CPL library – a minimal framework for coupled particle and continuum simulation

Smith E. R.\textsuperscript{1,2,*}, Trevelyan D. J.\textsuperscript{2}, Ramos-Fernandez E.\textsuperscript{2}, Sufian, A\textsuperscript{3}, O’Sullivan C.\textsuperscript{3}, D. Dini\textsuperscript{2}

Abstract

We present an open-source library for coupling particle codes, such as molecular dynamics (MD) or the discrete element method (DEM), and grid based computational fluid dynamics (CFD). The application is focused on domain decomposition coupling, where a particle and continuum software model different parts of a single simulation domain with information exchange. This focus allows a simple library to be developed, with core mapping and communication handled by just four functions. Emphasis is on scaling on supercomputers, a tested cross-language library, deployment with containers and well-documented simple examples. Building on this core, a template is provided to facilitate the user development of common features for coupling, such as averaging routines and functions to apply constraint forces. The interface code for LAMMPS and OpenFOAM is provided to both include molecular detail in a continuum solver and model fluids flowing through a granular system. Two novel development features are highlighted which will be useful in the development of the next generation of multi-scale software: \textit{i}) The division of coupled code into a smaller blocks with testing over a range of processor topologies. \textit{ii}) The use of coupled mocking to facilitate coverage of various parts of the code and allow rapid prototyping. These two features aim to help users develop coupled models in a test-driven

\footnote{edward.smith@brunel.ac.uk}
\footnote{\textsuperscript{1}Department of Mechanical and Aerospace Engineering, Brunel University London, Uxbridge, Middlesex UB8 3PH UK}
\footnote{\textsuperscript{2}Mechanical Engineering Department, Imperial College London, South Kensington Campus, SW79AZ, UK}
\footnote{\textsuperscript{3}Civil Engineering Department, Imperial College London, South Kensington Campus, SW79AZ, UK}
manner and focus on the physics of the problem instead of just software development. All presented code is open-source with detailed documentation on the dedicated website (cpl-library.org) permitting useful aspects to be evaluated and adopted in other projects.

**Keywords:** Multiscale, Coupling, Computational Fluid Dynamics, Molecular Dynamics, Discrete Element Method

---

**PROGRAM SUMMARY**

*Program title:* CPL LIBRARY

*Licensing provisions:* GPLv3

*Programming languages:* Fortran/C/C++/Python

*Operating system:* Linux with Docker image based on Ubuntu 16.04, continuous integration testing on Ubuntu 12.04/14.04 with Travis CI and used on CentOS, ARCH and a range of supercomputers

*Number of processors used:* 2 to 10,024

*External routines/libraries:* Message Passing Interface (MPI)

*Nature of problem:* Coupling of particle and continuum software to enable simulations not possible with either code alone. In particular, handling communication and interaction for computational fluid dynamics (CFD) software and either molecular dynamics (MD) or discrete element method (DEM) solvers on high performance computing (HPC) platforms.

*Solution method:* A shared library with a minimal set of functions to enable coupling, together with an entire infrastructure to facilitate development of validated coupled software including minimal Python interface to encourage mock testing, libraries to help develop coupled tools along with a wealth of pre-coupled examples including OpenFOAM, LAMMPS, Flowmol with interactive plotting using wxPython and matplotlib.

*Unusual features:* Minimal interface with simple setup. A CPL_Mocks framework to facilitate debugging and test driven development. Communication established between independent executables using MPI_Open_Port, reducing required changes to core source code. Coupled codes track the git repository with validation through continuous integration testing and deployment on DockerHub. All communication based on MPI_Cart and MPI_Graph so optimisation is possible through MPI implementation on supercomputers.
1. Introduction

Coupled simulations have the potential to revolutionise science. Consider the flow inside a chemical reactor, countless molecules driven by waves of fluid; the turbulent thermodynamics in a boiling vessel; the evaporation at the wall essential to the transfer of heat; the multi-fractal structure of soil soaked in fluid, driving and seeping until the interplay causes the failure of an embankment dam; the flow of blood, cells and viruses driven by a cascade of fluid, but fundamentally governed by details at the molecular scale. It is clear that such simulations are not possible with one modelling paradigm alone; however the combination of different models is often an overwhelmingly difficult problem.

The modelling framework proposed here aims to couple computational fluid dynamics (CFD) to a particle method, either molecular dynamics (MD) or the discrete element method (DEM). Coupling two simulation tools is a very challenging software development problem. CFD simulations require appropriate choice of boundary conditions and numerical scheme, with appropriate stability and monotony considerations. MD and DEM solvers require appropriate integration and force lists, choice of inter-atomic potentials, building of initial configurations and careful equilibration. Coupling brings in the implementation of constraint algorithms, appropriate averaging methods as well as domain termination and particle insertion considerations. Furthermore, both techniques are non-linear, making the origin of problems in either of the two codes very tricky to identify. CPL LIBRARY is a minimal framework to link two codes, with tools to enable the user to systematically test the software, allowing them to isolate and identify problems.

The two main application cases for CPL LIBRARY are shown graphically in Fig. 1, molecular-continuum coupling labelled a) and overlapping DEM and continuum coupling for granular flows labelled b). CPL LIBRARY originally focused on simulations where the domain is decomposed into two parts, as in Fig. 1(a) a CFD and an MD region with an interface between them. This type of coupling is known as domain decomposition, an approach first applied by O’Connell that has been widely adopted over the last two decades. Other names for this type of coupling used in the literature include geometrical coupling, concurrent coupling, or extreme scaling. The two parts of the domains overlap in order to stagger the interface between them and provides a buffer to facilitate coupling. Domain decomposition coupling
Figure 1: Schematic showing typically targeted coupled setup with a) a domain decomposition where a molecular model is only used near the wall and linked along an interface to the continuum shown by the grid while b) is the case of granular mechanics with discrete element method (DEM) and continuum CFD fully overlapping, with the CFD grid shown overlapping the DEM modelling of the particles.
involves the direct exchange of mass, momentum and energy fields between
a particle and continuum simulation, where each piece of software simulates
part of the total domain. Other coupling approaches are possible to link
nano-scale molecular models to the macro-scale continuum, including the
embedding of particle simulations into a continuum solver, or running
lower-level models to establish values for lookup tables or closure relations.
These alternate approaches aim to remove elements of empiricism in higher
level models, for example by calculating quantities in the MD which are
then set as coefficients in the CFD, e.g. viscosity, contact-line dynamics,
surface tension or heat flux. These are outside the focus of CPL LIBRARY,
which solely aims to facilitate the exchange of fields between
continuum and particle simulations in the region in which they overlap.

The approach of exchanging field is, however, applicable to granular me-
chanics simulations of the type shown in Fig. 1(b), where the average of
particle fields influences fluid flow features in the CFD and the resulting drag
is simply applied as a force field to the particle DEM system. For this type
of coupling, the theoretical foundations were set out in Anderson and Jack-
son (1967) and is a commonly applied method for coupled fluid-granular
systems with a large numbers of particles per CFD cell. The basic
premise can be considered a special case of domain-decomposition coupling
in which the overlap between both codes is the whole domain. In this case,
DEM and CFD model the particle and fluid parts of the granular simulation
respectively.

Table 2 gives an overview of three types of existing coupling software.
The first type of software is termed monolithic, which involves pre-packaging
two pieces of software as a single application. This provides an advantage
for the user as it simplifies the installation and allows a combined interface
to both codes. However, a potential disadvantage is that both codes are
adapted from their natural state to bring them together, so the versions are
frozen at the point at which they were coupled and can no longer be used
separately. Examples of monolithic codes typically occur in more applied ap-
lications, where the focus is on providing complete tools to solve engineering
problems. For granular systems, open source examples includes direct link-
ing of LAMMPS-OpenFOAM, adaptations like SediFOAM and mixed
commercial open-source such as CFDEM.

In the second software type, libraries, all coupling is handled through
shared libraries which facilitate information exchange between software. Li-
braries are lightweight and minimise the impact of coupling on either code,
Table showing existing software for coupling grouped into three types.

<table>
<thead>
<tr>
<th>Monolithic</th>
<th>Library</th>
<th>Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>SediFOAM\cite{15}</td>
<td>CPL LIBRARY\cite{19,61}</td>
<td>MUSCLE\cite{52}</td>
</tr>
<tr>
<td>LIGGHTS/CFDEM\cite{16}</td>
<td>MUI\cite{20}</td>
<td>MUSCLE\cite{52}</td>
</tr>
<tr>
<td>OpenFOAM/LAMMPS\cite{14}</td>
<td>MaMiCo\cite{17}</td>
<td>MCT\cite{54}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OASIS3\cite{53}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Precice\cite{50}</td>
</tr>
</tbody>
</table>

The focus of CPL LIBRARY is direct exchange between a fluid solver and a particle solver, linking in as a lightweight library and providing direct information exchange through MPI. The key is to provide ease of implementation and testing, which in turn facilitates the use of CPL LIBRARY to enable the coupling of complex software on supercomputers with minimal knowledge of their individual coding strategy and algorithmic deployment.
amples applications of CPL LIBRARY are then given in section 3 and the four functions required in order to implement coupling are outlined. A range of validation and verification examples as well as automated tests on continuous integration (CI) platforms are discussed in the results (section 4), followed by scaling test on supercomputers. Finally, we summarise the software in section 5.

2. Features

As noted in a recent review [5], existing coupling tools have generally not focused on deployment, optimisation and debugging. CPL LIBRARY aims to address these shortcomings, with one-click deployment using Docker containers and install scripts, along with provision of testing, debugging and visualisation tools. Together with a well documented interface designed for collaborative development, an open-source licence, availability on version control with automated testing and minimal interface, CPL LIBRARY is designed following the key principles of software best practice [6]. This is all built around an internal library designed for efficient communication on overlapping uniform grids. This focus on uniform grids allows us to make use of MPI cartesian communicators for intra-code exchanges and graphs for inter-code communication [6–8]. As MPI is often custom-built for a supercomputer architecture, use of intrinsic MPI cartesian and graph functions ensure communications can potentially be optimal on a given platform by utilising the bespoke MPI implementation.

The process of setting up a coupled simulation with CPL LIBRARY is shown in Fig. 3 starting with the creation of an MPI topology in each code, shown by the grid in Fig 3 a). Next in b), a shared communicator is created with CPL_init(), either by linking (MPI_Open_Port) or splitting (MPI_Comm_Split) existing communicators. The call to CPL_setup in c) then maps overlapping processes between the MPI topologies in each code, a process outlined by data passed to the setup function and the coupler input file. Finally, in d) the sending and receiving of information between the codes becomes as simple as calling CPL_send() and CPL_recv().
Figure 3: Overview of the four key functions provided by CPL LIBRARY and how they are used to setup mapping between two simulations (colour online), with a) creation of processor topology in each code with MPI calls, where the black grid indicates processor topology b) CPL_init() established a shared inter software communicator so information can be exchanged, c) establishing the mapping between processor topologies with CPL_setup and d) exchange of information based on establish topology mapping with CPL_send and CPL_recv.

As highly scalable CFD and MD software employs spatial decomposition for parallelisation[62–64], to ensure efficient scaling to thousands of cores, communication is limited to exchange between only the processes which overlap. No global communications are used after CPL LIBRARY has been setup.

The MPI_Open_Port feature of MPI-2 is used to make linking flexible, so both applications are started independently and retain their own isolated group of processors. This strategy differs from MPI’s multiple program multiple data (MPMD) run mode, where a coupled simulation is started with a command of the form,

```
mpirun -n N1 a.out : -n N2 b.out
```

which launches programs a.out and b.out on N1 and N2 processes respectively, sharing a single N1+N2 processor group known as MPI_Comm_World. Under the MPMD approach, as both a.out and b.out share MPI_Comm_World, any communications in either program can cause unexpected behaviour, with internal messages sent between programs. The source code of both pro-
grams must be edited to prevent this. The MPI_Open_ports strategy used by CPL LIBRARY ensures both codes have a unique MPI_Comm_World so internal messages cannot be sent between coupled codes. The required information to establish the connection between two codes is shared by writing port details to a file which can be copied between directories, hard drives, computers or simply read from the shared file system provided on HPC systems. The MPMD strategy is still supported in CPL LIBRARY, if required through COMM splitting during CPL_init but this requires any reference to MPI_Comm_World to be replaced by editing the source code of both programs. CPL LIBRARY automatically chooses the appropriate run type based on the number of processes found in MPI_Comm_World.

Both LAMMPS and OpenFOAM have been coupled along with a number of other in-house and minimal codes to allow users to mix and match working solutions. Each coupled code can then be linked to any of the other pool of coupled codes, which are designed to be individually tested. The individual testing, using CPLMocks discussed in Section 4.2 help to standardised the interface between codes and allow extension to new applications. The minimal nature of CPL LIBRARY, and the use of MPI_Open_Port prevents the need to patch the internals of OpenFOAM or LAMMPS to allow sharing of MPI communicators. It is therefore possible to keep codes virtually unchanged from the repository version, ensuring all developments follow the style and philosophy of the two linked codes. For LAMMPS, new development are included as fixes, so a USER-CPL package is simply added to the available packages and coupling is turned on or off by adding a single line to the LAMMPS input script. For OpenFOAM, a new top-level solver is written to include all CPL_send and CPL_recv commands together with calls to the OpenFOAM shared libraries, so only this solver needs to be developed.

3. Code

The CPL LIBRARY interface has been refined to four main functions: one to initialise the coupled communicator, one to setup the mapping between the two solvers, one to send and one to receive.

---

4At the time of writing, Cray supercomputers have not adopted the MPI-2 standard’s for dynamic processor allocation used by MPI_Open_Port. As a result, on these platforms an MPMD approach is the only solution.
**Table 1:** A table summarising the four main functions required for coupling: 
\begin{tabular}{|c|c|}
\hline
CPL\textunderscore init(COMM, realm) & CPL\textunderscore init(CART\_COMM, Domainsize, Origin, Ncells) \\
CPL\textunderscore setup\textunderscore cfd(CART\_COMM, Domainsize, Origin, Ncells) & CPL\textunderscore setup\textunderscore md(CART\_COMM, Domainsize, Origin) \\
CPL\textunderscore send(Array4D) & CPL\textunderscore recv(Array4D) \\
\hline
\end{tabular}

Figure 4: A table summarising the four main functions required for coupling: CPL\_init, CPL\_send, CPL\_recv and the two forms of CPL\_setup for MD/DEM or CFD codes.

A wider set of helper functions exist, such as CPL\_get and the various CPL\_mappings functions, all documented on the CPL LIBRARY website. However the minimal set of functions are sufficient to get started with a coupled simulation, as in the example shown in Fig. 5 the simplest additional Python code to couple a CFD solver.
from mpi4py import MPI
from cplpy import CPL

comm = MPI.COMM_WORLD
CPL = CPL()

COMM = CPL.init(CPL.CFD_REALM)
xyzL = [1.0, 1.0, 1.0]
xyz_orig = [0.0, 0.0, 0.0]

npxyz = [1, 1, 1]
ncxyz = [32, 32, 32]

Cart = COMM.Create_cart(npxyz)
CPL.setup_cfd(Cart, xyzL, xyz_orig, ncxyz)

R, S = CPL.get_arrays(
    recv_size=1,
    send_size=2)

for time in range(5):
    R, ierr = CPL.recv(R)
    print("CFD", time, R)
    S[:,:,:] = 2.71828
    CPL.send(S)
CPL.finalize()
MPI.Finalize()

with COUPLER.in of the form

OVERLAP_EXTENTS
  1
  32
  1
  4
  1
  1
  32
CONSTRAINT_INFO
  3
  0
  1
  32
  3
  3
  1
  32
BOUNDARY_EXTENTS
  1
  32
  1
  1
  1
  1
  32

Figure 5: Sample code showing minimal Python example of sending and receiving information between Python CFD and MD scripts with 32 cells in each direction, where the COUPLER.in configuration file is freeform using keyword where each data item for the keywords is specified on a new line, denoted by \n which is the newline character. The OVERLAP_EXTENTS input specifies a four cells overlap in y and all cells in x and z, with the CFD boundary region averaged in the MD set to be the bottom overlap row through BOUNDARY_EXTENTS, and the top row of the overlap defined to be the MD constraint region by CONSTRAINT_INFO, with the first two arguments specifying the constraint details.
The code here follows the steps shown in Figure 3. First, MPI is initialised and an intra software communicator is created, before CPL LIBRARY establishes the inter software communicator. Setup is called with the processor topology information, obtained from a call to MPI_CART_CREATE to specify the processor topology, along with values to define the CFD domain extents and the number of cells in the simulation. For simplicity, only one grid needs to be specified for coupling. All mapping between the overlapping codes is setup at this point, with the COUPLER.in file read to determine the actual cells which overlap between the codes, as shown in Fig. 5. Note a FULL_OVERLAP keyword can be used instead to skip all of this in granular cases. For simplicity, overlapping is specified in terms of cells so that exchanged information can be stored in the four-dimensional arrays, allocated for each processor by a call to CPL.get_arrays. The first dimension is therefore specified by the inputs to this function and all remaining grid dimensions are automatically calculated based on the local processes’ grid extents. The exchange of information is then a case of packing the data on the local processor into the send ‘S‘ array and unpacking from the receive ‘R‘ arrays. Examples in Python and Fortran follow a similar structure and are all included in the CPL LIBRARY examples directory.

For users who are interested in quickly getting started with coupling using OpenFOAM and LAMMPS, two commonly used and very general programs for CFD and MD/DEM respectively, two application (APPS) repositories have been developed which use CPL LIBRARY and extended tools to apply the coupling in LAMMPS and OpenFOAM. These are kept in separate repositories, partly to allow separate testing and partly because the main CPL LIBRARY repository is designed to manage data exchange between arbitrary codes, so specific details for any particular coupled package can be maintained independently with CPL LIBRARY an upstream dependency. A pre-packaged Docker environment is run using the code shown in Fig 6.
docker run --name cplrunOL -it cpllibrary/cpl-openfoam-lammps
# In the Docker Container, MD and CFD coupled example

cd /cpl-library/examples/LAMMPS_OPENFOAM
./run.sh

# Granular example

cd /cpl-library/test/granular/suzuki
./run.sh

Figure 6: Quickstart example showing Docker example of coupled Couette flow using OpenFOAM and LAMMPS

This Docker application includes MPI and CPL LIBRARY, as well as OpenFOAM 3.0.1 and the latest version of LAMMPS together with their interface repositories called CPL_APP_OPENFOAM and CPL_APP_LAMMPS respectively. A Couette flow example is provided for the MD-CFD coupling and a granular column example is provided for the DEM-CFD coupling. In the Couette flow example, a Lennard-Jones fluid (MD) is coupled to an incompressible CFD solver, where a top boundary condition of $U = 1$ is used to model a sliding wall. The two-way coupling ensures that as the fluid is dragged from a stationary state towards a linear profile, the two solvers evolve together in time, with the solution showing good agreement to the analytical solution for time evolving Couette flow. This used run.sh runs using a bash script which creates the parallel OpenFOAM case and runs with a command of the form,

cplexec -c 1 "cplicoFoam -case ./openfoam -parallel" \
  -m 1 "lmp_cpl < lammps.in"

In the granular example, a column of particles in the DEM simulations is coupled to an averaged Navier-Stokes CFD solver, where an upward flow is applied by a constant inlet velocity, resulting in the movement of the particles under the action of the imposed pressure gradient, buoyancy force and drag force. This coupling evolves as the particle position and velocity affect the fluid velocity and pressure profile, and the evolution with time is tracked until an equilibrium condition is reached and the resulting particle displacement and fluid pressure profile show good agreement with the analytical solutions. Again, this is run with a run.sh bash script, which creates the parallel OpenFOAM case and then,

cplexec -c 1 "cplsediFOAM -case openfoam/ -parallel" \
  -m 1 "lmp_cpl < lammps/suzuki.in"
Note that data created in the Docker container would need to be copied to the host operating system to allow post processing and visualisation, or a display attached to allow a GUI to be created and appropriate visualisation tools added to the Docker container.

The output can be visualised using the common post-processing tools for both OpenFOAM and LAMMPS respectively, however these tools will not combine the output for the coupled examples. For coupled visualisation, an open-source tool for uniform grids has been developed. Called Py-DataView[^1], it can be run using,

```
python pyDataView.py -d cpl-library/examples/LAMMPS_OPENFOAM/
```

and is designed to be a lightweight and quick way to plot coupled cases. The sliders allows quick exploration of the four dimensional data and a snapshot, csv file or Python script can be generated for further analysis.
Figure 7: The coupled example from Fig 6 with OpenFOAM and LAMMPS using a velocity constraint, with coupled data visualised using PyDataView. The blue points are from the averaged MD over 200 timesteps and the $x$ and $z$ directions, the red points denotes the constrained region of the MD, the green is the CFD and the cross denotes the CFD boundary condition, obtained from the averaged MD values at the bottom. The top of the MD above the constrained region is a buffer which has fewer molecules so is at lower momentum.

Both granular and molecular-continuum examples are provided with CPL LIBRARY, with examples generally in the interface code application (APPS) repositories. A range of solvers for OpenFOAM are provided, including granular examples adapted from the top-level solver of both CFDEM and SedFOAM. Any changes made to the main CPL LIBRARY repositories, the APPS or LAMMPS will trigger a rebuild of the Docker environment, ensuring this is always up to date. This can also be run in Singularity, which is designed for deployment on supercomputers, using an example on the CPL LIBRARY website. In addition, instructions for compiling on super-
computers are provided with tricks for using pre-compiled versions of codes, Anaconda deployments or patching existing module versions of the codes.

Use of CPL LIBRARY is not restricted to specific software. The provided compiler wrappers, cplf90 and cplc++, make it easy to build applications with CPL LIBRARY included. An application to manage coupled runs, cplexec, is provided to perform a number of consistency checks and handle exceptions. Assuming users have either MD or CFD expertise, the **CPL_Mocks** philosophy aims to make developing coupled code for a new application as simple as possible. This is done by providing the simplest possible case, using scripts located in the examples folder. The three languages supported are Fortran, the most common on supercomputers, C++ used in many scientific software packages and Python which has emerged as the most widely adopted scripting language in recent years, used for many applications in science and beyond. Fig 8 shows an example to compile two minimal code files and sends information between, where both have a similar structure to Fig 5 but written in Fortran and C++ respectively.

```bash
cd cpl-library/examples/minimal_send_recv_mocks
cplf90 minimal_CFD.f90 -o ./CFD
cplc++ minimal_MD.cpp -o ./MD
cplexec -c 1 ./CFD -m 1 ./MD
cplexec -c 1 ./minimal_CFD.py -m 1 ./MD
```

**Figure 8:** Example showing the minimal code for a coupled example, building Fortran and C++ code before running together and running with a Python script

From this simple example, users can see the minimal additions to their code required to implement coupling. However, the main advantage of this split approach is to promote collaboration between groups who are experts in the respective areas. This workflow would include them both agreeing on an interface in advance, using two **CPL_Mocks** scripts coupled together, as a kind of standardisation of an interface, and then they can develop both parts individually using their half of the **CPL_Mocks**. Combining the CFD-MD case together at the end becomes simpler and allows the coupled case to be developed starting from the knowledge that the two parts work individually as expected.
4. Results

In this section, some current applications are outlined, before moving on to the testing and validation of CPL LIBRARY. This includes an overview of a mocking strategy applied to coupling with suggestions for best practice coupled code development. Finally, the scaling of CPL LIBRARY is presented.

4.1. Applications

A range of basic applications are provided on the website:

- A minimal coupled initialisation and setup in Fortran, C++ and Python,
- Sending cell numbers to demonstrate the mapping works as expected.
- Sending a known function, here a sine wave, sent from a set of processors and plotted on the receiving side.
- A visual examples with a minimal user interface to show how real time control of a simulation could be implemented.
- A minimal MD and CFD solver written in Python is shown to highlight how the fully coupled problem would work.

The HTML for the CPL LIBRARY website is included in the repository so these examples remain available, with the code used to generate parts of the website tested automatically as described in the section 4.

Beyond basic examples, CPL LIBRARY has been used in a range of cases, including both laminar and turbulent Couette flow as shown in Fig 9a) and c) respectively, boiling simulation in Fig 9 b), granular mechanics in Fig 9 d), as well as real time visualisation and interactive control of simulation. In addition, it is possible to link to existing libraries such as Keras with Tensorflow so real-time and large datasets can be analysed on the fly during a simulation, allowing machine learning style training of very large fluid simulations with no storage.

4.2. Testing and Mocks

Best practice in software design includes the division of the code into modules to avoid repetition, providing a clear interface to allow automatic testing and validation for every incremental change. This is especially relevant to coupled simulation, when an error may come from one of the coupled codes,
Figure 9: An overview of coupling applications, including a) a laminar Couette flow example with molecular posts on the wall modelled in Flowmol [1] (green), coupled to OpenFOAM (red) and matched to analytical solution (black) with shifted zero location. b) an example of boiling with one-way coupling between Flowmol and a minimal 2D CFD script written in Python, where nucleation occurs naturally in the MD region and the resulting density, velocity and temperature fields are set in a limit region of the CFD and bubbles are created using interface tracking when they leave the MD region, rendered examples of boiling are included on the left. c) a coupled example of turbulence with Flowmol simulating the near-wall region and OpenFOAM the remaining channel (note logarithm axis in wall normal direction y), where isosurfaces of turbulent kinetic energy are shown from a full MD channel simulation [1] to highlight the location of the coupled interface and d) a granular fluid coupling for the Suzuki [6] column test with pressure field from the CFD shown by colours and particles shown from LAMMPS.
the coupling framework itself or emerge from the inherent instability of linking two highly non-linear codes. For this reason, CPL LIBRARY is designed with testing as a key part of the workflow. The first thing to consider is the unit tests which validate CPL LIBRARY itself, before moving on to discussion of more general testing of coupled software using CPL_Mocks.

The majority of testing is performed using pytest; As a high level language this allows us to run a range of different processor topologies using subprocesses. By testing the top-level Python code, the C++ and Fortran code is indirectly tested, as indicated in Fig 10 by the dotted line. The C/C++ bindings are linked to Fortran with compatibility ensured using the ISO_BINDINGS. The bindings for Python extend the C-Fortran interface using explicitly defined ctypes to ensure compatibility. In addition, cross-language coupling tests, e.g. where C++ sends information to Fortran or Python to C, are tested for a range of different processor topologies.

4.2.1. Unit Tests

The basic setup and exchange of CPL LIBRARY itself is unit tested, including, library initialisation and topological setup tests; communication test for the supported range of processor topology, overlap and grid setup options; Valgrind\cite{70} leak testing for memory consistency and minimal examples in all supported languages, which are themselves automatically tested and scraped to generate the website\cite{51} examples.

A range of basic assertion style tests are also included at run time to ensure impossible configurations are prevented, as an incorrect setup has the potential to be costly in supercomputing time. The setup requirements are deliberately stringent, insisting that the two coupled codes are consistent both in domain size and in cell sizes before a run is allowed. In addition, the size of each code is required to be an integer number of cells, so the mapping between the processor topology on both sides can be aligned. A graphical user interface is provided to experiment with different cell and processor topologies, allowing the user to visually establish a valid configuration before spawning a test case to verify this. These strict limitations aim to ensure basic functionality is ensured with the intention that the library will be refactored for more general cases in the future, with correctness ensured by the interface testing suite.

Beyond the testing of the core communication functionality, a range of coupling utilities are provided which are designed be run in serial so as to be independently testable, as shown in Figure\cite{10} These include, a) a four di-
Figure 10: Schematic showing the overall testing strategy for a coupled code, at the lowest level all communication is tested for different processor topologies and the functions used for coupling are tested in isolation. This is then used in each of the coupled codes and the results tested independently using CPL_Mocks which apply controlled inputs and ensure outputs work as expected, before the two validated model are brought together in a fully coupled simulation.
mensional Array class which allows a consistent description between Fortran, C++ and Python; b) a Field class to store continuum uniform values in cells and collect these from particle systems using molecular binning, interpolation and spherical overlap calculations; c) a Force class to apply coupled forces to MD or DEM systems with a range of drag and force models tested against values from the literature and designed to be easily extensible. In other coupling frameworks, this functionality is often taken from one of the linked codes, for example OpenFOAM provides array operations and a particle drag force framework. This limits the range of applicability to the specific code and prevents unit testing of the utilities without building the whole package. This is why CPL LIBRARY is designed to contain its own version of these operations, taking the best features of field operations, inspired by OpenFOAM, and incorporating open source software for calculations such as overlap [71]. By separating these utilities, they can be automatically tested as part of CPL LIBRARY, where low level C++ code is tested using google tests. The framework for applied forces is also designed to be easily extended using inheritance with only minimal code development needed using hook functions, bits of code inserted at pre-set locations such as pre-force or end of step, as inspired by LAMMPS. The process of developing new force models is fully documented on the CPL LIBRARY wiki on the website [51].

4.2.2. CPL_Mocks

The unit tests ensure the core communication and coupling utilities provide the required functionality, the next step is to verify these work when linked into CFD and MD/DEM codes. By its nature, a coupled simulation is difficult to break into independent components, so the concept of mocking becomes important. Mocks are a way of writing a piece of code to replicates the behaviour of part of an interconnected software package, monitoring the inputs and providing controlled outputs to ensure the software works as expected. CPL_Mocks uses this same philosophy in a coupled simulation, where the controlled inputs are provided using coupled information exchange.

In the Couette flow example, the CPL_Mocks test of the MD code (LAMMPS) checks the flow driven by varying applied forces from a mock script, while the CPL_Mocks test of the CFD code (OpenFOAM) checks the flow driven by boundary values provided from a CPL_Mocks script. In the granular column example, a CPL_Mocks test of the DEM code (LAMMPS) checks that the particle displacement is in agreement with the analytical solution for fluid fields obtained from a mock CFD script, while a CPL_Mocks test of the CFD
code (OpenFOAM) checks whether the steady-state pressure gradient in the column is obtained for particle fields obtained from a mock DEM script. The framework as applied to these examples is visualised in Fig 10. These tests are then run over a wide range of input values to identify corner cases.

All tests are automated and included on Travis CI, which builds the entire code from scratch in a new version of Linux every time a change is committed. As OpenFOAM is very slow to build, an anaconda deployment is used on Travis for the base version of OpenFOAM and the CPL_OPENFOAM_APP rebuilds only the latest changes to the coupled solver. The latest git (master branch) version of LAMMPS is built from scratch each time. By testing the codes in isolation, the chance they will work together is increased.

4.2.3. Coupled Testing Driven Development

Perhaps the most important contribution of the CPL LIBRARY project, valid beyond prospective users of this software, is the provision of a new philosophy for testing as part of the development of coupled codes. We provide a CPL_Mocks based framework which allows the development of coupled unit-tests in a controlled manner. This aims to encourage a new generation of test driven development (TDD) for the coupled problem. This is exemplified in Fig 10, where both codes are written to pass the tests designed in the linked CPL_Mocks scripts, before being combined to run a coupled example. The template for two possible testing strategies is provided in the CPL LIBRARY repositories.

In strategy one, we directly couple a code to a pytest script, which injects information through CPL_send and tests the resulting behaviour using CPL_recv. In this way, the CPL_Mocks can be designed to specify how the coupling should work before developing the code to do this, in the spirit of TDD. This has the advantage that the interface can be standardised, allowing new codes to be developed to this interface ensuring they will be compatible with any of the other software which has already been coupled.

The second strategy creates a range of coupled runs, using SimWrapLib, each with an MD or CFD example linked to a python dummy script. The input files and dummy scripts are changed as part of a parameter study driven by a single pytest run. This allows different processor topologies and input conditions to be explored, providing a much more general form of test and allowing identification of corner cases.

All of these tests are automated, so future correctness is ensured by continuous integration (currently Travis CI) which runs automatically in the
event of any changes to the Github repository for CPL LIBRARY. These are also triggered for changes in each of the APP repositories CPL_APP_LAMMPS or CPL_APP_OpenFOAM. When changes are made to the main code-base of LAMMPS or OpenFOAM, these tests can also serve to ensure the coupled package and basic physics is still working as expected. The latest codes are also rebuilt on DockerHub ensuring the Docker versions are always up to date.

4.3. Scaling

Choosing a meaningful metric for scaling of a coupled simulation is a non-trivial problem. As the DEM or MD code is often the rate limiting step, checking the scaling of a coupled MD example compared to the uncoupled case is a sensible first step, with results as shown in Figure 11a) for LAMMPS and OpenFOAM. These were run using OpenFOAM-3.0.1 and LAMMPS-2017Aug11 compiled with GCCv7.2.0 and MPICH3.2. The cluster used was the BP supercomputer based in Houston, US with 24 Intel Xeon E5-2690 v3 cores per node connected by a custom topology. OpenFOAM is run on a single processor with LAMMPS having a single process in y and scaled in x and z up to 4096 cores. The LAMMPS case is the standard Lennard Jones weak scaling benchmark with 32,000 atoms per processor. The coupling overlaps half the domains in y for both simulations, sending information every timestep for one layer of cells in y from CFD to MD, to use for constraints, and one layer of cells for the MD to CFD which is used as a boundary condition. The sources of coupling overhead are therefore i) Packing data and sending ii) Waiting for data to be received and unpacking iii) Applying boundary conditions to the MD code. The OpenFOAM case is an adapted version of the icoFOAM solver and a range of system sizes including 1, 9 and 100 cells and multiple processor in x and z were simulated showing negligible difference to the overall system scaling. This is because LAMMPS is the bottleneck in all calculations, so the worst case of one OpenFOAM and 4096 LAMMPS cores is shown in Figure 11a). The drop from 16 to 32 cores in the coupled case occurs when moving from intra-node communications inside a single node 24 core node to communication between multiple nodes. Despite this drop, performance for system sizes above 1024 cores appears to be similar for both LAMMPS alone and the case with LAMMPS coupled to OpenFOAM.

However, this does not expose the scaling of CPL LIBRARY itself, only slight overhead compared to the MD code. To test just the library itself, we
Figure 11: a) Weak scaling of LAMMPS (blue squares) compared to coupled example using CPL LIBRARY (red circles) on BP supercomputer in Houston with 24 cores/node. b) Weak scaling of CPL LIBRARY using a custom script to test scaling for cases where both codes use the same number of processors on ARCHER.

use a custom scaling program to model the scaling of the CPL LIBRARY in isolation.

This minimal script, included in the CPL LIBRARY utilities folder, is compiled into two executables flagged as CFD and MD. These are then run with both coupled together as part of a single simulation. As a weak scaling test, each process is setup to store approximately 600 cell subsection of the total domain, and only the number of processes is varied. The smallest size uses 24 processes in both the CFD and MD programs, so the total system has $600 \times 14,400$ = 14,400 cells per program. By scaling each program to 5016 processes (~3 million cells) in increments of processor number, the additional time taken for larger systems can be calculated. This arbitrary calculation sends three values for each of the 600 cell on each processor, receives the same volume of data and implements checks to ensure the communication is completed correctly. As the actual calculation is identical each time, the communication is the only change and we obtain a good insight into the overhead of scaling, as shown in Figure 11. The code is profiled using the CrayPat tool on ARCHER\textsuperscript{[73]}, showing packaging and unpacking of data takes about 5 % of the time in CPL\_send and CPL\_recv respectively. Given the large amounts of data sent, much larger than would be used in practice, a scaling of 70% at 10,000 cores seems to suggest reasonable behaviour.
5. Summary

A software library is presented to facilitate coupled simulation between two MPI based codes on high performance computers, where information is exchanged between part of the domain in both codes. The focus is on a minimal and simple library, with just four key routines: initialise, setup, send and receive. Simplicity is maintained by focusing the scope of application on uniform grids, a choice which allows minimal frontend scripts and provides a backend which uses just intrinsic MPI functions. This increases the chance that scaling will be maximised on supercomputing platforms. The uniform grid also allows a hierarchy of utilities to be built on this for use in coupled simulation. These utilities provide a template which can be easily extended by users to develop new forces and averaging methods, often by writing just a single function. We provide an open-source code base, a minimal API with Fortran, C, C++ and Python cross-language bindings with working examples in these multiple languages. Basic mapping and communication of the core library and utilities functions are automatically tested on continuous integration platform Travis CI. On top of the core coupling library, two APPS are provided which allow coupled simulation using LAMMPS and OpenFOAM. The development of these coupled APPS is guided using a mocking framework, a key feature of CPL LIBRARY which allows each part of the coupled model to be tested in isolation. Deployment of the code is provided by Docker and a range of build scripts for various supercomputing platforms are included with the distribution.

Acknowledgements

D. Dini and E. Smith acknowledge support received via the EPSRC Established Career Fellowship (EP/N025954/1). C O’Sullivan and A Sufian were funded by EPSRC grant EP/P010393/1. E Smith would also like to acknowledge support under both the Distributed and Embedded Computational Science and Engineering program (dCSE and eCSE respectively) as well as support from an EPSRC doctoral prize fellowship. E. Ramos-Fernandez, E. Smith, D. Dini and D. J. Trevelyan would like to acknowledge the funding and technical support from BP through the BP International Centre for Advanced Materials (BP-ICAM), which made this research possible. We would also like to acknowledge support from the Imperial College Research Computing Service, DOI: 10.14469/hpc/2232. Finally, we would
like to thank David Heyes for fruitful discussions and continued support over the many years this software has been developed.

References


29


URL https://docker.com

URL https://doi.org/10.1371/journal.pbio.1001745


URL https://github.com/edwardsmith999/pyDataView

URL https://zenodo.org/record/1308868

URL https://anaconda.com


URL https://github.com/edwardsmith999/SimWrapPy

URL https://pubs.cray.com/content/S-2376/7.0.0/cray-performance-measurement-and-analysis-tools-user-guide