University of Basel
Switzerland
Fall semester 2022

Praktikum

Computational Physics

Report by Sascha Tran

4th May 2023

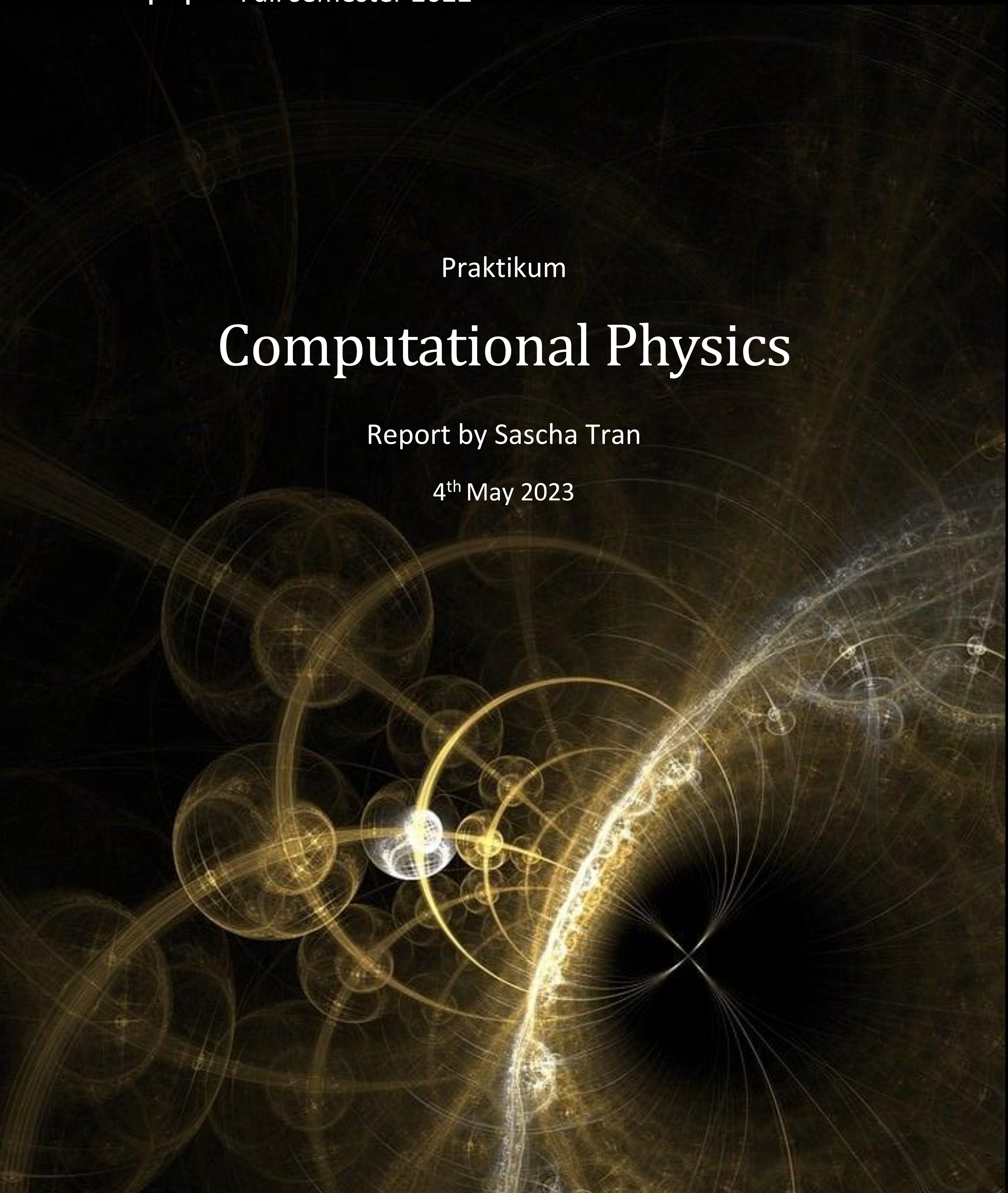

Table of Contents

1. Information

The task of this course was to solve two large projects out of 6 given. The goal of each project was to develop a short program which solves a simple physical problem.

The projects including their instructions can be found in the following script:
https://comphys.unibas.ch/TEACH/CP/course.pdf

Note that the subsection "Task" is a mere summary of the instructions. A more detailed description can be found in the script.

The aim of this report is to outline the solution of the following two projects:

- Sorting continuously growing data sets
- Determination of the melting temperature of silicon by a molecular dynamics simulation

2. Sorting continuously growing data sets

2.1. Introduction

The background and the instructions of this project can be found in the script mentioned in the beginning on page 0-18 to 0-22. All programs regarding this project can be found in folder P18.

In short, trivially inserting an element into a sorted list at the correct position using binary search has a computing time of $O(N)$ with N being the size of the current array. This is because every time an element is to be inserted, a new array with a size one bigger than the original must be created, and all elements must be copied into the new array. For large and growing data sets this kind of algorithm can be too slow, thus this project suggests using two-dimensional arrays.

2.2. Task and Expectation

- Implementation of the two-dimensional array including a subroutine that checks that the insertion was done correctly.

- Comparison of the performance of the trivial method with the two-dimensional method.
  It is expected that the two-dimensional method is much better than the trivial method, especially with greater amount of data.

- Examine the influence of the size of the data set of the two-dimensional array on the performance.
  It is expected that the greater the size of the initial data set is, the worse the performance of the two-dimensional array will become since it must distribute more data at the beginning.

- Examine the influence of the choice of the side lengths of the two-dimensional array on the performance.
  It is expected that a too big matrix will have a negative impact on the performance but also when the matrix is nearly as big as the initial length because redistribution is most likely then.

2.3. Results

2.3.1. Implementation of the two-dimensional array

The implementation of the two-dimensional array can be found in the file *sortingData.f90.* To check that the insertion is done correctly, the subroutines *matrixToVector* and *checkVector* were created. Note that the subroutine checkVector is commented out in the main program because the performance will be measured for the further tasks.

2.3.2. Performance of the trivial method with the two-dimensional method with different initial sizes of the data set

How the program works:

The program which measured the performance of the trivial and the two-dimensional methods can be found in the main program. To have a proper comparison, the randomly generated numbers are always the same for both methods.

The initial side lengths of the matrix were set to 500 x 500 which makes sure that it is never too small for all different sizes.

The program measures the time it took to insert 1000 elements with both methods for each of the following initial sizes of the data set: 1, 10, 100, 1'000, 10'000 and 100'000.

The results:

The data for the following plots can be found in the file *sumArr.csv* and *sumMat.csv.* There is a line "---" which separates the data with different initial sizes. The plots were created in the file *plotSize.ipynb.*

At the end of the 1000 insertions, the two-dimensional method is always faster than the trivial one in all plots.

However, it is also visible that the trivial method is performing better when the size of the data is rather small. In particular, the first couple of insertions are faster with the trivial method for all plots.

The bigger the initial size is, the longer it takes for both methods. Additionally, the two-dimensional method performs better than the trivial one after only a few insertions with a bigger initial size. For example, with initial size 1 it takes about 700 insertions before the trivial method is worse and with an initial size of 100 only about 300 insertions are necessary.

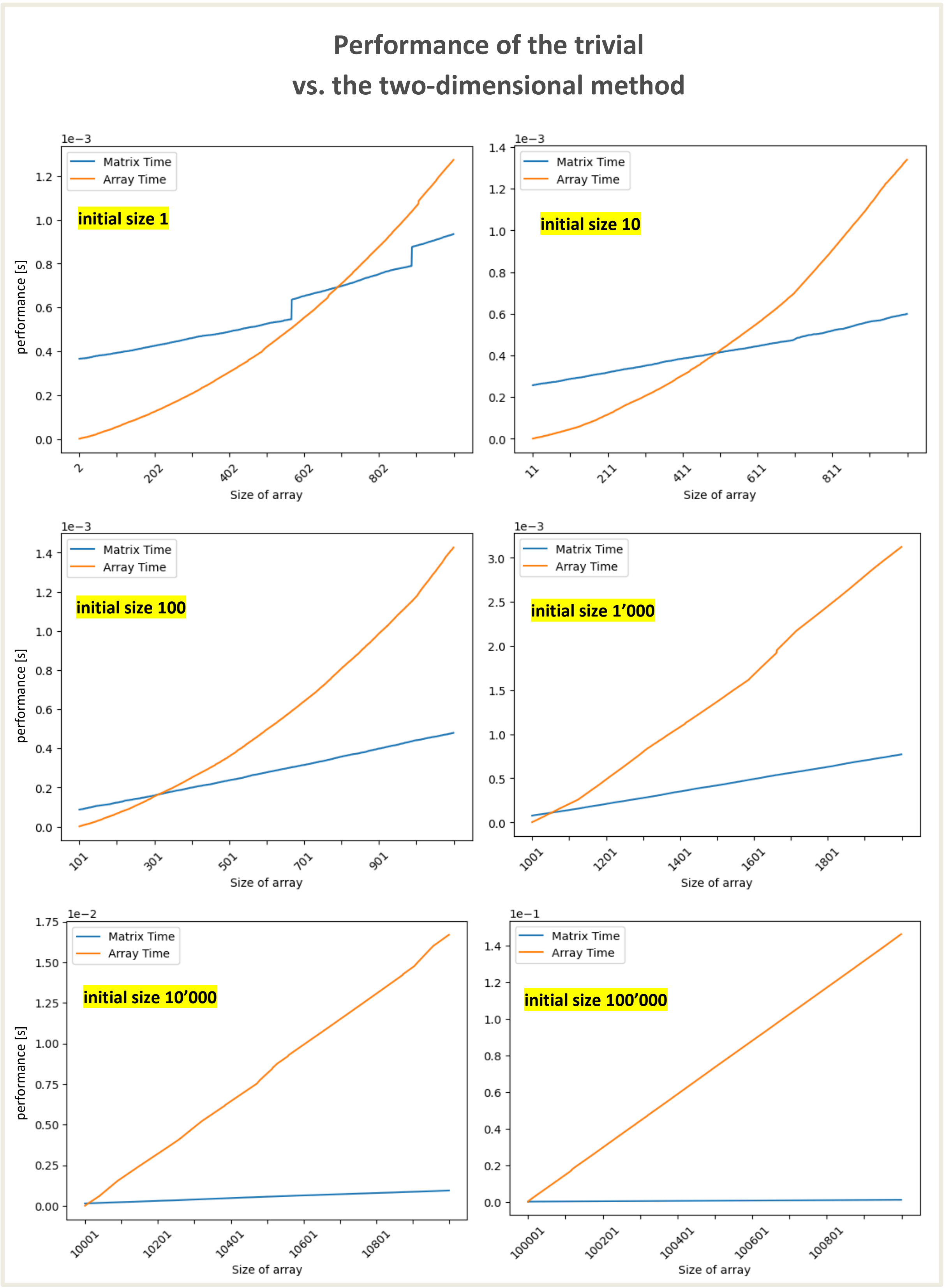

Figure 1: 6 plots visualizing the performance of the trivial method (Array Time) with the two-dimensional method (Matrix Time) with different initial sizes of the data set (highlighted in yellow).

2.3.3. Influence of the choice of the side lengths

How the program works:

The influence of the side lengths of the initial matrix with the dimension (*m* x *l)* was measured in the main program. The initial number of elements (length) was set to 1000 and 1000 insertions were done for each case.

The initial side lengths m and l were chosen as follows:

1. Initialize an array with 1000 elements and save them to a variable.
2. First set l = 100 and m = 20 (this guarantees that l x m is always greater than length = 1000).
3. Insert 1000 elements and measure the performance and the number of binary searches and save the data into a file.
4. Reset the list of elements to the initial array.
5. Keep the size of l but increase m by 10. If m is 100, increase it by 100. Redo steps 3 to 5 until m = 2100.
6. Now increase l by 100 and do the steps 3 to 5 again.
7. Do step 6 until l = 2100.
8. Redo steps 2 to 7 but with l and m switched.

The results:

The data for the following plots can be found in the file *mat.csv* and the plots were generated in the file *plotMat.ipynb.*

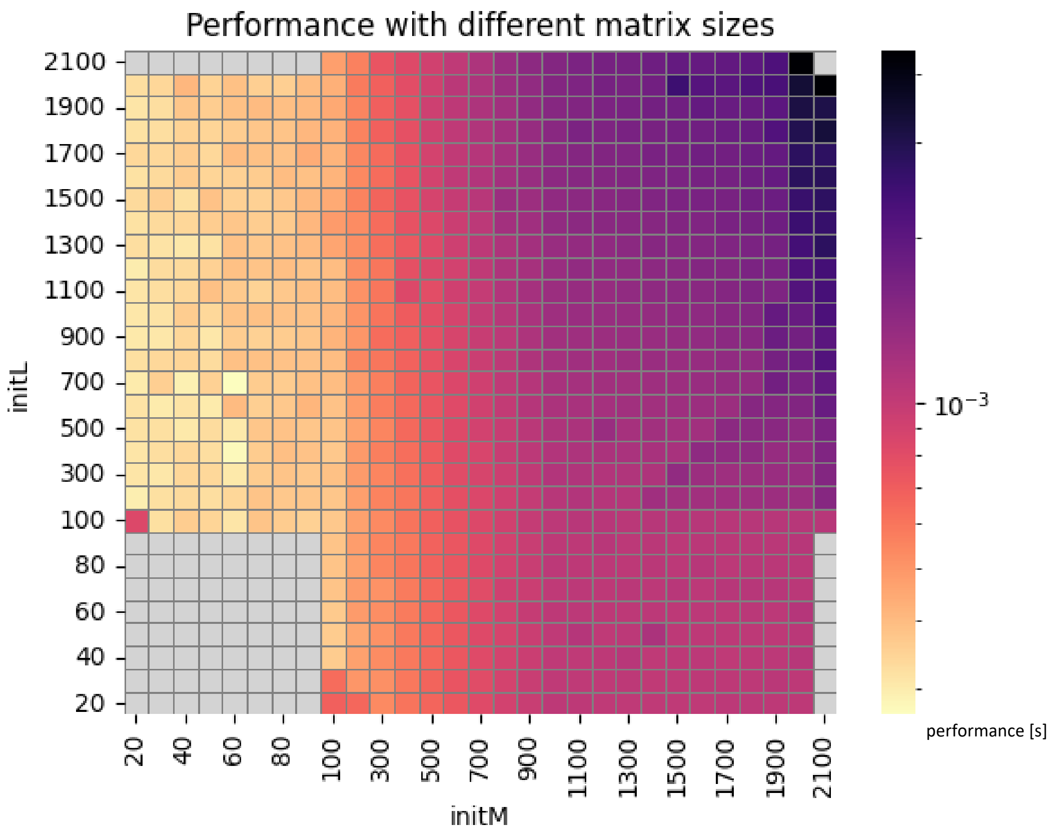

Figure 2: Performance of inserting 1000 elements in an array that already consists of 1000 elements using a two-dimensional array with different initial sizes m x l.

Figure 2 is a heatmap displaying the performance of inserting 1000 elements into an initial matrix with different sizes m and l. Note that the colour scale is logarithmically, and the greyish squares mean that there are no data for those sizes.

The darker the colour of a square is, the longer the performance was.

It can be observed that the squares become darker from left to right. This means that the bigger initM is, the slower the overall performance is. Additionally, at a certain width (around initM = 300), the squares become darker as well with bigger initL. But the colour transition is not as extreme from bottom to top than from left to right.

Furthermore, there are some darker squares around the grid size of 20 x 100 respectively 100 x 20.

In order to find out, what causes the performance difference between a l x m matrix and a m x l matrix, the total number of binary searches was also measured and plotted in the following figure.

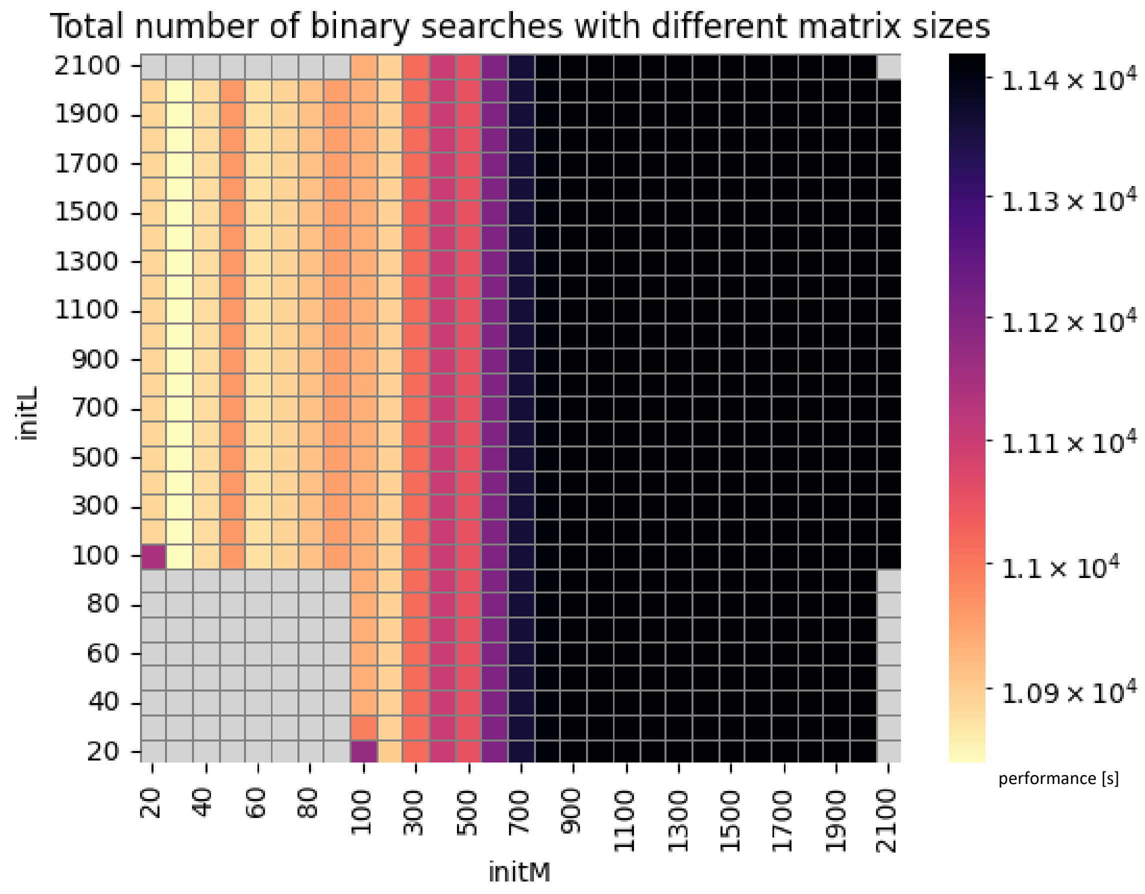

Figure 3: Number of binary searches when inserting 1000 elements in an array that already consists of 1000 elements using a two-dimensional array with different initial sizes m x l.

Figure 3 is a heatmap displaying the total number of binary searches needed, when inserting 1000 elements with a given initial matrix size m x l. Just like the above figure the greyish squares do not consist of data and the darker the colour of a square is, the higher the number of binary searches is.

It can be observed that overall, the greater initM is, the more binary searches were required.

It can also be observed that there was a higher number of binary searches required with the matrix sizes 20 x 100 and 100 x 20.

2.4. Discussion

2.4.1. Performance of the trivial method with the two-dimensional method with different initial sizes of the data set

After inserting 1000 elements, the two-dimensional method is always better in performance than the trivial one.

We can explain this observation by estimating the computing time for each method.

Trivial method:

- The algorithm uses binary search to find the position to insert the element. The computing time of a binary search is $O(log_2(N))$ with N being the number of elements in the array. This is binary search halves the number of possible positions after every iteration.
- To insert the element, space is needed. Since the array has a fixed size and is always full, a new array with a size one greater must be created and all elements must be copied into the new array. Therefore, the computing time here is $O(N)$.

All together this method has a computing time of $O\left(N + log_2(N)\right) \approx O(N)$.

Two-dimensional method:

- The algorithm uses binary search to find the position of which column the element is. This gives us $O(log_2(N_{col}))$ with N_{col} being the number of columns that are not empty.
- Then it again uses binary search to find the position within the found column to insert the element which has a computing time of $O(log_2(N_{row}))$ with N_{row} the number of non-empty rows in the column.
- To insert the element, it must shift at most N_{row} elements which is a computing time of $O(N_{row})$.

All together this method has a computing time of $O(log_2(N_{col}) + log_2(N_{row}) + N_{row})$.
If $N_{row} > log_2(N_{col})$, the computing time is $O(N_{row})$, otherwise it is $O(log_2(N_{col}))$.

With $N = 1000, N_{col} = 500 \; and \; N_{row} \leq 500$, the computing time of the two-dimensional method is always faster when inserting 1000 elements. (Initial length was set to the minimum of 0 elements.)

However, for the first few insertions the trivial method is faster because the matrix has to first be created which has a computing time of $O(N_{row} \times N_{col})$. This also explains the stair-like curve of the plot with initial size 1, because the matrix had to be redistributed.

2.4.2. Influence of the choice of the side lengths

The initial matrix sizes of 100 x 20 and 20 x 100 performed worse than there neighboring ones. The reason might be that when inserting 1000 elements into a 20 x 100 matrix (and 100 x 20 respectively), it is very likely that the matrix must be redistributed (even more than once).

Additionally, it seems that more binary searches were done with these sizes which naturally contributes to a worse performance.

However, even worse performance can be found at the top right corner where the matrix sizes are around 2000 x 2000. In general, this is as expected since initializing a big matrix takes much longer than a small one. Also, more binary searches are required to insert a number because the matrix is bigger.

Interestingly, the colour transition is much more visible from left to right (with increasing width initM) than from bottom to top (with increasing height initL). The transition in general makes sense as explained above but not the intensity. What I expected was a more symmetric heatmap.

When we look at the total number of binary searches, it is visible that the size of the height (initL) does not have an impact. The colour transition (light to dark) only goes from left to right. Therefore, it can be concluded that the binary searches mainly depend on the size of the width. This would also explain the phenomenon of the two different intense colour transitions above.

The question now is why the number of binary searches mainly depends on the width. When inserting an element into the matrix, the algorithm first searches for the column (using binary search) in which the element fits, then it searches for the position of where in the row of the column the element can be inserted. Therefore, when the matrix is extremely broad, independently of the number of elements existing in the matrix, many binary searches must be done, whereas a slimmer matrix does not require as many binary searches.

In summary, it can be concluded that the broader the initial matrix is, the worse the performance is, because the total number of binary searches is much higher. However, the height is not completely irrelevant if the width is at least 300 which can also be explained with the number of binary searches. Also, too small initial matrices reduce the performance because redistribution can occur.

Note that even though the heatmap in Figure 2 seems to outline the performance difference, it is on logarithmic scale which means the differences in performance are not that big. However, the initial size of the data set is only 1000 long in this measurement.

3. Determination of the melting temperature of silicon by a molecular dynamics simulation

3.1. Introduction

The background and the instructions of this project can be found in the script mentioned in the beginning on page 0-131 to 0-133. All programs regarding this project can be found in folder P131.

Briefly explained, in this project we want to find the melting temperature of silicon by running two molecular dynamics simulation. In both simulations a chunk of liquid silicon is sandwiched between crystalline silicon. The difference between these two simulations is the different initial velocities of the liquid part. One can imagine that one chunk is extremely hot and the other is only moderately hot.

3.2. Material

This section outlines the files and software specifically required for this project.

Files[1]:

- bazant_lib.f90
- md.f90
- cold.dat
- hot.dat

Software:

- Vesta[2]

3.3. Task and Expectation

3.3.1. Harmonic oscillator

- Write a little program that implements the velocity Verlet algorithm (Eq. 98 slide 0-113). Test the program for a harmonic oscillator.
- Check that the total energy (potential plus kinetic) is approximately conserved and that that the trajectories are periodic.
- Observe the quality of the energy conservation as a function of the size of the time step h.

  It is expected that the quality of the energy conservation increases with smaller time steps. After all, the trajectories are much more precise with smaller time steps.

3.3.2. Melting temperature of silicone

- Add the Verlet algorithm and the calculation of the temperature T of the system using the given equation (Eq. 113) to the file *md.f90*.

[1] https://comphys.unibas.ch/teaching.htm, The files are found in the first .tar folder of this website.
[2] https://jp-minerals.org/vesta/en/

- Verify that the energy is approximately conserved when running the MD program with hot.dat or cold.dat.
- Determine the melting temperature and visualize the system using V_Sim.

  The energy should be approximately conserved since the MD simulation is mainly based on the above Verlet algorithm.

3.4. Results

3.4.1. Harmonic oscillator

- The implementation can be found in the file *verlet.f90* and the plots were created in *plot.ipynb.*
- The simulation was held with the following initial parameters:
  R = 1.0, V = 0.0, M = 1.0, h = 0.001, k = 2.0, len = 30’000 (len is the number of time steps)
  With these initial parameters Figure 4 shows that the trajectory is indeed periodic (“Block 1”).

  Furthermore, the total energy is approximately conserved.
  The absolute difference between the first and last computed energy is 5.0 · 10^{-7} J which is nearly 0.
  The computation can be found in the plot-file “Block 2”.

- Table 1: Quality of energy conservation as a function of the time steps. displays the difference between the greatest and smallest total energy in each simulation. The initial parameters are the same as above except for the time steps. Each row is a simulation with a different time step.

Table 1: Quality of energy conservation as a function of the time steps.

Time step	**Max. difference in total energy [J]**
1	0.5000000
1E-01	0.4999997E-2
1E-02	0.5000000E-4
1E-03	0.5000000E-6
1E-04	0.5000004E-8
1E-05	0.8445467E-11
1E-06	0.2043921E-12
1E-07	0.4676259E-12
1E-08	0.6609158E-12
1E-09	0.6150636E-13
1E-10	0.1543210E-12
1E-11	0.1800782E-12
1E-12	0.1776357E-14
1E-13	0.000000
1E-14	0.000000
1E-15	0.000000
1E-16	0.000000
1E-17	0.000000
1E-18	0.000000
1E-19	0.000000
1E-20	0.000000

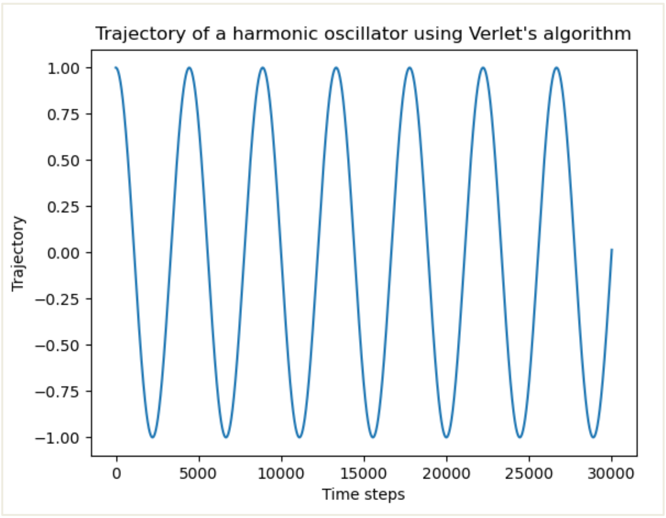

Figure 4: Trajectory of a harmonic oscillator

It is visible that in general, the smaller the time steps are the better the energy conservation is.

3.4.2. Melting temperature of silicone

- The Verlet algorithm as well as the calculation of the temperature can be found in the file *md.f90.*
- When running the MD simulation for 1’000 time steps with cold.dat, the absolute energy difference of the last time step and the first time step is 0.02 J and with hot.dat the difference is 0.008 J. Therefore, it can be confirmed that the energy is conserved. The computation can be found in the plot-file (“Block 3”).

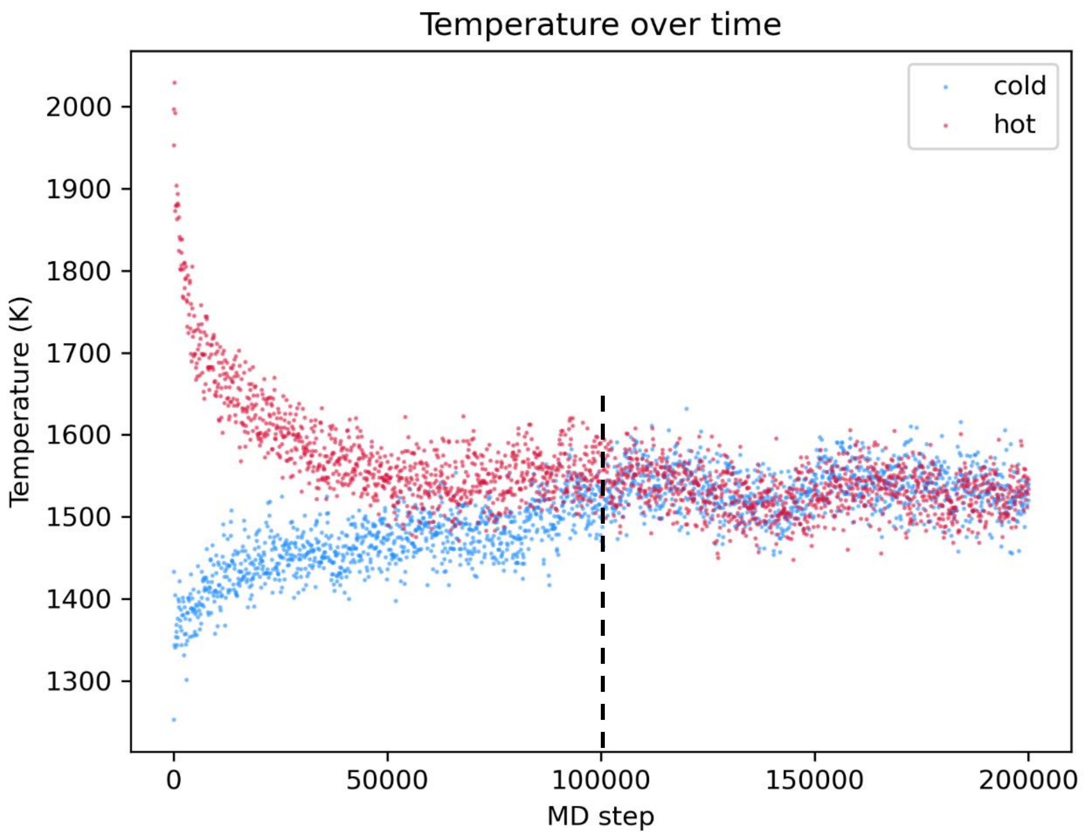

Figure 5: Temperature of MD simulation using hot.dat and cold.dat.

- In Figure 5, both graphs (red and blue) have reached their equilibrium after about 100’000 MD steps. To determine the melting temperature, the average of all data points of hot.dat respectively cold.dat starting at 100’000 MD steps was evaluated. Additionally, the average of both results was taken which gives us a melting temperature of 1532.61 K (highlighted in Table 2). The computation can be found in the plot-file (“Block 4”).

Table 2: Melting temperature of silicone. Temperature computed by taking the average of the data points of the MD simulation starting from the 100000th MD step.

Data points from	Melting Temperature [K]
hot.dat	1531.24
cold.dat	1533.98
both files	1532.61

- The following two figures are a visualization of the system in the initial state, when the chunk was inserted, and the final state after the MD simulation. As a result of a recommendation, the visualization was performed using Vesta instead of V_Sim.

Visualization of the system

with the insertion of the hot liquid silicone

hot.dat

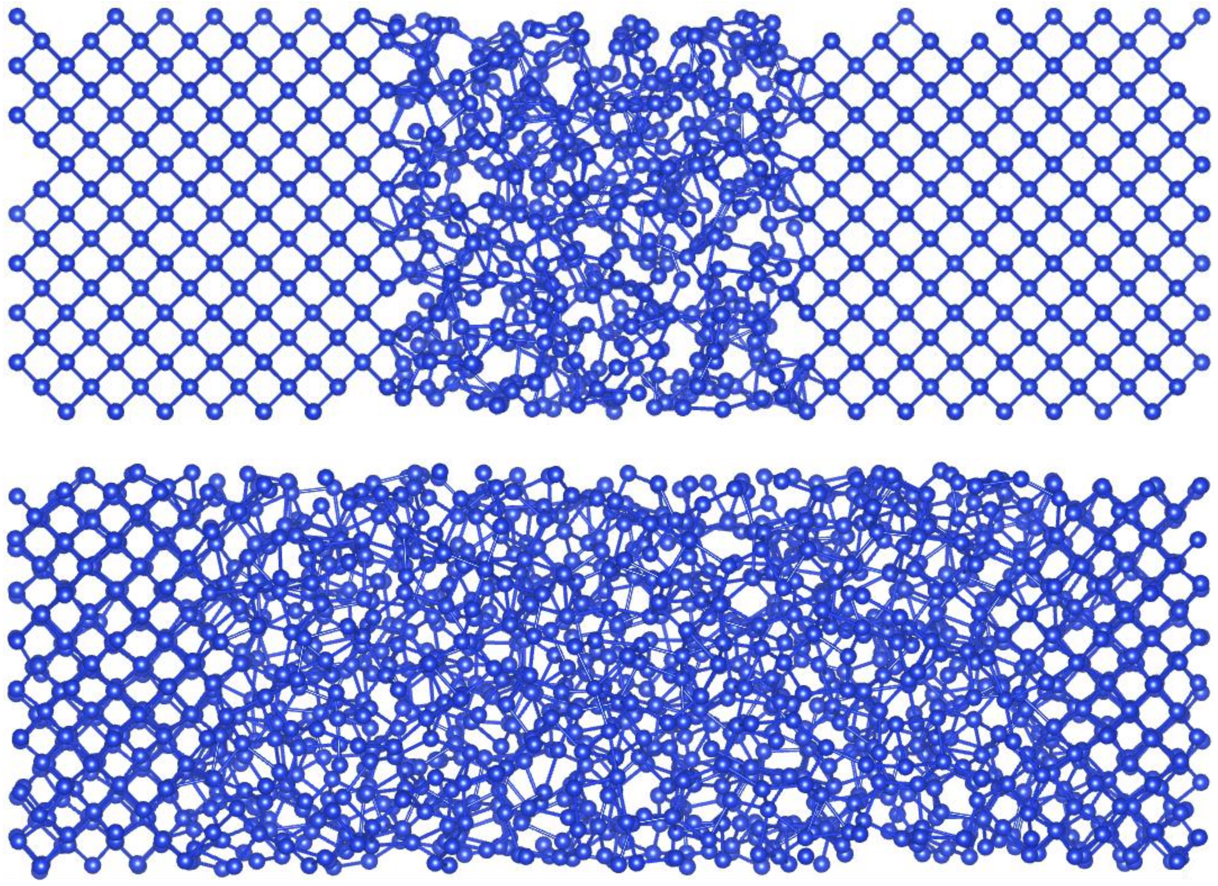

Figure 6: Initial state of the system (top) after insertion of the ***hot liquid*** *silicone and final state of the system (bottom) at the end of the MD simulation.*

The top molecule in Figure 6 is the state of the system at the time when the chunk was inserted. The part where the atoms are not perfectly aligned at all are the liquid regions. After the MD simulation, it can be observed that the liquid region has expanded.

As for Figure 7, the liquid region has slightly decreased after the simulation.

Visualization of the system

with the insertion of the moderately hot liquid silicone

cold.dat

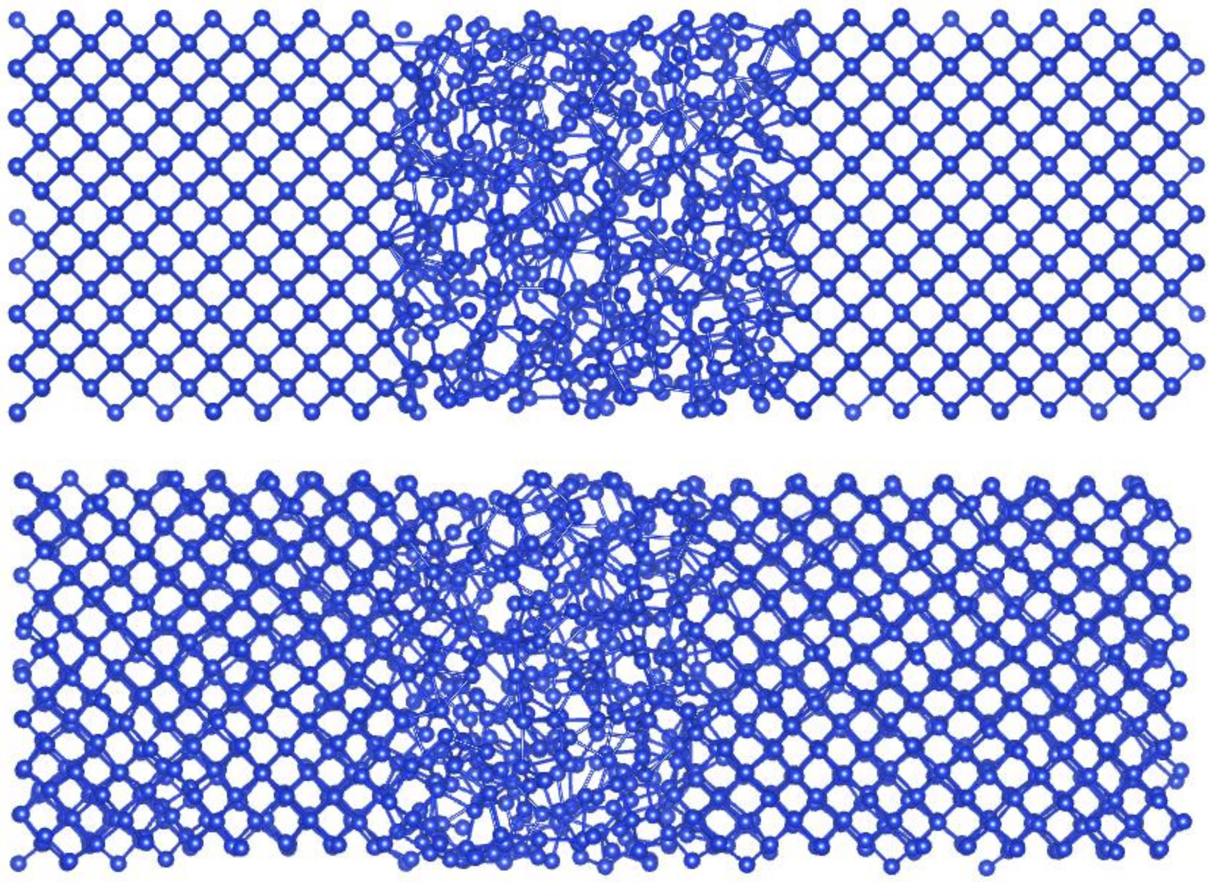

Figure 7: Initial state of the system (top) after insertion of the ***moderately hot liquid*** *silicone and final state of the system (bottom) at the end of the MD simulation.*

3.5. Discussion

3.5.1. Harmonic oscillator

The trajectory in Figure 4 looks like a harmonic oscillator. Since the program simulated the harmonic oscillator, this figure was expected. It is also clearly visible that the energy conservation increases with smaller time steps, this is most likely since the trajectories are computed more precise with smaller time steps.

3.5.2. Melting temperature of silicone

An explanation of the melting temperature of silicone can be found in the script. The temperature is as expected.

An explanation for the visualization can also be found in the script. The hotter liquid region expanded more than the cold part because the atoms are moving faster in a hot environment.

4. Summary

This report investigated two different projects.

The first project is about inserting elements into an increasing data set using two different methods, the trivial and the two-dimensional method.

It can be observed that the trivial method is faster with small sizes of a data set. But after inserting 1’000 elements, independently of the initial size of the data set, the two-dimensional method is faster.

A performance difference with different initial side lengths of the matrix was also observed. The initial matrix should not be too small relative to the size of the data set because the matrix has to be most likely redistributed. However, it should not be too big either because then the number of binary searches is very high, and this worsens the performance. An interesting observation was that the initial size of the width had a greater negative impact on the performance than the height, which was explained by the increasing number of binary searches with increasing width but not with increasing height.

As for the second project, it was about using MD simulation to determine the melting temperature of Silicone. The method to find the melting temperature was to simulate a solid system which has a hot liquid silicon inserted and another solid system with a colder liquid silicon. Their equilibrium was then the melting temperature which was determined at 1532.61 K.