Contents

1 A bottom-up introduction to CFD-DEM coupling with preCICE

This course develops a coupled CFD-DEM simulation one component at a time. All three cases use MercuryDPM for the particles. We first prescribe the flow analytically, then replace that function with a CFD solver, and finally let fluid and particles influence each other in a fluidized-bed simulation.

Case	Fluid participant	Particle participant	Coupling
`01` Rayleigh–Kothe	Analytical Python	MercuryDPM	One-way
`02` Channel transport	OpenFOAM [Nutils]	MercuryDPM	One-way
`03` Fluidized bed	OpenFOAM	MercuryDPM [LIGGGHTS]	Two-way

Work through the numbered directories in order. Each case README covers four steps: compiling only the newly required software, understanding the physical setup and preCICE configuration, running the participants, and inspecting the results.

1.1 Course bundle

The bundle-level `software/` directory contains source archives for preCICE 3.4.1, MercuryDPM, the OpenFOAM adapter, the custom OpenFOAM solver, and LIGGGHTS. This `experimental-setups/` directory contains the three cases and their common `tools/` directory.

Each coupled participant lives in its own subdirectory and is started in a separate terminal. The case-level `precice-config.xml` defines participants, meshes, exchanged data, communi-

cation, mapping, and the coupling scheme. After an interrupted run, use the local `clean.sh` scripts before restarting so that stale communication files do not interfere.

Commands in the course use the environment variables introduced at the start of case 1. You replace the course bundle path once; later paths inside the bundle and its local software installations are derived from it. The same section explains how to keep the variables available in a new terminal. Storing them in `~/.bashrc` (or the startup file for your shell) also makes restarting a participant easier after a terminal or solver crash.

2 Case 1: Particle transport in a Rayleigh–Kothe vortex

This first case couples an analytical velocity field implemented in Python to tracer particles simulated with MercuryDPM. Information travels only from the analytical participant to the particles. The case introduces the basic preCICE workflow without requiring a CFD solver. This case is part of MercuryDPM and is tested as part of its CI.

2.1 Step 1: Compiling Software: MercuryDPM and preCICE

First define where you extracted the course bundle. **Replace only the path on the right-hand side of the first line** with the absolute path on your machine. Do not change the derived paths, and use the same values in all three cases:

```
export COURSE_ROOT="/absolute/path/to/cfd-dem-course"
export SETUPS_ROOT="$COURSE_ROOT/experimental-setups"
export PRECICE_INSTALL_DIR="$COURSE_ROOT/software/precice-install"
export MERCURYDPM_BUILD_DIR="$COURSE_ROOT/software/mercurydpm-dev/build"
cd "$COURSE_ROOT/software"
```

All commands in this build step are executed in the terminal in which you set these variables. The commands below show every directory change explicitly.

Install the common build dependencies on Ubuntu:

```
sudo apt install build-essential cmake libboost-all-dev libeigen3-dev libxml2-dev \
  libopenmpi-dev openmpi-bin pkg-config python3-venv
```

For other operating systems and installation methods, consult the preCICE installation overview and the platform-specific dependency instructions. The source-build guide explains the CMake options in more detail.

Extract and install the bundled preCICE source:

```
cd "$COURSE_ROOT/software"
tar -xzf precice-v3.4.1.tar.gz
cd precice-v3.4.1
mkdir -p build
cd build
cmake \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_INSTALL_PREFIX="$PRECICE_INSTALL_DIR" \
  -DPRECICE_FEATURE_MPI_COMMUNICATION=ON \
  -DPRECICE_FEATURE_PETSC_MAPPING=OFF \
  -DPRECICE_FEATURE_GINKGO_MAPPING=OFF \
  -DPRECICE_FEATURE_PYTHON_ACTIONS=OFF \
  -DBUILD_TESTING=OFF \
  ..
make -j4
make install
```

The example uses four parallel build jobs. Change `4` to a suitable limit for your machine or the course computer; always give `-j` an explicit number. The course does not use preCICE's embedded Python actions, so they are disabled to avoid an unnecessary build dependency. This is separate from the `pyprecice` package installed below for the analytical participant.

The remaining builds and every simulation terminal must be able to find this preCICE installation. Export the paths in the current terminal now:

```
export CMAKE_PREFIX_PATH="$CMAKE_PREFIX_PATH:$PRECICE_INSTALL_DIR"
export PKG_CONFIG_PATH="$PKG_CONFIG_PATH:$PRECICE_INSTALL_DIR/lib/pkgconfig"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$PRECICE_INSTALL_DIR/lib"
```

These exports disappear when you close the terminal. Store them in `~/.bashrc` so they are set automatically in every new terminal, including one opened to restart a participant after its solver or terminal crashes. Add the following block, again replacing only the value assigned to `COURSE_ROOT`, and then open a new terminal (or run `source ~/.bashrc`):

```
# CFD-DEM course environment
export COURSE_ROOT="/absolute/path/to/cfd-dem-course"
export SETUPS_ROOT="$COURSE_ROOT/experimental-setups"
export PRECICE_INSTALL_DIR="$COURSE_ROOT/software/precice-install"
export MERCURYDPM_BUILD_DIR="$COURSE_ROOT/software/mercurydpm-dev/build"
export CMAKE_PREFIX_PATH="$CMAKE_PREFIX_PATH:$PRECICE_INSTALL_DIR"
export PKG_CONFIG_PATH="$PKG_CONFIG_PATH:$PRECICE_INSTALL_DIR/lib/pkgconfig"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$PRECICE_INSTALL_DIR/lib"
```

Add this block only once. If you use a shell other than Bash, put the equivalent exports in that shell's startup file. Also see the guide to making preCICE discoverable if CMake or the dynamic linker still cannot find it.

Extract MercuryDPM, configure it against that installation, and build the course executables:

```
cd "$COURSE_ROOT/software"
tar -xzf mercurydpm-dev.tar.gz
cd mercurydpm-dev
mkdir -p build
cd build
cmake \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_PREFIX_PATH="$PRECICE_INSTALL_DIR" \
  -DMercuryDPM_PreCICE_COUPLING=ON \
  -DMercuryDPM_USE_OpenMP=ON \
  ..
make -j4 \
  RayleighKothe ChannelTransport SettleFluidizedBed FluidizedBed MercuryCG
```

This creates every MercuryDPM executable needed by the three course cases in one build tree.

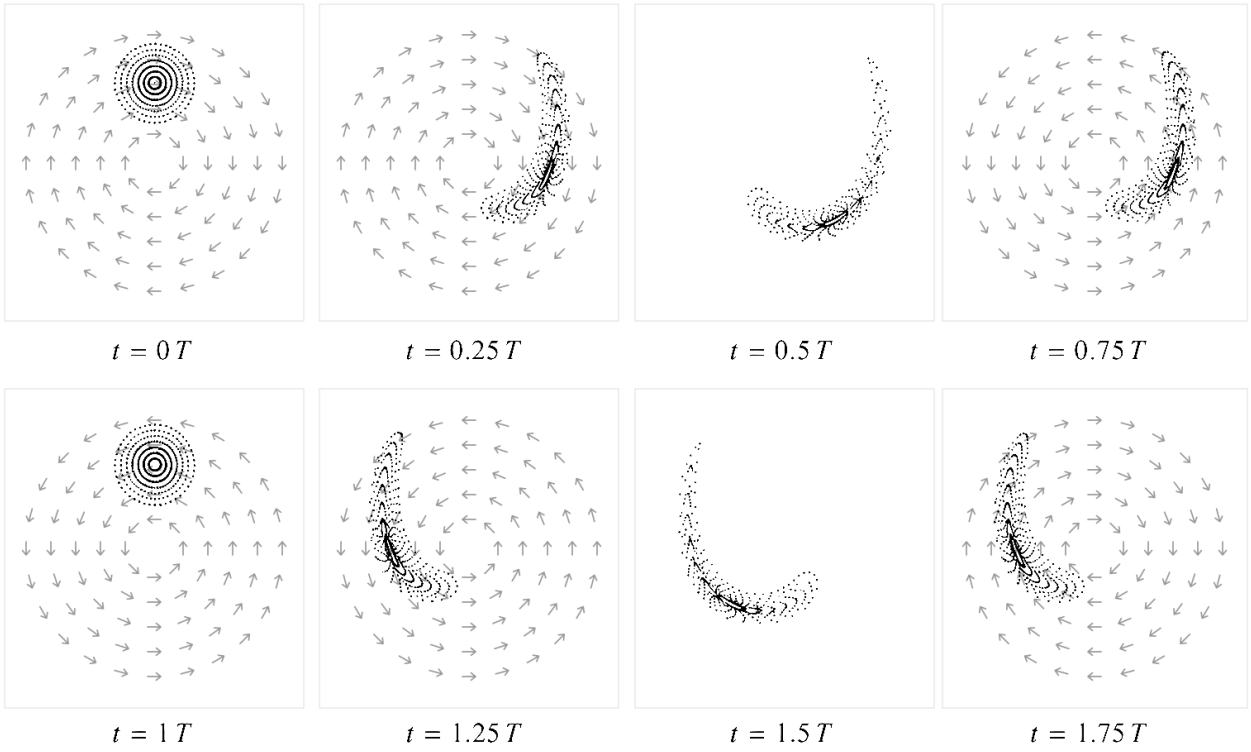
Finally, create and activate an isolated Python environment for the analytical participant:

```
cd "$SETUPS_ROOT/01-rayleigh-kothe/fluid-analytical"
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

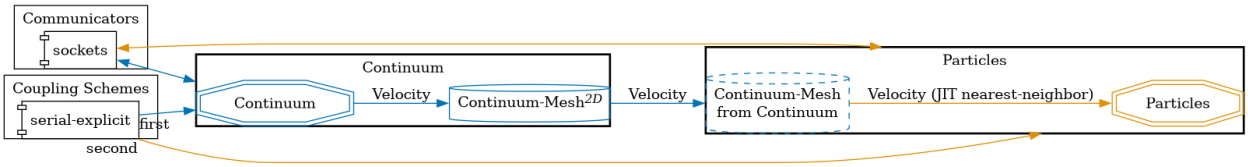
Activate this environment again whenever you open a new terminal for the Python participant.

2.2 Step 2: Understanding the Setup and Configuration

The Rayleigh-Kothe vortex periodically deforms a ring of particles and returns it close to its initial shape. MercuryDPM sends particle positions to preCICE, which evaluates the analytical velocity field at those positions using just-in-time nearest-neighbor mapping.



The case-level `precice-config.xml` defines the `Continuum` and `Particles` participants, the velocity data, their meshes, and a serial explicit coupling scheme.



2.3 Step 3: Running the Case

Open two terminals in this case. Start the analytical participant in the first:

```
cd "$SETUPS_ROOT/01-rayleigh-kothe/fluid-analytical"
source .venv/bin/activate
./run.sh
```

Start MercuryDPM in the second:

```
cd "$SETUPS_ROOT/01-rayleigh-kothe/particles-mercurydpm"
./run.sh --build-dir="$MERCURYDPM_BUILD_DIR"
```

It is normal for the first participant to wait until the second one starts. Use the participant-local `clean.sh` scripts before repeating a run.

2.4 Step 4: Inspecting the Result

Open the generated `RayleighKotheParticle_*.vtu` sequence in ParaView. Over one coupling period, the particles should deform with the vortex and then return close to their starting positions.

2.4.1 Optional accuracy task: compare two mappings

The supplied configuration uses nearest-neighbor mapping. This is a computationally cheap and robust choice, but it assigns the value of the closest continuum vertex and therefore produces a piecewise-constant approximation of the analytical velocity field.

First run the case with the supplied configuration. Then replace the `mapping:nearest-neighbor` XML tag in the `Particles` participant (line 30 in the `precice-config.xml`) with a partition-of-unity RBF mapping using a compact polynomial C6 basis function:

```
<mapping:rbf-pum-direct
  direction="read"
  from="Continuum-Mesh"
  constraint="consistent"
  project-to-input="false">
  <basis-function:compact-polynomial-c6 support-radius="2" />
</mapping:rbf-pum-direct>
```

Clean and rerun both participants, then compare the initial and final particle positions in ParaView. After one vortex period, do the particles reach their exact starting positions? You may also change the `110`-by-`110` continuum mesh resolution, e.g., reduce `num_vertices_x` and `num_vertices_y` in `fluid-analytical/rayleigh-kothe-function.py`, use the same resolution for both mappings, and repeat the comparison.

3 Case 2: Channel transport with OpenFOAM and MercuryDPM

This case replaces the analytical flow field from case 1 with a two-dimensional incompressible CFD simulation. OpenFOAM computes flow through a channel with an obstacle and sends velocity to MercuryDPM. The particles remain passive: they do not affect the fluid.

The setup follows the preCICE channel transport with particles tutorial.

3.1 Step 1: Compiling Software: OpenFOAM and Its preCICE Adapter

This case reuses `COURSE_ROOT`, `SETUPS_ROOT`, `PRECICE_INSTALL_DIR`, `MERCURYDPM_BUILD_DIR`, and the preCICE search paths configured in case 1. Make sure that persistent environment is loaded in each new terminal.

Keep the preCICE and MercuryDPM installations from case 1; its build step already created `ChannelTransport`. Follow the preCICE guide on how to get OpenFOAM for installation options. Activate the OpenFOAM environment in every terminal that builds or runs OpenFOAM. The supplied material is known to work with OpenFOAM v2406, v2412, and v2512. For an OpenFOAM v2512 package installation, the activation commands are typically:

```
export OPENFOAM_BASHRC=/usr/lib/openfoam/openfoam2512/etc/bashrc
source "$OPENFOAM_BASHRC"
```

Adjust only the assigned path for your installation, then use this same variable in every OpenFOAM terminal. You can add both lines to `~/.bashrc`, or repeat them whenever you open a terminal. Ensure that the `PKG_CONFIG_PATH` and `LD_LIBRARY_PATH` settings for the preCICE installation from case 1 are also active.

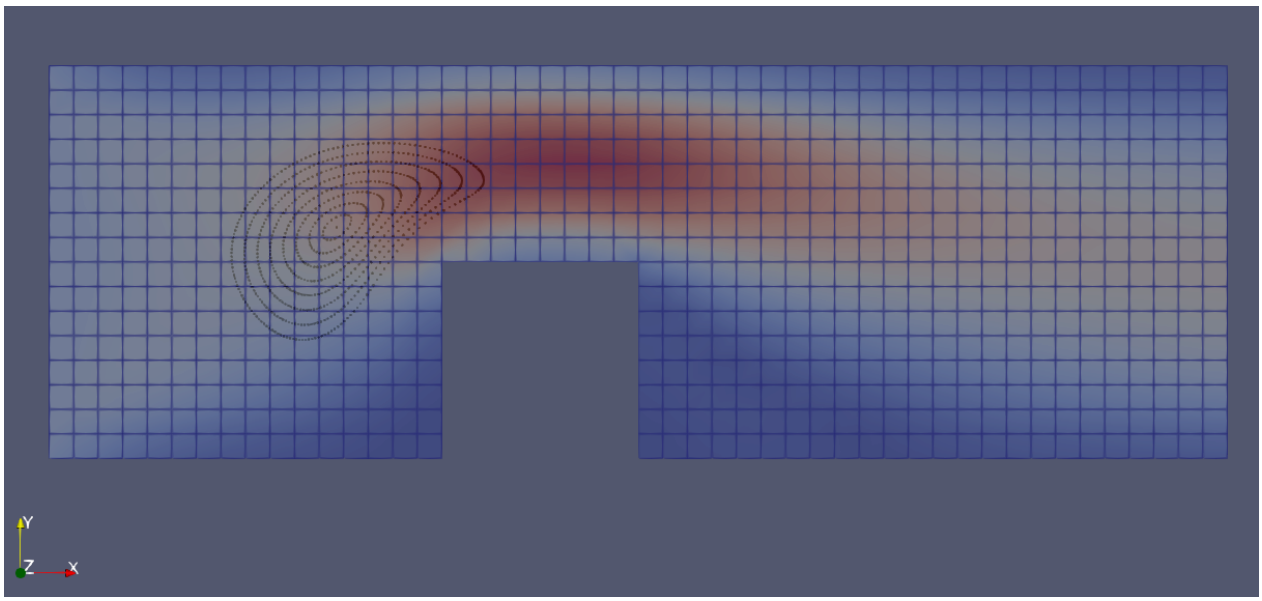
Extract and compile the bundled adapter after both OpenFOAM and preCICE are available:

```
cd "$COURSE_ROOT/software"
tar -xzf openfoam-adapter-dev.tar.gz
cd openfoam-adapter-dev
./Allwmake
foamHasLibrary -verbose precice libpreciceAdapterFunctionObject
```

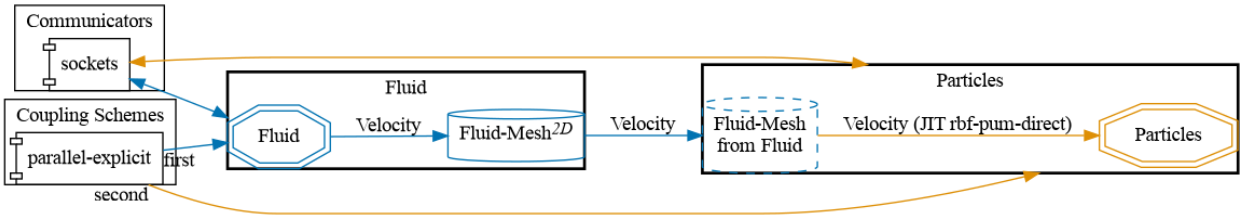
The final command should report that both libraries can be loaded. If compilation fails, inspect `wmake.log` and `ldd.log` in the adapter directory.

3.2 Step 2: Understanding the Setup and Configuration

Particles start as a compact circular cloud near the inlet. The fluid flows from left to right around the obstacle and transports the cloud downstream.



The OpenFOAM adapter is configured in `fluid-openfoam/system/preciceDict`. The case-level `precice-config.xml` couples `Fluid` to `Particles`, exchanges velocity in one direction, and maps it to particle positions just in time.



3.3 Step 3: Running the Case

Activate OpenFOAM and start the fluid participant in the first terminal:

```
source "$OPENFOAM_BASHRC"
cd "$SETUPS_ROOT/02-channel-transport-particles/fluid-openfoam"
./run.sh
```

Start the particle participant in the second terminal:

```
cd "$SETUPS_ROOT/02-channel-transport-particles/particles-mercurydpm"
./run.sh --build-dir="$MERCURYDPM_BUILD_DIR"
```

Use the participant-local `clean.sh` scripts before repeating a run.

3.4 Alternative: Using Nutils Instead of OpenFOAM

Nutils provides a lightweight Python alternative to OpenFOAM; see the preCICE Nutils adapter documentation and the same channel transport tutorial. Start `fluid-nutils/run.sh` instead of `fluid-openfoam/run.sh`, then run the unchanged MercuryDPM participant from Step 3. The Nutils launcher creates a local virtual environment and installs its Python requirements automatically. Note that if you choose Nutils here, OpenFOAM is required for the fluidized-bed case in case 3.

```
cd "$SETUPS_ROOT/02-channel-transport-particles/fluid-nutils"
./run.sh
```

3.5 Step 4: Inspecting the Result

Open the generated fluid and particle VTK files in ParaView. Compare the particle paths around the obstacle with the velocity field.

4 Case 3: Bidirectional CFD-DEM coupling in a fluidized bed

The final case reproduces Case A from J. M. Link, L. A. Cuypers, N. G. Deen, and J. A. M. Kuipers, "Flow regimes in a spout-fluid bed: a combined experimental and simulation study," *Chemical Engineering Science* 60(13) (2005), 3425–3442, doi:10.1016/j.ces.2005.01.027. The case and further methodological details are documented in D. Schneider, *Flexible and Efficient Data Mapping for Simulation of Coupled Problems*, PhD thesis, Universität Stuttgart, 2026, doi:10.18419/opus-18423. The setup and data provided here were largely adopted from D. Schneider, *Replication Data for: A Modular Approach for Mesh-Particle Coupling*, DaRUS, 2025, doi:10.18419/DARUS-5494.

OpenFOAM solves the volume-averaged Navier–Stokes equations and MercuryDPM resolves the particles. Fluid velocity and pressure gradient drive the particles, while particle volume fraction and momentum exchange feed back into the fluid.

4.1 Step 1: Compiling Software: Custom OpenFOAM and MercuryDPM Solvers

This case reuses the persistent course environment from case 1 and the OpenFOAM activation path from case 2. Make sure both are loaded in each new terminal.

Keep preCICE, MercuryDPM, OpenFOAM, and the OpenFOAM adapter from the previous cases. The MercuryDPM build from case 1 already contains `SettleFluidizedBed` and `FluidizedBed`.

Activate OpenFOAM, extract the custom solver, and compile it with `wmake`:

```
source "$OPENFOAM_BASHRC"
cd "$COURSE_ROOT/software"
tar -xzf openfoam-solver.tar.gz
cd openfoam-solver
wmake
command -v AndersonJacksonFoam
```

The final command should locate `AndersonJacksonFoam`, normally under `$FOAM_USER_APPBIN`. If `OPENFOAM_BASHRC` is empty, configure it as described in case 2 and use the same value in every OpenFOAM terminal.

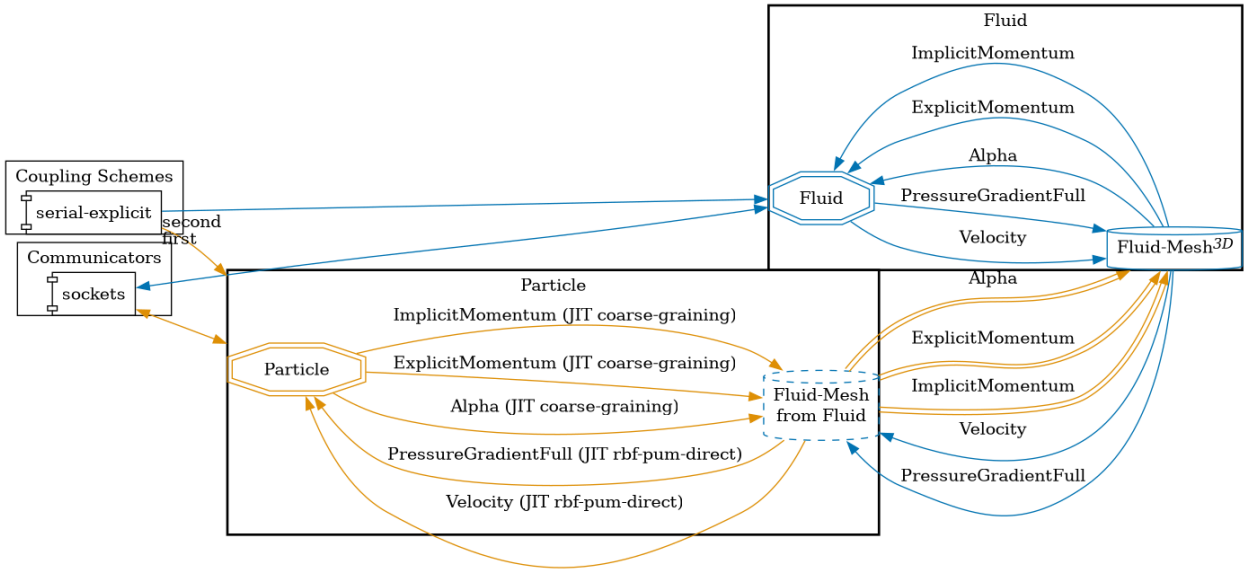
4.2 Step 2: Understanding the Setup and Configuration

The domain is a narrow fluidized bed with a packed particle layer and an upward gas flow. `fluid-openfoam/` contains the CFD case, while `particles-mercurydpm/` contains the DEM launcher and a prepared settling restart archive.

Before the first run, unpack the initial particle state:

```
cd "$SETUPS_ROOT/03-link-fluidized-bed/particles-mercurydpm"
tar -xzf settled-restart-file.tar.gz
```

The OpenFOAM adapter configuration is in `fluid-openfoam/system/preciceDict`. The case-level `precice-config.xml` exchanges velocity, pressure gradient, solid volume fraction, and explicit and implicit momentum terms using a serial explicit scheme and just-in-time mappings.



4.3 Step 3: Running the Case

Activate OpenFOAM and start the fluid participant in the first terminal:

```
source "$OPENFOAM_BASHRC"
cd "$SETUPS_ROOT/03-link-fluidized-bed/fluid-openfoam"
./run.sh
```

Start the MercuryDPM participant in the second terminal:

```
cd "$SETUPS_ROOT/03-link-fluidized-bed/particles-mercurydpm"
./run.sh --build-dir="$MERCURYDPM_BUILD_DIR"
```

The particle launcher runs `FluidizedBed` with six OpenMP threads. The prepared restart avoids recomputing the settling; `SettleFluidizedBed` is available if you want to regenerate it. Clean both participant directories before repeating an interrupted run.

4.4 Alternative: Using LIGGGHTS Instead of MercuryDPM

The course bundle also contains a modified LIGGGHTS version with the preCICE coupling and pressure-gradient exchange required by this case. It needs MPI and Voro++. On Ubuntu, install the library and its development files with:

```
sudo apt install voro++ voro++-dev
```

Alternatively, build Voro++ from source. The library must be compiled as position-independent code so that LIGGGHTS can link it into its shared library:

```
cd "$COURSE_ROOT/software"
git clone https://github.com/chr1shr/voro.git
cd voro
mkdir -p build
cd build
export VORO_INSTALL_DIR="$COURSE_ROOT/software/voro-install"
cmake \
  -DCMAKE_BUILD_TYPE=Release \
  -DCMAKE_POSITION_INDEPENDENT_CODE=ON \
  -DCMAKE_INSTALL_PREFIX="$VORO_INSTALL_DIR" \
  ..
make -j4
make install
export CMAKE_PREFIX_PATH="$CMAKE_PREFIX_PATH:$VORO_INSTALL_DIR"
```

With Voro++ installed and preCICE discoverable in the current environment, build LIGGGHTS from the bundle's `software/` directory:

```
cd "$COURSE_ROOT/software"
tar -xzf liggghts-dev.tar.gz
cd liggghts-dev
mkdir -p build
cd build
export LIGGGHTS_INSTALL_DIR="$COURSE_ROOT/software/liggghts-install"
cmake \
  -DCMAKE_INSTALL_PREFIX="$LIGGGHTS_INSTALL_DIR" \
  -DCMAKE_BUILD_TYPE=Release \
  -DENABLE_PRECICE=ON \
  -DENABLE_MODEL_HOOKE_STIFFNESS=ON \
  -DENABLE_MODEL_TANGENTIAL_HISTORY=ON \
  -DENABLE_MODEL_COHESION_OFF=ON \
  -DENABLE_MODEL_ROLLING_OFF=ON \
  -DENABLE_MODEL_SURFACE_DEFAULT=ON \
  -DENABLE_MPI=ON \
  -DENABLE_VORONOI=ON \
  -DENABLE_VTK=ON \
  ../src
make -j4
make install
export PATH="$LIGGGHTS_INSTALL_DIR/bin:$PATH"
```

As in case 1, the optional Voro++ and LIGGGHTS exports must be set in every terminal that builds or runs LIGGGHTS. Add `VORO_INSTALL_DIR`, `LIGGGHTS_INSTALL_DIR`, the updated `CMAKE_PREFIX_PATH`, and `PATH` commands to `~/.bashrc` if you want to keep them across terminals.

Keep the OpenFOAM participant from Step 3. In the particle terminal, use the LIGGGHTS case instead:

```
cd "$SETUPS_ROOT/03-link-fluidized-bed/particles-liggghts"
tar -xzf settled-restart-file.tar.gz
./run.sh
```

This launches two MPI ranks using `in.cfdem.liggghts`. The commented command in `run.sh` shows how to regenerate the settled state. Clean the OpenFOAM and LIGGGHTS directories before switching particle solvers or restarting a run.

The coupling commands are in `in.cfdem.liggghts`. If you change the preCICE mapping radius, keep the mirror width in the `fluid_coupling` command consistent with it (both are `0.008` in the supplied case). The `precice_initialize` command also specifies a `0.015` safety margin around the fluid mesh; revisit it if the geometry or mapping support changes.

4.5 Step 4: Inspecting and Comparing the Result

Open the OpenFOAM fields and the MercuryDPM or LIGGGHTS VTK snapshots in ParaView. The supplied result archives provide reference runs. The `postprocessing/` directory contains the Link et al. reference data, processed MercuryDPM and LIGGGHTS results, plotting scripts, and `comparison-all.pdf` for a compact quantitative comparison.

4.6 Advanced Topic: Postprocessing and Solver Comparison

The comparison evaluates the vertical particle mass flux across the bed. To coarse-grain a completed MercuryDPM run, execute `MercuryCG` from the case root, replacing the input name or time interval if your run differs:

```
cd "$SETUPS_ROOT/03-link-fluidized-bed"
"$MERCURYDPM_BUILD_DIR/Drivers/MercuryCG/MercuryCG" FluidizedBed \
  -coordinates XY -x -0.075 0.075 -y 0.125 0.135 -ny 1 -nx 40 \
  -timeaverage -tmin 2.5 -tmax 6 -function Lucy -w 0.005 \
  -o postprocessing/mercury-1-5s-to-5s-w0-005-x40.dat
```

For LIGGGHTS, unpack the averaged raw data and convert it to the common profile format:

```
cd "$SETUPS_ROOT/03-link-fluidized-bed/postprocessing"
tar -xzf liggghts-averaged-data.tar.gz
python3 process-liggghts-averaged-data.py \
  --file liggghts-results-1-5-to-5.dat --dx 0.00375 --dy 0.01
```

By default, the script uses the last available timestamp and writes its files under `liggghts-results/`. Use `--time <N>` to select another cumulative average. The supplied `liggghts-1-5s-to-5s-x40.csv` is the already processed profile from the reference run and is used in the command below.

Finally, compare both particle solvers with the Link et al. experimental data and Koch-Hill result:

```
python3 plot-comparison-all.py \
  --ref Link2005.dat \
  --mercury mercury-1-5s-to-5s-w0-005-x40.dat \
  --liggghts liggghts-1-5s-to-5s-x40.csv \
  --pdf comparison-all.pdf
```

The generated MercuryDPM particle data can be large. The processed profiles and `comparison-all.pdf` are therefore already included, so this advanced step can also be studied without rerunning or retaining every raw output file.