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

Boyao Wang

Department of Energy and Process Engineering,
Norwegian University of Science and Technology,
Trondheim, Norway

January 16, 2022

Table of Contents

Context: Biomass combustion

- Biomass: a environmental friendly fuel, neutral in greenhouse gas emissions and renewable
- The biomass combustion $<=>$ biomass thermal-chemical conversion
- Modeling method: CFD-DEM
- Thermal-chemical conversion of the thermally thick particle cannot be solved using a point particle model
- Thermally thick particle model: interface-based model (IBM), mesh-based particle model (MBM)

Mesh-based particle model (MBM)

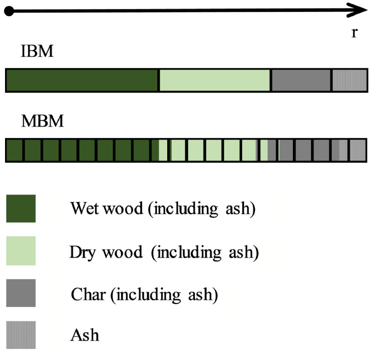


Fig.1 Schematics of one-dimensional MBM

The governing equation of the energy balance:

$$m_p C_p \frac{\partial T_s}{\partial t} = \frac{V_p}{\Gamma} \frac{\partial}{\partial r}(\Gamma_\kappa \frac{\partial T_s}{\partial r}) + S_{energy} \tag{1}$$

The coalCloud and parameterised inheritance

In $FOAM_SOLVERS/lagrangian/coalChemistryFoam/createClouds.H:

createClouds.H

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

The object coalParcels has a type of coalCloud.

The coalCloud and parameterised inheritance

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

coalCloud.H

```
    typedef ReactingMultiphaseCloud
    <
        ReactingCloud
        <
            ThermoCloud
            <
                KinematicCloud
                <
                    Cloud
                    <
                        coalParcel
                    >
                >
            >
        >
    > coalCloud;
```

The coalCloud and parameterised inheritance

coalParcel is defined in `$FOAM_SRC/lagrangian/coalCombustion/coalParcel/coalParcel.H/` as:

coalParcel.H

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

The coalCloud and parameterised inheritance

The ReactingMultiphaseCloud class, which is a templated class, is defined as follows:

ReactingMultiphaseCloud.H

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

The template parameter CloudType in ReactingMultiphaseCloud class when defining the coalCloud will be ReactingCloud. The ReactingMultiphaseCloud class also inherits the class CloudType, which is the ReactingCloud here.

The coalCloud and parameterised inheritance

Go the the definition of the ReactingCloud class:

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

The template parameter CloudType in ReactingCloud class when defining the coalCloud will be ThermoCloud. The ReactingCloud also inherits CloudType. The same logic goes until the innermost layer of the typedef of the coalCloud class. This is called parameterised inheritance.

Create the biomassCombustion library

Copy the `coalCombustion` library to a local folder and rename the keyword `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' {} \;
```

Modify the biomassCloud directory

Change 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/
```

Modify the biomassCloud directory

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' {} \;
```

Then, follow the report and modify the biomassMBMCloud.H and the basicReactingMultiphaseMBMCloud.H.

Modify the biomassParcel directory

Modify the file structure in the directory biomassParcel:

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

Modify the biomassParcel directory

```
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/*
```

Modify the biomassParcel directory

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/*
```

Then modify the biomassMBMParcel.H and the basicReactingMultiphaseMBMParcel.H following the text in report.

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' *
```

Then modify the ReactingMultiphaseMBMCloud class following the report text.

Modify the ReactingMultiphaseMBMParcel class

- Add members and member functions to ReactingMultiphaseMBMParcel.H. See details in the report, as too many members and member functions are added for implementing the MBM.
- Include the following code in the ReactingMultiphaseMBMParcel.H to add C language code (matrix2.h) to the C++ code.

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

  #include "matrix2.h"

  #ifdef __cplusplus
}
#endif
```

Modify the ReactingMultiphaseMBMParcel class

- Modify the calc function (where the MBM is implememted)

```
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();
```

Modify the ReactingMultiphaseMBMParcel class

```
  // * * * * * * * end initialization and start calculation * * * * * * * * * //
  while (tPar < dt)//t_stop)
  {
    ...
  }
  // To sum up all the mPars
  forAll(mPar_,i)
  {
    mParSum_ += sum(mPar_[i]);
  }
}
```

- Inside the `while` loop, the particle temperature, particle mass, and species fractions in the particle will be updated.

Modification to the constructors

```
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();
}
```

- Two new functions `readCoeff` and `initFields` are added to the body of the constructor to read extra data for the MBM.

Modify the ReactingMultiphaseMBMParcelIO.C

- readFields function in ReactingMultiphaseMBMParcelIO.C
- VCell_ which has a type of scalarField will be shown here as an example

```
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);

    ...
```

Modify the ReactingMultiphaseMBMParcelIO.C

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

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

- The VCell which has a type of IOField will read data from the run folder
- In the forAllIter loop, the data read by VCell will be assigned to the scalarField object called VCell_
- The same logic applies also for reading and assigning data

Modify the ReactingMultiphaseMBMParcelIO.C

- writeFields function in ReactingMultiphaseMBMParcelIO.C
- VCell_ which has a type of scalarField will be shown here as an example

```
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
```

Modify the ReactingMultiphaseMBMParcelIO.C

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

            ...

        }

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

- The `VCell` which has a type of `IOField` is constructed and ready to write data to the running folder.
- In the `forAllConstIter` loop, the data in the member `VCell_` will be assigned to `VCell`
- `VCell.write(np > 0)` will write the data to the running folder
- The same logic applies also for writing other data

Modify the ReactingMultiphaseMBMParcelIO.C

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

            ...

        }

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

- The VCell which has a type of IOField is constructed and ready to write data to the running folder.
- In the forAllConstIter loop, the data in the member VCell_ will be assigned to VCell
- VCell.write(np > 0) will write the data to the running folder
- The same logic applies also for writing other data

Finish of the modification to biomassCombustion

- The modification to the `biomassCombustion` directory is finished, the structure of this directory should look like:

The `clouds` and `parcels` directory

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

Modify the biomassChemistryFoam

- Starting by copying coalChemistryFoam and renaming the keyword coal to biomass

```
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
);
```

Modify the biomassChemistryFoam

- 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
- Remove rhoEffLagrangian in biomassChemistryFoam.C
- Rename coal* to biomassMBM* in EEqn.H, UEqn.H, setRDeltaT.H, YEqn.H, and biomassChemistryFoam.C using the following command.
- Modifications to the Make files refer to the supplied report

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

The Meschach library

The Meschach 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 conflict, 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
```

Meschach library code example

- Here is a typical and general example of using the Meschach to solve linear system ($Ax = b$) 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);
}
```

- In Meshchach, the user can also choose to use the QR and LLT decomposition

OpenFOAM library and code example

- Except for the iterative method, the linear system in OpenFOAM can be solved using LU or QR decomposition

```
    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));

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

The Eigen library and code example

- The Eigen library can be downloaded using the following command

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

- The Eigen library has both decomposition method and iterative methods for solving a linear system
- The Eigen does not need to compile, the user can include the required header file in the solver main file

Eigen library code example

- Here is a typical and general example of using the Eigen to solve linear system ($Ax = b$)

```
#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);
```

A tutorial case of a single biomass particle combustion

- The wood particle has a diameter 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
- In `particleInfoDict`, there are two main parts, where the user can set the mesh information in `meshInput` and reaction kinetics in `sigDevKin`

```
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
```

A tutorial case of a single biomass particle combustion

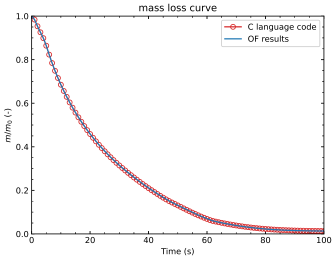
```
  FINE_PARTICLE_MESH 50;
  FINE_PARTICLE_MESH_DOUBLE 50.0;
  K_P_WET 0.26;
  K_P_DRY 0.12;
  K_P_CHAR 0.1;
  ASH_MASS_FRACTION 0.02;
  //daf in raw biomss the C H O ratios are from Neves review paper 2011 PECS
  //(poplar wood), char is only C.
  C_FRACTION 0.4941;
  H_FRACTION 0.0609;
  O_FRACTION 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;
}
```

A tutorial case of a single biomass particle combustion

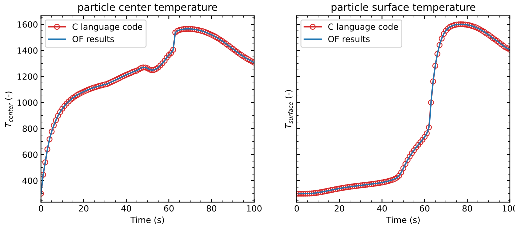
```
// single component kinetic ratio
sigDevKin
{
  DevKinA1 1.10e+11; /* (s-1) */
  DevKinE1 1.77e+05; /* (J/mol) */
  DevKinA2 9.28e+09; /* (s-1) */
  DevKinE2 1.49e+05; /* (J/mol) */
  DevKinA3 3.05e+07; /* (s-1) */
  DevKinE3 1.25e+05; /* (J/mol) */

  DryKinA 5.13e+10;
  DryKinE 88000.;
}
```

Code verification

- The ported code will be verified with the original code
- The particle mass loss and temperature profile during the thermal-chemical conversion will be shown



Particle mass loss curve



Particle surface and center temperature profile

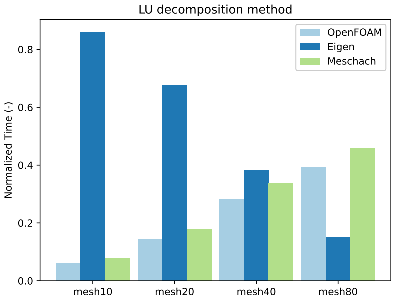
Linear system solver performance comparison

- The following command can be used to do the profiling.
- Two packages have to be installed to use these commands, which are `valgrind` and `kcachegrind`

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

- Three decomposition methods (LU, QR, LLT) and one iterative method will be used in terms of comparison
- The compared solvers are `Meschach`, OpenFOAM, and `Eigen`

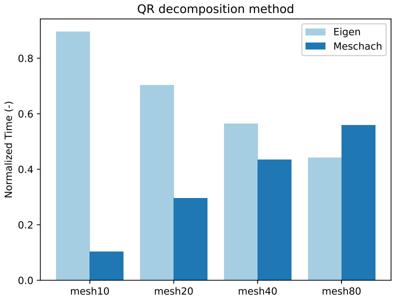
LU decomposition solver comparison



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

- 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

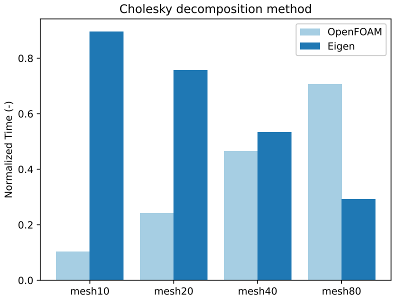
QR decomposition solver comparison



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

- The QR decomposition solver of `OpenFOAM` is too poor to compare with other libraries, therefore now shown
- When the number of meshes increases, the relative performance of the QR decomposition of `Eigen` increases relative to `Meschach`
- When the number of meshes increases to 80, the `Eigen` solver performs the best of all solvers

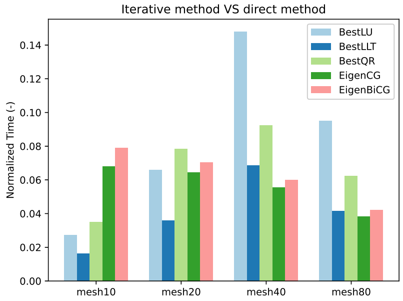
LLT decomposition solver comparison



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, the `Eigen` solver performs the best of all solvers

Iterative solver in Eigen



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.

- The figure 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
- When the mesh number is lower than 20, the direct method performs better, especially the LLT method used in OpenFOAM
- The best solver when mesh number is larger than 40 is the conjugate gradient (CG) method in `Eigen`