It is in your best interest to read through the rest of this document and schedule your time accordingly. Some exercises take much longer than others, and you may wish to skip one to better focus your efforts on another task. Here is the overall checklist detailing what should be turned in.

- Exercise 1
  - ☐ `ex1.pdf/docx`, A pdf or word document answering the questions associated with exercise 1.
  - ☐ `ex1-1.log`, Logfile generated from the first simulation.
  - ☐ `ex1-2.log`, Logfile generated from the second simulation.
  - ☐ `ex1-3.log`, Logfile generated from the third simulation.
- Exercise 2
  - ☐ `ex2.sh`, the shell script used to submit the replica-exchange NAMD run.
  - ☐ `ex2.log`, Logfile generated from the simulation.
  - ☐ `alanin.abf.czar.pmf`, the final PMF file generated by NAMD. To assess convergence, you may wish to plot the file using xmgrace or another plotting software.
- Exercise 3
  - ☐ `ex3.sh`, the shell script used to submit the replica-exchange NAMD run.
  - ☐ `ex3.log`, Logfile generated from one of the replicas within the run (use +stdout option to split apart the replica logs).
  - ☐ `tidata.npy`, an intermediate results file generated by the provided `TIprep.py` script.
  - ☐ tiplot.png, the final result output plot generated by the `TIcalc.py` script.
- Exercise 4 **(SCC teams only!)**
  - ☐ `ex4.sh`, the shell script used to submit the replica-exchange NAMD run.
  - ☐ `ex4.log`, Logfile generated from one of the replicas within the run (use +stdout option to split apart the replica logs).
  - ☐ `bayesresult.pkl`, an intermediate results file generated by the `BayesWHAM.py` script, and generated automatically by `analysis.sh`.
  - ☐ pmf.png, the final result output plot generated by the `plotBayesWHAM.py` script, and generated automatically by `analysis.sh`.
- BenchmarkTest
  - ☐ `apoa1.log`, the logfile from running the apoa1 benchmark.
  - ☐ `f1atpase.log`, the logfile from running the f1atpase benchmark.
  - ☐ `bmc.log`, the logfile from running the bmc benchmark.
  - ☐ `1stmv.log`, the logfile from running the stmv benchmark with approximately 1 million atoms.
  - ☐ `20stmv.log`, the logfile from running the larger version of the stmv benchmark with 20 million atoms.

1 Exercise 1

The goal for this exercise is to quantify how physical properties from a simulation depend on various parameters passed to NAMD. We will be analyzing multiple relatively short simulations for a box of water, where we can easily evaluate real quantities that could in principle be measured in the laboratory. By comparing these experimental values to simulations, we can assess the accuracy for our methods. There are three simulations we will be running for this exercise:

1. A simulation with standard 12 Å non-bonded cutoffs and 2 fs timesteps enabled by rigidbonds.
2. A simulation with shorter 9 Å non-bonded cutoffs and 2 fs timesteps enabled by rigidbonds.
3. A simulation with standard 12 Å non-bonded cutoffs and 1 fs timesteps.

The provided `run.namd` file within exercise 1 is set to run each simulation for 1 ns, corresponding to the first simulation. You will need to adjust the file to run the other two simulations, with potential optimizations to improve performance. If you run short on time, note that the analysis scripts provided can also work with shorter simulations. You WILL need to save the results from standard output, typically achieved by standard output redirection mechanisms.

Questions

1. In nanoseconds of simulation per day, how fast is the first simulation, with 12 Å cutoffs and rigidbonds?
2. In nanoseconds of simulation per day, how fast is the second simulation, with 9 Å cutoffs and rigidbonds?
3. In nanoseconds of simulation per day, how fast is the third simulation, with 12 Å cutoffs and no rigidbonds?

1.1 Calculating Heat Capacity

The heat capacity measures how much thermal energy is required to raise the temperature for a material. The heat capacity at constant volume (C_v) can be computed directly from simulation:

$$C_v = \frac{\langle E^2 \rangle - \langle E \rangle^2}{k_B T^2} \tag{1}$$

Here, E is the total energy of the system, k_B is Boltzmann's constant, and the angle brackets refer to taking the average over a trajectory. Using the provided `heatcapacity.py` script, which uses the pynamd library[1] to read in the energy output printed to standard output, and then calculates a heat capacity. This is a classical experiment, first credited to James Joule, who was a brewer in the 19th century that worked out the equivalence between mechanical work and heat. A calorie of mechanical work would raise the temperature for a gram of water one degree Celsius. Thus, the heat capacity of water is 1 calorie per gram per degree Celsius.

Questions

1. How close is your estimate for the heat capacity to this experimental value under standard simulation conditions?
2. Is the heat capacity value consistent across the whole trajectory? (Hint: modify the script to also report values for each half of the trajectory independently)
3. When the cutoff is reduced, what is your new estimate for the heat capacity?
4. Does the timestep used change the heat capacity?

1.2 Calculating Water Diffusion Coefficient

Einstein won a Nobel prize back in 1926 on Brownian motion, which describes how particles diffuse randomly over time. The basic quantity of interest is a diffusion coefficient, a measure of how far particles would be expected to move as a function of time. We can use the relationship Einstein determined to calculate the diffusion coefficient (D) from our simulation.

$$D = \frac{MSD}{2dt} \tag{2}$$

[1] If needed, see `https://github.com/radakb/pynamd/pull/12`

The mean squared displacement (MSD) will grow linearly over time (t), depending on the dimensionality (d) for the random walk. Our water can diffuse in three dimensions, so the slope for a plot between mean squared displacement vs. time is related to 6D.

This is implemented in `diffusion.py`, which uses MDAnalysis to compute the diffusion coefficient by reading in the dcd trajectory files that track the position for the atoms over time. It is well known that the water model that we are using does not match experimentally observed self-diffusion coefficients of $2.3 \times 10^{-9} \mathrm{m}^2\mathrm{s}^{-1}$.

Questions

1. What is the diffusion coefficient measured under standard simulation conditions with a 12 Å cutoff?
2. What is the diffusion coefficient measured with a 9 Å cutoff?
3. What could you say about when a shorter cutoff might be appropriate based on your results so far?

2 Exercise 2

Physical properties for materials are one thing molecular dynamics can be used for, but NAMD is far more commonly used in biological research where we are interested in specifics about nanoscale behavior for proteins. One fascinating question that we still don't have the full answer is how protein folds. In this exercise, we will study the folding of a toy model protein deca-alanine in water. When fully-folded, it exists as a α-helix. By using an enhanced sampling technique called extended-system adaptive biasing force (eABF), we can compute how the free energy of the system changes when the deca-alanine folds and unfolds. In eABF, such free energy change is represented by the potential of mean force (PMF) along a pre-defined collective variable (CV), which has been chosen to be the α-helical content of the protein (α) in this exercise.

The NAMD input files needed to run the eABF simulation are provided for you. Conveniently, NAMD computes the PMF on-the-fly every 0.1 ns, and you can refer to the generated file `alanin.abf.czar.pmf` for these results. No additional analysis is required, however you can review `alanin.abf.hist.czar.pmf` to observe the evolution of the PMF over time. The `czar.pmf` files are human readable, and are formatted such that a tool like xmgrace could plot it. When the simulation reaches convergence, you should see minimal changes of the PMF over time, indicating sufficient sampling along the chosen CV.

At convergence, the PMF should reveal two energy minima, one at α= 0.2, representing the unfolded state, and another at 0.9, representing the folded state. The unfolded state has higher entropy, while the folded state has higher enthalpy, leading to two distinct energy wells on the PMF. The energy difference between them is expected to be around 1.2 kcal/mol, with the folded state having higher energy. The energy barrier, represented by the highest point between the two energy minima on the PMF curve, is around 2.3 kcal/mol. Your result will be judged based on the accuracy of these two energy values. Please note that this simulation may take hundreds of nanoseconds to converge, so effective resources and time management is crucial. To help you along this path, this particular simulation uses hydrogen mass repartitioning to slow down bond vibrations by enough to allow for 4 fs timesteps, rather than the typical 2 fs timesteps enabled by SHAKE/RATTLE.

3 Exercise 3

One advantage of using molecular simulation rather than experiment is that we can ask really cool what-if questions. In my own research, one of these what-if questions studied a small antibiotic transport protein, EmrE, which confers resistance to small polyaromatic cations by pumping them out of the cell if they diffuse in. A key question raised by experiment was determining which of two glutamate residues inside the transport protein gets protonated first. Experimentally, this would be tricky to measure, since only a single hydrogen is being added to the protein, which is a very small signal to detect.

However, in simulation, we can do an unphysical alchemical transformation, where the proton is moved in stages from one glutamate to the other. By tracking the free energy change as a function of this slow transition, we can determine which glutamate is likely to be protonated first. The way we will do that

here is to use thermodynamic integration, where the free energy contributions are estimated from many intermediate contributions, and then integrated along an alchemical reaction coordinate.

The first step here is to run 19 different simulations simultaneously, using the replica exchange feature of NAMD to occasionally communicate between individual simulations to see if a swap along the alchemical reaction coordinate should occur, which can improve sampling and accelerate convergence. The `replicaconfig.namd` file sets up the exchanges, and should be run as 19 replicas from a single mpirun or srun command in NAMD from within a batch script submitted onto your cluster. If you cannot get 19 replicas to run simultaneously, you can consider editing the `lambvals` and `numreplicas` parameter in replicaconfig.namd to a cheaper alternative that is commented out. The default settings will generate output into the replicas directory, with each replica generating data in the directories labeled 0-18. The log for one of these replicates should be submitted to assess your simulation performance and measure how many independent samples you could collect in the allotted time. Note that the replicaconfig.namd predates the ability to run alchemical simulations on the GPU, and so adding in GPU-specific flags may dramatically improve performance if you chose to run this on the GPU.

The analysis performed on the simulation outputs takes the form of `TIprep.py`, which aggregates data from the multiple .tiout files in the different replicas to generate a tidata.npy file, which should be turned in. The intermediate `tidata.npy` file can then be plotted and analyzed by `TIcalc.py`. For this system, the published free energy difference between the two glutamates is 1.53 ± 0.12 kcal/mol, which required 10 ns of sampling to achieve. It is anticipated that 1 ns of trajectory would yield an answer in a similar range, albeit with a larger error estimate. Note that `TIcalc.py` uses matplotlib, numpy, and scipy libraries, which may need to be installed.

4 Exercise 4 (SCC only, not for IndySCC teams!)

Free energy is a really important concept, and one that can take multiple flavors. For those of you in the SCC, exercise 4 is designed to challenge your scaling even further to compute drug permeation through a biological membrane. The key observable when computing permeability is the free energy profile, which we compute through the analysis scripts called by `analysis.sh`. The individual python scripts will gather trajectories together, compute the potential of mean force, and plot the free energy profile computed by BayesWHAM. You can try this out right now, as I have already provided the output of 1 ns of trajectory, split into 200 picosecond and a 800 picosecond fragments. The figure you get from this limited sampling is quite misleading, as it greatly overestimates the free energy in solution. Your task is to increase the sampling 5-fold, running an additional 4 ns of simulation across the 64 replicas and comparing the result. You can accomplish this by submitting run.namd to your cluster, taking care to map GPUs to multiple tasks.

To facilitate this, an example submission script is provided in `submit.sh`. This script demonstrates how I map the 64 tasks onto 16 V100 GPUs here at Michigan State, yielding roughly 15 nanoseconds of simulation time per day based on the benchmark times in the logfiles. Now, typically accurate free energies can take multiple tens of nanoseconds to run, and you don't have that much time or hardware to throw at the problem. The assessment will be to compare your free energy profile against the result after 50 nanoseconds of simulation time, with points awarded based on the free energy difference between the minimum and the center of the membrane at 0, and the difference between the minimum and solution at 55. It is completely up to your best judgement of how long to run this calculation to best match the result of a much longer simulation.

5 Benchmark Test

Whereas replica exchange simulations scale across nodes easily through infrequent communication, this exercise will push the boundaries of what is possible by asking NAMD to collect benchmarks on larger and larger systems. Generally, these would be run on hardware commensurate with the challenge, rather than only a handful of nodes, and so the benchmarks are unlikely to be world-beating. However, this does offer the opportunity to sample from across biology different systems. These happen to have different length scales, ranging from around 100,000 atoms (the ApoA1 system we studied in the homework), to 300,000 atoms (the F1 ATPase) to roughly a million atoms (the tobacco mosaic virus), to full bacterial microcompartments

that are several million atoms. Your mission, simply put, is to simulate these systems as quickly as possible, changing whatever is required in the NAMD configuration files provided (except the nonbonded cutoff, those must remain at their defaults!) or whatever secret sauce your NAMD build requires to achieve maximum performance on your hardware. Note that the simulation must execute to completion to be scored.

The scoring for this exercise, unlike the previous two, is a direct competition, where a perfect score is determined by the highest performance (measured in ns/day) determined from a NAMD logfile submitted by one of the teams. The score for other teams will be directly proportional to their measured performance for the same standard benchmark. As a concrete example, if the best performance for a team on the given hardware is 100ns/day, and another team only achieves 80ns/day, the second team would receive a score of 80%.

There are five simulations to be run. In order of size, these are apoa1 (`apoa1.namd`), the f1atpase (`f1atpase.namd`), a bacterial microcompartment (`run.namd`), and two stmv-derived systems. The smaller stmv, run with `1stmv2fs.namd`, is only 1 million atoms large. The larger stmv, run with `20stmv2fs.namd`, is approximately 20 million atoms large, and may prove to be too large to run without using a memory-optimized build of NAMD.