Implementation of growing CCM library to reduce chemistry calculation time

Yuchen Zhou

Department of Energy Sciences/Fluid Dynamics,
Lund University,
Lund, Sweden

January 19, 2023

Backgrounds

Accurate numerical simulation results are demanded by both academic researches and industrial purposes

Simulations of reactive flows are often costly and unacceptable. Simplifications are required!

We can

- simplify the reaction mechanism
- decrease the amount of reaction calculation by reusing some results

This presentation will discuss the second method

A double swirl burner

The number of equations

I should put the continuity equation here! (1)

I should put the momentum equation here! (3)

The energy Equation (1)

$$\frac{\partial \rho h}{\partial t} + \frac{\partial \rho u_j h}{\partial x_j} + \frac{\partial \rho K}{\partial t} + \frac{\partial \rho u_j K}{\partial x_j} - \frac{\mathrm{d}p}{\mathrm{d}t} = \frac{\partial}{\partial x_j}(\alpha_{eff} \frac{\partial h}{\partial x_j}) + \dot{Q}$$

The species transport equation (N Equations for N species)

$$\frac{\partial \rho Y_i}{\partial t} + \frac{\partial \rho u_j Y_i}{\partial x_j} = \frac{\partial}{\partial x_j}(\rho D_i \frac{\partial Y_i}{\partial x_j}) + \omega_i$$

And usually a very stiff reaction rate ODE system!

Elementary reactions 1

What do I mean by saying a very stiff reaction rate ODE system?

A takeaway for CFDers is that solving this reaction rate system (chemistry simulations) can take up most of the time when simulating a reacting case.

A single step reaction might look like

$$2H_2 + O_2 \rightarrow 2H_2O$$

But in reality, this global reaction consists of many elementary reactions, which might look like

$$H_2 + M \rightarrow H + H + M$$

$$H + O_2 \rightarrow OH + O$$

$$OH + H_2 \rightarrow H_2O + H$$

Elementary reactions 2

(copied from the previous slide)

$$H_2 + M \rightarrow H + H + M$$

$$H + O_2 \rightarrow OH + O$$

$$OH + H_2 \rightarrow H_2O + H$$

We can see 9 species and 3 reactions are involved.

Please note this is just a route to reach the final product.

For a fuel as simple as methane (CH_4), it takes 53 species and 325 reactions to fully (maybe) describe that. This will indicate a very large reaction system, as well as 53 species transport equations to solve.

And the ODE system is very stiff (sensitive to the size of the time steps), which is hard to solve.

The calculation of reaction rates

The reaction rates can be calculated with the Arrhenius Formula.
For example, the reaction rate of a reaction given by

$$\nu_{\mathrm{A}}\mathrm{A} + \nu_{\mathrm{B}}\mathrm{B} \rightarrow \nu_{\mathrm{C}}\mathrm{C} + \nu_{\mathrm{D}}\mathrm{D}$$

can be calculated by

$$\omega_c = \frac{\mathrm{d}C_A}{\nu_A \mathrm{d}t} = \frac{\mathrm{d}C_B}{\nu_B \mathrm{d}t} = -\frac{\mathrm{d}C_C}{\nu_C \mathrm{d}t} = -\frac{\mathrm{d}C_D}{\nu_D \mathrm{d}t} == C_A^{\nu_A} C_B^{\nu_B} k(T)$$

where

$$k(T) = AT^b exp(-\frac{E_a}{R_u T})$$

Here,

- A, b, E_a are three empirical parameters
- T is temperature.
- C_i is the molar fraction of species i. It can be calculated from mass fraction of species i by $\rho Y_i / MW_i$, where the Y_i and MW_i is the mass fraction and molar weight of species i, respectively

How the reaction rate influence the solution system

Basically, we can see that the elementary reaction rate is a function of the molar fraction of reactants and temperature.
Also, turning reactants from the left to products on the right will generate/consume energy ($\dot{q}$) (or formation enthalpy difference), which will be proportional to the reaction rate.
By knowing how quick the reaction is, the heat release rate of the energy equation $\dot{Q}$ can be calculated by summing up the $\dot{q}$ of each individual elementary equation

$$\frac{\partial \rho h}{\partial t} + \frac{\partial \rho u_j h}{\partial x_j} + \frac{\partial \rho K}{\partial t} + \frac{\partial \rho u_j K}{\partial x_j} - \frac{\mathrm{d}p}{\mathrm{d}t} = \frac{\partial}{\partial x_j}(\alpha_{eff}\frac{\partial h}{\partial x_j}) + \dot{Q}$$

The source term of the species transport equation ω_i can also be obtained.

$$\frac{\partial \rho Y_i}{\partial t} + \frac{\partial \rho u_j Y_i}{\partial x_j} = \frac{\partial}{\partial x_j}(\rho D_i \frac{\partial Y_i}{\partial x_j}) + \omega_i$$

$$\dot{Q} = \sum \nu_k \omega_{c,k} \dot{q}_k \quad \omega_i = \sum \nu_k \omega_{c,k} MW_k$$

Then the system is completed!

A quick review and possible simplifications

The solution of chemical reactions in a reacting case is computationally expensive.

A global reaction usually consists of many elementary reactions, forming a very stiff ODE system.

The reaction rates can be calculated by Arrhenius Formula and will influence the source term of the energy equation and the species transfer equation.

The reaction rates are dependent on the temperature, pressure, and mass fraction of species. Luckily, not every component is important, some key variables (control variables, or CCM variables in this project) might be extracted to describe the behavior of the whole ODE system.

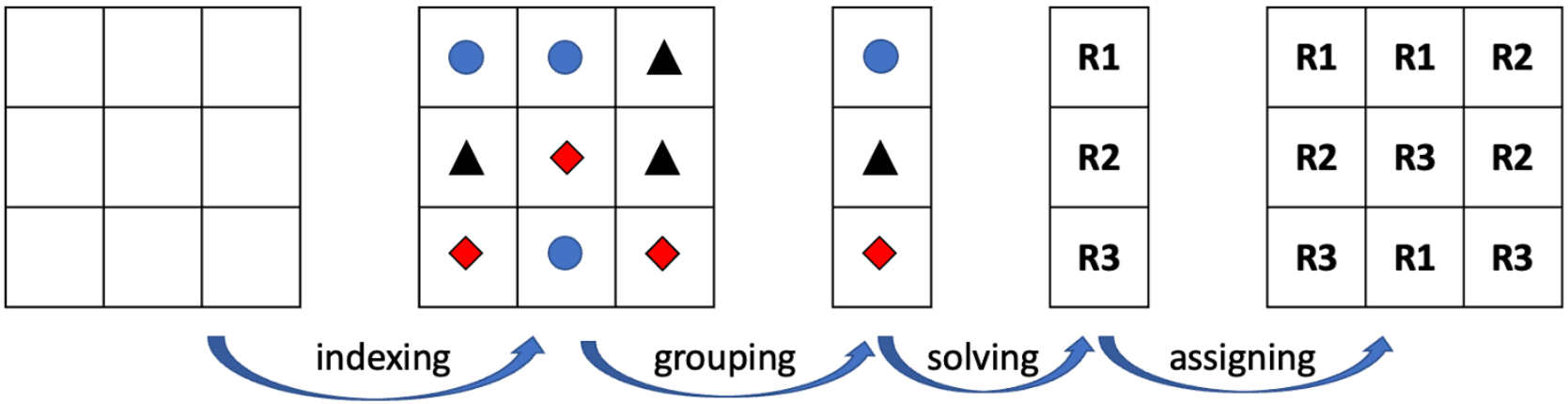
$$RR = RR(var_1, var_2, var_3, ...var_{n-1}, var_n)$$

The reaction states for many cells are very similar in most cases. They values of key variables are similar! And it is fine for them to SHARE the same reaction rate.

About CCM

The CCM method also intends to reduce the amount of chemistry calculation, by grouping the physical cells with the same "reaction states" and letting them share a common reaction rate.

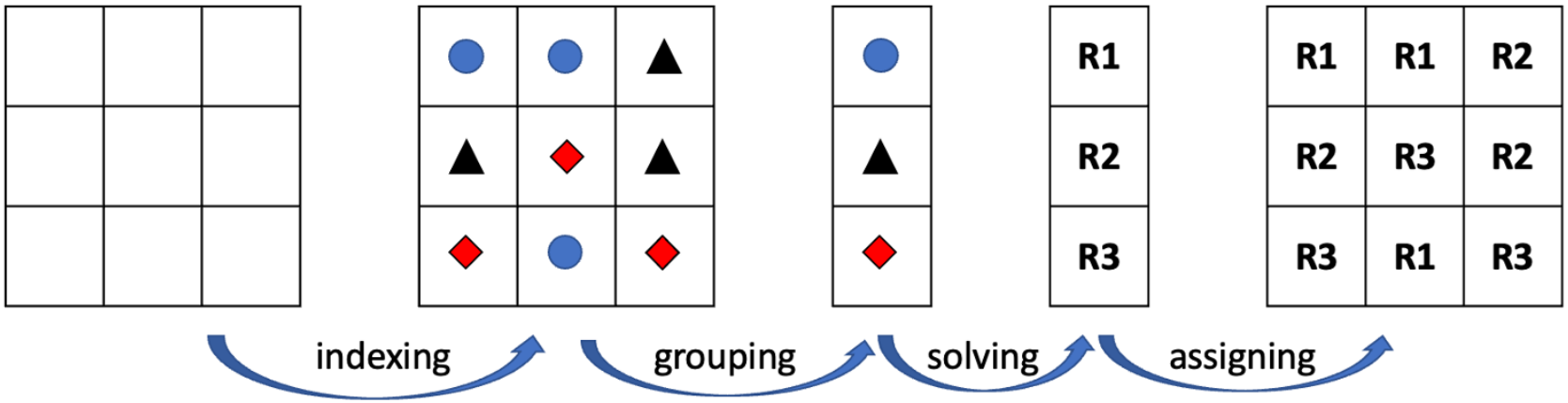
CCM variables : (T, p, ϕ, χ, Y_1, ... Y_N)



The procedure of CCM

CCM procedures

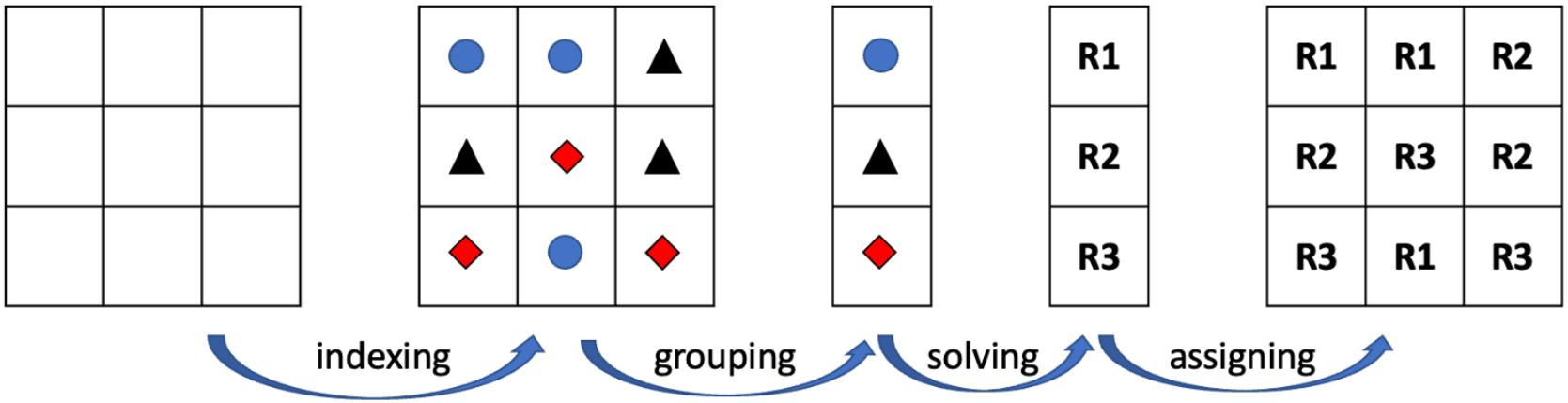
CCM variables : $(T,\ p,\ \phi,\ \chi,\ Y_1,\ \ldots\ Y_N)$



The procedure of CCM

- Suppose we have nine physical cells. We can calculate the values of their CCM variables.(indexing)
- Some cells will share the same values for their CCM variables. They can be classified into the same group (Map to the phase space).(grouping)
- Solve the reaction rates for cells in the phase space.(solving)
- attribute each physical cell a reaction rate.(assigning)

CCM procedures

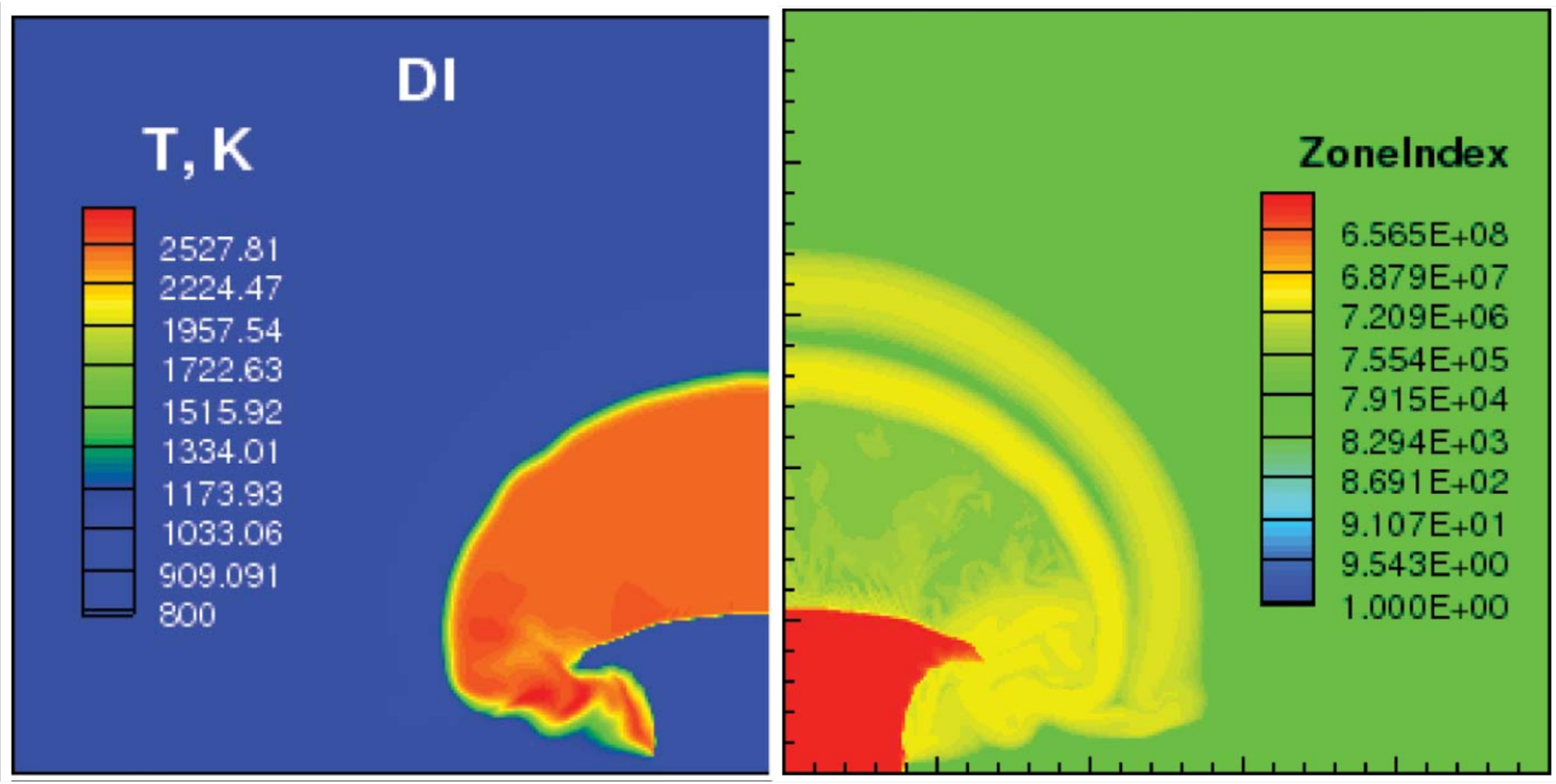


The procedure of CCM

By doing this, we will reuse the reaction rates in each time step.

If we reuse the results over many time steps, the method becomes **growing** CCM.

CCM procedures



ZoneIndex information for a jet flame

You can see that many cells actually share similar reaction states. It is possible to use this feature to achieve acceleration.

CCM settings

Necessary settings

- Choose a CCM variable set
- Set the min/max/step values of the CCM variables

Based on the above information, the program will calculate a index in each dimension for each cell.
Cells with similar reaction states will share the same index.
For dimension T, we set the min/max/step values as 300/2500/100 K.

For a cell with a 580K temperature, its ZoneIndex in this dimension will be

$$zIndex = floor(\frac{value - min}{step}) = floor(\frac{580 - 300}{100}) = 2$$

We can use four digits to represent it, making it `0002`.
A `ZoneIndex` considering all CCM dimensions can look like `(0030, 0025, 0002, 0044)`.
We usually concatenate them for convenience, making the final `ZoneIndex`
`0030 0025 0002 0044`

Set the chemistry model

The chemistry is set in `constant/chemistryProperties` file in the case folder.

chemistry settings in constant/chemistryProperties

```
chemistryType
{
    solver          ode; // noChemistrySolver, eulerImplicit
    method          standard;  // TDAC, CCM
}

chemistry       off;

initialChemicalTimeStep 1e-07;

odeCoeffs
{
    solver          seulex;
    absTol          1e-12;
    relTol          1e-07;
}
```

Call a chemistry model

reactingFoam.C

```
#include "rhoEqn.H"
while (pimple.loop())
{
    #include "UEqn.H"
    #include "YEqn.H"    // Species transport equations
    #include "EEqn.H"  // Energy equation
    // --- Pressure corrector loop
    while (pimple.correct())
    {
        ...// solve pEqn
    }

    if (pimple.turbCorr())
    {
        turbulence->correct();
    }
}
```

Call a chemistry model

reactingFoam.C

```
reaction->correct(); //// the chemistry is computed here!
Qdot = reaction->Qdot();
volScalarField Yt(0.0*Y[0]);

forAll(Y, i)
{
    ...
}
```

reactingFoam/createFields.H

```
autoPtr<CombustionModel<psiReactionThermo>> reaction
(
    CombustionModel<psiReactionThermo>::New(thermo, turbulence())
);
```

Note: There are some runtime selection stuff that the author has not fully understood. But the general route presented here is correct.

Solving the reaction rates

part from the solve function in StandardChemistryModel.C

```
template<class ReactionThermo, class ThermoType>
template<class DeltaTType>
Foam::scalar Foam::StandardChemistryModel<ReactionThermo, ThermoType>::solve
(
    const DeltaTType& deltaT
)
{
    ...
    if (Ti > Treact_)
    {
        const scalar rhoi = rho[celli];
        scalar pi = p[celli];

        for (label i=0; i<nSpecie_; i++)
        {
            // calculate the molar fractions from the mass fractions
            c_[i] = rhoi*Y_[i][celli]/specieThermo_[i].W();
            c0[i] = c_[i];
        }
    }
}
```

Solving the reaction rates

Continuing from the previous slide

Part of codes from the StandardChemistryModel.C

```
1
2  // Calculate the chemical source terms
3  while (timeLeft > SMALL)
4  {
5      scalar dt = timeLeft;
6      this->solve(c_, Ti, pi, dt, this->deltaTChem_[celli]); // update c
7      timeLeft -= dt;
8  }
9  ...
10 for (label i=0; i<nSpecie_; i++)
11 {
12     // calculate reaction rates(RR)
13     RR_[i][celli] =
14         (c_[i] - c0[i])*specieThermo_[i].W()/deltaT[celli];
15 }
```

You have seen that the solve function calculate the molar fraction from mass fraction, and then call the ODE solver to solve the reaction rates.

Some comments

The standard chemistry library can access the flow fields.

Based on the thermophysical states, the reaction rates can be determined!

Modifying the `solve` function will change the behaviors of chemistry calculation. We can make it zero (no chemistry, but I have not tried), or implement the CCM method!

Optional: compile your own standard chemistry model

It is a little bit tricky to do that, but it is possible. You might follow the procedures below but some adjustments might be required

Some general stuff

- Copy `$FOAM_SRC/thermophysicalModels/chemistryModel` to your working directory
- Delete the `functionObjects` and the `lnInclude`

In `chemistryModel` directory

- In `chemistryModel folder`, delete everything except the `StandardChemistryModel` and `BasicChemistryModel`.
- You should only keep `BasicChemistryModels.C` in the `BasicChemistryModel` directory.
- Replace `StandardChemistry` with `myStandardChemistry` in EVERY PLACE in the `chemistryModel` directory.
- Replace `standard` typename with `mystandard` or any name you like in `myStandardChemistryModel.H`. By doing this, the `mystandard` chemistry model will be added to the runtime selction options in the future procedures.

Optional: compile your own standard chemistry model 2

In `chemistrySolver`

- Delete everything except the `chemistrySolver` directory.
- In `chemistrySolver` subdirectory, only keeps `makeChemistrySolvers.C` and `makeChemistrySolverTypes.H`.
- In `makeChemistrySolverTypes.H`, find the macro part dealing with `StandardChemistryModel`, replace them with `myStandardChemistryModel`. Delete everything relevant with the `TDAC` model.
- Also, include `myStandardChemistryModel.H` in `makeChemistrySolverTypes.H`.

Then use the following `Make/{files,options}`

Make/files

```
chemistryModel/BasicChemistryModel/BasicChemistryModels.C
chemistrySolver/chemistrySolver/makeChemistrySolvers.C

LIB = $(FOAM_USER_LIBBIN)/libmychemistryModel
```

Optional: compile your own standard chemistry model 3

Use the following `Make/options`

Make/optionss

```
EXE_INC = \
    ... // keep original including files and add the following line
    -I$(LIB_SRC)/thermophysicalModels/chemistryModel/lnInclude

LIB_LIBS = \
    ... // keep original including files and add the following line
    -lchemistryModel
```

Then execute `wmake` to enjoy or wait for some instructions on further modifications...

Contributions

The authors of the CCM library include some former members of Bai's research group at Lund University and my current colleagues.

Yuchen(me) re-organized the structure and added the growing feature.

General structure

The solution of CCM follows the four stages we discussed before, consisting of indexing, grouping, solving, and assigning.

Before solving the functions, the thermophysical variables of physical cells sharing the same reaction states should be averaged to obtain their representatives, which is avging.

The `assigning` stage is done by the `attributeRR.H` header file.

solve function in CCM

```
1    #include "indexing.H"
2
3    #include "grouping.H"
4
5     // "avging.H" + "solving.H" are responsible for the solving stage
6    #include "avging.H"
7
8    #include "solving.H"
9
10   #include "attributeRR.H" // assigning stage
```

Indexing

In the indexing part, we iterate every CCM variables and calculate their index(pos) and concatenate them to form the final ZoneIndex

indexing

```
1    for(label icell= 0; icell < fieldSize; icell++) // for each cell
2    {
3        for(label mccm=0; mccm < nMCCM_; mccm ++) // for each dimension
4        {
5            scalar   pos = (MZ[mccm][icell]-MZmin_[mccm])/ZoneSpan_[mccm];
6            word tempWord = std::to_string(static_cast<label>(pos));
7            tempWord.insert(0,stringDigit_ - tempWord.size(),'0');
8            ZoneIndex[icell] = ZoneIndex[icell] + tempWord;
9        }
10   }
```

Grouping

grouping

```
1   // get ZoneNumber for each not calculated cells
2   auto NCResults = getZoneNumber(notCalculatedZoneIndex, stringDigit_); //not
     calculated results
3   scalarField NCZoneNumber = NCResults.ZoneNumber;
4   List<word> gCCMZoneIndex(NCResults.ZoneIndex);
5   label  numOfNCActive = NCResults.count;
6
7   if(growingCCM_)
8   {
9       oldTableSize_ = curTableSize_;
10      newTableSize_ = numOfNCActive;
11      curTableSize_ = oldTableSize_ + newTableSize_;
12  }
13  else
14  {
15      oldTableSize_ = 0;
16      newTableSize_ = numOfNCActive;
17      curTableSize_ = newTableSize_;
18  }
19  ... //The ZoneNumber should be corrected here for the growing situation.
     Please read the report for more information.
```

Grouping

Based on codes developed by my colleagues, I implemented another library to manage the grouping process.

grouping

```
1    // get ZoneNumber for each not calculated cells
2    auto NCResults = getZoneNumber(notCalculatedZoneIndex, stringDigit_);
3    scalarField NCZoneNumber = NCResults.ZoneNumber;  // the location to access
      RR
4    List<word> gCCMZoneIndex(NCResults.ZoneIndex); // the zoneIndex in the
      phaseSpace
5    label  numOfNCActive = NCResults.count;
```

Generally, the library does the following things, provided with the `ZoneIndex` of each physical cell.

- group them and obtain a much smaller `ZoneIndex` set in the phase space.
- tell each cell(in physical space) where (in phase space) it can access its reaction rates

Managing index in parallel is a little bit tricky. Please refer to the report for more information.

Grouping

The grouping process is achieved by constructing a hash table.

general grouping guide

```
1   List<word> lZoneIndex(0);
2   List<label> oinl(oZoneSize);
3   label       lZoneIndexCount(0);
4   HashTable<scalar, word> HashMapL(2*oZoneSize); // Here we construct a hashtable to store non-repeating index
5   forAll(ZoneIndex, zi)
6   {
7       word target = ZoneIndex[zi];
8       if (!HashMapL.found(target)) // new index will be added to the hashtable
9       {
10          // set
11          HashMapL.set(target,lZoneIndexCount);
12          // append
13          lZoneIndex.append(target);
14          // other
15          oinl[zi] = lZoneIndexCount; // record its location
16          lZoneIndexCount++;
17      }
18      else // record its position
19      {
20          oinl[zi] = HashMapL.find(target)();
21      }
22  }
```

Solving

The solving process is not very different from the `solve` function we discussed before in the `StandardChemistryModel`.

Similar parts The class dealing with chemistry can access the flow field information. Molar fractions of species will be calculated. Then the ODE solver is called to solve the reaction rates.

Different parts In StandardChemistryModel, each core will solve the RR of the local portion. But in CCM, the RR in the phase space is solved and mapped back. The number of cells are different from those in the physical space.

Assigning

assigning

```
1    forAll(this->mesh().C(), icell) // for each cell
2    {
3        const label izone = ZoneNumber[icell];
4        for(label i=0; i < this->nSpecie(); i++) // for species i
5        {
6            this->RR_[i][icell] = old_gRRCCM_[i][izone]*rho[icell]; // drho/dt =
     rho dYi/dt
7        }
8    }
```

Adding growing feature

pre-lookup

```
1     if (growingCCM_)
2     {
3         ...
4         for (label icell = 0;icell < fieldSize; icell++)
5         {
6             word index = ZoneIndex[icell];
7             if (totalHashTable_.found(index)) // already calculated before
8             {
9                 ZoneNumber[icell] = totalHashTable_.find(index)();
10                ...
11            }
12            else // not calculated cells
13            {
14                notCalculatedCells.append(icell);
15            }
16
17        }
18    }
```

Adding growing feature

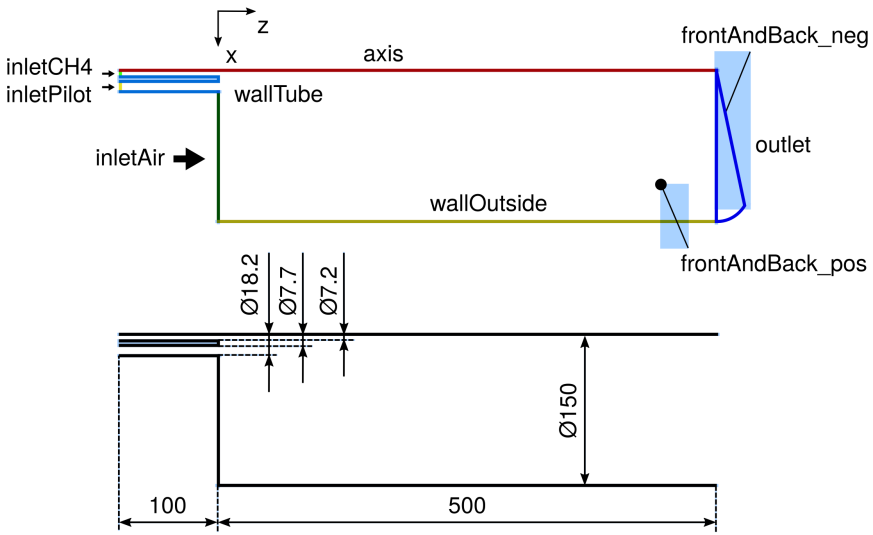
additional operations after grouping

```
1   if(growingCCM)
2   {
3       forAll(gCCMZoneIndex, i)
4       {
5           // store new reaction states to the totalHashTable
6           totalHashTable.set(gCCMZoneIndex[i],oldTableSize + i);
7       }
8       for (label i=0;i!=newTableSize;i++)
9       {
10          // store new reaction states
11          old_gCCMZoneIndex[oldTableSize+i] = gCCMZoneIndex[i];
12      }
13  }
```

additional assigning operations

```
1   if(growingCCM_)
2   {
3
4       forAll(gRRCCM,specie) // for each species
5       {
6           for (label i=0; i!=newTableSize_; i++) // for each cell
7           {
8               // store reaction rates in the list
9               old_gRRCCM_[specie][oldTableSize_+i] = gRRCCM[specie][i];
10          }
11      }
12  }
```

The SandiaD case



A schematic of Sandia burner

inletCH4:

- $\phi = 3$
- CH4 and air
- T 300K
- $v = 49.6\ m/s$

inletPilot:

- burnt methane products
- $T = 1880$K
- $v = 11.4\ m/s$

inletAir:

- air
- T 300K

Compile the code and prepare a case

Go to the code folder. Source the OpenFOAM environment.
Then execute:

```
./Allwmake
```

All relevant packages will be compiled!

Prepare a case

- Prepare a combustion case using `reactingFoam` / `rhoReactingFoam` / `sprayFoam`
- Set the CCM Method by changing `chemistryType.method` from `standard` or `TDAC` to `CCM` in `constant/chemistryProperties`
- Use `laminar` as the `combustionModel` in `constant/combustionProperties` (`EDC` is also okay but the reaction rates are not purely dependent on the reaction rates calculated in the chemistry solver. The turbulence effects will be considered in this case!).

Configure the CCM settings

The user are required to configure the CCM dictionary in `constant/chemistryProperties`

A piece of code

```
CCM
{
    growingCCM            off;
    ...
    stringDigit           4; //default to be 4
    maxTableSize      500000;
    hashTableSize     1500000;
    deleteRatio       0.5;

    ...
    min:max:SpanZoneJe 0 5 0.01;   // equivalence ratio phi
    min:max:SpanZoneT 300 2200 1;  // temperature T
    min:max:SpanZoneXi 0 1 0.025;  // scalar dissipation rate Xi
    chemicallyFrozenT 300;
    maxFlammabilityPhi 20;
    ...
}
```

Configure the CCM settings

The user also need to choose some species to add them to the CCM variables.

A piece of code

```
CCM
{
    ...
    CCMspecies
    {
        CH4             0 0.2 0.001;
        N2              0 1 0.001;
        HO2             0 1e-3 1e-6;
    }
}
```

Run the case

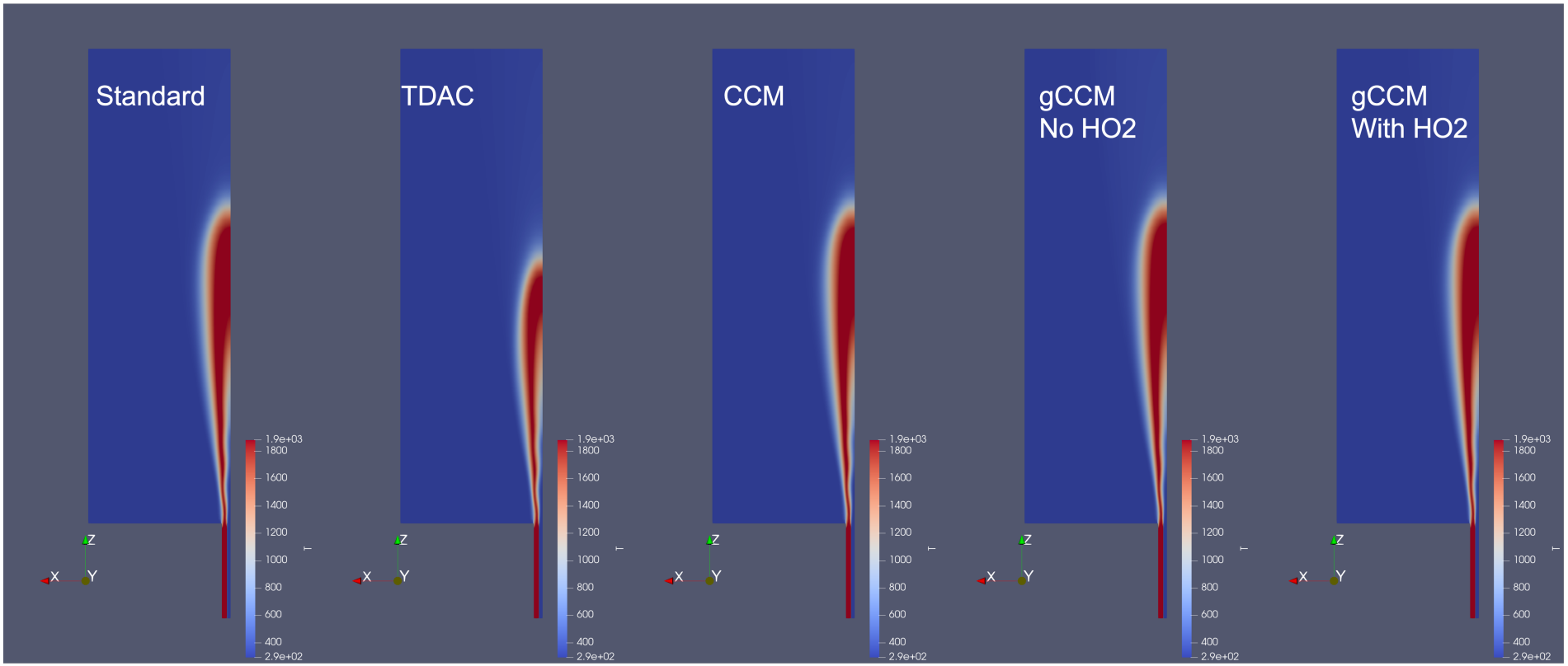
Run

```
reactingFoam
```

Or run in parallel (4 cores in this example) Note you might need to `decomposePar` first

```
mpirun -n 4 reactingFoam -parallel
```

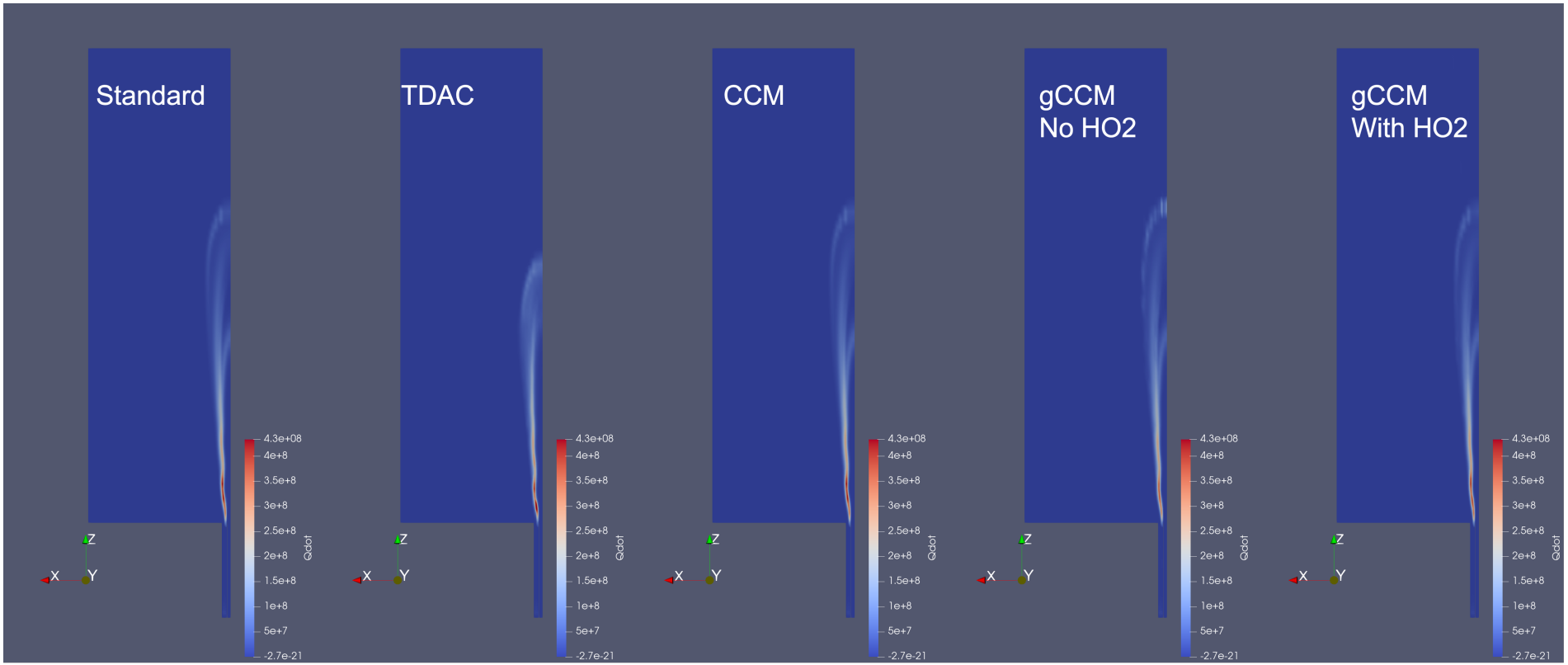
Results



T distribution

Compared to standard chemistry model, the CCM and growing CCM performs slightly better than the TDAC model in temperature predictions

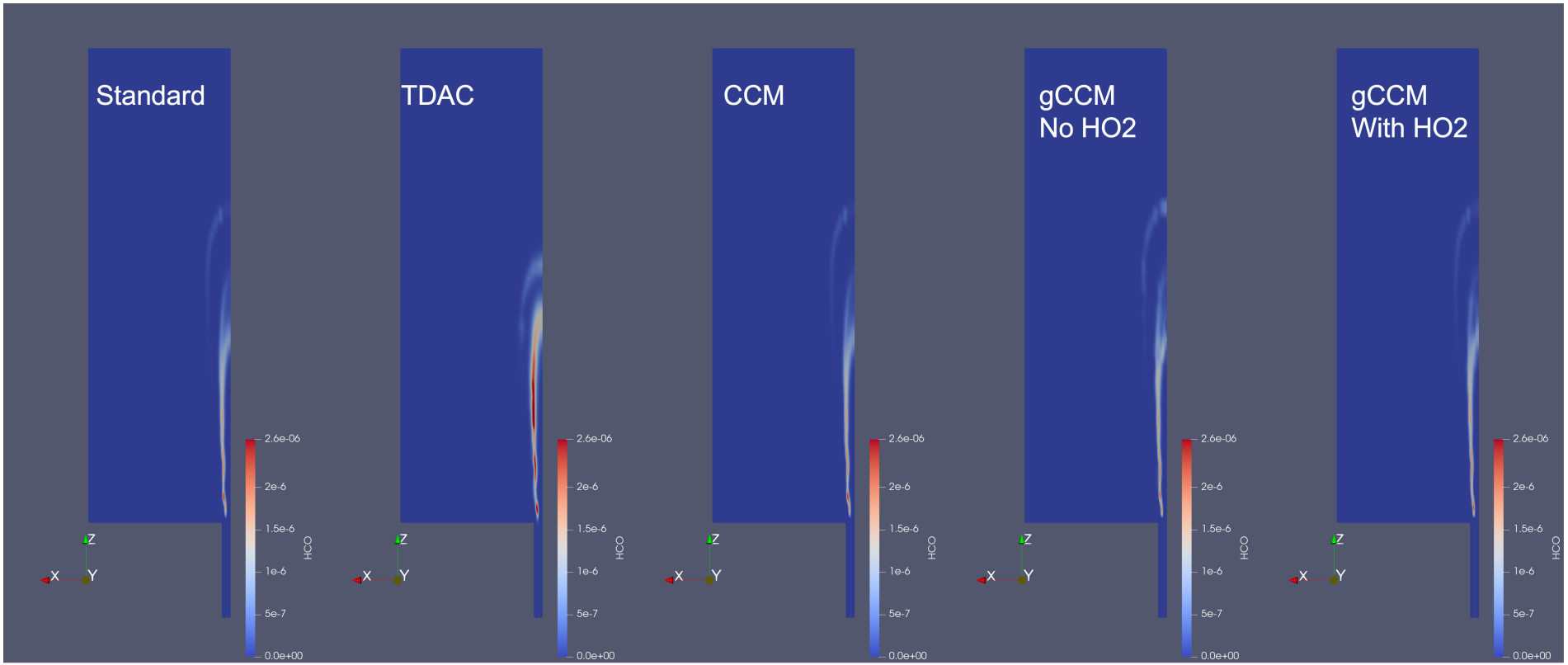
Results



Qdot distribution

Compared to standard chemistry model, the CCM and growing CCM performs slightly better than the TDAC model in heat release rate predictions.

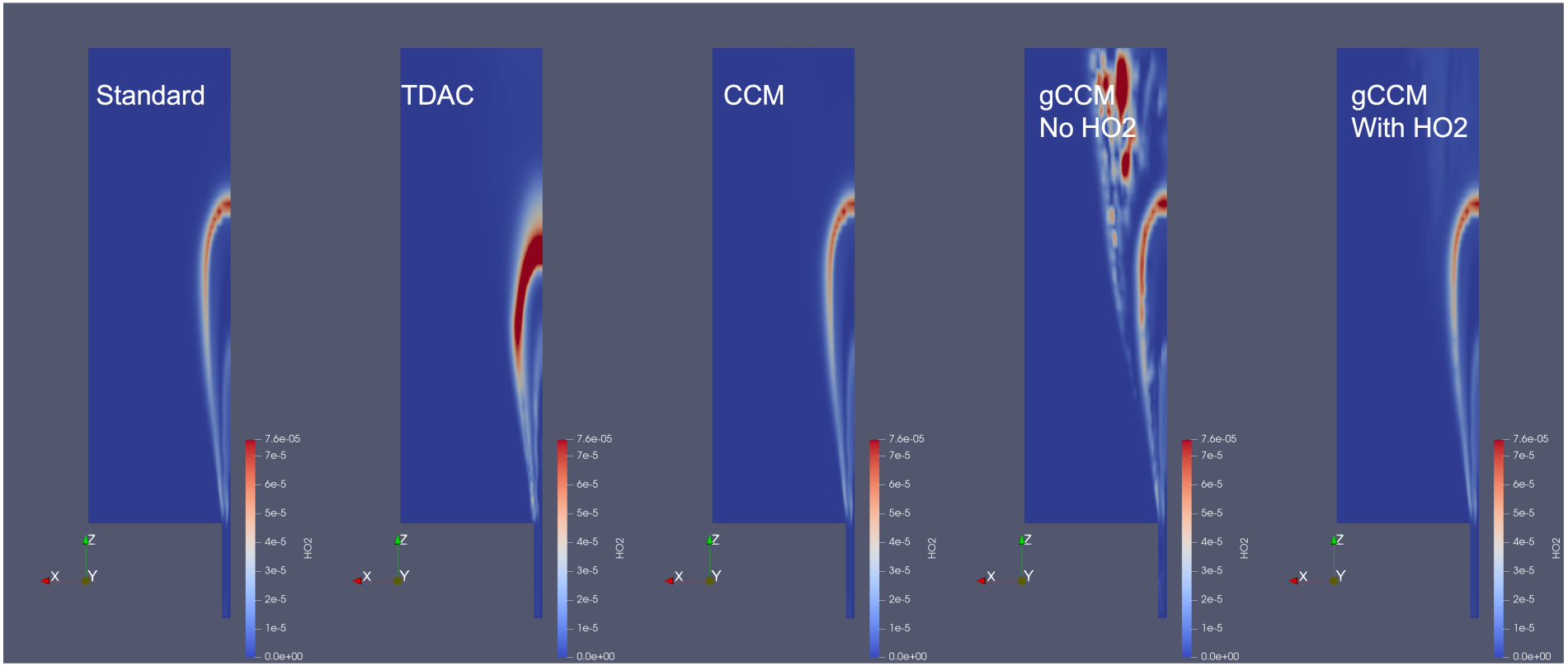
Results



HCO distribution

Compared to standard chemistry model, the CCM and growing CCM performs slightly better than the TDAC model in HCO predictions.

Results



HO2 distribution

Compared to standard chemistry model, the growing CCM cannot predict HO2 distribution very well. Adding this species to the CCM variables will improve the results.

Speed test

Generally, CCM has no speed advantage over the standard chemistry method.
This is because this two-dimensional case cannot reuse too many reaction rates.
Results will get better in three-dimensional cases with millions of cells.
The growing CCM method is still a little bit slower than the TDAC model. But it can perform better in the predictions of many fields.
However, owing to the inappropriate choice of CCM variables and the "history effects", the case might diverge.

Table: Base case benchmarks

Example	Time t
standard	1154.89 s
TDAC	232.11 s
CCM	1352.98 s
gCCM	351.11 s
gCCM+HO2index	454.74 s

Questions and answers

Many thanks for listening