Incorporation of OpenFOAM software into Computational Fluid Dynamics process in Volvo Technology

Master's Thesis in Computational Science and Engineering

Alexander Ivchenko
Incorporation of OpenFOAM software into Computational Fluid Dynamics process in Volvo Technology

Master’s thesis in Computational Science and Engineering

School of Information Science, Computer and Electrical Engineering
Halmstad University
Box 823, S-301 18 Halmstad, Sweden

June 2011
Acknowledgment

It is a pleasure for me to thank those who made this thesis possible and who helped me during my work.

I am heartily thankful to my supervisors at Volvo Technology Corporation: Anders Tenstam and Dalibor Cuturic for guiding me through the project and for the constant support that they gave to me. I am grateful to Peter Hansbo, my supervisor at Halmstad University, for his help and advice, especially at the late part of the thesis work.

I owe my deepest gratitude to 6170 department at Volvo Technology Corporation for providing the warm atmosphere I was working in. I would also like to give my special thanks to Maria Krantz, whose encouragement helped me a lot during the whole master’s project.

Alexander Ivchenko

Halmstad University, June 2011
Abstract

In this thesis work the feasibility of using open source OpenFOAM software as a solver part for Computation Fluid Dynamics in Volvo Technology is studied. Since the structure of the case in OpenFOAM is rather complex, one of the main purposes of this thesis work was also to make the process of using OpenFOAM as user-friendly as possible. The general conclusion that can be drawn from this work is that a very streamlined workflow can be, and has been, designed and created.
# Table of Contents

1  Introduction .................................................................................................................. 1

2  OpenFOAM ..................................................................................................................... 2

3  Overview of creating and solving a case in OpenFOAM ........................................... 3

3.1  Mesh ............................................................................................................................. 3

3.1.1  Creating mesh ...................................................................................................... 3

3.1.2  Importing mesh .................................................................................................... 3

3.1.3  Checking the mesh ............................................................................................... 3

3.2  Modifying the boundary conditions ........................................................................... 4

3.2.1  Modifying boundary file ....................................................................................... 4

3.2.2  Modifying boundary fields in 0 directory ............................................................ 4

3.3  Solving the case in parallel ......................................................................................... 4

3.3.1  Decomposing the case .......................................................................................... 4

3.3.2  Running the case in parallel ................................................................................ 5

3.3.3  Reconstructing the results ................................................................................... 5

3.4  Post-processing .......................................................................................................... 5

3.4.1  Using ParaView for post-processing .................................................................. 5

3.4.2  Mapping ................................................................................................................ 5

4  Drag and Lift calculations ............................................................................................. 6

5  Setting up and solving simpleTruck ............................................................................. 8

5.1  Mesh ............................................................................................................................. 8

5.1.1  Checking the mesh ............................................................................................... 10

5.2  Setting up boundary conditions ............................................................................... 11

5.3  Setting initial conditions .......................................................................................... 12

   Pressure p ...................................................................................................................... 13

   5.3.1  Velocity U .......................................................................................................... 14

   5.3.2  Turbulent kinetic energy K .............................................................................. 15

   5.3.3  Turbulent dissipation rate epsilon .................................................................. 16
1 Introduction

The Vehicle CFD team at Volvo Technology (VTEC) performs Computational Fluid Dynamics (CFD) analysis work within the Volvo Group. Some of the most resource demanding (both hardware and license-wise) analysis types encompass truck and bus aerodynamics, as well as the other sub-areas related to these. The main tool for CFD in VTEC is the proprietary software STAR-CCM+ made by CD-Adapco (1), which covers the whole computational process: pre-processing, solving and post-processing.

The main purpose of this thesis is to examine the feasibility of using open source OpenFOAM (2) as a code for the solver part of the process, using current pre-processing tools, and then to map results back to Star-CCM+. Since the case structure in OpenFOAM is quite complex and there are lots of options, it is necessary to make this process as user-friendly as possible. One of the main objectives of this thesis work was development, coding and testing of the family of scripts that takes into account the specific structure of Star-CCM + meshes.

The first step in the desirable working process with OpenFOAM is exporting mesh data from Star-CCM+. This mesh data should then be converted into the ready-to-be-solved OpenFOAM case. After that the cluster starts solving the case and during the solution process, the user should be able to judge the convergence. The final step is to map the results back to Star-CCM+, which will be used as a post-processing tool.
2 OpenFOAM

OpenFOAM is a free, open source CFD software package produced by a commercial company, OpenCFD Ltd (2). The core technology of OpenFOAM is a flexible collection of efficient C++ modules. It comes with a growing set of solvers applicable to a wide range of physical problems. The type of license is GPL (3). Generally, OpenFOAM has the same potential as most leading general-purpose proprietary packages, but it lacks native competitive pre- and post-processing utilities. However, it ships with: a script paraFoam for visualization meshes and solution data in ParaView (4), a wide range of mesh converters that allow import from a number of commercial packages. For this thesis work, we are particularly interested in the mesh-converter ccm26ToFoam that will be described in detail later on.
3 Overview of creating and solving a case in OpenFOAM

The usual start for creating a case in OpenFOAM is copying a tutorial with a solver that is physically similar to yours, to your run directory.

$ cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily $FOAM_RUN

Now you have a runnable case, but it has to be changed in order to suit your needs.

3.1 Mesh

3.1.1 Creating mesh

Meshes can be generated using blockMesh-utility, but it is not automatic and it can hardly be used for highly complex geometries that typically occur in commercial vehicle aerodynamics. The description of the mesh should be in the blockMeshDict file, and then the command for running blockMesh is:

$ blockMesh [-dict dictionary] [-case dir] [-blockTopo] [-region name]

3.1.2 Importing mesh

For aerodynamic purposes, where usually geometries are quite complex, the most interesting option is importing meshes generated by third-party software. For example

$ ccm26ToFoam <ccm26 file> [-case dir]

3.1.3 Checking the mesh

After the mesh has been generated it has to be checked. This can be done by the checkMesh-utility

If checkMesh found problematic areas in the mesh, it will place information about them in constant/polyMesh/sets directory. This information will be used later on to delete these problematic areas.

3.2 Modifying the boundary conditions

3.2.1 Modifying boundary file

If the mesh was imported, then, depending on whether the original mesh-file format from the third-party software has boundary-names and boundary-types or not, the conversion utility will direct this data to the file constant/polyMesh/boundary. The ccm-mesh files from Star-CCM+ contain boundary-names and boundary-types, thus ccm26ToFoam sets up appropriate names and types for boundaries in constant/polyMesh/boundary. Otherwise, they have to be changed manually.

3.2.2 Modifying boundary fields in 0 directory

After changing the boundary file, the field files in the 0 (zero) directory have to be either created or changed, depending on the turbulence model that will be used, and the original copied case. For a realizable k-epsilon model these fields are: U, p, k and epsilon, for k-omega SST these fields are: U, p, k, omega, nut. The conversion utilities do not change boundary fields and do not check the consistence between constant/polyMesh/boundary file and field-files, so it must be done manually.

3.3 Solving the case in parallel

3.3.1 Decomposing the case

In order to run the case in parallel, it needs first to be decomposed onto several processors. The description of the decomposition should be written in the file system/decomposeParDict. This file can be copied from tutorials

$ cp $FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak/system/decomposeParDict .
After copying, this file can be modified to the number of processors that will be used. The important point here is the type of decomposition; there are several possible types for that. We will use the “simple” type, because it is the easiest type for automatic decomposition, but later on it can be improved. Now the case can be decomposed


3.3.2 Running the case in parallel

After the case has been decomposed so that the fields and geometry are distributed across the desired number of processors the case is ready to be solved

$\text{foamRun <solver> <case dir> <number of nodes> <walltime>}

The output from the solver will be written in the file <case name>-<job number>.n200. From this file you can plot residuals using pyFoam library (5) that will be described later on.

3.3.3 Reconstructing the results

After the case has been successfully solved the solution from different processors and time directories should be merged into a single set of time directories for the whole model. The command for that is


Usually, for non-transient simulations, the interest is only the last time step, so the option -latestTime is very useful.

3.4 Post-processing

3.4.1 Using ParaView for post-processing

To see results in paraView you can simple write

$\text{paraFoam <-case dir> <-region name> <-touch>}

3.4.2 Mapping

The mapping procedure will be described later on.
4 Drag and Lift calculations

During the simulation, it is very useful to calculate the drag coefficient. The main interest here is not the final value of drag, because after the mapping procedure the final drag forces and associated coefficients will be calculated in Star-CCM+, but rather the iteration history of drag in order to judge convergence.

To calculate drag we need to add following functions from “libforces.so” library to the end of controlDict file

```
functions
{
    forces
    {
        type forces;
        functionObjectLibs ("libforces.so");
        patches (<list of patches, separated only with space>);
        rhoName rhoInf;
        rhoInf <reference density of the fluid>;
        CofR (0 0 0);
        outputControl timeStep;
        outputInterval 1;
    }
    forceCoeffs
    {
        // CofR - Centre of rotation
        // dragDir - Direction of drag coefficient
        // liftDir - Direction of lift coefficient
        // pitchAxis - Pitching moment axis

        type forceCoeffs;
        functionObjectLibs ("libforces.so");
        patches (Default Wheels1);
        rhoName rhoInf;
        rhoInf <reference density of the fluid>;
        CofR (0 0 0);

        liftDir (0 0 1);
        dragDir (1 0 0);
        pitchAxis (0 1 0);
    }
}
```

magUInf <free stream velocity magnitude>;
lRef <reference length of the truck>;
Aref <reference area of the truck>;

    outputControl timeStep;
    outputInterval 1;

After that, and during the simulation in the case directory, two folders will be created: forces and forceCoeffs. They will contain files with forces and force coefficients for each time step.
5 Setting up and solving simpleTruck

Let's take a closer look at the particular example of converting a simple truck model from Star-CCM+ to OpenFOAM. First we need to prepare the .ccm file, exported from Star-CCM+. This file will contain mesh and information about boundaries, such as names and types. At VTEC, this process is handled with an automatic Star-CCM+ script. This script is very powerful and these are the most crucial steps, from the point of OpenFOAM: a tunnel-box around the truck model will be created, all tunnel-box walls will be named according the certain convention, and the mesh will be generated. After that it is crucial to run another script that will delete prefixes like “000-” in names of the boundaries, which is essential because boundary-names in OpenFOAM, by definition, are identifiers: a sequence of digits and letters that begins with a letter. Then the case is ready to be exported into e.g. simpleTruck.ccm file.

Next step is to create a new OpenFOAM case. As was described in the previous chapter, the best way to do this is to copy an existing case. Since a realizable k-epsilon turbulence model in coupled with the simpleFoam solver is used, the most suitable case to copy is pitzDaily:

```
$ cp -r $FOAM_TUTORIALS/incompressible/simpleFoam/pitzDaily .
$ mv pitzDaily simpleTruckOF
$ cd simpleTruckOF
```

5.1 Mesh

Assuming that the simpleTruck.ccm file is located in the simpleTruckOF-directory, it is time to run the mesh converter

```
$ ccm26ToFoam simpleTruck.ccm
```

This utility will convert the mesh and overwrite files that contain mesh description in the constant/polyMesh directory: boundary, faces, neighbor, owner, points. It will also create two files in the 0/ directory: cellId and cellType. The first one (cellId) contains information about cell Id’s in Star-CCM+, but as it turns out it is (presently) a completely useless file because of a bug in the ccm26ToFoam utility: it loses precision, e.g., the cell with id 8540335 will be written as 8.54033e+06, and so this file can be immediately deleted. Since the code is open source, one
certainly has the option to change this and recompile, but is was not done here since the cell numbers are not crucial for the solving procedure. The second file (cellType) can be useful, because it has a template for defining initial conditions, for simpleTruck model it looks like:

//********************************************************************************
dimensions [0 0 0 0 0 0];
internalField uniform 1;
boundaryField
{
    Truck
    {
        type calculated;
        value uniform 0;
    }
    DOMAIN_INLET
    {
        type calculated;
        value uniform 0;
    }
    DOMAIN_OUTLET
    {
        type calculated;
        value uniform 0;
    }
    GROUND
    {
        type calculated;
        value uniform 0;
    }
    DOMAIN_SIDE_1
    {
        type symmetryPlane;
    }
    DOMAIN_SIDE_2
    {
        type symmetryPlane;
    }
    DOMAIN_TOP
    {
        type symmetryPlane;
    }
}

/****************************************************************************
5.1.1 Checking the mesh

Now the mesh has to be checked if it is good enough or not in terms of computational quality. If any problematic areas are found we try to modify them.

$ checkMesh$

After typing this command, the mesh is checked and the output is shown on the screen:

Create polyMesh for time = 0
Time = 0
Mesh stats
  points:      1114340
  faces:       3085643
  internal faces: 2913407
  cells:     983393
  boundary patches: 7
  point zones:   0
  face zones:   0
Overall number of cells of each type:
  hexahedra:  825671
  prisms:    49630
  wedges:    6634
  pyramids:  460
  tet wedges: 1394
  tetrahedra: 7008
  polyhedra:  92596
Checking topology...
  Boundary definition OK.
  Point usage OK.
<<Found 56 neighbouring cells with multiple inbetween faces.
  Upper triangular ordering OK.
<<Writing 114 unordered faces to set upperTriangularFace
  Face vertices OK.
Number of regions: 1 (OK).
Checking patch topology for multiply connected surfaces ...
  Patch  Faces  Points  Surface topology
  Truck   123489  121044  ok (non-closed singly connected)
  DOMAIN_INLET  340  379  ok (non-closed singly connected)
  DOMAIN_OUTLET 1363  1449  ok (non-closed singly connected)
  GROUND  45502  46543  ok (non-closed singly connected)
  DOMAIN_SIDE_1  691  750  ok (non-closed singly connected)
  DOMAIN_SIDE_2  691  750  ok (non-closed singly connected)
  DOMAIN_TOP   160  187  ok (non-closed singly connected)
Checking geometry...
Overall domain bounding box (-10 -15 -2.37671e-15) (40 15 20)
Mesh (non-empty, non-wedge) directions (1 1 1)
Mesh (non-empty) directions (1 1 1)
Boundary openness (-1.21492e-15 -8.63452e-16 -2.34319e-15) OK.
Max cell openness = 1.15815e-15 OK.
Max aspect ratio = 800.012 OK.
Minimum face area = 1.22407e-07. Maximum face area = 11.5449. Face area magnitudes OK.
Mesh non-orthogonality Max: 89.4533 average: 10.3337
*Number of severely non-orthogonal faces: 525.
Non-orthogonality check OK.
<<Writing 525 non-orthogonal faces to set nonOrthoFaces
***Error in face pyramids: 1 faces are incorrectly oriented.
<<Writing 1 faces with incorrect orientation to set wrongOrientedFaces
***Max skewness = 12.3452, 862 highly skew faces detected which may impair the quality of the results
<<Writing 862 skew faces to set skewFaces
Failed 2 mesh checks.
End

The utility makes a qualitative analysis of the mesh. As we can see, checkMesh reports several aspects of topological and geometric properties of the mesh, such as orthogonality, skewness, orientation or number of regions. The standards of quality in OpenFOAM and in Star-CCM+ are different: Star-CCM+ also has its own checking mesh utility but no clear correlation between the results of this one and the results of checkMesh have been obtained. For the simpleTruck model, checkMesh found: nonOrthoFaces, wrongOrientedFaces and skewFaces. NonOrthoFaces is considered as a “one star error”, which means it has a lower level of importance, so let's describe the procedure of deleting skewFaces and wrongOrientedFaces. Since the subsetMesh-utility which is used for deleting bad cells also rewrites field files, they have to be modified before.

5.2 Setting up boundary conditions

So far the polyMesh/boundary file that contains boundary-conditions looks like

```plaintext
// ************************************************************************** //
7
{
    Truck
    {
        type wall;
        nFaces 123489;
        startFace 2913407;
    }
    DOMAIN_INLET
    {
        type patch;
        nFaces 340;
        startFace 3036896;
    }
```
The above layout shows a set of desirable boundary conditions. Here we can see that tunnel walls have type symmetryPlane to reduce its influence on the solution; truck and ground have type wall; inlet and outlet have type patch. After checking the boundary file we are moving on to the 0 directory, where the initial conditions have to be set for our case.

5.3 Setting initial conditions

The 0 directory contains so far the files: epsilon, k, nut, nuTilda, p, R, U and temporary cellType. For k-epsilon turbulence model the essential files here are only U, p, k and epsilon. Therefore others can be deleted, except cellType which will be used as a template and be deleted later on.
5.3.1 Pressure $p$

Copy the boundary field content of the cellType file into the pressure after internal field. The outlet type should be fixedValue with the value 0 Pa, while the type of all other boundaries (except symmetryPlane, of course) should be zeroGradient

```plaintext
dimensions [0 2 -2 0 0 0 0];
internalField uniform 0;
boundaryField
{
    Truck
    {
        type zeroGradient;
    }
    DOMAIN_INLET
    {
        type zeroGradient;
    }
    DOMAIN_OUTLET
    {
        type fixedValue;
        value uniform 0;
    }
    GROUND
    {
        type zeroGradient;
    }
    DOMAIN_SIDE_1
    {
        type symmetryPlane;
    }
    DOMAIN_SIDE_2
    {
        type symmetryPlane;
    }
    DOMAIN_TOP
    {
        type symmetryPlane;
    }
}
```

The dimensions $[0 \ 2 \ -2 \ 0 \ 0 \ 0 \ 0]$; line in the beginning of the file describes the dimensions of the field. Each of the values corresponds to the power of each of the S.I. base units in $[\text{kg m s K mol A cd}]$. Here we can see that kinematic pressure is used $[\text{m}^2 / \text{s}^2]$. 


5.3.2 Velocity U

Let the value of the inlet velocity for this case be 25 km/h and let’s consider stationary ground. The type of the outlet should be zeroGradient. The internal field also has to be set; however, setting up the internal velocity the same as inlet velocity might not be a very good assumption for a non-homogenous field. The solution for that problem is another utility called potentialFoam which will be described later.

```plaintext
// *************************************************************************
// dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (25 0 0);
boundaryField
{
  Truck
  {
    type       calculated;
    value      uniform (0 0 0);
  }
  DOMAIN_INLET
  {
    type       calculated;
    value      uniform (25 0 0);
  }
  DOMAIN_OUTLET
  {
    type       zeroGradient;
  }
  GROUND
  {
    type       calculated;
    value      uniform (0 0 0);
  }
  DOMAIN_SIDE_1
  {
    type       symmetryPlane;
  }
  DOMAIN_SIDE_2
  {
    type       symmetryPlane;
  }
  DOMAIN_TOP
  {
    type       symmetryPlane;
  }
}
// *************************************************************************
```
5.3.3 Turbulent kinetic energy $K$

The value of turbulence kinetic energy may be calculated using the formula

$$k = 1.5 \cdot U^2 \cdot I^2$$

Where $U$ is velocity and $I$ is turbulence intensity, which has the usual value for controlled external flows of 0.001 (or 0.1%), but in real-life applications it may vary considerably. It gives $k = 0.0009375 \text{ m}^2 \text{s}^{-2}$. This value will be used as an internal field and the value at the inlet, other patches have type zeroGradient

```
// ************************************************************************* //
dimensions [0 2 -2 0 0 0];
internalField uniform 0.0009375;
boundaryField
{
  Truck
  {
    type zeroGradient;
  }

  DOMAIN_INLET
  {
    type fixedValue;
    value uniform 0.0009375;
  }

  DOMAIN_OUTLET
  {
    type zeroGradient;
  }

  GROUND
  {
    type zeroGradient;
  }

  DOMAIN_SIDE_1
  {
    type symmetryPlane;
  }

  DOMAIN_SIDE_2
  {
    type symmetryPlane;
  }

  DOMAIN_TOP
  {
    type symmetryPlane;
  }
}
// ************************************************************************* //
```
5.3.4 Turbulent dissipation rate epsilon

The value of turbulence dissipation rate may be calculated using the formula

$$\epsilon = C^{0.75} \cdot k^{1.5} / L$$

Where $C$ is a model constant that typically has a value of 0.09 and $L$ is turbulence or eddy length scale. For $L = 0.001$ m the value of $\epsilon = 0.000471$. This value will be used as an internal field and the value at the inlet, other patches have type zeroGradient

```plaintext
// ************************************************************************* //
dimensions [0 2 -3 0 0 0 0];
internalField uniform 0.000471;
boundaryField
{
    Truck
    {
        type zeroGradient;
    }
    DOMAIN_INLET
    {
        type fixedValue;
        value uniform 0.000471;
    }
    DOMAIN_OUTLET
    {
        type zeroGradient;
    }
    GROUND
    {
        type zeroGradient;
    }
    DOMAIN_SIDE_1
    {
        type symmetryPlane;
    }
    DOMAIN_SIDE_2
    {
        type symmetryPlane;
    }
    DOMAIN_TO
    {
        type symmetryPlane;
    }
}
// ************************************************************************* //
```
5.4 Deleting bad cells

All cells that are considered bad for OpenFOAM may impair the quality of the results, or might even prohibit obtaining a solution at all. One way of solving that problem is to delete these inappropriate cells. Usually they are located in very narrow places so deleting them will not dramatically affect the results of a typical aerodynamic case, but will help to avoid unphysical results in these particular cells.

Deleting cells can be done by using the setSet and subsetMesh utilities. When checkMesh utility finds a particular type of inadmissible cells or faces it writes the description of them into the set file. The next step is to combine these sets of cells or sets of faces into one set of cells that can be deleted further on. Of course this operation should find for each face the cells that contain it. The responsible utility for that is setSet, the syntax of which is:

```
```

When utility was run without -batch argument – it works in dialog mode. But a more efficient way of using this utility is by providing -batch file with commands. This will allow us to make an automatization of this problem later. Let's delete skewFaces and wrongOriented faces from the simpleTruck mesh by creating the batch file batchForDeletingCells.batch:

```
cellSet cellsToDelete new
cellSet cellsToDelete add faceToCell wrongOrientedFaces any
cellSet cellsToDelete add faceToCell skewFaces any
cellSet cellsToDelete invert
cellSet cellsToDelete subset
quit
```

The syntax of first four commands is:

```
<cellSet|faceSet|pointSet> <setName> <action> <source>
```

For example, by the third command in this file we add to cellSet “cellsToDelete” all cells that contain faces from skewFaces-set. Then the cell set “cellsToDelete” has to be inverted, so we will be able to generate a new mesh from this set without undesirable cells. And then finally the new subset is created.
To feed this batch file to the setSet-utility the following command has to be typed:

```sh
$ setSet -batch batchForDeletingCells.batch
```

Now everything is ready to run subsetMesh on subset cellsToDelete. This utility will create a sub-mesh only containing cells from cellsToDelete.

```sh
$ subsetMesh cellsToDelete -overwrite
```

The new boundary patch was created after deleting cells : oldInternalFaces; it has type “empty” by default and this type has to be changed to “symmetryPlane” in order to reduce the influence of this boundary on the solution.

Note that after this procedure of deleting cells several regions may be created, and in this case the work cannot be continued, because within the limited framework of this thesis work no solution as how to work with several regions in OpenFOAM could be presented.

In the general case it is not possible to run the solver when more than one isolated region is present. The reason for this is that the continuum region encompassing the whole geometry is split and this means that the flow cannot pass from region to region.
5.5 Decomposing

Let us copy the decomposeParDict from the damBreak case

```
$cp /$FOAM_TUTORIALS/multiphase/interFoam/laminar/damBreak/system/decomposeParDict
```

And then change the number of processors to 16:

```
numberOfSubdomains 16;
method simple;
simpleCoeffs
{
  n    ( 8 2 1 );
  delta 0.001;
}
```

This means that the domain will be split up in 8 pieces in the x-direction and in 2 pieces in the y-direction. Now it is time for decomposing

```
$ decomposePar
```

The case is decomposed and it is ready to be solved

5.6 Solving in Parallel

First it is necessary to connect to the cluster. For better initial solution we can run potentialFoam first

```
$ foamRun <full path to potentialFoam> simpleTruckOF 16 100:0:0
```

Now with better initial solution we can run the main script

```
$ foamRun <full path to simpleFoam> simpleTruckOF 16 100:0:0
```
6 Mapping procedure

Incorporation of OpenFOAM into the CFD process in Volvo Technology assumes that Star-CCM+ will be used for post-processing. The mapping procedure allows us to do that. The idea here is that the solution from OpenFOAM will be transferred not cell by cell (which wouldn't work in case some cells were deleted in OpenFOAM) but instead the information of cell centers, along with the solution in the cell centers, will be used. The potential error introduced by linear interpolation should then be minimal, since the cell centers in the map file will actually coincide exactly between the mapped data and the cell centers of the original Star-CCM+ model. While transferring to Star-CCM+, using information about cell centers, the mapping procedure will decide to which cell it will map the current solution data. After this is done, all transferred information is used as an initial solution for Star-CCM+, and one iteration of the solver, with under relaxation set to zero, is executed. The purpose is not to change the flow field, as it might, by running another solver, but rather to force Star-CCM+ to calculate any secondary set of scalars from the primary variables provided through the mapping.

The example of input for a script, that is responsible for mapping, is:

```
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>U</th>
<th>V</th>
<th>W</th>
<th>p</th>
<th>k</th>
<th>eps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.07975</td>
<td>-0.0115755</td>
<td>0.924951</td>
<td>-0.615643</td>
<td>-1.51125</td>
<td>2.51764</td>
<td>6092.56</td>
<td>1.79506</td>
<td>1674.21</td>
</tr>
<tr>
<td>1.07999</td>
<td>-0.0135049</td>
<td>0.924828</td>
<td>-1.43387</td>
<td>-1.15893</td>
<td>2.18116</td>
<td>6090.45</td>
<td>1.52231</td>
<td>1698.66</td>
</tr>
<tr>
<td>1.07994</td>
<td>-0.0216556</td>
<td>0.924643</td>
<td>0.871544</td>
<td>3.68627</td>
<td>-1.40476</td>
<td>6143.68</td>
<td>1.69272</td>
<td>1546.88</td>
</tr>
<tr>
<td>1.07993</td>
<td>-0.0262015</td>
<td>0.924417</td>
<td>-0.906594</td>
<td>1.96717</td>
<td>1.15657</td>
<td>6198.17</td>
<td>1.10568</td>
<td>943.347</td>
</tr>
<tr>
<td>1.07966</td>
<td>-0.0151754</td>
<td>0.924587</td>
<td>0.403754</td>
<td>-0.30058</td>
<td>2.11675</td>
<td>6088.72</td>
<td>1.76681</td>
<td>2102.1</td>
</tr>
</tbody>
</table>
```

Where X, Y, Z are coordinates of cell centers; U, V, W are components of the velocity vector; P – pressure, k – turbulence kinetic energy, eps – turbulence dissipation rate.
7 Script designing

All actions that have been described are quite repetitive and routine, and most of all time consuming. It is very easy to make a mistake or to forget to do something along the way, especially because all work is done in text file format. There exist a library – PyFoam (5) - for efficient working with OpenFOAM, but at the same time this library doesn't take into consideration issues of working with Star-CCM+. However, this is not a reason for avoiding using PyFoam for some routine sub-tasks e.g. changing boundary type from “empty” to “symmetryPlane” in all boundary fields. Python (6) was chosen as a programming language for scripts due to the possibility of using functions from PyFoam.

7.1 Building case

Schematically the construction of the case can be seen in Fig 1:
Build case skeleton process are shown in Fig. 2:

pyFoamCaseBuilder is a native utility from PyFoam. The purpose of the pyFoamCaseBuilder is to build OpenFOAM cases that minimize the amount of user inputs, using xml-file as a template. An Xml-template that describes the full process of constructing a case for Realizable K-Epsilon model has been written. It is responsible for every part of construction from ccm26ToFoam mesh-converter to adding correct boundary-conditions.

For deleting cells with high aspect ratio, open cells and zero area faces the procedure for deleting bad cells is used.
Set up control files are shown in Fig 3:

1. Start
2. Writing RASPProperties file
3. Writing fvSchemes file
4. Writing fvSolution file
5. Appending force and forceCoeff functions to controlDict
6. Check if the number of processors greater than 1?
   - Yes: Create decomposeParDict
   - No: Continue
7. End

Setting up control files process. Figure 3
7.2 Deleting bad cells

In Fig. 4 we show the scratch for the procedure for deleting cells with specified type:

Deletion of bad cells process. Figure 4
8 Feasibility of using OF in current CFD process in Volvo Technology

It turns out that OpenFOAM is very sensitive to mesh quality, while the tolerance for mesh quality in Star-CCM+ is very high. This property of Star-CCM+ allows its mesh generator to create meshes with fairly bad cell quality without detriment to the solution. A typical example of a cell produced by Star-CCM+ that will present the OpenFOAM solver with problems is shown in figure 5.
In the Appendix, Table 1 with information about meshes for several cases is shown. For example, the usual amount of non orthogonal cells for a mesh of approximately 13 million cells is about 13 000, and generally this is absolutely inappropriate since the number of cells to delete will most likely change the geometry to an extent that it will affect the aerodynamics of interest. We found parameters for the Star-CCM+ mesh generator that allow us to reduce the number of bad cells, but this reduction is not dramatic and in general we cannot benefit much from this methodology. For simpler geometries the amount of bad cells is not that large and OpenFOAM can be used for solving. Such simpler geometries can arise at, for instance, parameter studies from more generic shape type trucks, or simplified geometries excluding a detailed chassis or engine bay.
**prepareOFCase.java:** StarCCM+ Java script to prepare model for OpenFOAM (deletes inappropriate namings, exports to .ccm file)

**createCaseFromCCM.py** with arguments

**deleteBadCells.py**

**decomposePar**

**foamRun**
(potentialFoam/SimpleFoam)

**reconstructPar**

**writeCellCenters** -latestTime

**Mapping_script.py**

**mapOFData.java:** StarCCM+ Java script to read .csv file into mesh for post processing

**.ccm file**

Created OpenFOAM case structure, without critical bad cells

**.csv file with coordinates and primitive variables**

Thesis work main purpose
9 Results and Conclusion

In this work, a study of the feasibility of using open source OpenFOAM as an alternative in parallel computing for applications within the Vehicle CFD team was carried out.

One general conclusion that can be drawn from this work, and that was actually the most important aspect of this thesis work, is that a very streamlined workflow can be, and has been, designed. It strives to utilize the best of two worlds, namely the powerful mesh generation capabilities of a third-party proprietary meshing software on one hand, and the utility of parallelizing and solving the case at a lower cost than using commercial licensing all the way on the other hand. At the same time, it provides the user with a capability to run open source software demanding only a minimum of interaction with the rather complex, high-learning-threshold Open Source code structure. It also provides the user with a proper set of convergence-monitoring tools like residuals and forces history.

For this specific case, when StarCCM+ was used as the meshing tool, it turns out that currently, OpenFOAM and StarCCM+ stand too far apart when it comes to solver sensibility and appropriate mesh quality and no reliable mesh settings can be found in StarCCM+ that convert to a runnable case in OpenFOAM, considering a fully detailed, complex geometry description of a complete truck, aimed for aerodynamics simulations.

For simpler geometries, like more generic shape vehicles, or cases with excluded underhood and chassis structures, OpenFOAM may well comply with current simulation requirements, although in this thesis work, no qualitatively comparison of results with a commercial code has been made.

It goes without saying, though, that once the process workflow framework is there, it might easily be changed so that if another commercial package is used for meshing, e.g. ANSYS Fluent, ANSA, Hypermesh or other, the same scripts can be used either through mesh conversion to a .ccm file format, or by changing the script so that the utility used is switched from the ccm26ToFoam to another corresponding utility.

No effort has been made in trying to lump all subsets of scripts into one single top script for creating the OpenFOAM case, deleting cells, running the analysis and
map back the data into StarCCM+. Although this is quite possible, it will in practice only be of real interest for a very standardized case where every step of the process can be predicted to behave in a consistent manner. As an example, a shell top script may run the preparation java macro, work through the whole chain of python scripts and OpenFOAM commands, and finally run the java script to map the data back.
References

## Appendix

<table>
<thead>
<tr>
<th>Name</th>
<th>#High aspect ratio cells</th>
<th>#Open cells</th>
<th>#Zero faces</th>
<th>#several Non-ortho faces</th>
<th>#Non-ortho errors faces</th>
<th>#Face pyramids</th>
<th>#Skew faces</th>
</tr>
</thead>
<tbody>
<tr>
<td>VNTruck-L4H4InRUAGTunnel-ModPrismLayers-NoZeroAreaFaces.ccm</td>
<td>721</td>
<td>0</td>
<td>5</td>
<td>40263</td>
<td>165</td>
<td>397</td>
<td>4601</td>
</tr>
<tr>
<td>VNTruck-L4H4InRUAGTunnel-ModPrismLayers-NoZeroAreaFaces-RemovedInvalidDefault.ccm</td>
<td>715</td>
<td>0</td>
<td>5</td>
<td>40187</td>
<td>165</td>
<td>395</td>
<td>4725</td>
</tr>
<tr>
<td>VNTruck-L4H4InRUAGTunnel-ModPrismLayers-NoZeroAreaFaces-RemovedInvalidStep2.ccm</td>
<td>713</td>
<td>0</td>
<td>5</td>
<td>40084</td>
<td>163</td>
<td>395</td>
<td>4789</td>
</tr>
<tr>
<td>VNTruck-L4H4InRUAGTunnel-ModPrismLayers-NoZeroAreaFaces+Chevron-RemovedInvalidStep2.ccm</td>
<td>713</td>
<td>0</td>
<td>5</td>
<td>37939</td>
<td>158</td>
<td>381</td>
<td>5113</td>
</tr>
<tr>
<td>VNTruck-L4H4InRUAGTunnel-RemeshedRaw.ccm</td>
<td>1 (1873.64)</td>
<td>0</td>
<td>6</td>
<td>12660</td>
<td>131</td>
<td>270</td>
<td>5753</td>
</tr>
<tr>
<td>VNTruck-L4H4InRUAGTunnel-RemeshedRemovedInvalid+chevron+ZFA.ccm</td>
<td>0 (747.827)</td>
<td>0</td>
<td>1</td>
<td>11187</td>
<td>117</td>
<td>248</td>
<td>4996</td>
</tr>
<tr>
<td>VNTruck-L4H4InRUAGTunnel-Remeshed-PrismLayerVar1Raw</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>9464</td>
<td>99</td>
<td>246</td>
<td>2423</td>
</tr>
<tr>
<td>VNTruck-TetraWithoutPrisms</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>398</td>
<td>15</td>
<td>153</td>
<td>2788</td>
</tr>
<tr>
<td>VNTruck-TrimWithoutPrisms</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>23815</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

*Number of bad cells for different meshes. Table 1*