LABORATORY OF CRYOSPHERIC SCIENCES (CRYOS)

OpenFOAM Tutorial Project

Running a snow transport simulation with *snowBedFoam 1.0.*: Guidelines

Developed for OpenFOAM version 2.3.0.

Author:

Océane HAMES

Supervisors:

Mahdi JAFARI

Michael LEHNING

Thursday 22nd July, 2021

Contents

1 Legend & Symbols

OpenFOAM terminal command (within OpenFOAM environment)

regular Linux terminal command

`OpenFOAM dictionary name`

OpenFOAM-related variable

Figure X-*00*, with X the figure and *00* the line number

2 Introduction

The present document contains guidelines to run simulations with the aeolian snow transport model that was implemented in the open source computational fluid dynamics (CFD) software OpenFOAM (OF). The model code was developed in the context of a master thesis within the CRYOS Laboratory of the Ecole Polytechnique Fédérale de Lausanne (EPFL) and the WSL Institute for Snow and Avalanche Research SLF, Switzerland. Two submodels were added to the original OpenFOAM Lagrangian library to simulate the transport of snow particles by the wind, in particular for medium- (saltation) and small-sized (suspension) particles. The theoretical framework for snow transport processes and their related mathematical expressions, in addition to the OpenFOAM scripts embedding the different submodels for snow movement can be found in a complementary tutorial. The present document explains in details how to install OpenFOAM and run a standard snow transport case with the software.

3 Installing the core softwares

The standard version of the OpenFOAM software needs to be installed prior to running simulations with the *snowBedFoam 1.0.* model. It is recommended to employ a Linux system for running OpenFOAM. The installation guidelines that follow are meant for a computer running with Ubuntu 18.04 of the Linux distribution; they may slightly vary depending on the Linux version that is used.

3.1 OpenFOAM v2.3.0.

The Eulerian-Lagrangian snow transport model that we developed is implemented in the version 2.3.0. of OpenFOAM. This version can be installed following the same guidelines as for OpenFOAM 2.3.1: the only difference is to replace the 1 by 0 in the version number. To install OpenFOAM, do as follows:

1. Follow instructions for the given website, but with replacing 2.3.1. by 2.3.0:
https://openfoamwiki.net/index.php/Installation/Linux/OpenFOAM-2.3.1/Ubuntu#Ubuntu_18.04
2. Download the Source Pack for OpenFOAM 2.3.0 at:
https://openfoam.org/download/2-3-0-source
3. Copy the following line in your .bashrc file to set-up the OpenFOAM environment (should be done automatically if the guidelines are followed properly):
alias of230='source $HOME/OpenFOAM/OpenFOAM-2.3.0/etc/bashrc WM_NCOMPPROCS=10 WM_MPLIB=SYSTEMOPENMPI; export WM_CC=gcc-5; export WM_CXX=g++-5'
Once OpenFOAM is installed, the customized libraries can be added to the standard version of the model.

3.1.1 CRYOS-OF Snow transport model

In order to add the CRYOS aeolian snow transport model to the standard version of OpenFOAM 2.3.0, it is needed to copy within the **user** folder the modified version of the Lagrangian library (available on GitHub, see instructions at the end of this document). This directory can be created by typing in the terminal (with the OpenFOAM environment):
mkdir -p $WM_PROJECT_USER_DIR
Within this directory, copy the *src*, *run* and *applications* folders adapted to snow transport modelling and available on GitHub. Then, compile by running the commands:
of230 (to set-up the OF environment)
./wclean (to clean all the files that were compiled)
./wmake libso (to compile all the libraries)
./wmake libso (to have a summary of the compilation)
These commands need to be ran successively within the *src/lagrangianCRYOS/* folder inside the 1) *distributionModelsTriple* and 2) *intermediateCRYOS* subfolders. Once this is done, go into the *applications/solvers/snowBedFoam* folder and run the following commands:
./Allwclean (to clean all the files that were compiled)
./Allwmake (to compile all the libraries)
If there are any errors occurring during the compilation, their origin and type will appear in red

on the terminal window.

4 Meshing Procedure (before OpenFOAM)

4.1 Generating STL file

In order to create the specific topography on which the snowBedFoam solver is run, it is necessary to create a tri-dimensional representation of it. To do so, we created a STL file based on the ASC files of the sea ice relief. Multiple tri-surface format files exist - their type depends on the software that is employed to generate the mesh.

4.2 Meshing software

OpenFOAM supplies several meshing softwares such as SnappyHexMesh (*https://cfd.direct/openfoam/user-guide/v6-snappyhexmesh/*) or cfMesh (*https://cfmesh.com/cfmesh/*): more information is given on their respective websites. cfMesh has the advantage to allow parallel processing and a more automatized procedure, which can be a interesting depending on the situation. We employed the meshing software ANSYS® mesher because more familiar with the latter.

5 Running a case with OpenFOAM: General Procedure

5.1 Case structure

Before starting, note that it is always necessary to type the command of230 to set up the OpenFOAM environment. The simulation cases are located in the **run** folder of the user folder (accessible directly through the run command). We provided in our code repository a case example called "exampleCase" which should be copied inside the latter. To understand the structure of OpenFOAM cases, it is very useful to read the general OpenFOAM guide and more especially the case structure page at:
https://cfd.direct/openfoam/user-guide/v6-case-file-structure/.
To roughly summarize, the **constant** directory contains all the files related to flow and particle settings. This is where the flow (`turbulenceProperties`, Figure 8) and particle (`kinematicCloudProperties`, Figures 4-7) properties are controlled. The **system** directory contains all the files related to general simulation set-up such as start and end time, timestep etc.

located in the `controlDict` file (Figure 9). It is also in the **system** directory that pre-processing scripts such as `decomposeParDict` (Figure 11) or `createPatchDict` (Figures 2-3) are found. Finally, in the **0** directory are found for each computational variable (pressure, velocity, k, ϵ and others) the boundary conditions specified at every patch at the initial timestep. When running the simulations, some additional time directories are added with the variable values computed by OpenFOAM. They are many options possible and it is generally time-consuming to figure out the correct set of BCs: several tests are needed, for which the pressure and velocity behaviour should be thoroughly checked within the whole domain. Additional files directly located in the case directory such as `Allrunp` (Figure 1) or `Allrun` are used to call one by one the necessary applications to run a simulation. Basically, the whole simulation routine can be saved in one file (in our case: `Allrunp` to be ran in parallel) that can be run with the command ./file_name. The computer executes line by line the commands stored in these files. There are a good way to remember exactly the steps to run a case. In the next sections, we refer to the execution lines of the `Allrunp` file in Fig.1 to describe the main steps.

```
#!/bin/sh
cd ${0%/*} || exit 1    # run from this directory

# Source tutorial run functions
. $WM_PROJECT_DIR/bin/tools/RunFunctions

rm -r log.*
rm -r *.obj

# Get application directory
application=`getApplication`
echo $application

## Get the number of processors to run on from system/decomposeParDict
nProc=$(getNumberOfProcessors)
echo "the number of processors to run on from system/decomposeParDict: $nProc

# Create mesh
runApplication fluent3DMeshToFoam MOSAiC_seaice.msh

# Re-assign the patches
runApplication createPatch -overwrite

# Distribute domain among processors
runApplication decomposePar

# Run snow transport model
runApplication mpirun -np $nProc $application -parallel
```

Figure 1: Example of `Allrunp` file to run sea ice simulations.

5.2 Main Steps

To run an OpenFOAM simulation, it is first necessary to create the mesh on which to run the simulations. This is doable with several softwares, as described above (Section 4.2). Once the mesh is created, there is a need to translate it into an OpenFOAM mesh format (Fig.1-*19*) and to re-define the boundary conditions (if a non-OpenFOAM meshing software is used, Fig.1-*22*). Then, the mesh in the appropriate format can be decomposed on several processors (Fig.1-*25*) . Once this is done, the CFD solver `snowBedFoam` can be run (Fig.1-*28*) . Most of the data analysis and visualization occurs with Paraview which is one of the third parties softwares of OpenFOAM. The main steps are described in more details hereafter.

5.3 Forums

There are many questions that have been asked and answered through online forums. The main one is called CFD-Online (*https://www.cfd-online.com*). If issues are encountered, the first resource is to look up there. Open source softwares such as OpenFOAM have the advantage to have a strong user community which is ready to help.

6 Before the OpenFOAM run: Pre-Processing

The main steps for pre-processing a simulation case are summarized in this section. The purpose here is to show the set-up for our snow transport simulations rather than fully detail all the OpenFOAM option settings: there is an infinity of them and it is better to refer to the OpenFOAM official guide for more information.

6.1 Mesh set-up

6.1.1 Conversion to appropriate mesh format

Once that the mesh has been imported as a .msh file directly from ANSYS mesher, it can be converted into an OpenFOAM mesh via the command fluent3DToFoam $mesh\_name$.msh. The mesh coordinates and boundary types can be investigated into the folder **constant/polyMesh**. More especially in the `boundary` file, the names and the types of the patches can be read as well as the number of constitutive cells. Note that this command may vary depending on the meshing software that is used. The appropriate commands are stated in the official documentation.

6.1.2 Renaming patches and defining their types

Once that the mesh is in the appropriate OF format, all the patches should be given a right name and boundary condition with the command createPatch -overwrite. This command invokes the `createPatchDict` dictionary (Figures 2-3) into which the user can specify the accurate names needed for each patch and the correct boundary conditions. Note that this step is also mesher-dependent. The `createPatchDict` dictionary allows to link the cyclic patches together (defining the exact correspondance between cells) such as in line *43-44* of Figure 2. Regarding the renaming of the patches, names such as xMin (Fig.2-*39*) and xMax (Fig. 2-*57*) are a good option for the patches perpendicular to the x-axis while yMin (Fig.3-*75*) and yMax (Fig. 3-*91*) are suited for the patches perpendicular to the y-axis. These settings depend on the preferences of the user, however.

Once that the patch names have been changed, the mesh quality can be checked through the command checkMesh -allGeometry -allTopology. This tells directly if the mesh is OK or not. If the expression "Mesh OK" does not appear at the end of the command, this means that the simulations are going to blow up and the results will not be consistent. If the mesh-check did not work, it is necessary to go back to ANSYS mesher (or other) and to re-create the mesh. Usually a solution consists in using smaller cell dimensions.

6.2 Flow and particle parameters

In order to set up the flow and particle parameters, the `kinematicCloudProperties` and `turbulenceProperties` dictionaries located in the **constant** subfolder are used. Here, the particle properties correspond to the ones of snow but they can be easily changed to any kind of material that is needed for the simulations.

6.2.1 Setting up the snow particle properties: kinematicCloudProperties

In the **constant** folder, the `kinematicCloudProperties` file controls all the simulation settings related to the snow particles model, as shown in Figures 4 to 7. Note that the variables with a $ symbol in front are defined in the `setUp` file within the simulation case folder.

At first, when testing the flow, it is important to put the `coupling` (Fig.4-*24*) and `active` (Fig.4-*23*) settings to false. This will totally delete the production of particles and allow to restrictly study the flow field. To summarize, the aerodynamic lift of snow can be activated and set up with the `logLawShearStress` option of the sub-model

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.1.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      createPatchDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
// This application/dictionary controls:
// - optional: create new patches from boundary faces (either given as
//   a set of patches or as a faceSet)
// - always: order faces on coupled patches such that they are opposite. This
//   is done for all coupled faces, not just for any patches created.
// - optional: synchronise points on coupled patches.
// - always: remove zero-sized (non-coupled) patches (that were not added)

// 1. Create cyclic:
// - specify where the faces should come from
// - specify the type of cyclic. If a rotational specify the rotationAxis
//   and centre to make matching easier
// - always create both halves in one invocation with correct 'neighbourPatch'
//   setting.
// - optionally pointSync true to guarantee points to line up.

pointSync false;

// Patches to create.
patches
(
    {
        // Name of new patch
        name xMin;
        // Dictionary to construct new patch from
        patchInfo
        {
            type cyclic;
            neighbourPatch xMax;

            transform           unknown;
        matchTolerance  0.01;
        }
        // How to construct: either from 'patches' or 'set'
        constructFrom patches;

        // If constructFrom = patches : names of patches. Wildcards allowed.
        patches (xMi);
    }
    {
        // Name of new patch
        name xMax;
```

Figure 2: Example of `createPatchDict` file, lines 1 to 57.

`bedAerodynamicLiftInjectionModel` (Fig.6-7, lines *158-183*). The patch where the particles should be lifted from (here, `snowBed`) is defined as well as the way shear stress should be computed (`tauLogLaw`, false for universal shear stress computation). The diameter properties (including statistical distributions for random sampling) as well as the time when

```
58
59          // Dictionary to construct new patch from
60          patchInfo
61          {
62              type cyclic;
63              neighbourPatch xMin;
64
65              transform       unknown;//rotational;
66              matchTolerance 0.01;
67          }
68          // How to construct: either from 'patches' or 'set'
69          constructFrom patches;
70          // If constructFrom = patches : names of patches. Wildcards allowed.
71          patches (xMi_shadow);
72      }
73      {
74          // Name of new patch
75          name yMin;
76          // Dictionary to construct new patch from
77          patchInfo
78          {
79              type cyclic;
80              neighbourPatch yMax;
81              transform unknown;
82              matchTolerance  0.01;
83          }
84          // How to construct: either from 'patches' or 'set'
85          constructFrom patches;
86          // If constructFrom = patches : names of patches. Wildcards allowed.
87          patches (yMi);
88      }
89      {
90          // Name of new patch
91          name yMax;
92          // Dictionary to construct new patch from
93          patchInfo
94          {
95              type cyclic;
96              neighbourPatch yMin;
97              transform  unknown; //rotational;
98          matchTolerance  0.01;
99          }
100         // How to construct: either from 'patches' or 'set'
101         constructFrom patches;
102         // If constructFrom = patches : names of patches. Wildcards allowed.
103         patches (yMi_shadow);
104     }
105 );
106 // ************************************************************************* //
```

Figure 3: Example of `createPatchDict` file, lines 58 to 106

particles should start to be injected (start of activation, `SOA`) are also defined in the submodel coefficients. Similarly, the splashing-rebounding of particles can be controlled with the `patchInteractionModel` parameters (Figures 5-6, lines *100-147*). Several parameters can be set in this file such as the maximum probability of rebound P_r or the rebounding restitution coefficient. These parameters are detailed in the implementation tutorial complementary to the present one, together with the equations involving them. Finally, in case some precipitation particles need to be added, it is important to uncomment the `model1` in the `injectionModels` sub-section as show in Figure 5, lines *70-95*. The *massTotal* is computed based on the total number of particles that should be injected within *duration* seconds. Based on the mean particle diameter specified in *expectation*, the total particle volume can be found and then the mass to be injected can be derived using the particle density.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      kinematicCloudProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include        "../setUp"  // include file with all the settings

// NUMERICAL SETTINGS
solution
{
    // Activation of particles and coupling method
    active          true;//false; // activation of particles
    coupled         true;//false; //two-way coupling
    transient       yes;
    cellValueSourceCorrection on;

    // Interpolation method
    interpolationSchemes
    {
        rho             cell;
        U               cellPoint;
        mu              cell;
    }

    // Integration method
    integrationSchemes
    {
        U               analytical; //Euler;
    }

    // Relaxation
    sourceTerms
    {
        schemes
        {
           U semiImplicit 0.5; //the number is relaxCoeff for the field
        }
    }
}
// FLOW PROPERTIES
constantProperties
{
    rho0            $rhoPar;
    Omega0          (0 0 0);
    alphaMax        1;
}
```

Figure 4: `kinematicCloudProperties` file, lines 1 to 57.

```
// PARTICLE SUBMODELS
subModels
{
    // Acting forces
    particleForces
    {
        sphereDrag;

        gravity;
    }

    // Injection models
    injectionModels
    {
        model1
        {
            type            patchInjection;
            parcelBasisType mass;
            patchName       atmosphere;    //patch for injecting particles
            U0             (0 0 -1.544);   //terminal fall velocity
            sizeDistribution               //particle size distribution
            {
                type        normal;
                normalDistribution
                {
                    expectation 0.0002;
                    variance 0.00005;
                    minValue 5e-5;
                    maxValue 0.0005;
                }
            }
            flowRateProfile constant 1;   //flow rate
            massTotal       3543.45;      //kg. Mass injected in "duration"
            SOI             0;
            duration        3600;         //s. Time duration where massTotal is injected.
            parcelsPerSecond 0.05e6;      //number of particles per s.
        }
    }
    // Particle dispersion model
    dispersionModel none;

    // Rebound-splash of snow particles
    patchInteractionModel   localInteractionStickReboundSplash; //none;

     localInteractionStickReboundSplashCoeffs
    {

        patches
        (
            atmosphere
            {
                type    escape;
                e       1.0;
                mu      0.0;
            }

```

Figure 5: `kinematicCloudProperties` file, lines 58 to 113.

6.2.2 Setting up the flow properties: turbulenceProperties

The flow properties are specified in the `turbulenceProperties` file, as shown in Figure 8. The type of turbulence model (here, Reynolds-Averaged Navier-Stokes (`RAS`)) is specified at line *17*

```
114                 // Snow surface settings
115                 snowBed
116                 {
117                 // Rebound coefficients
118                 type          stickReboundSplash;
119                 Pm            0.9;      //(-) max probability of rebound.
120                 gamma         2.0;      //(-) rebound
121
122                 // Splash entrainment coefficients
123                 epsilonr      0.25;     //(-) energy balance
124                 mur           0.5;      //(-) momentum balance
125                 muf           0.4;      //(-) momentum balance
126                 bEne          10e-9;    //(-) bed cohesion
127                 corrm         0.0;      //(-) mass-velocity correlation
128                 corre         0.0;      //(-) mass-velocity correlation
129                 pppMax        $pppMax; //maximum number of particles per parcel
130
131                 // Particle diameter properties
132                 dm            $dm;      //m. mean particle diameter
133                 ds            $ds;      //m. std deviation of diameter
134                 d_max         $d_max;   //m. maximum particle diameter
135                 d_min         $d_min;   //m. minimum particle diameter
136                 }
137             );
138             // Particle probability distributions
139             sizeDistributionTriple
140             {
141                 type          normalLogNormalExponential;
142                 normalLogNormalExponentialDistribution
143                 {
144                     //nothing
145                 }
146             }
147         }
148         heatTransferModel none;
149
150         surfaceFilmModel none;
151
152         collisionModel none;
153
154         stochasticCollisionModel none;
155
156         radiation off;
157
158         // Aerodynamic entrainment of snow particles
159         bedAerodynamicLiftInjectionModel  logLawShearStress; //none;
160         logLawShearStressCoeffs
161         {
162             // Particle probability distributions
163             sizeDistributionTriple
164             {
165                 type          normalLogNormalExponential;
166                 normalLogNormalExponentialDistribution
167                 {
168                     //nothing
169                 }
170             };
```

Figure 6: `kinematicCloudProperties` file, lines 114 to 170.

while the specific submodel (here, k-Epsilon (`kEpsilon`)) and its coefficient values are defined at lines *28-36*. This model can be changed, depending on the needs of the user.

```
        // Particle and coefficient settings
        aerodynamicLiftPatch    snowBed;    //patch for entrainement
        tauLogLaw               false;      //shear stress computation method
        dm                      $dm;        //m. mean particle diameter.
        ds                      $ds;        //m. std deviation of diameter
        d_max                   $d_max;     //m. maximum particle diameter
        d_min                   $d_min;     //m. minimum particle diameter
        z0                      $Z0;        //m. aerodynamic roughness.
        pppMin                  $pppMin;    //number of particles per parcel.
        SOA                     100;        //s. start of activation.
        Acst                    0.2;        //(-). fluid threshold coefficient.
    }
}

cloudFunctions
{

}

// ************************************************************************* //
```

Figure 7: `kinematicCloudProperties` file, lines 171 to 190.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
simulationType  RAS;

RAS
{
    RASModel        kEpsilon;

    turbulence      on;
    printCoeffs     on;


    // Optional model coefficients, kEpsilon
    kEpsilonCoeffs
    {
        Cmu             0.033;
    C1              1.44;
        C2              1.92;
        C3              0.0;
        sigmak          1.0;
        sigmaEps        1.85;
    }

// ************************************************************************* //
```

Figure 8: Example of a set up for RANS simulations, defined in the `turbulenceProperties` file.

6.3 General simulation settings

6.3.1 controlDict

Figure 9 shows an example of the `controlDict` dictionary content. The type of the solver that is employed in the simulation is set at the `application` line (*17*). To activate the snow transport model, the application should be set as `snowBedFoam`. The total time of the simulation corresponds to `endTime` and is specified in seconds. For memory purposes, it is important to use the command `purgeWrite` which controls the amount of timesteps after which the time directories are deleted. For example, if `purgeWrite` is set to 20, there would constantly be 20 time directories in the case folder, with the earliest being progressively deleted. This prevents memory problems to happen. Set `latestTime` for the `startFrom` option which will make the simulations start every time from the latest timestep. The maximum timestep `maxDeltaT` should not exceed 0.01 seconds for RANS, and even less for Large Eddy Simulations (LES). Another important parameter to set up is the so-called `maxCo` or maximum Courant number (see next section). More information on this can be found on the forums. If you need to average some fields over time, you can set it up in the `functions` option under the `fieldAverage` parameter in the `controlDict` dictionary.

6.3.2 setUp file

An important file directly located in the case folder is `setUp` as shown in Figure 10. Many parameters related to the flow and particles are defined in that file. More especially, the minimum (*d_min*), maximum (*d_max*), mean (*dm*) and standard deviation (*ds*) of the particle diameter used in the `kinematicCloudProperties` dictionary are changed directly in `setUp` (lines *27-30*). Moreover, the *pppMax* and *pppMin* parameters which represent the amount of particles per parcel in the rebound-splash and aerodynamic lift model, respectively, are tuned here. These variables have an impact on the real time taken to run the simulation. Regarding the flow, the surface roughness *Z0*, friction velocity applied in the forcing *Ustar* and normalized direction of the fluid forcing *flowDirection* are set in this file (lines *34-41*). More information on the use of these parameters can be found in the code implementation tutorial.

6.4 Running in parallel: decomposePar

The last step before launching the simulation consists in distributing the different sub-parts of the mesh to a given processor (if needed to run in parallel) by running the command decomposePar.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.3.0                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
application     snowBedFoam;              //snow transport solver

startFrom       latestTime;

startTime       0;                        //initial timestep (s)

stopAt          endTime;

endTime         1000;                     //end of simulation time (s)

deltaT          1.e-3;                    //if fixed timestep (s)

writeControl    adjustableRunTime;        //timestep adjusted to maxCo

writeInterval   1;

purgeWrite      100;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

adjustTimeStep  yes;

maxCo           2;                         //maximum CFL number

maxDeltaT       0.01;                      //maximum timestep (s)

// ************************************************************************* //
```

Figure 9: Example of `controlDict` dictionary.

The settings are controlled within the `decomposeParDict` dictionnary which is found in the **system** folder (Figure 11). The parameter *numberOfSubdomains* corresponds to the number of processors on which the simulation will be run. Once this is defined, the only parameter that needs to be modified in this dictionary is under the *simpleCoeffs* entry of the dictionary. The expression **n (x y z)** corresponds to the way the domain is decomposed in the x, y, and z direction, respectively. It should be noted that it creates less problems when the number of processors in the

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.4.x                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/

// ************************************************************************* //

zMax                 15;      // maximum z-extent of domain (m).

// Initial values for the variables.
p0                   0.0;     // initial pressure (normalized by density) (m^2/s^2).
nut0                 0.0;     // initial turbulent viscosity (m^2/s).
nuTilda0             0.0;     // initial value for nuTilda (m^2/s).
k0                   0.24;    // initial turbulent kinetic energy (m^2/s^2).
epsilon0             14.855;  // initial turbulent dissipation energy (m^2/s^2).
omega0               440.15;  // initial value for omega (s^-1).

// General conditions and parameters for flow.
nu                   1.134e-5;// continuous phase field-kinetic viscosity (m^2/s).
rho                  1.41034; // continuous phase field density (kg/m3).

// General parameters for the Lagrangian particles (cfr kinematicCloudProperties).
rhoPar          900;          // particle density (kg m-3)
dm              0.0002;       // mean particle diameter (m)
ds              0.00005;      // std deviation of particle diameter (m)
d_max           0.0005;       // maximum particle diameter (m)
d_min           5e-5;         // minimum particle diameter (m)
pppMax          1000;         // number of particles per parcel (splash entrainment)
pppMin          100;          // number of particles per parcel (aerodynamic entrainment)

// General parameters for fluid forcing.
vKC            0.41;          // von Kármán constant (-)
Z0             0.00001;       // aerodynamic surface roughness (m)
Ustar          0.314;         // friction velocity (m s-1)
flowDirection  (0 1 0);       // direction vector of friction velocity, (x y z) with x+y+z=1
H              $zMax;         // vertical height for the fluid component (m)
noiseFactor     0.0;          // create artificial turbulence (noise factor)

// ************************************************************************* //
```

Figure 10: Example of the `setUp` file located in the simulation case.

z direction stays equall to 1. Thus, only the first two numbers in the brackets should be modified. The multiplication of the **x y z** terms must yield the number set in *numberOfSubdomains*. Other example, this time for 8 processors: n (4 2 1). The option *preservePatches* is used to put all of the cyclic patches on the same processors. Thus, the names in the brackets should be changed according to the names of the cyclic patches. Once the decomposition is done, directories called **processor*** will be created inside the case folder, and their number corresponds to the number of processors on which the numerical domain is decomposed.

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  2.2.2                                 |
|   \\  /    A nd           | Web:      www.OpenFOAM.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains  20;                     //number of processors
method          simple;                     //decomposition method
preservePatches (xMin xMax yMin yMax);  //for cyclic BCs

simpleCoeffs
{
    n           (5 4 1);                    //number of processors in x,y,z direction
    delta       0.001;
}

// ************************************************************************* //
```

Figure 11: Example of a decomposition in 20 processors as shown in the `decomposeParDict` file.

6.5 Launching the simulation

To run the simulation on one processor, start the simulation by going in the case folder and writing the name of your solver (after setting up the *of230* environment): snowBedFoam. To run in parallel, write the following command runParallel $application $nProc. All of these command lines, as well as the others, can found in the right order in the *Allrunp* file of the case directory. In case you want to run all the commands found in the *Allrunp* file, launch the following command within the terminal: ./Allrunp.

7 During the OpenFOAM run: Check

7.1 Courant Number

The Courant (Co) number is a very good indicator to check whether a simulation is set the right way. For example, the maximum Courant number should be less than 1 if using PISO algorithm. Note that most of the time when the simulations blow up (very high Co) it is due to skewed cells or other mesh issues. It should however be prevented by checking the mesh as explained above (section 6.1.2). The mesh is the core of the simulation and care should be taken when creating it. If the mesh is good, the simulations will for sure go without issues. If the Co number does not blow up after 5 seconds of simulation, this means the computations are stable and one can expect to lead successful simulations. Usually when a simulation is about to "explode" it is easily noticeable because the timesteps keep on getting smaller to respect the limit "maxCo" but in one of the cells the velocity is very high so at one point the timestep just cannot compensate to keep a low Co.

8 After the OpenFOAM run: Post-Processing

8.1 Reconstructing the simulation results

If ran in parallel, the OpenFOAM results are distributed within the time directories located in each processor folder. In order to re-asociate them into a single time directory, run the command reconstructPar in the terminal open within the case directory. To reconstruct the very last timestep that was simulated, run the command reconstructPar -latestTime. To reconstruct a given timestep XXX, run the command: reconstructPar -time XXX.

8.2 Paraview

Paraview allows to visualize the simulation results computed by OpenFOAM. To do so, create a file named *foam.foam* witihin the case directory (already present in *exampleCase*) and run the command paraview foam.foam in the OpenFOAM environment to visualize all the results. If the time directories were previously reconstructed, the results appear automatically. If the results are still distributed in the *processor* directories, select the option *decomposed case* in the options panel. This allows to visualize the results without reconstructing everything. In case you want to change the colors and the way the legend appears, activate the *View>Color Map Editor* in Paraview. There can also be set the limits and text of the color bar legend. If you want to add the contours of the surface topography on which the simulation was ran, select *Calculator* and choose the *zCoord* parameter. Then, select the *Contour* option and erase the default settings to set 10 for the countour lines (or whatever number of contour lines you want to appear in your results). There are other options for post-processing in OpenFOAM, which are fully detailed on the official OpenFOAM website.