Contents

List of Figures

Chapter 1

Introduction

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.

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

The forces can be derived from the potential Energy as shown in 2.1 [cf. 7].

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 2.2.

$$\overrightarrow{f_i} = \frac{\partial E_{pot}}{\partial \overrightarrow{r_i}} = m_i \overrightarrow{a_i} = m_i \overrightarrow{\dot{v}_i} = m_i \overrightarrow{\ddot{r}_i} \tag{2.2}$$

It can be seen that the system can be described just by the velocities and positions of the atoms. In the implementation we hold these values in an container-class to ensure that they are consecutive in memory. This speeds up the simulation.

In the next step we need to somehow propagate the simulation forward in time as a static simulation would be rather boring. For updating the positions and velocities one of the most used algorithm is the Velocity-Verlet integration. This scheme is often split up into two steps, the prediction shown in the equations 2.3 and 2.4 and the correction shown in equation 2.5 [cf. 7]. In between the two steps the force can be updated.

$$\overrightarrow{v_i}(t + \Delta t/2) = \overrightarrow{v_i}(t) + \frac{\overrightarrow{f_i}(t)\Delta t}{2m_i} \tag{2.3}$$

$$\overrightarrow{r_i}(t + \Delta t) = \overrightarrow{r_i}(t) + \overrightarrow{v_i}(t + \Delta t/2)\Delta t \tag{2.4}$$

$$\overrightarrow{v_i}(t + \Delta t) = \overrightarrow{v_i}(t + \Delta t/2) + \frac{\overrightarrow{f_i}(t + \Delta t)\Delta t}{2m_i} \tag{2.5}$$

We could now already propagate atoms forward with a constant force, so next we have to actually model the forces affecting each atom.

2.2 Lenard-Jones-Potential

The Lenard-Jones potential is most likely one of the more famous pair potentials. The goal of them is to model the Coulomb force and the Pauli repulsion.

$$V(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{2.6}$$

As shown in 2.1 we can derive the forces from the potential Energy. Now we have to formulate the equation for each atom, this was done in 2.7. We can get the force at the atom, if we get sum up the derivative of the potential energy between atom k and all the other atoms i times the normed vector between them.

$$\vec{f_k} = -\frac{\partial E_{pot}}{\partial \vec{r_k}} = \sum_i \frac{\partial V}{\partial r_{ik}} \hat{r_{ik}} \tag{2.7}$$

To actually implement the equation we have to actually derive $\delta V / \delta r_{ik}$ analytically. This could have been done by hand, but was actually just done with python.

```python
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)
```

With the codesnippet from above the derivative of the potential was obtained, shown in the next equation 2.8.

$$\frac{\partial V}{\partial r} = 4\epsilon\left(\frac{6\sigma^6}{r^7} - \frac{12\sigma^{12}}{r^{13}}\right) \tag{2.8}$$

Now we can implement those last two equations. We can even optimize it by subtracting the force that affects atom k from the forces of atom i, as they have to be the same just with opposite direction and therefore sign.

It can be seen that this algorithm behaves in the Order O(N^2) with increasing numbers of atoms in the simulation. This can be reduced to a linear order O(N) if we only consider atoms up to a certain distance (cutoff-distance) around each atom. This is possible as the forces get very small with a big distance between the atoms. The next goal would be to generate a list witch marks neighbors that are inside this cutoff-distance. This can be done using domain decomposition [cf. 7]

2.3 Berendsen-Thermostat

Thermostats are used in molecular simulation to control the heat in the system and preventing it from melting or evaporating. A simple form of temperature control is the Berendsen-Thermostat, which resales the temperature.

It is typically implemented by resealing the velocities by a factor as shown in equation 2.9.

$$\lambda = \sqrt{1 + \left(\frac{T_0}{T} - 1\right)\frac{\Delta t}{\tau}} \tag{2.9}$$

This can be done as the temperature is just another facet to kinetic energy and is connected via the following equation [cf. 7].

$$\frac{3}{2}k_B T = \sum_i \frac{1}{2}mv_i^2 \tag{2.10}$$

2.4 Embedded-Atom Method Potentials

Embedded atom-potentials are a good model for metallic bonding in solid or liquid states for different materials. As an implementation of the potential, developed by Gupta [4] and Cleri & Rosato [1] was provided in the course [7] itself, the details are best found there.

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 [5] for the unit-tests and eigen [3] 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 [7]. 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. While using the embedded-atom method potential it was necessary to generate a variety of different sized clusters in the form of an icosahedron. For this an external Mackay Icosahedron Structure Generator [8] was used.

The code is parted into a headerfile (.h) and a codefile (.cpp). The functions were 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 [6].

Chapter 4

Results

4.1 Results from the Lenard-Jones Potential with direct Summation

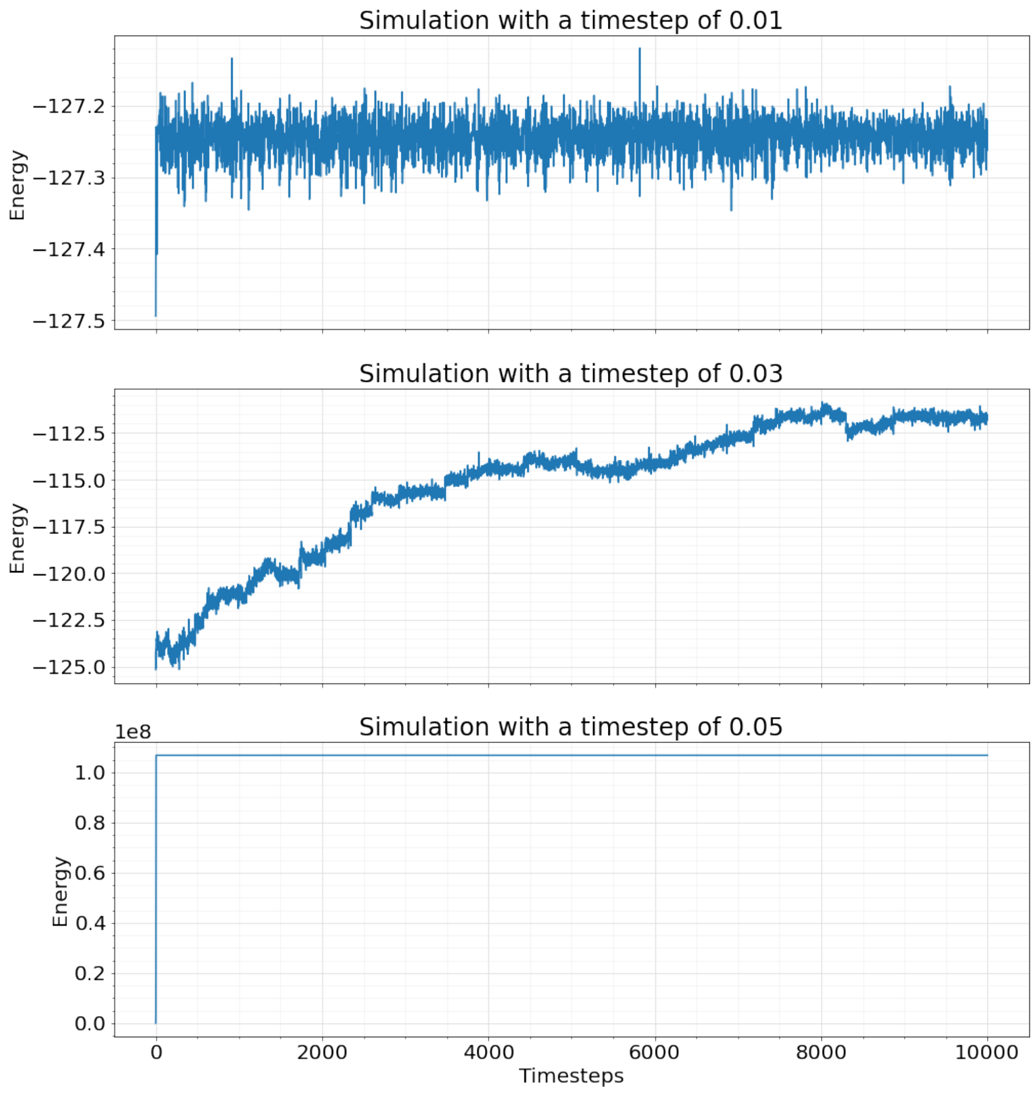

Figure 4.1: Simulation with different timesteps

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.

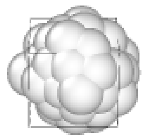
Figure 4.2: Simulation Snapshot

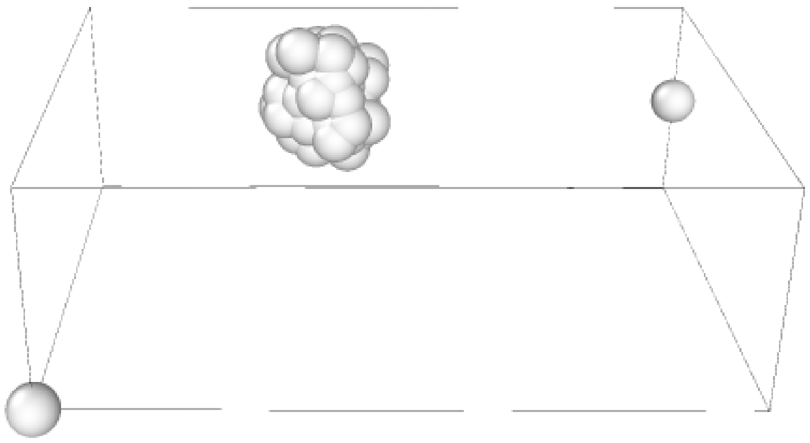
Figure 4.3: Simulation Snapshot

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

After incorporating the Berendsen Thermostat into the code it is interesting to look at the computational complexity of the simulation. With growing numbers of atoms the computation of should follow an order of something like O(N^2). The main culprit for this is the force-computation, as each interaction with all the other atoms in the simulation has to be computed. This is also shown in the next figure as the computation time follows a quadratic function.

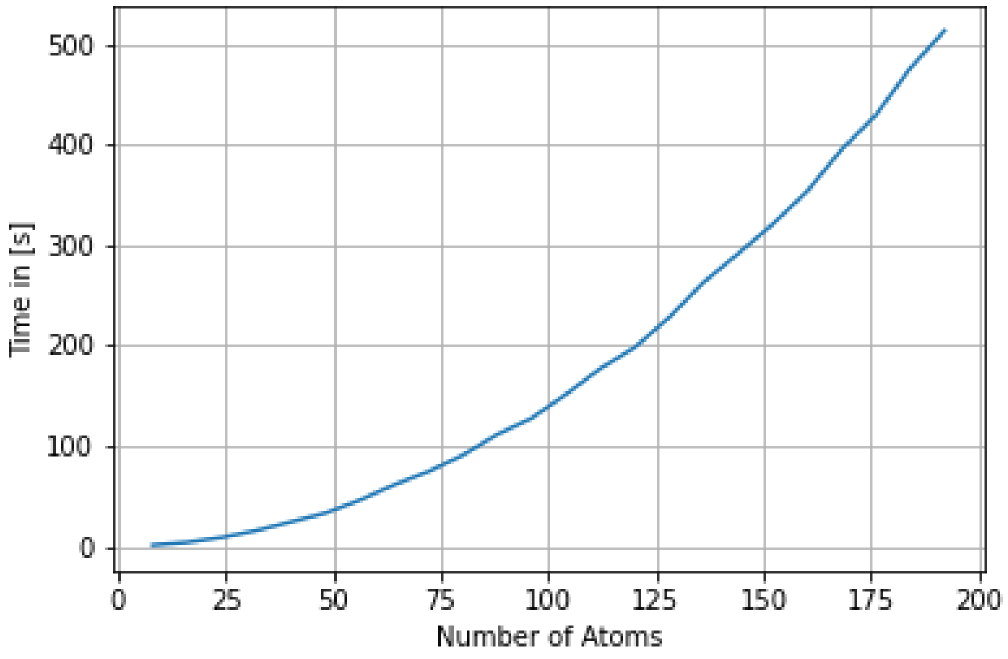

Figure 4.5: Simulationtime with the Berendsen Thermostat from 8 to 192 Atoms

Although all the interaction between all the atoms are computed they do not carry the same weight to the force that affects the atom. It should be rather clear that, the further apart atoms are, the smaller the force gets. After a certain distance it gets so small that it can be ignored. This leads to the idea to use neighborhood-lists that ignore the atoms outside of a certain radius, which was done in the next section.

4.3 Results from the Simulation with the Neighborhood-List

After running the simulation in the previous section, it was clear that they follow a computational complexity of the order O(N^2). This can be reduced to a linear order O(N) with the usage of neighborhood-lists. Only the atoms in a certain radius around the atom will be considered and a force will be added to the total force affecting the atom. This can be seen in figure 4.6 on a linear and a logarithmic scale.

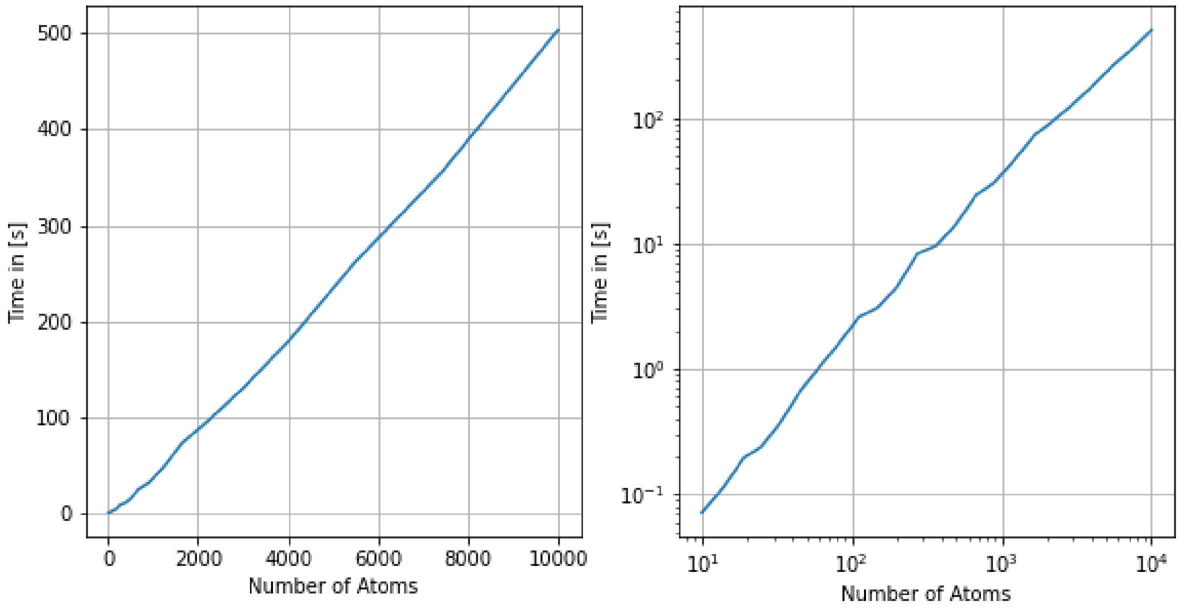

Figure 4.6: Simulationtime with the Neighborhood-List

4.4 Results from the Simulation with the Embedded-Atom Method Potential

In the next step the Embedded-Atom Method Potential was used to compute the potential energy and the forces between the atoms. A neighborhood-list was also used. With this it is possible to look at the actual physical properties of the materials in this case gold. In the figure 4.7 energy and temperature for different cluster sizes are plotted. It is possible to extract the melting point, the heat-capacity and the latent-heat from this diagram. This can be seen in the next figure 4.8 in relation to the size of the atom-cluster. All data in this figure has been read by hand so it has to be taken with a grain of salt.

The first interesting property is the melting point of the cluster, which can be seen in the different curvature in the graphs in figure 4.7 (for the red graph at 900K for example). The melting point increases as the clusters get bigger, but still is not near it's macroscopic equivalent of 1337 K.

It can also be seen that the potential energy of the median atom sinks with bigger clustersizes. To explain this, the ratio of surface to volume of the cluster and the fact that atoms at the border of the cluster need to have a higher energy, have to be considered. With increased clustersizes a smaller percentage of atoms is at the edge and increases the total energy of the cluster. The slope of the curvature at the melting point also increases with bigger clustersizes. In a big physical system this actually would be straight line where the melting point is and the simulated system seems to converge to that.

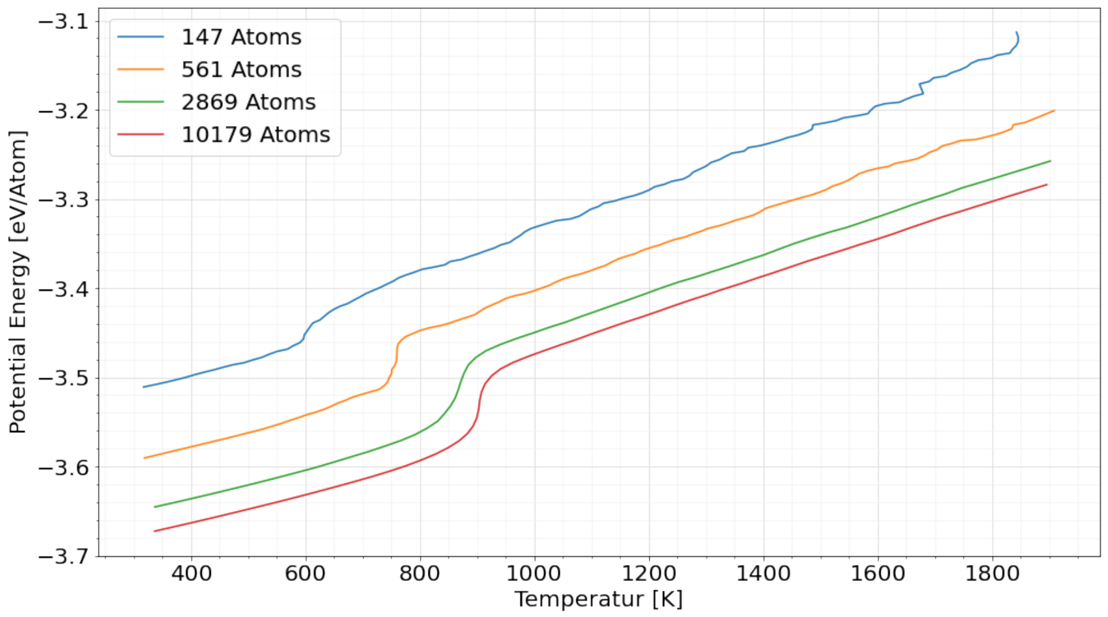

Figure 4.7: Gold Cluster Simulation

In the next figure 4.8 in the diagrams which plot the heat capacity and the latent heat, it can be seen that both seem to converge. The melting point on the other hand still seems to increase as it is still away from it's true melting point.

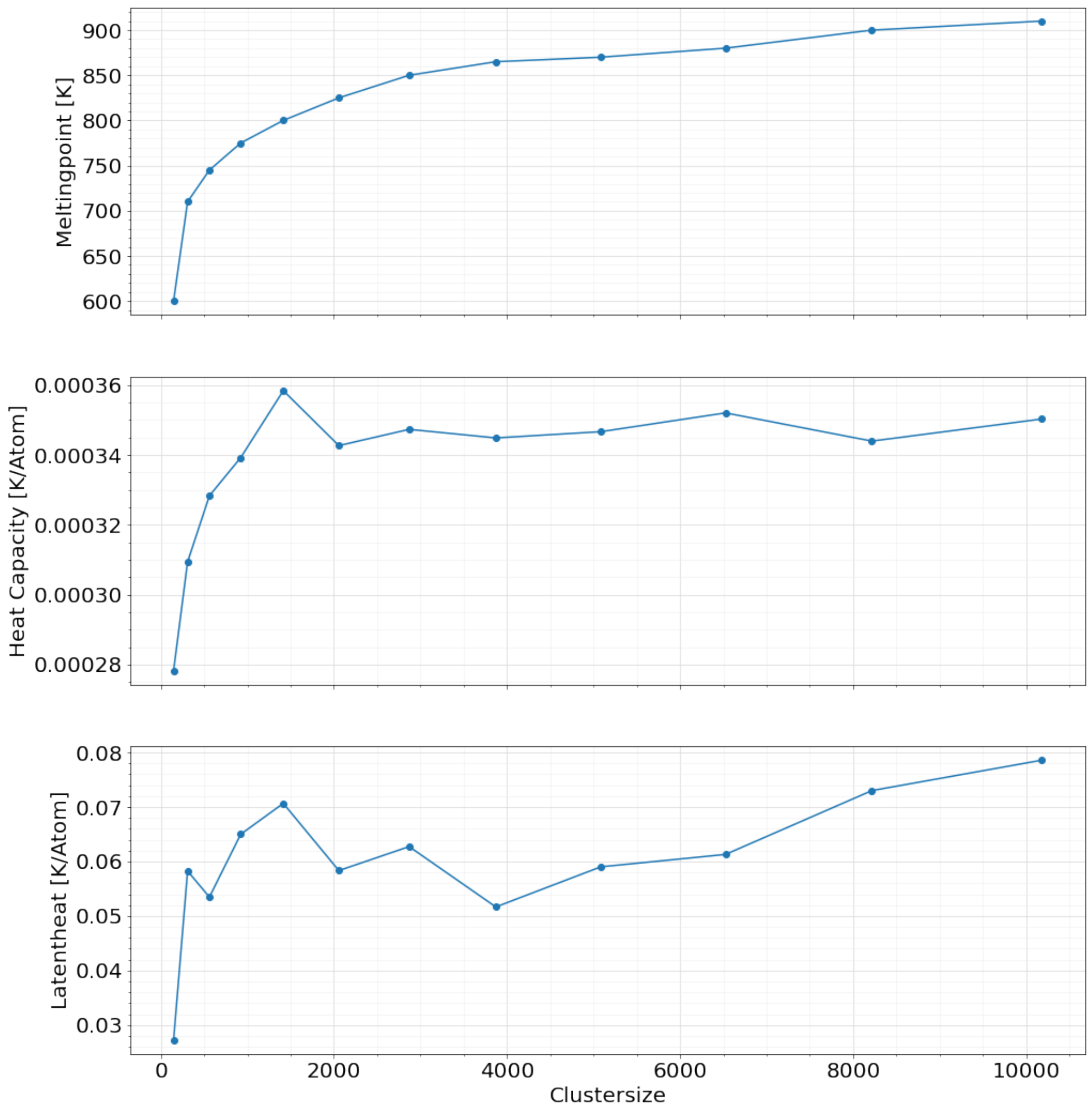

Figure 4.8: Melting Point, Heat Capacity and Latent Heat vs Clustersize

Chapter 5

Conclusion

Bibliography

[1] Fabrizio Cleri and Vittorio Rosato. “Tight-binding potentials for transition metals and alloys”. In: *Phys. Rev. B* 48 (1 1993), pp. 22–33. DOI: 10.1103/PhysRevB.48.22. URL: https://link.aps.org/doi/10.1103/PhysRevB.48.22.

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

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

[4] Raju P. Gupta. “Lattice relaxation at a metal surface”. In: *Phys. Rev. B* 23 (12 1981), pp. 6265–6270. DOI: 10.1103/PhysRevB.23.6265. URL: https://link.aps.org/doi/10.1103/PhysRevB.23.6265.

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

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

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

[8] Yanting Wang. *Mackay Icosahedron Structure Generator*. 2021. URL: http://www.pas.rochester.edu/~wangyt/algorithms/ih/ (visited on 08/01/2021).