DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Auto-Tuning Verlet List Skin Lengths in AutoPas

Daniel Asch

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Auto-Tuning Verlet List Skin Lengths in AutoPas

Auto-Tuning der Größe des "Verlet List Skin" in AutoPas

Author:	Daniel Asch
Supervisor:	Bungartz Hans-Joachim; Prof. Dr. rer. nat. habil.
Advisor:	Newcome Samuel
Submission Date:	15.03.2023

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.03.2023 Daniel Asch

Abstract

In molecular dynamics reducing the computation time is crucial for the simulation of large environments within reasonable time. For this purpose different algorithmic configurations can be used, with parameters that can be further adjusted to the environment. This is not easy to do manually and the optimal setup can change during the simulation. A solution to these issues is AutoPas, which automatically tunes the configuration. In this work, tuning over the rebuild frequency of the Verlet Lists-based configurations offered by AutoPas is investigated. The results show how this parameter affects the runtime in different scenarios and what the advantage is to use variable rebuild frequencies tuned by AutoPas.

Kurzfassung

In der Molekulardynamik ist die Reduzierung der Laufzeit von entscheidender Bedeutung, um große Umgebungen zu simulieren, ohne enorm viel Zeit zu beanspruchen. Um dies zu erreichen, können verschiedene algorithmische Konfigurationen verwendet werden, deren Parameter weiter an die Umgebung angepasst werden können. Dies ist manuell nicht einfach zu bewerkstelligen und die optimale Konfiguration kann sich während der Simulation ändern. Eine Lösung für diese Probleme ist "AutoPas", das die Konfiguration automatisch abstimmt. In dieser Arbeit wird die Abstimmung über die "Rebuild Frequency" der von "AutoPas" angebotenen "Verlet-Lists"-basierten Konfigurationen untersucht. Die Ergebnisse zeigen, wie sich dieser Parameter in verschiedenen Szenarien auf die Laufzeit auswirkt und welche Vorteile die Verwendung variabler, von "AutoPas"abgestimmter "Rebuild Frequencies" hat.

Contents

1. Introduction

In Formula 1, teams don't use the real car for tests in the wind tunnel to save money. This is possible because simulations are getting closer to the reality and don't consume too much time anymore [1].

This is just one of many examples of how simulations are becoming increasingly important and opening up new possibilities. Molecular systems typically contain a large number of instances making it impossible to research the properties analytically. Molecular Dynamics is able to help by simulating all particles numerically [2]. Furthermore, simulating an environment without gravity or an experiment in a vacuum may be cheaper and easier than performing it in real life. On the other hand this is computationally costly making efficient algorithms very important [3].

In order to reduce the time needed for the calculation some parameters can be optimized. The optimal configuration differs from one environment to the next and has to be found first.

This is the purpose of "AutoPas", a molecular dynamics software that is tuning automatically over several available setups to reduce computation time [4]. One of the used methods in this application is 'Verlet Lists' to minimize the calculations. In this work, the opportunity of further optimization of this particular approach is evaluated. This is done by replacing a single with a variable rebuild frequency that determines the interval between the steps for distance calculations.

Molecular Dynamics can be used in different research areas. For example to discover the functionality of drugs on the membrane proteins by understanding the membrane-based macromolecular targets [5]. Another topic where molecular simulations can be applied is predicting the structure of RNA to get information about the correlation of the structure and biological function [3].

2. Molecular Dynamics

2.1. What is Molecular Dynamics

The goal of Molecular Dynamics is to simulate and analyze the behavior and motion of a finite number of atoms interacting with each other. The complexity of the subject lies in the number of atoms. To obtain a relevant result and useful information, often a large number of particles is needed [6].

Nowadays, as computers become more powerful, performance is less and less of a problem, but it is still very important to be efficient in large simulations. There are many different ways to optimize molecular dynamics, such as using the Newton's 3rd law to cut the number of necessary calculations in half [4]. Other possibilities to decrease the duration of a simulation sometimes result in losing accuracy in the result by, for example, not calculating the force between each particle because it's negligible. Because of this it is inevitable to trade off between the time required and size of the error to the reality.

Since computers are discrete systems, there is always a certain difference to the actual physical event. This is because we have to calculate the simulation in time steps. Certainly one can shorten the time interval between the steps in order to get a more precise end result, but this automatically ends up in a longer calculation duration. Therefore, if it is possible to shorten the time that a calculation step takes, more steps can be simulated, which means smaller time intervals between steps, without affecting the total calculation time. That is why the computing duration for one iteration is an important indicator for the efficiency of the algorithm [7].

2.2. Particle Interactions

A very common way to predict the force between molecules in Molecular Dynamics is the Lennard-Jones potential, which includes van der Waals forces and Pauli repulsion. To get the most realistic result, it would be necessary to calculate the interaction for all Pairs of molecules. This would be a number of $(n-1)*n$ with a run time complexity of $\mathcal{O}(n^2)$. Using this procedure, the force approaches zero beyond a certain distance. To reduce the computing steps without increasing the error significantly, it is useful to introduce a cutoff horizon for each molecule to distinguish the particles for which the force to is relevant and where the interaction is negligible. Therefore, the simulation only needs to calculate $n*x$ times (x is a particular amount of atoms depending on the cutoff range and various parameters, but not on the total number of molecules) the force per time step, resulting in a complexity of $\mathcal{O}(n)$ [4].

2.3. Containers

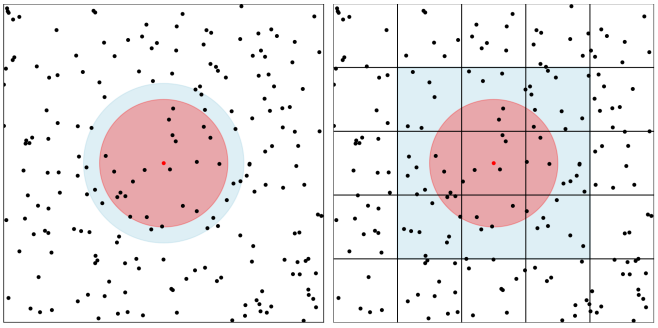

Figure 2.1.: Illustration of Verlet Lists (left) and Linked Cells (right) [8]

The most obvious approach to figure out which particles are inside the cutoff is to determine the distance to all other molecules at the beginning of each iteration. This is called Direct Sum and introduces the same issue of computation complexity as the force calculation. However, there are some ways to address this problem.

Linked Cells is a method where the simulation environment is divided into a grid based on which the particles are stored. Usually, the length of the cells is as big as the radius of the cutoff. Consequently, in order to calculate the neighbors of a molecule, only the particles in the same and in the neighboring cells need to be considered. This reduces the complexity of the distance calculations to $\mathcal{O}(n)$. It is possible to improve the performance by changing the size of the cell length or generating asymmetrical boxes in order to have a similar amount of atoms in all of them.

The focus in this thesis is on the third option called Verlet Lists (VL). This type of container determines the relevant neighbors by calculating the distance to all other particles, but instead of every time step, in a certain interval, named Rebuild Frequency (RF).

In this work with a RF of 12, is actually a frequency of $\frac{1}{12}$ or an interval of 12 steps between the reconstruction of the neighbor lists. Therefore a RF of 12 is higher than a RF of 24.

In order not to lose the accuracy of the simulation by overlooking atoms that are inside the cutoff, the radius of the horizon is extended by the so-called Verlet Skin. This skin must be big enough to prevent any molecule from entering the cutoff from the outside during the time between the rebuild steps. For this reason, a lower RF leads to a bigger skin length, which means more unnecessary force calculations per time step, but the advantage is that fewer neighbor lists rebuilds are performed on average. This results in a direct dependence between the RF and the verlet skin length. Whilst the complexity of the force updates between rebuilds is $\mathcal{O}(n)$, the complexity of the neighbor lists rebuilds is still $\mathcal{O}(n^2)$.

To solve this problem, a combination of Linked Cells and VL can be used. In this case, the particles are stored in cells used by VL only for neighbor lists rebuilds. This method is referred to as Verlet Lists Cells (VLC). Since it is necessary to take care of two types of containers simultaneously, the base load is higher. The advantage becomes apparent when many molecules are included in the simulation that are not crowded together [9] [10].

2.4. Traversals

To reduce the computation time, it is possible to use parallel threads for the distance calculation. To prevent race conditions when using shared memory, the threads have to be organized. For this purpose, traversals are introduced dividing the particles into groups that are assigned to the threads [4].

3. AutoPas

3.1. Motivation

It is possible to run an non-optimized molecule simulation and get accurate results. The disadvantage is that this could take a very long time and in most cases there is not unlimited time available. For this reason, it is important to increase the efficiency of molecular dynamics. The difficulty is to know what parameters are important and in which combination they are most effective. Therefore, it is necessary to be an expert in particle simulations and to have a lot of experience. In this case, there is another problem that is difficult to solve by hand. Most simulations have very varying states during the runtime, like a drop of water falling into a pool. In a short time there is a lot of movement, but not much happens before and after. Probably the optimal parameters would be different. That is why it makes sense to change the parameters during the simulation.

3.2. Functionality

Both issues are addressed by AutoPas by auto-tuning over specified parameters. This means that it is not necessary to be an expert, and the software is able to adapt the parameters to the environment before and during the simulation. To do this, the program iterates over different combinations of configurations selected by the tuning strategy. After a certain number of steps, the one with the fastest average computation time is used to proceed. This tuning phase is repeated after a certain number of iterations. To provide a simple but flexible experience for each user of the library, AutoPas works as a black box with an interface where short range forces can be set and particles can be accessed [4].

It is possible to create a particle and functor class to provide custom forces for the interactions or use the implemented ones. To ensure compatibility, the interface must be the same as in the base class of AutoPas. There are different strategies for choosing which configurations that are tested in the tuning phase. The simplest one is "full-search", where all possible combinations of parameters are tried [4].

4. Verlet Rebuild Frequency

4.1. Motivation

One of the containers integrated into AutoPas is Verlet Lists (VL), which calculates the distance to all particles to determine the relevant neighbors for the computation. A significant value is the RF, because it regulates the amount of time steps between each distance calculation and thus the size of the skin which determines how many molecules are in the extended cutoff. It can be assumed that the optimal number for this parameter is not the same in every simulation. As a consequence, it could be that the best value for the RF changes during runtime. In either case, a variable number for the RF in AutoPas would be useful to provide a efficient configuration for the particle simulation.

4.2. Approach

The first and most important thing to to determine is, if the optimal RF changes in different scenarios. This can be achieved by testing various possible parameters in some extreme environments.

If there are some decent results, the next step is to examine the simulations in more detail and see how the runtimes of the different configurations relate to each other. Finally, the most valuable information to investigate is the comparison between the total duration of computations with fixed and variable RF. To perform and compare different tests, three main simulation environments with different parameters and characteristics are used. Two of the them are standard experiments previously implemented for testing. The last one is created within this work to provide informative data. All yaml-files that were used as input for the different scenarios are listed in section A.3.

The first, Exploding Liquid (A.3.1), describes a cube of liquid represented by Lennard-Jones Molecules in a cuboid box that forms the framework of the simulation as shown in figure 4.1. During the experiment, the liquid splits into two equally sized parts that move away from each other. There is no external force acting, and the boundaries are periodic, what means molecules escaping from the border join on the opposite site of the box. With around 1900 particles this simulation is very small.

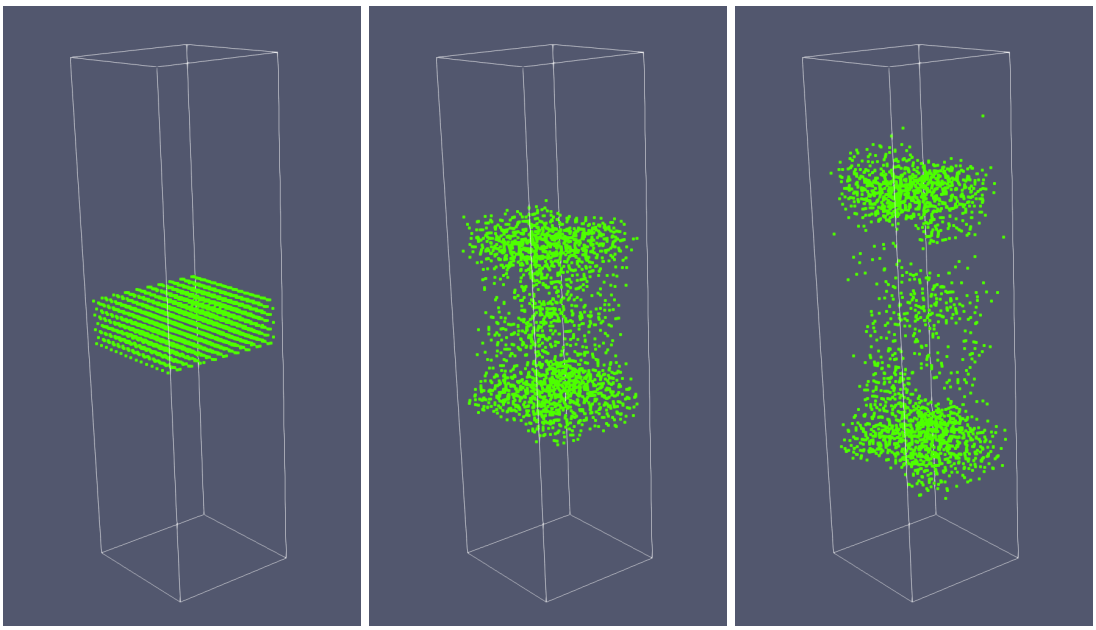

Figure 4.1.: Visualization of Exploding Liquid from left to right at the time steps 0, 2500 and 5000

The second scenario used to study the differences between different RF is called Falling Drop (A.3.2). It describes, as the name suggest, a drop falling into a pool of water. It starts with a sphere of liquid at the top and a cube at the bottom (figure 4.2, 4.3). An external force directed downwards simulates gravity. The boundaries are all reflective. This environment contains over 15000 molecules, some of which bump into the others. Therefore, it is very different to "exploding Liquid".

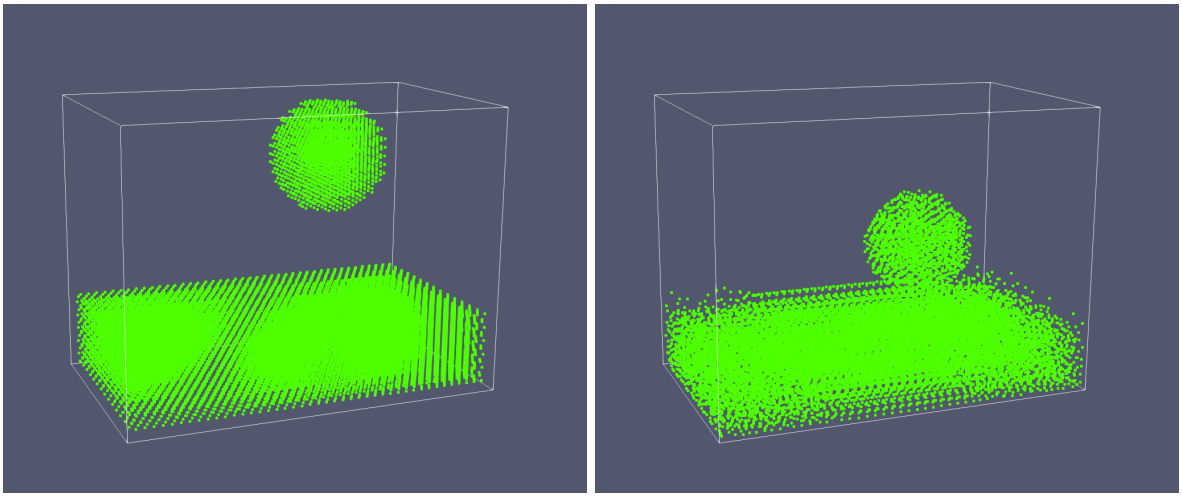

Figure 4.2.: Visualization of Falling Drop at the time steps 0 (left) and 2000 (right)

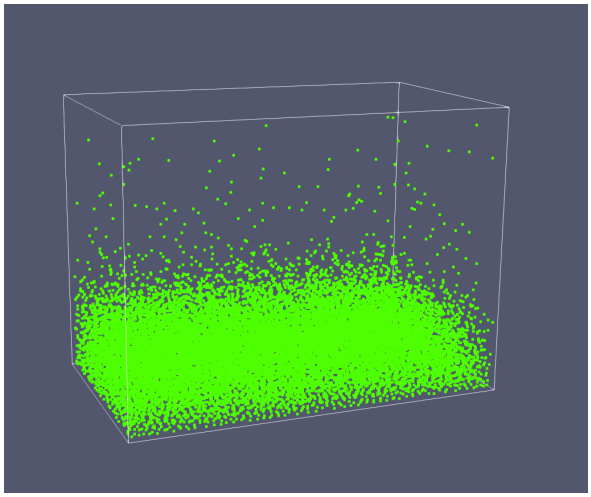

Figure 4.3.: Visualization of Falling Drop at the time step 8000

The last experiment Crashing Cubes (A.3.3), visualized in figure 4.4 and 4.5, is specifically designed to test the run time of variable RF. To achieve good results, it is necessary to have different states where the same RF is not always the best. To reach this goal, the scenario uses two cubes of particles moving towards each other at an initial velocity. The boundaries

are very wide making space to allow the molecules to spread out after the collision. This simulation is similar to Exploding Liquid, but contains much more particles with about 8400.

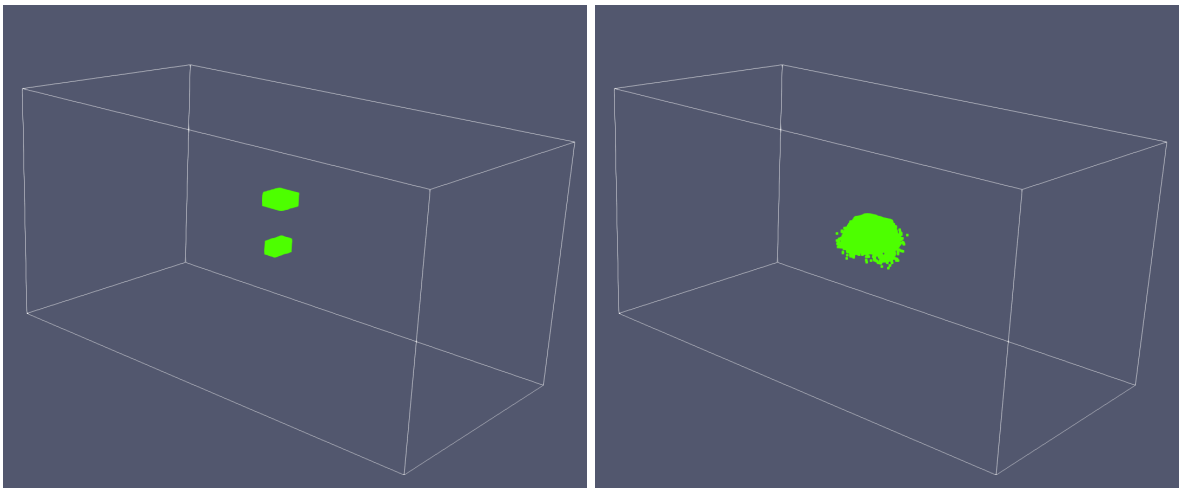

Figure 4.4.: Visualization of Crashing Cubes at the time steps 0 (left) and 672 (right)

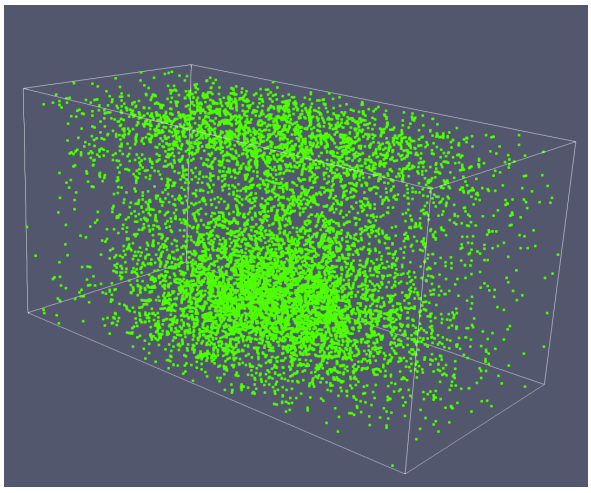

Figure 4.5.: Visualization of Crashing Cubes at the time step 4992

In order to compare the computation times between different runs of one scenario, it is useful to get the calculation times throughout the course of the simulation, without comparing the times of every single step. For this purpose it is reasonable to calculate the average of the runtimes in a certain interval. It is important to choose a multiple of the RF to prevent the calculated computing times from varying. Otherwise it can happen that there is one more rebuild step in some intervals than in others. Additionally it is most convenient to have the same interval, when comparing runs, which used different RF. For this reason, the data shown is always the average of 96 individual time steps. This number depends on the used frequencies 3, 6, 12, 24 and 48. When testing and analyzing the performance, there is

always some error from measuring and because the runtimes depend on what the computer is doing simultaneously. Therefore, all tests were run three times and the mean value of those is presented for comparison.

5. Results

5.1. Small Experiments

All tests were performed on a personal computer, whose detailed specifications can be found in section A.2.

As already mentioned, the first runs are used to determine if the same RF is not always the fastest in different environments. Just for this case, two very simple but extreme scenarios are used. The first one describes under 400 molecules formed in a cube at the beginning. The runtimes for different RF are shown in figure 5.1. The second simulation is very similar at the start, but contains only 210 particles, has a larger time interval between steps, and much wider borders. The runtime result of this scenarios are visualized in figure 5.2. On the left side the first one and on the right the second simulation.

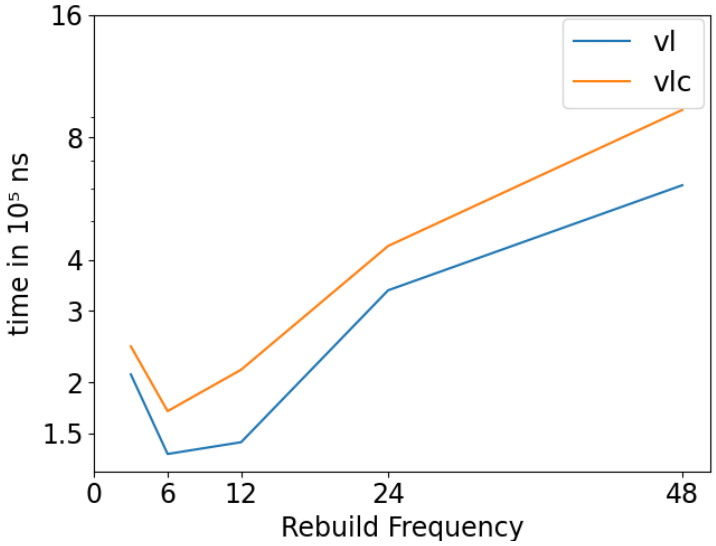

Figure 5.1.: Average runtimes of a simple experiment with under 400 molecules showing that the optimal RF is around 10

In both calculations, the configuration with VL is faster than with VLC. That is not surprising at all, considering that the cells gain an advantage in calculating the distance between the molecules.

As expected, different environments lead to different optimal RF. In the first graph, the minimum is around 10 and in the second the fastest option is around 48. Furthermore, it is noticeable that the computing time per step is overall higher in the left example than in the

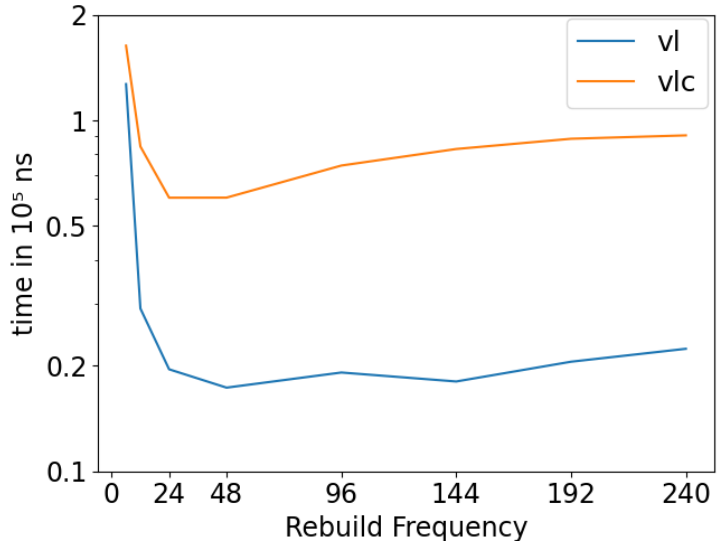

Figure 5.2.: Average runtimes of a simple experiment with 210 molecules showing that the optimal RF is over 24

right one. Not much more information can be derived from this since these scenarios are very small and far from practical use.

5.2. Exploding Liquid

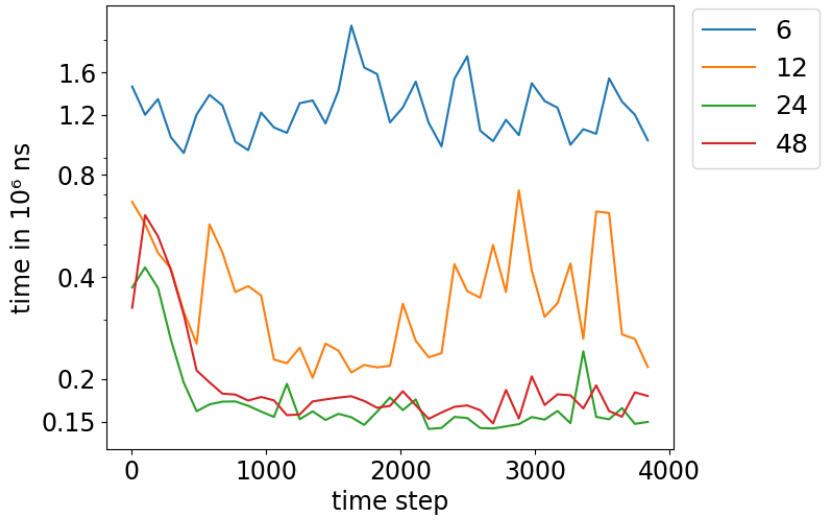

Figure 5.3.: Runtimes of Exploding Liquid with VL

A much more common simulation Exploding Liquid, described in section 4.2, is also light weighted with around 1900 molecules. It might be very interesting to see if the gap in the

required distance calculations between the container with cells and without is large enough to gain an advantage. The run times of VL in this scenario are shown in the graph of figure 5.3.

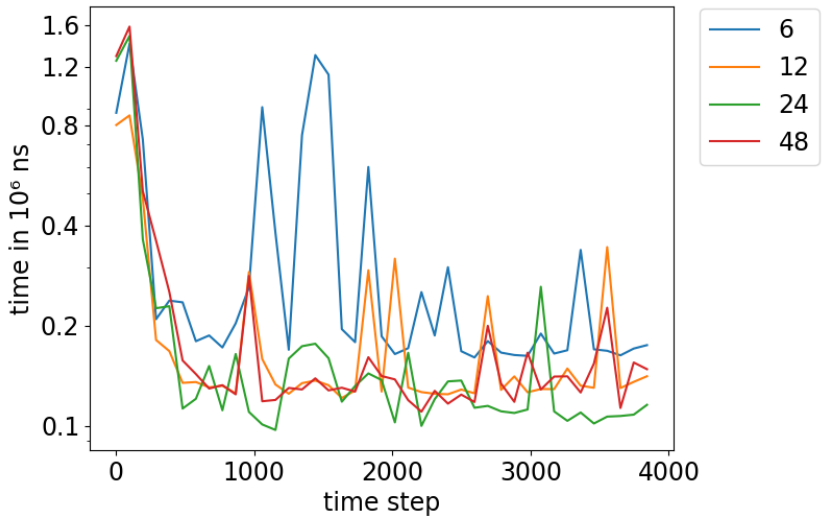

Figure 5.4.: Runtimes of Exploding Liquid with VLC

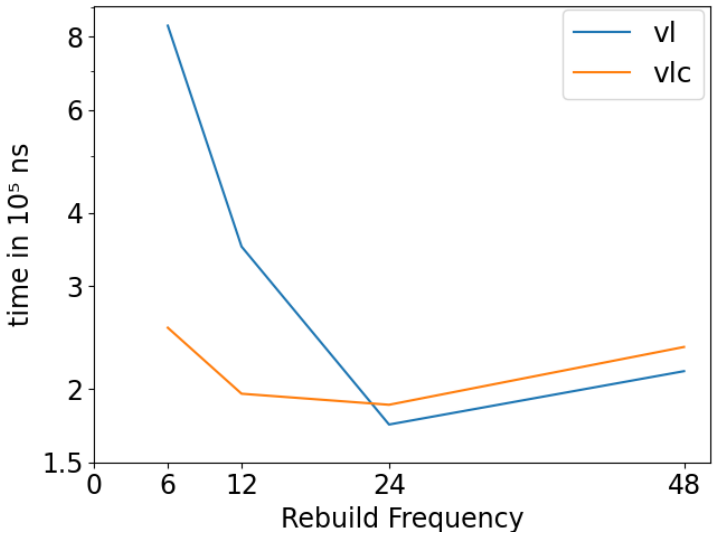

Figure 5.5.: Computing times for different RF in Exploding Liquid

It is recognizable, that the higher frequencies show a greater variance in simulation time required per step. From start to finish the calculation time hardly changes for a RF of 6. The situation is quite different for a higher number like 48, where the factor from maximum to minimum is more than three. This behavior leads to a different ratio between the configuration with a RF of 12 and 48. In the first 500 time steps, the run with the lower interval is almost as fast or faster, but after that it loses significant time compared to the higher interval. Overall,

the computing time is better with a RF of 24 over the entire duration. Similar observations can be made when observing the same scenario with VLC as container (runtimes shown in figure 5.4). However, there are some quite interesting differences. First of all, the computation time before the 1000th time step is significantly higher than afterwards, regardless of the RF. Additionally the run times of all configurations are very similar.

Figure 5.5 shows the comparison of the average total runtime between the two containers used. The optimal computation time is still achieved with VL at a RF of 24. Either the number of particles is not high enough to take advantage of VLC or the molecules are too close together.

5.3. Falling Drop

The second experiment Falling Drop (4.2) is the right candidate to test this with a multiple of particles. Moreover, it is a good opportunity to see how the computing duration changes when the distance calculation is much more expensive. At the first look, figure 5.6 looks very similar to figure 5.3.

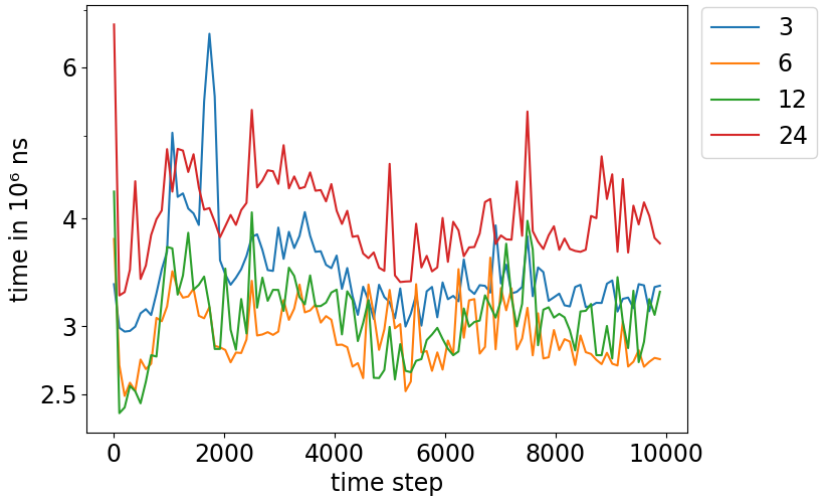

Figure 5.6.: Runtimes of Falling Drop with VL

The results are indeed very different, as the average times per step are much higher and the fastest setups have a far higher RF. Similarly, the lines do not intersect as often or are almost the same as in 6 and 12. The same behavior of the run times per step can be observed while using VLC, as shown in the graph of figure 5.7. In this simulation, the configuration with 6 as RF performs better than the one with 12. In general, it can be seen that higher RF perform better when using VLC. This makes perfect sense considering that the distance calculations are cheaper and therefore it is more efficient to do them more often. Comparing the average run times of the overall simulations with VL and VLC shown in figure 5.8, there is a different optimal configuration compared to the Exploding Liquid experiment.

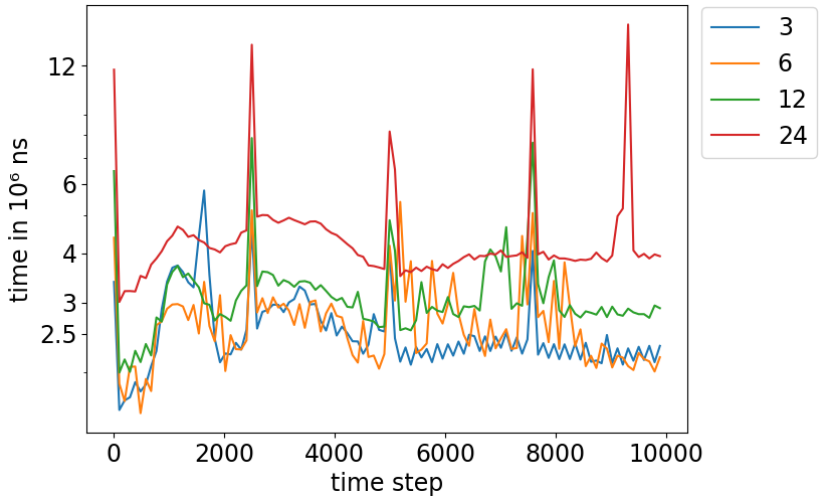

Figure 5.7.: Runtimes of Falling Drop with VLC

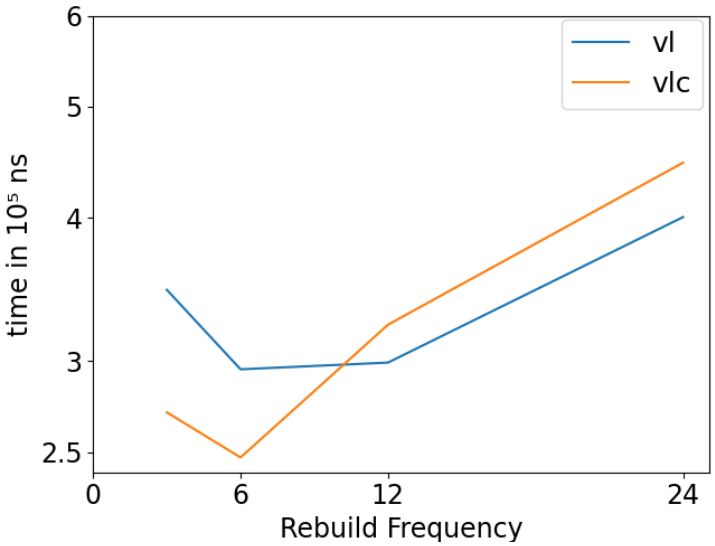

Figure 5.8.: Computing times for different RF in Falling Drop

5.4. Crashing Cubes

To investigate the behavior of the runtimes using different RF while the properties of the environment change, the last scenario Crashing Cubes is introduced. At first, when the two cubes collide, the simulation looks very similar to Falling Drop. But after around 1000 time steps the particles spread into a wide open space. The density of the molecules at the end is reminding of Exploding Liquid. This transformation in the simulation can also be observed in the computation times of all test shown in figure 5.9, 5.10, 5.11 and 5.12.

In this experiment all test were run with four different setups to show more detailed information. The first one is making use of VL that can only be combined with one specific

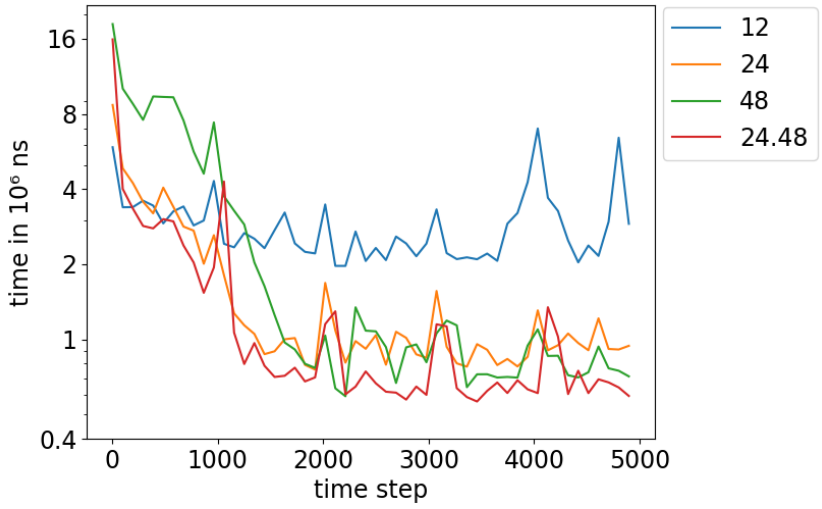

Figure 5.9.: Runtimes of Crashing Cubes using VL

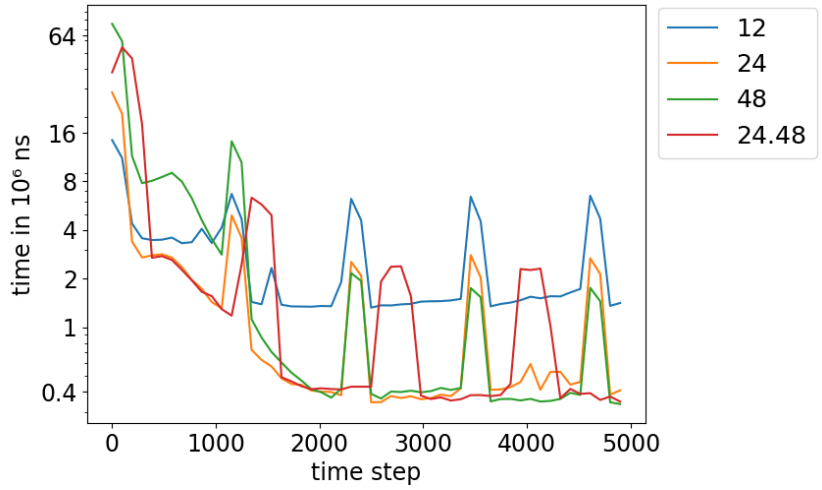

Figure 5.10.: Runtimes of Crashing Cubes using VLC with both traversals

traversal. For the other three configurations VLC is used as container, but with different available traversals. Two of them have a single traversal given. One of them uses the traversal c01 (non-sliced) and the other one the traversal sliced. The last setup tested, uses VLC in combination with both traversals to tune over. Therefore the tuning duration of the last one is longer compared to the others. The traversals included in the configurations with VLC were selected because of their very different approaches.

In all plots it can be seen that higher RF perform better at the beginning when the molecules are close together, and lower RF have an advantage when the distance between the instances increases. Almost throughout the test 24 is the fastest options between all single RF.

It is interesting to see how the computation times change when automatic tuning is used

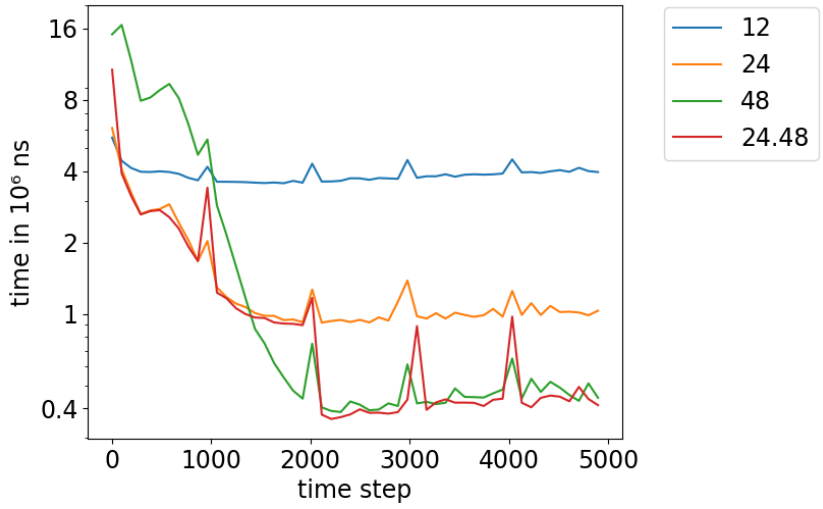

Figure 5.11.: Runtimes of Crashing Cubes using VLC with the traversal c01

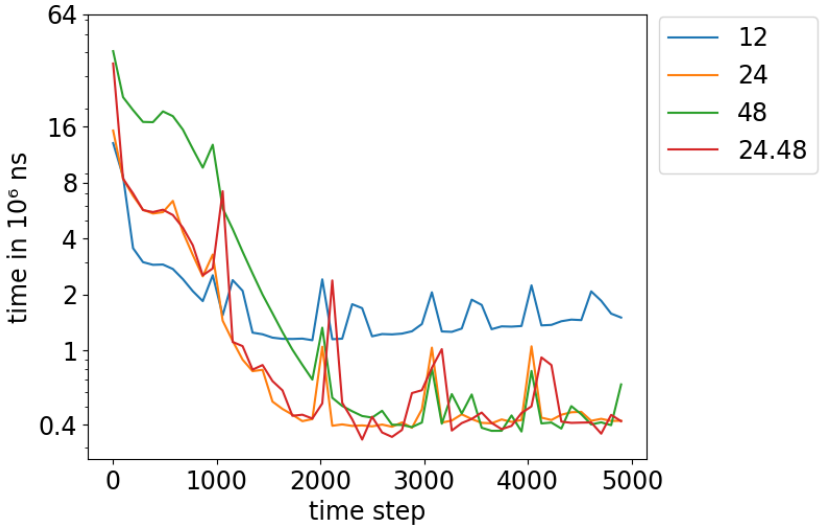

Figure 5.12.: Runtimes of Crashing Cubes using VLC with the traversal sliced

with more than one RF. This is presented in the red line in the graphs of figure 5.9, 5.10, 5.11 and 5.12. As expected, the red line, referring to the running times when tuning over 24 and 48 as RF, is very close to the one representing the times of a RF of 24 before the 1000th time step and the curve of 48 afterwards. How other combinations of RF look like compared is shown by figures in the appendix in section A.1.

The difference when using multiple RF instead of one is the time needed for tuning. With two allowed RF, the possible combinations of parameters are twice as large as if there is a static RF. That is the reason why the peaks in the red graph, which indicate the more elaborate tuning process, have a higher amplitude than the other curves.

In order to discover how much time can be saved by using autotune with more than one RF, all average total times are visualized in the figure 5.13 and listed in the table 5.1.

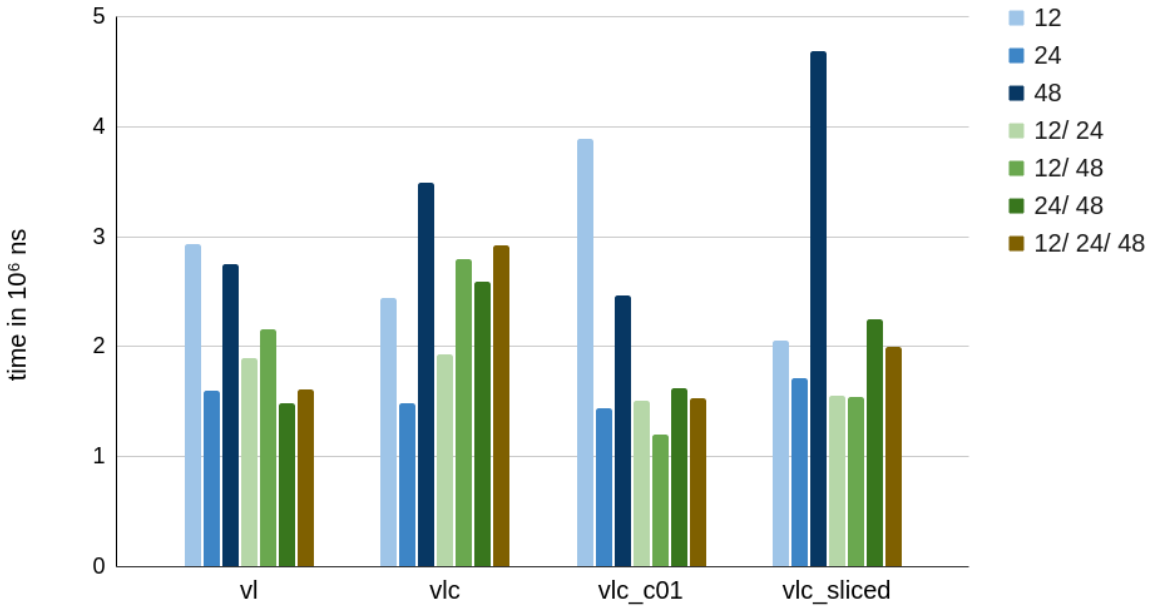

Figure 5.13.: Diagram of average runtimes of Crashing Cubes rounded in ns

RF	vl	vlc	vlc_c01	vlc_sliced
12	2932065	2448137	3890407	2056332
24	1598248	1489097	1436941	1714213
48	2751219	3488169	2462518	4684234
12/ 24	1891668	1927884	1509703	1554015
12/ 48	2156875	2798193	1199425	1547505
24/ 48	1485011	2591443	1621774	2248894
12/ 24/ 48	1610562	2918166	1531809	1997252

Table 5.1.: Table of average runtimes of Crashing Cubes rounded in ns

In general there is one RF for each simulation, that performs very good. Like, for example, 24 for the last experiment. If the tuning process is very light weighted because there are not many possible configurations like with VL or VLC with a fixed traversal, then tuning can be nearly as or more efficient than just running the best one. With a longer tuning period, as is the case when using VLC with two available traversals, the advantage of an optimal setup is not big enough.

However, usually the best RF is not known. Therefore, it is more useful to compare the non-optimal runtimes. From this viewpoint, two different patterns can be identified. For all

configurations, except of VLC using both traversals, the worst runtimes with more than one RF are better than the results with 12 or 48 (the non-optimal single RF). The only exception for this behavior can be observed when comparing using 12 to the pair of 24 and 48 when using the sliced traversal.

When using VLC with two traversals the relation of the times with tuned RF and static RF is different. The advantage of using multiple RF is not as clear as in the other configurations. Furthermore by using all three RF as option for tuning the runtime is longer than the in the worst-case of using two RF. The reason is the tuning process that needs more time for trying all possible combinations. Therefore the time necessary to find the best parameters is not worth the time gained with the tuned setup.

5.5. Interpretation

As mentioned before, one particular RF for a scenario is usually very fast. Especially when setting many other possible parameters. As previously seen, which RF is optimal depends on the environment of the simulation. With some experience, simply guessing a good RF is likely to outperform AutoPas, which uses more than one RF. But depending on the experiment, it is not always possible to get a faster computation compared to AutoPas because the optimal RF is sometimes changing during the scenario. For evaluating software, the best-case runtime is not the only important criterion because it doesn't guarantee a good runtime in every case. For this reason the duration needed in the worst-case is much more interesting. Therefore, using AutoPas with two or three tunable RF is a much safer choice most of the times. In all simulations tested for this paper, the optimal RF was between 6 and 96, but most of the times around 12 and 24.

When using VL leading to only one possible configuration, tuning over multiple RF is very cost-effective and almost always a good choice. The same applies if there are not many permissible configurations for some reason. The opposite case, if the tuning process takes longer, is somewhat more difficult to assess. It may be useful to shorten the tuning time by reducing the tuning samples or selecting a longer tuning interval. In general, the use of at least two different RF is much more reliable in terms of worst-case scenarios.

6. Future work

There are some options to gain efficiency in molecular dynamics by using the result of this paper. So far, varying RF have only been tested with full-search. The question is how the performance changes using other tuning strategies. Predictive tuning could reduce the combinations that have to be tested during the simulation [11]. This could lead to better runtimes when using multiple RF and traversals at the same time. A tuning strategy based on reinforcement learning could discover patterns between the size of the RF and different traversals depending on the environment [12]. Both approaches could fix the issue full-search has with too many possible combinations.

As shown in chapter 5, the runtimes between the RF differ when using different traversals. In this work only two traversals were tested separately. To see more detailed, how other traversals are working with variable RF would be interesting and could help to choose the right setup for a scenario. Furthermore it could help to use traversals and variable RF in a reasonable amount to gain efficiency. It could be a opportunity to use two or three different traversals for tuning instead of all or just one, that is not considered in this work.

Furthermore, when using linked cells or VLC, another parameter Cell Size Factor is tunable. Changing the size of the cells can lead to less unnecessary force calculations, but to more cells to take care of. The open question is, how is this factor behaving relative to variable RF. Maybe there is the potential to improve performance by using multiple options for both parameters simultaneously.

Verlet Cluster Lists and Pairwise Verlet Lists are special forms of VL as VLC is. It should be discovered how the runtimes of these four methods with variable RF are relative to each other. Maybe some get a greater advantage of it than others. This knowledge could help to choose better configurations or even lead to an optimized new tuning strategy.

In general, it is not clear how the computation times are varying when tuning over different containers. This is important to discover because using multiple containers in addition to variable RF could lead to a long tuning phase making the program inefficient. Again, this should depend on the strategy used. Therefore, it is important to be aware of this.

7. Conclusion

The original assumption was correct. Different RF perform different depending on the environment. In addition, the fastest configuration may change during a simulation. Therefore, tuning over multiple RF during the simulation has the potential to provide an advantage. However, during the testing process it became clear that this approach also has disadvantages. If there are many parameters available for tuning, resulting in more possible configurations, increases in tuning duration negate the benefit gained by using the best combination. For this reason, it is important to keep the number of allowed parameters within limits. This can be achieved by limiting other options or by using fewer different RF. As shown, there is a risk to just use one random RF. Therefore, at least two different RF should be used. For all scenarios tested in this work, either 12 or 24 performs very good. Consequently, tuning over both of them leads to a reasonable runtime without stretching the tuning process too much. If the user knows there are not many other tuning options for the simulation, it may be worth to iterate over more different RF. General it is possible to use other numbers, but when tuning between different ones they shouldn't be too similar, such as for example 12 and 14. Especially for inexperienced users, the runtime can be improved considerably in the worst-case by using variable RF.

A. Appendix

A.1. Additional Figures

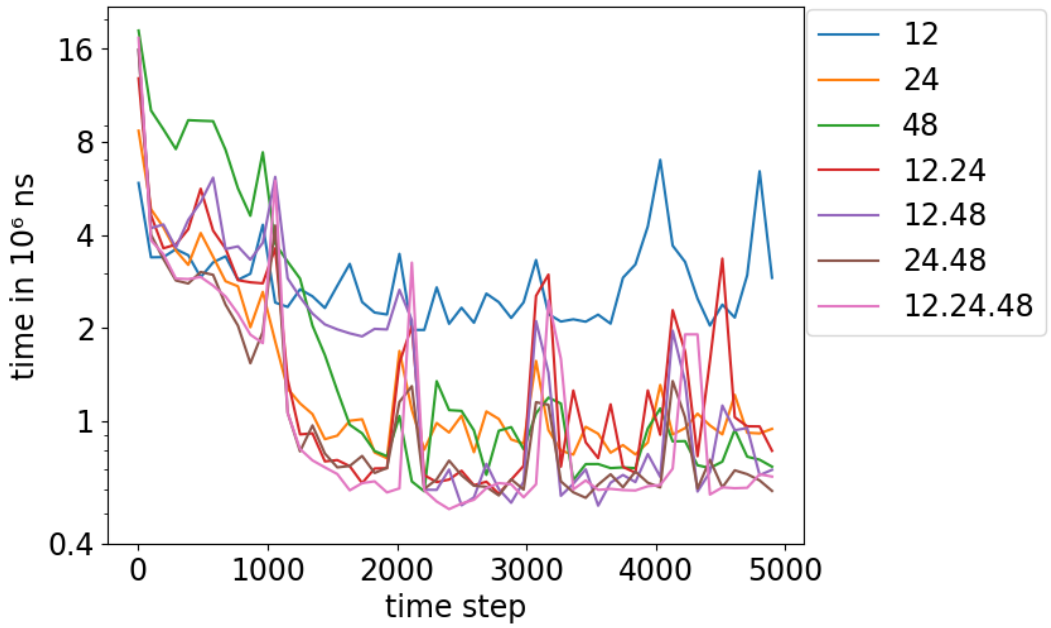

Figure A.1.: Runtimes of Crashing Cubes using VL

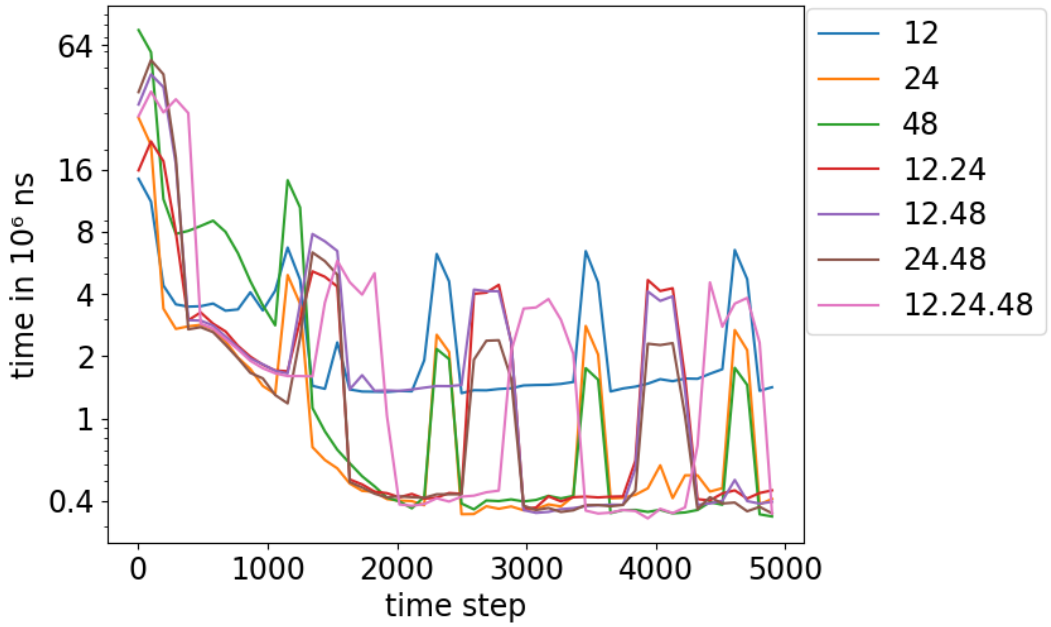

Figure A.2.: Runtimes of Crashing Cubes using VLC

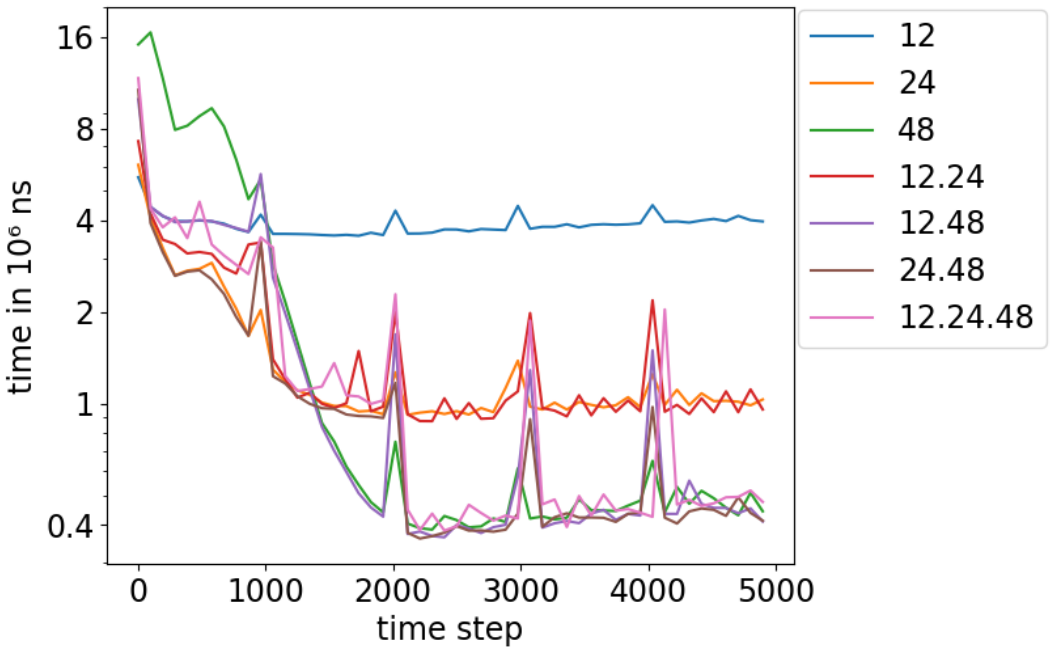

Figure A.3.: Runtimes of Crashing Cubes using VLC and the traversal c01

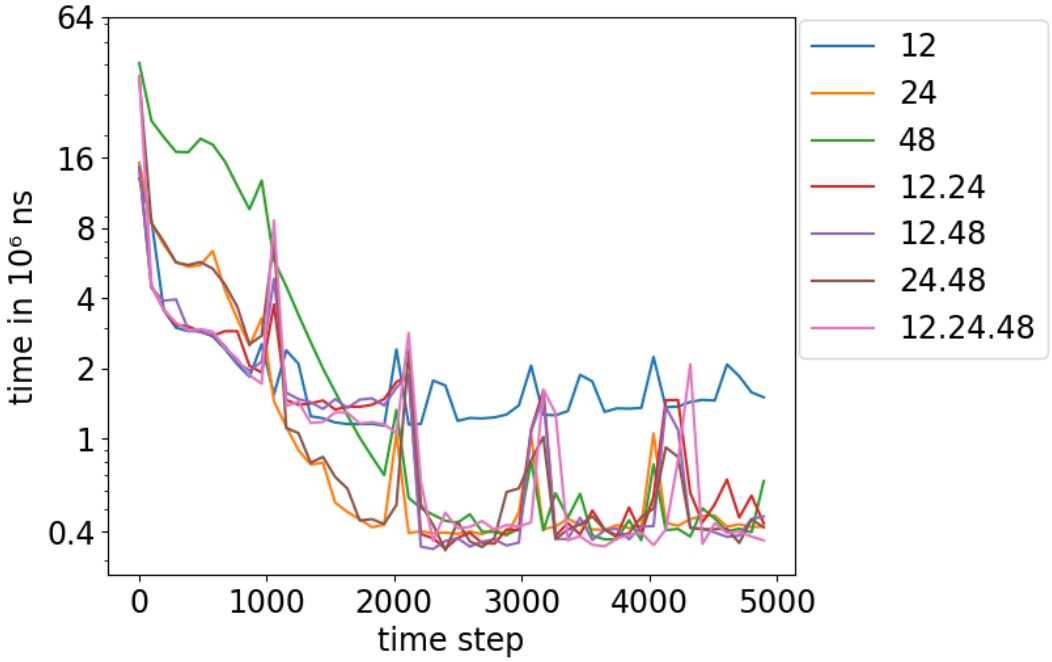

Figure A.4.: Runtimes of Crashing Cubes using VLC and the traversal sliced

A.2. Hardware Specifications

Processor: AMD Ryzen™ 7 5800X (8x 3.8GHz)
RAM: 32GB DDR4-3200
Memory: SSD
Operating System: Ubuntu 22.10

A.3. yaml-files

A.3.1. Exploding Liquid

```
verlet-skin-radius-per-timestep  :  0.02
verlet-cluster-size              :  4
selector-strategy                :  Fastest-Mean-Value
mpi-strategy                     :  no-mpi
tuning-strategy                  :  predictive-tuning
tuning-interval                  :  10000
tuning-samples                   :  6
functor                          :  Lennard-Jones-AVX2
cutoff                           :  2
box-min                          :  [0, 0, 0]
box-max                          :  [15, 60, 15]
cell-size                        :  [1]
deltaT                           :  0.00182367
iterations                       :  12000
periodic-boundaries              :  true
Objects:
  CubeClosestPacked:
    0:
      box-length                 :  [15, 6, 15]
      bottomLeftCorner           :  [0, 27, 0]
      particle-spacing           :  1.
      velocity                   :  [0, 0, 0]
      particle-type              :  0
      particle-epsilon           :  1
      particle-sigma             :  1
      particle-mass              :  1
no-flops                         :  false
no-end-config                    :  true
no-progress-bar                  :  false
vtk-filename                     :  explodingLiquid
vtk-write-frequency              :  100
```

A.3.2. Falling Drop

```
verlet-skin-radius-per-timestep  :  0.1
verlet-cluster-size              :  4
selector-strategy                :  Fastest-Absolute-Value
data-layout                      :  [AoS, SoA]
traversal                        :  [lc_c01, lc_c18, lc_c08, lc_sliced_c02,
                                        vl_list_iteration, vlc_c01, vlc_c18,
                                        vlc_sliced_c02, vcl_cluster_iteration,
                                        vcl_c01_balanced, vcl_c06]
tuning-strategy                  :  full-Search
tuning-interval                  :  2500
tuning-samples                   :  3
tuning-max-evidence              :  10
functor                          :  Lennard-Jones
newton3                          :  [disabled, enabled]
cutoff                           :  3
box-min                          :  [0, 0, 0]
box-max                          :  [7.25, 7.25, 7.25]
cell-size                        :  [1]
deltaT                           :  0.0005
iterations                       :  15000
boundary-type                    : [reflective,reflective,reflective]
globalForce                      :  [0,0,-12]
Objects:
  # "water"
  CubeClosestPacked:
    0:
      particle-spacing           :  1.122462048
      bottomLeftCorner           :  [1, 1, 1]
      box-length                 :  [48, 28, 10]
      velocity                   :  [0, 0, 0]
      particle-type              :  0
      particle-epsilon           :  1
      particle-sigma             :  1
      particle-mass              :  1
  Sphere:
    0:
      center                     :  [18, 15, 30]
      radius                     :  6
      particle-spacing           :  1.122462048
      velocity                   :  [0, 0, 0]
      particle-type              :  1
```

```
      particle-epsilon           :  1
      particle-sigma             :  1
      particle-mass              :  1
vtk-filename                     :  fallingDrop
vtk-write-frequency              :  1000
no-flops                         :  false
no-end-config                    :  true
log-level                        :  info
```

A.3.3. Crashing Cubes

```
verlet-skin-radius-per-timestep  :  0.2
verlet-cluster-size              :  3
selector-strategy                :  Fastest-Absolute-Value
tuning-strategy                  :  full-search
tuning-interval                  :  1000
tuning-samples                   :  6
functor                          :  Lennard-Jones-AVX2
cutoff                           :  2
box-min                          :  [0, 0, 0]
box-max                          :  [400, 200, 200]
cell-size                        :  [1, 0.5]
deltaT                           :  0.002
iterations                       :  5000
boundary-type                    : [periodic,periodic,periodic]
Objects:
  CubeClosestPacked:
    0:
      box-length                 :  [10, 10, 20]
      bottomLeftCorner           :  [195, 85, 90]
      particle-spacing           :  1.1
      velocity                   :  [0, 10, 0]
      particle-type              :  0
      particle-epsilon           :  1
      particle-sigma             :  1.1
      particle-mass              :  1
    1:
      box-length                  : [20, 10, 20]
      bottomLeftCorner            : [190, 125, 90]
      particle-spacing            : 1
      velocity                    : [ 0, -20, 0 ]
      particle-type               : 1
      particle-epsilon            : 1
      particle-sigma              : 1
      particle-mass               : 1
no-flops                         :  false
no-end-config                    :  true
no-progress-bar                  :  false
vtk-filename                     :  varibale_verlet
```

Bibliography

[1] *How Does An F1 Wind Tunnel Work?* URL: https://f1chronicle.com/how-does-an-f1-wind-tunnel-work-f1-technology/.

[2] M. E. Tuckerman* and G. J. Martyna. *Understanding Modern Molecular Dynamics: Techniques and Applications*. URL: https://pubs.acs.org/doi/pdf/10.1021/jp992433y.

[3] A. Krokhotin and N. V. Dokholyan. "Chapter Three - Computational Methods Toward Accurate RNA Structure Prediction Using Coarse-Grained and All-Atom Models". In: *Computational Methods for Understanding Riboswitches*. Ed. by S.-J. Chen and D. H. Burke-Aguero. Vol. 553. Methods in Enzymology. Academic Press, 2015, pp. 65–89. DOI: https://doi.org/10.1016/bs.mie.2014.10.052. URL: https://www.sciencedirect.com/science/article/pii/S0076687914000536.

[4] F. A. Gratl, S. Seckler, N. Tchipev, H.-J. Bungartz, and P. Neumann. "AutoPas: Auto-Tuning for Particle Simulations". In: (2019). DOI: 10.1109/IPDPSW.2019.00125.

[5] J. Loschwitz, O. O. Olubiyi, J. S. Hub, B. Strodel, and C. S. Poojari. "Chapter Seven - Computer simulations of protein–membrane systems". In: *Computational Approaches for Understanding Dynamical Systems: Protein Folding and Assembly*. Ed. by B. Strodel and B. Barz. Vol. 170. Progress in Molecular Biology and Translational Science. Academic Press, 2020, pp. 273–403. DOI: https://doi.org/10.1016/bs.pmbts.2020.01.001. URL: https://www.sciencedirect.com/science/article/pii/S1877117320300120.

[6] G. S. Grest, B. Dünweg, and K. Kremer. "Vectorized link cell Fortran code for molecular dynamics simulations for a large number of particles". In: *Computer Physics Communications* 55.3 (1989), pp. 269–285. ISSN: 0010-4655. DOI: https://doi.org/10.1016/0010-4655(89)90125-2. URL: https://www.sciencedirect.com/science/article/pii/0010465589901252.

[7] A. Hospital, J. Goñi, M. Orozco, and J. Gelpi. "Molecular dynamics simulations: advances and applications". In: (2015). DOI: 10.2147/AABC.S70333.

[8] S. J. Newcome. *AutoPas: Optimising Multi-site Molecular Dynamics Simulations with Autotuning and Kokkos*. URL: https://mediatum.ub.tum.de/doc/1689705/document.pdf.

[9] M. Papula. "Implementing the Linked Cell Algorithm in AutoPas using References". In: (2020). URL: https://mediatum.ub.tum.de/doc/1576172/1576172.pdf.

[10] T. Vladimirova. "Implementation and Evaluation of Verlet List-based Methods in AutoPas". In: (2021). URL: https://mediatum.ub.tum.de/doc/1601717/1601717.pdf.

[11] P. J. Mark. "Implementing a predictive tuning strategy in AutoPas using extrapolation". en. MA thesis. Technical University of Munich, Sept. 2020.

[12] L. Leonhard. “Can Reinforcement Learning be used to improve the autotuning process within AutoPas?” en. MA thesis. Technical University of Munich, Sept. 2022.