Cite as: Åkerblom, A.: Turbulence-chemistry interaction in OpenFOAM and how to implement a dynamic PaSR model for LES of turbulent combustion. In Proceedings of CFD with OpenSource Software, 2022, Edited by Nilsson. H., http://dx.doi.org/10.17196/OS_CFD#YEAR_2022

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Turbulence-chemistry interaction in OpenFOAM and how to implement a dynamic PaSR model for LES of turbulent combustion

Developed for OpenFOAM-v2112

Author:
Arvid Åkerblom
Lund University
arvid.akerblom@energy.lth.se

Peer reviewed by:
André Da Luz Moreira
Christer Fureby
Mohammad Hossein
Arabnejad Khanouki

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files. The material has gone through a review process. The role of the reviewer is to go through the tutorial and make sure that it works, that it is possible to follow, and to some extent correct the writing. The reviewer has no responsibility for the contents.

January 28, 2023

Learning outcomes

The main requirements of a tutorial in the course is that it should teach the four points: How to use it, The theory of it, How it is implemented, and How to modify it. Therefore the list of learning outcomes is organized with those headers.

The reader will learn:

How to use it:

- how turbulence-chemistry interaction models are used in OpenFOAM.
- how to use the new foxPaSR model when running a case.

The theory of it:

- why turbulence-chemistry interaction models are needed and how they affect the chemistry of a combustion simulation.
- which turbulence-chemistry interaction models are already available in OpenFOAM as well as their advantages and limitations.
- the physical and chemical motivation behind the Perfectly Stirred Reactor (PSR), Eddy Dissipation Concept (EDC), Fractal (FM) and Partially Stirred Reactor (PaSR) models.
- the physical and chemical motivation behind foxPaSR, the new variant of PaSR presented here.

How it is implemented:

- how reactive flow solvers in OpenFOAM retrieve and use information about the chemical reactions occurring in a simulation.
- the necessary functions of a turbulence-chemistry interaction model, which are required by the solver.

How to modify it:

- how to implement a new turbulence-chemistry interaction model, which utilizes the PaSR method but alters reaction rates individually and dynamically.
- how to set up a simple and coarse LES case to test and compare turbulence-chemistry interaction models on a desktop or laptop computer.

Prerequisites

The reader is expected to know the following in order to get maximum benefit out of this report:

- The fundamentals of fluid mechanics, combustion, and turbulence.
- Familiarity with CFD and turbulence modeling, in particular LES.
- How to set up and run cases in OpenFOAM.
- Basic knowledge of object orientation and C++ syntax.
- How to find files, classes and functions within the OpenFOAM source code.

Contents

Nomenclature

Acronyms

CFD	Computational Fluid Dynamics
DNS	Direct Numerical Simulation
EDC	Eddy Dissipation Concept
FM	Fractal Model
FRC	Finite-Rate Chemistry
LES	Large Eddy Simulation
LHS	Left-Hand Side
PaSR	Partially Stirred Reactor
PSR	Perfectly Stirred Reactor
RANS	Reynolds-Averaged Navier-Stokes
RHS	Right-Hand Side

English symbols

$\mathbf{b_i}$	Sub-grid source term
$\tilde{\mathbf{v}}$	Filtered velocity vector
$\tilde{T}$	Filtered temperature
$\tilde{Y}_i$	Filtered mass fraction of species i
A	Arrhenius law pre-exponential factor
C_γ	EDC model coefficient
c_{tot}	Total species concentration
D_3	Fractal dimension of sub-grid vortex cascade (FM)
D_i	Diffusion coefficient of species i
e_1	EDC model coefficient
e_2	EDC model coefficient
E_a	Arrhenius law activation energy
k	Sub-grid turbulent kinetic energy
$k_{f,j}$	Forward rate constant of reaction j
$k_{r,j}$	Reverse rate constant of reaction j
n	Arrhenius law temperature exponent
N_η	Number of dissipative length scales (FM)
n_R	Number of reactions in the reaction mechanism
N_T	Number of generated length scales (FM)
$N_{s,RHS}$	Number of species on RHS of a reaction
s_u	Laminar flame speed
v'	Magnitude of velocity fluctuations

Greek symbols

Δ	LES filter width
$\dot{\omega}_{i,j}$	Production rate of species i in reaction j
$\dot{\omega}_i$	Unfiltered production rate of species i

$\dot{\omega}_i^0$	Production rate of species i outside fine structures
ϵ	Turbulent dissipation
η	Dissipative length scale
γ^*	Volume fraction of reactive fine structures
γ_j^*	Reactive fine structure volume fraction belonging to reaction j
γ_{tot_l}	Volume fraction of turbulent scales (FM)
ν	Kinematic viscosity
ν_{eff}	Effective kinematic viscosity
$\nu_{i,j}$	Stoichiometric coefficient of species i in reaction j
$\overline{\dot{\omega}_i}$	Filtered production rate of species i
$\overline{\rho}$	Filtered density
τ^*	Mixing time scale
τ_Δ	Time scale of sub-grid velocity stretch
τ_c	Chemical time scale
τ_k	Kolmogorov time scale
$\tau_{c,j}$	Chemical time scale of reaction j

Chapter 1

Introduction

Combustion is a complex physical phenomenon that couples chemistry with fluid flow. The medium is typically a compressible gas, and the flow is turbulent in most engineering applications. Combustion is a hot topic among Computational Fluid Dynamics (CFD) engineers and CFD scientists, as a result of the many challenges involved in gathering quantitative and reliable data from flames by experimental methods, particularly for real engine designs.

When investigating a turbulent combustion process, the aspiring simulator must determine which phenomena and which scales should be directly resolved and which should be modeled. A good starting point is to decide whether to use Reynolds-Averaged Navier-Stokes (RANS) methods, Large Eddy Simulation (LES), or Direct Numerical Simulation (DNS). These methods pertain to the treatment of turbulence, but the choice has significant ramifications for the chemistry as well: turbulence affects the transport of products and reactants in a flame, and the flame affects the turbulence in turn in a complex two-way coupled process. If all turbulent motions are unresolved, as in RANS, then all turbulence-chemistry interactions are also unresolved. This puts a lot of weight on the shoulders of the turbulence and chemistry models, as well as their compatibility with each other. Furthermore, while RANS results can be accurate, unsteady motions and instantaneous results are often of key importance to scientists who wish to further their understanding of the actual processes involved in turbulent combustion. DNS is in many ways the ideal simulation method because it avoids the need to model turbulence and turbulence-chemistry interactions as all turbulent motions are resolved. However, as the reader is likely aware, DNS is often prohibitively expensive computationally.

Recently, the computational power available to scientists and engineers has grown large enough to allow for LES with full-scale engine geometries, opening up exciting opportunities to understand the turbulent flames that power them. LES offers a compromise between cost and accuracy by resolving the large scales of turbulence and modeling the small, or sub-grid, scales. Pope's recommendation [1] that at least 80% of the turbulent kinetic energy should belong to the resolved scales in order to have realistic large-scale results is often applied, but the LES filter width can be reduced further in order to increase the accuracy of the simulation. As the filter width approaches the Kolmogorov scales, the impact of the unresolved scales becomes smaller and the burden on the sub-grid models becomes lighter. As long as there is unresolved turbulence, however, a sub-grid turbulence model is required. There are a variety of well-validated sub-grid turbulence models for LES [2]. The topic is as deep as it is fascinating, and we will not delve into it here. What is important is that sub-grid models alter the viscosity experienced by resolved flow, often based on an estimate of the sub-grid kinetic energy k, or in other words, the kinetic energy of the unresolved turbulent scales. Several sub-grid turbulence models are included in OpenFOAM, such as Smagorinsky [3], WALE [4], kEqn [5], and dynamicKEqn [6].

On its own, a sub-grid turbulence model alters the resolved flow based on the turbulent kinetic energy of the unresolved scales. This is insufficient in the case of reactive flows as it does not take into account the turbulence-chemistry interactions that occur on the sub-grid scales. Such interactions may be negligible in some cases, such as for low Karlovitz numbers, but can have a significant impact on the overall reaction rate in other cases. Models for turbulence-chemistry interaction, which obtain

information from the resolved scales and the sub-grid turbulence model to adjust the reaction rate, have therefore been developed. Here we put the spotlight on these models. In the next chapter, we will study the physical and chemical theory of turbulence-chemistry interactions and discuss some existing models. Then, we will explore the source code of OpenFOAM to understand how these models are used in a CFD simulation. Finally, we will use the existing PaSR class to implement a new model, foxPaSR, which alters the rates of individual species and reactions rather than scaling all reaction rates by the same amount, and test it with a relatively simple LES case.

Chapter 2

Turbulence-chemistry interaction and its modeling

Turbulence and chemistry can interact in quite complex ways. In this chapter, we introduce the problem of determining the total effect of such interactions based on information obtained from the resolved flow and the sub-grid turbulence model. We then discuss a number of existing turbulence-interaction models which tackle the problem in different ways.

2.1 The filtering problem

Combustion chemistry can be accounted for in LES in a number of ways, but the arguably most direct method is referred to as Finite Rate Chemistry (FRC). In addition to the filtered Navier-Stokes equations for mass, momentum, and energy conservation, a balance equation for the filtered mass fraction $\tilde{Y}_i$ of each involved chemical species i is included as well. Each such species equation is given by

$$\frac{\partial \overline{\rho}\tilde{Y}_i}{\partial t} + \nabla \cdot (\overline{\rho}\tilde{Y}_i\tilde{\mathbf{v}}) = \nabla \cdot (D_i \nabla \tilde{Y}_i) + \overline{\dot{\omega}_i} - \nabla \cdot \mathbf{b_i}, \tag{2.1}$$

where $\overline{\rho}$ is the filtered density, $\tilde{\mathbf{v}}$ the filtered velocity, and D_i the diffusion coefficient of species i. The final term on the right hand side, $\nabla \cdot \mathbf{b_i}$, comes from the filtering of the convective term and is computed via the sub-grid turbulence model. The source term $\overline{\dot{\omega}_i}$ is the filtered production rate of species i due to chemical reactions. The role of a turbulence-chemistry interaction model is to adjust $\overline{\dot{\omega}_i}$ to account for sub-grid turbulence.

It would now be wise to make a distinction between the filtered production rate $\overline{\dot{\omega}_i}$ and the unfiltered production rate $\dot{\omega}_i$. The unfiltered production rate is computed under the assumption that chemistry-related variables, such as species concentrations and temperature, are uniformly distributed in the control volume of interest (which in the case of LES would be a single mesh cell). But how is it computed? In FRC, this is done by a *reaction mechanism*: a set of possible reactions and the coefficients needed to compute their rates under different thermochemical circumstances. The reaction mechanism is also what determines the list of involved species (like O_2, for example) and thus the amount of balance equations following Equation 2.1. Each reaction consists of a Left-Hand Side (LHS) and a Right-Hand Side (RHS), and can be either reversible or irreversible. Each reaction has a forward rate constant determined by the modified Arrhenius law,

$$k_{f,j} = A\tilde{T}^n \exp\Big(-\frac{E_a}{R\tilde{T}}\Big), \tag{2.2}$$

where the model coefficients A, n, and E_a are given by the mechanism. Reversible reactions also have a reverse rate constant, $k_{r,j}$. The production and consumption rates of all species are collectively determined by the rate constants of the reactions, the concentrations of the different species, and the stoichiometric coefficients of each species in each reaction. Therefore, a system of ordinary differential

equations, one for each species, must be solved to obtain the production and consumption rates. Such equation systems can be very large and numerically stiff, and are therefore costly to solve, which is why reaction mechanisms used in LES are typically heavily reduced. Heavy hydrocarbons such as kerosene may in reality contain hundreds of species participating in thousands of reactions, while some of the largest mechanisms [7] that are currently feasible for LES contain about 50 species and less than 300 reactions. If the details of the chemistry are of lesser importance, global mechanisms containing only a handful of species and reactions are often used as they greatly reduce the simulation cost.

The goal, then, is to first obtain $\dot{\omega}_i$ and then solve the equation $\overline{\dot{\omega}_i} = f(\dot{\omega}_i, x_{turb})$, where x_{turb} are any necessary variables pertaining to turbulence. In other words, we wish to compute the filtered reaction rate $\overline{\dot{\omega}_i}$ as a function of the unfiltered reaction rate $\dot{\omega}_i$ and any other information from the flow and/or sub-grid turbulence model that we may need. Can this be done? Yes. Can this be done accurately? That remains an open question. The answer is likely that it depends a lot on the flame under consideration, its Reynolds, Karlovitz and Damköhler numbers, and the filter width. In any case, it is wise to use a filter width that is as small as one can afford, in order to reduce the importance of the turbulence-chemistry interaction model.

2.2 Existing models

Let us now discuss some existing turbulence-chemistry interaction models.

2.2.1 Perfectly Stirred Reactor (a.k.a. "Laminar")

The simplest way to model turbulence-chemistry interaction is to not model it at all. If we assume that all interaction occurs on the resolved scales, there is no need to account for unresolved structures. Each control volume acts as a homogeneous reactor, or a Perfectly Stirred Reactor (PSR), which means that $\overline{\dot{\omega}_i} = \dot{\omega}_i$. This works well for laminar cases, but it also works for turbulent cases as long as the filter width is small enough that unresolved turbulence-chemistry interaction is negligible. This "model" is implemented in the laminar class in OpenFOAM.

2.2.2 Eddy Dissipation Concept

Batchelor *et al.* [8] and Chomiak *et al.* [9] noted that in turbulent flames at high Reynolds numbers, reactions are localized in turbulent fine structures concentrated in small regions. In a given control volume, the volume fraction of these reacting fine structures is denoted γ^*. For the filtered production rate, it follows that $\overline{\dot{\omega}_i} = \gamma^* \dot{\omega}_i^* + (1-\gamma^*)\dot{\omega}_i^0$, where $\dot{\omega}_i^*$ is the production rate in the reacting fine structures and $\dot{\omega}_i^0$ the production rate in the remainder of the volume. We now introduce two assumptions. First, we assume that $\dot{\omega}_i^* = \dot{\omega}_i$, or in other words, that the production rate in the fine structures is the same as the unfiltered production rate (per unit volume). Next, we assume that $\dot{\omega}_i^* \gg \dot{\omega}_i^0$, because the reacting fine structures are by definition the region where intense chemical activity occurs. With these assumptions, we get a simple linear relationship between the filtered and unfiltered production rates: $\overline{\dot{\omega}_i} = \gamma^* \dot{\omega}_i$. This is the basis of the Eddy Dissipation Concept (EDC) model as well as all models mentioned henceforth. In the EDC model, the reacting volume fraction is estimated with the equation

$$\gamma^* = \frac{\left(C_\gamma \left(\nu\epsilon/k^2\right)^{1/4}\right)^{e_1}}{1 - \left(C_\gamma \left(\nu\epsilon/k^2\right)^{1/4}\right)^{e_2}}, \tag{2.3}$$

where ν is the kinematic viscosity, ϵ the turbulent dissipation, and k the sub-grid kinetic energy. The constants C_γ, e_1, and e_2 are model coefficients. The estimation is based on a model of the turbulent cascade put forth by Magnussen [10]. EDC has existed for a long time and is well-validated. Because the reacting volume fraction is fully determined by turbulence, however, the characteristics of the chemistry itself are not considered. This may be problematic in situations where different fuels, with potentially different fine structure behavior, are compared at identical Reynolds numbers. In OpenFOAM, the model is implemented in the EDC class. Several versions are available, which use

different values for the model coefficients. In the OpenFOAM implementation, a specific time step length for solving the ordinary differential equation system of production rates is also computed.

2.2.3 Fractal Model

Giacomazzi *et al.* [11] have developed a sub-grid turbulence model based on fractal theory called the Fractal Model (FM). FM generates a cascade of vortices in each cell based on the filter width and local Reynolds number (with regard to the filter width), and alters the viscosity based on the amount of vortices generated during the cascade from the filter width scale to the dissipative length scale. This model was later extended [12] to account for sub-grid turbulence-chemistry interaction. In a nutshell, the fractal dimension D_3 of the sub-grid vortex cascade is first computed. The fraction γ_{tot_l} of the cell volume that contains turbulent scales of any kind is then computed as

$$\gamma_{tot_l} = \left(\frac{\Delta}{\eta}\right)^{D_3-3}, \tag{2.4}$$

where Δ is the filter width and η the dissipative length scale. Because the reacting fine structures are considered to exist close to the dissipative length scale, the volume fraction of reacting fine structures is then computed as

$$\gamma^* = \frac{N_\eta}{N_T}\gamma_{tot_l}, \tag{2.5}$$

where N_η is the number of dissipative length scales and N_T the total number of length scales generated in the cascade. FM is not included in OpenFOAM at the moment.

2.2.4 Partially Stirred Reactor

The Partially Stirred Reactor (PaSR) model estimates the reacting volume fraction by comparing the turbulent mixing time scale, τ^*, with the chemical time scale, τ_c, according to

$$\gamma^* = \frac{\tau_c}{\tau_c + \tau^*}. \tag{2.6}$$

This means that when the mixing is fast relative to the chemistry, $\gamma^* \approx 1$ and the model is simplified to the PSR model. There is no truly correct way to compute either τ^* or τ_c, but here we will present two alternatives for each, starting with the one that is included in OpenFOAM.

The OpenFOAM class PaSR contains an implementation where τ^* is equal to the Kolmogorov time scale, given by

$$\tau_k = \sqrt{\nu_{eff}/\epsilon}, \tag{2.7}$$

where ν_{eff} is the effective viscosity with turbulence taken into account and ϵ the turbulent dissipation rate. The chemical time scale is computed by the expression

$$\tau_c = \sum_{j=1}^{n_R} \frac{c_{tot}}{\sum_{i=1}^{N_{s,RHS}} \nu_{i,j} k_{f,j}}, \tag{2.8}$$

where n_R is the number of chemical reactions under consideration, c_{tot} the total species concentration (number of moles of all species per unit volume), and $N_{s,RHS}$ the number of product species in a given reaction. For each species in a reaction, $\nu_{i,j}$ is the stoichiometric coefficient and $k_{f,j}$ the production rate. In a nutshell, the chemical time scale is computed by dividing the total species concentration by the rate at which species are transferred from the LHS of the reactions to the RHS. The chemical time scale is thus computed dynamically based on information about the chemistry, allowing it to adapt to local conditions. On the other hand, the same reacting volume fraction γ^* is used to scale all species production rates in the end, even though they could differ from each other.

Another strategy for computing τ^* and τ_c is described by Sabelnikov & Fureby [13]. They argue that the mixing time scale should be given by

$$\tau^* = \sqrt{\tau_k \tau_\Delta}, \tag{2.9}$$

where $\tau_\Delta = \Delta/v'$ is the time scale of sub-grid velocity stretch (where v' is the magnitude of velocity fluctuations), which is typically longer than the Kolmogorov time scale. The reasoning behind this change is that turbulent fine structures are generally anisotropic, consisting of ribbon- and tube-like structures which are influenced by velocity stretch. They also use a simplified estimate for the chemical time scale:

$$\tau_c = \nu/s_u^2, \tag{2.10}$$

where ν is the kinematic viscosity and s_u the laminar flame speed. The flame speed pertains to laminar premixed flames, and can be extrapolated from experimental data or calculated with a one-dimensional simulation. If s_u is estimated beforehand based on the properties of the unburnt mixture in a simulated case, then the chemical time scale will be constant throughout the simulation. This is logical if combustion occurs in a premixed mode with an approximately uniform equivalence ratio. If the equivalence ratio varies due to mixing, or if the combustion occurs in a partially premixed or non-premixed mode, the concept of a uniform laminar flame speed stands on shakier foundations. The flame speed could be computed dynamically instead, based on local conditions, but the fact remains that it is not well-defined in the non-premixed mode.

2.3 The new model: foxPaSR

When describing PaSR in the previous section, we identified two potential problems:

- The existing PaSR implementation in OpenFOAM uses the Kolmogorov time scale as the mixing time scale, which has been shown by Sabelnikov & Fureby [13] to be inconsistent with the anisotropic nature of reactive fine structures.
- All reactions are scaled using the same value for γ^*. This may be fine when only one to a handful of reactions are considered, but larger reaction mechanisms may contain reactions with widely varying rates and time scales. It would thus make sense to scale all reactions individually, based on reaction-specific time scales.

We shall now derive a new PaSR model, *foxPaSR*, that attempts to resolve these issues. Solving the first problem is simply a matter of changing $\tau^* = \tau_k$ to $\tau^* = \sqrt{\tau_k \tau_\Delta}$ in the source code of OpenFOAM. In order to obtain $\tau_\Delta = \Delta/v'$, we note that $v' = \sqrt{2k}$ by definition and that $\epsilon = 1.048k^{3/2}/\Delta$ (see the LESModel class). This allows us to use the simplified expression

$$\tau_\Delta = \frac{k}{\epsilon}, \tag{2.11}$$

where model coefficients have been approximated to unity. To solve the reaction rate problem, we will use a novel method for computing a chemical time scale $\tau_{c,j}$ for each reaction j. In this new method, $\tau_{c,j}$ is given by

$$\tau_{c,j} = \min\left(\frac{\overline{\rho}\tilde{Y}_i}{|\dot{\omega}_{i,j}|}\right), \tag{2.12}$$

where the index i refers to a species on the *reactant* side and $\omega_{i,j}$ is the consumption rate [kg/m^3/s] of that species due to reaction j. In other words: a time scale is computed for each species on the reactant side, and the shortest of these is chosen as the chemical time scale of the reaction. This choice can be interpreted as the residence time of the reaction: the time it would take for the reaction to use its total supply of reactants if the reaction were to proceed at the current rate. Without taking other simultaneous reactions into consideration, the residence time appears to be a sound method for estimating "how quickly the reaction happens", but only a thorough validation study will determine whether this choice is well-founded. A separate fine structure volume fraction γ_j^* is then computed for each reaction following Equation 2.6. Finally, the filtered production rate of each species is computed by summing the scaled contributions of all reactions according to

$$\overline{\dot{\omega}_i} = \sum_{j=1}^{n_R} \gamma_j^* \dot{\omega}_{i,j}. \tag{2.13}$$

Note that $\dot{\omega}_{i,j} = 0$ for all reactions the species i is not involved in.

Chapter 3

Turbulence-chemistry interaction in OpenFOAM

In this chapter, we embark on a journey through the OpenFOAM source code in order to understand how turbulence-chemistry interaction models are implemented and used.

3.1 The solver

We begin our quest in the most natural place: the solver. If we want to run a simulation with finite-rate chemistry, we will have to run a solver which supports it. Some examples of such models include reactingFoam, fireFoam, and sprayFoam. Any of these examples work here, but we will choose reactingFoam for clarity's sake.

Near the beginning of the main function in reactingFoam.C, we find the line #include createFields.H. This line essentially executes every line in the file createFields.H which is found in the solver directory. The purpose of createFields.H is to create the various fields that will be solved for in the simulation (like U) but also to create pointers to certain classes based on the models that the user has specified in various dictionaries. In createFields.H we find the following lines.

$FOAM_APP/solvers/combustion/reactingFoam/createFields.H

```
Info<< "Creating reaction model\n" << endl;
autoPtr<CombustionModel<psiReactionThermo>> reaction
(
    CombustionModel<psiReactionThermo>::New(thermo, turbulence())
);
```

These lines create a pointer called reaction, which points to the CombustionModel<psiReactionThermo> class (an instantiation of CombustionModel using psiReactionThermo). This object is our turbulence-chemistry interaction model. The reason it has the generic name CombustionModel rather than something like "TurbulenceChemistryInteractionModel" is because the role of the class is not limited to handling turbulence-chemistry interactions. As we will soon see, turbulence-chemistry interaction models like EDC are sub-classes of CombustionModel, but there are other sub-classes that handle combustion in ways that are quite different from what was outlined in the previous chapter, where we limited ourselves to the finite-rate chemistry approach. The pointer reaction is used during the solver loop when the species equations (2.1) are solved. This happens in YEqn.H, where we find the following code.

$FOAM_APP/solvers/combustion/reactingFoam/YEqn.H

```
reaction->correct();
Qdot = reaction->Qdot();
volScalarField Yt(0.0*Y[0]);
```

```
forAll(Y, i)
{
    if (i != inertIndex && composition.active(i))
    {
        volScalarField& Yi = Y[i];

        fvScalarMatrix YiEqn
        (
            fvm::ddt(rho, Yi)
            + mvConvection->fvmDiv(phi, Yi)
            - fvm::laplacian(turbulence->muEff(), Yi)
            ==
            reaction->R(Yi)
            + fvOptions(rho, Yi)
        );

        YiEqn.relax();

        fvOptions.constrain(YiEqn);

        YiEqn.solve(mesh.solver("Yi"));

        fvOptions.correct(Yi);

        Yi.max(0.0);
        Yt += Yi;
    }
}
```

Before the species equation is defined and solved for each species (note the forAll(Y, i) loop), we find the lines reaction->correct() and Qdot = reaction->Qdot(). The first of these tells the turbulence-chemistry interaction model to correct all species reaction rates, and the second retrieves the updated heat release per unit volume, Qdot, from the turbulence-chemistry interaction model. Within the species equation itself, we also find the term reaction->R(Yi), which is the species production rate corresponding to the term $\overline{\dot{\omega}_i}$ in Equation 2.1.

3.2 The model

We will now assume that the user has selected the turbulence-chemistry interaction model PaSR, as we will use it later when implementing our new model. The laminar, PaSR and EDC classes inherit from CombustionModel according to Figure 3.1.

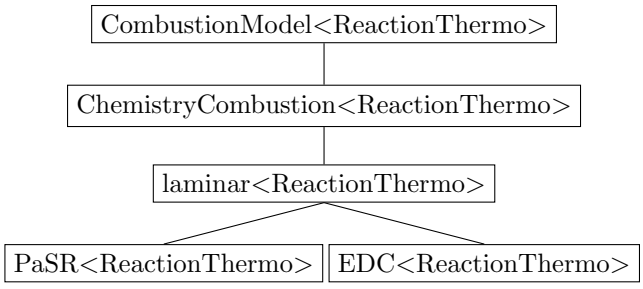

Figure 3.1: Inheritance diagram of turbulence-chemistry interaction models.

The ChemistryCombustion class is a chemistry model wrapper for combustion models. In other words, it provides laminar, PaSR and EDC with access to the chemistry-related functions implemented in the StandardChemistryModel. Crucially, it has the pointer chemistryPtr_ as a member, which points to the active chemistry model. The PSR model, which is implemented in the laminar class, contains the basic functionality needed to retrieve species production rates and heat release

which are both required by the solver. The PaSR and EDC classes are therefore sub-classes of laminar, as they simply extend laminar with methods for adjusting species production rates based on turbulence-chemistry interaction. In laminar.H, we find the following member function declarations:

$FOAM_SRC/combustionModels/laminar/laminar.H

```
//- Correct combustion rate
virtual void correct();

//- Fuel consumption rate matrix.
virtual tmp<fvScalarMatrix> R(volScalarField& Y) const;

//- Heat release rate [kg/m/s3]
virtual tmp<volScalarField> Qdot() const;
```

These are the functions that are called by the solver in YEqn.H. Note that the function R is said to return the "fuel consumption rate" even though it can return the production rate of any species, possibly because the author of the file assumed that one would be using single-step irreversible chemistry. The correct, R and Qdot functions are all reimplemented in PaSR.C. Let us begin with the latter two, as they are quite simple:

$FOAM_SRC/combustionModels/PaSR/PaSR.C

```
template<class ReactionThermo>
Foam::tmp<Foam::fvScalarMatrix>
Foam::combustionModels::PaSR<ReactionThermo>::R(volScalarField& Y) const
{
    return kappa_*laminar<ReactionThermo>::R(Y);
}


template<class ReactionThermo>
Foam::tmp<Foam::volScalarField>
Foam::combustionModels::PaSR<ReactionThermo>::Qdot() const
{
    return tmp<volScalarField>
    (
        new volScalarField
        (
            this->thermo().phasePropertyName(typeName + ":Qdot"),
            kappa_*laminar<ReactionThermo>::Qdot()
        )
    );
}
```

The R function simply returns the species production rates (as computed by laminar) multiplied by the member kappa_, which corresponds to the fine structure volume fraction γ^*. (This variable is also sometimes denoted κ.) The heat release gets the same treatment in the Qdot function, but in a way that ensures that the heat release can be saved as output. The correct function, which is called by the solver before R and Qdot, is responsible for updating kappa_. Let us go through this function, step by step.

$FOAM_SRC/combustionModels/PaSR/PaSR.C

```
template<class ReactionThermo>
void Foam::combustionModels::PaSR<ReactionThermo>::correct()
{
    if (this->active())
    {
        laminar<ReactionThermo>::correct();

        tmp<volScalarField> tepsilon(this->turbulence().epsilon());
        const scalarField& epsilon = tepsilon();

        tmp<volScalarField> tmuEff(this->turbulence().muEff());
        const scalarField& muEff = tmuEff();
```

```
        tmp<volScalarField> ttc(this->tc());
        const scalarField& tc = ttc();

        tmp<volScalarField> trho(this->rho());
        const scalarField& rho = trho();
```

First, laminar<ReactionThermo>::correct() is called to solve the chemistry equations and update all species production rates based on the PSR model. This is followed by the construction of references to any variables of interest - in this case, the turbulent dissipation ϵ, the effective viscosity μ_{Eff}, the chemical time scale τ_c and the density $\overline{\rho}$. The turbulence function is inherited from CombustionModel, and returns access to the turbulence model (which in the case of LES is the sub-grid turbulence model). The rho function is similarly inherited from CombustionModel, while the tc function is inherited from laminar where it uses its access to the chemistry model (via chemistryPtr_) to compute the chemical time scale for all cells according to Equation 2.8. The remainder of the function consists of the following code.

$FOAM_SRC/combustionModels/PaSR/PaSR.C

```
        forAll(epsilon, i)
        {
            const scalar tk =
                Cmix_*sqrt(max(muEff[i]/rho[i]/(epsilon[i] + SMALL), 0));

            if (tk > SMALL)
            {
                kappa_[i] = tc[i]/(tc[i] + tk);
            }
            else
            {
                kappa_[i] = 1.0;
            }
        }
    }
}
```

The forAll(epsilon, i) loop is simply a loop over all cells, utilizing a scalarField (in this case epsilon) that happened to be on hand. For each cell i, the Kolmogorov time scale, denoted tk, is first computed following Equation 2.7. Note the Cmix parameter, which is unity by default but can be given any value by the user *ad hoc* to account for coarse meshes. If the Kolmogorov time scale is greater than a microsecond (tk > SMALL) then kappa_ is computed following the PaSR approach given in Equation 2.6. Otherwise, the mixing is assumed to be infinitely fast and the PSR approach is used instead.

Chapter 4

Implementing the new PaSR model

We are now ready to implement foxPaSR, our new model. The first step is to create a working directory by copying the existing PaSR directory from the source code.

```
mkdir -p $WM_PROJECT_USER_DIR/src/combustionModels
cp -r $FOAM_SRC/combustionModels/PaSR $WM_PROJECT_USER_DIR/src/combustionModels/foxPaSR
mv $WM_PROJECT_USER_DIR/src/combustionModels/foxPaSR/PaSR.C $WM_PROJECT_USER_DIR/src/combustionModels/
    foxPaSR/foxPaSR.C
mv $WM_PROJECT_USER_DIR/src/combustionModels/foxPaSR/PaSR.H $WM_PROJECT_USER_DIR/src/combustionModels/
    foxPaSR/foxPaSR.H
mv $WM_PROJECT_USER_DIR/src/combustionModels/foxPaSR/PaSRs.C $WM_PROJECT_USER_DIR/src/combustionModels
    /foxPaSR/foxPaSRs.C
sed -i s/"PaSR"/"foxPaSR"/g $WM_PROJECT_USER_DIR/src/combustionModels/foxPaSR/foxPaSR.C
sed -i s/"PaSR"/"foxPaSR"/g $WM_PROJECT_USER_DIR/src/combustionModels/foxPaSR/foxPaSR.H
sed -i s/"PaSR"/"foxPaSR"/g $WM_PROJECT_USER_DIR/src/combustionModels/foxPaSR/foxPaSRs.C
```

We are going to compile foxPaSR as a library called FOXcombustionModels, which means that we need a Make directory in the combustionModels directory. That way, it will be easy to add additional turbulence-chemistry interaction models in the future.

```
mkdir $WM_PROJECT_USER_DIR/src/combustionModels/Make
touch $WM_PROJECT_USER_DIR/src/combustionModels/Make/files
touch $WM_PROJECT_USER_DIR/src/combustionModels/Make/options
```

The files in the Make directory should have the following contents, in order to ensure that all necessary libraries are included and the correct compilation destination is used.

Make/files

```
foxPaSR/foxPaSRs.C

LIB = $(FOAM_USER_LIBBIN)/libFOXcombustionModels
```

Make/options

```
EXE_INC = \
    -I$(LIB_SRC)/combustionModels/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/reactionThermo/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/chemistryModel/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/ODE/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/thermophysicalProperties/lnInclude
```

```
LIB_LIBS = \
    -lcombustionModels \
    -lcompressibleTransportModels \
    -lturbulenceModels \
    -lcompressibleTurbulenceModels \
    -lchemistryModel \
    -lfiniteVolume \
    -lmeshTools \
    -lreactionThermophysicalModels \
    -lspecie \
    -lfluidThermophysicalModels \
    -lthermophysicalProperties \
    -lODE
```

We will now begin modifying foxPaSR.C. When we are done with that file, some small changes to foxPaSR.H will also be required. There is no need to modify foxPaSRs.C.

In foxPaSR.C, we first remove kappa_ from the constructor because we don't want a whole field dedicated to each possible reaction. We also add three members: Y_ (the species mass fractions), reactions_ (the list of reactions), and modRR_ (the modified production rates of all species). These new members are initialized by the following lines.

foxPaSR.C

```
Y_(this->thermo().composition().Y()),
reactions_
(
    dynamic_cast<const reactingMixture<gasHThermoPhysics>&>(this->thermo())
),
modRR_(this->chemistryPtr_->nSpecie())
```

The species mass fractions are retrieved from the thermo package (returned by this->thermo()) and the modified production rates is simply a list with one entry per species. The definition of the reaction list is taken from the constructor of the StandardChemistryModel class, specifically instantiated with gasHThermoPhysics as its ThermoType; we will not try to understand what it does in detail, but we will add

foxPaSR.H

```
#include "thermoPhysicsTypes.H"
#include "StandardChemistryModel.H"
```

to the list of inclusions in PaSR.H. The modified reaction rates need to be further initialized as a list of volScalarFields, but this only has to happen once. We will therefore add the following lines to the constructor body.

foxPaSR.C

```
forAll(modRR_, i)
{
    modRR_.set
    (
        i,
        new volScalarField
        (
            IOobject
            (
                thermo.phasePropertyName(
                    typeName + ":modRR_" + Y_[i].name()
                ),
                this->mesh().time().timeName(),
                this->mesh(),
                IOobject::NO_READ,
                IOobject::NO_WRITE
            ),
            this->mesh(),
            dimensionedScalar("modRR",  dimensionSet(1, -3, -1, 0, 0), 0.0)
```

```
        )
    );
    Info << "Creating field " << modRR_[i].name() << endl;
}
```

These lines define each entry in modRR_ as a volScalarField with the dimension [kg/m^3/s]. Each volScalarField has a unique name which includes the name of the produced species.

We will now make a slight modification to the member functions R and Qdot. In foxPaSR, all production rates are modified inside the correct function and there is therefore no need to do any scaling elsewhere. We therefore simply remove the kappa_ factor from the return statements of these functions.

foxPaSR.C

```
template<class ReactionThermo>
Foam::tmp<Foam::fvScalarMatrix>
Foam::combustionModels::foxPaSR<ReactionThermo>::R(volScalarField& Y) const
{
    return laminar<ReactionThermo>::R(Y);
}


template<class ReactionThermo>
Foam::tmp<Foam::volScalarField>
Foam::combustionModels::foxPaSR<ReactionThermo>::Qdot() const
{
    return tmp<volScalarField>
    (
        new volScalarField
        (
            this->thermo().phasePropertyName(typeName + ":Qdot"),
            laminar<ReactionThermo>::Qdot()
        )
    );
}
```

It is time to start modifying the correct function, which will contain the actual implementations of foxPaSR's principles as outlined in chapter 2. We begin by removing the forAll(epsilon, i) loop and adding a reference to the turbulent kinetic energy (right after the definition of muEff). We also remove the reference to the chemical time scale, tc, as we will be using our own method for computing that variable. At this stage, our correct function has the following appearance.

foxPaSR.C

```
template<class ReactionThermo>
void Foam::combustionModels::PaSR<ReactionThermo>::correct()
{
    if (this->active())
    {
        laminar<ReactionThermo>::correct();

        tmp<volScalarField> tepsilon(this->turbulence().epsilon());
        const scalarField& epsilon = tepsilon();

        tmp<volScalarField> tmuEff(this->turbulence().muEff());
        const scalarField& muEff = tmuEff();

        tmp<volScalarField> ttke(this->turbulence().k());
        const scalarField& tke = ttke();

        tmp<volScalarField> trho(this->rho());
        const scalarField& rho = trho();
    }
}
```

The correct function is called once during each time step. After the declaration of rho, we therefore add some lines that set all modified production rates to zero in order to delete information left from the previous time step.

foxPaSR.C

```
forAll(modRR_, i)
{
    forAll(epsilon, celli)
    {
        modRR_[i][celli] = 0.0;
    }
}
```

We also declare a number of scalars that we will be using shortly. These are:

- tc, the chemical time scale τ_c
- tk, the Kolmogorov time scale τ_k
- tp, the time scale of sub-grid velocity stretch τ_Δ
- tm, the mixing time scale τ^*
- kappa, the reactive fine structure volume fraction
- tcSpecies, the residence time of a certain species in a certain reaction

foxPaSR.C

```
scalar tc, tk, tp, tm, kappa, tcSpecies;
```

We will use two nested loops for our calculations: an outer loop which goes through the list of reactions, and an inner loop which goes through all cells once a reaction is specified. Let us start by implementing the outer loop, leaving the inner loop blank for the moment.

foxPaSR.C

```
forAll(reactions_, ri)
{
    label refSpecies = reactions_[ri].lhs()[0].index;

    const scalar& refW =
        this->thermo().composition().W(refSpecies);

    scalarField refRR =
        this->chemistryPtr_->calculateRR(ri, refSpecies)/refW;

    forAll(epsilon, celli)
    {
        // To be filled in...
    }
}
```

Once a reaction has been selected and given the index ri, the index of the first species on the LHS is identified and stored as the label refSpecies. We will henceforth consider this species to be the *reference* species, as it should have a stoichiometric coefficient of 1. Its molecular weight is denoted refW and its production rate is stored as refRR. We use the calculateRR function from StandardChemistryModel, which returns the production rate of a certain species in a certain reaction in [kg/m^3/s]. For our purposes we prefer [kmol/m^3/s], which is why we divide by refW. This way, we get a refRR which represents the number of kmol being transferred from the LHS of the reaction to the RHS. As an example, consider a simple reaction representing the stoichiometric combustion of methane. For the sake of this example, assume that the reaction is reversible.

$$CH_4 + 2O_2 \leftrightarrow CO_2 + 2H_2O \tag{4.1}$$

In the reaction above, CH_4 would be identified as refSpecies. For a positive refRR, precisely refRR kmol of CH_4 and 2·refRR kmol of O_2 are consumed per second, while refRR kmol of CO_2 and 2·refRR kmol of H_2O are produced per second. The same would be true for a negative refRR, but then the species on the LHS would be produced while the species on the RHS would be consumed.

Our goal now is to go through all cells and ensure that the rate of reaction ri is scaled correctly based on local conditions. We begin the forAll(epsilon, celli) loop with the following lines, which set up scalar references to the local cell values of the variables we will need.

foxPaSR.C

```
const scalar& muC = muEff[celli];
const scalar& rhoC = rho[celli];
const scalar& epsC = epsilon[celli];
const scalar& tkeC = tke[celli];
const scalar& refRRC = refRR[celli];
```

We also initialize the chemical time scale using a large number (1 million seconds).

foxPaSR.C

```
tc = GREAT;
```

We want our model to compute the chemical time scale by investigating the residence time of the reactants. In an irreversible reaction, the reactants are easily identified because they are always on the LHS. But what about reversible reactions? One could conceivably split any reversible reaction into two opposing irreversible reactions, counting the LHS as reactants in one direction and the RHS as reactants in the other. We will instead consider the reactants to be on the LHS if refRRC is positive and on the RHS otherwise. We differentiate between these two cases using an if statement. Once the reactant side has been determined, a forAll(reactions_[ri].lhs(), si) (for positive refRRC) or forAll(reactions_[ri].rhs(), si) (for negative refRRC) loop is used to go through the species on the reactant side. For each species si, its index is denoted i, its stoichiometric coefficient is denoted stoichCoeff and its molecular weight is denoted W. Its residence time is computed by the line

foxPaSR.C

```
tcSpecies =
    mag(rhoC*Y_[i][celli]/(refRRC*W*stoichCoeff));
```

which is an implementation of Equation 2.12. The lines

foxPaSR.C

```
if (tcSpecies < tc)
{
    tc = tcSpecies;
}
```

ensure that the shortest residence time on the reactant side is chosen as the chemical time scale. The full procedure for computing the chemical time scale is presented below. Note the use of the SMALL number (10^{-6}) which we use to avoid floating point errors.

foxPaSR.C

```
if (refRRC > SMALL)
{
    forAll(reactions_[ri].lhs(), si)
    {
        const label& i = reactions_[ri].lhs()[si].index;

        const scalar& stoichCoeff =
            reactions_[ri].lhs()[si].stoichCoeff;

        const scalar& W =
            this->thermo().composition().W(i);
```

```
        tcSpecies =
            mag(rhoC*Y_[i][celli]/(refRRC*W*stoichCoeff));

        if (tcSpecies < tc)
        {
            tc = tcSpecies;
        }
    }
}
else if (refRRC < -SMALL)
{
    forAll(reactions_[ri].rhs(), si)
    {
        const label& i = reactions_[ri].rhs()[si].index;

        const scalar& stoichCoeff =
            reactions_[ri].rhs()[si].stoichCoeff;

        const scalar& W =
            this->thermo().composition().W(i);

        tcSpecies =
            mag(rhoC*Y_[i][celli]/(refRRC*W*stoichCoeff));

        if (tcSpecies < tc)
        {
            tc = tcSpecies;
        }
    }
}
```

We now compute the mixing time scale using the following three lines, based on the theory in chapter 2. Note that unlike the default PaSR class, we do not simply use the Kolmogorov time scale.

foxPaSR.C

```
tk =
    sqrt(max(muC/rhoC/(epsC + SMALL), 0));

tp =
    max(tkeC/(epsC + SMALL), 0);

tm = sqrt(tk*tp);
```

With both the chemical and mixing time scales in hand, we can compute the reactive fine structure volume fraction following Equation 2.6. However, it would be wise to set up some limiting cases. More specifically, the PaSR model should simplify to the PSR model if the mixing time scale is very short, or if the chemical time scale is very long. We implement this using an if statement.

foxPaSR.C

```
if (tm > SMALL && tc < GREAT)
{
    kappa = tc/(tc + tm);
}
else
{
    kappa = 1.0;
}
```

We have now estimated the reactive fine structure volume fraction, which means we are ready to compute modified production rates for all species involved in the reaction. We do this by first looping over all species on the LHS and executing the following line.

foxPaSR.C

```
modRR_[i][celli] -=
    kappa*refRRC*W*stoichCoeff;
```

Note the use of -=. This is done because modRR_[i] should contain the sum of all reaction-specific production rates for species i, following Equation 2.13. If refRRC is positive, then species on the LHS are consumed and thus have negative production rates. Conversely, when we next loop over all species on the RHS, += is used instead. The full procedure for adding the correctly modified contribution of reaction ri to the species production rates is presented below.

foxPaSR.C

```
forAll(reactions_[ri].lhs(), si)
{
    const label& i = reactions_[ri].lhs()[si].index;

    const scalar& stoichCoeff =
        reactions_[ri].lhs()[si].stoichCoeff;

    const scalar& W = this->thermo().composition().W(i);

    modRR_[i][celli] -=
        kappa*refRRC*W*stoichCoeff;
}
forAll(reactions_[ri].rhs(), si)
{
    const label& i = reactions_[ri].rhs()[si].index;

    const scalar& stoichCoeff =
        reactions_[ri].rhs()[si].stoichCoeff;

    const scalar& W = this->thermo().composition().W(i);

    modRR_[i][celli] +=
        kappa*refRRC*W*stoichCoeff;
}
```

When all reactions have been accounted for, modRR_ is complete. After the forAll(reactions_, ri) loop, we add the following lines to replace the reaction rates, previously computed by the PSR model, with modRR_.

foxPaSR.C

```
forAll(modRR_, i)
{
    this->chemistryPtr_->RR(i) = modRR_[i];
}
```

Note that we use a forAll(modRR_, i) loop to loop over all species. For each species i, the non-constant production rate, to which chemistryPtr_->RR returns access, is replaced with the modified reaction rate.

At this point foxPaSR.C has been fully implemented, but we must also make some modifications to foxPaSR.H to account for our new members. These only constitute changes to the private data of the class, so all we have to do is remove

foxPaSR.H

```
//- Mixing parameter
volScalarField kappa_;
```

from the list of private data and add the following entries.

foxPaSR.H

```
//- Species mass fractions
const PtrList<volScalarField>& Y_;

//- List of reactions
const PtrList<Reaction<gasHThermoPhysics>>& reactions_;
```

```
//- Scaling factors
PtrList<volScalarField> modRR_;
```

The implementation of foxPaSR is now complete and the library can be compiled with wmake. For the full source code, see appendix A.

Chapter 5

Testing the new PaSR model

In this chapter, we provide a simple demonstration of the new PaSR model, foxPaSR, in use. The model is primarily intended to be used for high Reynolds number LES with large chemical reaction mechanisms, which requires high-performance computing in order to achieve good results. Because we want to be able to test our model on a desktop machine, the test case will have to be an overly coarse-grained LES with a one-step reaction mechanism. This will only demonstrate that the model works, without showing its full potential. All dictionaries needed to run the case are included in appendix B.

The test case uses the reactingFoam solver and is based on the pitzDaily case, used in several OpenFOAM tutorials for various solvers. The geometry is extruded by two channel heights into the third dimension, to enable true (non-2D) LES. The two side boundaries that are introduced by this extrusion are treated as no-slip walls. The mesh consists of 341,250 hexahedral cells which are roughly uniform in size. A fixed velocity of 20 m/s is used at the inlet, resulting in a Reynolds number of approximately 50,000. The outlet pressure is fixed at 1 bar. The incoming gas is a lean mixture of n-heptane and air with an equivalence ratio of $\phi = 0.75$ and a temperature of $\tilde{T} = 400$ K. The combustion chemistry is handled by the one-step mechanism used in the aachenBomb tutorial for the sprayFoam solver. The flow field is allowed to develop for 20 ms, after which setFields is used to change the temperature to 2000 K everywhere beneath the shear layer, causing ignition. The transient ignition process is then flushed out over the course of 80 ms.

The turbulence-chemistry interaction model is chosen in constant/combustionProperties, in exactly the same way one would choose PaSR.

constant/combustionProperties

```
combustionModel foxPaSR;
active          yes;

foxPaSRCoeffs
{
    Cmix        1.0;
}
```

The FOXcombustionModels library must also be included in system/controlDict.

system/controlDict

```
libs ( "libFOXcombustionModels" );
```

Figure 5.1 shows instantaneous snapshots of the temperature for simulations with PSR, PaSR, and foxPaSR, while Figure 5.2 shows the heat release rate at the same moment in time. The premixed flame is stabilized at the shear layer, demarcating the border between the cold unburnt mixture at the top and the hot burnt mixture at the bottom. The influence of the turbulence-chemistry interaction model is quite evident; the PSR model, with its high reaction rate, predicts a lightly wrinkled flame while the PaSR predicts a more unstable and corrugated flame. The flame

predicted by foxPaSR is even more unstable, displaying the thin but unbroken reaction zones typical of flames with Karlovitz numbers $1 < \mathrm{Ka} < 100$.

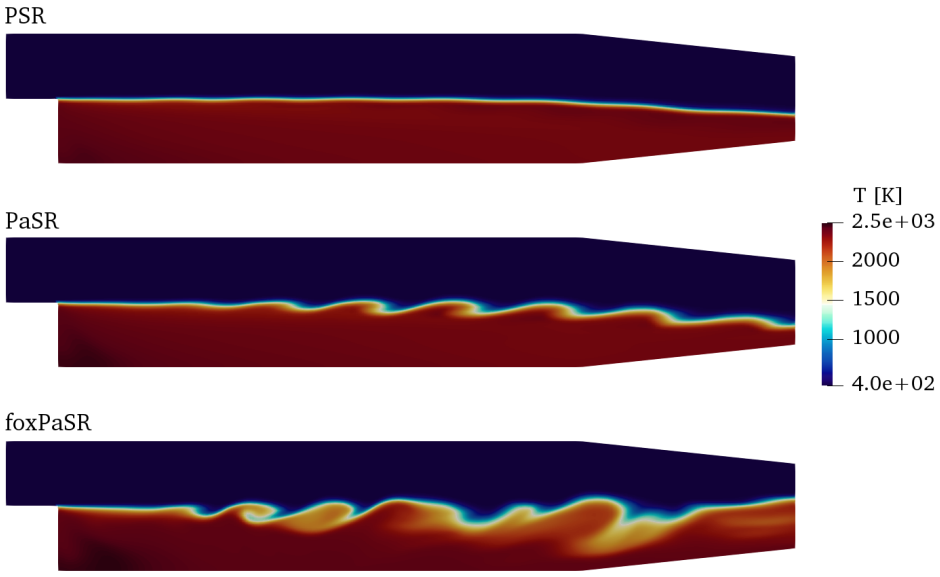

Figure 5.1: Instantaneous temperature plots for PSR, PaSR, and foxPaSR.

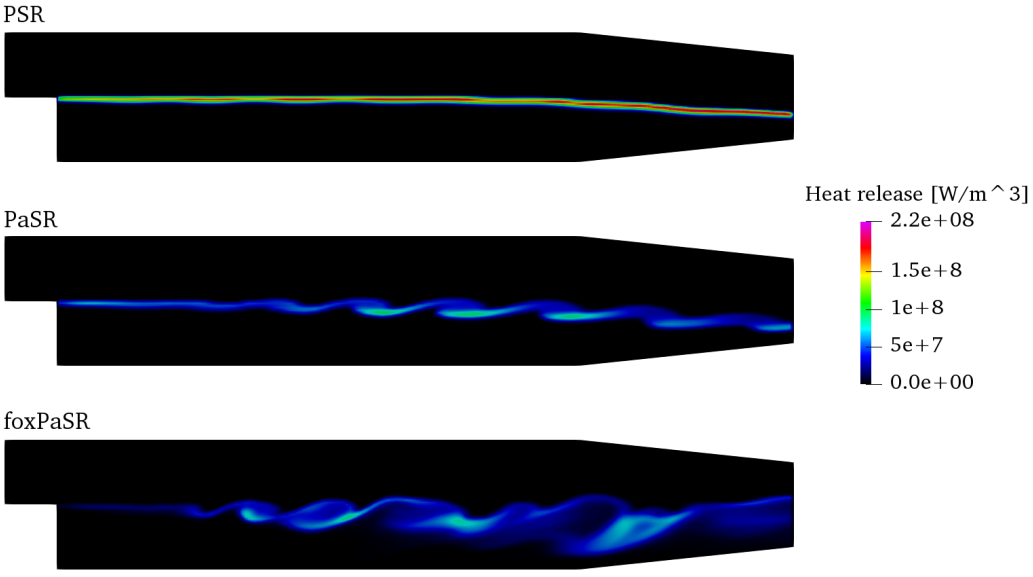

Figure 5.2: Instantaneous heat release plots for PSR, PaSR, and foxPaSR.

Bibliography

[1] S. Pope, "Ten questions concerning the large-eddy simulation of turbulent flows," *New Journal of Physics*, vol. 6, p. 35, 2004.

[2] P. Sagaut, *Large Eddy Simulation for Incompressible Flows*. Heidelberg, Germany: Springer Berlin, 3 ed., 2006.

[3] J. Smagorinsky, "General circulation experiments with the primitive equations: I. The basic experiment," *Monthly weather review*, vol. 91, no. 3, pp. 99–164, 1963.

[4] F. Nicoud and F. Ducros, "Subgrid-Scale Stress Modelling Based on the Square of the Velocity Gradient Tensor," *Flow, Turbulence and Combustion*, vol. 62, pp. 183–200, 1999.

[5] A. Yoshizawa, "A Statistically-derived Subgrid-scale Kinetic Energy Model for the Large Eddy Simulation of Turbulent Flows," *J. Phys. Soc. Jpn.*, vol. 54, p. 2834, 1985.

[6] W.-W. Kim and S. Menon, "A New Dynamic One Equation Subgrid-scale Model for Large Eddy Simulations," in *Proceedings of the 19th Aerospace Sciences Meeting*, (Reno, NV, USA), AIAA 95-0356, 1995.

[7] R. Xu, K. Wang, S. Banerjee, *et al.*, "A physics-based approach to modeling real-fuel combustion chemistry – II. Reaction kinetic models of jet and rocket fuels," *Combust. Flame*, vol. 193, pp. 520–537, 2018.

[8] G. Batchelor and A. Townsend, "The Nature of Turbulent Motion at Large Wave-numbers," *Proc. Roy. Soc. London A*, vol. 199, p. 238, 1949.

[9] J. Chomiak, "A Possible Propagation Mechanism of Turbulent Flames at High Reynolds Numbers," *Comb. Flame*, vol. 15, p. 319, 1970.

[10] B. Magnussen, "On the Structure of Turbulence and Generalized Eddy Dissipation Concept for Chemical Reactions in Turbulent Flow," in *Proceedings of the 9th Aerospace Sciences Meeting*, AIAA 1981-0042, 1981.

[11] E. Giacomazzi, C. Bruno, and B. Favini, "Fractal modeling of turbulent mixing," *Combust. Theory Modelling*, vol. 3, p. 637, 1999.

[12] E. Giacomazzi, C. Bruno, and B. Favini, "Fractal modeling of turbulent combustion," *Combust. Theory Modelling*, vol. 4, p. 391, 2000.

[13] V. Sabelnikov and C. Fureby, "LES combustion modeling for high Re flames using a multi-phase analogy," *Combust. Flame*, vol. 160, no. 1, pp. 83–96, 2013.

Study questions

How to use it:

- Which turbulence-chemistry interaction models are available in OpenFOAM, and what are they called?
- After compiling foxPaSR, how does one select it when running a case?

The theory of it:

- Why are turbulence-chemistry interaction models needed?
- What is the meaning of the "reactive fine structure volume fraction" γ^*?
- On a surface level, how is γ^* computed by EDC, FM, and PaSR, respectively?
- On a surface level, what separates foxPaSR from the PaSR implementation included in OpenFOAM?

How it is implemented:

- How does the solver (e.g. reactingFoam) incorporate the turbulence-chemistry interaction model into the solution algorithm?
- Which functions must be included in a turbulence-chemistry interaction model?
- How does a turbulence-chemistry interaction model access information and functionality in the chemistry model?

How to modify it:

- What do the different functions in a turbulence-chemistry interaction model do, and what do they return when called?
- The implementation of foxPaSR uses an outer loop and an inner loop. What is being looped over?

Appendix A

Source code

foxPaSR.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2017 OpenFOAM Foundation
    Copyright (C) 2019 OpenCFD Ltd.
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

\*---------------------------------------------------------------------------*/

#include "foxPaSR.H"

// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * //

template<class ReactionThermo>
Foam::combustionModels::foxPaSR<ReactionThermo>::foxPaSR
(
    const word& modelType,
    ReactionThermo& thermo,
    const compressibleTurbulenceModel& turb,
    const word& combustionProperties
)
:
    laminar<ReactionThermo>(modelType, thermo, turb, combustionProperties),
    Cmix_(this->coeffs().getScalar("Cmix")),
    Y_(this->thermo().composition().Y()),
    reactions_
    (
        dynamic_cast<const reactingMixture<gasHThermoPhysics>&>(this->thermo())
```

```
48      ),
49      modRR_(this->chemistryPtr_->nSpecie())
50  {
51      forAll(modRR_, i)
52      {
53          modRR_.set
54          (
55              i,
56              new volScalarField
57              (
58                  IOobject
59                  (
60                      thermo.phasePropertyName(
61                          typeName + ":modRR_" + Y_[i].name()
62                      ),
63                      this->mesh().time().timeName(),
64                      this->mesh(),
65                      IOobject::NO_READ,
66                      IOobject::NO_WRITE
67                  ),
68                  this->mesh(),
69                  dimensionedScalar("modRR",  dimensionSet(1, -3, -1, 0, 0), 0.0)
70              )
71          );
72          Info << "Creating field " << modRR_[i].name() << endl;
73      }
74  }
75
76
77  // * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * * //
78
79  template<class ReactionThermo>
80  Foam::combustionModels::foxPaSR<ReactionThermo>::~foxPaSR()
81  {}
82
83
84  // * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * * //
85
86  template<class ReactionThermo>
87  void Foam::combustionModels::foxPaSR<ReactionThermo>::correct()
88  {
89      if (this->active())
90      {
91          // Update chemistry
92          laminar<ReactionThermo>::correct();
93
94          // Reference to turbulent dissipation
95          tmp<volScalarField> tepsilon(this->turbulence().epsilon());
96          const scalarField& epsilon = tepsilon();
97
98          // Reference to effective viscosity
99          tmp<volScalarField> tmuEff(this->turbulence().muEff());
100         const scalarField& muEff = tmuEff();
101
102         // Reference to turbulent kinetic energy
103         tmp<volScalarField> ttke(this->turbulence().k());
104         const scalarField& tke = ttke();
105
106         // Reference to density
107         tmp<volScalarField> trho(this->rho());
108         const scalarField& rho = trho();
109
110         // Reset modified production rates of all species in all cells
111         forAll(modRR_, i)
112         {
113             forAll(epsilon, celli)
114             {
115                 modRR_[i][celli] = 0.0;
```

```
116             }
117         }
118
119         // Declare scalars used when scaling reaction rates
120         scalar tc, tk, tp, tm, kappa, tcSpecies;
121
122         // Loop over all reactions
123         forAll(reactions_, ri)
124         {
125             // Select the first species on the LHS as the reference species
126             label refSpecies = reactions_[ri].lhs()[0].index;
127
128             // Retrieve the molecular weight of the reference species
129             const scalar& refW =
130                 this->thermo().composition().W(refSpecies);
131
132             // Compute the reference reaction rate [kmol/s] of the
133             // reaction. In other words, the production/consumption
134             // rate of the first species on the LHS. We assume that
135             // its stoichiometric coefficient is unity.
136             scalarField refRR =
137                 this->chemistryPtr_->calculateRR(ri, refSpecies)/refW;
138
139             // Loop over all cells
140             forAll(epsilon, celli)
141             {
142                 // Declare references to local cell values
143                 const scalar& muC = muEff[celli];
144                 const scalar& rhoC = rho[celli];
145                 const scalar& epsC = epsilon[celli];
146                 const scalar& tkeC = tke[celli];
147                 const scalar& refRRC = refRR[celli];
148
149                 // Initialize the local chemical time scale
150                 tc = GREAT;
151
152                 // If the net reaction rate is positive, the reaction
153                 // is treated as an irreversible reaction with reactants
154                 // on the LHS and products on the RHS. If it is negative,
155                 // the reaction is treated as an irreversible reaction
156                 // with reactants on the RHS and products on the RHS.
157                 if (refRRC > SMALL)
158                 {
159                     // Loop over all species on LHS
160                     forAll(reactions_[ri].lhs(), si)
161                     {
162                         // Species index
163                         const label& i = reactions_[ri].lhs()[si].index;
164
165                         // Stoichiometric coefficient
166                         const scalar& stoichCoeff =
167                             reactions_[ri].lhs()[si].stoichCoeff;
168
169                         // Molecular weight
170                         const scalar& W =
171                             this->thermo().composition().W(i);
172
173                         // Compute the residence time of this species
174                         // (concentration / consumption)
175                         tcSpecies =
176                             mag(rhoC*Y_[i][celli]/(refRRC*W*stoichCoeff));
177
178                         // The shortest residence time is the chemical
179                         // time scale
180                         if (tcSpecies < tc)
181                         {
182                             tc = tcSpecies;
183                         }
```

```
184                     }
185                 }
186                 else if (refRRC < -SMALL)
187                 {
188                     // Loop over all species on RHS
189                     forAll(reactions_[ri].rhs(), si)
190                     {
191                         // Species index
192                         const label& i = reactions_[ri].rhs()[si].index;
193
194                         // Stoichiometric coefficient
195                         const scalar& stoichCoeff =
196                             reactions_[ri].rhs()[si].stoichCoeff;
197
198                         // Molecular weight
199                         const scalar& W = this->thermo().composition().W(i);
200
201                         // Compute the residence time of this species
202                         // (concentration / consumption)
203                         tcSpecies =
204                             mag(rhoC*Y_[i][celli]/(refRRC*W*stoichCoeff));
205
206                         // The shortest residence time is the chemical
207                         // time scale
208                         if (tcSpecies < tc)
209                         {
210                             tc = tcSpecies;
211                         }
212                     }
213                 }
214
215                 // Estimate the Kolmogorov time scale
216                 tk =
217                     sqrt(max(muC/rhoC/(epsC + SMALL), 0));
218
219                 tp =
220                     max(tkeC/(epsC + SMALL), 0);
221
222                 // Compute mixing time scale
223                 tm = sqrt(tk*tp);
224
225                 // If the mixing time scale is very short or
226                 // the chemical time scale very long, assume
227                 // perfect mixing. Otherwise, use partial mixing.
228                 if (tm > SMALL && tc < GREAT)
229                 {
230                     // Reaction rate scaling factor, a.k.a. the
231                     // volume fraction of reacting fine structures
232                     kappa = tc/(tc + tm);
233                 }
234                 else
235                 {
236                     // Perfect mixing
237                     kappa = 1.0;
238                 }
239                 // Update the modified production rates of involved
240                 // species by first looping over LHS species
241                 // and then over RHS species
242                 forAll(reactions_[ri].lhs(), si)
243                 {
244                     // Species index
245                     const label& i = reactions_[ri].lhs()[si].index;
246
247                     // Stoichiometric coefficient
248                     const scalar& stoichCoeff =
249                         reactions_[ri].lhs()[si].stoichCoeff;
250
251                     // Molecular weight
```

```
252                     const scalar& W = this->thermo().composition().W(i);
253
254                     // Compute the production rate of the species
255                     // and scale it with kappa. Add it to the total
256                     // modified production rate of the species.
257                     // Note that a positive refRRC means that
258                     // species on the LHS are consumed.
259                     modRR_[i][celli] -=
260                         kappa*refRRC*W*stoichCoeff;
261                 }
262                 forAll(reactions_[ri].rhs(), si)
263                 {
264                     // Species index
265                     const label& i = reactions_[ri].rhs()[si].index;
266
267                     // Stoichiometric coefficient
268                     const scalar& stoichCoeff =
269                         reactions_[ri].rhs()[si].stoichCoeff;
270
271                     // Molecular weight
272                     const scalar& W = this->thermo().composition().W(i);
273
274                     // Compute the production rate of the species
275                     // and scale it with kappa. Add it to the total
276                     // modified production rate of the species.
277                     // Note that a positive refRRC means that
278                     // species on the RHS are produced.
279                     modRR_[i][celli] +=
280                         kappa*refRRC*W*stoichCoeff;
281                 }
282             }
283         }
284
285         // Overwrite previously computed production rates with modified
286         // production rates
287         forAll(modRR_, i)
288         {
289             this->chemistryPtr_->RR(i) = modRR_[i];
290         }
291     }
292 }
293
294
295 template<class ReactionThermo>
296 Foam::tmp<Foam::fvScalarMatrix>
297 Foam::combustionModels::foxPaSR<ReactionThermo>::R(volScalarField& Y) const
298 {
299     return laminar<ReactionThermo>::R(Y);
300 }
301
302
303 template<class ReactionThermo>
304 Foam::tmp<Foam::volScalarField>
305 Foam::combustionModels::foxPaSR<ReactionThermo>::Qdot() const
306 {
307     return tmp<volScalarField>
308     (
309         new volScalarField
310         (
311             this->thermo().phasePropertyName(typeName + ":Qdot"),
312             laminar<ReactionThermo>::Qdot()
313         )
314     );
315 }
316
317
318 template<class ReactionThermo>
319 bool Foam::combustionModels::foxPaSR<ReactionThermo>::read()
```

```
{
    if (laminar<ReactionThermo>::read())
    {
        this->coeffs().readEntry("Cmix", Cmix_);
        return true;
    }

    return false;
}

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

foxPaSR.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2017 OpenFOAM Foundation
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Class
    Foam::combustionModels::foxPaSR

Group
    grpCombustionModels

Description
    Partially stirred reactor turbulent combustion model with individual
    reaction scaling.

    This model calculates finite rates based on both turbulent and chemical
    time scales.  Depending on mesh resolution, the Cmix parameter can be used
    to adjust the turbulent mixing time scale. A separate chemical time scale
    is computed for each reaction, and each reaction is scaled separately.
    The turbulent mixing time scale is a harmonic average between the
    Kolmogorov time scale and the time scale of sub-grid velocity stretch.

SourceFiles
    foxPaSR.C

\*---------------------------------------------------------------------------*/

#ifndef foxPaSR_H
#define foxPaSR_H

#include "laminar.H"
#include "thermoPhysicsTypes.H"
#include "StandardChemistryModel.H"

```

```
55 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
56
57 namespace Foam
58 {
59 namespace combustionModels
60 {
61
62 /*---------------------------------------------------------------------------*\
63                           Class foxPaSR Declaration
64 \*---------------------------------------------------------------------------*/
65
66 template<class ReactionThermo>
67 class foxPaSR
68 :
69     public laminar<ReactionThermo>
70 {
71     // Private data
72
73         //- Mixing parameter
74         scalar Cmix_;
75
76         //- Number of species
77         //label nSpecie_;
78
79         //- Species mass fractions
80         const PtrList<volScalarField>& Y_;
81
82         //- List of reactions
83         const PtrList<Reaction<gasHThermoPhysics>>& reactions_;
84
85         //- Scaling factors
86         PtrList<volScalarField> modRR_;
87
88     // Private Member Functions
89
90         //- No copy construct
91         foxPaSR(const foxPaSR&) = delete;
92
93         //- No copy assignment
94         void operator=(const foxPaSR&) = delete;
95
96
97 public:
98
99     //- Runtime type information
100    TypeName("foxPaSR");
101
102
103    // Constructors
104
105        //- Construct from components
106        foxPaSR
107        (
108            const word& modelType,
109            ReactionThermo& thermo,
110            const compressibleTurbulenceModel& turb,
111            const word& combustionProperties
112        );
113
114
115    //- Destructor
116    virtual ~foxPaSR();
117
118
119    // Member Functions
120
121        //- Correct combustion rate
122        virtual void correct();
```

```
123
124         //- Fuel consumption rate matrix
125         virtual tmp<fvScalarMatrix> R(volScalarField& Y) const;
126
127         //- Heat release rate [kg/m/s3]
128         virtual tmp<volScalarField> Qdot() const;
129
130         //- Update properties from given dictionary
131         virtual bool read();
132 };
133
134
135 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
136
137 } // End namespace combustionModels
138 } // End namespace Foam
139
140 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
141
142 #ifdef NoRepository
143     #include "foxPaSR.C"
144 #endif
145
146 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
147
148 #endif
149
150 // ************************************************************************* //
```

foxPaSRs.C

```
1  /*---------------------------------------------------------------------------*\
2    =========                 |
3    \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
4     \\    /   O peration     |
5      \\  /    A nd           | www.openfoam.com
6       \\/     M anipulation  |
7  -------------------------------------------------------------------------------
8      Copyright (C) 2011-2017 OpenFOAM Foundation
9  -------------------------------------------------------------------------------
10 License
11     This file is part of OpenFOAM.
12
13     OpenFOAM is free software: you can redistribute it and/or modify it
14     under the terms of the GNU General Public License as published by
15     the Free Software Foundation, either version 3 of the License, or
16     (at your option) any later version.
17
18     OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
19     ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
20     FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
21     for more details.
22
23     You should have received a copy of the GNU General Public License
24     along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.
25
26 \*---------------------------------------------------------------------------*/
27
28 #include "makeCombustionTypes.H"
29
30 #include "psiReactionThermo.H"
31 #include "rhoReactionThermo.H"
32 #include "foxPaSR.H"
33
34 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
35
36 namespace Foam
37 {
```

```
38
39 makeCombustionTypes(foxPaSR, psiReactionThermo);
40 makeCombustionTypes(foxPaSR, rhoReactionThermo);
41
42 }
43
44 // ************************************************************************* //
```

Appendix B

Test case dictionaries

B.1 Allrun and Allclean scripts

Allrun

```
#!/bin/sh
cd "${0%/*}" || exit                                # Run from this directory
. ${WM_PROJECT_DIR:?}/bin/tools/RunFunctions        # Tutorial run functions
#------------------------------------------------------------------------------

blockMesh > log.blockMesh

decomposePar -force -time 0 > log.decomposePar

mpirun -np 8 reactingFoam -parallel > log.initialRun

reconstructPar -time 0.02 > log.reconstructPar

sed -i s/"startTime       0;"/"startTime       0.02;"/g system/controlDict

setFields > log.setFields

decomposePar -force -time 0,0.02 >> log.decomposePar

sed -i s/"endTime         0.02;"/"endTime         0.05;"/g system/controlDict

mpirun -np 8 reactingFoam -parallel > log.ignitionRun

reconstructPar

#------------------------------------------------------------------------------
```

Allclean

```
#!/bin/sh
cd "${0%/*}" || exit                                # Run from this directory
. ${WM_PROJECT_DIR:?}/bin/tools/CleanFunctions      # Tutorial clean functions
#------------------------------------------------------------------------------

rm -r constant/polyMesh
rm -r 0.*
rm -r [123456789]*
rm -r processor*
rm log.*

sed -i s/"startTime       0.02;"/"startTime       0;"/g system/controlDict

sed -i s/"endTime         0.05;"/"endTime         0.02;"/g system/controlDict

```

```
#--------------------------------------------------------------------------
```

B.2 0 directory

0/alphat

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      alphat;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -1 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            zeroGradient;
    }

    upperWall
    {
        type            compressible::alphatWallFunction;
        Prt             0.85;
        value           uniform 0;
    }

    lowerWall
    {
        type            compressible::alphatWallFunction;
        Prt             0.85;
        value           uniform 0;
    }

    frontAndBack
    {
        type            compressible::alphatWallFunction;
        Prt             0.85;
        value           uniform 0;
    }
}


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

0/C7H16

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      C7H16;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.0474;
    }

    outlet
    {
        type            zeroGradient;
    }

    upperWall
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            zeroGradient;
    }
}


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

0/k

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      k;
```

```
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -2 0 0 0 0];

internalField   uniform 2e-5;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 2e-5;
    }

    outlet
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }

    upperWall
    {
        type            kqRWallFunction;
        value           uniform 0;
    }

    lowerWall
    {
        type            kqRWallFunction;
        value           uniform 0;
    }

    frontAndBack
    {
        type            kqRWallFunction;
        value           uniform 0;
    }
}


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

0/N2

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      N2;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0.767;

boundaryField
{
```

```
    inlet
    {
        type            fixedValue;
        value           uniform 0.2220;
    }

    outlet
    {
        type            zeroGradient;
    }

    upperWall
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            zeroGradient;
    }
}


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

0/nut

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      nut;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 2 -1 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            zeroGradient;
    }

    upperWall
    {
        type            nutkWallFunction;
        value           uniform 0;
```

```
    }

    lowerWall
    {
        type            nutkWallFunction;
        value           uniform 0;
    }

    frontAndBack
    {
        type            nutkWallFunction;
        value           uniform 0;
    }
}


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

0/O2

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      O2;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0.233;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0.2220;
    }

    outlet
    {
        type            zeroGradient;
    }

    upperWall
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            zeroGradient;
    }
}
```

```

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

0/p

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      p;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [1 -1 -2 0 0 0 0];

internalField   uniform 1e05;

boundaryField
{
    inlet
    {
        type            zeroGradient;
    }

    outlet
    {
        type            fixedValue;
        value           uniform 1e05;
    }

    upperWall
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            zeroGradient;
    }
}


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

0/T

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
```

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      T;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 400;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 400;
    }

    outlet
    {
        type            zeroGradient;
    }

    upperWall
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            zeroGradient;
    }
}


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

0/U

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volVectorField;
    object      U;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 1 -1 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
```

```
{
    inlet
    {
        type            fixedValue;
        value           uniform (20 0 0);
    }

    outlet
    {
        type            inletOutlet;
        inletValue      uniform (0 0 0);
        value           uniform (0 0 0);
    }

    upperWall
    {
        type            noSlip;
    }

    lowerWall
    {
        type            noSlip;
    }

    frontAndBack
    {
        type            noSlip;
    }
}


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

0/nut

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      Ydefault;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    inlet
    {
        type            fixedValue;
        value           uniform 0;
    }

    outlet
    {
        type            zeroGradient;
    }

```

```
    upperWall
    {
        type            zeroGradient;
    }

    lowerWall
    {
        type            zeroGradient;
    }

    frontAndBack
    {
        type            zeroGradient;
    }
}


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

B.3 chemkin directory

chemkin/chem.inp

```
ELEMENTS
 H   O    C   N   AR
END
SPECIE
C7H16 O2 N2 CO2 H2O
END
REACTIONS
 C7H16 + 11O2           => 7CO2 + 8H2O        5.00E+8  0.0   15780.0! 1
    FORD    / C7H16 0.25 /
    FORD    / O2 1.5 /
END
```

chemkin/therm.dat

```
THERMO ALL
   200.000  1000.000  6000.000
C7H16             P10/85C  7.H 16.   0.   0.G   200.000  6000.000 1000.        1
 2.04565203E+01 3.48575357E-02-1.09226846E-05 1.67201776E-09-9.81024850E-14    2
-3.25556365E+04-8.04405017E+01 1.11532994E+01-9.49419773E-03 1.95572075E-04    3
-2.49753662E-07 9.84877715E-11-2.67688904E+04-1.59096837E+01-2.25846141E+04    4
O2                ATcT06O  2.   0.   0.   0.G   200.000  6000.000 1000.        1
 3.45852381E+00 1.04045351E-03-2.79664041E-07 3.11439672E-11-8.55656058E-16    2
 1.02229063E+04 4.15264119E+00 3.78535371E+00-3.21928540E-03 1.12323443E-05    3
-1.17254068E-08 4.17659585E-12 1.02922572E+04 3.27320239E+00 1.13558105E+04    4
N2                G 8/02N  2.   0.   0.   0.G   200.000  6000.000 1000.        1
 2.95257637E+00 1.39690040E-03-4.92631603E-07 7.86010195E-11-4.60755204E-15    2
-9.23948688E+02 5.87188762E+00 3.53100528E+00-1.23660988E-04-5.02999433E-07    3
 2.43530612E-09-1.40881235E-12-1.04697628E+03 2.96747038E+00 0.00000000E+00    4
CO2               L 7/88C   1O   2    0    0G   200.000  6000.000 1000.        1
 0.46365111E+01 0.27414569E-02-0.99589759E-06 0.16038666E-09-0.91619857E-14    2
-0.49024904E+05-0.19348955E+01 0.23568130E+01 0.89841299E-02-0.71220632E-05    3
 0.24573008E-08-0.14288548E-12-0.48371971E+05 0.99009035E+01-0.47328105E+05    4
H2O               L 5/89H   2O   1    0    0G   200.000  6000.000 1000.        1
 0.26770389E+01 0.29731816E-02-0.77376889E-06 0.94433514E-10-0.42689991E-14    2
-0.29885894E+05 0.68825500E+01 0.41986352E+01-0.20364017E-02 0.65203416E-05    3
-0.54879269E-08 0.17719680E-11-0.30293726E+05-0.84900901E+00-0.29084817E+05    4
END
```

chemkin/transportProperties

```
/*---------------------------------*- C++ -*----------------------------------*\
```

```
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "chemkin";
    object      transportProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

".*"
{
    transport
    {
        As 1.67212e-6;
        Ts 170.672;
    }
}

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

B.4 constant directory

constant/chemistryProperties

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      chemistryProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

chemistryType
{
    solver            ode;
}

chemistry       on;

initialChemicalTimeStep 1e-07;

odeCoeffs
{
    solver          seulex;
    eps             0.05;
}


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

constant/combustionProperties

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      combustionProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

combustionModel foxPaSR;
active          yes;

foxPaSRCoeffs
{
    Cmix        1.0;
}


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

constant/radiationProperties

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      radiationProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

radiation       off;

radiationModel  none;


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

constant/thermophysicalProperties

```
/*---------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
```

```
    object      thermophysicalProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

thermoType
{
    type            hePsiThermo;
    mixture         reactingMixture;
    transport       sutherland;
    thermo          janaf;
    energy          sensibleEnthalpy;
    equationOfState perfectGas;
    specie          specie;
}

CHEMKINFile         "<case>/chemkin/chem.inp";
CHEMKINThermoFile   "<case>/chemkin/therm.dat";
CHEMKINTransportFile "<case>/chemkin/transportProperties";

newFormat       yes;

inertSpecie     N2;


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

constant/turbulenceProperties

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType  LES;

LES
{
    LESModel        dynamicKEqn;

    turbulence      on;

    printCoeffs     on;

    delta           cubeRootVol;

    dynamicKEqnCoeffs
    {
        filter simple;
    }

    cubeRootVolCoeffs
    {
        deltaCoeff      1;
    }

    PrandtlCoeffs
    {
```

```
        delta           cubeRootVol;
        cubeRootVolCoeffs
        {
            deltaCoeff      1;
        }

        smoothCoeffs
        {
            delta           cubeRootVol;
            cubeRootVolCoeffs
            {
                deltaCoeff      1;
            }

            maxDeltaRatio   1.1;
        }

        Cdelta          0.158;
    }

    vanDriestCoeffs
    {
        delta           cubeRootVol;
        cubeRootVolCoeffs
        {
            deltaCoeff      1;
        }

        smoothCoeffs
        {
            delta           cubeRootVol;
            cubeRootVolCoeffs
            {
                deltaCoeff      1;
            }

            maxDeltaRatio   1.1;
        }

        Aplus           26;
        Cdelta          0.158;
    }

    smoothCoeffs
    {
        delta           cubeRootVol;
        cubeRootVolCoeffs
        {
            deltaCoeff      1;
        }

        maxDeltaRatio   1.1;
    }
}


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

B.5 system directory

system/blockMeshDict

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

```
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

scale   0.001;

vertices
(
    (-20.6 0 -25)
    (-20.6 25.4 -25)
    (0 -25.4 -25)
    (0 0 -25)
    (0 25.4 -25)
    (206 -25.4 -25)
    (206 0 -25)
    (206 25.4 -25)
    (290 -16.6 -25)
    (290 0 -25)
    (290 16.6 -25)

    (-20.6 0 25)
    (-20.6 25.4 25)
    (0 -25.4 25)
    (0 0 25)
    (0 25.4 25)
    (206 -25.4 25)
    (206 0 25)
    (206 25.4 25)
    (290 -16.6 25)
    (290 0 25)
    (290 16.6 25)
);

negY
(
    (2 4 1)
    (1 3 0.3)
);

posY
(
    (1 4 2)
    (2 3 4)
    (2 4 0.25)
);

posYR
(
    (2 1 1)
    (1 1 0.25)
);


blocks
(
    hex (0 3 4 1 11 14 15 12)
    (18 30 25)
    simpleGrading (1 1 1)

```

```
    hex (2 5 6 3 13 16 17 14)
    (180 27 25)
    simpleGrading (1 1 1)

    hex (3 6 7 4 14 17 18 15)
    (180 30 25)
    simpleGrading (1 1 1)

    hex (5 8 9 6 16 19 20 17)
    (50 27 25)
    simpleGrading (1 1 1)

    hex (6 9 10 7 17 20 21 18)
    (50 30 25)
    simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    inlet
    {
        type patch;
        faces
        (
            (0 1 12 11)
        );
    }
    outlet
    {
        type patch;
        faces
        (
            (8 9 20 19)
            (9 10 21 20)
        );
    }
    upperWall
    {
        type wall;
        faces
        (
            (1 4 15 12)
            (4 7 18 15)
            (7 10 21 18)
        );
    }
    lowerWall
    {
        type wall;
        faces
        (
            (0 3 14 11)
            (3 2 13 14)
            (2 5 16 13)
            (5 8 19 16)
        );
    }
    frontAndBack
    {
        type wall;
        faces
        (
            (0 3 4 1)
            (2 5 6 3)
```

```
            (3 6 7 4)
            (5 8 9 6)
            (6 9 10 7)
            (11 14 15 12)
            (13 16 17 14)
            (14 17 18 15)
            (16 19 20 17)
            (17 20 21 18)
        );
    }
);


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

system/controlDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

application     reactingFoam;

startFrom       startTime;

startTime       0;

stopAt          endTime;

endTime         0.02;

deltaT          1e-05;

writeControl    timeStep;

writeInterval   500;

purgeWrite      0;

writeFormat     ascii;

writePrecision  6;

writeCompression off;

timeFormat      general;

timePrecision   6;

runTimeModifiable true;

libs ( "libFOXcombustionModels" );


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

system/decomposeParDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      decomposeParDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

numberOfSubdomains 8;

method          simple;

coeffs
{
    n           (8 1 1);
}


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

system/fvSchemes

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         leastSquares;
}

divSchemes
{
    default         none;
    div(phi,U)      Gauss GammaV 0.5;
    div(phi,h)      Gauss Gamma 0.5;
    div(phi,K)      Gauss Gamma 0.5;
    div(phiv,p)     Gauss Gamma 0.5;
    div(phi,k)      Gauss Minmod; // Gamma 0.1;
    div(phi,Yi_h)   Gauss Gamma 0.5;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
```

```
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

wallDist
{
  method meshWave;
}


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

system/fvSolution

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

solvers
{
    "(p|rho)"
    {
        solver          PBiCGStab;
        preconditioner  DIC;
        tolerance       1e-7;
        relTol          0.001;
    }

    "(p|rho)Final"
    {
        $p;
        relTol          0;
    }

    "(U|h|k|Yi)"
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-7;
        relTol          0.001;
    }

    "(U|h|k|Yi)Final"
```

```
    {
        $U;
        relTol          0;
    }
}

PIMPLE
{
    momentumPredictor yes;
    nOuterCorrectors 1;
    nCorrectors     3;
    nNonOrthogonalCorrectors 2;

    pMinFactor      0.75;
    pMaxFactor      1.25;
}
```

system/setFieldsDict

```
/*--------------------------------*- C++ -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Website:  www.openfoam.com                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      setFieldsDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

regions
(
    boxToCell
    {
        box (0 -0.03 -1) (0.3 0 1);
        fieldValues
        (
            volScalarFieldValue T 2000
        );
    }
);


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

Index