# ***Running this Repository***

1. To download the repository on your local machine, use the git clone command.
    a. HTTP: git clone https://github.com/estar29/NBodyParallel.git
    b. SSH: git clone git@github.com:estar29/NBodyParallel.git
2. Navigate to where on your local machine the directory was copied to.
3. Once inside the directory, there are multiple ways to run the program and gather results.
4. For predefined scenarios:
    a. For sequential implementation, change the directory to the sequential folder, then type any of the following commands:
        i. Solar system, sequential log files: make solar_seq.out
        ii. Solar system, sequential visualization: make solar_seq.pdf
        iii. Hundred particles, sequential log files: make hundred_seq.out
        iv. Hundred particles, sequential visualization: make hundred_seq.pdf
        v. Thousand particles, sequential log files: make thousand_seq.out
        vi. Thousand particles, sequential visualization: make thousand_seq.pdf
    b. For parallel implementation, change the directory to simply the NBodyParallel and run any of the following commands:
        i. Solar system, parallel log files: make solar_par.out
        ii. Solar system, parallel visualization: make solar_par.pdf
        iii. Hundred particles, parallel log files: make hundred_par.out
        iv. Hundred particles, parallel visualization: make hundred_par.pdf
        v. Thousand particles, parallel log files: make thousand_par.out
        vi. Thousand particles, parallel visualization: make thousand_par.pdf
5. For custom scenarios, run the following make commands:
    a. Sequential: change the directory to the sequential folder if not done so, then…
        i. make nbody
        ii. ./nbody (number_of_particles or “solar” for the solar system) (time_step_size) (number_of_time_steps) (how_often_to_print_state)
    b. Parallel: change the directory to simply the NBodyParallel directory, then…
        i. make nbody_par
        ii. ./nbody_par (number_of_particles or “solar” for the solar system) (time_step_size) (number_of_time_steps) (how_often_to_print_state)

## ***Findings***

After doing some testing, the sequential implementation erroneously appears to be around 9-10 times faster than the parallel implementation. This is not supposed to happen, indicating incorrect parallelization of the program. The most likely culprit is the cost of overhead from the numerous pragma calls to OpenMP greatly outweighing any potential computing time that could have been saved. A potential solution to make the parallel program more efficient is to decrease the number of pragmas used so that less overhead is produced.