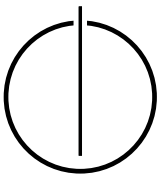
TUM SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE
TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Evaluating Data Structures in Multi-Site Molecular Dynamics

Johannes Riemenschneider



TUM SCHOOL OF COMPUTATION,
INFORMATION AND TECHNOLOGY

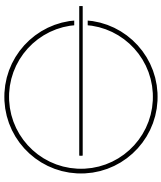
DEPARTMENT OF COMPUTER SCIENCE
TECHNICAL UNIVERSITY OF MUNICH

Bachelor's Thesis in Informatics

Evaluating Data Structures in Multi-Site Molecular Dynamics

Analyse von Datenstrukturen in Multi-Site Molekulardynamik

Author:	Johannes Riemenschneider
Supervisor:	Prof. Dr. Hans-Joachim Bungartz
Advisor:	Samuel James Newcome
Date:	February 23, 2024



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

Munich, February 23, 2024 Johannes Riemenschneider

Acknowledgements

This thesis would not have been possible without the support of many. First, I would like to thank my advisor, Sam, for the ideas and continuous feedback throughout the making of this thesis. Without you, this project would have never become what it is today.

Second, I am expressing my gratitude towards the Leibniz-Rechenzentrum for providing the computational resources required to run the necessary experiments, as well as the chair of Scientific Computing and Prof. Dr. Hans-Joachim Bungartz. Thank you for giving me the opportunity to work in this research area.

Finally, I would like to thank my family and friends for giving me the endurance and support I needed. In particular, I would like to thank Adrian, Lukas, and Sarah. Without you, my personal and academic journey would be significantly less successful but mostly a lot less fun.

Abstract

Molecular dynamics simulations have become crucial in various important research areas, such as thermodynamics, structural mechanics, and medicine. Since the effectiveness of such simulations in research is heavily dependent on their accuracy and fast availability, there is a continuous effort to optimize the algorithms involved in their making. Multi-site simulations form a subdiscipline within the molecular dynamics landscape, where every molecule gets treated as a rigid body of points. This thesis investigates the efficiency of different molecule representations and data structures that can be used to represent the molecules in such simulations. We show that the molecule representation chosen can significantly affect the runtimes of the molecular dynamics algorithms acting on it. The relation between molecule representation and molecular dynamics algorithm is captured by testing different combinations of the two in various simulation scenarios. We show´ that the introduction of new variables to reduce the computational workload of the torque evaluation can achieve a speedup of up to 1.4 depending on the scenario, whereas the explicit storage of site position actually decreases efficiency. Additionally, we discuss another approach, where every site gets treated as an independent entity during the force calculation process. We highlight that the efficiency of this representation strongly deviates in both directions from a more conventional approach depending on the data-layout and neighbor identification algorithm. We discuss this relation in greater detail.

Zusammenfassung

Moleukulardynamik-Simulationen sind inzwischen ein essenzieller Bestandteil verschiedener Forschungsgebiete, wie zum Beispiel der Thermodynamik, Strukturmechanik und der Medizin. Da die Effektivität dieser Simulationen stark von ihrer Genauigkeit und schnellen Verfügbarkeit abhängt, besteht ein hoher Anreiz die dafür benötigten Algorithmen stetig zu verbessern. Multi-Site Simulationen bilden eine Subdisziplin innerhalb der Molekulardynamik, in der Moleküle als Festkörper von Interaktionspunkten angesehen werden. Wir analysieren die Effizienz verschiedener Datenstrukturen und Molekülrepresentationen, welche in solchen Simulationen verwendet werden können. Wir zeigen, dass die gewählte Darstellung die Laufzeit signifikant beeinträchtigt. Das Verhältnis zwischen der gewählten Molekülrepräsentation und dem verwendeten Algorithmus wird durch die Verwendung verschiedener Kombinationen auf unterschiedlichen Eingaben quantifiziert. Wir zeigen, dass die Einführung neuer Variablen, um den Rechenaufwand der Drehmomentberechnung zu minimieren einen Speedup von bis zu 1.4 je nach Simulations-Input erzielen kann, während die explizite Speicherung der Site-Positionen die Laufzeit sogar verschlechtert. Zusätzlich führen wir einen neuen Anatz ein, in dem jede Site als eigenes Objekt während der Kraftberechnung behandelt wird. Wir zeigen, dass sich die Effizienz dieses Ansatzes in beide Richtungen stark von einem konventionelleren Ansatz je nach Data-Layout und Algorithmus unterscheidet. Dieses Verhältnis führen wir expliziter aus.

Contents

1. Introduction

With the increasing hardware capabilities of modern computers, particle simulations have become a crucial part of the modern research landscape in various application fields. For example, in the biomedical sector, they are deployed during multiple stages of the drug discovery process [1]. The same techniques can also be utilized in a different use case to gain valuable insights into the behavior of proteins on an atomic level for research in neuroscience [2]. However, they also find applications in more macroscopic research settings. The physical properties of aging asphalt under corroding environmental circumstances, such as water exposure and extreme heat, have also been explored using the same type of simulator [3]. On an even larger scale, smoothed particle simulations are being used to simulate the generation of entire galaxy formations [4].

This successful deployment of particle simulations throughout vastly different academic disciplines can mainly be attributed to the expansion in scale and accuracy of the possible simulations, along with the universal nature of the problem statement. In many research fields, the movement of bodies interacting with each other is highly important. For some of these use cases, simulating a molecule as a single interaction point is an oversimplification that leads to a distortion of measurements to an unacceptable level. In these cases, multi-site molecular dynamics can be used to represent molecules with a more accurate model at the cost of higher computational complexity. In addition to the increased computational cost, this approach introduces the new software-architectural challenge of finding a data structure to represent these molecules.

We highlight that there are conflicting requirements for the molecule representations. A representation with a small memory footprint is desirable, but the most compact representation introduces the need for a significant number of redundant floating point operations. Additionally, a poorly chosen representation can drastically decrease the ability to vectorize the multi-site molecule interactions in the implementation. We explore the effectiveness of several molecule representations in combination with different neighbor identification algorithms in the simulator md-flexible. md-flexible is a particle simulation program for short-range forces that utilizes the node-level auto-tuning library AutoPas [5].

2. Theoretical Background

In this chapter, we explain the theoretical framework required in multi-site MD by discussing the numerical algorithms, assumptions, and models utilized. We also introduce the simulator md-flexible and explain its relation to the library AutoPas.

2.1. Introduction to Molecular Dynamics

Simulating the movement of particles that are continuously exerting forces onto each other is the primary objective of molecular dynamics (MD). A specific instance of this problem is given by defining the particles with their mass, starting position, velocity at the beginning of the simulation, and other particle-specific parameters. Since these particles exert forces on each other, they accelerate according to Newton's second law of motion. Acceleration is the second derivative of the position function. Since the acceleration at any point in time depends on every particle position, this problem can be expressed as a system of differential equations.

If it were possible to solve these equations, the result would be a perfect description of the particle movements. Unfortunately, this problem cannot be solved analytically for any force function if three or more particles are involved. Therefore, we use numerical approximations instead. In order to determine the derivation of a function, the slope of an infinitesimally small slope triangle is evaluated. The fundamental assumption of molecular dynamics is the idea that this slope can be approximated sufficiently well by using a small enough Δt instead of letting the size of this time interval go to zero.

According to this idea, we slice the time axis into discrete time intervals. Then, the simulation state at each time step is computed iteratively, as shown in Figure 2.1. First, we update the position of each particle based on its current velocity. Then, we compute the force acting on each particle by determining the force between every particle pair. After that, we update each particle's velocity according to the forces they experience. This process is repeated until we reach the desired number of simulation steps.

2.2. The Lennard-Jones Potential

The approach of molecular dynamics is not tied to any specific kind of pairwise force. For example, when using gravitational force, it would be possible to simulate the planetary movement in a solar system. In molecular dynamics, the most commonly used force is the Lennard-Jones potential. It describes the van der Waals forces and Pauli repulsion between two uncharged molecules [6]. The potential U_{ij} between the particles i and j can be determined via the formula

$$U_{ij} = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r}\right)^{12} - \left(\frac{\sigma_{ij}}{r}\right)^{6}\right], \tag{2.1}$$

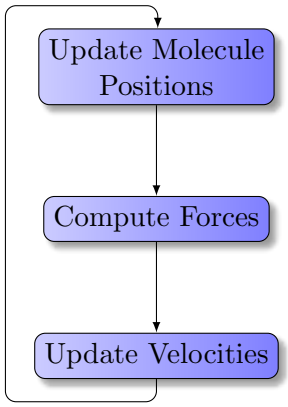


Figure 2.1.: Basic control flow of particle simulations

where r is the particle distance, σ_{ij}, and ϵ_{ij} are constants that only depend on the particle types interacting. The total force any particle i experiences can be obtained by summing up all the pairwise forces between this particle and the other particles in that simulation:

$$\vec{F}_i = \sum_{i \neq j} \vec{F}_{ij} \tag{2.2}$$

When the two particles are closely together, the minuend of the Lennard-Jones potential becomes more significant than the subtrahend, meaning that the particles are experiencing a repulsive force away from each other. For sufficiently large r, it is the subtrahend that is larger, meaning that the particles attract each other. Note that for large radii r, the absolute value of this force decreases with $\mathcal{O}(r^{-6})$.

2.3. Multi-Site Molecular Dynamics

The easiest way to model molecules in the simulation is by representing them as a single point in space. This approach is accurate enough to give meaningful results in many application fields. However, there are other use cases where this simplification distorts the result too drastically. Therefore, multi-site MD is trying to capture the molecule structure with higher accuracy. Instead of treating molecules as a single point, they are represented as a rigid body of points that are connected with each other. These molecule components are called *sites*. These sites do not have to correspond to actual atoms inside the molecule. For example, the accuracy of simulations involving water molecules can be greatly improved by introducing an imaginary fourth site located between the site representing the oxygen atom and the two sites corresponding to the hydrogen atoms [7]. This water-molecule representation is called the TIP4P model.

In order to compute the force between two molecules i and j, all the sites in i are interacting with all sites in j. We call the computation of forces two sites exert on each other a *site-to-site interaction*. The number of site-to-site interactions and, therefore, the number of Lennard-Jones computations required for a single *molecule-to-molecule interaction* increases quadratically with respect to the number of sites in a molecule. A molecule-to-molecule interaction describes the sum of all site-to-site interactions between these two molecules.

Additionally, sites have a position $\vec{d}$ relative to their molecule's center of mass (CoM). This means that any force acting on a site can exert a torque on the molecule, causing it to rotate. The total torque a molecule experiences at any point in time is given by the formula

$$\tau = \sum_{i \in S} \sum_{j \in M \setminus S} \vec{d_i} \times \vec{F_{ij}}. \tag{2.3}$$

S is the set of all sites in that molecule, $\vec{d_i}$ is the position of site i relative to the center of mass and M is the set of all sites in the simulation.

Analogously to acceleration and position, torque is the second derivative of the molecule's rotational orientation at any point in time. Therefore, the rotational movement can be approximated numerically using techniques that are analogous to the methods used to approximate the translational molecule behavior [8].

2.4. Eager and Lazy Torque Evaluation

The torque a molecule experiences is the sum of torques caused by site-to-site interactions. Since the cross-product is distributive, this term can be simplified:

$$\tau = \sum_{i \in S} \sum_{j \in M \setminus S} \vec{d_i} \times \vec{F_{ij}} \tag{2.4}$$

$$= \sum_{i \in S} \vec{d_i} \times \sum_{j \in M \setminus S} \vec{F_{ij}} \tag{2.5}$$

$$= \sum_{i \in S} \vec{d_i} \times \vec{F_i}. \tag{2.6}$$

A computer program using formula 2.4 updates the torque of a molecule any time a new pairwise force has been computed. This approach will be referred to as *eager torque evaluation* since the torque of a molecule is kept up to date at every point in time. When using the formula 2.6, the program computes the total force acting on every site first. After that, the torque is computed by summing up the torques caused by each individual site. This approach will be referred to as *lazy torque evaluation* since the torque calculation gets stalled until the end of the force calculation. Considering that the eager approach will perform a cross-product computation for every pairwise force while the lazy calculation only does so once for every site, the second approach reduces the workload quite drastically. The main drawback of lazy torque evaluation is that we need a variable for each site i to keep track of the force $\vec{F_i}$ that this site experiences. Eager torque evaluation does not require site-specific variables.

2.5. The Cutoff-radius

In order to compute the force acting on a molecule, it is theoretically necessary to compute the force between all pairs of particles. Since the number of pairs increases quadratically with the number of particles n, this approach would result in $\mathcal{O}(n^2)$ force calculations. However,

two particles with a large enough distance r between each other exert almost no force on each other. Since the magnitude of the Lennard-Jones potential decreases with $\mathcal{O}(r^{-6})$, it becomes negligibly small at a certain threshold distance. Therefore, not computing the force between these particles will not decrease the quality of the simulation significantly. Hence, a cutoff-radius r_{cutoff} is defined. Forces between particles that are further away than this distance do not get computed. This concept can only be applied as long as the force magnitude decreases quickly enough. If another force were used, where the magnitude decreases more slowly (for example, the gravitational force), this concept could not be used. Forces that diminish quickly enough are known as short-range forces.

When the molecules are spread out uniformly enough in space, the cutoff condition will reduce the number of required force calculations to $\mathcal{O}(n)$. Of course, a maximally inhomogeneous input, placing every molecule within the cutoff-radius of each other, would still result in $\mathcal{O}(n^2)$ force calculations. While it is true that less homogeneous scenarios are computationally more expensive, the effect of this phenomenon is bound by the fact that the Lennard-Jones potential increases with $\mathcal{O}(r^{12})$ for small distances. Molecules repel each other so violently when they get closer together that the accuracy of the simulation deteriorates faster than the runtime increases. Therefore, the density of a particle simulation cannot be increased indefinitely, which limits the negative effects inhomogeneity can have.

Unfortunately, filtering out the force computations that actually have to be done is no trivial task. The most naive implementation would be iterating over all pairs of molecules to check whether the force has to be computed between them. This approach is known as *Direct Sum*. Even though this successfully reduces the number of force calculations to $\mathcal{O}(n)$, the number of distance checks required remains to be in $\mathcal{O}(n^2)$. Therefore, more efficient data structures are required to limit the number of distance checks necessary.

2.6. Cutoff Conditions in Multi-Site Molecular Dynamics

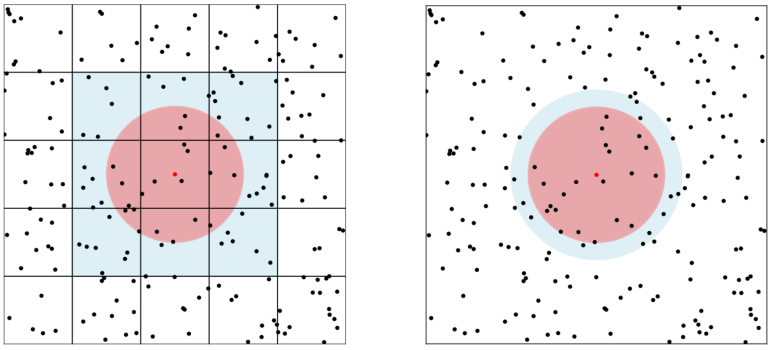
In the context of multi-site MD, there are multiple distances that can reasonably be used to determine the cutoff condition. One way this condition can be implemented is to use the spatial interval between the centers of masses. When this condition is used, all site-to-site interactions will be performed if the molecules are within the cutoff range. A partial molecule interaction is impossible. This approach is called the Center-of-Mass to Center-of-Mass (CtC) condition. Another approach is the Site-to-Site (StS) cutoff condition. As the name implies, the spatial interval between the site positions gets utilized rather than the distances between the centers of masses. When a molecule resides close to the cutoff-radius boundary, some sites can be within the cutoff range, while others remain barely outside. The application of the StS condition will result in the interaction with the sites that are barely within range while the others get ignored. This kind of partial interaction cannot occur with the CtC condition since it would determine whether or not the molecules as a whole should interact based on the positions of the centers of masses.

Even though the usage of the StS condition is generally more desirable from an accuracy perspective, the computational complexity of this condition surpasses the cost of the CtC condition by a significant margin. When the StS condition is used, the number of required distance checks increases quadratically with the number of sites per molecule. One way to mitigate this complexity increase is the introduction of a distance check between the

centers of masses in order to determine whether a partial interaction is even possible. If the molecules are far enough apart that no sites can be within the cutoff range, no matter the rotational orientations, the additional distance calculations can be omitted. Similarly, if the molecules are close enough together that no site can possibly be outside the cutoff-radius, the additional checks can also be disregarded. However, these optimizations are increasing the complexity of the control flow, which can hinder additional optimizations like vectorization.

2.7. Neighbor Identification Algorithms

Efficiently identifying all particle pairs that satisfy the cutoff condition is no trivial task in itself. This section introduces Linked Cells and Verlet Lists, which are two of the most commonly used neighbor identification algorithms to achieve that goal. Their respective advantages and disadvantages are also briefly mentioned.



(a) Linked Cells (b) Verlet Lists

Figure 2.2.: Graphical representation of the neighbor identification algorithms. All particles in the red circle are within the cutoff range of the highlighted particle. The blue area indicates for what particles a distance check is required, although they are outside the cutoff range. (Source: [9])

2.7.1. Linked Cells

In the Linked Cells algorithm, the simulation space gets divided into a three-dimensional grid of cubes with the length r_{cutoff}. Using this division, we can reduce the number of required distance checks significantly. A particle can only be within cutoff range to other particles in the same cell or in a directly adjacent cell. Distance checks with particles that are even further away can be omitted. This approach limits the number of required checks to $\mathcal{O}(n)$ for sufficiently homogeneous particle distributions. While this approach does filter out a lot of unnecessary distance checks, it does not do so quite as effectively as one might hope. For any molecule, the search for interaction candidates gets reduced to all molecules that reside in a cube of length $3 \cdot r_{\text{cutoff}}$. In contrast, the molecules that are actually within the cutoff range are located in a sphere of volume $\frac{4}{3}\pi r_{\text{cutoff}}$. Therefore, the Linked Cells algorithm limits the search space to a volume that is more than 6.4 times larger than the space where the actually interacting molecules can be located. This ratio between interactions that truly

should occur according to the cutoff condition and the ones that did not get filtered out by the neighbor identification algorithm is known as the *hit rate*. This hit rate can be improved by using a finer grid. Similar to an image of a circle at a higher resolution, the Linked Cell algorithm can use the smaller grid to approximate the sphere-shaped interaction space with higher accuracy. The downside of this method is the increased overhead cost of managing the grid structure. One of the advantages of the Linked Cell algorithm is the fact that it can be realized in a cache-friendly manner fairly easily by storing the particles within the same cell consecutively in memory. Additionally, the memory footprint and the computational overhead of the Linked Cells data structure is comparatively small relative to other neighbor identification algorithms.

2.7.2. Verlet Lists

Another neighbor identification algorithm uses Verlet Lists. In this approach, every molecule keeps track of all its neighbors by using a list. If these lists only contained molecules that were within cutoff range, they would have to be rebuilt for every force calculation since molecules move between iterations. To mitigate this, the Verlet Lists contain all the neighbors within a range of $r_{\text{cutoff}} + \Delta s$, where Δs is called the *verlet skin*. The idea behind this is that molecules move slowly enough so that penetrating through the verlet skin to get within the cutoff range will take numerous iterations. Therefore, we only have to rebuild the Verlet Lists in short enough intervals so that it would take longer for a molecule outside the Verlet List to get within the cutoff range. A trade-off emerges where a thicker verlet skin Δs will result in a more desirable rebuild frequency, but the Verlet List will be less restrictive in the interaction partner search space. In most scenarios, the verlet skin Δs is thin enough that Verlet Lists restrict the search space significantly more than the Linked Cells algorithm.

However, the question arises how these Verlet Lists are constructed. The simplest approach to construct these lists is to iterate over all molecule pairs. Unfortunately, as already established in Section 2.5, this would lead to a list construction runtime of $\mathcal{O}(n^2)$. Even though the asymptotic runtime of this algorithm would be the same as it is for the Direct Sum implementation, this approach can still yield high performance benefits. Verlet Lists do not have to be rebuilt for every iteration, which decreases the relevance of the quadratically scaling algorithm component. Another approach with a better asymptotic runtime is the usage of Linked Cells to construct Verlet Lists. This combination of the two approaches is called *Pairwise Verlet Lists*. It combines a lot of their advantages, such as the more restricted search space of Verlet Lists and the improved data locality of Linked Cells. However, the maintenance of these two data structures also results in their combined computational overhead.

Since the Verlet Lists algorithm requires the storage and maintenance of an additional data structure for every molecule in the simulation, it has a significantly higher memory footprint than the Linked Cells data structure. Additionally, the usage of Verlet Lists results in worse cache behavior since the algorithm iterates over Verlet List members, which are located at arbitrary memory locations. The main advantage of Verlet Lists is their ability to restrict the interaction partner search space significantly more strictly than the Linked Cells algorithm with a reasonably chosen verlet skin.

2.8. Newton's Third Law

Newton's third law (N3L) states that two bodies exerting a force on each other will experience forces of the same magnitude in opposite directions. This formula can be used to reduce the number of required force calculations by 50%:

$$\vec{F}_{ij} = -\vec{F}_{ji} \tag{2.7}$$

Even though it might seem like this reduction in computational workload should always be used to its full advantage, this is not always the runtime-optimal solution. In order to use N3L, the force accumulator of both interaction partners must be updated. As a result, we need write access to both particles. Ignoring N3L implies that write access to only one particle is required instead. Consequently, parallelizing the force calculation becomes significantly more challenging when using this law. Different approaches to parallelize the force calculation based on a certain neighbor identification algorithm are called *traversals*. As previous work has shown [5], the traversals that can be chosen when using N3L are parallelizable to a significantly lower degree than the traversals when this law is omitted. In fact, ignoring this law makes the problem embarrassingly parallel. Therefore, deciding whether or not this law should be used in any specific circumstance is a challenging optimization problem in itself.

2.9. AutoPas

Since the reduction of computation time is a high priority of molecular dynamics, determining and choosing the best neighbor identification algorithm is an important task. Unfortunately, it is common that the best algorithm at the start of a simulation is sub-optimal at a later stage. In order to deal with this phenomenon, AutoPas was created. AutoPas is a `C++` library capable of dynamically testing at runtime which algorithm has the best performance in the current simulation state. In order to do that, the library gathers information about the runtime of different algorithms in a process called the *tuning phase*. After the tuning phase, the best algorithm under the current circumstances gets used in the following *non-tuning phase*. Generally, the number of iterations used in the tuning phase is orders of magnitude lower than the number of non-tuning iterations, after which a new tuning phase commences.

In order to find the best algorithm, AutoPas deploys the so-called *Full Search Strategy*. All possible algorithm configurations are tested for a relatively small number of iterations, measuring the time needed in each case. After that, AutoPas chooses the configuration requiring the least amount of time. In addition to the neighbor identification algorithm, AutoPas also determines additional configuration parameters, such as the traversal and the data-layout.

2.10. md-flexible

Although AutoPas implements many of the data structures and algorithms required in MD simulations, it is not a simulator in itself. Instead, it is an auxiliary library that provides the neighbor identification algorithms and auto-tuning capabilities for a bigger framework to realize a simulator using these capabilities. One of the simulators built using AutoPas is md-flexible. Some of the features realized by md-flexible include the initialization of the

simulation based on an input file, the creation of output files, and the time integration steps during the simulation. While AutoPas selects the neighbor identification algorithm, it is also md-flexible's responsibility to perform the pairwise force computation itself. The particle management gets delegated to the `AutoPasContainer`-object. In order to do that, md-flexible defines a class that realizes the `Particle` Interface defined by AutoPas. These `Particle`s can then be inserted into the `AutoPasContainer`, from where the functionalities provided can be utilized.

Besides md-flexible, other MD simulators also use the functionalities provided by AutoPas. For example, it has also been integrated into other already established simulators, such as ls1 mardyn and LAMMPS [5].

2.11. Data-Layouts

When thinking about possible data structures to store molecules, the first thing that comes to mind is using an array, where every entry is a molecule. This structure is referred to as an Array of Structures (*AoS*). The i-th molecule can then be found at the i-th index of that array. Another way to represent the molecules would be creating an array for each member (such as mass, velocity in x-direction, etc.) and writing the values of the i-th molecule at the i-th position in the corresponding arrays. This data structure is called a Structure of Arrays (*SoA*). The information of a single molecule is spread out in memory, while values corresponding to the same member are grouped together. A graphical comparison of these two data-layouts can be seen in Figure 2.3.

In practice, it turns out that SoA is significantly more cache-friendly in a lot of cases. For example, when iterating over the positions of all molecules, a single cache line will be filled entirely with values that the CPU needs. An AoS would be better from a memory perspective if the program were interested in entire molecules with all of its members instead of a few members of many molecules.

AutoPas is capable of converting a given `Molecule` class into a suitable SoA at compile-time. In order to do this, the class has to inherit from the `Particle` class provided by AutoPas.

2.11.1. Entries of Non-constant Size

In general, every cell in an array needs to have the same size in order to enable constant-time random access. However, it is possible to create an array using an object that requires an arbitrary amount of space. One example would be an array of `Molecule` objects where one member of those `Molecule`s is a `std::vector`. This vector is represented as a reference to a memory location where the actual entries are stored. The reference itself only requires a constant amount of space, whereas the size of the data structure at that location is unknown.

Unfortunately, translating an AoS of those molecules into an SoA does not yield the memory benefits described in the previous section. Concatenating the vectors in an array means storing the memory *references* consecutively. The vectors themselves still remain at random locations. This problem is highlighted in Figure 2.3.

One possible approach to solve this issue would be flattening this two-dimensional data structure. In that process, we store the vectors consecutively in one large chunk of memory, redirecting the references accordingly. A graphical representation of this approach can be

seen in Figure 2.3. However, this solution also has its drawbacks. After the flattening process, it is no longer possible to append elements in an arbitrary vector cheaply. Additionally, copying all the vector entries to another memory location is very time-intensive.

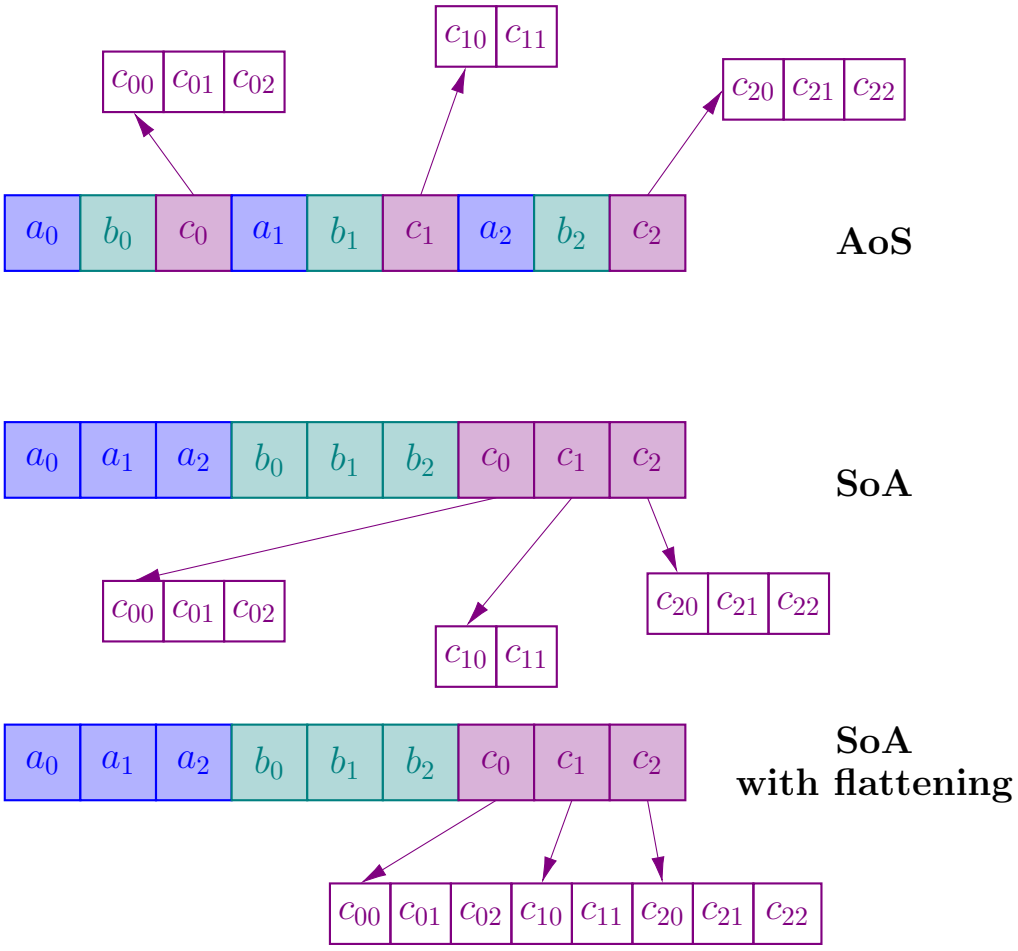


Figure 2.3.: Graphical representation of the AoS and SoA data-layout along with a flattened version of the two-dimensional array of the member c. c_i represents the memory reference, whereas c_ij correspond to the elements of the vector.

3. Technical Background

This chapter highlights the software-architectural structure of md-flexible. In order to do that, we explain the functionality provided by some of the most important classes. This context is necessary to understand the inefficiencies arising from the design decisions made in the making of md-flexible. The impact these decisions had on the overall runtime is analyzed from a theoretical perspective.

3.1. The AutoPasContainer

The `AutoPasContainer` is the central piece of the AutoPas library. It functions in a black-box fashion, where md-flexible can insert particles that should be governed by AutoPas. After inserting the particles, md-flexible can utilize AutoPas' functionalities, such as its neighbor identification algorithms and auto-tuning capabilities, through its provided interface to perform operations on them. However, fast random access to individual particles is no longer possible. The internal data structure used for these particles is abstracted away from md-flexible itself.

3.2. Particles

In order to initiate an `AutoPasContainer`, md-flexible has to define what objects AutoPas is supposed to treat as separate entities first. For that purpose, AutoPas provides a `Particle` class, which md-flexible inherits from in its own `Molecule` class. This `Particle` definition contains the set of minimal variables required so that AutoPas can apply its algorithms. For example, the `Particle` class contains a position variable since the neighbor identification algorithms of AutoPas would not really have any meaning on entities that do not have a defined position. As long as the `Molecule` class is an extension of the `Particle` class, md-flexible can freely choose what variables a `Molecule` object should have.

3.3. The Functor

As mentioned, AutoPas is a library responsible for determining every pairwise force calculation according to the cutoff condition in the most efficient way possible. However, it does not perform the force calculation itself. Instead, AutoPas provides a so-called `Functor`-Interface, which md-flexible realizes. The actual force calculation takes place in the implementation of the `Functor` methods. For example, the `AoSFunctor` can be called to compute the force between two particles. As the name implies, these particles are represented in an AoS fashion, meaning they are represented as objects that are given as function parameters. This method can be used for any neighbor identification algorithm. When AutoPas decides to utilize the SoA representation of the particles, it has methods depending on the neighbor identification

algorithm it is using at that time. For example, when using Verlet Lists, it can use the `SoAFunctorVerlet` method. The function parameters are the Verlet List itself, the index of the molecule for which this list is valid, and the SoA where those molecules are stored. In order to compute the forces when the Linked Cells algorithm is used, two functions are required. The `SoAFunctorSingle` is responsible for computing the pairwise forces within a cell, while `SoAFunctorPair` computes the pairwise forces between two neighboring cells.

```
1  /** Computes the force between Particle i and j after checking whether they
       are within cutoff range*/
2  void AoSFunctor(Particle &i, Particle &j, bool newton3);
3
4  /** Computes the forces between molecules inside the given cell.*/
5  void SoAFunctorSingle(SoA cell, bool newton3);
6
7  /** Computes the forces between particles in cell1 and cell2. These cells are
       located next to each other.*/
8  void SoAFunctorPair(SoA cell1, SoA, bool newton3);
9
10 /** Computes the forces of particle i with all particles inside of the
       verletList that are within cutoff range*/
11 void SoAFunctorVerlet(SoA soa, const size_t i,
12   const std::vector<size_t> &verletList, bool newton3);
```

Listing 3.1: Functions of the Functor-Interface.

3.3.1. Functor Calls

A functor call is the call of a function defined in the `Functor`-Interface. The number of functor calls per simulation step heavily depends on the strategy used by AutoPas at that time. For example, the usage of Verlet-Lists results in one functor call per molecule. This means that the number of `Functor` calls increases linearly with the number of molecules. With every functor call, we compute the force between the target molecule and all its Verlet List members.

In contrast, the number of functor calls is entirely independent of the number of molecules when using the Linked Cells algorithm. Since one call corresponds to two cells interacting, the only way to increase the number of calls is by increasing the number of cells (i.e., increasing the simulation space). As a result, the number of times an individual molecule is involved in a functor call also depends on the neighbor identification algorithm and the data-layout used.

3.4. The ParticlePropertiesLibrary

At runtime, md-flexible reads an input file defining all the necessary simulation parameters. This file contains information such as the initial molecule positions, the dimensions of the simulation space, and specifications on which data structures and algorithms md-flexible is allowed to use (for example, explicitly stating that only Linked Cells must be used). This file also defines the molecule types used in the simulation. This type definition can be interpreted as a blueprint explaining how to construct a certain molecule type. It defines the mass, ϵ, and σ of each site in that molecule, along with its position relative to the center of mass

and the inertia tensor of the entire molecule. When md-flexible parsed all this information successfully, it stores these molecule blueprints in the `ParticlePropertiesLibrary` for later access through an interface.

3.5. The Current Representation of Multi-Site Molecules

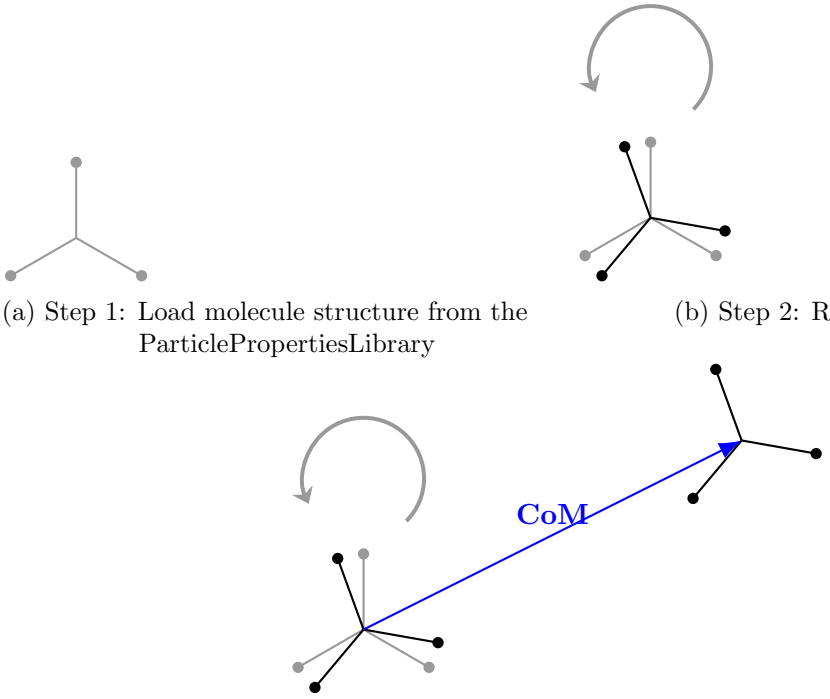
Prior to this work, md-flexible used to represent a molecule by storing its type, the location of its Center of Mass (CoM), and its current rotation. This representation contains all the information needed to fully define the state of a molecule. However, the site positions are not stored explicitly, meaning they have to be recomputed every time they are required. As displayed in Figure 3.1 below, the position computation consists of three steps. First, we read the position vectors, indicating the site positions relative to the molecule's CoM from the `ParticlePropertiesLibrary`. Second, we rotate these relative positions based on the current rotation of the molecule. Third, we perform a vector addition of the molecule's center of mass to the relative site positions we just computed. This process is very computationally intensive. Another drawback of this representation is that it does not enable lazy torque evaluation. As already mentioned in section 2.4, lazy torque evaluation requires site-specific accumulator variables to track the forces acting on each site. Since this molecule representation does not use site-specific parameters, the torques must be computed in an eager fashion. One major advantage of this representation is the fact that it only requires a constant amount of space. As a result, it can be stored in an SoA quite effectively. Additionally, this representation has a low memory footprint. All the variables are strictly necessary to define the state of a molecule; it does not have any kind of redundancy. In contrast, another representation storing the site positions explicitly would have redundant variables that could be inferred from other parameters.

3.6. Quantifying the Inefficiency of the Current Representation

The main drawbacks of this representation are the inability to use lazy torque evaluation and the need to recompute the site positions every time they are required. Both of these factors lead to an unnecessarily high number of floating point operations. Interestingly, the magnitude of these inefficiencies is not necessarily the same.

3.6.1. Lazy Torque

Using eager instead of lazy torque evaluation means performing a cross-product operation after every site-to-site interaction. A properly optimized implementation of the Lennard-Jones potential only requires 8 floating point operations for one site-to-site interaction. Computing a cross-product requires 9 operations. In other words, this inefficiency more than doubles the number of necessary computations even if the site-position problem did not exist. This observation does not mean a speed-up near 2 is possible in practice using lazy torque. This comparison solely regards the "useful" operations needed during the force calculation. The overhead associated with the control flow, the neighbor identification algorithm, the evaluation of the cutoff condition, etc., is a significant portion of the overall runtime, which was not taken into account in this analysis. Additionally, lazy torque evaluation introduces



(a) Step 1: Load molecule structure from the ParticlePropertiesLibrary

(b) Step 2: Rotate the molecule

(c) Step 3: Offset the rotated molecule by the center of mass

Figure 3.1.: Graphical representation of the Site position Calculation.

more variables, which increases memory workload. When eager torque is used, every molecule requires two variables to represent the force and torque it is experiencing. In a molecule-to-molecule interaction, the program only has to update these two variables. When lazy torque evaluation is used, the forces acting on each site have to be stored separately. As a result, the program has to update one variable for each site instead of updating exactly two variables for each molecule. Consequently, lazy torque reduces computational complexity at the cost of increasing memory workload. The usage of eager torque more than doubles the number of arithmetic operations required while reducing the memory requirements. Therefore, the impact of this trade-off depends on the type of bottlenecks the program currently has.

3.6.2. Site positions

Recomputing the site positions of a molecule with x sites requires $31 + 14x$ operations, which is considerably more expensive than the 9 operations required for a cross-product computation. However, these positions have to be evaluated significantly less frequently. In a `Functor` call, the site positions of each molecule only have to be computed once since they remain constant throughout the entire force calculation process. As a result, the number of site-recomputations does not depend on the number of site-to-site interactions (see eager torque in Section 3.6.1) but on the number of `Functor` calls the molecule is involved in. Since this amount varies depending on the neighbor identification algorithm and data-layout AutoPas uses at that time, so does the number of unnecessary computations.

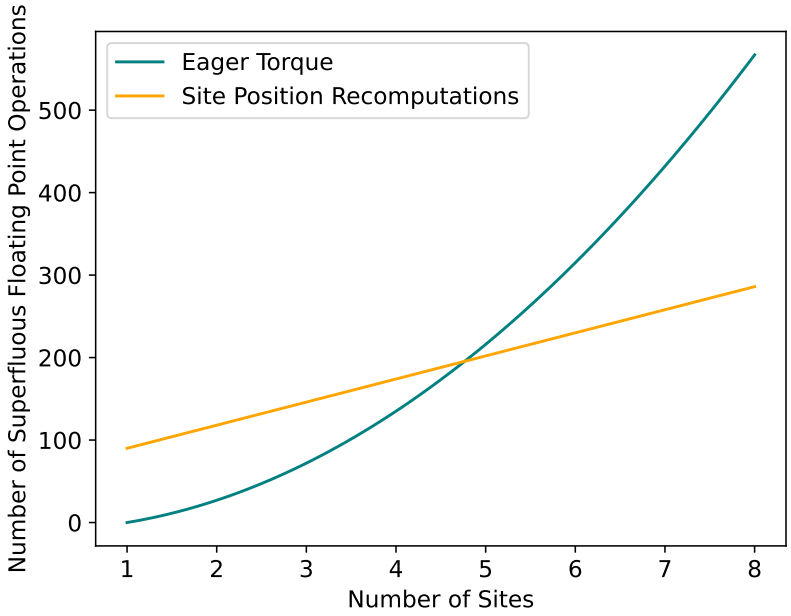


Figure 3.2.: Comparison of superfluous operations in a single molecule-to-molecule interaction when using the AoS data-layout

For example, when AutoPas is using the AoS data-layout, it will invoke a functor call for every molecule-to-molecule interaction, as described in section 3.3.1. Therefore, the number of required site-recomputations is twice as large as the number of molecule-to-molecule interactions in that scenario. Figure 3.2 displays the cost attributed to the site-position calculation and compares them to the number of superfluous operations caused by eager torque for molecules of different sizes. As can be seen, the linearly scaling cost of recomputing the site positions becomes decreasingly relevant for molecules with a higher site count. The cost of eager torque calculation is scaling quadratically, which becomes the dominant inefficiency for large enough molecules.

When AutoPas uses the SoA data-layout instead, determining the number of position recomputations required also depends on the neighbor identification algorithm.

One call of the `SoAFunctorVerlet` Function with a Verlet List of size m will result in the site position calculation of $m + 1$ molecules. The `Functor` computes the site positions of every list member and the molecule for which the list is valid. This molecule then interacts with all Verlet List members, meaning the $m + 1$ position calculations can be used for m molecule-to-molecule interactions. This is significantly better than the two position calculations for one molecule-to-molecule interaction in the AoS scenario.

When using the Linked Cell algorithm, every cell has to interact with every neighboring cell and itself. In total, every cell has to interact with 3^3 cells (including itself). Since every cell-to-cell interaction gets handled by a separate functor call, this means that the site positions of every molecule have to be recomputed 27 times. This value is independent of the number of force calculations that have to be performed.

In other words, the additional cost remains constant when the Linked Cells algorithm is used. When Verlet Lists are used instead, the site recomputation cost increases with the

number of molecule-to-molecule interactions. As a result, the number of superfluous position recomputations is larger for Verlet Lists as long as the simulation is sufficiently dense. The density, where the number of recomputations is exactly equal, is given by the term

$$\rho = \frac{26}{(r_{\text{cutoff}} + \Delta s)^3}. \tag{3.1}$$

To put this into perspective, when using a cutoff-radius $r_{\text{cutoff}} = 2$ and a verlet skin $\Delta s = 0.2$, using Verlet Lists will result in more position recomputations as long as the density is above 2.44. Considering that this density is unconventionally high, the number of required site recomputations would most likely be lower when using Verlet Lists with these specific parameters. However, increasing r_{cutoff} even slightly can lead to a significantly closer comparison.

In conclusion, this analysis shows that for large molecules, the usage of lazy torque evaluation reduces the number of floating point operations significantly more effectively than the explicit storage of site positions. Additionally, the number of site position recomputations required depends on the neighbor identification algorithm and data-layout used by AutoPas. When Linked Cells are used with the SoA data-layout, this overhead is constant with respect to the particle density. When Verlet Lists are used, this overhead increases with the density, but it remains lower than the overhead required for Linked Cells in many realistic simulation scenarios.

3.7. Alternative Molecule Representations

There are multiple changes that can be made to the internal molecule representation to address the previously discussed inefficiencies. This section highlights three solutions.

3.7.1. Approach I: Using Members of the Molecule class

The unnecessary site position recomputations can be avoided by explicitly storing the positions in a new member of the `Molecule` class. The `Functor` can access the site positions immediately through that member. Instead of recomputing the site positions multiple times throughout the force calculation phase, we only have to do so once after each time integration step because the positions and rotations of the molecules have changed. Therefore, the site positions are no longer valid and need to be updated.

Similarly, we can apply an analogous approach to realize lazy torque evaluation. As explained in Section 2.4, we need to store the forces acting on each site separately in order to be able to use the formula of lazy torque evaluation. We can realize these site-specific variables by adding another member to the `Molecule` class. This member keeps track of the forces each individual site experiences.

3.7.2. Approach II: Using an External Data Structure

The same benefits can also be obtained without storing the site-specific data in members of the `Molecule` class. Instead, we can also introduce an additional auxiliary data structure to which we delegate this responsibility. This new object will be referred to as the `SiteContainer`.

The molecules themselves only contain a reference they can use to access the data of their sites inside the container.

By utilizing an external data structure, this approach provides md-flexible with more control to use a beneficial memory layout. However, as will be discussed in greater detail, this approach is difficult to implement from a software-architectural perspective.

3.7.3. Approach III: Sites as Particles

As previously discussed in Section 3.2, md-flexible has to define what objects should be considered separate entities by AutoPas. This object has to inherit from the `Particle` class AutoPas provides.

One similarity between all the previously discussed approaches is that a particle always represents a molecule. The approaches only change the exact members of these objects or the information that is attached to them. However, md-flexible can also decide that AutoPas should treat all sites as separate entities instead of molecules. This idea will be referred to as the *site-based approach* in contrast to the previously utilized *molecule-based approach*. In order to implement this idea, we introduce a new `Site` class containing all the site-specific data.

Additionally, we introduce a new external `MoleculeContainer` class that governs the molecule-specific data, which is no longer delegated to AutoPas. This additional container is required in the time integration phase since the sites cannot be treated as individual entities during this phase.

4. Related work

Since multi-site simulations are supported in virtually all modern particle simulators, there are many academic projects that already had to settle on a molecule representation. This chapter compiles the design decisions made and the research about their effects.

4.1. md-flexible

The molecule representation originally utilized by md-flexible has already been explained in section 3.5. From a theoretical perspective, this representation leads to an unnecessarily high computational workload in the force calculation, as mentioned in Section 3.6. Previous work already quantified these negative consequences empirically for a vectorized functor implementation by investigating the runtime decomposition of the functor calls [11, p. 29]. According to those measurements, the runtime impact of the site recomputations decreases as the number of sites per molecule increases when using Verlet Lists. Considering that the number of site-to-site interactions increases quadratically, whereas the cost of recomputing the site positions increases linearly, this is expected. However, this impact only decreases from 80% of the overall functor runtime to 60% as the site count increases from 1 to 10. When using Linked Cells, recomputing the site positions requires a much more manageable 18% to 35% of the overall functor time. These results motivate the usage of explicitly stored site positions due to their potential to substantially improve the functor runtime. However, there are other factors at play that could negatively influence the overall runtime. The additional variables can only prove beneficial if the memory bandwidth is large enough to handle the increase in traffic. Additionally, due to their increased memory footprint, fewer molecules can be stored in the cache. A poorly chosen data structure to store the site positions can also hinder effective vectorization.

4.2. ls1 mardyn

The simulator ls1 mardyn does not utilize site-specific parameters in its multi-site molecule representation. Instead, the used data structure is analogous to the approach md-flexible has been using prior to this work [12, p. 842, Table 1]. While this design decision increases computational complexity, a significant focus in the development of ls1 mardyn is placed on reducing memory requirements. For the same reason, ls1 mardyn is predominantly focused on the Linked Cell algorithm. With the goal of reusing the molecule data loaded from previous cell-to-cell interactions, a sliding window traversal of the cell data structure is utilized [13, p. 7]. This sliding window traversal also motivates the use of the *Hybrid Scheme* [14, p. 69], where the newly computed site positions get stored until the corresponding cell moves out of the sliding window again.

This approach tries to capitalize on the benefits of a compact memory representation and the reduced workload of stored site positions. Consequently, the effectiveness of this method depends on the traversal used and is entirely inapplicable for some, such as lc_c01. See [5] for a more thorough explanation of traversals and their effects. This decision can be justified since such a traversal would only be chosen when the computational workload is not the bottleneck, to begin with. Since the design decisions in ls1 mardyn have been driven by the hardware specifications of its target architecture, the results can also change as requirements evolve.

A representation similar to the site-based approach (Section 5.3) has also been proposed [14, p. 75-76] and consciously rejected since this representation results in an unnecessarily high workload for the CtC cutoff condition. Considering that the StS cutoff condition should also be supported by md-flexible; this argument does not undermine the case for a performance increase with this adapted condition.

Additionally, the hybrid scheme used in ls1 mardyn is only applicable to the Linked Cell algorithm and cannot be easily extended to other neighbor identification algorithms, such as Verlet Lists. Because of memory requirements, using Verlet Lists has not been the focus of ls1 mardyn, to begin with. Considering that the previous research in md-flexible implied a particularly strong bottleneck when combining this molecule representation with Verlet Lists, a more thorough analysis of different data structures, their interaction with neighbor identification algorithms, and the effects on the overall runtime is needed.

4.3. GROMACS

In the Groningen Machine for Chemical Simulations (GROMACS), the term “molecule” does not correspond to a rigid body of sites, like in this thesis. GROMACS primarily focuses on the simulation of macromolecules, where molecule-internal forces have a significant enough effect on the simulation that they cannot be ignored. Therefore, the term “molecule” refers to a set of atoms with additional constraints imposed on them [15]. Among others, the possible angle between atoms can be limited to a defined range, or atoms can experience a corrective force when they leave their neutral position in the molecule. These constraints generally allow the atoms to move freely to some extent. However, they can also be chosen to effectively make the molecule rigid [16]. The position of each atom cannot be inferred by the state of the molecule in general. Therefore, every atom has to be modeled as a separate entity by GROMACS. Nevertheless, GROMACS also supports the use of rigid sites. These sites are then considered to be part of an atom. As a result, GROMACS can represent a molecule in what we consider a molecule-based or a site-based approach. However, if GROMACS is using the molecule-based approach, it would have to represent a molecule using the “atom” class.

A comparison of the runtime efficiencies of these different representations hasn’t been made yet. Other simulators like ls1 mardyn also support the simulation of macromolecules, where molecule-internal forces are also a significant factor. In the case of GROMACS, this functionality became the focus of the simulator to a degree, where it also affects the terminology used.

5. Implementation

This chapter elaborates on the software-architectural changes necessary to implement the alternative molecule representations. Along with the benefits these representations provide, they also introduce their own set of drawbacks. We also discuss these trade-offs in greater detail.

5.1. Approach I: Using Members of the Molecule-class

The first approach utilizes members of the `Molecule` class to store site-specific data such as the site positions. A simplified UML diagram of this approach can be seen in Figure 5.1. The newly added variables are marked in green. When trying to determine the possible type of these members, we have to realize that the number of sites per molecule can vary from molecule-type to molecule-type. Additionally, the molecule-types are defined in the input file that is not available at compile-time. In order to meet those requirements, the site positions need to be stored in a data structure of variable size, such as a `std::vector`. Unfortunately, as already described in section 2.11.1, the SoA representation of those molecules will be sub-optimal. The array of site positions will be an array of `std::vectors`, which means that the relevant data does not lie consecutively in memory.

Analogous to the site position, the same approach can be used to enable lazy torque evaluation. In order to implement lazy torque, it is necessary to track the forces acting on each site separately.

Since these force accumulators are site-specific parameters as well, they would also have to be stored in a data structure of arbitrary size, such as a `std::vector`. Similarly, the data structure of these force-accumulators will not be as intended when SoAs are used. This realization of lazy torque evaluation introduces the additional drawback that more variables have to be stored close to the CPU. As already described in section 3.6.1, every site needs its own force accumulator in order to enable lazy torque evaluation. While the site positions have to be computed regardless, only two variables are required to represent the force and the torque a molecule experiences. Choosing to store the forces exerted onto each site instead results in the usage of more variables at the benefit of less computational load. For this reason, it is not trivial whether or not these extended molecule representations have a beneficial effect on the overall runtime. Both the explicit storage of site positions and the usage of lazy torque evaluation are increasing memory requirements while decreasing computational workload to some extent. However, since the degree to which they are doing this differs, the impact of these two optimizations has to be evaluated separately.

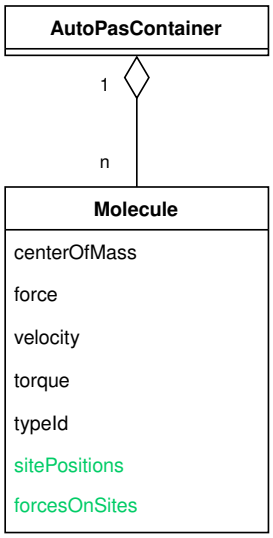


Figure 5.1.: UML representation of the molecule-based approach

5.2. Approach II: Using an External Data Structure

The main drawback in approach I is that the auto-conversion from AoS to SoA in AutoPas does not result in the site-specific variables being stored in the desired consecutive manner. The second approach is solving this issue by not using the SoA-conversion of AutoPas for site-management at all. Instead, we utilize our newly created `SiteContainer`, in which we can choose a more beneficial memory layout. Before the force calculation, we compute all site positions and store them consecutively in this data structure. A simplified version of the total molecule representation based on this approach is displayed in Figure 5.2.

This approach seems to solve the underlying problem. Unfortunately, it is not feasible from a software-architectural perspective for multiple reasons. First, the force functor needs to inherit from the `Functor` class provided by AutoPas. Since the newly introduced `SiteContainer` is not a part of the function call signatures, the functor cannot be given a reference to said structure in each call. Therefore, this `SiteContainer` would have to be accessible in a different manner, for example, by making it a member of the `Functor`. However, AutoPas is intended to be used as a black-box container [5]. From the outside perspective, it is unknown which neighbor identification algorithm it will use next. This poses an issue since this information is vital in determining the optimal data layout the `SiteContainer` should use. For example, when AutoPas is using the Linked Cells algorithm, it would be beneficial if the `SiteContainer` also stored the sites belonging to the same cell consecutively. If that were achievable, a functor call working on that cell would have all the required sites consecutively in memory. However, since the data structure used by AutoPas to represent the molecules is not accessible at runtime and the strategy it is about to deploy is also unknown, that vital piece of information cannot be used to build a `SiteContainer`. The best implementation representative of this approach that does not require the usage of AutoPas-internal states is the creation of such a data structure in the functor call itself. In this modified version, site positions remain a member of the molecules, but at the start of each functor call, this array of vectors gets flattened into one consecutive data structure.

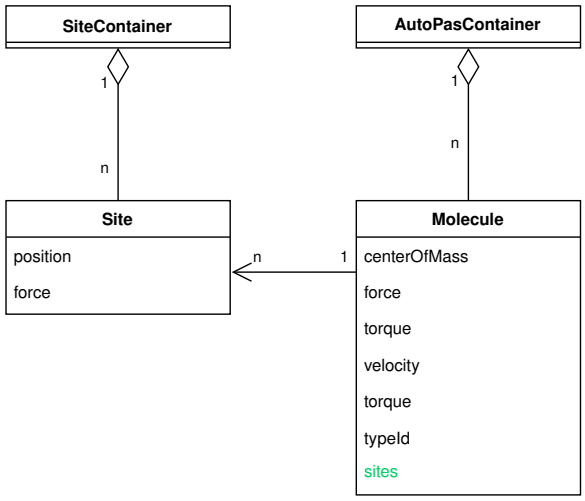


Figure 5.2.: UML representation of approach II

The drawback of this software-architectural workaround is that this data structure has to be rebuilt with every functor call, like the site positions have to be computed in every functor call with the baseline implementation. If this flattening process is significantly cheaper than the site recomputations, this approach can still result in a performance boost, but it does not solve the underlying problem to a satisfying degree.

5.3. Approach III: Sites as Particles

AutoPas is best suited to govern particles that require a constant amount of storage space. This poses a compatibility issue since the number of sites inside of a molecule is arbitrary. Therefore, a molecule representation containing site-specific data cannot be of constant size. However, the representation of each site individually does require only a constant amount of space. This idea is the main motivation behind approach III, where every site is treated as its own particle. The `Site` class inheriting from `Particle` contains all the site-related information such as σ, ϵ, the site position, and the force F it is experiencing. Additionally, every site stores the molecule-ID identifying the molecule it belongs to, along with a site index determining which site inside of the molecule it is. The `MoleculeContainer`, where all the molecules themselves are stored, has to provide fast random access to any molecule it is storing. In the current implementation, it is using a `std::vector` to achieve this goal. As additional requirements (such as the insertion or deletion of molecules during the simulation) become more relevant, this backend representation can be changed easily without affecting the interface provided.

In the force calculation phase, the `Sites` can be treated as separate particles interacting with each other. However, when two particles belonging to the same molecule (i.e., having the same molecule-ID) would interact with each other, we omit this force calculation. From a purely theoretical perspective, this condition is unnecessary. According to Newton's third law, particle i and j are exerting equally strong forces in opposite directions onto each other, which means that the molecule as a whole will not get accelerated. The forces cancel out.

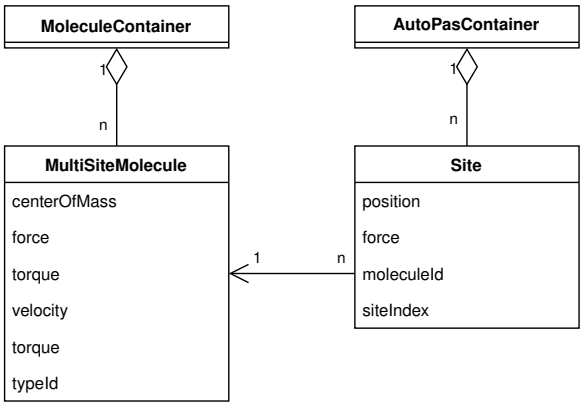


Figure 5.3.: UML representation of the site-based approach

Similarly, a molecule cannot exert a torque onto itself. However, sites belonging to the same molecule are significantly closer together than particles of different molecules. Therefore, the Lennard-Jones potential between them will be orders of magnitude higher. As a result, forces between particles of different molecules would get absorbed since they are represented using the IEEE 754 double-precision floating-point standard [10]. This representation only allows for a fixed number of relevant digits. When we add two values in that representation that vary by orders of magnitude, the last relevant digits of the smaller number will be ignored. When values are far enough apart, the smaller value can even be rounded away entirely. In order to avoid this kind of precision loss, sites of the same molecule must not interact.

5.4. Time Integration in the Site-based Approach

Since AutoPas only knows about sites in this approach, the force calculation will also not update the molecues' force- and torque parameters. Therefore, we must accumulate the site data before the time integration can commence. The most intuitive way to accomplish that goal would be to iterate over all molecules, summing up the forces their sites experience. Unfortunately, AutoPas does not provide fast random access to particles. Accessing all the sites of a certain molecule cannot be done efficiently. However, since the sites can reference their molecule via the molecule-ID, they can efficiently report the force exerted on them. The torque caused by that site can also be computed efficiently and stored in the molecule class. After this gathering phase, the molecules contain the same information as if they were multi-site particles governed by AutoPas. Therefore, the time integration on these molecules can be performed as usual. After the time integration phase, the molecule position, velocity, rotational orientation, and rotational velocity have been updated in the molecule objects. However, the sites governed by AutoPas have not updated their state based on the newly computed parameters of their molecule yet. Again, the most intuitive way to solve this problem would be iterating over all molecules and updating the parameters of all their sites based on the molecule state. However, this is not feasible since the molecules cannot access their sites efficiently. Analogously to the solution used during the gathering phase, this problem can be solved by iterating over all sites again. Every site accesses

Algorithm 1: Site-based time-integration

```
1 for site ∈ autoPasContainer do
2     molecule = moleculeContainer.getMolecule(site.getMoleculeID())
3     molecule.updateBasedOnSiteData(site)
4 for molecule ∈ moleculeContainer do
5     molecule.performTimeIntegration()
6 for site ∈ autoPasContainer do
7     molecule = moleculeContainer.getMolecule(site.getMoleculeID())
8     site.updateBasedOnMoleculeData(molecule)
```

Figure 5.4.: Pseudo-Code of the Site-based time-integration.

the newly computed molecule state by using its molecule-ID. Based on that information, the site computes the position it should be in and updates itself accordingly. The entire time-integration process, including the gathering phase and the site position updates, is displayed in Figure 5.4.

This time integration technique is significantly more expensive than the integration in the molecule-based approach. Instead of one loop (over all the molecules), three loops are required. However, the overall runtime of a particle simulation is dominated by its force calculation. In most simulations, the force calculation is responsible for significantly more than 90% of the overall runtime. Therefore, increasing the complexity of the time integration can be justified as long as it simplifies the force calculation in return.

6. Performance Comparison

As already established in Section 3.6, each molecule representation has its benefits and drawbacks. The impact of these trade-offs on the overall runtime depends on the algorithm chosen by AutoPas and the simulation size. Therefore, every combination of molecule representation and neighbor identification algorithm are tested on various inputs to capture these relations.

Every input tested has a simulation space of $21 \times 21 \times 21$. The molecules are placed in a `CubeClosestPacked`-pattern of size $20 \times 20 \times 20$, where a constant distance of 0.5 is maintained to each border. CubeClosestPacked is layering the molecules to fit as many of them as possible into the limited space while maintaining a defined minimal distance between each molecule and its neighbors. This molecule formation is then simulated by four threads for 100 simulation steps. In the first set of measurements, the number of sites per molecule is fixed at 5, while the density varies as the minimal distance between the molecules changes. In the second set of measurements, the number of sites varies while the particle density is fixed at around 0.87 (or 6970 molecules).

We take these measurements for every possible combination of molecule representation, neighbor identification algorithm, and data-layout. When Verlet Lists are used, the cutoff-radius and verlet skin are set to $r_{\text{cutoff}} = 2$ and $\Delta s = 0.2$. In this case, we use the traversal *vl_list_iteration*, which means that Newton's third law does not get applied, making the problem embarrassingly parallel [5]. Using a different traversal does not change the result significantly. When the Linked Cells algorithm is used, the cell size used is equal to the cutoff-radius $r_{\text{cutoff}} = 2$, and the traversal is *lc_c08*. This traversal makes use of Newton's third law while maintaining a high degree of parallelism [5].

Additionally, the time step size is set to $\Delta t = 0$. Thereby, the molecules will not move in the time integration phase. The program flow remains unchanged; at every iteration step, the pairwise forces get calculated, after which a time integration step computes the new positions, velocities, and rotational orientations. However, since the change in position includes a multiplication with Δt, the newly computed positions will be identical to the starting configuration. Particle simulations are chaotic systems, which means that the simulation states can diverge significantly when the order of floating point operations changes. Since some of the approaches are doing exactly that, any other choice of Δt would, therefore, result in diverging simulations, which can marginally distort the measurement results.

The measurement series described was taken on the CoolMUC2 system at Leibniz-Rechenzentrum (LRZ). It utilizes Intel Haswell nodes with 14 cores per CPU at a clock speed of 2.6 GHz. Additional information about the hardware specifications can be found online: https://doku.lrz.de/coolmuc-2-11484376.html.

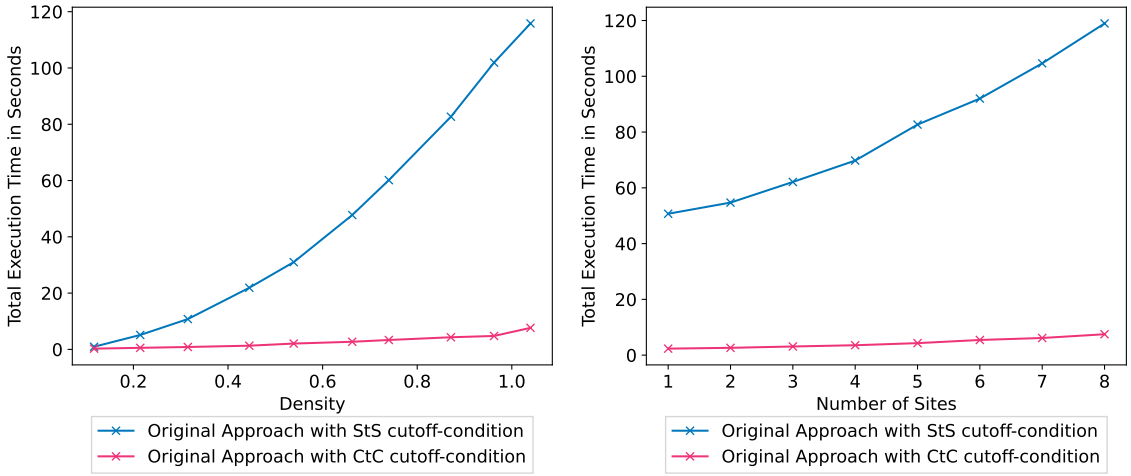
6.1. Runtime Comparison of StS and CtC Cutoff Conditions

In order to determine the utility of different molecule representations, it is crucial to quantify the impact of the cutoff condition on the total runtime first. As already established in section 2.6, the number of distance checks increases quadratically with the number of sites in a molecule when the StS condition is used. This amount could be lowered by introducing a check based on the centers of masses to determine whether a partial molecule interaction is even possible. However, the functor utilized for these measurements does not use this additional optimization. Instead, the site-to-site distance gets evaluated for every site-pair identified as "potentially interacting" by the neighbor identification algorithm. Therefore, this inefficiency becomes significantly more detrimental to the overall runtime when combined with an identification algorithm with a low hit rate. When using Linked Cells, this effect can slow down the program by a factor of 15 to 20, depending on the site count, as can be seen in Figure 6.1. The same phenomenon affects the runtime significantly less drastically when using Verlet Lists instead, as shown in Figure 6.2. Since this approach filters out molecules beyond the cutoff range significantly more effectively, the program uses a smaller portion of its overall runtime to perform distance checks with the remaining false positives. Therefore, managing these cases in a less efficient manner does not affect the overall runtime as much. It is noteworthy that all the portrayed results use the AoS data-layout, meaning that the site positions have to be recomputed for distance checks in every potential molecule-to-molecule interaction. These positions do not have to be recomputed when the CtC condition is used instead and the cutoff condition fails. These additional recomputations can be avoided by using a different molecule representation. Nevertheless, they are part of the measurements shown in Figures 6.1 and 6.2.

When utilizing the SoA data-layout, the functor used actually computes the Lennard-Jones potential between every site pair that has been identified by the neighbor identification algorithm as "potentially interacting". After that, we realize the cutoff condition by masking away the forces attributed to interactions beyond the cutoff-radius. As previous work has shown [11], this kind of site mask can be beneficial in combination with Verlet Lists, but it is sub-optimal for the Linked Cells algorithm and its low hit rate. However, since all molecule representations are equally influenced by these design decisions, a fair performance comparison can still be made based on the gathered data. The only factor that could lead to an improved functor design qualitatively changing the result of the following comparison is the fact that the functor can reduce the number of distance calculations more effectively in the molecule-based approaches than it can in the site-based approach. Discounting the runtime repercussions of this asymmetry, the performance comparison should be independent of the exact functor implementation used.

6.2. Comparison of the Molecule-based Approaches

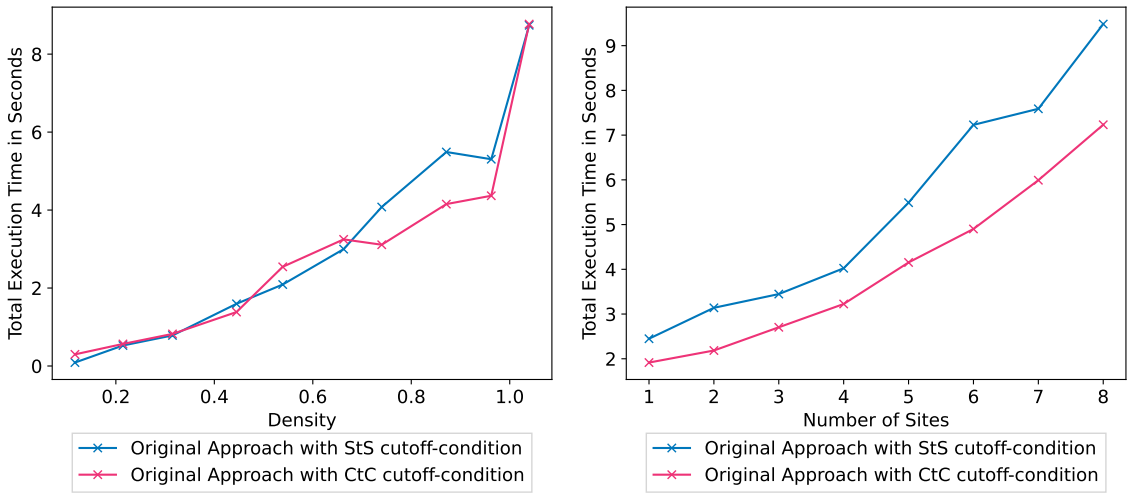
The performance benefits obtainable by using molecule representations with site-specific data are highly dependent on the data-layout, the neighbor identification algorithm, and the cutoff condition used. As the Figures 6.3a and 6.4a show, lazy torque evaluation in the Linked Cells algorithm results in a performance benefit relative to the original implementation for molecules with a high enough number of sites. This result is to be expected from a



(a) Number of Sites=5, varying Density

(b) Number of Molecules= 6970 (Density ≈ 0.87), varying Number of Sites

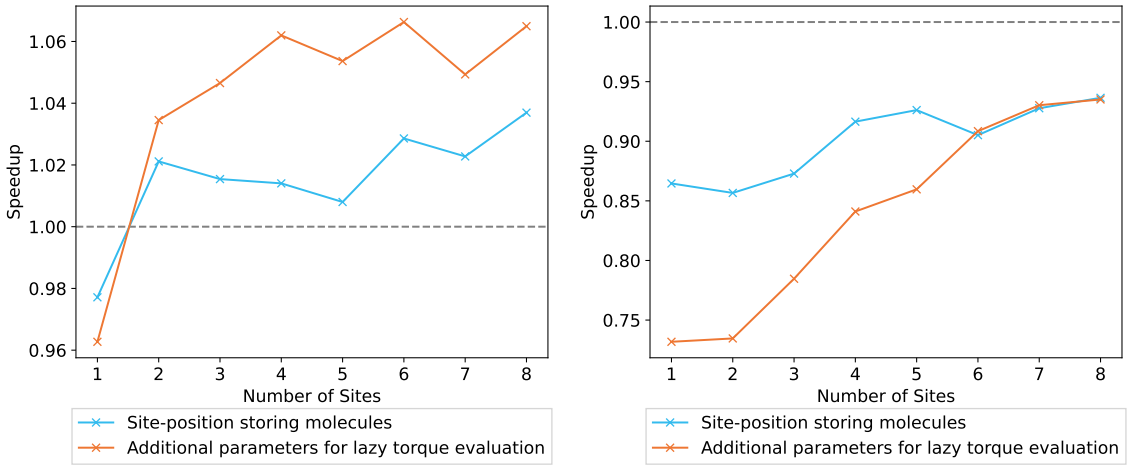
Figure 6.1.: Runtime comparison of different cutoff conditions using the **AoS** data-layout and the **Linked Cells** algorithm.



(a) Number of Sites=5, varying Density

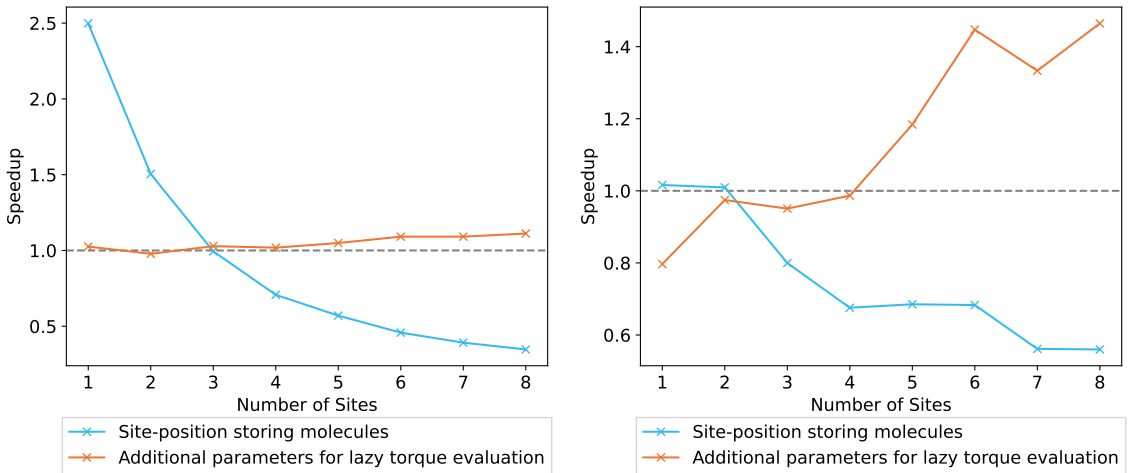
(b) Number of Molecules= 6970 (Density ≈ 0.87), varying Number of Sites

Figure 6.2.: Runtime comparison of different cutoff conditions using the **AoS** data-layout and **Verlet Lists**.



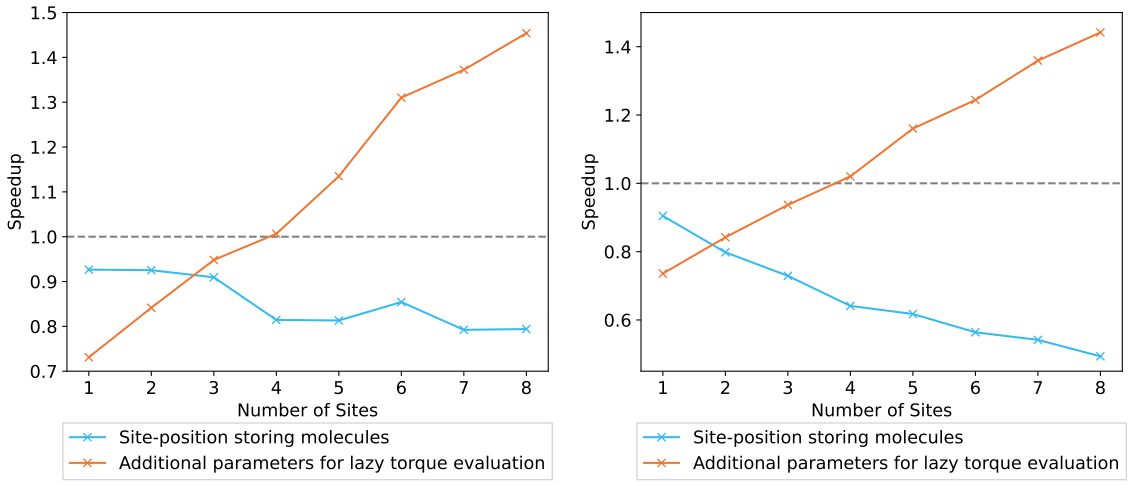
(a) Linked Cells (b) Verlet Lists

Figure 6.3.: Speedup relative to the original Implementation with the **SoA** data-layout at a density of approximately 0.87 (6970 molecules).



(a) Linked Cells (b) Verlet Lists

Figure 6.4.: Speedup relative to the original Implementation with the **AoS** data-layout at a density of approximately 0.87 (6970 molecules).



(a) Linked Cells

(b) Verlet Lists

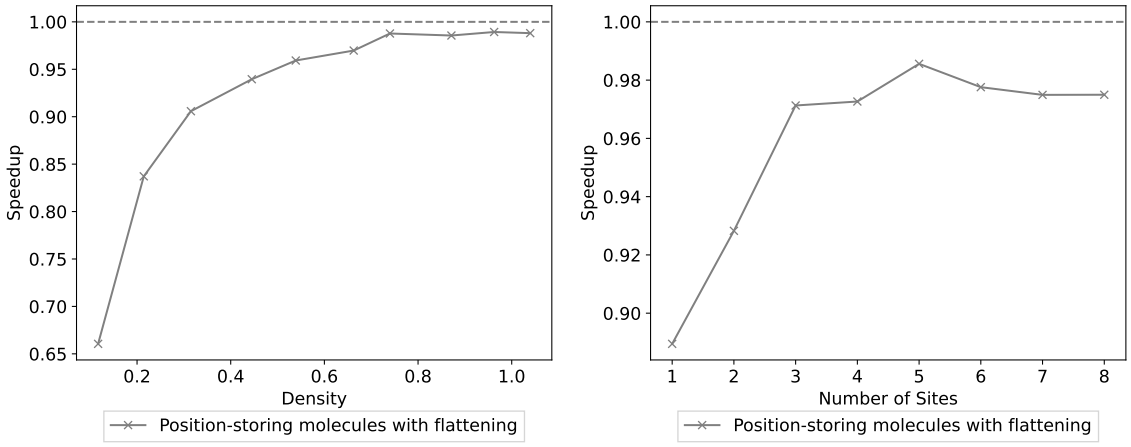
Figure 6.5.: Speedup relative to the original Implementation with the **AoS** data-layout, the **CtC** cutoff condition and a density of approximately 0.87 (6970 molecules).

theoretical perspective. With the increasing number of sites, the number of interactions increases quadratically, which implies that the interaction calculation represents a larger share of the overall workload. Optimizing this section of the program can, therefore, yield a higher performance benefit. However, the speedup only increases by a relatively modest factor of 1.06 for the SoA data-layout and 1.10 for the AoS data-layout.

This result can be explained by the inefficiencies surrounding the implementation of the StS cutoff condition. As already established in section 6.1, the force functor is investing a significant portion of its runtime into overhead computations that do not benefit from the reduction in runtime per site-to-site interaction. Therefore, optimizing this code section becomes less relevant to the overall runtime. When utilizing the CtC cutoff condition in combination with the AoS data-layout, the computational overhead is reduced. As a result, the removal of additional cross-product computations becomes more relevant to the overall runtime, and the speedup achievable with 8 sites per molecule increases to more than 1.4, as shown in Figure 6.5b.

Interestingly, the usage of explicitly stored site positions is unable to positively affect the overall runtime when using Verlet Lists. Considering that the usage of Verlet Lists results in the program utilizing a significant portion of its runtime on the recomputation of site positions, as previous work showed [11], the explicit storage of these positions falls short of the anticipated performance results in combination with this neighbor identification algorithm. These measurements can be better understood by not only modeling the computational cost of recomputing the site position but also the bandwidth required to access the additional variables from memory. Whenever a site position is required, it must be loaded from memory instead of being recomputed. If these values are not already present in the cache, accessing them can take significantly longer than it would have taken to simply recompute them.

When using Linked Cells with the AoS data-layout, the speedup depends on the cutoff condition used. When using the CtC cutoff condition, the site-position storing approach



(a) Number of Sites=5, varying Density

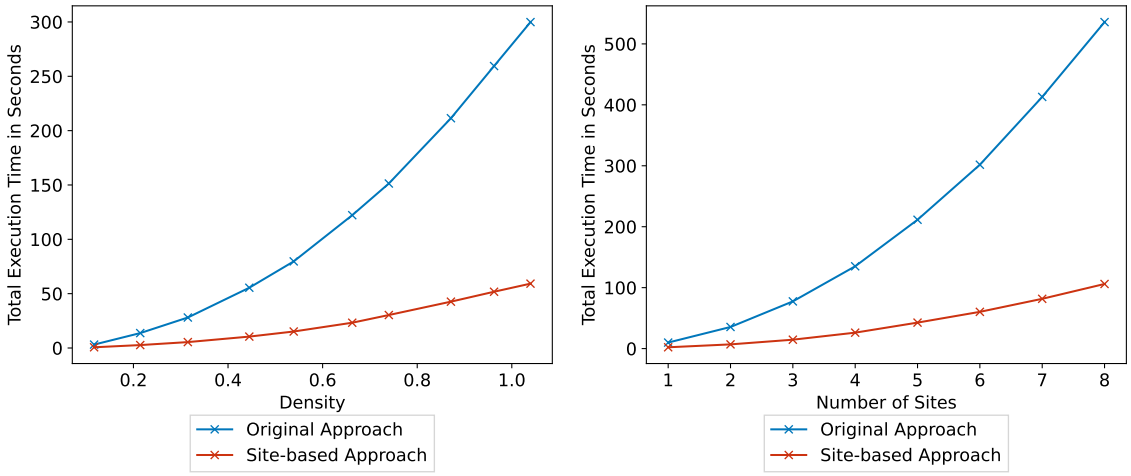
(b) Number of Molecules= 6970 (Density ≈ 0.87), varying Number of Sites

Figure 6.6.: Speedup achieved by flattening the site-positions array in the **Linked Cell** algorithm with an **SoA** data layout.

is worse in every tested scenario, as shown in Figure 6.5a, with a speedup decreasing from 0.93 to 0.8. When using the StS condition, a positive speedup can be achieved for small enough molecules, as shown in Figure 6.4. Additionally, the efficiency discrepancy becomes significantly wider. The speedup decreases from 2.5 for molecules with one site to 0.35 when using eight sites. This speedup is almost exactly 1 when each molecule has three sites. Considering that most multi-site simulations contain molecules with a site count greater than three, the scenarios where the speedup is beneficial are not useful in practice. The impact of the site position representation on the runtime can be significantly larger when using the StS condition since the position must be accessed significantly more frequently with this condition. When using CtC, we only have to access the site positions once we already determined that an interaction will take place. When using StS, we need to access the site positions in order to verify whether or not that will be the case. Therefore, changing the cost of accessing the site positions is significantly more impactful to the runtime when using the latter condition.

6.3. Site-Position Storing Molecules: The Impact of Flattening

Flattening the two-dimensional array of site positions can only yield a positive impact when these entries are used more than once. In order to construct the flattened array, we have to read every site position from its random memory location. When every site position only has to be read in once regardless, then using the two-dimensional array requires the same amount of read-instructions that building this additional data structure would need. In other words, the time benefit of building this new data structure would be smaller than the cost required to construct it. Therefore, flattening cannot have a positive impact when using Verlet Lists. The only neighbor identification algorithm that uses the site position of a molecule more than once is the Linked Cells algorithm.



(a) Number of Sites=5, varying Density

(b) Number of Molecules= 6970 (Density ≈ 0.87), varying Number of Sites

Figure 6.7.: Runtime Comparison of Site-based approach and Molecule-based original approach using **Linked Cells** algorithm and **SoA** data-layout.

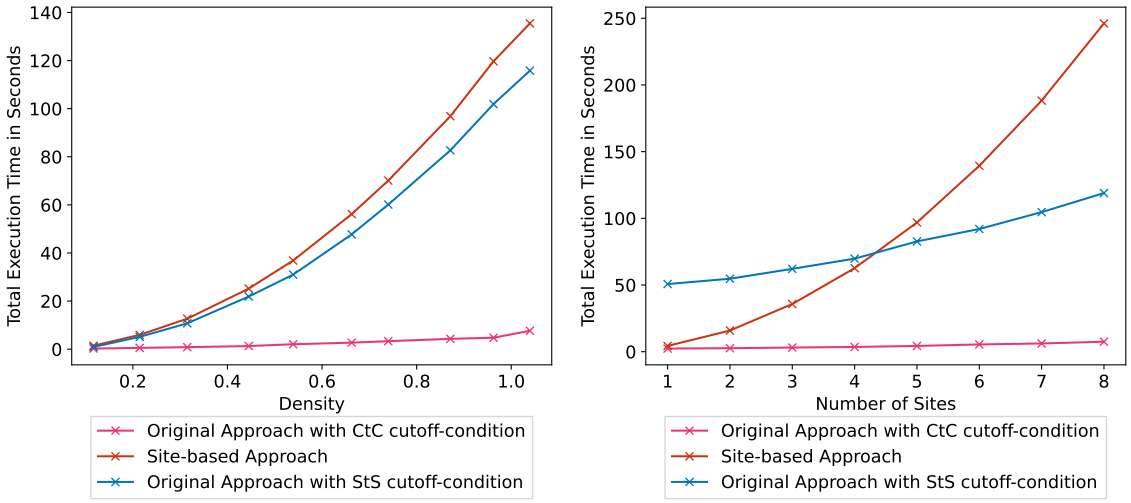
As shown in Figure 6.6, flattening the site position vector does not result in a better runtime for any tested simulation configuration. Considering that the construction of this one-dimensional array requires many copy-operations, this result is not surprising. Additionally, figure 6.6a shows that increasing the density will also increase the speedup achieved. This is to be expected since an increase in pairwise particle interactions means that the expensively built data structure can be used more often, driving down the amortized construction price. However, the positive benefit of higher densities is not strong enough to result in a performance increase with the flattening approach.

6.4. Comparison of Site-based Approach and Molecule-based Original Approach

While the different molecule-based approaches' runtime performances deviate slightly from each other, this discrepancy is significantly larger when comparing these representations with the site-based approach. For that reason, the figures used in this section do not display the runtime performance of site-position storing molecules or the extension for lazy torque evaluation. Including these approaches would worsen the visual clarity of the graphs without providing new, meaningful insights.

6.4.1. Linked Cells

As Figure 6.7 shows, the site-based approach is performing significantly better than the molecule-based approach when the Linked Cells algorithm is used with the SoA data-layout. The cause of this result is not to be confused with the explanation given for the significant runtime difference when comparing different cutoff conditions in section 6.1. Both functors



(a) Number of Sites=5, varying Density

(b) Number of Molecules= 6970 (Density ≈ 0.87), varying Number of Sites

Figure 6.8.: Runtime Comparison of Site-based approach and Molecule-based original approach using **Linked Cells** algorithm and **AoS** data-layout.

displayed in this figure use the StS condition. The observed speedup of approximately 5 is significantly larger than the measured speedups achieved in the molecule-based approaches through various optimizations. Therefore, the speedup achieved can not solely be attributed to the lazy torque evaluation and the omitted recomputations of site positions, which the site-based approach is taking advantage of. Instead, the cache-friendliness of the data structures constructed by AutoPas also has to play a major factor. Another indicator for this statement is the fact that the site-based approach performs significantly worse when using the AoS data-layout. In this scenario, the performance comparison of the different approaches mainly depends on the site count, while the density has a significantly smaller impact, as displayed in Figure 6.8. For large enough molecules, it is the molecule-based approach that has a more beneficial runtime with the StS-cutoff condition. These results indicate that the runtime superiority of the site-based approach with the SoA data-layout depends on the memory benefits the site-based approach could gather from the SoA structure.

These graphs also highlight again just how important the cutoff condition is. When comparing the site-based approach to the molecule-based approach with the CtC condition, the latter has a significantly superior runtime in every tested scenario.

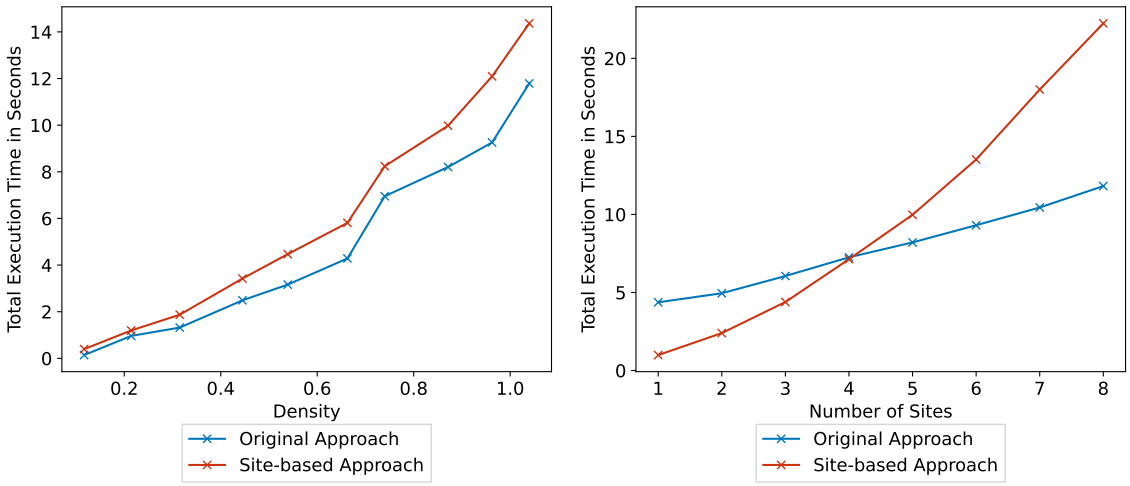
6.5. Verlet Lists

Figure 6.9 displays the runtime comparison between the site-based approach and the molecule-based approach when Verlet Lists are used in combination with the SoA data-layout. As can be seen in Figure 6.9b, the site-based approach performs better for molecules with a small number of sites, whereas the molecule-based approach has a better runtime for larger molecules. Altering the density does not affect the speedup (Figure 6.9a). In order to explain these results, a runtime decomposition of the force calculation is constructed using AutoPas'

built-in logging feature. Using this functionality, AutoPas logs the time spent in different phases of the control flow that are required during the force calculation. The data points used to construct the graphs shown in Figure 6.10 are the mean time spent in each displayed phase for a single simulation step. This additional measurement series was performed on a personal laptop instead of the CoolMUC2 cluster. The hardware specifications of this environment can be found in the appendix.

As Figure 6.10b shows, the majority of runtime is spent in the functor call when the molecule-based approach is used. The overhead required to rebuild the Verlet Lists remains constant as the number of sites increases. From a theoretical perspective, this is logical since the Verlet Lists construction costs are only dependent on the number of particles as well as their positions. Changing the number of sites means changing a property that is independent of the Verlet List construction process. Therefore, the cost attributed to it remains constant. However, this also means that the construction cost becomes dependent on the site count when every site is treated as its own particle. As can be seen in Figure 6.10a, the rebuilding process of Verlet Lists actually becomes the major bottleneck when this is the case. Since every site is its own particle, both the number of lists that have to be constructed, as well as the length of these lists increase simultaneously. Interestingly, the site-based approach requires less time in the actual functor calls. Figure 6.11 displays these computation times, disregarding the additional overhead attributed to the rebuilding process of the Verlet Lists. As can be seen, the site-based approach is significantly more efficient in that phase for small molecules. When the number of sites per molecule reaches 8, the site-based approach is just barely less efficient than the molecule-based approach. In simpler terms, the site-based approach has the potential to be more efficient than the molecule-based approach if the overhead attributed to the Verlet Lists can be overcome.

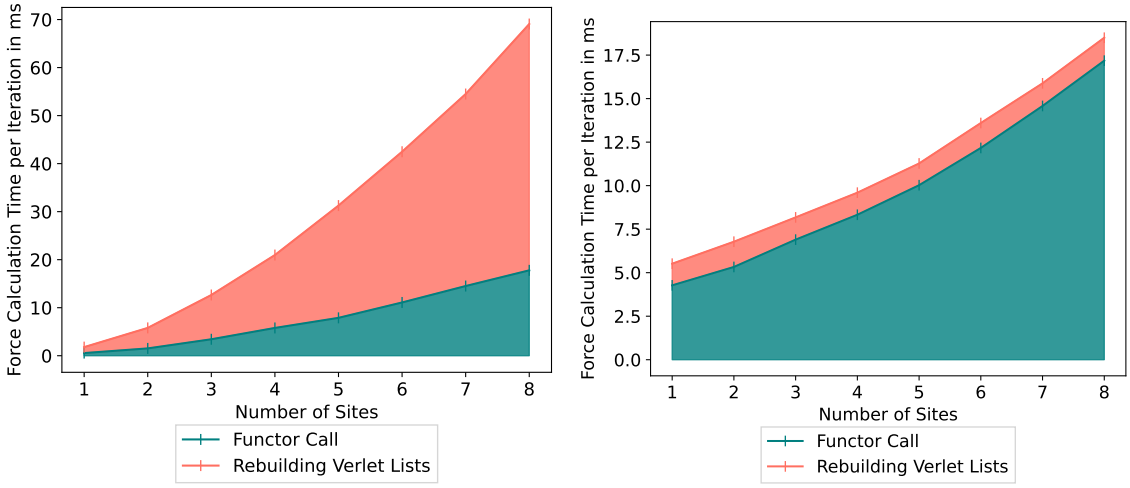
Considering that this phenomenon is independent of the data-layout chosen, it is not surprising that the trend in the runtime comparison graphs of the AoS layout is similar (compare Figure 6.12 and 6.9). However, with this layout, fewer sites per molecule are needed until the molecule-based approach becomes superior. This phenomenon can be explained by the fact that the runtime of the molecule-based approach is actually better with the AoS layout. Additionally, the site-based approach performs slightly worse. The latter observation can be explained by the fact that the interaction partners can lie consecutively in memory even though the use of Verlet Lists does not guarantee this sort of behavior. When the SoA data-layout is used, the site-based approach benefits from these coincidences since it has already loaded the necessary data into the cache. This benefit cannot occur when using the AoS data-layout.



(a) Number of Sites=5, varying Density

(b) Number of Molecules= 6970 (Density ≈ 0.87), varying Number of Sites

Figure 6.9.: Runtime Comparison of Site-based approach and Molecule-based original approach using **Verlet Lists** and **SoA** data-layout



(a) Site-based approach

(b) Molecule-based approach

Figure 6.10.: Runtime decomposition of the force calculation phase using **SoA** data-layout and **Verlet Lists**.

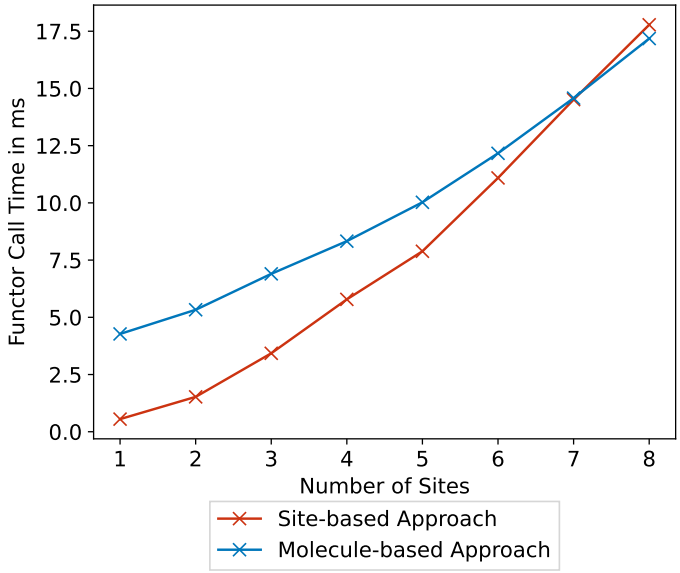
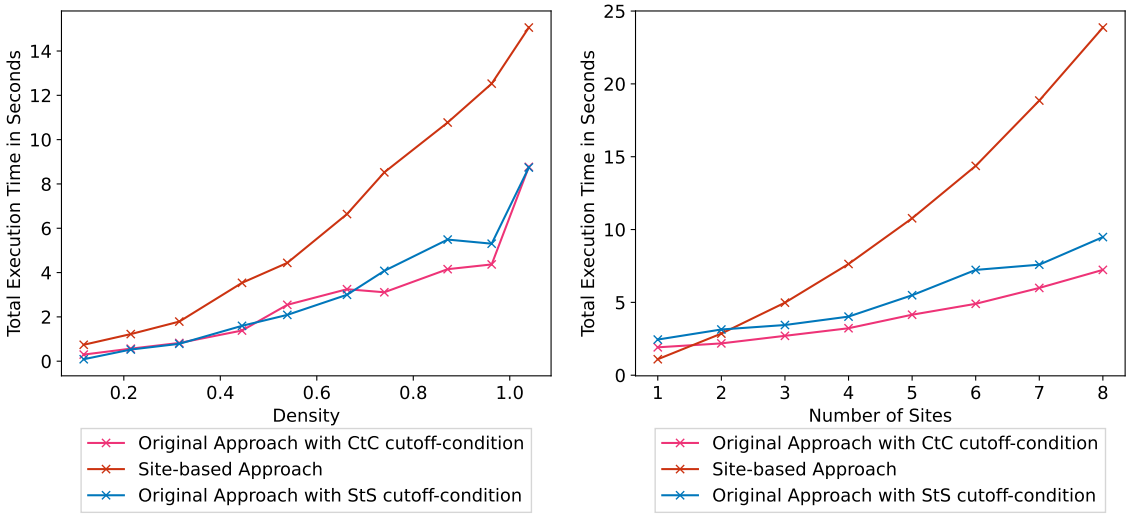


Figure 6.11.: Comparison of **runtime spent in Functor Calls** when using **site-based** approach or **molecule-based** approach at a fixed density of approximately 0.87 and various Numbers of Sites per molecule



(a) Number of Sites=5, varying Density

(b) Number of Molecules= 6970 (Density ≈ 0.87), varying Number of Sites

Figure 6.12.: Runtime Comparison of Site-based approach and Molecule-based original approach using **Verlet Lists** and **AoS** data-layout.

7. Future Work

In this thesis, we identified multiple opportunities to optimize the simulator even further. This chapter highlights the most promising approaches uncovered to advance md-flexible and AutoPas in the future.

7.1. Verlet Cluster Lists

As displayed in Figure 6.10, the primary bottleneck occurring when the site-based approach is combined with Verlet Lists is the increased cost attributed to the construction and maintenance of the lists. This overhead can be reduced significantly by using Verlet Cluster Lists. The efficiency of Verlet Cluster Lists relies on the idea that the Verlet Lists of particles that are closely together are almost identical. As a result, merging those lists and using them as the Verlet List of the entire cluster will not decrease the hit rate significantly. Considering that the sites belonging to the same molecule are very close together in space, this main assumption is valid as long as the sites belonging to the same molecule are grouped in the same cluster. In order to achieve this outcome, the cluster size should always be a multiple of the number of sites within a molecule. If this is not the case, there will be molecules whose sites get split up over multiple Cluster Lists, decreasing efficiency.

Additionally, it should be possible to construct these lists more cheaply, considering that the molecule-ID of each site already marks the sites that should be part of the same cluster. However, this additional piece of information has to be utilized via an extension of AutoPas since md-flexible is not responsible for the construction of the lists. Before committing to a software-architectural change within AutoPas itself, it should be empirically verified whether Verlet Cluster Lists can generally be used to reduce the runtime of the site-based approach without the usage of this additional optimization.

7.2. Hybrid Torque Evaluation

While this thesis only investigated the realization of lazy torque evaluation via site-specific parameters, there are other ways the distributivity of the cross-product can be used to minimize the number of required cross-product computations.

Instead of using site-specific permanent parameters, a temporary helper array can be constructed within each functor call. This array can then be utilized to monitor the forces acting on each site that are computed in this call. Instead of stalling the torque updates until the end of the entire force calculation phase (see Section 2.4), the torque gets updated before the functor deletes this temporary array. Similarly to the hybrid scheme [14, p. 69], this method is reducing the number of required cross-product computations without affecting the memory footprint of the molecules. Another similarity to this scheme is the fact that it

is only effectively applicable to the Linked Cells algorithm and cannot be easily extended to give a performance benefit when using Verlet Lists.

7.3. Vectorization

While this thesis did compare the effectiveness of multi-site molecule representations, it did not explicitly consider the effect of vectorization in that context. However, some representations should benefit considerably more from proper vectorization than others.

For example, the force functor used in the site-based approach is almost identical to a functor utilized in a single-site simulation. Considering that the vectorization of this simulation type has already resulted in high performance increases [17], it is safe to assume that a similarly successful result can be accomplished for the site-based approach as well.

The performance gain achievable via the vectorization of any molecule-based approach utilizing site-specific parameters is limited if the number of sites per molecule is not a multiple of the register size. Since the utilization of lazy torque evaluation did achieve a positive impact on the overall runtime, it would be worth investigating how well this approach can benefit from proper vectorization, given the upper mentioned limitations. Considering that the biggest advantage of the molecule-based approaches in comparison to the site-based approaches is their ability to limit the number of distance checks required, this functor should ensure to take advantage of this phenomenon. The results of section 6.1 imply that the proper utilization of the cutoff condition is critical to runtime performance.

7.4. Extension of the AutoPas Functionalities

This thesis highlighted that, in general, it appears to be suboptimal to permanently store redundant molecule information that can be computed from other variables. However, there is a middle ground between the recomputation of site positions in every functor call and their permanent storage. This goal can be accomplished by using a short-lived auxiliary data structure. Given the current black-box functionality provided by AutoPas, the efficient implementation of such an approach is challenging for reasons similar to the ones highlighted in section 5.2. However, the following functionalities could be added to allow for optimizations like these:

1. AutoPas can increase the runtime transparency about its internal state and the strategy it is about to deploy. A simulator can then build its own data structures before the force calculation commences. Doing so would also require to have information about the particle states and the way they are stored at that time. This approach starkly differs from the current black-box philosophy employed by AutoPas. Considering that a simulator using these features has a significantly higher level of cohesion with AutoPas itself, it also becomes more dependent on the exact implementation of that library. For that reason, this change would complicate further development.
2. AutoPas can alter the functor signatures to also include an auxiliary data structure option for all SoA objects provided. When the functor call realizes that no additional data structure has been built yet, it can choose to do so. Once this functor call returns, AutoPas can provide this already-built structure to other functor calls working on the

same SoA object. Also, AutoPas can delete this structure if it is only needed again far into the future or if it is not needed anymore at all. If AutoPas can guarantee write access to these data structures by a properly chosen traversal, these data structures can also be used to implement lazy torque, so long as the deconstructor of these structures can be defined by the simulator.

These extensions are useful in any type of particle simulation, where more complicated objects should still be considered single entities by AutoPas. Besides multi-site MD, there is an abundance of other use cases, such as the discrete element method (DEM), that fall into this category. Even though the proposed extensions increase software-architectural complexity significantly, providing an adequate option to temporarily store extended particle representations can drastically enhance the viability of AutoPas in other subdisciplines of particle simulations.

8. Conclusion

This thesis investigated the runtime efficiency of different molecule representations in multi-site molecular dynamics. Even though previous work hinted at a potential runtime benefit achievable by explicitly storing the site positions of a molecule, it turned out that this technique actually worsens the runtime in practice. From a theoretical perspective, the explicit site position storage reduces the number of floating point operations required, yet it increases the memory requirements to a degree that nullifies that positive effect.

In contrast, the utilization of lazy torque evaluation has been capable of increasing efficiency for large enough site counts except when Verlet Lists in combination with the SoA data-layout were used. In the other scenarios, the reduction in computational workload is high enough that the increased memory requirements do not overwrite the positive impact. Furthermore, these results highlight that the runtime repercussions of a molecule representation depend on the data-layout used and the algorithm acting on it. The utilization of a different neighbor identification algorithm changes important factors, such as the hit rate, the cache efficiency of memory access patterns, and the ratio of computational workload to memory requirements. Since the choice of a molecule representation includes trade-offs between similar properties, this decision interplays with the neighbor identification algorithm used.

Another molecule representation investigated in this thesis is the site-based approach in which every site is treated as a separate particle during the force calculation process. We explained the adapted control flow required to realize the time integration step with this implementation. Additionally, we showed that this approach can be highly beneficial to the overall runtime when the Linked Cells algorithm and the SoA data-layout is used. It is realizing the explicit site position storage and lazy torque evaluation in a much more memory-friendly way. A runtime decomposition of the force calculation phase showed that the maintenance of Verlet Lists is a major bottleneck when this neighbor identification algorithm is used.

However, this representation remains promising for future work since Verlet Cluster Lists can be employed to reduce this overhead drastically. Considering that the molecule-IDs are already indicating what particles should be within the same cluster, an adapted version of that algorithm should also be capable of reducing overhead costs even further. Additionally, the force computation on this molecule representation can easily be vectorized for an additional performance increase. These factors indicate that the site-based approach can become the predominant solution for multi-site molecular dynamics. However, the biggest disadvantage of this representation is its inability to reduce the number of distance calculations when employing the CtC cutoff condition. As this thesis highlighted, implementing the cutoff condition suboptimally can have detrimental effects on the overall runtime (see Section 6.1).

Therefore, it cannot be said that this implementation is strictly superior to the molecule-based approach.

Ultimately, the efficiency of these molecule representations depends on the degree to which

further improvements can capitalize on their respective advantages. Considering that the site-based approach offers significant potential for further optimization in every aspect but the cutoff condition, it has the potential to strictly outperform the molecule-based approach. Future work is needed to verify whether this assumption turns out to be correct.

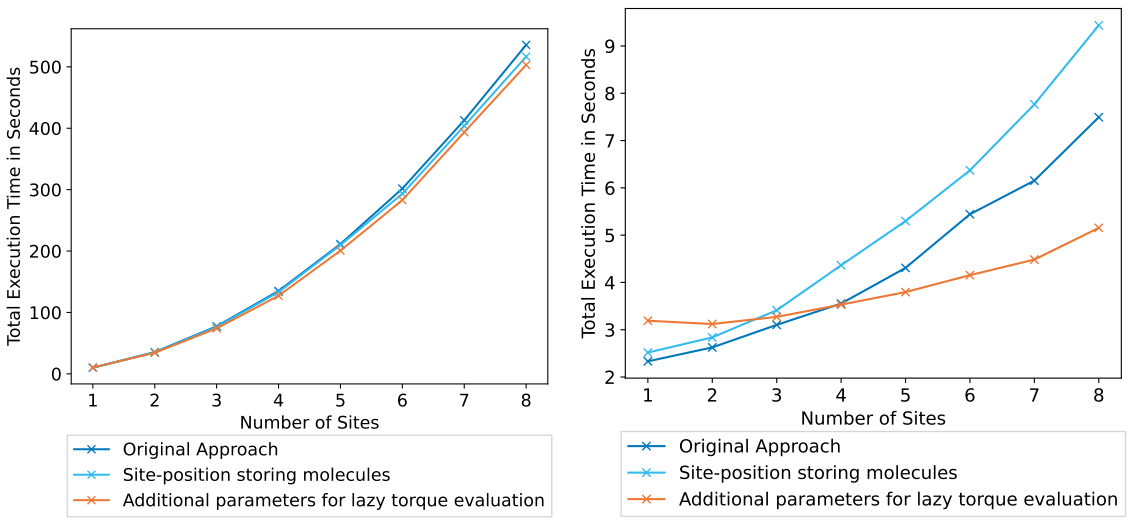
Part I.

Appendix

A. Hardware specifications of the remote laptop

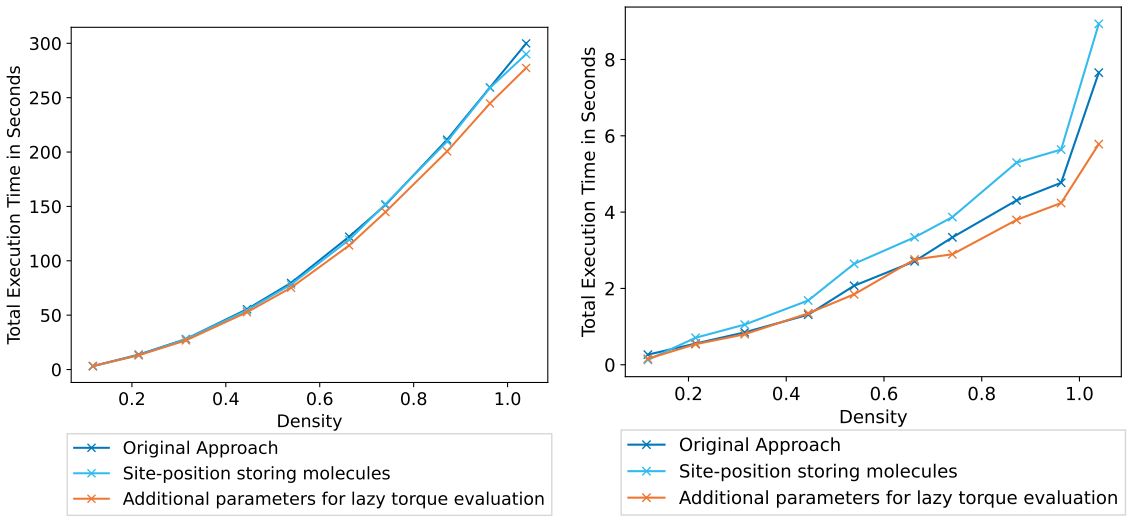
Processor: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz
RAM: 32 GB DDR4
Operating system: Ubuntu 22.04.3 LTS

B. Runtime comparison graphs



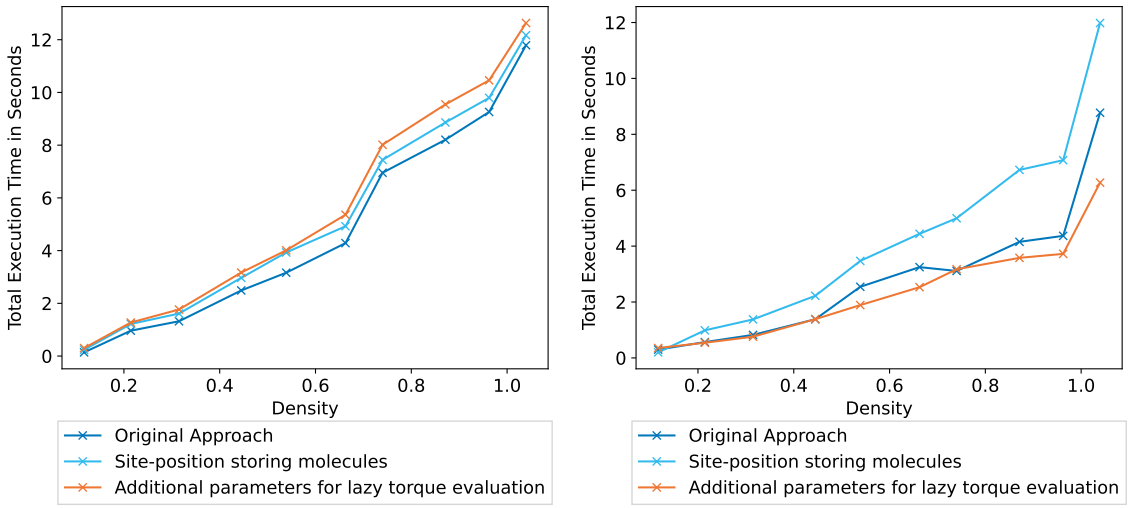
(a) SoA data-layout (b) AoS data-layout

Figure B.1.: Runtime comparison between AoS and SoA with a density of 0.87 and the Linked Cells algorithm



(a) SoA data-layout (b) AoS data-layout

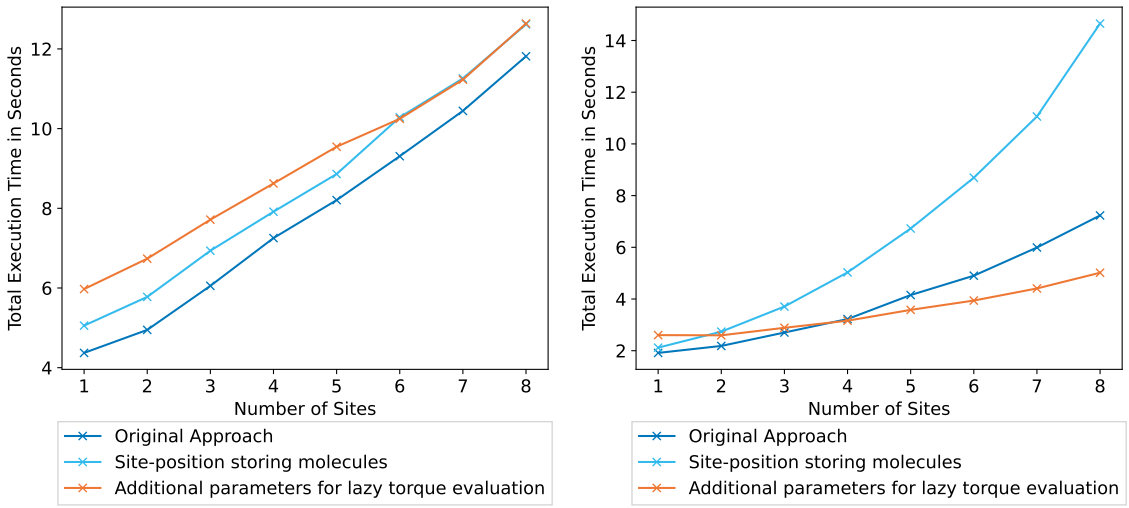
Figure B.2.: Runtime comparison between AoS and SoA data-layout with 5 Sites per molecule and the Linked Cells algorithm



(a) SoA data-layout

(b) AoS data-layout

Figure B.3.: Runtime comparison between AoS and SoA data-layout with 5 Sites per molecule and the usage of Verlet Lists



(a) SoA data-layout

(b) AoS data-layout

Figure B.4.: Runtime comparison between AoS and SoA with a density of 0.87 and the usage of Verlet Lists

Task	**Time in s**	**Runtime** %
Initialization	0.008	0.001%
ForceUpdate	**535.268**	99.923%
PositionUpdate	0.015	0.003%
QuaternionUpdate	0.036	0.007%
UpdateContainer	0.053	0.010%
Boundaries	0.255	0.048%
Vtk	0.055	0.010%
One iteration	5.357	1.000%
Total	535.753s	100.000%

Table B.1.: Runtime decomposition in a simulation utilizing the original molecule representation with a density of 0.87, 8 Sites per molecule and SoA data-layout. These results should highlight, that the force calculation phase can be the predominant factor defining the required runtime.

List of Figures

List of Tables

Bibliography

[1] Outi M. H. Salo-Ahen, Ida Alanko, Rajendra Bhadane, Alexandre M. J. J. Bonvin, Rodrigo Vargas Honorato, Shakhawath Hossain, André H. Juffer, Aleksei Kabedev, Maija Lahtela-Kakkonen, Anders Støttrup Larsen, Eveline Lescrinier, Parthiban Marimuthu, Muhammad Usman Mirza, Ghulam Mustafa, Ariane Nunes-Alves, Tatu Pantsar, Atefeh Saadabadi, Kalaimathy Singaravelu, and Michiel Vanmeert. Molecular dynamics simulations in drug discovery and pharmaceutical development. *Processes*, 9(1), 2021.

[2] Scott A Hollingsworth and Ron O Dror. Molecular dynamics simulation for all. *Neuron*, 99(6):1129–1143, 2018.

[3] Bingyan Cui, Xingyu Gu, Dongliang Hu, and Qiao Dong. A multiphysics evaluation of the rejuvenator effects on aged asphalt using molecular dynamics simulations. *Journal of Cleaner Production*, 259:120629, 2020.

[4] Josh Borrow, Matthieu Schaller, Richard G Bower, and Joop Schaye. Sphenix: smoothed particle hydrodynamics for the next generation of galaxy formation simulations. *Monthly Notices of the Royal Astronomical Society*, 511(2):2367–2389, 11 2021.

[5] Fabio Alexander Gratl, Steffen Seckler, Hans-Joachim Bungartz, and Philipp Neumann. N ways to simulate short-range particle systems: Automated algorithm selection with the node-level library autopas. *Computer Physics Communications*, 273:108262, 2022.

[6] Peter Atkins and Julio Paula. *Atkins' physical chemistry*. Oxford University press, 2008.

[7] J. L. F. Abascal and C. Vega. A general purpose model for the condensed phases of water: TIP4P/2005. *The Journal of Chemical Physics*, 123(23):234505, 12 2005.

[8] Dmitri Rozmanov and Peter G. Kusalik. Robust rotational-velocity-verlet integration methods. *Phys. Rev. E*, 81:056706, May 2010.

[9] Samuel James Newcome, Fabio Alexander Gratl, Philipp Neumann, and Hans-Joachim Bungartz. Towards auto-tuning multi-site molecular dynamics simulations with autopas. *Journal of Computational and Applied Mathematics*, 433:115278, 2023.

[10] David Goldberg. *What Every Computer Scientist Should Know About Floating-Point Arithmetic*. ACM Computing Surveys, 1991.

[11] Qendrim Behrami. Vectorization of the lennard-jones potential for multi-site molecules in autopas, July 2023.

[12] Nikola Tchipev, Steffen Seckler, Matthias Heinen, Jadran Vrabec, Fabio Gratl, Martin Horsch, Martin Bernreuther, Colin W Glass, Christoph Niethammer, Nicolay Hammer, Bernd Krischok, Michael Resch, Dieter Kranzlmüller, Hans Hasse, Hans-Joachim Bungartz, and Philipp Neumann. Twetris: Twenty trillion-atom simulation. *The International Journal of High Performance Computing Applications*, 33(5):838–854, 2019.

[13] Wolfgang Eckhardt, Alexander Heinecke, Reinhold Bader, Matthias Brehm, Nicolay Hammer, Herbert Huber, Hans-Georg Kleinhenz, Jadran Vrabec, Hans Hasse, Martin Horsch, et al. 591 tflops multi-trillion particles simulation on supermuc. In *International Supercomputing Conference*. Springer, 2013.

[14] Wolfgang Eckhardt. *Efficient HPC Implementations for Large-Scale Molecular Simulation in Process Engineering*. PhD thesis, Lehrstuhl für Informatik Schwerpunkt Wissenschaftliches Rechnen, June 2014.

[15] Gromacs documentation - atomproxy class. `https://manual.gromacs.org/current/doxygen/html-lib/classAtomProxy.xhtml`, Accessed Febuary 2024.

[16] Austin J. Barnes, William J. Orville-Thomas, and Jack Yarwood. Molecular liquids : dynamics and interactions. pages 492–500, 1984.

[17] Hiroshi Watanabe and Koh M. Nakagawa. Simd vectorization for the lennard-jones potential with avx2 and avx-512 instructions. *Computer Physics Communications*, 237:1–7, 2019.