Assignments P1.7 - Part 3

N-Body LJMD Refactoring, Optimization and Parallelization Group Project

You will work on this assignment in small groups with 3 members. The group assignment will be communicated to you. The evaluation of the group project will be based in part (1/3rd) on the performance of the group in total and on the individual contributions as documented by the git commits to the project (2/3rd). Please add a list of the group members and how their names will show up in the commit messages to the README file, so that your commits will be properly attributed.

Tasks for the entire group (to be completed first)

Set up a **shared** git repository on GitHub using the source code from session3/01-ljmd and then use the GitHub branch settings to **protect** the “master” so that no commits may be pushed into this branch but rather all changes to the “master” branch have to be done via pull requests, which have to go through review and merging requires approval of other group members (the PR submitter is not able to approve). Then each member **clones** this repository. You will work in your local repository on temporary feature branches, push them to GitHub and – if ready – submit a pull request to merge them into the master branch. To integrate changes that have been added to master, check out the master branch locally, pull all pending changes (this will be a fast-forward) and then switch to your feature branch and merge or rebase. Please note, that rebasing may only be done **before** a branch is pushed to a remote repo, as otherwise a forced push is needed since rebasing rewrites the commit history, which in turn can lead to all kinds of problems and inconsistencies (they can be fixed, but it is better to avoid them from the start).

- break down the single file ljmd.c into multiple files (force compute, verlet time integration (split into two functions in one file), input, output, utilities, cleanup, main function, and header for data structures and prototypes); update the CMakeLists.txt file accordingly so that you build one executable from main.cpp and a library, mdlib.
- set up some simple unit tests with the googletest library (write C++ code that allocates/fills data structure(s), calls the respective functions and uses assertions to check if the result is as expected) and integrate those tests into the CMake procedure so the tests can be run with ctest. For example:
  a) compute forces for a few 2-3 particle systems with atoms inside/outside the cutoff (directly and with images via PBC)
  b) compute part of the time integration for given positions, forces and velocities (no call to force())
  c) compute kinetic energy for given velocities and mass
  d) create a minimal input file (containing a few atoms) and matching restart on the fly and verify that that data is read correctly.

Tasks for **individual** group members (to be completed by December 3rd)

a) Optimize the force computation: refactor the code for better optimization and to avoid costly operations or redundant work. Adapt data structures as needed. Document improvements with profiling/benchmark data.
b) Add MPI parallelization. Document the parallel efficiency of changes.
c) Add OpenMP parallelization. Document the parallel efficiency of changes.
It is crucial that the individual tasks are implemented in small self-contained steps, so that they can be merged into the master branch frequently. Do not wait until it is too late.

Bonus tasks for extra credit (for individuals)

1) Implement an additional force function using Morse potential instead of Lennard-Jones
2) Implement cell list for better scaling with system size
3) Implement a python wrapper so the mdlib library can be called from python and thus the main function could be written in python.