



The University of
Nottingham

UNITED KINGDOM • CHINA • MALAYSIA



Brunel
University
London



Coupling LAMMPS and OpenFOAM for Multi-Scale Models

E. R. Smith², G. J. Pringle³, G. Gennari⁴, M. Magnini^{1,*}

¹*Dept. of Mechanical Engineering, University of Nottingham, Nottingham. E-mail: mirco.magnini@nottingham.ac.uk*

²*Mechanical and Aerospace Engineering, Brunel University London: edward.smith@brunel.ac.uk*

³*EPCC, University of Edinburgh: g.pringle@epcc.ed.ac.uk*

⁴*Dept. of Mechanical Engineering, University of Nottingham, Nottingham: gabriele.gennari@nottingham.ac.uk*



**eCSE06-01: “Hybrid Atomistic-Continuum Simulations
of Boiling Across Scales”**

CPL Library



CPL LIBRARY

eCSE06-01: “Hybrid Atomistic-Continuum Simulations
of Boiling Across Scales”

Domain Decomposition Coupling

- Finite Volume Solver

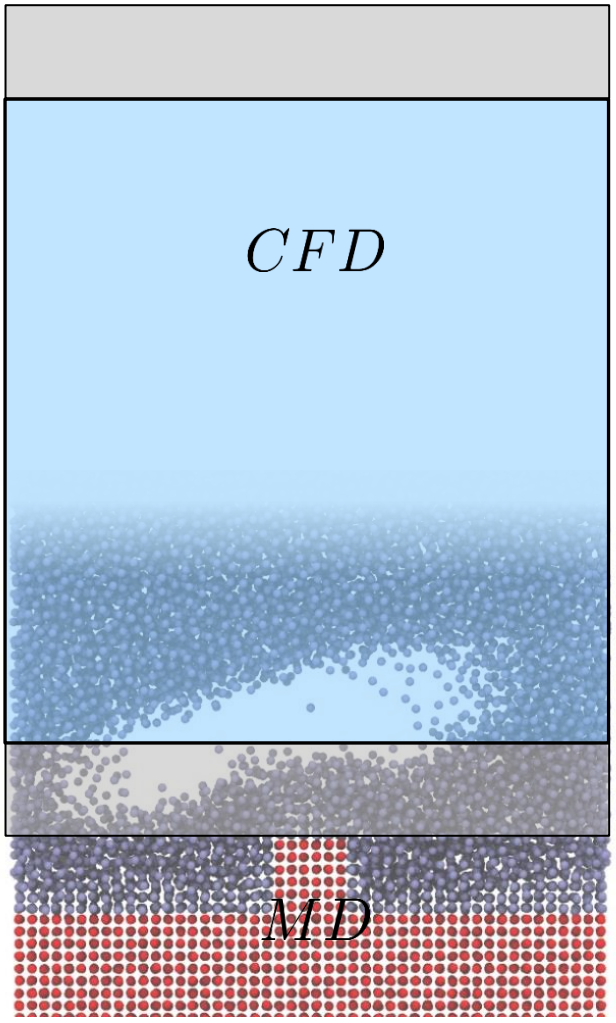
$$\frac{\partial}{\partial t} \int_V \rho u dV = - \oint_S \rho u u \cdot d\mathbf{S} - \oint_S \boldsymbol{\Pi} \cdot d\mathbf{S}$$

Share the same
time and **length**
scales



- Discrete molecules

$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i \text{ for all } i \text{ in } N$$



O'Connell Thompson (1995), Hadjiconstantinou (1998), Flekkoy (2000), Nie et al (2004).

Coupled CFD-MD Simulation

- Finite Volume Solver

$$\frac{\partial}{\partial t} \int_V \rho u dV = - \oint_S \rho u u \cdot dS - \oint_S \boldsymbol{\Pi} \cdot dS$$

$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i + \mathbf{F}_i^C \quad i \in \text{cell}$$

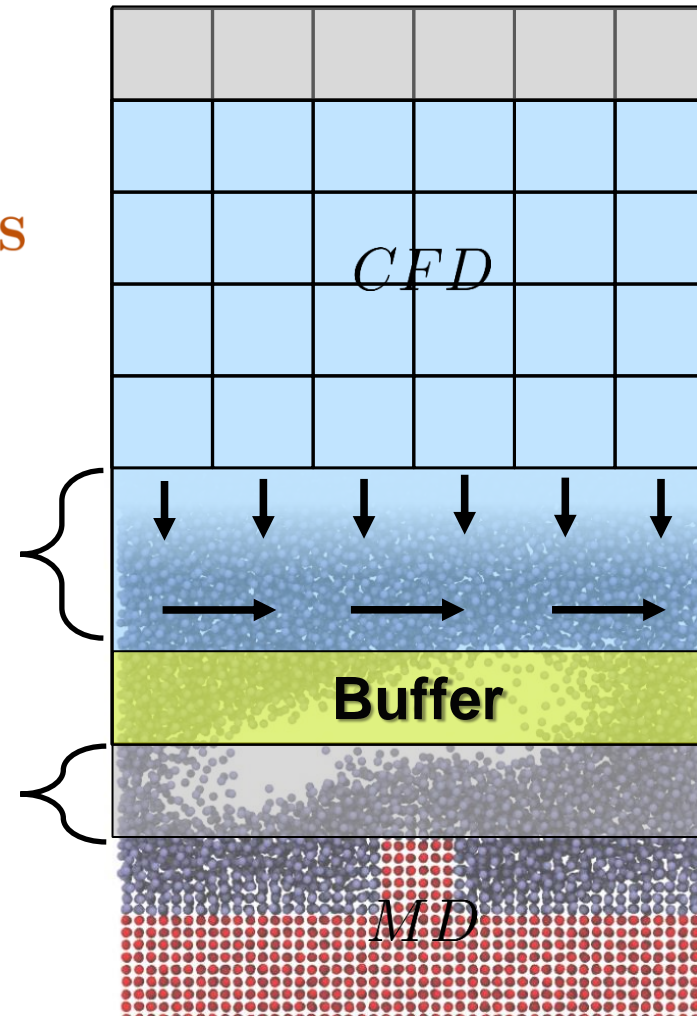
**CFD → MD
Boundary
condition**

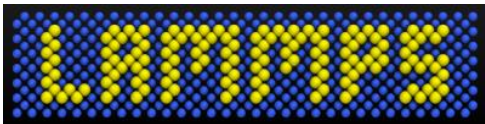
$$\int_V \rho u dV = \sum_{i=1}^N m_i \mathbf{v}_i \vartheta_i$$

**MD → CFD
Boundary
condition**

- Discrete molecules

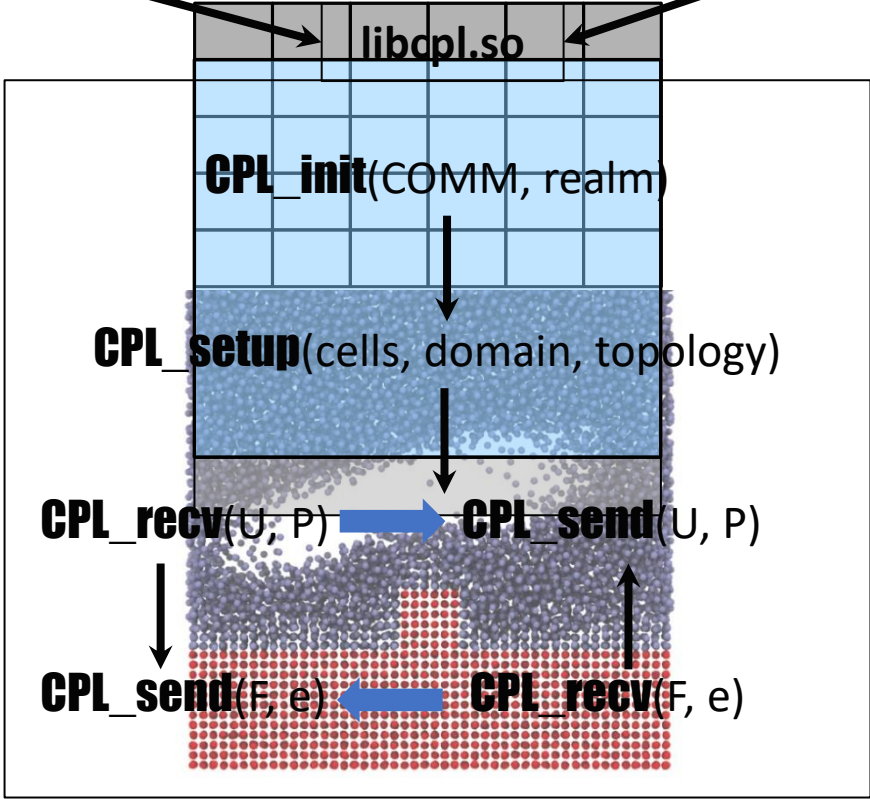
$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i \quad \text{for all } i \text{ in } N$$





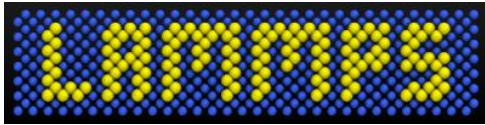
CPL LIBRARY

www.cpl-library.org



Granular Mechanics

- A special case of domain decomposition where they totally overlap

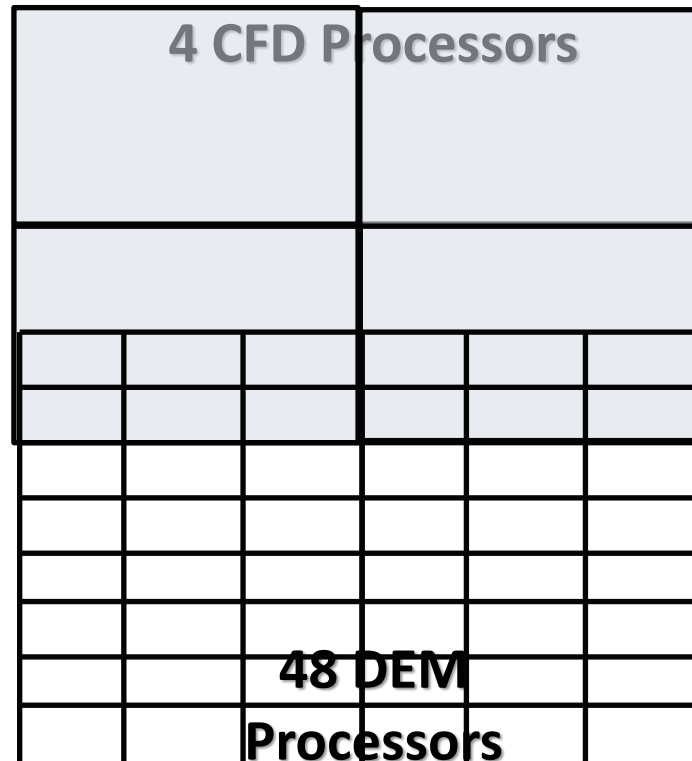


- Apply drag force based on continuum values to particles

$$F_i^c = C_{di}(U_{CFD} - u_i)$$

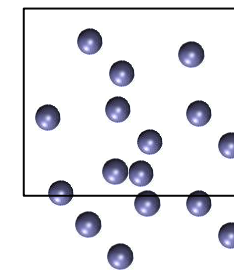
- Average values on a grid

$$u_a = \sum v_i \quad C_d = \sum C_{di}$$



- A Porous form of the Navier-Stokes Equations

$$\frac{\partial \rho \epsilon \mathbf{u}}{\partial t} + \nabla \cdot (\rho \epsilon \mathbf{u}) = -\epsilon \nabla P + \nabla \cdot (\epsilon \boldsymbol{\tau}) + \epsilon \rho g - \mathbf{F}^C$$



Assumes
Cell \gg particle

Porosity ϵ

MPI – the Inspiration for CPL Library

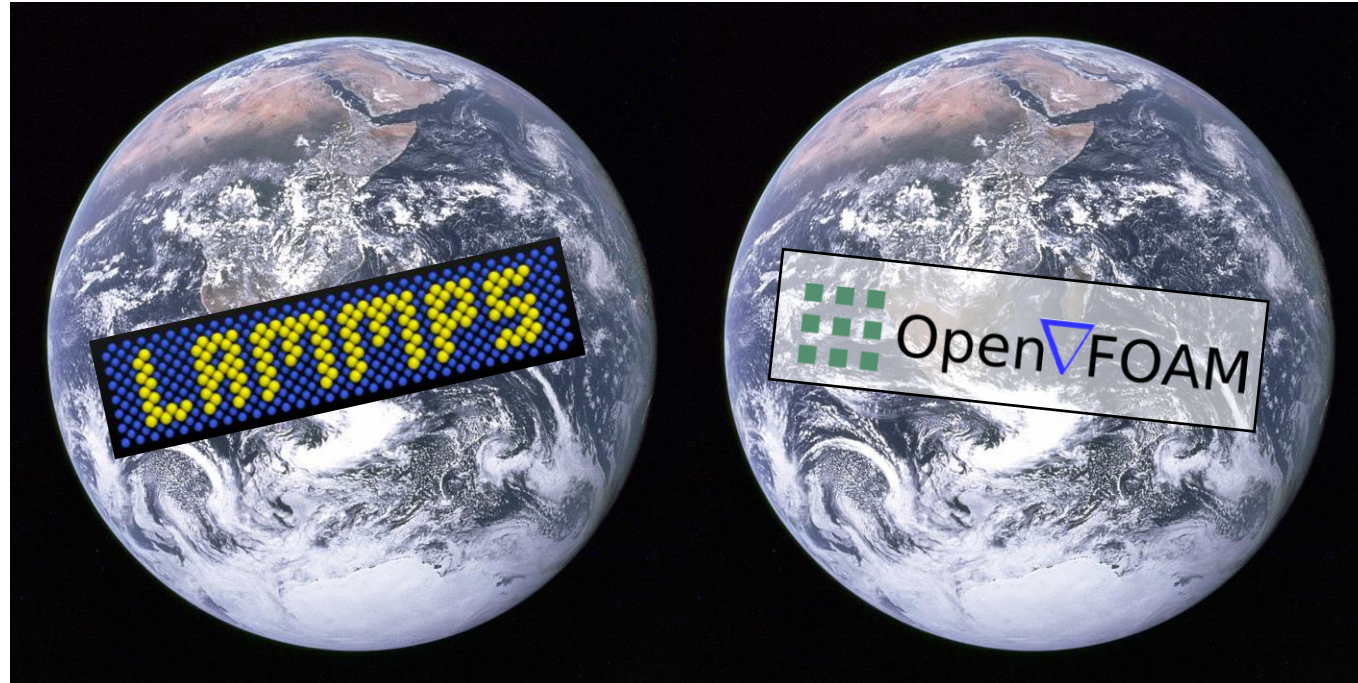
- Message Passing Interface (MPI) is a standardisation of communication for HPC
- The commands `MPI_send` and `MPI_recv` are used to exchange information between processes
- The processor topology can be setup using `MPI_Cart_create`
 - Assuming you have 128 cores available you might want 8 x 4 x 4 which is,

```
call MPI_Cart_create(COMM, ndims, [8, 4, 4], periods[],  
                    reorder, Ouput_cart_comm);
```
 - Allows scheduler/compiler to reorder to optimise topology based on node/core proximity in supercomputer (reorder flag)
 - Adjacent processes can be obtained with commands like `CART_SHIFT`

- Communicators are used to determine which processes communicate, e.g.

```
MPI_send(data, size, MPI_COMM)
```

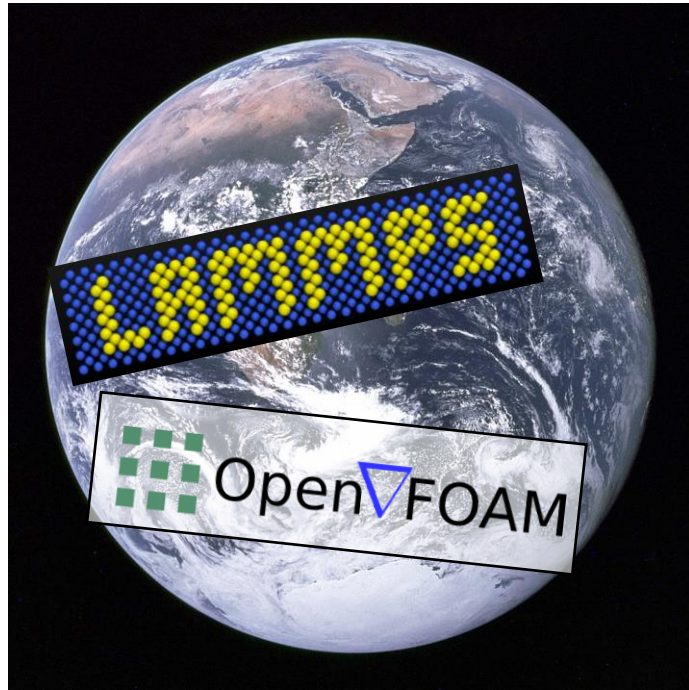
- The default that contains all communicators is MPI_COMM_WORLD
- Most codes (inc. LAMMPS/OpenFOAM) use this as they assume they are the only code in the world



CPL library employs a newer MPI feature to solve this

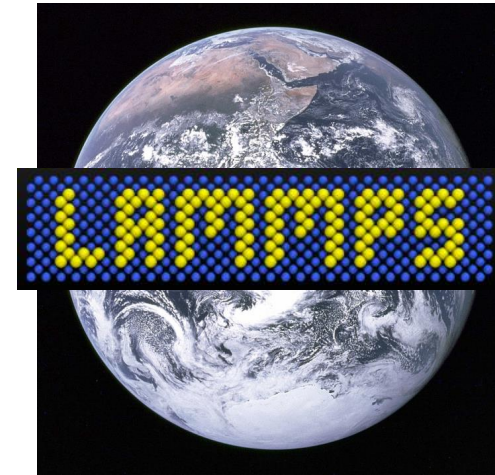
➤ Shared (same MPI_COMM_WORLD)

```
mpiexec -n 1 ./md : -n ./cfd
```



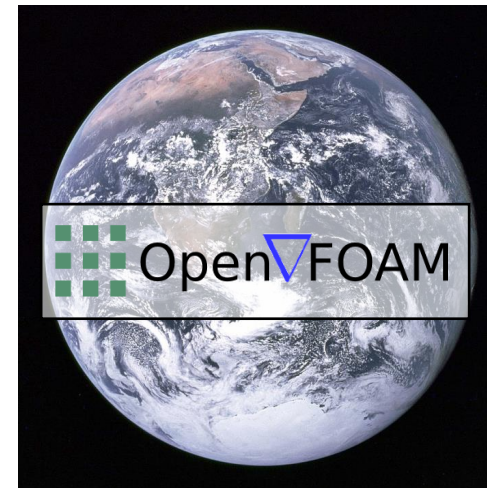
- Both codes must be patched to replace MPI_COMM_WORLD with CPL_COMM

➤ Distinct (Own MPI_COMM_WORLDs)



```
mpiexec -n 1 ./md
```

- Codes couple using MPI_port, no need to patch them as share COMM_UNIVERSE



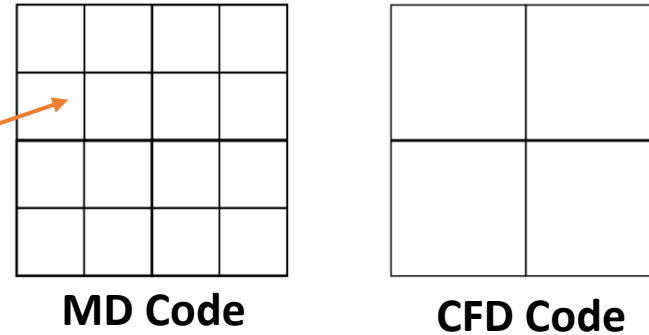
```
mpiexec -n 1 ./cfd
```

- ARCHER2 now works with MPI_port (since recent rebuild!)

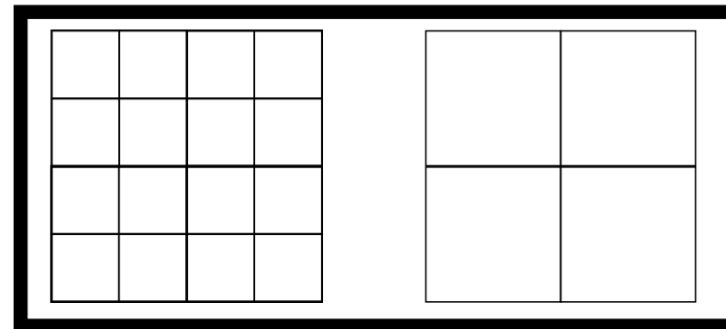
CPL Library - Linking Cartesian Grids Between 2 Codes

Squares are processors
(CFD computational grid not shown)

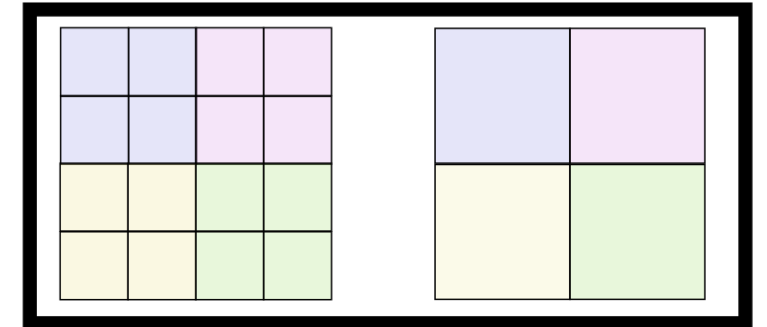
a) MPI_Init
MPI_Cart_Create



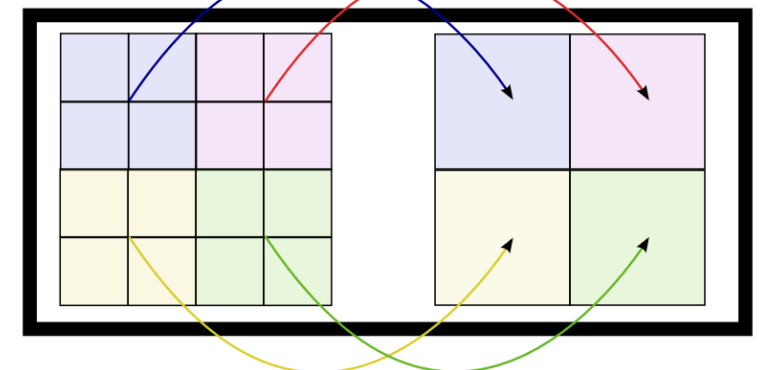
b) CPL_INIT()



c) CPL_SETUP(...)

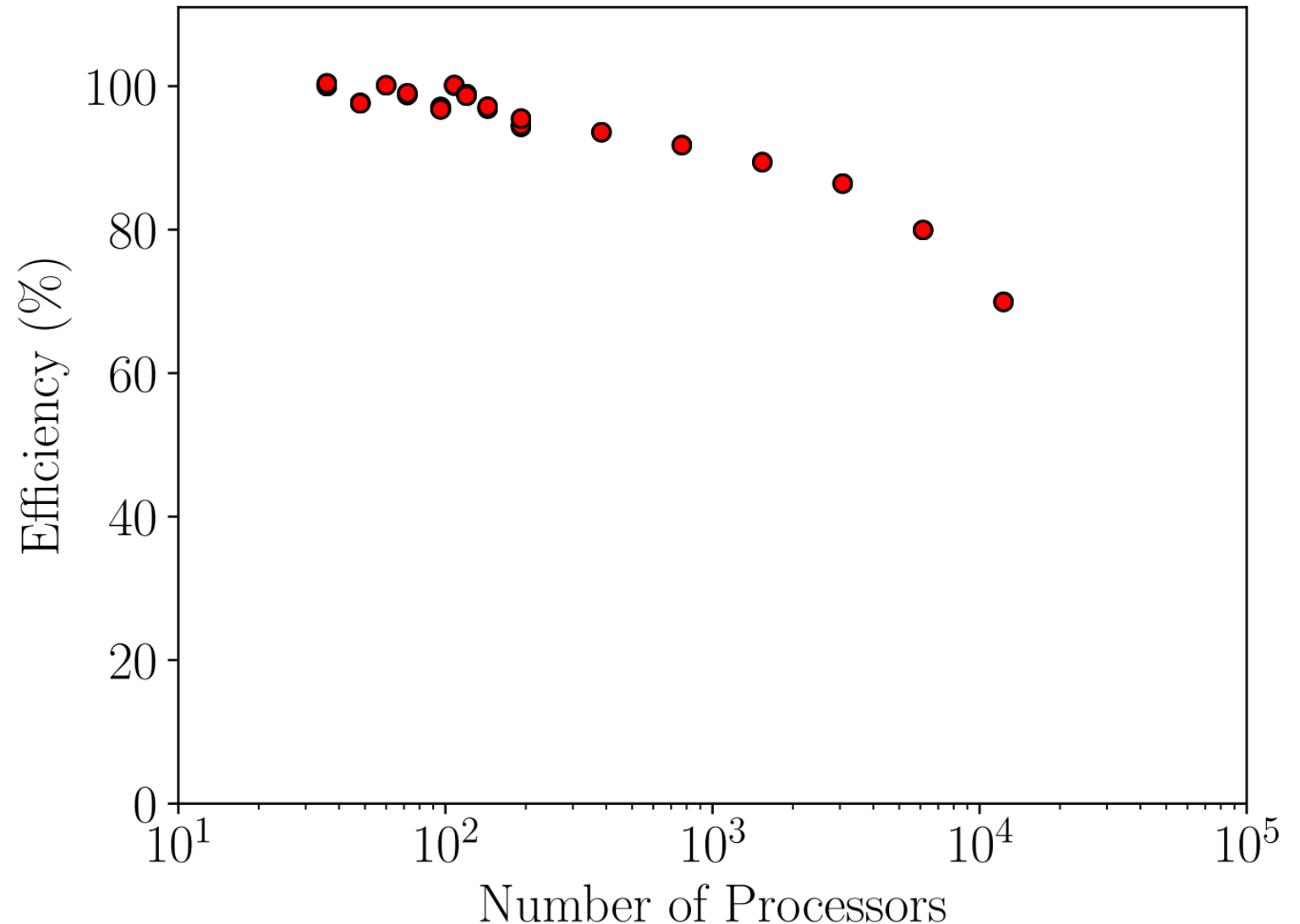


d) CPL_SEND(A)
CPL_RECV(A)



Weak Scaling of CPL-Library on ARCHER

- Scaling improved by
 - Using MPI CART CREATE
 - Ensuring all coupled communication is processor to processor
- Test here up to 10,000 cores
- Scaling example
 - Two dummy codes which do minimal work
 - Communication is a large fraction



Building CPL library

- Building creates a library `libcpl.so` in `lib` folder of `cpl-library`

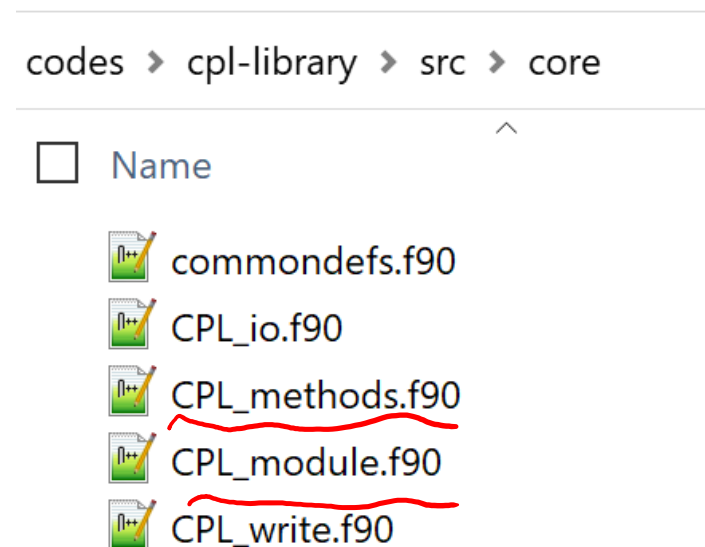
```
make PLATFORM=gcc
```






```
OR make PLATFORM=ARCHER2
```

- CPL library itself is very minimal
 - A core fortran source code with two main files
 - Bindings for C++ and Python
 - A set of utilities to help coupled simulations
- Prerequisite include a fortran compiler, a C++ compiler, Python and MPI
 - Python libraries like `numpy`, `scipy`, `matplotlib` and `pytest` are useful to run tests and `wxpython` allows some user interfaces
 - Has been built in a module on ARCHER2, please see:

codes > cpl-library > src > core

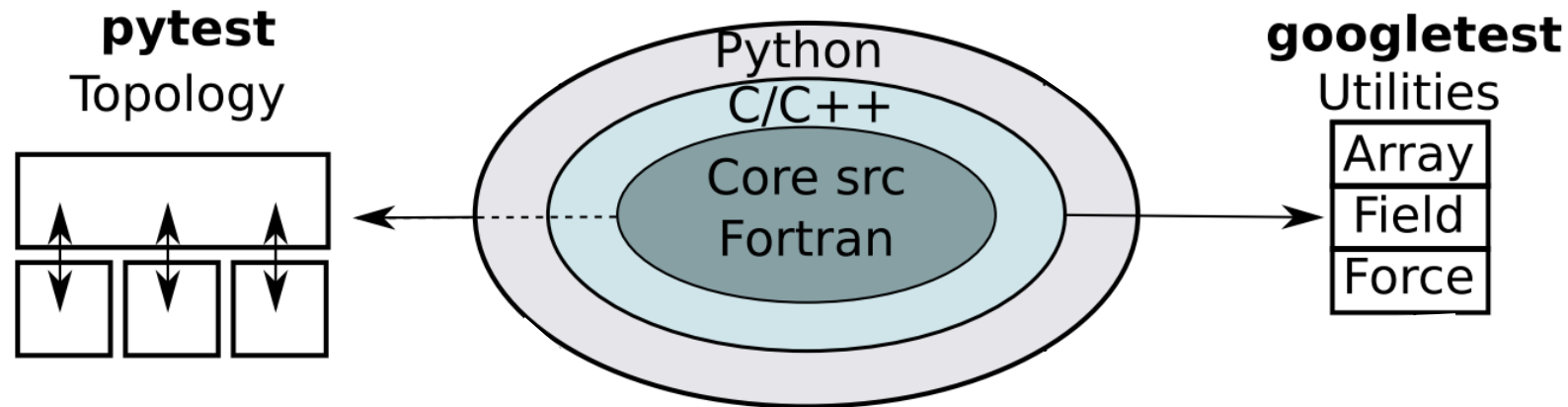
Name ^



 commondefs.f90
 CPL_io.f90
 CPL_methods.f90
 CPL_module.f90
 CPL_write.f90

Building CPL library

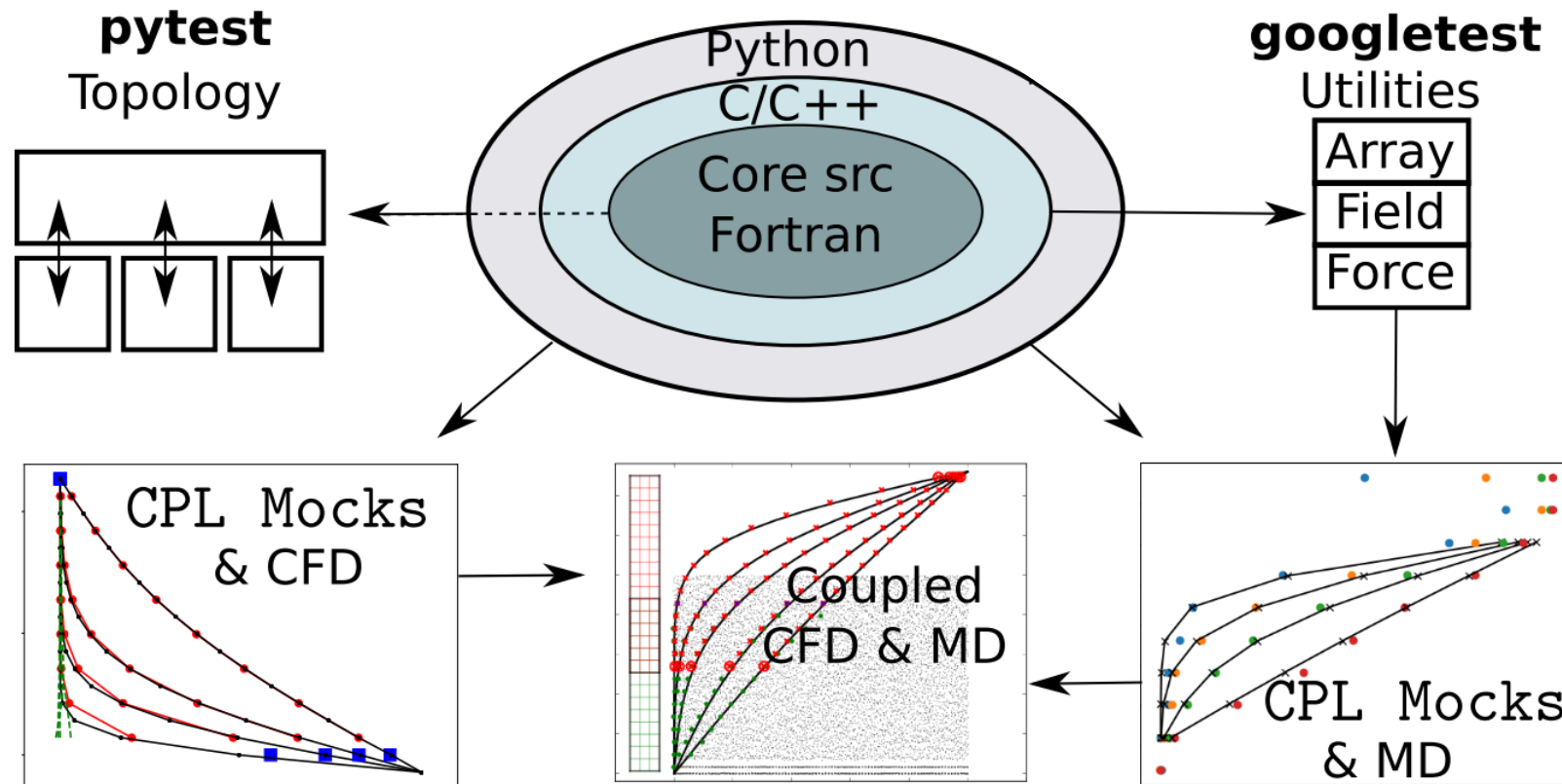
- CPL library itself is very minimal
 - A core fortran source code with two main files
 - Bindings for C++ and Python



- A set of utilities to help coupled simulations
- Wider utilities include interchangeable Fortran, C++ and Numpy arrays, objects to get field information (e.g. average MD) and apply MD forces
- Core exchange functionality tested using pytest for various topologies

Building CPL library

- Division of concern – CPL library can be tested by itself, codes kept isolated
- CFD code tested by coupling to a minimal script (software concept of mocking)
- MD code tested separately by coupling to a minimal script (mock CFD)

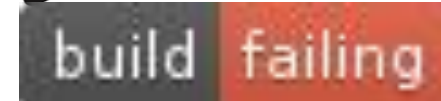
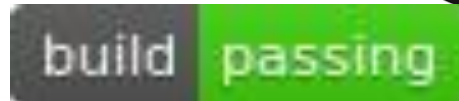


- Testing includes
 - Basic units test on low level code
 - Coupled runs on all permutations of coupled topologies
 - Coupling codes to Mocks to ensure expected behaviour
 - Complete physical examples compare to known analytical solutions
- Uses GitHub Actions to automate tests after any changes in the code

```
TEST_F(CPL_Force_Test, test_CPL_array_size) {  
    int nd = 9;  
    int icell = 3;  
    int jcell = 3;  
    int kcell = 3;  
    CPL::ndArray<double> buf;  
    int shape[4] = {nd, icell, jcell, kcell};  
    buf.resize (4, shape);  
  
    //Test sizes and shapes  
    ASSERT_EQ(buf.size(), nd*icell*jcell*kcell);  
    ASSERT_EQ(buf.shape(0), nd);  
    ASSERT_EQ(buf.shape(1), icell);  
    ASSERT_EQ(buf.shape(2), jcell);  
    ASSERT_EQ(buf.shape(3), kcell);  
};
```



Software
Sustainability
Institute



Coupling Mocks – Python CFD Mock

```
from mpi4py import MPI
from cplpy import CPL

comm = MPI.COMM_WORLD
CPL = CPL()
CFD_COMM = CPL.init(CPL.CFD_REALM)
CPL.setup_cfd(CFD_COMM.Create_cart([1, 1, 1]), xyzL=[1.0, 1.0, 1.0],
              xyz_orig=[0.0, 0.0, 0.0], ncxyz=[32, 32, 32])

recv_array, send_array = CPL.get_arrays(recv_size=4, send_size=1)

for time in range(5):
    recv_array, ierr = CPL.recv(recv_array)
    send_array[0, :, :, :] = 2.*time
    CPL.send(send_array)

CPL.finalize()
MPI.Finalize()
```

Coupling Mocks – Python CFD Mock

```
from mpi4py import MPI
from cplpy import CPL
```

```
comm = MPI.COMM_WORLD
CPL = CPL()
CFD_COMM = CPL.init(CPL.CFD_REALM)
CPL.setup_cfd(CFD_COMM.Create_cart([1, 1, 1]), xyzL=[1.0, 1.0, 1.0],
              xyz_orig=[0.0, 0.0, 0.0], ncxyz=[32, 32, 32])
```

```
recv_array, send_array = CPL.get_arrays(recv_size=4, send_size=1)
```

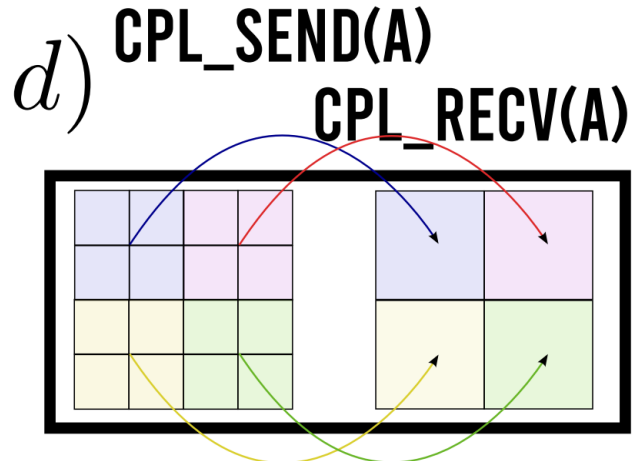
```
for time in range(5):
    recv_array, ierr = CPL.recv(recv_array)
    send_array[0, :, :, :] = 2.*time
    CPL.send(send_array)
```

```
CPL.finalize()
MPI.Finalize()
```

CFD Processors here to create MPI Cartesian Grid of processors

CFD Cells Specified here

- All mapping done by CPL_setup
- Send and recv then exchange data to the right place



Coupling Mocks – Python to C++

```
from mpi4py import MPI
from cplpy import CPL

comm = MPI.COMM_WORLD
CPL = CPL()
CFD_COMM = CPL.init(CPL.CFD_REALM)
CPL.setup_cfd(CFD_COMM.Create_cart([1, 1, 1]),
              xyzL=[1.0, 1.0, 1.0],
              xyz_orig=[0.0, 0.0, 0.0],
              ncxyz=[32, 32, 32])
recv_array, send_array =
CPL.get_arrays(recv_size=4, send_size=1)

for time in range(5):
    recv_array, ierr = CPL.recv(recv_array)
    send_array[0, :, :, :] = 2.*time
    CPL.send(send_array)

CPL.finalize()
MPI.Finalize()
```

CFD
Processors
here

CFD
Grid
here

```
#include "mpi.h"
#include "cpl.h"

...
int main() {

    MPI_Comm MD_COMM, CART_COMM;
    CPL::ndArray<double> send_array, recv_array;
    MPI_Init(NULL, NULL);
    CPL::init(CPL::MD_realm, MD_COMM);
    int npxyz[3] = {1, 1, 1}; ...
    MPI_Cart_create(MD_COMM, 3, npxyz,
                   periods, 1, &CART_COMM);
    double xyzL[3] = {1.0, 1.0, 1.0}; ...
    CPL::setup_md(CART_COMM, xyzL, xyz_orig);
    CPL::get_arrays(&recv_array, 1, &send_array, 4);

    for (int time = 0; time < 5; time++) {
        send_array(0,0,0,0) = 5.*time;
        bool flag = CPL::send(&send_array);
        bool flag = CPL::recv(&recv_array);
    }

    CPL::finalize();
    MPI_Finalize();
}
```

MD
Processors
here

No MD
grid
needed

Coupling Mocks – Fortran to C++

```
program main_CFD
  use cpl
  use mpi
  implicit none

  integer :: time, CFD_COMM, CART_COMM, ierr, CFD_realm=1
  double precision, dimension(:, :, :, :), &
    allocatable :: send_array, recv_array
  call MPI_Init(ierr)
  call CPL_init(CFD_realm, CFD_COMM, ierr)
  call MPI_Cart_create(CFD_COMM, 3, (/1, 1, 1/), &
    (/true., .true., .true./), &
    .true., CART_COMM, ierr)
  call CPL_setup_cfd(CART_COMM, (/1.d0, 1.d0, 1.d0/), &
    (/0.d0, 0.d0, 0.d0/), &
    (/32, 32, 32/))
  call CPL_get_arrays(recv_array, 4, send_array, 1)

  do time=1,5
    call CPL_recv(recv_array)
    send_array(1, :, :, :) = 2.*time
    call CPL_send(send_array)
  enddo

  call CPL_finalize(ierr)
  call MPI_finalize(ierr)

end program main_CFD
```

CFD
Processors
here

CFD
Grid
here

```
#include "mpi.h"
#include "cpl.h"

...
int main() {

  MPI_Comm MD_COMM, CART_COMM;
  CPL::ndArray<double> send_array, recv_array;
  MPI_Init(NULL, NULL);
  CPL::init(CPL::MD_realm, MD_COMM);
  int npxyz[3] = {1, 1, 1}; ...
  MPI_Cart_create(MD_COMM, 3, npxyz,
    periods, 1, &CART_COMM);
  double xyzL[3] = {1.0, 1.0, 1.0}; ...
  CPL::setup_md(CART_COMM, xyzL, xyz_orig);
  CPL::get_arrays(&recv_array, 1, &send_array, 4);

  for (int time = 0; time < 5; time++) {
    send_array(0,0,0,0) = 5.*time;
    bool flag = CPL::send(&send_array);
    bool flag = CPL::recv(&recv_array);

    CPL::finalize();
    MPI_Finalize();
  }
}
```

MD
Processors
here

No MD
grid
needed

Running CPL library

- Before running anything, you should add `libcpl` to your path using the following command:

```
source SOURCEME.sh
```

- Most common errors will be because you have not done this, e.g. “ImportError: No module named cplpy” or “error: cpl.h: No such file or directory”

- As it is a library, you don't run directly, must be linked into an executable, e.g.

```
cplf90 minimal_md.f90 -o ./md (or explicitly link -lcpl)
```

- which is run in shared or distinct mode

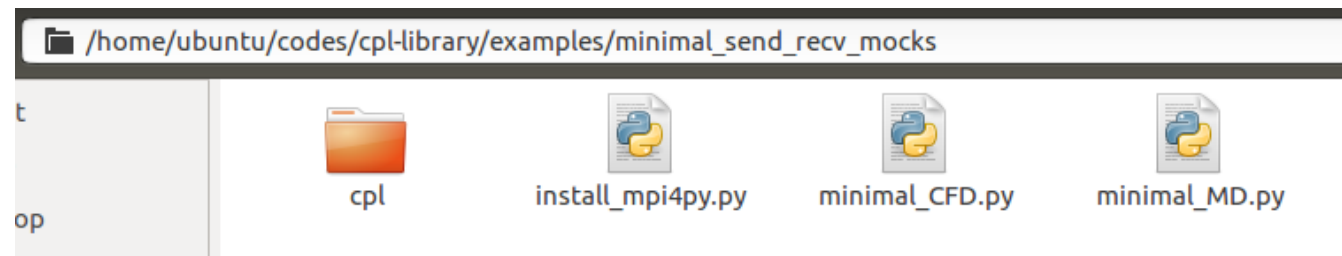
```
mpiexec -n 1 ./cfd : -n 1 ./md (shared)
```

```
mpiexec -n 1 ./cfd < different terminals > mpiexec -n 1 ./md
```

```
cplexec -c 1 ./cfd -m 1 ./md (cplexec Python wrapper)
```


Coupler Input file COUPLER.in

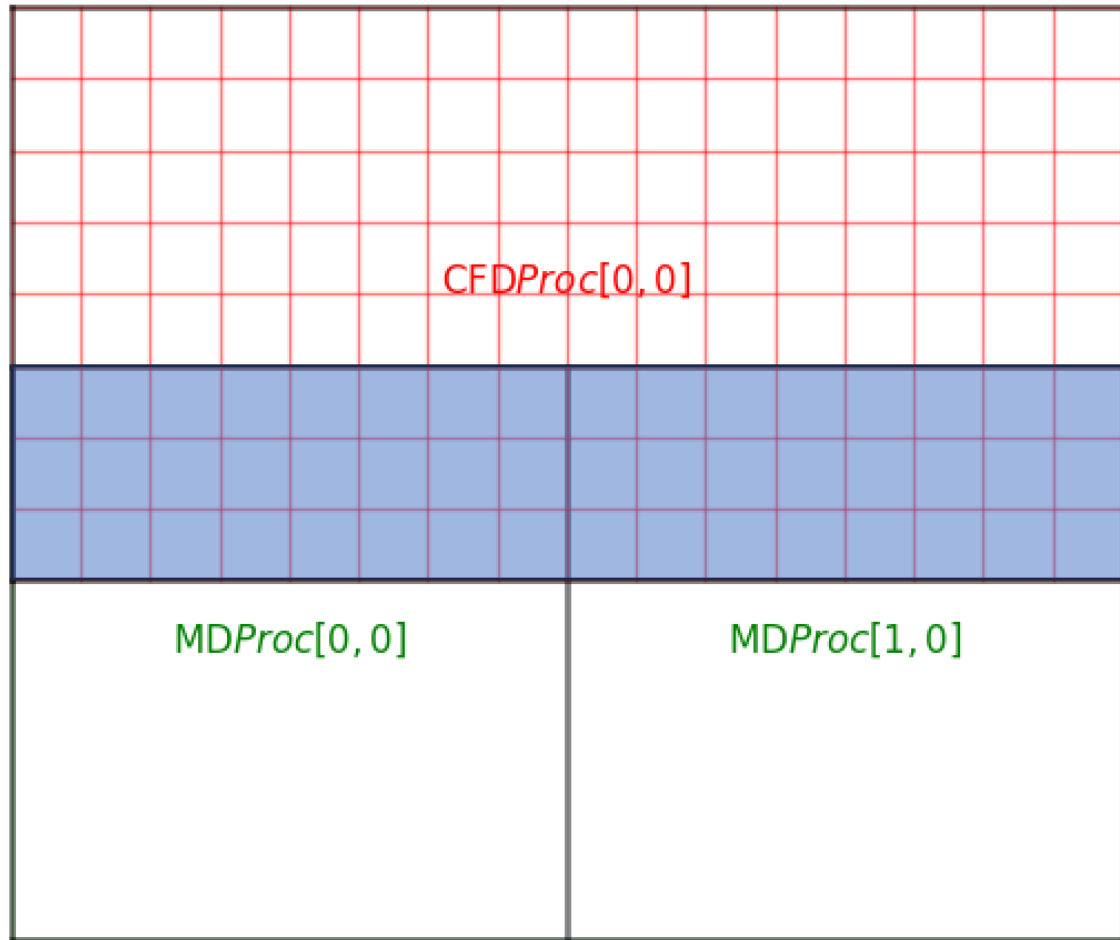
- The coupler setup (any inputs to coupled case which are not naturally specified in either code) are included in `cpl/COUPLER.in`
- `COUPLER.in` inside the `cpl` folder must exist to run a coupled case



- Some three important ones are:
 - 'FULL_OVERLAP' -- Specifies if overlap extents is all CFD cells (which is the case for granular coupling)
 - 'TIMESTEP_RATIO' -- ratio of timesteps in both MD/CFD codes
 - 'OVERLAP_EXTENTS' – The number of cells which overlap
 - 'BOUNDARY_EXTENTS; - The region of the overlap where boundary values are taken
 - 'CONSTRAINT_INFO' – The constraint to apply and region where it is applied

Changing Overlap Size

- Different COUPLER.in values set coupling overlap



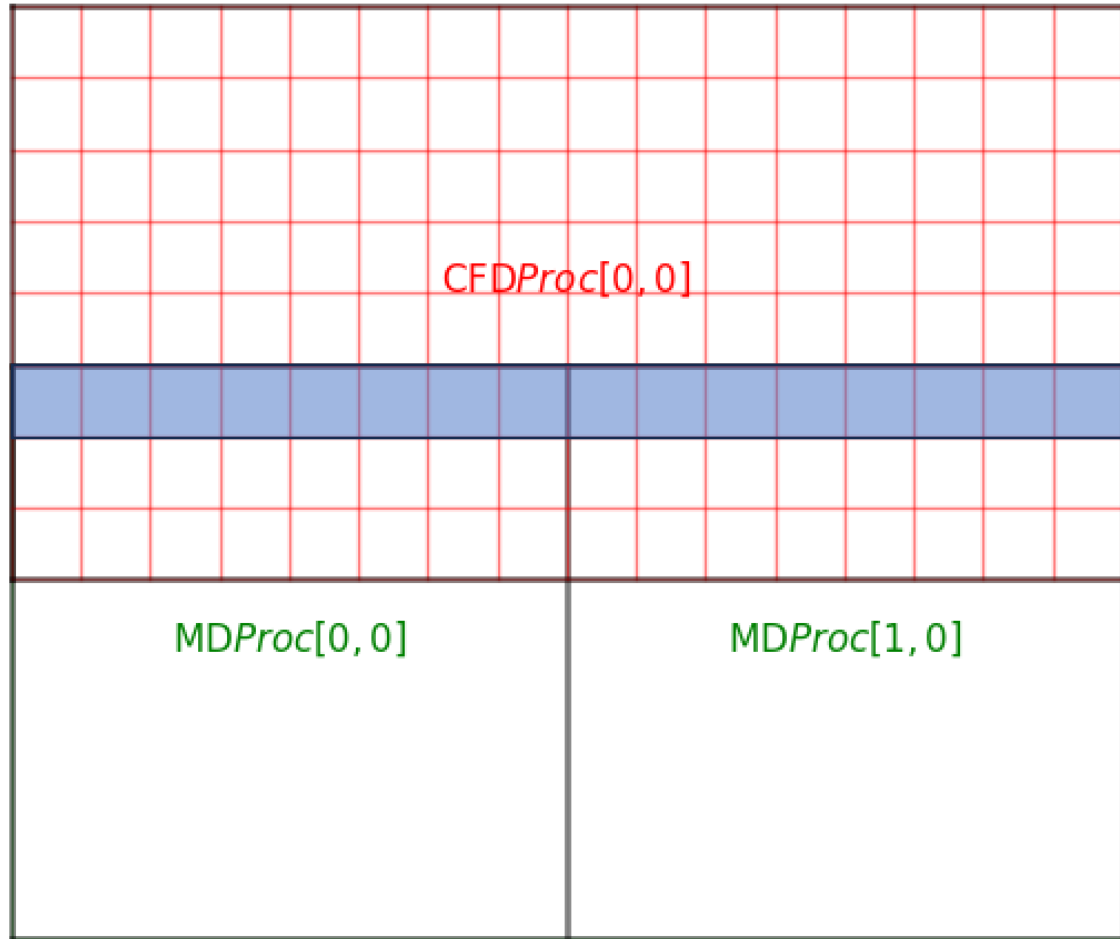
Specified as fraction of CFD grid

```
OVERLAP_EXTENTS
1           Start in x
16          End in x
1        Start in y
3        End in y
1           Start in z
16          End in z
```

Always use full overlap in x and z, only change y. Number of cells defines overlap (not origin or domain size)

Changing Location of Constrained Region

- Different COUPLER.in values set region of constraint applied in the overlap region (as well as type of applied force type)

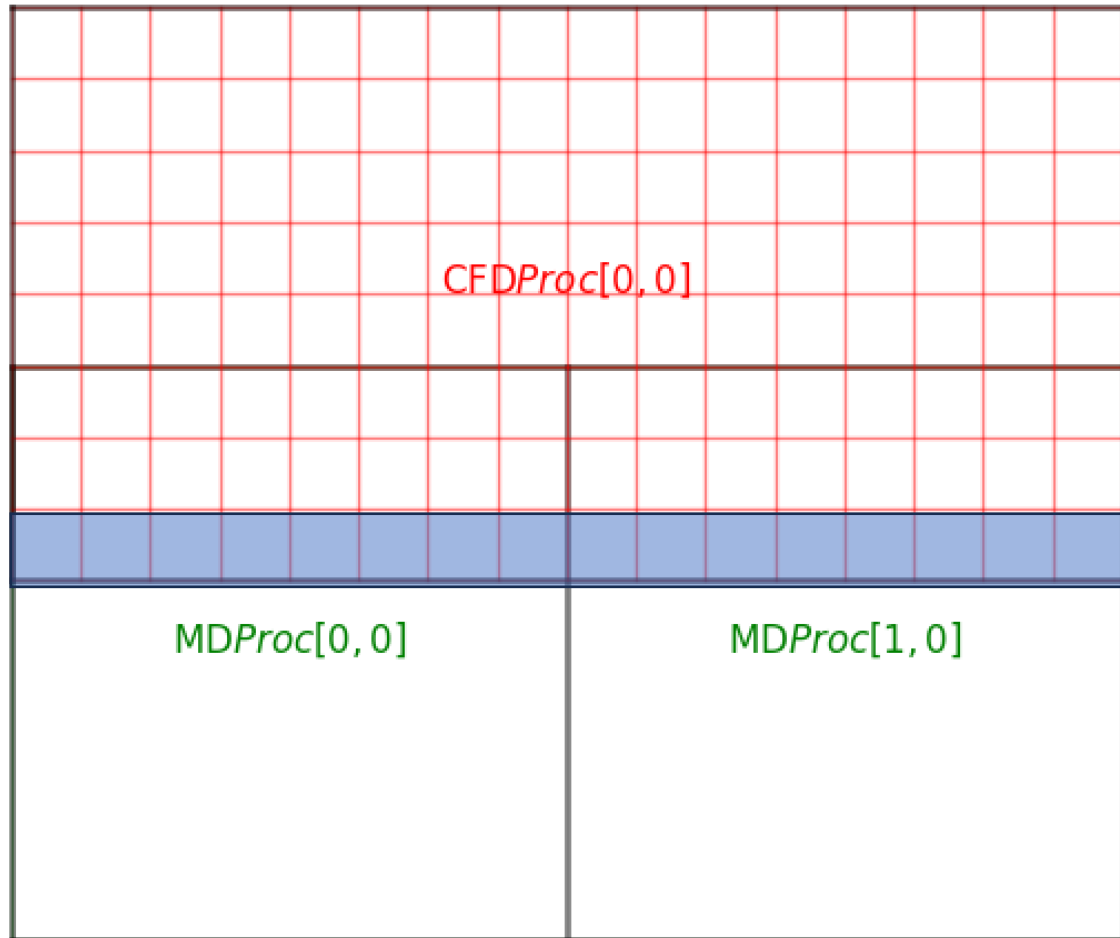


```
CONSTRAINT_INFO
2           Use Velocity Constraint
0           Flag for constraint control
1           Start in x
16          End in x
3           Start in y
3           End in y
1           Start in z
16          End in z
```

Constraint must be inside overlap

Changing Region Averaged to give CFD boundary

- Different COUPLER.in values set region to average for CFD boundary condition

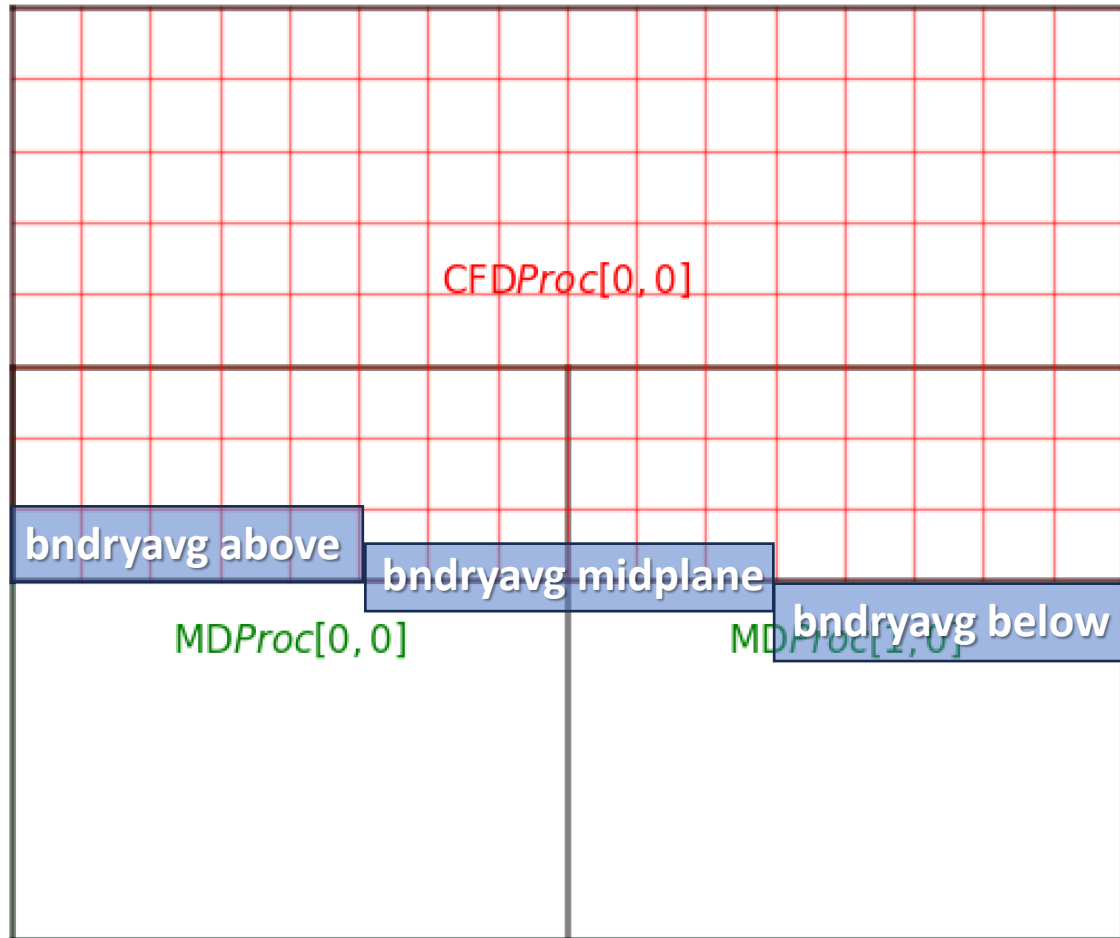


Boundary must be inside overlap

```
BOUNDARY_EXTENTS
1          Start in x
16         End in x
1        Start in y
1        End in y
1          Start in z
16         End in z
```

Changing Region Averaged to give CFD boundary

- MD code must also choose what is averaged to give this boundary condition



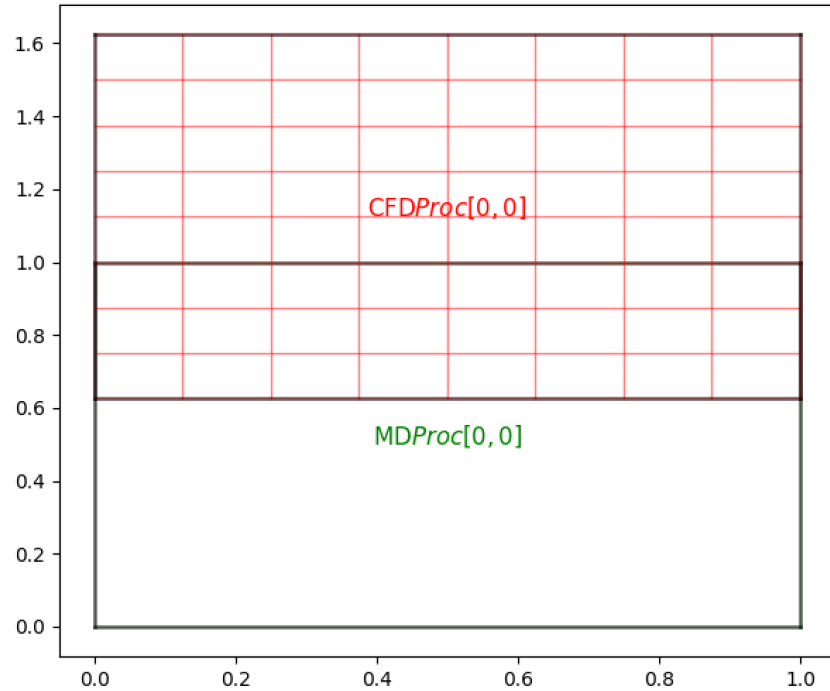
- In MD code, we choose convention
 - **below** CFD code might need halo cells which are cell below overlap domain
 - **midplane** surface fluxes use values at interface (midplane) between overlap and outside
 - **above** Bottom of overlap region is average and passed to CFD

This is still passed as data in the bottom cell of the overlap region.

Design and Test Different Topologies / Communication

- From `cpl-library/utils/design_topology/` a gui (needing wxPython) run with `python cpl_gridsetup.py`

cpl CPL Setup



CFD

Cells	8	8	8
Procs	1	1	1
Origin	0.00	0.00	0.00
Domain	1.00	1.00	1.00

MD

Procs	1	1	1
Origin	0.00	0.00	0.00
Domain	1.00	1.00	1.00

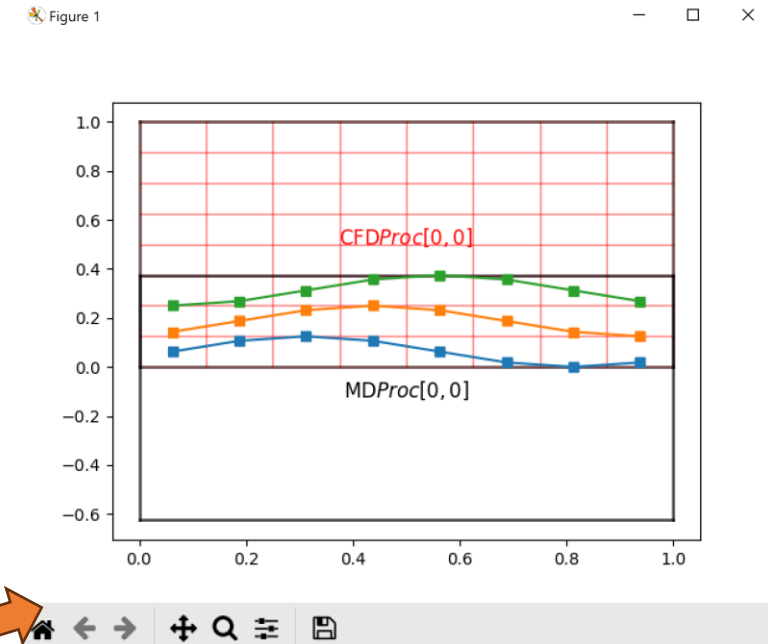
CPL

Mincell	1	1	1
Maxcell	8	3	8

Update SaveFig

Grid Style **RunCase**

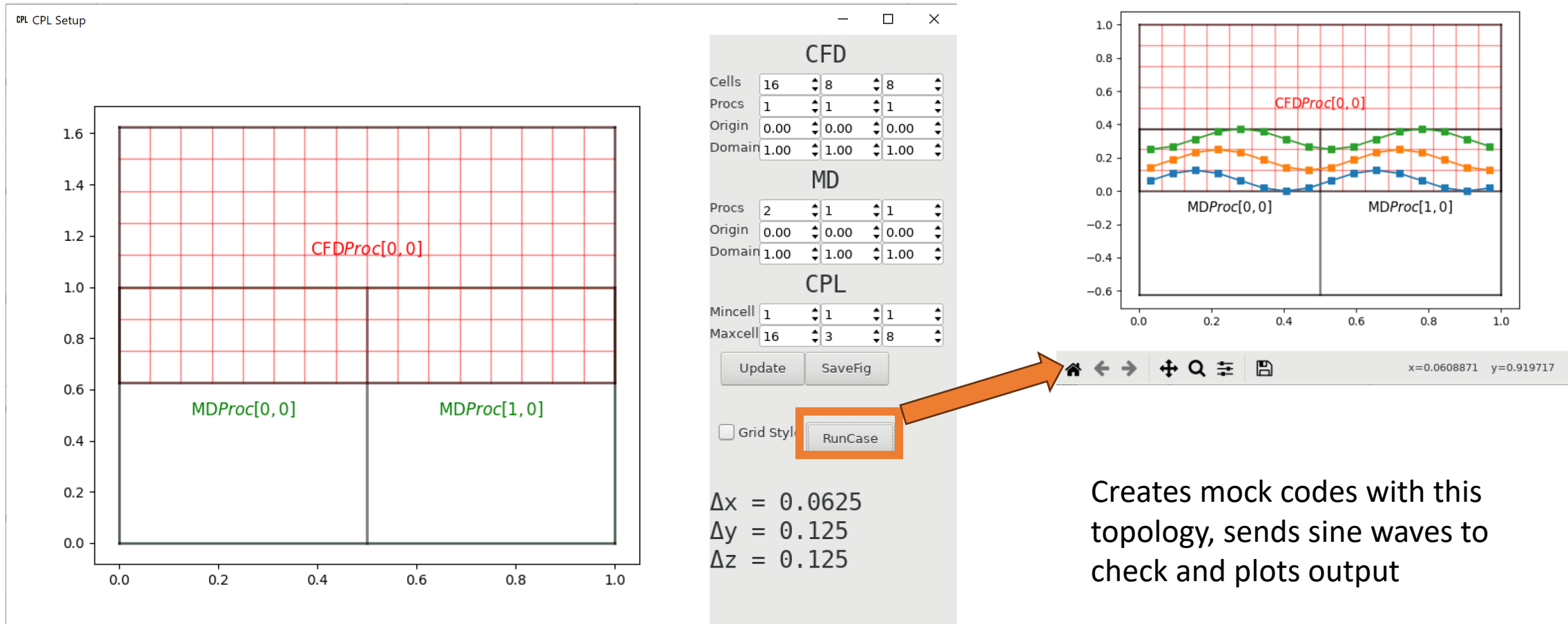
$\Delta x = 0.125$
 $\Delta y = 0.125$
 $\Delta z = 0.125$



Creates mock codes with this topology, sends sine waves to check and plots output

Design and Test Different Topologies / Communication

- Doubling Resolution in CFD and number of processors in CFD domain results in the following output



Creates mock codes with this topology, sends sine waves to check and plots output

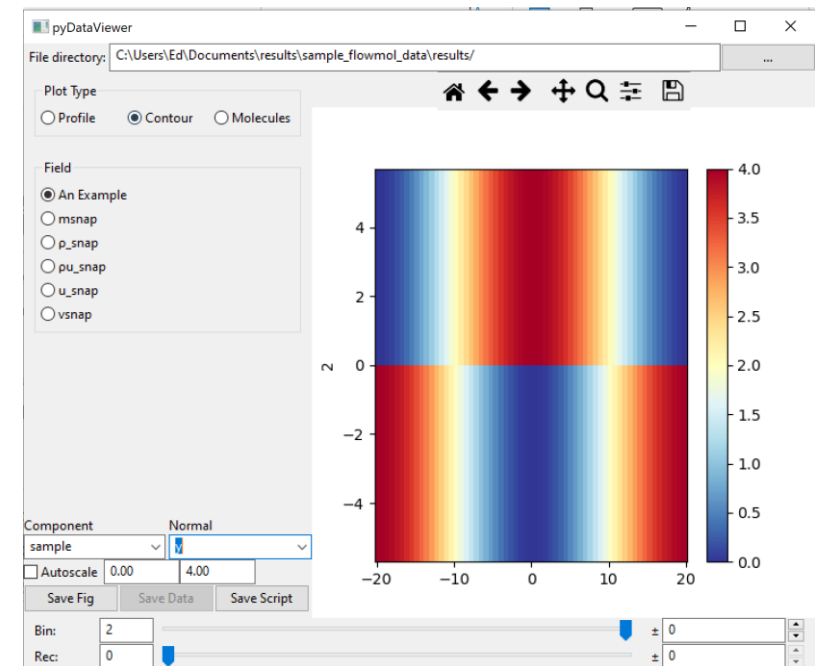
Other Software

➤ PyDataView

- A way of viewing OpenFOAM, LAMMPS and coupled runs
- `python3 pyDataView -d ./path/to/openfoam/output` but needs wxPython, matplotlib, numpy, scipy and vispy (if you want to view molecules)
- Cloned from <https://github.com/edwardsmith999/pyDataView>

➤ SimWrapLib

- Used to run coupled runs over a range of parameters
- Used for tests over a range of values
- <https://github.com/edwardsmith999/SimWrapPy>



Disclaimer

- This is still a research code, lots of things will be rough around the edges
 - The only case currently supported is total overlap in x and z with some number of cells overlapping in y (because that's the only case we needed for our work).
- Other cases can be setup that may not work – we have error to stop some cases but not possible to predict/catch all cases**
- We have built up a testing framework to hopefully extend to more complex cases in the future (if needed).
 - Through tests and minimal scripts, we have created a (hopefully) clear way to develop and test code
 - Deployment on ARCHER2 works with module system.
 - Please help by adding working cases with automated tests (GitHub pull request).
 - Please report anything which does not work as expected (GitHub issues).

- We are coupling two separate codes to run together
 - Computational Fluid Dynamics
 - Molecular Dynamics
- Build codes separately and exchange all information as average fields through shared library (CPL library)
- This is good because it:
 - Allows separate testing of both codes
 - Maintains scope of both codes
 - Promotes optimal scaling

LAMMPS



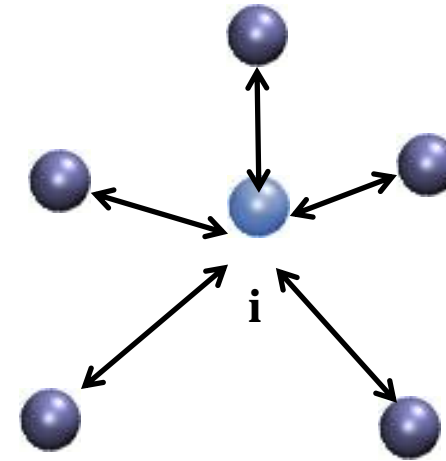
CPL LIBRARY

eCSE06-01: “Hybrid Atomistic-Continuum Simulations
of Boiling Across Scales”

- Discrete molecules in continuous space
 - Molecular position evolves continuously in time
 - Acceleration → Velocity → Position

$$\ddot{\mathbf{r}}_i \rightarrow \dot{\mathbf{r}}_i \rightarrow \mathbf{r}_i(t)$$

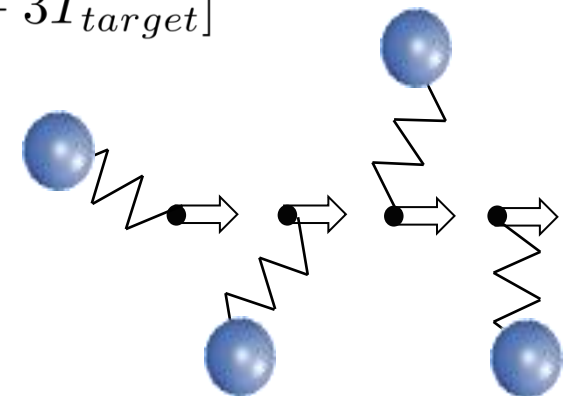
- Acceleration obtained from forces
 - Governed by Newton's law for an N-body system
 - Pairwise electrostatics interactions from quantum mechanics

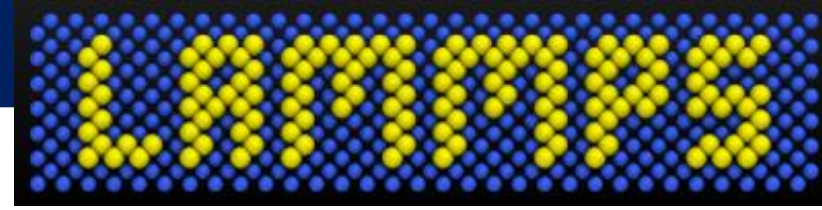


$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i = \sum_{i \neq j}^N \mathbf{f}_{ij} = \sum_{i \neq j}^N \nabla \Phi_{ij} \quad \Phi_{ij} = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^6 \right]$$

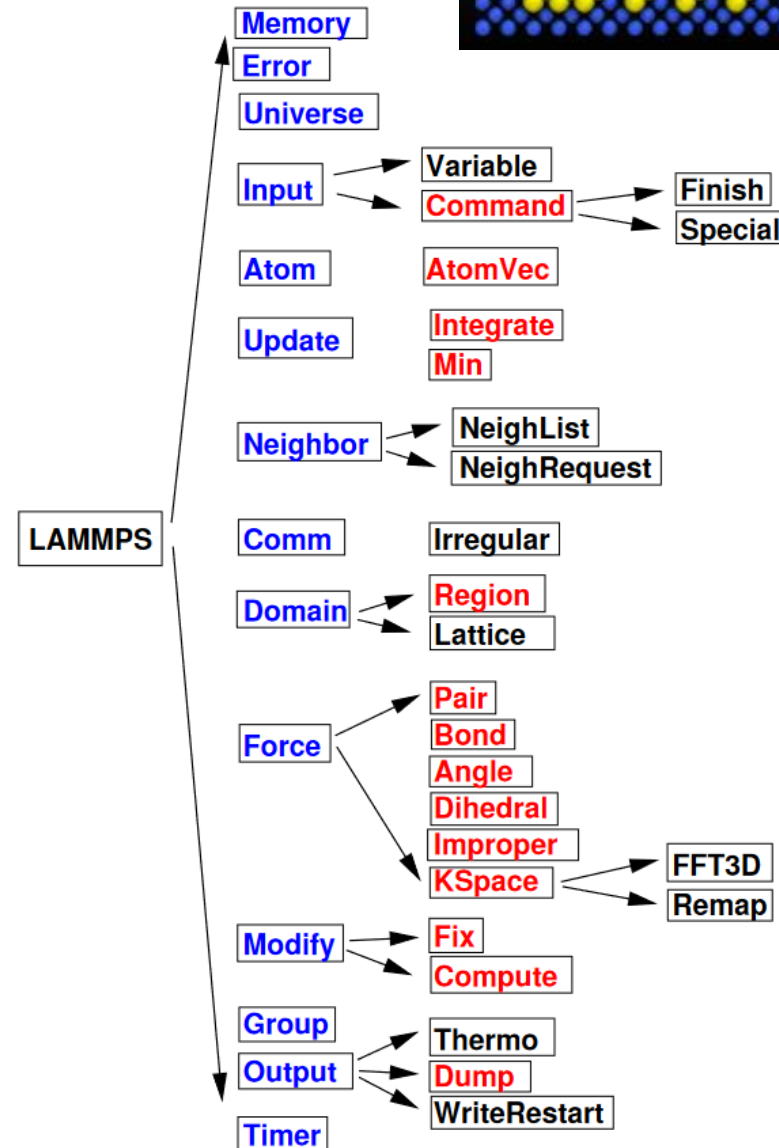
- Non-Equilibrium Molecular Dynamics (NEMD) is the study of cases beyond thermodynamic equilibrium, with:
 - Temperature gradients
 - Flow of fluid (e.g. Couette or Poiseuille flow)
- Essentially fluid dynamics - temperature gradients and flows
 - Thermostats (e.g. Nosé Hoover)
 - Solids of molecules with (an)harmonic springs linking them to tether site
 - Sliding walls by moving molecules
- Many other techniques for inducing flows...

$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i + \mathbf{F}_i^{teth} - \psi m_i \mathbf{c}_i$$
$$\dot{\psi} = \frac{1}{Q} [T - 3T_{target}]$$





- Sponsored by Sandia national labs with a great community of developers, documentation, etc
- Written in C++ and designed to be highly scalable and extensible
- A very minimal set of core code and a system of fixes which provide most functions



Compiling LAMMPS

- LAMMPS is made of a selection of packages – basically code in folders

- You can see included packages with

```
make ps
```

- Turn on packages with









```
make yes-package-name
```

- And build LAMMPS with MPI included using

```
make mpi
```

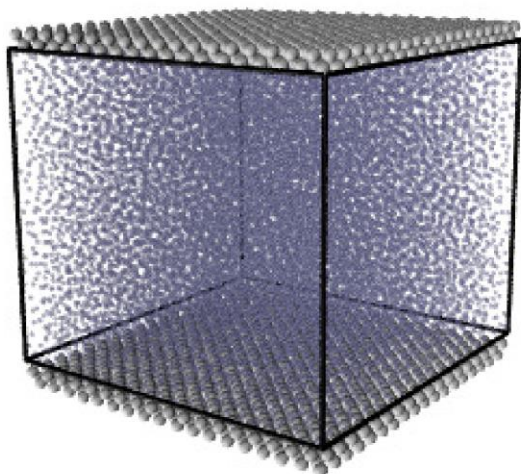
Which builds Makefile.mpi

codes > lammeps > src >

<input type="checkbox"/> Name	Date modified	Type
 ADIOS	27/03/2023 11:09	File folder
 AMOEBA	27/03/2023 11:09	File folder
 ASPHERE	27/03/2023 11:09	File folder
 ATC	27/03/2023 11:09	File folder
 AWPMD	27/03/2023 11:09	File folder
 BOCS	27/03/2023 11:09	File folder
 BODY	27/03/2023 11:09	File folder
 BPM	27/03/2023 11:09	File folder
 BROWNIAN	27/03/2023 11:09	File folder
 CG-DNA	27/03/2023 11:09	File folder
 CG-SPICA	27/03/2023 11:09	File folder
 CLASS2	27/03/2023 11:09	File folder
 COLLOID	27/03/2023 11:09	File folder

LAMMPS Input Format

- We can create walls and fluid in LAMMPS



```
# Domain size and walls
variable      x equal 12
variable      y equal 24
variable      z equal 10
variable      wallwidth equal 1.0
```

FCC lattice
units (not
domain size)

```
# Set outside of domain to be wall
variable      ylo equal -${wallwidth}
```

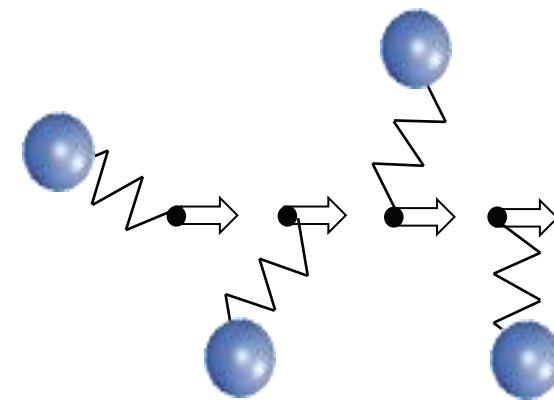
```
processors 2 1 1
```

Processor
topology

```
lattice      fcc ${rho}
region       simbox block 0 $x ${ylobuf} ${y} 0 $z
```

LAMMPS Input Format

- For tethered walls we need to create virtual atom sites (type 3) with bonds to wall atoms (type 2) and no interaction with fluid (type 1)



```
#Create a set of tethering sites (as molecules)
```

```
create_atoms      3 region lower
group             lowersites type 3
```

```
pair_style        lj/cut ${rc}
pair_coeff         1 1 1.0 1.0
pair_coeff         1 2 1.0 1.0
pair_coeff         1 3 0.0 0.0
```

Type 3 are sites so interactions set to zero

But bonds (spring) created to type 2

```
#Set imaginary site particles to not interact
bond_style        harmonic
bond_coeff         1 150.0 0.0
create_bonds      many lowersites lower 1 0.0 0.0001
```

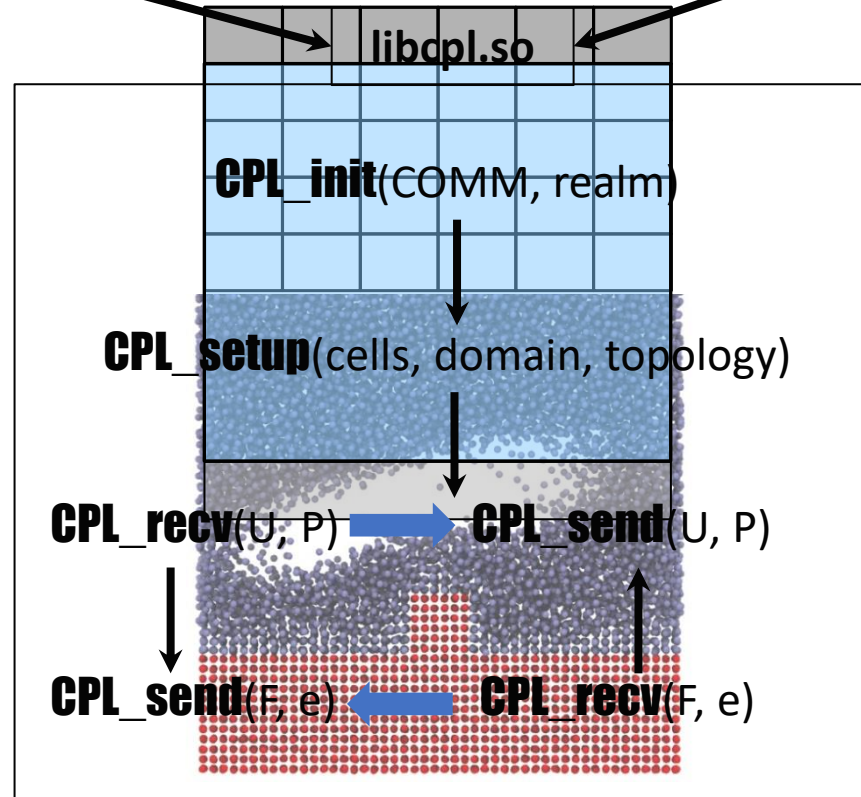
Coupling Overview

LAMMPS or
FlowMol

CPL LIBRARY

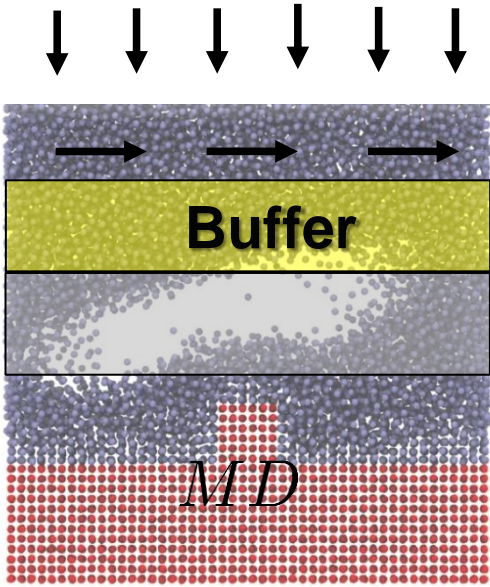
www.cpl-library.org

 OpenFOAM

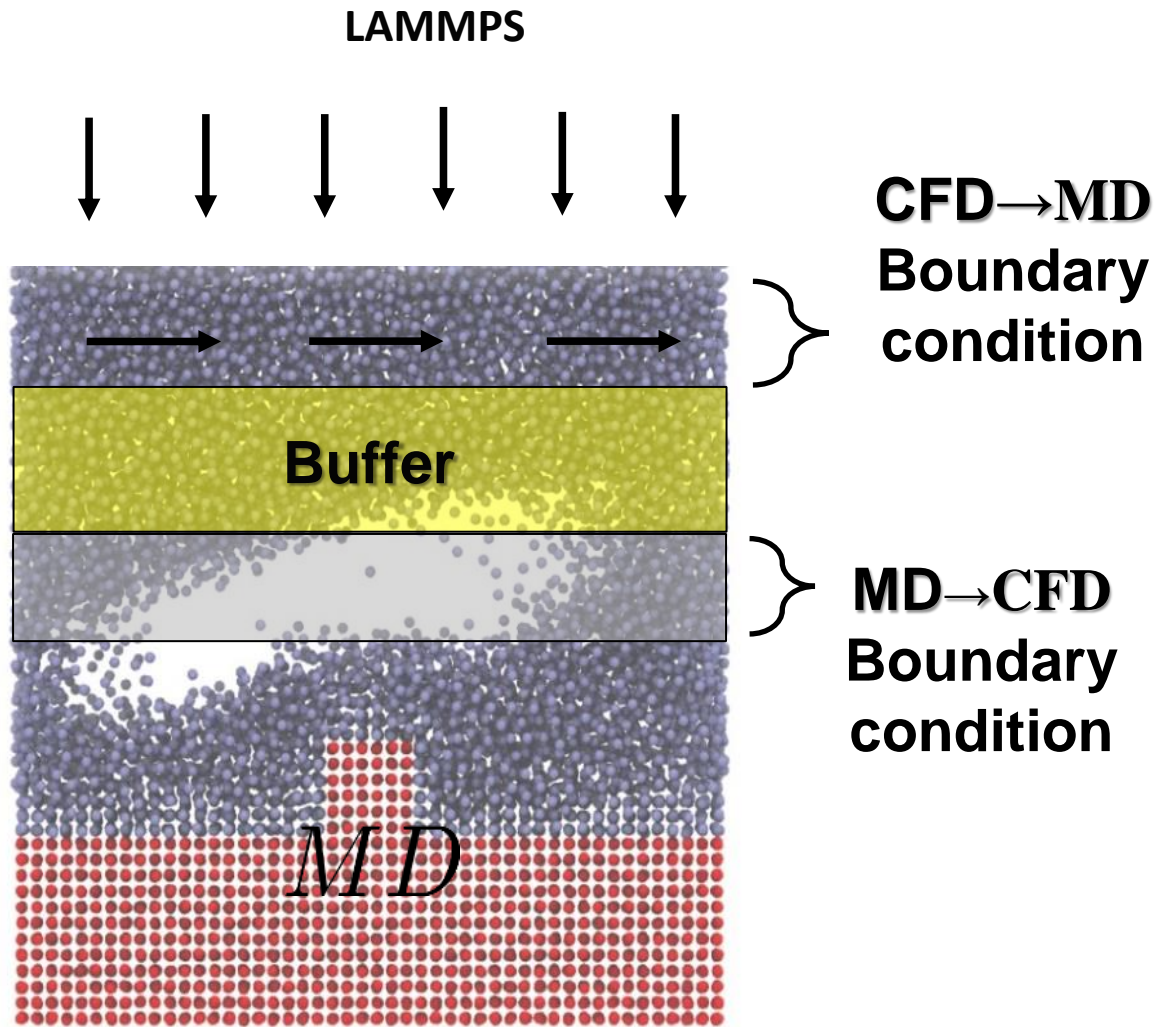


Coupling Overview

LAMMPS



Coupling Overview



Force applied to MD to make it agree with CFD here

$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i + \mathbf{F}_i^C \quad i \in \text{cell}$$

Average of MD to be passed to CFD as boundary condition

$$\sum_{i \in \text{cell}}^N m_i \mathbf{v}_i$$

Extending the Code of LAMMPS

- A “hook” system which allow users to develop code
- These can be inserted anywhere, e.g. `pre_force` or `end_of_step`
- Large portions of the code contributed by the community – “packages”

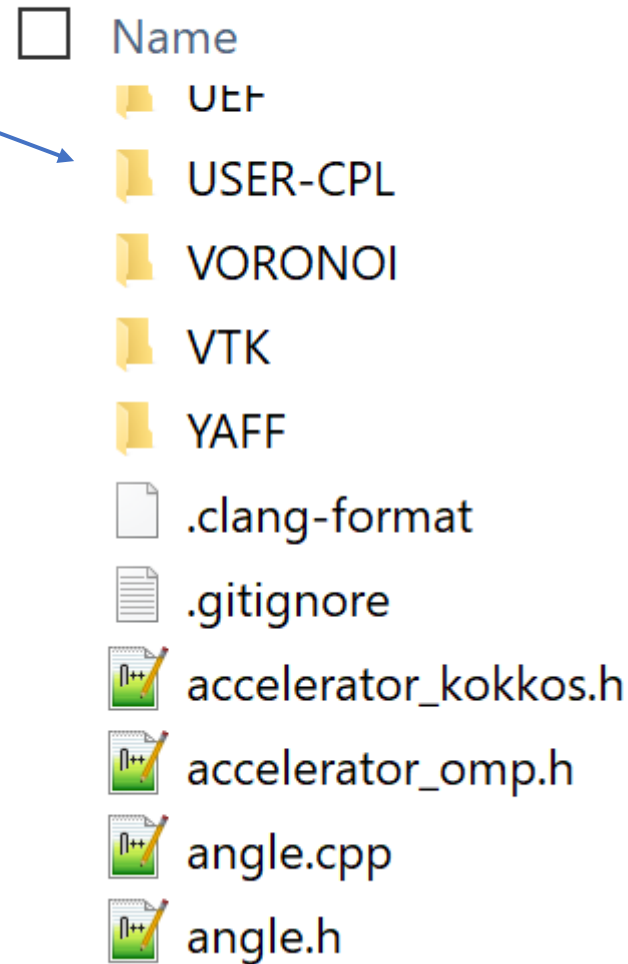
```
loop over N timesteps:  
ev_set()  
fix->initial_integrate()  
fix->post_integrate()  
nflag = neighbor->decide()  
if nflag:  
    fix->pre_exchange()  
    domain->pbc()  
    domain->reset_box()  
    comm->setup()  
    neighbor->setup_bins()  
    comm->exchange()  
    comm->borders()  
    fix->pre_neighbor()  
    neighbor->build()  
else  
    comm->forward_comm()
```

```
force_clear()  
fix->pre_force()  
pair->compute()  
bond->compute()  
angle->compute()  
dihedral->compute()  
improper->compute()  
kspace->compute()  
comm->reverse_comm()  
fix->post_force()  
fix->final_integrate()  
fix->end_of_step()  
output->write()
```

Extending LAMMPS for Coupling

- We add in USER-CPL package
- Turn on with
`make yes-user-cpl`
- And build LAMMPS including
`libcpl.so` with
`make cpl`
which builds `Makefile.cpl`
- Not in LAMMPS so we have to
copy these USER-CPL and
`Makefile.cpl` in

codes > lammps > src

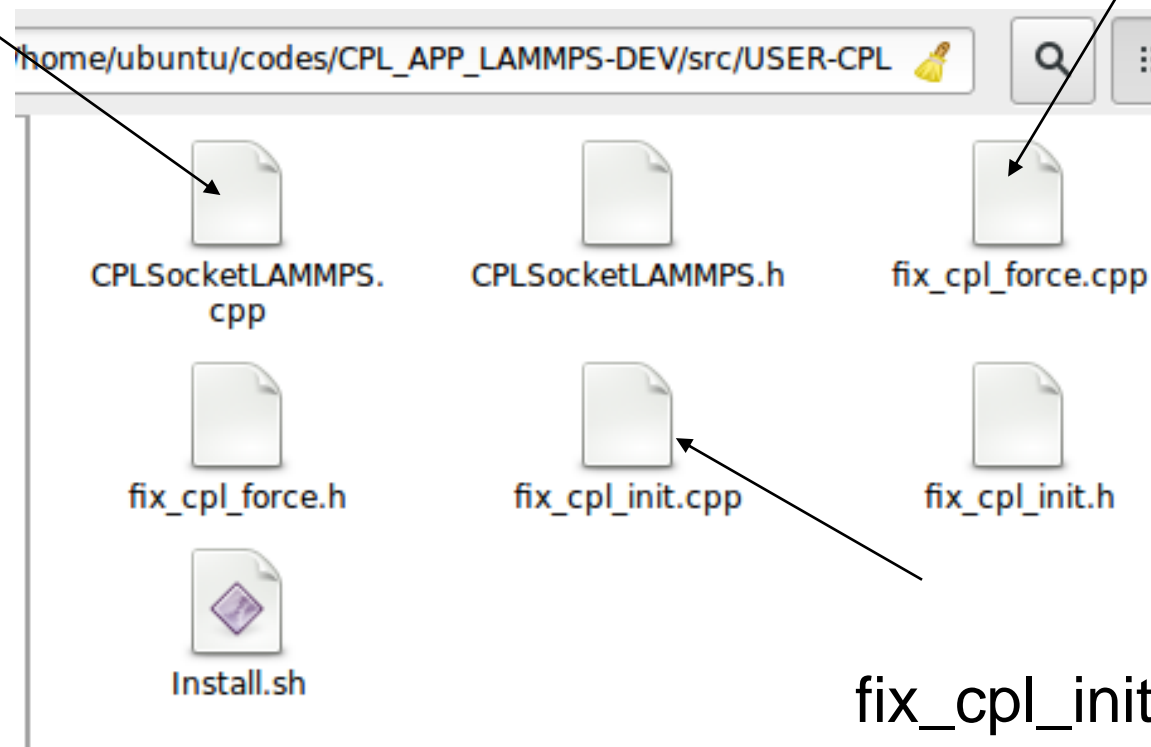


CPL_APP_LAMMPS

- We have an APP code to take care of this patching, etc

Common code here – could be used for other applications

fix_cpl_force is a fix to apply force to all particles in LAMMPS



fix_cpl_init
Setup a coupled run in LAMMPS

Compiling the APP

- The APP contains USER-CPL, a LAMMPS fix, copied to the LAMMPS directory
- Set the location of your version of LAMMPS in `CODE_INST_DIR`
- Once `CODE_INST_DIR` exists, linking to `libcpl.so` and building is all automated by a call to `make`
- Further patching needed for shared paradigm as LAMMPS assumed it has unique `MPI_COMM_WORLD`
- Add packages by changing `config/lammps_packages.in` in `CPL_APP_LAMMPS-DEV`
- You can also rebuild directly in `lammps/src` once USER-CPL is copied over

```
make yes-some-package
make cpl
```

Running Coupled LAMMPS

- A coupled run is triggered by adding the following to the input system in LAMMPS

```
fix ID all cpl/init region all args
```

- The args specify how to use CFD values to get a force (forcetype) and information to send to CFD (sendtype)
- The forcetype and sendtype must match the information sent and received by the CFD
- Example use

```
Fix cplfix all cpl/init region all forcetype Velocity xi 1.0  
sendtype velocity bndryavg below
```

forcetype

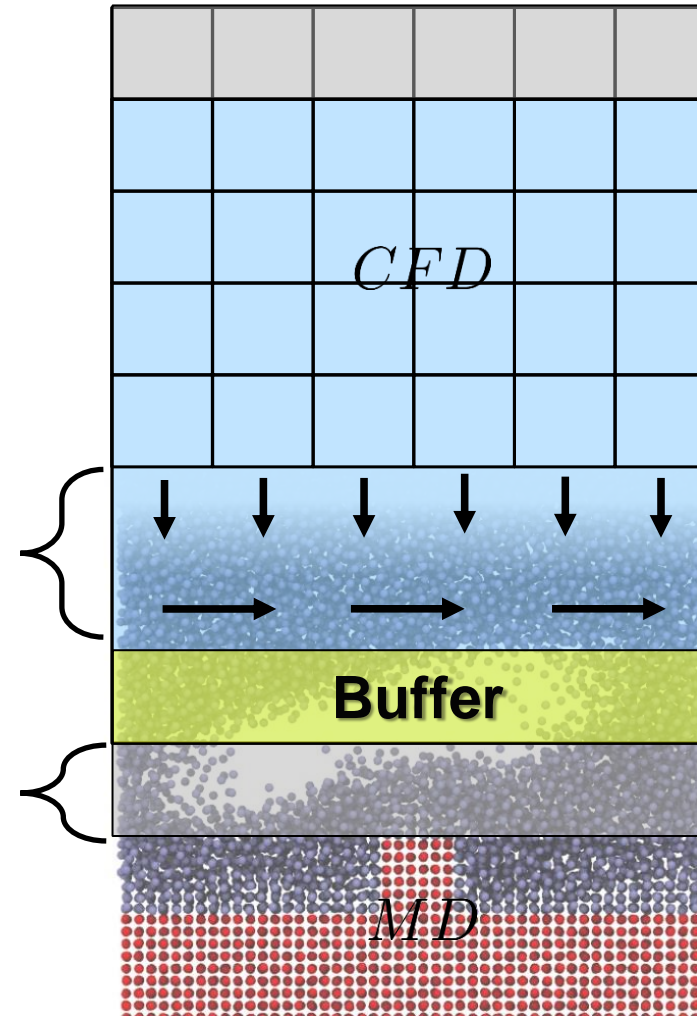
$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i + \mathbf{F}_i^C \quad i \in \text{cell}$$

**CFD→MD
Boundary
condition**

sendtype

$$\sum_{i \in \text{cell}}^N m_i \mathbf{v}_i$$

**MD→CFD
Boundary
condition**



- The forcetype itself is specified by next word,
 - For MD flows, this can either be Velocity or Stresses.
 - For granular flows can be a range of drag models Test, Drag, Stokes, Di_Felice, Ergun, Tang, BVK.
 - Designed to be easy to add new ones.
- Some forcetypes require additional inputs, added as words followed by true/false or setting of values (see www.cpl-library.org for documentation):
 - overlap, interpolate gradP, divStress, preforce_everytime (true/false)
 - Cd, mu, rho (values)

Types of MD Coupling Forces

- Velocity (State Coupling)* (CPLForceVelocity)

$$\mathbf{F}_i^C = \xi \left[\mathbf{u}^{CFD} - \sum_{i \in cell} \mathbf{v}_i \right]$$

- Stress (Flux Coupling)** (CPLForceFlekkoy)

$$\mathbf{F}_i^C = g(y) \mathbf{\Pi}^{CFD} \cdot \mathbf{n}$$

Weighting function to
distribute force on molecules

Stress tensor from CFD

$$\mathbf{\Pi} = P\mathbf{I} - \mu \nabla \mathbf{u}$$

*O'Connell and Thompson (1995) and Nie et al (2004) ** Flekkoy et al (2000)

Available Sendtypes

- A range of possible field based quantities can be calculated and sent by adding in any order after sendtype

- **Fundamental Types**

- NBINS – Number of particles in a volume
- VEL – sum of velocity in a volume
- STRESS – Virial style stress in particle system

$$\sum_{i \in cell}^N m_i$$

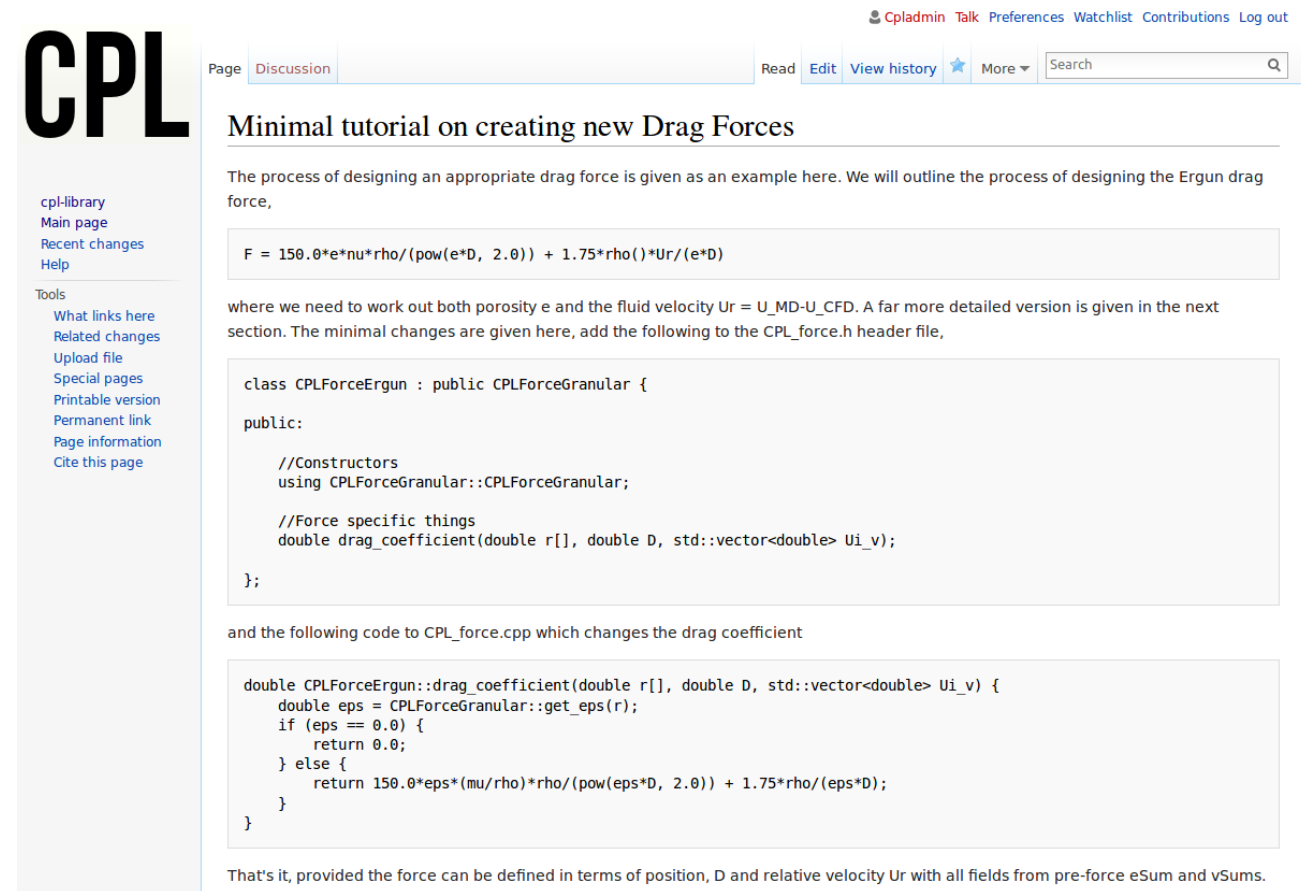
$$\sum_{i \in cell}^N m_i \mathbf{v}_i$$

- **Mixed types include:**

- velocity = VEL + NBIN

$$\sum_{i \in cell}^N m_i \mathbf{v}_i \mathbf{v}_i + \sum_{i,j \in cell}^N \mathbf{f}_{ij} \mathbf{r}_{ij}$$

- See CPL library wiki: https://www.cpl-library.org/docs/Main_Page.shtml
- The LAMMPS section has a tutorial on designing new force types
 - This force can then be included in LAMMPS as outlined in `fix_cpl_force`



The screenshot shows a wiki page for the CPL library. The page title is "Minimal tutorial on creating new Drag Forces". The content describes the process of designing a drag force, using the Ergun drag force as an example. It provides a mathematical formula for the force and a C++ code snippet for implementing it in the CPL library. The page also includes a sidebar with navigation links and a search bar.

CPL

Page: Discussion | Read | Edit | View history | More | Search

Minimal tutorial on creating new Drag Forces

The process of designing an appropriate drag force is given as an example here. We will outline the process of designing the Ergun drag force,

$$F = 150.0 * e * \nu * \rho / (\text{pow}(e * D, 2.0)) + 1.75 * \rho * U_r / (e * D)$$

where we need to work out both porosity e and the fluid velocity $U_r = U_{MD} - U_{CFD}$. A far more detailed version is given in the next section. The minimal changes are given here, add the following to the `CPL_force.h` header file,

```
class CPLForceErgun : public CPLForceGranular {
public:
    //Constructors
    using CPLForceGranular::CPLForceGranular;

    //Force specific things
    double drag_coefficient(double r[], double D, std::vector<double> U_i_v);
};
```

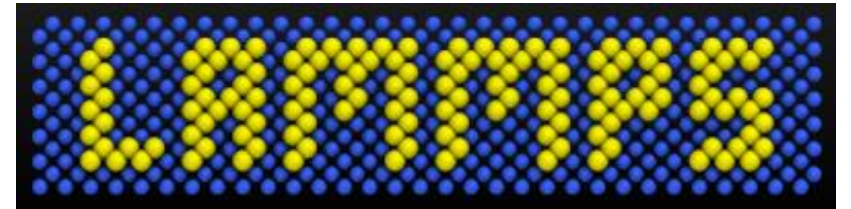
and the following code to `CPL_force.cpp` which changes the drag coefficient

```
double CPLForceErgun::drag_coefficient(double r[], double D, std::vector<double> U_i_v) {
    double eps = CPLForceGranular::get_eps(r);
    if (eps == 0.0) {
        return 0.0;
    } else {
        return 150.0 * eps * (mu / rho) * rho / (pow(eps * D, 2.0)) + 1.75 * rho / (eps * D);
    }
}
```

That's it, provided the force can be defined in terms of position, D and relative velocity U_r with all fields from pre-force `eSum` and `vSums`.

Summary

- LAMMPS is a powerful MD tool
 - We can extend by adding package USER-CPL for coupling
 - CPL_APP_LAMMPS-DEV does this for you (to build `lmp_cpl`)
 - Available pre-built as ARCHER2 module



- To setup a coupled run
 - Add a Fix cplfix to the LAMMPS input:

```
Fix cplfix all cpl/init region all forcetype Velocity xi 1.0  
sendtype velocity bndryavg below
```
 - Different sendtypes and forcetypes are available
 - Easy to extend in order to add your own
- A COUPLER.in file will also be needed (as discussed in the next section)

Coupled Couette Flow

Full example code can be found here:

https://github.com/Crompulence/cpl-library/tree/master/examples/LAMMPS_OPENFOAM

Needs both APPS installed:

https://github.com/Crompulence/CPL_APP_LAMMPS-DEV

https://github.com/Crompulence/CPL_APP_OPENFOAM



CPL LIBRARY

eCSE06-01: “Hybrid Atomistic-Continuum Simulations
of Boiling Across Scales”

Domain Decomposition Coupling

- Finite Volume Solver

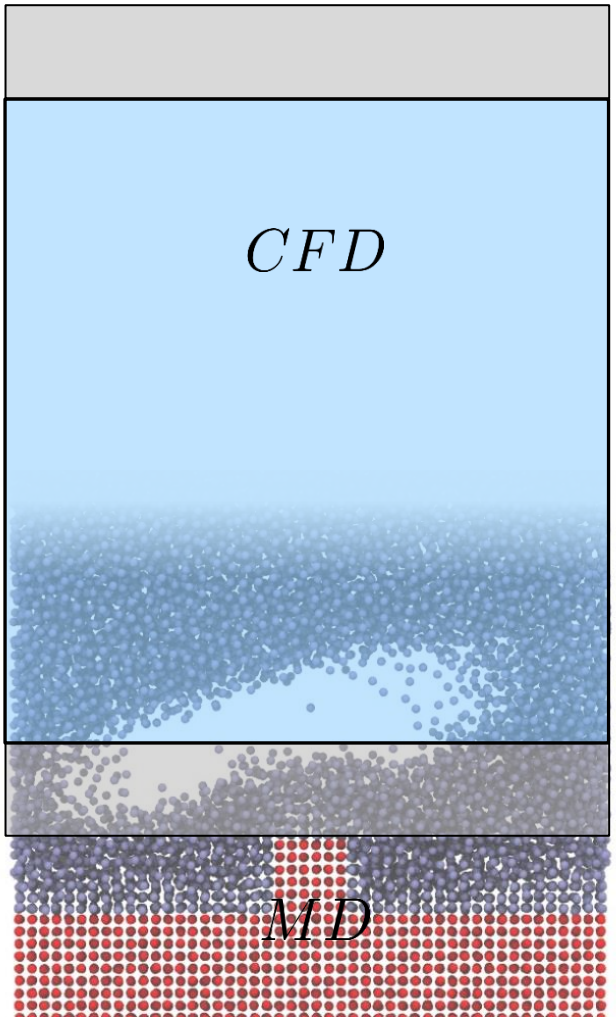
$$\frac{\partial}{\partial t} \int_V \rho u dV = - \oint_S \rho u u \cdot d\mathbf{S} - \oint_S \boldsymbol{\Pi} \cdot d\mathbf{S}$$

Share the same
time and **length**
scales



- Discrete molecules

$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i \text{ for all } i \text{ in } N$$



O'Connell Thompson (1995), Hadjiconstantinou (1998), Flekkoy (2000), Nie et al (2004).

Coupled CFD-MD Simulation

- Finite Volume Solver

$$\frac{\partial}{\partial t} \int_V \rho u dV = - \oint_S \rho u u \cdot d\mathbf{S} - \oint_S \boldsymbol{\Pi} \cdot d\mathbf{S}$$

forcetype

$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i + \mathbf{F}_i^C \quad i \in \text{cell}$$

**CFD→MD
Boundary
condition**

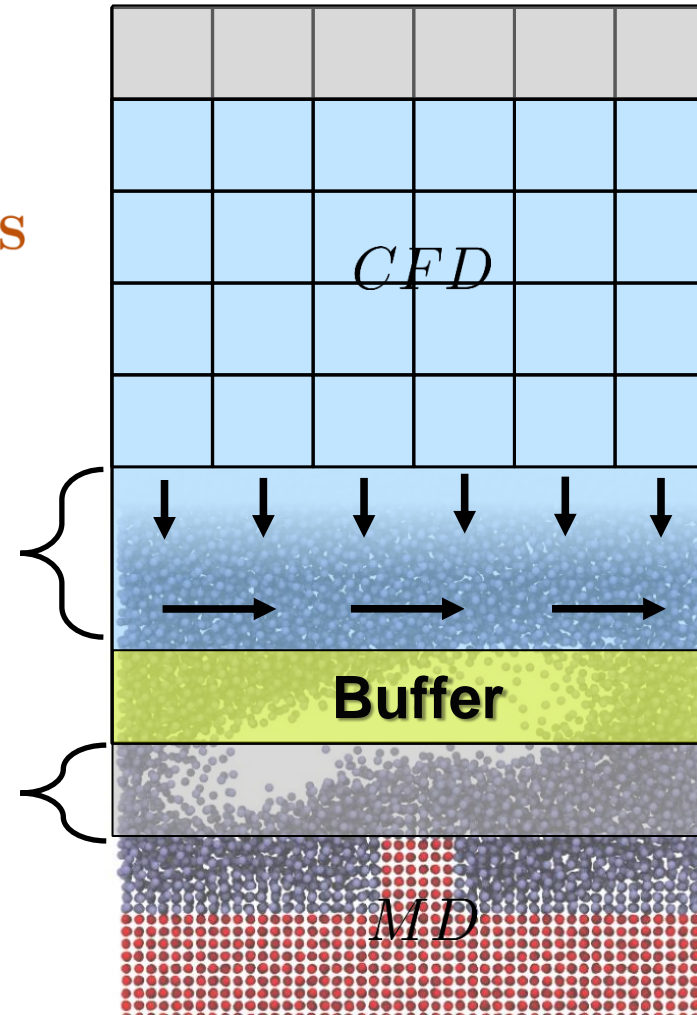
sendtype

$$\sum_{i \in \text{cell}}^N m_i \mathbf{v}_i$$

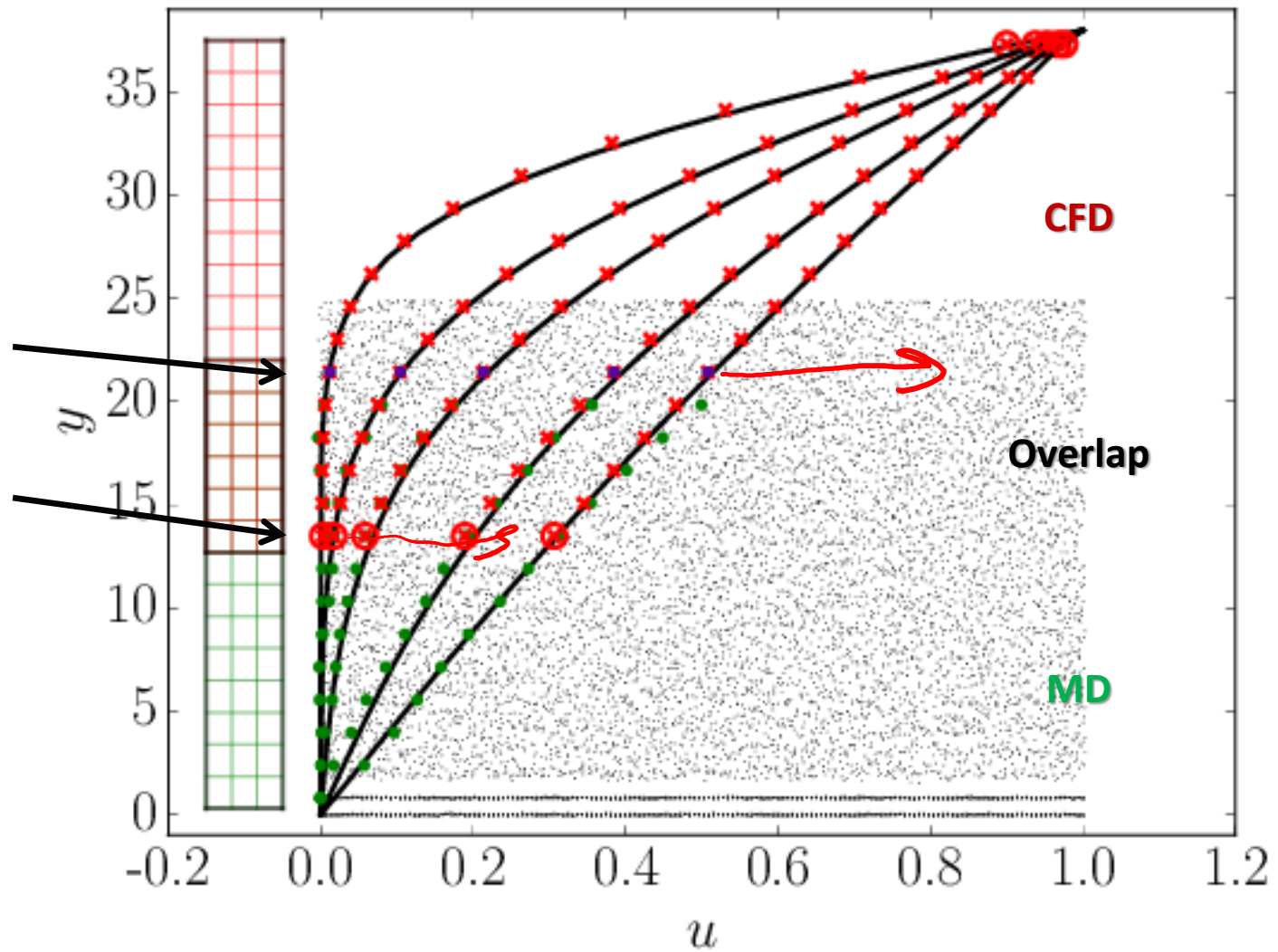
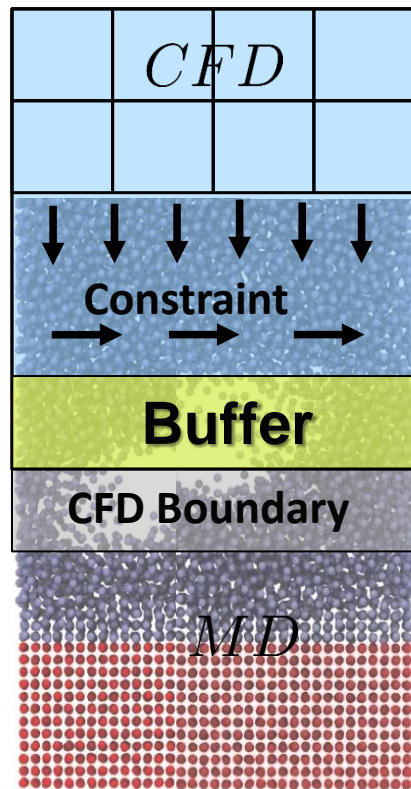
**MD→CFD
Boundary
condition**

- Discrete molecules

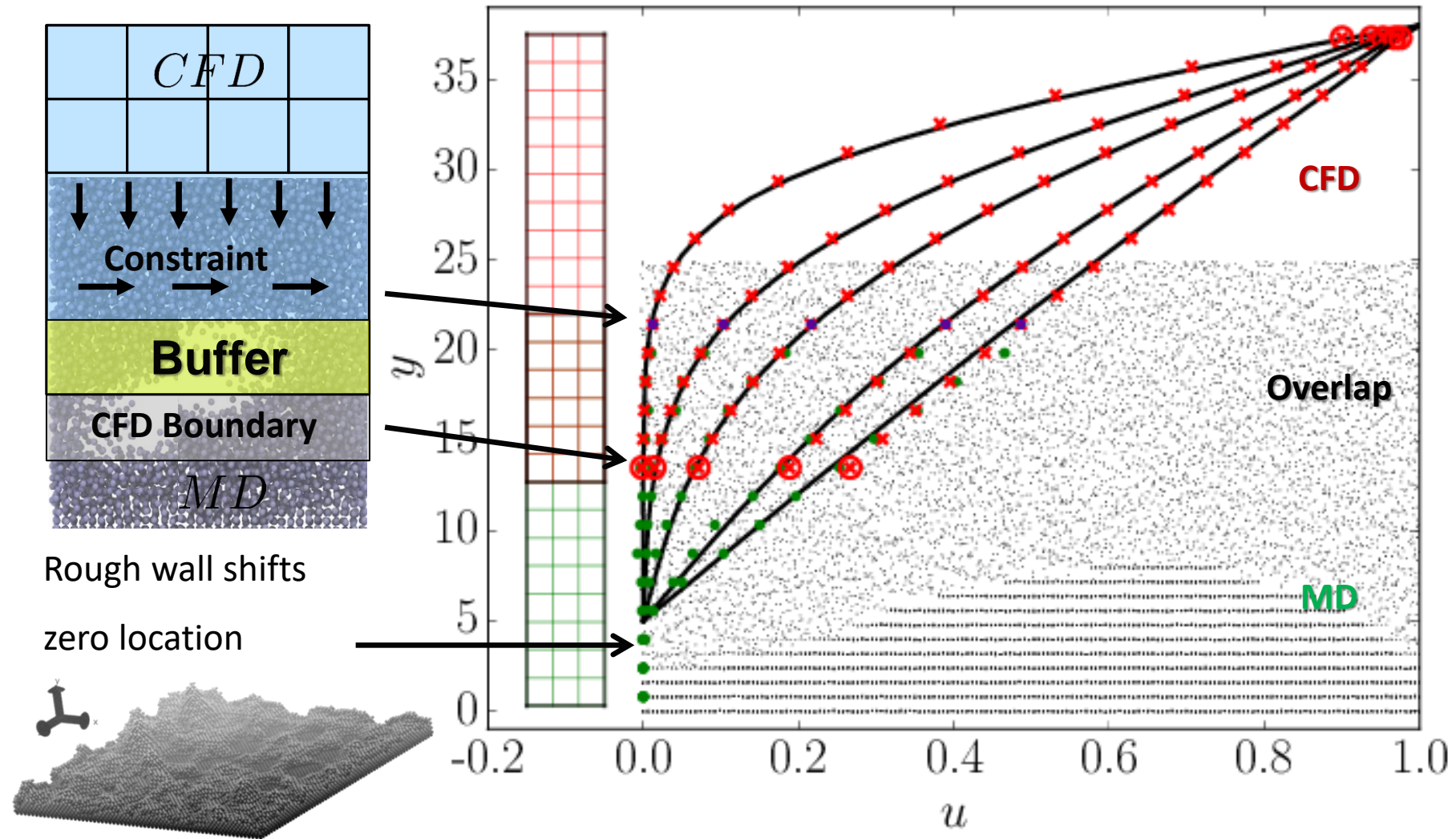
$$m_i \ddot{\mathbf{r}}_i = \mathbf{F}_i \text{ for all } i \text{ in } N$$



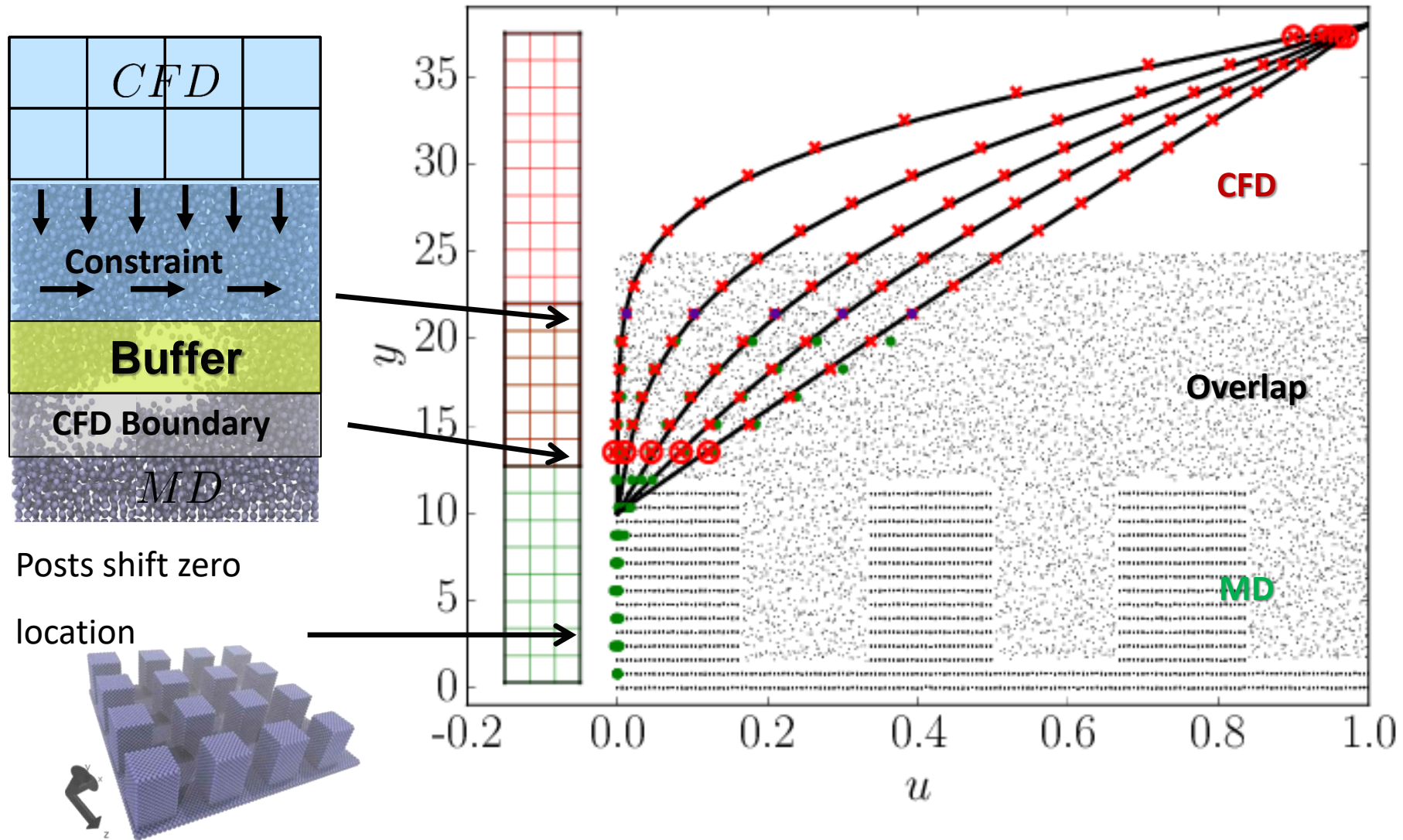
Coupling Results – Couette Flow



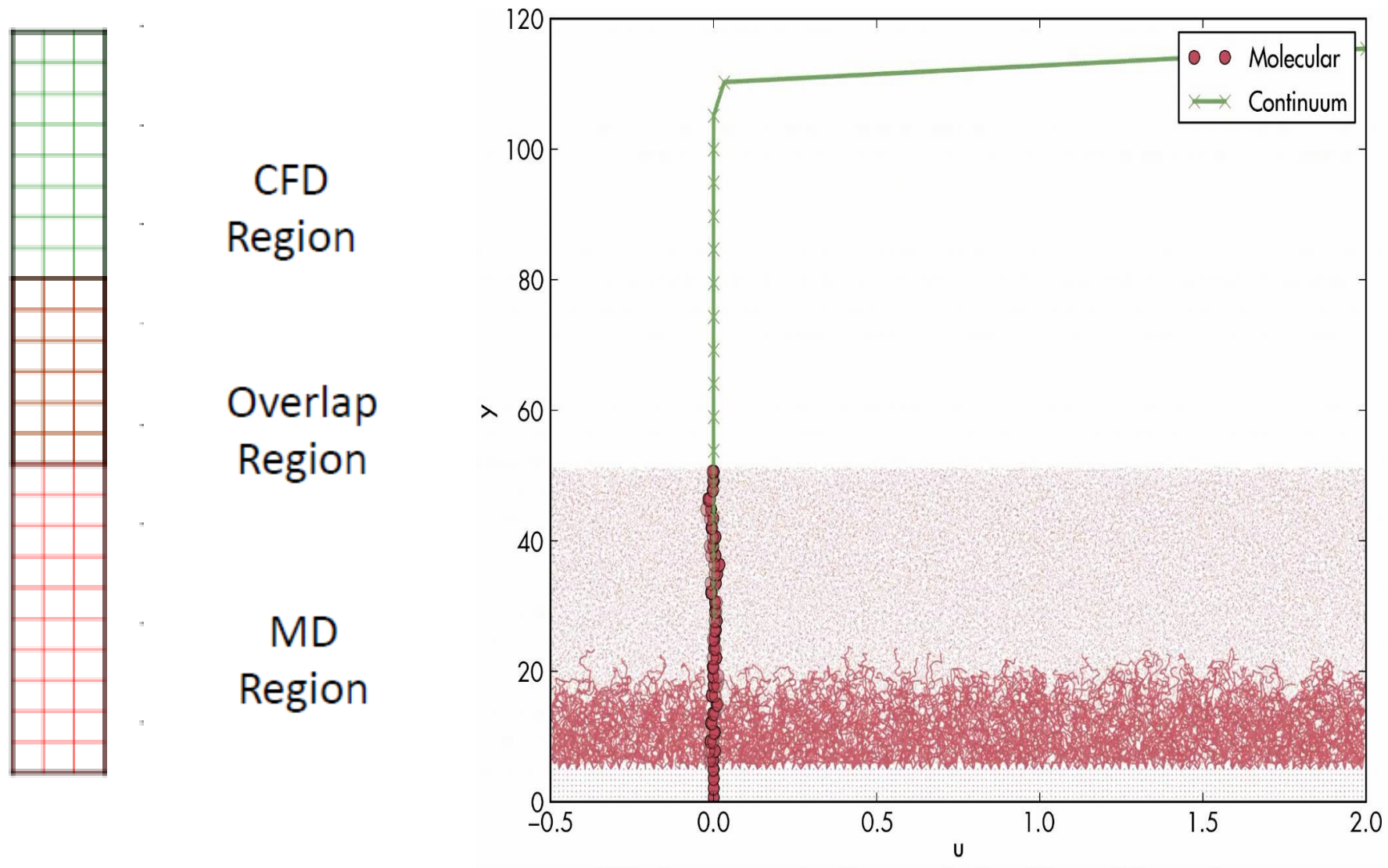
Coupling Results – Couette Flow with Wall Roughness



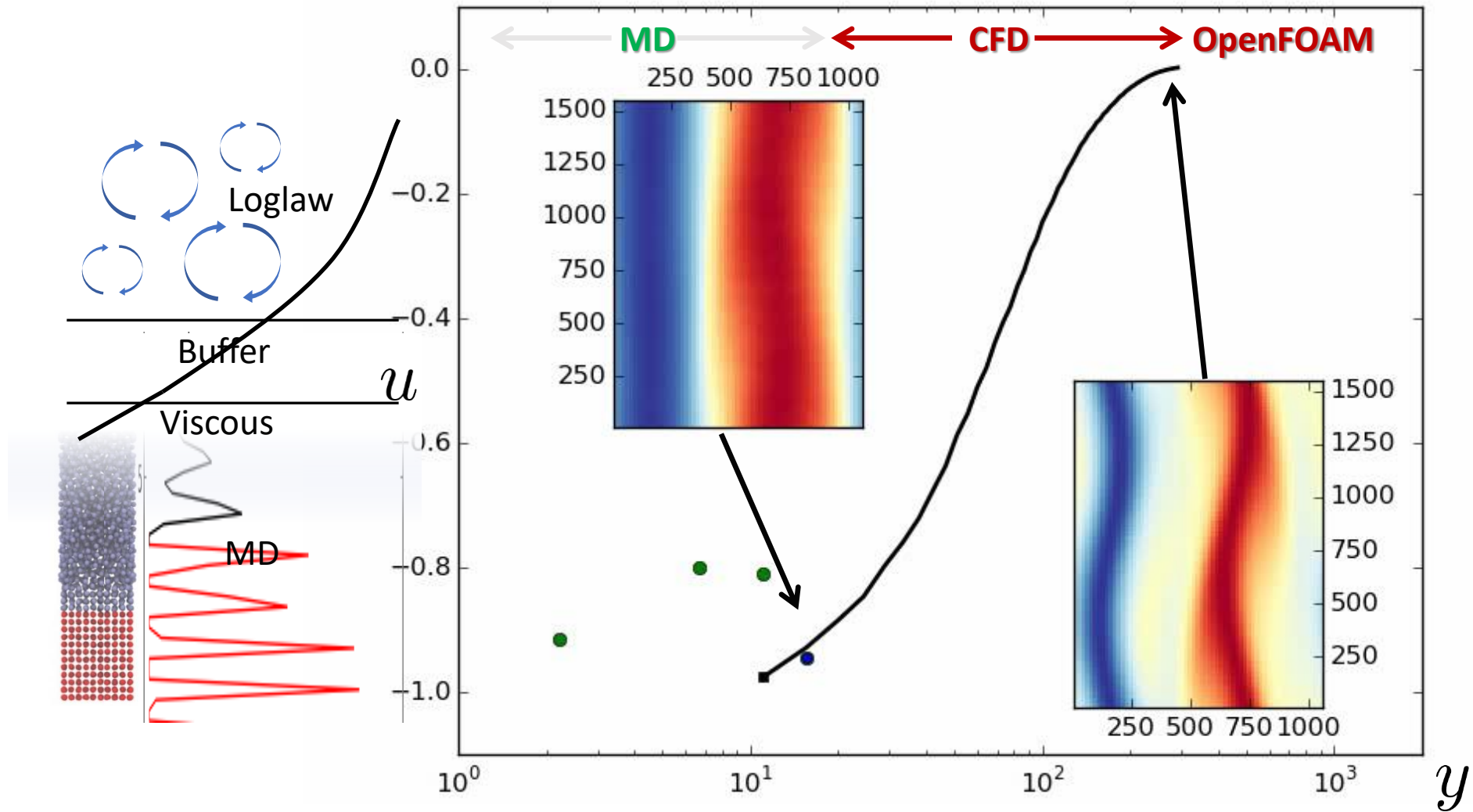
Coupling Results – Couette Flow with Wall Texture (superhydrophobic)



Coupling Results – Polymer Brushes for Tribology



Coupling Results – Turbulent Couette



- Stability – Numerical blowup is a big problem in CFD with CFL number based on timestep, velocity and grid resolution a first check
- OpenFOAM prioritizes stability but can still be subject to numerical instability, very difficult to debug if this is because of
 - An error in coupling exchange (coding/setup)
 - Too much noise passed from the MD causing an instability
 - A numerical instability in OpenFOAM itself (i.e. CFL violated or a non-linear instability)
- So, we suggest using Mocks

Start with the Mocks!

- Step 0 – check geometry, processor topology and visualise



- Step 1 – send a boundary condition and check CFD response



- Step 2 – apply a constant force and check MD averaging is correct



- Step 3 – compare overall system to analytical solution (if possible)



Start with the Mocks!

- Step 0 – check geometry, processor topology and visualise

minimal_MD.py



minimal_CFD.py

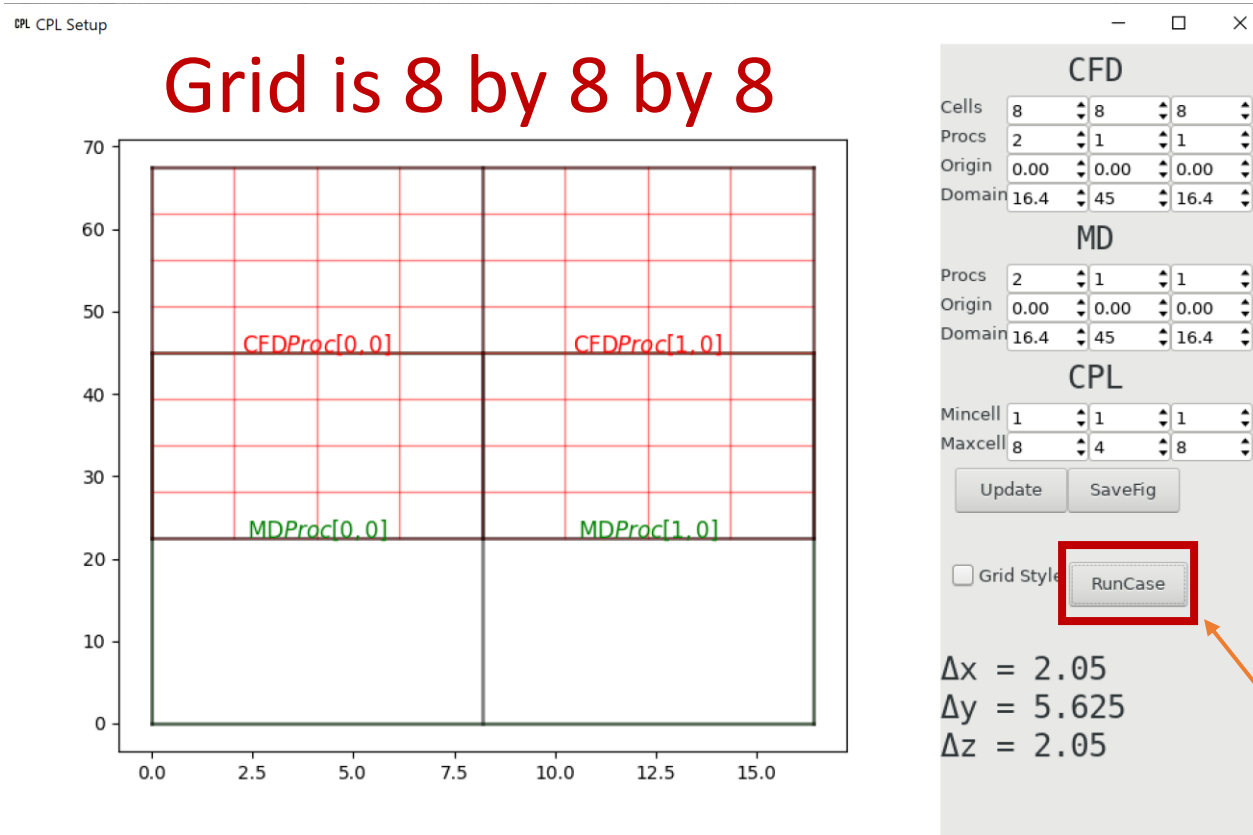
minimal MD and minimal CFD

- Step 0 – check geometry, processor topology and visualise

minimal_MD.py



minimal_CFD.py



- LAMMPS setup in FCC units

processors 2 1 1

variable x equal 10

variable y equal 24

variable z equal 10

region simbox block 0 \$x \${ylobuf} \$y 0 \$z

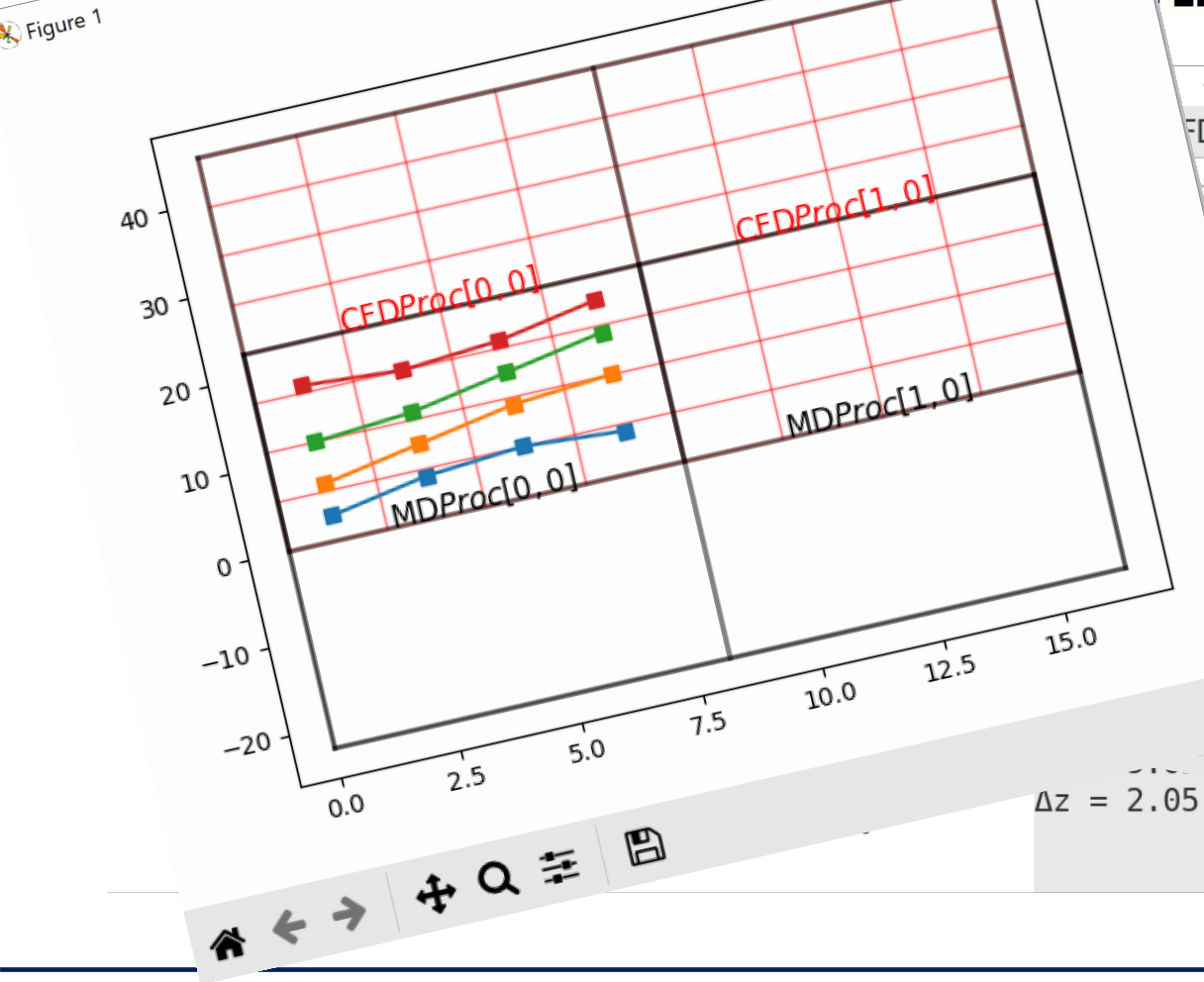
- Run to see physical domain size, in log.lammps we see

Created orthogonal box = (0 -5.0387886 0)
to (16.795962 40.310309 16.795962)

Then run to make sure it works

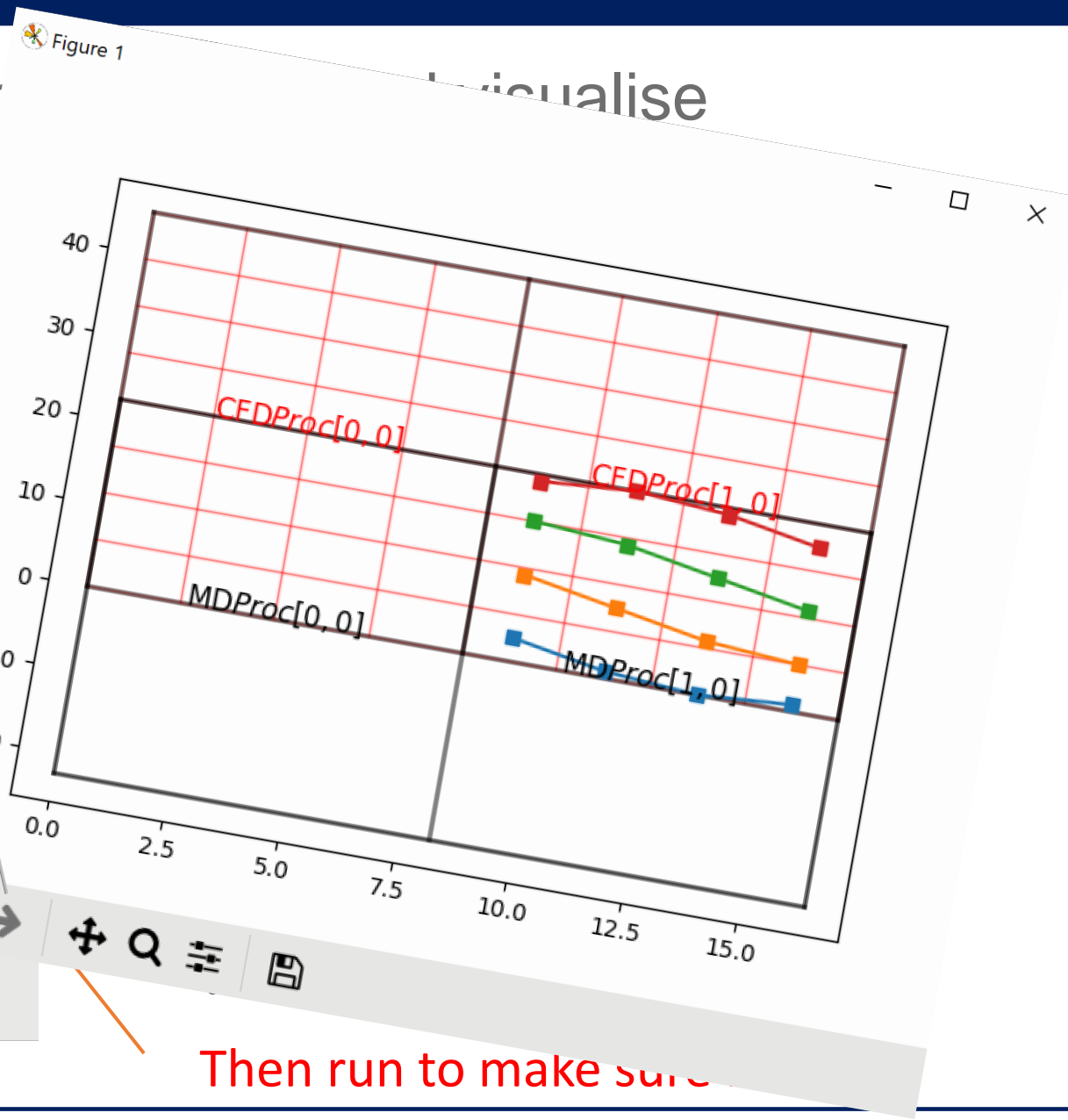
minimal MD and minimal CFD

- Step 0 – check accuracy



LIBRA

$\Delta z = 2.05$



Then run to make sure

minimal MD and minimal CFD

- Step 0 – check geometry, processor topology and visualise

minimal_MD.py

```
# Parameters of the cpu topology (cartesian grid)
```

```
npxyz = [2, 1, 1]
```

```
xyzL = [16.795961913825074, 45.349097,  
        16.795961913825074]
```

```
...
```

```
#Setup coupled simulation
```

```
...
```

```
CPL.setup_md(cart_comm, xyzL, xyz_orig)
```

Change domain to match
desired LAMMPS
domain size

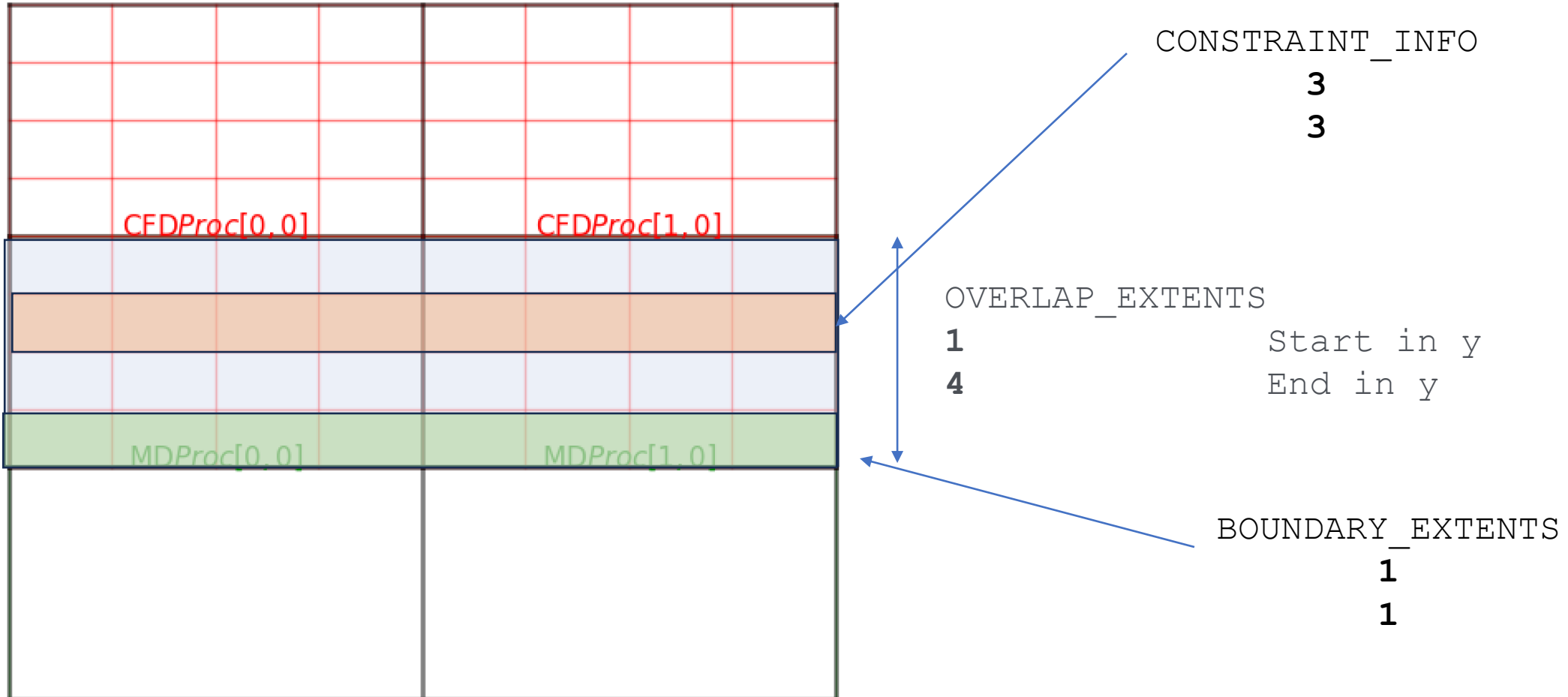
COUPLER.in

Variable	OVERLAP_ EXTENTS	CONSTRAINT _INFO	BOUNDARY_E XTENTS
Forcetype		2	
Flag		0	
xmin	1	1	1
xmax	8	8	8
ymin	1	3	1
ymax	4	3	1
zmin	1	1	1
zmax	8	8	8

Setup overlap region in cells

minimal MD and minimal CFD

- Change COUPLER.in to set coupling overlap



minimal MD and OpenFOAM

- Step 0 – check geometry, processor topology and visualise

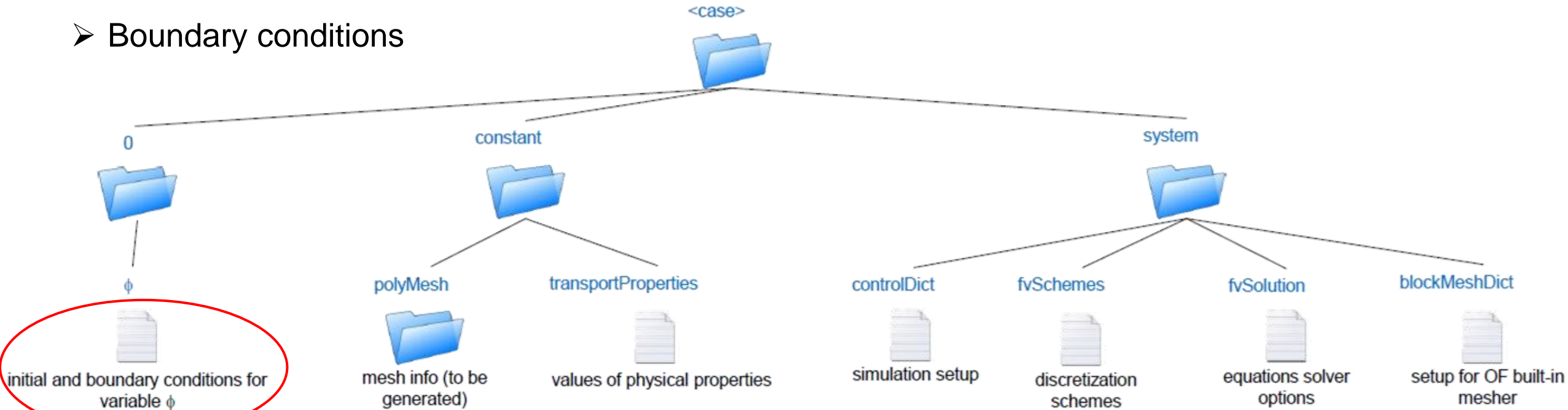


- Step 1 – send a boundary condition and check CFD response



Next, we need to setup OpenFOAM to match

- Typical folder tree structure of any OpenFOAM simulation
- Boundary conditions

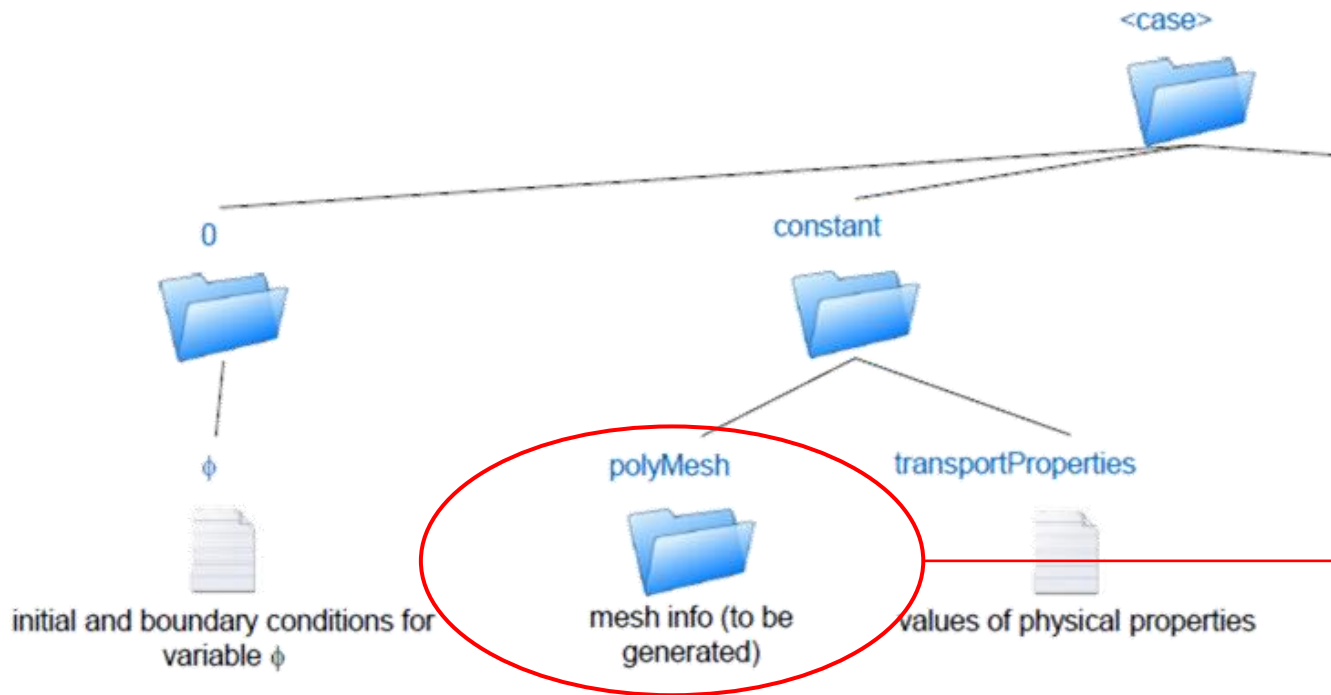


In 0.orig/U:
boundaryField
{
 CPLReceiveMD
 {
 type fixedValue;
 value \$internalField;
 }
}

- Bottom boundary is set to CPLReceieveMD type, the `sendType` from LAMMPS
- Exchange handled internally by CPL library

Overview of the OpenFOAM configuration files

➤ Mesh



In constant/polyMesh/blockMeshDict

```
scale 16.795961913825074;  
  
vertices  
(  
  (0 0 0)  
  (1.0 0 0)  
  (1.0 2.7 0)  
  (0 2.7 0)  
  (0 0 1.0)  
  (1.0 0 1.0)  
  (1.0 2.7 1.0)  
  (0 1.0 1.0)  
);  
  
blocks(hex (0 1 2 3 4 5 6 7) (8 8 8) simpleGrading (1 1 1));  
  
boundary  
(  
  ...  
  CPLReceiveMD  
  {  
    type patch;  
    faces  
    (  
      (1 5 4 0)  
    );  
  }  
)
```

Grid is 8 by 8 by 8

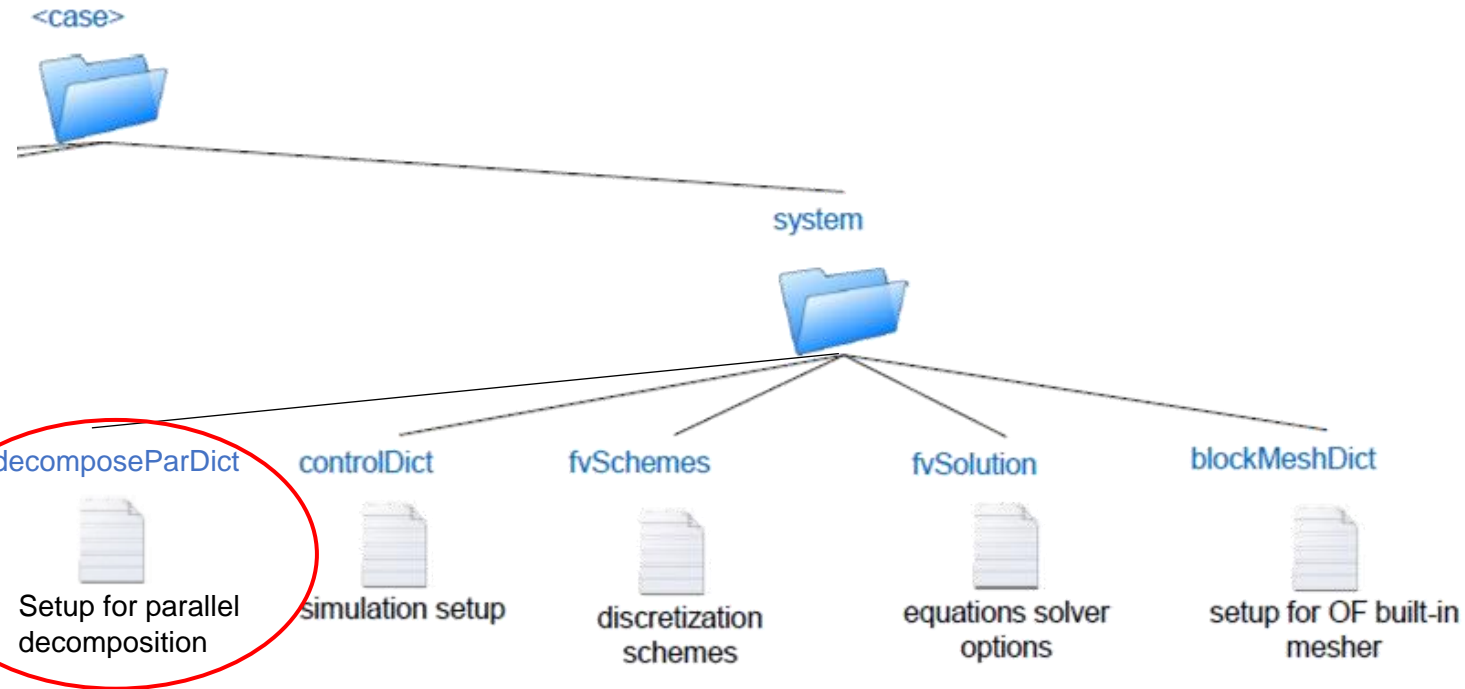
Overview of the OpenFOAM configuration files

➤ Domain parallel decomposition

In system/decomposeParDict

```
numberOfSubdomains 2;  
method simple;  
simpleCoeffs  
{  
  n (2 1 1);  
  delta 0.001;
```

Processor topology is
2 by 1 by 1



Overview of the OpenFOAM configuration files

Topology

In system/decomposeParDict

```
numberOfSubdomains 2;  
method simple;  
simpleCoeffs  
{  
  n (2 1 1);  
  delta 0.001;
```

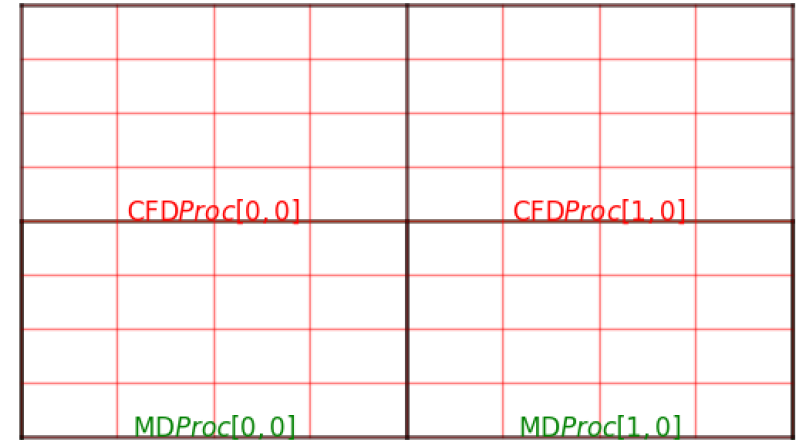
Boundary Condition

In 0.orig/U:

```
boundaryField  
{  
  CPLReceiveMD  
  {  
    type fixedValue;  
    value $internalField;  
  }
```

In constant/polyMesh/blockMeshDict

```
scale 16.795961913825074;  
vertices  
(  
  (0 0 0)  
  (1.0 0 0)  
  (1.0 2.7 0)  
  (0 2.7 0)  
  (0 0 1.0)  
  (1.0 0 1.0)  
  (1.0 2.7 1.0)  
  (0 1.0 1.0)  
);  
blocks(hex (0 1 2 3 4 5 6 7) (8 8 8) simpleGrading (1 1 1));  
boundary  
(  
  ...  
  CPLReceiveMD  
  {  
    type patch;  
    faces  
    (  
      (1 5 4 0)  
    );  
  }
```



Setup identical to
minimal_CFD.py
so we can swap out
for OpenFOAM

minimal MD and OpenFOAM

- Step 0 – check geometry, processor topology and visualise



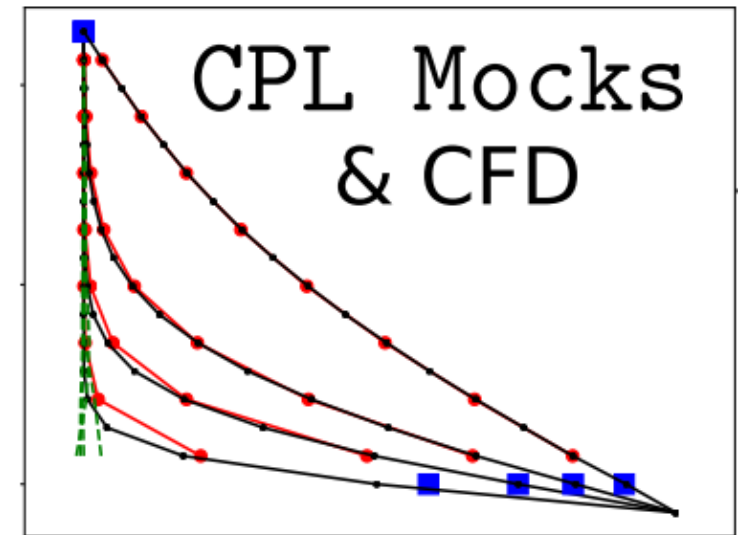
- Step 1 – send a boundary condition and check CFD response



```
send_array[0, :, :, :] = testBC  
CPL.send(send_array)
```

```
CPL.recv(recv_array)  
test_data(recv_array)
```

- Specify boundary condition to drive flow
- Test response against analytical solution for Couette flow



Start with the Mocks!

- Step 0 – check geometry, processor topology and visualise



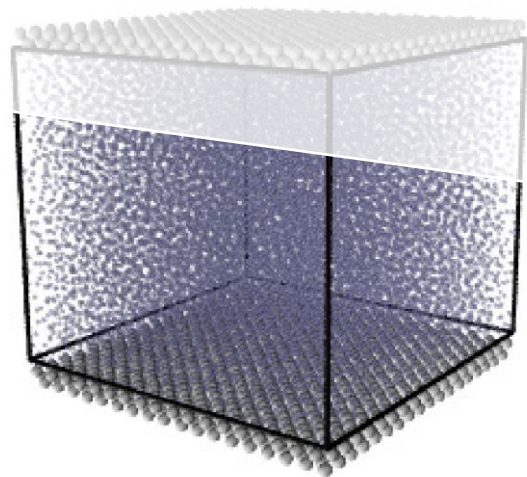
- Step 1 – send a boundary condition and check CFD response



- Step 2 – apply a constant force and check MD averaging is correct

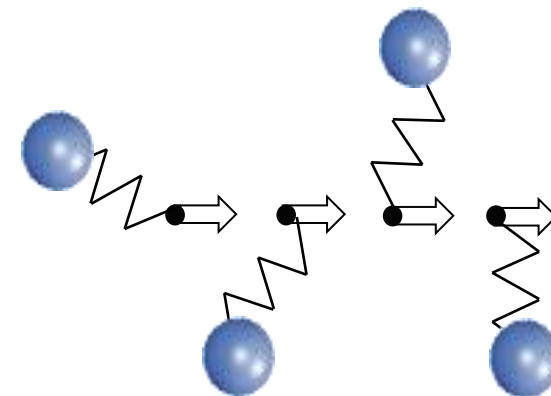


LAMMPS Input Format



```
# Domain size and walls  
variable      x equal 12  
variable      y equal 24  
variable      z equal 10
```

Take an MD Couette flow
case and cut the top off



```
#Create a set of tethering sites (as molecules)  
create_atoms 3 region lower  
group        lowersites type 3
```

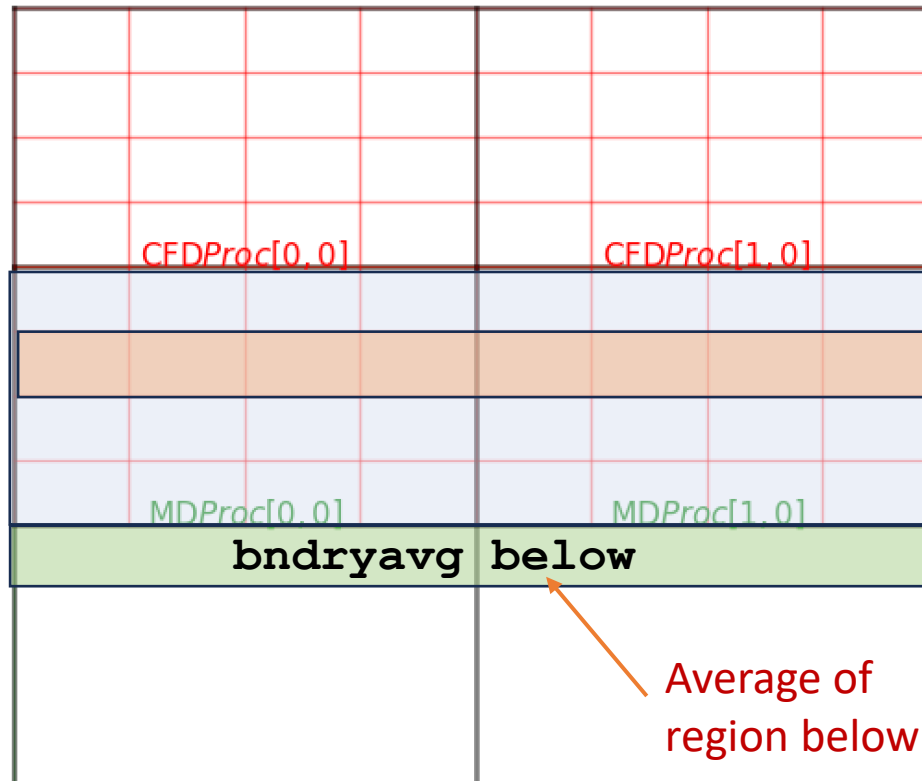
```
# Turn on coupling case and specify velocity constraint force (forcetype)  
# and velocity/density averaging and exchange (sendtype)  
fix cplfix all cpl/init region all forcetype Velocity xi 1.0  
      sendtype velocity bndryavg below
```

LAMMPS Input Format

forcetype

$$\mathbf{F}_i^C = \xi \left[\mathbf{u}^{CFD} - \sum_{i \in cell} \mathbf{v}_i \right]$$

Constraint location
specified in COUPLER.in
Forcetype chosen below



sendtype

$$\sum_{i \in cell}^N m_i \mathbf{v}_i$$

```
# Turn on coupling case and specify velocity constraint force (forcetype)
# and velocity/density averaging and exchange (sendtype)
fix cplfix all cpl/init region all forcetype Velocity xi 1.0
sendtype velocity bndryavg below
```

Start with the Mocks!

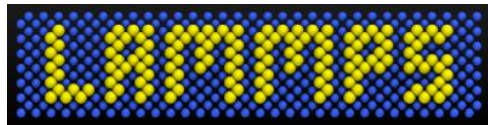
- Step 0 – check geometry, processor topology and visualise



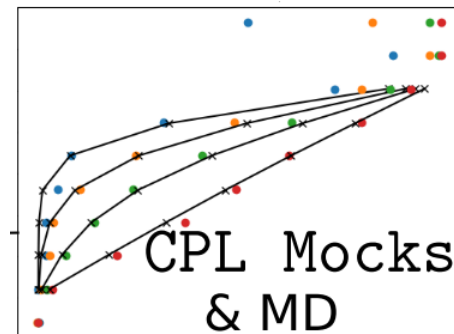
- Step 1 – send a boundary condition and check CFD response



- Step 2 – apply a constant force and check MD averaging is correct



- Apply force
- Compare received value to analytical Couette solution



```
send_array[0, :, :, :] = testforce
CPL.send(send_array)
CPL.recv(recv_array)
test_data(recv_array)
```

Start with the Mocks!

- Step 0 – check geometry, processor topology and visualise



- Step 1 – send a boundary condition and check CFD response



- Step 2 – apply a constant force and check MD averaging is correct

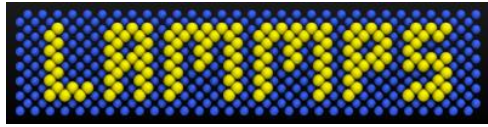


- Step 3 – compare overall system to analytical solution (if possible)



Hands on Example using ARCHER2

- Step 3 – compare overall system to analytical solution (if possible)



#Load Modules

```
module load other-software
module load cpl-openfoam
source $FOAM_CPL_APP/SOURCEME.sh
module load cpl-lammps
```

#copy examples

```
cd ${HOME}/home/work/
cp -r $CPL_PATH/examples/LAMMPS_OPENFOAM .
cd LAMMPS_OPENFOAM
```

Submit job

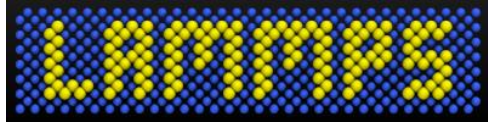
```
sbatch example_archer2.bat
```

A screenshot of a terminal window showing the execution of the commands listed on the left. The terminal title is "es205@DESKTOP-L6KBISU: ~" and the prompt is "es205@DESKTOP-L6KBISU:~\$". The terminal content is currently blank, indicating the commands have been entered but not yet executed or the output is not visible.

```
es205@DESKTOP-L6KBISU: ~
es205@DESKTOP-L6KBISU:~$
```

Hands on Example using ARCHER2

- Step 3 – compare overall system to analytical solution (if possible)



#Getting Plotting prerequisites

```
module load cray-python
python -m venv --system-site-packages
${HOME}/home/work/}/myvenv
source ${HOME}/home/work/}/myvenv/bin/activate
python -m pip install -U pip
python -m pip install -U matplotlib pyqt5
git clone github.com:edwardsmith999/pyDataView.git
```

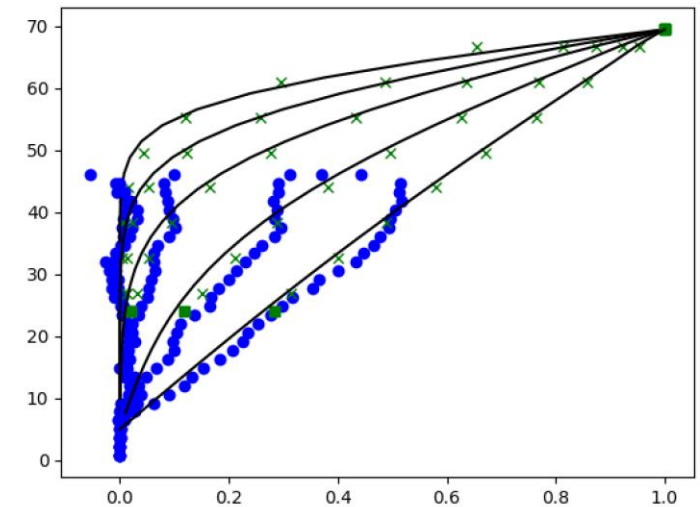
#Then edit

```
vi plot_coupled.py
```

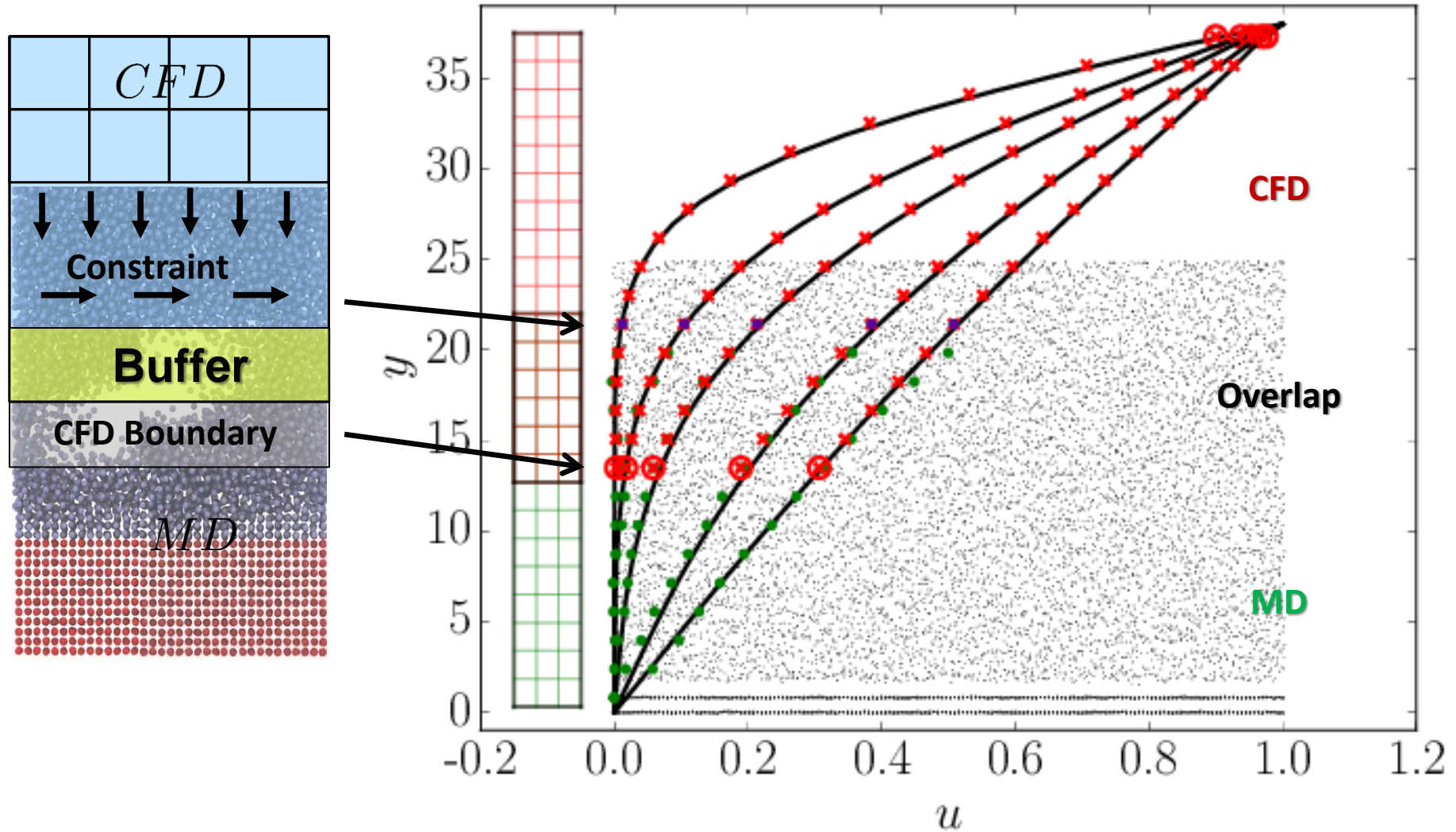
#To change path to where pyDataView is cloned

```
ppdir = './pyDataView/'
sys.path.append(ppdir)
import postproclib as ppl
```

```
# Be sure to have x
# forwarding on (ssh -X)
python plot_coupled.py
```



This Can be Improved With Bigger MD Domain



Summary

- Unsteady Couette Flow is the canonical test case for coupling
 - Wall driven and we have an analytical solution
 - Requires two-way coupling to be working in order to get correct agreement
- Shown the example of OpenFOAM and LAMMPS on ARCHER2
 - Starting from two mock or dummy scripts to get geometry
 - Then coupled each code with a dummy and test Couette flow
 - Finally coupled directly and validate with analytical solution
- The same workflow should be applied to all new cases developed
 - Tests should be designed and automated for both mock-code combinations
 - It is almost impossible to debug errors in the full coupled case