10th May 2018, Brighton

Numerical Simulations of evaporating droplet clouds using Fully Lagrangian Approach;The implementation of the model in OpenFoam

Introduction

- Basic Implementation
- Evaporation model in FLA
- Single droplet Calculation

A few equations

Two-way Coupling, what about folds?

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho U U) = S_{evap}$$

$$\frac{\partial \rho U}{\partial t} + \nabla \cdot (\rho U U) - \nabla \cdot \mu_{eff} \nabla U - \nabla \cdot \mu_{eff} \left[(\nabla U)^T - \frac{2}{3} tr\left((\nabla U)^T\right) I \right] = \rho g - \nabla p + S_{mom}$$

$$\frac{\partial \rho Y_i}{\partial t} + \nabla \cdot (\rho U Y_i) - \nabla \cdot \frac{\mu_{eff}}{Sc} \nabla Y_i = S_{evap}$$

$$\frac{\partial \rho h_s}{\partial t} + \nabla \cdot (\rho U h_s) - \nabla \cdot \alpha_{eff} \nabla h_s = \frac{Dp}{Dt} + S_{heat}$$

$$m_p \frac{du_p}{dt} = \vec{F}_D$$

$$n_p = \frac{n_0}{|J|} \qquad J_{ij} = \left(\frac{\partial x_i}{\partial x_{j0}} \right)$$

$$\frac{\partial \omega_{ij}}{\partial t} = \beta \left(\frac{\partial u_i}{\partial x_{j0}} - \omega_{ij} \right)$$

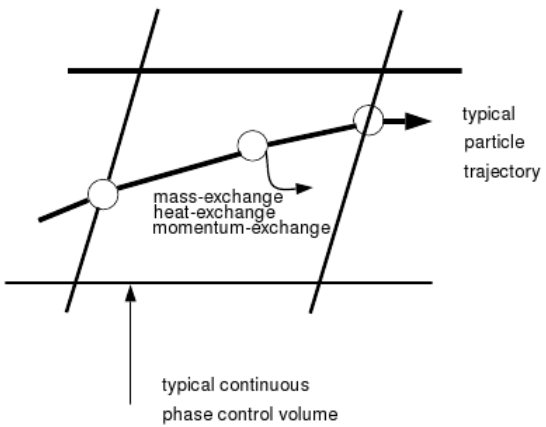
$$\omega_{ij} = \frac{\partial J_{ij}}{\partial t} = \frac{\partial v_{ij}}{\partial x_{j0}}$$

$$\frac{\partial J_{ij}}{\partial t} = \omega_{ij}$$

$$\frac{\partial \omega_{ij}}{\partial t} = \beta \left(\sum_k \left(J_{kj} \frac{\partial u_i}{\partial x_k} \right) - \omega_{ij} \right)$$

$$\dot{q}_{FLA} = \dot{q} n_d$$

$$\dot{m}_{FLA} = \dot{m} n_d$$



Implementation

- pierre@moco303-z640:~/OpenFOAM/pierre-v1712/src/lagrangian/intermediate/parcels/Templates/FlaParcel
  - FlaParcel.C FlaParcel.H FlaParcelI.H FlaParcelIO.C FlaParcelTrackingDataI.H
  - That’s it??
  - Not really.

Some code examples

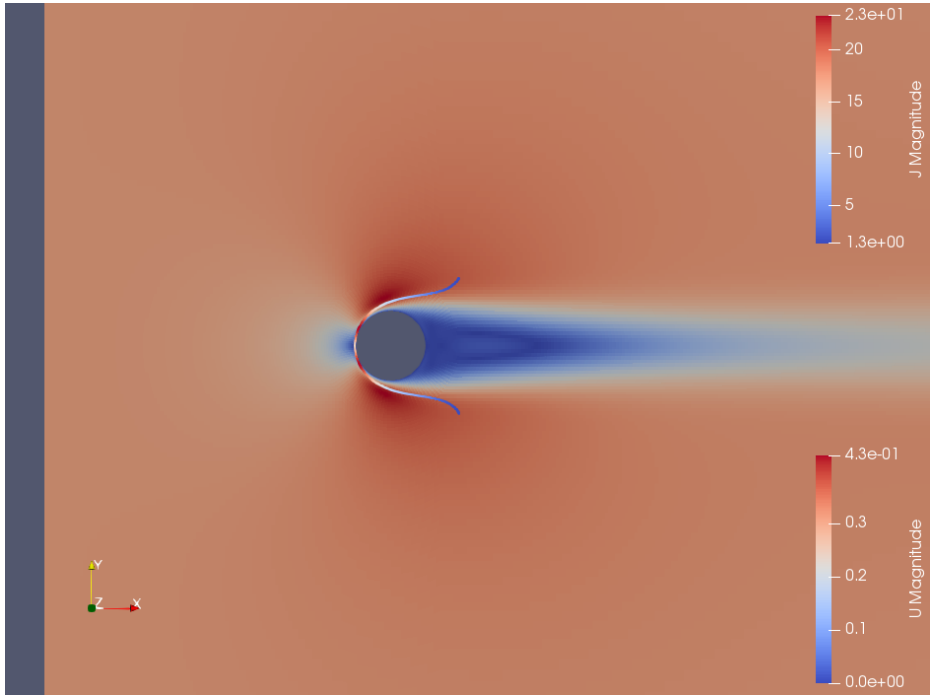
```
template<class ParcelType>
template<class TrackCloudType>
const Foam::tensor Foam::FlaParcel<ParcelType>::calcW
(
    TrackCloudType& cloud,
    trackingData& td,
    const scalar dt,
    const scalar Re,
    const scalar mu,
    const scalar mass,
    const vector& Su,
    vector& dUTrans,
    scalar& Spu
) const
{
    const typename TrackCloudType::parcelType& p =
        static_cast<const typename TrackCloudType::parcelType&>(*this);
    typename TrackCloudType::parcelType::trackingData& ttd =
        static_cast<typename TrackCloudType::parcelType::trackingData&>(td);

    const typename TrackCloudType::forceType& forces = cloud.forces();

    // Momentum source due to particle forces
    const forceSuSp Fcp = forces.calcCoupled(p, ttd, dt, mass, Re, mu);
    const forceSuSp Fncp = forces.calcNonCoupled(p, ttd, dt, mass, Re, mu);
    const forceSuSp Feff = Fcp + Fncp;
    const scalar massEff = forces.massEff(p, ttd, mass);
    // New particle deformation rate
    //~~~~~~~~~~~~~~~~~~~~~~
    const tensor FW = forces.calcWCoupled(p, ttd, dt, mass, Re, mu);

    // Update W - treat as 3-D
    const tensor abp =FW&J_/massEff;;
    const scalar bp = Feff.Sp()/massEff;
    IntegrationScheme<tensor>::integrationResult Wres =
        cloud.WIntegrator().integrate(W_, dt, abp, bp);

    tensor Wnew = Wres.value();
    return Wnew;
}
```

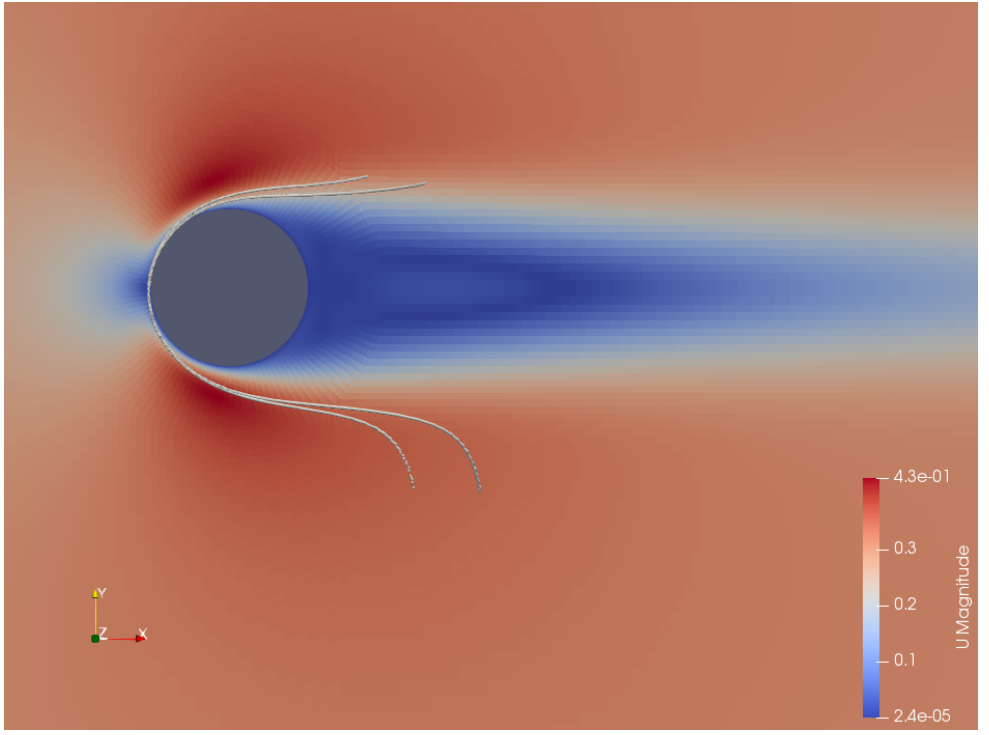
```
// calculate mass transfer of each specie in liquid
    forAll(activeLiquids_, i)
    {
        const label gid = liqToCarrierMap_[i];
        const label lid = liqToLiqMap_[i];

        // vapour diffusivity [m2/s]
        const scalar Dab = liquids_.properties()[lid].D(pc, Ts);

        // saturation pressure for species i [pa]
        // - carrier phase pressure assumed equal to the liquid vapour pressure
        //   close to the surface
        // NOTE: if pSat > pc then particle is superheated
        // calculated evaporation rate will be greater than that of a particle
        // at boiling point, but this is not a boiling model
        const scalar pSat = liquids_.properties()[lid].pv(pc, T);
        // Schmidt number
        const scalar Sc = nu/(Dab + ROOTVSMALL);

        // gas concentration in bulk gas [kmol/m3] at film temperature

        const scalar Xs = pSat/pc;
        // vapour concentration in bulk gas [kmol/m3] at film temperature
        Yinf =  Xc[gid]*this->owner().thermo().carrier().W(gid);

        const scalar Ys = Xs*liquids_.properties()[lid].W()/(Xs*liquids_.properties()[lid].W()+(1-Xs)*Xinf);

        Ys_tot += Ys;

       scalar rhogs = pc/(RR*Ts);

        // Total vaporisation rate
       scalar tot_vap_rate = pi*Dp * Dab * rhogs; //* Sh;
        // mass transfer [kg]

        dMassPC[lid] += tot_vap_rate*dt;
    }
```

Status

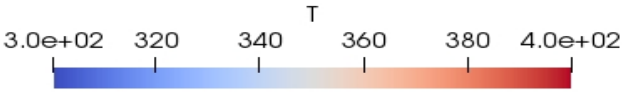
- Still in progress
  - Effect of interpolation should be studied, to validate the whole implementation.
  - Need to simplify handling of energy transfer and liquid properties libraries.
  - Test the evaporation model

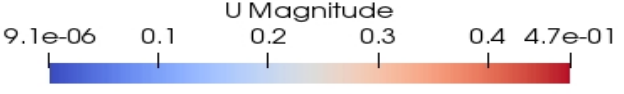
Cylinder case

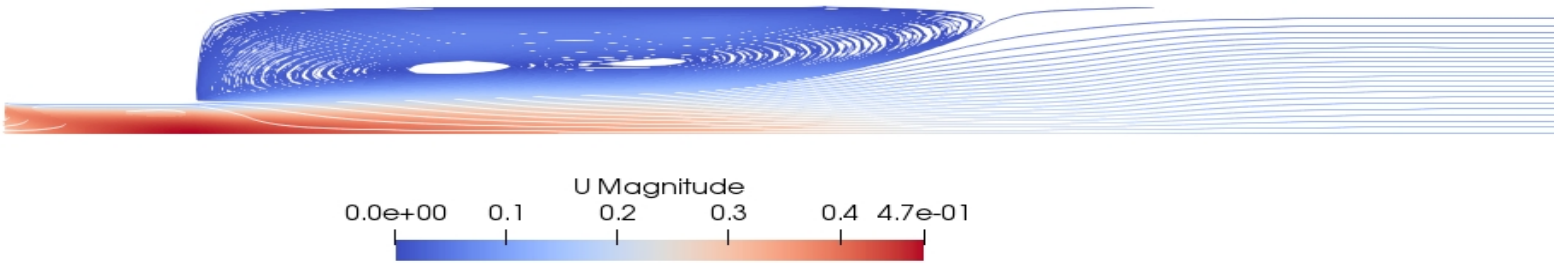




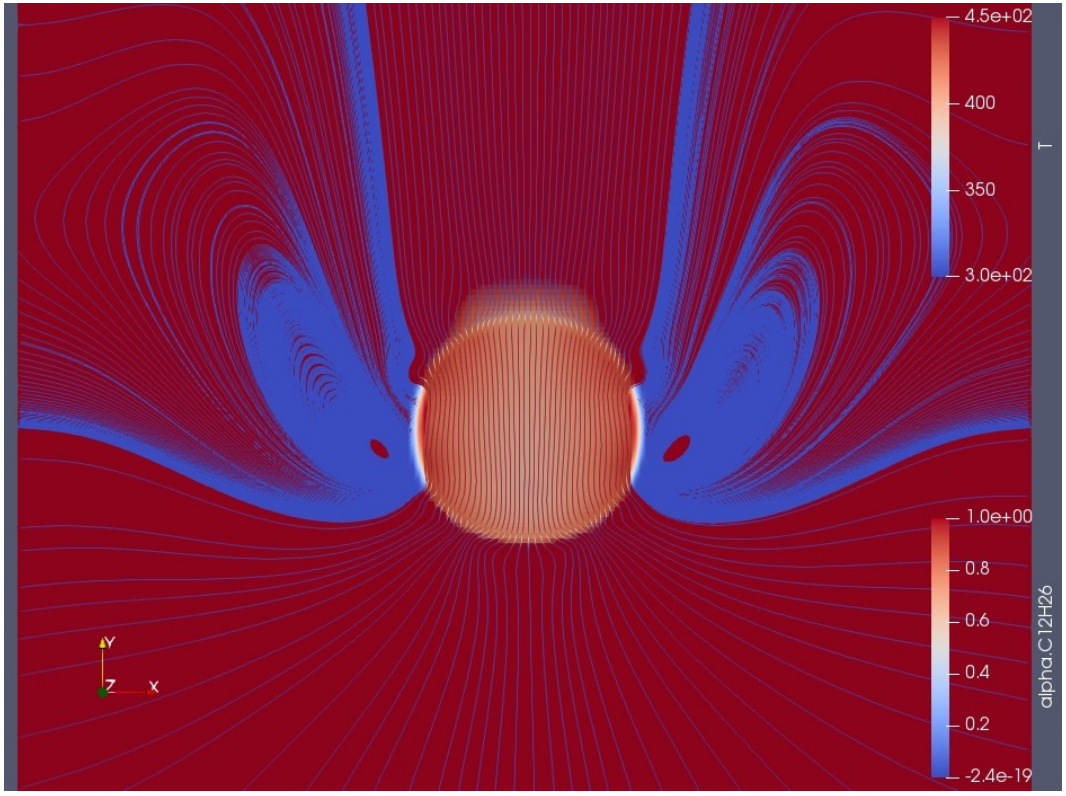
Backstep, work in progress for Evap.

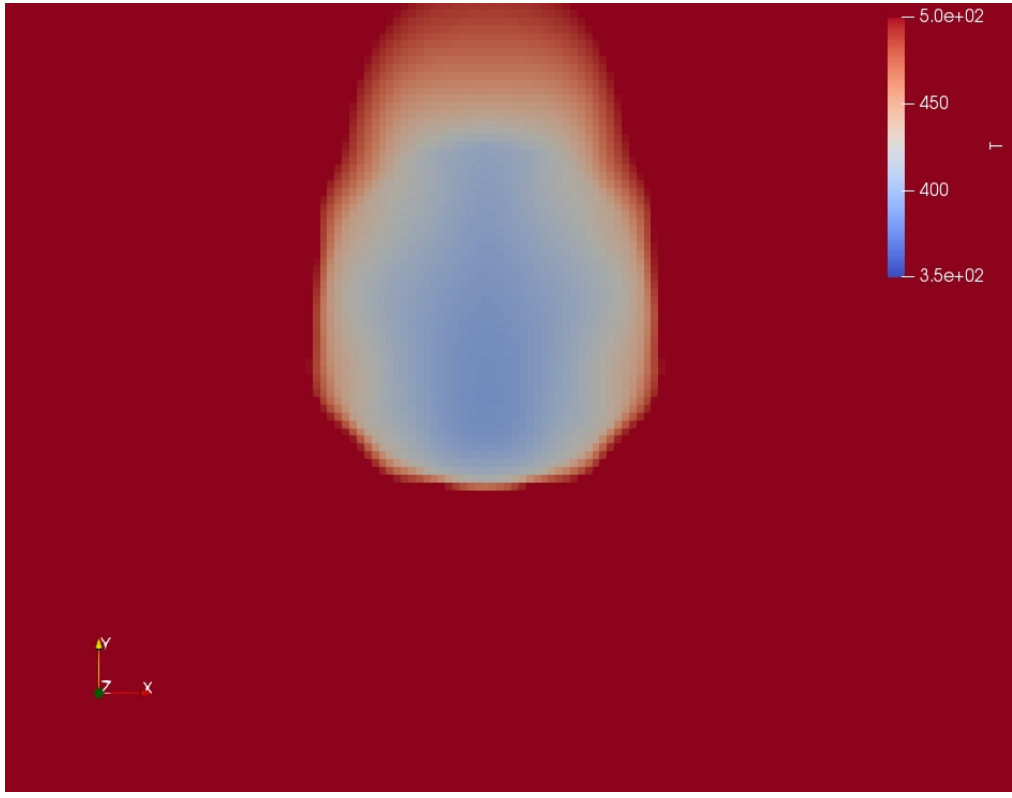





U Magnitude
0.0e+00
0.1
0.2
0.3
0.4
4.7e-01

Single Drop


4.5e+02
400
350
3.0e+02
T
1.0e+00
0.8
0.6
0.4
0.2
-2.4e-19
alpha.C12H26
Y
Z
X


5.0e+02
450
400
3.5e+02
T
Y
Z
X

Conclusion

- It’s all coming along nicely.
- By end of the month, implementation complete.