Coupling
The Discrete Element Method
to
Computational Fluid Dynamics

By
Edward Smith
and
Catherine O’Sullivan
Anderson and Jackson (1967)

Assumes
Cell >> particle

\[ \mathcal{E} \]

\[ F^C = \sum_i f^C_i \]

Porosity
Force
Anderson and Jackson (1967)

- A Porous form of the Navier-Stokes Equations

\[
\frac{\partial \rho \varepsilon u}{\partial t} + \nabla \cdot (\rho \varepsilon u) = -\varepsilon \nabla P + \nabla \cdot (\varepsilon \tau) + \varepsilon \rho g - F^C
\]

Assumes Cell >> particle

\(F^C = \sum_i f_i^C\)

- Force on particles: Added mass, Lubrication, Buoyancy, Drag forces (with empirical correlations), etc

\[m_i \ddot{r}_i = \sum_{i,j} f_{ij} + f_i^C\]
A Tale of Two Grids

CFD solved on grid

CPL library

Particles averaged on grid

\[ F^C = \sum_i f_i^C + \varepsilon \]
A Tale of Two Grids

Any CFD e.g. OpenFOAM

CPL library

libcpl.so

Any DEM e.g. LAMMPS

Two codes sharing a communicator

mpiexec -n 4 ./cfd.exe : -n 48 ./dem.exe
Developed for linking of particle and continuum code

Previous focus on scalability (for supercomputers)

Current focus on reliability and ease of use

Maintains separate scope of each code by linking shared library

- Particle only
- Particle Coupled

(a) Parallel speedup of the MD solver only (x), coupled code (o) against the ideal speedup (---)
• Testing the basic units of code

```cpp
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);
}
```
Validation (LAMMPS)

- Particle bouncing on a wall
Validation (OpenFOAM)

- Particle moving through a fluid
Summary

- We are coupling two separate codes to run together
  - Computational Fluid Dynamics
  - Discrete Element Method

- 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