Monte Carlo Simulation of a Lennard Jones Fluid

CHEM 280: **Foundations of Programming and Software Engineering for Molecular Science**

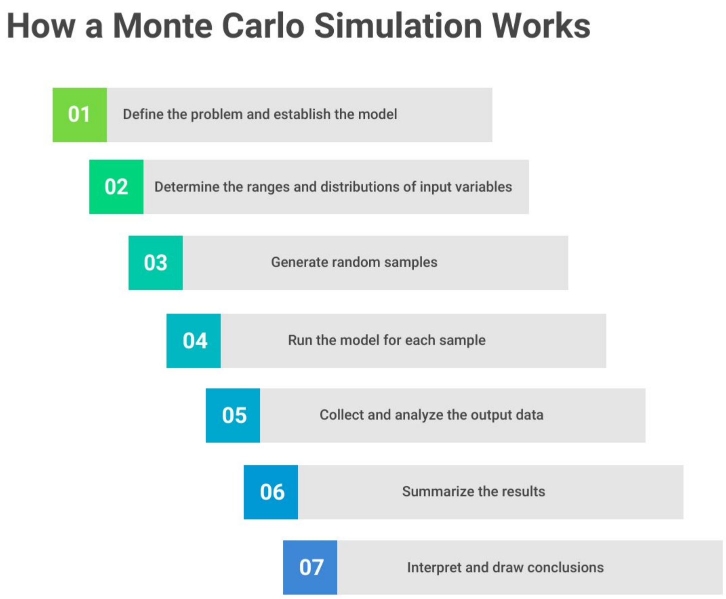
Team Sodium: Luis Hernandez, Thomas Janas, Casey Tomlin

Molecular Simulations

Casey Tomlin

Monte Carlo Simulations

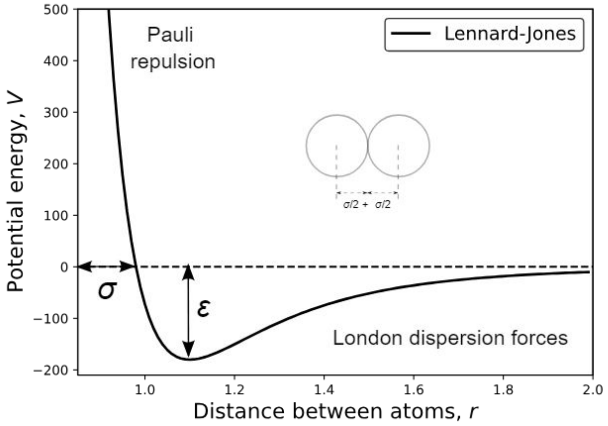
- Mathematical technique used to estimate the possible outcomes of an uncertain event
- Created by John von Neumann and Stanislaw Ulam during WWII
- Probe possible solutions to a problem defined with some element of randomness
    - Models a normal distribution for each variable
    - Obtains results and runs again
        - Each run uses a new set of random variables
- As more simulations are added, accuracy tends to increase



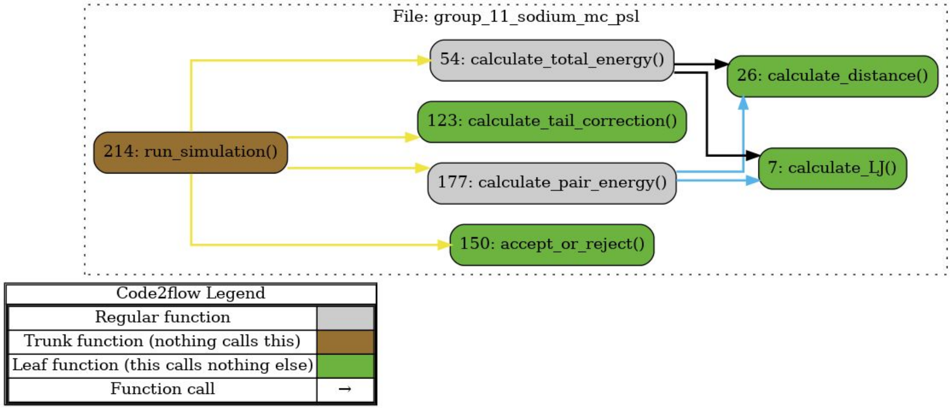
Lennard Jones Equation (12-6)

$$V(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

- Arguably the most widely-used pair potential in molecular simulations
- Models the pairwise interaction energy of two noble gas particles
- Reduced Units:
  - ε = 1 (depth of energy well, kJ/mol) and σ = 1 (van der Waals radius, Å)
  - r = distance between the two particles
  - Lennard Jones fluid possess near universal behavior with reduced units
  - Limits the range of influence of particles
- r^{-12} term approximates the strong Pauli repulsion
- r^{-6} term approximates the weaker attractive forces from locally induced dipoles



Implementation to Code



- Utilized data from NIST (National Institutes of Standards and Technology)
  - 3D coordinates of 800 particles
- Coordinates input to several functions:
  - Distance = returns distance between coordinates
  - Total Energy = returns the Lennard Jones energy of a system of particles
  - Pairwise Energy = returns the interaction energy of a particle with its environment
- Implementing a cutoff
  - Calculating tail correction accounts for this loss
- Monte Carlo Simulation of Lennard Jones Energy
  - Pick random particle
    - Interaction energy with system (current)
  - Displace particle randomly
    - Interaction energy with system (proposed)
  - Accept or reject proposed changes to particle
  - Print results

Programming

Luis Hernandez

Python vs C++

- Advantages/Disadvantages
- Performance
- Ease of Use
- Suitability for Projects

Interpreted VS. Compiled

Interpreted

- Python, JavaScript, Ruby, Perl, and PHP
- Code is translated to machine code at run-time
- Tend to be slower due to translation at run-time
- Debugging occurs at run-time so can be easier
- Easier for beginners
- More portable since it does not provide machine code

Compiled

- C, C++, Erlang, Haskell, Rust, and Go
- Code is compiled to machine code before execution
- Tend to be faster and more efficient since the machine code has already been compiled
- Code can fail compilation if errors are present
- Better for performance and memory management

Python VS. C++

Python

- Easier to code
- Typically fewer lines of code
- Dynamically typed
- Rapid development and deployment
- Automatic memory management
- Uses
  - Web development
  - Data Analysis
  - Machine learning

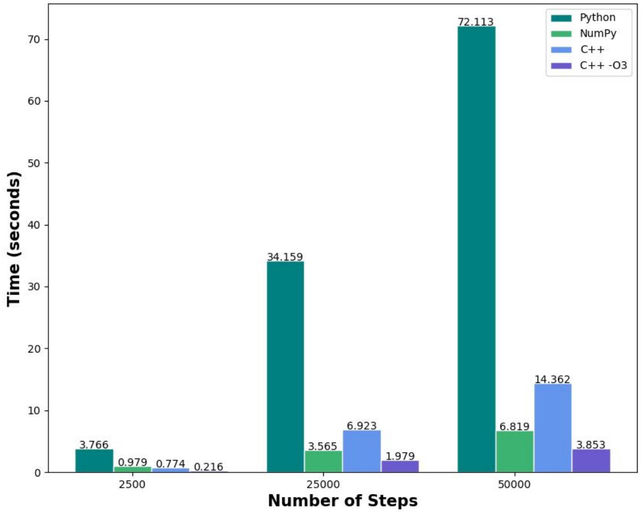
C++

- Typically more lines of code
- Variable type must be stated
- Must be compiled before execution
- Manual memory management
  - Increased control, but error prone
- Uses
  - Game Development
  - High Performance Computing
  - Embedded Systems

Best VS. Bestest Language

Implementations Evaluated

- Python Standard Library
- NumPy
- C++
  - Optimization flag -O0
  - Optimization flag -O3



Best VS. Bestest Language - Programming Ease

```python
import math
import random
import numpy as np

> def calculate_distance(coord1, coord2, box_length=None): ···

> def calculate_LJ(r_ij): ···

> def calculate_total_energy(coordinates, box_length, cutoff): ···

> def read_xyz(filepath): ···

> def calculate_tail_correction(n_particles, box_length, cutoff): ···

> def accept_or_reject(delta_e, beta): ···

> def calculate_pair_energy(coordinates, i_particle, box_length, cutoff): ···

# create a function called run_simulation that takes in simulation parameters and
> def run_simulation(coordinates, box_length, cutoff, reduced_temperature, num_steps
```

```cpp
#include <iostream>
#include <cmath>
#include <random> // From random_number_generator
#include <chrono> // From random_number_generator for generating random seeds
#include <fstream> // From read_xyz, for reading and writing files
#include <array> // From read_xyz, for std::array
#include <vector> // From read_xyz, for std::vector
#include <utility> // From read_xyz, for std::pair

// Make some types more convenient
typedef std::array<double, 3> AtomCoord; // From read_xyz
typedef std::vector<AtomCoord> Coordinates; // From read_xyz


std::default_random_engine re; // A Global! Probably shouldn't be used in real code - for Random Number Generator
std::pair<Coordinates, double> read_xyz(std::string file_path); // From read_xyz


double calculate_LJ(double r_ij);
bool accept_or_reject(double delta_e, double beta);
double random_double(double lower_bound, double upper_bound);
int random_integer(int lower_bound, int upper_bound);
double calculate_tail_correction(int n_particles, double box_length, double cutoff);
double calculate_pair_energy(const Coordinates & coordinates, int i_particle, double box_length, double cutoff);
double calculate_distance(const AtomCoord & coord1, const AtomCoord & coord2, double box_length);
double calculate_total_energy(const Coordinates & coordinates, double box_length, double cutoff);
std::pair<std::vector<double>, std::vector<double> > run_simulation(Coordinates coordinates, double box_length, dou
```

Code to ease code!!

Best VS. Bestest Language - Rewriting

- Coding .cpp from .py (PSL) was tedious, but straightforward
    - Significant boost in performance by using optimization flag -O3
- Rewriting PSL version to NumPy version was not initially straightforward
    - Once we had a better understanding of NumPy, it was much easier to program to gain significant performance improvement and complete the rewrites of the functions
    - Using NumPy:
        - Creating Arrays np.array()
        - Manipulating Arrays: np.delete(), np.concatenate(), slicing
        - Rapid Calculations
- In this case, the ease of writing and large gain in improvement using NumPy provided the most satisfaction

Best VS. Bestest Language - Conclusion

- The Bestest Language is determined by the use case
- Python programming
  - Rapid development and deployment
  - Great for beginners
  - Ease of use is more important than performance
- C++ programming
  - Use for complex deployments
  - Use when performance is most important

Software Engineering

Thomas Janas

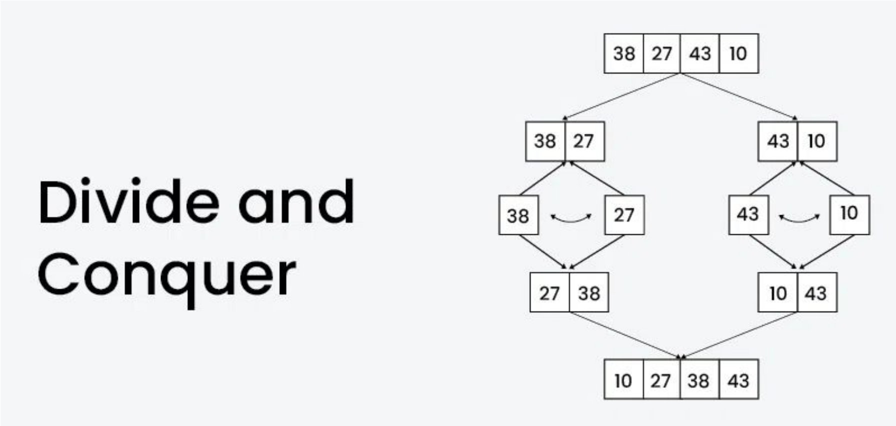
Software Testing

- Debugging
- Assert functions
- Printing variables
- Unit tests
    - “Sanity Checks”
    - Jupyter Notebook
        - Significant ease of producing and testing python code in real-time

Performance Improvements

- Switching from Python to NumPy or C++
- Instead of calculating the total energy for particles after a movement, we check for the differences associated with the moved particle
- Altering Python arrays in NumPy arrays for the calculate_LJ and calculate_distance functions
- Utilizing the optimization flag O-3, which optimizes the compiling to improve execution performance

Team Workflow



Individual level

- Drafting code
- Testing
- Debugging

Group level

- Code review
- Task balancing
- Group brainstorming
- Git/Github code management

Benefits

- Allows for quick solutions to be developed
- Ensures code is descriptive
- Code is built to be understandable
    - Allows coding tasks to be transferable

Git/GitHub - Useful tools for collaboration

Likes

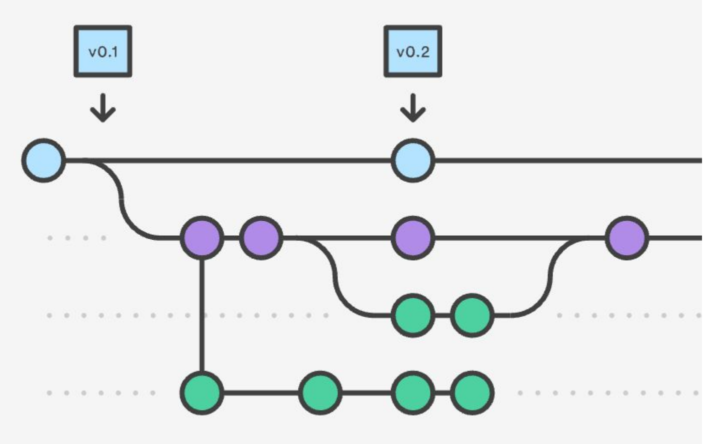
- Great online visualization for organizing files
- Provides all of the benefits of version control
- Allows teammates to pull branches and test code on their machines

Significantly helped with tracking changes and managing project

Dislikes

- Unique set of commands that are not entirely straightforward as a beginner
- Different operating systems and coding environments create frequent problems for beginners

Version Control



Definition: Method for tracking changes in code.

- Retains older code structures that was previously working
  - Allows for rollback of a current version to a previous version of code
- Allows changes to code without altering a current version that is being used by others
- Useful for managing a software project with a team
  - Delegated tasks can be tracked by users for progress
  - Any code changes which crashes a program can be directly sourced with proper management

Bibliography

- https://www.ibm.com/topics/monte-carlo-simulation
- https://pubs.acs.org/doi/10.1021/acs.jctc.4c00135
- https://education.ni.com/center-of-excellence/resources/448/intro-to-source-code-control-and-distributed-version-control-systems
- https://github.com/logos
- https://www.geeksforgeeks.org/divide-and-conquer/
- https://chem.libretexts.org
- https://www.geeksforgeeks.org/difference-between-python-and-c/
- https://www.ionos.com/digitalguide/websites/web-development/python-vs-c/#:~:text=Use%3A%20C%2B%2B%20is%20commonly%20employed,machine%20learning%20and%20artificial%20intelligence.