Molecular Dynamics with C++
Final Report

Christoph Moser

5203023

christoph.mos.studium@gmail.com

August 31, 2021

Contents

List of Figures

1

Introduction

Since the early 1800s, it is known that all matter we can see is build from tiny particles, called atoms. With computers getting faster and faster, it is possible to simulate those particles on a computer. But as with many things there is trade-off between accuracy of the simulation and actually being able to run said simulation in a reasonable time frame while still getting correct results. Developing methods for this problem is the goal of molecular dynamics.

In an introduction to molecular dynamics the students were tasked to write their own simulation code in C++. The methods used, the implementation itself and results gained with the code are described in this report.

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.

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

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

2.1 Integration

From Ep. 2.1 we can obtain the force acting on each individual atom with its acceleration, velocity and position as shown in Eq. 2.2.

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

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

In the next step we need to propagate the simulation forward in time. One of the most popular algorithms to update the positions and velocities is the Velocity-Verlet integration algorithm. This scheme is often split up into two steps, the prediction shown in the Eqs. 2.3 and 2.4, and the correction step shown in Eq. 2.5 [cf. 7]. The force which is needed in the second step can be updated beforehand, based on the new positions from the first step.

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

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

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

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

2.2 Lenard-Jones-Potential

A popular pair potential to model interatomic interactions is the Lenard-Jones-Potential. fThe goal 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 Eq. 2.1 we can derive the forces from the potential energy. Now we have to formulate the equation for each atom, which was done in Eq. 2.7. We can get the force at each atom if we sum up the derivative of the potential energy between atom k and all the other atoms i times the normed vector to atom i from atom k.

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

To implement the equation we have to derive $\delta V / \delta r_{ik}$ analytically, which was done using the symbolic python library sympy.

```
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 as shown in Eq. 2.8.

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

The result from Eq. 2.8 can be used to calculated the interatomic forces from Eq. 2.7. 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 the opposite direction and therefore sign.

It can be seen that this algorithm behaves in the order $O(N^2)$ with increasing numbers of atoms N 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 a good approximation as the forces get smaller as the interatomic distance between the atoms increases. The next goal is to generate a list containing the nearest neighbors that are inside this cutoff-distance, which can be done by using domain decomposition [cf. 7]

2.3 Berendsen-Thermostat

The initial state of the atomic system is often far away form the equilibrium. To prevent the atoms from melting or evaporating because of excessive energy, thermostats are used to control the heat/energy of the atoms. A rather simple thermostat is the Berendsen-thermostat, which was used in the simulation.

It is typically implemented by rescaling the velocities by a factor as shown in Eq. 2.9. The factor is composed of the target-temperature T_0, the current temperature T, the timestep of the simulation Δt and the relaxation time τ.

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

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

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

With Eq. 2.10 the current temperature can be calculated and put into Eq. 2.9.

2.4 Embedded-Atom Method Potentials

Embedded atom-potentials are a good model for metallic bonding in solid or liquid states for different materials. The method used in the code was developed by Gupta [3] and the parameters were taken from Cleri & Rosato [1].

Compared to pair potentials, the potential energy is calculated from two contributions, shown in Eq. 2.11. $E_{repulsion}$ again is modeled like a pair-potential. The second contribution $E_{bonding}$ models the electron density between the metal atoms.

$$E_{pot} = E_{repulsion} + E_{bonding} \tag{2.11}$$

The Eqs. 2.12 and 2.13 show how $E_{repulsion}$ and $E_{bonding}$ were implemented in the code. The parameter A, ξ, p and q are also taken from the paper and can be fitted to different types of metals [1].

$$\mathrm{E}_R^i = \sum_j A e^{-p(r_i j / r_0 - 1)} \tag{2.12}$$

$$\mathrm{E}_B^i = -\Big\{ \sum_j \xi^2 e^{-2q(r_{ij}/r_0 - 1)} \Big\}^{1/2} \tag{2.13}$$

To finalize the implementation, the forces between the atoms also have to be calculated. For this the Eq. 2.7 has to be considered again.

3

Implementation

The simulation code was written in C++, most of it as functions, although the states of the individual atoms were saved in a container-class. The functions were also tested with unit-tests during the implementation. Plot 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)[8]. The python-code was written in jupyter-notebook. Additional libraries used were: googletest [4] for the unit-tests and eigen [5] for the arrays used for data storage in the container-class. Further software from the course was used for reading xyz-data and for the implementation of the neighborhood-search and embedded-atoms-method algorithms [7]. To be able to visualize and anaylse the results of the simulated clusters, the positions of the atoms were recorded and put into OVITO Basic [2].

While using the embedded-atom method potential, it was necessary to generate a variety of different sized clusters in the form of an icosahedron. An external Mackay Icosahedron Structure Generator [9], was used for this.

The implementation itself followed the milestones given in the course and went up till milestone seven. The main-code of each milestone was saved into an extra function and can be commented in and out in the main.cpp.

In the first step all variables are initialized. For example the initial structure of the atoms, when and where to save data and the timestep of the verlet-integration have to be defined. Some of the initial parameters can also be given to the program externally. After the initialization the simulation is run in the form of a loop. This main-loop is run for a number of times specified before. In the loop itself, the two verlet-steps, the calculation of the kinetic and potential energy and the evaluation of the force at each atom, are executed. In later milestone a thermostat and other effects are added to the loop. After the main-loop the acquired data is saved into a file and then processed after the simulation itself with python.

In python the contents of the data-file, which was generated from the C++-program is read and then plotted.

4

Results

4.1 Lenard-Jones Potential with direct Summation

First we look at a simulation with just the lenard-jones-potential. To decide on a good timestep for the simulation, a sequence of plots was created as can be seen in Fig. 4.1.

As the timesteps gets bigger , 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 here, would be 0.01 as the simulation is still very accurate and still runs quite fast.

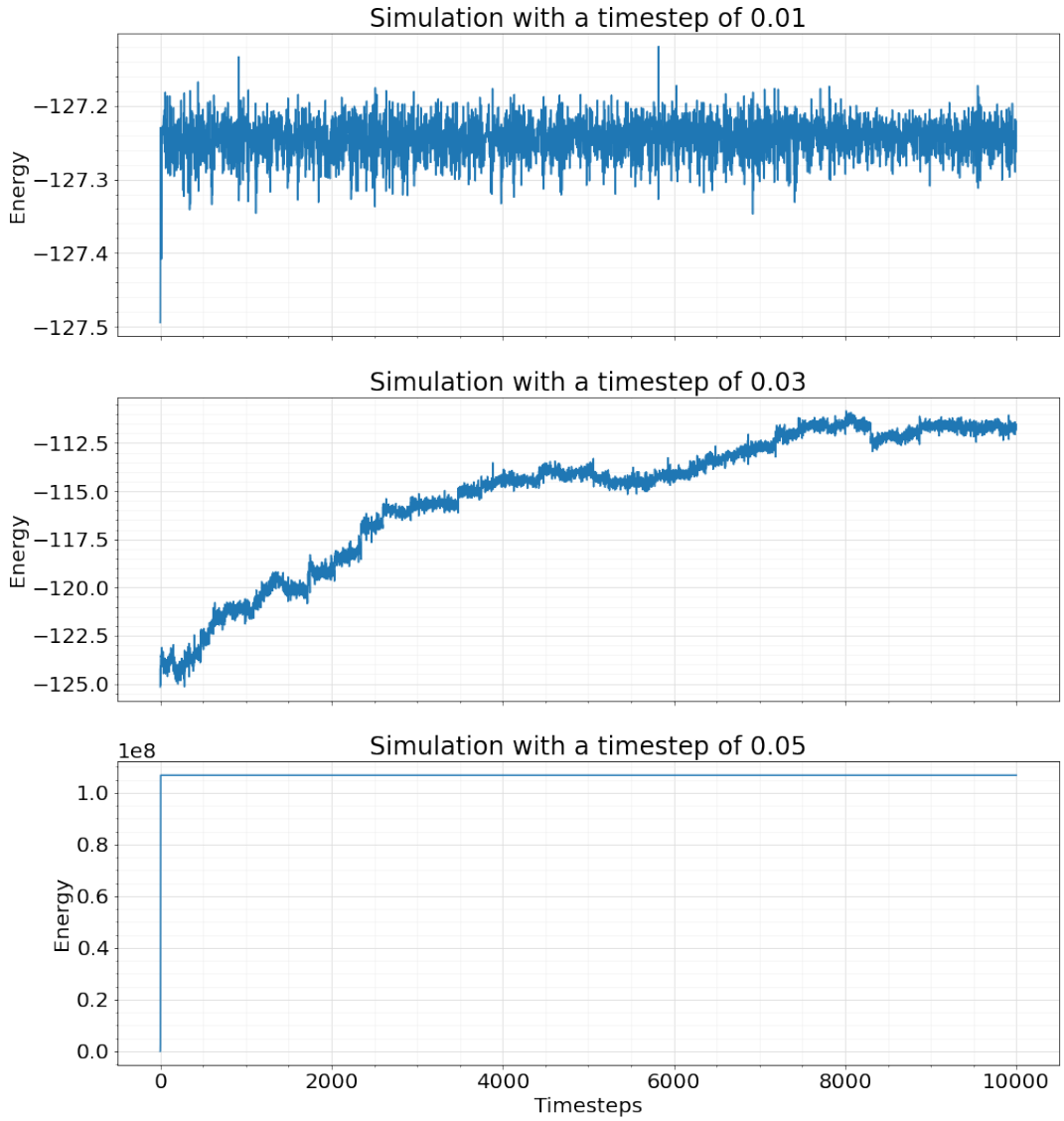

Figure 4.1: Comparison of the energy in the simulation with different timesteps

To visualize the simulation OVITO [2] was used and a series of snapshots, visible in Fig. 4.2 - 4.4, was created. The following images show one of the first simulations that was run, where the forces were calculated with the Lenard-Jones-Potential. The simulated cluster has a size of 52 atoms, and was given in the course lecture material [7].

Figure 4.2: Initial state of the cluster

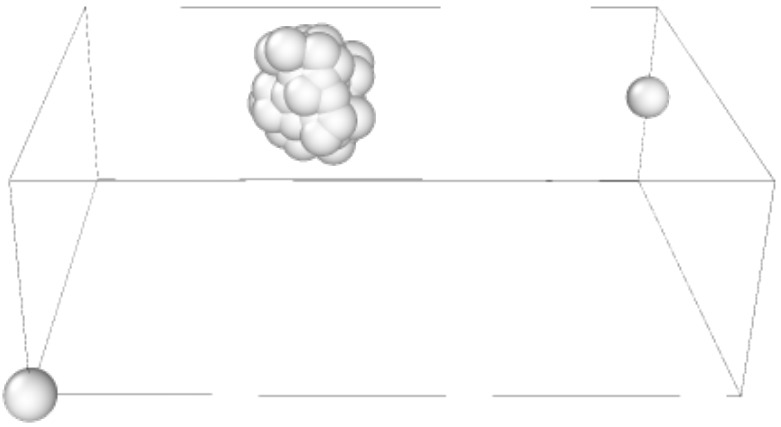

Figure 4.3: State after 2000 timesteps

Figure 4.4: State after 4000 timesteps

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

4.2 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-time should grow quadratically. 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 seems to follow a quadratic function.

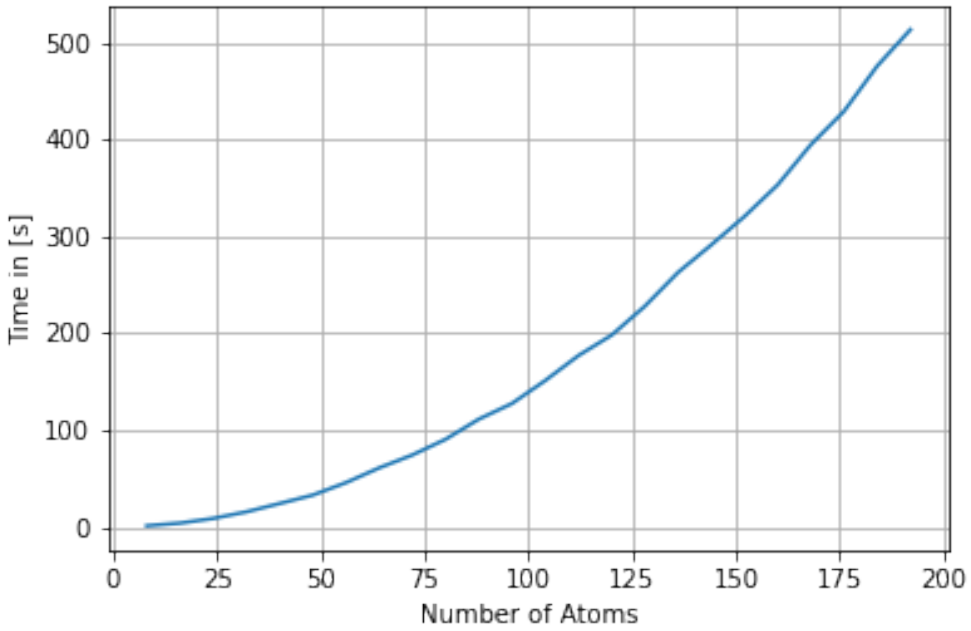

Figure 4.5: Comparison of the time needed to simulate 8 to 192 Atoms

Although all the interaction between all the atoms are computed, the individual forces 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 forces get. 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 has been done in the next section.

4.3 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 Fig. 4.6.

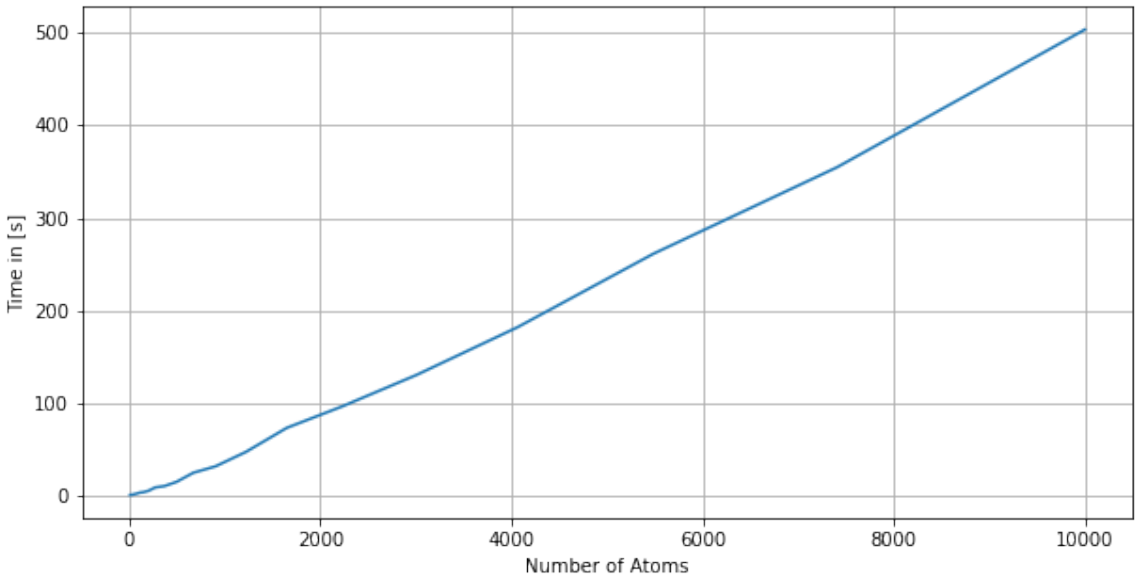

Figure 4.6: Comparison of the simulationtime with Neighborhood-lists

4.4 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 Fig. 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 Fig. 4.8 in relation to the size of the atom-cluster.

The first interesting property is the melting point of the cluster, which can be seen in the different curvatures 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 is still not not near it's macroscopic equivalent of 1337K.

It can also be seen that the potential energy of the average atom decreases 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 which leads to an increases in the total energy of the cluster. The slope of the jump at the melting point also increases with bigger clustersizes. For many atoms these curves would be piecewise linear with a high-slope section around the melting point. The simulated system seems to converge to that.

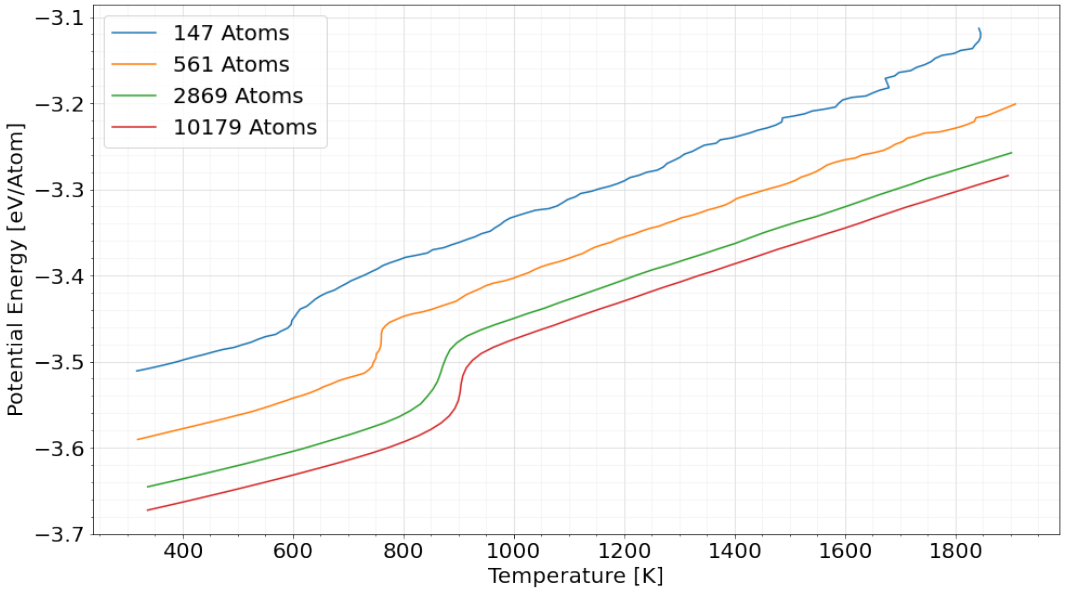

Figure 4.7: Gold cluster simulation with different quantities of atoms

In the diagram of Fig. 4.8 which plots the heat capacity and the latent heat, it can be seen that both seem to converge. On the other hand, the melting point still seems to increase as it is still away from it's true melting point.

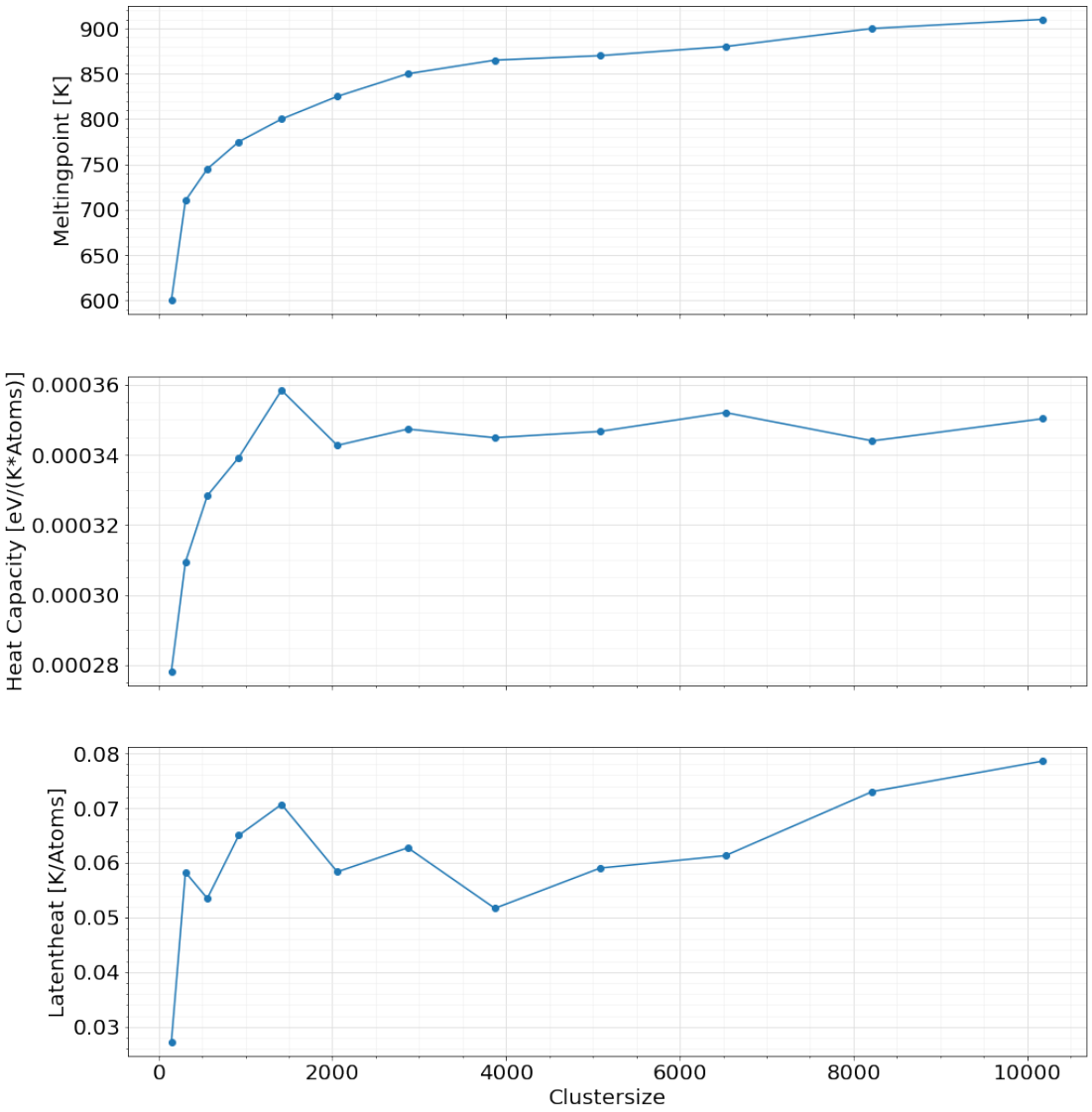

Figure 4.8: Melting point, heat capacity and latent heat vs clustersize

5

Conclusion

Over the duration of the course we dove into a very interesting mix between chemistry, physics and programming, and we were also able to look at different methods for simulating liquids and solids on an atomic scale.

First, integration-schemes were used to be able to propagate the particles forward in space with a constant force affecting the atoms. In the next step potentials were introduced to calculated forces and energies affecting the atoms. Initially the Lenard-Jones-Potential was used to calculate forces and energies. The potential was expanded with a neighborhood list, to reduce the computation time. Later the Embedded-Atoms-Method was used. With the introduction of thermostats a stable configuration of the atoms could be found without the cluster melting or evaporating.

Further improvements of the current implementation could include domain decomposition to better utilize modern parallel computer-architectures in a better way. There would also have been many more interesting themes to further dive into.

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] 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.

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

[5] Benoît Jacob and Gaël Guennebaud. *Eigen*. 2020. URL: https://eigen.tuxfamily.org/index.php?title=Main_Page (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 and Wolfram Nöhring. *Molecular Dynamics Course*. 2021. URL: https://imtek-simulation.github.io/MolecularDynamics/ (visited on 07/28/2021).

[8] JetBrains s.r.o. *CLion Eine plattformübergreifende IDE für C und C++*. 2021. URL: https://www.jetbrains.com/de-de/clion/ (visited on 08/16/2021).

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