Cite as: Boyao, W, F.: Implement a mesh-based particle model for the coalCombustionFoam for solving the biomass combustion. In Proceedings of CFD with OpenSource Software, 2021, Edited by Nilsson. H.,
http://dx.doi.org/10.17196/OS_CFD#YEAR_2021

CFD with OpenSource software

A course at Chalmers University of Technology
Taught by Håkan Nilsson

Implement a mesh-based particle model for the coalCombustionFoam for solving the biomass combustion

Developed for OpenFOAM-7
Requires: coalChemistryFoam, Meschach

Author:
Boyao Wang
Norwegian University of Science and Technology
boyao.wang@ntnu.no

Peer reviewed by:
Terese Løvås
Tian Li
Jingyuan Zhang
Ilya Morev
Mohammad Hossein Arabnejad

Licensed under CC-BY-NC-SA, https://creativecommons.org/licenses/

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.

February 9, 2022

Learning outcomes

The main requirement 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:

- Implement a mesh-based particle model for the coalCombustionFoam for simulating biomass combustion.

The theory of it:

- A description of the already existing particle model
- A general description of the mesh-based particle model
- CFD-DEM method used in OpenFOAM

How it is implemented:

- The cloud and parcel library design in OpenFOAM will be described, followed by the implementation of the mesh-based particle model.

How to modify it:

- A new parcel-level templated class will be created. The class will contain member data and functions that are related to the particle physical and chemical information. The model will be verified with the previous single-particle simulation.

Prerequisites

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

- How to run a tutorial case of coalChemistryFoam
- Knowledge of solving a linear system using direct and iterative methods
- How to compile a user-defined library and solver in OpenFOAM
- Knowledge of biomass combustion

Contents

Nomenclature

Acronyms

CFD	Computational Fluid Dynamics
DEM	Discrete Element Method
IBM	interface-based particle model
MBM	mesh-based particle model

English symbols

Bi	Biot number	$(-)$
C_p	Particle heat capacity	J/kgK
h_t	External heat transfer coefficient	W/m^2K
L_c	Particle characteristic length	m
m_p	Particle mass	kg
S_{energy}	source term of energy equation	J/s
T_s	Particle temperature	K
V_p	Particle volume	kg

Greek symbols

Γ	Particle shape factor	$(-)$
λ	Heat transfer coefficient	W/mK

Chapter 1

Introduction

1.1 Background

With the diminution of fossil fuel, biomass becomes gradually interesting in both scientific and industrial senses. Biomass (wood, agricultural residues, forestry residues, energy crops, etc.), a renewable fuel, has the advantage of being neutral in greenhouse gas emissions compared to fossil fuels through photosynthesis reactions [1]. My PhD topic "Numerical Modeling and Simulation of Efficient Biocarbon production", mainly focuses on the biocarbon (also known as biochar or charcoal) production through thermal conversion. The produced biochar can have a very high carbon content (90 %), meaning that biocarbon is a suitable reductant for the metallurgical process.

The thermal conversion of the biomass, e.g., combustion and pyrolysis, is a thermochemical process. In other words, heat transfer and chemical reactions occur in the biomass simultaneously. The chemical reaction rates depend highly on the temperature of the biomass particle. If the biomass particle is not isothermal and has an internal temperature gradient, the reaction rate will vary inside the particle. The dimensionless number, Biot number (Bi), can be used to understand which thermal regime the particle is in [2]. Biot number measures the relative importance of internal and external heat transfer. It is defined as

$$Bi = \frac{h_t L_c}{\lambda} \tag{1.1}$$

where h_t, L_c, and λ represents the external convective heat transfer coefficient, the characteristic length of the particle, and the internal heat conductive transfer coefficient, respectively. The particles can be classified into thermally thin and thick particles by Biot number. The thermally thin assumption is valid while $Bi < 0.1$. However, in biomass pyrolysis and combustion, the biomass particles can have large geometry and have to be considered as thermally thick particles. For thermally thick particles, the temperature gradient inside the particle is too large to be neglected, which means the point particle assumption is not valid.

The lagrangian method (for the solid phase) used in OpenFOAM is defined under the `src/lagrangian` folder. The particle that has been used in the OpenFOAM solver, e.g., coalChemestryFoam, is defined as a point particle, meaning that the particle is isothermal. This assumption is not valid for thermally thick particles. To address this shortcoming, the mesh-based particle model (MBM) [3] will be implemented in the OpenFOAM `lagrangian` library.

1.2 Theory of the Mesh-based particle model

The concept of the MBM, as shown in Figure 1.1, is developed based on the interface-based model (IBM) [4, 5]. The particle is divided into four layers: wet wood, dry wood, char, and ash. The raw biomass is considered as wet wood. When it is starting to be heated up, the wet wood will be dried up into dry wood, where pyrolysis occurs. Conversion of the biochar occurs at the char layer front.

The solid products, ash, are added to the ash layer, and the gas products will be released to the gas phase.

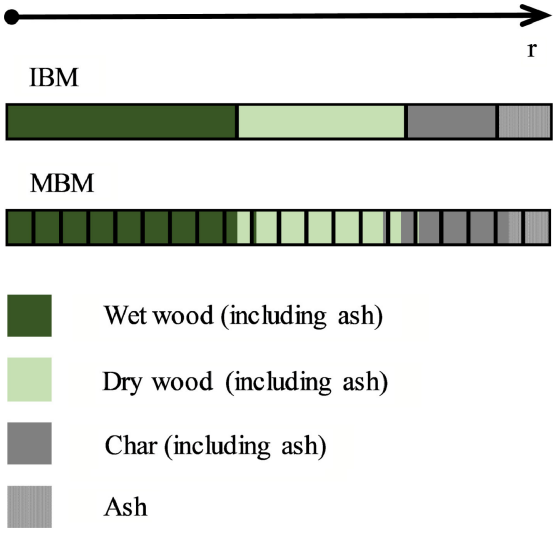

Figure 1.1: Schematics of one-dimensional MBM (Reproduced with permission of [3])

In the current MBM, the energy conservation is solved by the 1D finite volume method implicitly. The mass transport is not considered as the gas produced by thermochemical conversion is assumed to release to the gas phase instantaneously. The governing equation of the energy balance is:

$$m_p C_p \frac{\partial T_s}{\partial t} = \frac{V_p}{\Gamma} \frac{\partial}{\partial r} (\Gamma_\kappa \frac{\partial T_s}{\partial r}) + S_{energy} \tag{1.2}$$

where m_p is the particle mass, C_p is the particle heat capacity, T_s is the particle temperature, V_p is the particle volume, Γ is the particle shape factor, κ is the particle thermal conductivity, and the S_{energy} is the source term due to the thermochemical conversion processes (drying, pyrolysis, and char combustion).

Eq. (1.2) is solved by LU factorization with a matrix solver `Meschach` [6]. The discretization method is second-order central finite volume method. `Meschach` is a C language library, which can perform matrix computations. The code used by Li [3] to solve Eq. (1.2) is written as a stand-alone C language program by using the `Meschach` library. This report will discuss how to port this C language code into the OpenFOAM `lagrangian` library to solve a mesh-based particle instead of a point particle.

1.3 Outline

To give the reader a brief understanding of the story and to let the reader reproduce this exercise, the report is structured as:

- Chapter 1 provides a brief introduction to the physical phenomena that will be simulated and the theory of the mesh-based particle model (MBM).
- Chapter 2 will discuss how the particle model is solved in OpenFOAM solver. A walkthrough of the code will be addressed from the particle object instantiation to particle model calculation in `coalChemistryFoam` solver.
- Chapter 3 will show how the MBM written in C language is ported into the OpenFOAM `lagrangian` library.
- Chapter 4 will provide how to set up a simple case and conduct the code verification by comparing the results with the C language code.

Chapter 2

Codes of coalCloud particle

This section will discuss the particle object instantiation and how the particle model is calculated in the coalChemistryFoam solver. In OpenFOAM, particles are included in the parcel and parcels are included in the cloud. The particles used in coalChemistryFoam have a type of coalParcel, which is nested inside the coalCloud. The definition of coalParcel and coalCloud utilizes a C++ technique called parameterised inheritance. A similar inheritance method also exists in the turbulence model class of OpenFOAM. The discussion of the code will be limited in OpenFOAM-7.

2.1 The coalCloud and parameterised inheritance

In the solver coalChemistryFoam, the object of the particle cloud is called coalParcels and is declared in $FOAM_SOLVERS/lagrangian/coalChemistryFoam/createClouds.H. This object has a coalCloud type and is declared as follows:

createClouds.H

```
Info<< "\nConstructing coal cloud" << endl;
coalCloud coalParcels
(
    "coalCloud1",
    rho,
    U,
    g,
    slgThermo
);
```

The type coalCloud is defined in $FOAM_SRC/lagrangian/coalCombustion/coalCloud/coalCloud.H as:

where the type coalParcel is defined in $FOAM_SRC/lagrangian/coalCombustion/coalParcel/coalParcel.H/ as:

coalParcel.H

```
    typedef ReactingMultiphaseParcel
    <
        ReactingParcel
        <
            ThermoParcel
            <
                KinematicParcel
                <
                    particle
                >
            >
        >
    > coalParcel;
```

The types coalCloud and coalParcel are defined with typedef. The definition of the coalCloud type will be discussed first to let the reader understand the structural design of the code.

From the declaration of the coalCloud type, the coalCloud type is defined based on the ReactingMultiphaseCloud class, with the instantiation within the angular bracket. In other words, the coalCloud type is actually a ReactingMultiphaseCloud class with specific instantiation of its template parameters. The ReactingMultiphaseCloud class, which is a templated class, is defined as follows:

ReactingMultiphaseCloud.H

```
template<class CloudType>
class ReactingMultiphaseCloud
:
    public CloudType,
    public reactingMultiphaseCloud
{
...
}
```

The ReactingMultiphaseCloud class is inherited not only from an abstract class called reactingMultiphaseCloud but also its template type parameter CloudType. The CloudType, as shown in the declaration of coalCloud, has a type of:

```
        ReactingCloud
        <
            ThermoCloud
            <
                KinematicCloud
                <
                    Cloud
                    <
                        coalParcel
                    >
                >
            >
        >
```

The aforementioned type is the type parameter substituted for the ReactingMultiphaseCloud template parameter list. If we check how the object coalParcels is declared from its parameter list, the constructor of the class ReactingMultiphaseCloud will be called:

```
template<class CloudType>
Foam::ReactingMultiphaseCloud<CloudType>::ReactingMultiphaseCloud
(
    const word& cloudName,
    const volScalarField& rho,
    const volVectorField& U,
    const dimensionedVector& g,
    const SLGThermo& thermo,
    bool readFields
)
:
    CloudType(cloudName, rho, U, g, thermo, false),
    reactingMultiphaseCloud(),
    cloudCopyPtr_(nullptr),
    constProps_(this->particleProperties()),
    devolatilisationModel_(nullptr),
    surfaceReactionModel_(nullptr),
    dMassDevolatilisation_(0.0),
    dMassSurfaceReaction_(0.0)
{
   ...
}
```

And in the constructor of the ReactingMultiphaseCloud, the constructor of the CloudType class, from which the ReactingMultiphaseCloud class inherits, will be called. The parent class

CloudType as discussed above has a type of ReactingCloud, which means the constructor of the ReactingCloud class will be called. If we go to the definition of the ReactingCloud class:

```
template<class CloudType>
class ReactingCloud
:
    public CloudType,
    public reactingCloud
{
   ...
}
```

it is seen that the declaration of the ReactingCloud class is similar to that of the ReactingMultiphaseCloud class. The ReactingCloud class also inherits from an abstract class called reactingCloud and a class which type is CloudType. If we peel off another layer of the nested template, we will see this CloudType should have a type of:

```
            ThermoCloud
            <
                KinematicCloud
                <
                    Cloud
                    <
                        coalParcel
                    >
                >
            >
```

Again, the constructor of the ThermoCloud class will be called while constructing the ReactingCloud class. And the reader will also find that the ThermoCloud class is inherited from both an abstract class called thermoCloud and its template parameter CloudType, which is the KinematicCloud.

This method is called parameterised inheritance in C++. The reader can follow this coding logic to see how each constructor in the nested template type coalCloud is called up until the innermost class of the template.

2.2 The calc function

The calc function defined in the ReactingMultiphaseParcel will calculate the thermochemical conversion of the particle. The mesh-based particle model will also be implemented in the calc function. First, we will take a look how this function is called from the coalChemistryFoam solver. In the coalChemistryFoam.C, the coalParcels will be evolved:

```
int main(int argc, char *argv[])
{
...
    while (runTime.run())
    {
    ...
        coalParcels.evolve();
    ...
    }
...
}
```

The evolve() function of the ReactingMultiphaseCloud class will be called:

```
template<class CloudType>
void Foam::ReactingMultiphaseCloud<CloudType>::evolve()
{
    if (this->solution().canEvolve())
    {
        typename parcelType::trackingData td(*this);
```

```
7
8           this->solve(*this, td);
9       }
10  }
```

Because the ReactingMultiphaseCloud class does not declare the solve() function, the reader need to find the function from the inherited class. The solve() function is found in the class KinematicCloud:

```
1   template<class CloudType>
2   template<class TrackCloudType>
3   void Foam::KinematicCloud<CloudType>::solve
4   (
5       TrackCloudType& cloud,
6       typename parcelType::trackingData& td
7   )
8   {
9       if (solution_.steadyState())
10      {
11          ...
12      }
13      else
14      {
15          ...
16          evolveCloud(cloud, td);
17          ...
18      }
19      ...
20  }
```

Since we are looking at how the calc function is called, other functions that are called in this solve function will not be discussed. If we take a look into the function evolveCloud:

```
1   template<class CloudType>
2   template<class TrackCloudType>
3   void Foam::KinematicCloud<CloudType>::evolveCloud
4   (
5       TrackCloudType& cloud,
6       typename parcelType::trackingData& td
7   )
8   {
9       ...
10      if (solution_.transient())
11      {
12          ...
13          cloud.motion(cloud, td);
14          ...
15      }
16      else
17      {
18          ...
19      }
20  }
```

the motion function of the object cloud will be called:

```
1   template<class CloudType>
2   template<class TrackCloudType>
3   void Foam::KinematicCloud<CloudType>::motion
4   (
5       TrackCloudType& cloud,
6       typename parcelType::trackingData& td
7   )
8   {
9       td.part() = parcelType::trackingData::tpLinearTrack;
10      CloudType::move(cloud, td, solution_.trackTime());
11
12      updateCellOccupancy();
```

```
}
```

In the `motion` function, the `CloudType::move` function will be called. From the declaration of the type `coalCloud`, this `CloudType` should be `Cloud`. Furthermore, the `move` function in the `Cloud` class will be called:

```
template<class ParticleType>
template<class TrackCloudType>
void Foam::Cloud<ParticleType>::move
(
    TrackCloudType& cloud,
    typename ParticleType::trackingData& td,
    const scalar trackTime
)
{
    ...
    // While there are particles to transfer
    while (true)
    {
        ...
        // Loop over all particles
        forAllIter(typename Cloud<ParticleType>, *this, pIter)
        {
            ParticleType& p = pIter();

            // Move the particle
            bool keepParticle = p.move(cloud, td, trackTime);
            ...
        }

        ...
    }
}
```

In this `move` function, inside the `while` loop, an object called `p` with a type of `ParticleType` is created. From the declaration of the type `coalCloud` and `coalParcel`, the `ParticleType` here has a type of `ReactingMultiphaseParcel`. However, in the `ReactingMultiphaseParcel` class, there is no function called `move`. Therefore, we have to find the `move` function in its parent class. From the inherited classes, the `move` function is found in the `KinematicParcel` class:

```
template<class ParcelType>
template<class TrackCloudType>
bool Foam::KinematicParcel<ParcelType>::move
(
    TrackCloudType& cloud,
    trackingData& td,
    const scalar trackTime
)
{
    typename TrackCloudType::parcelType& p =
        static_cast<typename TrackCloudType::parcelType&>(*this);
    typename TrackCloudType::parcelType::trackingData& ttd =
        static_cast<typename TrackCloudType::parcelType::trackingData&>(td);

    ...

    while (ttd.keepParticle && !ttd.switchProcessor && p.stepFraction() < 1)
    {
        ...
        // Avoid problems with extremely small timesteps
        if (dt > rootVSmall)
        {
            ...
            p.calc(cloud, ttd, dt);
        }

        ...
```

```
    }

    return ttd.keepParticle;
}
```

In the code above, the object p has a type of TrackCloudType::parcelType. The type of TrackCloudType is the same as that of the function parameter cloud. To determine the type of cloud, return to the above-mentioned evolve function declared in the ReactingMultiphaseCloud class. In this evolve function, the *this is substituted in the function this->solve. The varible *this, which has a type of ReactingMultiphaseCloud, has been transferred through the functions mentioned above to this calc function as the cloud parameter. Therefore, the TrackCloudType here has the same type of *this, which is the ReactingMultiphaseCloud. Now, we are going to check how the parcelType is defined in the class ReactingMultiphaseCloud.

```
template<class CloudType>
class ReactingMultiphaseCloud
:
    public CloudType,
    public reactingMultiphaseCloud
{
public:

    // Public Typedefs

        //- Type of cloud this cloud was instantiated for
        typedef CloudType cloudType;

        //- Type of parcel the cloud was instantiated for
        typedef typename CloudType::particleType parcelType;

        ...
};


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

} // End namespace Foam
```

In the ReactingMultiphaseCloud class, the parcelType is defined as CloudType::particleType. From the declaration of the coalCloud type, the CloudType in this ReactingMultiphaseCloud class is ReactingCloud. In the ReactingCloud class, there is no declaration of the type particleType. Therefore, we have to find the declaration of the particleType in other inherited classes. The declaration of the particleType is found in the class Cloud:

```
template<class ParticleType>
class Cloud
:
    public cloud,
    public IDLList<ParticleType>
{
    ...
public:

    ...
    typedef ParticleType particleType;
    ...
};


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

} // End namespace Foam

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

The particleType has a type of ParticleType, which is the template parameter of the Cloud class. From the declaration of the type coalCloud, this ParticleType is coalParcel. According to the declaration of the type coalParcel, this ParticleType is ReactingMultiphaseParcel. If we go back to see where the calc function is called, p.calc(cloud, ttd, dt), the calc function in the ReactingMultiphaseParcel class is called. In this calc, the thermochemical conversion of the particle will be calculated. Therefore, we will add our mesh-based particle model in the calc function. In chapter 3, the implementaion of the mesh-based particle model will be introduced.

Chapter 3

Code implementation

In the previous chapter, we introduced the structure and design of particle evolution codes in OpenFOAM. This chapter will show how to implement mesh-based particle models (MBMs) by modifying the `coalCombustion` library of OpenFOAM. The new library is called `biomassCombustion`. A new solver call `biomassChemistryMBMFoam` will also be made. The MBM solver in this report can be found in github link. An interface-based method (IBM) model has been implemented in OpenFOAM before in our research group. The reader can find this solver in the a Github repository (link).

The MBM is solved by using the external linear system solver library. Three libraries have been chosen in this work, which are OpenFOAM, `Eigen`, and the `Meschach` library. A short introduction is provided to the use and compilation of each external linear system solver library.

3.1 The biomassCombustion library

The `biomassCombustion` library is created by modifying the `coalCombustion` library. First, we copy the `coalCombustion` folder to our working folder and rename `coal` to `biomass`:

```
cp -r $FOAM_SRC/lagrangian/coalCombustion/ .
rename 's/coalCombustion/biomassCombustion/' *
cd biomassCombustion
rename 's/coal/biomass/' *
find . -maxdepth 3 -name '*coal*' -exec rename 's/coal/biomass/ if -f;' {} \;
find . -maxdepth 3 -name '*Coal*' -exec rename 's/Coal/Biomass/ if -f;' {} \;
find . -maxdepth 3 -type f -exec sed -i 's/coal/biomass/g' {} \;
find . -maxdepth 3 -type f -exec sed -i 's/Coal/Biomass/g' {} \;
```

After the copying and renaming procedures, the structure of `biomassCombustion` is as follows:

```
biomassCloud
└── biomassCloud.H
biomassCloudList
├── biomassCloudList.C
├── biomassCloudList.H
└── biomassCloudListI.H
biomassParcel
├── biomassParcel.H
└── makeBiomassParcelSubmodels.C
include
└── makeBiomassParcelSurfaceReactionModels.H
Make
├── files
└── options
submodels
└── surfaceReactionModel
```

The type `coalCloud` and `coalParcel` are defined based on the `ReactingMultiphaseCloud` and the `ReactingMultiphaseParcel` class, which use the point particle model and are defined in the `intermediate` directory in the `lagrangian` directory. To implement the mesh-based particle model (MBM), two new classes, `ReactingMultiphaseMBMCloud` and `ReactingMultiphaseMBMParcel`, are created according to the code style of `intermediate`. To create these two new classes, we will copy the related file that was used to build the `ReactingMultiphaseCloud` and `ReactingMultiphaseParcel` classes. And to mimic the file structure of `parcels` and `clouds` directories within the `intermediate` folder, we have to move the `include` folder to the `biomassParcel` folder. Furthermore, the new folders called `derived` and `Templates` have to be created in the `biomassParcel` folder. In the `biomassClouds` directory, the `baseClasses`, `derived`, and `Templates` folders have to be created. The file structure of the `parcels` and `clouds` directories are shown as follows:

The `clouds` and `parcels` directory

```
clouds                  parcels
├── baseClasses         ├── derived
├── derived             ├── include
└── Templates           └── Templates
```

Following the code structure listed above, after modification, the `biomassCombustion` should have a structure as follows:

The `biomassCombustion` library structure

```
biomassCloud
├── baseClasses
├── derived
└── Templates
biomassCloudList
└── biomassMBMCloudList
biomassParcel
├── derived
├── include
└── Templates
Make
├── files
└── options
submodels
└── surfaceReactionModel
```

The following sections will show how to modify the structure of the `biomassCombustion` library.

3.1.1 Modify the `biomassCloud` directory

This section shows the changes executed in the directory `biomassCloud`. First, the reader can use the following command to change the structure of the directory `biomassCloud`:

```
cd biomassCloud
mkdir baseClasses derived Templates
mv biomassCloud.H biomassMBMCloud.H
mv biomassMBMCloud.H ./derived/
```

Copy the `basicReactingMultiphaseCloud` and `ReactingMultiphaseCloud` to the `derived` and `Templates` folders, which are located in the `biomassCloud` directory.

```
lagrangianCloud=$FOAM_SRC/lagrangian/intermediate/clouds
cp -r $lagrangianCloud/baseClasses/reactingMultiphaseCloud/ baseClasses/
cp -r $lagrangianCloud/derived/basicReactingMultiphaseCloud/ derived/
cp -r $lagrangianCloud/Templates/ReactingMultiphaseCloud/ Templates/
```

Rename the `basicReactingMultiphaseCloud` and `ReactingMultiphaseCloud` for files and folders, and substitute the `ReactingMultiphase` with `ReactingMultiphaseMBM` for all files:

```
find . -maxdepth 3 -name '*MultiphaseCloud*' -type f \
-execdir rename 's/MultiphaseCloud/MultiphaseMBMCloud/ if -f;' {} \;

find . -maxdepth 3 -name '*MultiphaseCloud*' -type d \
-execdir rename 's/MultiphaseCloud/MultiphaseMBMCloud/' {} \;

find . -maxdepth 3 -type f \
-exec sed -i 's/MultiphaseCloud/MultiphaseMBMCloud/g' {} \;
```

The comments above can bring some warnings, but it will finish the renaming work. The file structure modification in the directory `biomassCloud` has been completed.

3.1.2 Modify the `biomassMBMCloud.H`

The `biomassMBMCloud` type is defined as follows:

```
#ifndef biomassMBMCloud_H
#define biomassMBMCloud_H

#include "Cloud.H"
#include "KinematicCloud.H"
#include "ThermoCloud.H"
#include "ReactingCloud.H"
#include "ReactingMultiphaseCloud.H"
#include "ReactingMultiphaseMBMCloud.H"
#include "biomassMBMParcel.H"

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

namespace Foam
{
  typedef ReactingMultiphaseMBMCloud
  <
      ReactingMultiphaseCloud
      <
          ReactingCloud
          <
              ThermoCloud
              <
                  KinematicCloud
                  <
                      Cloud
                      <
                          biomassMBMParcel
                      >
                  >
              >
          >
      >
  >  biomassMBMCloud;
}

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

#endif

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

From the above declaration, we have defined the type `biomasMBMCloud` based on the class `ReactingMultiphaseMBMCloud`, with the instantiation within the angular bracket. This declaration allows the `biomassCloud` to inherit all classes defined in the angle bracket. As the `*Cloud` classes all use the parameterized inheritance method, which has been discussed in Chapter 2, that allowing these classes to have the ability to inherit from their template instantiation variables.

3.1.3 Modify the `basicReactingMultiphaseMBMCloud.H`

The `basicReactingMultiphaseMBMCloud` type is defined as follows:

```
#ifndef basicReactingMultiphaseMBMCloud_H
#define basicReactingMultiphaseMBMCloud_H

#include "Cloud.H"
#include "KinematicCloud.H"
#include "ThermoCloud.H"
#include "ReactingCloud.H"
#include "ReactingMultiphaseCloud.H"
#include "ReactingMultiphaseMBMCloud.H"
#include "basicReactingMultiphaseMBMParcel.H"

```

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

namespace Foam
{
    typedef ReactingMultiphaseMBMCloud
    <
        ReactingMultiphaseCloud
        <
            ReactingCloud
            <
                ThermoCloud
                <
                    KinematicCloud
                    <
                        Cloud
                        <
                            basicReactingMultiphaseMBMParcel
                        >
                    >
                >
            >
        >
    > basicReactingMultiphaseMBMCloud;
}

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

#endif
```

As mentioned in the declaration of the `biomassMBMCloud` type, the code above will declare a new type called `ReactingMultiphaseMBMCloud`, which inherits all classes that are defined within the angle brackets.

3.1.4 Modify the `biomassParcel` directory

This section begins by modifying the stuctrue of the directory `biomassParcel`. The following commands are used to modify the file structure in the directory `biomassParcel`:

```
cd ..
mv include/ biomassParcel/
cd biomassParcel/
mkdir derived Templates
```

The `basicReactingMultiphaseParcel` and `ReactingMultiphaseParcel` will be copied to the `derived` and `Templates` folders, which are located in the `biomassParcel` directory, using the following command:

```
lagrangianParcel=$FOAM_SRC/lagrangian/intermediate/parcels
cp -r $lagrangianParcel/derived/basicReactingMultiphaseParcel/ derived/
cp -r $lagrangianParcel/Templates/ReactingMultiphaseParcel Templates/
```

Rename the `basicReactingMultiphaseParcel` and `ReactingMultiphaseParcel` for files and folders, and substitute the `ReactingMultiphase` to `ReactingMultiphaseMBM` for all files:

```
cd derived
mv basicReactingMultiphaseParcel basicReactingMultiphaseMBMParcel
rename 's/ReactingMultiphaseParcel/ReactingMultiphaseMBMParcel/' \
 ./basicReactingMultiphaseMBMParcel/*
sed -i 's/ReactingMultiphase/ReactingMultiphaseMBM/' \
./basicReactingMultiphaseMBMParcel/basicReactingMultiphaseMBMParcel.H
sed -i 's/ReactingMultiphase/ReactingMultiphaseMBM/' \
./basicReactingMultiphaseMBMParcel/defineBasicReactingMultiphaseMBMParcel.C
sed -i 's/basicReactingMultiphaseCloud/basicReactingMultiphaseMBMCloud/' \
./basicReactingMultiphaseMBMParcel/\
makeBasicReactingMultiphaseMBMParcelSubmodels.C

cd ../Templates
mv ReactingMultiphaseParcel ReactingMultiphaseMBMParcel
rename 's/ReactingMultiphaseParcel/ReactingMultiphaseMBMParcel/' \
./ReactingMultiphaseMBMParcel/*
sed -i 's/ReactingMultiphase/ReactingMultiphaseMBM/g' \
./ReactingMultiphaseMBMParcel/*
```

Go back to the `biomassParcel` folder and move the `biomassParcel.H` to the `derived` folder. Rename the `biomassParcel.H` to `biomassMBMParcel.H`.

```
cd ..
mv biomassParcel.H derived/biomassMBMParcel.H
```

Move the `makeBiomassParcelSubmodels.C` to the `include` folder and rename the file name from `makeBiomassParcelSubmodels.C` to `makeBiomassMBMParcelSubmodels.C`. At the same time, replace the word `biomass` with `biomassMBM` in the files of this directory.

```
mv makeBiomassParcelSubmodels.C include/
rename 's/BiomassParcel/BiomassMBMParcel/' include/*
sed -i 's/makeBiomassParcelSurfaceReactionModels/\
makeBiomassMBMParcelSurfaceReactionModels/g' include/*
sed -i 's/biomassCloud/biomassMBMCloud/g' include/*
```

3.1.5 Modify the `biomassMBMParcel.H`

The `biomassMBMParcel` type is defined as follows:

```
1  #ifndef biomassMBMParcel_H
2  #define biomassMBMParcel_H
3
4  #include "contiguous.H"
5  #include "particle.H"
6  #include "KinematicParcel.H"
7  #include "ThermoParcel.H"
8  #include "ReactingParcel.H"
9  #include "ReactingMultiphaseParcel.H"
10 #include "ReactingMultiphaseMBMParcel.H"
11
12 // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
13
14 namespace Foam
15 {
16     typedef ReactingMultiphaseMBMParcel
17     <
```

```
        ReactingMultiphaseParcel
        <
            ReactingParcel
            <
                ThermoParcel
                <
                    KinematicParcel
                    <
                        particle
                    >
                >
            >
        >
    > biomassMBMParcel;

    template<>
    inline bool contiguous<biomassMBMParcel>()
    {
        return false;
    }
}

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

#endif

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

The reason for declaring the type `biomassMBMParcel` as above can refer to the `biomassMBMCloud` type declaration.

3.1.6 Modify the `basicReactingMultiphaseMBMParcel.H`

The `basicReactingMultiphaseMBMParcel` type is defined as follows:

```
#ifndef basicReactingMultiphaseMBMParcel_H
#define basicReactingMultiphaseMBMParcel_H

#include "contiguous.H"
#include "particle.H"
#include "KinematicParcel.H"
#include "ThermoParcel.H"
#include "ReactingParcel.H"
#include "ReactingMultiphaseParcel.H"
#include "ReactingMultiphaseMBMParcel.H"

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

namespace Foam
{
    typedef ReactingMultiphaseMBMParcel
    <
        ReactingMultiphaseParcel
        <
            ReactingParcel
            <
                ThermoParcel
                <
                    KinematicParcel
                    <
                        particle
                    >
                >
            >
        >
    > basicReactingMultiphaseMBMParcel;

```

```
    template<>
    inline bool contiguous<basicReactingMultiphaseMBMParcel>()
    {
        return false;
    }
}


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

#endif

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

The reason for declaring the type `ReactingMultiphaseMBMParcel` as above can refer to the `ReactingMultiphaseMBMCloud` type declaration.

3.1.7 Modify the `biomassCloudList` directory

Go to the `biomassCloudList` directory and make a folder called `biomassMBMCloudList`.

```
cd ../biomassCloudList/
mkdir biomassMBMCloudList
```

Move files to `biomassMBMCloudList` folder and rename the files. Replace the word `biomass` with `biomassMBM` in the files of this directory.

```
mv biomassCloudList* biomassMBMCloudList/
cd biomassMBMCloudList/
rename 's/biomassCloud/biomassMBMCloud/' *
sed -i 's/biomassCloud/biomassMBMCloud/g' *
```

The file structure modification in the directory `biomassParcel` has been completed.

3.1.8 Modify the `ReactingMultiphaseMBMCloud` class

No new functions and members need to be added to this `ReactingMultiphaseMBMCloud` class. In `ReactingMultiphaseMBMCloud.H`, the following codes need to be removed in case of code conflicts:

```
            //- Devolatilisation model
            autoPtr
            <
                DevolatilisationModel<ReactingMultiphaseMBMCloud<CloudType>>
            >
            devolatilisationModel_;

            //- Surface reaction model
            autoPtr
            <
                SurfaceReactionModel<ReactingMultiphaseMBMCloud<CloudType>>
            >
            surfaceReactionModel_;
```

The following codes in the `setModels()` function have to be removed in case of code conflicts:

```
    devolatilisationModel_.reset
    (
        DevolatilisationModel<ReactingMultiphaseMBMCloud<CloudType>>::New
        (
            this->subModelProperties(),
            *this
        ).ptr()
```

```
    );

    surfaceReactionModel_.reset
    (
        SurfaceReactionModel<ReactingMultiphaseMBMCloud<CloudType>>::New
        (
            this->subModelProperties(),
            *this
        ).ptr()
    );
```

Since these submodels have already been declared in the ReactingMultiphaseCloud, if they are declared again, they will cause conflicts during code compilation.

3.2 Modify the ReactingMultiphaseMBMParcel class

The mesh-based particle model will be implemented in the ReactingMultiphaseMBMParcel class. The in-house mesh-based code used in paper [3], was written in C language. In this report, we will port the C language code in to the OpenFOAM C++ library.

3.2.1 Add members and member functions

Before adding new members and member functions, old members and unnecessary member functions have to be removed. The reader can compare the file in the ReactingMultiphaseParcel folder and ReactingMultiphaseMBMParcel folder to see what has been removed.

After removing the unnecessary members and member functions, some members and member functions will be added to the ReactingMultiphaseMBMParcel class. Some implementations of the members and member functions are omitted in the text due to report space limitations. Please check the ReactingMultiphaseMBMParcel.H in Appendix A.1. Meanwhile, to access some of these members, access functions need to be added. The access functions are declared in the public part of the ReactingMultiphaseMBMParcel class and defined in ReactingMultiphaseMBMParcelI.H as inline functions.

After adding the members and access functions, some helper functions, which will be used in the mesh-based particle model, are added in the private part of the ReactingMultiphaseMBMParcel class. These functions are declared in ReactingMultiphaseMBMParcel.H and defined in a file called externalFunctions.H (see Appendix A.1). These helper functions are used for calculating the physical models explained in paper [3]. The externalFunctions.H will be included in ReactingMultiphaseMBMParcel.C to make sure they are properly compiled.

Since the linear system solver Meschach is written in C, the reader must add a keyword, the extern "C", to C++ if the reader want to combine C code and C++ code. In ReactingMultiphaseMBMParcel.H, the following codes are added before the declaration of the ReactingMultiphaseMBMParcel class to include the C code defined in matrix2.h.

```
#ifdef __cplusplus
extern "C"
{
  #endif

  #include "matrix2.h"

  #ifdef __cplusplus
}
#endif
```

3.2.2 Modify the calc function

The calc function in ReactingMultiphaseMBMParcel class is the main calculation function for the mesh-based particle model. After adding a mesh-based particle model, the calculation function

becomes very large, so we create a separate file called `MBMParcelMainCalcFunction.H` to include the calculation function. By omitting part of the code, the `calc` function can be shown as follows.

```

template<class ParcelType>
template<class TrackCloudType>
void Foam::ReactingMultiphaseMBMParcel<ParcelType>::calc
(
    TrackCloudType& cloud,
    trackingData& td,
    const scalar dt
)
{
  // set the TInf_ as gas temperature
  TInf_ = td.Tc();

  // initialize the time to zero
  tPar = 0.0;

  // initialize the total mass of the particle to zero
  mParSum_ = 0.0;

  // set up mesh particle time step
  scalar t_step = TIME_DT_;//cloud.constProps().deltaTime();

  // read the emissivity of the particle
  scalar emissivity = cloud.constProps().epsilon0();

  // * * * * * * * end initialization and start calculation * * * * * * * * * //
  while (tPar < dt)//t_stop)
  {
    /* Initialize matrix system */
    m_zero(A);
    v_zero(b);
    ...
    /* Solve matrix system */
    /* ------------------------------------------------------------------- */
    LU = m_get(A->m, A->n);
    LU = m_copy(A, LU);
    pivot = px_get(A->m);
    LUfactor(LU, pivot);
    Tp_ = LUsolve(LU, pivot, b, Tp_);
    PX_FREE(pivot);
    M_FREE(LU);
    ...
    iter_count = iter_count+1;
    /* ------------------------------------------------------------------- */

    //- Update time
    tPar = tPar + t_step;
    ...

    /* ------------------------------------------------------------------- */
  }
  // To sum up all the mPars
  forAll(mPar_,i)
  {
    mParSum_ += sum(mPar_[i]);
  }
}
```

Due to most parts of the code are ported from the in-house C language code, we will briefly introduce this `calc` function. Before the `while` loop, some initializations are done. Inside the `while` loop, the mesh-based particle model will be calculated. The `while` loop ends until `tPar` (particle simulation time) is smaller than `dt` (the gas phase simulation time), the step-fractioned gas-phase time step. It has to be mentioned here that `dt` is not equal to the gas phase time step. Inside the `while` loop, remember to include the ommiting part of the code. In this `calc` function example, the

Meschach library is used to solve the Eq.(1.2). Other linear system solvers could also be used here to solve the Eq.(1.2), which will be introduced in section 3.3.

3.2.3 Modification to the constructors

The constructor of a class is always used to initialize the members of a class and its inherited class. As too many members have been created as the requirement of the mesh-based particle model, we create two new member functions called `readCoeff` and `initFields` to read data from the running folder and initialize the members. The definition can be found in `externalFunction.H`, see appendix A.1. An example for the modification to the constructor in `ReactingMultiphaseMBMParcelI.H` is shown as below:

```
template<class ParcelType>
inline Foam::ReactingMultiphaseMBMParcel<ParcelType>::ReactingMultiphaseMBMParcel
(
    const polyMesh& mesh,
    const barycentric& coordinates,
    const label celli,
    const label tetFacei,
    const label tetPti
)
:
    ParcelType(mesh, coordinates, celli, tetFacei, tetPti)
{
    readCoeff(mesh.time());
    initFields();
}
```

The above code inserts two new functions, `readCoeff()` and `initFields()`, into the body of the `ReactingMultiphaseMBMParcel` constructor to read data and initialize members. Similar changes to the constructor in `ReactingMultiphaseMBMParcelIO.C` and `ReactingMultiphaseMBMParcel.C` have to be done. The reader can refer to the supplied source code to see the difference.

3.2.4 Modify the `ReactingMultiphaseMBMParcelIO.C`

Some modifications to `ReactingMultiphaseMBMParcelIO.C` must be made to read and write files during simulation and ensure that the code can start at any latest time. There are two functions in the `ReactingMultiphaseMBMParcelIO.C` that take care of reading and writing files to the running folder. Function `readFields` reads while the function `writeFields` writes. An example of how the field `VCell` is read from the running files shows as follows:

```
template<class ParcelType>
template<class CloudType, class CompositionType>
void Foam::ReactingMultiphaseMBMParcel<ParcelType>::readFields
(
    CloudType& c,
    const CompositionType& compModel
)
{
    bool valid = c.size();

    ParcelType::readFields(c, compModel);

    IOField<scalarField> VCell(c.fieldIOobject("VCell", IOobject::READ_IF_PRESENT), valid);

    ...

    label i = 0;
    forAllIter(typename Cloud<ReactingMultiphaseMBMParcel<ParcelType> >, c, iter)
    {
        ReactingMultiphaseMBMParcel<ParcelType>& p = iter();

        p.VCell_ = VCell[i];
```

```
        ...
    }
}
```

In the above code, the `c.size()` will check whether a particle exists and return a value to the Boolean object `valid`. `ParcelType::readFields(c, compModel)` will call the `readFields` function of the inherited class. Then an object called `VCell`, which has a type of `IOField<scalarField>`, is created. This `IOField` object will read the file, which has a name of "VCell" in the `lagrangian` folder in the running case folder when the particles exist. The `forAllIter` will loop all particles in the cloud and give each particle the value read from the running case folder.

The field `VCell` will be used as an example to show how it is written to the running folder:

```
template<class ParcelType>
template<class CloudType, class CompositionType>
void Foam::ReactingMultiphaseMBMParcel<ParcelType>::writeFields
(
    const CloudType& c,
    const CompositionType& compModel
)
{
    ParcelType::writeFields(c, compModel);

    label np = c.size();
    {
        IOField<scalarField> VCell(c.fieldIOobject("VCell", IOobject::NO_READ), np);

        ...

        label i = 0;
        forAllConstIter(typename Cloud<ReactingMultiphaseMBMParcel<ParcelType> >, c, iter)
        {
            const ReactingMultiphaseMBMParcel<ParcelType>& p = iter();
            VCell[i] = p.VCell_;

            ...

        }

        VCell.write(np > 0);
        ...
    }
}
```

This `writeFields` function will first call the `writeFields` function of its inherited class. Then an object `np` is created to count the number of particles. The `VCell` object with a type of `IOField` is created, which has a read option of `IOobject::NO_READ`. In the `forAllConstIter` loop, each element of `VCell` will be substituted with the `VCell_` of each particle. After the `forAllConstIter`, the `VCell` will be written to the run folder using a function `write`.

3.3 The linear system solver libraries

This section will introduce the external linear system solver libraries we have chosen to solve the Eq.(1.2) and how to compile these libraries with OpenFOAM solver. We will also compare the efficiency of each solver in each library in Chapter 4.

3.3.1 The `Meschach` library

The original in-house C language code solves the Eq.(1.2) by using a matrix computation library (`Meschach` [6]), which is also written in C language. This library can be downloaded using the following command:

```
wget https://homepage.divms.uiowa.edu/~dstewart/meschach/mesch12b.tar.gz
```

Make a folder called `mesch12b` and unpack the downloaded file:

```
mkdir mesch12b
tar -xzf mesch12b.tar.gz -C mesch12b
```

To compile this C language library together with the OpenFOAM solver without conflicts, the `-fPIC` flag must be added to the `CFLAGS` variable in the `makefile`. To compile the `Meschach` library, go to the `mesch12b` folder and follow the commands (on a `Ubuntu` machine):

```
./configure --with-all
make all
make clean
```

After the compilation, a file called `meschach.a` will be generated in the `mesch12b` folder. Remember to copy the `mesch12b` folder to the `biomassCombustion` folder as it is needed for compilation. Here is a typical and general example of using the `Meschach` to solve linear system using LU decomposition method:

```
main()
{
    MAT  *A, *Acopy;
    VEC  *b, *x;
    PERM *pivot;

    // fill in A and b
    pivot = px_get(A->m);  /* get pivot permutation -- size = # rows of A */
    LUfactor(A,pivot);     /* LU factorisation */
    x = LUsolve(A,pivot,b,VNULL);  /* ... and solve A.x = b; result is in x */
    PX_FREE(pivot);
    M_FREE(LU);
}
```

Instead of the LU decomposition method, QR decomposition method can also be chosen. An example of the use of QR decomposition in the `Meschach` library is as follows:

```
main()
{
    MAT *QR;
    VEC *diag, *x, *b,;
     // fill in A and b
    QR = m_get(A->m,A->n);
    QR = m_copy(A,QR);
    diag = v_get(A->n);
    QRfactor(QR,diag); /* QR factorisation */
    T2 = QRsolve(QR,diag,b,x); /* ... and solve A.x = b; result is in x */
    M_FREE(QR);
    V_FREE(diag);
}
```

3.3.2 The `OpenFOAM` simple matrix solver library

In OpenFOAM, the linear system can also be solved by using the LU decomposition or QR decomposition. The following code shows an example of using the LU and QR decomposition method solving a linear system in OpenFOAM:

```
    scalarSquareMatrix squareMatrix(FILL_IN_THE_MATRIX_SIZE, Zero);
    scalarField source(FILL_IN_THE_SOURCETERM_SIZE, 0.0);

    // fill in the squareMatrix and source

    LUscalarMatrix LU(squareMatrix);
    scalarField x1(LU.solve(source));
```

```
8
9       QRMatrix<scalarSquareMatrix> QR(squareMatrix);
10      scalarField x2(QROF.solve(sourceOF));
```

Examples of how to use the decomposition method LU and QR in OpenFOAM can be found in `$FOAM_APP/test/Matrix/Test-Matrix.C`.

3.3.3 The Eigen library

Due to the descretization of Eq.(1.2) results in a sparse matrix with only diagonals and its neighbors, using LU decomposition will be costly when the matrix becomes larger. Therefore, we introduce another library called Eigen to solve this linear system using the iterative method.

This library can be downloaded using the following command:

```
wget https://gitlab.com/libeigen/eigen/-/archive/3.4.0/eigen-3.4.0.tar.gz
```

In terms of compilation, as there is no library to link to, the user does not need to pre-compile the `Eigen` library. Remember to copy the `eigen-3.4.0` folder to the `biomassCombustion` and `biomassChemistryFoam` folder as it is needed for compilation. Here is a typical and general example of using the `Eigen` library to solve the linear system using the iterative method:

```
#include <Eigen/RequiredModuleName>
// ...
SparseMatrix<double> A;
// fill A
VectorXd b, x;
// fill b
// solve Ax = b
SolverClassName<SparseMatrix<double> > solver;
solver.compute(A);
if(solver.info()!=Success) {
  // decomposition failed
  return;
}
x = solver.solve(b);
if(solver.info()!=Success) {
  // solving failed
  return;
}
// solve for another right hand side:
x1 = solver.solve(b1);
```

3.4 The Make file in biomassCombustion library

This section will show how to modify the `files` and `options` defined in the `Make` folder of the `biomassCombustion` library. The `files` is shown as:

```
/* Biomass parcel and sub-models */
BIOMASSPARCEL=biomassParcel
BIOMASSCLOUD=biomassCloud
BIOMASSBASECLOUDS=$(BIOMASSCLOUD)/baseClasses

DERIVEDPARCEL=$(BIOMASSPARCEL)/derived

/* Cloud base classes */
$(BIOMASSBASECLOUDS)/reactingMultiphaseMBMCloud/reactingMultiphaseMBMCloud.C

/* reacting multiphase MBM parcel sub-models */
REACTINGMPMPMPARCEL=$(DERIVEDPARCEL)/basicReactingMultiphaseMBMParcel
$(REACTINGMPMPMPARCEL)/defineBasicReactingMultiphaseMBMParcel.C
$(REACTINGMPMPMPARCEL)/makeBasicReactingMultiphaseMBMParcelSubmodels.C

$(BIOMASSPARCEL)/include/makeBiomassMBMParcelSubmodels.C

biomassCloudList/biomassMBMCloudList/biomassMBMCloudList.C

LIB = $(FOAM_USER_LIBBIN)/libbiomassCombustion
```

First, this is a user-defined library, hence the compiled library should locate at `FOAM_USER_LIBBIN` instead of `FOAM_LIBBIN`. The `*.C` files are included to tell the compiler which files need to be compiled. The `options`, which is used to include files and other libraries, is shown as below:

```
EXE_INC = \
    -Wno-old-style-cast \
    -Wno-sign-compare \
    -I$(LIB_SRC)/lagrangian/basic/lnInclude \
    -I$(LIB_SRC)/lagrangian/intermediate/lnInclude \
    -I$(LIB_SRC)/lagrangian/distributionModels/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/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)/radiationModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/incompressible/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(LIB_SRC)/transportModels \
    -I$(LIB_SRC)/regionModels/regionModel/lnInclude \
    -I$(LIB_SRC)/regionModels/surfaceFilmModels/lnInclude \
    -I$(LIB_SRC)/dynamicFvMesh/lnInclude \
    -I$(LIB_SRC)/sampling/lnInclude \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude

LIB_LIBS = \
    -llagrangian \
    -llagrangianIntermediate \
    -llagrangianTurbulence \
    -ldistributionModels \
    -lspecie \
    -lcompressibleTransportModels \
    -lfluidThermophysicalModels \
    -lthermophysicalProperties \
    -lreactionThermophysicalModels \
    -lSLGThermo \
    -lradiationModels \
    -lturbulenceModels \
    -lincompressibleTurbulenceModels \
    -lcompressibleTurbulenceModels \
    -lincompressibleTransportModels \
    -lregionModels \
    -lsurfaceFilmModels \
    -ldynamicFvMesh \
    -lsampling \
    -lfiniteVolume \
    -lmeshTools \
    ./mesch12b/meschach.a
```

As there are old style C++ code in the `Eigen` library, we add the `-Wno-old-style-cast` and `-Wno-sign-compare` flags to get rid of the warnings shown during the compilation. To link the `meschach` library, we add `./mesch12b/meschach.a` in the `options` file. After all these steps, we can compile the `biomassCombustion` library using `wmake`.

3.5 Modify the biomassChemistryFoam

This section explains how the solver coalChemistryFoam is modified to biomassChemistryFoam.

3.5.1 Modify the source codes

Using the following commands to copy the coalChemistryFoam solver from OpenFOAM and rename the main file coalChemistryFoam to biomassChemistryFoam:

```
cd biomassChemistryFoam
cp -r $FOAM_APP/solvers/lagrangian/coalChemistryFoam .
mv coalChemistryFoam biomassChemistryFoam
mv coalChemistryFoam.C biomassChemistryFoam.C
```

Remove the rhoEffLagrangian in createFields.H. In createCloud.H, the old clouds objects are removed and the object biomassMBMParcels which is a biomassMBMCloud is created.

```
Info<< "\nConstructing biomassMBM cloud" << endl;
biomassMBMCloud biomassMBMParcels
(
    "biomassMBMCloud",
    rho,
    U,
    g,
    slgThermo
);
```

Rename the cloud in pEqn.H and rhoEqn.H:

```
sed -i 's/coal/biomassMBM/g' pEqn.H rhoEqn.H
```

Remove limestoneParcels in EEqn.H, UEqn.H, setRDeltaT.H, and biomassChemistryFoam.C. After this, remove rhoEffLagrangian in biomassChemistryFoam.C. The removal is done manually without using commands. Then, rename coal* to biomassMBM* in EEqn.H, UEqn.H, setRDeltaT.H, YEqn.H, and biomassChemistryFoam.C using the following command:

```
sed -i 's/coal/biomassMBM/g' EEqn.H UEqn.H setRDeltaT.H YEqn.H \
biomassChemistryFoam.C
```

At the end, the eigen-3.4.0 folder has to be copied into the biomassChemistryFoam folder as the Eigen solver is going to be used.

3.5.2 Modify the Make files

Modify the Make/files and make biomassChemistryFoam as a user-defined application:

```
sed -i 's/FOAM_APPBIN/FOAM_USER_APPBIN/g' Make/files
sed -i 's/coalChemistryFoam/biomassChemistryFoam/g' Make/files
```

After modification, the file Make/options is shown as follows:

```
EXE_INC = \
    -Wno-old-style-cast \
    -Wno-sign-compare \
    -I$(LIB_SRC)/finiteVolume/lnInclude \
    -I$(LIB_SRC)/meshTools/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/turbulenceModels/lnInclude \
    -I$(LIB_SRC)/TurbulenceModels/compressible/lnInclude \
    -I$(LIB_SRC)/lagrangian/basic/lnInclude \
    -I$(LIB_SRC)/lagrangian/intermediate/lnInclude \
    -I../biomassCombustion/lnInclude \
    -I$(LIB_SRC)/lagrangian/distributionModels/lnInclude \
    -I$(LIB_SRC)/thermophysicalModels/specie/lnInclude \
    -I$(LIB_SRC)/transportModels/compressible/lnInclude \
    -I$(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)/thermophysicalModels/chemistryModel/lnInclude \
    -I$(LIB_SRC)/radiationModels/lnInclude \
    -I$(LIB_SRC)/regionModels/regionModel/lnInclude \
    -I$(LIB_SRC)/regionModels/surfaceFilmModels/lnInclude \
    -I$(LIB_SRC)/ODE/lnInclude \
    -I$(LIB_SRC)/combustionModels/lnInclude \
    -I$(FOAM_SOLVERS)/combustion/reactingFoam \
    -I$(LIB_SRC)/sampling/lnInclude

EXE_LIBS = \
    -L$(FOAM_USER_LIBBIN) \
    -lfiniteVolume \
    -lmeshTools \
    -lturbulenceModels \
    -lcompressibleTurbulenceModels \
    -llagrangian \
    -llagrangianIntermediate \
    -llagrangianTurbulence \
    -lbiomassCombustion\
    -lspecie \
    -lcompressibleTransportModels \
    -lfluidThermophysicalModels \
    -lthermophysicalProperties \
    -lreactionThermophysicalModels \
    -lSLGThermo \
    -lchemistryModel \
    -lradiationModels \
    -lregionModels \
    -lsurfaceFilmModels \
    -lODE \
    -lcombustionModels \
    -lfvOptions \
    -lsampling
```

In the file above, `-Wno-old-style-cast` and `-Wno-sign-compare` flags are added to avoid warnings during the compilation. The following codes in the `Make/options` of `coalChemistryFoam`:

```
-I$(LIB_SRC)/lagrangian/coalCombustion/lnInclude \
...
-lcoalCombustion\
```

is changed to:

```
-I../biomassCombustion/lnInclude \
...
-lbiomassCombustion \
```

As the `-libbiomassCombustion.so` is now in the `FOAM_USER_LIBBIN`, the following codes have to be added to the `Make/options`:

```
-L$(FOAM_USER_LIBBIN) \
```

Remember to remove the `err.H` file before the compilation. Use the `wmake` command to compile the `biomassChemistryFoam` solver. Use the `Allwmake` in the supplied files.

```
#!/bin/bash
wmake -j biomassCombustion
rm -rf biomassCombustion/lnInclude/err.h
wmake -j biomassMBMChemistryFoam
```

Chapter 4

Tutorial case and discussion

In this chapter, we will first give the reader tutorial case as a code verification. Then, the comparison between different linear system solvers will be given specifically for this mesh-based particle case.

4.1 Single particle simulation

4.1.1 Case setup

In this tutorial case, a single particle combustion simulation will be conducted using the mesh-based particle model (MBM). The wood particle has a parameter of 9.5 mm, the ambient temperature is 1050 K. The parameters of the MBM are listed in a file called `particleInfoDict` in the `constant` folder. Here is an example of the `particleInfoDict`:

particleInfoDict

```
/*--------------------------------*- C++ -*----------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Version:  7
     \\/     M anipulation  |
\*---------------------------------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      particleInfoDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

meshInput
{
  VERBOSITY 0;
  SAVE_EVERY 10000;
  TIME_DT 1e-04;
  TIME_TOTAL 100;
  MOISTURE_FRACTION_DB 0.666667;
  PARTICLE_RADIUS 4.75E-03;
  DRYING_SHRINKAGE_VOLUME 0.127;
  T_REACTOR 1050.0;
  T_RADIATION 1276.0;
  T_P_INIT 300.0;
  MASS_FRAC_O2 0.23;
  DENSITY_INITIAL 844.0; // poplar wood
  FINE_PARTICLE_MESH 50;
  FINE_PARTICLE_MESH_DOUBLE 50.0;
```

```
  K_P_WET 0.26;
  K_P_DRY 0.12;
  K_P_CHAR 0.1;
  ASH_MASS_FRACTION 0.02;
  //daf in raw biomss the C H O ratios are from Neves review paper 2011 PECS
  //(poplar wood), char is only C.
  C_FRACTION 0.4941;
  H_FRACTION 0.0609;
  O_FRACTION 0.4353;
  CHARCONVERSION_SHRINKAGE_VOLUME 0.95;
  DEVOL_SHRINKAGE_VOLUME 0.28;
  P_ATM 101325.0; /* (Pa) */
  MASS_FRAC_H2O 0.0;
  MASS_FRAC_H2 0.0;
  MASS_FRAC_CO2 0.0;
  MY_MAXIMUM_NEWTON_ITER 500;
  N_SPECIES 12;
}

// single component kinetic ratio
sigDevKin
{
  DevKinA1 1.10e+11; /* (s-1) */
  DevKinE1 1.77e+05; /* (J/mol) */
  DevKinA2 9.28e+09; /* (s-1) */
  DevKinE2 1.49e+05; /* (J/mol) */
  DevKinA3 3.05e+07; /* (s-1) */
  DevKinE3 1.25e+05; /* (J/mol) */

  DryKinA 5.13e+10;
  DryKinE 88000.;
}

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

Table 4.1 describes the meaning of the parameters defined in the `meshInput` subdictionary of `particleInfoDict`. The subdictionary `sigKin` describes the single component kinetic coefficients. The `A` parameter stands for the pre-exponential factor, while the `E` parameter stands for the activation energy in the Arrhenius equation.

Table 4.1: particleInfoDict explanation

parameters	**explanation**
VERBOSITY	1 for true, 0 for false
SAVE_EVERY	save to file every 10000 iteration steps
TIME_DT	mesh-based particle model time step
MOISTURE_FRACTION_DB	dry based moisture contents of the wood particle
PARTICLE_RADIUS	wood particle radius
DRYING_SHRINKAGE_VOLUME	wood particle drying shrinkage coefficient
T_REACTOR	reactor temperature (ambient gas temperature)
T_RADIATION	radiation temperature (reactor wall temperature)
T_P_INIT	initial temperature of the wood particle
MASS_FRAC_O2	oxygen mass fraction
DENSITY_INITIAL	initial density of the wet (poplar) wood particle
FINE_PARTICLE_MESH	number of meshes (int form)
FINE_PARTICLE_MESH_DOUBLE	number of meshes (double form)
K_P_WET	wet wood heat conductivity
K_P_DRY	dry wood heat conductivity
K_P_CHAR	char heat conductivity
ASH_MASS_FRACTION	mass fraction of ash
C_FRACTION	C fraction of the wood particle

H_FRACTION	H fraction of the wood particle
O_FRACTION	O fraction of the wood particle
CHARCONVERSION_SHRINKAGE_VOLUME	char conversion shrinkage volume coefficient
DEVOL_SHRINKAGE_VOLUME	devolatilization shrinkage volume coefficient
P_ATM	ambient pressure
MASS_FRAC_H2O	H2O fraction in gas phase
MASS_FRAC_H2	H2 fraction in gas phase
MASS_FRAC_CO2	CO2 fraction in gas phase
MY_MAXIMUM_NEWTON_ITER	Maximum newton iteration steps
N_SPECIES	number of species

The same setup as described above is used in the in-house C language code to verify our implementation.

4.1.2 Code verification

Simulation of a single biomass particle combustion under the conditions described in the Section 4.1.1 was carried out by `biomassChemistryFoam` and the original C language code. Figure 4.1 shows the mass loss curve of a single biomass particle calculated by both the OpenFOAM and the C language code.

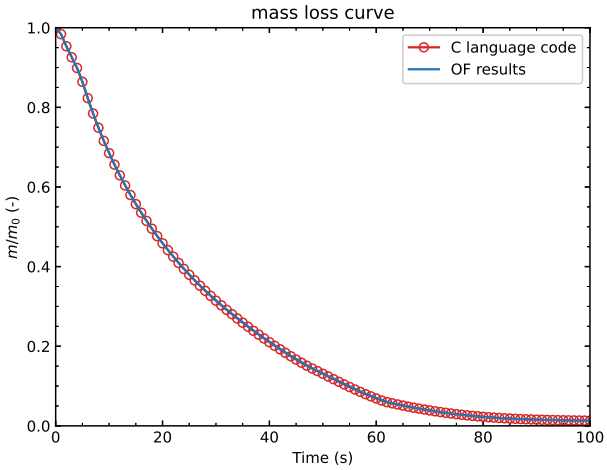

Figure 4.1: The mass loss curve calculated by OpenFOAM and the stand alone code [3]

Figure 4.2 shows the results of the center and surface temperature of the particle calculated by both OpenFOAM and the original C language code. Figure 4.1 and Figure 4.2 show that the code has been verified with the original C language code.

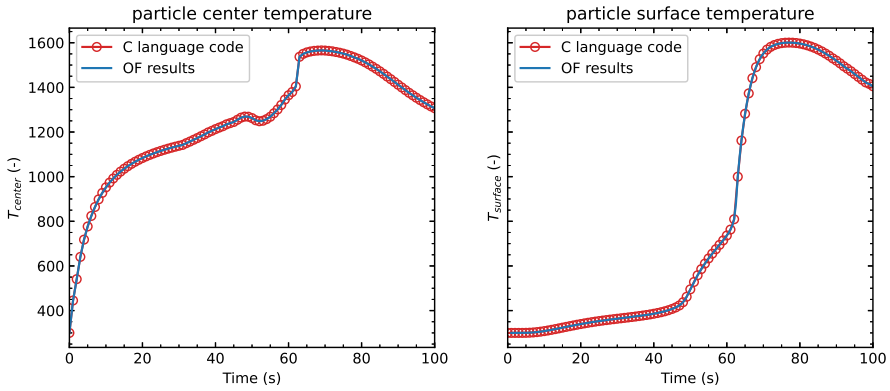

Figure 4.2: The mass loss curve calculated by OpenFOAM and the stand alone code [3]

4.2 Mesh convergence study

This section investigates the sensitivity of meshes. Same simulation conditions as described in Section 4.1.1 are used here. The number of meshes used in the study is 10, 20, 40, 80, and 160. Mass loss, surface and center temperature of the particle are used to study the mesh convergence as shown in Figure 4.3 and Figure 4.4

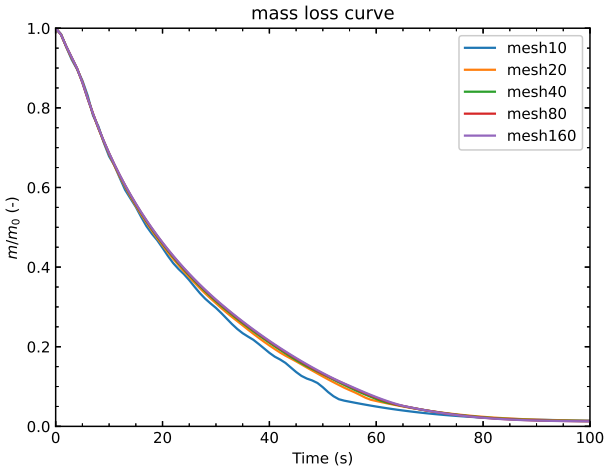

Figure 4.3: The mass loss curve when the number of meshes are 10, 20, 40, 80, and 160

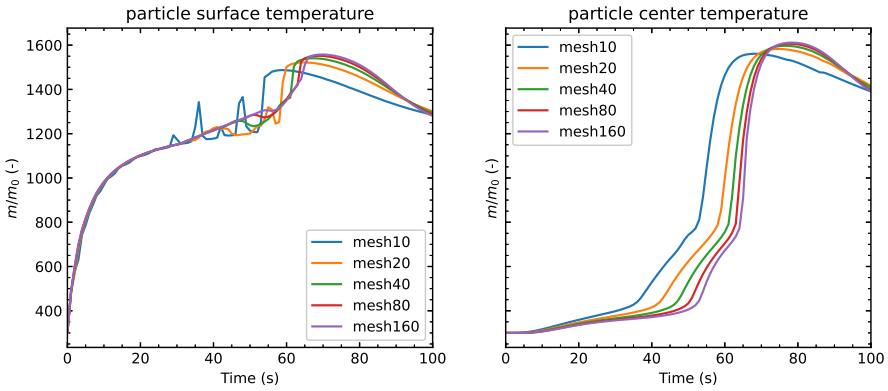

Figure 4.4: The center and surface temperature of the particle when the number of meshes are 10, 20, 40, 80, and 160

In terms of the mass loss curve shown in Figure 4.3, the results are already converged when the number of meshes is 40. However, the particle's surface and center temperature profile does not show a convergence when the number of meshes equals 40. This is due to the particle center temperature is the temperature profile of the innermost cell. The larger the inner cell is, the sooner the center temperature appears to rise inside the particle.

4.3 Linear system solver performance comparison

This section will compare different linear system solvers in different libraries. We use `valgrind` to do the profiling. To show the performance of each function, `kcachegrind` is used. The following commands can execute the requirement of profiling:

```
valgrind --tool=callgrind <executableName>
kcachegrind <file created by valgrind>
```

To test the performance of each method, we put all solvers in one file and let them solve the same linear system simultaneously. The following sections will discuss the comparison of the solvers from the point of view of the solution method, including LU decomposition, QR decomposition, Cholesky decomposition, and interactive methods.

4.3.1 LU decomposition solver comparison

The `meschach`, `OpenFOAM`, and `Eigen` solvers all have the LU decomposition method for solving the linear system. Figure 4.5 shows the performance of each solver when the number of meshes is equal to 10, 20, 40, and 80. The normalized time is calculated as the time used by the solver divided by the time used by all solvers.

Figure 4.5 tells that the LU decomposition solver of `OpenFOAM` is always faster than that of the `Meschach` solver. When the number of meshes increases, the relative performance of the `Eigen` solver increases. When the number of meshes increases to 80, the `Eigen` solver performs the best of all solvers.

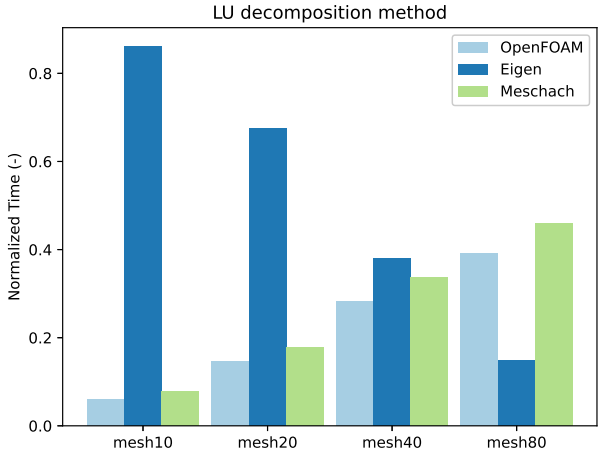

Figure 4.5: Performance of LU decomposition method in `OpenFOAM`, `Eigen`, and `Meschach` when the number of meshes is equal to 10, 20, 40, and 80.

4.3.2 QR decomposition solver comparison

The `meschach`, `OpenFOAM`, and `Eigen` solvers all have the QR decomposition method for solving the linear system. Figure 4.6 shows the performance of QR decomposition in `meschach` and `Eigen` when the number of meshes is equal to 10, 20, 40, and 80. As the QR decomposition in `OpenFOAM` performs too poor compared with other solvers, it is not shown in Figure.4.6. Figure 4.6 shows that when the number of meshes increases, the relative performance of the QR decomposition of `Eigen` increases relative to `Meschach`.

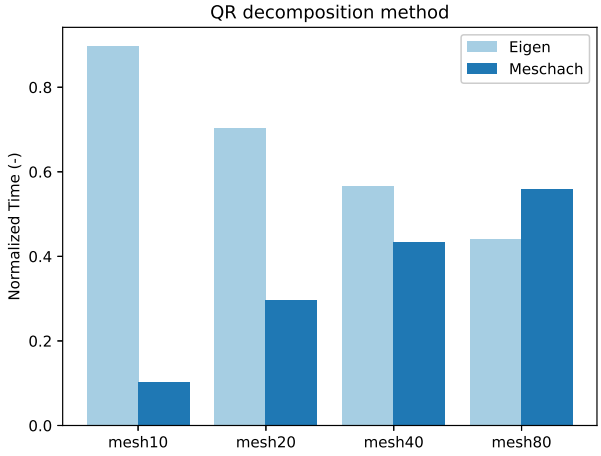

Figure 4.6: Performance of QR decomposition method in `Eigen` and `Meschach` when the number of meshes is equal to 10, 20, 40, and 80.

4.3.3 Cholesky decomposition (LLT) solver comparison

The `OpenFOAM` and `Eigen` solvers have the LLT decomposition method for solving the linear system. Figure 4.7 shows the performance of QR decomposition in `OpenFOAM` and `Eigen` when the number

of meshes is equal to 10, 20, 40, and 80.

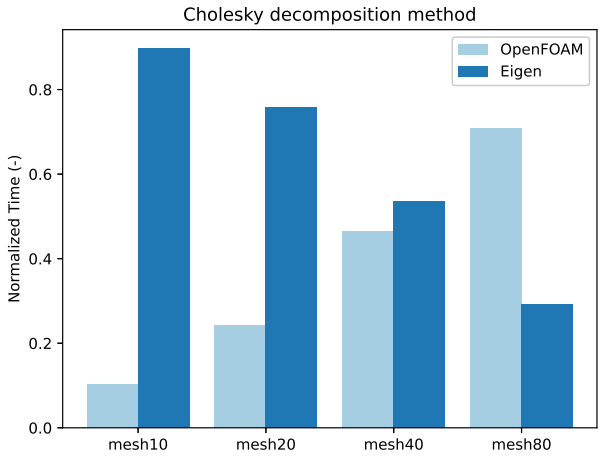

Figure 4.7: Performance of LLT decomposition method in `OpenFOAM` and `Eigen` when the number of meshes is equal to 10, 20, 40, and 80.

The LLT decomposition method of `Eigen` performs relatively poor when the mesh number is low. When the number of meshes increases to 80, it outperforms the `OpenFOAM` solver.

4.3.4 Iterative solver in `Eigen`

In `Eigen` solver, the linear system could be solved by using the iterative method (e.g., conjugate gradient method). Direct methods can perform slowly when operating large sparse matrices, while the iterative methods are good at solving these systems. This means that when the number of meshes increases, at a certain point, the iterative method performs better than the direct method.

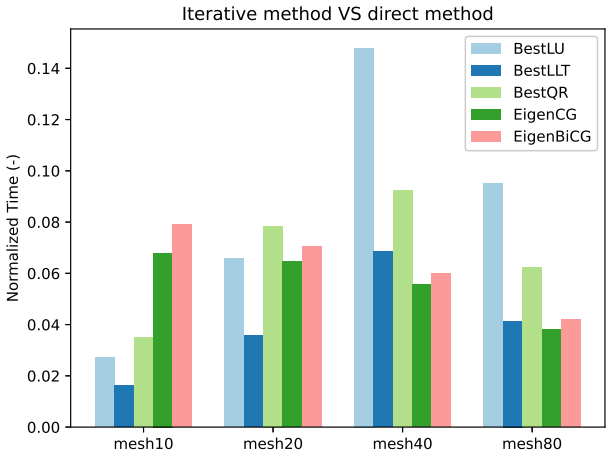

Figure 4.8: Performance of`Eigen` iterative method compared with the best LU, QR, and LLT method when number of meshes is equal to 10, 20, 40, and 80.

Figure 4.8 shows the comparison between the interactive method used in `Eigen` and the best direct method (LU, QR, and LLT) used in other solvers at different mesh numbers. The best direct methods at different mesh numbers are shown in Table 4.2.

Table 4.2: Best direct method at different mesh numbers

	LU	QR	LLT
mesh10	OpenFOAM	Meschach	OpenFOAM
mesh20	OpenFOAM	Meschach	OpenFOAM
mesh40	OpenFOAM	Meschach	OpenFOAM
mesh80	Eigen	Eigen	Eigen

Figure 4.8 tells that when the mesh number is lower than 20, the direct method performs better, especially the LLT method used in OpenFOAM. From the mesh convergence study in Section 4.2, 40 meshes will be sufficient for the mass loss study, while 80 meshes for the temperature profile study. When the number of meshes equals 40, all methods except for LU decomposition can be the choice. The best solver at this mesh number is the conjugate gradient (CG) method in `Eigen`. If the temperature profile is going to be studied, which means the mesh number has to be 80, the LLT in OpenFOAM solver or the CG in `Eigen` could be the choice. It should be mentioned that the CG in `Eigen` outperforms a bit the LLT in OpenFOAM.

Bibliography

[1] C. Di Blasi, "Combustion and gasification rates of lignocellulosic chars," *Progress in Energy and Combustion Science*, vol. 35, no. 2, pp. 121–140, 2009.

[2] K. M. Bryden, K. W. Ragland, and C. J. Rutland, "Modeling thermally thick pyrolysis of wood," *Biomass and Bioenergy*, vol. 22, no. 1, pp. 41–53, 2002.

[3] T. Li, H. Thunman, and H. Ström, "A fast-solving particle model for thermochemical conversion of biomass," *Combustion and Flame*, vol. 213, pp. 117–131, 2020.

[4] H. Ström and H. Thunman, "Cfd simulations of biofuel bed conversion: A submodel for the drying and devolatilization of thermally thick wood particles," *Combustion and Flame*, vol. 160, no. 2, pp. 417–431, 2013.

[5] H. Thunman, B. Leckner, F. Niklasson, and F. Johnsson, "Combustion of wood particles—a particle model for eulerian calculations," *Combustion and Flame*, vol. 129, no. 1-2, pp. 30–46, 2002.

[6] D. E. Stewart and Z. Leyk, *Meschach: Matrix computations in C*, vol. 32. Centre for Mathematics and its Applications, Australian National University . . . , 1994.

Study questions

1. What has to be modified in the Make folder to include an external library outside the OpenFOAM solver?
2. In which function of which class the mesh-based particle model is included?
3. How the parcel class read data from the cloud class?
4. If you have a limited amount of mesh, which method (direct or iterative) will you choose to solve the linear system when matrix A is a sparse matrix?
5. How does the parameterised inheritance method work in `coalCloud` type definition?

Appendix A

Developed codes

Because of the length of the code, only the code referred in the text is displayed in the appendix.

A.1 biomassParcel

Files mentioned in the text but not shown in the text in the `biomassParcel` folder are listed in this section.

A.1.1 ReactingMultiphaseMBMParcel.H

biomassParcel/Templates/ReactingMultiphaseMBMParcel/ReactingMultiphaseMBMParcel.H

```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     | Website:  https://openfoam.org
    \\  /    A nd           | Copyright (C) 2011-2019 OpenFOAM Foundation
     \\/     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/>.

Class
    Foam::ReactingMultiphaseMBMParcel

Description
    Multiphase variant of the reacting parcel class with one/two-way coupling
    with the continuous phase.

SourceFiles
    ReactingMultiphaseMBMParcelI.H
    ReactingMultiphaseMBMParcel.C
    ReactingMultiphaseMBMParcelIO.C
```

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

#ifndef ReactingMultiphaseMBMParcel_H
#define ReactingMultiphaseMBMParcel_H

// include Eigen solver
#include <eigen-3.4.0/Eigen/Dense>
#include <eigen-3.4.0/Eigen/Sparse>
#include <eigen-3.4.0/Eigen/IterativeLinearSolvers>

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

#ifdef __cplusplus
extern "C"
{
  #endif

  #include "matrix2.h"

  #ifdef __cplusplus
}
#endif

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

namespace Foam
{

template<class ParcelType>
class ReactingMultiphaseMBMParcel;

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

/*---------------------------------------------------------------------------*\
                Class ReactingMultiphaseMBMParcel Declaration
\*---------------------------------------------------------------------------*/

template<class ParcelType>
class ReactingMultiphaseMBMParcel
:
    public ParcelType
{
    // Private Data

        /* model parameters */
        #define MY_QUITE_SMALL 1.0e-3
        #define MY_VERY_SMALL 1.0e-6
        #define MY_EXTREMELY_SMALL 1.0e-20
        #define MY_QUITE_LARGE 1.0e3
        #define MY_VERY_LARGE 1.0e6
        // #define T_P_INIT 300.0
        // #define emissivity 0.85

        /* molar masses */
        const scalar M_C = 12.011e-3;           /* kg/mol */
        const scalar M_O = 15.999e-3;           /* kg/mol */
        const scalar M_H = 1.008e-3;            /* kg/mol */
        const scalar M_H2O = 18.01528e-3;       /* kg/mol */
        const scalar M_O2 = (2.0 * M_O);        /* kg/mol */
        const scalar M_H2 = (2.0 * M_H);        /* kg/mol */
        const scalar M_N2 = (2.0 * 14.007e-3);  /* kg/mol */
```

```
        const scalar M_CO2 = (M_C + M_O2);      /* kg/mol */
        const scalar M_CO = (M_C + M_O);        /* kg/mol */
        const scalar M_CH4 = (M_C + 4.0 * M_H); /* kg/mol */

        /* model indices */
        enum
        {
          layer_1,
          layer_2,
          layer_3,
          layer_4
        };
        enum
        {
          boundary_1,
          boundary_2,
          boundary_3,
          boundary_4
        };
        enum
        {
          drying_front, //drf
          devol_front, //dvf
          char_front, //cf
          char_produced //cp
        };
        enum
        {
          charring,
          tarring,
          gassing,
        };
        enum
        {
          evaporation, //ev
          devolatilization, //dv
          char_combustion //cc
        };
        enum
        {
          cylinder,
          sphere,
          parallelepiped
        };

        /* property indices */
        enum
        {
          index_CO2,
          index_H2,
          index_CO,
          index_H2O,
          index_CH4,
          index_C2H4,
          index_C10H8,
          index_C,
          index_C2H6,
          index_C3H8,
          index_C6H6O,
          index_volatiles
        };

        enum
        {
          H2O_index,
          O2_index,
          CH4_index,
          H2_index,
```

```
        CO_index,
        CO2_index,
        N2_index
    };

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

    /* static interpolation lists */
    scalar X_list[50];
    scalar Y_list[50];

    /* static message passing */
    scalar T_send[3];

    scalar view_factor = 1.;
    scalar m_init;
    scalar r_init;
    scalar L1, L2, L3;
    scalar particle_volume0;

    /* Redux statics */
    label material, shape;

    /* read some parameters from file */
    label VERBOSITY_, SAVE_EVERY_, FINE_PARTICLE_MESH_, N_SPECIES_;

    scalar TIME_DT_, PARTICLE_RADIUS_, T_REACTOR_,
    T_RADIATION_,T_P_INIT_, MASS_FRAC_O2_, DENSITY_INITIAL_, MOISTURE_FRACTION_DB_,
    MOISTURE_FRACTION_WB, FINE_PARTICLE_MESH_DOUBLE_,
    DRYING_SHRINKAGE_VOLUME_, ASH_MASS_FRACTION_DB, K_P_WET_, K_P_DRY_,
    K_P_CHAR_,ASH_MASS_FRACTION_, C_FRACTION_, H_FRACTION_, O_FRACTION_,
    CHARCONVERSION_SHRINKAGE_VOLUME_,DEVOL_SHRINKAGE_VOLUME_,
    P_ATM_, MASS_FRAC_H2O_, MASS_FRAC_H2_, MASS_FRAC_CO2_,
    MY_MAXIMUM_NEWTON_ITER_; /*DRY_DENSITY*/;

    /* drying kinetics parameters */
    scalar DryKinA_, DryKinE_;

    /* pyrolysis kinetics parameters */
    scalar DevKinA1_, DevKinE1_, DevKinA2_, DevKinE2_, DevKinA3_, DevKinE3_;

    // count interatrion number for print out results
    label iter_count, output_every;

    scalar tPar; //,t_stop;
    scalar aw, ae, ap, ap0, Su__b, Sp__b;
    scalar TOuterBoundary_;
    scalar Tf, Af;
    scalar rb, TInf_;
    scalar S, Q, q, a, heatHKp, e, delta_0, delta_1;
    scalar rho_g, mu, cp_g, k_g, vel_mag, h;
    scalar RCharReac_1S;

    scalar total_mass;

    scalar R_drySourceimins1High, R_drySourceiHigh, R_devoSourceimins1High,
    R_devoSourceiHigh, R_devoSourceimins1HighChar, R_devoSourceiHighChar,
    R_drySourceimins1Low, R_drySourceiLow, R_devoSourceimins1Low,
    R_devoSourceiLow, R_devoSourceimins1LowChar, R_devoSourceiLowChar,
    density_drywood;

    scalar D_g, D_gAB;

    scalar hmi = 0.0, hmia, beta_d, alpha = 0.0, conc_O2, conc_O2_correct,
    O2_diff_suface = 0.0, omegaC_S;

    scalar volume_sum;
```

```
        scalar mParSum_;

        scalar high = 0.0, low = 0.0;

        scalar R_O2_mole_need = 0.;

        scalar charFrontRadius_;

        scalar tReal_ = 0.0;

        label combustion_flag, combustion_layer;

        List<scalarField> xi_,QReaction_,Rb_,RCharReac_;
        List<scalarField> mPar_,mLast_;
        List<scalarField> QEquilibrate_,RDevolReac_;
        List<scalarField> dmdt;

        scalarField VCell_;
        scalarField VCharLast_,VChar_,xc_,Sp_,Su_,xif_,omegaC_;
        scalarField sStar_,dSdT_star;
        scalarField charFromDevol_,totalDevol_;
        scalarField RProduct_;
        scalarField charYield_;
        scalarField betaR_;
        scalarField R_O2_mole_need_list;
        scalarField ThermalTimeScale;

        /* static library variables */
        scalarField LHV,M;

        // meschach members
        VEC *b, *Tp_;
        MAT *A, *LU;
        PERM *pivot;


public:

    //- Class to hold reacting multiphase particle constant properties
    class constantProperties
    :
        public ParcelType::constantProperties
    {
        // Private Data

    public:

        // Constructors

            //- Null constructor
            constantProperties();

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

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


        // Access

    };


    //- Use base tracking data
    typedef typename ParcelType::trackingData trackingData;
```

```
private:

    // Private Member Functions
    // initialize
    void ReadLHVM(scalar YC); //YC is the char Fraction

    void initFields();

    void readMeshInput(const dictionary &particleInfoDict_);

    void readDevKin(const dictionary &particleInfoDict_);

    // call both readMeshInput and readDevKin
    void readCoeff(const Time &time);

    // Helper functions(Functions added from the mesh code)
    scalar c_p(label layer, scalar T);

    scalar k_p(label layer, scalar T);

    scalar rhoPar(label layer, scalar T);

    scalar gas_cp(label index, scalar T);

    scalar deltaHvap(scalar T);

    scalar deltaHdevol(label i, scalar T, scalar char_fraction);

    scalar volume_of_element(scalar r1, scalar r2);

    scalar surface_area(scalar r);

    scalar radius_from_volume(scalar V);

    scalar radius_from_volume_element(scalar V, scalar r_inner);

    scalar temperature_gradient(scalar Ti, scalar Tj, scalar ri, scalar rj);

    void update_surface_temperatures
    (
        scalar nextT_p3, scalar k_p3, scalar A, scalar r_b3, scalar r_c3,
        scalar Q_particle_conduction, scalar TInf_, scalar h,
        scalar emissivity
    );

    scalar heatRatio(scalar temperature);

    scalar charFrontLocation
    (
        scalar rb,
        label i,
        scalarField V_cell,
        scalarField VChar_,
        scalarField xc_,
        scalar xi_
    );

    void update_boundary_temperatures
    (
        scalar k_p1, scalar k_p2, scalar k_p3, scalar k_p4,
        scalar A[4], scalar rb[4], scalar rp[4],
        scalar T_p[4], scalar T_b[4],
        scalar Q_evap, scalar Q_devol, scalar Q_char_comb
    );

    //Do linear interpolation
    scalar linInterp(scalar X, label n_elements_in_list);
```

```
    // Eigen CG solver
    void EigenCGFunc(MAT* A, VEC* T, VEC* b);

protected:

    // Protected data

        // Parcel properties


    // Protected Member Functions



public:

    // Static Data Members

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

        //- String representation of properties
        AddToPropertyList
        (
            ParcelType,
            " TOuterBoundary"
          + " charFrontRadius"
          + " TInf"
          + " mParSum"
          + " VCell"
          + " Tp"
          + " VChar"
          + " VCharLast"
          + " xc"
          + " RProduct"
          + " omegaC"
          + " betaR"
          + " Su"
          + " Sp"
          + " totalDevol"
          + " charFromDevol"
          + " xi"
          + " mPar"
          + " mLast"
          + " RCharReac"
          + " QEquilibrate"
          + " Rb"
          + " QReaction"
          + " RDevolReac"
        );

    // Constructors

        //- Construct from mesh, position and topology
        //  Other properties initialised as null
        inline ReactingMultiphaseMBMParcel
        (
            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 ReactingMultiphaseMBMParcel
```

```
        (
            const polyMesh& mesh,
            const vector& position,
            const label celli
        );

        //- Construct from components
        inline ReactingMultiphaseMBMParcel
        (
            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 scalarField& Y0,
            const scalarField& YGas0,
            const scalarField& YLiquid0,
            const scalarField& YSolid0,
            const constantProperties& constProps
        );

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

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

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

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

        //- Construct and return a (basic particle) clone
        virtual autoPtr<particle> clone(const polyMesh& mesh) const
        {
            return autoPtr<particle>(new ReactingMultiphaseMBMParcel(*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<ReactingMultiphaseMBMParcel<ParcelType>> operator()
            (
                Istream& is
            ) const
            {
                return autoPtr<ReactingMultiphaseMBMParcel<ParcelType>>
                (
                    new ReactingMultiphaseMBMParcel<ParcelType>(mesh_, is, true)
                );
            }
        };


    // Member Functions

        // Access

            //- Return const access to mass fractions of gases
            inline scalar TOuterBoundary() const;
            inline scalar charFrontRadius() const;
            inline scalar TInf() const;
            inline scalar mParSum() const;

            inline scalarField VCell() const;
            inline scalarField Tp() const;
            inline scalarField VChar() const;
            inline scalarField VCharLast() const;
            inline scalarField xc() const;
            inline scalarField RProduct() const;
            inline scalarField omegaC() const;
            inline scalarField betaR() const;
            inline scalarField Su() const;
            inline scalarField Sp() const;
            inline scalarField totalDevol() const;
            inline scalarField charFromDevol() const;

            inline List<scalarField> xi() const;
            inline List<scalarField> mPar() const;
            inline List<scalarField> mLast() const;
            inline List<scalarField> RCharReac() const;
            inline List<scalarField> QEquilibrate() const;
            inline List<scalarField> Rb() const;
            inline List<scalarField> QReaction() const;
            inline List<scalarField> RDevolReac() const;

            // some initial constant value
            inline scalar MOISTURE_FRACTION_DB() const;



        // Edit

            //- Return access to front tempearture
            inline scalar& TOuterBoundary();
            inline scalar& charFrontRadius();
            inline scalar& TInf();
            inline scalar& mParSum();

            inline scalarField& VCell();
            inline scalarField& Tp();
            inline scalarField& VChar();
            inline scalarField& VCharLast();
            inline scalarField& xc();
            inline scalarField& RProduct();
```

```
            inline scalarField& omegaC();
            inline scalarField& betaR();
            inline scalarField& Su();
            inline scalarField& Sp();
            inline scalarField& totalDevol();
            inline scalarField& charFromDevol();

            inline List<scalarField>& xi();
            inline List<scalarField>& mPar();
            inline List<scalarField>& mLast();
            inline List<scalarField>& RCharReac();
            inline List<scalarField>& QEquilibrate();
            inline List<scalarField>& Rb();
            inline List<scalarField>& QReaction();
            inline List<scalarField>& RDevolReac();

            // some initial constant value
            inline scalar& MOISTURE_FRACTION_DB();


        // 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
            );

            //- 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, class CompositionType>
            static void readFields
            (
                CloudType& c,
                const CompositionType& compModel
            );

            //- Read - no composition
            template<class CloudType>
            static void readFields(CloudType& c);

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

            //- Read - composition supplied
```

```
            template<class CloudType>
            static void writeFields(const CloudType& c);


    // Ostream Operator

        friend Ostream& operator<< <ParcelType>
        (
            Ostream&,
            const ReactingMultiphaseMBMParcel<ParcelType>&
        );
};


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

} // End namespace Foam

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

#include "ReactingMultiphaseMBMParcelI.H"

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

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

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

#endif

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

A.1.2 externalFunctions.H

biomassParcel/Templates/externalFunctions.H

```
template <class ParcelType>
void Foam::ReactingMultiphaseMBMParcel<ParcelType>::ReadLHVM(scalar YC)
{
    /* Lower heating values (LHVs) */
    LHV[index_CO2] = 0.0;                                          /* per definition */
    LHV[index_H2] = 120.0e6;                                       /* J/kg D&D p. 26 */
    LHV[index_CO] = 10.25e6;                                       /* J/kg D&D p. 26 */
    LHV[index_H2O] = 0.0;                                          /* per definition */
    LHV[index_CH4] = 50.0e6;                                       /* J/kg D&D p. 26 */
    LHV[index_C2H4] = 47.7e6;                                      /* J/kg D&D p. 26 */
    LHV[index_C10H8] = 39.48e6;                                    /* J/kg Alberto's code */
    LHV[index_C] = 32.79e6;                                        /* J/kg Alberto's code */
    LHV[index_C2H6] = 47.6e6;                                      /* J/kg D&D p. 26 */
    LHV[index_C3H8] = 46.3e6;                                      /* J/kg D&D p. 26 */
    LHV[index_C6H6O] = 32.45e6;                                    /* J/kg http://www.nist.gov/data/
     PDFfiles/jpcrd6.pdf OR http://pac.iupac.org/publications/pac/pdf/1961/pdf/0201x0125.pdf */
    LHV[index_volatiles] = (18.93e6 - 32.57e6 * YC) / (1.0 - YC); /* J/kg Alberto's code */
    /* Molar masses */
    M[index_CO2] = 44.01e-3;      /* kg/mole */
    M[index_H2] = 2.0 * 1.008e-3; /* kg/mole */
    M[index_CO] = 28.010e-3;      /* kg/mole */
    M[index_H2O] = 18.01528e-3;   /* kg/mole */
    M[index_CH4] = 16.04e-3;      /* kg/mole */
    M[index_C2H4] = 28.05e-3;     /* kg/mole */
    M[index_C10H8] = 128.17e-3;   /* kg/mole */
    M[index_C] = 12.011e-3;       /* kg/mole */
    M[index_C2H6] = 30.07e-3;     /* kg/mole */
    M[index_C3H8] = 44.1e-3;      /* kg/mole */
```

```
    M[index_C6H6O] = 94.11e-3;    /* kg/mole */
}

template <class ParcelType>
void Foam::ReactingMultiphaseMBMParcel<ParcelType>::readMeshInput
(
   const dictionary &particleInfoDict_
)
{

  #define MESHINPUT particleInfoDict_.subDict("meshInput").lookup

  MESHINPUT("VERBOSITY") >> VERBOSITY_;
  MESHINPUT("SAVE_EVERY") >> SAVE_EVERY_;
  MESHINPUT("TIME_DT") >> TIME_DT_;
  MESHINPUT("TIME_TOTAL") >> TIME_TOTAL_;
  MESHINPUT("MOISTURE_FRACTION_DB") >> MOISTURE_FRACTION_DB_;
  MESHINPUT("PARTICLE_RADIUS") >> PARTICLE_RADIUS_;
  MESHINPUT("DRYING_SHRINKAGE_VOLUME") >> DRYING_SHRINKAGE_VOLUME_;
  MESHINPUT("T_REACTOR") >> T_REACTOR_;
  MESHINPUT("T_RADIATION") >> T_RADIATION_;
  MESHINPUT("T_P_INIT") >> T_P_INIT_;
  MESHINPUT("MASS_FRAC_O2") >> MASS_FRAC_O2_;
  MESHINPUT("DENSITY_INITIAL") >> DENSITY_INITIAL_;
  MESHINPUT("FINE_PARTICLE_MESH") >> FINE_PARTICLE_MESH_;
  MESHINPUT("FINE_PARTICLE_MESH_DOUBLE") >> FINE_PARTICLE_MESH_DOUBLE_;
  MESHINPUT("K_P_WET") >> K_P_WET_;
  MESHINPUT("K_P_DRY") >> K_P_DRY_;
  MESHINPUT("K_P_CHAR") >> K_P_CHAR_;
  MESHINPUT("ASH_MASS_FRACTION") >> ASH_MASS_FRACTION_;
  MESHINPUT("C_FRACTION") >> C_FRACTION_;
  MESHINPUT("H_FRACTION") >> H_FRACTION_;
  MESHINPUT("O_FRACTION") >> O_FRACTION_;
  MESHINPUT("CHARCONVERSION_SHRINKAGE_VOLUME") >> CHARCONVERSION_SHRINKAGE_VOLUME_;
  MESHINPUT("DEVOL_SHRINKAGE_VOLUME") >> DEVOL_SHRINKAGE_VOLUME_;
  MESHINPUT("P_ATM") >> P_ATM_;
  MESHINPUT("MASS_FRAC_H2O") >> MASS_FRAC_H2O_;
  MESHINPUT("MASS_FRAC_H2") >> MASS_FRAC_H2_;
  MESHINPUT("MASS_FRAC_CO2") >> MASS_FRAC_CO2_;
  MESHINPUT("MY_MAXIMUM_NEWTON_ITER") >> MY_MAXIMUM_NEWTON_ITER_;
  MESHINPUT("N_SPECIES") >> N_SPECIES_;
}

template <class ParcelType>
void Foam::ReactingMultiphaseMBMParcel<ParcelType>::readDevKin
(
   const dictionary &particleInfoDict_
)
{
  #define SIGDEVKIN particleInfoDict_.subDict("sigDevKin").lookup

  /* pyrolysis kinetics parameters */

  SIGDEVKIN("DevKinA1") >> DevKinA1_;
  SIGDEVKIN("DevKinE1") >> DevKinE1_;
  SIGDEVKIN("DevKinA2") >> DevKinA2_;
  SIGDEVKIN("DevKinE2") >> DevKinE2_;
  SIGDEVKIN("DevKinA3") >> DevKinA3_;
  SIGDEVKIN("DevKinE3") >> DevKinE3_;

  /* drying kinetics parameters */
  SIGDEVKIN("DryKinA") >> DryKinA_;
  SIGDEVKIN("DryKinE") >> DryKinE_;
}

template <class ParcelType>
void Foam::ReactingMultiphaseMBMParcel<ParcelType>::readCoeff(const Time &time)
{
```

```
    // dictionary particleInfoDict_("constant/particleInfoDict");
    // Info << "particleInfoDict_ is " << particleInfoDict_ << nl;
    const dictionary& particleInfoDict_
    (
        IFstream("constant/particleInfoDict")()
    );

    Info << particleInfoDict_ << nl;
    readMeshInput(particleInfoDict_);
    readDevKin(particleInfoDict_);

    MOISTURE_FRACTION_WB = MOISTURE_FRACTION_DB_/(1.0+MOISTURE_FRACTION_DB_);
    ASH_MASS_FRACTION_DB = ASH_MASS_FRACTION_/(1.0-MOISTURE_FRACTION_WB);

}

template <class ParcelType>
void Foam::ReactingMultiphaseMBMParcel<ParcelType>::initFields()
{
    fp = fopen("codeMeshOutput.txt", "w");
    // initialize some properties of the particle
    material = 1;
    shape = sphere;
    r_init = PARTICLE_RADIUS_;
    Info << "PARTICLE_RADIUS_ " << PARTICLE_RADIUS_ << nl;
    L1 = 2.0*r_init;
    L2 = 2.0*r_init;
    L3 = 2.0*r_init;

    // initial mass of the biomass particle
    m_init = rhoPar(layer_1, T_P_INIT_)*volume_of_element(0.0, r_init);

    // initial volume of the biomass particle
    particle_volume0 = volume_of_element(0.0, r_init);

    LHV.resize(N_SPECIES_, 0.0);
    M.resize(N_SPECIES_-1, 0.0);
    ReadLHVM(0.18); /* assumes a char fraction of 0.18 (only relevant for volatile LHV) */

    #define LISTINIT2D(NAME,SIZE1,SIZE2) NAME.resize(SIZE1); \
    forAll(NAME,i)                                           \
    {                                                        \
        NAME[i].resize(SIZE2, 0.0);                              \
    }

    LISTINIT2D(xi_,4,FINE_PARTICLE_MESH_);
    LISTINIT2D(QReaction_,3,FINE_PARTICLE_MESH_);
    LISTINIT2D(Rb_,4,FINE_PARTICLE_MESH_);
    LISTINIT2D(RCharReac_,4,FINE_PARTICLE_MESH_);
    LISTINIT2D(mPar_,4,FINE_PARTICLE_MESH_);
    LISTINIT2D(mLast_,4,FINE_PARTICLE_MESH_);
    LISTINIT2D(QEquilibrate_,3,FINE_PARTICLE_MESH_);
    LISTINIT2D(RDevolReac_,3,FINE_PARTICLE_MESH_);
    LISTINIT2D(dmdt,4,FINE_PARTICLE_MESH_); //pass

    VCell_.resize(FINE_PARTICLE_MESH_, 0.0);
    VCharLast_.resize(FINE_PARTICLE_MESH_, 0.0);
    VChar_.resize(FINE_PARTICLE_MESH_, 0.0);
    xc_.resize(FINE_PARTICLE_MESH_, 0.0);
    Sp_.resize(FINE_PARTICLE_MESH_, 0.0);
    Su_.resize(FINE_PARTICLE_MESH_);
    xif_.resize(FINE_PARTICLE_MESH_); //
    omegaC_.resize(FINE_PARTICLE_MESH_, 0.0);
    sStar_.resize(FINE_PARTICLE_MESH_, 0.0); //
    dSdT_star.resize(FINE_PARTICLE_MESH_, 0.0); //
    charFromDevol_.resize(FINE_PARTICLE_MESH_, 0.0); //
    totalDevol_.resize(FINE_PARTICLE_MESH_, 0.0); //
    RProduct_.resize(7, 0.0); // TO DO as different model
```

```
    charYield_.resize(FINE_PARTICLE_MESH_, 0.0); //
    betaR_.resize(4, 0.0);
    R_O2_mole_need_list.resize(FINE_PARTICLE_MESH_, 0.0); //
    ThermalTimeScale.resize(FINE_PARTICLE_MESH_, 0.0); //

    /* Assemble matrix system */
    A = m_get(FINE_PARTICLE_MESH_, FINE_PARTICLE_MESH_);
    Tp_ = v_get(FINE_PARTICLE_MESH_);
    b = v_get(FINE_PARTICLE_MESH_);

    /*  ap*Tp_ = aw*Tw+ae*Te+ap0*Tp0+Su_
    *       ap = ap0+aw+aE-Sp_
    *       ap0 = rho*cp*deltax/deltat
    *       aw = kw/deltax_wp
    *       ae = ke/deltax_pe
    *       b = S*deltax = Su_+Sp_*Tp_      */
    /* --- formulated as a matrix problem:
    *       aw*Tw-ap*Tp_+ae*Te =-ap0*Tp0-Su_    */

    /* Set some parameters */
    output_every = SAVE_EVERY_; //((label) (1.0/dt));
    // t_stop = TIME_TOTAL_;

    /* Initialize variables */
    // particle time
    // tPar = 0.0;
    // set the particle temperature to be zero
    v_zero(Tp_); // take old values or not? No!!

    // initialize the iteration number to be 0
    // this was used for print out result before
    iter_count = 0;

    // read the particle radius
    // this value needs to be updated!!
    rb = PARTICLE_RADIUS_; //

    /*Print out KP*/
    // Info << "K_P_WET_: " << K_P_WET_ << nl;
    // Info << "K_P_DRY_: " << K_P_DRY_ << nl;
    // Info << "K_P_CHAR_: " << K_P_CHAR_ << nl;

    /* Initialize fields */
    if (VERBOSITY_)
    {
    printf("Init Tp_:\n");
    }

    for (unsigned int i = 0; i < Tp_->max_dim; i++)
    {
        // set the initial temperature to the Tp_ vector
        // in IO.C file, this value should not be set again, but read
        Tp_->ve[i] = T_P_INIT_;
        if (VERBOSITY_)
        {
            // printf("%i: %f ", i, Tp_->ve[i]);
            Info << i << ": " << Tp_->ve[i] << " ";
        }
    }

    // Set the initial boundary temperature
    TOuterBoundary_ = T_P_INIT_;
    if (VERBOSITY_)
    {
        printf("\n");
    }

    /* Particle is meshed from the outside in */
```

```
    if (VERBOSITY_)
    {
        printf("Particle mesh:\n");
    }

    // calculate the cell Volume
    scalar cellVol = volume_of_element(0.0, rb)/FINE_PARTICLE_MESH_DOUBLE_;

    // volume_sum at the beginning is equal to one cell volume
    volume_sum = 0.5*cellVol;

    // Set the cell center location from the outmost cell using volume
    // cell number 1 is the one close to the particle surface
    for (j = FINE_PARTICLE_MESH_; j >= 1; j--)
    {
        // xc_ means the cell center location
        Info << "volume_sum " << volume_sum << nl;
        xc_[j-1] = radius_from_volume(volume_sum);
        volume_sum = volume_sum+cellVol;
        if (VERBOSITY_)
        {
            printf("%i: %e-", j, xc_[j-1]);
        }
        charFromDevol_[j-1] = 0.0;
        totalDevol_[j-1] = 0.0;
    }

    if (VERBOSITY_)
    {
        printf("\n");
    }

    // initialize the total mass to be zero at the beginning
    // in IO.C file, this value should not be set again, but read
    total_mass = 0.0;
    // four layers
    for (i = 0; i <= 3; i++)
    {
        // each cell in layer i
        for (j = 0; j < FINE_PARTICLE_MESH_; j++)
        {
            // if it is layer 1
            if (i == layer_1)
            {
                // initial composition of wet wood is set as 1 for all the cells
                xi_[i][j] = 1.0;
            }
            else
            {
                // other layer (dry wood, char, and ash) does not exit yet
                xi_[i][j] = 0.0;
            }

            // mass of each layer is initialize
            // mass = xi_ (fraction) * rho * volume
            mPar_[i][j] =
                xi_[i][j]*rhoPar(i, Tp_->ve[i])*cellVol;

            printf("x %e ", xi_[i][j]);
            printf("mPar_ %e ", mPar_[i][j]);
            // initialize the total mass before any reactions
            total_mass = total_mass+mPar_[i][j];
        }
    }
    printf("\n");

    /* Set gas phase data at the reactor temperature */
    TInf_ = T_REACTOR_;
```

```
    // gas density
    rho_g = (3.550951e2*Foam::pow(TInf_, -1.000866));

    // viscosity?
    mu =
        1.716e-5*Foam::pow((TInf_/273.11), 3./2.)
        *(273.11+110.56)/(TInf_+110.56);

    // gas heat capacity
    cp_g = 1.0e3*
    (
        (-0.000000000153767)*Foam::pow(TInf_, 3.0)
        +0.000000363209494*Foam::pow(TInf_, 2.0)
        +(-0.000051486068111)*TInf_+1.009174922600606
    ); /* polyfit from table data for N2 on the interval 500-1300 K */

    k_g = (-0.000000000003493)*Foam::pow(TInf_, 3.0)
            +0.000000003319264*Foam::pow(TInf_, 2.0)
            +0.000060059499759*TInf_+0.008533051948052;
    /* polyfit from table data for N2 on the interval 500-1300 K */

    if (VERBOSITY_)
    {
        printf("Gas phase data to use at Tp_ = %f\n", TInf_);
        printf("  rho  = %e\n", rho_g);
        printf("  mu   = %e\n", mu);
        printf("  cp_g = %e\n", cp_g);
        printf("  k_g  = %e\n", k_g);
    }

    // velocity?
    vel_mag = 0.5; /* (mPar_/s) */

    /* molar concentration of O2 in the gas phase */
    // conc_O2 = MASS_FRAC_O2_*(M_O2/M_N2)*P_ATM_/(8.3145*TInf_);
    conc_O2 = MASS_FRAC_O2_*(M_N2/M_O2)*P_ATM_/(8.3145*TInf_);
    // conc_CO2 = MASS_FRAC_CO2_*(M_CO2/M_N2)*P_ATM_/(8.3145*TInf_); /* molar concentration of CO2 in
     the gas phase */
    // conc_H2O = MASS_FRAC_H2O_*(M_H2O/M_N2)*P_ATM_/(8.3145*TInf_); /* molar concentration of H2O in
     the gas phase */
    // conc_H2 = MASS_FRAC_H2_*(M_H2/M_N2)*P_ATM_/(8.3145*TInf_); /* molar concentration of H2 in the
     gas phase */

    /*NOT USED IN THIS CODE!!!!*/
    /* Devolatilization product distribution */
    /* NEW TAKE 2016-10-10-use Neves data at 900C */
    // Y_H2 = 0.015; /* (kg H2/kg daf fuel) */
    // Y_CO = 0.3; /* (kg CO/kg daf fuel) */
    // Y_CH4 = 0.05;   /* (kg CH4/kg daf fuel) */
    // Y_CO2 = 0.1; /* (kg CO2/kg daf fuel) */
    // Y_G = 0.55;
    // Y_H2O = Y_G-(Y_H2+Y_CO+Y_CH4+Y_CO2); /* (kg CO2/kg daf fuel) */

    //  initialize volume of each components
    for (j = FINE_PARTICLE_MESH_; j >= 1; j--)
    {
        VCell_[j-1] = cellVol;
        VChar_[j-1] = 0.0;
        VCharLast_[j-1] = 0.0;

        for (i = 1; i <= 4; i++)
        {
            // mLast_ is the mass from the last time step
            mLast_[i-1][j-1] = mPar_[i-1][j-1];
        }
        // The thermal time scale should not be initialized again in IO
        if (j == 1)
```

```
        {
            ThermalTimeScale[j-1] =
            Foam::pow(rb-xc_[0], 2.0)
            /(k_p(layer_1, Tp_->ve[j-1])
            /(c_p(layer_1, Tp_->ve[j-1])
            *rhoPar(layer_1, Tp_->ve[j-1])));
        }
        else
        {
            ThermalTimeScale[j-1] =
            Foam::pow(xc_[j-2]-xc_[j-1], 2.0)
            /(k_p(layer_1, Tp_->ve[j-1])
            /(c_p(layer_1, Tp_->ve[j-1])
            *rhoPar(layer_1, Tp_->ve[j-1])));
        }
    }

    // for (int i = 0; i < FINE_PARTICLE_MESH_; i++) {
    //   Info << xc_[i] << nl;
    // }

    density_drywood =
        DENSITY_INITIAL_
        *(1.0-MOISTURE_FRACTION_WB)
        /(1.0-DRYING_SHRINKAGE_VOLUME_);

    combustion_layer = 99999999;

    // write down time and temperature to file "fp" with a name codeMeshInput.txt
    fprintf(fp, "time %e ", tPar);
    fprintf(fp, "temperatures ");

    // print the particle initial temperature to the screen
    for (i = 1; i <= FINE_PARTICLE_MESH_; i++)
    {
        fprintf(fp, "%e ", Tp_->ve[i-1]);
    }

    // print the particle outer boundary temperature
    fprintf(fp, "%e ", TOuterBoundary_);

    // write down masses and radius to file "fp"
    for (j = 1; j <= 4; j++)
    {
        fprintf(fp, "masses ");

        for (i = 1; i <= FINE_PARTICLE_MESH_; i++)
        {
            for (j = 1; j <= 4; j++)
            {
                fprintf(fp, "%e ", mPar_[j-1][i-1]);
            }
        }
    fprintf(fp, "radius %e ", rb);
    }

    fprintf(fp, "Rb_ ");

    for (i = 1; i <= FINE_PARTICLE_MESH_; i++)
    {
        for (j = 1; j <= 4; j++)
        {
            fprintf(fp, "%e ", Rb_[j-1][i-1]);
        }
    }

    fprintf(fp, "Thermal_time ");
```

```
    for (i = 1; i <= FINE_PARTICLE_MESH_; i++)
    {
        fprintf(fp, "%e ", ThermalTimeScale[i-1]);
    }

    fprintf(fp, "\n");

    Info << "A->max_m " << A->max_m << nl;
    Info << "A->max_n " << A->max_n << nl;
    Info << "A->max_size " << A->max_size << nl;

    Info << "xi_" << nl << xi_ << nl;

    // FatalErrorInFunction
    //           << "Unknown scatterModel type "
    //           << exit(FatalError);
};


template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::linInterp
(
    scalar X, label n_elements_in_list
)
{
    label i = 0;
    scalar my_value;

    if (X <= X_list[0])
    {
        my_value = Y_list[0];
    }
    else if (X >= X_list[n_elements_in_list - 1])
    {
        my_value = Y_list[n_elements_in_list - 1];
    }
    else
    {
        while (!((X >= X_list[i - 1]) && (X < X_list[i])))
        {
            i = i + 1;
        }
        my_value = Y_list[i - 1]
                   +
                   (Y_list[i] - Y_list[i - 1]) * (X - X_list[i - 1])
                   / (X_list[i] - X_list[i - 1]);
    }

    return my_value;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::c_p
(
  label layer, scalar T
)
{
    /* returns c_p in J/kg,K for the particle layer */
    /* N.B.: currently, all materials use the same expressions! */
    scalar heat_capacity = 0.0;
    scalar c_p_wood;
    scalar c_p_water = 4.309e3; /* c_p for water in J/kg,K */
    scalar A;                   /* correction factor in J/kg,K */
    scalar Y_mois = 1.0-1.0/(1.0+MOISTURE_FRACTION_DB_);
    //     scalar hcgas;

    if (layer == layer_1)
    {
```

```
        /* moist wood */
        c_p_wood = 4.206*T-37.7; /* eq. (24.i) Thunman et al., Energy & Fuels 2001 */
        A = 1.0e3*
            ((0.02355*T-1.320*Y_mois/(1.0-Y_mois)-6.191)*Y_mois/(1.0-Y_mois));
        heat_capacity = c_p_wood*(1.0-Y_mois)+c_p_water*Y_mois+A;
    }
    else if (layer == layer_2)
    {
        /* dry wood */
        heat_capacity = 4.206*T-37.7; /* eq. (24.i) Thunman et al., Energy & Fuels 2001 */
        // heat_capacity = 1000.*(0.1031+0.003867*T); /* Properties of Wood for Combustion Analysis */
    }
    else if (layer == layer_3)
    {
        /* char */
        heat_capacity = -334.0+4410.0e-3*T-3160.0e-6*Foam::pow(T, 2.0)
            +1010.0e-9*Foam::pow(T, 3.0)-119.0e-12*Foam::pow(T, 4.0); /* Table 5 Thunman et al.,
     Energy & Fuels 2001 */
        //heat_capacity = 1000.*(1.39+0.00036*T); /* Properties of Wood for Combustion Analysis */
    }
    else if (layer == layer_4)
    {
        heat_capacity = 754.0+0.586*(T-273.15); /* Sven */
    }
    else
    {
        printf("ERROR in c_p function!!!\n");
    }

    return heat_capacity;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::k_p
(
  label layer, scalar T
)
{
    /* returns heat conductivity (W/m,K) for each layer as a function of temperature */
    scalar heat_conductivity = 0.0, kgas = 0.0;

    if (layer == layer_1)
    {
        heat_conductivity = K_P_WET_; //
    }
    else if (layer == layer_2)
    {
        heat_conductivity = K_P_DRY_; //
    }
    else if (layer == layer_3)
    {
        //         heat_conductivity = 0.052; //K. W. Ragland, D. J. Aerts, Properties of Wood for
     Combustion Analysis Wheast cites "R. C. (1976). Handbook of Chemistry and Physics, CRC Press,
     Cleveland, Ohio, p. E-5."
        heat_conductivity = K_P_CHAR_;
    }
    else if (layer == layer_4)
    {
        kgas = (-0.000000000003493)*Foam::pow(T, 3.0)
            +0.000000003319264*Foam::pow(T, 2.0)
            +0.000060059499759*T+0.008533051948052;

        heat_conductivity = 1.03*(1.0-0.65)+0.65*kgas; /* Sven */
    }
    else
    {
        printf("ERROR in heat conductivity function!!\n");
    }
```

```
    return heat_conductivity;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::rhoPar
(
  label layer, scalar T
)
{
    /* returns density (kg/m3) for each layer as a function of temperature */
    scalar density = 0.0;
    scalar dgas;

    if (material == 1) /* in this version of the code there is only one material implemented in this
     function */
    {
        if (layer == layer_1)
        {
            density = DENSITY_INITIAL_;
        }
        else if (layer == layer_2)
        {
            density = (1.0-MOISTURE_FRACTION_WB)*DENSITY_INITIAL_;
        }
        else if (layer == layer_3)
        {
            density = 350.0;
        }
        else if (layer == layer_4)
        {
            dgas = 101325.0*(2.0*14.00672*1.0e-3)/(8.3145*T);
            density = 2000.0*(1.0-0.65)+0.65*dgas; /* Sven */
        }
        else
        {
            printf("ERROR in density function!!\n");
        }
    }
    else
    {
        printf("ERROR in density function!!\n");
    }

    return density;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::gas_cp
(
  label index, scalar T
)
{
    /* returns heat capacity of the gas in J/mol,K */
    label i;
    scalar A, B, C, D, E;
    scalar temp = 0.0;
    scalar cp = 0.0;

    scalar T_list_C10H8[19] =
    {50.0, 100.0, 150.0, 200.0, 273.15, 298.15, 300.0, 400.0, 500.0, 600.0,
    700.0, 800.0, 900.0, 1000.0, 1100.0, 1200.0, 1300.0, 1400.0, 1500.0};

    scalar Cp_list_C10H8[19] =
    {36.18, 47.50, 63.89, 84.99, 120.52, 133.02, 133.94, 181.16, 220.70,
    252.37, 277.77, 298.43, 315.50, 329.77, 341.8, 352.0, 360.8, 368.2, 374.7};

    scalar T_list_C2H6[31] =
```

```
    {100.0, 200.0, 298.15, 300.0, 400.0, 500.0, 600.0, 700.0, 800.0, 900.0,
    1000.0, 1100.0, 1200.0, 1300.0, 1400.0, 1500.0, 1600.0, 1700.0, 1800.0,
    1900.0, 2000.0, 2100.0, 2200.0, 2300.0, 2400.0, 2500.0, 2600.0, 2700.0,
    2800.0, 2900.0, 3000.0};

    scalar Cp_list_C2H6[31] =
    {35.70, 42.30, 52.49, 52.71, 65.46, 77.94, 89.19, 99.14, 107.94, 115.71,
    122.55, 128.55, 133.80, 138.39, 142.40, 145.90, 148.98, 151.67, 154.04,
    156.14, 158.00, 159.65, 161.12, 162.43, 163.61, 164.67, 165.63, 166.49,
    167.28, 168.00, 168.65};

    scalar T_list_C3H8[19] =
    {50.0, 100.0, 150.0, 200.0, 273.15, 298.15, 300.0, 400.0, 500.0, 600.0,
    700.0, 800.0, 900.0, 1000.0, 1100.0, 1200.0, 1300.0, 1400.0, 1500.0};

    scalar Cp_list_C3H8[19] =
    {34.06, 41.30, 48.79, 56.07, 68.74, 73.60, 73.93, 94.01, 112.59, 128.70,
    142.67, 154.77, 165.35, 174.60, 182.67, 189.74, 195.85, 201.21, 205.89};

    scalar T_list_C6H6O[19] =
    {50.0, 100.0, 150.0, 200.0, 273.15, 298.15, 300.0, 400.0, 500.0, 600.0,
    700.0, 800.0, 900.0, 1000.0, 1100.0, 1200.0, 1300.0, 1400.0, 1500.0};

    scalar Cp_list_C6H6O[19] =
    {33.91, 41.38, 54.19, 69.65, 94.61, 103.22, 103.86, 135.79, 161.91, 182.48,
    198.84, 212.14, 223.19, 232.49, 240.41, 247.20, 253.06, 258.12, 262.52};
    //     scalar BensonBuss_H = 0.85;
    //     scalar BensonBuss_C = 3.75;
    //     scalar BensonBuss_O = 3.40;

    if (index == index_CO2)
    {
        if (T < 1200.0)
        {
            /* valid to 1200K */
            A = 24.99735;
            B = 55.18696;
            C = -33.69137;
            D = 7.948387;
            E = -0.136638;
        }
        else
        {
            /* valid from 1200 to 6000 K */
            A = 58.16639;
            B = 2.720074;
            C = -0.492289;
            D = 0.038844;
            E = -6.447293;
        }
        /* Shomate Equation from NIST Webbook */
        temp = T/1000.0;
        cp = A+B*temp+C*Foam::pow(temp, 2.0)+D*Foam::pow(temp, 3.0)+
            E/Foam::pow(temp, 2.0);
    }
    else if (index == index_H2)
    {
        if (T < 1000.0)
        {
            /* valid from 298 to 1000K */
            A = 33.066178;
            B = -11.363417;
            C = 11.432816;
            D = -2.772874;
            E = -0.158558;
        }
        else
        {
```

```
            /* valid from 1000 to 2500K */
            A = 18.563083;
            B = 12.257357;
            C = -2.859786;
            D = 0.268238;
            E = 1.977990;
        }
        /* Shomate Equation from NIST Webbook */
        temp = T/1000.0;
        cp = A+B*temp+C*Foam::pow(temp, 2.0)+D*Foam::pow(temp, 3.0)+
            E/Foam::pow(temp, 2.0);
    }
    else if (index == index_CO)
    {
        /* valid to 1300K */
        A = 25.56759;
        B = 6.096130;
        C = 4.054656;
        D = -2.671301;
        E = 0.131021;
        /* Shomate Equation from NIST Webbook */
        temp = T/1000.0;
        cp = A+B*temp+C*Foam::pow(temp, 2.0)+D*Foam::pow(temp, 3.0)+
            E/Foam::pow(temp, 2.0);
    }
    else if (index == index_H2O)
    {
        /* valid from 500 to 1700 K */
        A = 30.09200;
        B = 6.832514;
        C = 6.793435;
        D = -2.534480;
        E = 0.082139;
        /* Shomate Equation from NIST Webbook */
        temp = T/1000.0;
        cp = A+B*temp+C*Foam::pow(temp, 2.0)+D*Foam::pow(temp, 3.0)+
            E/Foam::pow(temp, 2.0);
    }
    else if (index == index_CH4)
    {
        /* valid to 1300K */
        A = -0.703029;
        B = 108.4773;
        C = -42.52157;
        D = 5.862788;
        E = 0.678565;
        /* Shomate Equation from NIST Webbook */
        temp = T/1000.0;
        cp = A+B*temp+C*Foam::pow(temp, 2.0)+D*Foam::pow(temp, 3.0)+
            E/Foam::pow(temp, 2.0);
    }
    else if (index == index_C2H4)
    {
        if (T < 1200.0)
        {
            /* valid from 298 to 1200K */
            A = -6.387880;
            B = 184.4019;
            C = -112.9718;
            D = 28.49593;
            E = 0.315540;
        }
        else
        {
            /* valid from 1200 K to 6000 K */
            A = 106.5104;
            B = 13.73260;
            C = -2.628481;
```

```
            D = 0.174595;
            E = -26.14469;
        }
        /* Shomate Equation from NIST Webbook */
        temp = T/1000.0;
        cp = A+B*temp+C*Foam::pow(temp, 2.0)+D*Foam::pow(temp, 3.0)+
            E/Foam::pow(temp, 2.0);
    }
    else if (index == index_C10H8)
    {
        for (i = 1; i <= 19; i++)
        {
            X_list[i-1] = T_list_C10H8[i-1];
            Y_list[i-1] = Cp_list_C10H8[i-1];
        }
        cp = linInterp(T, 19);
    }
    else if (index == index_C)
    {
        /* specific heat of graphite (solid)-valid from 250 to 3000 K (taken from NIST Webbook)
        *       %     cp = 0.538657+9.11129e-6*T-90.2725*T^(-1)-43449.3*T^(-2)+1.59309e7*T^(-3)
     -1.43688e9*T^(-4); % cal/g*K
        *       %     cp = cp*12.0107;    % (*g/mol) --> cal/mol,K
        *       %     cp = cp*4.184;      % (*J/cal) --> J/mol,K */
        cp =
            (
                0.538657+9.11129e-6*T-90.2725*Foam::pow(T, -1.0)
                -43449.3*Foam::pow(T, -2.0)+1.59309e7*Foam::pow(T, -3.0)
                -1.43688e9*Foam::pow(T, -4.0)
            )*12.0107*4.184; /* J/mol*K */
    }
    else if (index == index_C2H6)
    {
        for (i = 1; i <= 31; i++)
        {
            X_list[i-1] = T_list_C2H6[i-1];
            Y_list[i-1] = Cp_list_C2H6[i-1];
        }
        cp = linInterp(T, 31);
    }
    else if (index == index_C3H8)
    {
        for (i = 1; i <= 19; i++)
        {
            X_list[i-1] = T_list_C3H8[i-1];
            Y_list[i-1] = Cp_list_C3H8[i-1];
        }
        cp = linInterp(T, 19);
    }
    else if (index == index_C6H6O)
    {
        for (i = 1; i <= 19; i++)
        {
            X_list[i-1] = T_list_C6H6O[i-1];
            Y_list[i-1] = Cp_list_C6H6O[i-1];
        }
        cp = linInterp(T, 19);
    }
    else
    {
        printf("ERROR in gas_cp function!!! (1)\n");
    }

    return cp;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::deltaHvap(scalar T)
```

```
{
    /* returns heat of vaporization of water in J/kg for T in Kelvin */
    /* on the interval 273-433K */
    label i;
    scalar T_list[9] =
        {273.15, 298.15, 313.15, 333.15, 353.15,
        373.15, 393.15, 413.15, 433.15};

    scalar dH_list[9] =
        {45.054e3, 43.99e3, 43.35e3, 42.482e3,
        41.585e3, 40.657e3, 39.684e3, 38.643e3, 37.518e3};

    scalar dH;

    for (i = 1; i <= 9; i++)
    {
        X_list[i-1] = T_list[i-1];
        Y_list[i-1] = dH_list[i-1];
    }

    dH = linInterp(T, 9);
    dH = dH/18.0e-3; /* convert J/mol to J/kg */

    return dH;
    //     return 2264000.705;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::deltaHdevol
(
  label i, scalar T, scalar char_fraction
)
{
    /* returns heat of devolatilization in J/mole for T in Kelvin */
    scalar dH = 0.0;

    return dH;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::volume_of_element
(
  scalar r1, scalar r2
)
{
    /* returns volume in m3 of element */
    /* 1 = cylinder, 2 = sphere, 3 = parallelepiped */
    scalar vol = 0.0;
    scalar eta = 0.0;

    if (shape == cylinder)
    {
        eta = L1-L3;
        if (L3 == L1)
        {
            vol = 2*M_PI*(Foam::pow(r2, 3.0)-Foam::pow(r1, 3.0));
        }
        else if (L3 > L1)
        {
            vol = 2*M_PI*(Foam::pow(r2, 3.0)-Foam::pow(r2, 2.0)*eta/2)
                 -2*M_PI*(Foam::pow(r1, 3.0)-Foam::pow(r1, 2.0)*eta/2);
        }
        else if (L3 < L1)
        {
            vol =
                2*M_PI*
                    (
                        Foam::pow(r2, 3.0)+Foam::pow(r2, 2.0)*eta
```

```
                        +0.25*Foam::pow(eta, 2.0)
                    )
                -2*M_PI*
                    (
                        Foam::pow(r1, 3.0)+Foam::pow(r1, 2.0)*eta
                        +0.25*Foam::pow(eta, 2.0)
                    );
        }
    }
    else if (shape == sphere)
    {
        vol =
            4.0*M_PI*(Foam::pow(r2, 3.0))/3.0-4.0*M_PI*(Foam::pow(r1, 3.0))/3.0;
    }
    else if (shape == parallelepiped)
    {
        vol = 8*Foam::pow(r2, 3.0)
              +4*Foam::pow(r2, 2.0)*((L2-L1)+(L3-L1)+2*(L2-L1)*(L3-L1))
              -
              (
                8*Foam::pow(r1, 3.0)
                +4*Foam::pow(r1, 2.0)*((L2-L1)+(L3-L1)+2*(L2-L1)*(L3-L1))
              );
    }
    else
    {
        printf("ERROR in volume of element function!!!\n");
    }

    return vol;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::surface_area
(
  scalar r
)
{
    /* returns surface area in m2 for the surface */
    /* 1 = cylinder, 2 = sphere, 3 = parallellepiped */
    scalar surf = 0.0;
    scalar eta = 0.0;

    if (shape == cylinder)
    {
        eta = L1-L3;
        if (L3 == L1)
        {
            surf = 6.0*M_PI*Foam::pow(r, 2.0);
        }
        else if (L3 > L1)
        {
            surf = 2.0*M_PI*(3.0*Foam::pow(r, 2.0)-r*eta);
        }
        else
        {
            surf = 2.0*M_PI*
            (
                3.0*Foam::pow(r, 2.0)+2.0*r*eta+0.25*Foam::pow(eta, 2.0)
            );
        }
    }
    else if (shape == sphere)
    {
        surf = 4.0*M_PI*(Foam::pow(r, 2.0));
    }
    else if (shape == parallelepiped)
    {
```

```
        surf = 24.0*Foam::pow(r, 2.0)
            +8.0*r*((L2-L1)+(L3-L1)+2.0*(L2-L1)*(L3-L1));
    }
    else
    {
        printf("ERROR in surface area function!!!\n");
    }

    return surf;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::radius_from_volume
(
  scalar V
)
{
    scalar r0, r1;
    scalar f = 0.0, f_prime = 0.0;
    scalar eta = L1-L3;
    label i = 0;

    r0 = 1.0*L1; /* initial guess */
    r1 = 0.5*L1;

    while
    (
        (Foam::mag((r0-r1)/r0) > MY_QUITE_SMALL)&&(i <= MY_MAXIMUM_NEWTON_ITER_)
    )
    {
        i = i+1;
        r0 = r1;

        if (shape == cylinder)
        {
            if (eta == 0)
            {
                f = (2.0*M_PI*Foam::pow(r0, 3.0)-V);
                f_prime = 6.0*M_PI*Foam::pow(r0, 2.0);
            }
            else if (L3 > L1)
            {
                f =
                    (
                        2.0*M_PI*
                            (Foam::pow(r0, 3.0)-Foam::pow(r0, 2.0)*eta/2.0)-V
                    );
                f_prime = 2.0*M_PI*(3.0*Foam::pow(r0, 2.0)-r0*eta);
            }
            else
            {
                f =
                    (
                        2.0*M_PI*
                            (
                                Foam::pow(r0, 3.0)+Foam::pow(r0, 2.0)*eta
                                +0.25*Foam::pow(eta, 2.0)
                            )
                        -V
                    );
                f_prime = 2.0*M_PI*(3.0*Foam::pow(r0, 2.0)+2.0*r0*eta);
            }
        }
        else if (shape == sphere)
        {
            f = (4.0*M_PI*Foam::pow(r0, 3.0)/3.0-V);
            f_prime = 4.0*M_PI*Foam::pow(r0, 2.0);
        }
```

```
        else if (shape == parallelepiped)
        {
            f =
                (
                    8.0*Foam::pow(r0, 3.0)
                    +4.0*Foam::pow(r0, 2.0)*((L2-L1)+(L3-L1)+2*(L2-L1)*(L3-L1))
                    -V
                );
            f_prime = 8.0*3.0*Foam::pow(r0, 2.0)
                +4.0*2.0*r0*((L2-L1)+(L3-L1)+2*(L2-L1)*(L3-L1));
        }
        else
        {
            printf("ERROR in radius from volume function!!\n");
        }

        r1 = r0-f/f_prime;
    }


    return r1;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::
radius_from_volume_element(scalar V, scalar r_inner)
{
    scalar r0, r1;
    scalar f = 0.0, f_prime = 0.0;
    scalar eta = L1-L3;
    label i = 0;

    r0 = 1.0*L1; /* initial guess */
    r1 = 0.5*L1;

    while (
        (Foam::mag((r0-r1)/r0) > MY_QUITE_SMALL)&&(i <= MY_MAXIMUM_NEWTON_ITER_))
    {
        i = i+1;
        r0 = r1;

        if (shape == cylinder)
        {
            if (eta == 0)
            {
                f = (2.0*M_PI*Foam::pow(r0, 3.0)-V);
                f_prime = 6.0*M_PI*Foam::pow(r0, 2.0);
            }
            else if (L3 > L1)
            {
                f =
                    (
                        2.0*M_PI*(Foam::pow(r0, 3.0)-Foam::pow(r0, 2.0)*eta/2.0)
                        -V
                    );
                f_prime = 2.0*M_PI*(3.0*Foam::pow(r0, 2.0)-r0*eta);
            }
            else
            {
                f =
                    (
                        2.0*M_PI*
                            (
                                Foam::pow(r0, 3.0)+Foam::pow(r0, 2.0)*eta
                                +0.25*Foam::pow(eta, 2.0)
                            )
                            -V
                    );
```

```
                f_prime = 2.0*M_PI*(3.0*Foam::pow(r0, 2.0)+2.0*r0*eta);
            }
            printf
            (
                "ERROR in radius from volume element function -- NOT PREPARED\n"
            );
        }
        else if (shape == sphere)
        {
            f =
                (
                    4.0*M_PI*Foam::pow(r0, 3.0)/3.0
                   -4.0*M_PI*Foam::pow(r_inner, 3.0)/3.0
                   -V
                );
            f_prime = 4.0*M_PI*Foam::pow(r0, 2.0);
        }
        else if (shape == parallelepiped)
        {
            f =
                (
                    8.0*Foam::pow(r0, 3.0)
                   +4.0*Foam::pow(r0, 2.0)*((L2-L1)+(L3-L1)+2*(L2-L1)*(L3-L1))
                   -V
                );
            f_prime = 8.0*3.0*Foam::pow(r0, 2.0)
                +4.0*2.0*r0*((L2-L1)+(L3-L1)+2*(L2-L1)*(L3-L1));
            printf
            (
                "ERROR in radius from volume element function -- NOT PREPARED\n"
            );
        }
        else
        {
            printf("ERROR in radius from volume element function!!\n");
        }

        r1 = r0-f/f_prime;
    }

    return r1;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::
temperature_gradient(scalar Ti, scalar Tj, scalar ri, scalar rj)
{
    scalar dTdr = 0.0;

    /* currently only for spheres in this version of the code */
    if (shape == sphere)
    {
        /* dTdr = (Ti-Tj)/(ri*(ri/rj-1.0)); */
        dTdr = (Ti-Tj)/(ri-rj);
    }
    else
    {
        printf("Error in temperature gradient function!\n");
    }

    return dTdr;
}

template <class ParcelType>
void Foam::ReactingMultiphaseMBMParcel<ParcelType>::update_surface_temperatures
(
    scalar nextT_p3, scalar k_p3, scalar A, scalar r_b3, scalar r_c3,
    scalar Q_particle_conduction, scalar TInf_, scalar h, scalar emissivity
```

```
)
{
    /* currently only for spheres */
    label i = 0;
    scalar f = 0.0, f_prime = 0.0;
    scalar T0, T1;
    // scalar eta = L1-L3;

    T0 = TInf_;
    T1 = (TInf_+nextT_p3);

    while
    (
        (Foam::mag((T0-T1)/T0) > MY_QUITE_SMALL)&&(i <= MY_MAXIMUM_NEWTON_ITER_)
    )
    {
        i = i+1;
        T0 = T1;

        if (shape == sphere)
        {
            f =
                (h*A*(TInf_-T0)
                +
                view_factor*emissivity
                *Foam::constant::physicoChemical::sigma.value()
                *A*(Foam::pow(TInf_, 4.0)-Foam::pow(T0, 4.0))
                -
                k_p3*A*(T0-nextT_p3)/(r_b3-r_c3));
            f_prime =
                (
                    -h*A
                    +
                    view_factor*emissivity
                    *Foam::constant::physicoChemical::sigma.value()
                    *A*(-4.0*Foam::pow(T0, 3.0))
                    -
                    k_p3*A/(r_b3-r_c3)
                );
        }
        else
        {
            printf("ERROR in update surface temperatures function!!\n");
        }

        T1 = T0-f/f_prime;
    }

    T_send[0] = T1;
}

template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::heatRatio
(
  scalar temperature
)
{
    // scalar COCO2Ratio = 4.3*Foam::exp(-3390./temperature);
    //     return (COCO2Ratio+0.3)/(COCO2Ratio+1.);
    return 1.; //strange
}


template <class ParcelType>
Foam::scalar Foam::ReactingMultiphaseMBMParcel<ParcelType>::charFrontLocation
(
    scalar rb,
    label i,
```

```
    scalarField VCell_,
    scalarField VChar_,
    scalarField xc_,
    scalar xi_
)
{
    scalar rCharFront;

    if (i == FINE_PARTICLE_MESH_-1)
    {
        rCharFront = radius_from_volume(VChar_[i]);
    }
    else if (i == 0)
    {
        rCharFront = radius_from_volume
            (volume_of_element(0., xc_[i+1])+0.5*VCell_[i+1]+xi_*VChar_[i]);
    }
    else
    {
        rCharFront = radius_from_volume
            (volume_of_element(0., xc_[i+1])+0.5*VCell_[i+1]+VChar_[i]);
    }

    return rCharFront;
}

template <class ParcelType>
void Foam::ReactingMultiphaseMBMParcel<ParcelType>::
update_boundary_temperatures
(
    scalar k_p1, scalar k_p2, scalar k_p3, scalar k_p4,
    scalar A[4], scalar rb[4], scalar rp[4],
    scalar T_p[4], scalar T_b[4],
    scalar Q_evap, scalar Q_devol, scalar Q_char_comb
)
{
    //     scalar T0 = 0.0;
    scalar T1 = 0.0;
    scalar T2 = 0.0;
    scalar T3 = 0.0;
    //     scalar eta = L1-L3;
    scalar nextT_b1;
    scalar nextT_b2;
    scalar nextT_b3;
    scalar CA, CB;

    /* currently only for spheres */

    if (shape == sphere)
    {
        /* new Tb3: */
        CA = A[boundary_3]*k_p4/(rp[layer_4]-rb[boundary_3]);
        CB = A[boundary_3]*k_p3/(rb[boundary_3]-rp[layer_3]);
        T3 = (CA*T_p[layer_4]+CB*T_p[layer_3]-Q_char_comb)/(CA+CB);
        /* new Tb2: */
        CA = A[boundary_2]*k_p3/(rp[layer_3]-rb[boundary_2]);
        CB = A[boundary_2]*k_p2/(rb[boundary_2]-rp[layer_2]);
        T2 = (CA*T_p[layer_3]+CB*T_p[layer_2]-Q_devol)/(CA+CB);
        /* new Tb1: */
        CA = A[boundary_1]*k_p2/(rp[layer_2]-rb[boundary_1]);
        CB = A[boundary_1]*k_p1/(rb[boundary_1]-rp[layer_1]);
        T1 = (CA*T_p[layer_2]+CB*T_p[layer_1]-Q_evap)/(CA+CB);
    }
    else
    {
        printf("ERROR in update boundary temperatures function!!\n");
    }
```

```
    nextT_b3 = T3;
    nextT_b2 = T2;
    nextT_b1 = T1;

    /* return temperatures in array T_send */
    T_send[0] = nextT_b1;
    T_send[1] = nextT_b2;
    T_send[2] = nextT_b3;
}
```