AALBORG UNIVERSITY

Title :	**Numerical investigation of a BFR using OpenFOAM**
Topic :	**Fluids and Combustion Engineering**
Project period :	**10th Semester**
Project start :	**February 3rd 2008**
Report submitted :	**June 3rd 2008**
Page count :	**57**
Appendix :	**A-G**
Supplement :	**Found on enclosed CD**
Number printed :	**4**
Supervisors :	**Søren K. Kær**
Group :	**Face 10**
Institute :	**AAU - Institute of Energy Technology**
Written by :	

Christian Andersen **Niels E. L. Nielsen**

Abstract

The opensource CFD (Computational Fluid Dynamics) software package OpenFOAM has been investigated in this projekt. OpenFOAM was evaluated against results obtained from the commercial CFD program Fluent. The comparison was conducted using geometry of a BFR (Burner Flow Reactor) as base. The BFR have previously been investigated with particle combustion. OpenFOAM has no solver for particle combustion so the comparison are done using two approaches; a cold-flow simulation using a turbulent incompressible solver, and a gas combustion simulation with methane as fuel. The cold-flow simulation showed similar results for both Fluent and OpenFOAM. The gas combustion simulation were done using both fuel-lean and fuel-rich environment. For the fuel-lean simulation, the two codes, were very similar, but in the fuel-rich simulation the temperature profile deviated. The gas combustion model in OpenFOAM is a transient model and significant calculation time were needed. To compensate for this, development of a steady-state gas combustion model have been initiated. The results of the developed combustion models still need some work, before they can compete with commercial software. Overall the OpenFOAM toolbox is considered a solid starting point for developing new code, although considerable time is needed to "reverse engineer" the code.

Preface

This report have been written under the *Fluids and Combustion Engineering* graduate programme, 10th semester in the *Institute of Energy Technology - AAU*.

The report consists of three parts: the main report, a set of appendixes and a CD-rom. On the CD-rom all relevant source and case data can be found.

Tables and figures have been enumerated with the number of the chapter and the number of the figure in that chapter, *e.g.* "Figure 3.1". This figure will be the first figure in chapter 3. Appendixes are indicated with letters, *e.g.* "Appendix A".

Citations in the report have been made by the Harvard method, *e.g.* Jensen (1999).

Contents

Nomenclature

Latin Letters

A	Arrhenius constant	$cm^3/mol \cdot s$
b	Arrhenius constant	-
C_R	EBU model constant	-
C_r	Courant Number	-
C_{EDC}	EDC model constant	-
C'_R	EBU model constant	-
δ	Small number	-
Δt	Time step	s
E_A	Arrhenius constant	cal/mol
f_{mix}	Mixture fraction	-
f_{stoich}	Stoechiometric ratio	-
h	Enthalpy	kJ/kg
k	Turbulent kinetic energy	m^2/s^2
$\dot{m}$	Mass flow	kg/s
nuTilda	Turbulent kinetic viscosity	m^2/s
p	Pressure	Pa
R	Reynolds stress tensor	m^2/s^2
S_i	Source term	-
s	Stoechiometric ratio	-
Sc_T	Turbulent Schmidt number	-
$\bar{v}$	Mean linear velocity	m/s
T	Temperature	K
$\vec{u}$	Velocity vector	m/s
U	Velocity	m/s
$\tilde{Y}_i$	Favre average mass fraction	-
Y_i	Mass fraction	-
$Y_{fu,1}$	Mass fraction of fuel at inlet	-

Greek Letters

χ	Reacting fraction of fine structures	
δx	Cell dimension	-
ε	Turbulent dissipation rate	m^2/s^3
Γ	Mass diffusion coefficient	m/s^2
γ^*	Mass fraction of fine structures	-
ν	Kinematic viscosity	m/s^2
$\nabla\cdot$	Divergence operator	-
∇^2	Laplacian operator	-
ν_T	Turbulent kinematic viscosity	m/s^2
$\tilde{\dot{\omega}}$	Favre average reaction rate on mass basis	-
Φ	Equivalence ratio	-
ϕ	Random variable	-
$\bar{\rho}$	Mean density	kg/m^3
μ	Dynamic viscosity	$kg/m\cdot s$
ρ	Density	kg/m^3

Abbreviations

ASCII	American Standard Code for Information Interchange
BC	Boundary Condition
BFR	Burner Flow Reactor
BYU	Brigham Young University
CFD	Computational Fluid Dynamics
EBU	Eddy Break-Up
EDC	Eddy Dissipation Concept
GCC	Gnu Compiler Collection
PaSR	Partially stirred Reactor
TVD	Total Variation Diminishing

Subscripts

fu	Fuel
i	Fuel, oxidiser, product etc.
ox	Oxidiser
pr	Product

1 Introduction

The purpose of the present work is to investigate how open software for computational fluid dynamics (CFD) perform against commercial software. When using the term *open* it implies that the source code for the software is fully available and documented, also known as open-source software. OpenFOAM (Open Field Operation and Manipulation) is a open-source toolbox for solving anything from complex fluid flows involving chemical reactions, turbulence and heat transfer, to solid dynamics and electromagnetics. The structure of OpenFOAM is an environment, where it is relative easy to formulate systems of partial differential equations and solve them for a discretized field of operation.

The main advantage of OpenFOAM compared to the commercial counterparts, *e.g.* Fluent and Ansys CFX etc. is that the commercial programs are closed source. OpenFOAM is interesting because of the possibilities the open source has to offer the user i.e. to create custom solvers using already existing modules in the OpenFOAM toolbox or extending physical models ad hoc. Most commercial software offers a secondary language for customised models, but the interaction with the solver is limited by the software programmers. The user defined models are not an integrated part of the main solver in most commercial CFD packages, which makes the models inefficient compared to a fully integrated program.

OpenFOAM is not point-and-click CFD, however it offers the solvers and environment to extend them to individual needs. According to Olesen (2007) the CFD software is approximately four times the price of computer hardware at present time. The cost of CFD software limits the use to larger companies. OpenFOAM offers a free advanced toolbox for solving complex physical problems only limited by the users imagination and capabilities. The time to develop new models also has to be taken into account.

1.1 Problem orientation

To compare OpenFOAM with other software the present work is based on the geometry of the Burner Flow Reactor (BFR), located at Brigham Young University (BYU) in Utah USA. The BFR is a co-fired coal biomass burner and thus involves many physical areas such as flow, chemistry, thermodynamics, particles etc. On this basis OpenFOAM will be used to see what possibilities are available compared with Fluent and what results can be obtained using free software.

1.1.1 Introduction to the Burner Flow Reactor

The purpose of the Burner Flow Reactor (BFR) is to simulate the region of one burner in a full-size industrial powerplant. The BFR is generally used for validating new CFD code. The advantage is that it can be run under stable operating conditions with easier access for sampling species and temperatures. The BFR is an axi-symmetric, 200 kW, pulverised fuel, vertical-fired reactor with a swirling flow.

The dimensions used for the BFR model is depicted in a 2D drawing in figure 1.1. Figure 1.2 shows the location of air and fuel inlets as they are used in the present work.

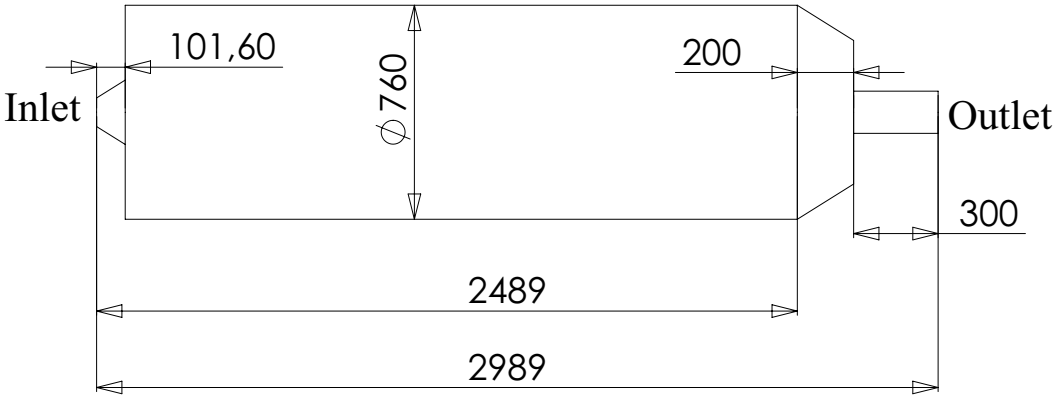


Figure 1.1: *Sketch of internal dimensions of the Burner Flow reactor. Dimensions are in mm.*

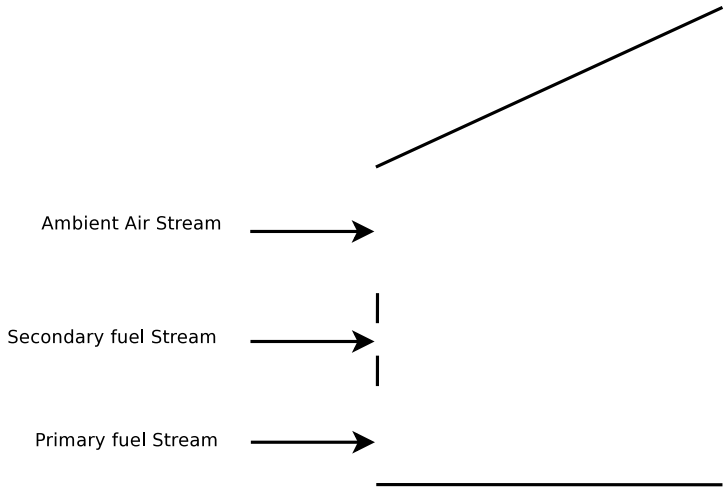


Figure 1.2: *Sketch of inlet conditions.*

1.1.2 Modeling the Burner Flow Reactor

Simulating turbulent combustion of coal and biomass particles is no trivial task. The Burner Flow Reactor combines many physical problems that need to be modelled or solved depending on the available resources. The model considerations in the present work are listed below for overview.

- Turbulent flow domain
- Particle trajectory
- Solid fuel pyrolysis (devolatilisation)
- Solid fuel combustion
- Gas combustion
- Thermodynamic model related to the chemistry
- Thermal radiation model

The turbulent flow controls the transport of both species and energy, therefore it is of great interest to have an accurate calculation of the flow field. The majority of industrial CFD that involves combustion make use of RANS (Reynolds Average Navier Stokes) turbulence models or Large eddy simulation, which is getting increasing popular because of increasing computational resources.

The combustion of solids introduce the challenge of tracking particles in the flow domain. Gas emission from coal or biomass particles are controlled by temperature, higher temperature means faster devolatilisation. According to Turns (2006), volatiles and tar make up to ∼70% of the mass of coal. The trajectory of the carbonaceous particles determines the combustion stages (gaseous combustion and char burnout) and thereby becomes an important part of the combustion model.

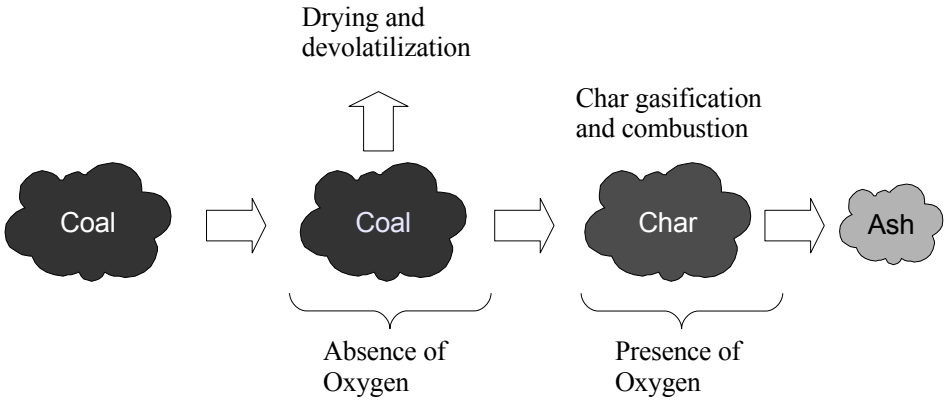
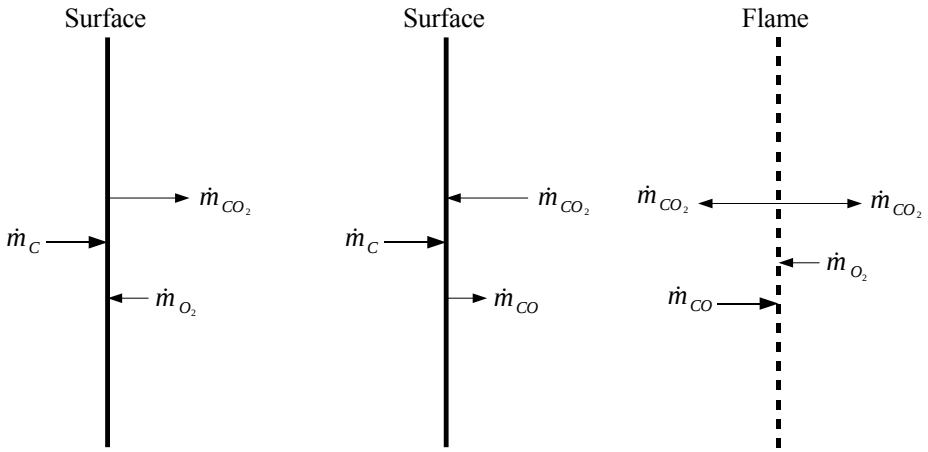


Figure 1.3: *Sketch of the devolatilization process (pyrolysis).*

In figure 1.3 the devolatilization process of coal is sketched. Pyrolysis is chemical decomposition of coal (or other organic materials) by heating in the absence of oxygen. The devolatilisation process can be modelled using Arrhenius-type rate coefficients, and can for the case of biomass or large particles be extended with multiple coefficients to account for non-isothermal pyrolysis in the particle as proposed by Smoot and Smith (1985).

Coal combustion implies modeling combustion of solid fuels, which is complex process to model. Two approaches are listed in Turns (2006); an one film model and a two

film model. The one film model assumes that the oxidation process occur at the coal particle surface and the intermediate specie CO is neglected. The two film model is somewhat more physical realistic, since it captures the reaction between carbon dioxide and carbon at the particle surface ($C + CO_2 \rightarrow 2CO$).



(a) One film carbon burning approach, the particle surface of the coal particle is sketched.

(b) Two film carbon burning approach, the particle surface and flame sheet are sketched.

Figure 1.4: *Schematics of the film modelling approach for carbon combustion, Turns (2006).*

The process depicted in figure 1.3 shows the initial drying and devolatilisation of the coal in a non-reactive environment. The heat supplied for heating the coal particle comes from external heating or from flame radiation. When the coal particle is "dried out" char remains, which is mostly carbon. The oxidation of char to form CO is a slow process since it is governed by diffusive mechanisms at the surface of the particle. Combustion of char is also depicted more schematic in figure 1.4 to give an overview of the reaction mechanisms.

Reaction rates in gaseous combustion are controlled by either kinetics or mixing rate, depending on the type of reaction. Empirical results have confirmed that most chemical reactions can be fitted to the Arrhenius collision theory, but also the influence of turbulence should be taken into account depending on the Damköhler number. Gaseous combustion is easier to account for, since it does not involve phase change and can be implemented through the source terms.

Thermodynamics play a significant role in simulating combustion, since it is the link between temperature and flow properties. The coupling between thermo-physical properties of mixture and energy release from combustion or other heat sources has significant influence on temperature distribution. Radiation makes up a substantial part of heat transfer in combusting flows and has a major influence on temperature distribution.

Transient simulation of a combustion might be more accurate, theoretically, but also demanding in computational resources and disk space. Time averaged results are easier to interpret and often produce reasonable accuracy for most applications. A steady-state combustion solver is therefore considered the most suitable choice for modeling the Burner Flow Reactor.

1.2 Problem definition

In section 1.1 the extent of modelling particle combustion in the BFR has been introduced. The purpose of the present work is to give an overview of the OpenFOAM toolbox and explore the possibilities available for developing new models.

For modeling solid fuel combustion, the first step is to develop a steady state gas combustion model. According to Wiki (2008), the most popular steady state combustion models are *mixture fraction*, *Eddy Break-Up (EBU)* and *Eddy Dissipation Concept (EDC)*. During the present work steady state combustion models will be implemented in OpenFOAM.

OpenFOAM contains a pre-build solver for transient-combustion using a RANS turbulence model and Chemkin thermodynamic tables. Cold flow and combustion simulations will be subject for comparison in order to evaluate how OpenFOAM performs relative to Fluent.


AALBORG UNIVERSITY
DENMARK

2
Introduction to OpenFOAM

2.1 Introduction

This chapter will give a brief introduction to OpenFOAM and how the program is used. To do this the structure of a case file will be described and how all the relevant constants and values are set. This will give the reader a better insight of the subsequent chapters which has focus on comparing some existing solvers to Fluent.
As mentioned earlier FOAM is short for *Field Operation and Manipulation*. The following is the developers own description of OpenFOAM, OpenFOAM (2008).

"OpenFOAM at its core, is a flexible set of C++ written modules. These are used to build solvers, to simulate specific problems in engineering mechanics. Utilities, to perform pre- and post-processing tasks and libraries, to create toolboxes that are accessible to the solvers/utilities, such as libraries of physical models.

OpenFOAM is shipped with numerous pre-configured solvers, utilities and libraries and so can be used like any typical simulation package. The difference is that FOAM is open, both in terms of source code and in its structure and hierarchical design. This makes the solvers, utilities and libraries fully extensible.

OpenFOAM employs finite volume numerics to solve systems of partial differential equations on any structured or unstructured mesh. The fluid flow solvers are developed within a robust, implicit, pressure-velocity, iterative solution framework. Domain decomposition parallelism is fundamental to the design of OpenFOAM and integrated at a low level so that solvers can generally be developed without the need for any "parallel-specific" coding."
A comparison of the functions in OpenFOAM versus Fluent can be found in appendix A.

2.2 OpenFOAM structure

OpenFOAM has different built-in utilities and solvers and relies on external programs for other tasks like other commercial CFD applications. This includes mesh generation, although a simple mesh tool called BlockMesh is available. BlockMesh is at present not an easy approach to mesh generation since all setup is done manually in text-files. It is preferable to use other mesh generation applications on complex geometries. OpenFOAM has mesh conversion tools which can transform mesh from other mesh generation programs to native OpenFOAM format.
Post-processing in OpenFOAM relies on external programs like ParaView, Fluent, Fieldview or Ensight. ParaView is although the preferable program because it has native reader for OpenFOAM and is free. OpenFOAM has the utilities to convert the results to the other commercially available post-processing applications. Figure 2.2 displays the OpenFOAM structure.

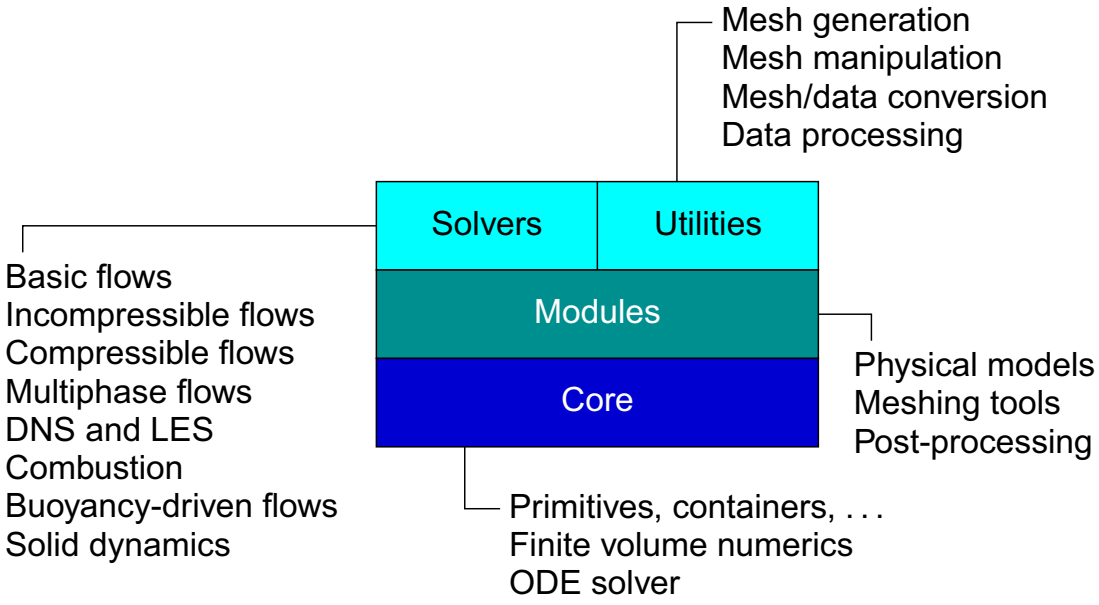


Figure 2.1: *OpenFOAM structure view, OpenCFD (2007)*

OpenFOAM is developed natively on a Linux/UNIX platform and specifically the GCC (Gnu Compiler Collection) C++ compiler. No ports to the windows platform has yet been developed although some have had mixed success using Cygwin for windows which is a Linux-like environment for Windows. Virtualization software like VirtualBox or WMware can be used to run a Linux environment on top of a windows installation. This effectively means that OpenFOAM is inherently free since no licenses are needed for the operating system (excluding UNIX). Figure 2.2 shows the implementation of OpenFOAM in a Linux/UNIX environment.

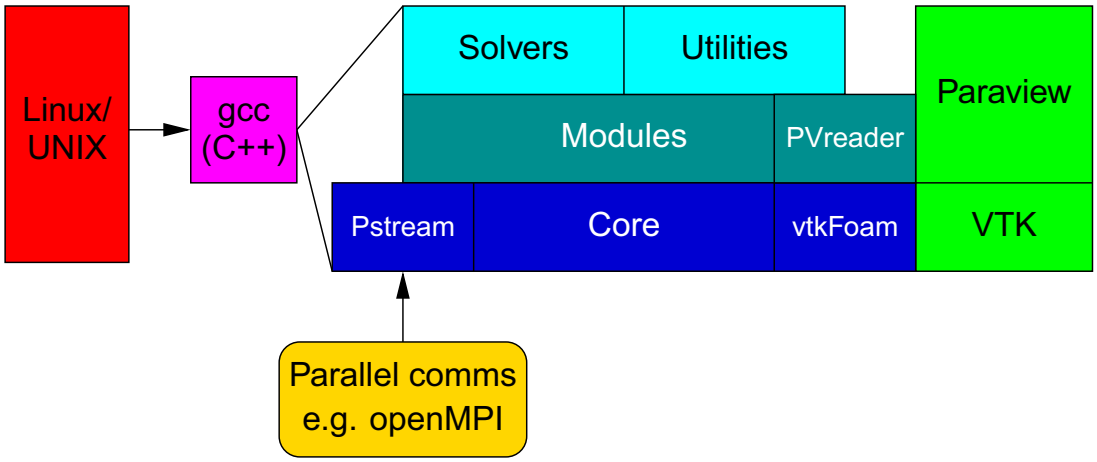


Figure 2.2: *OpenFOAM Linux/UNIX implementation, OpenCFD (2007)*

2.3 Basic case setup

Setting up a case to be used in OpenFOAM is significantly different from commercial CFD codes such as Fluent or CFX. All boundary conditions, mesh and case setup is done using text files in a specific folder structure. The folder structure is shown in figure 2.3. The following is a description of how a case is setup for a steady-state incompressible turbulence model. The description will give the reader a feel of where the values and constants for the case are given.

2.3.1 System

The *system* folder contains three files for setting various system specific properties.

controlDict

Contains the most basic setup for the case such as, start time, end time, write interval and iteration step. The values in this file can be updated while solving.

fvSchemes

Has the setup for gradient, divergence and laplacian schemes. A list of available schemes can be found in (OpenFOAM 2007, U-103). The schemes can be changed during solving so the user do not need to stop calculations for changing to higher or lower order schemes.

fvSolution

This file has the setup for the convergence criterion of the different variables and the setup for the pressure-velocity coupling algorithms. Also under-relaxation factors for the variables are defined in this file.

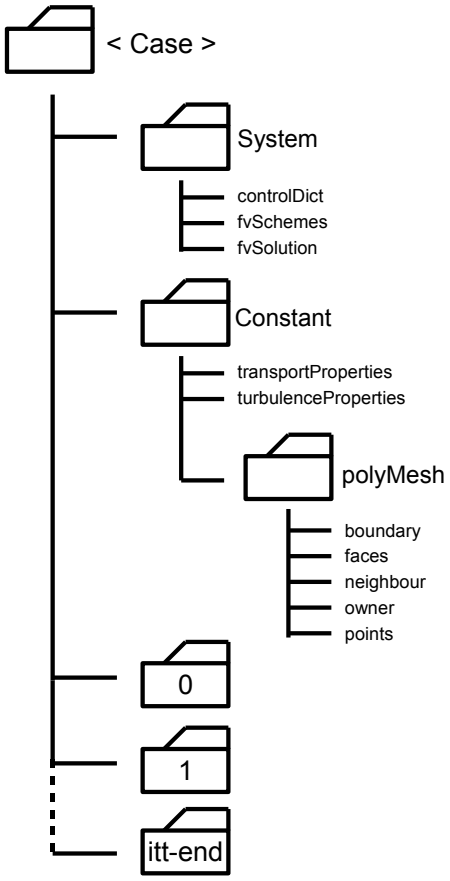


Figure 2.3: *Folder structure of a case file for* simpleFoam *steady-state incompressible model.*

2.3.2 Constant

The folder has two files where the physical properties of the fluid/species and the turbulence model are set.

transportProperties

In this file the physical properties for the fluid are set. For a steady-state incompressible solver only the kinematic viscosity ν of the fluid, is needed as input.

turbulenceProperties

Here the turbulence model is set and the constant for the turbulence models are also set here. OpenFOAM has both k-ε and k-ω models and derivatives of them. A list of the available turbulence models in OpenFOAM can be found in (OpenFOAM 2007, U-93). The turbulence model can also be switched off in this file.

2.3.3 polyMesh

This sub-folder has the geometry and boundary condition patches for the mesh. Using a mesh conversion tool will automatically create these files. A list of these tools can be found in (OpenFOAM 2007, U-86).

2.3.4 0,1,...,itt_end

These folders contain the calculated properties, such as velocity, temperature etc. The 0 folder are used to set the initial boundary conditions. The boundary conditions are set in separate files, so the velocity boundary conditions are set in a file *U* with patches corresponding to the boundary-type set in the *boundary* file in the *polyMesh* folder. For an incompressible solver six files are needed in the folder, *U, p, k, epsilon, R* and *nuTilda*. The amount of folders with calculated values depend on the write-interval set in the *controlDict* file.

2.4 Solving the case

When the case is setup with the desired mesh and boundary conditions the case are solved from command line using one of the solver listed in (OpenFOAM 2007, Table 3.10). The previous exposition of the case setup were done using *simpleFoam* as a base. To run this solver on the described case the following command is used.

```
$ simpleFoam . case
```

The . means that the case-folder is located in the folder that the user presently is in. The residuals are printed directly to the screen and no log files are created. If the user is interested in a log file in order to plot the residual curves another command can be run.

```
$ foamJob simpleFoam . case
```

This command automatically creates a file called *log* in the case folder. Running the command **foamLog** on this log file will create individual files for U, k, epsilon etc. that can be plotted using programs such as xmgrace, gnuplot or MatLab. The case will run until the specified end-time, set in the controlDict folder, has been reached.

2.5 Postprocessing the case

When an obtained solution have been reached the user can post-process the case using ParaView by issuing the command.

```
$ paraFoam . case
```

Figure 2.4 show a screen-dump of the post-processing program ParaView.

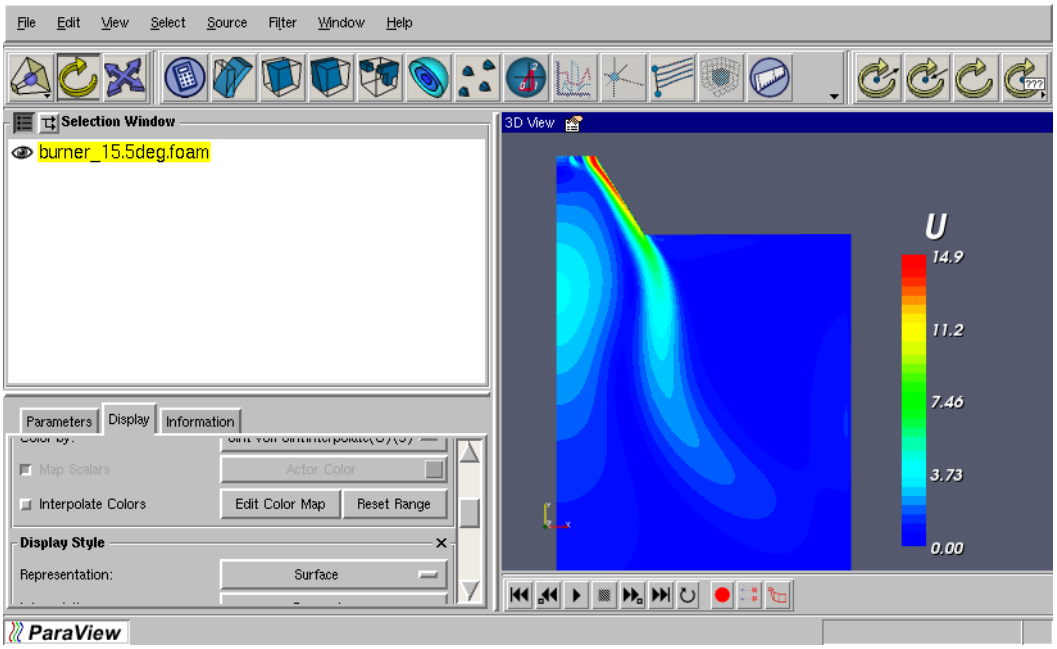


Figure 2.4: *ParaView graphical user interface.*

2.6 FoamX

OpenFOAM has a graphical user interface called FoamX, see figure 2.5, which can be used to pre-process the case. The interface is although not trivial to use and is prone to instability. Also an insight of how Linux operation systems handle libraries is needed since FoamX relies on very mandatory versions of each library in order to run properly.

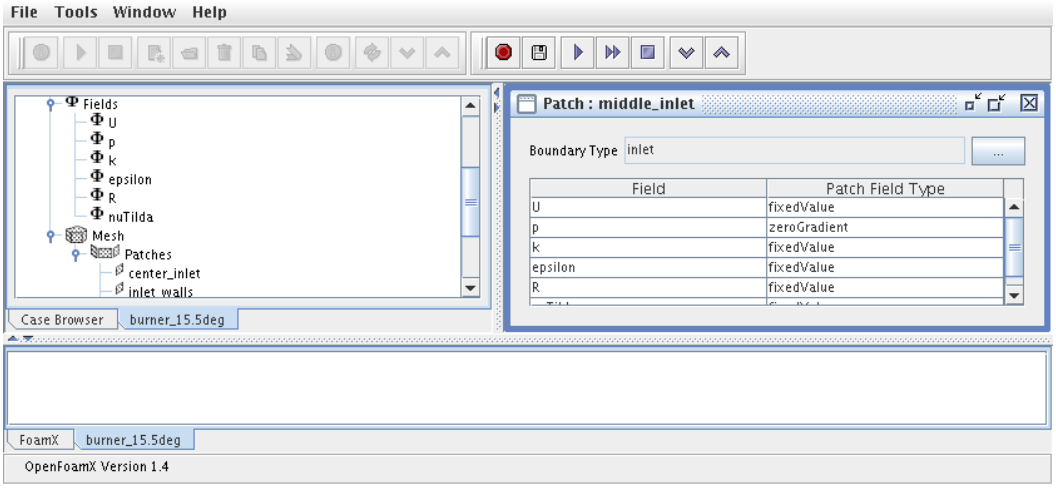


Figure 2.5: *FoamX graphical user interface.*

FoamX can be a good starting point when converting the mesh and setting boundary condition types. When saving the case all the needed files and folders, described earlier, are created. It is recommended to use this approach, but if modifications are needed to the boundary conditions or the input/output control, one should edit the files manually. The cases can also be solved from within the FoamX program, but the authors of this

report strongly advise against this.

2.7 Summary

The chapter has briefly described the OpenFOAM CFD toolbox and the requirements to run a OpenFOAM simulation. The initial steps for creating a case has been described which is not as straight forward as OpenFOAM's commercial counterparts. When the structure of OpenFOAM is understood and one gets familiar with the syntax of OpenFOAM it provides a great set of tools, but the time to familiarise oneself with OpenFOAM can be time consuming.


AALBORG UNIVERSITY
DENMARK

3

Cold flow comparison

3.1 Introduction

The purpose of this chapter is to compare an OpenFOAM simulation with a similar case in Fluent. The case in question is based on the geometry of the BFR described in chapter 1. The case setup is an comparison of the two codes using a steady-state incompressible turbulence model. This is done to see how the turbulence models compare, if these are similar, later calculations using reactions and chemistry can rule the turbulence from any errors. The chapter will describe how the case is setup, in detail, using OpenFOAM to give the reader a better understanding of how this is done.

3.2 Fluent setup

The burner geometry and case was obtained from Muff (2007) who has analysed the combustion of particles in the BFR by experiments and using Fluent. The case from Muff (2007) was modified only to solve for flow and pressure so all chemistry and energy equations are disabled. The boundary conditions for the case were obtained from Hvid (2006), and can be seen in table 3.1. The solver used is the standard k-ε turbulence model and *2nd order upwind* discretization scheme are used for all properties.

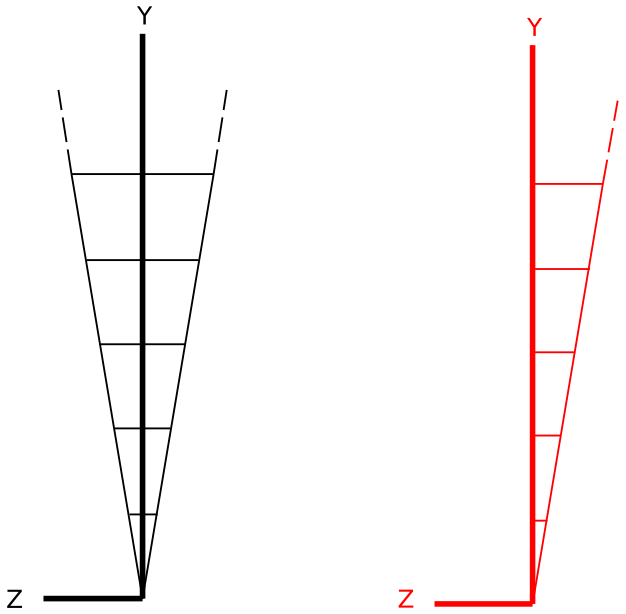
3.3 OpenFOAM setup

Since OpenFOAM is different from Fluent in certain areas, the OpenFOAM setup is described in greater detail. An understanding of OpenFOAM and how it is used was described in chapter 2.

3.3.1 Mesh

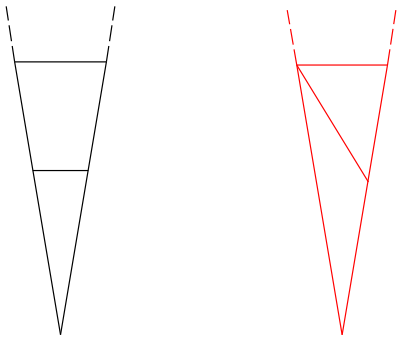
OpenFOAM is strictly a 3D code and thus cannot use the axi-symmetric mesh from Fluent. In order to create an axi-symmetric mesh for OpenFOAM some guidelines must be followed.

The mesh has to be created as a wedge straddling along the X-Y plane and the centre of the wedge **must** be in the Y-axis, see figure 3.1(a). The mesh in figure 3.1(b) cannot be used. The cone must have an angle of 2.5-5° and must only be **one** cell thick. The grid layout in figure 3.2(b) will not work even though it seems to be only one cell thick. The mesh has to be created as displayed in figure 3.2(a) to make it work with OpenFOAM.



(a) Correct (b) Incorrect

Figure 3.1: *How to create a successful axi-symmetric mesh for OpenFOAM, (a) correct (b) wrong*



(a) Correct (b) Incorrect

Figure 3.2: *How to make the mesh with a cell thickness of 1, (a) correct (b) wrong*

No mesh independency investigation has or was made. The mesh for OpenFOAM and Fluent can be seen in figure 3.3. The OpenFOAM mesh is more dense near the inlet than the Fluent mesh, but the overall cell count is the same.

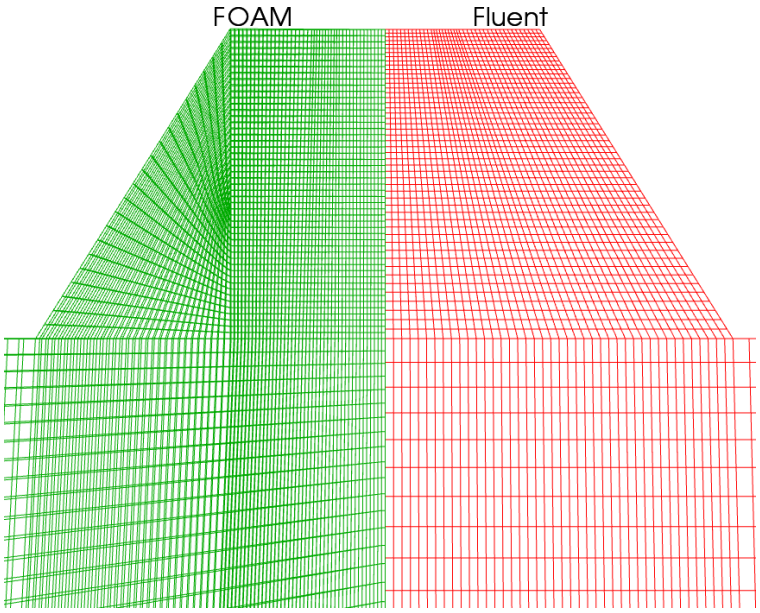


Figure 3.3: *OpenFOAM and Fluent mesh near the BFR inlet, the green mesh is OpenFOAM and the red is Fluent.*

3.3.2 Boundary conditions

The mesh was created using Gambit and converted to OpenFOAM format using the command

```
$ fluentMeshToFoam . case Fluent-mesh-file.msh -scale 0.001
```

If the mesh have been created in millimetres the *-scale 0.001* option, can be used to scale the mesh to meters while converting. After converting the mesh, the OpenFOAM mesh files are located in the *polyMesh* folder, see figure 2.3.
To setup the initial boundary conditions the *FoamX* case handler were used. Velocity inlets are used for the inlet conditions and pressure outlet is used for the outlet.
OpenFOAM also handles the physical properties of the fluid, that has to be solved for, differently than Fluent. Fluent use the dynamic viscosity μ and the density ρ whereas OpenFOAM use the kinematic viscosity ν for the incompressible solvers. The physical properties are obtained from Cengel (2003) and the values for μ, ρ and ν at 20°C are used in both OpenFOAM and Fluent. The kinematic viscosity is specified in the *transportProperties* file located in the *constant* folder in the OpenFOAM case.

3.3.3 Solver setup

The OpenFOAM case was run using the *simpleFoam* solver which is a steady-state incompressible turbulent solver. The boundary conditions were exported from Fluent into an ASCII text format so that the boundary conditions, for the secondary inlet,

would be the same. The discretization scheme used in OpenFOAM, for the vector field, is *limitedLinear* which is a 2nd order accurate bounded scheme. The reason for using bounded scheme was that using unbounded 2nd order scheme proved unstable. The schemes are set in the *fvSchemes* file in the *system* folder.

3.4 Boundary conditions

The boundary conditions for Fluent and OpenFOAM are listed below.

BC	U	k	ε
Centre inlet	2.053477	0.0375	0.15
Middle inlet	4.827447	0.0375	0.15
Secondary inlet	vector app. C	0.0375	0.15
Internal field	0	0.0375	0.15
Walls	0	zero gradient	zero gradient

Table 3.1: *Boundary conditions for simpleFoam and Fluent.*

3.5 Contour plot

Figure 3.4 displays combined vector, contour plots of the results from running the cases with 6000 iterations and a swirl angle of 9.5°.

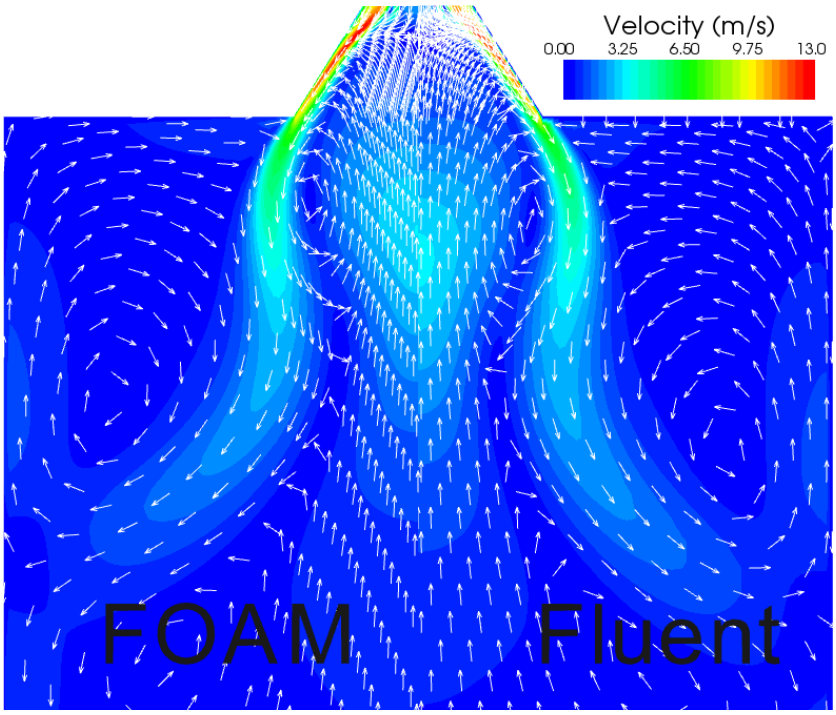


Figure 3.4: *OpenFOAM and Fluent vector/contour plot, of the velocity magnitude, near the inlet of the BFR.*

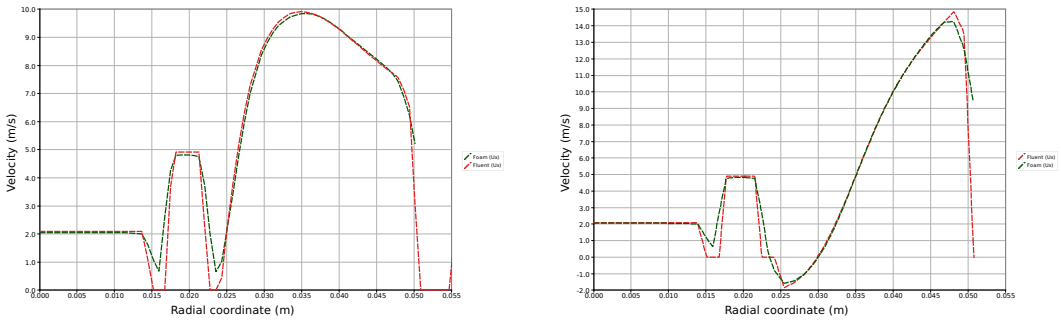
The figure shows the contours at a swirler angle of 9.5°. No significant difference in the two solvers are noticeable, and the difference that does exist is ascribed to be from the mesh and solver setups being slightly different.

3.6 Line plots

To compare the two cases quantitatively, the data was exported from Fluent to *Ensight GOLD* format. The results from OpenFOAM are cell based and thus ParaView interpolates the cell values to the node values. Lines at different axial locations were used to plot the axial, radial and tangential velocity.

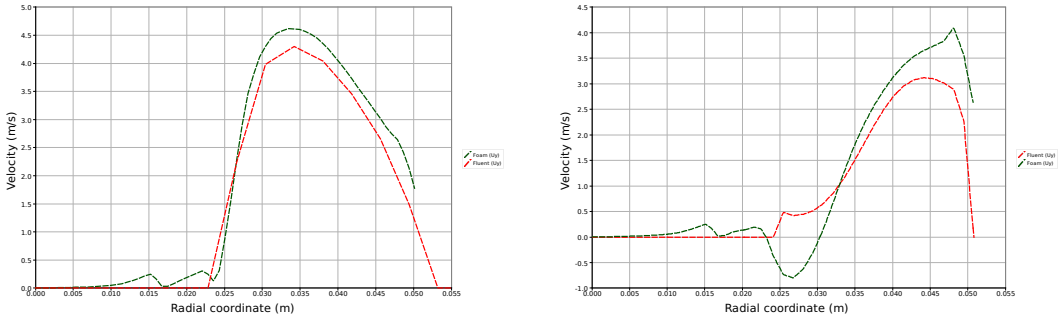
3.6.1 Inlet

Figure 3.5, 3.6 and 3.7 show the axial, radial and tangential inlet velocity profiles for the two cases. The difference in the plots are likely due to different interpolation for the different data sets in ParaView. The figures is with the swirler set at 9.5° and 15.5°.



(a) 9.5° (b) 15.5°

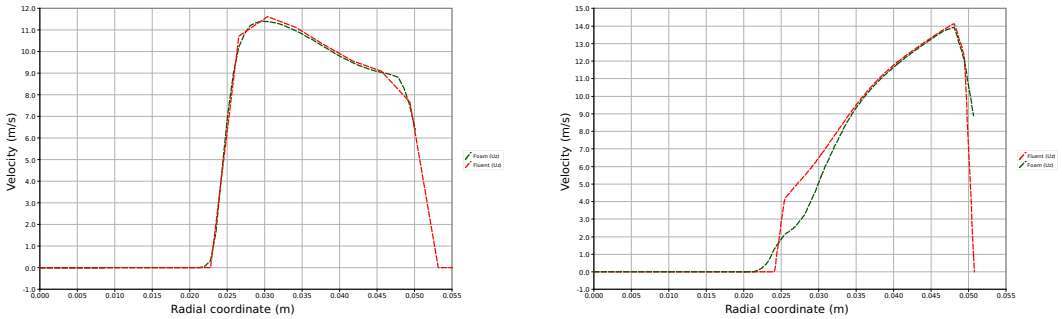
Figure 3.5: *Comparison of OpenFOAM and Fluent. Axial velocity at inlet face.*



(a) 9.5° (b) 15.5°

Figure 3.6: *Comparison of OpenFOAM and Fluent. Radial velocity at inlet face.*

With the swirler set at 15.5° the data are somewhat different but the gradients at this swirler angle is also higher. The discrepancy are accredited to the high gradients and the interpolation used in the post-processing program.

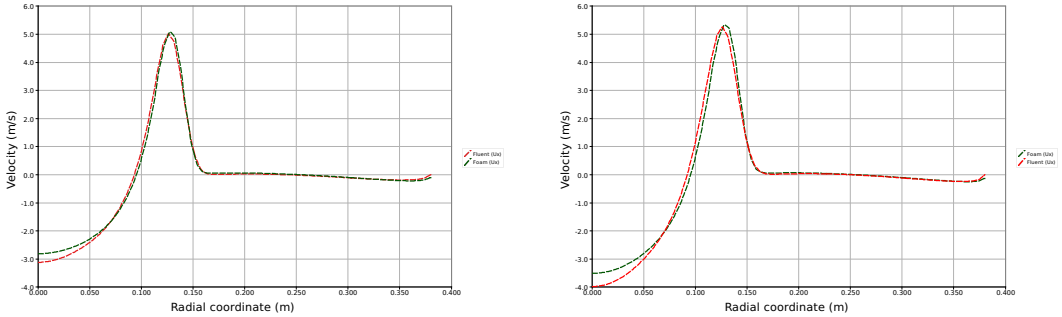


(a) 9.5° (b) 15.5°

Figure 3.7: *Comparison of OpenFOAM and Fluent. Tangential velocity at inlet face.*

3.6.2 15cm downstream

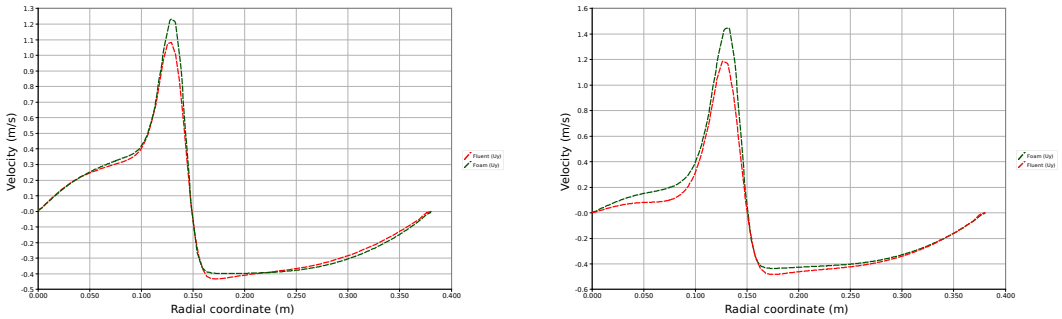
The same tendency can be seen further down the burner. Figure 3.8, 3.9 and 3.10 show the velocity profile for the cases 15cm downstream in the burner.



(a) 9.5° (b) 15.5°

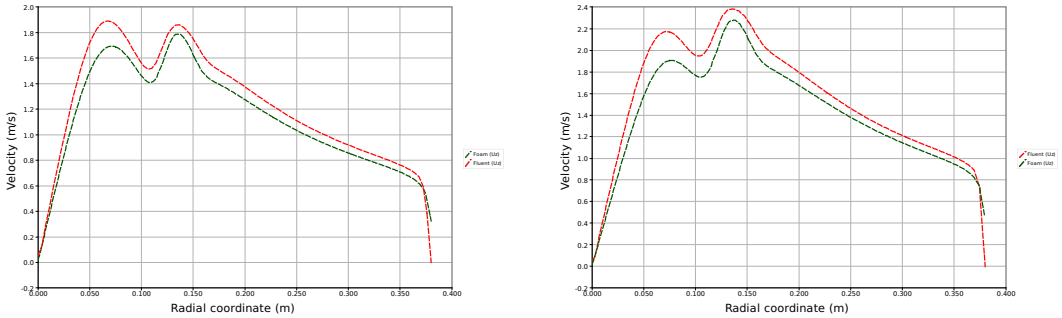
Figure 3.8: *Comparison of OpenFOAM and Fluent. Axial velocity at x=0.15m.*

Better agreement is obtained 15cm downstream than at the inlet. OpenFOAM has a slight offset compared to the Fluent data but the tendency in the curves is always the same. Experimental measurements could be conducted to verify which code is more accurate. The uncertainty in the experiments are although expected to be larger than the difference between the two codes. More plots are shown in appendix B.



(a) 9.5° (b) 15.5°

Figure 3.9: *Comparison of OpenFOAM and Fluent. Radial velocity at x=0.15m.*



(a) 9.5° (b) 15.5°

Figure 3.10: *Comparison of OpenFOAM and Fluent. Tangential velocity at x=0.15m.*

3.7 Summary

The chapter has compared Fluent and OpenFOAM using an incompressible turbulence model on the BFR. Only slight differences exists between the two solvers. The differences that exists between the two results are ascribed to the different file-formats and how ParaView interpolates the cell based results from OpenFOAM to the points. Also the strong gradients in the flow will contribute with some different results. Overall the two codes are considered to be on par with regards to the results, setup time and calculation time.


AALBORG UNIVERSITY
DENMARK

4

Reacting flow comparison

4.1 Introduction

The chapter is an extension of the previous chapter to include combustion into the turbulence model. OpenFOAM has solvers which can solve for different chemical gas reaction and combustion. OpenFOAM also has solvers for diesel particle combustion but only gas combustion models are considered.

Three solvers exist which can be handle gas combustion, this includes reactingFoam, XiFoam and Xoodles. All three solvers are transient models.

reactingFoam

The chemistry model in reactingFoam is a Partially Stirred Reactor Combustion Model or PaSR, which generally is used for turbulent non-premixed combustion. The PaSR model is a modified version of the Eddy Dissipation Concept (EDC) where the chemical timescale is handled differently, Chomiak and Karlsson (1996), Bhave and Kraft (2004). A description of the EDC can be found i chapter 5.

XiFoam

XiFoam is a compressible premixed/partially premixed combustion solver with turbulence modelling. XiFOAM is based on the Weller flamelet combustion model for RANS turbulence models, Weller et al. (1998).

Xoodles

Xoodles is a compressible premixed/partially premixed combustion solver also based on the Weller flamelet combustion model for Large Eddy Simulations (LES), Weller et al. (1998).

Generally all three models can be applied. Xoodles is although not considered since it is a LES (Large Eddy simulation) model and a much finer mesh is needed. In order to test the models the results will be compared to similar results from Fluent. To do this only methane (CH_4) will be considered as a fuel, since it is the simplest hydrocarbon and is often used for benchmarking. This provides a non-premixed mixture of gasses where the two coal/biomass fuel inlets, are substituted with methane and the secondary inlet is air. This rules out XiFoam as solver, since it is for premixed / partially premixed problems only. The comparison is based on reactingFoam and Fluent results.

4.2 Boundary conditions

The case is modelled with a swirl angle of 9.5° only. Two cases are simulated, fuel-rich and fuel-lean conditions, to see how the two codes compare. The fuel-rich case has an equivalence ratio of $\Phi = 1.21$ and the fuel-lean case has a an equivalence ratio of $\Phi = 0.855$. The equivalence ratio Φ is given as the stoichiometric air to fuel ratio divided by the actual air to fuel ratio, equation 4.1, Turns (2006).

$$\Phi = \frac{Air/Fuel_{stoich}}{Air/Fuel} \tag{4.1}$$

The fuel-rich boundary conditions can be seen in table 4.1 and the fuel-lean conditions in table 4.2.

BC	T(K)	CH_4	O_2	N_2
Centre inlet	300	1.0	0	0
Middle inlet	300	1.0	0	0
Secondary inlet	411	0	0.234	0.766
Internal field	2000	0	0.234	0.766
Walls	550	-	-	-

Table 4.1: *Boundary conditions for reactingFoam in the fuel-rich case $\Phi = 1.21$, temperature in Kelvin and species in mass fraction. Flow boundary conditions can be seen in table 3.1.*

The thermal boundary conditions for the walls is adiabatic with constant temperature. OpenFOAM does not support walls with energy-flux at present time.

BC	T(K)	CH_4	O_2	N_2	U
Centre inlet	300	1	0	0	2.339441
Middle inlet	300	1	0	0	2.382512
Secondary inlet	411	0	0.234	0.766	Vector
Internal field	2000	0	0.234	0.766	-
Walls	550	-	-	-	-

Table 4.2: *Boundary conditions for reactingFoam in the fuel-lean case* $\Phi = 0.855$*, temperature in Kelvin and species in mass fraction. Turbulence BC's is the same as for the fuel-rich case.*

4.3 Fluent

The Fluent case is setup with the EDC (Eddy Dissipation Concept) model, with a single step methane air reaction and the same boundary conditions as in table 4.1 and 4.2 are used. The EDC model is used since Fluent has no implementation of the PaSR model which reactingFoam is based on, a further description of the EDC model can be found in chapter 5. The Fluent case is solved steady-state with the recommendations on chemistry from Fluent (2005). The schemes used in reactingFoam, are sought to be used in Fluent.

Solver	
Steady-state	
Pressure based	
Pressure velocity coupling	SIMPLE
Schemes	
Pressure	Standard
Velocity	2nd. order upwind
Swirl	QUICK
Turbulence	1st. order upwind
Species	QUICK
Energy	QUICK

Table 4.3: *Fluent solver setup*

4.4 reactingFoam

reactingFoam is as mentioned earlier a transient chemical reaction solver. reactingFoam is the only model in OpenFOAM that can handle turbulent non-premixed chemical reactions/combustion.
Step for step this is how reactingFoam is executed. The code for reactingFoam is included in appendix D.

1. Calculate the chemistry based on the turbulent and chemical timescales (chemistry.H)

2. Calculate ρ (rhoEqn.H)
3. Calculate the velocity/pressure field (UEqn.H)
4. Read species and feed them to the chemistry solver (YEqn.H)
5. Calculate the temperature from the chemical reactions (hEqn.H) enthalpy lookup
6. Calculate the pressure field using PISO (pEqn.H)
7. Correct the turbulence (turbulence->correct()) pressure-corrector
8. Update ρ from the temperature (rho = thermo->rho())
9. return to step 1.

This is done until the specified end-time has been reached. The convergence criterion for each time-step is defined in the *fvSolution* file described in chapter 2

4.4.1 Chemistry

The chemistry solver in OpenFOAM is a strictly kinetic model and can thus model all reactions, being global or elementary. The chemistry reader is build around the Chemkin file format and can thus read any reaction scheme from Chemkin or other chemistry applications that can export in this format.

Using the Chemkin file format allows one to construct reaction schemes and thermophysical properties for species not included in OpenFOAM's standard libraries, without having to modify the underlying solver code. The thermophysical properties are based on the NASA profiles for specific heat, enthalpy and entropy, Mech (2008). The chemistry in the reactingFoam solver used for this specific case is a single step global reaction with only CH_4 and O_2 reacting to produce CO_2 and H_2O. OpenFOAM has the thermophysical properties for the most common species and thus only the reaction has to be defined. The Chemkin file is based on the elements in the reacting such as H, N etc. and the species in the reaction, the reaction itself and the Arrhenius equation constants seen in equation 4.2.

$$k(T) = AT^b exp(-E_A/R_u T) \tag{4.2}$$

Where A, b, and E_A are three empirical parameters, Turns (2006). The Arrhenius equation is only used for laminar reactions. If turbulent combustion is enabled, reaction rates are based on which is the slowest, chemistry or mixing. The mixing or turbulent reaction rate is defined as the turbulent dissipation rate ε divided by the turbulent kinetic energy k, $turb_{mix} = \frac{\epsilon}{k}$. $\frac{\epsilon}{k}$ is also known as the turbulent time-scale, Fluent (2005). See chapter 5 for a further description of the mixing. The Chemkin file used for the reactingFoam case is as follows:

```
ELEMENTS
H   O    C N
END
SPECIES
CH4 O2 N2 CO2 H2O
```

```
END
REACTIONS
CH4 + 2O2 => CO2 + 2H2O 6.70091E+12 0.0 4.84149E+04! A, b, E_a
FORD / CH4 0.2 /
FORD / O2 1.3 /
END
```

The Arrhenius constants, and forward reaction values, are taken from Fluent in order to use the same values, Turns (2006). Fluent has the frequency factor A in $Kmol/m^3s$ and activation energy E_A in $J/Kmol$ where the Chemkin format are in $cm^3/mol - s$ and cal/mol respectively. Thus the Fluent constant for A has to be multiplied with 31.6 to go from $Kmol/m^3s$ to $cm^3/mol - s$ and E_A must be divided by 4184 to go from $J/Kmol$ to cal/mol.

4.4.2 Boundary conditions

The boundary conditions for reactingFoam is the same as in chapter 3 for the flow and turbulence. The new conditions is for the temperature and species. To add these boundary conditions, four more files has to be in the *0* directory. One for temperature T, methane CH_4, oxygen O_2 and nitrogen N_2. The product files will automatically be generated when reactions start. The boundary conditions for reactingFoam can be seen in table 4.1 and 4.2. Initialising both flow and chemistry at the same time proved to be unstable and thus initial guesses for turbulence and velocity were used from the cold flow steady-state solution obtained in chapter 3. This approach is also given as a good starting point in Fluent (2005).

4.4.3 Solution setup

To speed up the calculations of the transient reactingFoam model it is run in a parallel mode. When doing this considerations about the physics of the model has to be taken into account. Since the model handles chemical reactions the portion of the mesh that has these reactions will be computational expensive. Based on the solution of the cold-flow comparison some assumptions about the model can be made.

- Reactions occur near the inlet where mixing is high
- Flow downstream in the BFR has low velocities and no reactions

The assumptions are also graphically displayed in figure 4.1.

Figure 4.1: *Assumptions for making the model parallel.*

The grid is discretised into four domains. This is done since a cluster-node with four processors is available. Two ways of discretising the mesh are considered, horisontal and vertical discretisation. Figure 4.2 and 4.3 shows the two considered ways.

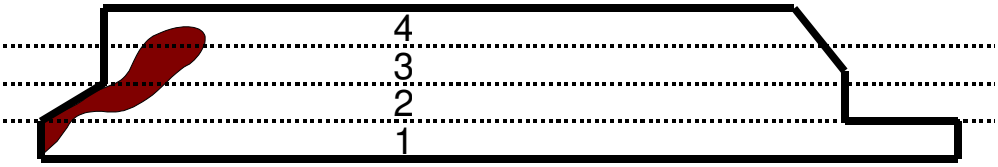


Figure 4.2: *Vertical discretization, here the processors see the same amount of computational mesh.*

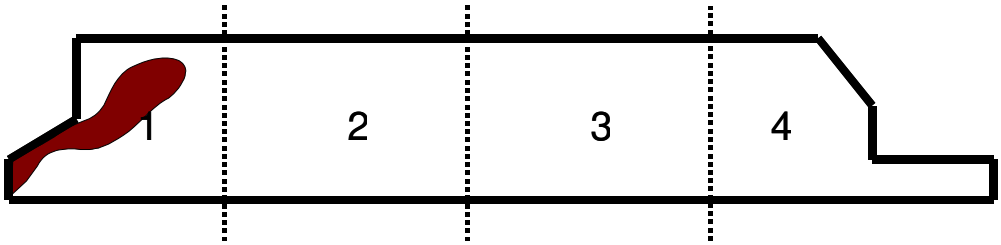


Figure 4.3: *Horizontal discretization, here only the first processor has to handle almost all the chemical calculations and the other processors will have to "wait" for the first one to finish.*

Vertical discretization is used to ensure that each processor has almost the same amount of flow and reaction equations to solve. The horizontal discretization is not considered since it is assumed that no significant speed-up can be gained.
To start the calculations, the mesh has to be decomposed according to the setup by the user. Parallel mesh setup is defined in the *decomposeParDict* located in the *system* folder of the case. The content of the file can be found in appendix E. Generally only two inputs are needed, How many sub-domains are wanted and in which directions should the mesh be decomposed. The processor weighing can also be defined if the solver is run on different hardware platforms. Once the *decomposeParDict* have been correctly setup, the command *decomposePar* must be run on the case.

```
$ decomposePar . case
```

If the mesh is discretised into four sub-domains four folders in the case folder are created. These folders are named *processor0, processor1, processor2* and *processor3*. Each folder then has the boundary conditions matching the portions of the mesh that each processor has to handle.
Once the mesh has been decomposed the solver has to be run parallel. This is done using an option to the *foamJob* command described earlier.
For running reactingFoam in parallel the following command is used:

```
$ foamJob -p 4 reactingFoam . case
```

Using the *-p* options tells the *foamJob* command that this is a parallel case and it initialises *reacingFoam* on the amount of processors specified in the *decomposeParDict* file.
Once the calculations have ended the mesh/output has to be reconstructed, for post processing, using the command

```
$ reconstructPar . case
```

The parallelisation of the case give speed up in calculation time about 2.5 times, compared to calculations on only one processor.

4.4.4 Scheme setup

Since reactingFoam is a transient model some different values have to be set. The solution setup can be seen in table 4.4. When using a transient model in OpenFOAM some additional options has to be set in the *controlDict* file. The most important option that has to be set is the time-step size. OpenFOAM can handle this in two different ways. The first option is to set the time-step manually, but choosing a too large time-step will cause divergence an lead to floating point errors. Instead a different approach can be used by using an adjustable time-step where OpenFOAM automatically sets the time-step based on the Courant number. The Courant number reflects the portion of a cell that a variable will travel in one time-step, equation 4.3.

$$C_r = \frac{\bar{v}\Delta t}{\delta x} \tag{4.3}$$

Where C_r is the Courant number, $\bar{v}$ is the average linear velocity in that cell. δx is the dimension of the cell and Δt is the time-step.
When advection dominates dispersion, a model with a small ($C_r \leq 1$) Courant number will decrease oscillations, improve accuracy and decrease numerical dispersion. Lower Courant number ($0 < C_r \leq 0.5$) means better stability but slower calculation, higher Courant number ($1 \geq C_r \geq 0.5$) is just the opposite, Fluent (2005). A Courant number of 0.2 was used since higher numbers proved to be unstable.

Solver	
Transient	$C_r = 0.2$
Pressure based	
Pressure velocity coupling	PISO
Schemes	
Pressure	limitedLinear
Velocity	limitedLinearV
Turbulence	upwind
Species	cubicCorrected
Energy	cubicCorrected

Table 4.4: *reactingFoam solver setup*

The *limitedLinear* scheme used is a second order bounded scheme and the *limitedLinearV* is a TVD (Total Variation Diminishing) scheme. OpenFOAM (2007) recommends using TVD schemes in swirling flow. The reason for using the upwind scheme for *turbulence* is that higher order schemes proved to be unstable for the turbulence properties k and ε.

4.5 Results

Here the results from OpenFOAM and Fluent are presented. reactingFoam is as mentioned earlier transient whereas the Fluent case has been solved steady-state. react-

ingFoam proves to approach a steady solution and comparison of the two solvers is considered valid. It is interesting to see that the transient model becomes steady and this also means that modelling this case transient is somewhat an unnecessary waste of computational resources.
To get a steady solution using reactingFoam at least 10 seconds has to be simulated. The comparison of the two codes is based on contour plot of the temperature and species involved in the reaction. The contour plots are made by comparing reactingFoam results (left) directly against Fluent results (right).

4.5.1 Fuel-rich conditions

The two cases are compared at two different air to fuel ratios. In the fuel-rich case an excess of fuel should be present after combustion is complete. From figure 4.4(b) this can be hard to see , but as seen in table 4.5 some excess fuel is present at the outlet. The temperatures for the under-stoichiometric should be $\sim$ 2235K based on an analytical solutions for $\Phi = 1.21$. As seen in figure 4.4(a) reactingFoam has a maximum value of 2220K and Fluent has a maximum of 2130K. reactingFoam thus have values closer to the analytical solution. Both reactingFoam and Fluent has similar results for the CO_2 and H_2O production, only CO_2 is shown in figure 4.4(c).

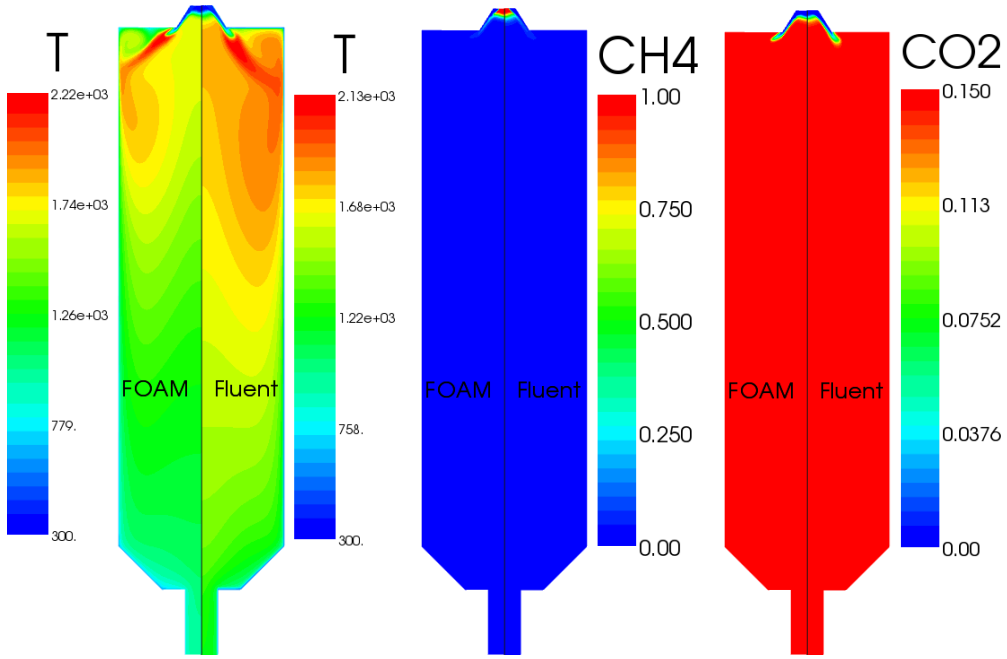


Figure 4.4: *Comparison of reactingFoam and Fluent. (a) temperature contours, (b) CH_4 mass fraction and (c) CO_2 mass fraction.*

A noticeable difference between reactingFoam and Fluent is the temperature field. Fluent has a larger area in the BFR where the temperatures are higher. Changing the schemes to higher orders in Fluent did not give significantly different temperatures. One of the influencing things contributing to this higher downstream temperature could be caused by the walls cooling the flow faster in the OpenFOAM model. The heat-flux from the wall can although not be obtained from reactingFoam so this cannot be verified.The

O_2 consumption in the two models are almost identical as is the CO_2 production, see table 4.5.

Specie	Calculated	Fluent	reactingFoam
CO_2	0.1239	0.1503	0.1499
H_2O	0.0943	0.1231	0.1228
CH_4	0.0675	0.0128	0.0133
N_2	0.7141	0.7138	0.7140

Table 4.5: *Numerical results for the mass fractions after fuel-rich combustion. The numerical values are taken at the outlet.*

The calculated values in table 4.5 are different from the numerical values. The calculated values for CH_4 is ∼5 times larger than the numerical. Experimental verifications, at the outlet, are needed to verify the results.

4.5.2 Fuel-lean conditions

In the fuel-lean case an excess of oxidiser should be present after combustion is complete. The temperatures for the fuel-lean should be $\sim$ 2194K based on an analytical solutions for $\Phi = 0.855$ as seen in figure 4.5(a) reactingFoam has a maximum value of 2220K and Fluent 2200 which both are close to the analytical value. only CO_2 is shown in figure 4.5(c).

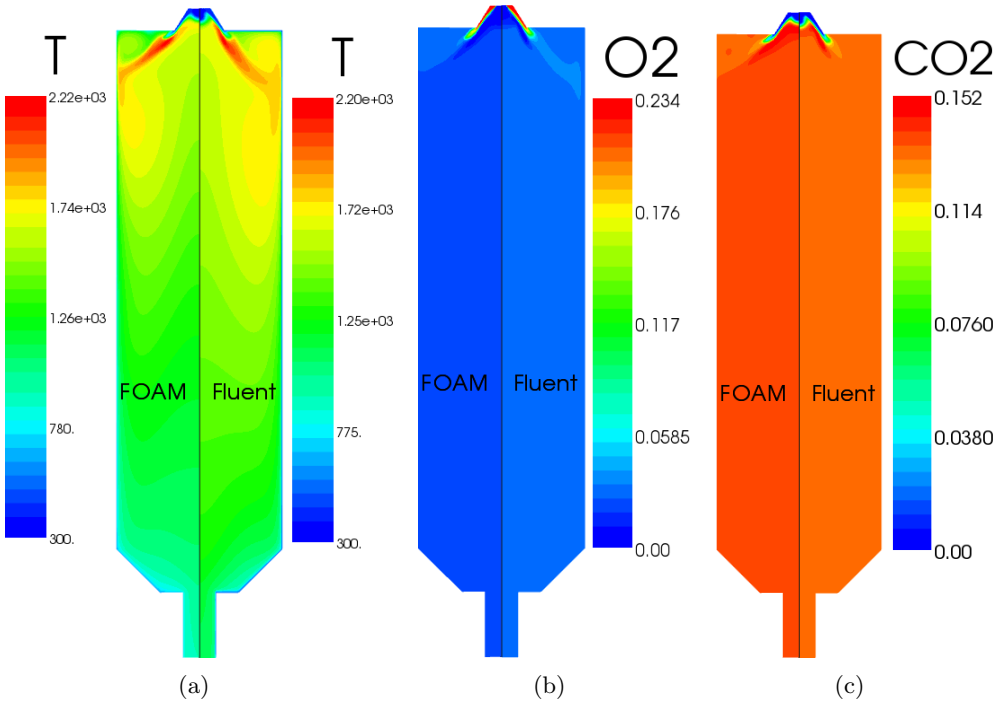


Figure 4.5: *Comparison of reactingFoam and Fluent. (a) shows the temperature contours, (b) O_2 mass fraction and (c) CO_2 mass fraction.*

In the fuel-lean case the temperature profiles inside the BFR are more similar although

some deviation is still present between the two models. reactingFoam has a slightly larger CO_2 and H_2O concentration downstream in the BFR, see figure 4.5(c). The O_2 concentration is on the other hand smaller, figure 4.5(b). Overall the two models are similar in results and experimental verification could be conducted to verify the models validity.

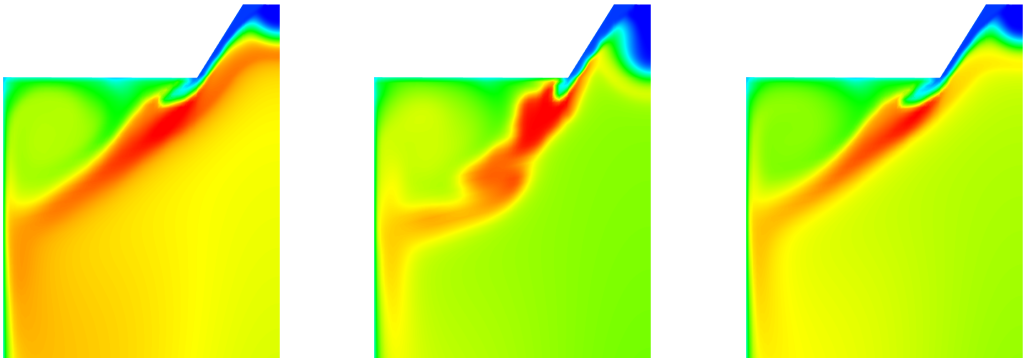
Specie	Calculated	Fluent	reactingFoam
CO_2	0.1297	0.1314	0.1403
H_2O	0.1061	0.1075	0.1148
O_2	0.0334	0.0318	0.0188
N_2	0.7308	0.7292	0.7260

Table 4.6: *Calculated and numerical results for the mass fractions after fuel-lean combustion. The numerical values are taken at the outlet.*

The numerical values obtained for the mass fraction in Fluent are in good agreement with the calculated, see table 4.6. The results from OpenFOAM differs from Fluent and the calculated solution, but are still within acceptable range.

4.6 Scheme discussion

For evaluating the stability of the OpenFOAM solver, the influence of the convection scheme was investigated. Three different schemes on the energy and species were tried. First it was run with only upwind, then a QUICK and lastly cubicCorrected which is a fourth order bounded scheme. The reason for using different schemes, was that after changing the upwind scheme, see figure 4.6(a), to a QUICK scheme, "wiggles" or instability occurred in the temperature profiles, see figure 4.6(b). Changing this to the cubicCorrected scheme cancelled these "wiggles", see figure 4.6(c). It is assumed that the "wiggles" are a function of the QUICK schemes nature, where it has a tendency to produce over-shoots and under-shoots in the results and thus producing the obtained "wiggles" which is also discussed by Versteeg and Malalasekera (2007).



(a) Upwind (b) QUICK (c) cubicCorrected

Figure 4.6: *Difference in temperature field using three different schemes.*

4.7 Summary

The combustion models available in the OpenFOAM packages have briefly been introduced and the reactingFoam PaSR solver was chosen to compare with results from a Fluent EDC model. reactingFoam is a transient model but the results proved to be steady and was compared to the steady-state results obtained from Fluent. The results for both reactingFoam and Fluent are similar, with some slight deviations. The only significant difference between reactingFoam and Fluent is the time needed to get the results. Table 4.7 summarises the differences between the two solvers.

	reactingFoam (transient)	**Fluent (steady state)**
Setup time	$\sim$ 1 hour	$\sim$ 1 hour
Solver time	$\sim$ 7 days	$\sim$ 1 hour
Solver cost	Free	$\sim$100K kr + $\sim$25K kr pr. node

Table 4.7: *Comparison of setting up each solver and the time needed to obtain a solution*

reactingFoam is free and can be run parallel for free, but Fluent has steady state models which can get a solution faster. So the free nature off OpenFOAM comes at a price, calculation time. If a steady state EDC model where available in OpenFOAM this would be the favourable choice since both solvers produce similar results.


AALBORG UNIVERSITY
DENMARK

5

Steady state combustion model

5.1 Introduction

This chapter will give an introduction to the development of models in OpenFOAM. Three models are developed in OpenFOAM the mixture fraction model, Eddy Break-Up and Eddy Dissipation Concept. Not enough time where available to implement thermodynamics in the models. This is mainly because the object oriented programming nature in OpenFOAM which have a steep learning curve. The purpose of object oriented programming is to make code-reuse easier and avoid inefficient code. Most of the physical models in CFD are modelled using partial differential equations, which makes parts of the solver repeatable.
The disadvantage with object oriented programming is that it can be difficult to get an overview of existing code, unless the user is an experienced programmer. Presumably most CFD software is used by Engineers which have little or no experience in programming.
The big advantage programming object oriented is the reuse of functions in the code. On the other hand it makes it harder to read, since one has to look into anything between 2-50 separate files in different locations to locate the implementation or equations. To illustrate the file dependency for a solver, the file structure of the steady state incompressible solver *simpleFoam* is shown in appendix G.

The programming language used in OpenFOAM is standard C++ and it widely incorporates the use of classes, templates and pointers in the C++ language. For a description of classes, templates and pointers see Soulie (2008), Hubbard (2000). When OpenFOAM is coded using classes these functions can be called in the solver but the actual calculations are handled in another file containing the called class function. Equation

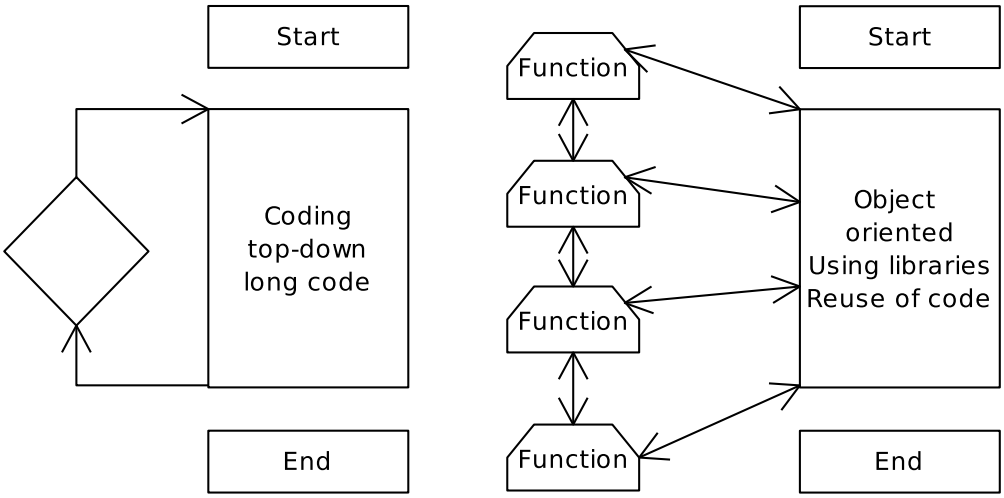


Figure 5.1: *Difference between object oriented and top-down programming*

5.1 and the subsequent code is example of how the equations is written in OpenFOAM syntax.

$$\frac{\partial \rho U}{\partial t} + \nabla \bullet \rho U U - \nabla \bullet \mu \nabla U = -\nabla p \tag{5.1}$$

```
solve
(
    fvm::ddt(rho, U)
  + fvm::div(phi, U)
  - fvm::laplacian(mu, U)
    ==
  - fvc::grad(p)
);
```

Here *fvm* (finite volume matrix) is the class and *::* calls the class *fvm* with the input *ddt(rho, U)*. The *fvm* class solves a partial differential equation with a matrix input, where the *fvc* (finite volume calculus) is a scalar array. Time derivative is called with the function "ddt()" and is a member og the *fvm* class as well as "div()" (divergence operator) and "laplacian" (laplacian operator).
The C++ standard supports overloading of functions, which means that the same class can have multiple inputs and depending on the input and variables the appropriate equations are used.

Starting to understand the code and creating new code can be a time consuming task, so a good understanding of the C++ language is highly recommended. The specific OpenFOAM syntax, classes and templates can be even more time consuming to grasp since the official user guide and programmers guide is more a guide on existing libraries/solvers and how they are used on specific tutorial cases. Little documentation on the syntax is available in the official documentation and significant reverse engineering of the code is needed to develop a new solver.

5.2 Introduction to combustion modeling

Combustion is one of the most important processes in energy engineering, which involves turbulent fluid flow, heat transfer, chemical reaction and thermal radiation. Computational Fluid Dynamics is currently on of the only alternative to carrying out expensive experiments in scaled or full scale models.

The objective of a combustion model is to relate the turbulent fluid flow to chemical reactions. Equation 5.2 is the transport equation for any scalar (ϕ). The number of transport equations varies with different combustion models depending on the approach.

$$\underbrace{\frac{\partial \rho\phi}{\partial t}}_{\text{Unsteady term}} + \underbrace{\nabla \cdot (\rho \vec{u} \phi)}_{\text{Convection term}} = \underbrace{\nabla^2(\Gamma\phi)}_{\text{Diffusion term}} + \underbrace{S_\phi}_{\text{Source term}} \tag{5.2}$$

Combustion models are widely available in commercial CFD packages and are generally divided into *equilibrium chemistry* and *detailed chemistry* as depicted in figure 5.2.
In the field of coal and biomass combustion, the eddy-dissipation model along with a probability density function (PDF) is often preferred for its simplicity Fluent (2005).

	Premixed flames	Diffusion flames	Partially premixed flames
Equilibrium chemistry	Zimont model (Reaction progress variable)	Mixture fraction model	Zimont-/ Mixture fraction approach
	Eddy Dissipation model (Magnussen and Hjertager)		
Detailed chemistry		Flamelet model	
	Finite Rate model Eddy-Dissipation-Concept model PDF Transport model		

Figure 5.2: *Overview of available combustion models in Fluent.*

5.3 Arrhenius kinetic model

The Arrhenius kinetic model is mainly used in laminar combustion modelling Turns (2006), but can be used in simulations where slow chemical reactions as carbon monoxide and nitrous oxide needs to be taken into account.

In CFD the Arrhenius model is not commonly used as primary model, but rather a sub model for simulating slow reactions. Because of shortness of time the Arrhenius kinetic model has not been implemented in the Eddy break-up model or the Eddy dissipation model.

5.4 Mixture fraction theory

The mixture fraction approach is among the simplest formulations of a combusting flow. The model assumes “mixed is burnt” and does not account for dissociation

The basis of this, non-premixed combustion model, approach is that under a certain set of simplifying assumptions, the instantaneous thermochemical state of the fluid can be related to a conserved scalar known as the mixture fraction, f_{mix}.

By assuming equal diffusion rates for each specie and steady state conditions, oxidiser, fuel and products can be derived from one conserved scalar, the mixture fraction. Equation 5.3 show the equation for the mixture fraction. The mass diffusion is calculated as the ration between the turbulent kinematic viscosity and the turbulent Schmidt number. According to Fluent (2005) the turbulent Schmidt number (Sc_T) can be assumed to be a constant value of 0.7.

$$\frac{\partial \rho f_{mix}}{\partial t} + \nabla \cdot (\rho \vec{u} f_{mix}) = \nabla^2 \left(\frac{\nu_T}{Sc_T} f_{mix} \right) \tag{5.3}$$

The conserved scalar concept greatly simplifies the solution of reacting flow problems, and can be used for initializing other combustion models, with stability or convergence problems.

Equation 5.4 is first simplified as oxidiser, fuel and products as shown in equation 5.5. From the reaction scheme the mass fractions of the components of the products can later be extracted.

$$CH_4 + s(O_2 + 3.76N_2) \rightarrow CO_2 + (2 \times s)H_2O + 3.76 \times sN_2 \tag{5.4}$$

$$Y_{fu} + s \cdot Y_{ox} \rightarrow (1 + s)Y_{pr} \tag{5.5}$$

The definition of the mixture fraction is shown in equation 5.6.

$$f_{\mathbf{mix}} = Y_{fu} + \frac{Y_{pr}}{1 + s} \tag{5.6}$$

The implementation of mixture fraction in OpenFOAM has the advantage of no source terms (being a passive scalar), since this makes the simulation convergence both stable and faster.

5.4.1 Mixture fraction results

In this section the results from the mixture fraction model are presented and discussed. Since the mixture fraction approach is the most simple and primitive combustion model its performance will be evaluated relative to Eddy break-up model and Eddy dissipation model.
The scalar property "f_{mix}" is plotted along with the mass fraction of fuel in figure 5.3. The correlation used is as follows:

$$f_{stoich} \leq f_{mix} < 1.0: \qquad Y_{ox} = 0.0 \qquad Y_{fu} = \frac{f_{mix} - f_{stoich}}{1 - f_{stoich}} \cdot Y_{fu,1} \tag{5.7}$$

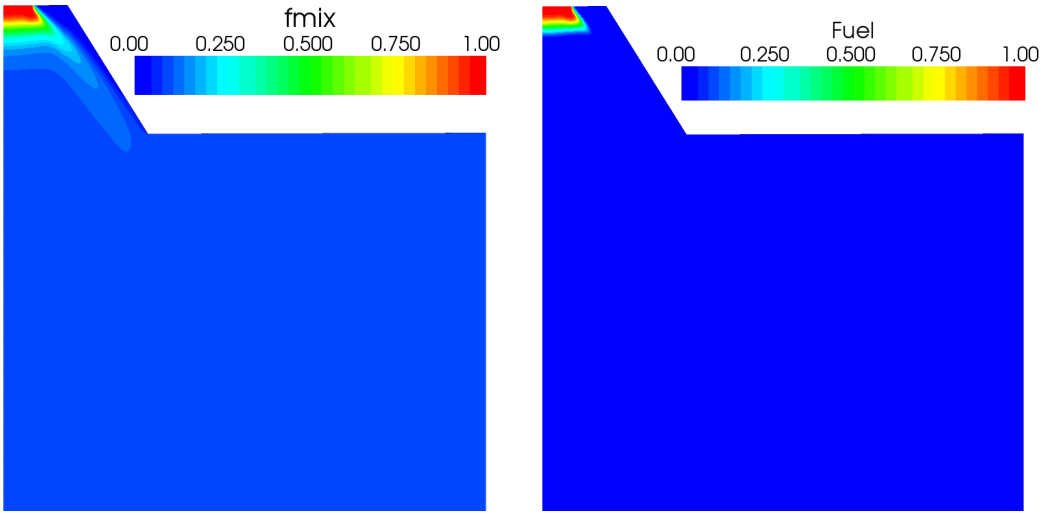
$$0 \leq f_{mix} < f_{stoich}: \qquad Y_{fu} = 0.0 \qquad Y_{ox} = \frac{f_{stoich} - f_{mix}}{f_{stoich}} \tag{5.8}$$

BC	fmix	U
Center inlet	1.0	2.339441
Middle inlet	1.0	2.382512
Secondary inlet	0.0	Vector
Internal field	0.0	0.0
Walls	zero gradient	0.0

Table 5.1: *Boundary condition specification for the mixture fraction model*

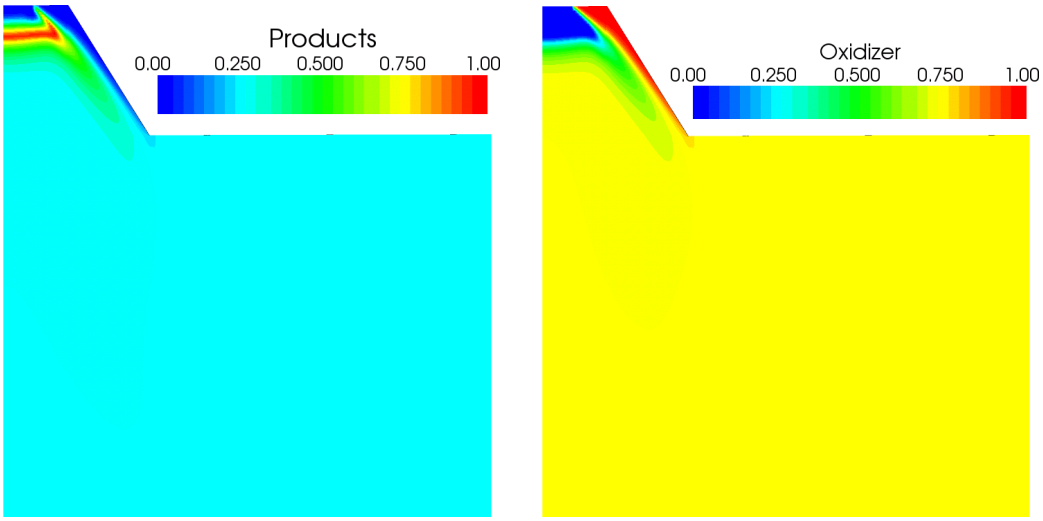
where f_{stoich} is the stoichiometric ratio, Y_{ox} is the mass fraction of oxidizer, Y_{fu} is the mass fraction of fuel and $Y_{fu,1}$ is the mass fraction of fuel at inlet.

The flamesheet shown by the products in figure 5.4 takes a form as a plane front, which is caused by the recirculations zone at the centreline of the burner.
The methane penetration in the flow is lower than expected, but since the reaction is mixing controlled and the burner has a preswirler the mixing is expected to be fast.



(a) Mixture fraction (b) Fuel mass fraction

Figure 5.3: *Mixture fraction and fuel distribution near the inlet of the burner.*



(a) Products mass fraction (b) Oxidizer mass fraction

Figure 5.4: *Mass fraction of products and oxidizer near the inlet of the burner.*

5.5 Eddy Break-Up model

The Eddy Break-Up model is a simple and efficient model used in combustion calculations. The intrinsic idea behind the Eddy Break-Up model is that the rate of combustion is determined by the rate at which parcels of unburned gas are broken down to into smaller ones. This in such a way that there is sufficient interfacial area between the unburned mixture and hot gasses to permit reaction. The implication of this is that chemical kinetics has no role in determining the burning rate, but rather turbulent mixing rate completely controls combustion. Since the model does not involve solving expensive Arrhenius chemical kinetic calculations, and gives reasonable results for highly turbulent combustion. The mixing controlled combustion can be expressed as an infinite Damköhler number, which expresses the ratio between the speed of chemical reactions compared to the time scale of the flow.

The Eddy Break-Up model is implemented in OpenFOAM on the basis on its theoretical describtion in Versteeg and Malalasekera (2007) and is identical with the Eddy dissipation model in Fluent. According to Versteeg and Malalasekera (2007) the Eddy Break-Up model was originally propose by Spalding in 1971, however other sources suggests that Magnussen and Hjertager also contributed to the development, Wiki (2008).

The chemical model used in the present work only accounts for the global reaction for methane-air combustion, and neglects intermediate species and dissociation. The reaction scheme is presented in equation 5.9.

$$CH_4 + 2 \times (O_2 + 3.76N_2) \rightarrow CO_2 + 2 \times H_2O + 7.52 \times N_2 \tag{5.9}$$

The chemical reaction scheme is converted to mass basis, for easier implementation in the CFD code, as shown in equation 5.10.

$$1 \text{ kg } CH_4 + 4 \text{ kg } O_2 \rightarrow 2.75 \text{ kg } CO_2 + 2.25 \text{ kg } H_2O \tag{5.10}$$

The limiting reaction rates are listed below, the concept is to have the lowest reaction rate control the chemical mechanism.

$$\tilde{\dot{\omega}}_{fu} = - C_R \cdot \overline{\rho} \cdot \tilde{Y}_{fu} \cdot \frac{\varepsilon}{k} \tag{5.11}$$

$$\tilde{\dot{\omega}}_{ox} = - C_R \cdot \overline{\rho} \cdot \frac{\tilde{Y}_{ox}}{s} \cdot \frac{\varepsilon}{k} \tag{5.12}$$

$$\tilde{\dot{\omega}}_{pr} = - C'_R \cdot \overline{\rho} \cdot \frac{\tilde{Y}_{pr}}{1+s} \cdot \frac{\varepsilon}{k} \tag{5.13}$$

Where C_R, C'_R are model constants with the value of 1.0 and 0.5 respectively, according to Versteeg and Malalasekera (2007). s is the stoichiometric ratio, $\overline{\rho}$ is the mean density, ε is the turbulent dissipation rate, k is the turbulent kinetic energy and ω_i is the reaction rate on mass basis.

$$\tilde{\dot{\omega}}_{fu} = -\overline{\rho}\frac{\varepsilon}{k}\min\left[C_R \cdot \tilde{Y}_{fu}, C_R \cdot \frac{\tilde{Y}_{ox}}{s}, C'_R\frac{\tilde{Y}_{pr}}{1+s}\right] \tag{5.14}$$

The implementation of the Eddy Break-Up model in OpenFOAM requires (N-1) transport equations for the combustion model and a reaction code to calculate source terms for the transport equations. According to Fluent (2005) numerical accuracy is improved by choosing Nitrogen as the non-transport specie, since Nitrogen has the highest mass fraction. The implementation of the EBU model is further described in appendix F.

If the reaction rate of the products is taken into account, it is important to initialize the field with non-zero values to avoid zero being the lowest reaction rate (will produce no products).

5.5.1 Eddy Break-Up results

In this section Eddy break-up results from Fluent and OpenFOAM are compared. The basis of the case setup in Fluent is identical to the one applied in chapter 4, only difference is that the energy equation is disabled. The purpose of this section is validation of the steady state combustion code developed in the present work against commercial software.

As seen in figure 5.5 the flow recirculation zone is predicted stronger by OpenFOAM than by Fluent. However, the flow field comparison in chapter 3 showed better agreement. The reason for the change in flow predicted by Fluent is not known since all boundary conditions are the same.

When comparing combustion models the best variable to compare is usually the temperature, since determination of temperature distribution often is the purpose of a combustion model. The variable selected for final comparison is the products, as the time to fully implement the energy equation in OpenFOAM was not available. The distribution of products in a burner can implicit be related to the temperature, since forming of products is an exoterm process. The difference seen in the distribution of

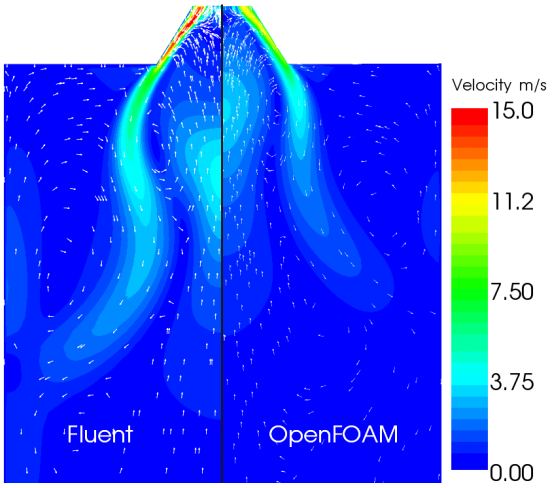


Figure 5.5: *Comparison of between Fluent and OpenFOAM flow field. Fluent with disabled energy equation.*

methane directly influences the products as seen in figure 5.7 and figure 5.8.

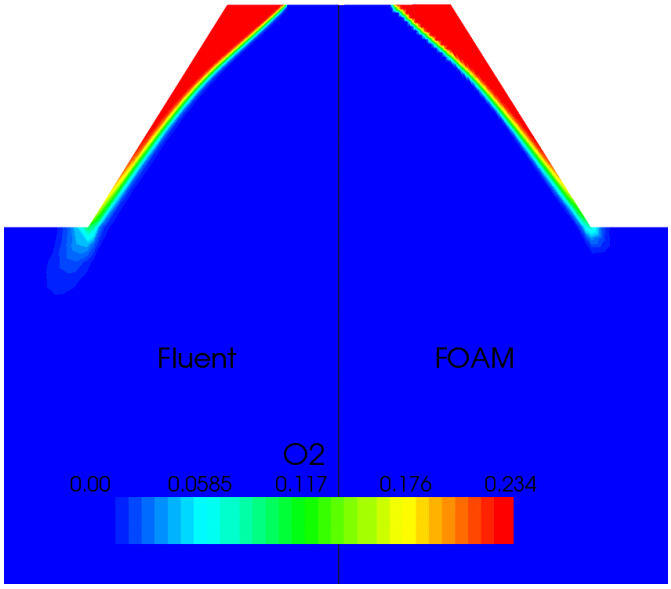


Figure 5.6: *Comparison of Eddy Break-Up model with the mass fraction for oxygen.*

The reaction in OpenFOAM is pushed towards the inlet by the upstream recirculation at the center of the burner. The tendency of both predictions are very similar, but with small differences at the outer edge of the inlet. When accounting for the differences in the flow field, the agreement between the OpenFOAM and Fluent shows reasonable accuracy for the distribution of products.

The combustion code written in the present work does not include source term linearisation, which influences the convergence time and stability. However, the QUICK

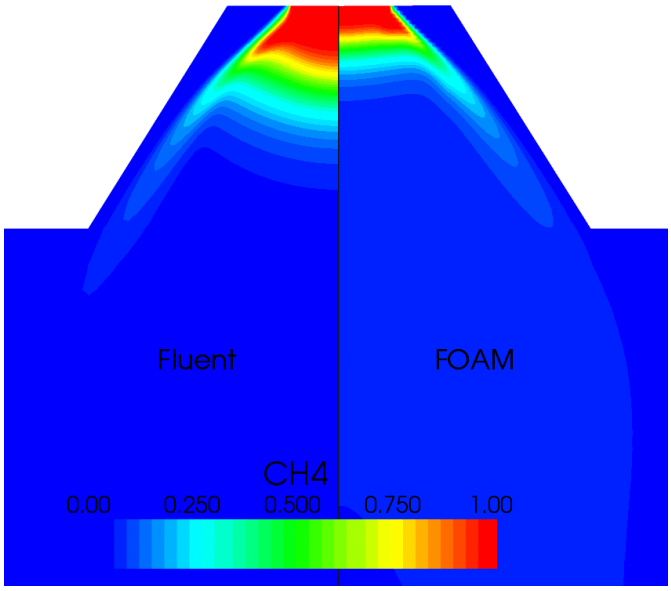


Figure 5.7: *Comparison of Eddy Break-Up model with the mass fraction for methane.*

convection scheme along with a under relaxation factor of 0.7, on the species, made it possible to control the convergence process and give a stable solution. It should be kept in mind that OpenFOAM is just a toolbox, the user developed code should by itself have limiting effects that ensure valid results, *e.g.* limiting functions for mass fractions and reaction rate.

The results for both the software simulations and the simple continuity calculation are listed in table 5.2. The Eddy Break-Up model developed in the present work deviates from Fluent approximately 6% for mass fractions of CO_2, H_2O and N_2. However the predicted mass fraction at outlet for CH_4 deviates 131%. The reason for the large deviation is ascribed to numerical instability, meaning that the code still needs improvement to make it more stable and easier to converge.

Specie	**Calculated**	**Fluent**	**reactingFoam**	**OpenFOAM EBU**
CO_2	0.1239	0.1503	0.1499	0.1590
H_2O	0.0943	0.1231	0.1228	0.1301
CH_4	0.0676	0.0128	0.0133	0.0296
N_2	0.7141	0.7138	0.7140	0.6813

Table 5.2: *Mass fractions after fuel-rich combustion. The values are extracted at the outlet.*

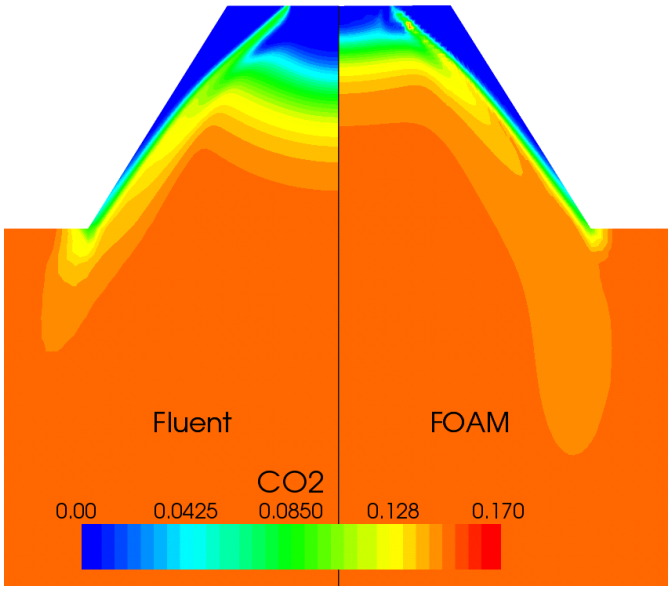


Figure 5.8: *Comparison of Eddy Break-Up model with the mass fraction for carbondioxide.*

5.6 Eddy Dissipation model

The Eddy Dissipation model is a revised version of the Eddy Break-Up model proposed by Ertesvag and Magnussen (2000). The EDC model attempts to in cooperate the significance of fine structures in a turbulent reacting flow in which combustion chemistry is important.

The fine structure approach of the Eddy Dissipation model is implemented in OpenFOAM through the present work. However, Fluent also includes Arrhenius kinetic model, where the EDC model in the present work models the combustion as mixing limited. The Eddy Dissipation model is implemented in OpenFOAM on the basis on its theoretical desciption in Versteeg and Malalasekera (2007).

In the EDC model it is assumed that the mass fraction occupied by the fine structures can be expressed as equation 5.15.

$$\gamma^* = 4.6 \left(\frac{\nu \varepsilon}{k^2} \right)^{1/2} \tag{5.15}$$

Where k and ε are the turbulent kinetic energy and dissipation, and 4.6 is a model constant. Not all the fine structures will be sufficiently heated to react. The fraction of the fine structures which react is assumed proportional to the ratio between local concentration of reacted fuel and the total fuel concentration. The reacting fraction of the fine structures is defined in equation 5.16.

$$\chi = \frac{\tilde{Y}_{pr}/(1+st)}{Y_{\min} + \tilde{Y}_{pr}/(1+st)} \tag{5.16}$$

where, $Y_{\min} = \min \left[\tilde{Y}_{fu}, \tilde{Y}_{ox}/st \right]$. The reaction rate on mass basis can finally be determined from equation 5.17.

$$\tilde{\dot{\omega}}_{fu} = -\overline{\rho}\frac{\varepsilon}{k}C_{EDC}\ \min\left[\tilde{Y}_{fu}, \frac{\tilde{Y}_{ox}}{s}\right] \cdot \left(\frac{\chi}{1-\gamma^{*}\chi}\right) \tag{5.17}$$

Where C_{EDC} is a model constant with a recommended value of 11.2 according to Versteeg and Malalasekera (2007). A more detailed description of the Eddy Dissipation model is available in Magnussen (1981). When applying the EDC model it is also important to initialize the field with non-zero values for products to avoid the zero reaction rate also mentioned for the Eddy Break-Up model.

5.6.1 Eddy Dissipation results

In this section the Eddy Dissipation model and the Eddy Break-Up model are compared. Fluent has implemented the Eddy Dissipation model differently than the description in Versteeg and Malalasekera (2007), however with the same concept of small structures. Since it is un-common to use the EDC in Fluent without the Arrhenius kinetic model enabled, the Eddy Break-Up model and the Eddy dissipation model as described by Versteeg and Malalasekera (2007) will be compared.

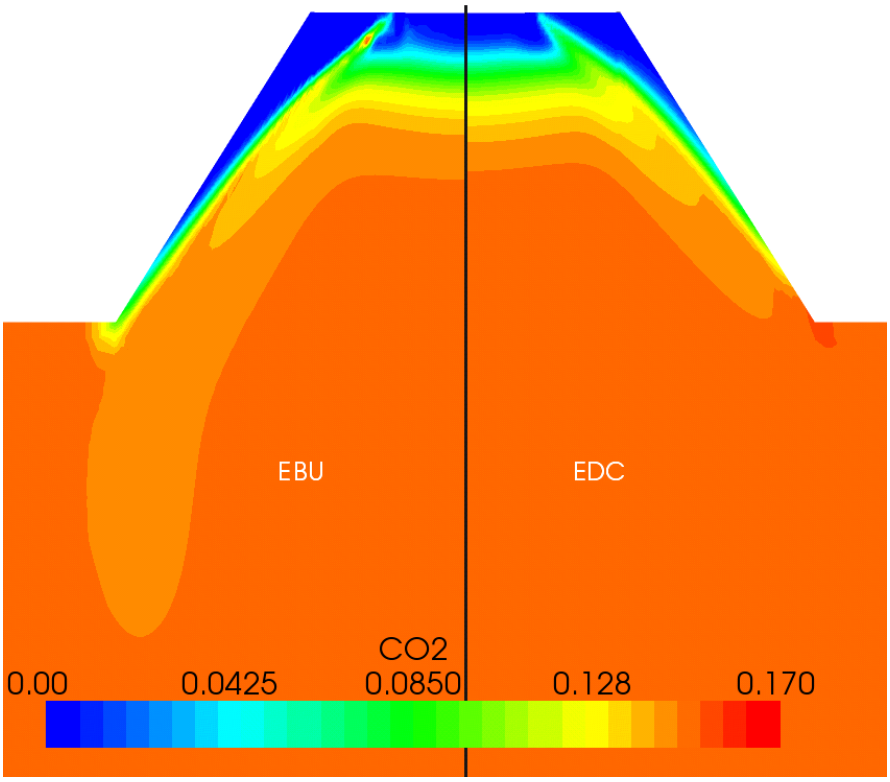
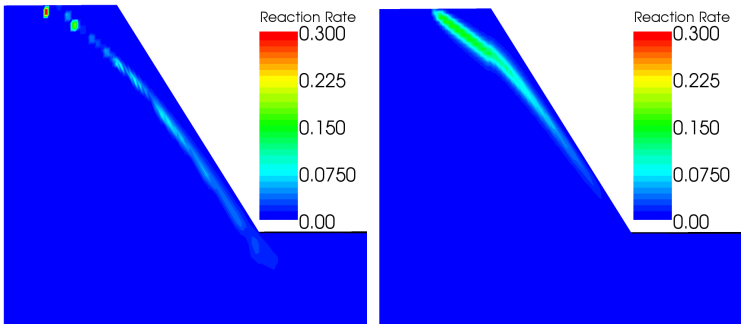


Figure 5.9: *Comparison of Eddy Break-Up model and Eddy Dissipation model with respect to distribution of carbon dioxide.*

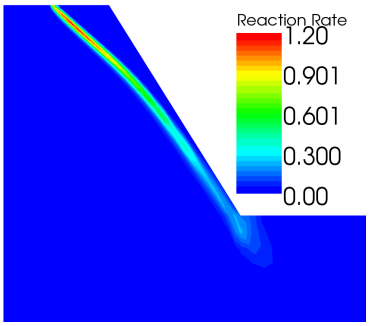
The difference between the Eddy Break-Up model and the Eddy Dissipation model in the present work is depicted in figure 5.9. The CO_2 distribution is almost identical, however the EBU model predicts a lower concentration of CO_2 in the high velocity region in the burner zone. The theoretical basis of the two models are similar, thus the

result is expectedly similar.

For an overview of the models both in Fluent and OpenFOAM the turbulent reaction rate is depicted in figure 5.10. The reaction rate calculated by Fluent shows a well-defined region with an expected decreasing value downstream.



(a) EBU in OpenFOAM (b) EDC in OpenFOAM



(c) EBU in Fluent

Figure 5.10: *Comparison of the reaction rate for both OpenFOAM models and the EBU model in Fluent (Eddy-Dissipation).*

The comparison in figure 5.10 show similar tendency for the reaction rate. The reaction rate predicted by Fluent is approximately four times the ones predicted by the EBU and EDC. The values in the EBU/EDC model to weight the reactants against the products as showed in equation 5.11 through 5.13 by the constants (C_R and C'_R) can be fine tuned for better agreement with Fluent.

The overall agreement between the models implemented in OpenFOAM and Fluent, show that the models in OpenFOAM still need some work before they can compete with commercial software. It will be necessary to implement the thermodynamic coupling between flow and chemistry before experimental validation is possible.

5.7 Numerical stabilisation

Stability in numerical simulations is a common concern, when simulating physical phenomena. The combination of large time steps and large gradients has a destabilising effect on the convergence in CFD calculations.

The combustion source terms have a dominant influence on the solution of species and energy equation. In the present work some stability problems with the implementation of the energy equation were not solved. According to Fluent (2005) it is important to limit and control the source terms carefully because of their non-linearity.

Combustion tends to drive the concentration of reactants towards the lower limit and product concentrations towards the upper limit. The mass fraction of the participating species needs to be bounded between 0 and 1 to ensure stability and convergence. The combustion source terms may need to be moderated to maintain physically realistic mass fractions.

Information regarding linearisation of combustion source terms was acquired late in the present work, so there was no time for its implementation, which could have made the combustion models more stable. However the instability was avoided by disabling the chemistry until the flow and species were converged. The procedure for its implementation is described shortly according to ANSYS (2006).

$$\mathrm{S}_i = \left(\frac{\omega_i - |\omega_i|}{2Y_i^*}\right) + \left(\frac{\omega_i + |\omega_i|}{2(1 - Y_i^{**})}\right)(1 - Y_i) \tag{5.18}$$

Where $Y_i^* = \max(\delta, Y_i)$ and $Y_i^{**} = \max(\delta, 1 - Y_i)$

$$\text{IF} Y_i \geq (1 - \delta) \qquad \mathrm{S}_i = Y_i \tag{5.19}$$

$$\text{ELSE} \qquad \mathrm{S}_i = \omega_i \frac{1 - Y_i}{\delta} \tag{5.20}$$

where Y_i is mass fraction, ω_i is the reaction rate, δ is machine epsilon (small number). Thus as the products increases towards 1, the source decreases towards 0.

5.8 Summary

In this chapter the development environment in OpenFOAM is briefly described along with the structure of the libraries. The steady state combustion models, which were considered favourable to implement in OpenFOAM are described.

The combustion models implemented in OpenFOAM in the present work showed varying agreement with the commercial solver, Fluent. The standard Eddy Break-Up showed fine agreement between the model in Fluent and in OpenFOAM.

Developing new models in OpenFOAM is not only expressing the differential equations, but also making limiter functions and bounding variables. In the present work it was discovered that the implicit formulation of Nitrogen along with the upwind divergence scheme had a significant effect on the stability of the combustion model.

The drawback with OpenFOAM at present state is the lack of documentation for developing new models. However, as the OpenFOAM community grows the access to help and discussion forums increase. The present work makes an good staring point for new developers, because it describes basic procedures for programming in OpenFOAM.


AALBORG UNIVERSITY
DENMARK

6 Conclusion

6.1 Primary conclusion

The free open-source CFD toolbox OpenFOAM has been investigated in this project. The OpenFOAM software was considered because of increasing license fees and dropping hardware cost. To evaluate the performance of the OpenFOAM solver it has been compared to results from Fluent. The comparison was based on a burner flow reactor which has been thoroughly investigated using experiments and CFD on particle combustion. Since OpenFOAM has no particle combustion model at present time, comparison is performed using cold-flow and gas-combustion models.

The OpenFOAM toolbox has been described and the basic simulation setup is presented. The description of the simulation-setup serves as an overview for using OpenFOAM.

The cold-flow comparison is performed using an incompressible turbulence model on similar meshes. OpenFOAM handles axi-symmetric simulation differently from Fluent and has to be setup in a specific way, which is described in detail. The cold-flow comparison gave almost identical results for both OpenFOAM and Fluent. The small difference in the results is ascribed the mesh being slightly different and the interpolation of cell values to point values in post-processing. Overall both codes perform equally in this type of flow problems. When the OpenFOAM syntax is understood, using a steady flow solver in OpenFOAM was found to be as stable and straightforward to setup as using Fluent.

The gas combustion model were compared using a transient model in OpenFOAM, since no steady state model exists, and a steady state model in Fluent. Two cases were modelled, a fuel-lean and a fuel-rich. In the fuel-lean case both codes were similar in results and have mass fraction values close to computed values. In the fuel-rich case the high temperature region extends further downstream in the burner using Fluent. Overall both codes have reasonable agreement, although the results with the transient model in OpenFOAM take longer to obtain. To get reactingFoam to be stable proved to be somewhat of a troublesome task and a good understanding of schemes and initial boundary conditions was needed.

To reduce calculation time, steady state gas-combustion models in OpenFOAM were developed. Three different models have been developed for comparison with Fluent.

A Mixture Fraction model, an Eddy Break-Up model and an Eddy Dissipation Model. The models handle chemical reactions without coupling to the energy.

The combustion models implemented in OpenFOAM showed varying agreement with Fluent. The standard Eddy Break-Up model showed the best agreement for the distribution of products in the burner.

It was discovered that the implicit formulation of Nitrogen along with the upwind divergence scheme had a significant effect on the stability for the combustion model. Numerical stabilization in OpenFOAM is an important part of developing new models.

Overall the OpenFOAM toolbox is a good solid code, which still need some physical models. The time needed to use the existing models can be slightly larger than when starting with a commercial code. The stability of some of the more "advanced" models which include more than flow calculation can be unstable. Good initial guesses and lower order divergence schemes must be used for these models. The stability and calculation time for the steady state models in OpenFOAM are considered equally to Fluent.

Developing with OpenFOAM was another issue addressed in this project and significant time was needed to learn the programming language and syntax of OpenFOAM. The report serves as a good starting point for new-comers to the OpenFOAM toolbox. Although tutorial guides are available, some of the topics addressed in this report, are not covered in the user guides.

OpenFOAM is also a good way for companies and individuals, who does not have the finance for commercial programs, to get started with CFD. OpenFOAM does not contain the same advanced models as some commercial counterparts, but provides a broad package of tools and gives the users opportunity to extend or improve the product.

6.2 Perspective

To replace the commercial CFD software, extensive development is still needed to build the missing models. The task of creating new models is demanding, and should be carried out in cooperation between experienced programmers and engineers. When the missing models are developed, OpenFOAM is considered a good and likely replacement for expensive commercial software. Table 6.1 lists the authors opinion regarding areas it would be beneficial to use OpenFOAM at present time.

Field of work	**Beneficiary**		
Industrial	Proprietary work	Quality assurance	Parallel use
Consultants	Customisation	Asses capability	Use in perpetuity
Academic	Research	Teaching	Limited resources
Overall	Can be developed	Source access	No licence fee

Table 6.1: *Benefits of OpenFOAM depending on field of work.*

The work needed for a complete steady state combustion model in OpenFOAM is as follows:

- Energy equation is written, but needs to be activated.
- The thermodynamic tables in OpenFOAM needs to be implemented, to calculate the correct mixture properties as enthalpy, density and viscosity.

The main concern is to couple existing libraries in OpenFOAM with the user written code.

6.3 Future work

In future work it would be of great interest to develop the missing physical models described in chapter 1. It is estimated that it will take an experienced user from anything between 3 months to a year to implement the missing models. By experienced user, it mean users with similar knowledge as the authors have described in this report.

Moreover OpenFOAM would be a good way to teach students the concept of Computational Fluid Dynamics, since OpenFOAM is free and is not "point and click" CFD. The students will have to get knowledge about initial guesses and how the divergence schemes to get a stable model. Also the students can be taught to develop with OpenFOAM as it has all the basic equations solvers and libraries for students to create their own code with focus on the models they want to implement.


AALBORG UNIVERSITY
DENMARK

Bibliography

Anderson, J. D. *Computational Fluid Dynamics.* McGraw-Hill, 1995.

ANSYS. *ANSYS CFX Theory Guide*, 11b ed., December 2006.

Babushok, V. I. and Dakdancha, A. N. Global kinetic parameters for high-temperature gas-phase reactions. *Translated from Fizika Goreniya i Vzryva 29* (1993), pp. 48–80.

BELL, J. B., DAY, M. S. and GRCAR, J. F. Numerical simulation of premixed turbulent methane combustion. *Proceedings of the Combustion Institute 29* (2002), pp. 1987–1993.

Bhave, A. and Kraft, M. Partially stirred reactor model: Analytical solutions and numerical convergence study of a pdf/monte carlo method. *SIAM J. Sci. Comput. 25*, 5 (2004), pp. 1798–1823.

Brink, A., Mueller, C., Kilpinen, P. and Hupa, M. Possibilities and limitations of the eddy break-up model. *Combustion and flame 123* (2000), pp. 275–279.

Buckmaster, J. D. *The mathematics of combustion.* Society for Industrial and Applied Mathematics, 1985. ISBN 85-50339.

Casey, M. and Wintergerste, T. *Best practice guidelines*, 1 ed. ERCOFTAC, 2000.

Cengel, Y. A. *Heat Transfer: A Practical Approach*, 2 ed. McGraw-Hill, 2003. ISBN 0-07-245893-3.

Chomiak, J. and Karlsson, A. Flame liftoff in diesel sprays. *Twenty-Sixth Symposium (International) on Combustion*, 26 (1996), pp. 2557–2564.

Ertesvåg, I. S. and Magnussen, B. F. The eddy dissipation turbulence energy cascade model. *Combust. Sci. and Tech. 159* (December 1999), pp. 213–235.

Fluent. *Fluent 6.2 Users Guide.* Fluent Inc., January 2005.

Fox, R. O. *Computational Models for Turbulent Reacting Flows.* Cambridge University Press, 2003. ISBN 0-521-65907-8.

Hubbard, J. R. *Programming with C++*, second ed. McGraw Hill, 2000. ISBN 0-07-135346-1.

Hvid, S. L. Determination of BFR SA inlet swirling BC. Tech. rep., Dong Energy, November 2006.

Lee, J. H. and Trimm, D. L. Catalytic combustion of methane. *Fuel Processing Technology 42* (1995), pp. 339–359.

Libby, P. A. and Williams, F. A. *Turbulent Reacting Flows*. Academic Press, 1994. ISBN 0-12-447945-6.

Magnussen, B. F. On the structure of turbulence and a generalized eddy dissipation concept for chemical reaction in turbulent flow. *19th AIAA Aerospace Science Meeting* (1981).

Mech, G. Nasa polynomials, April 2008. Available at `http://www.me.berkeley.edu/gri_mech/data/nasa_plnm.html`.

Muff, M. V. CFD modeling of co-firing of straw and coal.

Munson, B. R., Young, D. F. and Qkiishi, T. H. *Fundamentals of fluid mechanics*, 4 ed. John Wiley & Sons, Inc., 2002. ISBN 0-471-44250-X.

Nazeer, W. A., Jackson, R. E., Peart, J. A. and Tree, D. R. Detailed measurements in a pulverized coal flame with natural gas reburning. *FUEL 78* (November 1999), pp. 689–699.

Norton, D. G. and Vlachos, D. G. Combustion characteristics and fame stability at the microscale: a cfd study of premixed methane/air mixtures. *Chemical Engineering Science 58* (December 2002), pp. 4871–4882.

Olesen, M. OpenFOAM and STAR-CD, integration, interoperability and Symbiosis. In *Dansis - New trends in CFD* (October 2007).

OpenCFD, L. Open-source CFD for commercial use. In *Dansis - New trends in CFD* (October 2007).

OpenFOAM. Openfoam homepage, March 2008. Available at `http://www.opencfd.co.uk/openfoam/`.

OpenFOAM. *OpenFoam user guide*, 1.4.1 ed., 2007.

Schlichting, H. and Gersten, K. *Boundary-Layer Theory*, 8 ed. Springer, 2000.

Smoot, L. D. and Smith, P. J. *Coal Combustion And Gasification*. Plenum Press, 1985. ISBN 0-306-41750-2.

Soulie, J. Cplusplus homepage, March 2008. Available at `http://www.cplusplus.com/doc/tutorial/`.

Thiessen, S., Karim, G. and Seyedeyn-Azad, F. Constant volume autoignition of premixed methane-carbon dioxide mixtures. *International Journal of Green Energy 4* (2007), pp. 535–547.

Tree, D. R. Temperature, velocity and species profile measurements for reburning in a pulverized, entrained flow, coal combustor. *Grant Number DE-FG22-95PC95223* (1999).

Turns, S. R. *An Introduction to Combustion*, second ed. McGraw Hill, 2006. ISBN 978-007-126072-5.

Versteeg, H. K. and Malalasekera, W. *An introduction to computational fluid dynamics, The finite volume method*, second ed. Pearson Education Limited, 2007. ISBN 978-0-13-127498-1.

Weller, H. G., Tabor, G., Gosman, A. D. and Fureby, C. Application of a flame-wrinkling les combustion model to a turbulent mixing layer. *Twenty-Seventh Symposium (International) on Combustion* (1998), pp. 899–907.

Westbrook, C. K. Chemical kinetics of hydrocarbon ignition in practical combustion systems. *Proceedings of the Combustion Institute 28* (2000), pp. 1563–1577.

Westbrook, C. K., Yasuhiro, Pointsot, T. J., phillips J. Smith and Warnatz, J. Computational combustion. *Proceedings of the Combustion Institute*, 30 (2005), pp. 125–157.

Wiki, C. Cfd online homepage, May 2008. Available at `http://www.cfd-online.com/Wiki/Combustion`.


AALBORG UNIVERSITY
DENMARK

A

Solver capability comparison

A.1 Introduction

The following tables, A.1 through A.4 are used as a short summary of the possibilities available in OpenFOAM compared to those in Fluent. The comparison is based on the authors opinion and findings in regards to the experience of the authors.

A.2 General

	OpenFOAM	**Fluent**
Parallelisation	Yes	Yes
Scripting	Yes	Yes (but not good)
Solution monitoring	Yes	Yes
Adjustment of parameters during solving	Yes (easy)	Yes (could be better)

Table A.1: *Runtime options and general considerations*

A.3 Thermo physical

	OpenFOAM	Fluent
Chemistry		
Transient global/elementary reactions	Yes	Yes
Steady global/elementary reactions	No	Yes
Thermodynamics		
Conduction/convection	Yes	Yes
Radiation	No	Yes
Particle flow		
Particle tracking	No (langerian library avaiable)	Yes
Particle combustion	No	Yes
Multiphase flow	Yes	Yes

Table A.2: *Physical models avaliable in OpenFOAM and Fluent*

A.4 Mesh and boundary conditions

	OpenFOAM	Fluent
Mesh		
Polyhedral (tets, hex etc.)	Yes	Yes
2D and axi-symmetric	Yes (3d, 1 cell thick)	Yes
BC type		
Exterior/interior wall	Yes	Yes
Interior face	No	Yes
Porous jump	No	Yes
Porous zone	No (exist as library)	Yes
Pressure outlet, pressure inlet, velocity inlet	Yes	Yes
Mass flow inlet	No	Yes
Exhaust fan, inlet vent, outlet vent, intake fan	No	Yes
Outflow, pressure far field,	No	Yes

Table A.3: *Mesh types and boundary conditions avaliable in OpenFOAM and Fluent*

A.5 Solver setup

	OpenFOAM	Fluent
Graphical user interface	Yes (needs improvement)	Yes
Turbulence models		
k-ε, k-ω, (wall functions)	Yes	Yes
Large Eddy Simulation	Yes	Yes
Direct Numerical Simulation	Yes	Yes
Laminar flow, transient and steady state	Yes	Yes
Numerical Interpolation scheme		
Spatial 1. order upwind (upwind)	Yes	Yes
Spatial 2. order upwind (Quick)	Yes	Yes
Time 1. order (Euler)	Yes	Yes
Time 2. order (CrankNicholson)	Yes	Yes
Pressure velocity coupling		
Simple	Yes	Yes
PISO	Yes	Yes
Coupled	No	Yes

Table A.4: *Solvers and schemes avaliable in OpenFOAM and Fluent*


AALBORG UNIVERSITY
DENMARK

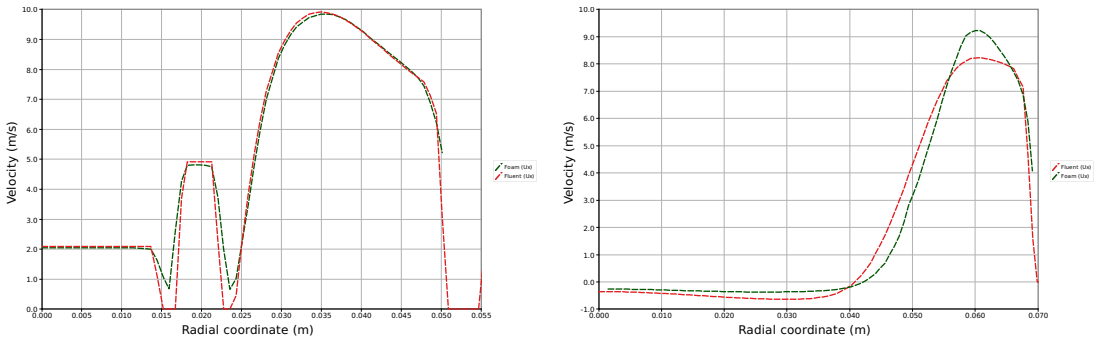
B

OpenFOAM vs Fluent cold-flow line plots

The green line is OpenFOAM results and Fluent is the red color.

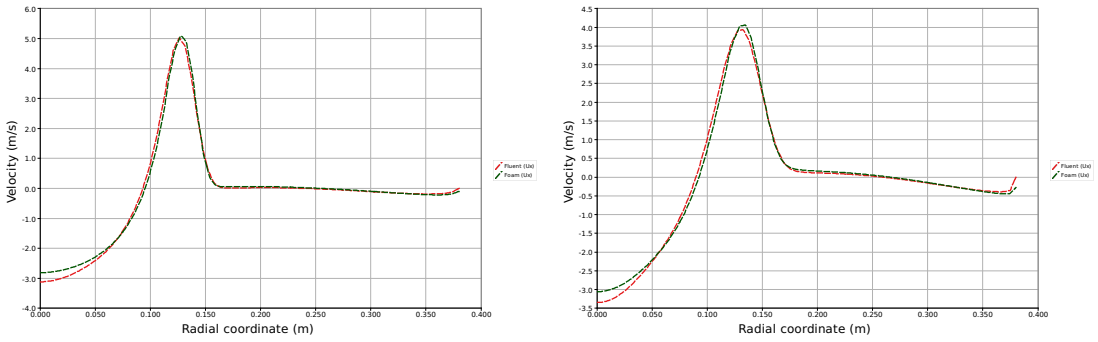
B.1 9.5deg

This is the line plots at different axial locations for the BFR with a swirl angle of 9.5°.



(a) (b)

Figure B.1: *Comparison of OpenFOAM and Fluent, axial velocity at inlet (a), and x=0.03m (b).*



(a) (b)

Figure B.2: *Comparison of OpenFOAM and Fluent, axial velocity at x=0.15m (a), and x=0.2m (b).*

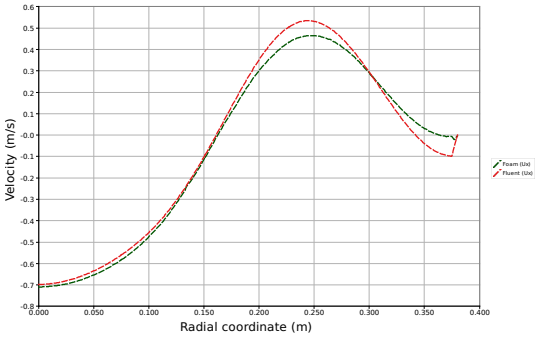
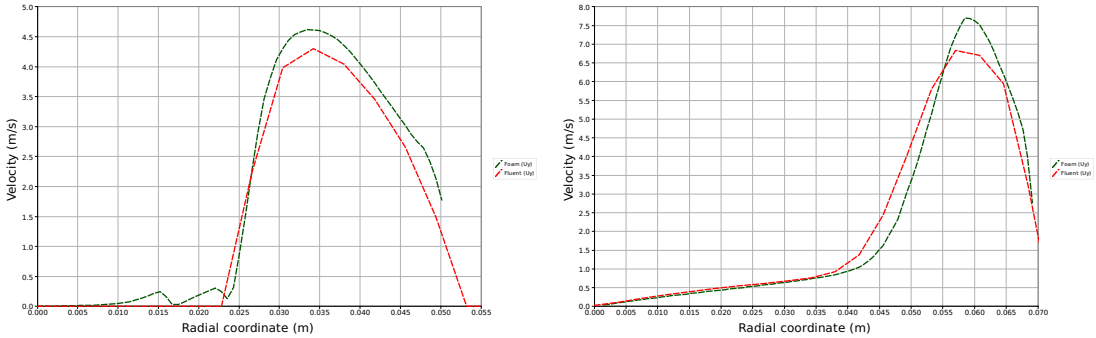
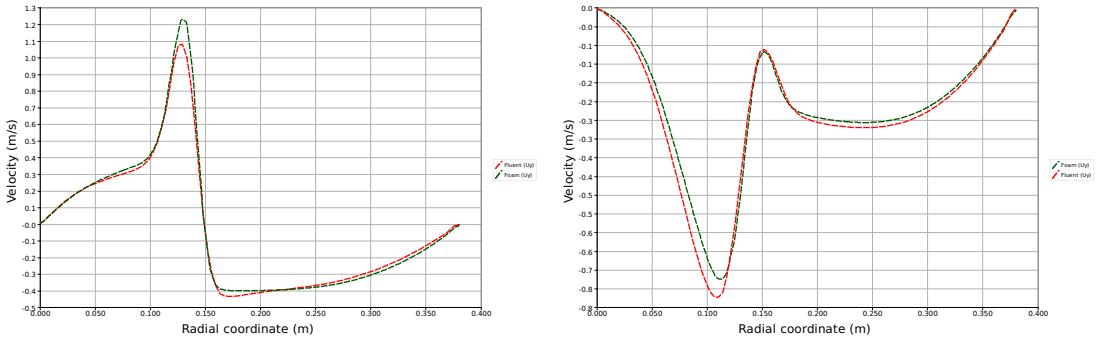


Figure B.3: *Comparison of OpenFOAM and Fluent, axial velocity x=0.5m.*



(a) (b)

Figure B.4: *Comparison of OpenFOAM and Fluent, radial velocity at inlet (a), and x=0.03m (b).*



(a) (b)

Figure B.5: *Comparison of OpenFOAM and Fluent, radial velocity at x=0.15m (a), and x=0.2m (b).*

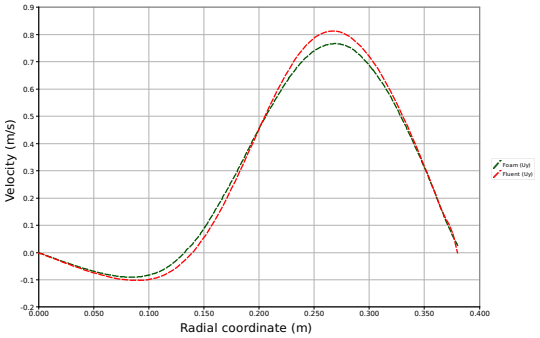
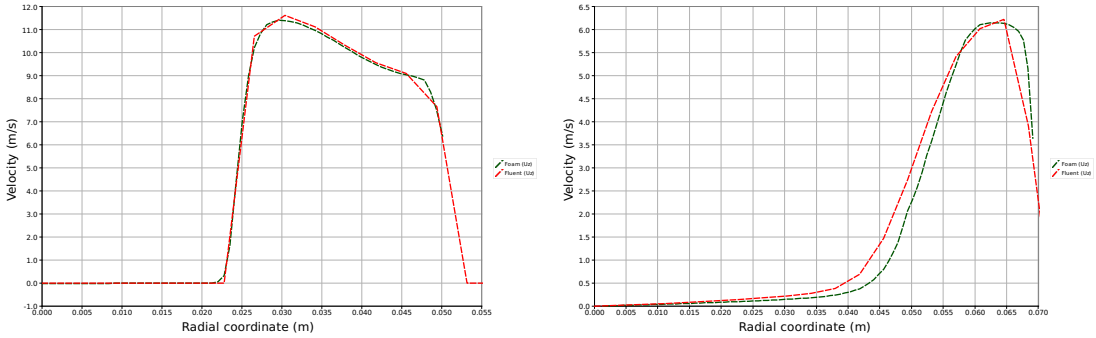
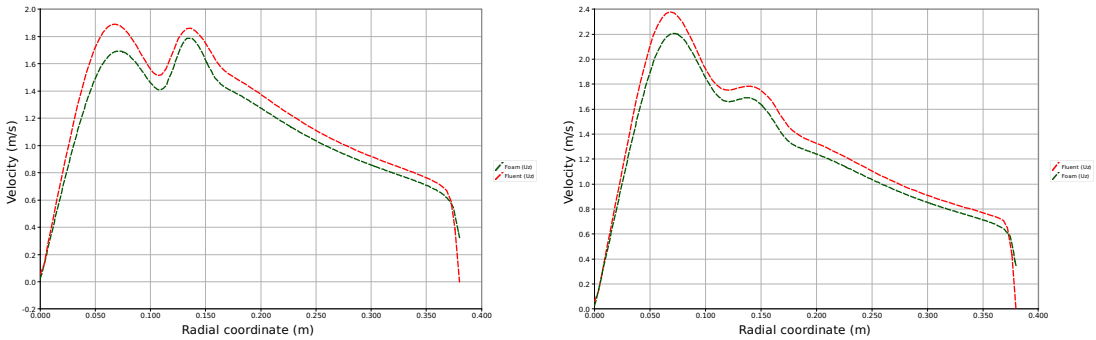


Figure B.6: *Comparison of OpenFOAM and Fluent, radial velocity at x=0.5m.*



(a) (b)

Figure B.7: *Comparison of OpenFOAM and Fluent tangential velocity at inlet (a), and x=0.03m (b).*



(a) (b)

Figure B.8: *Comparison of OpenFOAM and Fluent tangential velocity at x=0.15m (a), and x=0.2m (b).*

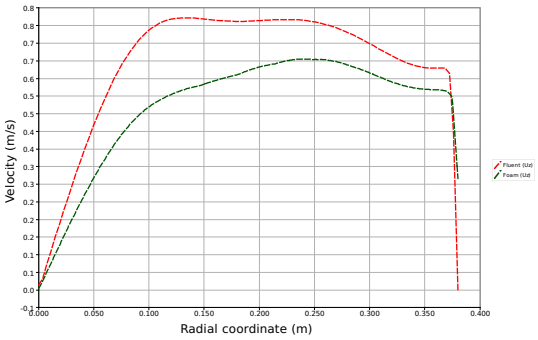
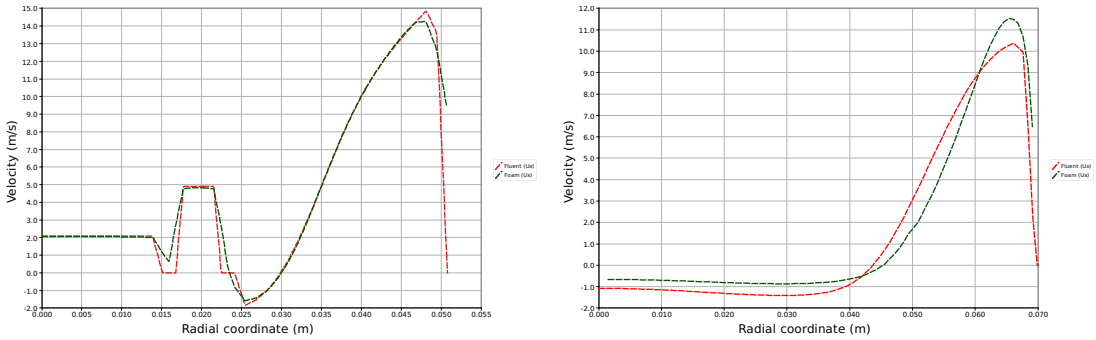


Figure B.9: *Comparison of OpenFOAM and Fluent tangential velocity at x=0.5m.*

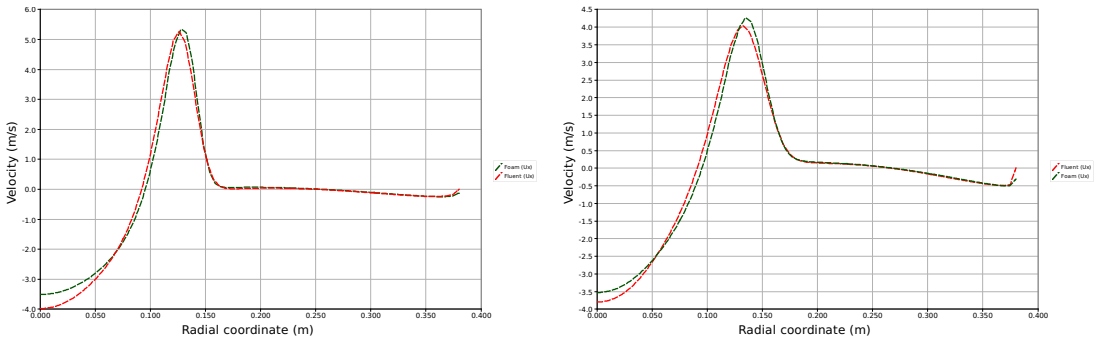
B.2 15.5deg

This is the line plot at different axial locations for the BFR with a swirl angle of 15.5°.



(a) (b)

Figure B.10: *Comparison of OpenFOAM and Fluent, axial velocity at inlet (a), and x=0.03m (b).*



(a) (b)

Figure B.11: *Comparison of OpenFOAM and Fluent, axial velocity at x=0.15m (a), and x=0.2m (b).*

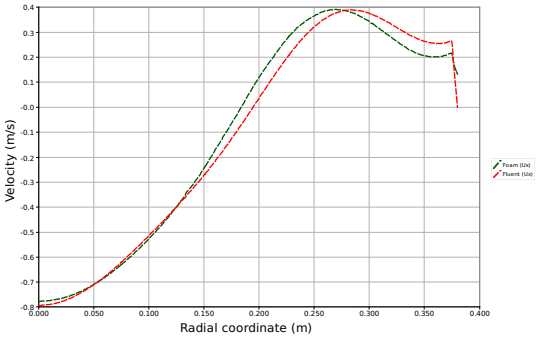
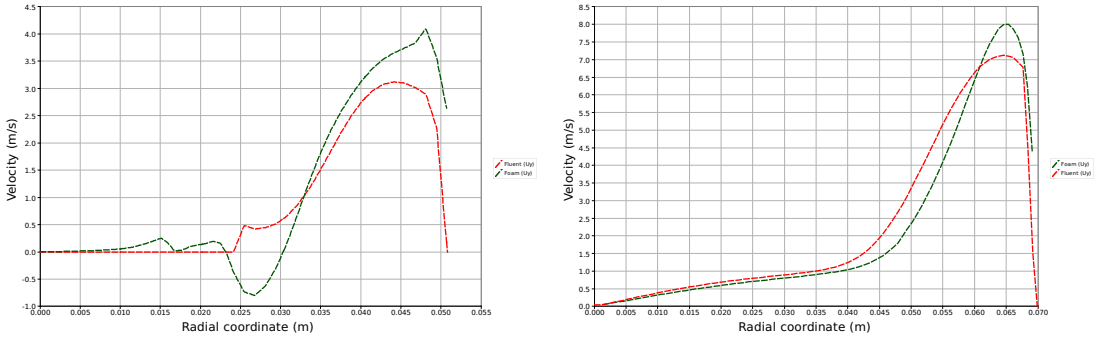
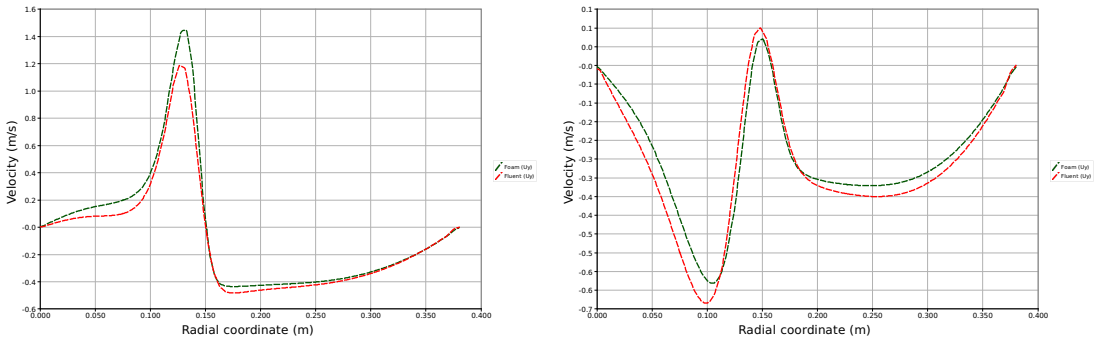


Figure B.12: *Comparison of OpenFOAM and Fluent, axial velocity at x=0.5m.*



(a) (b)

Figure B.13: *Comparison of OpenFOAM and Fluent, radial velocity at inlet (a), and x=0.03m (b).*



(a) (b)

Figure B.14: *Comparison of OpenFOAM and Fluent, radial velocity at x=0.15m (a), and x=0.2m (b).*

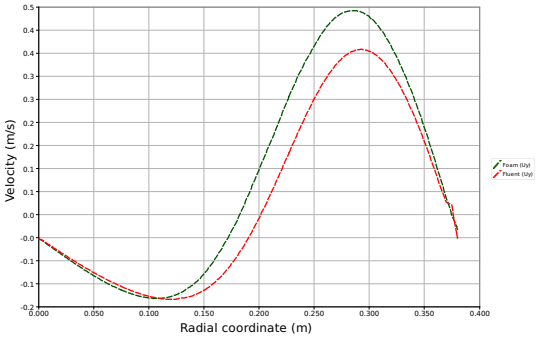
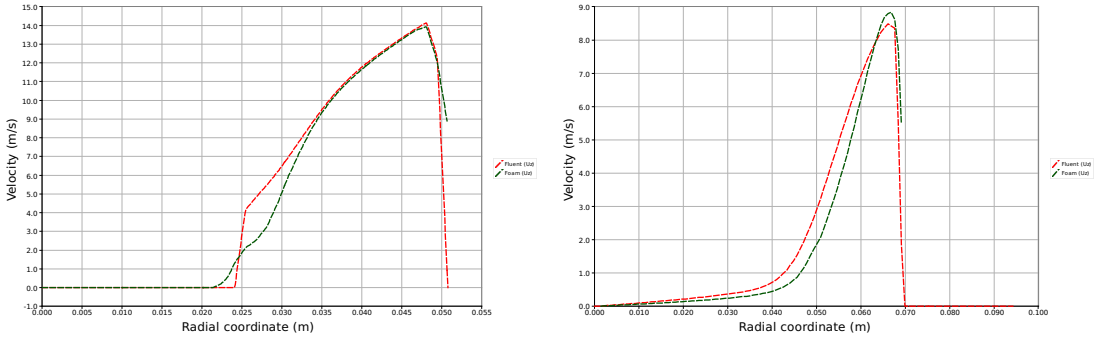
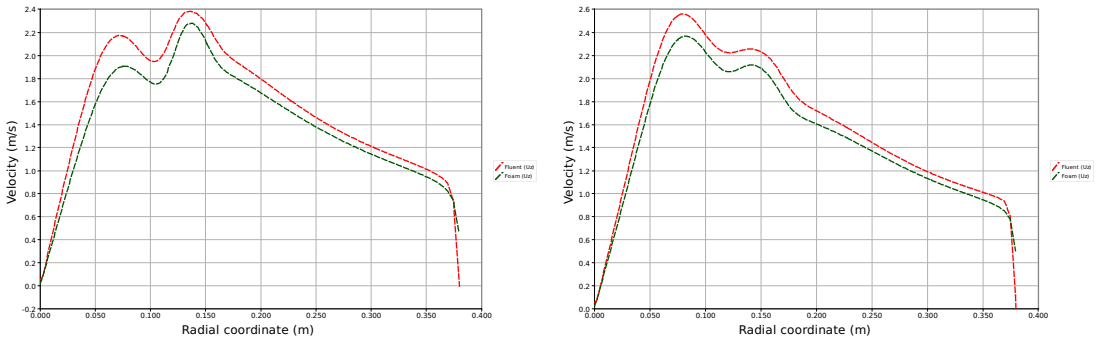


Figure B.15: *Comparison of OpenFOAM and Fluent, radial velocity at x=0.5m.*



(a) (b)

Figure B.16: *Comparison of OpenFOAM and Fluent tangential velocity at inlet (a), and x=0.03m (b).*



(a) (b)

Figure B.17: *Comparison of OpenFOAM and Fluent tangential velocity at x=0.15m (a), and x=0.2m (b).*

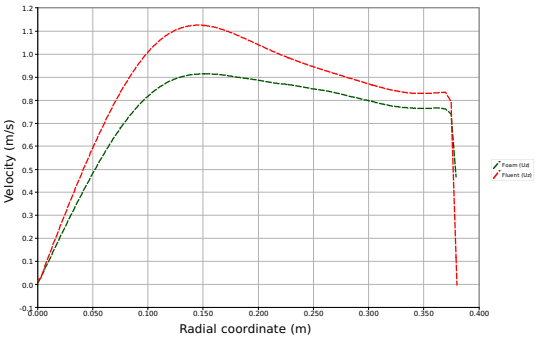


Figure B.18: *Comparison of OpenFOAM and Fluent tangential velocity at x=0.5m.*

C

Boundary conditions for the secondary inlet

This is the boundary conditions for the vector inlet for the cold flow simultaion in chapter 3. The vector is generated using a user defined script for Fluent provided by Søren K. Kær.

C.1 9.5deg swirl

```
secondary_inlet
{
        type            fixedValue;
        value           nonuniform
        (
(5.038372517  1.318976521  5.948118210)
(6.983522415  2.280838966  8.717851639)
(7.614884377  2.468930721  8.916778564)
(7.937578201  2.712002039  9.017991066)
(8.216681480  2.985629082  9.129273415)
(8.504680634  3.262526274  9.270412445)
(8.802019119  3.529788017  9.446068764)
(9.104012489  3.777939796  9.664211273)
(9.382325172  4.001732349  9.919239998)
(9.615644455  4.192168236  10.20353889)
(9.794603348  4.343037128  10.49393559)
(9.881971359  4.448816299  10.77063465)
(9.866350174  4.495499134  11.02808857)
(9.719843864  4.476028442  11.24520302)
(9.397566795  4.372411728  11.41704082)
(8.842724800  4.149442673  11.51684952)
(7.948215008  3.759859800  11.49510479)
(6.524629116  3.090357304  11.21446896)
(4.469147682  2.027052641  10.29091644)
(1.868039012  0.7867641449  7.147181034)
        );
}
```

C.2 15.5deg swirl

```
secondary_inlet
{
        type            fixedValue;
        value           nonuniform
        (
(10.81075478  2.245611906  9.646548271)
(14.07942581  3.912551403  14.09775543)
(14.50617790  3.561279774  13.96904087)
(14.02873039  3.420178890  13.64336872)
(13.31704330  3.378517628  13.26274586)
(12.55818272  3.336833477  12.86591911)
(11.73062229  3.247148037  12.45657349)
(10.79064274  3.087699890  12.02578545)
(9.699790001  2.861071587  11.55841637)
(8.446446419  2.578186035  11.03434658)
(7.055155277  2.259540558  10.43528271)
(5.579270840  1.916506767  9.744189262)
(4.062003136  1.544011235  8.935476303)
(2.601680279  1.122992158  7.978330612)
(1.304342985  0.6289692521  6.862952232)
(0.2164029926  0.2028484792  5.566578865)
(-0.6440873742 -0.1735256463  3.539119959)
(-1.271425724 -0.4080839157  2.511145115)
(-1.670281649 -0.5768163800  1.906583071)
(-1.944216013 -0.4505060911  1.592400432)
        );
}
```

D

reactingFoam code

This is the code for reactingFoam.

```
1: #include "fvCFD.H"
2: #include "hCombustionThermo.H"
3: #include "compressible/turbulenceModel/turbulenceModel.H"
4: #include "chemistryModel.H"
5: #include "chemistrySolver.H"
6: #include "multivariateScheme.H"
7:
8: // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
9:
10: int main(int argc, char *argv[])
11: {
12: #   include "setRootCase.H"
13: #   include "createTime.H"
14: #   include "createMesh.H"
15: #   include "readChemistryProperties.H"
16: #   include "readEnvironmentalProperties.H"
17: #   include "createFields.H"
18: #   include "initContinuityErrs.H"
19: #   include "readTimeControls.H"
20: #   include "setInitialDeltaT.H"
21:
22: // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
23:
24:     Info << "\nStarting time loop\n" << endl;
25:
26:     while (runTime.run())
27:     {
28: #       include "readTimeControls.H"
29: #       include "readPISOControls.H"
30: #       include "compressibleCourantNo.H"
31: #       include "setDeltaT.H"
32:
33:         runTime++;
34:         Info<< "Time = " << runTime.timeName() << nl << endl;
35:
36: #       include "chemistry.H"
37: #       include "rhoEqn.H"
```

```
38: #       include "UEqn.H"
39:
40:         for (label ocorr=1; ocorr <= nOuterCorr; ocorr++)
41:         {
42: #           include "YEqn.H"
43:
44: #           define Db turbulence->alphaEff()
45: #           include "hEqn.H"
46:
47:             // --- PISO loop
48:             for (int corr=1; corr<=nCorr; corr++)
49:             {
50: #               include "pEqn.H"
51:             }
52:         }
53:
54:         turbulence->correct();
55:
56:         rho = thermo->rho();
57:
58:         runTime.write();
59:
60:         Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
61:             << "  ClockTime = " << runTime.elapsedClockTime() << " s"
62:             << nl << endl;
63:     }
64:
65:     Info<< "End\n" << endl;
66:
67:     return(0);
68: }
```

E

OpenFOAM parallelisation

This appendix has the content for the *decomposeParDict* used for parallelising the mesh described in chapter 4.

```
numberOfSubdomains 4;

method          simple;

simpleCoeffs
{
    n               (1 4 1);
    delta           0.001;
}
hierarchicalCoeffs
{
    n               (1 1 1);
    delta           0.001;
    order           xyz;
}
metisCoeffs
{
    processorWeights
    (
        1
        1
        1
        1
    );
}
manualCoeffs
{
    dataFile        "";
}
distributed     no;

roots
(
);
```


AALBORG UNIVERSITY
DENMARK

F

Programming with OpenFOAM

F.1 Mesh variables

In OpenFOAM the declaration of new mesh-variables is made more flexible by the use of classes. The mesh-variables can either be *volScalarField*, *volVectorField* or *volTensorField* which refers to the dimension of the allocated matrix. The read and write settings have several options depending on the type of variable and the need for postprocessing.

An example of a variable declarations is presented below, the placement of this piece of code is usually in the declarationfile *createFields.H*. In the presented example the dimensions and scalar value are defined directly in the code (hardcoding), but could also be specified as a file: *case/0/CO2*.

```
1: volScalarField CO2
2: (
3:     IOobject
4:     (
5:         "CO2",
6:         runTime.timeName(),
7:         mesh,
8:         IOobject::READ_IF_PRESENT,
9:         IOobject::AUTO_WRITE
10:     ),
11:     mesh,
12: // Optional declaration, this can be done by accessing a file in "case/0/"
13: //   dimensionedScalar("zero", dimensionSet(1,-1,-3,0,0,0,0), value)
14: );
```

Mesh variable	Read option	Write option
volScalarField	NO_READ	NO_WRITE
volVectorField	READ_IF_PRESENT	AUTO_WRITE
volTensorField	MUST_READ	

Table F.1: *Overview of mesh-variable input/output options.*

F.2 Mesh loop

For changing variables inside the mesh it is necessary to be able to call and internal mesh values. This section will shortly describe how to read/write mesh data sequential. First the matrix is initialized i line (2)

Example of a mass fraction limiter used in this project:

```
1: // Initialize the variable Y_i for use in a loop
2: scalarField& CO2Internal = CO2.internalField();
3:
4: // Loop for all mesh points
5: forAll(CO2, celli)
6: {
7: // Limits the mass fraction to a positive number
8:     if (CO2Internal[celli] < 0.0)
9:     {
10:     CO2Internal[celli] = 0.0;
11:     }
12: // Limits the mass fraction to max 1.0
13:     if (CO2Internal[celli] > 1.0)
14:     {
15:     CO2Internal[celli] = 1.0;
16:     }
17: }
```

When evaluating cell values it is important to keep a decimal so that the evaluation function knows that it is a float.

F.3 Transport equation in OpenFOAM

The OpenFOAM enviroment makes it easy to implement new transport equations for other species, both passive and reactive. Here is a example of the implementation of the carbon dioxide specie transport. The numerical scheme for both the divergence scheme and the laplacian scheme is defined in the file *case/system/fvSchemes*.

```
1: // Store previous value for under-relaxation
2: CO2.storePrevIter();
3:
4: // Define a ScalarMatrix as a object
5:         fvScalarMatrix CO2Eqn
6:             (
7:               fvm::div(phi, CO2)
8:                 - fvm::laplacian(turbulence->nuEff(),CO2)
9:                 == S_CO2
10:             );
11:
12: // Apply underrelaxation to the equation
13: // Under relaxation factors defined in file: fvSolution
14:         CO2Eqn.relax();
```

```
15:
16: // Solve the differential equation
17:         CO2Eqn.solve();
```

F.4 EBU in OpenFOAM

This is the implementation of the Eddy Break Up model in OpenFOAM.

```
1: /* Modeling the chemical mechanisms for the transport equations */
2:
3: double st, C_R, C_RR, ohno;
4:
5:     st   = 4.0;
6:     C_R  = 4.0;
7:     C_RR = 2.0;
8:     ohno = 0.0;
9:
10: scalarField& omegaInternal = omega.internalField();
11: scalarField& omega_fuInternal = omega_fu.internalField();
12: scalarField& omega_oxInternal = omega_ox.internalField();
13: scalarField& omega_prInternal = omega_pr.internalField();
14:
15: scalarField& epsilonInternal = epsilon.internalField();
16: scalarField& kInternal = k.internalField();
17: scalarField& CH4Internal = CH4.internalField();
18: scalarField& O2Internal = O2.internalField();
19: scalarField& CO2Internal = CO2.internalField();
20: scalarField& H2OInternal = H2O.internalField();
21: scalarField& N2Internal = N2.internalField();
22:
23: scalarField& S_CH4Internal = S_CH4.internalField();
24: scalarField& S_O2Internal = S_O2.internalField();
25: scalarField& S_CO2Internal = S_CO2.internalField();
26: scalarField& S_H2OInternal = S_H2O.internalField();
27:
28: // Nitrogen is the specie which is not solver for
29: forAll (N2Internal, celli)
30: {
31: N2Internal[celli] = 1.0 -
(CH4Internal[celli]+O2Internal[celli]+CO2Internal[celli]+H2OInternal[celli]);
32: }
33:
34: // Reaction rate loop, determine the minimum of the reaction rates
35:
36: forAll (omegaInternal, celli)
37: {
38:     omega_fuInternal[celli] = C_R*CH4Internal[celli];
39:     omega_oxInternal[celli] = C_R*O2Internal[celli]/st;
40:     omega_prInternal[celli] = C_RR*(CO2Internal[celli]+
```

```
H2OInternal[celli])/(1+st);
41:
42: if (omega_fuInternal[celli] <= omega_oxInternal[celli])
43:     {
44:     omegaInternal[celli] = omega_fuInternal[celli];
45:     }
46: if (omega_fuInternal[celli] >= omega_oxInternal[celli])
47:     {
48:     omegaInternal[celli] = omega_oxInternal[celli];
49:     }
50: if (omega_prInternal[celli] < omegaInternal[celli])
51:     {
52:     omegaInternal[celli] = omega_prInternal[celli];
53:     }
54: if (omegaInternal[celli] < 0.0)
55:     {
56:     omegaInternal[celli] = 0.0;
57:     }
58: // If there is an error determing the reaction rate
59: // this will keep the calculation goind and display an error message
60: else
61:     {
62:     ohno = ohno + 1.0;
63:     omegaInternal[celli] = omega_fuInternal[celli];
64:     }
65:
66: // Definition of the source terms
67:
68:     S_CH4Internal[celli] = -omegaInternal[celli]*
epsilonInternal[celli]/kInternal[celli];
69:     S_O2Internal[celli]  = -4*omegaInternal[celli]*
epsilonInternal[celli]/kInternal[celli];
70:     S_CO2Internal[celli] = 2.75*omegaInternal[celli]*
epsilonInternal[celli]/kInternal[celli];
71:     S_H2OInternal[celli] = 2.25*omegaInternal[celli]*
epsilonInternal[celli]/kInternal[celli];
72:
73: }
74:     Info << "Determination of the source term did not succed: " << ohno <<
" Number of times!" << endl;
75:     Info << "Minimum/Maximun reaction rate: " << min(omega.internalField())
<< "/" << max(omega.internalField()) << endl;
76:     Info << "Minimum/Maximun omega_fu: " << min(omega_fu.internalField())
<< "/" << max(omega_fu.internalField()) << endl;
77:     Info << "Minimum/Maximun omega_ox: " << min(omega_ox.internalField())
<< "/" << max(omega_ox.internalField()) << endl;
78:     Info << "Minimum/Maximun omega_pr: " << min(omega_pr.internalField())
<< "/" << max(omega_pr.internalField()) << endl;
```

```
79: }
```

F.5 EDC in OpenFOAM

This is the implementation of the Eddy Dissipation Concept in OpenFOAM.

```
1: /* Initializing  */
2: double st, C_R, C_RR;
3:
4:     st   = 4.0;
5:     C_R  = 1.0;
6:     C_RR = 0.5;
7:
8: // Define scalerfields for input to the loop
9: scalarField& omegaInternal = omega.internalField();
10: scalarField& omega_fuInternal = omega_fu.internalField();
11: scalarField& omega_oxInternal = omega_ox.internalField();
12:
13: scalarField& epsilonInternal = epsilon.internalField();
14: scalarField& kInternal = k.internalField();
15: scalarField& CH4Internal = CH4.internalField();
16: scalarField& O2Internal = O2.internalField();
17: scalarField& CO2Internal = CO2.internalField();
18: scalarField& H2OInternal = H2O.internalField();
19: scalarField& N2Internal = N2.internalField();
20:
21: scalarField& S_CH4Internal = S_CH4.internalField();
22: scalarField& S_O2Internal = S_O2.internalField();
23: scalarField& S_CO2Internal = S_CO2.internalField();
24: scalarField& S_H2OInternal = S_H2O.internalField();
25:
26: // Nitrogen is the specie which is not solver for
27: forAll (N2Internal, celli)
28: {
29: N2Internal[celli] = 1.0 -
(CH4Internal[celli]+O2Internal[celli]+CO2Internal[celli]+H2OInternal[celli]);
30: }
31:
32: // Eddy dissipation model
33: scalarField& gammaInternal = gamma.internalField();
34: scalarField& xiInternal = xi.internalField();
35: scalarField& viscInternal = visc.internalField();
36:
37: // Calculating the viscosity
38: visc = turbulence->nuEff()-turbulence->nut();
39:
40: // Calculating the mass fraction of fine scales and the fraction of reaction
41: forAll(gammaInternal, celli)
42: {
```

```
43: gammaInternal[celli] = viscInternal[celli]/kInternal[celli]*
epsilonInternal[celli]/kInternal[celli];
44: gammaInternal[celli] = 4.6 * (gammaInternal[celli])/3.0;
45:
46: if (CO2Internal[celli]+H2OInternal[celli] <= 0.0)
47: {
48: xiInternal[celli] = 0.0;
49: }
50: if (CO2Internal[celli]+H2OInternal[celli] > 0.0)
51: {
52: xiInternal[celli] = ((CO2Internal[celli]+H2OInternal[celli])/(1+st))/
(min(CH4Internal[celli],O2Internal[celli]) + (CO2Internal[celli]+H2OInternal[celli])/(1+
53: }
54: }
55:
56: // Reaction rate loop, determine the minimum of the reaction rates
57: forAll (omegaInternal, celli)
58: {
59:     omega_fuInternal[celli] = C_R*CH4Internal[celli];
60:     omega_oxInternal[celli] = C_R*O2Internal[celli]/st;
61:
62: if (omega_fuInternal[celli] <= omega_oxInternal[celli])
63:     {
64:     omegaInternal[celli] = omega_fuInternal[celli];
65:     }
66: if (omega_fuInternal[celli] >= omega_oxInternal[celli])
67:     {
68:     omegaInternal[celli] = omega_oxInternal[celli];
69:     }
70: // If there is an error determing the reaction rate
71: // this will keep the calculation goind and display an error message
72: else
73:     {
74:     omegaInternal[celli] = omega_fuInternal[celli];
75:     Info << "An Error in the reaction rate loop has occured" << endl;
76:     }
77:
78: // Definition of the source terms
79:
80:     S_CH4Internal[celli] = -omegaInternal[celli]*epsilonInternal[celli]/
kInternal[celli]*(xiInternal[celli]/(1.0-gammaInternal[celli]*xiInternal[celli]));
81:     S_O2Internal[celli]  = -4*omegaInternal[celli]*epsilonInternal[celli]/
kInternal[celli]*(xiInternal[celli]/(1.0-gammaInternal[celli]*xiInternal[celli]));
82:     S_CO2Internal[celli] = 2.75*omegaInternal[celli]*epsilonInternal[celli]/
kInternal[celli]*(xiInternal[celli]/(1.0-gammaInternal[celli]*xiInternal[celli]));
83:     S_H2OInternal[celli] = 2.25*omegaInternal[celli]*epsilonInternal[celli]/
kInternal[celli]*(xiInternal[celli]/(1.0-gammaInternal[celli]*xiInternal[celli]));
84:
```

```
85: }
86:     Info << "Minimum  reaction rate: " << min(omega.internalField()) << endl;
87:     Info << "Maximum reaction rate: " << max(omega.internalField()) << endl;
88: }
```


AALBORG UNIVERSITY
DENMARK

G

SimpleFoam - Steady state turbulence solver

G.1 Introduction

In this chapter the OpenFOAM implementation of the steady state incompressible turbulence solver is presented. The solver is divided into sections of the physical models for better reuse of the code. To give the best overview the solver code is presented in the general way (the solver code) and afterwards more specific which header files are combined.

G.2 Solver code

```
#include "fvCFD.H"
#include "incompressible/singlePhaseTransportModel/singlePhaseTransportModel.H"
#include "incompressible/turbulenceModel/turbulenceModel.H"

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

int main(int argc, char *argv[])
{

#   include "setRootCase.H"

#   include "createTime.H"
#   include "createMesh.H"
#   include "createFields.H"
#   include "initContinuityErrs.H"

    //mesh.clearPrimitives();

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

    Info<< "\nStarting time loop\n" << endl;

    for (runTime++; !runTime.end(); runTime++)
    {
        Info<< "Time = " << runTime.timeName() << nl << endl;

#       include "readSIMPLEControls.H"
```

```
            p.storePrevIter();

            // Pressure-velocity SIMPLE corrector
            {
                // Momentum predictor

                tmp<fvVectorMatrix> UEqn
                (
                    fvm::div(phi, U)
                  + turbulence->divR(U)
                );

                UEqn().relax();

                solve(UEqn() == -fvc::grad(p));

                p.boundaryField().updateCoeffs();
                volScalarField AU = UEqn().A();
                U = UEqn().H()/AU;
                UEqn.clear();
                phi = fvc::interpolate(U) & mesh.Sf();
                adjustPhi(phi, U, p);

                // Non-orthogonal pressure corrector loop
                for (int nonOrth=0; nonOrth<=nNonOrthCorr; nonOrth++)
                {
                    fvScalarMatrix pEqn
                    (
                        fvm::laplacian(1.0/AU, p) == fvc::div(phi)
                    );

                    pEqn.setReference(pRefCell, pRefValue);
                    pEqn.solve();

                    if (nonOrth == nNonOrthCorr)
                    {
                        phi -= pEqn.flux();
                    }
                }

#               include "continuityErrs.H"

                // Explicitly relax pressure for momentum corrector
                p.relax();

                // Momentum corrector
                U -= fvc::grad(p)/AU;
```

```
            U.correctBoundaryConditions();
        }

        turbulence->correct();

        runTime.write();

        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }

    Info<< "End\n" << endl;

    return(0);
}

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

G.3 Overview of header files

fvCFD.H

1. parRun.H
    (a) OPstream.H
    (b) IPstream.H
    (c) IOstreams.H
2. Time.H
    (a) TimePaths.H
        i. fileName.H
        ii. word.H
    (b) objectRegistry.H
        i. HashTable.H
        ii. label.H
        iii. word.H
        iv. className.H
        v. regIOobject.H
    (c) Iodictionary.H

i. dictionary.H
ii. regIOobject.H

(d) FIFOStack.H

i. SLList.H
ii. Llist.H
iii. label.H
iv. SLListBase.H

(e) clock.H

i. ctime

(f) cpuTime.H

(g) TimeState.H

i. dimensionedScalar.H
ii. dimensionedType.H
iii. word.H
iv. direction.H
v. dimensionSet.H
vi. VectorSpace.H
vii. scalar.H
viii. dimensionedScalarFwd.H
ix. scalar.H
x. floatScalar.H
xi. doubleFloat.H
xii. products.H
xiii. label.H
xiv. direction.H
xv. word.H
xvi. doubleScalar.H
xvii. doubleFloat.H
xviii. products.H
xix. pTraits.H
xx. label.H
xxi. direction.H
xxii. word.H

(h) Switch.H

i. bool.H
ii. word.H

(i) instantList.H

i. instant.H
ii. List.H
iii. Ulist.H
iv. label.H
v. bool.H
vi. autoPtr.H

(j) NamedEnum.H

(k) typeInfo.H

(l) dlLibraryTable.H

(m) functionObjectList.H

i. functionObject.H
A. typeInfo.H
B. autoPtr.H
C. runTimeSelectionTables.H
D. HashPtrTable.H
E. HashTable.H
F. label.H
G. int.H
H. word.H
I. string.H
J. char.H
K. string
L. className.H
M. word.H
N. debug.H

3. fvMesh.H

(a) polyMesh.H

(b) lduMesh.H

(c) primitiveMesh.H

(d) fvBoundaryMesh.H

(e) surfaceInterpolation.H

(f) DimensionedField.H

(g) volFieldsFwd.H

(h) surfaceFieldsFwd.H

(i) pointFieldsFwd.H

(j) SlicedGeometricField.H

(k) slicedVolFieldsFwd.H

(l) slicedSurfaceFieldsFwd.H

(m) className.H

4. fvc.H

(a) fv.H

(b) surfaceInterpolate.H

(c) fvcVolumeIntegrate.H

(d) fvcSurfaceIntegrate.H

(e) fvcAverage.H

(f) fvcReconstruct.H

(g) fvcDdt.H

(h) fvcDDt.H

(i) fvcD2dt2.H

(j) fvcDiv.H

(k) fvcFlux.H

(l) fvcGrad.H

(m) fvcMagSqrGradGrad.H

(n) fvcSnGrad.H

(o) fvcCurl.H

(p) fvcLaplacian.H

(q) fvcSup.H

(r) fvcMeshPhi.H

5. fvMatrices.H

(a) fvMatricesFwd.H

   (b) fvScalarMatrix.H

6. fvm.H
   (a) fvmDdt.H
   (b) fvmD2dt2.H
   (c) fvmDiv.H
   (d) fvmLaplacian.H
   (e) fvmSup.H
7. linear.H
   (a) surfaceInterpolationScheme.H
   (b) volFields.H
8. calculatedFvPatchFields.H
9. fixedValueFvPatchFields.H
10. adjustPhi.H
    (a) volFieldsFwd.H
    (b) surfaceFieldsFwd.H
11. findRefCell.H
12. mathematicalConstants.H
    (a) scalar.H
13. Osspecific.H
    (a) fileNameList.H
    (b) long.H
14. argList.H
    (a) stringList.H
    (b) SubList.H
    (c) SLList.H
    (d) HashTable.H
    (e) word.H

(f) fileName.H

(g) parRun.H

(h) sigFpe.H

(i) sigInt.H

(j) sigQuit.H

(k) sigSegv.H

singlePhaseTransportModel.H

1. transportModel.H

   (a) IOdictionary.H

   (b) volFieldsFwd.H

   (c) surfaceFieldsFwd.H

2. autoPtr.H

turbulenceModel.H

1. volFields.H
2. surfaceFields.H

   (a) GeometricFields.H

   (b) surfaceMesh.H

   (c) fvMesh.H

   (d) fvsPatchFields.H

   (e) surfaceFieldsFwd.H

   (f) calculatedFvsPatchFields.H

3. nearWallDist.H

   (a) volFields.H

      i. GeometricFields.H
      ii. GeometricScalarField.H
      iii. GeometricTensorField.H
      iv. GeometricSphericalTensorField.H
      v. volMesh.H
      vi. GeoMesh.H

vii. fvMesh.H

viii. primitiveMesh.H

ix. fvMesh.H

A. polyMesh.H
B. lduMesh.H
C. primitiveMesh.H
D. fvBoundaryMesh.H
E. surfaceInterpolation.H
F. DimensionedField.H
G. volFieldsFwd.H
H. surfaceFieldsFwd.H
I. pointFieldsFwd.H
J. SlicedGeometricField.H
K. slicedVolFieldsFwd.H
L. fieldTypes.H
M. slicedSurfaceFieldsFwd.H
N. className.H
O. fvPatchField.H
P. fvPatch.H
Q. DimensionedField.H
R. volFieldsFwd.H
S. fieldTypes.H
T. calculatedFvPatchFields.H
U. calculatedFvPatchField.H
V. fieldTypes.H

(b) fvm.H

(c) fvc.H

(d) fvMatrices.H

(e) incompressible/transportModel/transportModel.H

(f) IOdictionary.H

(g) Switch.H

(h) bound.H

(i) autoPtr.H

(j) runTimeSelectionTables.H

i. token.H

A. label.H
B. scalar.H

C. word.H
D. InfoProxy.H
E. refCount.H
F. bool.H
G. typeInfo.H
H. error.H
I. className.H
J. runTimeSelectionTables.H

ii. autoPtr.H

iii. HashTable.H

A. label.H
B. word.H
C. className.H

setRootCase.H

createTime.H

createMesh.H

createFields.H

(a) createPhi.H

initContinuityErrs.H

readSIMPLEControls.H

continuityErrs.H