4 Technical Report (publishable)

Abstract

In this report we detail the development, deployment and testing of a coupled model of LAMMPS and OpenFOAM on ARCHER. In line with the objectives of the project, we ported a coupled code using CPL library to ARCHER, developing automated testing to check data exchange and scaling tests up to 10,000 cores. Next, we extended the capabilities of CPL library to include the fully overlapping domains required for geomechanics, including topological and communications tests and a range of LAMMPS-OpenFOAM validation cases. A range of documentation was developed with scripts, to allow a novice user to download and build in a single command on ARCHER, as well as more detailed instructions on the API with example usage so more advanced users can develop their own functions. This required careful design of the user interface and consideration of future extension. Finally, scaling and performance checking was performed with serial and parallel optimisation, along with load balancing. In addition to the above work, a whole range of additional functionality was developed including OpenFOAM solver codes, a strategy for poly-dispersed clumps on large parallel systems, a set of extensible field and force objects which can be unit tested and finally a novel mock coupling based approach, which we believe can form the basis of a new way to develop coupled models.

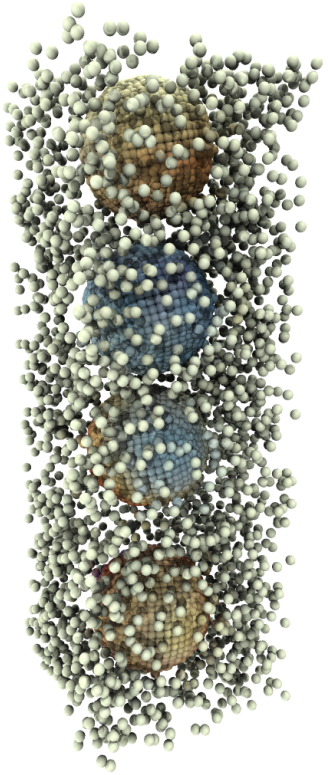


Figure 1: Granular DEM simulation in LAMMPS with particles coloured by overlapping velocity field from CFD simulation in OpenFOAM

Introduction

Combining the discrete element method (DEM) and computational fluid dynamics (CFD) is an essential step in the next generation of geotechnical design tools. However, the problem is complex, requiring the linking of techniques and models from two active areas

of research, each complex with many potential sources of uncertainty. The project has aimed to solve this problem by developing efficient, tested and minimal coupling between the molecular dynamics code LAMMPS and the computational fluid dynamics code OpenFOAM to run simulations on ARCHER. The resultant system will be of use to anyone wanting to fluid-particle systems; the relevant disciplines include geotechnical (civil) engineering, geology, and petroleum engineering. Hitherto this type of problem had not been simulated on ARCHER.

Specific examples of research it is in the process of enabling are:

- Seepage-induced instability including internal erosion. The software developed is currently in use by Dr. Adnan Sufian who is working to improve understanding of the mechanisms whereby fluid flow can induce failure in dams and flood embankments. Water seeps continuously and slowly through embankment dams and flood embankments. If the structures are poorly designed this flowing water can preferentially erode the finer grains in the embankment. This erosion can develop in a progressive manner and ultimately cause embankment failure. Dr. Sufian is developing models using the software where the flow of water through a sample of soil will be simulated. During these simulations the movement of particles will be closely monitored. These simulations will help us better understand the link between particle movement under seepage flow, the stresses on the particles and the flow velocity.
- During earthquakes or during the construction of some embankment structures there can be an increase in the pressure of the water in the void space between grains sufficiently high so as to cause failure. This phenomenon is called liquefaction and it triggered the tragic Aberfan landslide in 1966. Liquefaction also has cause significant damage during the Japanese earthquake in 2011 and researchers from the University of Tokyo (Dr. M. Otsubo working with Prof. R. Kuwano) aim to use the software to improve understanding of liquefaction.
- Researchers at the University of Cardiff are currently seeking funding (from NERC) to support use of the code to simulate segregation of minerals in lava flows. This research would improve the understanding as to how platinum rich layers develop in geological strata and ultimately benefit those interpreting geological data to extra high-value minerals.

Sample input files for simulations relating to each of the following three applications will be included in the project website.

Software Overview

The software aims to combine separate, modular, and tested codes stored on separate repositories. In this way, we avoid the monolithic and static packages typical of previous scientific software in order to develop an open-source framework, linked to github repository which can include the latest bug fixes and changes while allowing the project to scale as more users and developers join. The emphasis is on breaking the problem into separate software packages which can be tested and developed as units. This division of responsibility allows components to be unit tested, creates a new framework for mocking coupled elements and provides Python interfaces to allow rapid coupled development. As shown in Figure 2 the monolithic solution is a bundled OpenFOAM and LAMMPS executable. The first level of division splits these codes into separate executables each linked to a shared library which handles all communication (CPL). In this way, the communication and processor mapping itself can be tested in isolation, before each part of OpenFOAM and LAMMPS can be linked to a mock script for development and testing. The interface between the CPL library and the two codes is handled by application code (APPS), stored on separate repository and designed to provide bespoke functionality for coupling. These APPS use common design patterns from the CPL libraries utilities folder in the form of unit-tested array, force and field objects in order to allow common functionality to be validated and reused for other coupling type simulations (e.g. MD-CFD domain decomposition see Mohamed and Mohamad, 2009).

CPL library

CPL library emerged from from Edward Smith's Phd work (Smith 2014) and was extended via two successful dCSE projects (Smith et al 2012, 2013). CPL aims to provide a minimal set of commands, similar to MPI, which link two codes together. Coupling is achieved through just four key commands,

- CPL_init(ream)
- CPL_Setup(CART_COMM, domain_size, origin, ncells)
- CPL_send(A)
- CPL_recv(A)

The minimum of information is required, the domain size, the number of cells and the Cartesian layout of processors. Everything else is handled by CPL library from this minimal information (although a more complex, lower level interface is possible). Averaging is performed on a uniform grid of cells, a key design decision to simplify coupled domain setup based on the assumption that all complex grids can be mapped to a uniform grid.

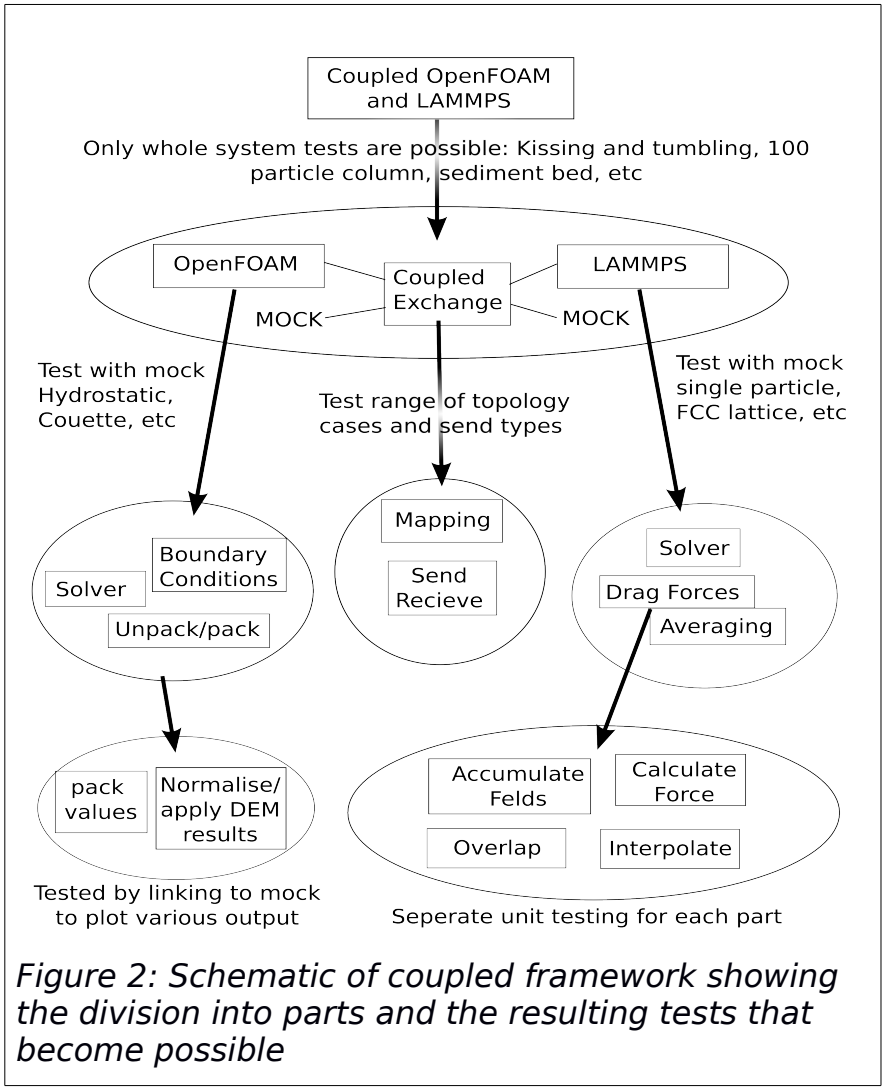
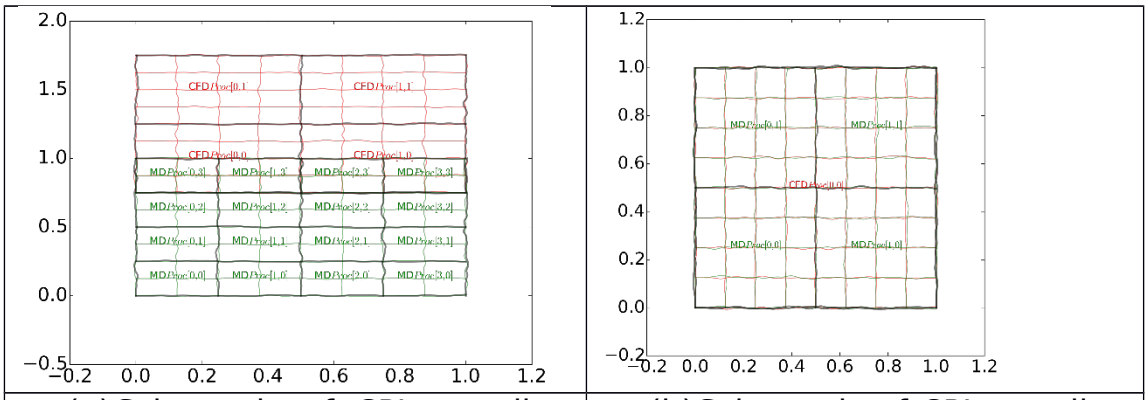


Figure 2: Schematic of coupled framework showing the division into parts and the resulting tests that become possible

The aim of this work was to extend this to the fully overlapping case required for granular mechanics (Fig 3b) with all information exchanged. This required new inputs to switch to the granular case and to allow multiple processors in the y-direction (removing a previous limitation of the software). In addition, this development had to be backward compatible for users of the previous MD-CFD style of coupling. To this end, a new bit encoding system was developed to allow easy swapping of send and receive information types and an input system based on C++ maps.



(a) Schematic of CPL coupling prior to eCSE project

(b) Schematic of CPL coupling enabled by project

Figure 3: Previous and new topologies supported as a result of this project, note y axis is the vertical and horizontal could be x or z.

LAMMPS, GranLAMMPS and the LAMMP_APP

GranLAMMPS is a custom fork of the main LAMMPS repository which allows simulation of soil mechanics laboratory element tests (e.g. Hanley et al.(2014), Huang et al (2014), Shire et al. (2014)). It was developed by Dr. Kevin Hanley working with Dr. Catherine O'Sullivan and their collaborators. Although closed source, the modular nature of the CPL APPS and sockets developed in this project means that the granular branch of LAMMPS can use the developments with minimal adaptation, with the GranLAMMPS branch available to collaborators and developments aiming to be included in the main version of LAMMPS in the future.

LAMMPS works through a series of user defined fixes. These are objects the user can define by inheriting from a base class called fix with a pre-defined structure. These are automatically registered with LAMMPS and instantiated (created when LAMMPS runs). The user can define a range of hook functions with names like post_setup or pre_force, which can be used to inject code at the appropriate location in the overall LAMMPS solver (i.e. after the setup or before the force calculation respectively). The addition code is included in LAMMPS by writing user add on packages which can be included when LAMMPS is complied. Provided LAMMPS has been built with the extra package, they can then be used, often by keywords to switch these on from the user input.

The coupler application has been developed to be applied entirely as a fix, this fits within the code design philosophy of LAMMPS and allows a single line of input to switch coupled simulation or off, with a single command of the form:

```
fix ID group cpl/init region all forcetype X sendtype Y
```

The first two arguments are standard LAMMPS: ID = user-assigned name for the fix group-ID = ID of the group of atoms to apply the fix to The next part, cpl/init, is the "style", i.e. the fixes' name. "region all" specifies that the fix is applied to the entire box in space. forcetype X allows you to specify which constraint force system to use, including

1) test -- A simple test for debugging (sends cell indices)
2) Flekkoy -- stress based coupling
3) Velocity -- applies (U_particle - U_CFD) (2004) 4) Drag -- base drag class for granular type drag forces and extended to a range of the main drag models.

However, this input system has been designed so the user can add anything they want here easily with any arguments cascaded using a C++ arg_map to a user defined extensible force object, as detailed in the tutorial here: *http://www.cpl-library.org/wiki/index.php/Main_Page*. The Field and Force objects have a full unit-testing framework, an extensive range of drag models have been developed and they provide interpolation and overlap libraries.

The sendtype Y specify which data is sent to the CFD solver, which can be any combination of inputs (note they are additive such as a) VEL b) NBIN c) STRESS d) FORCE e) FORCECOEFF f) VOIDRATIO as well as pre-defined collections.

Another major consideration of this project was an efficient method for treating polydisperse systems, i.e. systems with a wide range of particle sizes. After extensive reading, a strategy was developed to use compound spheres or clumps based on multiple particles and a volume penalisation method Tsuji (1993) in order to model polydispersed systems. This is attractive from an HPC perspective as it allows us to use of the highly optimised parallel framework of LAMMPS which would not be possible using single large spheres which may be spread over multiple processes.

OpenFOAM, SediFOAM and the OpenFOAM_APP

A major part of this project involved establishing the correct form of CFD solver to use. The original proposal suggested the use of the Navier Stokes equation in the form given by Tsuji (1993), although many others exist (Xu & Yu 1997; Kafui et al. 2002). After careful literature review, it was deemed that SediFOAM by Sun et al (2016) at https://github.com/xiaoh/sediFoam was the most promising candidate for coupled simulation. Some changes were made to the solver, adapting SediFOAM for use in the low Reynolds number domains of interest for the target application and changing the algorithm to receive all information through CPL library from the

separate LAMMPS code in line with the modular approach enabled by CPL library. SediFOAM has been extensively validated and includes a number of consideration which stabilise the multi-phase solver for OpenFOAM, based on the PhD thesis of Henrik Rusche (2002). The ideas is as follows.

1) Exclude pressure gradient from the momentum prediction step, calculate only a single Jacobi iteration to find a guess of the velocity before the PISO pressure solver is applied to the particle/fluid mixture.

2) The gravity and drag term is applied inside the pressure equation, in a process known as semi-implicit coupling (Issa 1986).

3) The combined fluid/particle pressure equation is used to enforce mass conservation for the mixture and the flux is adjusted instead of the velocity field.

4) Finally the velocity field is obtained from a reconstruction of the flux correction. This approach is used to improve the numerical stability of a granular simulation. However, it was found that the case of Couette flow, the correct solution is not obtained from this method, and a full “Solve” is required in step 1) above instead of a single “Relax” step to get correct results for Couette flow case.

This highlights a common feature of CFD: balancing accuracy and stability of a numerical scheme. It is therefore unclear if this is the right choice of solver for all cases, so the strategy used in this project was to provide software which allows a range of testing tools in the form of mock coupled examples and a socket to allow easy development of other coupled solvers as needed. A few other porous solvers developed during this project are also provided in the OpenFOAM_APP source file and these can be tested for each case to decide suitability.

Deployment

The CPL library software is kept very simple, entirely self contained with no dependencies, requiring only a Fortran compiler for the core code. The C++ wrapper needs to use a compiler which supports the 2011 standard and the Python wrapper is based on Python 2.7 using Numpy and Pytest.

The main deployment challenge comes from compiling the codes which are to be coupled, namely OpenFOAM and LAMMPS, external software packages which we have limited control over but want to allow the latest version from the repository to be used. We therefore aim to keep both OpenFOAM and LAMMPS as close to the repository

version as possible, with every change representing a maintenance burden.

LAMMPS builds fairly quickly and the required additions for coupled granular simulation have been made entirely self-contained using the package system, so building the coupled version of LAMMPS is fairly trivial. A major part of the early work in this project was to make a single USER-CPL module to ensure deployment of the coupled model with LAMMPS as simple as possible. A patch is still required on ARCHER as described below.

OpenFOAM takes over 8 hours to build, this prevents test automation using a continuous integration platform and makes it very time consuming to develop and provide building instructions. The solution developed for ARCHER is to allow patching of an existing OpenFOAM installation with minimal rebuild required.

The reason a patch is essential in both OpenFOAM and LAMMPS on ARCHER, is because we need to run the coupled codes in Multiple Processor Multiple Data (MPMD) mode, with a command of the form,

```
mpiexec -n 64 ./md: -n 8 ./cfd
```

where they share a single `MPI_COMM_WORLD`. Both LAMMPS and OpenFOAM assume they are the only code in the MPI_COMM_WORLD so this must be patched. Our previous solution to this problem, using MPI ports connected two unrelated MPI jobs each with an `MPI_COMM_WORLD`. However, this is not possible on ARCHER as Cray do not support processor spawning or MPI_port. This means that we are back to the MPMD strategy and maintaining a separate patch for every version of both MD and CFD code.

An Anaconda based approach was also developed, which allows the code to be downloaded into a virtual environment which contains all dependencies, including MPI (tested on the local Imperial supercomputer and a BP cluster). This did not work on the Cray based ARCHER where a wrapper is required to run MPI jobs (aprun) and node allocation was found to not work as expected. A solution using the ABI interface module was put on hold as we waited for GCC7.2 which has only recently become available, and is currently being developed.

Singularity (HPC for Docker) was also developed for CPL library itself and some minor scripts. As singularity and Docker are not currently supported on ARCHER, this solutions will be adapted when these become available.

Testing and Validation

One of the main aims of this project was to implement modern software practices such as design to an interface, test driven development, continuous integration and version control. This is made difficult by the nature of MPI programs which require considerable setup before a run, including MPI initialisation and the creation of an appropriate communicator which includes a number of opaque side effects. The use of coupled simulation makes these even more challenging as the setup of two separate codes is required before we can even begin testing. We have approached this by testing CPL library in isolation; then separating off the entire array, field and force system to test in serial; before building a validated range of drag force models on top of this and finally developing a system of coupled mocks which use dummy Python scripts to test the coupled codes feature by feature. In this section, we give a number of examples of how we use the idea of mocks to test the coupled LAMMPS and OpenFOAM, gradually building up a framework which we can have faith in for use in coupled granular simulation.

The starting point is to ensure the validity of CPL library, achieved with basic tests of CPL_send and CPL_recv operations on a wide range of processor topologies for codes in Fortran, C++ and Python, all automated using the Travis CI integration from Github.

A major part of the project was to split off the entire granular drag force implementation into a field and force objects, which allows complex coupling based serial operations to be developed using test driven development, resulting in a comprehensive unit testing framework. This includes both a calculation of sphere-cube overlap, interpolation within cells (which requires communication to exchange halo cells) and a wide range of other functionality all tested using googletest, an example of some of the unit-tests from the googletest output are shown below:

```
[----------] 27 tests from CPL_Force_Test
[ RUN      ] CPL_Force_Test.test_CPL_array_size
[ RUN      ] CPL_Force_Test.test_CPL_field_name
[ RUN      ] CPL_Force_Test.test_CPL_field_setters
[ RUN      ] CPL_Force_Test.test_CPL_field_overlap
[ RUN      ] CPL_Force_Test.test_CPL_field_getters
[ RUN      ] CPL_Force_Test.test_CPL_force_constructor
[ RUN      ] CPL_Force_Test.test_CPL_force_get_set_field
[ RUN      ] CPL_Force_Test.test_CPL_get_cell
...
[ RUN      ] CPL_Force_Test.test_CPL_Force_get_force
[ RUN      ] CPL_Force_Test.test_CPL_force_internalfields
[ RUN      ] CPL_Force_Test.test_CPLForce_Drag
```

```
[ RUN      ] CPL_Force_Test.test_CPLForce_Drag_argmap
[ RUN      ] CPL_Force_Test.test_CPLForce_Drag_check_volSumsFsum
[ RUN      ] CPL_Force_Test.test_CPLForce_Drag_check_overlap_field
[ RUN      ] CPL_Force_Test.test_Granular_CPL_inhereted
[ RUN      ] CPL_Force_Test.test_Granular_CPL_forces
[  PASSED  ] 27 tests.
```

Memory allocation and leak checking is performed as part of the test suite using Valgrind to check all new development both locally and on Travis CI.

The range of possible drag models in coupled DEM is extensive, including Ergun, Di Felice, Tang, Tenneti and BVK (Ergun 1952; Di Felice 1994; Tang et al. 2014; Tenneti et al. 2011; Beetstra et al. 2007). These have all been implemented using the CPL_force object framework and are automatically tested by comparison to an existing implementation (https://github.com/chrisk314/drag-utils).

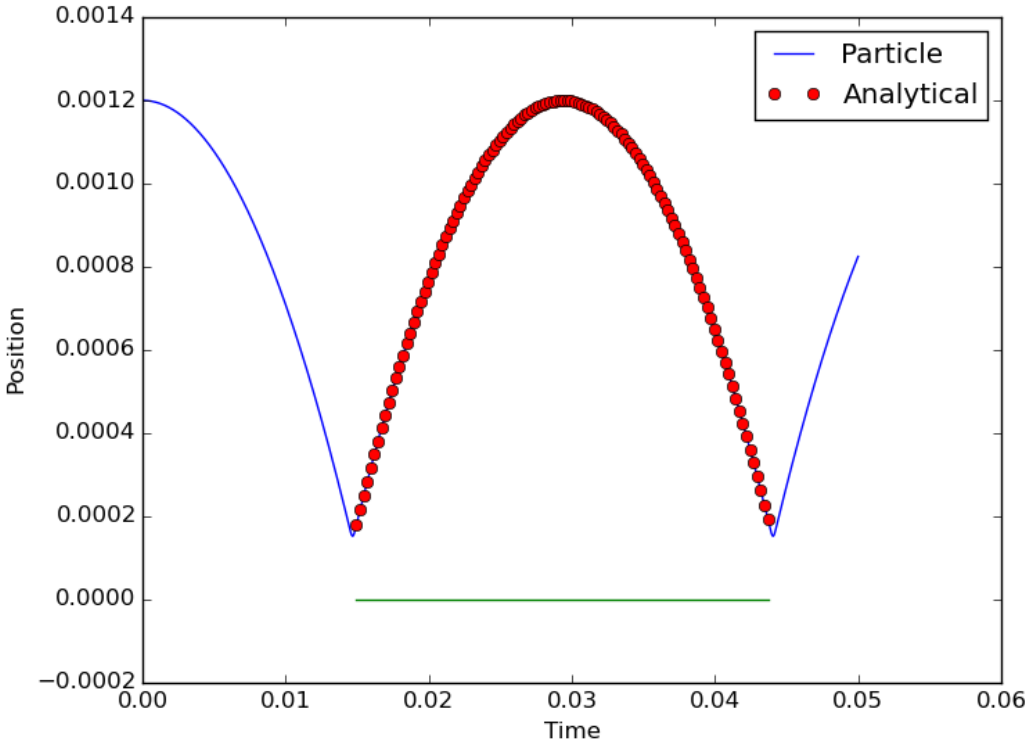


Figure 4: Bouncing LAMMPS particle with force field from coupled exchange compared to analytical solution and regression test

Validation of LAMMPS with coupling is done through a range of test cases. The test cases suggested in the original proposal were found to be too complex and instead a range of simpler cases were used, each designed to test one additional aspect of the coupled

simulation. Starting from a single particle in a periodic box, we apply a field obtained through the coupler, then add the same single particle bouncing on a wall, then the particle interacting with a fluid field through drag and buoyancy, then flow over an FCC lattice. Each case has analytical or other solutions we can compare to, for example the bouncing ball, shown in Figure 4, can be tested against constant acceleration equations for most of the trajectory, while the (non-linear) interaction with the wall is tested with a regression test to a previous simulation.

The coupling to a dummy code can then test spatially and temporally varying force fields; probe different processor topologies and the communication exchange within LAMMPS; ensure the correct application of the drag force and the correct application of dynamics within LAMMPS. In this way, the coupled dummy scripts can be seen to work like the concept of a mock class, allowing values to be injected and exploring how the code responds, thereby separating the problem into its various parts.

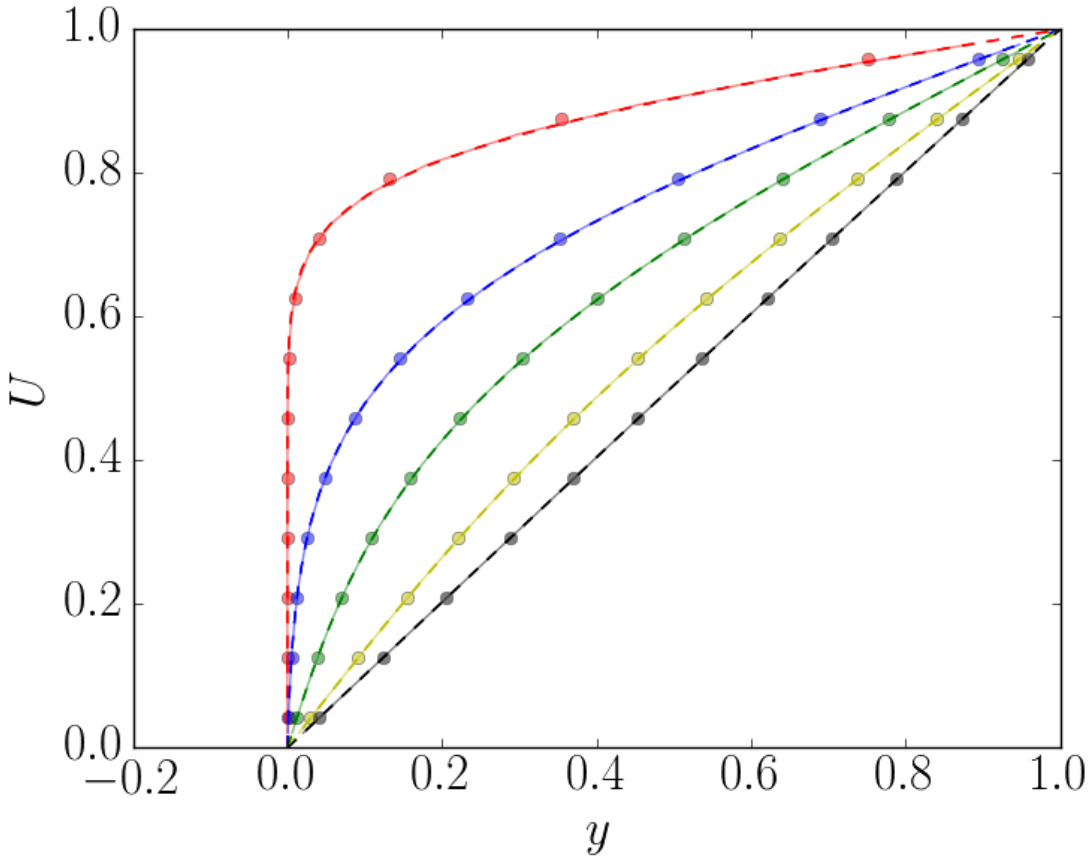


Figure 5: SediFOAM simulation (circles) compared to the Couette flow analytical solution (lines) with L_2 norm assertion test.

The OpenFOAM simulations are based on SediFOAM (Sun et al , 2016), a complex multi-phase solver based on years of research in two fluid modelling and additions for turbulent granular flows. To ensure this is valid for our problem, we start with a simple boundary enforced hydrostatic gradient which is the basis for much of the work in sediment flows and is used here to ensure the pressure solver is working as expected. Another canonical test case with an analytical solution is Couette flow, Figure 5, which tests the diffusive part of the fluid simulation is working as expected. We start with the case with zero porosity, which is coupled to a mock Python script and matched to the analytical solution with an assertion based on the L_2 norm error. Next, to test the effect of porosity and drag force on the fluid solver, a region of constraint force and zero porosity is applied by a mock Python code, as shown in Figure 6. The part of the domain above this can again be matched to the Couette analytical solution, validating that the drag force works as expected.

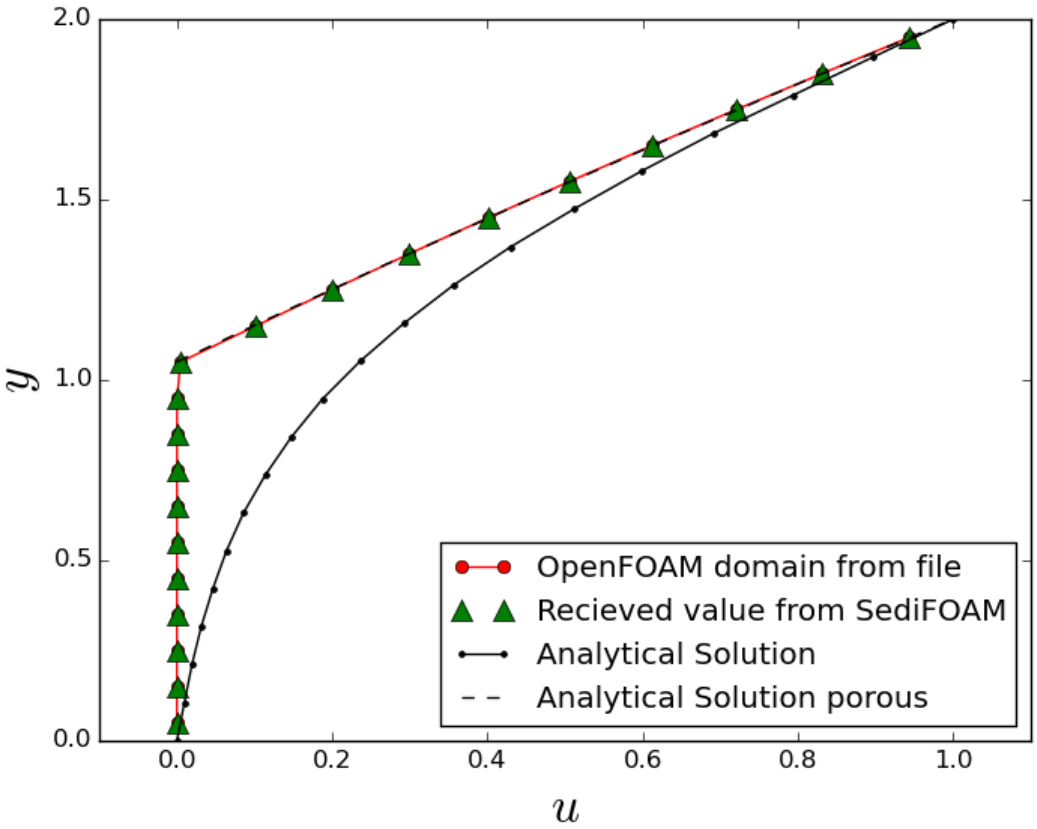
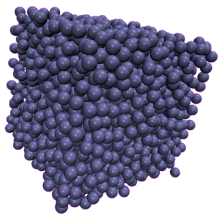


Figure 6: Couette flow with drag force and porosity applied in a local region with evolution checked against analytical solution by python dummy script.

The next model looks at applying a porosity that is representative of a DEM simulation, with the MOCK code reproducing a controlled FCC lattice. As a DEM simulation would be averaged for this case and pass a uniform field, the mock field is identical and we can therefore developed the CFD code in isolation and ensure this works as expected.



```
import numpy as np
from mpi4py import MPI
from cplpy import CPL

#initialise MPI
comm = MPI.COMM_WORLD
CPL = CPL()
MD_COMM = CPL.init(CPL.MD_REALM)
CPL.setup_md(MD_COMM.Create_cart([1, 1, 1]),
                    xyzL=[1., 1., 1.], xyz_orig=[0., 0., 0.])

#Setup send and recv buffers
recvbuf, sendbuf = CPL.get_arrays(recv_size=9, send_size=8)

#FCC properties
dp = 0.01; phi = 0.74; mu = 1e-3; K = 435.
cvol =CPL.get("dx")*CPL.get("dy")*CPL.get("dz")
cnp = cvol*phi/((np.pi/6)*dp**3); cCd = cnp*3*np.pi*mu*dp*K

#Main time loop
for time in range(501):
    # Recv data [Ux,Uy,Uz,dPx,dPy, dPz, dTaux, dTauy, dTauz]
    recvbuf, ierr = CPL.recv(recvbuf)
    Uy = recvbuf[1,:,:,:]
    gradPy = recvbuf[4,:,:,:]

    # Send data [Ux, Uy, Uz, Fx, Fy, Fz, Cd, e]
    sendbuf[4,:,:,:] = -cCd*Uy + (cvol*phi)*gradPy
    sendbuf[6,:,:,:] = cCd*np.ones_like(Uy)
    sendbuf[7,:,:,:] = 1 - phi
    CPL.send(sendbuf)

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

Figure 7: Snippet of Python mock code to be linked to OpenFOAM to mock LAMMPS FCC lattice

Once the CFD code has been tested in isolation, we setup an FCC lattice in LAMMPS and apply the corresponding information from a dummy script modelling the CFD code, in order to debug the DEM

code separately. The two validated codes can then be connected directly and any problems resulting from this final “integration test” therefore isolate the problem and tweaks to the input system applied to fix this.

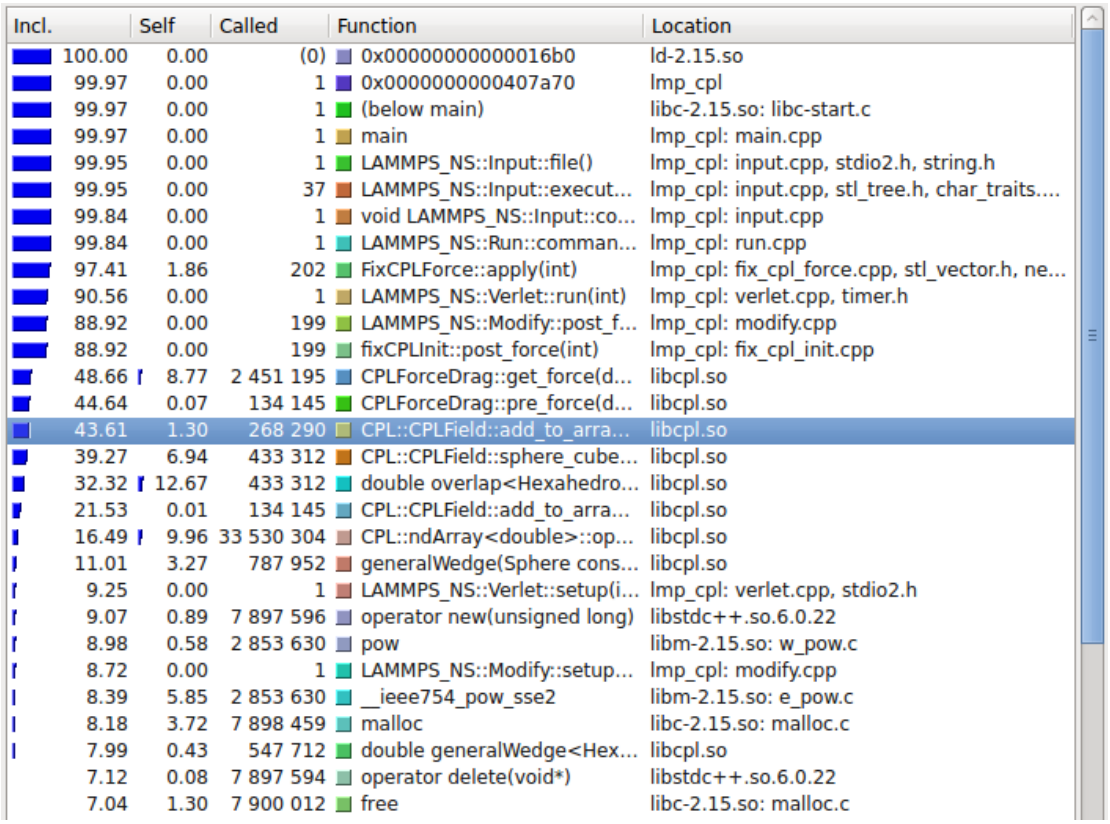
In this way, the CPL framework provides an elegant way of developing software with integrated testing and allows two complex codes to be developed separately. This also allows the automation of the test of each code against the various releases, so any errors introduced by future changes to OpenFOAM, LAMMPS, CPL library (as well as upstream dependencies such as MPI) can be isolated to one or other code.

Scaling

Choosing a meaningful metric for scaling of a coupled simulation is not a trivial problem. As we are linking two existing codes, scaling is limited to the worst of these codes. The DEM/MD code is often the rate limiting step and in the previous dCSE reports (Smith et al 2012 and Smith et al 2013) we simply showed that scaling of a coupled MD code was comparable to an uncoupled case. However, a good load balancing strategy should prevent the DEM being a bottleneck; so we consider first the relative cost of both codes to establish the ratio of system sizes to get good performance. We have then developed a custom framework to model the scaling of the CPL library software developed during this project.

Serial Optimisation

Before considering a parallel run on ARCHER, it is essential that the serial efficiency of any newly developed code is optimised. This is achieved by profiling on a local compute using Valgrind's cachegrind, with the output shown here:



Incl.	Self	Called	Function	Location
100.00	0.00	(0)	0x00000000000016b0	ld-2.15.so
99.97	0.00	1	0x0000000000407a70	lmp_cpl
99.97	0.00	1	(below main)	libc-2.15.so: libc-start.c
99.97	0.00	1	main	lmp_cpl: main.cpp
99.95	0.00	1	LAMMPS_NS::Input::file()	lmp_cpl: input.cpp, stdio2.h, string.h
99.95	0.00	37	LAMMPS_NS::Input::execut...	lmp_cpl: input.cpp, stl_tree.h, char_traits....
99.84	0.00	1	void LAMMPS_NS::Input::co...	lmp_cpl: input.cpp
99.84	0.00	1	LAMMPS_NS::Run::comman...	lmp_cpl: run.cpp
97.41	1.86	202	FixCPLForce::apply(int)	lmp_cpl: fix_cpl_force.cpp, stl_vector.h, ne...
90.56	0.00	1	LAMMPS_NS::Verlet::run(int)	lmp_cpl: verlet.cpp, timer.h
88.92	0.00	199	LAMMPS_NS::Modify::post_f...	lmp_cpl: modify.cpp
88.92	0.00	199	fixCPLInit::post_force(int)	lmp_cpl: fix_cpl_init.cpp
48.66	8.77	2 451 195	CPLForceDrag::get_force(d...	libcpl.so
44.64	0.07	134 145	CPLForceDrag::pre_force(d...	libcpl.so
43.61	1.30	268 290	CPL::CPLField::add_to_arra...	libcpl.so
39.27	6.94	433 312	CPL::CPLField::sphere_cube...	libcpl.so
32.32	12.67	433 312	double overlap<Hexahedro...	libcpl.so
21.53	0.01	134 145	CPL::CPLField::add_to_arra...	libcpl.so
16.49	9.96	33 530 304	CPL::ndArray<double>::op...	libcpl.so
11.01	3.27	787 952	generalWedge(Sphere cons...	libcpl.so
9.25	0.00	1	LAMMPS_NS::Verlet::setup(i...	lmp_cpl: verlet.cpp, stdio2.h
9.07	0.89	7 897 596	operator new(unsigned long)	libstdc++.so.6.0.22
8.98	0.58	2 853 630	pow	libm-2.15.so: w_pow.c
8.72	0.00	1	LAMMPS_NS::Modify::setup...	lmp_cpl: modify.cpp
8.39	5.85	2 853 630	__ieee754_pow_sse2	libm-2.15.so: e_pow.c
8.18	3.72	7 898 459	malloc	libc-2.15.so: malloc.c
7.99	0.43	547 712	double generalWedge<Hex...	libcpl.so
7.12	0.08	7 897 594	operator delete(void*)	libstdc++.so.6.0.22
7.04	1.30	7 900 012	free	libc-2.15.so: malloc.c

Figure 8: Output from cachegrind coupled LAMMPS case

It is clear from Figure 8 that get_force and pre_force, both new routines developed in this project, dominate the calculation. Given the extensive calculation required for the intersection of a sphere/cube, it is not surprising that this operation is expensive (see self time of overlap<hexahderon> in Figure 8). The overlap calculation itself is developed for the most general possible case and uses the library developed by (http://dx.doi.org/10.1016/j.jcp.2016.02.003), a header only C++ library. A range of unit tests were written to ensure the outputs of this library gave the expected results before incorporation into the CPL_field class described above.

Based on the profiling, a range of optimisations were undertaken. In order to accelerate the simulation, only spheres which are within a radius of the cell's edge are considered for overlap calculation, with the whole spherical volume simply added to the cell if not. Spheres which are only near one surface are calculated using the spherical cap calculation. Only the very rare cases of spheres located near the edge or corner of a cell are then passed to the fully general overlap library calculation. These changes resulted in a speedup of orders of magnitude. The accelerated approach detailed here is then checked against the overlap library for millions of random particle positions

to ensure this accelerated code gives the same results as the full overlap calculation. In addition, memory allocations were greatly reduced through the use of pointers to allow a further speed up in the various code developments.

Serial Scaling

We measured ratio of OpenFOAM and LAMMPS codes as a function of number of cells or particles respectively in a representative simulation (hydrostatic for OpenFOAM, FCC lattice for LAMMPS). The serial scaling suggests that designing systems with approximately two hundred particles per cell will give good load balancing.

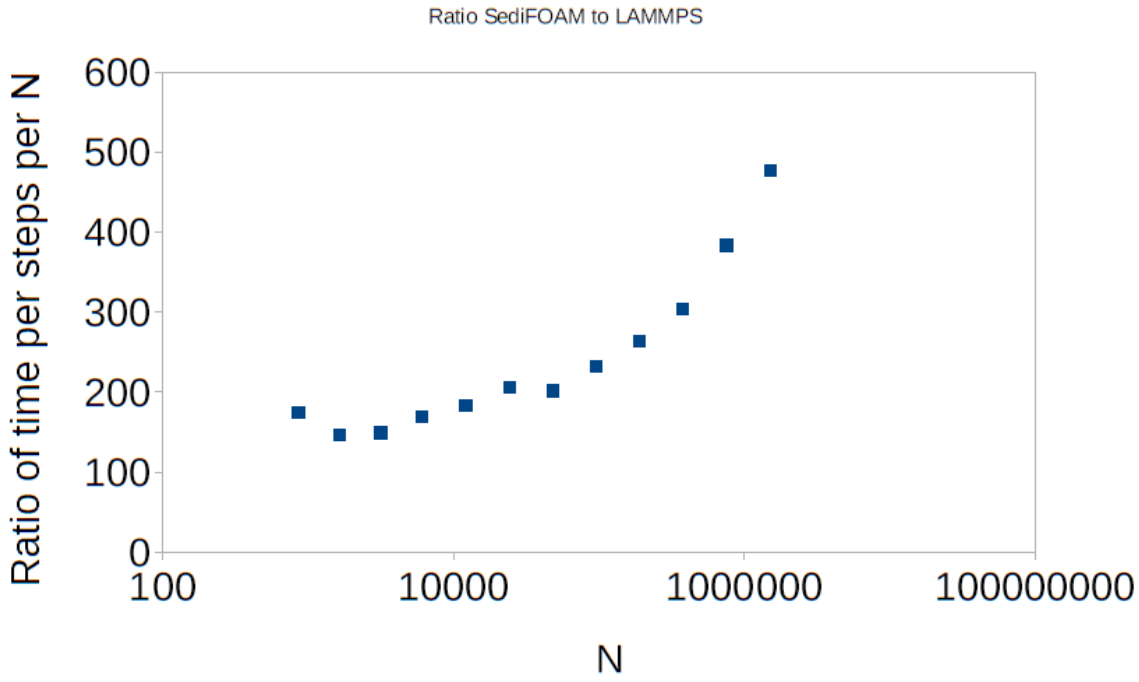


Figure 9: Ratio of calculation time of SediFOAM to LAMMPS as a function of number of cells or particles

Parallel Scaling

The basic premise of CPL library is to set up only local mapping between processes which overlap physically, using a mapping set up by MPI_graph. Each CFD processor receives data from one or more DEM processor, with MPI_Wait used to hold until all overlapping information has arrived before unpacking and returning the data to the user. As all communication is local, there is no expected bottleneck to good scaling.

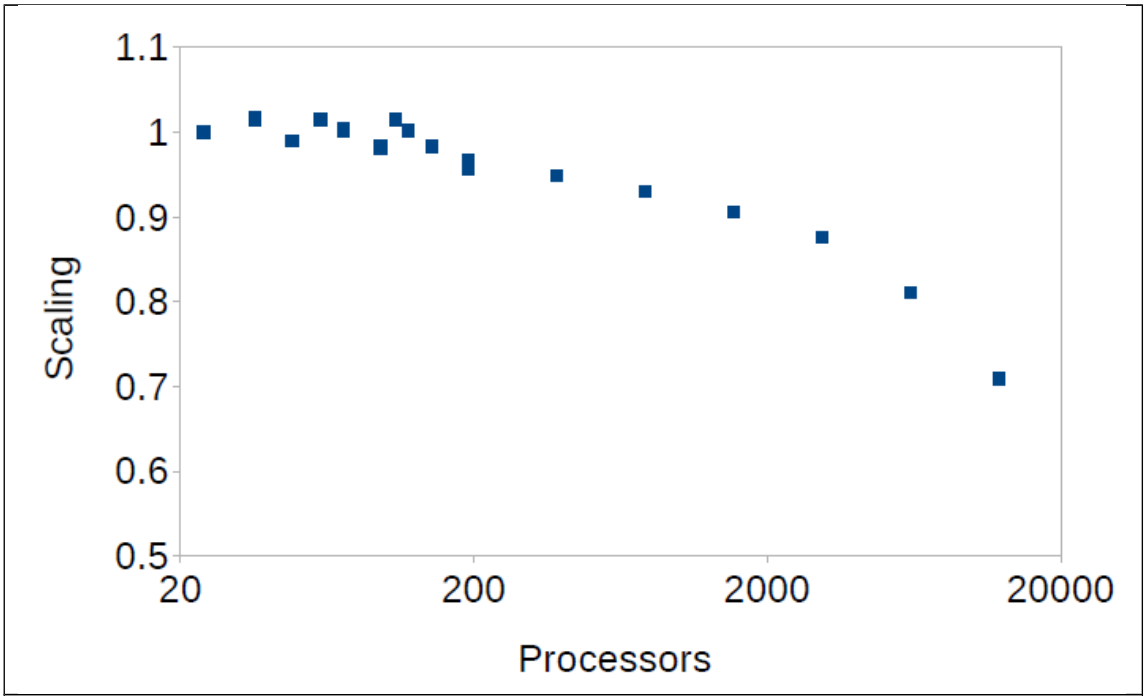


Figure 10: Weak scaling of CPL library with both codes using the same number of processors

However, we want to ensure that the mapping scales well in coupling of any two coupled codes for any situation. A parallel scaling tests is developed which uses a minimal Fortran script compiled into two executables flagged as CFD and DEM and this is run with both connected as part of a coupled simulation. In each case, all coupled communication is local to the overlapping processors, each of which always has 60^3 cells (weak scaling). The arbitrary calculation sends three values for each of the 60^3 cell on each processor, receives the same volume of data and then checks it is correct, before performing an arbitrary calculation. This is repeated twenty times for each run. The smallest size uses 24 processes in both the CFD and DEM codes. The system is then scaled up to 5016 processes per code (10,032 cores in total) and the time taken is calculated. As the actual calculation is identical each time on a given processor, the coupled communication is the only change and we obtain a good insight to the scaling of the system, shown in Figure 10. We use the Craypat tool on ARCHER to check overhead and costs due to parallel communication, shown in Table 1. It is seen that the majority of the time is spent waiting for the messages to arrive. All broadcast and barrier communications are in the setup so can be ignored. Packaging and unpacking the data is seen to take about 5% of the time in CPL send and CPL_recv respectivly. This suggests that coupled scaling is working as expected and, given the large amounts of data sent, the scaling of 70% at 10,000 cores seems reasonable. As the data is used directly after CPL_recv, non-blocking communication are not possible

without reorganising of the coupling algorithm, a possible future consideration.

Samp%	Samp	Imb. Samp	Imb. Samp%	Group Function
DEM code				
41.00%	2144.1	--	--	**MPI**
22.50%	1176.3	221.7	16.00%	mpi_waitall
7.50%	394	193	33.20%	MPI_SSEND
5.60%	294.8	89.2	23.50%	MPI_BARRIER
4.70%	246.6	184.4	43.20%	mpi_bcast
28.20%	1473.9	--	--	**USER**
9.80%	511.2	50.8	9.10%	__stuff_MOD_create_array
8.40%	438	96	18.20%	__stuff_MOD_check_array
4.90%	258.4	43.6	14.60%	__coupler_MOD_cpl_recv
4.40%	229.2	40.8	15.30%	__coupler_MOD_cpl_send
CFD code				
Samp%	Samp	Imb. Samp	Imb. Samp%	Group Function
41.40%	2125.5	--	--	**USER**
19.70%	1010.6	604.4	37.80%	__stuff_MOD_check_array
10.80%	554.8	53.2	8.90%	__stuff_MOD_create_array
5.30%	273.3	38.7	12.50%	__coupler_MOD_cpl_recv
5.00%	255.3	51.7	17.00%	__coupler_MOD_cpl_send
33.00%	1694.2	--	--	**MPI**
14.70%	756.7	218.3	22.60%	mpi_waitall
8.80%	451.9	168.1	27.40%	MPI_SSEND
5.90%	303.9	99.1	24.90%	MPI_BARRIER
2.70%	140.3	286.7	67.80%	mpi_bcast

Table 1: Cray pat profile of parallel code

Conclusions

In this work, OpenFOAM was coupled to LAMMPS using CPL library with the code deployed on ARCHER. The development included an extension to allow data exchange with fully overlapping domains; design of a modular, extensible and unit-tested drag force framework for granular systems; scaling studies of both the CPL library and LAMMPS-OpenFOAM; documentation and interface design for use by both novice users and programmers; deployment on ARCHER using a single script and Anaconda packages; as well as the development of a modular framework which facilitates testing of components and mocking of coupled runs.

The developed software contains many tests and the aim was to develop validated building blocks which could be used to construct coupled simulation projects. As an instability in either code is very difficult to predict and almost impossible to debug in a monolithic coupled executable, we focused on splitting the software and designing tools to probe the coupled problem. By making testing and coupled mock scripts an integral part of the development process, we hope to develop more reliable coupled software and provide easier deployment on HPC platforms.

Acknowledgement

“This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk)” Dr. Adnan Sufian funded via EPSRC grant EP/P010393/1 provided invaluable assistance on this project.

References

- Hanley, K.J., O’Sullivan, C. ,and Huang, X.(2014) “Particle-scale mechanics of sand crushing in compression and shearing using DEM” Soils and Foundations, 55(5), pp 1100–1112, doi:10.1016/j.sandf.2015.09.011
- Huang, X., O’Sullivan, C., Hanley, K. J., Kwok, C.Y. (2014) “DEM Analysis of the State Parameter”, Géotechnique 64(12) 954-965 DOI 10.1680/geot./14-P-013
- Shire, T.; O’Sullivan, C. ; Fannin, R.J.; Hanley, K. (2014) “Fabric and effective stress distribution in internally unstable soils” ASCE Journal of Geotechnical and Geoenvironmental Engineering, 140(12) DOI 10.1061/ (ASCE)GT.1943-5606.0001184
- Smith, E. R. , Heyes, D. M. , Dini, D. and Zaki, T. A. (2015) “A localized momentum constraint for nonequilibrium molecular dynamics simulations” J. Chem. Phys. 142, 074110 (2015) [hdl.handle.net/10044/1/21849]
- Smith, E. R. (2014) On the coupling of molecular dynamics to continuum computational fluid dynamics PhD Thesis Imperial College London
- Smith E, Anton L, (2012), Scalable coupling of Molecular Dynamics (MD) and Direct Numerical Simulation (DNS) of multi-scale flows, http://www.hector.ac.uk/cse/distributedcse/reports/transflow01 /transflow01.pdf (dCSE report)
- Smith E, Trevelyan D, Zaki T, 2013, Scalable coupling of Molecular Dynamics (MD) and Direct Numerical Simulation (DNS) of Multi-scale Flows — Part 2

http://www.hector.ac.uk/cse/distributedcse/reports/transflow02/transflow02.pdf. (dCSE report)

- Tsuji, Y., T. Kawaguchi, and T. Tanaka (1993). Discrete particle simulation of two dimensional fluidized bed. Powder Technology 77, 79-87.
- Xu, B. H. & Yu, A. B. (1997). Numerical simulation of the gas solid flow in a fluidized bed by combining discrete particle method with computational fluid dynamics. Chem. Eng. Sci. 52, 2785–2809
- R. Sun and H. Xiao. 'SediFoam: A general-purpose, open-source CFD-DEM solver for particle-laden flows with emphasis on sediment transport'. Computers and Geosciences, 89, 207-219, 2016. DOI:10.1016/j.cageo.2016.01.011
- Mohamed, K. M. & Mohamad, A. A. 2009 A review of the development of hybrid atomistic-continuum methods for dense fluids. Microfluidics and Nanofluidics 8, 283.
- Issa, Raad. (1986). Solution of the Implicit Discretized Fluid Flow Equations by Operator Splitting. Journal of Computational Physics. 62. 10.1016/0021-9991(86)90099-9.
- Rushe, H. (2002) Computational Fluid Dynamics of Dispersed Two-Phase Flows at High Phase Fractions PhD Thesis Imperial College London
- Kafui, K.D., Thornton, C. & Adams, M.J., 2002. Discrete particle-continuum fluid modelling of gas-solid fluidised beds. *Chemical Engineering Science*, 57(13), pp.2395–2410.

- Xu, B.H. & Yu, a. B., 1997. Numerical simulation of the gas-solid flow in a fluidized bed by combining discrete particle method with computational fluid dynamics. *Chemical Engineering Science*, 52(16), pp.2785–2809. Available at: http://www.sciencedirect.com/science/article/pii/S000925099700081X.

- Beetstra, R., van der Hoef, M.A. & Kuipers, J.A.M., 2007. Drag force of intermediate Reynolds number flow past mono- and bidisperse arrays of spheres. *AIChE Journal*, 53(2), pp.489–501. Available at: http://doi.wiley.com/10.1002/aic.11065.

- Ergun, S., 1952. Fluid Flow Through Packed Columns. *Journal of Chemical Engineering Progress*, 48(2), pp.89–94.

- Di Felice, R.D., 1994. The voidage function for fluid-particle interaction systems. *Int. J. Multiphase Flow*, 20(1), pp.153–159.
- Tang, Y. et al., 2014. A methodology for highly accurate results of direct numerical simulations: Drag force in dense gas-solid flows at intermediate Reynolds number. *International Journal of Multiphase Flow*, 62, pp.73–86.
- Tenneti, S., Garg, R. & Subramaniam, S., 2011. Drag law for monodisperse gas–solid systems using particle-resolved direct numerical simulation of flow past fixed assemblies of spheres. *International Journal of Multiphase Flow*, 37(9), pp.1072–1092. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0301932211001170.