

# **2/11/2020 Week 5 Module 1**

## **Interactive Exercise for Molecular Dynamics**

- In this exercise we will
  - prepare a python script to run OpenMM using the OpenMM script builder:  
<http://builder.openmm.org>
  - run a short MD simulation on our desktops
  - visualize the simulation with VMD

- This is the default script in the OpenMM script builder: <http://builder.openmm.org>
- Let's go through what the parameters mean and I'll give you values for a simple simulation of ubiquitin, a protein that we looked at during the VMD tutorial
- If you hover your mouse over the parameter names, it'll also give a brief description
- A more detailed description of simulation parameters is available at <http://docs.openmm.org/latest/userguide/application.html#simulation-parameters>

The screenshot shows the OpenMM Script Builder web interface. The 'General' tab is selected, displaying the following parameters:

- Input coordinates: input.pdb
- Input topology: input.prmtp
- Forcefield: AMBER99sb-ildn
- Water Model: TIP3P
- Platform: CUDA
- Precision: mixed
- Device index: (empty)
- OpenCL platform indx: (empty)

The right panel shows the generated Python script, which is a template for running a simulation. The script includes comments and code for setting up the simulation environment, loading the input files, creating the system, integrator, and platform, and running the simulation.

```
#####
# this script was generated by openmm-builder. to customize it further,
# you can save the file to disk and edit it with your favorite editor.
#####

from __future__ import print_function
from simtk.openmm import app
import simtk.openmm as mm
from simtk import unit
from sys import stdout

pdb = app.PDBFile('input.pdb')
forcefield = app.ForceField('amber99sbildn.xml', 'tip3p.xml')

system = forcefield.createSystem(pdb.topology, nonbondedMethod=app.PME,
                                nonbondedCutoff=1.0*unit.nanometers, constraints=app.HBonds, rigidWater=True,
                                ewaldErrorTolerance=0.0005)
integrator = mm.LangevinIntegrator(300*unit.kelvin, 1.0/unit.picoseconds,
                                   2.0*unit.femtoseconds)
integrator.setConstraintTolerance(0.00001)

platform = mm.Platform.getPlatformByName('CUDA')
properties = {'CudaPrecision': 'mixed'}
simulation = app.Simulation(pdb.topology, system, integrator, platform,
                           properties)
simulation.context.setPositions(pdb.positions)

print('Minimizing...')
simulation.minimizeEnergy()

simulation.context.setVelocitiesToTemperature(300*unit.kelvin)
print('Equilibrating...')
simulation.step(100)

simulation.reporters.append(app.DCDReporter('trajectory.dcd', 1000))
simulation.reporters.append(app.StateDataReporter(stdout, 1000, step=True,
                                                  potentialEnergy=True, temperature=True, progress=True, remainingTime=True,
                                                  speed=True, totalSteps=1000, separator='\t'))
```