



Molecular Dynamics with C++ Final Report

CHRISTOPH MOSER

5203023

christoph.mos.studium@gmail.com

August 16, 2021

Contents

1	Introduction	3
2	Methods	4
2.1	Integration	4
2.2	Lenard-Jones-Potential	5
2.3	Berendsen-Thermostat	5
2.4	Embedded-Atom Method Potentials	6
3	Implementation	7
4	Results	9
4.1	Results from the Lenard-Jones Potential with direct Summation	10
4.2	Result from the Simulation with the Berendsen Thermostat	12
4.3	Results from the Simulation with the Neighborhood-List	12
4.4	Results from the Simulation with the Embedded-Atom Method Potential	13
5	Conclusion	16

List of Figures

4.1	Comparison of the Energy in the Simulation with different timesteps . .	10
4.2	Simulation Snapshot 1	11
4.3	Simulation Snapshot 2	11
4.4	Simulation Snapshot 3	11
4.5	Comparison of the time needed to simulate 8 to 192 Atoms	12
4.6	Comparison of the simulationtime with Neighborhood-lists	13
4.7	Gold Cluster Simulation	14
4.8	Melting Point, Heat Capacity and Latent Heat vs Clustersize	15

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 in a computer. But as with many things there has to be made a 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.

$$\vec{f}_k = -\frac{\partial}{\partial \vec{r}_k} E_{pot}(\{\vec{r}_k\}) \quad (2.1)$$

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 acting on each individual atom with its acceleration, velocity and position as shown in 2.2.

$$\vec{f}_i = \frac{\partial E_{pot}}{\partial \vec{r}_i} = m_i \vec{a}_i = m_i \vec{\dot{v}}_i = m_i \vec{\ddot{r}}_i \quad (2.2)$$

As it 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. 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 algorithms 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]. Between the two steps the force can be updated.

$$\vec{v}_i(t + \Delta t/2) = \vec{v}_i(t) + \frac{\vec{f}_i(t)\Delta t}{2m_i} \quad (2.3)$$

$$\vec{r}_i(t + \Delta t) = \vec{r}_i(t) + \vec{v}_i(t + \Delta t/2)\Delta t \quad (2.4)$$

$$\vec{v}_i(t + \Delta t) = \vec{v}_i(t + \Delta t/2) + \frac{\vec{f}_i(t + \Delta t)\Delta t}{2m_i} \quad (2.5)$$

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

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

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

As shown in 2.1 we can derive the forces from the potential energy. Now we have to formulate the equation for each atom, which was done in 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 them.

$$\vec{f}_k = -\frac{\partial E_{pot}}{\partial \vec{r}_k} = \sum_i \frac{\partial V}{\partial r_{ik}} \hat{r}_{ik} \quad (2.7)$$

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

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

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 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 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 smaller, as the distance between the atoms increases. The next goal would be to generate a list with marked neighbors that are inside this cutoff-distance, which can be done by using domain decomposition [cf. 7]

2.3 Berendsen-Thermostat

The initial structure of the atoms is often times far away from 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 equation 2.9.

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

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

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

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.

3

Implementation

The simulation code was written in C++, most of it as functions, although the positions, velocities, etc. of the individual atoms were saved in a container-class. While writing the functions, these were also tested with unit-tests. 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. Alternatively, this could have been done directly in CLion with a plugin. Additional libraries used were: googletest [5] for the unit-tests and eigen [3] 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 Gupta-Potential algorithms [7]. To be able to visualize and analyse 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 code is partitioned 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 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 can be commented back into the code again in the true main.cpp file.

Independent of the milestones the code is generally structured similarly. It has to be noted that here just a generalization is given, and for further information it would be best to just look at the code itself as it should be well commented [6].

In the first step the variables have to be 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 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. In the later milestones the program was often directly called in python and more then one instance run in parallel to speed up the data-acquisition.

4

Results

4.1 Results from the Lenard-Jones Potential with direct Summation

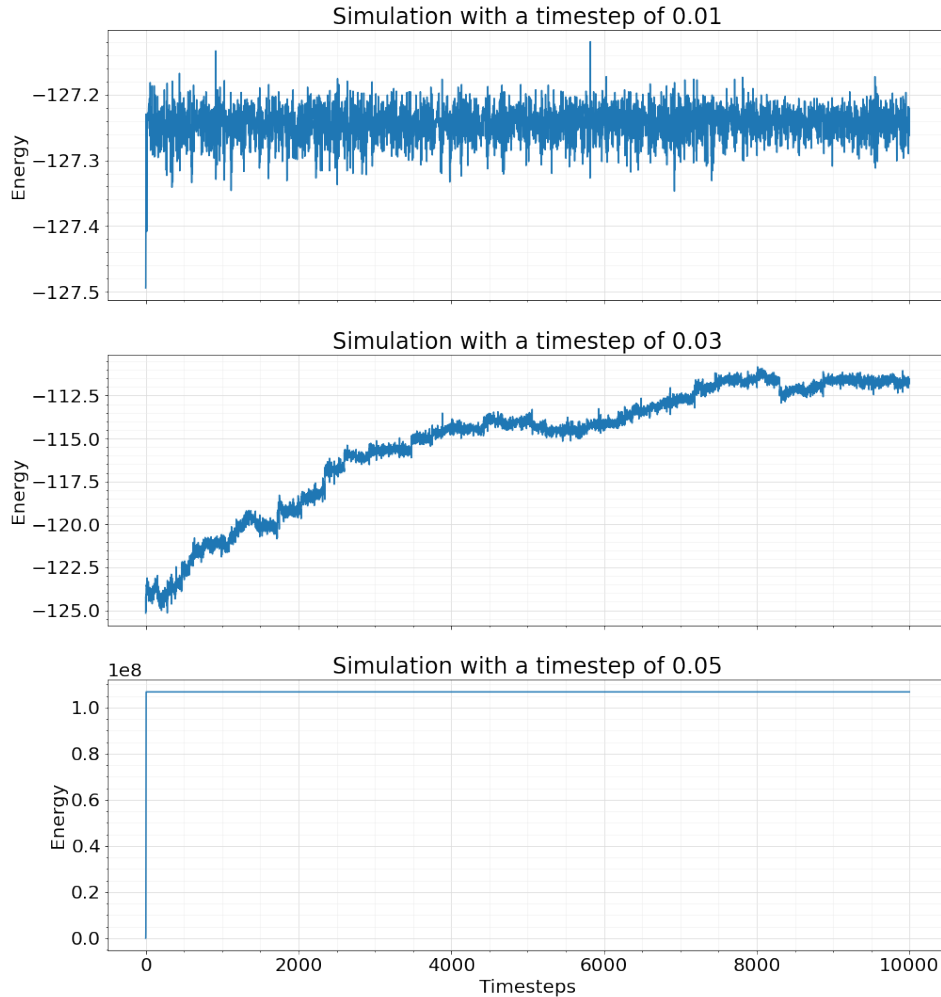


Figure 4.1: Comparison of the Energy in the Simulation with different timesteps

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, 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.

To visualize the simulation OVITO [2] was used and this series of snapshots 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 lecture [7].



Figure 4.2: Initial structure of the atoms

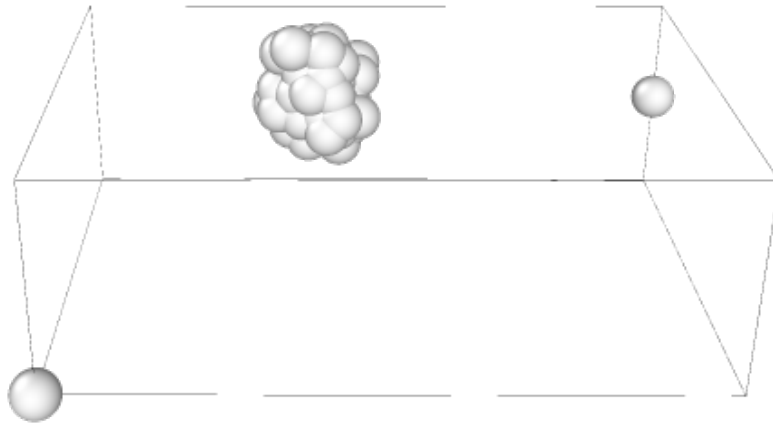


Figure 4.3: Structure of the atoms, after 2000 repetitions

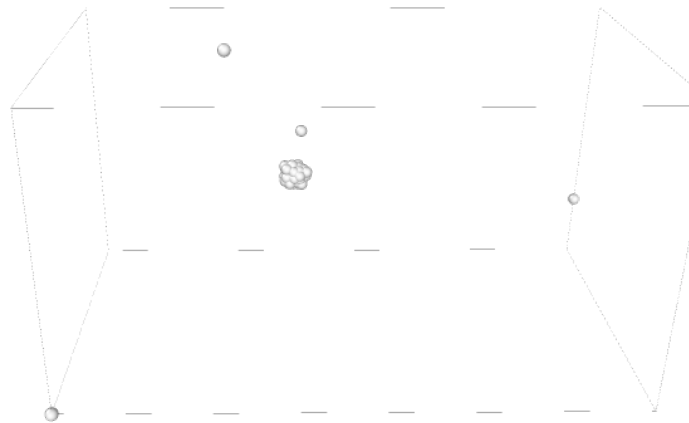


Figure 4.4: Structure of the atoms, after 4000 repetitions

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 escaped the initial blob and flew 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 seems to follow a quadratic function.

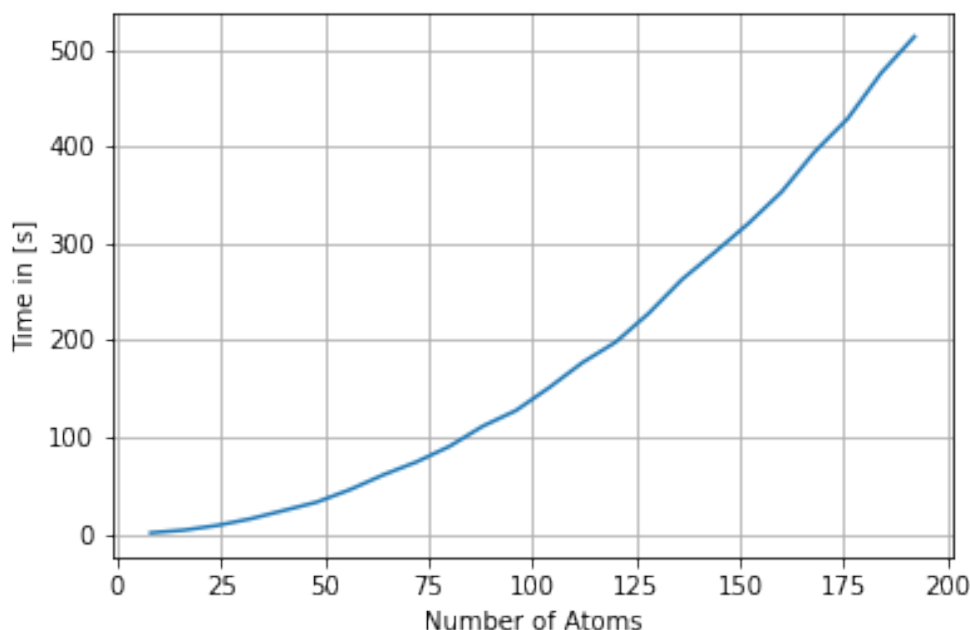


Figure 4.5: Comparison of the time needed to simulate 8 to 192 Atoms

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 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.

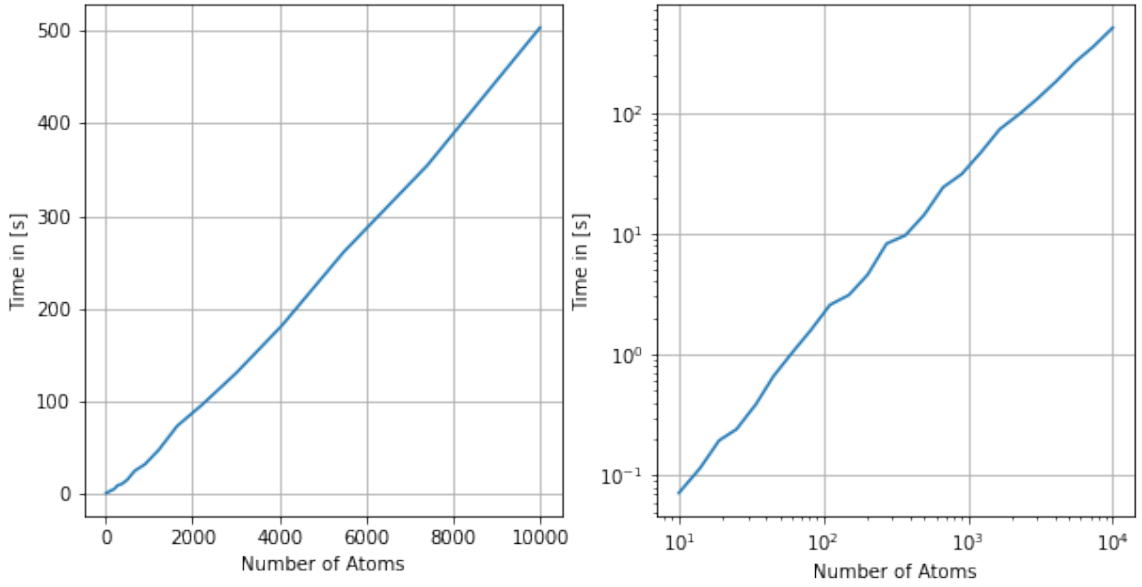


Figure 4.6: Comparison of the simulation time with Neighborhood-lists

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 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 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 near its macroscopic equivalent of 1337 K.

It can also be seen that the potential energy of the median 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 increase in 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 would be a straight line where the actual melting point is and the simulated system seems to converge to that.

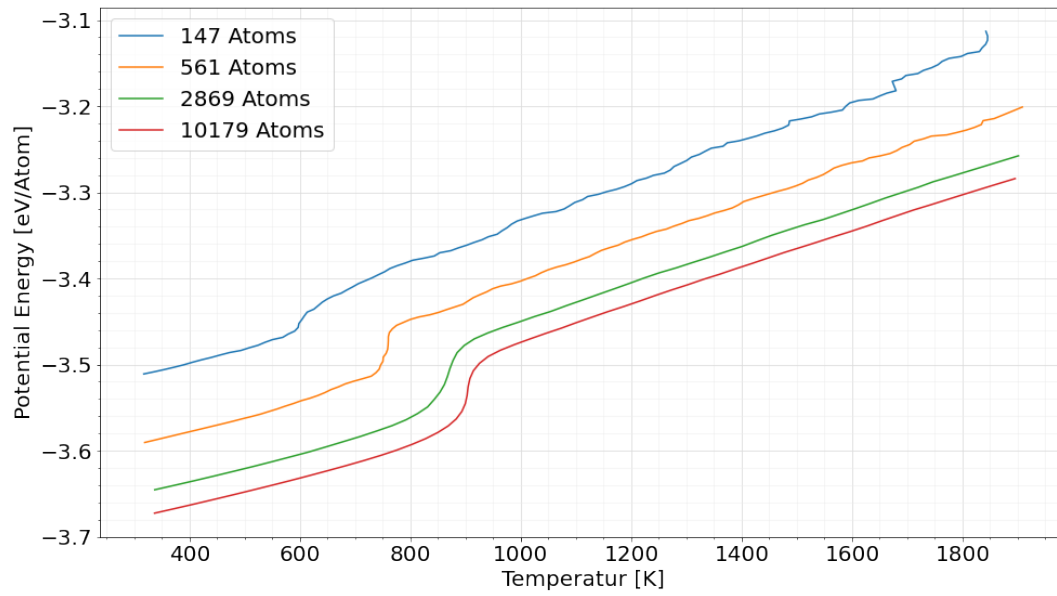


Figure 4.7: Gold Cluster Simulation with different quantities of atoms

In the diagram of figure 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.

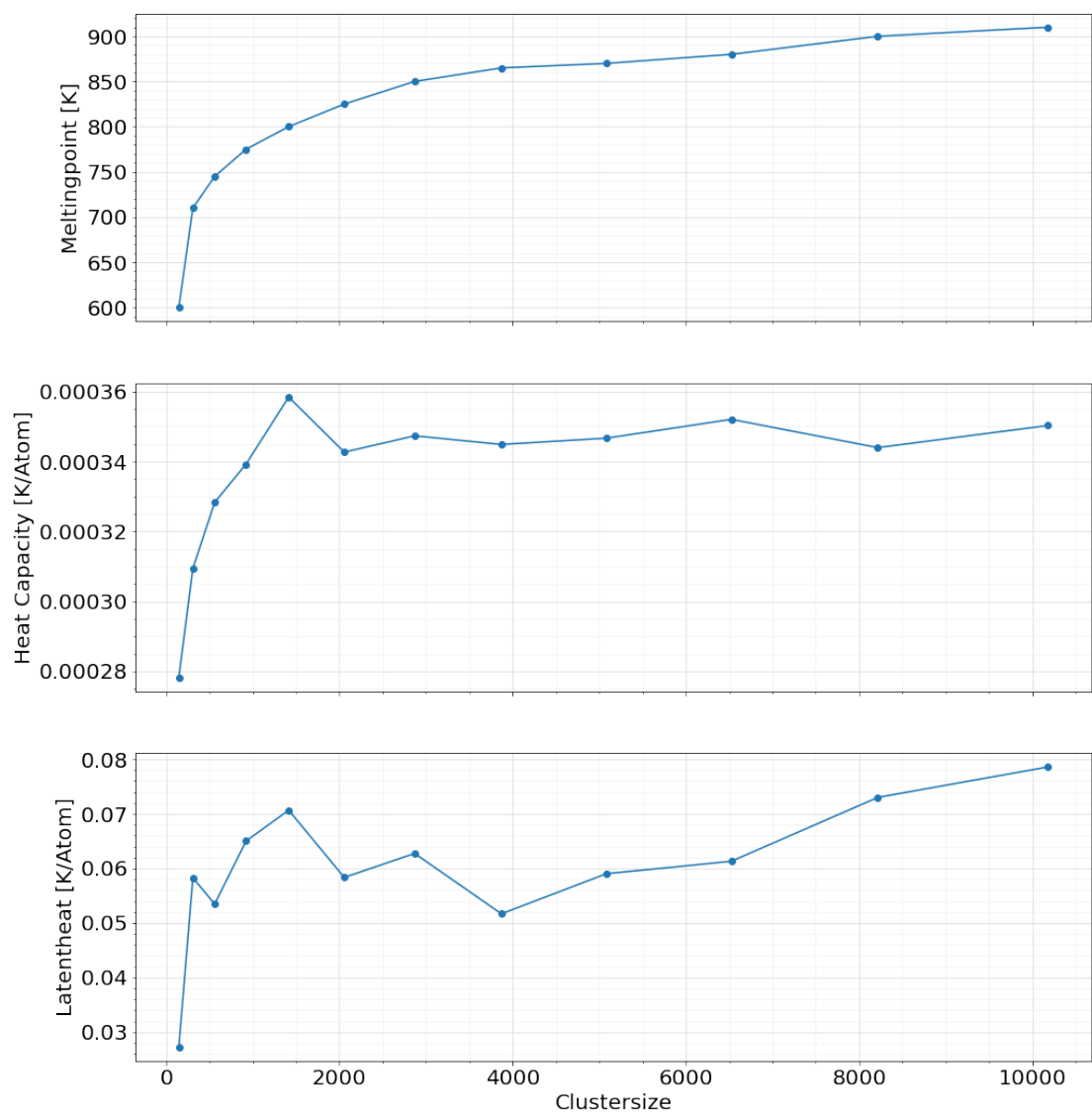


Figure 4.8: Melting Point, Heat Capacity and Latent Heat vs Clustersize

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.

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 calculate forces and energies affecting the atoms. Initially the Lenard-Jones-Potential was used to calculate forces and energies. 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.

There would have been many more interesting themes to look further dive into, which could not have been looked into because of a lack of time. For example the program could have been modified via domain decomposition to better utilize modern parallel computer-architectures in a better way.

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](https://doi.org/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](https://doi.org/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] 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).