2019年9月4日 (Ver.2)

第122回お試しアカウント付き
並列プログラミング講習会
「OpenFOAM初級」

今野 雅

(東京大学情報基盤センター客員研究員，株式会社OCAEL)

目次

1　Ahmed bodyとは

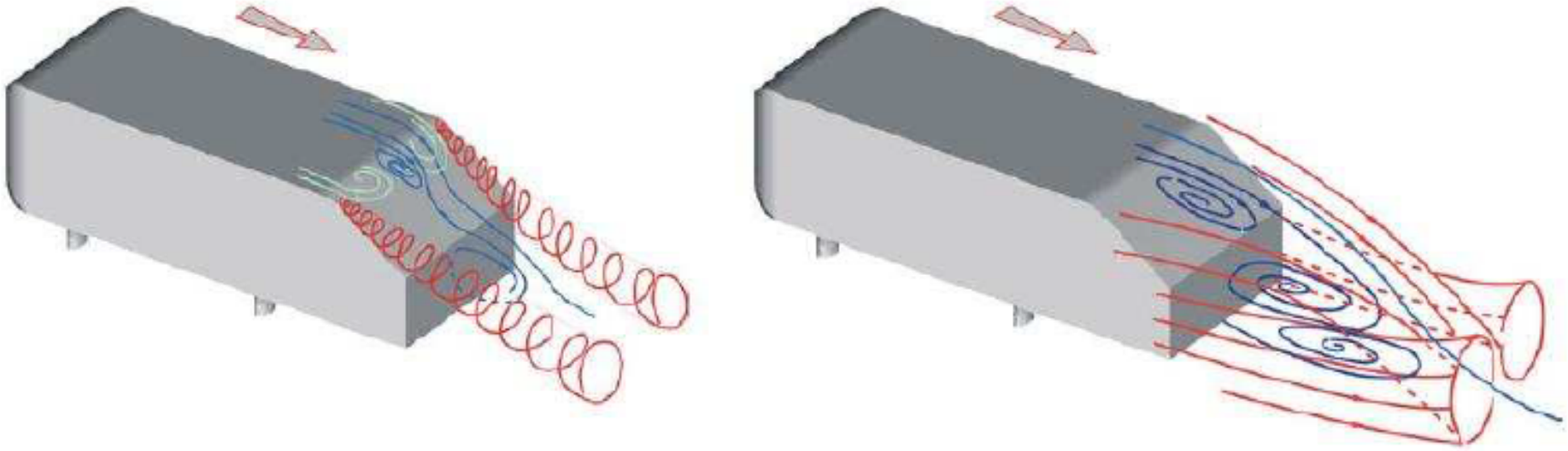


図1　Ahmed body後流の特性(左図：傾斜角度 $\theta = 25^\circ$ ，右図：傾斜角度 $\theta = 35^\circ$)[6]

- Ahmed bodyは，1984年に当時のドイツ航空宇宙研究所(DFVLR)，現在のドイツ航空宇宙センター(DLR)のSyed R. AhmedがSAE(Society of Automotive Engineers)に投稿した論文[3]において風洞実験を行なっている自動車モデルの略称である.
- このモデルは図1に示す通り，非常に単純化されたモデルであるため，風洞実験やCFD解析での再現が容易である.
- 一方で，車体後部上面の傾斜角度が $\theta = 25^\circ$ では，車体後部上面の両端に後引き渦が発生しているなど，複雑な流れ場となっている.

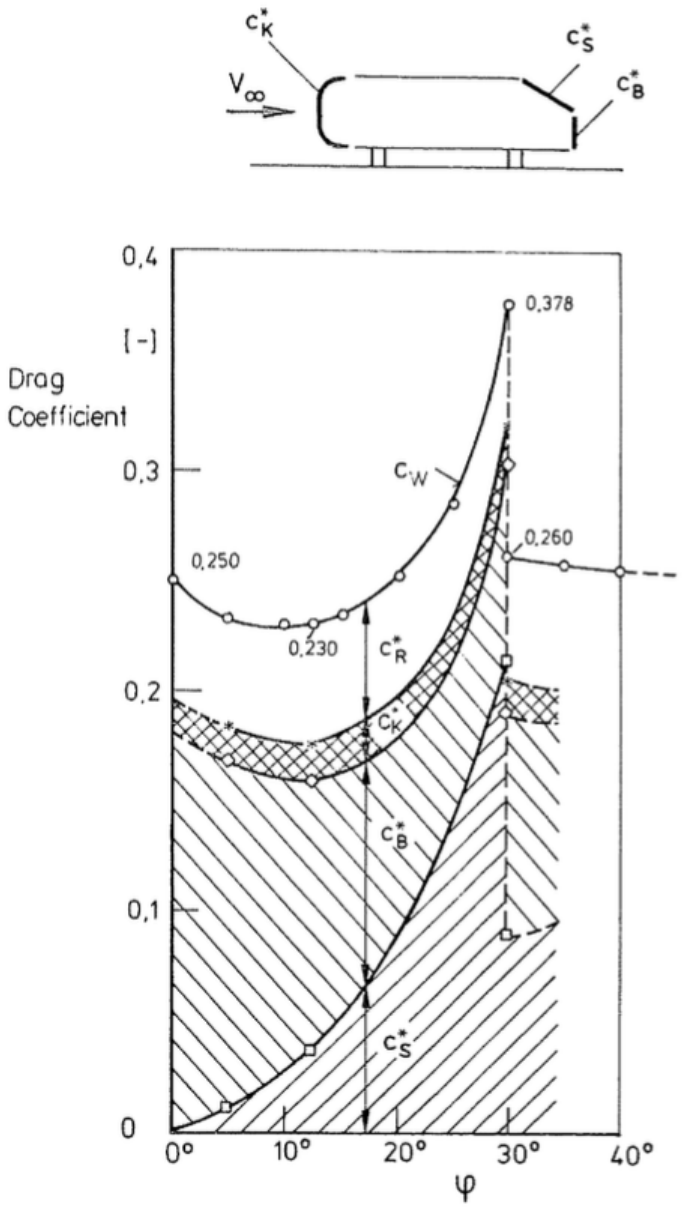


図2　傾斜角度による空気抗力係数の特性[3]

- 図2に示す通り，抗力係数C_d値が車体後部上面(スラント)の傾斜角度θが$25° \sim 30°$の範囲で急増し，$\theta = 30°$を境に急減する特徴を持つことから，スラント角によって引き起こされる車体後方領域の流れを調査するのによく用いられた.

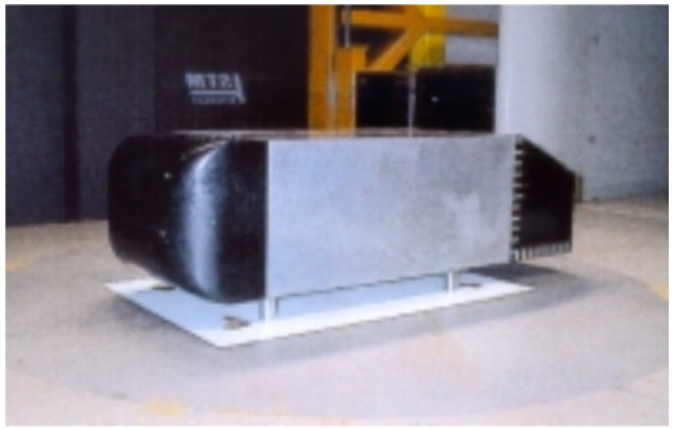


図3　LSTM低風速風洞での Ahmed bodyの実験)[5]

図4　オイルと煤によるAhmed body後部表面の流脈可視化(左図：傾斜角度 $\theta = 25^\circ$，右図：傾斜角度 $\theta = 35^\circ$)[5]

- ヨーロッパのModels for Vehicle Aerodynamics(MOVA) プロジェクトとして，ドイツ，エアランゲン大学の流体力学研究室(通称LSTM)のLienhartら[5]により，図3に示すLSTMの低風速風洞において，スラント角が25^o，35^oのAhmed bodyについて，2成分レーザードップラー風速計(LDA)を用いた詳細な流速の計測や，図4に示すスラント表面の流脈可視等が行われた.
- これらの実験結果およびAhmed bodyの形状データは，ERCOFTAC(European Research Community On Flow, Turbulence And Combustion)のClassicケース8.2としてWeb[1]で掲載されており，簡単に詳細なデータ入手できることから，自動車空力解析用のベンチマークモデルとして良く用いられる.

2　解析モデル

- 車体の形状は車長は $D = 1.044$ m，車幅は $W = 0.389$ m，車高は $h = 0.288$ m である.
- 本演習では，車体後部上面の傾斜角度(以下，スラント角)は $\varphi = 25°$ とする.

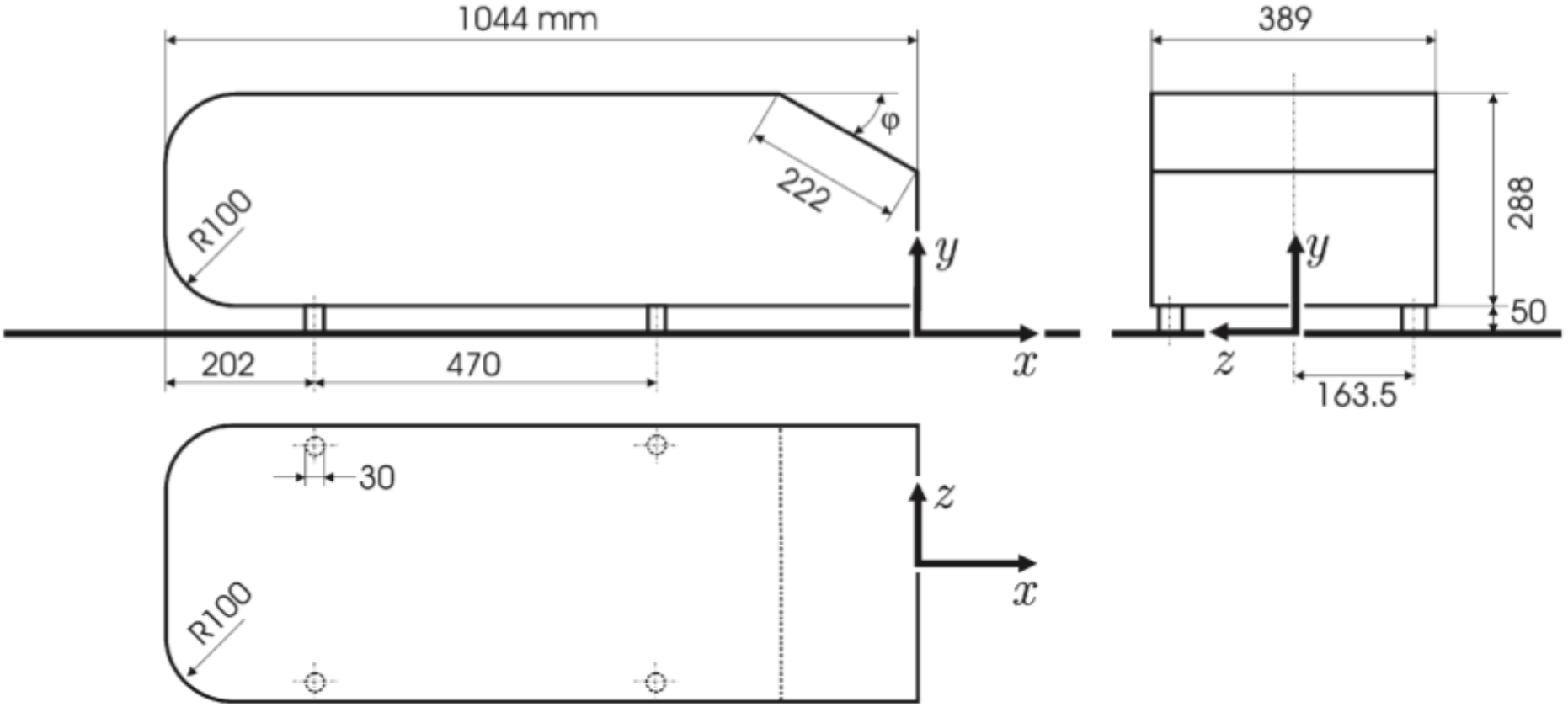


図5　Ahmed Bodyモデル[4]

3　基本的な解析条件

- RANS乱流モデルを用いた非圧縮性流体の定常解析を行う.
- 基本的な解析条件は，ERCOFTAC[1]記載の表1を用いる.

表1　解析条件

風洞高さ	$H = 1.4$ [m]
空気の動粘性係数	$\nu = 15 \times 10^{-6}$ [m^2/s]
流入速度	$U = 40$ [m/s] (風洞実験でのバルク風速)
車高	$h = 0.288$ [m]
レイノルズ数(車高ベース)	$Re = Uh/\nu = 768,000$ [-]

4　CFD解析条件

表2　CFD解析条件

上面・側面境界	すべり壁(slip wall)
壁面境界	すべり無し壁(no-slip wall)，滑面に対する壁関数
乱流モデル	標準$k-\omega$ SSTモデル
流入面の乱れ強さ	$I=0.25$[%]
流入面の混合長さ	$l=0.07H=0.098$[m]
移流項の離散化スキーム	全ての場についてTVD(limited linear)
解析ソルバ	simpleFoam(定常乱流解析ソルバ)
圧力・速度連成手法	SIMPLEC法

- 風洞実験では天井面と側面は開放されていたが，CFD解析ではすべり壁とする.
- 床面と車体表面はすべり無し壁とし，壁関数を設定する.
- 乱流モデルは k - ω SSTモデルを使用する.
- 風洞実験での測定を行なったドイツ・バイエルン州・エアランゲン大学のLSTM低風速風洞の流入風は，平均乱流強度が0.25%以下と記載されているので，乱流強度Iは0.25%.
- 混合長さlは，風洞高さをH=1.4[m]として，$l=0.07\times H=0.098$[m].

5　解析領域

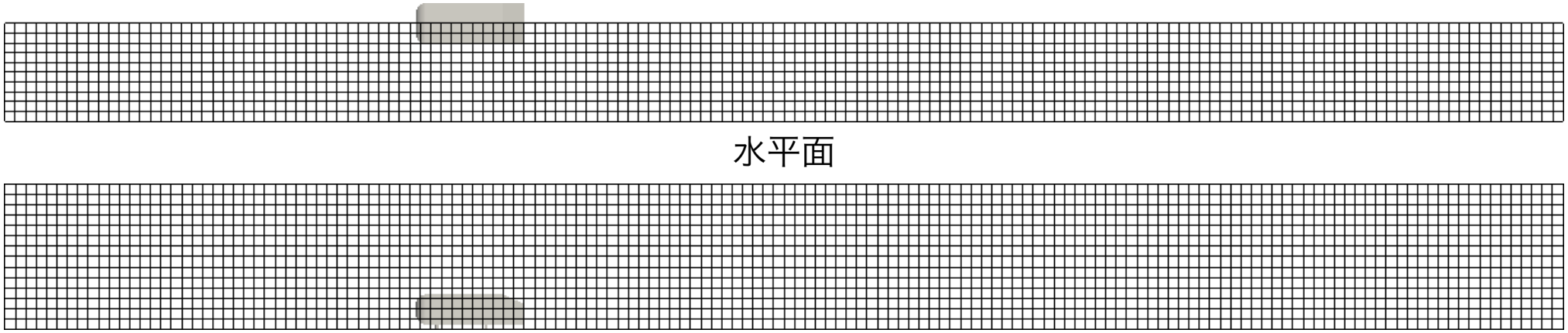


図6　解析領域の格子分割とAhmedbodyモデル

- 本解析では，車体が対称面を持つため，解析領域も空間対称に設定することが可能である.
- また，RANS乱流モデルを用いた定常解析を行うので，車体の対称面に対して，流れ場の定常解も対称であると仮定し，対称面を設けた半分の領域の解析を行う.
- 解析領域は15[m] $\times 0.935$[m] $\times 1.4$[m]とする.
- 計算格子はベース格子の分割幅Δxが概ね0.1[m]となるように分割数を指定する.
- そのベースメッシュを基準とし，車体周辺の流れを捉えるための再分割領域の設定と，形状を表現するための細分割の設定を行う.

6　解析ケースの作成

- OpenFOAMでは，基本的に付属しているチュートリアルモデルの中から解きたい解析モデルをコピーして編集するのが一般的な解析ケースの作成方法である.
- 本解析は非圧縮性流体解析であるから，カテゴリーは*incompressible*であり，また定常乱流解析のため，ソルバーはsimpleFoamとなる.
- 対応するチュートリアルのディレクトリから，本解析に最も近いケースを探すと，バイクの空力解析*motorBike*および車の2次元空力解析*simpleCar*となるので，これらを参考に設定を作成する方法がある.
- 今回は，演習の時間が限られているので，既に作成済みの解析ケースをコピーする.

Oakforest-PACSにログインして，以下のコマンドを実行し，講習会用ファイルを展開する.
なお，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っている.

計算用ディレクトリへの移動と講習会用ファイルの展開

```
cd /work/gt00/$USER
cp -a /work/gt00/share/lecture-OpenFOAM ./
ln -s /work/gt00/$USER/lecture-OpenFOAM ~/
```

作成済の解析ケースのコピー

```
cd ~/lecture-OpenFOAM/AhmedBody/Ahmed25
cp -a template case-1
cd case-1
```

7　解析手順

1. **前処理 (スクリプト: *ofp0pre.sh*)**
（a）特徴辺抽出 (surfaceFeatureExtract)
（b）ベース格子生成 (blockMesh)
（c）並列格子生成用領域分割 (decomposePar -decomposeParDict 辞書名)
（d）並列格子生成 (snappyHexMesh -decomposeParDict 辞書名 -overwrite -parallel)
（e）格子の品質チェック (checkMesh -decomposeParDict 辞書名 -constant)
（f）格子の再構築(注) (reconstructParMesh -constant)
2. **流体解析用の領域分割 (スクリプト: *ofp1de.sh*)**
（a）流体解析用の領域分割 (decomposePar -force)
3. **初期値作成 (スクリプト: *ofp2solveInit.sh*)**
（a）格子順変更による行列バンド幅縮小 (renumberMesh -overwrite -parallel)
（b）ポテンシャル流れを解いて初期値作成 (potentialFoam -parallel)
4. **流体解析 (スクリプト: *ofp3solve.sh*. 性能プロファイル実行時は *solveVTune.sh*)**
（a）ソルバ実行 (simpleFoam -parallel)
5. **再構築 (スクリプト: *ofp4re.sh*)**
（a）計算結果の再構築 (reconstructPar -latestTime)
6. **後処理 (スクリプト: *ofp5post.sh*)**
（a）解析結果のサンプリングなど (postProcess -func sample -parallel など)
7. **結果プロット (スクリプト: *ofp6plot.sh*)**

8　前処理

Code 1　ofp0pre.sh

```
1  #!/bin/bash
2  #PJM -S
3  #PJM -g gt00
4  #PJM -L elapse=0:15:00
5  #PJM -L rscgrp=tutorial-flat
6  #PJM -L node=1
7  #PJM --mpi proc=64
8  source ofpshare.sh # 共通の設定
9  $numactl surfaceFeatureExtract >& log.surfaceFeatureExtract # 特徴辺抽出
10 $numactl blockMesh >& log.blockMesh # ベース格子生成
11 $numactl decomposePar -decomposeParDict system/decomposeParDict.mesh \
12   >& log.decomposePar.mesh # 格子生成用領域分割. -decomposeParDictで領域分割の辞書指定
13 $mpirun $numactl snappyHexMesh -decomposeParDict system/decomposeParDict.mesh \
14   -overwrite -parallel >& log.snappyHexMesh # 並列格子生成
15 $mpirun $numactl checkMesh  -decomposeParDict system/decomposeParDict.mesh \
16 -parallel -constant >& log.checkMesh # 格子の品質チェック
17 $numactl reconstructParMesh -constant >& log.reconstructParMesh # 格子の再構築
```

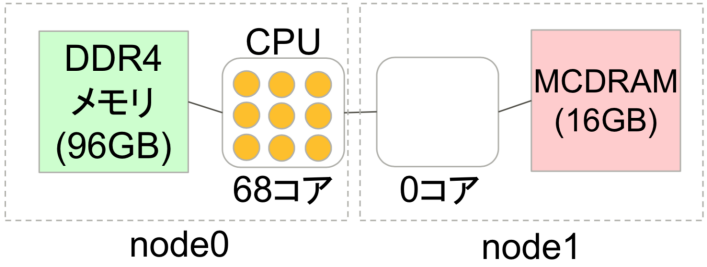
ジョブ投入コマンドpjsubに与えるオプションを#PJM 以降に記述できる．また，これらの指定は，ジョブ投入コマンドpjsubのオプションにより上書きされる．

- -S：ジョブ統計情報とノードごとの詳細情報をファイルに出力

- `-g gt00`：課題番号=gt00(講習用)
- `-L elapse=0:15:00`：経過時間制限=15分(講習用リソースグループの最大値)
- `-L rscgrp=tutorial-flat`：リソースグループ=tutorial-flat(講習会の時間のみ有効)
  講習会終了後はリソースグループをlecture-flatに書き変えるか，`pjsub`のオプションに`-L rscgrp=lecture-flat`を指定する.
- `-L node=1`：ノード数=1
- `-mpi proc=64`：総プロセス数=64 (ノード毎のプロセス数ではない事に注意)

Code 2　ofpshare.sh

```
1 module purge # 標準で有効なmoduleに依存しないよう全moduleをunload
2 module load gcc/4.8.5 # Gccコンパイラのmoudleをload
3 source /work/gt00/share/OpenFOAM/OpenFOAM-v1712/etc/bashrc \
4 WM_COMPILER=Gcc4_8_5 WM_MPLIB=INTELMPI2017_3_196 # OpenFOAM環境設定
5 source /usr/local/bin/mpi_core_setting.sh # MPIプロセスのピニング
6 export I_MPI_DEBUG=5 # MPIプロセスのピニング等のデバッグ情報取得
7 env # 念のため実行時の環境変数を記録しておく
8 numactl="numactl␣-p␣1" # numactl -p 1 : MCDRAMをできるだけ使用
9 mpirun="mpiexec.hydra␣-n␣$PJM_MPI_PROC"  # MPI実行用コマンド($PJM_MPI_PROC=プロセス数)
```



リソースグループが*-cacheの時はCacheモードとなり，高バンド幅メモリのMCDRAMはキャッシュメモリとして利用されるので，特に設定の必要はないが，リソースグループが*-flatの時はFlatモードとなり，MCDRAMを優先的に使うには，numactlなどで設定する．図引用元[2]

Tips

- 並列格子生成による格子数が非常に多い場合等，格子の再構築と本解析用の領域再分割も行わない手順もあるが，本手順では格子生成と流体解析の領域分割が独立となり自由度が高く，計算効率も上げやすい．
- 特徴辺抽出(surfaceFeatureExtract)，ベース格子生成(blockMesh)，領域分割(decomposePar)，格子再構築(reconstructParMesh)などの並列処理できないユーティリティは，後述するような複数ノードを使用するジョブ内ではなく，*ofp0pre.sh*のように1ノードで実行するジョブで実行させるほうがトークンを節約できるので，適宜ジョブスクリプトを分けるほうが良い．
- 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，できるだけ適切な経過時間制限値を`-L elapse`で設定する．

8.1 前処理ジョブの投入

ジョブスクリプト*ofp0pre.sh*をpjsubで投入する．

ジョブの投入

```
pjsub ofp0pre.sh
# 講習会終了後は pjsub -L rscgrp=lecture-flat ofp0pre.sh
```

以下のコマンドにより，投入したジョブの状況を確認する．

ジョブの状況確認

```
pjstat
```

本演習において，今後pjsubでジョブを投入したら，pjstatで適宜ジョブの状態を確認する．ジョブが実行状態になったら，tailコマンドを用いて，ログのトレースを行う．

ログファイルのトレース

```
tail -f log.*
```

ユーティリティ毎にログファイルが異なるので，Endなどのメッセージが出るなどしてログファイルの更新が止まったら，Ctrl+C(コントロールキーとCキー)を押してtailコマンドを一旦終了させて，カーソル↑＋リターン，または，!!で再度上記コマンドを実行する．

9　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

図引用元[2]

ピニングを適切に行わないと，プロセスが偏って割り当てられ，性能が劣化する可能性が高い．最初は必ずピニング結果を確認する

先頭の物理コアは，CPUスケージューリングクロック割り込み抑止設定(動的Tick-less)がされていないので，なるべく先頭タイルは使用しない

一般的に，ノード当り66プロセス以下のL1専有より，ノード当り33プロセス以下のL2専有のほうが，メモリ帯域幅も多く使用でき，ピーク性能が高くなるが，L1専有のほうが費用(トークン)対性能が高い事も多いので，事前のベンチマークテストが重要

10　特徴辺抽出 (surfaceFeatureExtract)

- 格子生成において特徴辺を再現するため，形状データから特徴辺を抽出する.
- 特徴辺の抽出する形状を *system/surfaceFeatureExtractDict* で定義する.
- 辺に隣接する2つの三角形の法線ベクトルの成す角度が `includedAngle` 以下であれば特徴辺として抽出される.
- ここでは，車体の先端の特徴辺を取り込み，かつ平面内の分割線を特徴辺として認識しないよう `includedAngle` を $179.9[^o]$ としている.

Code 3　system/surfaceFeatureExtractDict

```
17 AhmedBody.stl // 三角分割表面形状ファイル名
18 {
19     // 特徴辺の抽出法
20     extractionMethod    extractFromSurface; // 表面形状ファイルから抽出
21
22     extractFromSurfaceCoeffs
23     {
24         // 隣接する2つの面の法線の角度が以下より小さい辺を特徴辺とする
25         includedAngle   179.9;
26     }
27
28     // 可視化用に特徴辺をobj形式で出力
29     writeObj                yes;
```

11　ベース格子生成 (blockMesh)

11.1　ベース格子での解析領域の指定

- ベース格子の作成にはblockMeshを用いる.
- 計算格子はベースメッシュの幅が約$\Delta x = 0.1$[m]となるように分割数を指定し，各境界面にもパッチ名を付与する.
- 車体後部が $x = 0$ ，車体の対称面が $y = 0$ ，床面が $z = 0$ となるように解析領域(15[m] $\times 0.935$[m] $\times 1.4$[m])を指定する.

Code 4　system/blockMeshDict

```
20 xmin -5;
21 xmax 10;
22 ymin -0.935;
23 ymax  0;
24 zmin  0;
25 zmax  1.4;
26
27 vertices
28 (
29     ( $xmin $ymin $zmin ) // 0
30     ( $xmax $ymin $zmin ) // 1
31     ( $xmax $ymax $zmin ) // 2
32     ( $xmin $ymax $zmin ) // 3
```

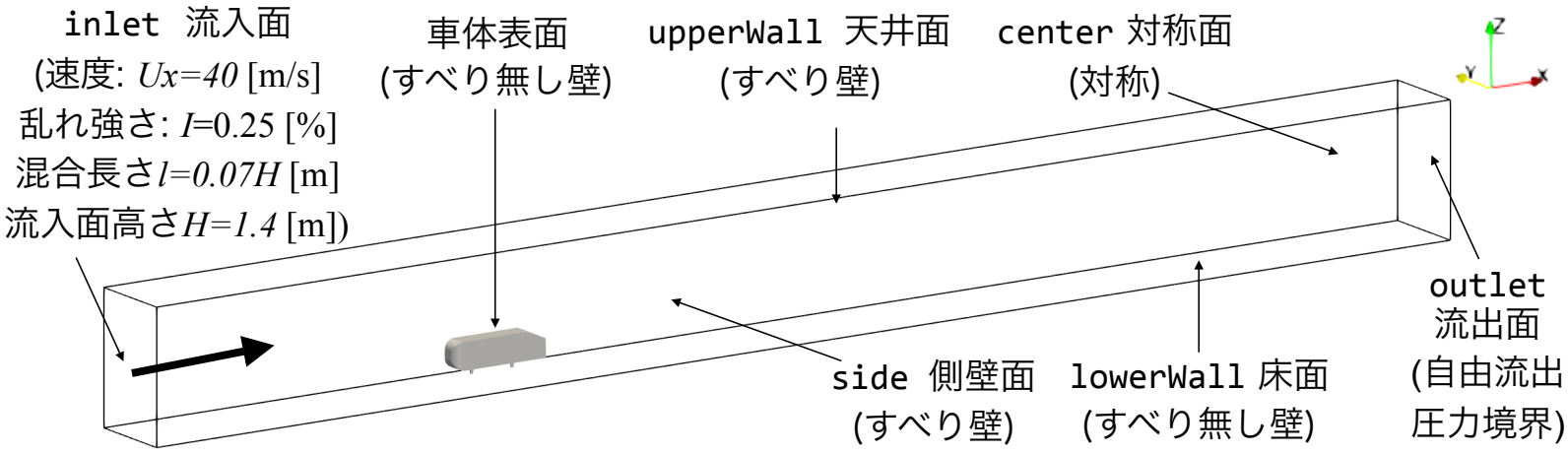
```
33      ( $xmin $ymin $zmax ) // 4
34      ( $xmax $ymin $zmax ) // 5
35      ( $xmax $ymax $zmax ) // 6
36      ( $xmin $ymax $zmax ) // 7
37 );
```

11.2　ベース格子での分割数・パッチの指定

ベース格子の分割幅Δxが，約0.1[m]となるように分割数を指定する．また，解析領域の各面に対して，以下のようにパッチを指定する．

表3　解析領域の境界面

境界名	パッチ名	パッチ型
対称面	center	symmetry
側壁面	side	wall
流入面	inlet	patch
流出面	outlet	patch
床面	lowerWall	wall
天井面	upperWall	wall



Code 5　system/blockMeshDict

```
38 blocks
39 (
40     hex ( 0 1 2 3 4 5 6 7)  (150 10 14) simpleGrading (1 1 1)
```

```
41 );
42 boundary
43 (
44     center
45     {
46         type  symmetry;
47         faces ((3 7 6 2));
48     }
49     side
50     {
51         type  wall;
52         faces ((1 5 4 0));
53 
54     }
55     inlet
56     {
57         type  patch;
58         faces ((0 4 7 3));
59 
60     }
61     outlet
62     {
63         type  patch;
64         faces ((2 6 5 1));
```

```
    }
    lowerWall
    {
        type  wall;
        faces ((0 3 2 1));
    }
    upperWall
    {
        type  wall;
        faces ((4 5 6 7));
    }
);
edges
(
);
```

12　並列格子生成用領域分割 (decomposePar)

- 生成する格子が複雑になるに従って，snappyHexMeshの計算時間が長くなる.
- 特にXeon Phiではコアの性能が低く時間がかかるので，できれば並列計算する.
- 今回格子生成用の領域分割の設定は，本解析用の領域再分割とは別に行うため，通常の *system/decomposeParDict* ではなく，*system/decomposeParDict.mesh* で行う.
- `numberOfSubdomains` で領域分割数(並列数)，`method` で分割手法を指定する.
- ここでは，1ノードあたり68物理コアのうち64コアを用いて並列計算を行う(2の階乗のほうが `simple` や `hierarchical` での分割数指定が容易であり，MPI通信も一般に高速).

Code 6　system/decomposeParDict.mesh

```
18 numberOfSubdomains 64; // 領域分割数(並列数)
19 method      hierarchical; // 分割手法: hierarchical(分割順序が指定できる)
20 hierarchicalCoeffs
21 {
22     n           (4 4 4); // x,y,z方向の分割数
23     delta       0.001;
24     order       xyz; // 分割順序
25 }
```

13　並列格子生成 (snappyHexMesh)

blockMeshで解析領域を作成したので，次はsnappyHexMeshで車両周りのメッシュ作成，細分割領域の作成，境界層レイヤーの挿入を行う．ここでは，snappyHexMeshを用いて，細かなメッシュ操作について説明する．

13.1　三角分割表面形状の定義

snappyHexMeshの設定ファイル*snappyHexMeshDict*のgeometryで形状の指定と細分割領域を定義する．基本的な記述方法としては，形状の種類，形状の識別名，形状の領域を指定する．形状の識別名は細分割や境界層の挿入のメッシュ操作の際に用いられる．

Code 7　system/snappyHexMeshDict

```
28 geometry
29 {
30     // CADデータのファイル名(constant/triSurfaceディレクトリ内)
31     AhmedBody.stl
32     {
33         type triSurfaceMesh; // 形状の種類
34         name AhmedBody;      // 形状の識別名
35     }
```

13.2　直方体形状の定義

本解析モデルでは車両周りの流れを捉えるために4段階の細分割領域を定義した．車両に近づくにつれて細かくなるように領域refinebox1-refibebox4を定義した．

Code 8　system/snappyHexMeshDict

```
37      refinebox1  // 形状の識別名
38      {
39          type searchableBox;              // 形状の種類
40          min ( -2.610 -0.935  0.0   ); // 直方体の座標最小値
41          max (  2.610  0.935  0.845 ); // 直方体の座標最大値
42      }
```

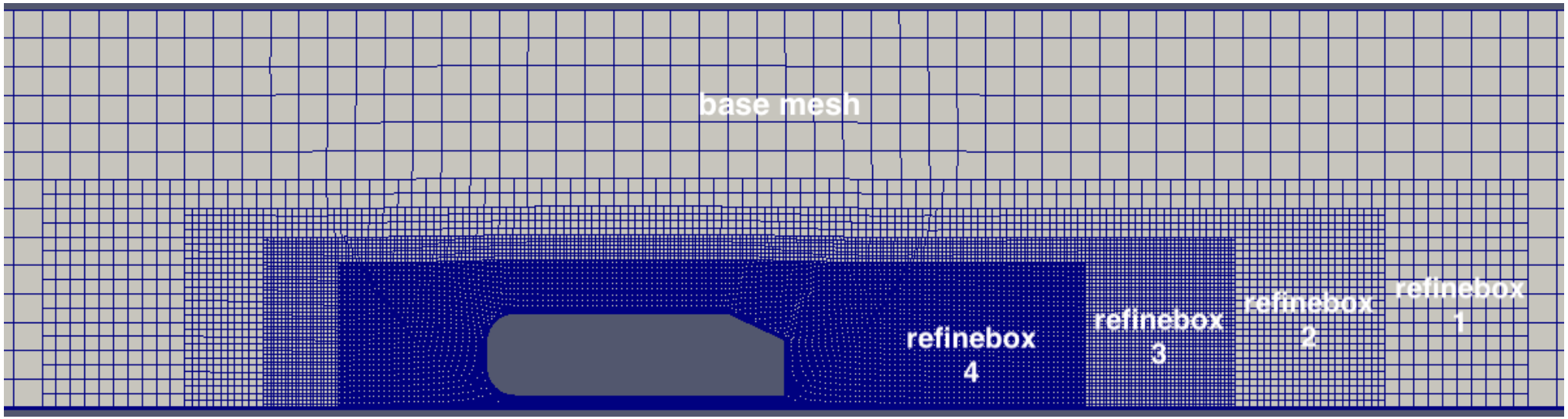


図7　領域の細分割

13.3　特徴辺の設定

後述するsurfaceFeatureExtractで抽出される特徴辺に対する分割レベルを設定する.

Code 9　system/snappyHexMeshDict

```
81      features
82      (
83          {
84              file "AhmedBody.eMesh";   // 特徴線ファイル
85              level 4;                  // 分割レベル
86          }
```

13.4　階段状格子の設定パラメータ

階段状格子を作成するために必要なメッシュ数の制限や格子間のレイヤ数に関するパラメータを設定する. 指定した細分割レベルまで細分割が進むように, maxLocalCells や maxGlobalCellsは大きくしておく.

Code 10　system/snappyHexMeshDict

```
67  castellatedMeshControls
68  {
69      maxLocalCells 100000000;  // プロセッサあたりの最大メッシュ数
70      maxGlobalCells 200000000; // 全体の最大メッシュ数
71      minRefinementCells 10;    // 細分割される最小メッシュ数
```

```
72      nCellsBetweenLevels 3;      // 細分割レベルが異なる格子間のレイヤ数
73      maxLoadUnbalance 1;         // 並列時ロードバランシングを行なわない割合
74      // 本解析用の領域分割を後で行うので, ここではロードバランシングを行なわない
```

13.5 形状表面の細分割

refinementSurfaces で geometry で定義した形状表面に細かいメッシュを作成する．分割レベルを level で指定し，ベースメッシュの分割レベルを 0 として、1 上がるごとに 格子の辺を2分割する細分割を行う．分割レベルは最小値と最大値を指定するが，なるべく同一にするほうが望ましい．形状再現性や格子分解能を考慮して，必要に応じて分割レベルを調整する．また，作成されるパッチ面の型定義やグループ化には patchInfo を用いる．

Code 11 system/snappyHexMeshDict

```
98      refinementSurfaces
99      {
100         AhmedBody // geometryブロックで定義した形状の識別名
101         {
102             level (4 4); // 形状表面の細分割レベル（最小値 最大値）
103
104             patchInfo
105             {
106                 type wall; // パッチの境界条件
107                 inGroups (AhmedBodyGroup); // グループ名
```

```
108             }
109         }
110     }
```

13.6 領域の細分割

refinementRegions でgemetryで定義した形状の領域内(外も可能)のメッシュを細分化を行う．細分割する領域をmodeで設定するが，指定した形状の内部(inside)、外部(outside)、距離(distance)を選択できる．また，levelsに対して((距離 分割レベル))を指定する．ここで，距離はmodeがdistance以外の時は，特に意味を持たない．

Code 12 system/snappyHexMeshDict

```
129     refinementRegions
130     {
131         refinebox1
132         {
133             mode inside; // 領域内が分割される
134             levels ((1E15 1)); // 2番目の数字が細分割レベル
135             // 最初の数字はmodeがdistanceの場合のみ, 1番目が距離を表わす.
136         }
137 
138         refinebox2
```

13.7　解析領域の内部点指定

locationInMeshで解析領域の内部点を指定する．この点により定義した形状の内外判定が行われる．この点は，格子生成の途中の段階でも格子の界面や節点に一致してはいけない．

Code 13　system/snappyHexMeshDict

```
166     locationInMesh ( -1e-10 -1e-10 1e-10 );
```

13.8　境界適合

snapControlsで境界適合過程の制御パラメータを指定する．今回はデフォルトのままである．

13.9　境界層レイヤーの挿入

addLayersControlsのlayersで壁面に対して，境界層レイヤーを挿入することができる．境界層レイヤーを挿入するために必要な設定パラメータは「レイヤ厚さ」，「レイヤ数」，「拡大比」である．これらのパタメータを組み合わせて，解析に適したレイヤーを挿入する．

Code 14　system/snappyHexMeshDict

```
213 addLayersControls
214 {
215     // 以下で指定するレイヤのサイズが相対値(true)か絶対値(false)か？
216     // 相対値の場合, レイヤの外側の格子幅に対する比を指定する.
```

```
217     relativeSizes false;
218
219     // パッチ毎のレイヤー情報
220     layers
221     {
222       lowerWall // パッチ名
223       {
224           nSurfaceLayers 3; // レイヤ数
225       }
226
227       "AhmedBody_.*" // パッチ名
228       {
229           nSurfaceLayers 5; // レイヤ数
230       }
231     }
232
233     expansionRatio 1.2; //レイヤー拡大比
234
235     firstLayerThickness 0.0025; //最初のレイヤー厚さ [m]
236 //    finalLayerThickness 1; // 壁面最遠界層レイヤー厚さ [m]
```

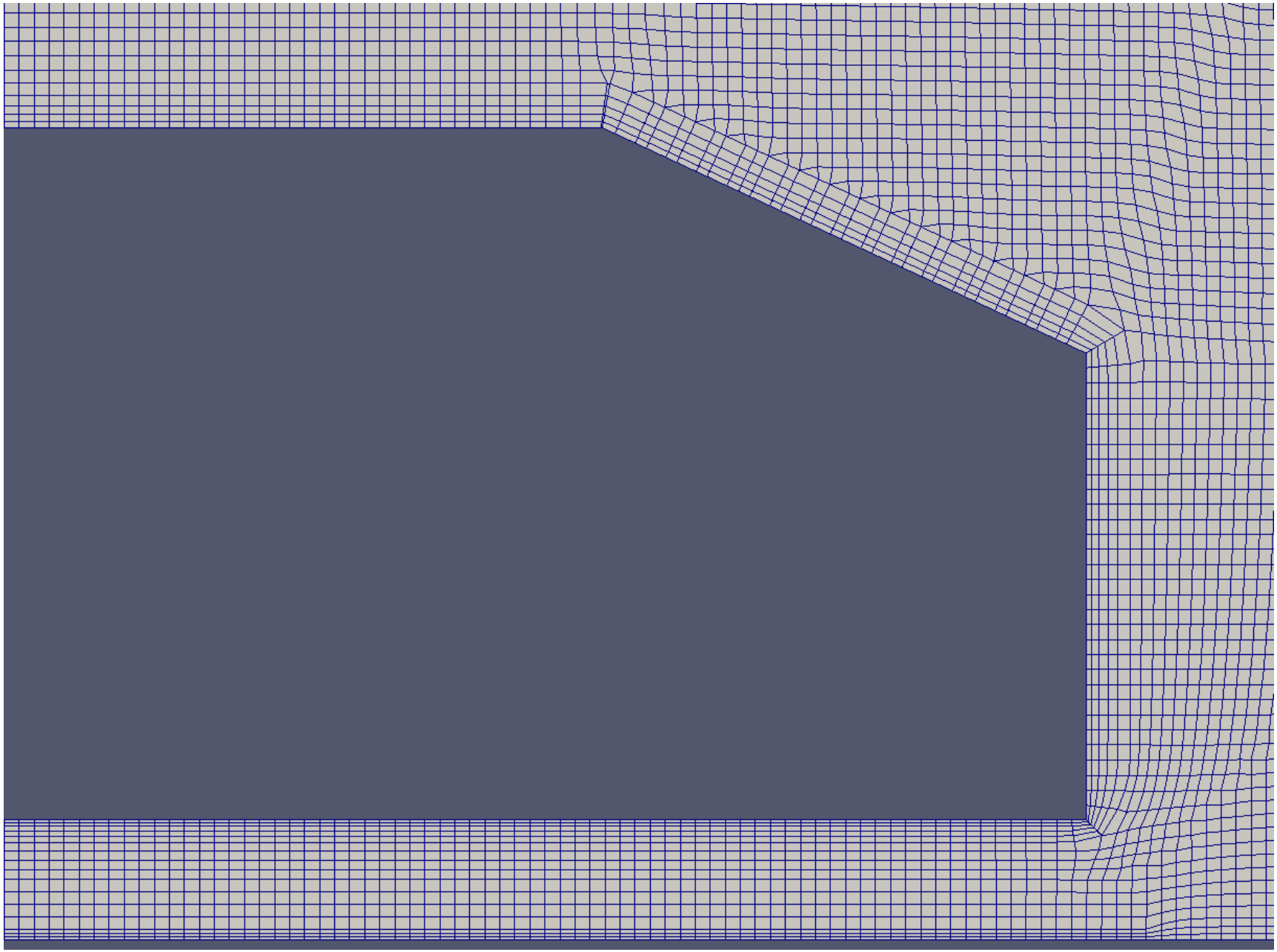


図8　境界層レイヤー(車体表面5層，地表面3層)

13.10　前処理のログの確認

ジョブが終了したら，必ずジョブのエラー出力ファイル*.sh.e[ジョブID]にエラーが出力されていない無い事を確認する.

```
more *.sh.e*
# または，ls -l *.sh.e* でファイルサイズが0である事を確認する
```

次に，moreコマンドを用いて，snappyHexMeshログの確認を行う.

```
more log.snappyHexMesh
```

最初にIntel MPIのデバッグ情報が出力されているので，プロセスのピニングが意図したものか確認する.

log.snappyHexMesh

```
[0] MPI startup(): Multi-threaded optimized library
[0] MPI startup(): shm data transfer mode   # [ランク] 使用されるファブリック
# 略
[63] MPI startup(): shm data transfer mode  # [ランク] 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)
[0] MPI startup(): Rank    Pid      Node name  Pin cpu # プロセスのピニング情報
[0] MPI startup(): 0       48661    c3847.ofp  2       # Rank0 → Pin2
# 略
[0] MPI startup(): 63      48733    c3847.ofp  65      # Rank63 → Pin65
[0] MPI startup(): I_MPI_DEBUG=5
```

```
[0] MPI startup(): I_MPI_FABRICS_LIST=tmi # 使用するファブリックリストの指定(注)
[0] MPI startup(): I_MPI_FALLBACK=0       # 指定したファブリックを必ず使用する(注)
[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 # 高速プロセス起動アルゴリズムon(注)
# 略
[0] MPI startup(): I_MPI_PIN_MAPPING=64:0 2,1 3,2 4,3 5,4 6,5 7,6 8,7 9,8 10,9 11,10
  12,11 13,12 14,13 15,14 16,15 17,16 18,17 19,18 20,19 21,20 22,21 23,22 24,23 25,24
  26,25 27,26 28,27 29,28 30,29 31,30 32,31 33,32 34,33 35,34 36,35 37,36 38,37 39,38
  40,39 41,40 42,41 43,42 44,43 45,44 46,45 47,46 48,47 49,48 50,49 51,50 52,51 53,52
  54,53 55,54 56,55 57,56 58,57 59,58 60,59 61,60 62,61 63,62 64,63 65 # ピニング設定(注)
# (注) これらはmpi_core_setting.shで設定される環境変数
```

また，OpenFOAMのアプリケーションのログにおけるnProcsやHostsの出力が意図したプロセッサ数やノード数となっていない場合には，ジョブファイルの指定を確認する．

log.snappyHexMesh

```
Build  : v1712 # ビルドバージョン
Arch   : "LSB;label=32;scalar=64" # "バイト順;ラベルbit数;実数変数bit数"
Exec   : snappyHexMesh -decomposeParDict system/decomposeParDict.mesh -overwrite -
  parallel # 実行コマンド
Date   : Jan 1 1970 # 開始日時
Time   : 00:00:00    # 開始時刻
Host   : "cxxxx.ofp" # ホスト名
PID    : xxxxx       # プロセスID
```

```
I/O    : uncollated  # 領域分割ファイルを1ファイルにまとめない
Case   : /work/0/gt00/txxxxx/lecture/Ahmed25/case-1 # ケースディレクトリ
nProcs : 64 # 計算で使用されているプロセス数
Hosts  :
(
    (cxxxx.ofp 64) # (計算ノードのホスト名 MPIプロセス数)
)
```

パッチに付加されたレイヤの統計情報は，snappyHexMeshのログの最後に出力される．

Code 15　log.snappyHexMesh

```
2282 patch            faces    layers   overall thickness
2283                                    [m]       [%]
2284 -----            -----    ------   ---       ---
2285 lowerWall        38953    2.99     0.00831   91.4
2286 AhmedBody_body   16595    4.85     0.0155    83.1
2287 AhmedBody_head   2288     4.65     0.0161    86.4
2288 AhmedBody_stilt  240      1.3      0.0027    14.5
2289 AhmedBody_slant  1062     4.58     0.0176    94.5
```

また，checkMeshのログにエラーが無い事を確認する．

checkMeshのログ確認

```
more log.checkMesh
```

Code 16　log.checkMesh

```
Checking geometry...
    Overall domain bounding box (-5 -0.935 -2.1684e-18) (10 0 1.4)
    Mesh has 3 geometric (non-empty/wedge) directions (1 1 1)
    Mesh has 3 solution (non-empty) directions (1 1 1)
    Boundary openness (5.94108e-18 -1.07564e-15 -6.09034e-16) OK.
    Max cell openness = 3.69328e-16 OK.
    Max aspect ratio = 40 OK.
    Minimum face area = 1.2894e-06. Maximum face area = 0.0103729.  Face area magnitudes
  OK.
    Min volume = 5.54246e-09. Max volume = 0.000975025.  Total volume = 19.5796.  Cell
  volumes OK.
    Mesh non-orthogonality Max: 64.5554 average: 4.78279
    Non-orthogonality check OK.
    Face pyramids OK.
    Max skewness = 2.71299 OK.
    Coupled point location match (average 0) OK.

Mesh OK.

End

Finalising parallel run
```

14　ファイルの転送

今回は，データをクライアント側に転送して可視化するので，ユーザマシンの端末で以下のコマンドを実行して，講習用ファイル一式をユーザマシンに転送する.

(ユーザマシン) 講習会用ファイルの転送

```
cd
mkdir lecture-OpenFOAM
rsync txxxxx@ofp.jcahpc.jp:lecture-OpenFOAM/ ~/lecture-OpenFOAM/ -auv --exclude=pro*
```

ここで，txxxxxは利用者番号である．なお，転送元と転送先どちらもに末尾に/(スラッシュ)を付ける．オプションの意味は，以下の通りである.

- -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・更新されたファイル・ディレクトリのみ転送.
- -v（-verbose）：転送情報を表示.
- -exclude=pro*：processorディレクトリを除外して転送.

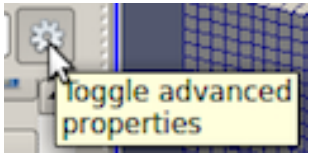
転送元の講習会用ディレクトリの実体は*/work/gt00/$USER/lecture-OpenFOAM*であるが，講習会用ディレクトリの参照を容易にするために，ホームディレクトリにシンボリックリンクを貼っているので，上記の指定で参照可能である.

15　格子の可視化

15.1　最終生成格子の可視化

ユーザマシンでParaViewを立ち上げて，*/lecture-OpenFOAM/AhmedBody/Ahmed25/case-1*にあるOpenFOAMデータ可視化用のダミーファイルpv.foamを読み込み，格子を可視化する．

表4　ParaViewによるベース格子の可視化

1. ‘File/Open‘メニューで，ケースのディレクトリのpv.foamを選択し，OKを押す．
2. `Mesh Regions`でcenter，lowerWall，AhmedBody_*を選択する(レイヤを可視化する場合にはcentarのみを選択する)．
3. ‘Apply‘ボタンを押す．
4. ‘Properties‘タブ内の‘Representation‘で‘Surface With Edges‘を選択する．
5. ‘Properties‘タブ内の‘Coloring‘で‘Solid Color‘を選択する．
6. 歯車状のボタンを押して，‘Advanced Properties‘を有効にする． 

7. ‘Camera Parallel Projection‘をチェックして，格子の分割幅がわかりやすい平行投影モードにする．
8. RenderView画面において，マウス等で視点を操作する．

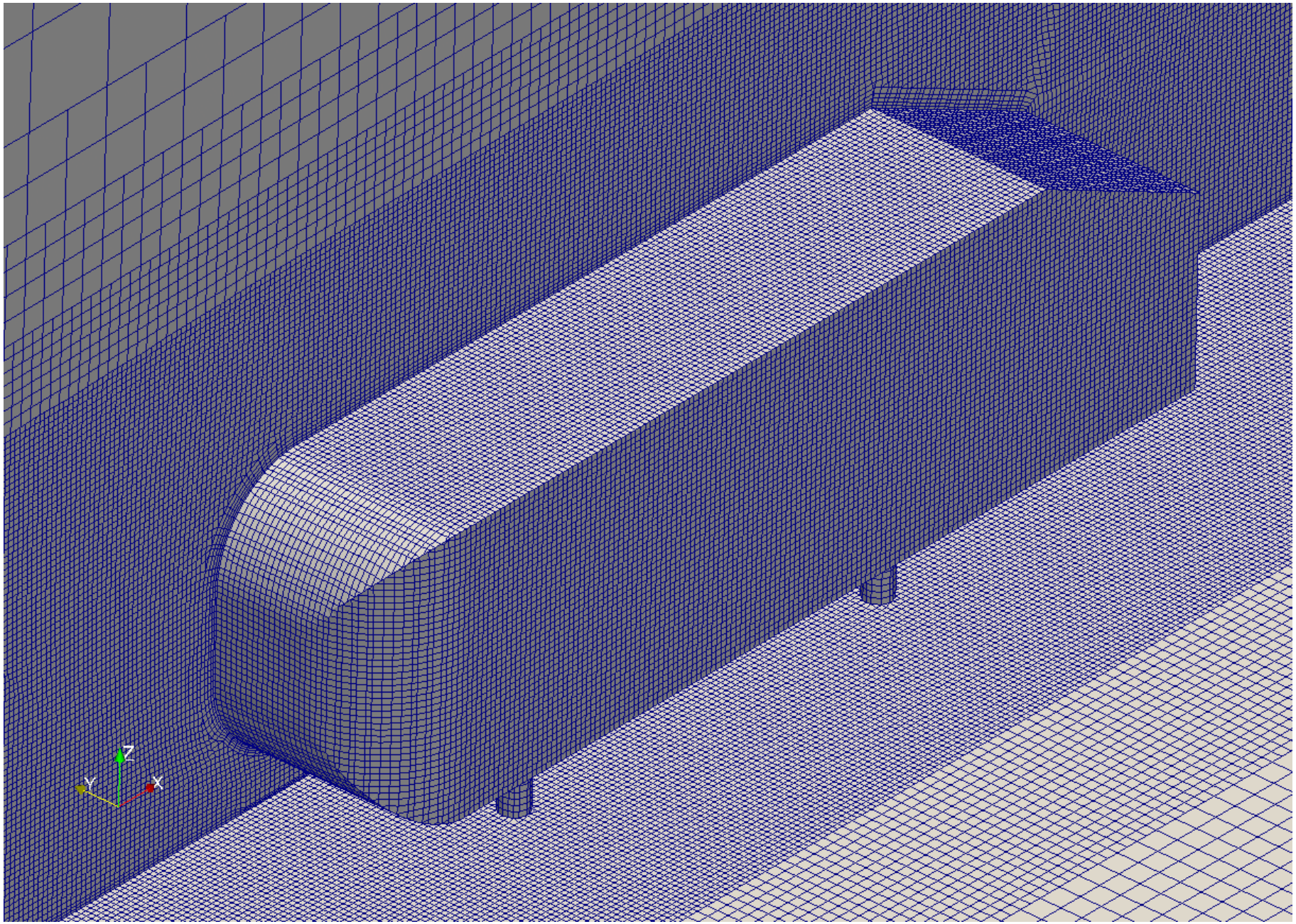


図9　生成格子

15.2　特徴辺の可視化

- surfaceFeatureExtract コマンドを実行すると，*constant/triSurface* に特徴辺ファイル .eMesh が作成される.
- 特徴辺ファイルは OpenFOAM の独自形式であるため ParaView 等で可視化できない.
- `writeObj` が `yes` の場合には，*constant/extendedFeatureEdgeMesh* に WaveFront OBJ 形式の特徴辺が *AhmedBody_edgeMesh.obj* として出力される.
- ParaView の ‘File/Open‘ メニューから *constant/extendedFeatureEdgeMesh/AhmedBody_edgeMesh.obj* を読み込む.
- Pipeline Browser で *pv.foam* フィルターを選択し，ボタンを押して，格子を非表示にする.

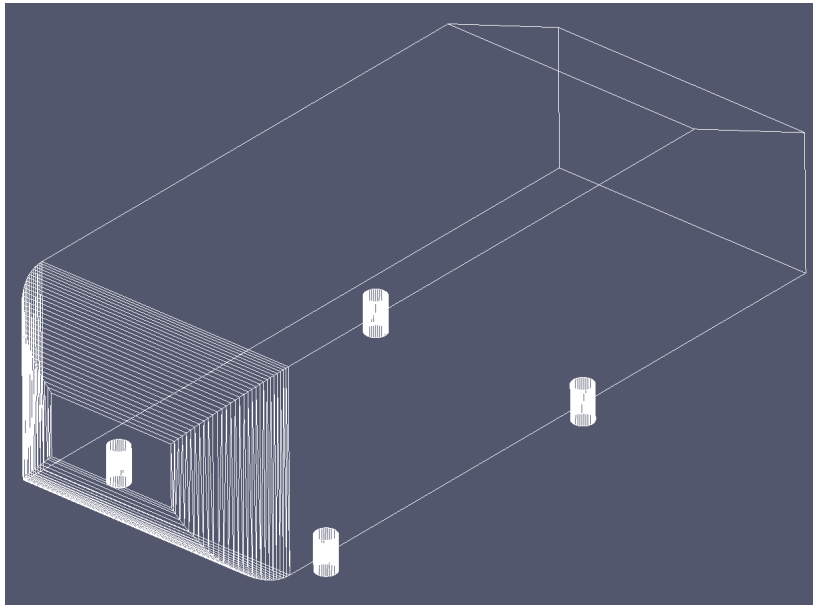


図 10　特徴辺

16　流体解析用の領域分割

Code 17　ofp1de.sh

```
1  #!/bin/bash
2  #PJM -S
3  #PJM -g gt00
4  #PJM -L elapse=0:15:00
5  #PJM -L rscgrp=tutorial-flat
6  #PJM -L node=1
7  source ofpshare.sh # 共通の設定
8  cp -a 0.orig 0 # 初期時刻ディレクトリを用意
9  $numactl decomposePar -force >& log.decomposePar # 流体解析用の領域分割
10 # -force :並列格子生成用のプロセッサディレクトリが残っているので消去する
```

Code 18　system/decomposeParDict

```
18 numberOfSubdomains 256; // 並列数
19 method       scotch; // 分割手法: scotch(領域分割最適化ライブラリを用いる)
```

今回は，1ノードあたり64プロセスで4ノード使用し，計256プロセスの並列解析を行う．

流体解析用の領域分割

```
pjsub ofp1de.sh
# 講習会終了後は pjsub -L rscgrp=lecture-flat ofp1de.sh
```

17　初期値作成

Code 19　ofp2solveInit.sh

```
1  #!/bin/bash
2  #PJM -S
3  #PJM -g gt00
4  #PJM -L elapse=0:15:00
5  #PJM -L rscgrp=tutorial-flat
6  #PJM -L node=4
7  #PJM --mpi proc=256
8  source ofpshare.sh # 共通の設定
9  $mpirun $numactl \
10 renumberMesh -overwrite -parallel >& log.renumberMesh # 格子順変更による行列バンド幅縮小
11 $mpirun $numactl \
12 potentialFoam -parallel >& log.potentialFoam # ポテンシャル流れを解いて初期値とする
```

- -L node=4：ノード数=4
- -mpi proc=256：総プロセス数=256=4×64(ノード毎のプロセス数ではない)
- renumberMesh：格子順変更→行列バンド幅縮小
  - 有限体積法では，圧力のPoisson方程式および運動量(速度)や乱流統計量の輸送方程式は，各種モデル化やsystem/fvSchemesで設定した離散化スキームを用いて離散化され，最終的には線型一次方程式に帰着される．
  - 線型一次方程式は，system/fvSolutionで設定した線型ソルバを用い，許容残差の範

囲で解かれる.
  - 係数行列のバンド幅が小さいほうが一般に計算効率が良いため，バンド幅を縮小させるために，格子の順番を変更する.
- potentialFoam：ポテンシャル流れを解いて初期値とする
  - 流体の粘性を無視しているため短時間で解けるポテンシャル流れを解き，流体解析の初期値として用いる.
  - ポテンシャル流れを初期値とすると，通常安定かつ早く収束できる.

17.1 初期値作成ジョブの投入

初期値作成

```
pjsub ofp2solveInit.sh
# 講習会終了後は pjsub -L rscgrp=lecture-flat ofp2solveInit.sh
pjstat # ジョブが終了したら, 以下でログを確認する
more log.renumberMesh
more log.potentialFoam
```

18　流体解析

Code 20　ofp3solve.sh

```
#!/bin/bash
#PJM -S
#PJM -g gt00
#PJM -L elapse=0:15:00
#PJM -L rscgrp=tutorial-flat
#PJM -L node=4
#PJM --mpi proc=256
source ofpshare.sh # 共通の設定
jobid=${PJM_SUBJOBID:-$PJM_JOBID} # ジョブID
$mpirun $numactl \
simpleFoam -parallel >& log.simpleFoam.$jobid # 並列流体解析
```

流体解析実行

```
pjsub ofp3solve.sh
# 講習会終了後は pjsub -L rscgrp=lecture-flat ofp3solve.sh
pjstat # ジョブが開始されたら, 以下でログをモニターする
tail -f log.simpleFoam.* # tailコマンドはCtrl+Cで終了できる
```

19　関数出力のモニター

線型ソルバーの残差や空力係数をモニターする場合はfoamMonitorコマンドを用いる.

Code 21　foamMonitor.sh

```
1 #!/bin/bash
2 foamMonitor -r 1 postProcessing/forceCoeffs/0/coefficient.dat &    # -r :更新秒
3 foamMonitor -r 1 -l postProcessing/residuals/0/residuals.dat &    # -l : 縦軸log
```

ログインノードでOpenFOAMのコマンドであるfoamMonitorを実行するので，以下のスクリプトによりログインノードにおいてOpenFOAMの環境設定を行う.

Code 22　ofpOpenFOAM.sh

```
1 #!/bin/bash
2 module purge # 標準で有効なmoduleに依存しないよう全moduleをunload
3 module load gcc/4.8.5 # Gccコンパイラのmoudulをload
4 source /work/gt00/share/OpenFOAM/OpenFOAM-v1712/etc/bashrc \
5 WM_COMPILER=Gcc4_8_5 WM_MPLIB=INTELMPI2017_3_196 # OpenFOAM環境設定
```

ログインノードにおけるOpenFOAMの環境設定

```
. ofpOpenFOAM.sh # .(ドット)と空白の後にofpOpenFOAM.sh
```

*ofp3solve.sh*のジョブが開始されたら，以下を実行する.

関数出力のモニター

```
./foamMonitor.sh
```

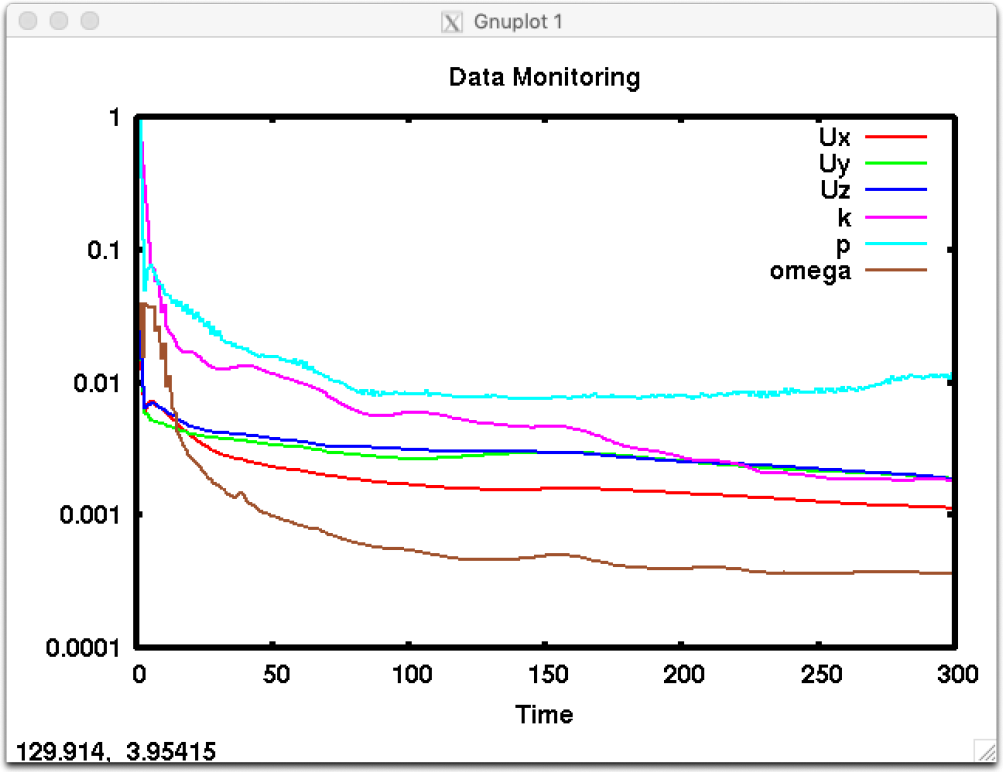


図11　線型ソルバー残差のモニター

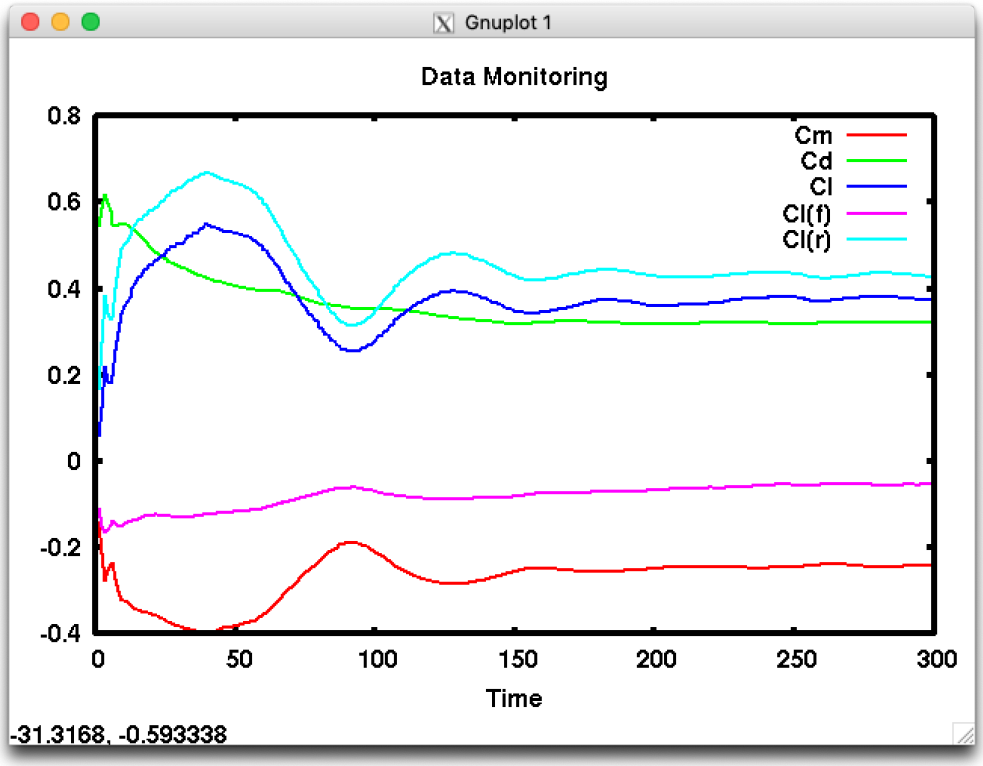


図12　空力係数のモニター

20　流体解析の設定

20.1　場の初期値と境界条件の設定

境界条件を図13に示す.

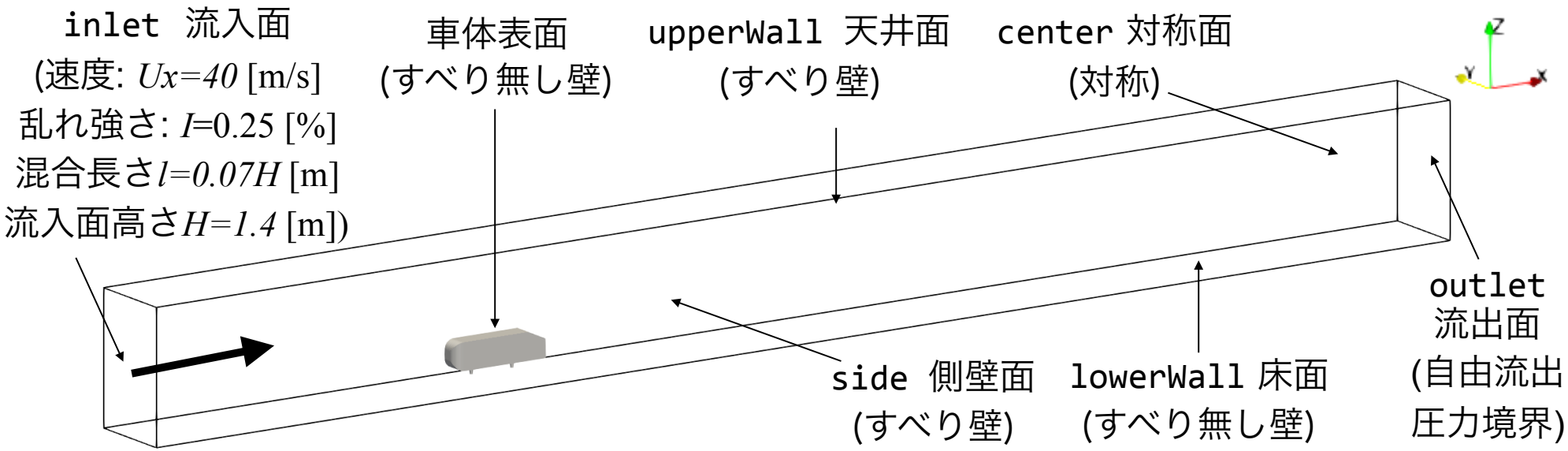


図13　境界条件

場の初期値と境界条件は0.origディレクトリに置かれた物理量ファイル(U, p, k, omega, nut...)で設定する. 本モデルの境界条件の型を表13に示す.

表5　境界条件の型

境界名	境界型	U	p	k	ε	ω	ν_t
inlet	patch	fixedValue	zero Gradient	turbulent Intensity Kinetic Energy Inlet	turbulent MixingLength Dissipation Rate Inlet	turbulent MixingLength Frequency Inlet	calculated
outlet	patch	pressure InletOutlet Velocity	fixedValue	inletOutlet	inletOutlet	inletOutlet	calculated
lowerWall, Ahmed.*	wall	noSlip	zero Gradient	kqRWall Function	epsilonWall Function	omegaWall Function	nutkWall Function
upperWall, side	wall	slip					
center	symmetry	symmetry					

Code 23　0.orig/U：速度の初期値と境界条件

```
17 dimensions      [0 1 -1 0 0 0 0]; // 単位: [m s^-1]
18 internalField   uniform (40 0 0); // 内部の初期値(主流方向に速度40[m/s])
19 boundaryField
20 {
21     inlet // 流入面
22     {
23       type              fixedValue; // 固定値
24       value             uniform (40 0 0); // 主流方向に速度40[m/s]
25     }
26     outlet // 流出面
27     {
28         type             pressureInletOutletVelocity;
29         // 圧力規定境界条件下の流入/流出用速度境界条件.
30         // 流出時はパッチ法線方向の速度勾配0の自由流出.
```

```
31          // 流入時は, 境界に隣接する内部格子の速度のパッチ法線方向の成分が与えられる.
32          value           $internalField;
33      }
34      "(side|upperWall)" // 側壁・天井
35      {
36          type            slip; // すべり壁
37      }
38      "(lowerWall|AhmedBody_.*)" // 床面・車体表面
39      {
40          type            noSlip; // すべり無し壁
41      }
42      center // 対称面
43      {
44          type            symmetry; // 対称
45      }
46  }
```

Code 24　0.orig/p : 圧力の初期値と境界条件

```
17  dimensions      [0 2 -2 0 0 0 0]; // 単位: [m^2 s^-2]
18  internalField   uniform 0; // 内部の初期値(差圧)
19  boundaryField
20  {
21      "(inlet|lowerWall|AhmedBody_.*)" // 床面・車体表面
22      {
23          type            zeroGradient;
24      }
25      outlet // 流出面
26      {
27          type            fixedValue; // 固定値
28          value           $internalField; // 内部の初期値
29      }
30      "(side|upperWall)" // 側壁・天井
```

```
    {
        type            slip; // すべり壁
    }
    center // 対称面
    {
        type            symmetry; // 対称
    }
}
```

Code 25　0.orig/nut：動粘性係数の初期値と境界条件

```
dimensions      [0 2 -1 0 0 0 0]; // 単位: [m^2 s^-1]
internalField   uniform 0; // 内部の初期値
boundaryField
{
    "(inlet|outlet)" // 流入面・流出面
    {
        type            calculated; // 自動的に算出
        value           uniform 0;
    }
    "(side|upperWall)" // 側壁・天井
    {
        type            slip; // すべり壁
    }
    "(lowerWall|AhmedBody_.*)" // 床面・車体表面
    {
        type            nutkWallFunction; // 壁関数(kより算出)
//        type            nutUWallFunction; // 壁関数(Uより算出)
//        type            nutUSpaldingWallFunction; // 壁関数(Uより算出・Spalding型)
        value           uniform 0;
    }
    center // 対称面
    {
```

```
        type            symmetry; // 対称
    }
}
```

Code 26　0.orig/k：乱流エネルギーの初期値と境界条件

```
dimensions      [0 2 -2 0 0 0 0]; // 単位: [m^2 s^-2]
internalField   uniform 0.1; // 内部の初期値
boundaryField
{
    inlet // 流入面
    {
        type            turbulentIntensityKineticEnergyInlet; // 乱れ強さ指定
        intensity       0.0025;          // 乱れ強さI=0.25%
        value           $internalField; // 内部の初期値(ここでの値は影響しない)
    }
    outlet // 流出面
    {
        type            inletOutlet;     // 自由流出
        inletValue      $internalField; // 流入時の値は内部の初期値
        value           $internalField;
    }
    "(side|upperWall)" // 側壁・天井
    {
        type            slip;
    }
    "(lowerWall|AhmedBody_.*)" // 床面・車体
    {
        type            kqRWallFunction; // 壁関数(実際には法線方向勾配0)
        value           $internalField;
    }
    center // 対称面
    {
```

```
44          type            symmetry; // 対称
45      }
46  }
47
48
49  // ************************************************************************* //
```

Code 27　0.orig/omega：比消散率の初期値と境界条件

```
17  dimensions      [0 0 -1 0 0 0 0]; // 単位: [s^-1]
18  internalField   uniform 11.1; // 内部の初期値(omega=epsilon/k/Cmu=0.1/0.1/0.09)
19  boundaryField
20  {
21      inlet
22      {
23          type            turbulentMixingLengthFrequencyInlet; // 混合長さスケール指定
24          mixingLength    0.098; // 混合長さスケールl=0.07*H. H=1.4m(流入面高さ).
25          value           $internalField; // 内部の初期値(ここでの値は影響しない)
26      }
27      outlet
28      {
29          type            inletOutlet;     // 自由流出
30          inletValue      $internalField; // 流入時の値は内部の初期値
31          value           $internalField;
32      }
33      "(side|upperWall)" // 側壁面・天井面
34      {
35          type            slip; // すべり壁
36      }
37      "(lowerWall|AhmedBody_.*)" // 床面・車体表面
38      {
39          type            omegaWallFunction; // omega用の壁関数
40          value           $internalField;
```

```
    }
    center // 対称面
    {
        type            symmetry; // 対称
    }
}
```

Code 28　0.orig/epsilon：乱流エネルギー消散率の初期値と境界条件

```
dimensions      [0 2 -3 0 0 0 0]; // 単位: [m^2 s^-3]
internalField   uniform 0.1; // 内部の初期値
boundaryField
{
    inlet // 流入面
    {
        type            turbulentMixingLengthDissipationRateInlet; // 混合長さスケール指定
        mixingLength    0.098; // 混合長さスケールl=0.07*H. H=1.4m(流入面高さ).
        value           $internalField; // 内部の初期値(ここでの値は影響しない)
    }
    outlet // 流出面
    {
        type            inletOutlet;    // 自由流出
        inletValue      $internalField; // 流入時の値は内部の初期値
        value           $internalField;
    }
    "(side|upperWall)" // 側壁面・天井面
    {
        type            slip;
    }
    "(lowerWall|AhmedBody_.*)" // 床面・車体表面
    {
        type            epsilonWallFunction; // epsilon用の壁関数
        value           $internalField;
```

```
41      }
42      center // 対称面
43      {
44          type            symmetry; // 対称
45      }
46  }
```

20.2　流体物性値の設定

粘性係数などの流体物性値はconstant/transportPropertiesで設定する．空気の動粘性係数は $\nu = 1.5 \times 10^{-5}$ m ^{2}/s を設定する．

Code 29　constant/transportProperties

```
19  nu [0 2 -1 0 0 0 0] 1.5e-05; // 動粘性係数 [m^2/s]
```

20.3　乱流モデルの設定

乱流モデルはconstant/turbulencePropertiesで設定する．RANSの k - ω SSTモデルの乱流モデルを設定する．

Code 30　system/turbulenceProperties

```
17  simulationType RAS;
18
19  RAS
```

```
20 {
21     RASModel            kOmegaSST;     // k-ω SSTモデル
22 //  RASModel            kEpsilon;      // 標準k-εモデル
23 //  RASModel            RNGkEpsilon;   // RNG k-εモデル
24 //  RASModel            realizableKE;  // Realizable k-εモデル
25 //  RASModel            kOmega;        // k-ωモデル
26
27   turbulence          on; // 乱流の計算：on=有効、off=無効
28
29   printCoeffs         on; // 乱流モデル係数の表示：on=有効、off=無効
30 }
```

RASModel で乱流モデルの種類を設定する．また，printCoeffsがonの場合は，計算初期に乱流モデル係数が表示される．バージョンによって，デフォルトの係数が変更される場合もあるので，計算条件保存のためonにしておくほうが良い．

Tips

例えば，OpenFOAM-5.0およびv1712から，標準$k-\varepsilon$モデル(kEpsion)のモデル係数C3のデフォルト値が-0.33から0に変更されている．

20.4 離散化スキームの設定

離散化スキームはsystem/fvSchemesで設定する．移流項のスキームdivSchemes(phi,変数)は，速度についてTVD系スキームlimitedLinearのベクトル版limitedLinearV を使用する．limitedLinearスキームのパラメータは0から1の範囲を取るが，最も安定となる1を指定した．乱流統計量については，最も安定であるupwind(一次風上)を用いた．もちろん，他のスキームでの解析可能である．

Code 31　system/fvSchemes

```
27 divSchemes
28 {
29     default           none;
30
31     // 移流項の離散化スキーム
32     div(phi,U)        bounded Gauss limitedLinearV 1; // TVD系スキーム
33 //  div(phi,U)        bounded Gauss upwind;
34 //  div(phi,k)        bounded Gauss limitedLinear 1;
35     div(phi,k)        bounded Gauss upwind;           // 風上(最安定・1次精度・数値拡散過大)
36 //  div(phi,omega)    bounded Gauss limitedLinear 1;
37     div(phi,omega)    bounded Gauss upwind;           // 風上(最安定・1次精度・数値拡散過大)
```

20.5　解法の設定

Code 32　system/fvSolution

```
17 solvers
```

```
18 {
19     // 圧力のPoisson方程式の線型ソルバの設定
20     p
21     {
22         solver          PCG;  // 線型ソルバ:PCG(前処理付き共役勾配法)
23         preconditioner  DIC;  // 前処理:DIC(diagonal incomplete-Cholesky)
24         tolerance       1e-7; // 最終残差の許容誤差
25         relTol          0.01; // 相対残差(最終残差/初期残差)の許容誤差
26     }
27
28     // potentialfoam用の線型ソルバの設定
29     Phi
30     {
31         $p; // p(圧力)の定義を参照する(pと同じ設定を用いる)
32     }
33
34     // U, k, omega, epsilonの輸送方程式の線型ソルバ
35     "(U|k|omega|epsilon)"
36     {
37         solver          smoothSolver; // 線型ソルバ:定常反復法
38         smoother        GaussSeidel; // スムーサ:ガウス-ザイデル法
39         tolerance       1e-8; // 最終残差の許容誤差
40         relTol          0.1; // 相対残差(最終残差/初期残差)の許容誤差
41         nSweeps         1; // 反復内のスイープ数
```

```
42      }
43  }
44
45  SIMPLE
46  {
47      // 非直交補正反復回数
48      // 格子の非直交性が悪く初期に発散する場合には非直交補正の反復を1以上にする
49      nNonOrthogonalCorrectors 1;
50
51      // SIMPLEC(SIMPLE consistent)型か？
52      consistent yes;
53  }
54
55  // SIMPLE, SIMPLEC法の緩和係数
56  // 発散する場合には小さくする
57  relaxationFactors
58  {
59      equations
60      {
61          U                       0.9;
62          "(k|omega|epsilon)"     0.9;
63      }
64  }
65
```

```
66 // potentialFoamの反復回数
67 potentialFlow
68 {
69     nNonOrthogonalCorrectors 15;
70 }
```

20.6 実行制御の設定

Code 33 system/controlDict

```
17 application         simpleFoam;// アプリケーション名
18 startFrom           startTime; // 解析開始の設定法
19 startTime           0;         // 解析の開始時間(定常計算では, 時間=反復番号)
20 stopAt              endTime;   // 解析終了の設定法
21 endTime             300;       // 解析の終了時間
22 deltaT              1;         // 解析時間刻み(定常計算では通常1)
23 writeControl        timeStep;  // 解析結果書き出しの決定法
24 writeInterval       300;       // 書き出す間隔
25 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
26 writeFormat         binary;    // データファイルのフォーマット(binary, ascii)
27 writePrecision      6;         // データファイルの有効桁(上記がasciiの場合)
28 writeCompression    off;       // データファイルの圧縮(off, on)
29 timeFormat          general;   // 時刻ディレクトリのフォーマット
30 timePrecision       6;         // 時刻ディレクトリのフォーマット有効桁
```

20.7　計算履歴の出力

抗力・揚力係数(C_d , C_l)や，線型ソルバ残差の履歴を出力するには，OpenFOAMの関数機能functionsを利用するため，system/controlDictのfunctionsに，#includeを用いて関数を定義したファイルを設定する．関数定義のファイルはsystemに置く．

Code 34　system/controlDict

```
32 functions
33 {
34     #include "forceCoeffs"
35     #include "forces"
36     #include "residuals"
37 }
```

空力係数の関数を定義したsystem/forceCoeffsでは，以下のように，車体のパッチ名，自由流速度(流入速度)，代表長さ，代表面積(見付面積)などを定義する．空力の関数を定義したsystem/forcesもほぼ同様である．

Code 35　system/forceCoeffs

```
 9 forceCoeffs
10 {
11     type          forceCoeffs;        // 関数型
12     libs          ("libforces.so");   // 関数のライブラリ
13     writeControl timeStep;            // 出力間隔手法(時刻ステップベース)
14     timeInterval 1;                   // 算出間隔(毎回)
```

```
15      log           yes;                  // ログ出力の有無
16      patches       ( "AhmedBody_.*");    // 車体のパッチ名
17      rho           rhoInf;               // 密度の定義
18      rhoInf        1.225;                // 流体密度(非圧縮性解析の場合, ダミー)
19      liftDir       (0 0 1);              // 揚力方向
20      dragDir       (1 0 0);              // 抗力方向
21      pitchAxis     (0 1 0);              // ピッチング軸方向
22      CofR          (-0.5269 0 0.186);    // モーメント中心
23      magUInf       40;                   // 自由流速度(流入速度)
24      lRef          0.47;                 // モーメント算出用代表長さ
25      Aref          0.0575;               // 代表面積(見付面積. 0.389*0.288/2+0.03*0.05)
26  }
```

線型ソルバ残差の出力の関数定義は以下のように，残差を出力する場のリストなどを定義する.

Code 36　system/residuals

```
 9  residuals
10  {
11    type            residuals;          // 関数型
12    libs            ("libutilityFunctionObjects.so"); // 関数のライブラリ
13    writeControl    timeStep;           // 出力間隔手法(時刻ステップベース)
14    writeInterval   1;                  // 算出間隔(毎回)
15    fields (p U k omega);               // 残差を出力する場のリスト
16  }
```

20.8　解析結果の再構築

Code 37　ofp4re.sh

```
7 source ofpshare.sh # 共通の設定
8 $numactl reconstructPar -latestTime >& log.reconstructPar # 最終時刻の解析結果の再構築
```

解析結果の再構築

```
pjsub ofp4re.sh
# 講習会終了後は pjsub -L rscgrp=lecture-flat ofp4re.sh
```

21　空力係数の実測値との比較

Ercoftacベンチマークでは車両の空気抗力係数 C_d と車両周りの対称面における主流方向速度分布について検証されているので，実験値との比較検証を行う．
空力係数の関数の算出結果は，流体解析実行時に *postProcessing/forceCoeffs/0/coefficient.dat* 出力されているので，以下のようにして値を確認し，風洞実験値と比較する．結果を表6に示す．

空力係数の関数出力の表示

```
more postProcessing/forceCoeffs/0/coefficient.dat
```

Code 38　空力係数の関数出力

```
10 # Time          Cm              Cd              Cl              Cl(f)           Cl(r)
11 300             -2.399826e-01   3.218229e-01    3.747412e-01    -5.261198e-02   4.273532e-01
```

表6　空力係数の比較

	実験	CFD(template)
抗力係数C_d	0.299	0.321
揚力係数C_l	0.345	0.376

22　後処理

Code 39　ofp5post.sh

```
1  #!/bin/bash
2  #PJM -S
3  #PJM -g gt00
4  #PJM -L elapse=0:15:00
5  #PJM -L rscgrp=tutorial-flat
6  #PJM -L node=4
7  #PJM --mpi proc=256
8  source ofpshare.sh # 共通の設定
9  $mpirun $numactl postProcess -parallel -latestTime -func sample >& log.sample # サンプリング
10 $mpirun $numactl simpleFoam -parallel -latestTime -postProcess -func yPlus >& log.yPlus # y+算出
```

解析結果をサンプリングしたり，無次元化壁座標 y^+ を算出するなどの後処理についても並列実行が可能である．

後処理ジョブの投入

```
pjsub ofp5post.sh
# 講習会終了後は pjsub -L rscgrp=lecture-flat ofp5post.sh
```

サンプリングの設定はsystem/sampleで行う．

Code 40　system/sample

```
type sets;                  // 関数型
libs ("libsampling.so");    // 関数のライブラリ

// 補間方法: 格子中心と節点, 界面における値を用いて補間
interpolationScheme cellPointFace;

// 集合サンプリング出力形式 raw:ASCII生データ形式
setFormat raw;

 // サンプルするフィールドのリスト
fields
(
    U
);

// 集合サンプリングの定義
sets
(
    x-243 // 鉛直プロファイルラインのサンプリング名
    {
        // サンプルタイプ: 線分と格子界面との交点の中点および界面
        type            midPointAndFace;
        // サンプリング座標値の出力軸
```

また，各パッチにおける無次元化壁座標 y^+ の統計値は，*log.yPlus*で確認できる.

Code 41　run/log.yPlus

```
yPlus yPlus write:
    writing field yPlus
    patch side y+ : min = 0.155623, max = 518.978, average = 55.0887
    patch lowerWall y+ : min = 24.3891, max = 434.433, average = 109.457
    patch upperWall y+ : min = 2.74571, max = 175.691, average = 95.3974
    patch AhmedBody_body y+ : min = 27.6063, max = 460.109, average = 116.808
    patch AhmedBody_head y+ : min = 13.2498, max = 143.364, average = 59.8158
    patch AhmedBody_stilt y+ : min = 33.0915, max = 635.635, average = 127.666
    patch AhmedBody_slant y+ : min = 66.5122, max = 366.126, average = 138.547
```

23　速度分布のプロットと実験値との比較

- 実験値と解析結果のデータをプロットしてグラフ化するが，ここではgnuplotを用いる.
- 今回は車体後部上面の各点についての速度分布を確認するために車体後部上面の形状もプロットした．他にも，線型ソルバーの残差，空力係数，空力のプロット用gnuplot入力ファイルがplotディレクリに収めてあるので，これらをgnuplotでプロットする(もちろん手動でgnuplotを実行しても良い).

Code 42　ofp6plot.sh

```
7  for file in plot/*.gp
8  do
9      gnuplot $file
10 done
```

上記を実行し，画像ビューワevinceで表示するには以下のようにする.

プロット実行・表示

```
./ofp6plot.sh
evince *.pdf
```

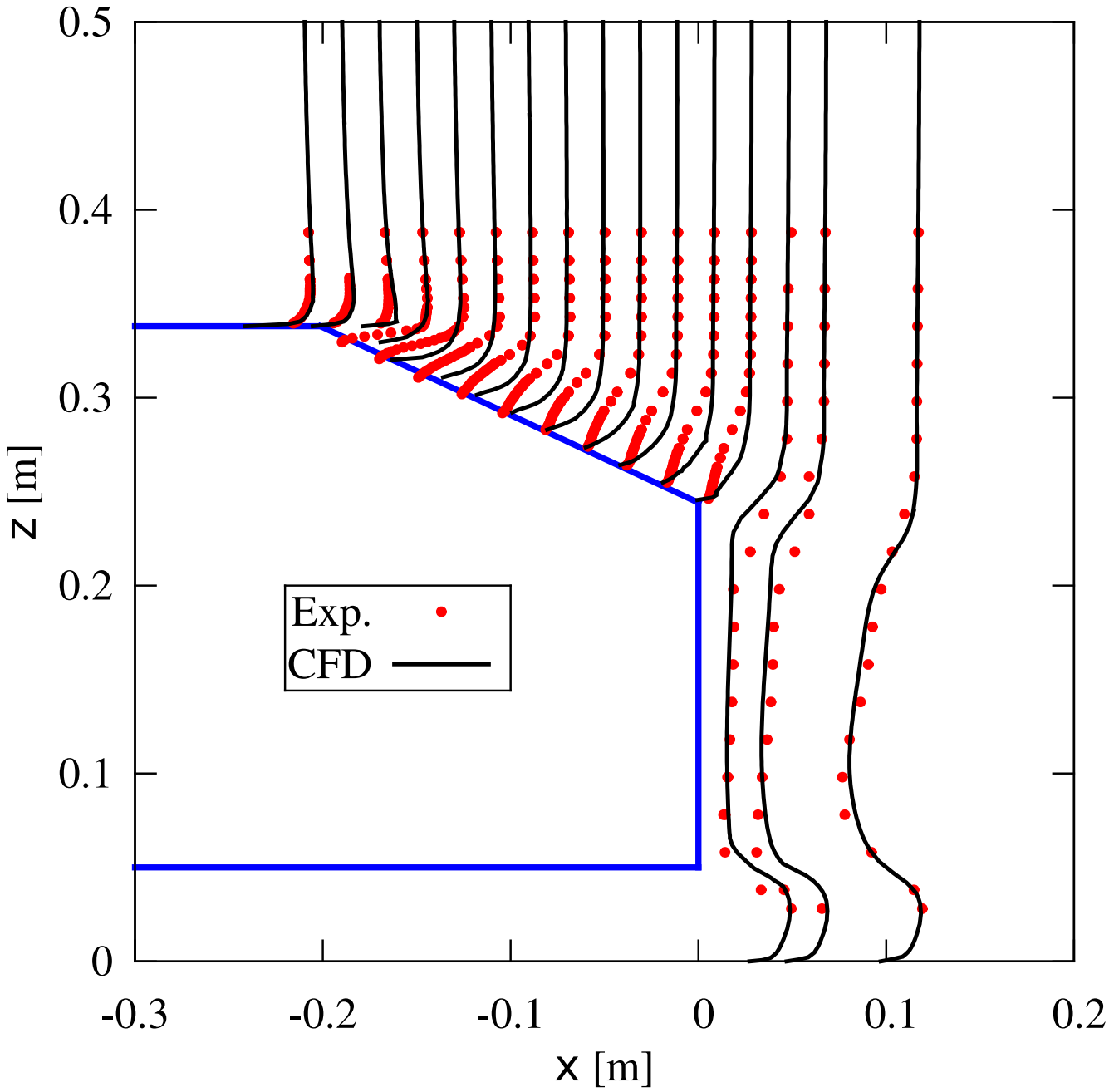


図14　速度の鉛直プロファイルの実験値との比較

24　ParaViewによる計算結果の確認

解析結果をユーザマシンに転送し，ParaViewで流線を描き，後引き渦の存在を確認する.

(ユーザマシン) 計算結果のファイル転送

```
rsync txxxxx@ofp.jcahpc.jp:lecture-OpenFOAM/ ~/lecture-OpenFOAM/ -auv --exclude=pro*
#または, カーソル↑で上記を呼び出してリターン(エンター)キーを押す
```

1. Refreshボタンを押して，転送した解析結果を読み込む.
2. Mesh RegionsでinternalMesh, center，lowerWallおよびAhmedBody_*を選択する.
3. Cell ArraysでUのみを選択する.
4. Last Frameボタンを押して，Applyボタンを押す.
5. Pipeline Browserでpv.foamを選択し，Extract Blockフィルターを選択後, internalMesh以外の全パッチを選択し，Applyボタンを押す.
6. Coloringを○Uにする.
7. Pipeline Browserでpv.foamを選択し，Extract Blockフィルターを選択後, internalMeshを選択し，Applyボタンを押す.
8. さらにStream Tracerフィルターを選択し，Seed typeにHigh resolution line source を選択し，Point1=(-5,-0.935,0.2)，Point2=(-5,0,0.2)，Resolution=100を入力する.
9. Applyボタンを押して流線を表示する.
10. Coloringを○Uにする.

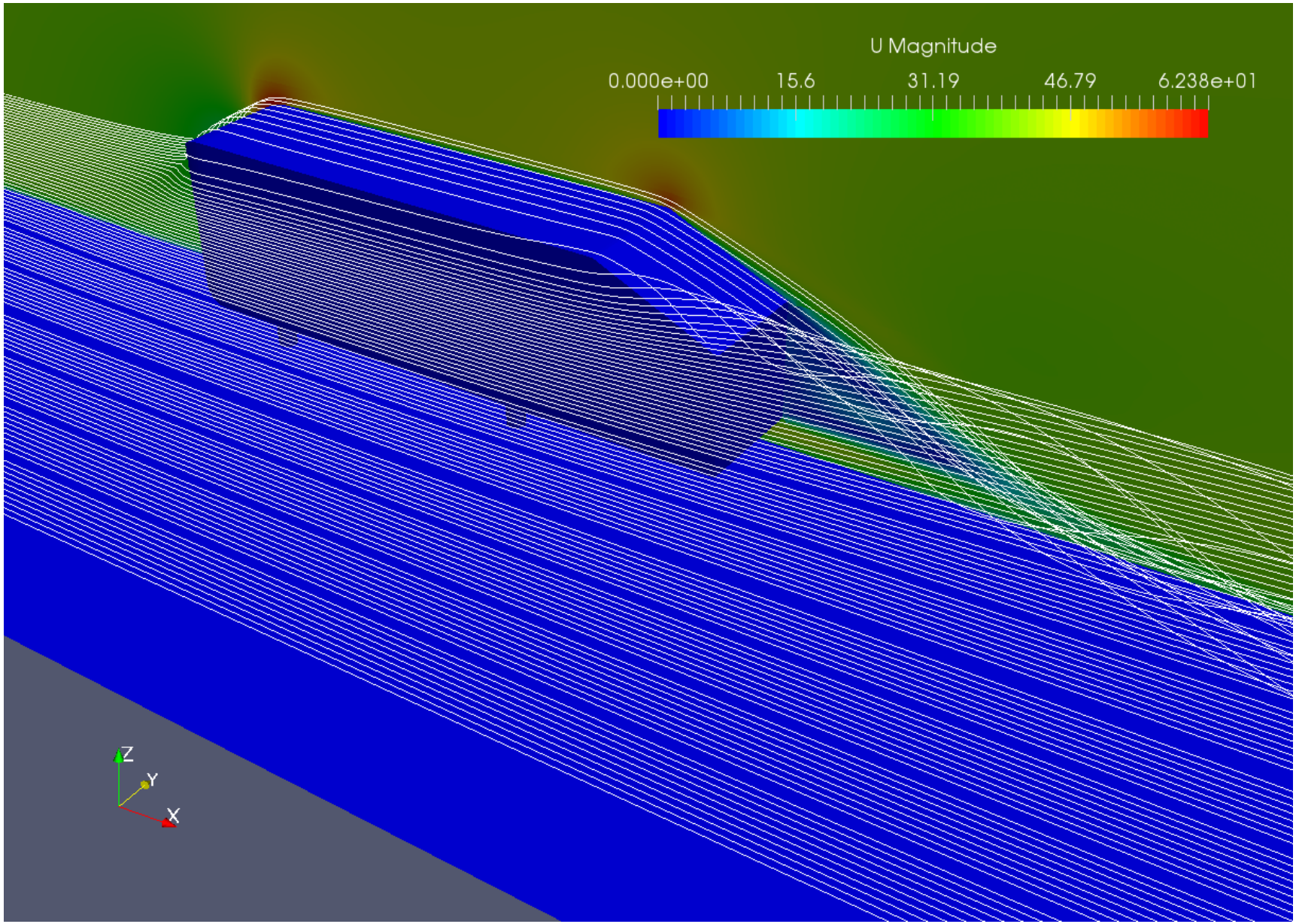


図15　流線と対称面速度分布(Color Mapや Color legendの位置等は適宜変更する)

25　流体解析の性能プロファイル実行

25.1　流体解析の性能プロファイル

- 性能プロファイラにより，計算負荷が高い部分(hotspots)など計算効率改善のためのデータを，ソースの変更や再コンパイル無しに取得可能である.
- ソースレベルの詳細なプロファイルには，デバッグオプションを付けた再コンパイルが必要である.
- VTune Amplifierを用いて，流体解析の性能プロファイル実行を行うスクリプトを示す.

Code 43　ofpsolveVtune.sh

```
 9 module load vtune # VTune Amplifier XEのmoduleをload
10 jobid=${PJM_SUBJOBID:-$PJM_JOBID} # ジョブID
11 # VTune Amplifier XEのコマンドamplxe-clをgtoolオプションで指定
12 # -collect hotspots : hotspot(計算負荷が高い部分)の解析を行う
13 # -r ディレトリ名:MPIランク範囲 : MPIランク範囲に関する解析結果をディレトリ名に保存
14 $mpirun -gtool "amplxe-cl␣-collect␣hotspots␣-r␣vtune.$jobid:0" \
15 $numactl simpleFoam -parallel >& log.simpleFoam.$jobid # 並列流体解析
```

上記のジョブを投入することで，流体解析の性能サンプリングを行うことができる.

流体解析の性能サンプリング

```
cd ~/lecture-OpenFOAM/AhmedBody/Ahmed25
cp -a case-1 case-profile
cd case-profile
```

```
vi system/controlDict # endTimeを101と短くする(計算負荷の概要把握には十分)
pjsub ofpsolveVtune.sh
# 講習会終了後は pjsub -L rscgrp=lecture-flat ofpsolveVtune.sh
```

25.2 プロファイラ結果の表示

Code 44 ofpvtune.sh

```
2  module load vtune # VTune Amplifier XEのmoduleをload
3  for dir in vtune.*
4  do
5      [ -d $dir ] || continue
6      # amplxe-cl : VTune Amplifier XEのコマンドライン版コマンド
7      # -r データディレトリ名
8      # -show-as=percent : 関数の実行時間を秒ではなく, 割合で示す
9      # -format=csv : テキスト形式ではなく, CSV形式で出力する(区切記号はコンマではなくタブ)
10     amplxe-cl -R hotspots -r $dir -show-as=percent -format=csv > $dir.csv
11 done
```

- ここでは，GUIツールを用いず，CLIツールによりプロファイル結果をCSV形式に変換して表示する.
- ジョブの実行終了後，上記のスクリプトを実行して，プロファイルデータをテキスト変換して後，表示する.

```
./ofpvtune.sh
more *.csv
```

以下がCPU時間の割合を示したCSV形式の結果ファイルの一部である.

Code 45　vtune.*.csv

```
Function          CPU Time
Foam::DICPreconditioner::precondition    11.565199 # 圧力PCG線型ソルバのDIC前処理
PMPI_Allreduce  5.774741                           # MPIのAllreduce(線型ソルバのベクトル内積)
Foam::lduMatrix::Amul    6.064315                  # 線型ソルバーでの疎行列ベクトル積SpMV
Foam::PCG::solve         4.559271                  # 圧力PCG線型ソルバ
```

- 計算時間の割合は，Foam::DICPreconditioner::precondition(PCG線型ソルバのDIC前処理)が1位，PMPI_Allreduce(線形ソルバでのベクトルの内積で必要となるMPIプロセスの全体通信)が2位，Foam::lduMatrix::Amul(線型ソルバーでの疎行列ベクトル積SpMV)が3位である.
- MPIプロセス数が増加するに伴ない，線型ソルバにおけるMPIのAllreduceが計算時間の多くを占めるようになるので，超大規模ノードの解析ではMPI通信時間を削減・隠蔽する必要がある[7].

26　格子生成・空力解析演習

26.1　格子生成・空力解析演習手順

- *template*ディレクトリをコピーし，設定の変更を行なう
- 流体解析を実行し，空力係数，速度分布の実験値との比較を行う．
- *./Allrun.batch*を実行すると，前処理から結果プロットまでの全ジョブを，依存関係があるサブジョブとするステップジョブとして投入する．

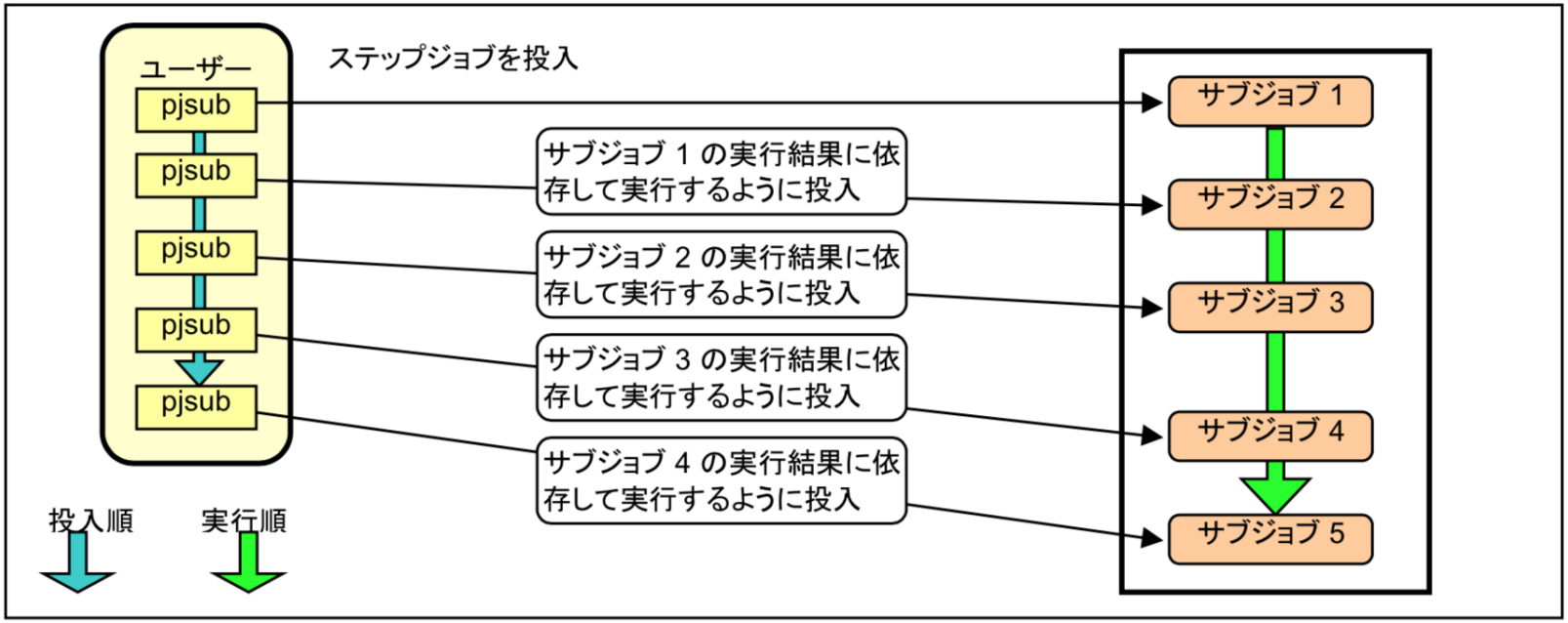


図16　ステップジョブイメージ(図引用元:[2])

26.2　格子生成・空力解析演習設定変更例

- 生成格子を変更(設定: *system/snappyHexMeshDict*)
  - 床面や車体表面のレイヤ数・レイヤ最小厚み変更
  - 車体表面の分割レベルの変更
- 流体解析の初期値の境界条件を変更
  - 乱流動粘性係数の nut の壁関数変更(設定: *0.orig/nut*)
- 流体解析の計算条件を変更
  - 乱流モデルの変更(設定:*constant/turbulenceProperties*)
  - 移流項離散化スキームの変更(設定:*system/fvSchemes*)

26.3　格子生成・空力解析演習実行例

```
cd ~/lecture-OpenFOAM/AhmedBody/Ahmed25
cp -a template case-2 # コピー先のディレクトリ名は任意
cd case-2             # この後, 適宜設定変更
./Allrun.batch        # 全ジョブを実行
pjstat -E             # -E: ステップジョブ内のサブジョブを展開して表示
cd ..
cp -a template case-3 # 以下同様
```

27　並列計算ベンチマークテスト

- 領域分割数や領域分割方法，メモリモード，圧力の線型ソルバ等を変更して流体解析を行い，解析時間を比較する.
  - 領域分割数 (*system/decomposeParDict* の `numberOfSubdomains`)
  - 領域分割法: scotch, multiLevel, simple, hierarchical など (*system/decomposeParDict* の `method`)()
  - ノード数 (`pjsub -L node=`ノード数 `-mpi proc=`領域分割数)
  - 圧力の線型ソルバ，前処理，スムーサ (*system/fvSolution*)
- 予備検討で行う並列計算のベンチマークテストでは，時間 (反復) ステップ数を本計算より短くして検討する事が多い.
- 解析の初期化にかかる時間は，ステップ数が多い本計算では相対的に小さくなる.
- ここでは，初期ステップ完了から最終ステップ完了までの解析時間のステップあたりの平均値を比較する.
- P ノード並列時の初期と最終のステップの解析時間 (ExecutionTime) の差を $t(P)$ として，以下を求める.
  - スピードアップ率：$S_P = t(1)/t(P)$
  - 並列化効率 [%]：$E_P = S_P/P \times 100 = (t(1)/t(P))/P \times 100$

並列計算ベンチマークテスト実行例

```
cd ~/lecture-OpenFOAM/AhmedBody/Ahmed25
cp -a template case-mesh          # 格子のみ作成されたケースを作成
cp -a case-1/constant/polyMesh case-mesh/constant # case-1の格子データをコピー
vi case-mesh/system/controlDict # endTimeを101と小さい値にする
cp -a case-mesh case-16-1024-scotch # コピー先のディレクトリ名は任意
cd case-16-1024-scotch            # この後, 適宜設定変更
vi system/decomposeParDict        # 領域分割の変更(例えば, numberOfSubdomainsを1024に変更)
vi ofp2solveInit.sh               # ノード数やMPIプロセス数を変更(例えば, node=16, proc=1024)
vi ofp3solve.sh                   # 同上
vi system/fvSolution              # 圧力の線型ソルバを必要に応じて変更
./Allrun.batch ofp1de.sh ofp2solveInit.sh ofp3solve.sh # 領域分割からソルバ解析までを実行
pjstat -E                         # -E: ステップジョブ内のサブジョブを展開して表示
cd ..
cp -a case-mesh case-16-512-scotch # 以下同様
```

解析時間の集計結果表示

```
cd ~/lecture-OpenFOAM/AhmedBody/Ahmed25
./averageExecutionTime.sh
```

28　補足

28.1　Intel MPIライブラリでの実行エラー

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，以下のようにして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.

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
```

28.2　Oakforest-PACSでのOpenFOAMのコンパイル

OpenFOAM自動ビルドスクリプトinstallOpenFOAMのWebページを参考にコンパイルする．URL: https://gitlab.com/OpenCAE/installOpenFOAM/blob/master/README.md

参考文献

[1] Ercoftac classic database. URL: http://cfd.mace.manchester.ac.uk/ercoftac/.

[2] Oakforest-PACS利用支援ポータル「Oakforest-PACS システム 利用手引書」. URL: https://ofp-www.jcahpc.jp/.

[3] S.R. Ahmed, G. Ramm, and G. Faltin. Some salient features of the time-averaged ground vehicle wake. In *SAE Technical Paper*. SAE International, 02 1984. URL: https://doi.org/10.4271/840300, doi:10.4271/840300.

[4] Christof Hinterberger, M GarcÃŋa-Villalba, and W Rodi. Large eddy simulation of flow around the ahmed body. the aerodynamics of heavy vehicles: trucks, buses, and trains. *Numer Heat Transfer Part B*, pages 77–88, 01 2004.

[5] Hermann Lienhart, C Stoots, and S Becker. Flow and turbulence structures in the wake of a simplified car model (ahmed model). *Notes on Numerical Fluid Mechanics*, 77, 01 2002. doi:10.1007/978-3-540-45466-3_39.

[6] Matthieu Minguez, Richard Pasquetti, and Eric Serre. Spectral vanishing viscosity stabilized les of the ahmed body turbulent wake. *COMMUNICATIONS IN COMPUTATIONAL PHYSICS Commun. Comput. Phys*, 5:635–648, 03 2009.

[7] 今野 雅. OpenFOAMにおけるCommunication Avoiding CG法の実装と性能評価. 12 2017. URL: https://www.slideshare.net/MasashiImano/openfoamcommunication-avoiding-cg-84835454.

Copyright