Cite as: Fjällborg, Ö.: Implementation of a new heat transfer model in OpenFOAM for lagrangian particle tracking solvers for use in porous media. 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

Implementation of a new heat transfer model in OpenFOAM for lagrangian particle tracking solvers for use in porous media

Developed for OpenFOAM-v2112
Requires: PyFoam

Author:
Örjan Fjällborg
Luleå University of Technology
orjan.fjallborg@lkab.com

Peer reviewed by:
Mo Adib
Saeed Salehi
Charlotte Andersson

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 15, 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 to use existing LPT heat transfer models.

The theory of it:

- the theory of heat transfer between gas and particle, and the assumptions, theory and limitations for the existing Ranz-Marshall heat transfer model and theory for the new heat transfer model.

How it is implemented:

- the general mechanism and implementation of LPT heat transfer and how the specific Ranz-Marshall heat transfer model is implemented and fits in the general code structure.

How to modify it:

- how to implement a new LPT heat transfer submodel as shown in a class diagram, and how to plug it in with a minimal footprint into an existing OpenFOAM v2112 installation.
- how to implement and plug in a new parcel or cloud type.
- how an existing tutorial case can be modified and simulated with both the existing Ranz-Marshall model and the new heat transfer model and then compared.

Prerequisites

The reader is expected to have some prior knowledge on the following in order to get maximum benefit out of this report:

- How to run standard tutorials like the damBreak tutorial.
- Fundamentals of Computational Methods for Fluid Dynamics, Book by J. H. Ferziger and M. Peric.
- How to customize a solver and do top-level application programming.
- Fundamentals on multiphase heat transfer.
- Lagrangian particle tracking.

Contents

Nomenclature

Acronyms

CFD	Computational Fluid Dynamics
LPT	Lagrangian Particle Tracking

Definitions

Particle	A solid that contains porosity
Pellet	Same as particle

English symbols

$\mathbf{q}$	Heat flux vector	$\mathrm{J/(m^2 \cdot s)}$
$\mathbf{U}$	Velocity	$\mathrm{m/s}$
Δt_e	Effective time-step	s
$\dot{Q}$	Heat rate of change from chemical reactions	$\mathrm{J/(s \cdot m^3)}$
$\dot{q}$	Heat flow	$\mathrm{J/s}$
$\dot{q}_\mathrm{Rtot}$	Total radiative heat flow	$\mathrm{J/s}$
$\dot{q}_\mathrm{R}$	Emitted radiative heat flow	$\mathrm{J/s}$
$\dot{q}_\mathrm{r}$	Chemical reaction heat flow	$\mathrm{J/s}$
A	Area	$\mathrm{m^2}$
A_pore	Pore surface area	$\mathrm{m^2}$
C_p	Specific heat capacity	$\mathrm{J/(kg \cdot K)}$
d	Diameter	m
e	Internal energy	$\mathrm{J/kg}$
h	Enthalpy	$\mathrm{J/kg}$
h	Heat transfer coefficient	$\mathrm{J/(m^2 \cdot s \cdot K)}$
K	Kinetic energy	$\mathrm{J/kg}$
k	Thermal conductivity	$\mathrm{J/(m \cdot s \cdot K)}$
L	Caracteristic length	m
q	Heat flux	$\mathrm{J/(m^2 \cdot s)}$
S_hR	Energy rate of change source term due to radiation	$\mathrm{J/(m^3 \cdot s)}$
S_h	Enthapy rate of change source term	$\mathrm{J/(m^3 \cdot s)}$
T	Temperature	K
t	Time	s
U	Linear velocity	$\mathrm{m/s}$
V	Volume	$\mathrm{m^3}$

Greek symbols

α_eff	Sum of laminar and turbulent thermal diffusivites	$\mathrm{kg/(m \cdot s)}$
$\boldsymbol{\tau}$	Viscous momentum flux	$\mathrm{J/m^3}$
ϵ	Emissivity of the surface	
ϵ_v	Void fraction in a bed	
κ	Thermal diffusivity	$\mathrm{m^2/s}$

μ	Fluid dynamic viscosity	$\mathrm{kg/(m \cdot s)}$
ν	Fluid kinematic viscosity	$\mathrm{m^2/s}$
ρ	Fluid density	$\mathrm{kg/m^3}$
σ	Stefan-Boltzmann constant	$5.67037 \times 10^{-8}\ \mathrm{Wm^{-2}K^{-4}}$

Superscripts

n	current time step
n+1	next time step

Subscripts

p	pressure
0	bulk
E	environment
e	effective
f	fluid
g	gas
p	particle
pz	particle with zero porosity
R	radiation
r	reaction
s	solid
srfc	surface

Other symbols

Bi	Biot number
Gr	Grashof number
Nu	Nusselt number
Pr	Prandlt number
Re	Reynolds number
Sc	Schmidth number
Sh	Sherwood number

Chapter 1

Introduction

1.1 Background

This report focuses on heat transfer between lagrangian particles and a continous phase. This is important in all high-temperature solid reactions, and not at least in the steel and mining industry where gas-solid reactions and heat transfer is common in porous packed beds, where the packed bed can be seen as a collection of lagrangian particles and the gas flowing through it as the continous phase. This can for example be found in the production of iron ore pellets, in the upper parts of blast furnaces or in the production of sponge iron in DR-shafts.

Iron ore pellet production consist of two major steps. The first is the balling of iron ore "green" pellets, were fine-ground iron oxide is mixed with binders and additives and agglomerated into pellets. Production of green pellets is followed by heat hardening at high temperature. The heat hardening or "induration" of pellets involves drying, preheating, firing, and cooling. During induration, the pellets are heated and exposed to oxygen-containing gas. If the iron oxide is magnetite (Fe_3O_4) an oxidation reaction and phase change occurs simultaneously as the heat hardening, and the oxidation reaction results in a conversion of magnetite to hematite (Fe_2O_3).

The heat transfer in a porous packed bed is therefore of major importance to model accurately, but OpenFOAM only has one LPT based heat transfer model, which is limited to single particles, not a packed bed of particles. Validation of the existing Ranz-Marshall model to experimental data, also shows that the heating up of a porous packed bed is too slow, so there is a need of developing a heat transfer model that is better adapted for porous packed beds. This report describes the theory and implementation of the existing heat transfer model and also shows how to extend OpenFOAM with another heat transfer model and the theory of the model.

1.1.1 Selection of solver

Since a packed porous bed can be seen as a collection of particles with some degree of void between then, selecting the lagrangian solver `reactingHeterogenousParcelFoam` which also contains turbulence, transient or steady state operation, mass and heat transport between the phases, compositions and chemical reactions is a good starting point. Compared to other LPT solvers (`reactingParcelFoam`, `simpleReactingParcelFoam`, `coalChemistryFoam`, `simpleCoalParcelFoam` or `sprayFoam`) that contains an energy equation, `reactingHeterogenousParcelFoam` is more complete. The other solvers either only works in steady state mode or lack some terms in the energy equation or the possibility for simulating heterogenous reactions.

All of the solvers does, however, contain the term ρr found in the energy equations Eq. (2.33) and Eq. (2.35) which makes it possible to use a new heat transfer model in all of the solvers.

Chapter 2

Theory

There are three types of heat transfer in a solid particle - gas phase system. The most dominant type is heat convection between a gas and a particle, which can be determined by the dimensionless Nusselt number which compares convection to conduction in the gas. The Nusselt number is also often empirically correlated to the Reynolds number and the Prandlt number and can be used to determine the convective heat transfer coefficient.

The second type is thermal radiation between the particle and the environment. This mechanism gets higher importance at higher temperatures.

And the third type is heat conduction in the gas or in the particle.

Additional theory for heat transfer in a porous packed bed is described in Section 2.7. Note that, the only difference between a single particle and a porous packed bed is that considered heat transfer models only calculates different values for the convective heat transfer coefficient h. All other theory applies to both cases.

2.1 Convective heat transfer

Let us consider a solid particle with a surface temperature (and average temperature) of T_{srfc} and moving gas stream with a mean bulk temperature of T_{g} then the convective heat flow $\dot{q}$ is given by

$$\dot{q} = hA_{\mathrm{srfc}}(T_{\mathrm{g}} - T_{\mathrm{srfc}}). \tag{2.1}$$

Here, h is a convective heat transfer coefficient and A_{srfc} is the surface area. Heat transfer is analogous to mass transfer, and Eq. (2.1) resembles very much the equation for convective mass transfer. The Nusselt number is defined as the relative importance of convection to conduction (or external to internal heat transfer) and is given by

$$\mathrm{Nu} = hL/k_{\mathrm{g}}, \tag{2.2}$$

and the Nusselt number is also often empirically set from correlations of the form

$$\mathrm{Nu} = f(\mathrm{Re}, \mathrm{Pr}), \tag{2.3}$$

for forced convection, where the Reynolds number is given by

$$\mathrm{Re} = UL/\nu, \tag{2.4}$$

and the Prandtl number by

$$\mathrm{Pr} = C_{\mathrm{p}}\mu/k_{\mathrm{g}}, \tag{2.5}$$

where k_{g} is the thermal conductivity of the gas, L the characteristic length (or diameter of the particle), U the magnitude of the velocity, ν the kinematic viscosity, μ the dynamic viscosity and C_{p} the heat capacity.

These correlations are very much like the mass transfer correlations with the difference that the Sherwood number is replaced with the Nusselt number (Nu), and the Schmidth number is replaced with the Prandtl number (Pr). This gives the Ranz-Marshall correlation for heat transfer

$$\mathrm{Nu} = 2.0 + 0.6\mathrm{Re}^{1/2}\mathrm{Pr}^{1/3}. \tag{2.6}$$

2.2 Thermal radiation

At high temperatures, thermal radiation may be an important mechanism for heat transfer. The rate a body emits radiation is given by the Stefan-Boltzmann law, as

$$\dot{q}_\mathrm{R} = A_\mathrm{srfc}\epsilon\sigma T_\mathrm{srfc}^4, \tag{2.7}$$

where $\dot{q}_\mathrm{R}$ is the radiant heat flux, ϵ is the emissivity of the surface, σ is the Stefan-Boltzmann constant and T_srfc the absolute surface temperature.

For a single particle in a large cavity, an approximate expression for the net received radiative flux $\dot{q}_\mathrm{r}'$ at the solid surface is given by

$$\dot{q}_\mathrm{R}' = A_\mathrm{srfc}\epsilon\sigma(T_\mathrm{E}^4 - T_\mathrm{srfc}^4), \tag{2.8}$$

where T_E is the temperature of the environment.

To compare radiative heat transfer and convective heat transfer, a radiation heat transfer coefficient can be defined as

$$h_\mathrm{R} = \epsilon\sigma(T_\mathrm{E}^2 + T_\mathrm{srfc}^2)(T_\mathrm{E} + T_\mathrm{srfc}), \tag{2.9}$$

and used

$$\dot{q}_\mathrm{R}' = A_\mathrm{srfc}h_\mathrm{R}(T_\mathrm{E} - T_\mathrm{srfc}). \tag{2.10}$$

2.3 Heat conduction in a porous particle

Fourier's law describes the conduction of heat, which is proportional to the temperature gradient as

$$\mathbf{q} = -k\nabla T. \tag{2.11}$$

Heat conduction in porous particles is less well-known. One approximation is to use an effective conductivity, given by

$$k_\mathrm{e} = (1 - \epsilon_\mathrm{p})k_\mathrm{pz}, \tag{2.12}$$

where k_e is the effective conductivity and k_pz is the conductivity for the particle at zero porosity, and ϵ_p is the porosity of the particle.

For Eq. (2.12) to be valid, the particle needs to be continuous with closed and isolated pores. Thermal conductivity of the gas in the pores needs also to be much lower than in the solid phase. If the particle consists of loosely packed grains instead, then heat conduction in the pore gas gives most resistance to heat conduction, because that only a small part of the heat flow is conducted through the contact points of the solid grains. The magnitude of the conduction in the contact points is dependent on surface roughness and other geometrical details which makes it is very difficult to predict the effective thermal conductivity under these conditions and measuring of it might be the safest to do. If the loosed packed particle is sintered then Eq. (2.12) gets more valid again.

2.4 Combined conduction and convection

The general unsteady case for conduction in a particle with chemical reaction is given by Fourier's second law, which stated

$$\frac{\partial T}{\partial t} = \nabla \cdot \kappa_\mathrm{e}\nabla T + \frac{\dot{Q}}{(1 - \epsilon_\mathrm{p})\rho_\mathrm{pz}C_\mathrm{p}}, \tag{2.13}$$

where κ_e is effective thermal diffusivity, $\dot{Q}$ is rate of heat generated from chemical reaction, ρ_pz density of the particle with zero porosity and C_p the specific heat capacity. The convective heat transfer out of the solid is described by a boundary condition as

$$q = -\left|k_\mathrm{e}\nabla T\right| = h(T_\mathrm{srfc} - T_0), \tag{2.14}$$

at the particle surface where q is the convective heat flux. The Biot number compares heat convection and conduction relatively (or external and internal heat transfer) as

$$\mathrm{Bi} = hL/k_\mathrm{e}. \tag{2.15}$$

When the Biot number is < 0.2, then conduction within the particle can be ignored [1], and the convective boundary condition Eq. (2.14) can replace the conduction term in Eq. (2.13). By using the relation between thermal diffusivity and thermal conductivity given by

$$\kappa_\mathrm{e} = \frac{k_\mathrm{e}}{\rho_\mathrm{p} C_\mathrm{p}}, \tag{2.16}$$

and multiplying the equation Eq. (2.13) with the particle volume V_p, we can do the following derivation

$$\begin{aligned}
V_\mathrm{p}\frac{\partial T}{\partial t} &= V_\mathrm{p}\nabla\kappa_e\nabla T + V_\mathrm{p}\frac{Q}{(1-\epsilon_\mathrm{p})\rho_\mathrm{pz}C_p} \\
V_\mathrm{p}\rho_\mathrm{p}(1-\epsilon_\mathrm{p})C_p\frac{\partial T}{\partial t} &= V_\mathrm{p}Q + V_\mathrm{p}\rho_\mathrm{p}(1-\epsilon_\mathrm{p})C_p\nabla\kappa_e\nabla T \\
&= V_\mathrm{p}Q + V_\mathrm{p}\nabla k_e\nabla T && \text{by use of Eq. (2.16)} \\
&= V_\mathrm{p}Q - V_\mathrm{p}\nabla h(T_\mathrm{p} - T_0) && \text{by use of Eq. (2.14)} \\
&= V_\mathrm{p}Q + V_\mathrm{p}\nabla h(T_0 - T_\mathrm{p}) \\
&= V_\mathrm{p}Q + hA_\mathrm{pore}(T_0 - T_\mathrm{p}),
\end{aligned} \tag{2.17}$$

where T_p is the uniform temperature of the particle and A_pore is the pore surface area of the particle. If Eq. (2.17) is integrated, it can be used for calculating the time for a particle to reach a certain temperature.

2.5 Total heat transfer without particle conduction

When the Biot number is below 0.2 [1], the particle internal heat conduction can be neglected. The temperature at the surface is then assumed to be the same as the internal temperature and T_p is used instead of T_srfc in this section. For the other mechanisms, an energy balance with a transient heat change for the particles internal energy and convective heat transfer, total radiative heat transfer $\dot{q}_{\mathrm{R_{tot}}}$ and a chemical reaction enthalpy term $\dot{q}_\mathrm{r}$ is given by

$$mC_p\frac{\mathrm{d}T}{\mathrm{d}t} = \underbrace{hA_\mathrm{srfc}(T_\mathrm{g} - T_\mathrm{p})}_{\dot{q}} + \underbrace{A_\mathrm{srfc}\epsilon\left(\frac{G}{4} - \sigma T_\mathrm{p}^4\right)}_{\dot{q}_{\mathrm{R_{tot}}}} + \dot{q}_\mathrm{r}. \tag{2.18}$$

The first order differential equation Eq. (2.18) can be written in the general form as

$$\frac{\mathrm{d}\phi}{\mathrm{d}t} = A - B\phi, \tag{2.19}$$

where in this case

$$B = \frac{hA_\mathrm{srfc}}{mC_p}, \tag{2.20}$$

$$A = B\cdot T_\mathrm{g} + \frac{A_\mathrm{srfc}\epsilon\left(\frac{G}{4} - \sigma T_\mathrm{p}^4\right) + \dot{q}_\mathrm{r}}{mC_p}, \tag{2.21}$$

and

$$\phi = T_\mathrm{p}. \tag{2.22}$$

To calculate a new temperature T_p^{n+1} for the particle an effective time-step Δt_e is used with the old temperature T^n to calculate a delta temperature which is added to the old temperature. These steps are given by

$$\Delta t_\mathrm{e} = f(\Delta t, B), \tag{2.23}$$
$$\Delta T_\mathrm{p} = (A - B \cdot T_\mathrm{p}^n)\Delta t_\mathrm{e} \tag{2.24}$$

and

$$T_\mathrm{p}^{n+1} = T_\mathrm{p}^n + \Delta T_\mathrm{p}. \tag{2.25}$$

2.5.1 The analytical solution transformed to an effective time-step

If Eq. (2.18) is solved analytically the following solution is found

$$T_\mathrm{p} = \frac{Ce^{-Bt} - A}{-B}, \tag{2.26}$$

where C is the constant of integration. By stating an expression for ΔT

$$\Delta T_\mathrm{p} = T_\mathrm{p}^{n+1} - T_\mathrm{p}^n, \tag{2.27}$$

and plugging in the solution in Eq. (2.26)

$$\Delta T_\mathrm{p} = \frac{Ce^{-Bt^{n+1}} - A}{-B} - \frac{Ce^{-Bt^n} - A}{-B}$$

we get the intermediate result

$$-\Delta T_\mathrm{p} \cdot B = Ce^{-Bt^{n+1}} - Ce^{-Bt^n}, \tag{2.28}$$

which can be divided by Ce^{-Bt^n} and transformed with a logarithm law, giving

$$\frac{-\Delta T_\mathrm{p} \cdot B}{Ce^{-Bt^n}} = Ce^{-B\Delta t} - 1, \tag{2.29}$$

which can be rearranged with use of Eq. (2.26), where Ce^{-Bt^n} is replaced with $-BT^n + A$, giving

$$\Delta T_\mathrm{p} = (A - B \cdot T_\mathrm{p}^n) \underbrace{\frac{1 - Ce^{-B\Delta t}}{B}}_{\Delta t_\mathrm{e}}. \tag{2.30}$$

In equation Eq. (2.30) the effective time-step Δt_e have been extracted for the analytical case if $C = 1$.

2.5.2 The Eulerian solution transformed to an effective time-step

Solving Eq. (2.18) (where $T = \phi$) with the implicit Eulerian method (central differencing) gives

$$\frac{T_\mathrm{p}^{n+1} - T_\mathrm{p}^n}{\Delta t} = A - BT_\mathrm{p}^{n+1}, \tag{2.31}$$

and by transforming it with some algebra

$$\begin{aligned} T_{\mathrm{p}}^{n+1} - T_{\mathrm{p}}^{n} &= \Delta t(A - BT_{\mathrm{p}}^{n+1}) \\ T_{\mathrm{p}}^{n+1} - T_{\mathrm{p}}^{n} - B\Delta t T_{\mathrm{p}}^{n} &= \Delta t(A - BT_{\mathrm{p}}^{n+1}) - B\Delta t T_{\mathrm{p}}^{n} \\ T_{\mathrm{p}}^{n+1} - T_{\mathrm{p}}^{n} - B\Delta t T_{\mathrm{p}}^{n} + \Delta t B T_{\mathrm{p}}^{n+1} &= \Delta t A - B\Delta t T_{\mathrm{p}}^{n} \\ (T_{\mathrm{p}}^{n+1} - T_{\mathrm{p}}^{n})(1 + B\Delta t) &= \Delta t(A - BT_{\mathrm{p}}^{n}) \end{aligned}$$

we get

$$\Delta T_{\mathrm{p}} = (A - BT_{\mathrm{p}}^{n}) \underbrace{\frac{\Delta t}{1 + B\Delta t}}_{\Delta t_{\mathrm{e}}}. \tag{2.32}$$

2.6 Heat transfer in the energy equations

In the energy equations for the Eulerian-Lagrangian formulation, the heat transfer between the Lagrangian particles and the continous Eulerian phase are given by the term ρr in the equations Eq. (2.33) and Eq.(2.35). ρr can then be divided to the terms found in Eq. (2.36) and Eq. (2.37). The internal energy equation is given by

$$\frac{\partial \rho e}{\partial t} + \nabla \cdot (\rho \mathbf{U} e) + \frac{\partial \rho K}{\partial t} + \nabla \cdot (\rho \mathbf{U} K) + \nabla \cdot (\mathbf{U} p) = -\nabla \cdot \mathbf{q} + \nabla \cdot (\boldsymbol{\tau} \cdot \mathbf{U}) + \rho r + \rho \mathbf{g} \cdot \mathbf{U}, \tag{2.33}$$

where $\mathbf{q} = -\alpha_{\mathrm{eff}} \nabla e$.

By defining enthalpy as the sum of internal energy e and kinematic pressure

$$h \equiv e + p/\rho \tag{2.34}$$

gives the enthalpy energy equation in Eq. (2.35)

$$\frac{\partial \rho h}{\partial t} + \nabla \cdot (\rho \mathbf{U} h) + \frac{\partial \rho K}{\partial t} + \nabla \cdot (\rho \mathbf{U} K) - \frac{\partial p}{\partial t} = -\nabla \cdot \mathbf{q} + \nabla \cdot (\boldsymbol{\tau} \cdot \mathbf{U}) + \rho r + \rho \mathbf{g} \cdot \mathbf{U}, \tag{2.35}$$

where $\mathbf{q} = -\alpha_{\mathrm{eff}} \nabla h$, and ρr is a sum of other thermal source terms given by

$$\rho r = S_{\mathrm{h}} + S_{\mathrm{hSf}} + \dot{Q}, \tag{2.36}$$

where S_{hSf} is the energy rate of change for the particle surface film and $\dot{Q}$ is the energy rate of change due to combustion and S_{h} the energy released or absorbed by the lagrangian particle which is given by the right hand side of Eq. (2.18) converted to a volumetric source term in Eq. (2.37)

$$S_{\mathrm{h}} = \frac{\dot{q} + \dot{q}_{\mathrm{R}} + \dot{q}_{\mathrm{r}}}{V_{\mathrm{p}}}. \tag{2.37}$$

2.7 Theory for porous packed beds

2.7.1 Rowe heat transfer correlation

The Ranz-Marshal correlation Eq. (2.6) designed for single particles is not appropriate for a porous packed bed where additional problems arise, where we also must consider radiative particle-particle heat transfer and convective heat transfer between the wall and the bed. In this case the correlation proposed by Rowe *et al.* [2] in the form

$$\mathrm{Nu} = f(\mathrm{Re}, \mathrm{Pr}, \epsilon_{\mathrm{v}}), \tag{2.38}$$

where ϵ_v is the void fraction in the bed, is given by

$$\mathrm{Nu} = A + B\mathrm{Re}^n\mathrm{Pr}^{2/3}, \tag{2.39}$$

where

$$A = \frac{2}{1-(1-\epsilon_\mathrm{v})^{1/3}} \tag{2.40}$$

and

$$B = \frac{2}{3\epsilon_\mathrm{v}} \tag{2.41}$$

and where n is given by

$$\frac{2-3n}{3n-1} = \underbrace{4.65\mathrm{Re}^{-0.28}}_{\equiv\hat{R}} \tag{2.42}$$

and when n is extracted

$$n = \frac{2+\hat{R}}{3\hat{R}+3}, \tag{2.43}$$

where the Nusselt number Nu in Eq. (2.39) is defined by Eq. (2.2) with the characteristic length L set to the particle diameter d_p.

Chapter 3

Heat transfer modelling

In this chapter, the existing solver `reactingHeterogenousParcelFoam` and existing code related to LPT heat transfer will be described. By following the code in the solver and through the different cloud- and parcel templated classes, code and theory can be connected. OpenFOAM handles clouds which consists of a number of parcels. Each parcel is a calculational unit. A parcel have a decimal number that says how many particles it has. All particles in a parcel have the same properties. In Figure 3.1 the collaboration of cloud-classes is shown.

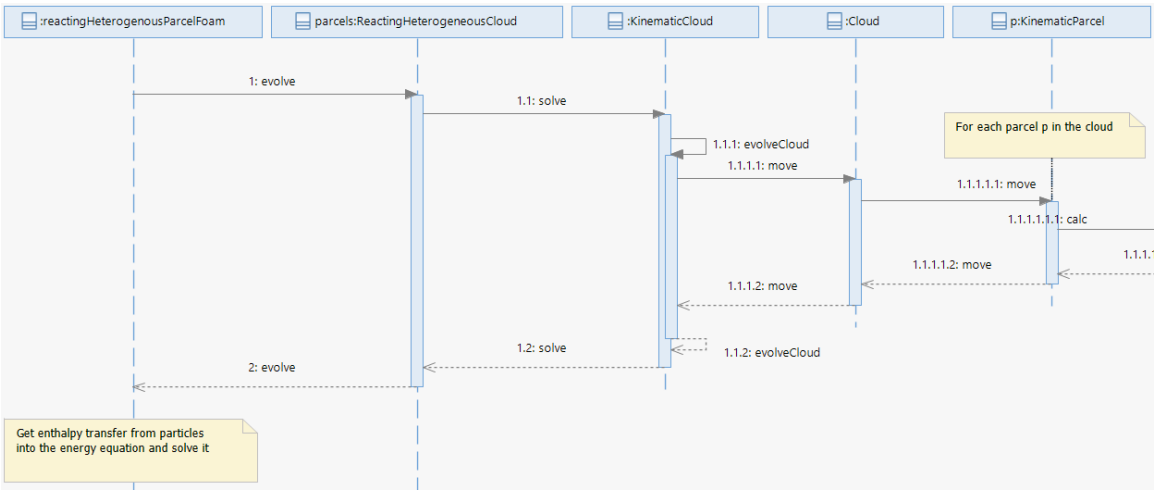

Figure 3.1: Collaboration of cloud-classes for heat transfer

In Figure 3.2, lower level heat transfer calculations in collaborating parcel-classes are shown. The result from this sequence is sent back through the calling cloud classes and into the energy equation as an enthalpy transfer variable. It is also shown in the diagram where the surface values for the particles is calculated and where the heterogeneous reaction calculations are initiated and where the heat transfer coefficient is calculated. In this case it is assumed that the Nusselt number is calculated in the Rowe subclass of `HeatTransferModel`.

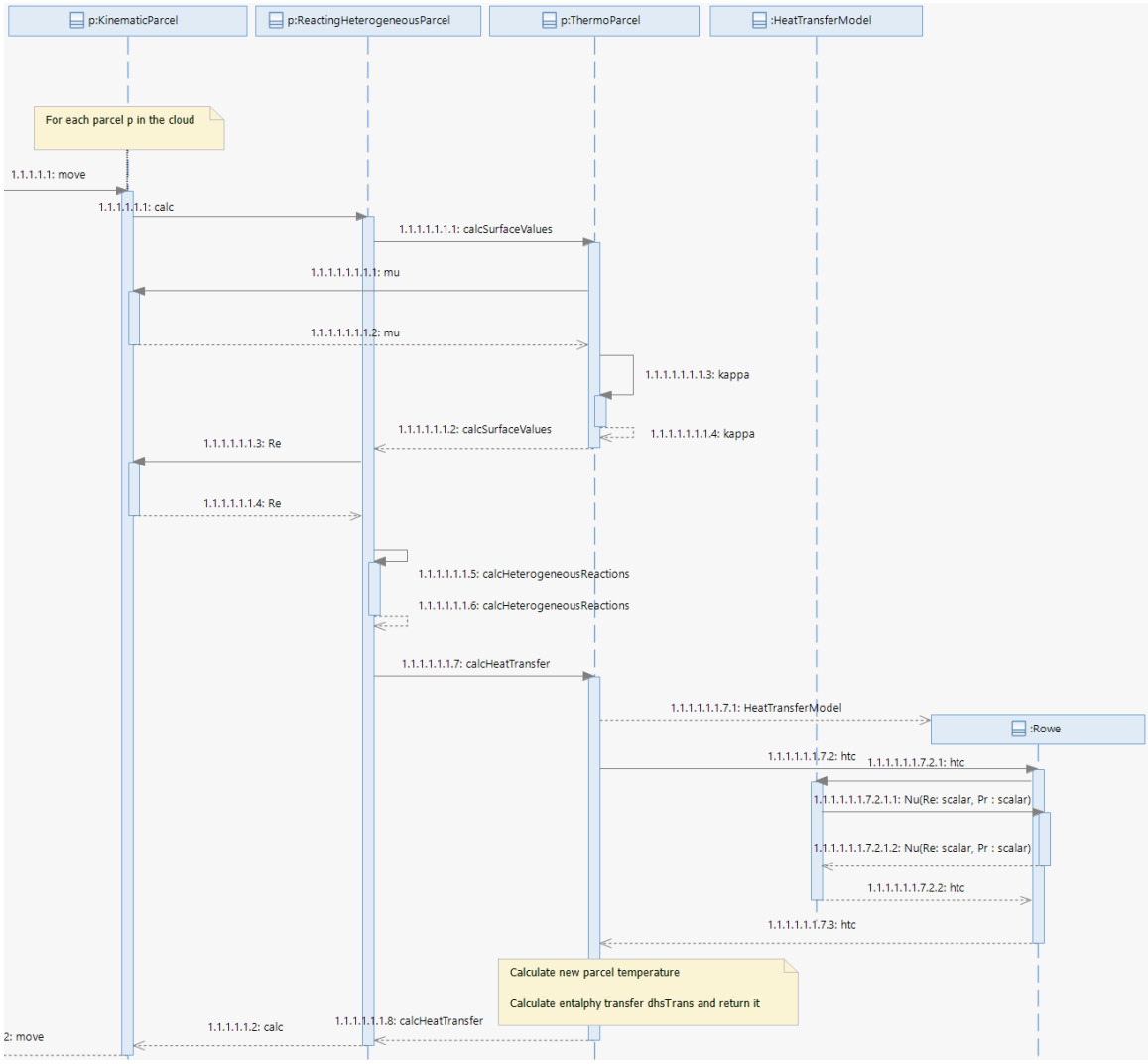

Figure 3.2: Collaboration of parcel-classes for heat transfer

The solver includes the energy equation at line 187 below, and this code path is described in Section 3.1. The code path for calculating how much mass, energy and momentum that is transferred to/from the parcels in the cloud (line 172 below) is described in Section 3.2, where the part in Section 3.2.2 handles the most of the heat transfer. The sequence diagrams shown in Figure 3.1 and in Figure 3.2, describes the calculations done before the enthalpy transfer term is used in the energy equation.

`reactingParcelFoam.C`

```
170         }
171
172         parcels.evolve();
173         surfaceFilm.evolve();
174
175         if (solvePrimaryRegion)
176         {
177             if (pimple.nCorrPIMPLE() <= 1)
178             {
179                 #include "rhoEqn.H"
180             }
181
```

```
182             // --- PIMPLE loop
183             while (pimple.loop())
184             {
185                 #include "UEqn.H"
186                 #include "YEqn.H"
187                 #include "EEqn.H"
188
189                 // --- Pressure corrector loop
190                 while (pimple.correct())
191                 {
192                     #include "pEqn.H"
193                 }
194
195                 if (pimple.turbCorr())
196                 {
197                     turbulence->correct();
198                 }
199             }
200
201             rho = thermo.rho();
202         }
203
204         runTime.write();
205
206         runTime.printExecutionTime(Info);
207     }
208
209     Info<< "End\n" << endl;
210
211     return 0;
212 }
```

In this solver, the LPT cloud found in the variable `parcels` at line 172 above, gets the following type when macros in `basicHeterogeneousReactingCloud.H` and `basicHeterogeneousReactingParcel.H` are used:

Cloud type

```
1     Foam::ReactingHeterogeneousCloud
2     <
3         Foam::ReactingCloud
4         <
5             Foam::ThermoCloud
6             <
7                 Foam::KinematicCloud
8                 <
9                     Foam::Cloud
10                    <
11                        Foam::ReactingHeterogeneousParcel
12                        <
13                            Foam::ReactingParcel
14                            <
15                                Foam::ThermoParcel
16                                <
17                                    Foam::KinematicParcel
18                                    <
19                                        Foam::particle
20                                    >
21                                >
22                            >
23                        >
24                    >
25                >
26            >
27        >
28    >
```

When a method is called for an object of the cloud type above, it can be a bit tricky to find out which class and method that is actually invoked, due to the technique used in OpenFOAM called parameterized inheritance. How this works can be read more about in Boyao Wang's work [3]. One way to find the class is to use doxygen to search for the method and then if there are multiple found classes, use the knowledge that the cloud- and parcel classes also have different responsibilities, and then pick the most probable class. To be absolutely sure then a debugger should be used to follow the execution of the solver.

3.1 Energy equations

In this section the energy equations are followed in the code, which uses data that is calculated and described in Section 3.2. The energy equations Eq. (2.33) and Eq. (2.35) are found in `$FOAM_SOLVERS/lagrangian/reactingParcelFoam/EEqn.H` as shown below:

`EEqn.H`

```
2     volScalarField& he = thermo.he();
3
4     fvScalarMatrix EEqn
5     (
6         fvm::ddt(rho, he) + mvConvection->fvmDiv(phi, he)
7       + fvc::ddt(rho, K) + fvc::div(phi, K)
8       + (
9             he.name() == "e"
10          ? fvc::div
11            (
12                fvc::absolute(phi/fvc::interpolate(rho), U),
13                p,
14                "div(phiv,p)"
15            )
16          : -dpdt
17        )
18      - fvm::laplacian(turbulence->alphaEff(), he)
19     ==
20        rho*(U&g)
21      + parcels.Sh(he)
22      + surfaceFilm.Sh()
23      + radiation->Sh(thermo, he)
24      + Qdot
25      + fvOptions(rho, he)
26    );
```

At line 21 above, the convective, radiative and chemical heat transferred between the parcels and the continuous phase are added as a source term in the equation by: `parcels.Sh(he)` which corresponds to Eq. (2.37). Note that in the energy equation `EEqn` that the terms are discretized so the unit changes from J/(m^3·s) to J/s when the `fvMatrix` is assembled according to the finite volume method. Fields in the equation like `Qdot` is multiplied by cell volume when the resulting `fvMatrix` is added together in difference to the terms that are an `fvMatrix` from the beginning, like `parcels.Sh(he)` which are already discretized and have unit J/s.

The variable `he`, passed to the `Sh` method, is a volume scalar field containing the enthalpy or the internal energy. Which form of energy that is found in `he` depends on the dictionary `thermoType` in the configuration file `constant/thermophysicalProperties` shown below. The entry `energy` specifies the energy form. When the energy form is `sensibleEnthalpy` or `absoluteEnthalpy` the name of the field is `"h"` otherwise the energy form is `sensibleInternalEnergy` with the name `"e"` of the field. If the selected energy form is enthalpy then it can be absolute (when heat of formation is included) or sensible (without heat of formation).

`thermophysicalProperties`

```
17 thermoType
18 {
```

```
    type            heRhoThermo;
    mixture         reactingMixture;
    transport       sutherland;
    thermo          janaf;
    energy          sensibleEnthalpy;
    equationOfState perfectGas;
    specie          specie;
}
```

Thermophysical models are described in [4]. The method `Sh()` is found in the file `ThermoCloudI.H` shown below, and if the solution is coupled then at line 255 the enthalpy transfer for all internal cells are divided by the time step and put in the source term side of the `fvMatrix`.

`ThermoCloudI.H`

```
225 template<class CloudType>
226 inline Foam::tmp<Foam::fvScalarMatrix>
227 Foam::ThermoCloud<CloudType>::Sh(volScalarField& hs) const
228 {
229     if (debug)
230     {
231         Info<< "hsTrans min/max = " << min(hsTrans()).value() << ", "
232             << max(hsTrans()).value() << nl
233             << "hsCoeff min/max = " << min(hsCoeff()).value() << ", "
234             << max(hsCoeff()).value() << endl;
235     }
236
237     if (this->solution().coupled())
238     {
239         if (this->solution().semiImplicit("h"))
240         {
241             const volScalarField Cp(thermo_.thermo().Cp());
242             const volScalarField::Internal
243                 Vdt(this->mesh().V()*this->db().time().deltaT());
244
245             return
246                 hsTrans()/Vdt
247               - fvm::SuSp(hsCoeff()/(Cp*Vdt), hs)
248               + hsCoeff()/(Cp*Vdt)*hs;
249         }
250         else
251         {
252             tmp<fvScalarMatrix> tfvm(new fvScalarMatrix(hs, dimEnergy/dimTime));
253             fvScalarMatrix& fvm = tfvm.ref();
254
255             fvm.source() = -hsTrans()/(this->db().time().deltaT());
256
257             return tfvm;
258         }
259     }
260
261     return tmp<fvScalarMatrix>::New(hs, dimEnergy/dimTime);
262 }
```

3.2 Cloud and parcel calculations

In each iteration of the solver, before the energy equation is solved in `EEqn.H` the parcels in the LPT cloud are evolved, i.e. injected, moved and calculated. This description is specific for the solver `reactingHeterogenousParcelFoam` but most of this description should also be relevant to the other LPT solvers with an energy equation. In the solver code `reactingParcelFoam.C` the calculation in the `evolve` method at line 172 regards how much mass, energy and momentum that is transferred to/from the parcels in the cloud

When `parcels.evolve()` is called, the `evolve()` implementation in the class `ReactingHeterogeneousCloud` is executed as shown below:

ReactingHeterogeneousCloud.C

```
222 template<class CloudType>
223 void Foam::ReactingHeterogeneousCloud<CloudType>::evolve()
224 {
225     if (this->solution().canEvolve())
226     {
227         typename parcelType::trackingData td(*this);
228 
229         this->solve(*this, td);
230     }
231 }
```

At line 227 above, a tracking data object is created. And at line 229 above, a call to `solve()` is done. Since the method `solve()` is not in the same class, using of the `this` pointer looks up the `solve()` method in one of the base classes for the templated cloud class. The inherited classes are given by the template parameters in `ReactingHeterogeneousCloud.H` shown below. This technique is called parameterized inheritance in C++.

ReactingHeterogeneousCloud.H

```
63 template<class CloudType>
64 class ReactingHeterogeneousCloud
65 :
66     public CloudType,
67     public reactingHeterogeneousCloud
68 {
```

The `solve()` method is dynamically bound to the class `KinematicCloud` and the central calculations are done in `KinematicCloud::evolveCloud()` at line 142 for steady-state and at line 153 for transient mode in the code below:

KinematicCloud.C

```
126 template<class CloudType>
127 template<class TrackCloudType>
128 void Foam::KinematicCloud<CloudType>::solve
129 (
130     TrackCloudType& cloud,
131     typename parcelType::trackingData& td
132 )
133 {
134     addProfiling(prof, "cloud::solve");
135 
136     if (solution_.steadyState())
137     {
138         cloud.storeState();
139 
140         cloud.preEvolve(td);
141 
142         evolveCloud(cloud, td);
143 
144         if (solution_.coupled())
145         {
146             cloud.relaxSources(cloud.cloudCopy());
147         }
148     }
149     else
150     {
151         cloud.preEvolve(td);
152 
153         evolveCloud(cloud, td);
154 
155         if (solution_.coupled())
156         {
157             cloud.scaleSources();
158         }
159     }
```

```

    cloud.info();

    cloud.postEvolve(td);

    if (solution_.steadyState())
    {
        cloud.restoreState();
    }
}
```

Injecting, moving and calculating on the parcels in a cloud are initiated in `KinematicCloud::evolveCloud()` shown below. Injecting the parcels are handled a bit different if the solver is running in transient mode or in steady-state mode. For steady-state mode, the injection is a separate step at line 256, in transient mode all steps are done at line 244.

`KinematicCloud.C`

```
217 template<class CloudType>
218 template<class TrackCloudType>
219 void Foam::KinematicCloud<CloudType>::evolveCloud
220 (
221     TrackCloudType& cloud,
222     typename parcelType::trackingData& td
223 )
224 {
225     if (solution_.coupled())
226     {
227         cloud.resetSourceTerms();
228     }
229
230     if (solution_.transient())
231     {
232         label preInjectionSize = this->size();
233
234         this->surfaceFilm().inject(cloud);
235
236         // Update the cellOccupancy if the size of the cloud has changed
237         // during the injection.
238         if (preInjectionSize != this->size())
239         {
240             updateCellOccupancy();
241             preInjectionSize = this->size();
242         }
243
244         injectors_.inject(cloud, td);
245
246         // Assume that motion will update the cellOccupancy as necessary
247         // before it is required.
248         cloud.motion(cloud, td);
249
250         stochasticCollision().update(td, solution_.trackTime());
251     }
252     else
253     {
254 //        this->surfaceFilm().injectSteadyState(cloud);
255
256         injectors_.injectSteadyState(cloud, td, solution_.trackTime());
257
258         td.part() = parcelType::trackingData::tpLinearTrack;
259         CloudType::move(cloud, td, solution_.trackTime());
260     }
261 }
```

In the file `Cloud.C` the method `move()` moves and calculates on all parcels in the cloud in the loop at lines 210-213 below:

Cloud.C

```
210         for (ParticleType& p : *this)
211         {
212             // Move the particle
213             bool keepParticle = p.move(cloud, td, trackTime);
```

Moving and calculating on a single parcel in the cloud is handled in `KinematicParcel::move()`, which in turn calls the `calc()` method on the parcel at line 379

KinematicParcel.C

```
379             p.calc(cloud, ttd, dt);
```

The `calc()` method is found in the class `ReactingHeterogeousParcel` for the cloud type used by the solver. In the method `ReactingHeterogeousParcel::calc()`, surface values are first calculated at line 136 shown below:

ReactingHeterogeneousParcel.C (1)

```
107 template<class ParcelType>
108 template<class TrackCloudType>
109 void Foam::ReactingHeterogeneousParcel<ParcelType>::calc
110 (
111     TrackCloudType& cloud,
112     trackingData& td,
113     const scalar dt
114 )
115 {
116     typedef typename TrackCloudType::reactingCloudType reactingCloudType;
117
118     const CompositionModel<reactingCloudType>& composition =
119         cloud.composition();
120
121     // Define local properties at beginning of timestep
122     // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
123
124     const scalar np0 = this->nParticle_;
125     const scalar d0 = this->d_;
126     const vector& U0 = this->U_;
127     const scalar T0 = this->T_;
128     const scalar mass0 = this->mass();
129
130     const scalar pc = td.pc();
131
132     const label idS = composition.idSolid();
133
134     // Calc surface values
135     scalar Ts, rhos, mus, Prs, kappas;
136     this->calcSurfaceValues(cloud, td, T0, Ts, rhos, mus, Prs, kappas);
137     scalar Res = this->Re(rhos, U0, td.Uc(), d0, mus);
```

Calculations of surface values are done in `ThermoParcel::calcSurfaceValues()` as shown below, and those surface values are then used above at line 137 to calculate the Reynolds number for the properties at the surface for the continous phase according to Eq. (2.4).

ThermoParcel.C

```
template<class ParcelType>
template<class TrackCloudType>
void Foam::ThermoParcel<ParcelType>::calcSurfaceValues
(
    TrackCloudType& cloud,
    trackingData& td,
    const scalar T,
    scalar& Ts,
    scalar& rhos,
    scalar& mus,
```

```
    scalar& Pr,
    scalar& kappas
) const
{
    // Surface temperature using two thirds rule
    Ts = (2.0*T + td.Tc())/3.0;

    if (Ts < cloud.constProps().TMin())
    {
        if (debug)
        {
            WarningInFunction
                << "Limiting parcel surface temperature to "
                << cloud.constProps().TMin() <<  nl << endl;
        }

        Ts = cloud.constProps().TMin();
    }

    // Assuming thermo props vary linearly with T for small d(T)
    const scalar TRatio = td.Tc()/Ts;

    rhos = td.rhoc()*TRatio;

    tetIndices tetIs = this->currentTetIndices();
    mus = td.muInterp().interpolate(this->coordinates(), tetIs)/TRatio;
    kappas = td.kappaInterp().interpolate(this->coordinates(), tetIs)/TRatio;

    Pr = td.Cpc()*mus/kappas;
    Pr = max(ROOTVSMALL, Pr);
}
```

In the code above, the dynamic viscosity for the continous phase is interpolated to the particle surface at line 220 and the thermal conductivity is interpolated to the surface at line 221, and Eq. (2.5) is solved at line 223. The interpolation is done by the parcel internal class `trackingData`, and the dynamic viscosity is handled by `KinematicParcel` in difference to the thermal conductivity that is handled by `ThermoParcel`. Back in the class `ReactingHeterogeneousParcel`, after the surface values have been calculated, the transferred heat `dhsTrans` is first set in the call to `calcHeterogeneousReactions()` at line 193 below:

`ReactingHeterogeneousParcel.C (2)`

```
    // Heterogeneous reactions
    // ~~~~~~~~~~~~~~~~~

    // Change in carrier phase composition due to surface reactions
    scalarField dMassSRSolid(this->Y_.size(), 0.0);
    scalarField dMassSRCarrier(composition.carrier().species().size(), 0.0);

    // Calc mass and enthalpy transfer due to reactions
    calcHeterogeneousReactions
    (
        cloud,
        td,
        dt,
        Res,
        mus/rhos,
        d0,
        T0,
        mass0,
        canCombust_,
        Ne,
        NCpW,
        this->Y_,
        F_,
        dMassSRSolid,
        dMassSRCarrier,
```

```
210         Sh,
211         dhsTrans
212     );
```

Then the heat in `dhsTrans` should be[1] used for calculating a new temperature for the parcel at line 291 below. `dhsTrans` can at the same time be changed.

ReactingHeterogeneousParcel.C (3)

```
287     // Heat transfer
288     // ~~~~~~~~~~~~~
289
290     // Calculate new particle temperature
291     this->T_ =
292         this->calcHeatTransfer
293         (
294             cloud,
295             td,
296             dt,
297             Res,
298             Prs,
299             kappas,
300             NCpW,
301             Sh,
302             dhsTrans,
303             Sph
304         );
```

Then the heat is transferred from `dhsTrans` to the cloud at line 350 below, into the class `ThermoCloud`'s instance variable `hsTrans` which is a `volScalarField` if the solution is coupled. This `hsTrans` instance variable is a storage for all internal cells where all parcels that are in the same cell are accumulated. `np0` contains the number of particles per parcel.

ReactingHeterogeneousParcel.C (4)

```
336         forAll(dMassSRCarrier, i)
337         {
338             scalar dm = np0*dMassSRCarrier[i];
339             scalar hs = composition.carrier().Hs(i, pc, T0);
340             cloud.rhoTrans(i)[this->cell()] += dm;
341             cloud.UTrans()[this->cell()] += dm*U0;
342             cloud.hsTrans()[this->cell()] += dm*hs;
343         }
344
345         // Update momentum transfer
346         cloud.UTrans()[this->cell()] += np0*dUTrans;
347         cloud.UCoeff()[this->cell()] += np0*Spu;
348
349         // Update sensible enthalpy transfer
350         cloud.hsTrans()[this->cell()] += np0*dhsTrans;
351         cloud.hsCoeff()[this->cell()] += np0*Sph;
352
353         // Update radiation fields
354         if (cloud.radiation())
355         {
356             const scalar ap = this->areaP();
357             const scalar T4 = pow4(T0);
358             cloud.radAreaP()[this->cell()] += dt*np0*ap;
359             cloud.radT4()[this->cell()] += dt*np0*T4;
360             cloud.radAreaPT4()[this->cell()] += dt*np0*ap*T4;
361         }
362     }
363 }
```

[1] In version 2112 of OpenFOAM, heat of reaction is not directly affecting the particle temperature.

3.2.1 Calling the heat transfer model

At line 279 in `ThermoParcel.C` the Heat transfer coefficient calculation is initiated:

```
279     scalar htc = cloud.heatTransfer().htc(d, Re, Pr, kappa, NCpW);
```

In `ThermoCloud.H` a smart pointer to the template class `HeatTransferModel` is declared:

```
125             //- Heat transfer model
126             autoPtr<HeatTransferModel<ThermoCloud<CloudType>>>
127                 heatTransferModel_;
```

The code `cloud.heatTransfer()` above, returns this smart pointer which points to a subclass of `HeatTransferModel`. This feature in C++ is called dynamic binding, and the name of the subclass is set in the cloud properties dictionary and an object of the class is created through the runtime selection table mechanism. The method `htc()` is called which is found in the base class below:

`HeatTransferModel.C`

```
67  template<class CloudType>
68  Foam::scalar Foam::HeatTransferModel<CloudType>::htc
69  (
70      const scalar dp,
71      const scalar Re,
72      const scalar Pr,
73      const scalar kappa,
74      const scalar NCpW
75  ) const
76  {
77      const scalar Nu = this->Nu(Re, Pr);
78
79      scalar htc = Nu*kappa/dp;
80
81      if (BirdCorrection_ && (mag(htc) > ROOTVSMALL) && (mag(NCpW) > ROOTVSMALL))
82      {
83          const scalar phit = min(NCpW/htc, 50);
84          if (phit > 0.001)
85          {
86              htc *= phit/(exp(phit) - 1.0);
87          }
88      }
89
90      return htc;
91  }
```

At line 77 in `HeatTransferModel.C` above, the Nusselt number should be calculated by the `Nu()` method which is abstract in the base class but must be implemented in all subclasses. The implementation in the single available heat transfer model in OpenFOAM looks like:

`RanzMarshall.C`

```
61  template<class CloudType>
62  Foam::scalar Foam::RanzMarshall<CloudType>::Nu
63  (
64      const scalar Re,
65      const scalar Pr
66  ) const
67  {
68      // (AOB:p. 18 below Eq. 42)
69      return a_ + b_*pow(Re, m_)*pow(Pr, n_);
70  }
```

After the Nusselt number is calculated, the Heat transfer coefficient h or the variable `htc` at line 77 can be calculated by using Eq. (2.2):

`HeatTransferModel.C`

```
79      scalar htc = Nu*kappa/dp;
```

3.2.2 Calculating heat transferred

The method `calcHeatTransfer()` is the top level method for solving the equations in Section 2.5. Eq. (2.20) is calculated at line 282, and Eq. (2.21) is calculated at line 283 except for the radiation and reaction terms. The radiation term is later calculated at line 292 and 294 and added at line 297 to the integration constant. NOTE: the chemical reaction heat term, which is already in `dhsTrans` is not used in integrating a new parcel temperature! [2]

In line 305 below in the file `ThermoParcel.C`, the amount:

$$-m \cdot C_p \cdot \Delta T_\mathrm{s} \tag{3.1}$$

is added to `dhsTrans` which already contains the heat (J) from chemical reactions for the parcel, and where $m = \rho_\mathrm{s} V_\mathrm{s}$.

`ThermoParcel.C`

```
251 template<class ParcelType>
252 template<class TrackCloudType>
253 Foam::scalar Foam::ThermoParcel<ParcelType>::calcHeatTransfer
254 (
255     TrackCloudType& cloud,
256     trackingData& td,
257     const scalar dt,
258     const scalar Re,
259     const scalar Pr,
260     const scalar kappa,
261     const scalar NCpW,
262     const scalar Sh,
263     scalar& dhsTrans,
264     scalar& Sph
265 )
266 {
267     if (!cloud.heatTransfer().active())
268     {
269         return T_;
270     }
271 
272     const scalar d = this->d();
273     const scalar rho = this->rho();
274     const scalar As = this->areaS(d);
275     const scalar V = this->volume(d);
276     const scalar m = rho*V;
277 
278     // Calc heat transfer coefficient
279     scalar htc = cloud.heatTransfer().htc(d, Re, Pr, kappa, NCpW);
280 
281     // Calculate the integration coefficients
282     const scalar bcp = htc*As/(m*Cp_);
283     const scalar acp = bcp*td.Tc();
284     scalar ancp = Sh;
285     if (cloud.radiation())
286     {
287         const tetIndices tetIs = this->currentTetIndices();
288         const scalar Gc = td.GInterp().interpolate(this->coordinates(), tetIs);
289         const scalar sigma = physicoChemical::sigma.value();
290         const scalar epsilon = cloud.constProps().epsilon0();
291 
292         ancp += As*epsilon*(Gc/4.0 - sigma*pow4(T_));
293     }
294     ancp /= m*Cp_;
295 
296     // Integrate to find the new parcel temperature
297     const scalar deltaT = cloud.TIntegrator().delta(T_, dt, acp + ancp, bcp);
298     const scalar deltaTncp = ancp*dt;
299     const scalar deltaTcp = deltaT - deltaTncp;
```

[2] In Chapter 5 a new thermo parcel is implemented to fix this problem.

```
300
301     // Calculate the new temperature and the enthalpy transfer terms
302     scalar Tnew = T_ + deltaT;
303     Tnew = min(max(Tnew, cloud.constProps().TMin()), cloud.constProps().TMax());
304
305     dhsTrans -= m*Cp_*deltaTcp;
306
307     Sph = dt*m*Cp_*bcp;
308
309     return Tnew;
310 }
```

Integration is done at line 297 in `ThermoParcel.C` above, to calculate the temperature difference Eq. (2.24). In `integrationSchemeTemplates.C` the effective time step is first calculated in the `delta()` method by calling `dtEff()` at line 54 below:

`integrationSchemeTemplates.C`

```
32 template<class Type>
33 inline Type Foam::integrationScheme::explicitDelta
34 (
35     const Type& phi,
36     const scalar dtEff,
37     const Type& Alpha,
38     const scalar Beta
39 )
40 {
41     return (Alpha - Beta*phi)*dtEff;
42 }
43
44
45 template<class Type>
46 inline Type Foam::integrationScheme::delta
47 (
48     const Type& phi,
49     const scalar dt,
50     const Type& Alpha,
51     const scalar Beta
52 ) const
53 {
54     return explicitDelta(phi, dtEff(dt, Beta), Alpha, Beta);
55 }
```

Depending on which integration scheme that is used, the analytical- or the Euler scheme, via the runtime selection table, the correct code is called. For the analytical solution the code below in `analytical.C` is called, which solves Eq. (2.30)

`analytical.C`

```
56 Foam::scalar Foam::integrationSchemes::analytical::dtEff
57 (
58     const scalar dt,
59     const scalar Beta
60 ) const
61 {
62     return
63         mag(Beta*dt) > SMALL
64       ? (1 - exp(- Beta*dt))/Beta
65       : dt;
66 }
```

and for the Euler solution the code below in `Euler.C` is called, which solves Eq. (2.32):

`Euler.C`

```
56 Foam::scalar Foam::integrationSchemes::Euler::dtEff
57 (
58     const scalar dt,
```

```
    const scalar Beta
) const
{
    return dt/(1 + Beta*dt);
}
```

Then the `explicitDelta()` method is called in `integrationSchemeTemplates.C` to calculate Eq. (2.24)

Chapter 4

Implementing a new heat transfer model

4.1 Code structure

As shown before in Section 3.2.1 the sub class `RanzMarshall` inherits from the base class `HeatTransferModel` which is shown in Figure 4.1. Also a new heat transfer model called `Rowe` is added in the diagram for later implementation. It is also noticed in the diagram that the method `Nu()` is abstract in the base class and implemented in all subclasses. The base class also contains support for the runtime selection table so that changing heat transfer model in the dictionary can be done without re-compiling.

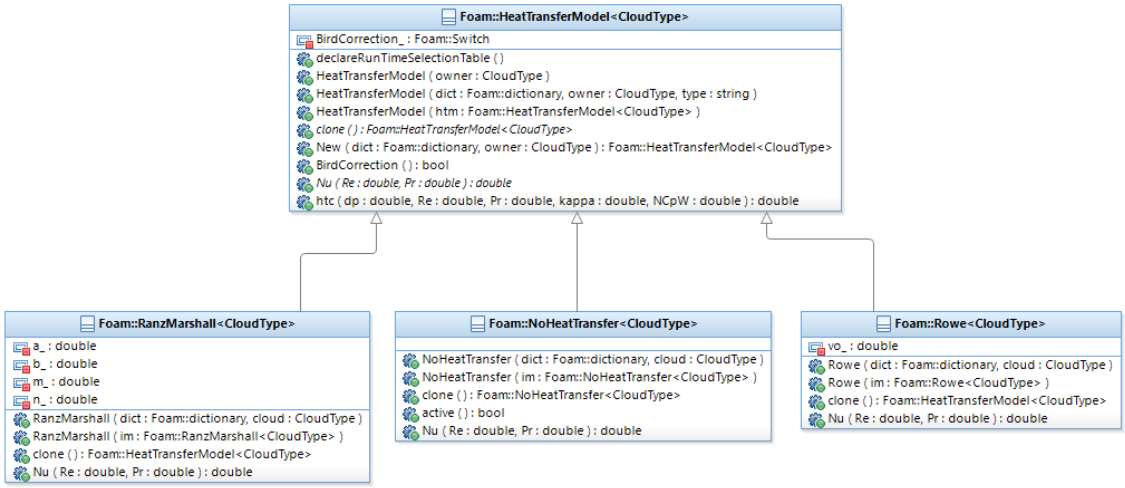

Figure 4.1: HeatTransferModel class diagram

4.2 Assembling an LPT submodel

LPT submodels are located in `src/lagrangian/intermediate/submodels` and are compiled into the library `liblagrangianIntermediate.so`. The Ranz-Marshall heat transfer model is located in `src/lagrangian/intermediate/submodels/Thermodynamic/HeatTransferModel/RanzMarshall/` and the Rowe model should be at the same relative location in the file structure but in the user directory instead of in the OpenFOAM installation directory.

File structure

```
└── src
    └── lagrangian
        └── intermediate
            ├── Make
            │   ├── files
            │   └── options
            ├── parcels
            │   ├── derived
            │   │   └── basicMyHeterogeneousReactingParcel
            │   │       └── makeBasicMyHeterogeneousReactingParcelSubmodels.C
            │   └── include
            │       └── makeParcelHeatTransferModels.H
            └── submodels
                └── Thermodynamic
                    └── HeatTransferModel
                        └── Rowe
                            ├── Rowe.C
                            └── Rowe.H
```

To make a directory for the `Rowe` heat transfer model in the user directory, use the commands below:

```
export D=$WM_PROJECT_USER_DIR/src/lagrangian/intermediate
mkdir -p $D/submodels/Thermodynamic/HeatTransferModel/Rowe
```

Create the directory `Make` for compilation with:

```
mkdir -p $D/Make
```

And a directory for template macros:

```
mkdir -p $D/parcels/include/
```

And a directory for the definition (or template instantiation) of the Rowe heat transfer model with:

```
mkdir -p $D/parcels/derived/basicMyHeterogeneousReactingParcel
```

4.2.1 Make files

Create the files `options` and `files` in the `Make` directory and paste the content for the files from Appendix A. The `files` file contains which files to compile and in which library to assemble the compiled files into. For this submodel the template instantiation file `makeBasicHeterogeneousReactingParcelSubmodels.C` should be compiled and put into the library `libmylagrangianIntermediate.so` as shown below:

`files`

```
12 $(REACTINGHETERMPPARCEL)/makeBasicMyHeterogeneousReactingParcelSubmodels.C
13
14 LIB = $(FOAM_USER_LIBBIN)/libmylagrangianIntermediate
```

Paths to include files and libraries to link to is put in the `options` file. Note that include files from the user directory needs to be in the path as shown at line 2 below:

`options`

```
1 EXE_INC = \
2     -IlnInclude \
3     -I$(LIB_SRC)/finiteVolume/lnInclude \
```

4.2.2 Template macros

Create the file `makeParcelHeatTransferModels.H` in the directory `$WM_PROJECT_USER_DIR/src/lagrangian/intermediate/parcels/include` with the content from Appendix A:

makeParcelHeatTransferModels.H

```
#include "Rowe.H"

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

#define makeParcelHeatTransferModels(CloudType)                                \
                                                                               \
    makeHeatTransferModel(CloudType);                                          \
                                                                               \
    makeHeatTransferModelType(Rowe, CloudType);
```

4.2.3 Template instantiation

Create the file `makeBasicMyHeterogeneousReactingParcelSubmodels.C` in the directory `$WM_PROJECT_USER_DIR/src/lagrangian/intermediate/parcels/derived/ /basicMyHeterogeneousReactingParcel/` with the content from Appendix A:

makeBasicMyHeterogeneousReactingParcelSubmodels.C

```
#include "makeParcelHeatTransferModels.H"

// Thermo sub-models
makeParcelHeatTransferModels(basicHeterogeneousReactingCloud);
```

4.2.4 Rowe heat transfer submodel

Create the files `Rowe.H` and `Rowe.C` in the directory `$WM_PROJECT_USER_DIR/src/lagrangian/intermediate/submodels/Thermodynamic/ /HeatTransferModel/Rowe/` with the content from Appendix A:

Rowe.C

```
template<class CloudType>
Foam::scalar Foam::Rowe<CloudType>::Nu
(
    const scalar Re,
    const scalar Pr
) const
{
    const scalar a = 2 / (1 - cbrt(1 - vo_));
    const scalar b = 2 / (3 * vo_);
    const scalar m = 2.0 / 3.0;
    scalar n = 1.0 / 3.0;
    if (Re != 0) {
        const scalar R_hat = 4.65 * pow(Re, -0.28);
        n = (2 + R_hat) / (3 * R_hat + 3);
    }
    const scalar Nu = a + b * pow(Re, n) * pow(Pr, m);

    if (debug)
    {
        Info << "------------------------" << nl
            << "Re       = " << Re << nl
            << "Pr       = " << Pr << nl
            << "vo       = " << vo_ << nl
            << "a        = " << a << nl
            << "b        = " << b << nl
```

```
80              << "m         = " << m << nl
81              << "n         = " << n << nl
82              << "Nu        = " << Nu << nl
83              << endl;
84      }
85      return Nu;
86  }
87
88
89  // ************************************************************************ //
```

As shown above, Eq. (2.39) is implemented at lines 62-70.

Chapter 5

Implementing a new thermo parcel

The problem with handling heat of reaction found in Section 3.2.2 is addressed in this chapter by creating a new parcel type `MyThermoParcel` and plugging it in the templated class structure instead of the original `ThermoParcel`. It is also possible to keep the same class name for the parcel, but with a new implementation. This alternative will be used for the class `ReactingHeterogeneousParcel`, which needs to be modified in a few lines to work with the new `MyThermoParcel`. Below is the new cloud structure shown:

New cloud type

```
Foam::ReactingHeterogeneousCloud
<
    Foam::ReactingCloud
    <
        Foam::ThermoCloud
        <
            Foam::KinematicCloud
            <
                Foam::Cloud
                <
                    Foam::ReactingHeterogeneousParcel
                    <
                        Foam::ReactingParcel
                        <
                            Foam::MyThermoParcel
                            <
                                Foam::KinematicParcel
                                <
                                    Foam::particle
                                >
                            >
                        >
                    >
                >
            >
        >
    >
```

5.1 File structure for a new thermo parcel

Files that need to be added or modified are shown in the file structure below together with the files needed for the new heat transfer model:

File structure

```
.
├── Make
│   ├── files
│   └── options
├── clouds
│   └── derived
│       └── basicMyHeterogeneousReactingCloud
│           └── basicMyHeterogeneousReactingCloud.H
├── parcels
│   ├── Templates
│   │   ├── MyThermoParcel
│   │   │   ├── MyThermoParcel.C
│   │   │   ├── MyThermoParcel.H
│   │   │   ├── MyThermoParcelI.H
│   │   │   ├── MyThermoParcelIO.C
│   │   │   └── MyThermoParcelTrackingDataI.H
│   │   └── ReactingHeterogeneousParcel
│   │       ├── ReactingHeterogeneousParcel.C
│   │       ├── ReactingHeterogeneousParcel.H
│   │       ├── ReactingHeterogeneousParcelI.H
│   │       └── ReactingHeterogeneousParcelIO.C
│   ├── derived
│   │   ├── basicMyHeterogeneousReactingParcel
│   │   │   ├── basicMyHeterogeneousReactingParcel.H
│   │   │   ├── defineBasicMyHeterogeneousReactingParcel.C
│   │   │   └── makeBasicMyHeterogeneousReactingParcelSubmodels.C
│   │   ├── basicMyReactingParcel
│   │   │   ├── basicMyReactingParcel.H
│   │   │   └── defineBasicMyReactingParcel.C
│   │   └── basicMyThermoParcel
│   │       ├── basicMyThermoParcel.H
│   │       └── defineBasicMyThermoParcel.C
│   └── include
│       └── makeParcelHeatTransferModels.H
└── submodels
    └── Thermodynamic
        └── HeatTransferModel
            └── Rowe
                ├── Rowe.C
                └── Rowe.H
```

Create the directories and files for the `Rowe` heat transfer model by following Section 4.2. Then create the additional directories for `MyThermoParcel` and `ReactingHeterogeneousParcel`. First, the directories for the class templates:

```
export S=$FOAM_SRC/lagrangian/intermediate
export D=$WM_PROJECT_USER_DIR/src/lagrangian/intermediate
mkdir -p $D/parcels/Templates/MyThermoParcel
mkdir -p $D/parcels/Templates/ReactingHeterogeneousParcel
```

Create the directory `basicMyHeterogenousReactingCloud` for a cloud type declaration:

```
mkdir -p $D/clouds/derived/basicMyHeterogeneousReactingCloud
```

And a directory for the definition (or template instantiation) of the heterogeneous reacting parcle with:

```
mkdir -p $D/parcels/derived/basicMyHeterogeneousReactingParcel
```

And a directory for the definition (or template instantiation) of the reacting parcle with:

```
mkdir -p $D/parcels/derived/basicMyReactingParcel
```

And a directory for the definition (or template instantiation) of the thermo parcle with:

```
mkdir -p $D/parcels/derived/basicMyThermoParcel
```

And a Make directory with:

```
mkdir -p $D/Make
```

5.1.1 Make files

Create the files `options` and `files` in the `Make` directory and paste the content for the files from Appendix B. The `files` file contains which files to compile and in which library to assemble the compiled files into. The thermo parcel template instantiation in the file `defineBasicMyThermoParcel.C` should be compiled, and also definitions for parcel classes that are affected: `defineBasicMyReactingParcel.C` and `defineBasicMyHeterogeneousReactingParcel.C`.

As before, the heat transfer submodel template instantiation file `makeBasicHeterogeneousReactingParcelSubmodels.C` should also be compiled as shown below:

`Make/files`

```
PARCELS=parcels
BASEPARCELS=$(PARCELS)/baseClasses
DERIVEDPARCELS=$(PARCELS)/derived

CLOUDS=clouds
BASECLOUDS=$(CLOUDS)/baseClasses
DERIVEDCLOUDS=$(CLOUDS)/derived

/* thermo parcel sub-models */
THERMOPARCEL=$(DERIVEDPARCELS)/basicMyThermoParcel
$(THERMOPARCEL)/defineBasicMyThermoParcel.C

/* reacting parcel sub-models */
REACTINGPARCEL=$(DERIVEDPARCELS)/basicMyReactingParcel
$(REACTINGPARCEL)/defineBasicMyReactingParcel.C

/* heterogeous reacting parcel sub-models */
REACTINGHETERMPPARCEL=$(DERIVEDPARCELS)/basicMyHeterogeneousReactingParcel
$(REACTINGHETERMPPARCEL)/defineBasicMyHeterogeneousReactingParcel.C
$(REACTINGHETERMPPARCEL)/makeBasicMyHeterogeneousReactingParcelSubmodels.C
LIB = $(FOAM_USER_LIBBIN)/libmylagrangianIntermediate
```

The `options` file do not need to be changed. The same file can be used as in implementing a new heat transfer model in Chapter 4, so it can be copy-pasted from Appendix A.

5.1.2 Cloud type declaration

Below is the new cloud type declaration shown:

`basicMyHeterogeneousReactingCloud.H`

```
namespace Foam
{
    typedef ReactingHeterogeneousCloud
    <
        ReactingCloud
        <
            ThermoCloud
            <
                KinematicCloud
                <
                    Cloud
                    <
                        basicMyHeterogeneousReactingParcel
                    >
                >
            >
        >
```

```
    > basicMyHeterogeneousReactingCloud;
}
```

And can be created from the original file by:

```
export S2=$S/clouds/derived/basicHeterogeneousReactingCloud
export D2=$D/clouds/derived/basicMyHeterogeneousReactingCloud

cp $S2/basicHeterogeneousReactingCloud.H $D2/basicMyHeterogeneousReactingCloud.H

sed -i 's/basicHetero/basicMyHetero/g' $D2/basicMyHeterogeneousReactingCloud.H
```

5.1.3 Parcel types declarations

Below is the new parcel type declaration shown:

basicMyHeterogeneousReactingParcel.H

```
    typedef ReactingHeterogeneousParcel
    <
        ReactingParcel
        <
            MyThermoParcel
            <
                KinematicParcel
                <
                    particle
                >
            >
        >
    > basicMyHeterogeneousReactingParcel;
```

And can be created from the original file by:

```
export S2=$S/parcels/derived/basicHeterogeneousReactingParcel
export D2=$D/parcels/derived/basicMyHeterogeneousReactingParcel

cp $S2/basicHeterogeneousReactingParcel.H $D2/basicMyHeterogeneousReactingParcel.H

sed -i 's/basicHetero/basicMyHetero/g' $D2/basicMyHeterogeneousReactingParcel.H
sed -i 's/ThermoParcel/MyThermoParcel/g' $D2/basicMyHeterogeneousReactingParcel.H
```

5.1.4 Templated parcel classes

The `MyThermoParcel` class needs to be created to fix the problem in Section 3.2.2. This can be done by copying the original `ThermoParcel` implementation:

```
export S2=$S/parcels/Templates/ThermoParcel
export D2=$D/parcels/Templates/MyThermoParcel
pushd $S2; for i in Thermo*; do cp $i "`echo "$D2/My$i"`"; done; popd
```

The class name can be changed in all `MyThermoParcel` files by:

```
sed -i 's/ThermoParcel/MyThermoParcel/g' $D2/MyThermoParcel*
```

`ReactingHeterogeneousParcel` is also copied to the user directory:

```
export S2=$S/parcels/Templates/ReactingHeterogeneousParcel
export D2=$D/parcels/Templates/ReactingHeterogeneousParcel
pushd $S2; for i in Reacting*; do cp $i "`echo "$D2/$i"`"; done; popd
```

And all dependencies to `ThermoParcel` from `ReactingHeterogeneousParcel` is changed to `MyThermoParcel` by:

```
sed -i 's/ThermoParcel/MyThermoParcel/g' $D2/Reacting*
```

5.1.5 Template instantiation

We need to modify `typedef`'s for new and affected parcel types and instantiate them, which can be done for `ReactingHeterogeneosParcel` with:

```
export S2=$S/parcels/derived/basicHeterogeneousReactingParcel
export D2=$D/parcels/derived/basicMyHeterogeneousReactingParcel

cp $S2/basicHeterogeneousReactingParcel.H $D2/basicMyHeterogeneousReactingParcel.H

sed -i 's/basicHetero/basicMyHetero/g' $D2/basicMyHeterogeneousReactingParcel.H
sed -i 's/ThermoParcel/MyThermoParcel/g' $D2/basicMyHeterogeneousReactingParcel.H
```

and use the above file in the template instantiation:

```
cp $S2/defineBasicHeterogeneousReactingParcel.C $D2/defineBasicMyHeterogeneousReactingParcel.C

sed -i 's/basicHetero/basicMyHetero/g' $D2/defineBasicMyHeterogeneousReactingParcel.C
sed -i 's/ThermoParcel/MyThermoParcel/g' $D2/defineBasicMyHeterogeneousReactingParcel.C
```

and for `ReactingParcel` with:

```
export S2=$S/parcels/derived/basicReactingParcel
export D2=$D/parcels/derived/basicMyReactingParcel

cp $S2/basicReactingParcel.H $D2/basicMyReactingParcel.H

sed -i 's/basicReactingParcel/basicMyReactingParcel/g' $D2/basicMyReactingParcel.H
sed -i 's/ThermoParcel/MyThermoParcel/g' $D2/basicMyReactingParcel.H
```

and use the above file in the template instantiation:

```
cp $S2/defineBasicReactingParcel.C $D2/defineBasicMyReactingParcel.C

sed -i 's/basicReactingParcel/basicMyReactingParcel/g' $D2/defineBasicMyReactingParcel.C
```

and for `MyThermoParcel` with:

```
export S2=$S/parcels/derived/basicThermoParcel
export D2=$D/parcels/derived/basicMyThermoParcel

cp $S2/basicThermoParcel.H $D2/basicMyThermoParcel.H

sed -i 's/ThermoParcel/MyThermoParcel/g' $D2/basicMyThermoParcel.H
```

and use the above file in the template instantiation:

```
cp $S2/defineBasicThermoParcel.C $D2/defineBasicMyThermoParcel.C

sed -i 's/ThermoParcel/MyThermoParcel/g' $D2/defineBasicMyThermoParcel.C
```

Since we have a new cloud type, we need to define submodels that depend on the new cloud type also. The following commands can be used to do that, and the resulting file is also found in Appendix B.

```
export S2=$S/parcels/derived/basicHeterogeneousReactingParcel
export D2=$D/parcels/derived/basicMyHeterogeneousReactingParcel

cp $S2/makeBasicHeterogeneousReactingParcelSubmodels.C
$D2/makeBasicMyHeterogeneousReactingParcelSubmodels.C

sed -i 's/basicHetero/basicMyHetero/g' $D2/makeBasicMyHeterogeneousReactingParcelSubmodels.C
```

5.2 Modifying MyThermoParcel

The boolean flag `internalHeatOfReaction` is added and declared in `MyThermoParcel.H` with access methods, and the methods are defined in `MyThermoParcelI.H` to read it's value from the dictionary `constantProperties` in the file `reactingCloud1Properties`. Then modify the integration constant term `ancp` to include heat of reaction by replacing line 294 in `MyThermoParcel` below:

```
294     ancp /= m*Cp_;
```

with these lines:

```
    if (cloud.constProps().internalHeatOfReaction())
    {
        ancp += dhsTrans / dt;
    }
    ancp /= m*Cp_;
```

`ancp` corresponds to the second term in the right hand side of Eq. (2.21) which is

$$A = B \cdot T_{\mathrm{g}} + \frac{A_{\mathrm{s}}\epsilon\left(\frac{G}{4} - \sigma T_{\mathrm{s}}^4\right) + \dot{q}_{\mathrm{r}}}{mC_p}.$$

The enthalpy transfer variable `dhsTrans` must also be modified when reaction heat should go to the particle directly instead of to the gas stream. The boolean flag `internalHeatOfReaction` is used to chose where the heat of reaction goes as shown below:

```
    if (cloud.constProps().internalHeatOfReaction())
    {
        dhsTrans = -m*Cp_*deltaTcp;
    }
    else
    {
        dhsTrans -= m*Cp_*deltaTcp;
    }
```

The original implementation is in the `else` block above, where all heat of reaction goes to the gas stream minus the enthalpy to heat up the parcel with only convection (in the variable `deltaTcp` the radiation contribution have been removed and also heat of reaction, which leaves only the convective contribution remaining). The whole modified method is shown below:

`MyThermoParcel.C`

```
251 template<class ParcelType>
252 template<class TrackCloudType>
253 Foam::scalar Foam::MyThermoParcel<ParcelType>::calcHeatTransfer
254 (
255     TrackCloudType& cloud,
256     trackingData& td,
257     const scalar dt,
258     const scalar Re,
259     const scalar Pr,
260     const scalar kappa,
261     const scalar NCpW,
262     const scalar Sh,
263     scalar& dhsTrans,
264     scalar& Sph
265 )
266 {
267     if (!cloud.heatTransfer().active())
268     {
269         return T_;
270     }
271
272     const scalar d = this->d();
273     const scalar rho = this->rho();
```

```
    const scalar As = this->areaS(d);
    const scalar V = this->volume(d);
    const scalar m = rho*V;

    // Calc heat transfer coefficient
    scalar htc = cloud.heatTransfer().htc(d, Re, Pr, kappa, NCpW);

    // Calculate the integration coefficients
    const scalar bcp = htc*As/(m*Cp_);
    const scalar acp = bcp*td.Tc();
    scalar ancp = Sh;
    if (cloud.radiation())
    {
        const tetIndices tetIs = this->currentTetIndices();
        const scalar Gc = td.GInterp().interpolate(this->coordinates(), tetIs);
        const scalar sigma = physicoChemical::sigma.value();
        const scalar epsilon = cloud.constProps().epsilon0();

        ancp += As*epsilon*(Gc/4.0 - sigma*pow4(T_));
    }
    if (cloud.constProps().internalHeatOfReaction())
    {
        ancp += dhsTrans / dt;
    }
    ancp /= m*Cp_;

    // Integrate to find the new parcel temperature
    const scalar deltaT = cloud.TIntegrator().delta(T_, dt, acp + ancp, bcp);
    const scalar deltaTncp = ancp*dt;
    const scalar deltaTcp = deltaT - deltaTncp;

    // Calculate the new temperature and the enthalpy transfer terms
    scalar Tnew = T_ + deltaT;
    Tnew = min(max(Tnew, cloud.constProps().TMin()), cloud.constProps().TMax());

    if (cloud.constProps().internalHeatOfReaction())
    {
        dhsTrans = -m*Cp_*deltaTcp;
    }
    else
    {
        dhsTrans -= m*Cp_*deltaTcp;
    }

    Sph = dt*m*Cp_*bcp;

    if (debug)
    {
        Info << "MyThermoParcel v2: Adding heat of reaction "
            << (
                    cloud.constProps().internalHeatOfReaction()
                  ? "internally"
                  : "externally"
               )
            << nl;
    }
    return Tnew;
}
```

Compile the library with:

```
wmake $D
```

5.3 Using the new thermo parcel

The solver `reactingHeterogenousParcelFoam` needs to be connected to the new cloud type `basicMyHeterogeneousReactingCloud` which contains the new and modified parcel types. Start with copying the original solver to the user directory, and move to that directory:

```
export S2=$FOAM_APP/solvers/lagrangian/reactingParcelFoam/reactingHeterogenousParcelFoam
export D2=$WM_PROJECT_USER_DIR/applications/solvers/lagrangian/reactingParcelFoam/
cp -rp $S2 $D2
cd $D2/reactingHeterogenousParcelFoam
```

Modify the solver to use the new cloud type:

reactingHeterogenousParcelFoam.C

```
#define CLOUD_BASE_TYPE MyHeterogeneousReacting
#define CLOUD_BASE_TYPE_NAME "MyHeterogeneousReacting"

#include "reactingParcelFoam.C"
```

5.3.1 Make files

To compile a new executable solver, replace `$FOAM_APPBIN` with `$FOAM_USER_APPBIN` in:

Make/files

```
1 reactingHeterogenousParcelFoam.C
2
3 EXE = $(FOAM_USER_APPBIN)/reactingHeterogenousParcelFoam
```

And add lines 3, 27, 33, 54 in the `Make/options` file, so that include files and libraries are picked up from the user directories also.

Make/options

```
1  EXE_INC = \
2      -I.. \
3      -I$(FOAM_SOLVERS)/lagrangian/reactingParcelFoam \
4      -I$(LIB_SRC)/finiteVolume/lnInclude \
5      -I$(LIB_SRC)/finiteArea/lnInclude \
6      -I${LIB_SRC}/sampling/lnInclude \
7      -I${LIB_SRC}/meshTools/lnInclude \
8      -I$(LIB_SRC)/dynamicMesh/lnInclude \
9      -I$(LIB_SRC)/dynamicFvMesh/lnInclude \
10     -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
11     -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
12     -I$(LIB_SRC)/lagrangian/distributionModels/lnInclude \
13     -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
14     -I$(LIB_SRC)/transportModels/compressible/lnInclude \
15     -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
16     -I$(LIB_SRC)/thermophysicalModels/thermophysicalProperties/lnInclude \
17     -I$(LIB_SRC)/thermophysicalModels/reactionThermo/lnInclude \
18     -I$(LIB_SRC)/thermophysicalModels/SLGThermo/lnInclude \
19     -I$(LIB_SRC)/thermophysicalModels/chemistryModel/lnInclude \
20     -I$(LIB_SRC)/thermophysicalModels/radiation/lnInclude \
21     -I$(LIB_SRC)/regionModels/regionModel/lnInclude \
22     -I$(LIB_SRC)/regionModels/surfaceFilmModels/lnInclude \
23     -I$(LIB_SRC)/regionFaModels/lnInclude \
24     -I$(LIB_SRC)/faOptions/lnInclude \
25     -I$(LIB_SRC)/lagrangian/basic/lnInclude \
26     -I$(LIB_SRC)/lagrangian/intermediate/lnInclude \
27     -I../../../../../src/lagrangian/intermediate/lnInclude \
28     -I$(LIB_SRC)/ODE/lnInclude \
29     -I$(LIB_SRC)/combustionModels/lnInclude \
30     -I$(FOAM_SOLVERS)/combustion/reactingFoam
31
```

```
EXE_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lfiniteVolume \
    -lfvOptions \
    -lsampling \
    -lmeshTools \
    -ldynamicMesh \
    -ldynamicFvMesh \
    -lturbulenceModels \
    -lcompressibleTurbulenceModels \
    -lspecie \
    -lcompressibleTransportModels \
    -lfluidThermophysicalModels \
    -lreactionThermophysicalModels \
    -lSLGThermo \
    -lchemistryModel \
    -lregionModels \
    -lradiationModels \
    -lsurfaceFilmModels \
    -lsurfaceFilmDerivedFvPatchFields \
    -llagrangian \
    -llagrangianIntermediate \
    -lmylagrangianIntermediate \
    -llagrangianTurbulence \
    -lODE \
    -lcombustionModels \
    -lregionFaModels \
    -lfiniteArea \
    -lfaOptions
```

Compile the solver with:

```
wmake
```

Chapter 6

Tutorial case

This chapter describes how to setup a tutorial case where the newly implemented heat transfer model is compared to the RanzMarshall model. The tutorial case `rectangularDuct` is used and modified to look more like a porous packed bed of iron ore pellets.

6.1 Transient 1-D simulation of a stationary porous packed bed

6.1.1 Original case

The file structure for the tutorial original case is to start with as below:

Original case file structure

```
.
├── 0
│   ├── N2
│   ├── O2
│   ├── T
│   ├── U
│   ├── alphat
│   ├── k
│   ├── nut
│   ├── omega
│   ├── p
│   └── p_rgh
├── constant
│   ├── chemistryProperties
│   ├── combustionProperties
│   ├── foam.dat
│   ├── foam.inp
│   ├── g
│   ├── particleTrackDict
│   ├── radiationProperties
│   ├── reactingCloud1Properties
│   ├── thermophysicalProperties
│   └── turbulenceProperties
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── decomposeParDict
    ├── fvSchemes
    └── fvSolution
```

This case is for a transient simulation of LPT-particles that are injected at an inlet patch with velocity 0.1 m/s in the x-direction and with a diameter of 0.011 m for the particles. Total mass injected is 30 kg during one second with 8442 parcels per second. The initial temperature of the

particles are 303 K and the gas temperature is initially 900 K everywhere. The gas is composed of 80% N_2 and 20% O_2 and have a velocity of 2 m/s initially. The particles are composed of 100% magnetite (Fe_3O_4) initially and the mesh is rectangular with the dimensions 90×10×10 m (L×B×H) and with 40000 cells.

6.1.2 Setup

Create a tutorial directory and copy the `rectangularDuct` case into it with:

```
mkdir $FOAM_RUN/reactingHeterogenousParcelFoam_1D
cd $FOAM_RUN/reactingHeterogenousParcelFoam_1D
cp -r $FOAM_TUTORIALS/lagrangian/reactingHeterogenousParcelFoam/rectangularDuct/ .
```

Create an `Allrun`, `Allclean` and a `changeDictionaryDict` script from Appendix A.2. Also add the python scripts for creating particle positions `createReactingCloud1Positions.py` and for extracting LPT data `runLagrangian.py` and plotting; `plot_lagrangian_layers_T.py, plot_common.py` and `PlotData.py` from the same appendix.

6.1.3 Run

The `Allrun` script can be executed directly to modify the `rectangularDuct` case and start the simulation and do post processing in an automated sequence. But to explain how it works, different parts are extracted, executed and explained. The `Allrun` script should modify the tutorial case file structure to look like below before simulations starts:

Modified case file structure

```
.
├── Allclean
├── Allrun
├── PlotData.py
├── changeDictionaryDict
├── createReactingCloud1Positions.py
├── plot_common.py
├── plot_lagrangian_layers_T.py
├── plot_residual.py
├── rectangularDuct
│       └── ... (original tutorial)
├── rectangularDuct0
│     ├── 0
│     │     ├── N2
│     │     ├── O2
│     │     ├── T
│     │     ├── U
│     │     ├── alphat
│     │     ├── k
│     │     ├── nut
│     │     ├── omega
│     │     ├── p
│     │     └── p_rgh
│     ├── constant
│     │     ├── chemistryProperties
│     │     ├── combustionProperties
│     │     ├── foam.dat
│     │     ├── foam.inp
│     │     ├── g
│     │     ├── particleTrackDict
│     │     ├── polyMesh (generated from blockMesh)
│     │     │     ├── boundary
│     │     │     ├── faces
│     │     │     ├── neighbour
│     │     │     ├── owner
│     │     │     └── points
│     │     ├── radiationProperties
```

```
│   │   │   ├── reactingCloud1Positions (new file)
│   │   │   ├── reactingCloud1Properties
│   │   │   ├── thermophysicalProperties
│   │   │   └── turbulenceProperties
│   │   └── system
│   │       ├── blockMeshDict
│   │       ├── changeDictionaryDict (new file)
│   │       ├── controlDict
│   │       ├── decomposeParDict
│   │       ├── fvSchemes
│   │       └── fvSolution
│   ├── rectangularDuct1
│   │       └── ... (same structure as rectangularDuct0)
│   ├── rectangularDuct2
│   │       └── ... (same structure as rectangularDuct0)
└── runLagrangian.py
```

As a first step in the execution of the `Allrun` script, the original case is copied into `rectangularDuct0` which is later modified and made into a base case for heat transfer model comparison. Then the dictionary `changeDictionaryDict` is copied into the base case, and at line 12, `blockMesh` is run to get something in `constant/polyMesh` required by `changeDictionary`.

`Allrun`

```
1  #!/bin/sh
2  cd "${0%/*}" || exit                                # Run from this directory
3  . ${WM_PROJECT_DIR:?}/bin/tools/RunFunctions        # Tutorial run functions
4  #------------------------------------------------------------------------------
5
6  cp -rf rectangularDuct rectangularDuct0
7  cp -rf changeDictionaryDict rectangularDuct0/system
8  cd rectangularDuct0
9  chmod a+w system/*
10 chmod a+w constant/*
11 chmod a+w 0/*
12 blockMesh
```

Then `changeDictionary` is executed at line 13:

`Allrun`

```
13 changeDictionary
```

with the dictionary content:

`changeDictionaryDict`

```
1  /*--------------------------------*- C++ -*----------------------------------*\
2  | =========                 |                                                 |
3  | \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
4  |  \\    /   O peration     | Version:  v2112                                 |
5  |   \\  /    A nd           | Website:  www.openfoam.com                      |
6  |    \\/     M anipulation  |                                                 |
7  \*---------------------------------------------------------------------------*/
8  FoamFile
9  {
10     version     2.0;
11     format      ascii;
12     class       dictionary;
13     object      changeDictionaryDict;
14 }
15 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
16
17 U
18 {
19     internalField   uniform (3.2 0 0);
20     boundaryField
21     {
```

```
        inlet
        {
            value       uniform (3.2 0 0);
        }
    }
}

T
{
    internalField   uniform 573;
    boundaryField
    {
        inlet
        {
            value       uniform 573;
        }
        outlet
        {
            inletValue      uniform 573;
        }
    }
}

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

which changes the temperature to 573 K and the velocity of the gas to 3.2 m/s. Turbulence is switched off at line 15 in the `Allrun` script:

Allrun

```
15 foamDictionary constant/turbulenceProperties -entry RAS.turbulence -set off
```

The mesh is changed at line 17-73 below, so that the scale gets ten times smaller giving a mesh size of 9x1x1 m, and the cells are a bit thinner in the middle part of the mesh where the LPT-particles later are placed. The mesh is also modified to a 1-D mesh.

Allrun

```
17 foamDictionary system/blockMeshDict -entry scale -set 0.1
18 foamDictionary system/blockMeshDict -entry vertices -set '(
19     (0 0 0)
20     (40 0 0)
21     (40 10 0)
22     (0 10 0)
23     (0 0 10)
24     (40 0 10)
25     (40 10 10)
26     (0 10 10)
27
28     (45 0 0)        // 8
29     (45 10 0)       // 9
30     (45 0 10)       // 10
31     (45 10 10)      // 11
32
33     (90 0 0)        // 12
34     (90 10 0)       // 13
35     (90 0 10)       // 14
36     (90 10 10)      // 15
37 )'
38 foamDictionary system/blockMeshDict -entry blocks -set '(
39 hex (0 1 2 3 4 5 6 7) (40 1 1) simpleGrading (1 1 1)
40 hex (1 8 9 2 5 10 11 6) (50 1 1) simpleGrading (1 1 1)
41 hex (8 12 13 9 10 14 15 11) (45 1 1) simpleGrading (1 1 1))'
42
43 foamDictionary system/blockMeshDict -entry boundary -set '
44 (
45     inlet
46     {
```

```
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }

    outlet
    {
        type patch;
        faces
        (
            (13 15 14 12)
        );
    }
    walls
    {
        type empty;
        faces
        (
            (3 7 6 2)(2 6 11 9)(9 11 15 13)
            (1 5 4 0)(8 10 5 1)(12 14 10 8)
            (0 3 2 1)(1 2 9 8)(8 9 13 12)
            (4 5 6 7)(5 10 11 6)(10 14 15 11)
        );
    }
)'
```

The resulting mesh is shown in Figure 6.1 where the middle part has 50 cells in the x-direction

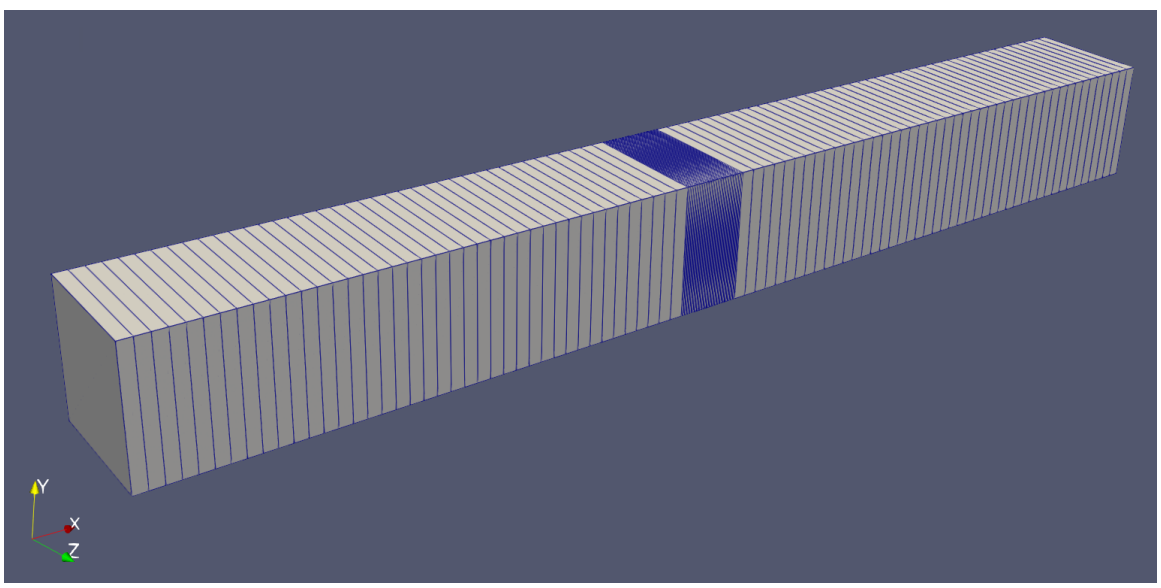

Figure 6.1: Modified mesh for the tutorial case

shown in Figure 6.2

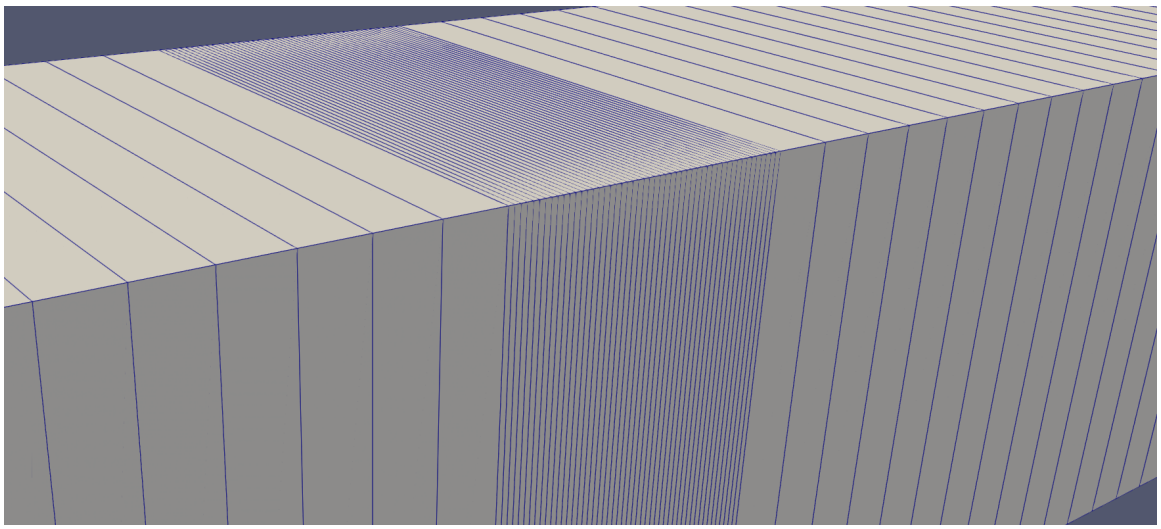

Figure 6.2: Close up of the mesh

At lines 75-78, changes in the dictionary `system/controlDict` is performed with purpose to control the simulation, where settings is changed to use an ASCII format, run for 1500 seconds and set debug switches to off. Debug switches can easily be turned on instead if needed.

`Allrun`

```
75 foamDictionary system/controlDict -entry writeFormat -set ascii
76 foamDictionary system/controlDict -entry endTime -set 1500
77 foamDictionary system/controlDict -entry DebugSwitches.RanzMarshall -set 0
78 foamDictionary system/controlDict -entry DebugSwitches.Rowe -set 0
```

Discretization schemes are changed to second order at lines 80-87 from the first order `Upwind` scheme:

`Allrun`

```
87 foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,U)' -set 'Gauss linearUpwind grad(U)'
88 foamDictionary system/fvSchemes -entry 'divSchemes.div(phid,p)' -set 'Gauss linearUpwind grad(p)'
89 foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,K)' -set 'Gauss linearUpwind grad(K)'
90 foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,h)' -set 'Gauss linearUpwind grad(h)'
91 foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,k)' -set 'Gauss linearUpwind grad(k)'
92 foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,epsilon)' -set 'Gauss linearUpwind grad(
     epsilon)'
93 foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,omega)' -set 'Gauss linearUpwind grad(omega
     )'
94 foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,Yi_h)' -set 'Gauss upwind grad(Yi_h)'
```

Properties for the LPT-particles are changed at lines 89-104:

Allrun

```
89 # Initial magnetite particle density = 5100 * (1 - porosity pellet) = 5140 * (1 - 0.3) = 3598
90 foamDictionary constant/reactingCloud1Properties -entry constantProperties.rho0 -set 3600
91 foamDictionary constant/reactingCloud1Properties -entry constantProperties.Cp0 -set 649
92 foamDictionary constant/reactingCloud1Properties -entry subModels.dispersionModel -set none
93 foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.type -set
       manualInjection
94 foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.patch -remove
95 foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.duration -
       remove
96 foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.
       flowRateProfile -remove
97 foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.
       parcelsPerSecond -remove
98 foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.positionsFile
        -set '"reactingCloud1Positions"'
99 foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.U0 -set '(0 0
        0)'
100 foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.
       sizeDistribution.fixedValueDistribution.value -set 0.012
101 # massTotal = V * rho * (1 - void bed) = 0.5 * 3600 * (1 - 0.4) = 1.08e3
102 foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.massTotal -
       set 1.08e3
103 foamDictionary constant/reactingCloud1Properties -entry subModels.MUCSheterogeneousRateCoeffs.epsilon
       -set 0.4
104 foamDictionary constant/reactingCloud1Properties -entry subModels.singleMixtureFractionCoeffs.phases -
       set '( gas { } liquid { } solid { Fe3O4 0 ; Fe2O3 1 ; } )'
```

Resulting in the following content in `constant/reactingCloud1Properties` when executed:

```
53 constantProperties
54 {
55     rho0            3600;
56     T0              303;
57     Cp0             649;
58     hRetentionCoeff 0;
59     volumeUpdateMethod constantVolume;
60 }
61
62 subModels
63 {
64     particleForces
65     {
66     }
67     injectionModels
68     {
69         model1
70         {
71             type            manualInjection;
72             parcelBasisType mass;
73             U0              ( 0 0 0 );
74             massTotal       1080;
75             SOI             0;
76             minParticlesPerParcel 0.7;
77             sizeDistribution
78             {
79                 type            fixedValue;
80                 fixedValueDistribution
81                 {
82                     value           0.012;
83                 }
84             }
85             positionsFile   "reactingCloud1Positions";
86         }
87     }
88     dispersionModel none;
```

```
    patchInteractionModel standardWallInteraction;
    heatTransferModel RanzMarshall;
    compositionModel singleMixtureFraction;
    phaseChangeModel none;
    stochasticCollisionModel none;
    surfaceFilmModel none;
    radiation       off;
    standardWallInteractionCoeffs
    {
        type            rebound;
    }
    RanzMarshallCoeffs
    {
        BirdCorrection  off;
    }
    heterogeneousReactingModel MUCSheterogeneousRate;
    MUCSheterogeneousRateCoeffs
    {
        D12             0.0002724;
        epsilon         0.4;
        gamma           3.07;
        sigma           1;
        E               1;
        A               31400;
        Aeff            0.7;
        Ea              165100;
        O2              O2;
        nuFuel          2;
        nuProd          3;
        nuOx            0.5;
        fuel            Fe3O4;
        product         Fe2O3;
    }
    singleMixtureFractionCoeffs
    {
        phases          ( gas { } liquid { } solid { Fe3O4 0 ; Fe2O3 1 ; } );
        YGasTot0        0;
        YLiquidTot0     0;
        YSolidTot0      1;
    }
}
```

Table 6.1: Particle properties

Property	Symbol	Value	Description
`constantProperties.rho0`	ρ_p	3600	initial density $(1-\epsilon_\mathrm{p})\rho_\mathrm{pz} = (1-0.3)*5140 = 3598$
`constantProperties.T0`	T_p	303	initial temperature (K)
`constantProperties.Cp0`	C_p	649	initial heat capacity
`subModels.injectionModels .model1.sizeDistribution .fixedValueDistribution.value`	d_p	0.012	diameter
`subModels.injectionModels .model1.U0`	$\mathbf{U}_\mathrm{p}$	(0 0 0)	initial velocity
`subModels.singleMixtureFractionCoeffs .phases.solid.YFe3O4`	$Y_{\mathrm{Fe_3O_4}}$	0	mass fraction of magnetite
`subModels.singleMixtureFractionCoeffs .phases.solid.YFe2O3`	$Y_{\mathrm{Fe_2O_3}}$	1	mass fraction of hematite

Table 6.2: Particle injection properties

Property	Value	Description
`type`	`manualInjection`	manual injection
`parcelBasisType`	`mass`	mass based
`massTotal`	1080	total mass $V_\mathrm{bed}(1-\epsilon_\mathrm{bed})\rho_\mathrm{p} = 0.5*(1-0.4)*3600 = 1.08e3$
`positionsFile`	`reactingCloud1Positions`	parcel positions

Table 6.1 explains the most important particle properties and Table 6.2 shows the most important properties for the particle injection subdictionary `subModels.injectionModels.model1`.

To generate the positions for the particles in the dictionary `constant/reactingCloud1Position`, the Python script `createReactingCloud1Positions.py` is executed on row 106:

`Allrun`

```
106 python ../createReactingCloud1Positions.py
```

The base case is copied to two new cases; `rectangularDuct1` for the `RanzMarshall` heat transfer model and `rectangularDuct2` for the `Rowe` model at lines 112-114:

`Allrun`

```
112 cd ..
113 cp -rf rectangularDuct0 rectangularDuct1
114 cp -rf rectangularDuct0 rectangularDuct2
```

At line 119 the `RanzMarshall` model is executed after the mesh is generated at line 118. Then the post-processing is done at lines 120-122.

Allrun

```
117 cd rectangularDuct1
118 runApplication blockMesh
119 runApplication $(getApplication)
120 python ../runLagrangian.py
121 cp -rf lagrangian.csv ../lagrangian1.csv
122 runApplication foamLog log.reactingHeterogenousParcelFoam
```

At line 128 the `Rowe` model is executed after the `Rowe` model is selected and configured at lines 125-126 and the mesh is generated at line 127. Then the post-processing is done at lines 129-134.

Allrun

```
124 cd ../rectangularDuct2
125 foamDictionary constant/reactingCloud1Properties -entry subModels.heatTransferModel -set Rowe
126 foamDictionary constant/reactingCloud1Properties -entry subModels.RoweCoeffs -set '{vo 0.3;
        BirdCorrection  off;}'
127 runApplication blockMesh
128 runApplication $(getApplication)
129 python ../runLagrangian.py
130 cp -rf lagrangian.csv ../lagrangian2.csv
131 runApplication foamLog log.reactingHeterogenousParcelFoam
132
133 cd ..
134 python plot_lagrangian_layers_T.py
```

6.1.4 Result

Results from the heat transfer simulations are compared in the Figure 6.3, where we can see that a much faster heat transfer is taking place with the Rowe model than with the Ranz-Marshall model. This faster heat transfer using the Rowe model is also a desired effect, since when simulating and comparing to experimental data, there is a too slow heating up of the packed bed with the Ranz-Marshall model.

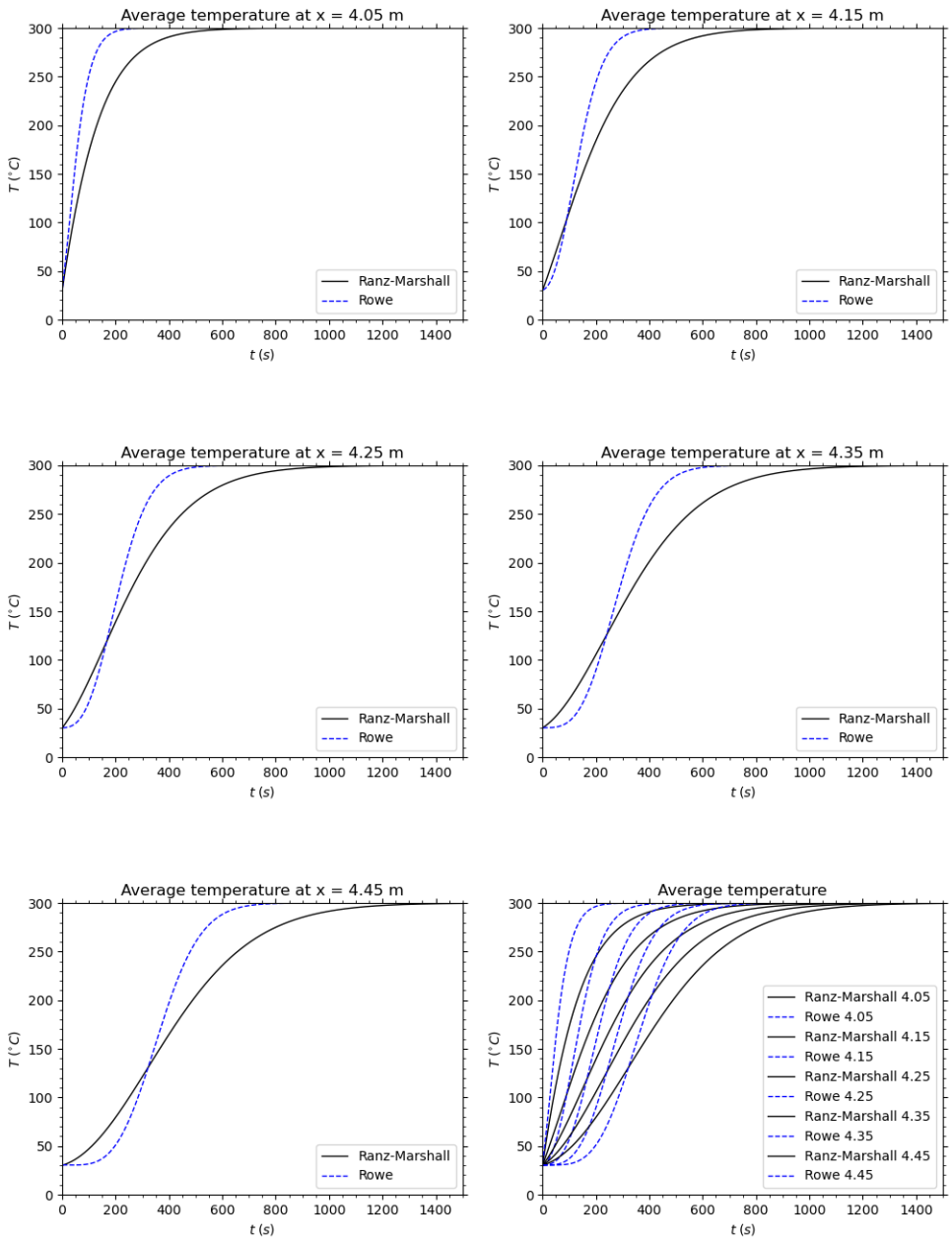

Figure 6.3: Heat transfer models comparison

For Figure 6.3 the configuration as shown in Table 6.1 is used, and it is noticed that there is only hematite in the bed. This means that no chemical reaction is taking place because the hematite is a stable iron oxide in air. This also means that the code modification for heat of reaction in MyThermoParcel is not affecting the result. To see the effect of the code modification, another simulation is done with 100% magnetite instead, which oxidizes in warm air and gives much heat of reaction.

The chemical reaction model used is `MUCSheterogeneousRate` which builds on the work by D. Papanastassiou and G. Bitsianes [5]. Unfortunately there is a bug in the OpenFOAM implementation at line 177 shown below:

`MUCSheterogeneousRate.C`

```
172     const scalar dfdt =
173         Aeff_*(Cb/deltaRho0)
174        /(
175            r/3/alpha
176          + sqr(r)*(1/cbrt(1-f)-1)/3/Deff
177          - (1/sqr(cbrt(1-f)))*r/k/sigma_/E_/3
178         );
```

The last term in the calculation of the conversion rate `dfdt` is subtracted instead of added (as it is in the article [5]). A correction for this can be done by creating the directory:

```
mkdir -p $D/submodels/HeterogeneousReactingModel/MUCSheterogeneousRate/
```

And copy the header- and source file and modify it with:

```
export S2=$S/submodels/HeterogeneousReactingModel/MUCSheterogeneousRate
export D2=$D/submodels/HeterogeneousReactingModel/MUCSheterogeneousRate

cp $S2/MUCSheterogeneousRate.H $D2
cp $S2/MUCSheterogeneousRate.C $D2

sed -i 's/- (1/sqr\/+ (1\/sqr/g' $D2/MUCSheterogeneousRate.C
```

The header file needs also be copied, even if it is not modified, because it includes the .C file. Otherwise the original `MUCSheterogeneousRate.C` is picked up in the compilation. The template macros, used for declaring the submodel, needs also be in the user directory for the same reason:

```
export S2=$S/parcels/include
export D2=$D/parcels/include

cp $S2/makeHeterogeneousReactingParcelHeterogeneousReactingModels.H $D2
```

The files in the `Make` directory are already modified and do not need to be changed to include the modified MUCS model. The added files are shown below:

```
.
├── parcels
│     └── include
│         └── makeHeterogeneousReactingParcelHeterogeneousReactingModels.H
└── submodels
    └── HeterogeneousReactingModel
        └── MUCSheterogeneousRate
            ├── MUCSheterogeneousRate.C
            └── MUCSheterogeneousRate.H
```

The `Allrun` script is extended with setting up two new cases for comparing chemical reaction, without including heat of reaction in integrating the particle temperature (`rectangularDuct3`) and one with heat of reaction included in the integration (`rectangularDuct4`):

`Allrun`

```
137 cp -rf rectangularDuct0 rectangularDuct3
138 cd rectangularDuct3
```

```
foamDictionary constant/reactingCloud1Properties -entry subModels.heatTransferModel -set Rowe
foamDictionary constant/reactingCloud1Properties -entry subModels.RoweCoeffs -set '{vo 0.3;
    BirdCorrection  off;}'
foamDictionary constant/reactingCloud1Properties -entry subModels.singleMixtureFractionCoeffs.phases -
    set '( gas { } liquid { } solid { Fe3O4 1 ; Fe2O3 0 ; } )'
foamDictionary system/controlDict -entry DebugSwitches.basicMyThermoParcel -set 2
runApplication blockMesh
runApplication $(getApplication)
python ../runLagrangian.py
cp -rf lagrangian.csv ../lagrangian3.csv
runApplication foamLog log.reactingHeterogenousParcelFoam


cd ..
cp -rf rectangularDuct0 rectangularDuct4
cd rectangularDuct4
foamDictionary constant/reactingCloud1Properties -entry subModels.heatTransferModel -set Rowe
foamDictionary constant/reactingCloud1Properties -entry subModels.RoweCoeffs -set '{vo 0.3;
    BirdCorrection  off;}'
foamDictionary constant/reactingCloud1Properties -entry subModels.singleMixtureFractionCoeffs.phases -
    set '( gas { } liquid { } solid { Fe3O4 1 ; Fe2O3 0 ; } )'
foamDictionary system/controlDict -entry DebugSwitches.basicMyThermoParcel -set 0
runApplication blockMesh
runApplication $(getApplication)
python ../runLagrangian.py
cp -rf lagrangian.csv ../lagrangian4.csv
runApplication foamLog log.reactingHeterogenousParcelFoam

cd ..
python plot_lagrangian_layers_T_chem.py
#------------------------------------------------------------------------------
```

Result from comparing particle temperatures is shown in Figure 6.4, where the particles gets a higher temperature with the heat of reaction included in the integration.

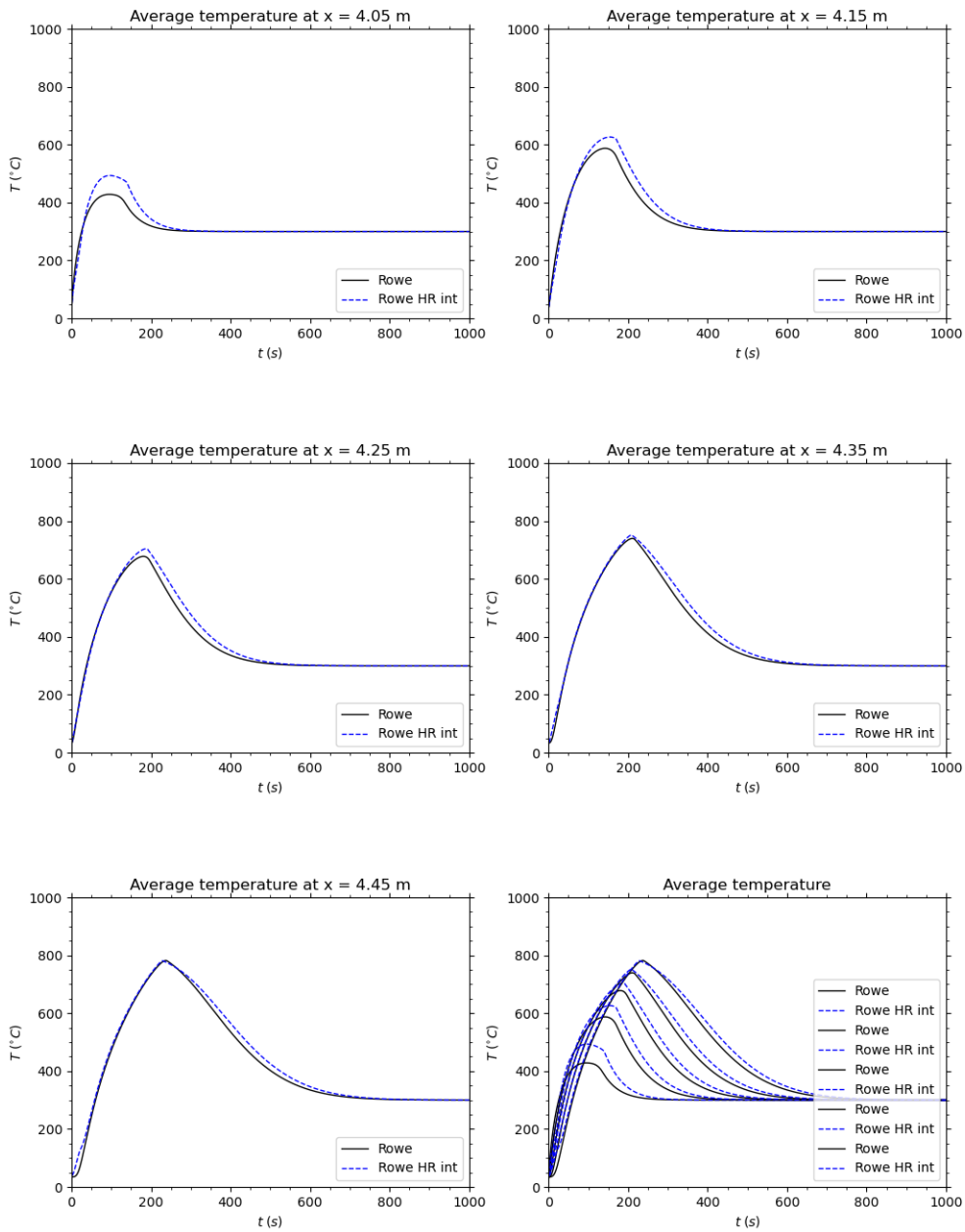

Figure 6.4: Heat transfer models comparison with heat of reaction

Chapter 7

Conclusions

In the present work a new lagrangian heat transfer model has been developed and compared to the only one existing in OpenFOAM v2112. The result from the comparison shows that the new heat transfer model gives a much faster heating up of a porous packet bed, which was desired since the existing model was to slow compared to experimental data.

At the same time problems found in other parts of OpenFOAM regarding handling of chemical reaction heat for porous particles are solved with the development of a new thermo parcel which is successfully compiled and plugged into the `reactingHeterogeneousParcelFoam` solver. Also a bug in the submodel `MUCSheterogeneousRate` is corrected and the modified submodel is compiled in the user directory and successfully used during simulation.

This work hopefully also bring the reader some more knowledge about lagrangian heat transfer through the derived equations for calculating the particle temperature in `ThermoParcel` and for other equation implementations.

Bibliography

[1] J. Szekely, J. W. Evans and H. Y. Sohn, *Gas-Solid Reactions*. New York, San Fransico, London: Academic Press, 1976.

[2] P. N. Rowe, K. T. Claxton and J. B. Lewis, "Heat and mass transfer from a single sphere in an extensive flowing fluid.," *Transaction of Institute Chemical Engineering*, vol. 43, pp. 14–31, 1965.

[3] B. Wang, "Implement a mesh-based particle model for the coalCombustionFoam for solving the biomass combustion," 2021. Edited by Nilsson. H., available online., 2021.

[4] "Thermophysical models." https://doc.cfd.direct/openfoam/user-guide-v6/thermophysical/. Accessed: 2022-11-04.

[5] D. Papanastassiou and G. Bitsianes, "Modelling of Heterogeneous Gas-Solid Reactions," *Metallurgical Transsactions*, vol. 4, p. 480, 1973.

[6] M. Kampili, "Implementation of decay heat model as a submodel in lagrangian library for reactingParcelFoam solver," 2017. Edited by Nilsson. H., available online., 2017.

Study questions

1. How do you switch heat transfer model?
2. How do you check intermediate values in the Rowe heat transfer model?
3. What is the difference between enthalpy and internal energy?
4. How can radiative decay heat be included? Hint: see [6]

Appendix A

Developed codes

A.1 Rowe heat transfer model

A.1.1 files

src/lagrangian/intermediate/Make/files

```
PARCELS=parcels
BASEPARCELS=$(PARCELS)/baseClasses
DERIVEDPARCELS=$(PARCELS)/derived

CLOUDS=clouds
BASECLOUDS=$(CLOUDS)/baseClasses
DERIVEDCLOUDS=$(CLOUDS)/derived

/* heterogeous reacting parcel sub-models */
REACTINGHETERMPPARCEL=$(DERIVEDPARCELS)/basicMyHeterogeneousReactingParcel
$(REACTINGHETERMPPARCEL)/makeBasicMyHeterogeneousReactingParcelSubmodels.C

LIB = $(FOAM_USER_LIBBIN)/libmylagrangianIntermediate
```

A.1.2 options

src/lagrangian/intermediate/Make/options

```
EXE_INC = \
    -IlnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/surfMesh/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/lagrangian/basic/lnInclude \
    -I$(LIB_SRC)/lagrangian/intermediate/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(LIB_SRC)/lagrangian/distributionModels/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/basic/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/thermophysicalProperties/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/reactionThermo/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/SLGThermo/lnInclude \
    -I$(LIB_SRC)/regionModels/regionModel/lnInclude \
    -I$(LIB_SRC)/regionModels/surfaceFilmModels/lnInclude \
    -I$(LIB_SRC)/regionFaModels/lnInclude \
    -I$(LIB_SRC)/finiteArea/lnInclude \
    -I$(LIB_SRC)/faOptions/lnInclude \
    -I$(LIB_SRC)/lagrangian/intermediate/lnInclude
LIB_LIBS =
```

A.1.3 makeBasicMyHeterogeneousReactingParcelSubmodels.C

src/lagrangian/intermediate/parcels/derived/basicMyHeterogeneousReactingParcel/makeBasicMyHeterogeneousReactingParcelSubmodels.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2018-2021 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 "basicHeterogeneousReactingCloud.H"

// Thermodynamic
#include "makeParcelHeatTransferModels.H"

// Thermo sub-models
makeParcelHeatTransferModels(basicHeterogeneousReactingCloud);

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

A.1.4 makeParcelHeatTransferModels.H

src/lagrangian/intermediate/parcels/include/basicMyHeterogeneousReactingParcel/makeParcelHeatTransferModels.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2016 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/>.

\*---------------------------------------------------------------------------*/
#ifndef makeParcelHeatTransferModels_H
#define makeParcelHeatTransferModels_H
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#include "Rowe.H"

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

#define makeParcelHeatTransferModels(CloudType)                             \
                                                                            \
    makeHeatTransferModel(CloudType);                                       \
                                                                            \
    makeHeatTransferModelType(Rowe, CloudType);


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
#endif
// ************************************************************************* //
```

A.1.5 Rowe.H

src/lagrangian/intermediate/submodels/Thermodynamic/HeatTransferModel/Rowe/Rowe.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2016 OpenFOAM Foundation
    Copyright (C) 2021 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/>.

Class
    Foam::Rowe

Group
    grpLagrangianIntermediateHeatTransferSubModels

Description
    Nusselt-number model using the empirical Rowe correlation
    to be used in modelling of the fluid-particle heat transfer coefficient:
```

```
    \f[
        \mathrm{Nu} = A + B\mathrm{Re}^n\mathrm{Pr}^{2/3}
    \f]
    with

    \f[
        \mathrm{Re} =
            \frac{\rho_c \, | \mathbf{u}_\mathrm{rel} | \, d_p}{\mu_c}
    \f]
    with

    \f[
        A = \frac{2}{1 - (1 - \epsilon_\mathrm{v})^{1/3}}
    \f]

    \f[
        B = \frac{2}{3\epsilon_\mathrm{v}}
    \f]

    \f[
        \frac{2 - 3n}{3n-1} = 4.65\mathrm{Re}^{-0.28}
    \f]

    \f[
        \mathrm{Pr} = \frac{ C_p \, \mu_c }{ \kappa_c }
    \f]
    where

    \vartable
        \mathrm{Nu}    | Nusselt number
        \mathrm{Re}    | Reynolds number
        \mathrm{Pr}    | Prandtl number
        d_p            | Particle diameter
        \rho_c         | Density of carrier in the film surrounding particle
        \mu_c    | Dynamic viscosity of carrier in the film surrounding particle
        \mathbf{u}_\mathrm{rel} | Relative velocity between particle and carrier
        \epsilon_\mathrm{v} | Bed voidage
        C_p            | Specific heat capacity
        \kappa_c       | Thermal conductivity of carrier in the film
    \endvartable

    Reference:
    \verbatim
        Standard model:
            P. N. Rowe, K. T. Claxton and J. B. Lewis (1965)
            Heat and mass transfer from a single sphere in an extensive flowing fluid.
            Transaction of Institute Chemical Engineering, 43, pp. 14-31.

    \endverbatim

Usage
    Minimal example by using \c constant/\<CloudProperties\>:
    \verbatim
    subModels
    {
        // Mandatory entries
        heatTransferModel    Rowe;

        // Optional entries
        RoweCoeffs
        {
            vo    0.3;
        }
    }
    \endverbatim

    where the entries mean:
    \table
```

```
      Property           | Description              | Type | Reqd | Deflt
      heatTransferModel  | Type name: Rowe  | word | yes  | -
      vo                 | Bed voidage  | scalar | yes | -
    \endtable

SourceFiles
    Rowe.C

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

#ifndef Rowe_H
#define Rowe_H

#include "HeatTransferModel.H"

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

namespace Foam
{

/*---------------------------------------------------------------------------*\
                        Class Rowe Declaration
\*---------------------------------------------------------------------------*/

template<class CloudType>
class Rowe
:
    public HeatTransferModel<CloudType>
{
    // Private Data

        //- Voidage
        const scalar vo_;


public:

    //- Runtime type information
    TypeName("Rowe");


    // Generated Methods

        //- No copy assignment
        void operator=(const Rowe&) = delete;


    // Constructors

        //- Construct from dictionary
        Rowe(const dictionary& dict, CloudType& cloud);

        //- Copy construct
        Rowe(const Rowe<CloudType>& im);

        //- Construct and return a clone
        virtual autoPtr<HeatTransferModel<CloudType>> clone() const
        {
            return autoPtr<HeatTransferModel<CloudType>>
            (
                new Rowe<CloudType>(*this)
            );
        }


    //- Destructor
    virtual ~Rowe() = default;
```

```
    // Member Functions

        // Evaluation

            //- Return Nusselt number
            virtual scalar Nu
            (
                const scalar Re,
                const scalar Pr
            ) const;
};


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

} // End namespace Foam

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

#ifdef NoRepository
    #include "Rowe.C"
#endif

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

#endif

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

A.1.6 Rowe.C

src/lagrangian/intermediate/submodels/Thermodynamic/HeatTransferModel/Rowe/Rowe.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011 OpenFOAM Foundation
    Copyright (C) 2021 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 "Rowe.H"

// * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * //
```

```
template<class CloudType>
Foam::Rowe<CloudType>::Rowe
(
    const dictionary& dict,
    CloudType& cloud
)
:
    HeatTransferModel<CloudType>(dict, cloud, typeName),
    vo_(this->coeffDict().template get<scalar>("vo"))
{}


template<class CloudType>
Foam::Rowe<CloudType>::Rowe(const Rowe<CloudType>& htm)
:
    HeatTransferModel<CloudType>(htm),
    vo_(htm.vo_)
{}


// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * //

template<class CloudType>
Foam::scalar Foam::Rowe<CloudType>::Nu
(
    const scalar Re,
    const scalar Pr
) const
{
    const scalar a = 2 / (1 - cbrt(1 - vo_));
    const scalar b = 2 / (3 * vo_);
    const scalar m = 2.0 / 3.0;
    scalar n = 1.0 / 3.0;
    if (Re != 0) {
        const scalar R_hat = 4.65 * pow(Re, -0.28);
        n = (2 + R_hat) / (3 * R_hat + 3);
    }
    const scalar Nu = a + b * pow(Re, n) * pow(Pr, m);

    if (debug)
    {
        Info << "-----------------------" << nl
            << "Re       = " << Re << nl
            << "Pr       = " << Pr << nl
            << "vo       = " << vo_ << nl
            << "a        = " << a << nl
            << "b        = " << b << nl
            << "m        = " << m << nl
            << "n        = " << n << nl
            << "Nu       = " << Nu << nl
            << endl;
    }
    return Nu;
}


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

A.2 Tutorial case

A.2.1 Allrun

Allrun

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

cp -rf rectangularDuct rectangularDuct0
cp -rf changeDictionaryDict rectangularDuct0/system
cd rectangularDuct0
chmod a+w system/*
chmod a+w constant/*
chmod a+w 0/*
blockMesh
changeDictionary

foamDictionary constant/turbulenceProperties -entry RAS.turbulence -set off

foamDictionary system/blockMeshDict -entry scale -set 0.1
foamDictionary system/blockMeshDict -entry vertices -set '(
    (0 0 0)
    (40 0 0)
    (40 10 0)
    (0 10 0)
    (0 0 10)
    (40 0 10)
    (40 10 10)
    (0 10 10)

    (45 0 0)        // 8
    (45 10 0)       // 9
    (45 0 10)       // 10
    (45 10 10)      // 11

    (90 0 0)        // 12
    (90 10 0)       // 13
    (90 0 10)       // 14
    (90 10 10)      // 15
)'
foamDictionary system/blockMeshDict -entry blocks -set '(
hex (0 1 2 3 4 5 6 7) (40 1 1) simpleGrading (1 1 1)
hex (1 8 9 2 5 10 11 6) (50 1 1) simpleGrading (1 1 1)
hex (8 12 13 9 10 14 15 11) (45 1 1) simpleGrading (1 1 1))'

foamDictionary system/blockMeshDict -entry boundary -set '
(
    inlet
    {
        type patch;
        faces
        (
            (0 4 7 3)
        );
    }

    outlet
    {
        type patch;
        faces
        (
            (13 15 14 12)
        );
    }
    walls
    {
        type empty;
        faces
        (
            (3 7 6 2)(2 6 11 9)(9 11 15 13)
            (1 5 4 0)(8 10 5 1)(12 14 10 8)
```

```
            (0 3 2 1)(1 2 9 8)(8 9 13 12)
            (4 5 6 7)(5 10 11 6)(10 14 15 11)
        );
    }
)'

foamDictionary system/controlDict -entry writeFormat -set ascii
foamDictionary system/controlDict -entry endTime -set 1500
foamDictionary system/controlDict -entry DebugSwitches.RanzMarshall -set 0
foamDictionary system/controlDict -entry DebugSwitches.Rowe -set 0

foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,U)' -set 'Gauss linearUpwind grad(U)'
foamDictionary system/fvSchemes -entry 'divSchemes.div(phid,p)' -set 'Gauss linearUpwind grad(p)'
foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,K)' -set 'Gauss linearUpwind grad(K)'
foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,h)' -set 'Gauss linearUpwind grad(h)'
foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,k)' -set 'Gauss linearUpwind grad(k)'
foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,epsilon)' -set 'Gauss linearUpwind grad(
    epsilon)'
foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,omega)' -set 'Gauss linearUpwind grad(omega
    )'
foamDictionary system/fvSchemes -entry 'divSchemes.div(phi,Yi_h)' -set 'Gauss upwind grad(Yi_h)'

# Initial magnetite particle density = 5100 * (1 - porosity pellet) = 5140 * (1 - 0.3) = 3598
foamDictionary constant/reactingCloud1Properties -entry constantProperties.rho0 -set 3600
foamDictionary constant/reactingCloud1Properties -entry constantProperties.Cp0 -set 649
foamDictionary constant/reactingCloud1Properties -entry subModels.dispersionModel -set none
foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.type -set
    manualInjection
foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.patch -remove
foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.duration -
    remove
foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.
    flowRateProfile -remove
foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.
    parcelsPerSecond -remove
foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.positionsFile
     -set '"reactingCloud1Positions"'
foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.U0 -set '(0 0
     0)'
foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.
    sizeDistribution.fixedValueDistribution.value -set 0.012
# massTotal = V * rho * (1 - void bed) = 0.5 * 3600 * (1 - 0.4) = 1.08e3
foamDictionary constant/reactingCloud1Properties -entry subModels.injectionModels.model1.massTotal -
    set 1.08e3
foamDictionary constant/reactingCloud1Properties -entry subModels.MUCSheterogeneousRateCoeffs.epsilon
    -set 0.4
foamDictionary constant/reactingCloud1Properties -entry subModels.singleMixtureFractionCoeffs.phases -
    set '( gas { } liquid { } solid { Fe3O4 0 ; Fe2O3 1 ; } )'

python ../createReactingCloud1Positions.py
cp -f constant/reactingCloud1Properties constant/reactingCloud2Properties
cp -f constant/reactingCloud1Properties constant/reactingCloud3Properties
foamDictionary constant/reactingCloud2Properties -entry subModels.injectionModels.model1.massTotal -
    set 0
foamDictionary constant/reactingCloud3Properties -entry subModels.injectionModels.model1.massTotal -
    set 0

cd ..
cp -rf rectangularDuct0 rectangularDuct1
cp -rf rectangularDuct0 rectangularDuct2


cd rectangularDuct1
runApplication blockMesh
runApplication $(getApplication)
python ../runLagrangian.py
cp -rf lagrangian.csv ../lagrangian1.csv
runApplication foamLog log.reactingHeterogenousParcelFoam
```

```
cd ../rectangularDuct2
foamDictionary constant/reactingCloud1Properties -entry subModels.heatTransferModel -set Rowe
foamDictionary constant/reactingCloud1Properties -entry subModels.RoweCoeffs -set '{vo 0.3;
    BirdCorrection  off;}'
runApplication blockMesh
runApplication $(getApplication)
python ../runLagrangian.py
cp -rf lagrangian.csv ../lagrangian2.csv
runApplication foamLog log.reactingHeterogenousParcelFoam

cd ..
python plot_lagrangian_layers_T.py


cp -rf rectangularDuct0 rectangularDuct3
cd rectangularDuct3
foamDictionary constant/reactingCloud1Properties -entry subModels.heatTransferModel -set Rowe
foamDictionary constant/reactingCloud1Properties -entry subModels.RoweCoeffs -set '{vo 0.3;
    BirdCorrection  off;}'
foamDictionary constant/reactingCloud1Properties -entry subModels.singleMixtureFractionCoeffs.phases -
    set '( gas { } liquid { } solid { Fe3O4 1 ; Fe2O3 0 ; } )'
foamDictionary system/controlDict -entry DebugSwitches.basicMyThermoParcel -set 2
runApplication blockMesh
runApplication $(getApplication)
python ../runLagrangian.py
cp -rf lagrangian.csv ../lagrangian3.csv
runApplication foamLog log.reactingHeterogenousParcelFoam


cd ..
cp -rf rectangularDuct0 rectangularDuct4
cd rectangularDuct4
foamDictionary constant/reactingCloud1Properties -entry subModels.heatTransferModel -set Rowe
foamDictionary constant/reactingCloud1Properties -entry subModels.RoweCoeffs -set '{vo 0.3;
    BirdCorrection  off;}'
foamDictionary constant/reactingCloud1Properties -entry subModels.singleMixtureFractionCoeffs.phases -
    set '( gas { } liquid { } solid { Fe3O4 1 ; Fe2O3 0 ; } )'
foamDictionary system/controlDict -entry DebugSwitches.basicMyThermoParcel -set 0
runApplication blockMesh
runApplication $(getApplication)
python ../runLagrangian.py
cp -rf lagrangian.csv ../lagrangian4.csv
runApplication foamLog log.reactingHeterogenousParcelFoam

cd ..
python plot_lagrangian_layers_T_chem.py
#------------------------------------------------------------------------------
```

A.2.2 Allclean

Allclean

```
#!/bin/sh
cd "${0%/*}" || exit                                # Run from this directory
. ${WM_PROJECT_DIR:?}/bin/tools/CleanFunctions      # Tutorial clean functions
#------------------------------------------------------------------------------
rm -rf rectangularDuct0
rm -rf rectangularDuct1
rm -rf rectangularDuct2
rm -rf rectangularDuct3
rm -rf rectangularDuct4
rm -f *.png
rm -f *.eps
rm -f *.csv
#------------------------------------------------------------------------------
```

A.2.3 changeDictionaryDict

changeDictionaryDict

```
/*--------------------------------*- 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      changeDictionaryDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

U
{
    internalField   uniform (3.2 0 0);
    boundaryField
    {
        inlet
        {
            value       uniform (3.2 0 0);
        }
    }
}

T
{
    internalField   uniform 573;
    boundaryField
    {
        inlet
        {
            value       uniform 573;
        }
        outlet
        {
            inletValue      uniform 573;
        }
    }
}

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

A.2.4 createReactingCloud1Positions.py

createReactingCloud1Positions.py

```
# ---------------------------------------------------------------------------
# Project      : OSCFD 2022
#
# Name         : createReactingCloud1Positions.py
#
# Prepared     : Orjan Fjallborg LKAB
#
# Description  : Creates the file reactingCloud1Positions
#
# History      :
# 1, 2022-12-07, klorfj, Created
# ---------------------------------------------------------------------------
```

```
import io
from string import Template
import os

# ---------------------------------------------------------------------------
# Name:         createPositions
# Summary:      creates the positions assuming rectangular and uniformly
#               sized cells in the mesh
# Returns:      see above
# ---------------------------------------------------------------------------
def createPositions():
    buf = io.StringIO()

    Nx, Ny, Nz = (50, 1, 1)       # no of cells in x- y- and z direction
    min = [4, 0, 0]                # min coordinate of bounding box
    max = [4.5, 1, 1]              # max coordinate of bounding box
    cs_x = (max[0] - min[0]) / Nx
    cs_y = (max[1] - min[1]) / Ny
    cs_z = (max[2] - min[2]) / Nz

    for k in range(Nz):
        z = cs_z / 2 + k * cs_z + min[2]
        for j in range(Ny):
            y = cs_y / 2 + j * cs_y + min[1]
            for i in range(Nx):
                x = cs_x / 2 + i * cs_x + min[0]
                buf.write('    ({0} {1} {2})\n'.format(x, y, z))

    return buf.getvalue()

# ---------------------------------------------------------------------------
# Name:         createContent
# Summary:      Creates the position vector field as a string
# Returns:      see above
# ---------------------------------------------------------------------------
def createContent():
    template = Template(r'''/*--------------------------------*- C++
     -*----------------------------------*\
| =========                 |                                                 |
| \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox           |
|  \\    /   O peration     | Version:  v2112                                 |
|   \\  /    A nd           | Web:      www.openfoam.org                      |
|    \\/     M anipulation  |                                                 |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       vectorField;
    object      reactingCloud1Positions;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
(
$internalField
)
// ************************************************************************* //
''')

    return template.substitute(
        internalField = createPositions()
    )




f = open(os.getcwd() + "/constant/reactingCloud1Positions", 'w')
f.write(createContent())
f.close()
```

A.2.5 runLagrangian.py

runLagrangian.py

```
from PyFoam.RunDictionary.SolutionDirectory import SolutionDirectory
from PyFoam.RunDictionary.LagrangianCloudData import LagrangianCloudData
import pandas as pd
sol = SolutionDirectory(".")
data = pd.concat([LagrangianCloudData(".","reactingCloud1",t).data for t in sol.times if float(t) >
     0])
data = data.round({'Px': 4, 'Py': 4, 'Pz': 4})
data.to_csv("lagrangian.csv")
```

A.2.6 plot_lagrangian_layers_T.py

plot_lagrangian_layers_T.py

```
# ---------------------------------------------------------------------------
# Project      : OSCFD 2022
#
# Name         : plot_lagrangian_layers_T.py
#
# Prepared     : Orjan Fjallborg LKAB
#
# Description  : Plots the temperature from LPT data for heat transfer model comparison.
#                One subplot is created for each position in x.
#
# History      :
# 1, 2022-12-07, klorfj, Created
# ---------------------------------------------------------------------------
import pandas as ps
import numpy as np
import os, sys
import math
import matplotlib.pyplot as plt
from matplotlib.ticker import (MultipleLocator)
import datetime
import plot_common as pc
from PlotData import PlotData

# ---------------------------------------------------------------------------
# Name:         add_sim_T_to_plot
# Summary:      queries the LPT data and adds temperature to the Axes ax.
# Params:       ax      (I/O) - Axes object
#               x_layer   (I) - x layer
#               data      (I) - PyFoam LPT pandas data
#               pd        (I) - a PlotData object
#               add_label (I) - Set label if True
# ---------------------------------------------------------------------------
def add_sim_T_to_plot(ax, x_layer, data, pd, add_label=False):
    xa=[]
    ya=[]

    ds = 0.005

    query = (
        'Px >= ' + str(x_layer) + ' - ' + str(ds) + ' and '
        'Px <= ' + str(x_layer) + ' + ' + str(ds)
        )
    data.query(query, inplace=True)

    grouped = data.groupby(["age"])
    avg = grouped.aggregate(np.average)

    xa = data['age'].apply(np.array).unique()
    ya = avg['T'].values-273
```

```
    pc.plot_array(ax, xa, ya, pd, add_label)



fig = plt.figure(figsize=(20, 27))
gs = fig.add_gridspec(3, 2, hspace=0.5)
axs = gs.subplots()
ax_ix = 0
for x_layer in [4.05, 4.15, 4.25, 4.35, 4.45]:
    ax = axs[math.floor(ax_ix / 2), ax_ix % 2]
    ax.set_xlim(0, 1500)
    ax.set_ylim(0, 300)
    ax.set_xmargin(0)
    ax.set_ymargin(0)
    ax.xaxis.set_minor_locator(MultipleLocator(5))
    ax.yaxis.set_minor_locator(MultipleLocator(5))
    ax.minorticks_on()
    ax.tick_params(top=True, right=True)
    ax.tick_params(which = 'major', top = True, right = True)
    ax.tick_params(which = 'minor', top = True, right = True)
    dir = os.getcwd()

    data1 = ps.read_csv(dir + "/lagrangian1.csv")
    data2 = ps.read_csv(dir + "/lagrangian2.csv")
    add_sim_T_to_plot(ax, x_layer, data1, pc.PLOT_DATA_ARR[0], True)
    add_sim_T_to_plot(ax, x_layer, data2, pc.PLOT_DATA_ARR[1], True)

    ax.set_xlabel("$t \; (s)$")
    ax.set_ylabel("$T \; (^{\circ} C)$")
    ax.set_title("Average temperature at x = " + str(x_layer) + " m")
    ax.legend(loc="lower right")
    ax_ix = ax_ix + 1


# Plot for all positions in the last subplot
ax = axs[2, 1]
txt1 = pc.PLOT_DATA_ARR[0].text
txt2 = pc.PLOT_DATA_ARR[1].text
for x_layer in [4.05, 4.15, 4.25, 4.35, 4.45]:
    ax.set_xlim(0, 1500)
    ax.set_ylim(0, 300)
    ax.set_xmargin(0)
    ax.set_ymargin(0)
    ax.xaxis.set_minor_locator(MultipleLocator(5))
    ax.yaxis.set_minor_locator(MultipleLocator(5))
    ax.minorticks_on()
    ax.tick_params(top=True, right=True)
    ax.tick_params(which = 'major', top = True, right = True)
    ax.tick_params(which = 'minor', top = True, right = True)
    dir = os.getcwd()

    data1 = ps.read_csv(dir + "/lagrangian1.csv")
    data2 = ps.read_csv(dir + "/lagrangian2.csv")
    pc.PLOT_DATA_ARR[0].text = txt1 + " " + str(x_layer)
    pc.PLOT_DATA_ARR[1].text = txt2 + " " + str(x_layer)
    add_sim_T_to_plot(ax, x_layer, data1, pc.PLOT_DATA_ARR[0], True)
    add_sim_T_to_plot(ax, x_layer, data2, pc.PLOT_DATA_ARR[1], True)

    ax.set_xlabel("$t \; (s)$")
    ax.set_ylabel("$T \; (^{\circ} C)$")
    ax.set_title("Average temperature")
    ax.legend(loc="lower right")

plt.savefig("plot_lagrangian_layers_T.eps")
plt.savefig("plot_lagrangian_layers_T.png")
```

A.2.7 plot_common.py

plot_common.py

```
# ---------------------------------------------------------------------------
# Project      : OSCFD 2022
#
# Name         : plot_common.py
#
# Prepared     : Orjan Fjallborg LKAB
#
# Description  : Global parameters and plot functionality.
#
# History      :
# 1, 2022-12-07, klorfj, Created
# ---------------------------------------------------------------------------
import numpy as np
import os, sys
import math
import matplotlib.pyplot as plt
from matplotlib.ticker import (MultipleLocator)
import datetime
from PlotData import PlotData

# Global parameters

PLOT_DATA_ARR = [
    PlotData(color='black', line_style='solid', text='Ranz-Marshall', case='rectangularDuct1'),
    PlotData(color='blue', line_style='dashed', text='Rowe', case='rectangularDuct2'),
    PlotData(color='black', line_style='solid', text='Rowe', case='rectangularDuct3'),
    PlotData(color='blue', line_style='dashed', text='Rowe HR int', case='rectangularDuct4')
]



# ---------------------------------------------------------------------------
# Name:         plot_array
# Summary:      Plotts x- and y- data arrays to the Axes ax.
# Params:       ax      (I/O) - Axes object
#               xa        (I) - x data array
#               ya        (I) - y data array
#               pd        (I) - a PlotData object
#               add_label (I) - Set label if True
# ---------------------------------------------------------------------------
def plot_array(ax, xa, ya, pd, add_label=False):

    if (add_label):
        ax.plot(xa, ya, color=pd.color, linewidth=1, linestyle=pd.line_style,
            label = pd.text)
    else:
        ax.plot(xa, ya, color=pd.color, linewidth=1, linestyle=pd.line_style)

# ---------------------------------------------------------------------------
# Name:         plot_array_exp
# Summary:      Plotts x- and y- experimental data arrays to the Axes ax.
# Params:       ax      (I/O) - Axes object
#               xa        (I) - x data array
#               ya        (I) - y data array
#               add_label (I) - Set label if True
# ---------------------------------------------------------------------------
def plot_array_exp(ax, xa, ya, add_label=False):

    if (add_label):
        ax.plot(xa, ya, linestyle='none', marker='o', fillstyle='none',
            markersize=3, markeredgecolor = 'b',
            label = "Experiment")
    else:
        ax.plot(xa, ya, linestyle='none', marker='o', fillstyle='none',
```

```
            markersize=3, markeredgecolor = 'b', linewidth=1)
```

A.2.8 PlotData.py

PlotData.py

```
# ---------------------------------------------------------------------------
# Project      : OSCFD 2022
#
# Name         : PlotData.py
#
# Prepared     : Orjan Fjallborg LKAB
#
# Description  : Class handling data for different case trends in the same plot
#
# History      :
# 1, 2022-12-07, klorfj, Created
# ---------------------------------------------------------------------------

class PlotData(object):
    # -----------------------------------------------------------------------------
    # Name:         __init__
    # Summary:      Constructor
    # Params:
    # Returns:      the contructed object
    # -----------------------------------------------------------------------------
    def __init__(self, color, line_style, text, case):
        self.color = color
        self.line_style = line_style
        self.text = text
        self.case = case
```

A.2.9 plot_lagrangian_layers_T_chem.py

plot_lagrangian_layers_T_chem.py

```
# ---------------------------------------------------------------------------
# Project      : OSCFD 2022
#
# Name         : plot_lagrangian_layers_T_chem.py
#
# Prepared     : Orjan Fjallborg LKAB
#
# Description  : Plots the temperature from LPT data for heat transfer model comparison.
#                One subplot is created for each position in x.
#
# History      :
# 1, 2022-12-18, klorfj, Created
# ---------------------------------------------------------------------------
import pandas as ps
import numpy as np
import os, sys
import math
import matplotlib.pyplot as plt
from matplotlib.ticker import (MultipleLocator)
import datetime
import plot_common as pc
from PlotData import PlotData

# -----------------------------------------------------------------------------
# Name:         add_sim_T_to_plot
# Summary:      queries the LPT data and adds temperature to the Axes ax.
# Params:       ax      (I/O) - Axes object
#               x_layer   (I) - x layer
```

```
#               data      (I) - PyFoam LPT pandas data
#               pd        (I) - a PlotData object
#               add_label (I) - Set label if True
# ---------------------------------------------------------------------------
def add_sim_T_to_plot(ax, x_layer, data, pd, add_label=False):
    xa=[]
    ya=[]

    ds = 0.005

    query = (
        'Px >= ' + str(x_layer) + ' - ' + str(ds) + ' and '
        'Px <= ' + str(x_layer) + ' + ' + str(ds)
        )
    data.query(query, inplace=True)

    grouped = data.groupby(["age"])
    avg = grouped.aggregate(np.average)

    xa = data['age'].apply(np.array).unique()
    ya = avg['T'].values-273

    pc.plot_array(ax, xa, ya, pd, add_label)



fig = plt.figure(figsize=(12, 16))
gs = fig.add_gridspec(3, 2, hspace=0.5)
axs = gs.subplots()
ax_ix = 0
for x_layer in [4.05, 4.15, 4.25, 4.35, 4.45]:
    ax = axs[math.floor(ax_ix / 2), ax_ix % 2]
    ax.set_xlim(0, 1000)
    ax.set_ylim(0, 1000)
    ax.set_xmargin(0)
    ax.set_ymargin(0)
    ax.xaxis.set_minor_locator(MultipleLocator(5))
    ax.yaxis.set_minor_locator(MultipleLocator(5))
    ax.minorticks_on()
    ax.tick_params(top=True, right=True)
    ax.tick_params(which = 'major', top = True, right = True)
    ax.tick_params(which = 'minor', top = True, right = True)
    dir = os.getcwd()

    data1 = ps.read_csv(dir + "/lagrangian3.csv")
    data2 = ps.read_csv(dir + "/lagrangian4.csv")
    add_sim_T_to_plot(ax, x_layer, data1, pc.PLOT_DATA_ARR[2], True)
    add_sim_T_to_plot(ax, x_layer, data2, pc.PLOT_DATA_ARR[3], True)

    ax.set_xlabel("$t \; (s)$")
    ax.set_ylabel("$T \; (^{\circ} C)$")
    ax.set_title("Average temperature at x = " + str(x_layer) + " m")
    ax.legend(loc="lower right")
    ax_ix = ax_ix + 1


# Plot for all positions in the last subplot
ax = axs[2, 1]
txt1 = pc.PLOT_DATA_ARR[0].text
txt2 = pc.PLOT_DATA_ARR[1].text
for x_layer in [4.05, 4.15, 4.25, 4.35, 4.45]:
    ax.set_xlim(0, 1000)
    ax.set_ylim(0, 1000)
    ax.set_xmargin(0)
    ax.set_ymargin(0)
    ax.xaxis.set_minor_locator(MultipleLocator(5))
    ax.yaxis.set_minor_locator(MultipleLocator(5))
    ax.minorticks_on()
```

```
    ax.tick_params(top=True, right=True)
    ax.tick_params(which = 'major', top = True, right = True)
    ax.tick_params(which = 'minor', top = True, right = True)
    dir = os.getcwd()

    data1 = ps.read_csv(dir + "/lagrangian3.csv")
    data2 = ps.read_csv(dir + "/lagrangian4.csv")
    pc.PLOT_DATA_ARR[0].text = txt1 + " " + str(x_layer)
    pc.PLOT_DATA_ARR[1].text = txt2 + " " + str(x_layer)
    add_sim_T_to_plot(ax, x_layer, data1, pc.PLOT_DATA_ARR[2], True)
    add_sim_T_to_plot(ax, x_layer, data2, pc.PLOT_DATA_ARR[3], True)

    ax.set_xlabel("$t \; (s)$")
    ax.set_ylabel("$T \; (^{\circ} C)$")
    ax.set_title("Average temperature")
    ax.legend(loc="lower right")

plt.savefig("plot_lagrangian_layers_T_chem.eps")
plt.savefig("plot_lagrangian_layers_T_chem.png")
```

Appendix B

Developed codes for new parcel types

B.1 Parcel types

B.1.1 files

src/lagrangian/intermediate/Make/files

```
PARCELS=parcels
BASEPARCELS=$(PARCELS)/baseClasses
DERIVEDPARCELS=$(PARCELS)/derived

CLOUDS=clouds
BASECLOUDS=$(CLOUDS)/baseClasses
DERIVEDCLOUDS=$(CLOUDS)/derived

/* thermo parcel sub-models */
THERMOPARCEL=$(DERIVEDPARCELS)/basicMyThermoParcel
$(THERMOPARCEL)/defineBasicMyThermoParcel.C

/* reacting parcel sub-models */
REACTINGPARCEL=$(DERIVEDPARCELS)/basicMyReactingParcel
$(REACTINGPARCEL)/defineBasicMyReactingParcel.C

/* heterogeous reacting parcel sub-models */
REACTINGHETERMPPARCEL=$(DERIVEDPARCELS)/basicMyHeterogeneousReactingParcel
$(REACTINGHETERMPPARCEL)/defineBasicMyHeterogeneousReactingParcel.C
$(REACTINGHETERMPPARCEL)/makeBasicMyHeterogeneousReactingParcelSubmodels.C
LIB = $(FOAM_USER_LIBBIN)/libmylagrangianIntermediate
```

B.1.2 MyThermoParcel.H

src/lagrangian/intermediate/parcels/Templates/MyThermoParcel/MyThermoParcel.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  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) 2016-2019 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/>.

Class
    Foam::MyThermoParcel

Group
    grpLagrangianIntermediateParcels

Description
    Thermodynamic parcel class with one/two-way coupling with the continuous
    phase. Includes Kinematic parcel sub-models, plus:
    - heat transfer

SourceFiles
    MyThermoParcelI.H
    MyThermoParcel.C
    MyThermoParcelIO.C

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

#ifndef MyThermoParcel_H
#define MyThermoParcel_H

#include "particle.H"
#include "SLGThermo.H"
#include "demandDrivenEntry.H"

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

namespace Foam
{

template<class ParcelType>
class MyThermoParcel;

template<class ParcelType>
Ostream& operator<<
(
    Ostream&,
    const MyThermoParcel<ParcelType>&
);

/*---------------------------------------------------------------------------*\
                        Class MyThermoParcel Declaration
\*---------------------------------------------------------------------------*/

template<class ParcelType>
class MyThermoParcel
:
    public ParcelType
{
public:
```

```
    //- Size in bytes of the fields
    static const std::size_t sizeofFields;


    //- Class to hold thermo particle constant properties
    class constantProperties
    :
        public ParcelType::constantProperties
    {

        // Private data

            //- Particle initial temperature [K]
            demandDrivenEntry<scalar> T0_;

            //- Minimum temperature [K]
            demandDrivenEntry<scalar> TMin_;

            //- Maximum temperature [K]
            demandDrivenEntry<scalar> TMax_;

            //- Particle specific heat capacity [J/(kg.K)]
            demandDrivenEntry<scalar> Cp0_;

            //- Particle emissivity [] (radiation)
            demandDrivenEntry<scalar> epsilon0_;

            //- Particle scattering factor [] (radiation)
            demandDrivenEntry<scalar> f0_;

            //- Particle internal heat of reaction flag.
            // 0 - heat of reaction going directly to/from continous phase.
            // 1 - heat of reaction going directly into the particle
            demandDrivenEntry<bool> internalHeatOfReaction_;
    public:

        // Constructors

            //- Null constructor
            constantProperties();

            //- Copy constructor
            constantProperties(const constantProperties& cp);

            //- Construct from dictionary
            constantProperties(const dictionary& parentDict);


        // Member functions

            // Access

                //- Return const access to the particle initial temperature [K]
                inline scalar T0() const;

                //- Return const access to minimum temperature [K]
                inline scalar TMin() const;

                //- Return const access to maximum temperature [K]
                inline scalar TMax() const;

                //- Set the maximum temperature [K]
                inline void setTMax(const scalar TMax);

                //- Return const access to the particle specific heat capacity
                //  [J/(kg.K)]
                inline scalar Cp0() const;
```

```
            //- Return const access to the particle emissivity []
            //  Active for radiation only
            inline scalar epsilon0() const;

            //- Return const access to the particle scattering factor []
            //  Active for radiation only
            inline scalar f0() const;

            //- Return const access to internal heat of reaction flag
            inline bool internalHeatOfReaction() const;
};


class trackingData
:
    public ParcelType::trackingData
{
private:

    // Private data

        //- Local copy of carrier specific heat field
        //  Cp not stored on carrier thermo, but returned as tmp<...>
        const volScalarField Cp_;

        //- Local copy of carrier thermal conductivity field
        //  kappa not stored on carrier thermo, but returned as tmp<...>
        const volScalarField kappa_;


        // Interpolators for continuous phase fields

            //- Temperature field interpolator
            autoPtr<interpolation<scalar>> TInterp_;

            //- Specific heat capacity field interpolator
            autoPtr<interpolation<scalar>> CpInterp_;

            //- Thermal conductivity field interpolator
            autoPtr<interpolation<scalar>> kappaInterp_;

            //- Radiation field interpolator
            autoPtr<interpolation<scalar>> GInterp_;


        // Cached continuous phase properties

            //- Temperature [K]
            scalar Tc_;

            //- Specific heat capacity [J/(kg.K)]
            scalar Cpc_;


public:

    typedef typename ParcelType::trackingData::trackPart trackPart;

    // Constructors

        //- Construct from components
        template <class TrackCloudType>
        inline trackingData
        (
            const TrackCloudType& cloud,
            trackPart part = ParcelType::trackingData::tpLinearTrack
        );
```

```
        // Member functions

            //- Return access to the locally stored carrier Cp field
            inline const volScalarField& Cp() const;

            //- Return access to the locally stored carrier kappa field
            inline const volScalarField& kappa() const;

            //- Return const access to the interpolator for continuous
            //  phase temperature field
            inline const interpolation<scalar>& TInterp() const;

            //- Return const access to the interpolator for continuous
            //  phase specific heat capacity field
            inline const interpolation<scalar>& CpInterp() const;

            //- Return const access to the interpolator for continuous
            //  phase thermal conductivity field
            inline const interpolation<scalar>& kappaInterp() const;

            //- Return const access to the interpolator for continuous
            //  radiation field
            inline const interpolation<scalar>& GInterp() const;

            //- Return the continuous phase temperature
            inline scalar Tc() const;

            //- Access the continuous phase temperature
            inline scalar& Tc();

            //- Return the continuous phase specific heat capacity
            inline scalar Cpc() const;

            //- Access the continuous phase specific heat capacity
            inline scalar& Cpc();
    };


protected:

    // Protected data

        // Parcel properties

            //- Temperature [K]
            scalar T_;

            //- Specific heat capacity [J/(kg.K)]
            scalar Cp_;


    // Protected Member Functions

        //- Calculate new particle temperature
        template<class TrackCloudType>
        scalar calcHeatTransfer
        (
            TrackCloudType& cloud,
            trackingData& td,
            const scalar dt,           // timestep
            const scalar Re,           // Reynolds number
            const scalar Pr,           // Prandtl number - surface
            const scalar kappa,        // Thermal conductivity - surface
            const scalar NCpW,         // Sum of N*Cp*W of emission species
            const scalar Sh,           // explicit particle enthalpy source
            scalar& dhsTrans,          // sensible enthalpy transfer to carrier
            scalar& Sph                // linearised heat transfer coefficient
```

```
        );


public:

    // Static data members

        //- Runtime type information
        TypeName("MyThermoParcel");

        //- String representation of properties
        AddToPropertyList
        (
            ParcelType,
            " T"
          + " Cp"
        );


    // Constructors

        //- Construct from mesh, coordinates and topology
        //  Other properties initialised as null
        inline MyThermoParcel
        (
            const polyMesh& mesh,
            const barycentric& coordinates,
            const label celli,
            const label tetFacei,
            const label tetPti
        );

        //- Construct from a position and a cell, searching for the rest of the
        //  required topology. Other properties are initialised as null.
        inline MyThermoParcel
        (
            const polyMesh& mesh,
            const vector& position,
            const label celli
        );

        //- Construct from components
        inline MyThermoParcel
        (
            const polyMesh& mesh,
            const barycentric& coordinates,
            const label celli,
            const label tetFacei,
            const label tetPti,
            const label typeId,
            const scalar nParticle0,
            const scalar d0,
            const scalar dTarget0,
            const vector& U0,
            const vector& f0,
            const vector& angularMomentum0,
            const vector& torque0,
            const constantProperties& constProps
        );

        //- Construct from Istream
        MyThermoParcel
        (
            const polyMesh& mesh,
            Istream& is,
            bool readFields = true,
            bool newFormat = true
        );
```

```
        //- Construct as a copy
        MyThermoParcel(const MyThermoParcel& p);

        //- Construct as a copy
        MyThermoParcel(const MyThermoParcel& p, const polyMesh& mesh);

        //- Construct and return a (basic particle) clone
        virtual autoPtr<particle> clone() const
        {
            return autoPtr<particle>(new MyThermoParcel(*this));
        }

        //- Construct and return a (basic particle) clone
        virtual autoPtr<particle> clone(const polyMesh& mesh) const
        {
            return autoPtr<particle>(new MyThermoParcel(*this, mesh));
        }

        //- Factory class to read-construct particles used for
        //  parallel transfer
        class iNew
        {
            const polyMesh& mesh_;

        public:

            iNew(const polyMesh& mesh)
            :
                mesh_(mesh)
            {}

            autoPtr<MyThermoParcel<ParcelType>> operator()(Istream& is) const
            {
                return autoPtr<MyThermoParcel<ParcelType>>
                (
                    new MyThermoParcel<ParcelType>(mesh_, is, true)
                );
            }
        };


    // Member Functions

        // Access

            //- Return const access to temperature
            inline scalar T() const;

            //- Return const access to specific heat capacity
            inline scalar Cp() const;

            //- Return the parcel sensible enthalpy
            inline scalar hs() const;


        // Edit

            //- Return access to temperature
            inline scalar& T();

            //- Return access to specific heat capacity
            inline scalar& Cp();


        // Main calculation loop

            //- Set cell values
```

```
        template<class TrackCloudType>
        void setCellValues(TrackCloudType& cloud, trackingData& td);

        //- Correct cell values using latest transfer information
        template<class TrackCloudType>
        void cellValueSourceCorrection
        (
            TrackCloudType& cloud,
            trackingData& td,
            const scalar dt
        );

        //- Calculate surface thermo properties
        template<class TrackCloudType>
        void calcSurfaceValues
        (
            TrackCloudType& cloud,
            trackingData& td,
            const scalar T,
            scalar& Ts,
            scalar& rhos,
            scalar& mus,
            scalar& Pr,
            scalar& kappas
        ) const;

        //- Update parcel properties over the time interval
        template<class TrackCloudType>
        void calc
        (
            TrackCloudType& cloud,
            trackingData& td,
            const scalar dt
        );


    // I-O

        //- Read
        template<class CloudType>
        static void readFields(CloudType& c);

        //- Write
        template<class CloudType>
        static void writeFields(const CloudType& c);

        //- Write individual parcel properties to stream
        void writeProperties
        (
            Ostream& os,
            const wordRes& filters,
            const word& delim,
            const bool namesOnly = false
        ) const;

        //- Read particle fields as objects from the obr registry
        template<class CloudType>
        static void readObjects(CloudType& c, const objectRegistry& obr);

        //- Write particle fields as objects into the obr registry
        template<class CloudType>
        static void writeObjects(const CloudType& c, objectRegistry& obr);


// Ostream Operator

    friend Ostream& operator<< <ParcelType>
    (
```

```
            Ostream&,
            const MyThermoParcel<ParcelType>&
        );
};


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

} // End namespace Foam

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

#include "MyThermoParcelI.H"
#include "MyThermoParcelTrackingDataI.H"

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

#ifdef NoRepository
    #include "MyThermoParcel.C"
#endif

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

#endif

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

B.1.3 MyThermoParcelI.H

src/lagrangian/intermediate/parcels/Templates/MyThermoParcel/MyThermoParcelI.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/>.

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

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

template<class ParcelType>
inline Foam::MyThermoParcel<ParcelType>::constantProperties::constantProperties()
:
    ParcelType::constantProperties(),
    T0_(this->dict_, 0.0),
```

```
    TMin_(this->dict_, 0.0),
    TMax_(this->dict_, VGREAT),
    Cp0_(this->dict_, 0.0),
    epsilon0_(this->dict_, 0.0),
    f0_(this->dict_, 0.0)
{}


template<class ParcelType>
inline Foam::MyThermoParcel<ParcelType>::constantProperties::constantProperties
(
    const constantProperties& cp
)
:
    ParcelType::constantProperties(cp),
    T0_(cp.T0_),
    TMin_(cp.TMin_),
    TMax_(cp.TMax_),
    Cp0_(cp.Cp0_),
    epsilon0_(cp.epsilon0_),
    f0_(cp.f0_)
{}


template<class ParcelType>
inline Foam::MyThermoParcel<ParcelType>::constantProperties::constantProperties
(
    const dictionary& parentDict
)
:
    ParcelType::constantProperties(parentDict),
    T0_(this->dict_, "T0"),
    TMin_(this->dict_, "TMin", 200.0),
    TMax_(this->dict_, "TMax", 5000.0),
    Cp0_(this->dict_, "Cp0"),
    epsilon0_(this->dict_, "epsilon0"),
    f0_(this->dict_, "f0")
{}


template<class ParcelType>
inline Foam::MyThermoParcel<ParcelType>::MyThermoParcel
(
    const polyMesh& mesh,
    const barycentric& coordinates,
    const label celli,
    const label tetFacei,
    const label tetPti
)
:
    ParcelType(mesh, coordinates, celli, tetFacei, tetPti),
    T_(0.0),
    Cp_(0.0)
{}


template<class ParcelType>
inline Foam::MyThermoParcel<ParcelType>::MyThermoParcel
(
    const polyMesh& mesh,
    const vector& position,
    const label celli
)
:
    ParcelType(mesh, position, celli),
    T_(0.0),
    Cp_(0.0)
{}
```

```
template<class ParcelType>
inline Foam::MyThermoParcel<ParcelType>::MyThermoParcel
(
    const polyMesh& mesh,
    const barycentric& coordinates,
    const label celli,
    const label tetFacei,
    const label tetPti,
    const label typeId,
    const scalar nParticle0,
    const scalar d0,
    const scalar dTarget0,
    const vector& U0,
    const vector& f0,
    const vector& angularMomentum0,
    const vector& torque0,
    const constantProperties& constProps
)
:
    ParcelType
    (
        mesh,
        coordinates,
        celli,
        tetFacei,
        tetPti,
        typeId,
        nParticle0,
        d0,
        dTarget0,
        U0,
        f0,
        angularMomentum0,
        torque0,
        constProps
    ),
    T_(constProps.T0()),
    Cp_(constProps.Cp0())
{}


// * * * * * * * * * constantProperties Member Functions * * * * * * * * * * //

template<class ParcelType>
inline Foam::scalar
Foam::MyThermoParcel<ParcelType>::constantProperties::T0() const
{
    return T0_.value();
}


template<class ParcelType>
inline Foam::scalar
Foam::MyThermoParcel<ParcelType>::constantProperties::TMin() const
{
    return TMin_.value();
}


template<class ParcelType>
inline Foam::scalar
Foam::MyThermoParcel<ParcelType>::constantProperties::TMax() const
{
    return TMax_.value();
}
```

```
template<class ParcelType>
inline void
Foam::MyThermoParcel<ParcelType>::constantProperties::setTMax(const scalar TMax)
{
    TMax_.setValue(TMax);
}


template<class ParcelType>
inline Foam::scalar
Foam::MyThermoParcel<ParcelType>::constantProperties::Cp0() const
{
    return Cp0_.value();
}


template<class ParcelType>
inline Foam::scalar
Foam::MyThermoParcel<ParcelType>::constantProperties::epsilon0() const
{
    return epsilon0_.value();
}


template<class ParcelType>
inline Foam::scalar
Foam::MyThermoParcel<ParcelType>::constantProperties::f0() const
{
    return f0_.value();
}


// * * * * * * * * * * MyThermoParcel Member Functions * * * * * * * * * * * * //

template<class ParcelType>
inline Foam::scalar Foam::MyThermoParcel<ParcelType>::T() const
{
    return T_;
}


template<class ParcelType>
inline Foam::scalar Foam::MyThermoParcel<ParcelType>::Cp() const
{
    return Cp_;
}


template<class ParcelType>
inline Foam::scalar Foam::MyThermoParcel<ParcelType>::hs() const
{
    return Cp_*(T_ - 298.15);
}


template<class ParcelType>
inline Foam::scalar& Foam::MyThermoParcel<ParcelType>::T()
{
    return T_;
}


template<class ParcelType>
inline Foam::scalar& Foam::MyThermoParcel<ParcelType>::Cp()
{
    return Cp_;
}
```

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

B.1.4 MyThermoParcel.C

src/lagrangian/intermediate/parcels/Templates/MyThermoParcel/
MyThermoParcel.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  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/>.

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

#include "MyThermoParcel.H"
#include "physicoChemicalConstants.H"

using namespace Foam::constant;

// * * * * * * * * * * * *  Protected Member Functions * * * * * * * * * * * * * //

template<class ParcelType>
template<class TrackCloudType>
void Foam::MyThermoParcel<ParcelType>::setCellValues
(
    TrackCloudType& cloud,
    trackingData& td
)
{
    ParcelType::setCellValues(cloud, td);

    tetIndices tetIs = this->currentTetIndices();

    td.Cpc() = td.CpInterp().interpolate(this->coordinates(), tetIs);

    td.Tc() = td.TInterp().interpolate(this->coordinates(), tetIs);

    if (td.Tc() < cloud.constProps().TMin())
    {
        if (debug)
        {
            WarningInFunction
                << "Limiting observed temperature in cell " << this->cell()
                << " to " << cloud.constProps().TMin() <<  nl << endl;
```

```
        }

        td.Tc() = cloud.constProps().TMin();
    }
}


template<class ParcelType>
template<class TrackCloudType>
void Foam::MyThermoParcel<ParcelType>::cellValueSourceCorrection
(
    TrackCloudType& cloud,
    trackingData& td,
    const scalar dt
)
{
    const label celli = this->cell();
    const scalar massCell = this->massCell(td);

    td.Uc() += cloud.UTrans()[celli]/massCell;

//    tetIndices tetIs = this->currentTetIndices();
//    Tc_ = td.TInterp().interpolate(this->coordinates(), tetIs);

    const scalar CpMean = td.CpInterp().psi()[celli];
    td.Tc() += cloud.hsTrans()[celli]/(CpMean*massCell);

    if (td.Tc() < cloud.constProps().TMin())
    {
        if (debug)
        {
            WarningInFunction
                << "Limiting observed temperature in cell " << celli
                << " to " << cloud.constProps().TMin() <<  nl << endl;
        }

        td.Tc() = cloud.constProps().TMin();
    }
}


template<class ParcelType>
template<class TrackCloudType>
void Foam::MyThermoParcel<ParcelType>::calcSurfaceValues
(
    TrackCloudType& cloud,
    trackingData& td,
    const scalar T,
    scalar& Ts,
    scalar& rhos,
    scalar& mus,
    scalar& Pr,
    scalar& kappas
) const
{
    // Surface temperature using two thirds rule
    Ts = (2.0*T + td.Tc())/3.0;

    if (Ts < cloud.constProps().TMin())
    {
        if (debug)
        {
            WarningInFunction
                << "Limiting parcel surface temperature to "
                << cloud.constProps().TMin() <<  nl << endl;
        }

        Ts = cloud.constProps().TMin();
```

```
    }

    // Assuming thermo props vary linearly with T for small d(T)
    const scalar TRatio = td.Tc()/Ts;

    rhos = td.rhoc()*TRatio;

    tetIndices tetIs = this->currentTetIndices();
    mus = td.muInterp().interpolate(this->coordinates(), tetIs)/TRatio;
    kappas = td.kappaInterp().interpolate(this->coordinates(), tetIs)/TRatio;

    Pr = td.Cpc()*mus/kappas;
    Pr = max(ROOTVSMALL, Pr);
}


template<class ParcelType>
template<class TrackCloudType>
void Foam::MyThermoParcel<ParcelType>::calc
(
    TrackCloudType& cloud,
    trackingData& td,
    const scalar dt
)
{
    // Define local properties at beginning of time step
    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    const scalar np0 = this->nParticle_;
    const scalar mass0 = this->mass();

    // Store T for consistent radiation source
    const scalar T0 = this->T_;


    // Calc surface values
    // ~~~~~~~~~~~~~~~~~~~
    scalar Ts, rhos, mus, Pr, kappas;
    calcSurfaceValues(cloud, td, this->T_, Ts, rhos, mus, Pr, kappas);

    // Reynolds number
    scalar Re = this->Re(rhos, this->U_, td.Uc(), this->d_, mus);


    // Sources
    // ~~~~~~~

    // Explicit momentum source for particle
    vector Su = Zero;

    // Linearised momentum source coefficient
    scalar Spu = 0.0;

    // Momentum transfer from the particle to the carrier phase
    vector dUTrans = Zero;

    // Explicit enthalpy source for particle
    scalar Sh = 0.0;

    // Linearised enthalpy source coefficient
    scalar Sph = 0.0;

    // Sensible enthalpy transfer from the particle to the carrier phase
    scalar dhsTrans = 0.0;


    // Heat transfer
    // ~~~~~~~~~~~~~
```

```
    // Sum Ni*Cpi*Wi of emission species
    scalar NCpW = 0.0;

    // Calculate new particle temperature
    this->T_ =
        this->calcHeatTransfer
        (
            cloud,
            td,
            dt,
            Re,
            Pr,
            kappas,
            NCpW,
            Sh,
            dhsTrans,
            Sph
        );


    // Motion
    // ~~~~~~

    // Calculate new particle velocity
    this->U_ =
        this->calcVelocity(cloud, td, dt, Re, mus, mass0, Su, dUTrans, Spu);


    //  Accumulate carrier phase source terms
    // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    if (cloud.solution().coupled())
    {
        // Update momentum transfer
        cloud.UTrans()[this->cell()] += np0*dUTrans;

        // Update momentum transfer coefficient
        cloud.UCoeff()[this->cell()] += np0*Spu;

        // Update sensible enthalpy transfer
        cloud.hsTrans()[this->cell()] += np0*dhsTrans;

        // Update sensible enthalpy coefficient
        cloud.hsCoeff()[this->cell()] += np0*Sph;

        // Update radiation fields
        if (cloud.radiation())
        {
            const scalar ap = this->areaP();
            const scalar T4 = pow4(T0);
            cloud.radAreaP()[this->cell()] += dt*np0*ap;
            cloud.radT4()[this->cell()] += dt*np0*T4;
            cloud.radAreaPT4()[this->cell()] += dt*np0*ap*T4;
        }
    }
}


template<class ParcelType>
template<class TrackCloudType>
Foam::scalar Foam::MyThermoParcel<ParcelType>::calcHeatTransfer
(
    TrackCloudType& cloud,
    trackingData& td,
    const scalar dt,
    const scalar Re,
    const scalar Pr,
    const scalar kappa,
    const scalar NCpW,
```

```
    const scalar Sh,
    scalar& dhsTrans,
    scalar& Sph
)
{
    if (!cloud.heatTransfer().active())
    {
        return T_;
    }

    const scalar d = this->d();
    const scalar rho = this->rho();
    const scalar As = this->areaS(d);
    const scalar V = this->volume(d);
    const scalar m = rho*V;

    // Calc heat transfer coefficient
    scalar htc = cloud.heatTransfer().htc(d, Re, Pr, kappa, NCpW);

    // Calculate the integration coefficients
    const scalar bcp = htc*As/(m*Cp_);
    const scalar acp = bcp*td.Tc();
    scalar ancp = Sh;
    if (cloud.radiation())
    {
        const tetIndices tetIs = this->currentTetIndices();
        const scalar Gc = td.GInterp().interpolate(this->coordinates(), tetIs);
        const scalar sigma = physicoChemical::sigma.value();
        const scalar epsilon = cloud.constProps().epsilon0();

        ancp += As*epsilon*(Gc/4.0 - sigma*pow4(T_));
    }
    if (cloud.constProps().internalHeatOfReaction())
    {
        ancp += dhsTrans / dt;
    }
    ancp /= m*Cp_;

    // Integrate to find the new parcel temperature
    const scalar deltaT = cloud.TIntegrator().delta(T_, dt, acp + ancp, bcp);
    const scalar deltaTncp = ancp*dt;
    const scalar deltaTcp = deltaT - deltaTncp;

    // Calculate the new temperature and the enthalpy transfer terms
    scalar Tnew = T_ + deltaT;
    Tnew = min(max(Tnew, cloud.constProps().TMin()), cloud.constProps().TMax());

    if (cloud.constProps().internalHeatOfReaction())
    {
        dhsTrans = -m*Cp_*deltaTcp;
    }
    else
    {
        dhsTrans -= m*Cp_*deltaTcp;
    }

    Sph = dt*m*Cp_*bcp;

    if (debug)
    {
        Info << "MyThermoParcel v2: Adding heat of reaction "
            << (
                    cloud.constProps().internalHeatOfReaction()
                ? "internally"
                : "externally"
               )
            << nl;
    }
```

```
    return Tnew;
}


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

template<class ParcelType>
Foam::MyThermoParcel<ParcelType>::MyThermoParcel
(
    const MyThermoParcel<ParcelType>& p
)
:
    ParcelType(p),
    T_(p.T_),
    Cp_(p.Cp_)
{}


template<class ParcelType>
Foam::MyThermoParcel<ParcelType>::MyThermoParcel
(
    const MyThermoParcel<ParcelType>& p,
    const polyMesh& mesh
)
:
    ParcelType(p, mesh),
    T_(p.T_),
    Cp_(p.Cp_)
{}


// * * * * * * * * * * * * * * * IOStream operators  * * * * * * * * * * * * * * //

#include "MyThermoParcelIO.C"

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

B.1.5 basicMyHeterogeneousReactingCloud.H

src/lagrangian/intermediate/clouds/derived/basicMyHeterogeneousReactingCloud/basicMyHeterogeneousReactingCloud.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2018-2019 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/>.
```

```
Class
    Foam::basicMyHeterogeneousReactingCloud

Description
    Cloud class to introduce heterogeneou reacting parcels

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

#ifndef basicMyHeterogeneousReactingCloud_H
#define basicMyHeterogeneousReactingCloud_H

#include "Cloud.H"
#include "KinematicCloud.H"
#include "ThermoCloud.H"
#include "ReactingCloud.H"
#include "ReactingHeterogeneousCloud.H"
#include "basicMyHeterogeneousReactingParcel.H"

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

namespace Foam
{
    typedef ReactingHeterogeneousCloud
    <
        ReactingCloud
        <
            ThermoCloud
            <
                KinematicCloud
                <
                    Cloud
                    <
                        basicMyHeterogeneousReactingParcel
                    >
                >
            >
        >
    > basicMyHeterogeneousReactingCloud;
}

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

#endif

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

B.1.6 basicMyHeterogeneousReactingParcel.H

src/lagrangian/intermediate/parcels/derived/basicMyHeterogeneousReactingParcel/basicMyHeterogeneousReactingParcel.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2018-2019 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/>.

Class
    Foam::basicMyHeterogeneousReactingParcel

Description
    Definition of reacting heterogeneous parcel

SourceFiles
    basicMyHeterogeneousReactingParcel.C

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

#ifndef basicMyHeterogeneousReactingParcel_H
#define basicMyHeterogeneousReactingParcel_H

#include "contiguous.H"
#include "particle.H"
#include "KinematicParcel.H"
#include "MyThermoParcel.H"
#include "ReactingParcel.H"
#include "ReactingHeterogeneousParcel.H"

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

namespace Foam
{
    typedef ReactingHeterogeneousParcel
    <
        ReactingParcel
        <
            MyThermoParcel
            <
                KinematicParcel
                <
                    particle
                >
            >
        >
    > basicMyHeterogeneousReactingParcel;

    //- Non-contiguous data for basicMyHeterogeneousReactingParcel
    template<>
    struct is_contiguous<basicMyHeterogeneousReactingParcel> : std::false_type {};
}

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

#endif

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

B.1.7 defineBasicMyHeterogeneousReactingParcel.C

src/lagrangian/intermediate/parcels/derived/basicMyHeterogeneousReactingParcel/defineBasicMyHeterogeneousReactingParcel.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2018 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 "basicMyHeterogeneousReactingParcel.H"
#include "Cloud.H"

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

namespace Foam
{
    defineTemplateTypeNameAndDebug(basicMyHeterogeneousReactingParcel, 0);
    defineTemplateTypeNameAndDebug(Cloud<basicMyHeterogeneousReactingParcel>, 0);
}


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

B.1.8 makeBasicMyHeterogeneousReactingParcelSubmodels.C

src/lagrangian/intermediate/parcels/derived/basicMyHeterogeneousReactingParcel/makeBasicMyHeterogeneousReactingParcelSubmodels.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2018-2021 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 "basicMyHeterogeneousReactingCloud.H"

#include "makeReactingParcelCloudFunctionObjects.H"

// Kinematic
#include "makeThermoParcelForces.H" // thermo variant
#include "makeParcelDispersionModels.H"
#include "makeReactingParcelInjectionModels.H" // Reacting variant
#include "makeParcelPatchInteractionModels.H"
#include "makeParcelStochasticCollisionModels.H"
#include "makeReactingParcelSurfaceFilmModels.H" // Reacting variant
#include "makeHeterogeneousReactingParcelHeterogeneousReactingModels.H"

// Thermodynamic
#include "makeParcelHeatTransferModels.H"

// Reacting
#include "makeReactingMultiphaseParcelCompositionModels.H"
#include "makeReactingParcelPhaseChangeModels.H"

// MPPIC sub-models
#include "makeMPPICParcelDampingModels.H"
#include "makeMPPICParcelIsotropyModels.H"
#include "makeMPPICParcelPackingModels.H"

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

makeReactingParcelCloudFunctionObjects(basicMyHeterogeneousReactingCloud);

// Kinematic sub-models
makeThermoParcelForces(basicMyHeterogeneousReactingCloud);
makeParcelDispersionModels(basicMyHeterogeneousReactingCloud);
makeReactingParcelInjectionModels(basicMyHeterogeneousReactingCloud);
makeParcelPatchInteractionModels(basicMyHeterogeneousReactingCloud);
makeParcelStochasticCollisionModels(basicMyHeterogeneousReactingCloud);
makeReactingParcelSurfaceFilmModels(basicMyHeterogeneousReactingCloud);

// Thermo sub-models
makeParcelHeatTransferModels(basicMyHeterogeneousReactingCloud);

// Reacting sub-models
makeReactingMultiphaseParcelCompositionModels(basicMyHeterogeneousReactingCloud);
makeReactingParcelPhaseChangeModels(basicMyHeterogeneousReactingCloud);
makeHeterogeneousReactingParcelHeterogeneousReactingModels
(
    basicMyHeterogeneousReactingCloud
);

// MPPIC sub-models
makeMPPICParcelDampingModels(basicMyHeterogeneousReactingCloud);
makeMPPICParcelIsotropyModels(basicMyHeterogeneousReactingCloud);
makeMPPICParcelPackingModels(basicMyHeterogeneousReactingCloud);

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

B.1.9 basicMyReactingParcel.H

src/lagrangian/intermediate/parcels/derived/basicMyReactingParcel/
basicMyReactingParcel.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2016 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/>.

Class
    Foam::basicMyReactingParcel

Description
    Definition of basic reacting parcel

SourceFiles
    basicMyReactingParcel.C

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

#ifndef basicMyReactingParcel_H
#define basicMyReactingParcel_H

#include "contiguous.H"
#include "particle.H"
#include "KinematicParcel.H"
#include "MyThermoParcel.H"
#include "ReactingParcel.H"

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

namespace Foam
{
    typedef ReactingParcel<MyThermoParcel<KinematicParcel<particle>>>
        basicMyReactingParcel;

    //- Non-contiguous data for basicMyReactingParcel
    template<> struct is_contiguous<basicMyReactingParcel> : std::false_type {};
}

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

#endif

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

B.1.10 defineBasicMyReactingParcel.C

src/lagrangian/intermediate/parcels/derived/basicMyReactingParcel/
defineBasicMyReactingParcel.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011 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/>.

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

#include "basicMyReactingParcel.H"
#include "Cloud.H"

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

namespace Foam
{
    defineTemplateTypeNameAndDebug(basicMyReactingParcel, 0);
    defineTemplateTypeNameAndDebug(Cloud<basicMyReactingParcel>, 0);
}


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

B.1.11 basicMyThermoParcel.H

src/lagrangian/intermediate/parcels/derived/basicMyThermoParcel/
basicMyThermoParcel.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2016 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/>.

Class
    Foam::basicMyThermoParcel

Description
    Definition of basic thermo parcel

SourceFiles
    basicMyThermoParcel.C

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

#ifndef basicMyThermoParcel_H
#define basicMyThermoParcel_H

#include "contiguous.H"
#include "particle.H"
#include "KinematicParcel.H"
#include "MyThermoParcel.H"

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

namespace Foam
{
    typedef MyThermoParcel<KinematicParcel<particle>> basicMyThermoParcel;

    //- Contiguous data for basicMyThermoParcel
    template<> struct is_contiguous<basicMyThermoParcel> : std::true_type {};
}

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

#endif

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

B.1.12 defineBasicMyThermoParcel.C

src/lagrangian/intermediate/parcels/derived/basicMyThermoParcel/defineBasicMyThermoParcel.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | www.openfoam.com
     \\/     M anipulation  |
-------------------------------------------------------------------------------
    Copyright (C) 2011-2013 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/>.

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

#include "basicMyThermoParcel.H"
#include "Cloud.H"

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

namespace Foam
{
    defineTemplateTypeNameAndDebug(basicMyThermoParcel, 0);
    defineTemplateTypeNameAndDebug(Cloud<basicMyThermoParcel>, 0);
}

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

B.2 Solver

B.2.1 reactingHeterogenousParcelFoam.C

applications/solvers/lagrangian/reactingParcelFoam/reactingHeterogenousParcelFoam/reactingHeterogenousParcelFoam.C

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2018-2019 OpenCFD Ltd.
     \\/     M anipulation  |
-------------------------------------------------------------------------------
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/>.

Application
    reactingHeterogenousParcelFoam

Group
    grpLagrangianSolvers

Description
    Transient solver for the coupled transport of a single kinematic particle
    cloud including the effect of the volume fraction of particles on the
```

```
    continuous phase. Multi-Phase Particle In Cell (MPPIC) modeling is used to
    represent collisions without resolving particle-particle interactions.

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

#define CLOUD_BASE_TYPE MyHeterogeneousReacting
#define CLOUD_BASE_TYPE_NAME "MyHeterogeneousReacting"

#include "reactingParcelFoam.C"

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

Index