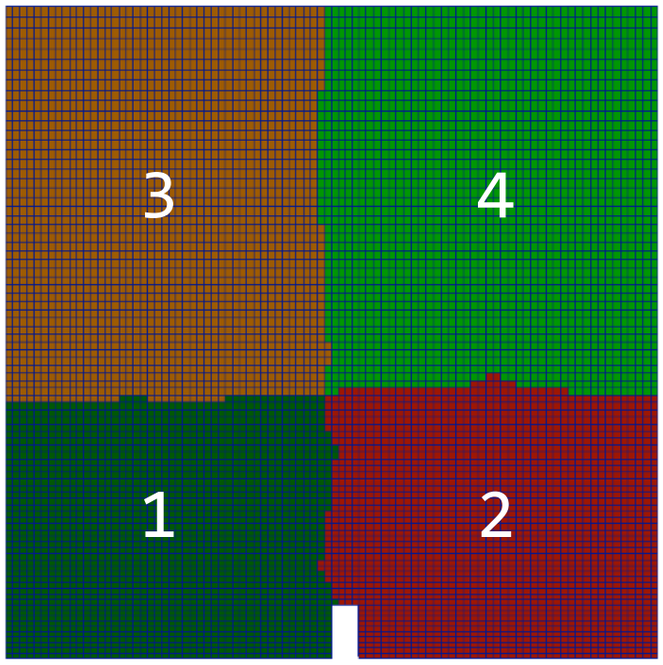
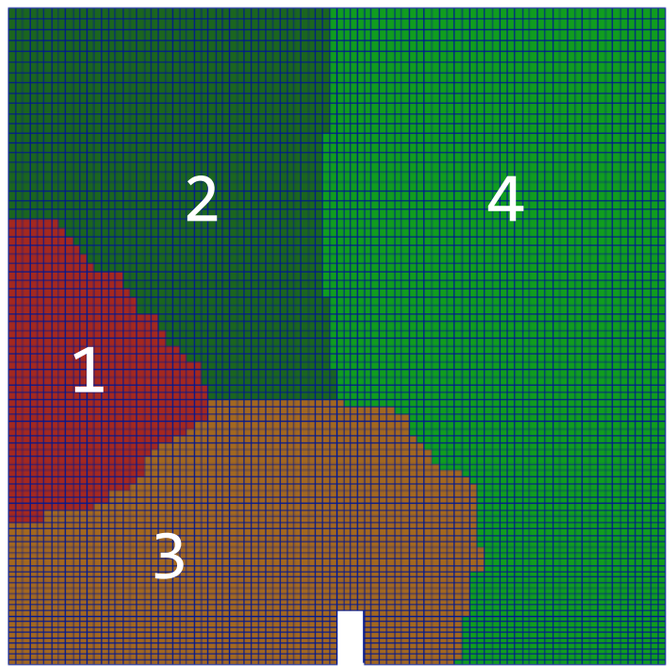
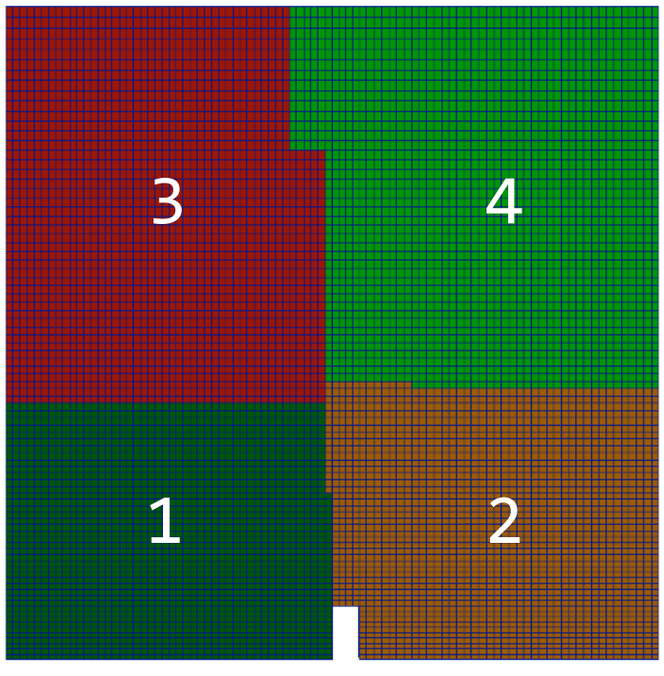
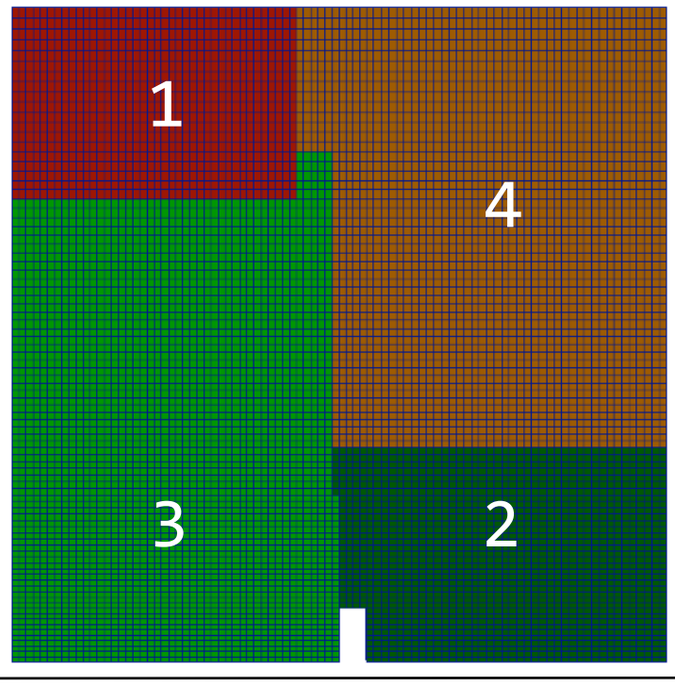
sctoch

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
#PBS -q OCTPHI
#PBS -l elapstim_req=0:10:00
#PBS -l cpunum_job=1
#PBS -b 1

# ジョブ投入時のディレクトリに移動
cd $PBS_O_WORKDIR
# OpenFOAM-4.1の環境設定
source /octfs/apl/OpenFOAM/4.1/OpenFOAM-4.1/etc/bashrc
# 念のため実行時の環境変数を記録
env
# decomposeParの実行
# -cellDistオプションは分割領域可視化用なので，省略可
decomposePar -cellDist >& $PBS_JOBNAME.l${PBS_JOBID#*:}
# end
```

領域分割ジョブ投入とログの確認

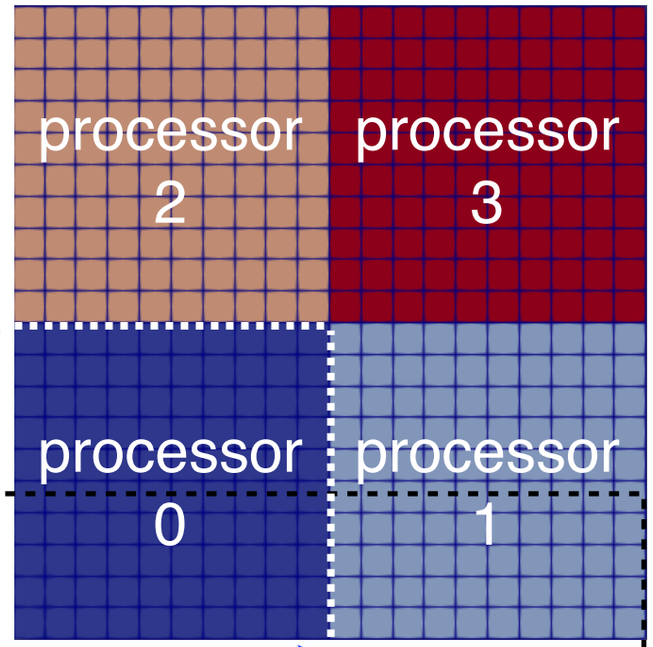
領域分割ジョブの投入

```
qsub decomposePar.sh
```

領域分割のログの確認(ジョブ完了後)

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

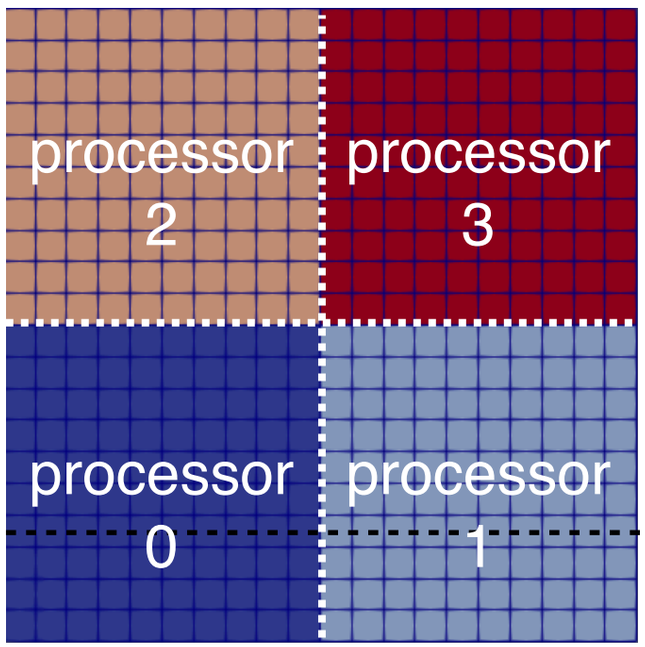
```
Processor 0 #プロセッサ0の担当分割領域
    Number of cells = 100 #格子数
    Number of faces shared with processor 1 = 10
    #プロセッサ番号1の担当分割領域との共有界面数
    Number of faces shared with processor 2 = 10
    #プロセッサ番号2の担当分割領域との共有界面数
    Number of processor patches = 2 #上記の界面を共有するプロセッサ数
    Number of processor faces =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



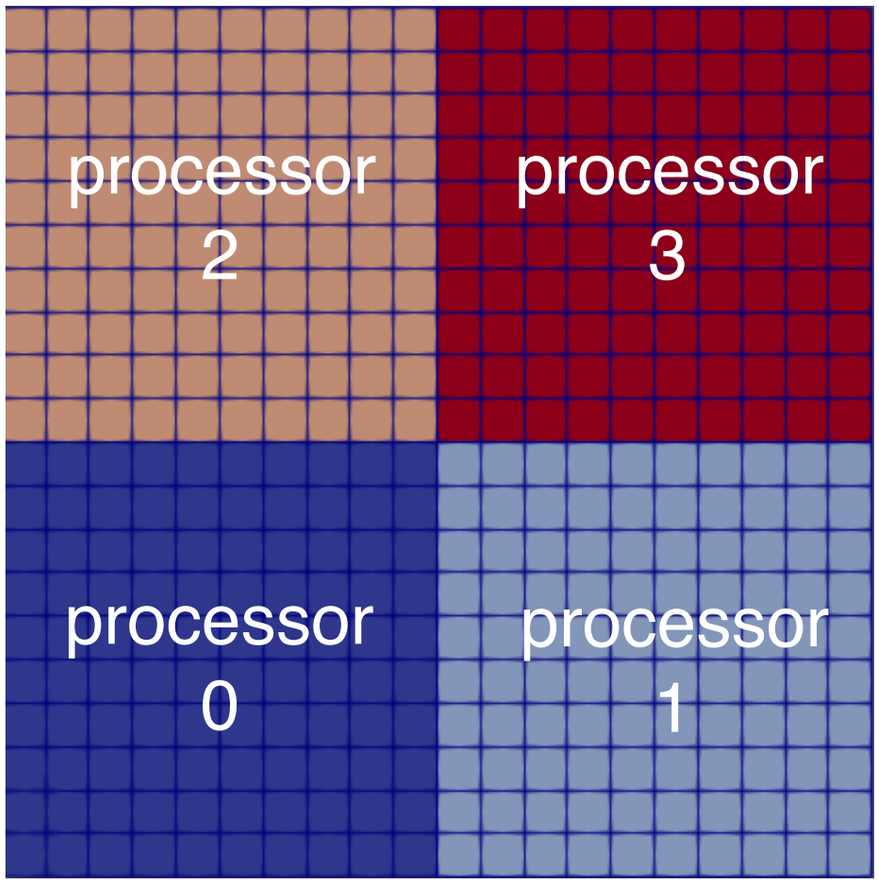
領域分割のログの確認(続き)

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/
```

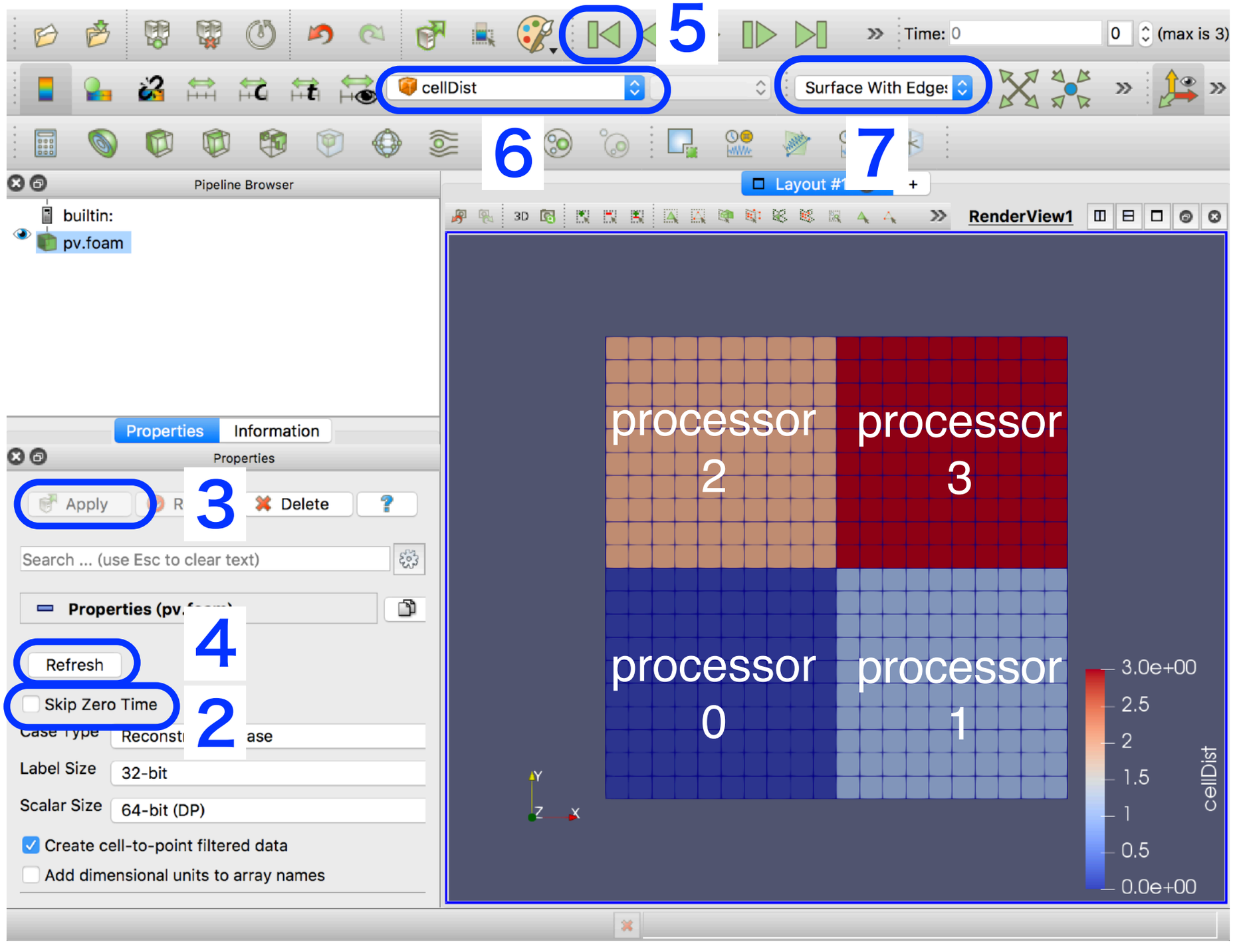
領域分割により生成されたデータ

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"↑"で前のコマンドが出る(ヒストリー機能). カーソル上下で履歴を辿れる
rsync -auv xxxxxx@octopus.hpc.cmc.osaka-u.ac.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，赤字が並列用修正・追加分）

```
#PBS -l cpunum_job=4
#PBS -b 1
#PBS -T intmpi

# ジョブ投入時のディレクトリに移動
cd $PBS_O_WORKDIR
# OpenFOAM-4.1の環境設定
source /octfs/apl/OpenFOAM/4.1/OpenFOAM-4.1/etc/bashrc
# MPIプロセスのコア割り当てなどIntel MPIのデバッグ情報取得
export I_MPI_DEBUG=5
# 念のため実行時の環境変数を記録
env
# icoFoamの実行
# $NQSII_MPIOPTS : mpirunのオプション(使用ノード名・プロセス数情報)
# -parallel : OpenFOAMのアプリケーションは並列計算時，本オプションが必要
mpirun $NQSII_MPIOPTS -np 4 \
numactl -p 1 \
icoFoam -parallel >& $PBS_JOBNAME.l${PBS_JOBID#*:}
```

ノード毎のジョブ数(フラットMPIの場合，MPIプロセス数)

Intel MPIを用いる場合に必要

並列計算とログ確認

```
qsub par.sh
more par.sh.l*
```

ジョブ開始後

```
[0] MPI startup(): Multi-threaded optimized library
[2] MPI startup(): shm data transfer mode
[3] MPI startup(): shm data transfer mode
[0] MPI startup(): shm data transfer mode
[1] MPI startup(): shm data transfer mode
[0] MPI startup(): Rank    Pid      Node name  Pin cpu
[0] MPI startup(): 0       304413   octopus03  {0,1,2,3,4,5,24,25,26,27,28,29}
[0] MPI startup(): 1       304414   octopus03  {6,7,8,9,10,11,30,31,32,33,34,35}
[0] MPI startup(): 2       304415   octopus03  {12,13,14,15,16,17,36,37,38,39,40,41}
[0] MPI startup(): 3       304416   octopus03  {18,19,20,21,22,23,42,43,44,45,46,47}
[0] MPI startup(): I_MPI_DEBUG=5
[0] MPI startup(): I_MPI_INFO_NUMA_NODE_MAP=mlx5_0:0
[0] MPI startup(): I_MPI_INFO_NUMA_NODE_NUM=2
[0] MPI startup(): I_MPI_PIN_MAPPING=4:0 0,1 6,2 12,3 18

nProcs : 4
Slaves :
3
(
"octopus03.304414"
"octopus03.304415"
"octopus03.304416"
)
```

使用されるファブリック．デフォルトでは，ノード内はshm(共有メモリ)，ノード間はdapl(Direct Access Programming Library)

MPIプロセスのコア割付情報

計算で使用されている総MPIプロセス数

スレーブプロセス数(総MPIプロセス数-1)

スレーブプロセスの計算ノードのホスト名:PID

意図したノード・プロセス数・コア割付で動いているか確認(失敗→ジョブファイルの指定確認)

並列計算結果の再構築用ジョブスクリプト

reconstructPar.sh

```
#!/bin/bash
#PBS -q OCTPHI
#PBS -l elapstim_req=0:10:00
#PBS -l cpunum_job=1
#PBS -b 1

# ジョブ投入時のディレクトリに移動
cd $PBS_O_WORKDIR
# OpenFOAM-4.1の環境設定
source /octfs/apl/OpenFOAM/4.1/OpenFOAM-4.1/etc/bashrc
# 念のため実行時の環境変数を記録
env
# reconstructParの実行
reconstructPar >& $PBS_JOBNAME.l${PBS_JOBID#*:}
```

領域分割ジョブの投入

```
qsub reconstructPar.sh
```

並列計算結果の再構築

並列計算時の解析結果

processor0/ プロセッサディレクトリ
  5/ 分割領域の時刻ディレクトリ
    U, p, phi
  :
  15/
processor1/
  5/
    U, p, phi
processor2/
  5/
    U, p, phi
processor3/
  5/
    U, p, phi

分割済み解析結果

再構築後の解析結果
(通常と同じ場所)

5/ 時刻ディレクトリ
  U, p, phi
10/
  U, p, phi
  :
15/
  U, p, phi

並列計算結果と再構築結果の確認

```
find | sort | 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

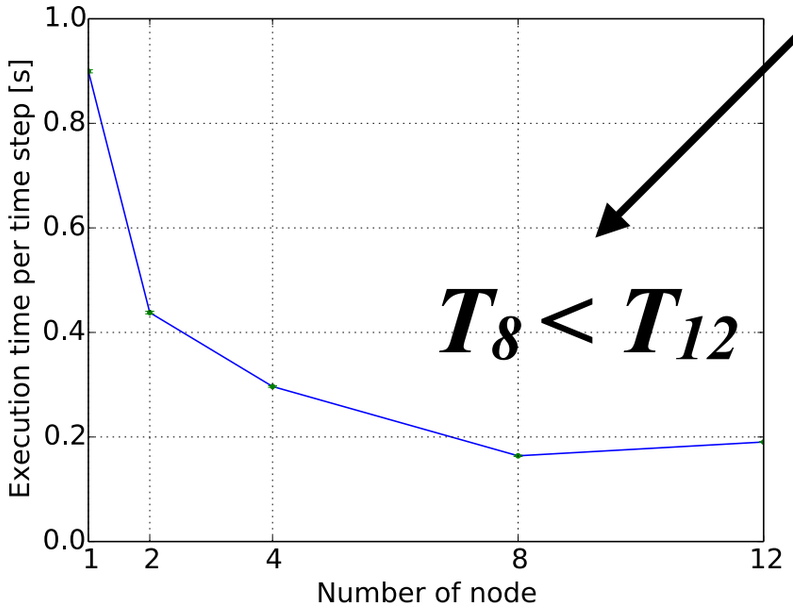
T_S ： ベースとなるプロセス数(1プロセス，1ノード等)での実行時間

T_P ： ベースとなるプロセス数×Pでの実行時間

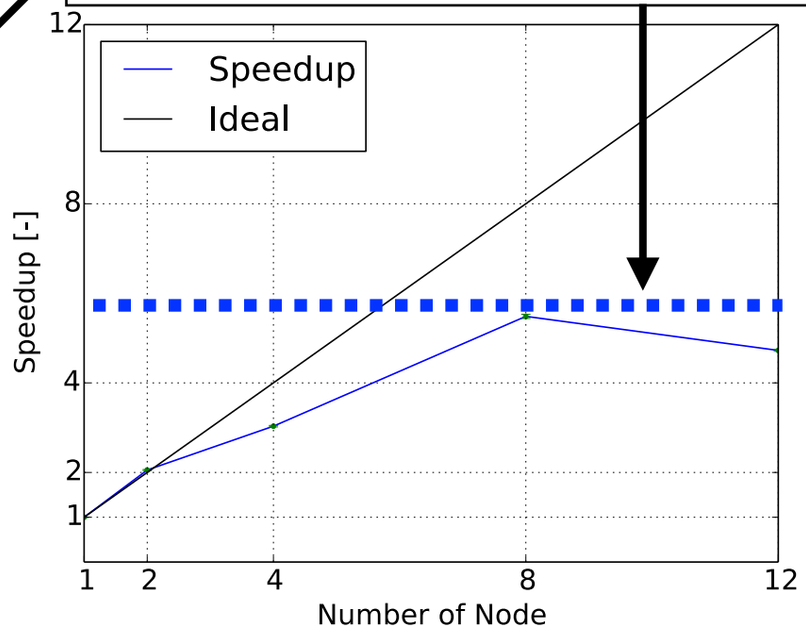
- 並列化効率 E_P

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

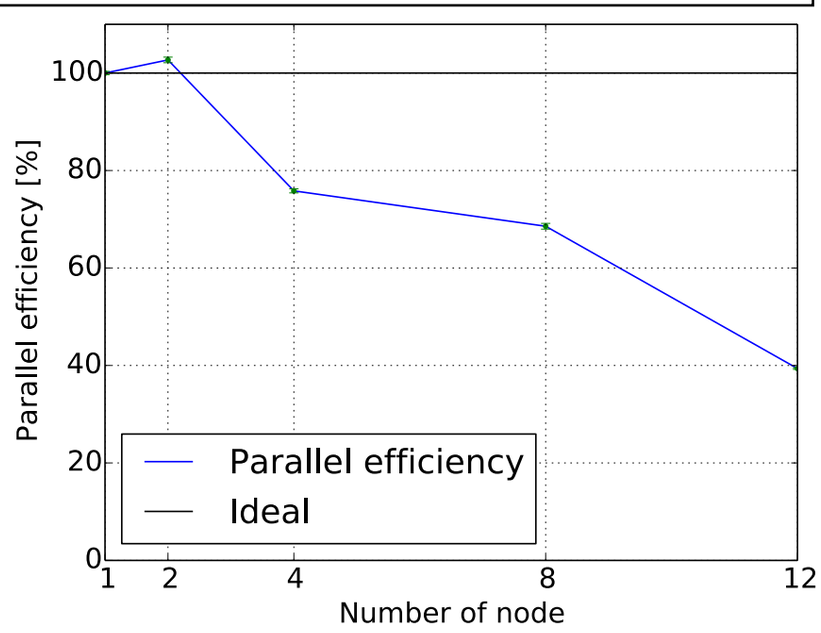
このケースでは12ノード以上使用しても非効率
→台数効果の飽和(Saturation)をベンチマークテストで事前に把握し，非効率な計算を行わないことが重要



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などでスピードアップ率などを計算

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

```
0.68282753515 #逆に遅くなっている
```

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と相対的に多いので，スピードアップ率や並列化効率が悪い.

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
method               simple;  //simpleをscotchに変更しても良い
simpleCoeffs //単純に軸方向に分割
{
    n        ( 8 8 1 ); //分割数(methodがscotchの場合には，変更不要)
```

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
```

```
#PBS -l cpunum_job=64
#PBS -b 1
```

ノード毎のMPIプロセス数，ノード数を変更する

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

今回は実行しない

```
qsub blockMesh.sh
sstat
qsub seq.sh
sstat
qsub decomposePar.sh
sstat
qsub par.sh
```

ジョブを続けて実行するには以下のように実行するほうが便利(直列連携)

```
qsub blockMesh.sh seq.sh decomposePar.sh par.sh
```

並列にジョブを入れる場合にはコロンで区切る(並列連携，今回は実行しない)

```
qsub blockMesh.sh:seq.sh:decomposePar.sh:par.sh
```

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

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

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

```
qsub foamRunTutorials.sh
sstat
```

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)

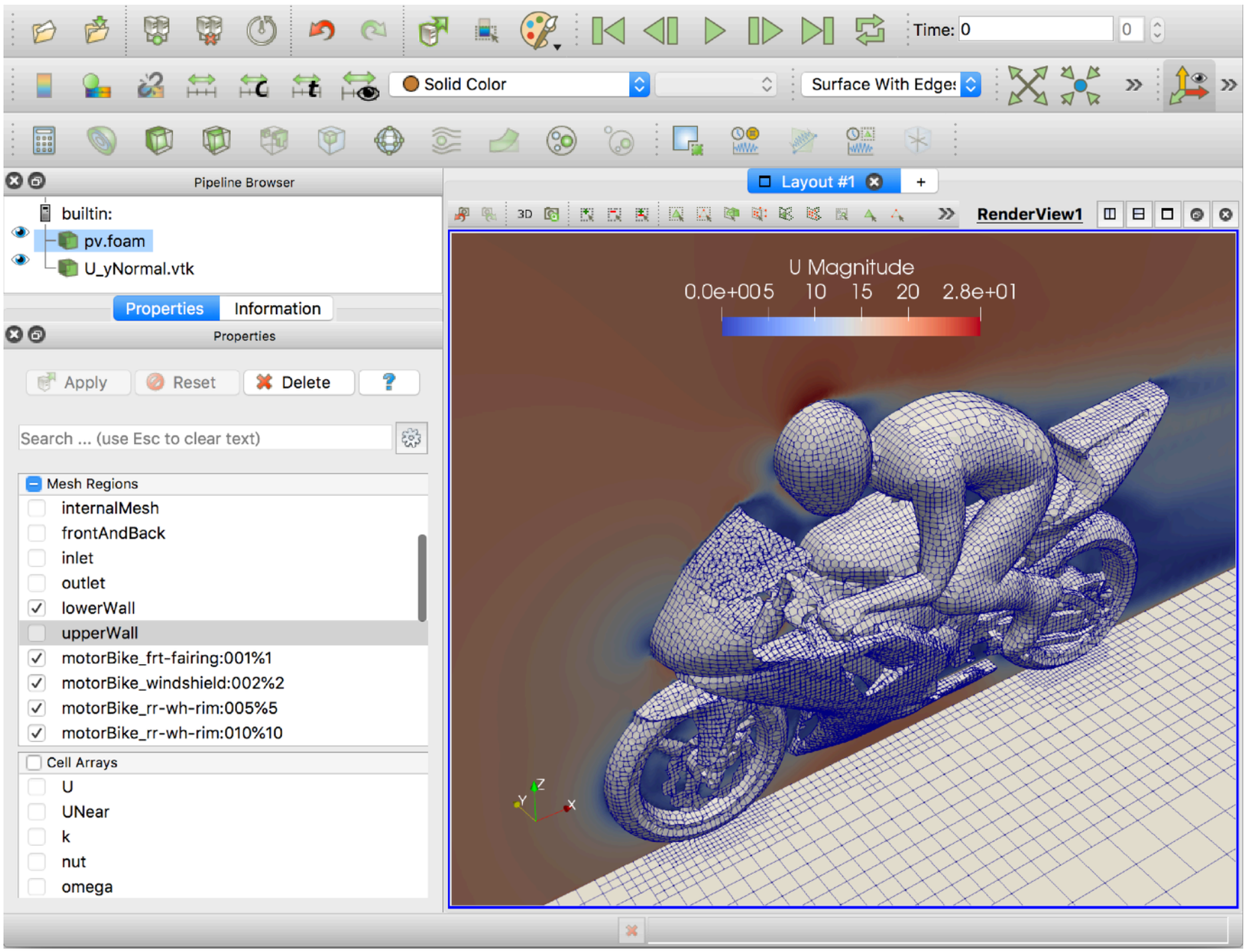
チュートリアルの実行結果の転送(別端末で実行)

```
# 並列計算ディレクトリprocessor*は可視化に不要なので，excludeオプションで指定
rsync -auv xxxxxx@octopus.hpc.cmc.osaka-u.ac.jp:lecture/ ~/lecture/ \
 --exclude=processor*
cd ~/lecture/motorBike/
touch pv.foam
```

ログファイルや前回の計算結果が残っていると，foamRunTutorialsによるチュート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
```

チュートリアルの実行例(続き)



1. File/Open *pv.foam*→OK
2. Mesh Regions *select* →
InternalMesh,frontAndBack,inlet,outlet,upperWall *unselect* →
3. *Cell Arrays unselect*
4. Apply
5. Coloring/Solid Color
6. Representation/ Surface With Edges
7. File/Open → postProcessing/ cuttingPlane/500/ U_yNormal.vtk → OK → Apply
8. Coloring/・U
9. マウスで調整

postProcessing以下は実行時作成・可視化されたファイル
setsには流線・表面流線があるので，こちらも可視化してみる

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)

```
# amplxe-cl : intel VTuneのコマンド(コマンドライン版)
# -c 収集するデータタイプ(advanced-hotspots, hotspots等)
# -r データ保存ディレクトリ(既に存在するとエラーになる)
/octfs/apl/intel/vtune_amplifier_xe/bin64/amplxe-cl \
-c advanced-hotspots -r $PBS_JOBNAME.v${PBS_JOBID#*:} \
numactl -p 1 \
icoFoam >& $PBS_JOBNAME.l${PBS_JOBID#*:}
```

par-vtune.sh (赤字が並列計算用追加分)

```
# Intel MPIの場合, -gtoolでamplex-clコマンドを指定可能
# -r データ保存ディレクトリ:データ収集対象のプロセス番号(または範囲)
mpirun $NQSII_MPIOPTS \
-gtool "/octfs/apl/intel/vtune_amplifier_xe/bin64/amplxe-cl \
-c advanced-hotspots -r $PBS_JOBNAME.v${PBS_JOBID#*:}:0" \
numactl -p 1 \
icoFoam -parallel >& $PBS_JOBNAME.l${PBS_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

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

ソルバのログを追跡

```
Elapsed Time:          45.086
CPU Time:              44.580
Average CPU Usage:     0.995
CPI Rate:              1.193
```

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 99.5%

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）

```
/octfs/apl/intel/vtune_amplifier_xe/bin64//amplxe-cl -R hotspots -r seq-vtune.sh.v* > seq-vtune.txt # ../bin/vtune-report.sh でも可
more seq-vtune.txt
```

```
Function                                          CPU Time
Foam::DICPreconditioner::precondition   14.770s
Foam::lduMatrix::Amul                   12.440
```

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)

```
PMPIDI_CH3I_Progress       2.760s
:
Foam::DICPreconditioner::precondition   0.250s
```

MPI関連が支配的となる→並列化効率が落ちる

線形ソルバPCGのDIC前処理

Intel MPIライブラリ使用時の注意点

- Intel MPIライブラリを用いてOpenFOAMを実行すると，多ノードで実行エラーになる場合があるが，以下の色付きの部分をジョブスクリプトに追加し，RDMA translationキャッシュ機能をOFFにすると，実行エラーを回避できる可能性がある．なお，この設定による速度低下は僅かである．

```
# MPIプロセスのコア割り当てなどIntel MPIのデバッグ情報取得
export I_MPI_DEBUG=5
# RDMA translationキャッシュ機能OFF
export I_MPI_DAPL_TRANSLATION_CACHE=0
export I_MPI_DAPL_UD_TRANSLATION_CACHE=0
# 念のため実行時の環境変数を記録
env
```

OCTOPUSへのOpenFOAMのインストール

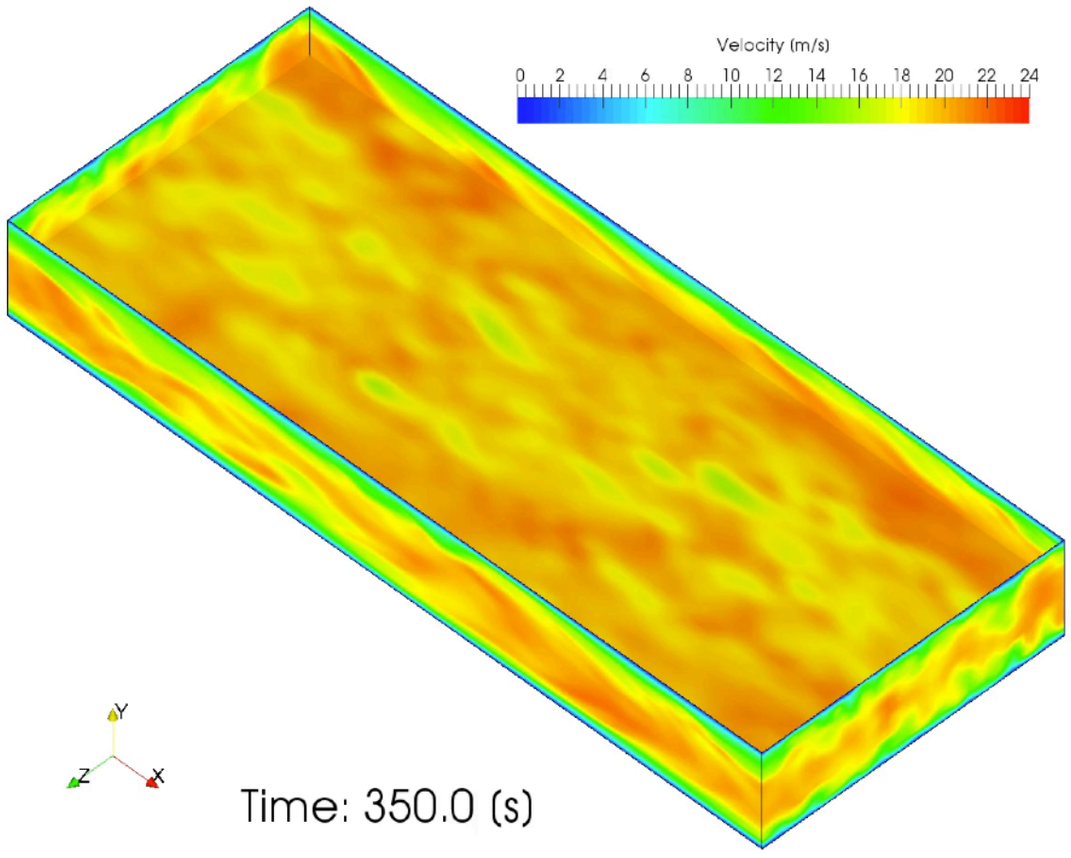
- OpenFOAM自動ビルドスクリプト(https://gitlab.com/OpenCAE/installOpenFOAM/blob/master/README.md) を参考にソースからビルドする

OCTUPUSでのOpenFOAMベンチマークテスト

格子生成 (blockMesh)

チャネル流れ ($Re_\tau = 110$)

格子数約3M



解析条件

$L_x \times L_y \times L_z = 5\pi \times 2 \times 2\pi$

$Re_\tau = u_\tau \, \delta / \mu = 110$ [-]

ここで

L_x, L_y, L_z：各方向のチャネル幅 [m]

u_τ：壁面摩擦速度 [m/s]

δ ：チャネル半幅 [m] ($=L_y / 2$)

μ ：動粘性係数 [m^2/s^2]

主流方向(x): 一定の圧力勾配

主流方向(x), スパン方向(z): 周期境界

ソルバ: OpenFOAM pimpleFoam

乱流モデル: 無し(laminar)

速度線型ソルバ: BiCG (前処理DILU)

圧力線型ソルバ: PCG (前処理DIC)

領域分割手法：scotch(周期境界面は同領域)

- **2〜51ステップのCPU時間(Execution time)から1時間あたりのステップ数を算出**

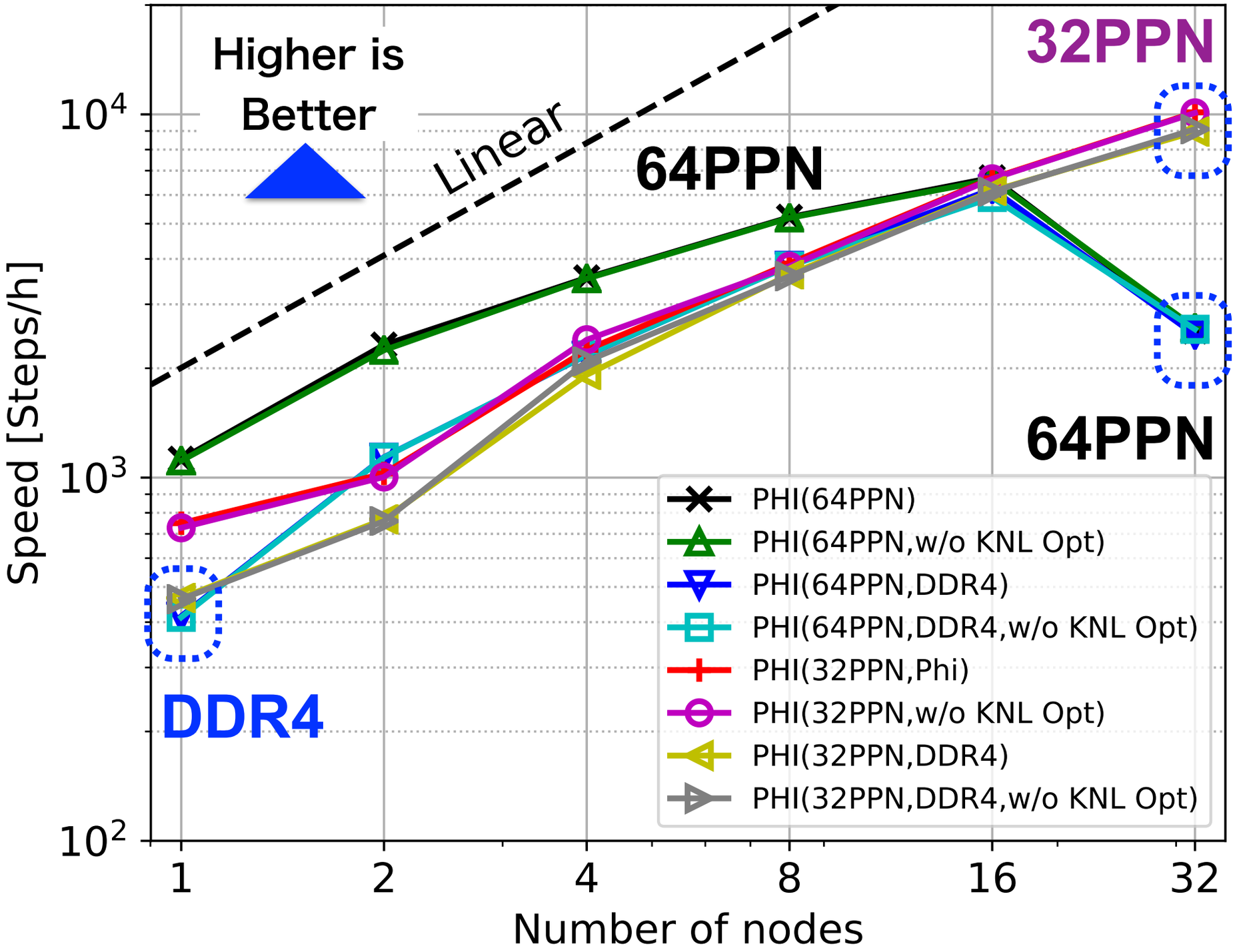
チャネル流ベンチマーク計測システム

機関	システム (略称)	CPU [GPU] (周波数[GHz])	CPU数 (コア)	倍精度性能 [GFlops]	メモリ[GiB] (帯域幅[GB/s])	インターコネクト (帯域幅[Gbps])
JCAHPC	Oakforest-PACS (OFP)	Intel Xeon Phi 7250, KNL(1.4)	1(68)	3046	96(115.2), MCDRAM 16(490)	Intel Omni-Path (100)
東京大学	Reedbush-U (RBU)	Intel Xeon E5-2695 v4 (2.1-3.3)	2(36)	1210	256 (76.8×2)	Infiniband EDR(100)
九州大学	ITOサブシステム A (ITO-A)	Intel Xeon Gold 6154 (3.0-3.7)	2(36)	3450	192(255.9)	Infiniband EDR(100)
大阪大学	OCTOPUS 汎用 CPU (OCT)	Intel Xeon Gold 6126(2.6)	2(24)	1997	192(255.9)	Infiniband EDR(100)
	OCTOPUS Xeon Phi (PHI)	Intel Xeon Phi 7210, KNL(1.3)	1(64)	2662	192(102.4), MCDRAM 16(400〜)	

システム	バージョン	コンパイラ(※1)	MPI
OFP	v1612+	Icc 2017.1(KNL向け最適化)	Intel MPI 2017.1(※2)
RBU	2.3.0	Gcc-4.8.5	OpenMPI 1.8.3
ITO-A	v1706	Icc 2018.1	Intel MPI 2018.1
OCT	v1612+	Icc 2017.0(KNL向け最適化)	IntelMPI 2017.5
PHI			

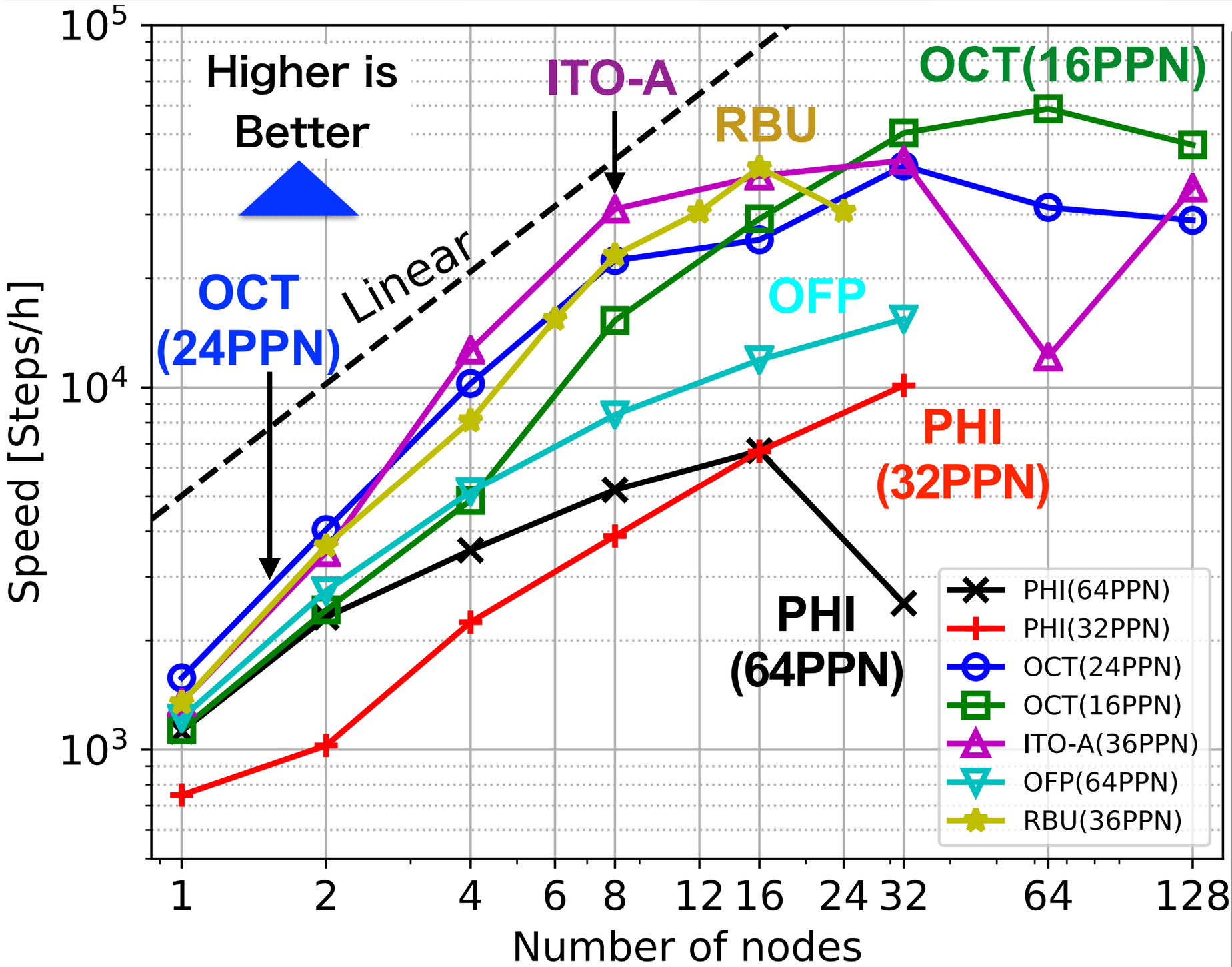
※1) 最適化フラグ: -O3，OakforestPACSとOCTPHIはO3 -DvectorMachine -xmic-avx512 ※2) unset KMP_AFFINIY;mpirun -env LD_PRELOAD libhbm.so -env HBM_SIZE 100 -env HBM_THRESHOLD 16 -env MPI_BUFFER_SIZE 1000000 -env I_MPI_PIN_PROCESSOR_EXCLUDE_LIST 0,1,68,69,136,137,204,205 -env KMP_HW_SUBSET 1T -env I_MPI_PIN_DOMAIN 4(または8)

OCTPUS Xeon Phiノードの解析速度比較



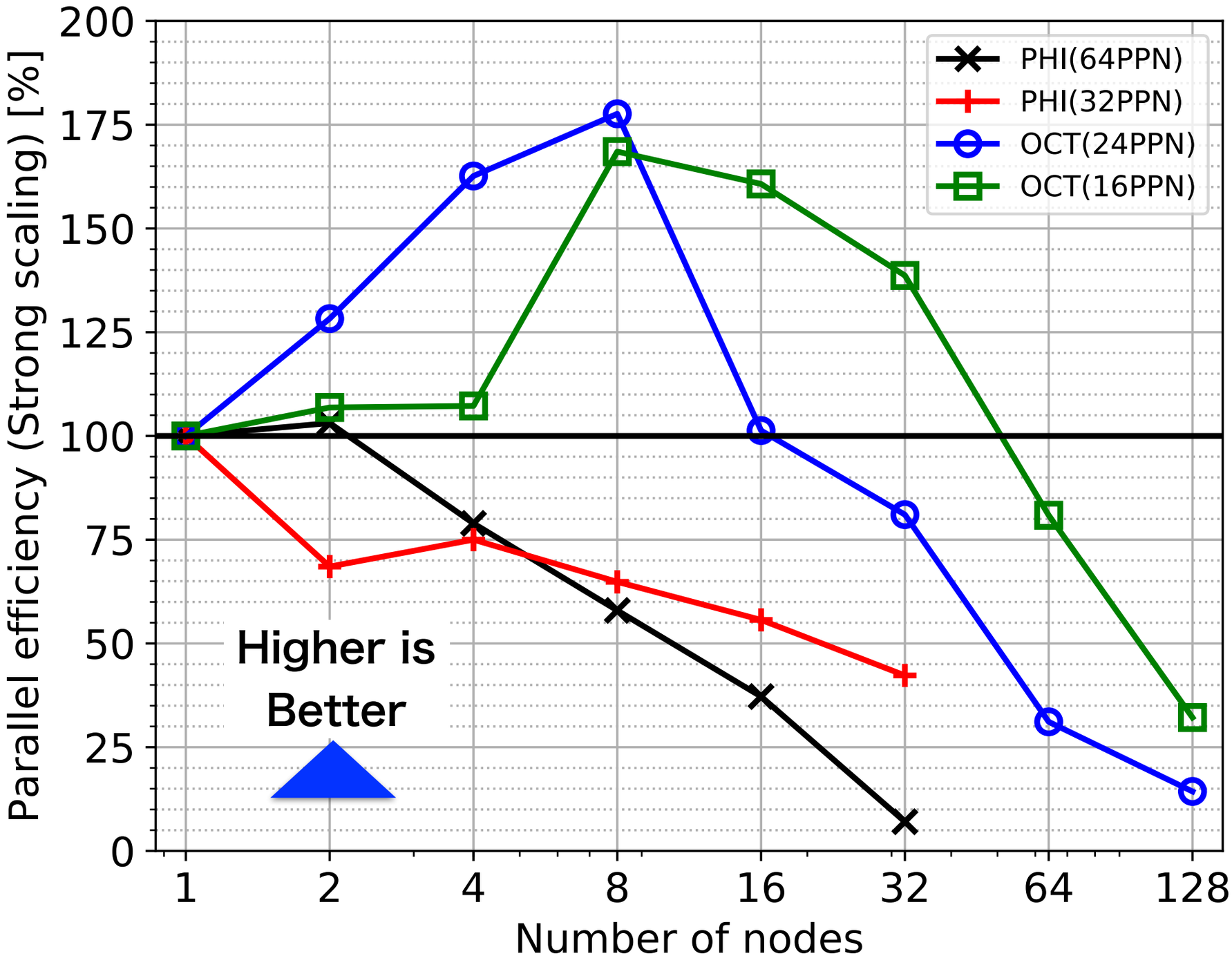
- DDR4のみ使用より，MCDRAMを使用(numactl -p 1)のほうが高速
- KNLオプション(-xmic-avx512)での高速化は僅か
- 16ノード以下ではフルコアの64PPNのほうが速いが，32ノードでは，コアがL2キャッシュを占有する32PPNのほうが高速

各システムの解析速度比較



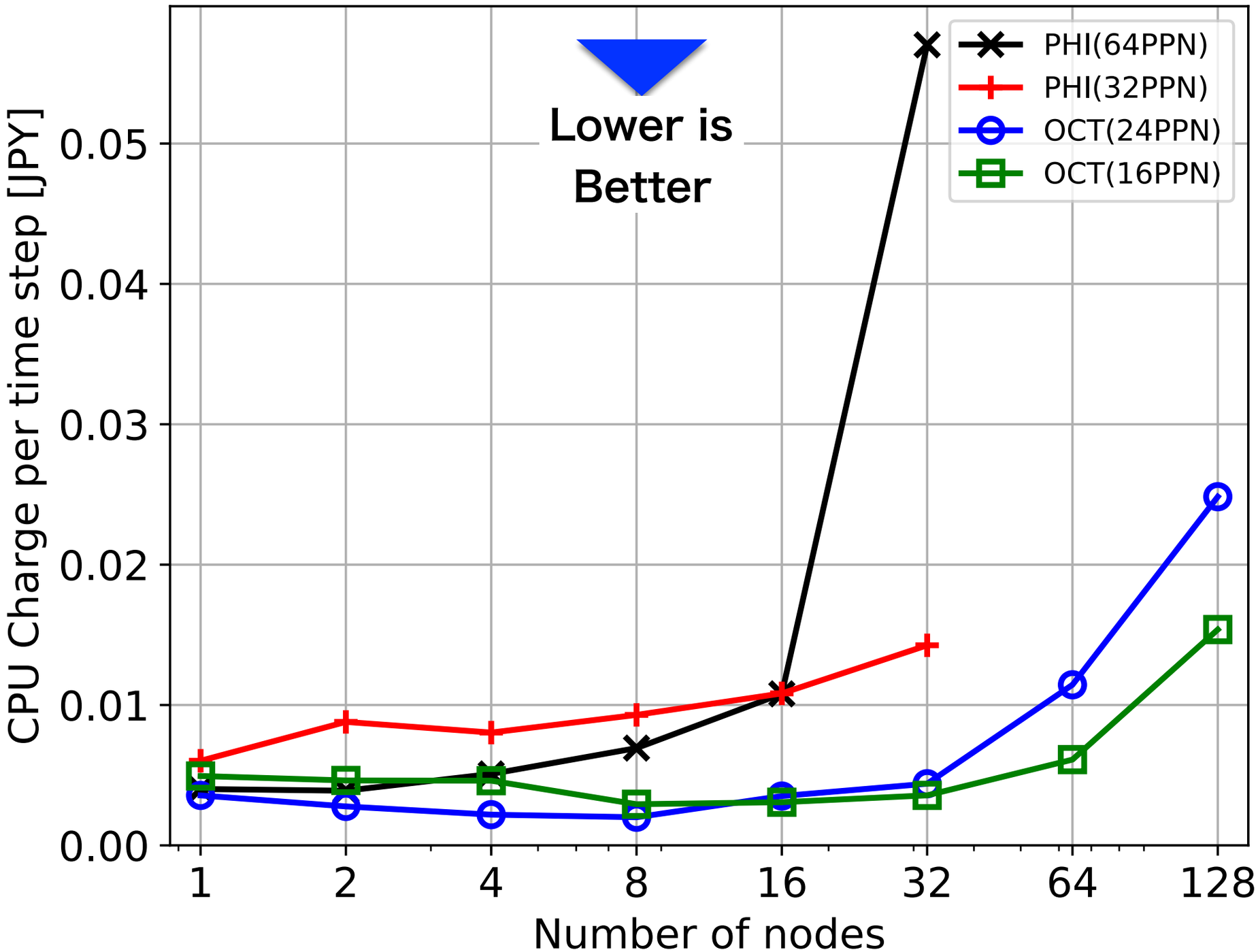
- PHI(OCTOPUSのXeon Phi)の64PPNと32PPNについては，最速条件のみ掲載
- OCT(OCTOPUSの汎用CPU)は8ノード以下ではフルコアの24PPNのほうが速いが，16ノード以上では16PPNのほうが高速
- OFPとPHIの速度比はノード数の増加で高くなる

OCTOPUSでの並列化効率



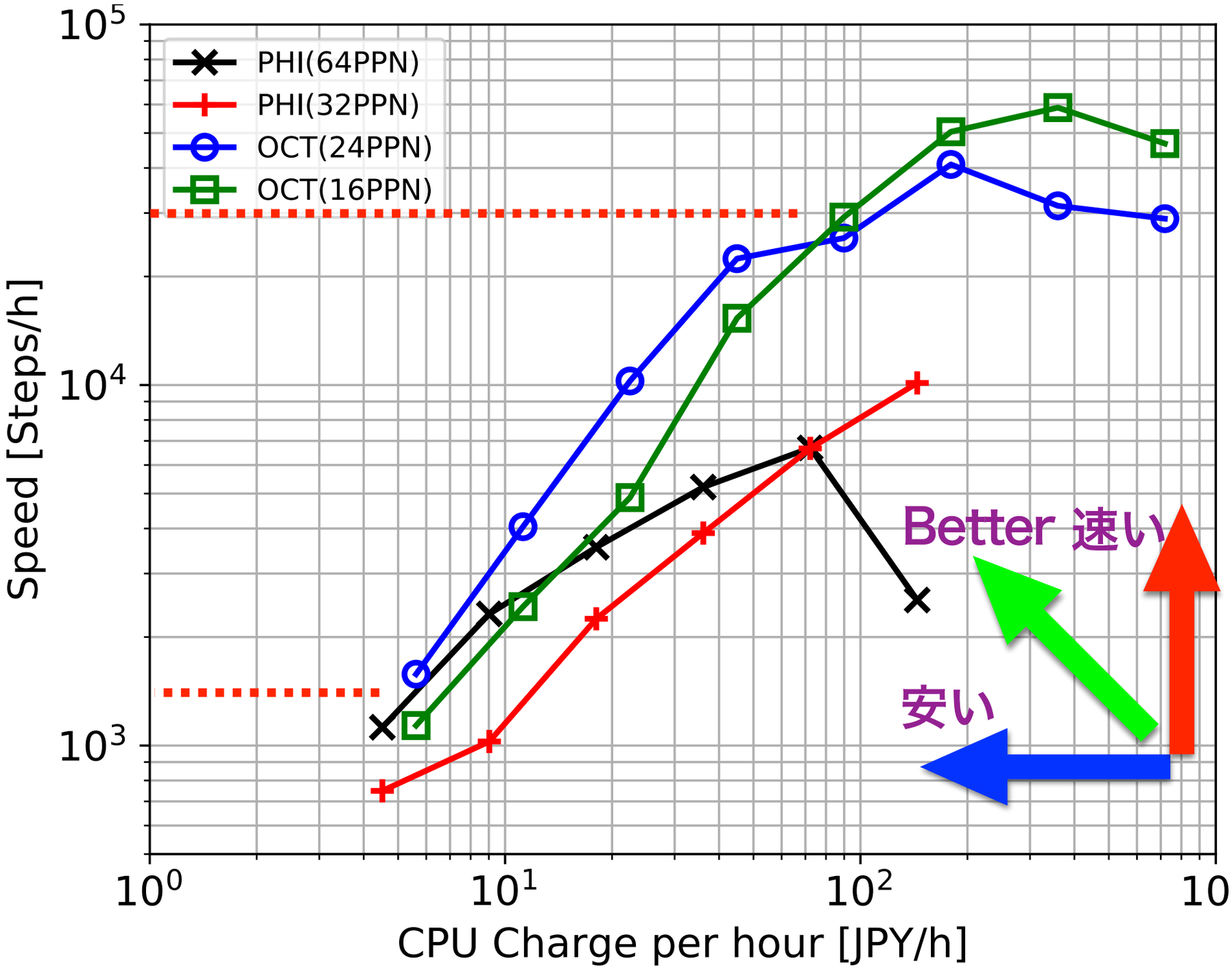
- 汎用CPU(Xeon)では，16~32ノードまでスーパーリニア

OCTOPUSでのステップ毎の課金比較



- 課金額は基本負担額10万円(税抜)の共有コースを仮定して算出
- 安価
  - ✓ 1~8ノード: OCT(24PPN)
  - ✓ 16~ノード：OCT(16PPN)
- 最安価：OCT(24PPN)を8ノード使用(並列化効率が最大)

OCTOPUSでの課金–解析速度曲線

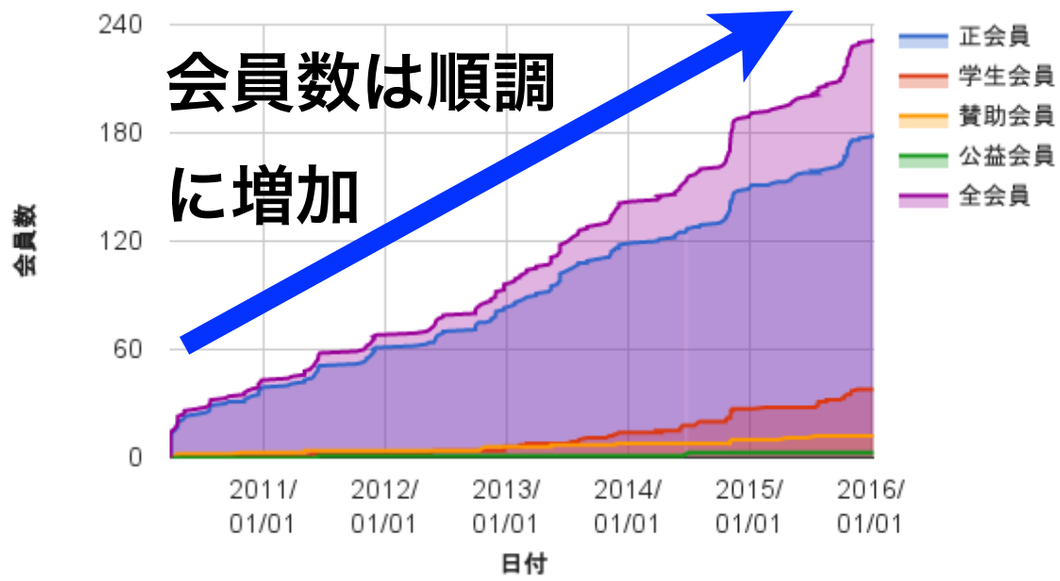


ステップ数	最安価
30,000 ～	OCT (16PPN) 16～ノード
～ 30,000	OCT (24PPN) 1～8ノード
～1500	PHI(64PPN) 1ノード

オープンCAE学会の紹介

- ✓ **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

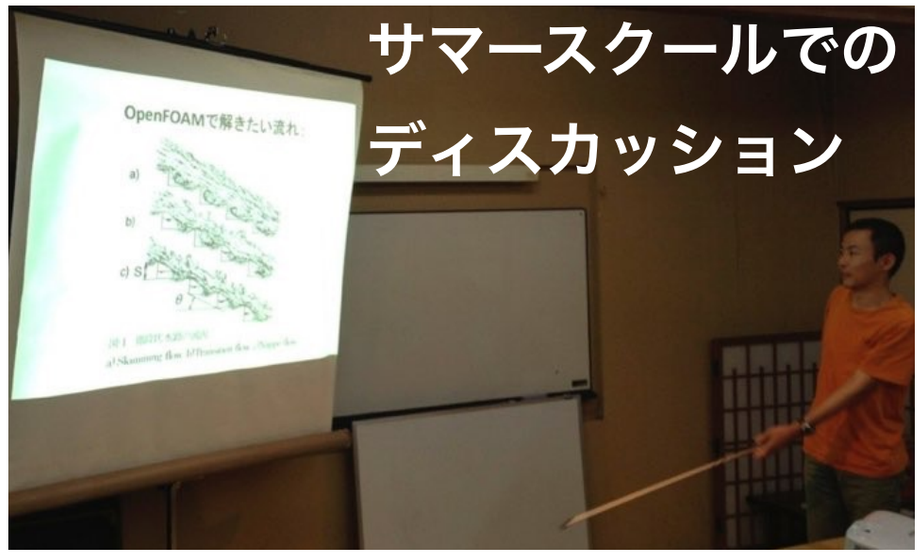


主な年間行事

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
12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×4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6日〜8日@川崎)**
  - ✓ 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

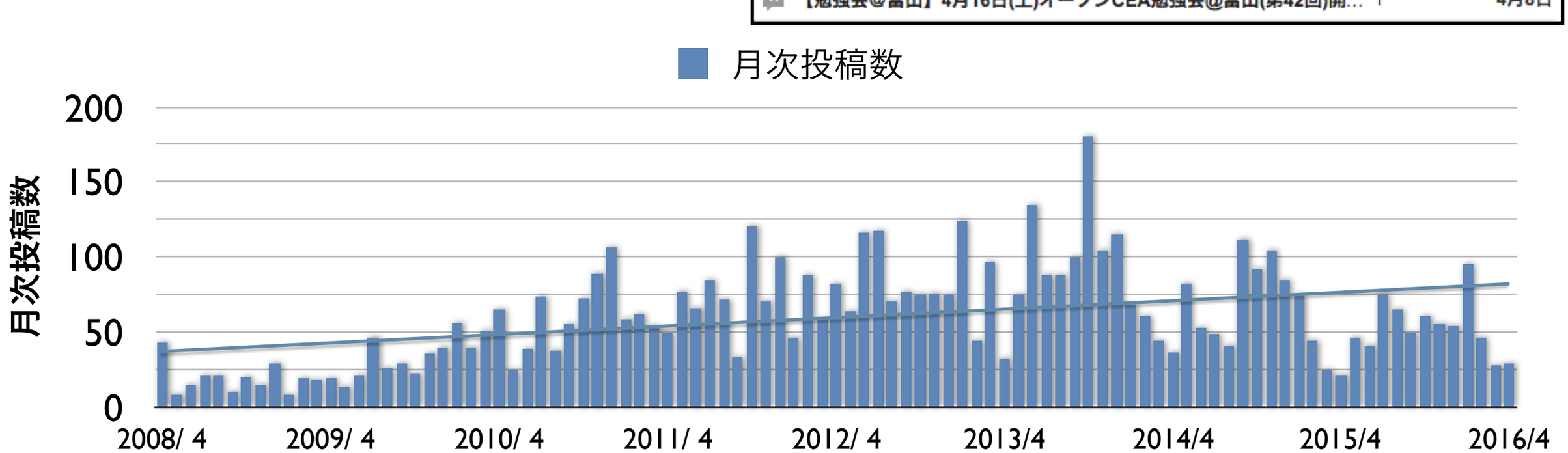
https://github.com/opencae/OpenFOAM

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

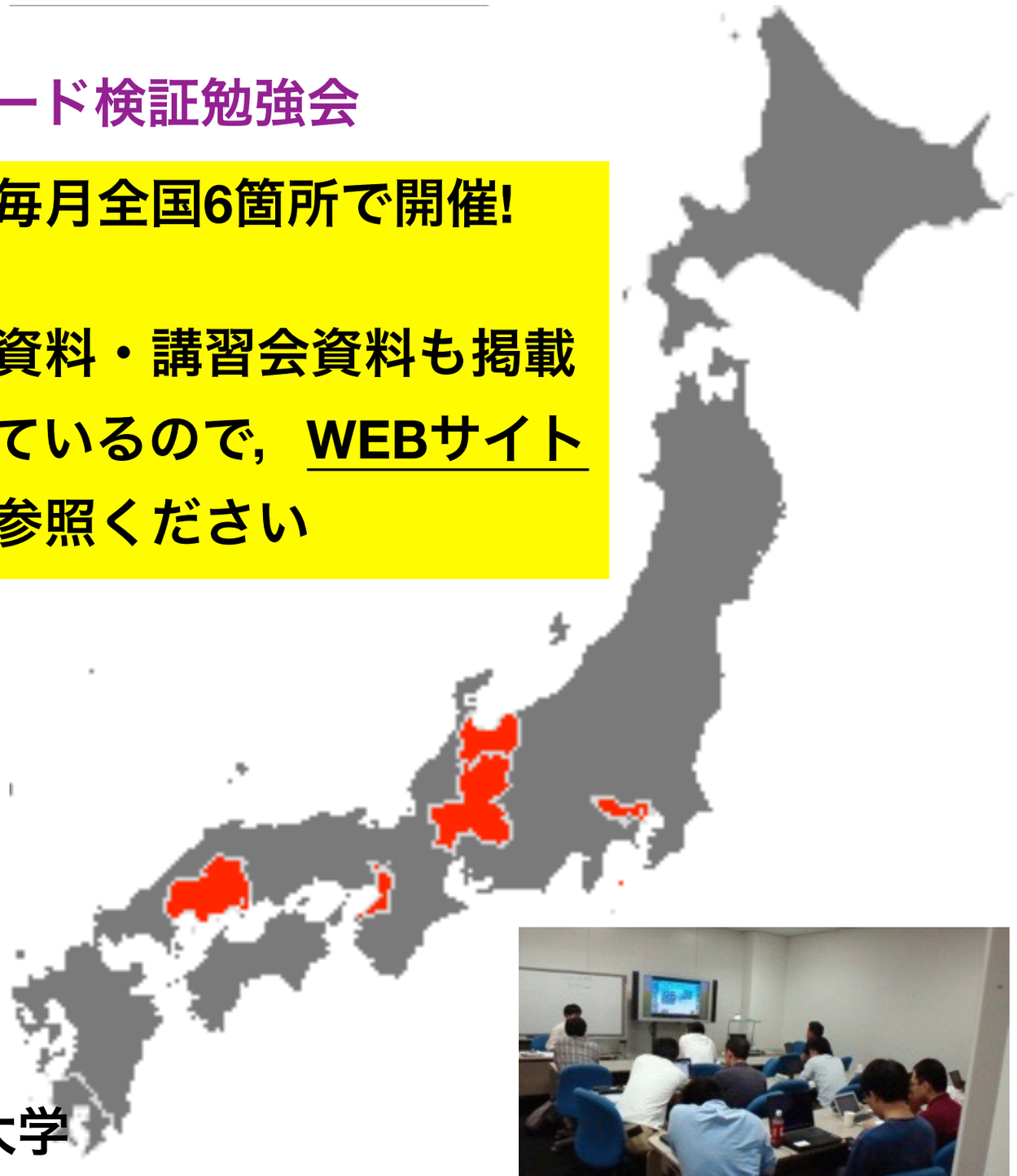
コミュニティ委員会: オープンCAE勉強会後援

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

合同勉強会(2018年6月23日)@東洋大学

ほぼ毎月全国6箇所で開催!

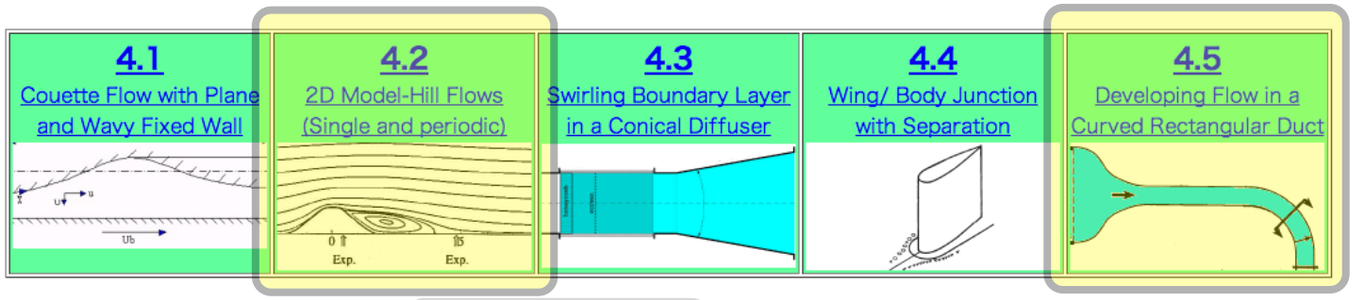
発表資料・講習会資料も掲載されているので，WEBサイトをご参照ください

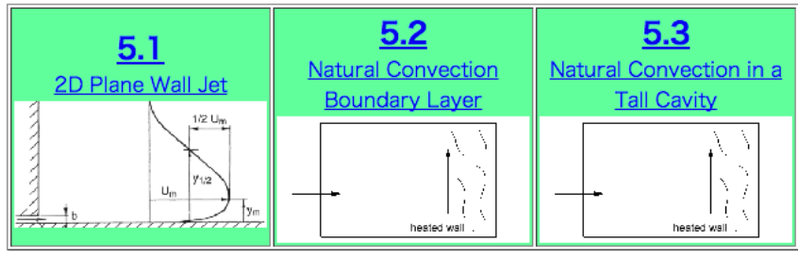


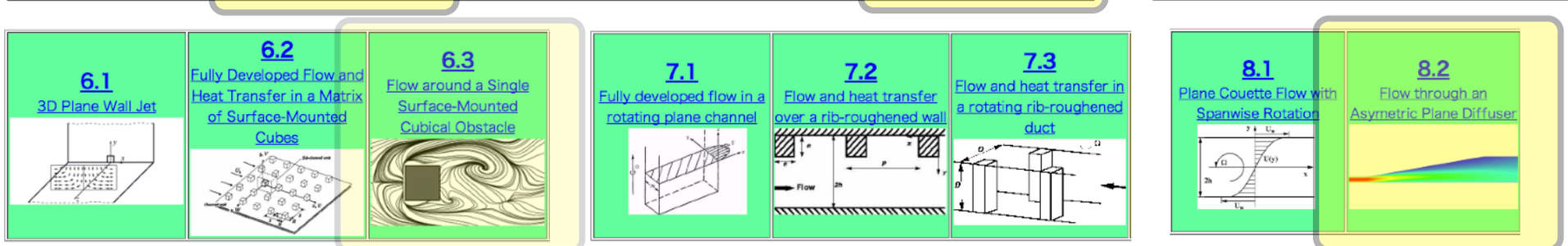
V&V委員会: ERCOFTAC SIG15ベンチマーク

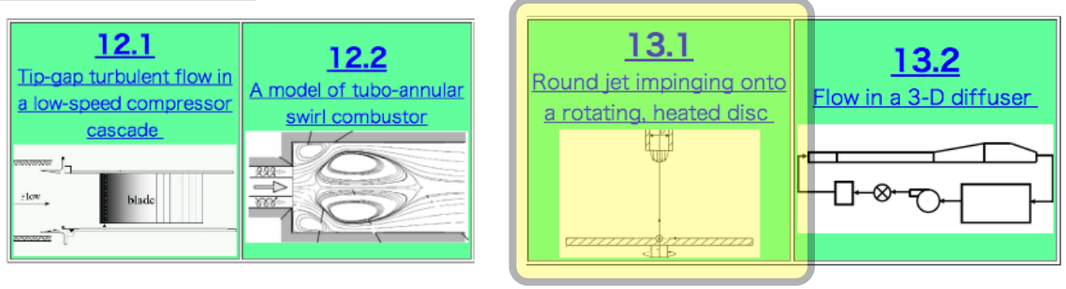
学会のレポジトリで実験値との検証ケースを公開．実験値との比較プロットも自動で可能

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

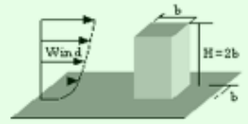
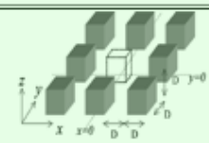
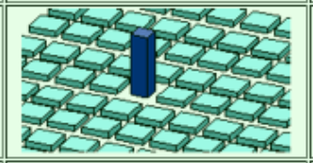
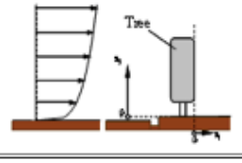
OpenFOAMで10ケースを実施・整備

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．

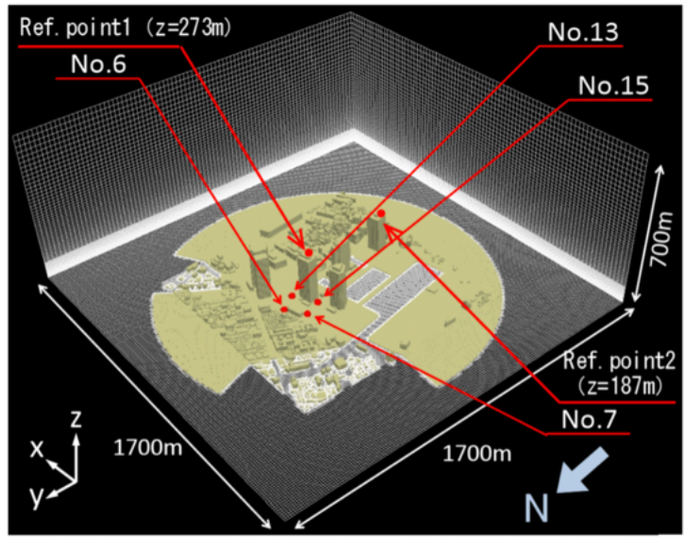
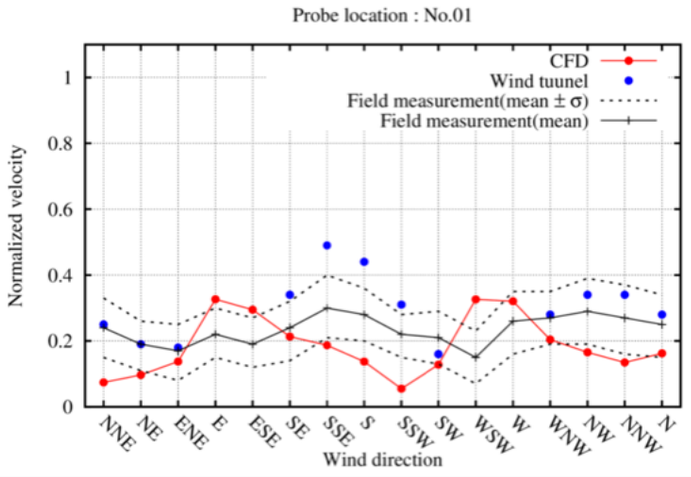


Fig. Calculation mesh (参考文献[Imano2010]から引用)

計算結果

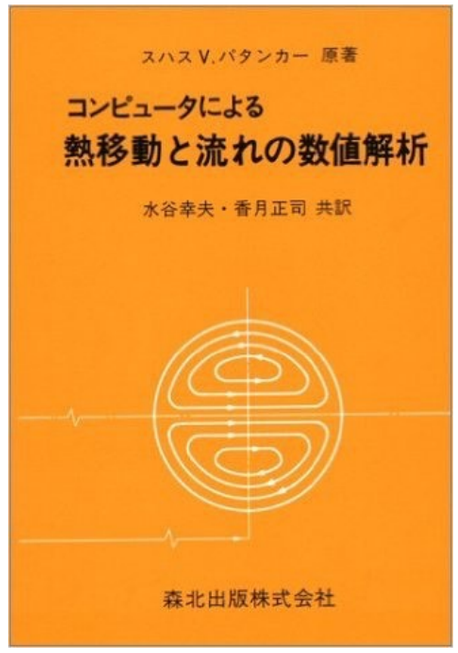
各計測点における風向別の風速比を以下の図に示す．風洞実験値及び実測値[AIJ]も合わせて示す．何れの結果も，風向NE～N～NWでは参照点1(D)，それ以外の風向は参照点2(C)の風速で基準化してある．



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

[OCT] OCTOPUSの利用方法（ http://www.hpc.cmc.osaka-u.ac.jp/system/manual/octopus-use/ ）必ず一度は目を通したほうが良い

[OCT-SMAP] OCTOPUS CPU Scheduling Condition **(** https://portal.hpc.cmc.osaka-u.ac.jp/secure/oct-smap/oct-smap.html ）ノード予約情報を見てからジョブを投入すると良い

[NQSII] NQSIIマニュアル**(** https://portal.hpc.cmc.osaka-u.ac.jp/secure/manual/OCTOPUS/J/NQSII_UsersGuide-Operation.pdf) ジョブシステムの詳細

[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://gitlab.com/OpenCAE/installOpenFOAM/blob/master/README.md ）スパコンでOpenFOAMを自動でビルドする