0.0 Using these instructions

Terminal commands that you can use (or potentially even copy-paste) are denoted using the following formatting:

```
this is an example terminal command
```

This homework was tested on a CC-supported Ubuntu 22 image on a CHI@TACC Haswell node. **You will not be graded on compute performance for this homework**, but please run this on a machine with at least 48 threads available.

1.0 Installing GROMACS

GROMACS can be compiled from source, but can also be installed using mambaforge, which is a far simpler method. However, in order to use mambaforge, we need to install that first!

1.1 Installing Mambaforge

Conda is a method of managing both python packages and python environments, that has since been co-opted as a general package-manager/environment manager. Mamba and Mambaforge are re-implementations of conda and condaforge that are significantly faster. You can get a brief overview of mambaforge here.

1. On the terminal, download mambaforge using `wget`:
   `wget https://github.com/conda-forge/miniforge/releases/latest/download/Mambaforge-Linux-x86_64.sh`
   (apologies, the font is small to accommodate the long link)
2. Give Mambaforge-Linux-x86_64.sh executable permissions, and then run the executable:
   ```
   chmod +x Mambaforge-Linux-x86_64.sh
   ./Mambaforge-Linux-x86_64.sh
   ```
3. The default options for Mambaforge should be fine for what we are doing, so just say yes to everything, including asking Mambaforge to run conda init
4. Once everything is installed, close your terminal, and then re-open it to make a new session.
5. Your command prompt may change to indicate that you are now in the base environment. We don't want this behavior by default, so let's change that with this command:
   ```
   conda config --set auto_activate_base false
   ```
6. You can close and re-open your terminal again, and you should no longer automatically load the base environment.

1.2 Installing GROMACS using Mamba

The recommended best practice for installing packages using Mamba is to not install things in the base environment, but instead create a new environment, and install things there.

1. Let's create an environment called gromacs_2023.1, and then activate the environment:
   ```
   mamba create -n gromacs_2023.1
   mamba activate gromacs_2023.1
   ```
2. Your command prompt should now show that you are in the gromacs_2023.1 environment. In this environment, let's install a specific version of GROMACS, as well as ACPYPE (we'll discuss this later), and let's set our python version to version 3.9:
   ```
   mamba install gromacs=2023.1 acpype python=3.9
   ```

3. After installing GROMACS (and every time you will activate your gromacs_2023.1 environment), you'll get a warning that no ICDs were found. As we won't be using GPU accelerators during this event, you can disregard this warning.
4. GROMACS should be installed now, but we can confirm it by using the `which` command:
   `which gmx`
   The executable to run gromacs is called gmx, and the which command will provide the location of that executable. If we didn't have GROMACS installed (or it's not visible in our environment), we would get a message such as `which: no gmx in (.....`.

2.0 Creating a Protein+Ligand System using GROMACS

GROMACS contains a number of modules, which are listed in a somewhat confusing way in the man pages (access with the command `man gmx`). If you need help understanding the inputs for a particular module, you can access the help menu for it with `gmx <your-module-of-interest> -h` . Other resources for understanding GROMACS include the documentation, which has user guides and short how-to guides. Justin Lemkul has also created a set of good tutorials for GROMACS which you can access here.

2.1.0 Acquiring the necessary input files

Let's use the Focal Adhesion Kinase (FAK) as an example. We can download the structure of FAK bound to an inhibitor directly from the RCSB Protein Data Bank.

1. Make a new directory where we can try running gromacs, and go into it.
   `mkdir fak_example`
   `cd fak_example`
2. Our structure of interest has a PDB ID code of 4I4F (the code is caps-insensitive). We can download the crystal structure from the Protein Data Bank using `wget`:
   `wget https://files.rcsb.org/download/4i4f.pdb`
   You should see a new file called `4i4f.pdb`
3. The crystal structure will contain the protein, ligand, crystallographic waters, and sometimes other solvent molecules (in this case, isopropanol). PDB files are human-readable, so I invite you to use your favorite text editor to check it out yourself. Because we are only interested in the protein and ligand, let's isolate those parts.
   a. The ligand residue is labeled as '1BR' in the crystal structure. We can make a file with only the ligand like so:
      `grep 1BR 4i4f.pdb > 4i4f_ligand.pdb`
   b. The protein is a little tricky to get on it's own, here, we just get rid of all of the residues in the PDB file that don't correspond to our protein (using `grep -v`)
      `grep -v HOH 4i4f.pdb | grep -v IPA | grep -v 1BR > 4i4f_protein.pdb`
   c. You should have two new files `4i4f_ligand.pdb` and `4i4f_protein.pdb`.

2.1.1 Parameterizing the Protein

In molecular dynamics, we need to define ahead of time how different types of atoms (atomtypes) will interact with each other. This is described in what is called the "forcefield". In this step, we will parameterize both the protein, and we will tackle the ligand in the next step.

1. If you haven't already done so, enter your GROMACS environment.
   `mamba activate gromacs_2023.1`
2. Let's start by parameterizing the protein. We will use the pdb2gmx module within gromacs here, which you can learn about with the command `gmx pdb2gmx -h` . In our specific case, we can use the command:
   `gmx pdb2gmx -f 4i4f_protein.pdb -o 4i4f_protein.gro`
3. You should enter an interface to choose what kind of forcefield you want to parameterize with. For our purposes, let's go with ***Amber99SB*** (not ILDN). You will also be asked what parameters we want our solvent molecules to be. Let's choose ***TIP3P***.

2.1.2 Parameterizing the Ligand

While all proteins are made up of 20 amino acids (21 if you include selenocysteine), Ligand molecules tend to have unique, individual connectivity and atom-types, which requires specialized tools for parameterization. We will be using the ACPYPE tool that we also installed into our GROMACS to parameterize our ligand.

1. If you haven't already done so, enter your GROMACS environment, as it also contains ACPYPE.
   `mamba activate gromacs_2023.1`
2. We first need to add hydrogens to our ligand, which we can do with OpenBabel (which was installed with ACPYPE). The -h flag will add hydrogens to our system, our inputs and outputs are both going to be pdb files.
   `obabel -ipdb 4i4f_ligand.pdb -h -opdb -O 4i4f_ligand-prepared.pdb`
3. Now, we will parameterize the ligand with hydrogens using ACPYPE. We will name our ligand as LIG (this name is important, because we'll reference it alot in the future), and we will parameterize using the GAFF2 method. You can learn more about using acpype with the command `acpype -h`
   `acpype -i 4i4f_ligand-prepared.pdb -b LIG -a gaff2`
   Expect this to take a little bit of time (sometimes up to 10 minutes) as a semi-empirical DFT method is used to calculate the charges of the atom. When the process is finished, you should see a new directory called `LIG.acpype` to show up in your working directory.

2.1.3 Combining the Ligand and Protein to Create a Protein+Ligand System

We needed to separate the protein and ligand to parameterize them separately, but now we need to recombine them for the rest of the setup pipeline. You can manually cut and paste the files together, but instead, we will use a provided script that does that for us.

1. If you haven't already done so, enter your GROMACS environment, so that we can be sure we have python in our environment.
   `mamba activate gromacs_2023.1`
2. Copy the script `join_protein_and_ligand.py` from the shared files into this directory.
3. The script needs the ligand basename (the -b argument from acpype) the protein file, and the topology file (topol.top)
   `python join_protein_and_ligand.py LIG 4i4f_protein.gro topol.top`

You should see a new file called `complex.gro`, and we've also changed the topology file to look for a new ligand and include extra positional restraints. Note that this script will modify the topology file, but will save the old topology file as `bak_topol.top` if you need to use an unmodified version again later

2.2 Creating Periodic Boundary Conditions and Adding Water and Ions

In real life, proteins are surrounded by water, and to properly simulate them, we need to surround our protein with water as well. As our bodies contain non-trivial concentrations of different charged ions, we will need to add those to the system as well. Furthermore, because we are interested in how the protein and ligand interact with each other, we want to keep the system as small as possible, but we don't want artifacts at the edges of our simulated system. The solution is to use what is called periodic boundary conditions, where the simulation repeats itself, so that molecules that leave one boundary come in from the opposite side. We still need a buffer between the edges of the periodic boundary box and the protein, as we don't want the protein to end up interacting with itself!

Fortunately, GROMACS contains a number of in-built commands that help us with this process. For each of the modules that I describe in this instruction set, you can learn more about them using the command `gmx <your-module-of-interest> -h`.

1. If you haven't already done so, enter your GROMACS environment
   `mamba activate gromacs_2023.1`
2. First, let's define the periodic boundary box we want to use. GROMACS gives a number of different shapes that are possible, including 'cubic', 'dodecahedron', 'octahedron' and 'triclinic'. Triclinic is generally the most efficient in terms of reducing the number of waters in the system, but we will use a truncated octahedron here because it looks cool. We'll also include a 1.2 nanometer buffer between our protein and the solvent box.
   `gmx editconf -f complex.gro -o 01_boxgen.gro -bt octahedron -d 1.2`
   Note that we are using the file `complex.gro` that we generated in the previous step, and we generate `01_boxgen.gro` as our output.
3. With the box defined, we can fill the rest of the box with waters. There's a stochastic element to the water-filling step, so if your system has a different number of atoms than one of your group-mate's, this is probably why.
   `gmx solvate -cp 01_boxgen.gro -cs spc216.gro -p topol.top -o 02_solvated.gro`
   PLEASE NOTE this command will modify your `topol.top` file. If you need to re-do this step (or previous steps), you can't use the same `topol.top` file. GROMACS will automatically back up your original `topol.top` to something like `#topol.top.1#`, so copy one of those topology files back to `topol.top` try again. The most recently backed up file will have the highest number. If you can't recover the topol.top file, generate a new one by repeating sections **2.1.1** and **2.1.3**.
4. Next we need to compile our inputs from a .mdp file to generate an input .tpr file using the GROMACS pre-processor (grompp). I'll go into more detail about what's going on

here in section **4.0**, but for now, copy the file `add_ions.mdp` from the shared files so that we can use it for this step.

```
gmx grompp -f add_ions.mdp -c 02_solvated.gro -p topol.top -o 03_ions.tpr
```

The generated .tpr file will not be human-readable.

5. With our compiled input in hand, we can then add in our ions. Some of our solvent will be replaced with positive and negatively charged ions, and if our protein has any charge imbalances, we'll add ions to balance them out. We will add sodium (NA) and chloride (CL) ions to our system, at a concentration of 0.1 M.

   ```
   gmx genion -s 03_ions.tpr -o 03_solv-ion.gro -p topol.top \
   -pname NA -nname CL -conc 0.10 -neutral
   ```

   GROMACS will ask us what molecules we want to replace with solvent. If we choose the first option (System), that will potentially replace parts of our protein with ions! We want to replace the water with ions, but our topology doesn't have things labeled water (weird, I know), so using that option doesn't work (at least in my hands). Instead let's choose ***SOL*** for solvent instead.

   PLEASE NOTE this command will also modify your `topol.top` file. If you need to redo it, use a backed up file, or re-create it by redoing sections **2.1.1**, **2.1.3**, and step 3 in this list

6. Finally, because we want to be able to select both of the protein and the ligand to this system, we should create an index file that includes the selection of both our protein and our ligand.

   ```
   gmx make_ndx -f 03_solv-ion.gro -o index.ndx
   ```

   This will ask for input to create a new selection, but annoyingly, there's no LIG option (it's named by its residue name MOL instead). Instead, we should join ***Protein*** with ***Other*** and then quit the process with ***q***. In short, the two options you should put into the console here are:

   ```
   "Protein" | "Other"
   q
   ```

   The output of this process will be a file called `index.ndx`. The last selection of that index file should be the new one you created, `Protein_Other`.

3.0 Creating a Ligand-Only System using GROMACS

Molecular dynamics is also used on ligands in solution as well. To evaluate the binding affinity of a drug, you need to be able to account for the differences in ligand motion in solution, as opposed to ligand motions bound to a protein. We will run a molecular dynamics simulation of the ligand in crystal structure 4i4f on its own, in an octahedral box of water.

1. If you haven't already done so, enter your GROMACS environment

   ```
   mamba activate gromacs_2023.1
   ```

2. Let's create a new directory called `lig-only`. We'll copy the ligand parameter directory that acpype made into there. Go into `lig-only`.

   ```
   mkdir lig-only
   cp -r LIG.acpype lig-only
   ```

3. Copy `generate_ligand_topology.py` into your new directory, and run the script, using the basename that we used with acpype.
   `python generate_ligand-topology.py LIG`
   This should generate a `topol.top` file, a `LIG_GMX.gro` file, a `LIG_GMX.itp` file and a `posre_LIG.itp` file.cd ../
4. Let's put the ligand by itself into a new box. The directory `LIG.acpype` has formatted .gro file that we can use
   `gmx editconf -f LIG_GMX.gro -o 01_boxgen.gro -bt octahedron -d 1.2`
5. You can repeat steps 3-5 of section **2.2** to fully solvate the ligand and add ions.
6. We will also create an index file for this system, too, but we don't need to create any special selections, so we can just exit with **q** after we get prompted
   `gmx make_ndx -f 03_solv-ion.gro -o index.ndx`

4.0 Minimizing and Equilibrating our Systems

When we set up our systems, GROMACS magically poofed molecules into existence, with some regard to how close they would be with other molecules, but not in a necessarily natural way. If you visualize the `03_solv-ion.gro` structures with VMD, you can see that the waters are aligned up in an orderly fashion, which is not what actually happens inside of a liquid. We don't want the energy of rearrangement to affect our final production results, so when setting up a molecular dynamics simulation, we can take steps to **minimize** the energy of our system with the protein and ligand constrained, and then we can **equilibrate** the system to let it move into more natural positions while we remove the constraints.

To run a simulation in GROMACS, you first compile the instructions that you want to give to the program into a binary .tpr file that is generated using the GROMACS pre-processor. Generating the .tpr file will involve a human-readable .mdp file that includes instructions for what you want GROMACS to do.

We will minimize and equilibrate **both** the protein+ligand and ligand-only systems. Rather than run each step individually, we will pipeline them into a script that you can run once, which is `run_min-equil.sh` in the shared files. But there's no reason to treat this script as a black box. Let's look at it:

```
#!/bin/bash
gmx grompp -f 04_minimize.mdp -c 03_solv-ion.gro -p topol.top -maxwarn 4 -o 04_minimization.tpr
gmx mdrun -v -nt 8 -deffnm 04_minimization

gmx grompp -f 05_nvt_equil.mdp -c 04_minimization.gro -r 04_minimization.gro -p topol.top \
-maxwarn 4 -n index.ndx -o 05_nvt_equil.tpr
```

```
gmx mdrun -v -nt 8 -deffnm 05_nvt_equil

gmx grompp -f 06_npt_equil.mdp -c 05_nvt_equil.gro -t 05_nvt_equil.cpt
-r 05_nvt_equil.gro  \
-p topol.top -maxwarn 4 -n index.ndx -o 06_npt_equil.tpr
gmx mdrun -v -nt 8 -deffnm 06_npt_equil

gmx grompp -f 07_md_run.mdp -c 06_npt_equil.gro -t 06_npt_equil.cpt -p
topol.top \
-n index.ndx -maxwarn 4 -o 07_md_run.tpr
```

You can see the that the script is comprised of alternating grompp and mdrun commands. You should already have some familiarity with grompp, that's the command that compiles a .tpr input file. The mdrun command tell GROMACS to actually perform the simulation. We don't specify the .tpr command itself, instead we use the -deffnm option to tell GROMACS to look for a tpr file with a certain prefix, and to output files with that same prefix. The -nt option tells GROMACS to use 8 threads for the minimization and equilibration steps.

Perform the following steps in the `fak_example` directories:

1. If you haven't already done so, enter your GROMACS environment,
   `mamba activate gromacs_2023.1`
2. Copy `04_minimize.mdp`, `05_nvt_equil.mdp`, `06_npt_equil.mdp`, `07_md_run.mdp`, and `run_min-equil.sh` into the directory
3. Give the `run_min-equil.sh` executable permissions
   `chmod +x run_min-equil.sh`
4. Make sure that you have at least 8 threads available, and then run the script
   `./run_min-equil.sh`
   You should get a `07_md_run.tpr` file in each directory that you can use for the production MD simulations.

In the `lig-only` directory, we can perform similar steps, but we need to use slightly different mdp files:

5. If you haven't already done so, enter your GROMACS environment,
   `mamba activate gromacs_2023.1`
6. Copy `04_minimize.mdp`, `L05_nvt_equil.mdp`, `L06_npt_equil.mdp`, `L07_md_run.mdp`, and `run_min-equil_lig-only.sh` into the directory
7. Give the `run_min-equil_lig-only.sh` executable permissions
   `chmod +x run_min-equil_lig-only.sh`
8. Make sure that you have at least 8 threads available, and then run the script
   `./run_min-equil_lig-only.sh`

You should get a `07_md_run.tpr` file in each directory that you can use for the production MD simulations.

5.0 Production Runs for the Protein+Ligand and Ligand-Only Simulations

We want to see how the speed of the molecular dynamics runs changes as a function of the number of threads, so we will create a bunch of directories where we copy the tpr file into, and then run the script from each one.

Perform the following steps in both the `lig-only` and `fak_example` directories:

1. If you haven't already done so, enter your GROMACS environment,
   `mamba activate gromacs_2023.1`
2. Create directories for 1,2,4,8,12,16,24,28,32,40 and 48 thread runs
   `mkdir mdrun_{01,02,04,08,12,16,24,28,32,40,48}_threads`
3. In each directory, copy `07_md_run.tpr` into it, and then run GROMACS with the following command:
   `gmx mdrun -v -nt <number-of-threads> -deffnm 07_md_run`
   a. I recommend running each simulation one at a time. GROMACS by default does not play nice with other processes running on the same system.
   b. Some of the high-thread ligand runs will fail. If they do, don't include them in your final plot. The failed runs will still produce a logfile.
   c. I'll also note that people on other systems have needed to change OMP_NUM_THREADS to a higher value to utilize more threads, but I haven't experienced this behavior in my tests personally.
   d. On a single Haswell node, the 1 thread run of the protein+ligand system took roughly 80 minutes. Plan accordingly.
4. You'll need to upload these log files to Google Classroom. Within the `fak_example` directory, type the commands:
   `tar -cf mdrun_logs.tar mdrun_??_threads/07_md_run.log lig-only/mdrun_??_threads/07_md_run.log`
   `gzip mdrun_logs.tar`
   These two commands should create a file called `mdrun_logs.tar.gz` that you can upload to Google Classroom
5. dd

Each run will create a `07_md_run.log` file, which when completed, will give an estimate for the rate at which the simulation progresses in ns/day (at the bottom of the log file). When each run is done, tabulate the ns/day value with the number of threads used for the simulation, as well as the ratio of ns/day divided by core time (Core t). Plot both of these values against the number of threads for both the protein+ligand system and the ligand-only system using your favorite plotting software.

6.0 Visualizing A Trajectory

Visualization is an important aspect of running molecular dynamics, however, creating an output that is easy for a human to understand is somewhat difficult. Let's use VMD as our visualization software. After registering, you can download the software for free at https://www.ks.uiuc.edu/Development/Download/download.cgi?PackageName=VMD . Install the software on a device with a screen. Once you've done that, let's process one of the trajectories (it doesn't matter which) from the protein+ligand system.

1. If you haven't already done so, enter your GROMACS environment,
   `mamba activate gromacs_2023.1`
2. By default, XTC files can't be read by VMD (although there is a plugin for VMD version 1.9.2 that can directly read GROMACS output). Additionally, we want to correct for the effects of our periodic boundary condition. We can convert our trajectory using the gromacs `trjconv` command.
   ```
   gmx trjconv -f 07_md_run.xtc -s 07_md_run.tpr -n ../index.ndx -pbc mol \
   -ur compact -o 07_md_run.pdb
   ```
   When prompted, choose to output the entire system.
3. The previous command will correct for any PBC adjustments, but the protein+ligand system will still tumble in the simulation, making it difficult for us to ascertain what's going on. We need to further refine our
   ```
   gmx trjconv -f 07_md_run.pdb -s 07_md_run.tpr -n ../index.ndx -fit rot+trans \
   -o 07_md_run_FITTED.pdb
   ```
   When prompted, choose to fit the Protein_Other selection to least squares fit (so that the protein and ligand won't tumble), and then output the whole system.

Transfer the file `07_md_run_FITTED.pdb` to your machine with VMD installed, and open it in VMD. You should see the truncated octahedron crystal structure, with a protein in the middle of it. Play around with the trajectory in VMD to get a sense of what's going on. The graphical representations pane (Graphics > Representations) will be helpful, as you can set it to view only the protein (replace the "Selected Atoms" section with "protein"), the ligand (either "resname MOL" or "resname 1BR"). If you want to play around with VMD even more, I'll refer you to the user guide, particularly section 3.5.

7.0 End of Homework Questions

Create a separate document that answers each of these questions. Only one set of answers needs to be turned in per group.

1. Include your plots of benchmarking the performance and efficiency of the simulations of the protein+ligand and ligand-only systems at varying thread counts.
   a. What are the general trends in performance (ns/day)?
   b. What are the trends in efficiency ((ns/day)/(Core t))?
2. Look at the atom counts of the final set up protein+ligand and ligand-only systems.
   a. How many atoms does each system have?

    b. How many solvent molecules, chloride ions, and sodium ions does each system have?
3. In the benchmarking of the protein+ligand system and ligand-only system, why is there such a large disparity in ns/day between the two systems when the same number of threads are used?
4. Take a look at the log for the 48 thread protein+ligand system (in `fak_example/mdrun_48_threads/`).
    a. How many ranks did GROMACS assign to PP?
    b. How many ranks did GROMACS assign to PME?
    c. How many threads were assigned per rank?
    d. If you wanted to get better performance, how would you change the ratio of ranks?
5. Let's look at those failed ligand-only runs.
    a. At higher thread counts, why did the ligand-only system become sensitive to the number of threads used? Specifically, why were there cases where a lower thread count failed, but a higher thread count would run normally?
    b. Why didn't you see this behavior in the protein+ligand system?
    c. Are there specific differences between the log files of the protein+ligand and ligand-only runs at the same thread count that you can point to that explain the change in behavior?
    d. In the cases where the ligand-only system failed to run, what changes to your mdrun command could you make to fully utilize your requested thread-count?
6. Include a screenshot (including the Display, graphical representations, and VMD main windows, similar to slide 49 of the powerpoint) of your VMD visualization of the protein+ligand system from the first frame and the last frame. Keep the axis in the lower left corner (this is done by default, just don't get rid of it).
    a. How many frames does your trajectory have?

Upload your answer document as well as the .tar.gz file with your logs into the GROMACS section of Google Classroom.