Postprocessing and Particle Tracking Code

1 Dependencies

The postprocessing operations we perform at the end of a simulation aim at computing parameters of possible biological interest. Due to the complexity of the geometries that we normally consider and the large dimensions of our domains of interest, we typically rely on a number of popular external libraries to help dealing with this task in a HPC enviroment. This unfortunately means that a number of dependencies must be met in order for our software to run, but luckily many of this libraries are often found already installed on HPC systems. Here is a comprehensive list of all the dependencies we have at the moment

- **MPI** – we use OpenMPI 1.7.4
- **MPI4PY** – a wrapper of MPI in Python (our version on the cluster is 1.3.1)
- **HDF5** – a library for the (parallel) management of large composite datasets (version 1.8.10). **NOTE: The parallel component must be enabled, either via `./configure --enable-parallel` or by directly compiling with `mpicc` (e.g. after `export CC=mpicc`)**
- **H5PY** – Python wrapper of HDF5. Must be built against a parallel-enabled HDF5 library (our version is 2.2.1)
- **CMake** – Configuring utility necessary to build VTK (and other Kitware products). Our version is 3.1.3
- **VTK** – a library for the management of complex 3D meshes (and much more). Version 5.8.0 with Python modules enabled. This library is better built after configuring with **CMake** (e.g. calling `ccmake` and explicitly activating the Python section)
- **NUMPY** – fast library for linear algebra computations in Python. Our version is 1.8.1

2 Files

We here report a list of the files included in the tarball. These utilities should be able to cover the whole workflow that allows particle tracking and TFP computations once the results are stored in the ordinary `.vis` files that the CRIMSON postprocessor produces.

2.1 mesh2vtk.py

This script converts coordinate and connectivity files produced by the mesher into a VTK unstructured grid file (`.vtu`).

Usage Example:

```
python mesh2vtk.py fname
```

Command Line Arguments:

- `fname` – name of the mesh. The coordinates and connectivity files must be named `fname.coordinates` and `fname.connectivity`, respectively

Input Files:

- `fname.coordinates` – coordinate file produced by the mesher
- `fname.connectivity` – connectivity file produced by the mesher

Output Files:

- `fname.vtu` – VTK unstructured grid file of the mesh

2.2 mesh2vtkexternal.py

This script is very helpful to create a VTK polydata file of the external surface of a mesh.

Usage Example:

```
python mesh2vkexternal.py fname allwelements.ebc
```

Command Line Arguments:

- `fname` – Name of the mesh. The coordinates and connectivity files must be named `fname.coordinates` and `fname.connectivity`, respectively
- `allwelements.ebc` – Element file corresponding to the external surface we want to extract from the mesh (e.g. concatenation of all the `wall_*.ebc` files produced by the CRIMSON mesher)

Input Files:

- `fname.coordinates` – Coordinate file produced by the mesher
- `fname.connectivity` – Connectivity file produced by the mesher
- `allwelements.ebc` – Element file of the external surface we want to extract from the mesh

Output Files:

- `allwelements.ebc.vtp` – External surface in VTK polydata format (all nodes of initial mesh are present but only those on the surface are connected)
- `allwelements.ebc-clean.vtp` – External surface in VTK polydata format (no extra nodes)

2.3 vis2hdf5.py

This Python script reads the `.vis` files produced by the CRIMSON postprocessor and reduces them into one unique HDF5 file. Information on the geometry (mesh and external surface of the mesh) must be provided through two VTU and VTP files of the geometry. The script produces in output also a metadata (`.xdmf`) so that the `.h5` file can be directly visualized in Paraview.

Usage Example:

```
mpirun -np 16 python vis2hdf5.py fname start stop step \
    meshfname.vtu surffname.vtp
```

Command Line Arguments:

- `fname` – Representative name of the simulation. The `.vis` filenames have the format `fname-%i.vis`, where `%i` is an integer representing the timestep number
- `start` – Initial timestep (integer)
- `stop` – Final timestep (integer)
- `step` – Interval between saved timesteps (integer)
- `meshfname.vtu` – Filename of `.vtu` file containing mesh information (coordinates and connectivity)
- `surffname.vtp` – Filename of `.vtp` file containing external surface information (coordinates and connectivity)

Input Files:

- `fname-%i.vis` – Series of `.vis` files containing the simulation results for each timestep. `%i` is an integer representing the timestep number
- `meshfname.vtu` – VTK unstructured grid file containing the mesh information (coordinates and connectivity) as an unstructured grid
- `surfname.vtp` – VTK polydata file containing geometric (coordinates and connectivity) information of the external mesh boundary

Output Files:

- `fname.h5` – HDF5 file containing the simulation results for each timestep
- `fname.xdmf` – Metadata file to instruct Paraview on how to read the HDF5 file
- `fname-wss.vtp` – VTK polydata file containing distribution of computed WSS-based indices on the external surface of the mesh

2.4 particletracking.py

This script provides a Python (parallel) version of the particle tracking code that maintains the same features of its C/C++ counterpart, but it is more flexible, it has reduced dependencies and it is only minimally penalized in terms of speed since it relies on the same libraries with the only overhead being Python calls to VTK, HDF5, and NUMPY. We typically call this script in an array job that varies the command line argument `stepstart` to probe throughout the cardiac cycle (typically 40/50 times).

Usage Example:

```
mpirun -np 16 python particletrackingpython.py fname \
    start stop step stepstart ncycles dt disp repartition
```

Command Line Arguments:

- `fname` – Representative name of the simulation (also filename of the `.h5` file without extension)
- `start` – Timestep number corresponding to the first step of the cardiac cycle
- `stop` – Timestep number corresponding to the last step of the cardiac cycle
- `step` – Interval between saved timestep (integer)
- `stepstart` – Timestep number of injection. Particle advection starts at this timestep and continues for `ncycles` full cardiac cycles
- `ncycles` – Integer number of cardiac cycles that we would like the advection to continue for
- `dt` – Time interval between `step` timesteps (in seconds)
- `disp` – Flag integer indicating whether the simulation is deformable (`1`) or not (`0`)
- `repartition` – Integer indicating the frequency of repartitioning (e.g. 2 means repartitioning occurs every 2 steps)

Input Files:

- `particles.vtu` – VTK filename that stores a high resolution mesh whose nodes coordinates will be used as location for initial particle injection
- `fname.h5` – HDF5 file containing the simulation results
- `allwnodes.nbc` – Text file obtained concatenating all the `wall_*.nbc` files in the `mesh-surfaces` folder
- `rp-sp.vtp` – VTK polydata file containing geometric (coordinates and connectivity) information of the external mesh boundary

Output Files:

- `fname-particles-stepstart.h5` – HDF5 files containing tracking of the particles injected at `stepstart`
- `fname-particles-stepstart.xdmf` – XDMF file that allows Paraview to visualize the particle tracking results file
- `fname-particles-stepstart.vtu` – VTK unstructured grid file that contains the computed particle indices "accumulated" at the end of the advection (i.e. after `ncycles` cardiac cycles)

2.5 tfp.py

This (serial) script can be run after the particle tracking simulations are over in order to average the results and extract indices of interest, especially in regions close to the wall, such as TFP, PRT, and FTLE. It also computes upper percentiles of the indices, which is particularly helpful for particle based indices.

Usage Example:

```
python tfp.py fname normal_ids aaa_id
```

Command Line Arguments:

- `fname` – Representative name of the simulation (also filename of the `.vtp` containing the WSS based indices)
- `percent` – Integer indicating how much the domain close to the wall will be probed to determine "near wall" quantities (in percents of local radius)

Input Files:

- `fname-wss.vtp` – VTK polydata file with the WSS results (produced by `vis2hdf5.py`)
- `mesh.vtu` – VTU unstructured file HDF5 file containing the mesh information (produced by `mesh2vtk.py`)
- `fname-particles-*.vtu` – Series of VTK unstructured grid files containing the results at the end of the particle advection (produced by `particletracking.py`)

Output Files:

- `fname-tfp.vtp` – Results file (in VTK polydata format) that containes indices averaged in the near-wall region HDF5 files containing tracking of the particles injected at `stepstart` (`%i` assumes the value of `stepstart`)
- `fname-particles-%i.xdmf` – XDMF file that allows Paraview to visualize the particle tracking results file
- `$fname-particles-%i.vtu` – VTK unstructured grid file that contains the computed particle indices "accumulated" at the end of the advection (i.e. after `$ncycles` cardiac cycles)

3 Test Example on a Simple Cylinder

As a showcase example we test the entire workflow on a simple cylindrical geometry provided in the `example` folder.

Mesh and Particle Mesh The CRIMSON mesher is used to produce two meshes at different resolutions for CFD and particle simulations (max edge size 1.0 and 0.5, respectively)

Conversion of Meshes to VTK The script `mesh2vtk.py` converts the meshes into a VTK format. The connectivity and coordinates files must be gunzipped prior to this operation.

```
python ..\src\mesh2vtk.py mesh
cd particles-mesh
python ..\..\src\mesh2vtk.py particles
cd ..
```

External surface extraction After concatenating the `wall*.nbc` and `wall*.ebc` into the `allwnodes.nbc` and `allwelements.ebc` files, respectively, the external surface of the mesh is extracted

```
python ..\src\mesh2vtkexternal.py mesh allwelements.ebc
```

VMTK to compute centerline and radius At this point, we employ VMTK to compute the locally varying radius on the surface. This comes in handy in real cases since at the end of the particle tracking we usually want to extract near wall properties, where "near-wall" is typically a fraction of the local radius. In the VMTK `pypepad` you can use the following command (on one line)

```
vmtkcenterlines -ifile path/to/example/allwelements.ebc-clean.vtp
-radiusarray Radius -endpoints 1 -ofile path/to/example/rp-cl.vtp --pipe
vmtksurfaceprojection -ifile path/to/example/allwelements.ebc-clean.vtp
-rfile path/to/example/rp-cl.vtp -ofile path/to/example/rp-sp.vtp
```

Now in the `rp-sp.vtp` file we have the external surface populated with a "Radius" array, while `rp-cl.vtp` contains the centerline

Simulation We run the fluid simulation and postprocess the results to extract the `.vis` files (see `simulation-files/particles`). In this example we run the fluid simulation only for one cardiac cycle (9000 timesteps, timestep duration 0.0001, saving every 50 timesteps)

Convert VIS files to HDF5 After copying the mesh and surface files, from the folder where the `.vis` files are stored we execute

```
mpirun -np 16 python vis2hdf5.py example 50 9000 50 mesh.vtu rp-sp.vtp
```

This generates the files `example.h5`, `example-wss.vtp`, and `example.xdmf`

Particle Tracking Particles injected at the 50th and 4500th timesteps are advected forward in time for 4 cardiac cycles with the commands

```
mpirun -np 16 python particletracking.py example 50 9000 50 50 0.005 0 1
mpirun -np 16 python particletracking.py example 50 9000 50 4500 0.005 0 1
```

Similarly particles could be advected backward in time with the commands

```
mpirun -np 16 python particletracking.py example 9000 50 -50 50 -0.005 0 1
mpirun -np 16 python particletracking.py example 9000 50 -50 4500 -0.005 0 1
```

Average Result and Near-Wall Indices In this final step we average the results obtained from different injection times and compute the near wall indices (probing the domain close to the wall as far as 5% of the local radius. The command

```
python tfp.py example 5
```

outputs the results in `example-tfp.vtp`