OpenMM Tutorial for Sugita Group, Wako

December 2, 2019

What is so special about OpenMM?

Optimized C++ + CUDA/OpenCL MD engine that requires only 1 GPU (+1 CPU master)

Powerful parsers for CHARMM, AMBER, and GROMACS-format topology files

Allows users to write custom forces and custom integrators *from python* that are bound to the engine in C++ via SWIG

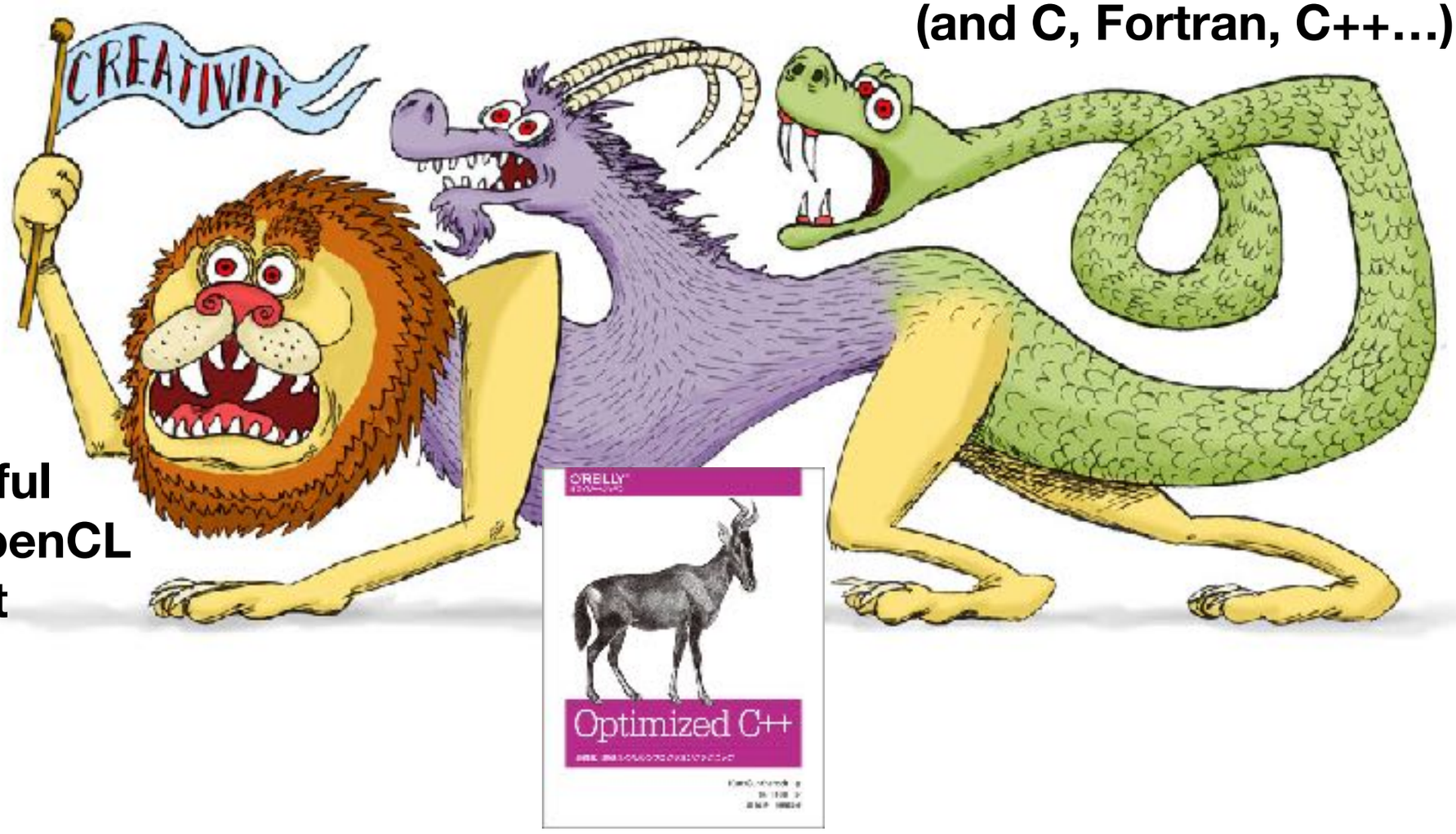
Allows users to manipulate simulations in python *during* simulation

Flexible and easy platform for testing new ideas

In addition to python, there are also APIs for C++, C, and Fortran

What makes OpenMM so great… also makes it difficult to make a <u>general</u> tutorial

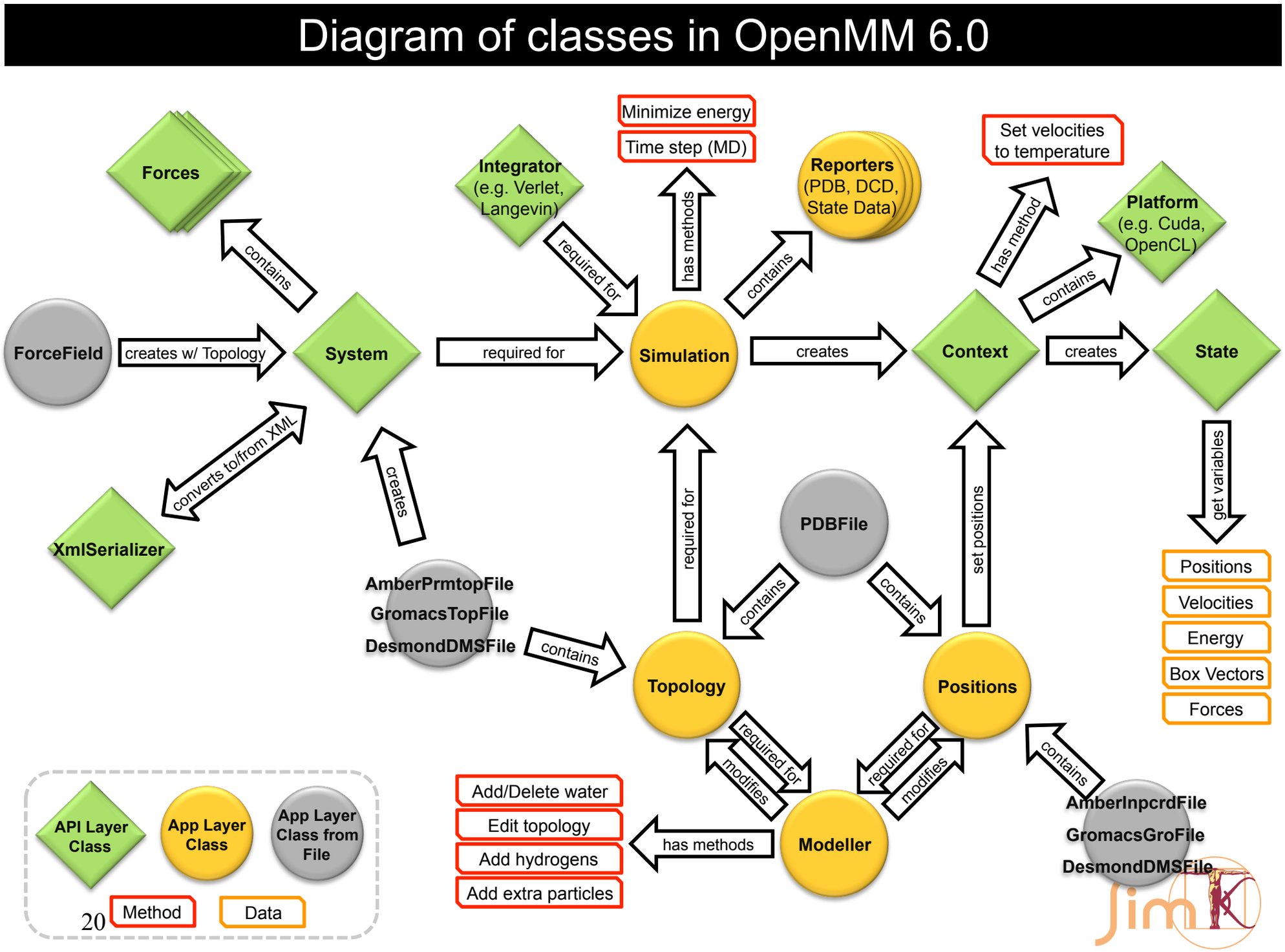
Extensive python API (and C, Fortran, C++…)



Powerful CUDA&OpenCL heart

The aim here is to learn how to utilize OpenMM on a basic level, and give a taste of the unique things you can do entirely from the python “head” of OpenMM.

On the level of python *classes*… OpenMM kind of looks like this



We will learn about how to use OpenMM via example applications

1) Conventional MD simulation using files prepared outside of OpenMM

Trp-Cage folding

Implicit Solvent: AMBER file topology + OBC2 model (GB-type implicit solvent)
Explicit Solvent: CHARMM, GROMACS (w/ CMAP) file topologies

2) Custom Forces, Performing restarts, and Storing Systems in XML Format

O_2 in water

Alchemical insertion of 1 O_2 to water

Performing a restart using XML files

Serializing systems to and loading from XML format

Harmonic position restraint
Harmonic flat-well position restraint

3) Explaining/discussing the CHARMM-GUI OpenMM control script

Sphingomyelin + Cholesterol lipid bilayer

We are going to use GPUs on KELP for these exercises

KELP's specs…

4 machines
4 GTX 1080 TI GPUs, 24 CPU each — 16 GPUs total

Copy the tutorial files to your own directory (SETUPNOTES.txt to copy-paste)

cp -r /home/gpantel/OpenMM_Tutorial /home/USERNAME/

Let's try to log in and individually choose unique GPUs to run on…

(1) ssh in to kelp (via RIKEN network or VPN). USERNAME@kelp
(2) ssh in to one of the FOUR kelp machines, kelp0[1-4]. e.g. ssh kelp04
(3) Select which GPU you want to use… each GPU is has an ID of 0, 1, 2, or 3

To make sure we are all using different GPUs…

enter your kelp + GPUID selection into a table we will write on the board

Setting up your environment in KELP to use OpenMM

You can copy-paste these from SETUPNOTES.txt in OpenMM_Tutorial

First set up our environment to use CUDA 10.0

```
export PATH=/usr/local/cuda-10.0/bin:/usr/local/cuda-10.0/NsightCompute-1.0${PATH:+:${PATH}}
export LD_LIBRARY_PATH=/usr/local/cuda-10.0/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
```

Set up a conda environment for python3.7 (your own python environment)

```
conda create -n py37 python=3.7
conda activate py37
```

Install OpenMM with pre-compiled binaries for use with CUDA 10.0

```
conda install -c omnia/label/cuda100 openmm
```

Test to make sure that OpenMM sees CUDA and has no errorrs

```
python -m simtk.testInstallation
```

Also install openmmtools. We will use it for building a water box.

```
conda install -c omnia openmmtools
```

(1) Trp-Cage folding in implicit solvent

OpenMM_Tutorial/trpcage_implicit

Following Simmerling's seminal paper[1] and AMBER basic tutorial #3[2]

NLYIQWLKDGGPSSGRPPPS

linear_trpcage.pdb

5 ns MD

[1] Simmerling, C., Strockbine, B. & Roitberg, A. E. "All-atom structure prediction and folding simulations of a stable protein." ***J. Am. Chem. Soc.*** **124, 11258–11259 (2002).**
[2] http://ambermd.org/tutorials/basic/tutorial3/

Let's discuss how OpenMM works via “trpcage_implicit/implicit_amber_tutorial.py”

OpenMM’s python modules are part of the "simtk" package

```
import simtk.openmm as mm # contains functions MD work
import simtk.openmm.app as app # contains functions for i/o
from simtk import unit # controls unique object types for physical units
```

First we initialize the system

Parse the topology

```
# app.AmberPrmtopFile parses the system topology and force field parameters, constructing an object
# We'll store the constructed object as "forcefield"
# there are other topology file parsers in app: GromacsTopFile, CharmmPsfFile+CharmmParameterSet
# these parsers actually quite work well (except for restraints and constraints)
print('Parsing system topology')
forcefield = app.AmberPrmtopFile('ff99sb_jacs2002.prmtop')
```

Parse the coordinates

```
# app.PDBfile parses the system coordinates, constructing an object.
# We'll store the constructed object as "coord"
# there are several other parses in app: AmberInpCrd, CharmmCrdFile, GromacsGroFile
print('Parsing system coordinates')
coord = app.PDBFile('linear_trpcage.pdb')
```

Parameterize the system

```
# forcefield.createSystem constructs an object containing the complete force field parameters of the
system
# We can actually modify "system" after construction and even between MD steps
print('Constructing sytem')
system = forcefield.createSystem(implicitSolvent=app.OBC2, soluteDielectric=1.0,
solventDielectric=78.5)
```

Let's discuss how OpenMM works via “trpcage_implicit/implicit_amber_tutorial.py”

Next we initialize the MD engine and link it to the system, forming a “context” which belongs to a “simulation,” which controls I/O.

Initialize the integrator

```
# the integrator object is only the thermostat+MD integrator
# There is only a Langevin and Andersen thermostat in OpenMM in the pre-compiled distribution
# However, you can define custom integrators (more on this later)
print('Constructing integrator')
integrator = mm.LangevinIntegrator(325*unit.kelvin, 1.0/unit.picosecond, 2.0*unit.femtosecond)
```

ref. temperature **friction** **timestep**

Initialize the MD platform

```
# mm.Platform selects the MD engine to use. There are engines for CPU, CUDA, and OpenCL
# CUDA is the most-optimized MD engine
print('Selecting MD platform')
platform = mm.Platform.getPlatformByName('CUDA')

# We can set the precision scheme to use for MD.
# "single": Nearly all work is done with single precision.
# "mixed": Forces are in single precision. Integration steps are in double precision.
# "double": All work is done with double precision. GPUs are bad at this. Can't really use it.
# "mixed" is generally fast and stable. I always use "mixed".
properties = {'CudaPrecision': 'mixed'}

# We should also specify which GPU we want to use for the simulation
properties["DeviceIndex"] = "0"
```

on kelp0[1-4] one GPU each corresponds to “DeviceIndex” 0, 1, 2, and 3

Let's discuss how OpenMM works via “trpcage_implicit/implicit_amber_tutorial.py”

Now we construct the simulation and set the initial condition

```
simulation = app.Simulation(topology, system, integrator, platform, properties)

print('Setting initial condition')
simulation.context.setPositions(coord.positions)
simulation.context.setVelocitiesToTemperature(325*unit.kelvin)
```

Then add “reporters” to the simulation to write out thermodynamic data and the trajectory coordinates.

```
# let's decide here now long we want to run the simulation and the file writing period
mdsteps    =  2500000  #  5  ns at 2 fs timestep
dcdperiod  =    50000  #  10 ps at 2 fs timestep
logperiod  =    50000  #  10 ps at 2 fs timestep

# now we will set up "reporters" to write out thermodynamic data and coordinates
# you can choose which system variables you want to have output from app.StateDataReporter
print('Setting reporters')
from sys import stdout # we'll use this to print output to the terminal during simulation
simulation.reporters.append(app.StateDataReporter(stdout, logperiod, step=True,
    time=True, potentialEnergy=True, kineticEnergy=True, totalEnergy=True,
    temperature=True, progress=True, volume=True, density=True,
    remainingTime=True, speed=True, totalSteps=mdsteps, separator='\t'))
simulation.reporters.append(app.StateDataReporter('trpcage_implicit.log', logperiod, step=True,
    time=True, potentialEnergy=True, kineticEnergy=True, totalEnergy=True,
    temperature=True, progress=True, volume=True, density=True,
    remainingTime=True, speed=True, totalSteps=mdsteps, separator='\t'))
simulation.reporters.append(app.DCDReporter('trpcage_implicit.dcd', dcdperiod))
```

Let's discuss how OpenMM works via “trpcage_implicit/implicit_amber_tutorial.py”

Now we minimize the system

```
# now let's minimize the system to convergence (steepest descent)
print('Minimizing')
simulation.minimizeEnergy()

# let's save the minimized structure as a PDB by
# (1) exctracting the system positions from the context
# (2) Using app.PDBfile to write a PDB of the system with these positions
minpositions = simulation.context.getState(getPositions=True).getPositions()
app.PDBFile.writeFile(forcefield.topology, minpositions, open('trpcage_implicit_min.pdb', 'w'))
```

And run the MD simulation. The reporters will take care of the output.

```
# and now we run the simulation for "mdsteps" number of steps
print('Running MD')
simulation.step(mdsteps)

# just for this tutorial, let's also save a PDB of the last frame of the simulation
lastpositions = simulation.context.getState(getPositions=True).getPositions()
app.PDBFile.writeFile(forcefield.topology, lastpositions, open('trpcage_implicit_last.pdb', 'w'))
```

(1) Trp-Cage simulation w/ CHARMM and GROMACS format

OpenMM_Tutorial/trpcage_explicit

We will simulate a collapsed conformation of Trp-Cage (after 5 ns of simulation in implicit solvent) in explicit water + 150mM NaCl

We will apply a cutoff in non-bonded interactions

We will add a barostat to do NPT simulation

We will anneal the temperature from 5 to 325 K

We will use **CHARMM36** with
CHARMM and GROMACS-format topology files

Let's simulate Trp-Cage using “trpcage_explicit/charmm/explicit_charmm_tutorial.py”

And discuss the changes from “trpcage_implicit/implicit_charmm_tutorial.py”

```
# input topology, psf, and force field files generated from CHARMM-GUI Solution Builder
print('Parsing system topology')
topology = app.CharmmPsfFile('step3_charmm2omm.psf')
parameters = app.CharmmParameterSet('top_all36_prot.rtf',\
             'par_all36_prot.prm' ,'toppar_water_ions.str')

print('Parsing system coordinates')
# for using PBC in OpenMM, we need to make sure that
# the origin of the sytem is at (0,0,0)
# and that the extremum of the system is at (Lx, Ly, Lz)
coord = app.PDBFile('step3_pbcsetup.pdb')
# translate the coordinates, we'll use numpy here.
import numpy as np
xyz = np.array(coord.positions/unit.nanometer)
xyz[:,0] -= np.amin(xyz[:,0])
xyz[:,1] -= np.amin(xyz[:,1])
xyz[:,2] -= np.amin(xyz[:,2])
coord.positions = xyz*unit.nanometer



print('Constructing sytem')
# set periodic box vectors
topology.setBox(7.5*unit.nanometer, 7.5*unit.nanometer, 7.5*unit.nanometer)
# use PME for long-range electrostatics, cutoff for short-range interactions
# constrain H-bonds with RATTLE, constrain water with SETTLE
system = topology.createSystem(parameters, nonbondedMethod=app.PME,
    nonbondedCutoff=1.2*unit.nanometers, constraints=app.HBonds, rigidWater=True,
    ewaldErrorTolerance=0.0005)
```

To prepare the system, we will add a barostat to perform NPT simulation and anneal it from 5 K to 325 K over 130 ps

```
print('Constructing and adding Barostat to system')
barostat = mm.MonteCarloBarostat(1.0*unit.bar, 5.0*unit.kelvin, 25)
system.addForce(barostat)
```

target pressure, temperature for metropolis, MC attempt period

```
.
.
.
simulation = app.Simulation(topology, system, integrator, platform, properties)
print('Setting initial condition')
simulation.context.setPositions(coord.positions)
# we'll set the initial temperature to 5 K, before we do simulated annealing
simulation.context.setVelocitiesToTemperature(5*unit.kelvin)

mdsteps   = 65000 # 100 ps at 2 fs timestep
dcdperiod =   500 # 100 fs at 2 fs timestep
logperiod =   500 # 100 fs at 2 fs timestep
.
.
.
```

Changes can be made to the simulation between steps.
Here, to change the temperature by +5 K every 500 steps, we use a *for* loop 65 times:

```
print('Running Simulated Annealing MD')
# every 1000 steps raise the temperature by 5 K, ending at 325 K
T = 5
for i in range(65):
    simulation.step( int(mdsteps/65) )
    integrator.setTemperature( (T+(i*T))*unit.kelvin )
    barostat.setDefaultTemperature( (T+(i*T))*unit.kelvin )
```

We could also simulate Trp-Cage using "trpcage_explicit/gromacs/explicit_gromacs_tutorial.py" (same CHARMM36 parameters) with files generated by CHARMM-GUI

The main difference comes at the very beginning to construct the system:

```
# input topology, psf, and force field files generated from CHARMM-GUI Solution Builder
print('Parsing system topology and coordinates')
# charmm-gui does not generate a .gro file, but it can easily be prepared in gromacs via
# gmx editconf -f step3_charmm2gmx.pdb -box 7.5 7.5 7.5 -o step3_charmm2gmx.gro
# gmx editconf autmatically translates coordinates to be within (0,Lx), (0,Ly), (0,Lz)
coord    = app.GromacsGroFile('step3_charmm2gmx.gro')
topology = app.GromacsTopFile('topol.top', periodicBoxVectors=coord.getPeriodicBoxVectors())

print('Constructing sytem')
# use PME for long-range electrostatics, cutoff for short-range interactions
# constrain H-bonds with RATTLE, constrain water with SETTLE
system = topology.createSystem(nonbondedMethod=app.PME,
    nonbondedCutoff=1.2*unit.nanometers, constraints=app.HBonds, rigidWater=True,
    ewaldErrorTolerance=0.0005)
```

A critical difference: we *must* use the native GROMACS itp file for TIP3P: "toppar/TIP3.itp" must have these arguments included…

```
#ifndef FLEXIBLE
[ settles ]
; i     j       funct   length
1       1       0.09572 0.15139

[ exclusions ]
1       2       3
2       1       3
3       1       2
```

```
#else
[ bonds ]
; i     j       funct   length  force.c.
1       2       1       0.09572 502416.0 0.09572        502416.0
1       3       1       0.09572 502416.0 0.09572        502416.0


[ angles ]
; i     j       k       funct   angle   force.c.
2       1       3       1       104.52  628.02  104.52  628.02
#endif
```

(2) Alchemically inserting and positionally restraining O_2 in water

We will build the whole system in OpenMM without parsing any files

We will alchemically insert O_2 into TIP3P water

We will learn how to restart a simulation

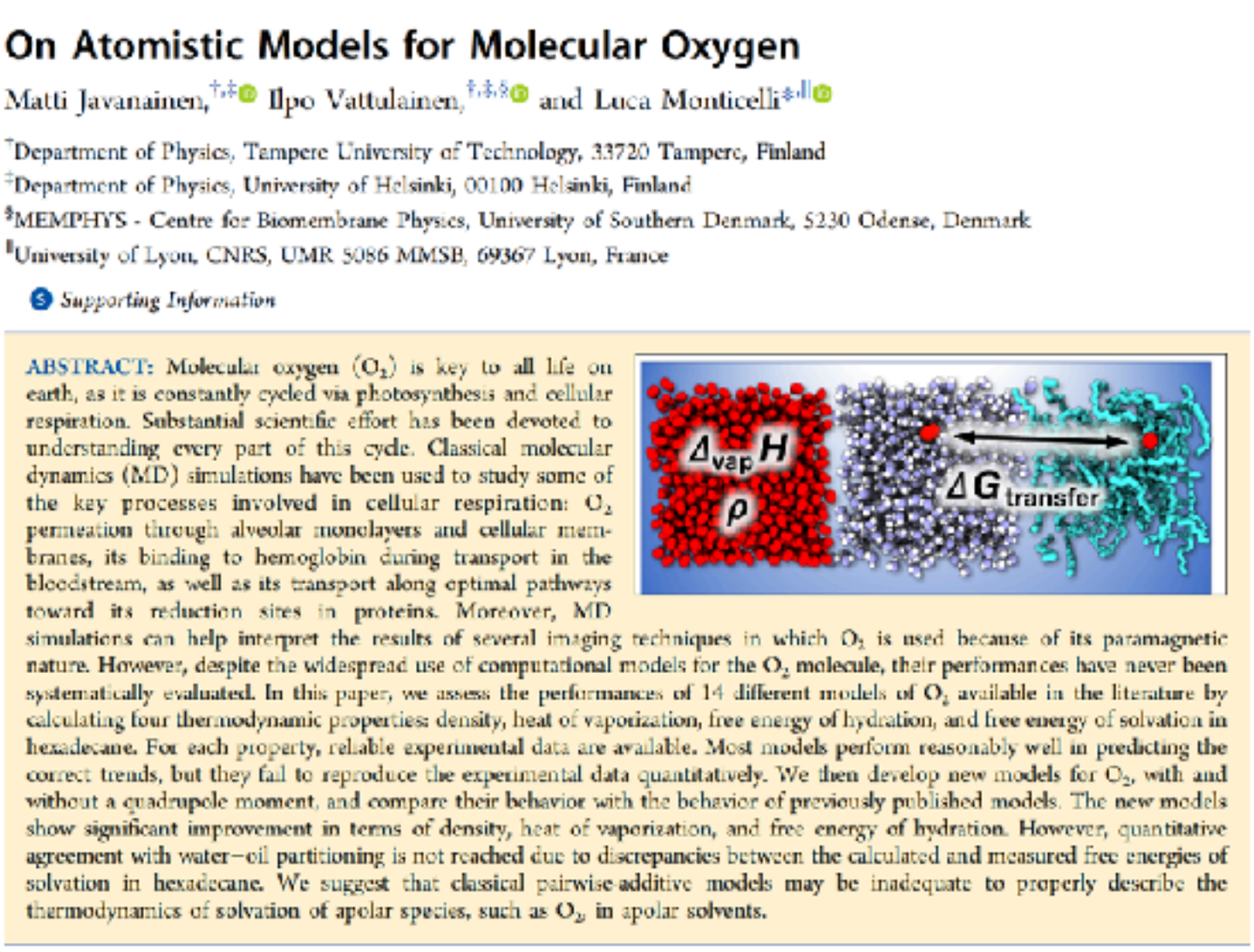
We will apply restraints on O_2

O_2 in water: 1728 waters

"noQ" model from Javanainen et al.[1]

ε: 0.4029 kJ/mol
σ: 0.313 nm
d: 0.1016 nm
constrained O-O bond

310 K, 1 atm NPT ensemble



On Atomistic Models for Molecular Oxygen

Matti Javanainen,[†,‡] Ilpo Vattulainen,[†,‡,§] and Luca Monticelli[*,∥]

[†]Department of Physics, Tampere University of Technology, 33720 Tampere, Finland
[‡]Department of Physics, University of Helsinki, 00100 Helsinki, Finland
[§]MEMPHYS - Centre for Biomembrane Physics, University of Southern Denmark, 5230 Odense, Denmark
[∥]University of Lyon, CNRS, UMR 5086 MMSB, 69367 Lyon, France

Supporting Information

ABSTRACT: Molecular oxygen (O_2) is key to all life on earth, as it is constantly cycled via photosynthesis and cellular respiration. Substantial scientific effort has been devoted to understanding every part of this cycle. Classical molecular dynamics (MD) simulations have been used to study some of the key processes involved in cellular respiration: O_2 permeation through alveolar monolayers and cellular membranes, its binding to hemoglobin during transport in the bloodstream, as well as its transport along optimal pathways toward its reduction sites in proteins. Moreover, MD simulations can help interpret the results of several imaging techniques in which O_2 is used because of its paramagnetic nature. However, despite the widespread use of computational models for the O_2 molecule, their performances have never been systematically evaluated. In this paper, we assess the performances of 14 different models of O_2 available in the literature by calculating four thermodynamic properties: density, heat of vaporization, free energy of hydration, and free energy of solvation in hexadecane. For each property, reliable experimental data are available. Most models perform reasonably well in predicting the correct trends, but they fail to reproduce the experimental data quantitatively. We then develop new models for O_2, with and without a quadrupole moment, and compare their behavior with the behavior of previously published models. The new models show significant improvement in terms of density, heat of vaporization, and free energy of hydration. However, quantitative agreement with water−oil partitioning is not reached due to discrepancies between the calculated and measured free energies of solvation in hexadecane. We suggest that classical pairwise-additive models may be inadequate to properly describe the thermodynamics of solvation of apolar species, such as O_2, in apolar solvents.

[1] Javanainen, M., Vattulainen, I. & Monticelli, L. "On atomistic models for molecular oxygen." J. Phys. Chem. B 121, 518–528 (2017).

We will construct a water box, add an O_2 to the system with alchemical LJ terms, and alchemically insert it “O2_in_water/insert/O2_alchemical_insertion.py”

```
import simtk.openmm as mm
import simtk.openmm.app as app
from simtk import unit
# we will use openmmtools for building a TIP3P water box
# this can be installed via conda:
# conda install -c omnia openmmtools
import openmmtools

# have openmmtools construct a fully-parameterized cubic 1728-water box
waterbox = openmmtools.testsystems.WaterBox(box_edge=26.2*unit.angstrom, cutoff=1.2*unit.nanometer,\
   model='tip3p', switch_width=2.0*unit.angstrom, constrained=True, dispersion_correction=True,
nonbondedMethod=app.PME)

# waterbox contains a "system" object that is fully ready for MD simulation, but we're going to
# forget about this for the moment, and totally parameterize the system from the ground-up in OpenMM
# we are using the coordinates and parameters
system = waterbox.system

# let's add the 1st and 2nd atom of O2 ("OA" and "OB") to the system
# we will insert it to (-0.0508, 0, 0), (0.0508, 0, 0)
import numpy as np
system.addParticle(15.999*unit.amu)
system.addParticle(15.999*unit.amu)
# the two particles we've added have indices 1728 and 1729.

positions_list = (waterbox.positions/unit.nanometer).tolist()
positions_list.append([-0.0508, 0.0, 0.0])
positions_list.append([0.0508, 0.0, 0.0])
positions = np.array(positions_list)*unit.nanometer
# and now let's transpose the position again s.t. it sits within (0,0,0),(Lx,Ly,Lz)
positions[:,0] -= np.amin(positions[:,0])
positions[:,1] -= np.amin(positions[:,1])
positions[:,2] -= np.amin(positions[:,2])
```

Adding constraints is super easy…
but adding to the system topology is awkward

```
# add a constraint between the 1st and 2nd atom of O2
system.addConstraint(1728, 1729, 0.1016*unit.nanometer)

# add O2 to the system topology
topology = waterbox.topology
topology.addChain('O2') # this is the 2nd Chain in the topology
O2chain = list(topology.chains())[-1] # select the 2nd chain object
topology.addResidue('O2', O2chain) # add a residue called 'O2' to the Chain 'O2'
O2residue = list(topology.residues())[-1]
topology.addAtom('OA', app.Element.getBySymbol('O'), O2residue)
topology.addAtom('OB', app.Element.getBySymbol('O'), O2residue)
```

We’re going to build a totally new “NonbondedForce” object to describe LJ+electrostatics in the system including O_2. We can use the parameters are already present in “waterbox”.

```
# we're going to pull the parameters for water out of the waterbox system to form the alchemical
part of the LJ interaction
# get the list of all Force objects in the sytem
forces = { force.__class__.__name__ : force for force in system.getForces() }
# pull out the NonbondedForce. We're going to use all of the parameter values in this.
waterbox_nbforce = forces['NonbondedForce']
```

Constructing a force that depends on a "λ" value for scaling soft-core non-bonded interactions of O with the rest of the system

$$4\epsilon\left[1-(1-\lambda)\right]\left[\left(\frac{1}{\alpha(1-(1-\lambda))}\right)^2+\left(\frac{r}{\sigma}\right)^{12}-\left(\frac{1}{\alpha(1-(1-\lambda))}\right)-\left(\frac{r}{\sigma}\right)^6\right]$$

```
# we will create a CustomNonbondedForce to represent all alchemical LJ interactions
# via this force, water-water interactions are zero, water-O2 interactions are scaled by lambda
# we will set the initial value of lambda to 0 -- no interaction between O2 and water
lambda_value = 0.0
alchemical_nbforce = mm.CustomNonbondedForce("""4*epsilon*l12*(1/((alphaLJ*(1-l12) + (r/sigma)^6)^2) - 1/
(alphaLJ*(1-l12) + (r/sigma)^6));
                                             sigma=0.5*(sigma1+sigma2);
                                             epsilon=sqrt(epsilon1*epsilon2);
                                             alphaLJ=0.5;
                                             l12=1-(1-lambda)*step(useLambda1+useLambda2-0.5)""")
alchemical_nbforce.addPerParticleParameter("sigma") # parameter #1
alchemical_nbforce.addPerParticleParameter("epsilon") # parameter #2
alchemical_nbforce.addPerParticleParameter("useLambda") # parameter #3. 1==Alchemical, 0==Not Alchemical
alchemical_nbforce.addGlobalParameter("lambda", lambda_value) # set the initial lambda to 0 (O2 turned off)
alchemical_nbforce.setNonbondedMethod(mm.NonbondedForce.CutoffPeriodic)
alchemical_nbforce.setCutoffDistance(1.2*unit.nanometer)
for particle_index in range(system.getNumParticles()):
    if particle_index in [1728, 1729]:
        # Add nonbonded LJ parameters for our O2
        sigma = 0.313*unit.nanometer
        epsilon = 0.4029*unit.kilojoule/unit.mole # unit.kilojoules_per_mole is exactly the same
        alchemical_nbforce.addParticle([sigma, epsilon, 1]) # this is alchemical
    elif particle_index not in [1728, 1729]:
        # Add nonbonded LJ parameters for a water
        sigma = waterbox_nbforce.getParticleParameters(particle_index)[1]
        epsilon = waterbox_nbforce.getParticleParameters(particle_index)[2]
        alchemical_nbforce.addParticle([sigma, epsilon, 0]) # this is not alchemical
system.addForce(alchemical_nbforce)
# use a switching function to smoothly truncate forces to zero from 10-12 angstroms
alchemical_nbforce.setUseSwitchingFunction(use=True)
alchemical_nbforce.setSwitchingDistance(2.0*unit.angstroms)
```

this step function makes sure nothing happens when useLambda = 0 for both atoms in an interaction

Constructing a normal NonbondedForce that describes water-water interactions

```
# and now we're going to build out a "normal", non-alchemical NonbondedForce.
# Even though O2 will not be part of this force, any NonbondedForce object
# must contain parameters for EVERY atom in the system, even if the parameters are zero
nbforce = mm.NonbondedForce()
nbforce.setNonbondedMethod(mm.NonbondedForce.CutoffPeriodic)
nbforce.setCutoffDistance(12.0*unit.angstroms)
nbforce.setUseSwitchingFunction(use=True)
nbforce.setSwitchingDistance(2.0*unit.angstroms)
# use the long-range dispersion correction for isotropic fluids in NPT
# Michael R. Shirts, David L. Mobley, John D. Chodera, and Vijay S. Pande.
# Accurate and efficient corrections for missing dispersion interactions in molecular simulations.
# Journal of Physical Chemistry B, 111:13052â€šÀÓ13063, 2007.
nbforce.setUseDispersionCorrection(True)
for particle_index in range(system.getNumParticles()):
    # set LJ parameters of each paricle
    if particle_index in [1728, 1729]:
        # O2 has 0 -- the only way it interats with water is via alchemical_nbforce
        charge = 0.0*unit.coulomb
        sigma = 0.0*unit.nanometer
        epsilon = 0.0*unit.kilojoule/unit.mole
        nbforce.addParticle(charge, sigma, epsilon)
    elif particle_index not in [1728, 1729]:
        # Add nonbonded LJ parameters for a water
        charge = waterbox_nbforce.getParticleParameters(particle_index)[0]
        sigma = waterbox_nbforce.getParticleParameters(particle_index)[1]
        epsilon = waterbox_nbforce.getParticleParameters(particle_index)[2]
        nbforce.addParticle(charge, sigma, epsilon)
system.addForce(nbforce)
```

We don’t need the NonbondedForce from “waterbox” anymore (it does not describe all atoms). We delete it based on the index in system.getForces()

```
# now let's delete the original normal NonbondedForce of the system (waterbox_nbforce)
system.removeForce(2)
```

Now we finish setting up the system as usual and minimize at λ=0

```
print('Constructing integrator')
integrator = mm.LangevinIntegrator(310*unit.kelvin, 1.0/unit.picosecond, 2.0*unit.femtosecond)

print('Constructing and adding Barostat to system')
barostat = mm.MonteCarloBarostat(1.0*unit.bar, 310*unit.kelvin, 25)
system.addForce(barostat)

print('Selecting MD platform')
platform = mm.Platform.getPlatformByName('CUDA')
properties = {'CudaPrecision': 'mixed'}
properties["DeviceIndex"] = "0"

print('Constructing simulation context')
simulation = app.Simulation(topology, system, integrator, platform, properties)

print('Setting initial condition')
simulation.context.setPositions(positions)
simulation.context.setVelocitiesToTemperature(310*unit.kelvin)

print('Minimizing with lambda=0')
simulation.minimizeEnergy()
minpositions = simulation.context.getState(getPositions=True).getPositions()
app.PDBFile.writeFile(topology, minpositions, open('O2_alchemical_insertion_min.pdb', 'w'))
```

And now we set up the reporters and perform a simulation in which we increase λ by 0.1 after every 1 ps of simulation.

```
print('Setting reporters')
mdsteps    = 500*11 #   1 ps per lambda condition at 2 fs timestep
dcdperiod =  500    #   1 ps at 2 fs timestep
logperiod =   50    # 0.1 ps at 2 fs timestep
from sys import stdout # we'll use this to print output to the terminal during simulation
simulation.reporters.append(app.StateDataReporter(stdout, logperiod, step=True,
    time=True, potentialEnergy=True, kineticEnergy=True, totalEnergy=True,
    temperature=True, progress=True, volume=True, density=True,
    remainingTime=True, speed=True, totalSteps=mdsteps, separator='\t'))
simulation.reporters.append(app.StateDataReporter('O2_alchemical_insertion.log', logperiod,
step=True,
    time=True, potentialEnergy=True, kineticEnergy=True, totalEnergy=True,
    temperature=True, progress=True, volume=True, density=True,
    remainingTime=True, speed=True, totalSteps=mdsteps, separator='\t'))
simulation.reporters.append(app.DCDReporter('O2_alchemical_insertion.dcd', dcdperiod))

# Now let's do the alchemical insertion... every 1 ps we'll increase lambda by 0.1
# We'll be able to indirectly observe the alchemical insertion via Total Energy over time
for i in range(10):
    print('Simulating for 1 ps at lambda=%f'%lambda_value)
    simulation.step(500)
    lambda_value += 0.1
    simulation.context.setParameter('lambda', lambda_value)
print('Simulating for 1 ps at lambda=%f'%lambda_value)
simulation.step(500)

lastpositions = simulation.context.getState(getPositions=True).getPositions()
app.PDBFile.writeFile(topology, lastpositions, open('O2_alchemical_insertion_last.pdb', 'w'))
```

Finally, let's try something of practical importance: saving the system parameters to initialize again later while avoid all of that messy set-up…

```
# Now we will save a serialization of this simulation into OpenMM's native XML format
# We can re-initialize the system later for further simulations without all of the
bothersome set-up by loading these files!
# We'll write exactly the same XML files Folding@home uses to transfer simulation data for
restarts to/from users
state = simulation.context.getState(getPositions=True, getVelocities=True, getForces=True,
getEnergy=True, getParameters=True, enforcePeriodicBox=True)

# system.xml contains all of the force field parameters
with open('O2_system.xml', 'w') as f:
     system_xml = mm.XmlSerializer.serialize(system)
     f.write(system_xml)
# integrator.xml contains the confiruation for the integrator, RNG seed
with open('O2_integrator.xml', 'w') as f:
     integrator_xml = mm.XmlSerializer.serialize(integrator)
     f.write(integrator_xml)

# state.xml contains positions, velocities, forces, the barostat
with open('O2_state.xml', 'w') as f:
     f.write(mm.XmlSerializer.serialize(state))




# there is also a binary "Checkpoint" file
# using a "Checkpoint" file only work on the same hardware+software combination.
simulation.loadCheckpoint('O2_state.chk')
```

We will push our O_2 in water to the center of the system via a linear restraint "O2_in_water/restrain/O2_center_restraint.py"

First, let's try using these XML files to make system setup really easy.

```
import simtk.openmm as mm
import simtk.openmm.app as app
from simtk import unit

# load up the system, integrator, and state. Easy!
system = mm.XmlSerializer.deserialize(open('../insert/O2_system.xml').read())
integrator = mm.XmlSerializer.deserialize(open('../insert/O2_integrator.xml').read())
state = mm.XmlSerializer.deserialize(open('../insert/O2_state.xml').read())

# we'll just take the topology from here...
pdb = app.PDBFile('../insert/O2_alchemical_insertion_last.pdb')
topology = pdb.topology

# let's specify our simulation platform again
platform = mm.Platform.getPlatformByName('CUDA')
properties = {'CudaPrecision': 'mixed'}
properties["DeviceIndex"] = "0"

# and we could reconstruct the simulation no problem, if we wanted to
#simulation = app.Simulation(topology, system, integrator, platform, properties)
#simulation.context.setState(state)
# but we won't do that yet. Let's make a restraint to pull O2 to the center of the system
```

Now let's push that O_2 to the center of the system

$$k\left(|x-x_0|+|y-y_0|+|z-z_0|\right)$$

```
# let's make the linear restraint, setting x0, y0, z0 to the center of the system
centerforce = mm.CustomExternalForce("k*(abs(x-x0)+abs(y-y0)+abs(z-z0))")
centerforce.addGlobalParameter("k", 5.0*unit.kilojoule/unit.angstrom/unit.mole)
centerforce.addPerParticleParameter("x0")
centerforce.addPerParticleParameter("y0")
centerforce.addPerParticleParameter("z0")
import numpy as np
xmean = np.mean(np.array(state.getPositions()/unit.nanometer)[:,0])*unit.nanometer
ymean = np.mean(np.array(state.getPositions()/unit.nanometer)[:,1])*unit.nanometer
zmean = np.mean(np.array(state.getPositions()/unit.nanometer)[:,2])*unit.nanometer
centerforce.addParticle(1728, mm.Vec3(xmean, ymean, zmean))
centerforce.addParticle(1729, mm.Vec3(xmean, ymean, zmean))
system.addForce(centerforce)

# ok now let's do some simulation using this restraint
simulation = app.Simulation(topology, system, integrator, platform, properties)
simulation.context.setState(state)

# set up reporters so we can see what's going on...
mdsteps   = 55000   # 110 ps total simulation
dcdperiod =    50   # 0.1 ps at 2 fs timestep
logperiod =    50   # 0.1 ps at 2 fs timestep
from sys import stdout # we'll use this to print output to the terminal during simulation
simulation.reporters.append(app.StateDataReporter(stdout, logperiod, step=True,
    time=True, potentialEnergy=True, kineticEnergy=True, totalEnergy=True,
    temperature=True, progress=True, volume=True, density=True,
    remainingTime=True, speed=True, totalSteps=mdsteps, separator='\t'))
simulation.reporters.append(app.StateDataReporter('O2_restraints.log', logperiod, step=True,
    time=True, potentialEnergy=True, kineticEnergy=True, totalEnergy=True,
    temperature=True, progress=True, volume=True, density=True,
    remainingTime=True, speed=True, totalSteps=mdsteps, separator='\t'))
simulation.reporters.append(app.DCDReporter('O2_restraints.dcd', dcdperiod))

# 5000 steps should be more than enough to pull O2 to the center
simulation.step(5000)
```

Now let's restrain O_2 to sit between (z_0-3Å,z_0+3Å)

$$k \max\left(0, \mid z - z_0 \mid - w\right)^2$$

```
# ok now let's remove centerforce from the system
simulation.system.removeForce(5)

# and now let's make a new force restraining O2 to a 6-angstrom flat-well harmonic potential
flatzforce = mm.CustomExternalForce('k * (pz^2); \
                                     pz = max(0, delta); \
                                     delta = r - width; \
                                     r = abs(periodicdistance(x, y, z, x, y, z0));')
flatzforce.addGlobalParameter('k', 1.0*unit.kilojoule/unit.angstrom**2/unit.mole)
flatzforce.addGlobalParameter('width', 0.3*unit.nanometer)
flatzforce.addPerParticleParameter('z0')
flatzforce.addParticle(1728, [zmean])
flatzforce.addParticle(1729, [zmean])
simulation.system.addForce(flatzforce)

# we're going to "reinitialize" the simulation context to totally remove "centerforce"
# if we don't do this, the last force of "centerforce" lingers in the system
# we also need to do this to now include "flatzforce" in the system
positions = simulation.context.getState(getPositions=True).getPositions()
velocities = simulation.context.getState(getVelocities=True).getVelocities()
simulation.context.reinitialize()
simulation.context.setPositions(positions)
simulation.context.setVelocities(velocities)

# Let's simulate O2 in this flat well for 100 ps
simulation.step(50000)
```

(3) Using and discussing the CHARMM-GUI OpenMM scripts (example: simulate a Sphingomyelin+Cholesterol membrane)

Composition:
35 water/lipid
9 Na, 9 Cl
80 PSM
20 Cholesterol
CHARMM36
22,152 atoms

Target condition:
310 K, 1 atm NPT

files in:
charmm-gui-SM+CHOL

CHARMM-GUI generates input scripts for OpenMM

It is intended to act like the input scripts of an normal MD package (like GENESIS) to **regain the clarity we often lose in OpenMM scripts**

Using an input script like this is much less stressful~

Let's look at an "input" script created to parse instructions to openmm_run.py

"charmm-gui-SM+CHOL/openmm/step6.1_equilibration.inp"

```
mini_nstep  = 5000              # Number of steps for minimization
mini_Tol    = 100.0             # Minimization energy tolerance

gen_vel     = yes               # Generate initial velocities
gen_temp    = 310               # Temperature for generating initial velocities (K)

nstep       = 25000             # number of steps to run
dt          = 0.001             # time-step (ps)

nstout      = 1000              # Writing output frequency (steps)
nstdcd      = 1000              # Writing coordinates trajectory frequency (steps)

coulomb     = PME               # Electrostatic cut-off method
ewald_Tol   = 0.0005            # Ewald error tolerance
vdw         = Force-switch      # vdW cut-off method
r_on        = 1.0               # Switch-on distance (nm)
r_off       = 1.2               # Switch-off distance (nm)

temp        = 310               # Temperature (K)
fric_coeff  = 1                 # Friction coefficient for Langevin dynamics

pcouple     = no                # Turn on/off pressure coupling

cons        = HBonds            # Constraints mehtod

rest        = yes               # Turn on/off restraints
fc_lpos     = 1000.0            # Positional restraint force constant for lipids (kJ/mol/nm^2)
fc_ldih     = 1000.0            # Dihedral restraint force constant for lipids (kJ/mol/rad^2)
```

And let's see how we just do the prescribed sequence of simulations for equilibration prior to production…

In "README" (generated by CHARMM-GUI)

The following csh script is included

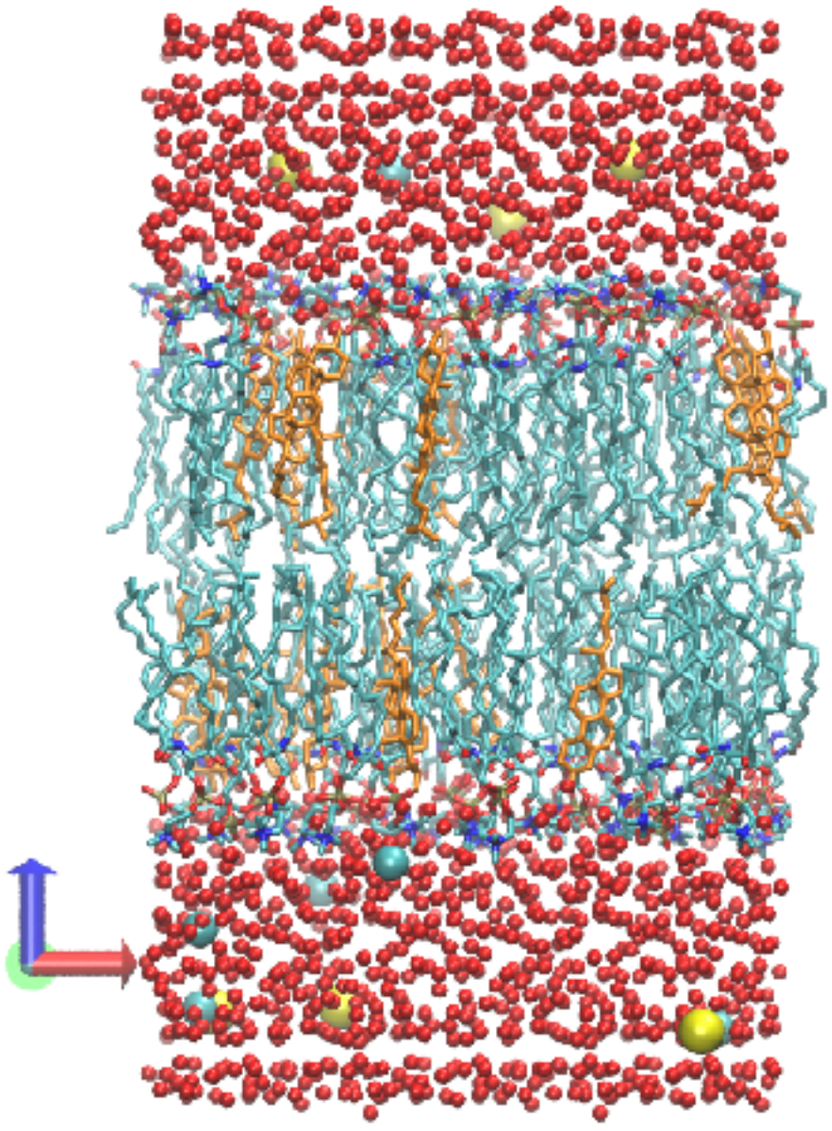
```
set init = step5_charmm2omm
set cnt = 1

while ( ${cnt} <= 6 )
    @ pcnt = ${cnt} - 1
    set istep = step6.${cnt}_equilibration
    set pstep = step6.${pcnt}_equilibration

    if ( ${cnt} == 1 ) then
        python -u openmm_run.py -i ${istep}.inp -t toppar.str -p ${init}.psf -c ${init}.crd -b ${init}.str -orst ${istep}.rst -odcd ${istep}.dcd > ${istep}.out
    else
        python -u openmm_run.py -i ${istep}.inp -t toppar.str -p ${init}.psf -c ${init}.crd -irst ${pstep}.rst -orst ${istep}.rst -odcd ${istep}.dcd > ${istep}.out
    endif
    @ cnt += 1
end
```

You could just run this with "csh README" and all pre-equilibration steps will run.

Let’s spend time looking & discussing openmm_run.py carefully (off slides)



As convenient as the CHARMM-GUI input script “openmm_run.py” is, there are a few things you need to be aware of

(1) The platform is set to “CUDA” at “single” precision

(2) The CUDA device index is not specified

(3) It does not write a log file using StateDataReporter. You could add this to openmm_run.py

(4) It reconstructs the system at every restart. This can cost several minutes of time in larger systems. You could modify openmm_run.py to read an XML file of the system, state, and integrator during restarts to save time.

Let’s consider the code for these three points in openmm_run.py…

‘single’ may be dangerous! I strongly suggest changing this to ‘mixed’, if you change anything.

```
# Set platform
platform = Platform.getPlatformByName('CUDA')
prop = dict(CudaPrecision='single')

# Production
if inputs.nstep > 0:
    print("\nMD run: %s steps" % inputs.nstep)
    if inputs.nstdcd > 0:
        if not args.odcd: args.odcd = 'output.dcd'
        simulation.reporters.append(DCDReporter(args.odcd, inputs.nstdcd))
    simulation.reporters.append(
        StateDataReporter(sys.stdout, inputs.nstout, step=True, time=True, potentialEnergy=True,
temperature=True, progress=True,
                          remainingTime=True, speed=True, totalSteps=inputs.nstep, separator='\t')
    )
    simulation.step(inputs.nstep)
```

sys.stdout prints results to the terminal… could change this or add a second “StateDataReporter” which outputs to some file, like “output.log” or whatever

The only XML file being written here is for “state”. This could be done for “system” and “integrator” just once, then loaded at the start to save hours of time for long, multiple-restart production runs.

```
# Write restart file
if not (args.orst or args.ochk): args.orst = 'output.rst'
if args.orst:
    state = simulation.context.getState( getPositions=True, getVelocities=True )
    with open(args.orst, 'w') as f:
        f.write(XmlSerializer.serialize(state))
if args.ochk:
    with open(args.ochk, 'wb') as f:
        f.write(simulation.context.createCheckpoint())
if args.opdb:
    crd = simulation.context.getState(getPositions=True).getPositions()
    PDBFile.writeFile(psf.topology, crd, open(args.opdb, 'w'))
```

(4) Other resources for Understanding how to use OpenMM…

Note there is no truly complete manual for OpenMM

(1) The online documents**: http://openmm.org/documentation.html**

(2) The source code, usually under:**https://github.com/openmm/openmm/tree/master/openmmapi/include/openmm**

(3) The OpenMM forums: **https://simtk.org/plugins/phpBB/indexPhpbb.php?group_id=161&pluginname=phpBB**

(4) Resolved OpenMM GitHub issues: **https://github.com/openmm/openmm/issues**

(5) *openmmtools*: **https://openmmtools.readthedocs.io/en/0.18.1/#**

(6) Examples from other people, often via GitHub (e.g. my “LJsimulator”: **https://gpantel.github.io/computational-method/LJsimulation/)**