Contents

List of Figures

Chapter 1

Introduction

1

Chapter 2

Methods

The goal of a molecular dynamics simulation is to determine the motion of each individual atom. For this we have to solve the equations of motion and compute the forces that affect each atom.

$$\vec{f_k} = -\frac{\partial}{\partial \vec{r_k}} E_{pot}(\{\vec{r_k}\}) \tag{2.1}$$

The forces can be derived from the potential Energy as shown in 2.1 [cf. 4]. The simulation was run in a discretized space, so all updates to the motion of atoms was only done after a time step.

2.1 Integration

From the equation 2.1 we can obtain the force at each individual atom with it's acceleration, velocity and position as shown in 2.1.

$$\vec{f_i} = \frac{\partial E_{pot}}{\partial \vec{r_i}} = m_i \vec{a_i} = m_i \vec{v_i} = m_i \vec{r_i} \tag{2.2}$$

2.2 Potentials

$$\vec{f_k} = -\frac{\partial E_{pot}}{\partial \vec{r_k}} = \sum_i \frac{\partial V}{\partial r_{ik}} \hat{r_{ik}} \tag{2.3}$$

2.2.1 Lenard-Jones-Potential

$$V(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{2.4}$$

[4]:
```
import sympy as sp
import warnings
warnings.filterwarnings('ignore')
sp.init_printing()
eps = sp.Symbol("e")
sig = sp.Symbol("s")
rad = sp.Symbol("r")
```

```
energyRad = 4 * eps * ((sig/rad)**12 - (sig/rad)**6)
energyRad.diff(rad)
```

[4]:
$$4e\left(\frac{6s^6}{r^7} - \frac{12s^{12}}{r^{13}}\right)$$

2.2.2 Neighborhood-Search Algorithm

2.2.3 Embedded-Atom Method Potentials

2.2.4 Berendsen-Thermostat

$$\lambda = \sqrt{1 + \left(\frac{T_0}{T} - 1\right)\frac{\Delta t}{\tau}} \tag{2.5}$$

$$T(t) = T_0 + (T_1 - T_0)e^{-t/\tau} \tag{2.6}$$

Chapter 3

Implementation

The simulation code was written in C++, most of it just as functions, although the positions, velocities, etc. of the individual atoms where saved in a container-class. While writing the functions, these were also tested with unit-tests. Plots generation and automation for running the project were written in python.

The C++-code was developed in CLion, an IDE which bundles many useful features together (CMake, GDB and Git). The python-code was written in jupyter-notebook, alternatively this could also have been done directly in CLion with a plugin. Additional libaries used where: googletest [3] for the unit-tests and eigen [1] for the arrays used for data storage in the container-class. Further software form the course was used for reading xyz-data and for the implementation of the Neighborhood-Search and Gupta-Potential algorithms [4]. To visualize the simulation the positions of the atoms at a given time where recorded and then put into OVITO Basic [2] to be able to look at them.

The code is parted into a headerfile (.h) and a codefile (.cpp). The functions where generally structured into a related header- and codefile. For example all functions regarding the Lenard-Jones-Potential can be found in an appropriately named filecombo. Furthermore a test file exists where the unit-tests can be found.

The implementation itself just followed the milestones given in the course and went up till milestone seven. In case any mistakes happened the main of each milestone was saved into an extra file and just commented into the code again in the true main.cpp file.

For further information it would be best to just look at the code itself as it should be well commented [5].

Chapter 4

Results

4.1 Results from the Lenard-Jones Potential with direct Summation

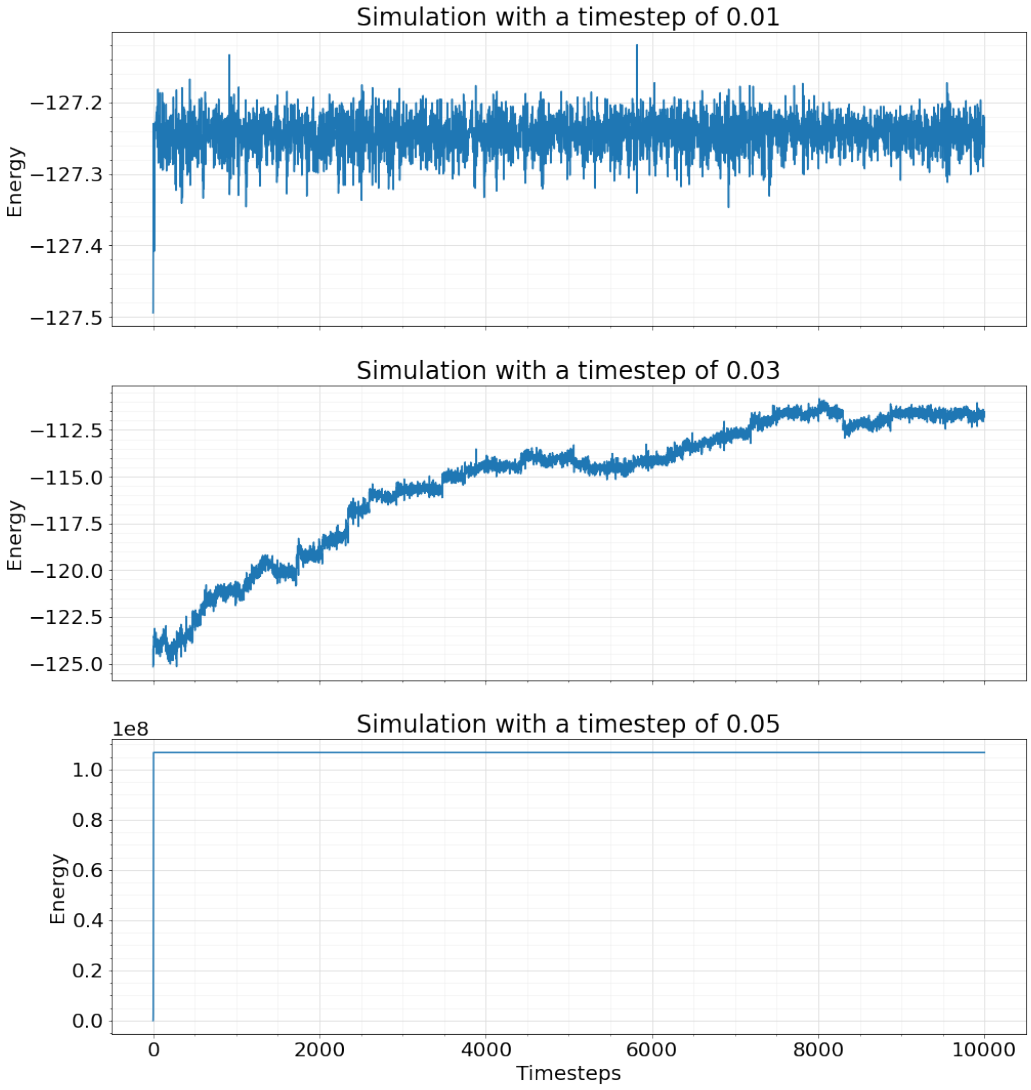

Figure 4.1: Simulation with different timesteps

The first task of the course asks to plot the total energy of the simulation for different time steps. The units were in a Lenard-Jones equivalent and not given here. As can be seen in the sequence 4.1, with a bigger and bigger timestep the energy in the simulation goes from stable (timestep 0.01) over a drifting behavior (timestep 0.03) to being unstable (timestep 0.05). A good timestep for this simulation would be 0.01.

To visualize the simulation OVITO [2] was used and this series of images was created.

Figure 4.2: Simulation Snapshot

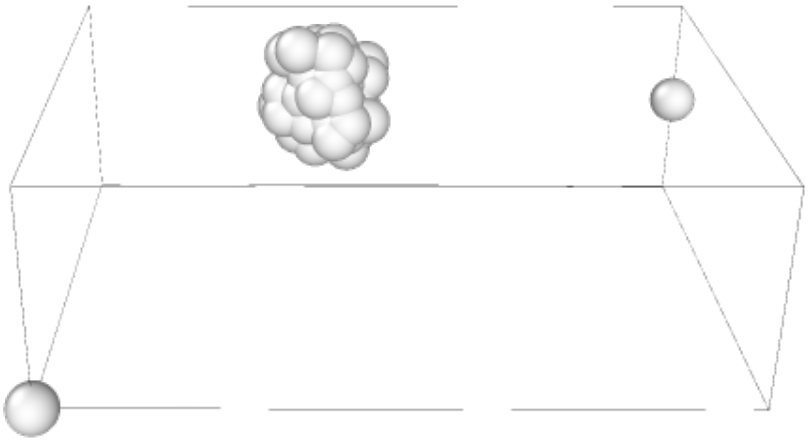

Figure 4.3: Simulation Snapshot

Figure 4.4: Simulation Snapshot

As it can be seen in the images 4.2, 4.3 and 4.4 the Atoms are initially ordered into a blob, which was given in the course. Later in the simulation some of the atoms escape the initial blob and fly outward separately.

4.2 Result from the Simulation with the Berendsen Thermostat

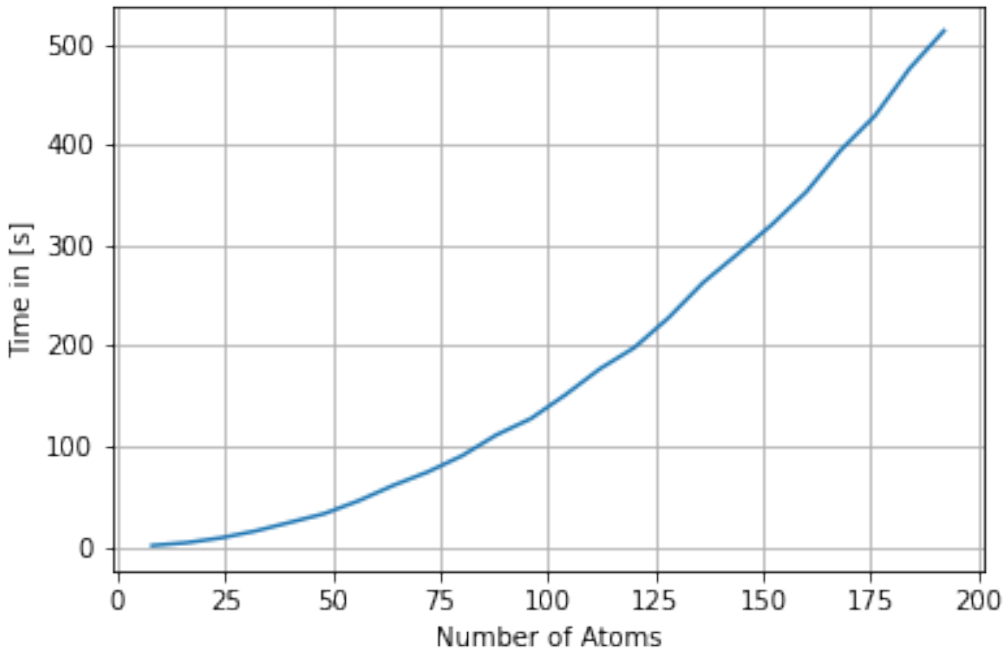

Figure 4.5: Simulationtime with the Berendsen Thermostat from 8 to 192 Atoms

4.3 Results from the Simulation with the Neighborhood-List

4.4 Results from the Simulation with the Gupta-Potential

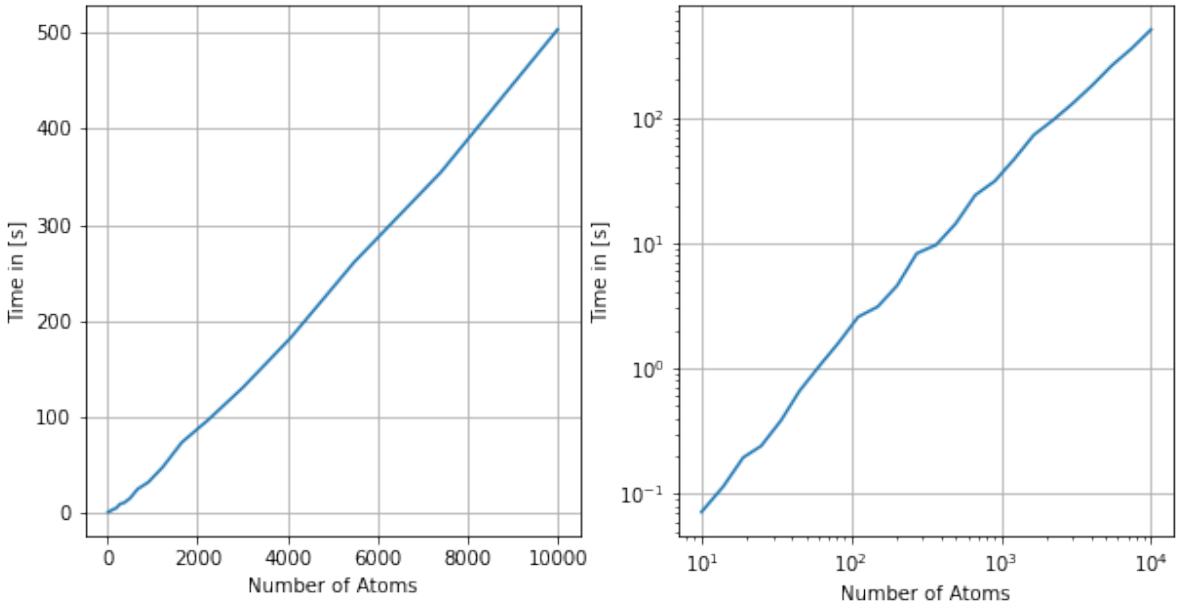

Figure 4.6: Simulationtime with the Neighborhood-list

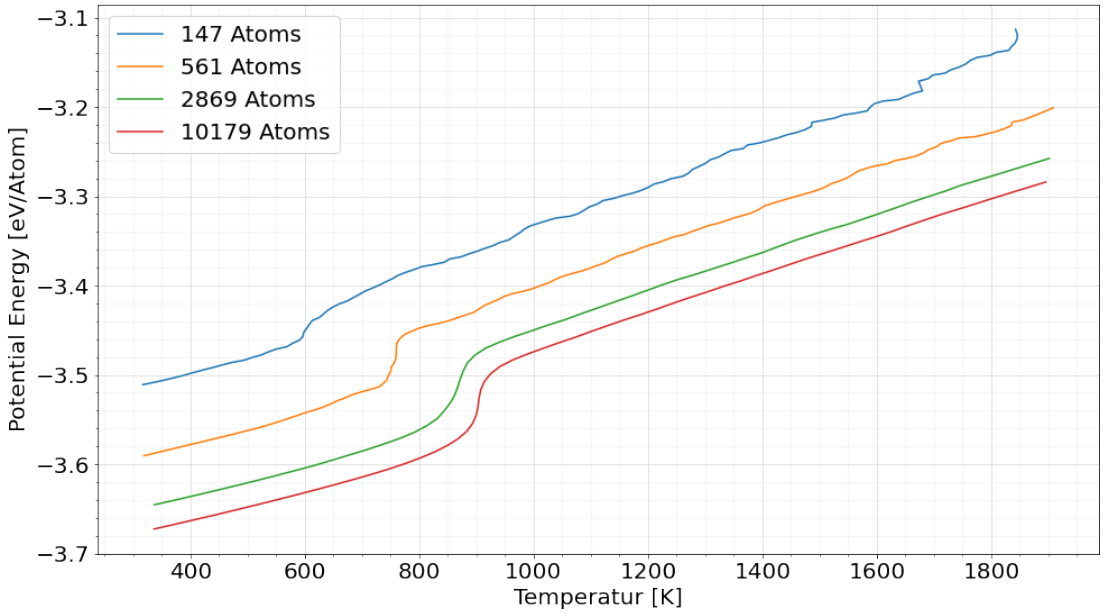

Figure 4.7: Gold Cluster Simulation

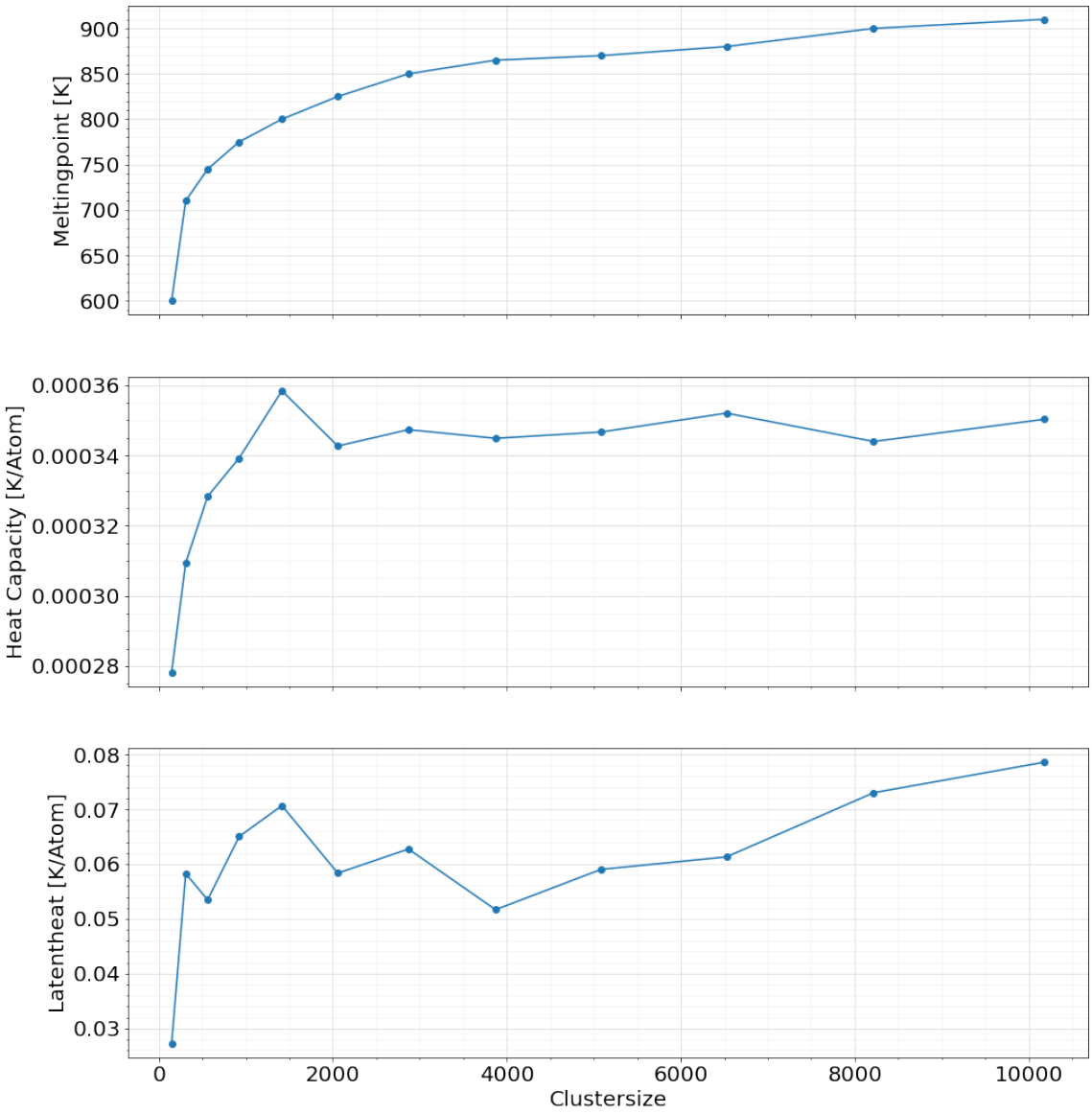

Figure 4.8: Melting Point, Heat Capacity and Latent Heat vs Clustersize

Chapter 5

Conclusion

Bibliography

[1] Gaël Guennebaud Benoît Jacob. *Eigen*. 2020. URL: https://eigen.tuxfamily.org/index.php?title=Main_Page (visited on 07/28/2021).

[2] OVITO GmbH. *Ovito*. 2021. URL: https://www.ovito.org (visited on 08/01/2021).

[3] Google Inc. *GoogleTest, Google's C++ test framework*. 2021. URL: https://github.com/google/googletest (visited on 07/28/2021).

[4] Wolfram Nöhring Lars Pastewka. *Molecular Dynamics Course*. 2021. URL: https://imtek-simulation.github.io/MolecularDynamics/ (visited on 07/28/2021).

[5] Christoph Moser. *Code Repository*. 2021. URL: https://github.com/cmoser8892/MoleDymCode (visited on 08/01/2021).