SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY MUNICH

Bachelor's Thesis in Informatics

Integrating Three-Body Interactions for Molecular Dynamics Simulation into SimpleMD

Maximilian Mayr

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNICAL UNIVERSITY MUNICH

Bachelor's Thesis in Informatics

Integrating Three-Body Interactions for Molecular Dynamics Simulation into SimpleMD

Integration von Dreikörperwechselwirkungen für die Moleküldynamiksimulation in SimpleMD

Author:	Maximilian Mayr
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisors:	Univ.-Prof. Dr. habil. Philipp Neumann
	Samuel Newcome, M.Sc.
	Amartya Das Sharma, M.Sc.
Date:	16.08.2023

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

Munich, 16.08.2023 Maximilian Mayr

Acknowledgements

First of all, I would like to thank Sam and Amartya, the advisors of this thesis. Thank you for your assistance and valuable feedback throughout these last months. I appreciate the time working together.

Furthermore, I would like to thank the Chair of Scientific Computing and Prof. Dr. Hans-Joachim Bungartz, as well as Helmut Schmidt University's Chair for High Performance Computing and Prof. Dr. Philipp Neumann for enabling this cooperation and giving me the opportunity to work on my thesis in this interesting area of research.

I would like to thank the Leibniz Supercomputing Centre and Helmut Schmidt University's Container-based High Performance Computing Center for providing me with the computational resources and granting me access to CoolMUC-2 and HSUper. Without these resources parts of this work would not have been possible.

Finally, I would like to thank my family and friends who have supported me throughout my entire studies and made studying informatics possible, in the first place.

Abstract

The molecular dynamics simulation tool SimpleMD is, like most other simulation software, based on force calculations between pairs of particles. However, the inclusion of three-body force terms increases the accuracy of the results in certain cases, like the vapor-liquid equilibria of fluids. Adaptive resolution enables the use of different force types in distinct areas of the domain, as required by the specific simulation scenario, for instance to reduce the costs of a computationally expensive three-body simulation. Therefore, it is useful to include both three-body interactions and adaptive resolution, in SimpleMD. The purpose of this thesis is to describe the extension of SimpleMD with three-body interactions and a simple adaptive resolution scheme, which allows for simulations whose domains have a volume solely based on two-body interactions and another volume based on three-body interactions, connected by an interface region that ensures a smooth transition between the regions. A performance analysis of the implemented extensions is conducted to show how well three-body interactions scale in a system originally designed for two-body interactions, and how the performances of two-body interactions, three-body interactions, and different adaptive resolution use cases behave in comparison to each other.

Zusammenfassung

Das Moleküldynamiksimulationsprogramm SimpleMD ist, wie die meisten anderen Simulationsprogramme, auf Kraftberechnungen zwischen Partikelpaaren basiert. Allerdings erhöht die Inklusion von Dreikörperkrafttermen in gewissen Fällen, wie bei Dampf-Flüssigkeit-Gleichgewichten, die Genauigkeit der Ergebnisse. Adaptive Auflösungsverfahren ermöglichen die Benutzung verschiedener Krafttypen in unterschiedlichen Teilen der Simulationsumgebung, was in gewissen Simulationen gewünscht ist, um zum Beispiel den Anteil laufzeitintensiver Dreikörperinteraktionen zu reduzieren. Es ist deshalb nützlich, sowohl Dreikörperwechselwirkungen als auch adaptive Auflösungsverfahren in SimpleMD zu integrieren. Der Zweck dieser Arbeit ist es, die Erweiterung von SimpleMD um Dreikörperwechselwirkungen und ein einfaches adaptives Auflösungsverfahren, das die Simulationsumgebung in einen Bereich in dem nur Zweikörperkräfte wirken und einen anderen in dem nur Dreikörperkräfte wirken aufteilt, wobei durch eine Übergangszone für einen glatten Übergang gesorgt wird, zu beschreiben. Außerdem, wurde eine Performanzanalyse der implementierten Erweiterungen durchgeführt, um zu zeigen, wie gut Dreikörperwechselwirkungen in einem auf Zweikörperwechselwirkungen ausgelegten System skalieren und wie sich die Performanzen von Zweikörperinteraktionen, Dreikörperinteraktionen und verschieden adaptiven Auflösungsverfahren im Vergleich zueinander verhalten.

Contents

Part I.

Introduction and Background

1. Introduction

In the field of molecular dynamics (MD) multi-body potentials are used to compute the forces between particles. In this thesis, the terms *particle* and *molecule* are used interchangeably to denote one of the simulated atoms or molecules. Although the real quantum interactions in a chemical system cannot be reduced to two-body and three-body potentials, these two are considered as the main contributors to the total energy, in most cases. [POM$^+$23]

It is common practice to use effective many-body potentials, like the Lennard-Jones potential, to base MD simulations on. However, these can hide intermolecular influences, for instance the vapor-liquid equilibria of fluids are significantly influenced by three-body interactions. For instance, the contribution of three-body interactions to the liquid phase of argon is considerable. Therefore, the addition of a three-body term, like the Axilrod-Teller potential, to the simulation can result in a considerably higher agreement between experiment and theory. Thus, three-body interactions play an important role in these cases, and hence are indispensable despite being computationally expensive. As shown later in this thesis (see chapter 5.2), their calculation takes remarkably longer than computing two-body interactions. [MTS01, MS99]

The tool SimpleMD is part of the macro-micro-coupling tool MaMiCo[1]. It has been developed and provided by Helmut Schmidt University and Technical University of Munich. Its purpose is to run MD simulations based solely on the Lennard-Jones two-body potential. Hence, there were no three-body interactions available within the tool prior to the current work. [NFA$^+$16]

As part of this work, we extended SimpleMD to support three-body interactions and adaptive resolution between the two force types. We start with a basic description of SimpleMD, explaining all relevant parts for the extension, and then with that backdrop we describe the current work, which includes extending the library with three-body interactions and an adaptive resolution scheme, which works with these three-body interactions and the already existing two-body interactions. Thereafter, we analyze the performance of the different types of force calculation, compare these performances to each other, and investigate how they improve when more computational resources are added.

The structure of this thesis is as follows: Chapter 2 provides the background knowledge used in the other chapters. Chapter 3 describes the structure and functionality of SimpleMD before the extensions were implemented. In Chapter 4, the changes made throughout the implementation process are described. It is shown how the existing structures have been used to realize three-body interactions and adaptive resolution in the system. In chapter 5,

[1] `https://github.com/HSU-HPC/MaMiCo`

the performance of the implementation is analyzed. Here is presented how well the program scales when parallelization is applied and how the execution times of a three-body simulation, a two-body simulation, and adaptive resolution simulations behave in comparison to each other. Chapter 6 describes how SimpleMD can be further enhanced and which aspects of its performance need to be investigated in the future.

2. Theoretical Background

2.1. Molecular Dynamics Simulation

A MD simulation simulates the development of a system over time, by tracking the particle positions and interactions. Therefore, the observed time interval is split in (usually very small) time steps. In each time step, the forces between the molecules are determined by calculating the partial derivatives of their potential energy. After this, the molecules' positions are updated according to a time integration algorithm.

The following formula shows the theoretical force calculation for total force taking effect on molecule i. The formula consists of its own potential energy and the potentials of the many-body interactions involving particle i and all other particles $j, k, \ldots$ with $i \neq j \neq k \neq \ldots$, used in the simulation. Thus, theoretically, up to n-body potentials have to be evaluated for a simulation containing n molecules.

$$\mathbf{F}_i = -\nabla(\phi_1(i) + \sum_{\substack{j \\ j \neq i}} \phi_2(i, j) + \sum_{\substack{j,k \\ j \neq k \neq i}} \phi_3(i, j, k) + \ldots)$$

In the formula, ∇ is the gradient of the sum of all potential energies affecting i and ϕ_k is the function describing the k-body potential energy. [KY14] This thesis only focuses on pairwise ($k = 2$) and three-body interactions ($k = 3$).

The particles are located and move in the simulation area, which is called a *domain*. When particles are close to the boundaries of the domain or leave the domain, they have to be treated specially, according to the boundary condition of this boundary. In case there is no boundary condition, a particle that has left the domain is removed from the simulation. Another possibility is the use of open boundaries. When they are in use, external forces are applied to the particles to simulate the real forces between them and particles outside of the domain. If periodic boundaries are used, the particle leaving the domain reenters it on the opposite side. This simulation of an infinite space additionally requires force computations between particles on the opposite sides of the domain. In SimpleMD, the boundary condition of the boundaries of distributed memory parallelization subdomains, which border other subdomains, are called parallel boundary conditions. They resemble periodic boundaries but interact with other subdomains instead of the domain's opposite side. Another method, which, in contrast to the others, will be covered throughout this thesis, are reflecting boundaries. The reflection is applied in a way that at the moment the particle hits the border, its velocity in this direction is reversed and potentially rescaled. [NW20, LMV16]

2.2. Time Integration

The purpose of the time integration is to update the molecules' positions according to the forces influencing them, resulting from the force computations, and their momenta, and thus, simulate the movements of the molecules in the time step.

In SimpleMD, the updates of the position and velocity of each molecule are performed with the velocity form of the Verlet algorithm introduced in [SABW82]. First the velocity of the current time step n is computed using the forces of the current and previous time step, the velocity computed in the previous time step $n-1$, the length of one time step δt, and the molecule's mass m.

$$\mathbf{v}_n = \mathbf{v}_{n-1} + \tfrac{\delta t}{2 \cdot m} \cdot (\mathbf{F}_n + \mathbf{F}_{n-1})$$

The calculated velocity is immediately used, along with the force, the length of one time step, and the mass, to calculate the new position of the molecule, which is used in the next time step.

$$\mathbf{x}_{n+1} = \mathbf{x}_n + \delta t \cdot (\mathbf{v}_n + \tfrac{\delta t}{2 \cdot m} \cdot \mathbf{F}_n)$$

2.3. Force Computations

2.3.1. Short-Range and Long-Range Forces

We can distinguish between short-range forces[1], like the Lennard-Jones force and the Axilrod-Teller force, and long-range forces, such as Coulomb forces or gravitational forces, depending on how fast the function decays with increasing distances. Hence, if a short-range potential is used, particles only exert significant forces on each other when the distance between them is small. Thus, the force calculations can be omitted for particles with a large enough distance, which allows for more efficient force computations. For instance, the computation of two-body forces only requires $\mathcal{O}(n)$ calculations of partial forces per time step instead of $\mathcal{O}(n^2)$. Potentials decaying faster than r^{-d} are considered short-range, otherwise they are long-range potentials, where r is the distance between the particles and d is the number of spatial dimensions of the simulation domain. [GZK07]

Since SimpleMD only works with short-range potentials, only these are described in the next chapters.

2.3.2. Cutoff Radius

When using short-range potentials, a cutoff radius can be used to improve the efficiency. This approach enables omitting the force calculations of the majority of molecule-pairs and molecule-triplets[2] because their impact is insignificant for the result. In the thesis the cutoff radius is denoted by r_{cutoff}.

[1]The same differentiation applies to the corresponding potentials.
[2]In the context of this thesis, a group of three molecules is called *triplet*.

For pairwise interactions the cutoff radius is exceeded when the distance between the particles is larger than r_{cutoff}.

However, for three-body interactions, there are different variations of cutoff radii that can be applied. Some examples are:

1. The distances between all pairs of particles of the triplet have to be within r_{cutoff}. [Mar01]
2. The sum of all distances between the respective particles and the center of mass is less than r_{cutoff}. [O'S13]
3. Both other particles j and k lie within the cutoff radius to particle i. No requirements about the distance between j and k exist. However, this breaks the symmetry when the distance between j and k is larger than r_{cutoff}, because then the force only gets applied to particle i. [CW00]
4. Variant 3 can also be applied symmetrically, with the force affecting all particles, regardless of the distance between j and k. [KKN+13]

The previously discussed periodic boundary condition can imply that particles on the opposite side of the domain are within the cutoff distance.

2.3.3. Two-Body Force

Since SimpleMD is only based on two-body interactions, the force computations determine the forces between pairs of molecules. Therefore, SimpleMD uses the Lennard-Jones 12-6 potential to calculate the two-body forces between the particles.

$$\phi_2 = 4 \cdot \epsilon((\tfrac{\sigma}{r_{ij}})^{12} - (\tfrac{\sigma}{r_{ij}})^6)$$

In the formula, r_{ij} denotes the distance between the two particles i and j, and ϵ and σ are the two Lennard-Jones parameters.

A cutoff radius ensures that only forces between particles which are closer than the cutoff radius are calculated. The force $\mathbf{F}_{ij}$ between the particles i and j is computed by deriving the potential energy between these molecules with respect to $\mathbf{x}_{ij}$, the vector from particle i to particle j.

$$\mathbf{F}_{ij} = -\tfrac{\partial \phi_2(i,j)}{\partial \mathbf{x}_{ij}}$$

By using $\frac{\partial r_{ij}}{\partial \mathbf{x}_{ij}} = \frac{\mathbf{x}_{ij}}{r_{ij}}$, the formula results in:

$$-\tfrac{\partial \phi_2(i,j)}{\partial \mathbf{x}_{ij}} = -\tfrac{\partial r_{ij}}{\partial \mathbf{x}_{ij}} \cdot \tfrac{\partial \phi_2(i,j)}{\partial r_{ij}} = -\tfrac{\mathbf{x}_{ij}}{r_{ij}} \cdot \tfrac{\partial \phi_2(i,j)}{\partial r_{ij}}$$

Thus, the resulting formula for the force is:

$$\mathbf{F}_{ij} = -24 \cdot \tfrac{\epsilon}{r_{ij}^2} \cdot (\tfrac{\sigma}{r_{ij}})^6 \cdot (1 - 2 \cdot (\tfrac{\sigma}{r_{ij}})^6) \cdot \mathbf{x}_{ij}$$

However, the force on particle i, resulting from this pair of particles, is calculated according to

$$\mathbf{F}_i = -\frac{\partial \phi_2(i,j)}{\partial \mathbf{x}_i}.$$

Hence, the ansatz has to be extended like this, where $\mathbf{I}$ is the identity matrix:

$$-\frac{\partial \phi_2(i,j)}{\partial \mathbf{x}_i} = -\frac{\partial \mathbf{x}_{ij}}{\partial \mathbf{x}_i} \cdot \frac{\partial r_{ij}}{\partial \mathbf{x}_{ij}} \cdot \frac{\partial \phi_2(i,j)}{\partial r_{ij}} = -(-\mathbf{I}) \cdot \frac{\mathbf{x}_{ij}}{r_{ij}} \cdot \frac{\partial \phi_2(i,j)}{\partial r_{ij}} = \frac{\mathbf{x}_{ij}}{r_{ij}} \cdot \frac{\partial \phi_2(i,j)}{\partial r_{ij}}$$

Thus, the force on particle i is $\mathbf{F}_i = -\mathbf{F}_{ij}$. (Ansatz from [Mar01])

2.3.4. Three-Body Force

The aim of this work is to integrate three-body interactions and adaptive resolution into SimpleMD. For the calculation of forces between the triplets of particles, we used the Axilrod-Teller potential, involving any three particles i, j, and k. The formula is

$$\phi_3 = \nu \cdot \frac{1+3\cdot\cos\theta_i\cdot\cos\theta_j\cdot\cos\theta_k}{(r_{ij}\cdot r_{ik}\cdot r_{jk})^3},$$

where θ_i is the angle between $\mathbf{x}_{ij}$ and $\mathbf{x}_{ik}$, and ν is a positive parameter, depending on the ionization energy and the dipole polarizability of the particles. [AT04]

The potential can be rewritten as

$$\phi_3 = \nu \cdot \left(\frac{1}{(r_{ij}\cdot r_{ik}\cdot r_{jk})^3} + \frac{3\cdot(-r_{ij}^2+r_{ik}^2+r_{jk}^2)\cdot(r_{ij}^2-r_{ik}^2+r_{jk}^2)\cdot(r_{ij}^2+r_{ik}^2-r_{jk}^2)}{8\cdot(r_{ij}\cdot r_{ik}\cdot r_{jk})^5}\right)$$

by application of the cosine law. The latter version of the formula can be used immediately by the program without calculating the angles resulting from the positions prior.

Using the same ansatz as in chapter 2.3.3, the force $\mathbf{F}_{ij}$, affecting particles i and j, can be calculated as shown below.

$$\mathbf{F}_{ij} = \frac{3\cdot\nu}{8\cdot r_{ij}} \cdot \Big(-\frac{8}{r_{ij}^4\cdot r_{ik}^3\cdot r_{jk}^3} - \frac{1}{r_{ik}^5\cdot r_{jk}^5} + \frac{5\cdot r_{ik}}{r_{ij}^6\cdot r_{jk}^5} + \frac{5\cdot r_{jk}}{r_{ij}^6\cdot r_{ik}^5} - \frac{1}{r_{ij}^2\cdot r_{ik}^3\cdot r_{jk}^5} - \frac{1}{r_{ij}^2\cdot r_{ik}^5\cdot r_{jk}^3} - \frac{3}{r_{ij}^4\cdot r_{ik}\cdot r_{jk}^5} - \frac{3}{r_{ij}^4\cdot r_{ik}^5\cdot r_{jk}} - \frac{5}{r_{ij}^6\cdot r_{ik}\cdot r_{jk}^3} - \frac{5}{r_{ij}^6\cdot r_{ik}^3\cdot r_{jk}} + \frac{6}{r_{ij}^4\cdot r_{ik}^3\cdot r_{jk}^3}\Big) \cdot \mathbf{x}_{ij}$$

The remaining two forces $\mathbf{F}_{ik}$ and $\mathbf{F}_{jk}$ can be calculated analogously. In the formula, all indices ij and ik, or ij and jk respectively have to be exchanged.

For the same reason as described in chapter 2.3.3, $\mathbf{F}_{ij}$ affects $\mathbf{F}_i$ negatively. However, $\mathbf{F}_{ik}$ affects the force of particle i as well. Thus, the total force of particle i resulting from this triplet is $\mathbf{F}_i = -\mathbf{F}_{ij} - \mathbf{F}_{ik}$. [Mar01]

2.3.5. Newton's Third Law of Motion

Newton's third law of motion states that "the mutual actions of two bodies upon each other are always equal, and directed to contrary parts," [New50] or in other words: Whenever a force acts from one body on another, a force with the same magnitude acts in the opposite direction as well.

After applying the law on two-body forces according to chapter 2.3.3, it holds:

$$\begin{gathered}\mathbf{F}_{ij} = -\mathbf{F}_{ji} \text{ and} \\ \mathbf{F}_i = -\mathbf{F}_j\end{gathered}$$

The application of the law on the three partial forces affecting two particles described in chapter 2.3.4 results in the following formulas for forces of the molecules j and k resulting from the triplet.

$$\begin{gathered}\mathbf{F}_j = \mathbf{F}_{ij} - \mathbf{F}_{jk} \\ \mathbf{F}_k = \mathbf{F}_{ik} + \mathbf{F}_{jk}\end{gathered}$$

Furthermore, Newton's third law can be applied on the entire triplet as described by [Mar01]:

$$\mathbf{F}_i + \mathbf{F}_j = -\mathbf{F}_k$$

However, this symmetry is not usable if a variation of the cutoff radius, like the third variant mentioned in chapter 2.3.2, is used, where the force is not applied to all particles.

2.4. Adaptive Resolution

Sometimes, it is desired to use different potentials in different areas of the domain.[3] A naive approach would be to only consider the forces resulting from the potential relevant in the volume of the domain the particle is located in. However, this would result in neighboring molecules along the border being treated completely differently, which is unrealistic in most cases. Therefore, an interface layer is embedded between the areas, which use only one type of potential, to allow for a smooth transition.

In this interface layer, both types of forces have a partial effect on the total force of one particle. The impact of each partial force is determined by a weighting function w. In order to achieve a smooth transition, the weighting function should ideally fulfill the following conditions.

1. $w : [x_0, x_n] \to [0, 1]$ where x_0 is the start of the interface region and x_n is its end.
2. $w(x_0) = 0$ and $w(x_n) = 1$ or vice versa, depending on the alignment of two-body and three-body areas.

[3]In this thesis, adaptive resolution is only covered for two different potentials used in the domain, although more are possible.

3. w is monotonic, continuous, differentiable and has a zero slope at the boundaries of the interface layer.

Functions fulfilling these conditions can, for instance, be based on sines and cosines.

The total potential ϕ of a particle is the weighted sum of its partial potentials. In the thesis the partial potentials are always the Lennard-Jones potential ϕ_2 and the Axilrod-Teller potential ϕ_3. In this chapter the reference to the specific particles is omitted. The calculation of the total force works analogously. The actual formula depends on how the weighting function is defined, precisely on whether the result is the weight of the two-body term or the three-body term. In the formulas below, w outputs the weight of the two-body parts depending on the position x of the particle.

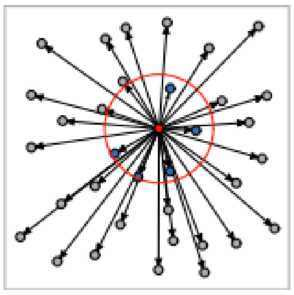
$$\phi = w(x) \cdot \phi_2 + (1 - w(x)) \cdot \phi_3$$
$$\mathbf{F} = w(x) \cdot \mathbf{F}_2 + (1 - w(x)) \cdot \mathbf{F}_3$$

On the other hand, if w's results are the weights of the three-body areas, the formulas are the same with reversed indices. [PDSK05]

2.5. Particle Containers

The domain of a MD simulation can be decomposed in subdomains for multiple reasons. One of them, which will not be subject of this thesis, is decomposing the domain into subdomains for parallelization purposes, so that every process can work on its own subdomain, which is especially useful for distributed memory parallelization.

Another reason for decomposing the domain is to improve the performance by reducing the number of distance checks. For this purposes, the domain is decomposed in particle containers in a way, that the majority of particles outside of particle i's cutoff radius are not even considered for the force computations to avoid unnecessary distance evaluations. In the following chapters, two types of particle containers, which are all illustrated in figure 2.1, are described. There are other types, like Verlet lists and Verlet cluster lists, as well. However, they are not subject of this thesis and are described in [GSBN21].


(a) Direct Cutoff

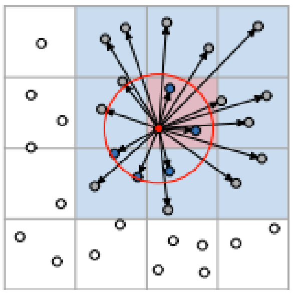

(b) Linked Cells

Figure 2.1.: The two particle container types described in the sections below. In all figures, the forces involving the red particle are calculated. The red circle is r_{cutoff}. All gray particles are part of distance checks and all blue particles are part of force calculations. The remaining particles are ignored. Source: [GSBN21]

2.5.1. Direct Cutoff

The naive approach, shown in figure 2.1a, is to make no prior decomposition. This is called *direct cutoff*. It leads to $\mathcal{O}(n^2)$ distance checks because every molecule has to evaluate the distances to all remaining $n-1$ molecules when two-body interactions are used. [GSBN21]

2.5.2. Linked Cells

Other methods, like the linked cell approach, are used to reduce the number of necessary distance checks. The linked cell approach normally divides the domain in a regular Cartesian grid of so-called *linked cells*, which is illustrated in figure 2.1b. The mesh width has to be at least as big as r_{cutoff} and can vary in the different directions, although the cells are usually squares or cubes. The particles are sorted into the cells according to their position. The particles within each cell now only have to evaluate the distance between them and all particles in the neighboring cells. This reduces the number of distance checks to $\mathcal{O}(n)$ if the particles are distributed homogeneously and two-body interactions are used. [GSBN21]

SimpleMD uses a linked cell approach to improve performance by reducing the number of distance checks.

2.6. Scalability

In high performance computing (HPC), it is possible to use a large amount of resources to solve problems. The concept of scalability can be used to measure the ability of software to increase the computational power when more resources are provided.

The primary criterion for indicating the scalability of software is the *speedup*. It can be calculated as

$$speedup(n) = \frac{T_1}{T_n}$$

where n is the number of processors, that the speedup should be evaluated for, and for any number of processes i, T_i is the execution time of the software using this many processes.

Ideally, the speedup for n processors is equal to n. However, achieving such a speedup is unrealistic because it would require a perfectly equal workload among all processors and the absence of any communication between the processors, which is not attainable in realistic applications. [Li18]

2.6.1. Strong Scaling

Strong scaling measures the speedup of the software, in the case where more resources are used to solve the same problem size. According to Amdahl, the theoretical upper limit of the possible speedup is determined by the share of the program that cannot be executed parallelly. [Amd67] The formula widely used to describe Amdahl's law is

$$speedup_{max}(n) = \frac{1}{s+\frac{p}{n}}$$

where n is the number of processes, s is the proportion of the execution time that is used to compute the parts that cannot be parallelized, and p is the remaining proportion of the execution time.

As a result, the more resources are added, the lower is the relative improvement, given that realistically $s \neq 0$.

However, the real speedups are usually lower than proposed by Amdahl's law because an increase in computational units leads to an increase in the serial share due to maintenance and communication. [Li18]

2.6.2. Weak Scaling

Weak scaling measures the speedup of the software, where more resources are used to solve the same problem size per computational unit. Gustafson proposed, that it is more reasonable to assign higher computational power to bigger problem sizes instead of spending them on smaller problems, where the limits pointed out by Amdahl have a more severe effect. Gustafson's law for the scaled speedup is

$$speedup_{scaled}(n) = s + p \cdot n$$

where n, s, and p represent the same proportions as in Amdahl's law. [Gus88]

Again, an increasing number of processes leads to a higher value of s. Thus, the actual scaled speedup usually has a logarithmic behavior and not the linear characteristic described in the formula. [Li18]

In the performance measurements, the speedup of the weak scaling computation times is calculated according to the formula in the beginning of chapter 2.6. The relation $\frac{T_1}{T_n}$ is normally lower then 1 for weak scaling. Therefore, the relation is called *efficiency*, in this context.

3. Structure and Functionality of SimpleMD

3.1. MaMiCo

The macro-micro-coupling tool MaMiCo[1] is a software which allows for coupling of arbitrary continuum and molecular dynamics solvers. It supports several MD solvers, like the Lennard-Jones based SimpleMD[2] and the node-level optimized ls1-MarDyn[3]. [JN19, NFA$^+$16]

Additionally, there are utility functions, which will be described in the next chapter, in the subdirectory tarch. The two features primarily used in SimpleMD are the configuration functions and vectors.

The code is written in C++ and the Makefile and other compile instructions are created by CMake. Important compile time decisions, like the simulation to be used, the number of dimensions, whether the simulation should be parallelized with OpenMP and/or MPI, and more, are set in the compile options. Therefore, a different combination of compile options generates a different program.

3.2. External Utility

Utility functions that can be used in all parts of MaMiCo are located in the technical architecture directory *tarch*[4]. It contains the folder tinyxml2 which provides functions to read and write on XML files and is used by the `Configuration` classes that parse XML files. The subdirectory *la* contains vector and matrix classes. Finally, there are also functions for using multiple MD simulations parallelly, prevent copying, and generating random numbers.

The two utility functions used in SimpleMD, the configurations (and, therefore, also tinyxml2) and vectors, are described in detail in the following parts.

3.2.1. Vectors

`Vector` and `Matrix` classes are located in the linear algebra subdirectory *la*[5]. However, since matrices are not used in SimpleMD, they will not be discussed here.

[1] `https://github.com/HSU-HPC/MaMiCo`
[2] `https://github.com/HSU-HPC/MaMiCo/tree/master/simplemd`
[3] `https://github.com/ls1mardyn/ls1-mardyn`
[4] `https://github.com/HSU-HPC/MaMiCo/tree/master/tarch` (tarch is copied from the software Peano `https://gitlab.lrz.de/hpcsoftware/Peano/-/tree/p4/src/tarch`)
[5] `https://github.com/HSU-HPC/MaMiCo/tree/master/tarch/la`

The generic class `Vector` has the two template parameters `size` and `T`. During the initialization, a vector with as many elements as defined in `size` of type `T` is created.

Some math, logic and access functions are achieved by using operator overloading. All functions can only be used on two vectors of the same size and type or on one vector of arbitrary size and a scalar of the same type as the elements of the vector.

The most important functions are the addition and subtraction of two vectors using `+` and `-`, as well as `+=` and `-=` with the same functionality and the first input variable being the output variable. Moreover, `*` multiplies a vector and a scalar and `dot` computes the dot product of two vectors.

3.2.2. Configuration

The input of SimpleMD is passed in one XML file that contains all data to set up the simulation in form of key-value-pairs and another file containing the information about the position, velocity, and force of all molecules. The parsing of the XML file is handled with the configuration utility functions.

There are two classes in the subfolder *configuration*[6], `Configuration` and `ParseConfiguration`. All configuration classes of SimpleMD inherit the structure of the class `Configuration`. Its most important method is `parseSubtag`, which parses all key-value-pairs of one subtag of the XML file and stores them in the local attributes defined by the child class. The class `ParseConfiguration` provides the necessary functions to parse a XML file.

3.3. Force Calculation and Time Integration

3.3.1. Force Calculation

SimpleMD uses the Lennard-Jones potential to calculate the forces between the molecules. A cutoff radius ensures that only forces between molecules which are closer than the cutoff radius are calculated. The values for the cutoff radius r_{cutoff} and the Lennard-Jones parameters ϵ and σ are values of the XML file. The force $\mathbf{F}_i$ of molecule i, resulting from the interaction between any two molecules i and j, is computed as described in chapter 2.3.3.

The calculated force is added to the force buffer, which stores the total sum of forces during one time step, calculated so far, of molecule i and subtracted from the buffer of molecule j, according to Newton's third law of motion.

3.3.2. Time Integration

The time integration is performed with a velocity Verlet algorithm as described in chapter 2.2.

[6] `https://github.com/HSU-HPC/MaMiCo/tree/master/tarch/configuration`

In case, after the time integration one molecule's position is outside of the (sub-)domain, measures have to be taken. In some cases, which will not be discussed in this thesis, the respective molecule has to be sent to another process or the same process. However, if the boundaries of the simulation are reflecting and the molecule has moved outside of the whole global domain, a reflection is applied. The molecule's position is mirrored on the boundary, in a way that the distance between the boundary and the coordinate, in this direction d, outside the domain equals the distance between the boundary and the new coordinate in the domain. The velocity in this direction is reversed.

$$x_{d,inside} = x_{d,outside} + 2 \cdot (x_{d,domain} - x_{d,outside})$$
$$v_{d,inside} = -v_{d,outside}$$

However, during the initial time step[7] the velocity computation does not happen. The velocity in the formula of the positional update is replaced by the velocity v_{init}, passed in the file containing the positions, velocities, and forces of all molecules. The formula for this time step, which resembles the formula of the first time step of a classic Störmer-Verlet algorithm [Ver67], is

$$\mathbf{x}_1 = \mathbf{x}_{init} + \delta t \cdot (\mathbf{v}_{init} + \tfrac{\delta t}{2 \cdot m} \cdot \mathbf{F}_0).$$

Additionally, no reflection is applied, during this time step.

3.4. Linked Cell Approach

SimpleMD uses a linked cell approach to decompose the domain in particle containers. The size of each linked cell is defined in the XML file. If the domain size is not divisible by the size of the linked cell in a dimension, depending on the dimension, the last cell on the right, back, or top side of the domain is smaller than the rest. Normally, the size in each dimension is identical and at least as big as the cutoff radius, which is also defined in the XML file.

Coordinates

Each linked cell can be identified within the local domain by its index and its coordinates. The coordinates are a tuple with as many elements as the dimensions of the simulation. The coordinates will be represented as (x, y), in the two-dimensional case, and (x, y, z), in the three-dimensional case. The index can be calculated according to the respective formula

$$index = x + y \cdot cells_x$$
$$index = x + y \cdot cells_x + z \cdot cells_x \cdot cells_y$$

where $cells_x$ and $cells_y$ are the total number of cells, including all ghost layers, in the x-direction and y-direction respectively. The first version of the equation is used for two-dimensional simulations and the second one for three-dimensional simulations.

[7] In this time step all objects are created and only the force calculation, computing $\mathbf{F}_0$, and time integration, computing $\mathbf{x}_1$, happen.

Visibility

To avoid multiple calculations of the same forces, the interactions between the cells only happen in a controlled manner. Each cell can only interact with cells that have an equal or higher coordinate in each dimension. If one cell can interact with another, we call the other cell visible to the previous cell. The way this interaction control works guarantees that if $cell_A$ is visible to $cell_B$, $cell_B$ is always invisible for $cell_A$. However, it is also possible that two cells which need to interact with another are both invisible to each other. In this case a third cell which can see both cells needs to initiate the interaction. This phenomenon occurs during the force calculations and is described in further detail in the next part.

Force Calculations

SimpleMD expects the cell size to be at least as large as the cutoff radius. In that case, all molecules outside of a linked cell and outside of all 8 neighboring cells, in the two-dimensional case, or all 26 neighboring cells, in the three-dimensional case, have a larger distance to molecules inside the central cell than the cutoff radius. Thus, the forces between two such molecules do not need to be calculated and two linked cells which do not neighbor each other do not interact with each other during the force calculations.

In the process of force computations, all linked cells are traversed. When a cell is handled, first, the forces between all molecule-pairs in the cell are calculated. After that, the force computations for all cell-pairs which are visible and need to be calculated by the handled cell are initiated. The cell-pairs whose force calculations need to be initiated by a cell are described below. During the latter step, forces are calculated for every molecule-pair with one molecule being located in each cell, respectively.

A cell always needs to initiate cell-pair calculations if the following two conditions are met.

1. Both cells are visible to the currently handled cell.
2. There is no other cell with an equal or higher coordinate in each dimension that can see both cells, as well.

Two-Dimensional Force Calculations

In the two-dimensional case, each cell has to initiate the computations for 4 cell-pairs. These are the three interactions between the initiating cell and its right[8], top[9], and top-right neighbors. Additionally, the force calculation between the molecules in the neighboring cells on the right and top needs to be triggered, as well, because the top neighbor is invisible to the right neighbor due to a lower x-coordinate and the vice versa due to a lower y-coordinate. The second condition is fulfilled because the initiating cell is in the top-right corner of the subdomain containing all cells that can see both cells.

[8]in positive x-direction
[9]in positive y-direction

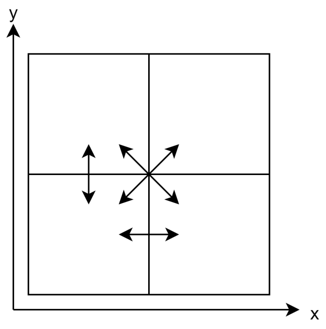


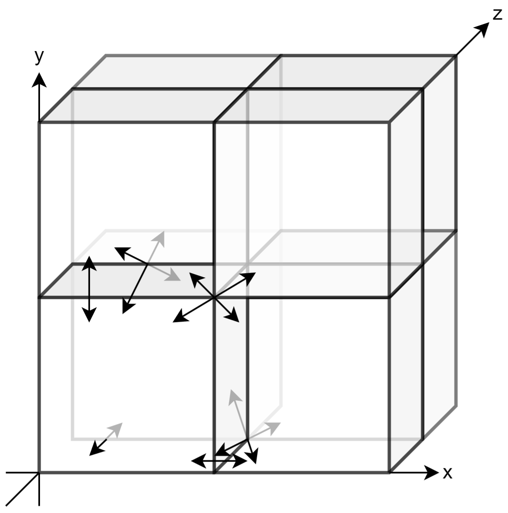
Figure 3.1.: Force calculations that need to be initiated by the bottom-left cell.

The remaining 5 computations including the handled cell are triggered by its left, bottom, and left-bottom neighbor. Each of these cells initiates the calculating including itself. The bottom cell also sets off the interaction with the bottom-right neighbor and the left cell sets off the interaction with the left-top neighbor.

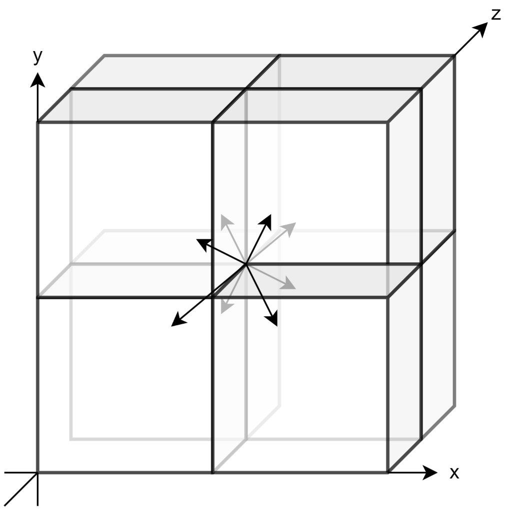
Three-Dimensional Force Calculations

In the three-dimensional case, each cell has to initiate 13 cell-pair computations. These contain the 4 interactions of the two-dimensional case in the xy-plane and also their equivalents in the xz-plane and yz-plane. This results in a total of 9 interactions because the pairings of the handled cell and its right, top, and back[10] neighbor are part of two planes, but only computed once per time step. The remaining four pairs are between the handled cell and its top-right-back neighbor, between the right and the top-back neighbor, between the top and the right-back neighbor, and between the back and the top-right neighbor. For all of these pairs, the handled cell is, again, in the top-right-back corner of the subdomain containing the cells which are able to see both cells.

[10]in positive z-direction



(a) Force calculations between molecules whose cells are in the bottom-left-front cell's xy-plane, xz-plane, and yz-plane.



(b) Remaining force calculations.

Figure 3.2.: Force calculations, initiated by the bottom-left-front cell.

The remaining 19 cell-pairs including the handled cell are handled by its 7 neighbors on the left-bottom-front side of the $3 \times 3 \times 3$ cell-cube with the handled cell in the middle.

3.5. Structure of SimpleMD

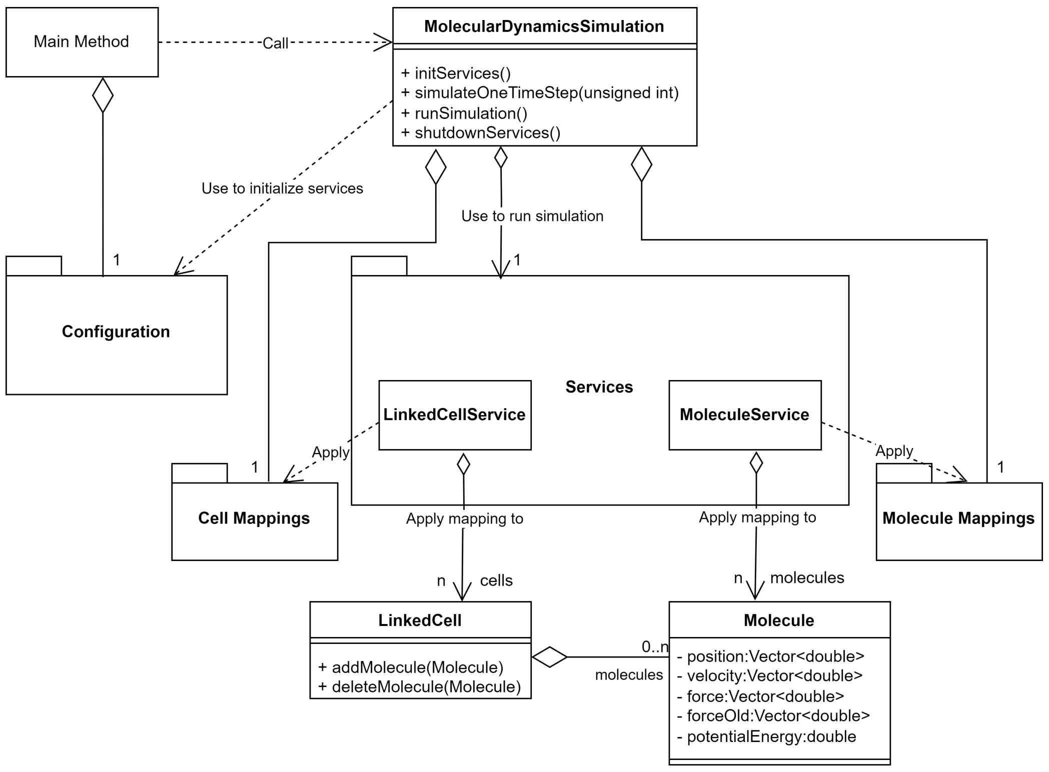


Figure 3.3.: Simplified class diagram of SimpleMD, containing the most relevant classes of the main directory and the subdirectories as packages. All vectors of the class Molecule have the number of dimensions as size. Irrelevant attributes and methods, as well as getter and setter methods are omitted.

SimpleMD consist of a main directory and 4 subdirectories. The main directory includes a file providing the `main` method, files defining constants, the classes for molecules and linked cells, and the simulation. The subdirectories are *cell-mappings*[11] and *molecule-mappings*[12], which provide the functions applied on a cell level and molecule level respectively, *configurations*[13], which is responsible for the parsing of the XML file, during the initialization of the system, and *services*[14], which provides services to operate on linked cells and molecules that are used by the simulation class. The following sections describe the most important parts of SimpleMD. A more detailed description is provided in appendix A.1.

3.5.1. Main Directory

Main Method: The main method is located in `main.cpp`. It sets up, runs, and shuts down one MD simulation.

[11]`https://github.com/HSU-HPC/MaMiCo/tree/master/simplemd/cell-mappings`
[12]`https://github.com/HSU-HPC/MaMiCo/tree/master/simplemd/molecule-mappings`
[13]`https://github.com/HSU-HPC/MaMiCo/tree/master/simplemd/configurations`
[14]`https://github.com/HSU-HPC/MaMiCo/tree/master/simplemd/services`

Constants: Global constants are defined in the files `MolecularDynamicsDefinitions.h` and `MolecularDynamicsUserInput.h`. Some of these constants are used in preprocessor directives.

MolecularProperties: The class stores the properties that all molecules have in common.

Molecule: The class contains all properties that differ from molecule to molecule. Some of these are the position, the velocity, and the forces of the current and previous time step.

LinkedCell: This class provides the functionality to store molecules in a list, access this list and iterate over it.

MolecularDynamicsSimulation: This class runs the MD simulation. Its methods allow to initialize the system, simulate one time step, run the entire simulation, and shut down the system. During the initialization, it uses the functions of the directory configurations and during each time step, it operates by using the services. The control flow of SimpleMD is described in chapter 3.6.

The remaining classes and more information about the classes described here can be found in appendix A.1.1.

3.5.2. Subdirectories

Configurations

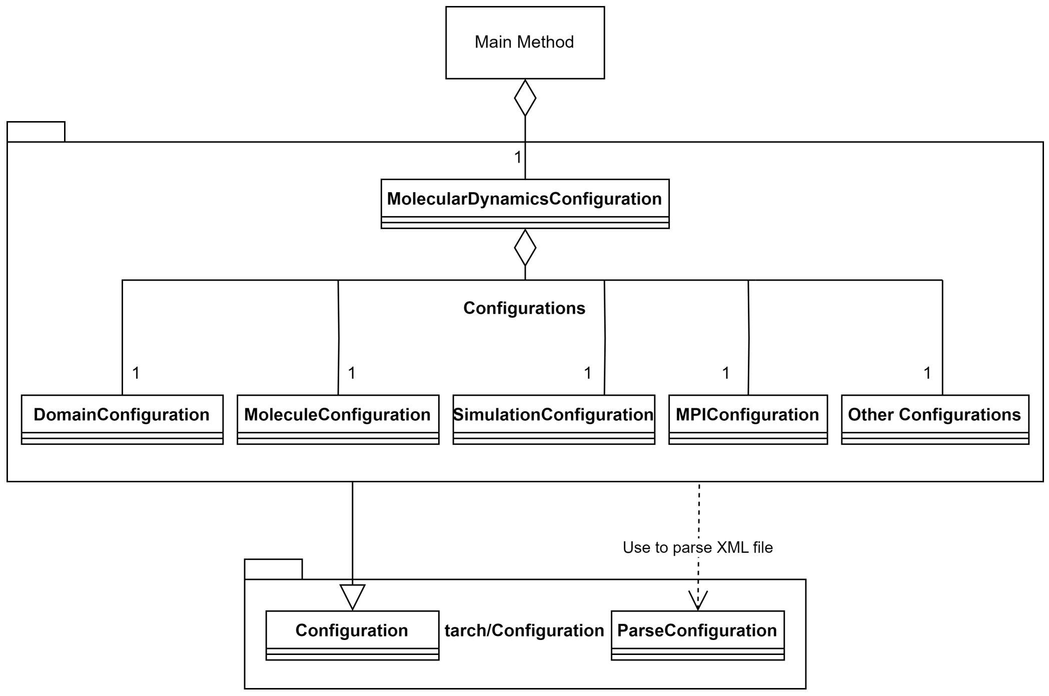


Figure 3.4.: Simplified structure of the subdirectory configurations. All attributes and methods are omitted. Most can be found at table 3.1 and chapter 3.2.2.

All classes in the subdirectory configurations inherit their structure from the utility class `Configuration`, which is described in chapter 3.2.2. Each class stores the parsed values in local attributes and provides getter methods for them. Table 3.1 shows the most important values parsed in the respective classes.

Class	Properties
`DomainConfiguration`	• Molecules per direction • Domain size per direction • Domain offset (coordinates of the bottom-left-front point of the domain) • Cutoff radius • Linked cell size per direction • Boundary types • Checkpoint file name (optional)
`MoleculeConfiguration`	• Mass • Lennard-Jones parameter ϵ • Lennard-Jones parameter σ
`SimulationConfiguration`	• Time step length δt • Number of time steps
`MPIConfiguration`	• Number of processes per direction

Table 3.1.: The most important values parsed in the most important configuration classes.

Additionally, there are further configuration classes. Most of them provide values used to set up plotters and writers for checkpoint files, which store data about all molecules. The other two classes are

1. `ExternalForceService`, which parses constant external forces to be applied to the entire domain in every time step, and
2. `MolecularDynamicsConfiguration`, which stores all other configuration classes based on their tag. `MolecularDynamicsSimulation` uses this class to access all configurations.

Services

The service classes are primarily used by the class `MolecularDynamicsSimulation` to execute simulation of one time step. The most important services are the `LinkedCellService` and the `MoleculeService`. The other services are the `ExternalForceService`, adding the external forces to the molecules' force buffers, the `MolecularPropertiesService`, enabling access to the class `MolecularProperties`, and the `ParallelAndLocalBufferService` and the `ParallelTopologyService`, which manage the processes and the communication between them in case of distributed memory parallelization, parallel and periodic boundaries. The latter two will not be covered in this thesis.

MoleculeService

The `MoleculeService` stores all molecules in a `std::vector`[15] and allows for adding and deleting molecules. The most important method for the simulation, however, is the generic method `iterateMolecules`. It receives an instance of a molecule mapping and applies this mapping to all stored molecules. Therefore, it iterates over all molecules stored in the vector and calls the mapping's method `handleMolecule` with the currently handled molecule as input argument. Shared memory parallelization with OpenMP is used for this iteration if the corresponding compile option has been enabled.

LinkedCellService

The class `LinkedCellService` stores all linked cells and all ghost cells of the (sub-)domain in an array. The index of each linked cell can be calculated as described in chapter 3.4. It provides the functions for adding and deleting molecules from linked cells. The cell structure cannot be changed after it has been generated by the method `initCellStructure`. In contrast to the `MoleculeService`, there are two frequently used methods in the `LinkedCellService`. These methods are `iterateCells` and `iterateCellPairs`.

iterateCells: This generic method iterates over all non-ghost cells and calls the method handleCell of the cell mapping it received as input. The iteration is parallelized using OpenMP when the compile option has been enabled.

iterateCellPairs: This generic method works similar to `iterateCells`. It receives a cell mapping, iterates over all non-ghost cells and applies the mapping to all cells and neighboring cell-pairs. To avoid the cell mapping being applied multiple times on the same cell-pair, the visibility restriction mentioned in chapter 3.4 have to be complied with. During the iteration each non-ghost cell is handled once. First the handled cell gets passed to the cell mapping's method `handleCell`, then, all cell-pairs, whose interactions needs to be executed during the handling of this cell, get passed to the method `handleCellPair`.

[15]`https://en.cppreference.com/w/cpp/container/vector`

Shared memory parallelization with OpenMP is possible for the iteration in `iterateCellPairs`, as well. However, a red-black traversal, or a colored traversal in general, has to be applied to avoid race conditions. Race conditions can occur if any two neighboring cells are handled parallelly because one may accesses the other and/or both may access a shared neighbor at the same time. All cells are split in 2^d groups. (d is the number of dimensions.) Every group contains all cells that share the same combination of even or odd coordinates in the respective dimensions. Afterwards, the cells in one group are iterated. This iteration can be parallelized because no race conditions can occur in the simulation use case. The iteration of the next group can only start if the iteration of its predecessor is completed.

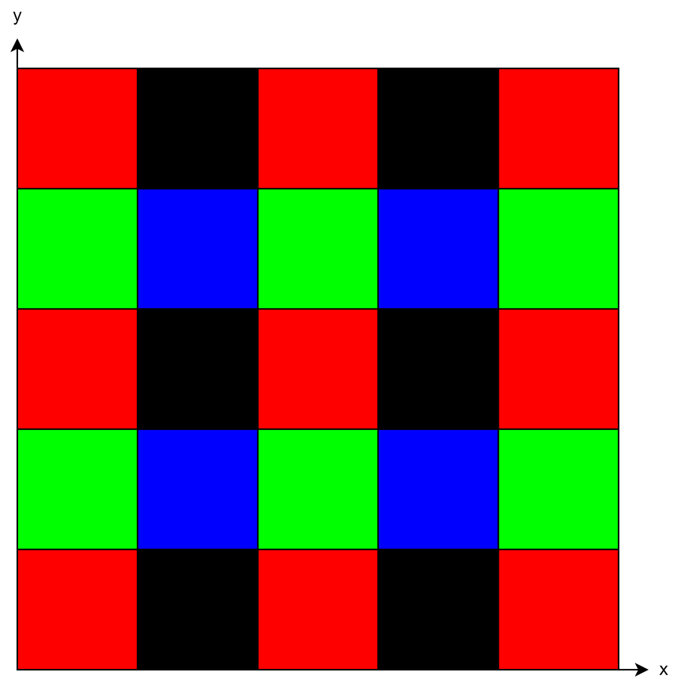


Figure 3.5.: Distribution of linked cells of a section of a two-dimensional domain into groups according to the red-black traversal. All cells of one color can be handled parallelly. However, no differently colored cells may be handled, meanwhile. In the three-dimensional case, this distribution is valid for every second value of the z-coordinate. For the remaining half of the three-dimensional domain, the same scheme can be applied with different colors.

Cell Mappings

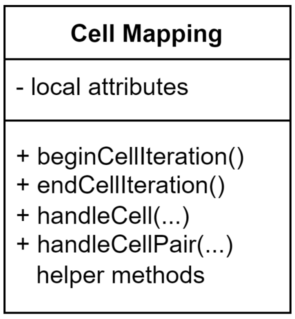


Figure 3.6.: Class diagram of the general structure of all cell mappings.

All cell mapping classes are structured similarly. They feature the methods `beginCellIteration` and `endCellIteration`, which are further explained in appendix A.1.2. Additionally, there are the two methods `handleCell` and `handleCellPair`. They receive one or two cells respectively and their indices, and apply the mapping to the molecules in these cells. The most relevant cell mappings are:

EmptyLinkedListMapping: After the last cell mapping has been applied in a time step, this mapping is used to remove all molecules from each cell because, after time integration, they might belong in another cell.

LennardJonesForceMapping: This mapping computes the Lennard-Jones forces, as described in chapter 3.3.1 for all molecule-pairs in one cell if `handleCell` is called, and for all molecule-pairs with one molecule being located in each of the two cells if `handleCellPair` is executed. The forces are added to or subtracted from the force buffer of the molecules. Additionally, if `handleCell` is called, constant external forces are added to the force buffer of each molecule. The external forces are added by the `ExternalForceService`, which is mentioned in chapter 3.5.2.

The remaining mappings are mentioned or described in appendix A.1.2.

Molecule Mappings

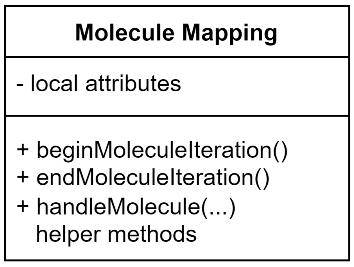


Figure 3.7.: Class diagram of the general structure of all molecule mappings.

Like the cell mappings, all molecule mappings have a similar structure. This structure contains the methods `beginMoleculeIteration` and `endMoleculeIteration`, which serve the same purpose as their equivalents in the cell mappings, and `handleMolecule` which applies the mapping to a passed molecule. The most important molecule mappings are:

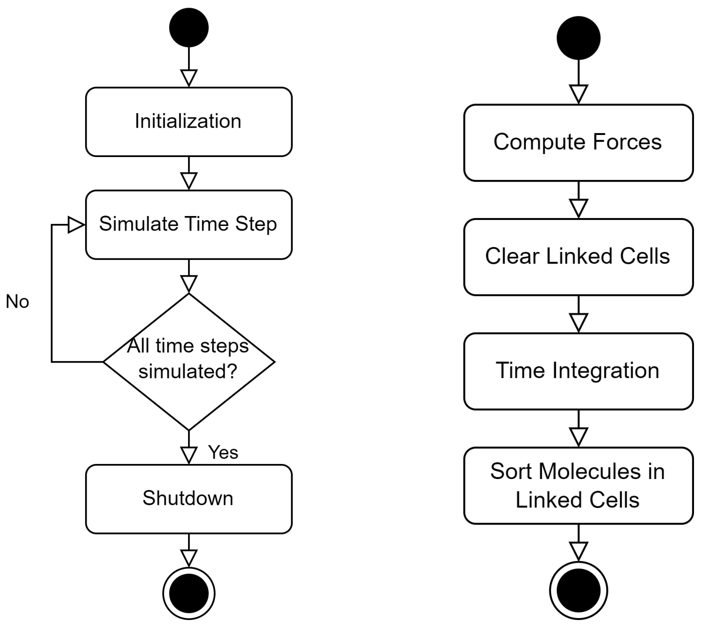
InitialPositionAndForceUpdate: The mapping replaces the time integration in the initial time step. Its functionality is described in chapter 3.3.2. After the calculation it stores the force calculated in the initial time step in the buffer for the force of the previous time step and resets the force buffer to 0.

UpdateLinkedCellListMapping: This mapping sorts the molecules into the linked cells according to the coordinates. It is used at the end of every time step.

VelocityStoermerVerletMapping: This mapping applies the time integration as described in chapters 2.2 and 3.3.2. Similar to the `InitialPositionAndForceUpdate`, the force of the current time step is stored in `forceOld` and the buffer `force` is reset to **0**.

The remaining molecule mappings are mentioned in appendix A.1.2.

3.6. Control Flow of SimpleMD



(a) Simulation (b) Time step

Figure 3.8.: Simplified control flow of a SimpleMD simulation and one time step.

Figure 3.8a shows the simplified control flow of SimpleMD. During the initialization, the XML file is parsed and the values are used for the creation of all objects. Then, the number of time steps defined in the XML file are simulated. The procedure during one time step is illustrated in figure 3.8b. Finally, all structures are cleared and the objects are deleted during the shutdown phase. More details about the specific phases of a SimpleMD simulation can be found in appendix A.2. There is information about the actions during each time step, which are not shown in the figure.

Part II.

Implementation and Performance

4. Implementation

This chapter showcases the changes and extensions that were implemented in SimpleMD.[1] Chapter 4.1 explains, how the program was extended with three-body interactions. Chapter 4.2 shows the approach of using both two-body interactions and three-body interactions, to run a MD simulation with adaptive resolution. In chapter 5, the performance improvements achieved by shared memory parallelization (using OpenMP), and distributed memory parallelization (via MPI), are shown and analyzed. Additionally, the differences in execution times between three-body interactions only, different adaptive resolution cases, and two-body interactions only, are investigated.

4.1. Three-Body Interactions

The implementation of three-body interactions required additional mappings, as well as changes in existing classes. Backwards compatibility is guaranteed because the new code is only used if the corresponding constant `MD_BODY`, used for preprocessor directives, is set accordingly. The implementation of the compile option, switching between two and three body interactions, and its impact on the program are described in chapter 4.1.4.

4.1.1. Cell Mappings

Handling Triplets of Cells

Three-body interactions operate on three different molecules which may be located in three different cells. However, the cell mappings currently only support computations with molecules that are in one cell or in a pair of two cells. Therefore, the new method **`handleCellTriplet`**, which receives instances of all three cells and their indices, similar to **`handleCell`** and **`handleCellPair`**, is added. All currently existing cell mappings do not operate on triplets of cells. Thus, the new method has no function in all cell mappings existing so far. It is only used in the two new cell mappings described below.

[1]The implementation at the time of writing can be found on a dedicated branch of MaMiCo at `https://github.com/HSU-HPC/MaMiCo/tree/simplemd-three-body`

Force Calculation

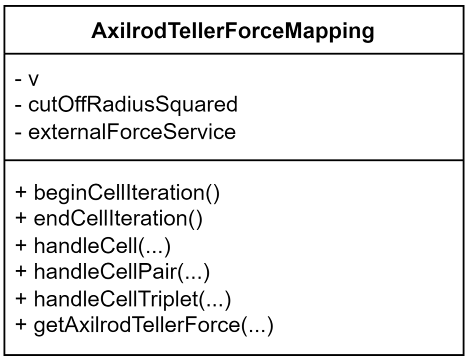


Figure 4.1.: Simplified class diagram of the AxilrodTellerForceMapping.

The forces of the molecule-triplets are computed by the new class `AxilrodTellerForceMapping`. As the name suggests, the Axilrod-Teller potential is used as a basis for the force calculations. The forces are calculated according to the formula shown in chapter 2.3.4. The class has three attributes. They are the Axilrod-Teller parameter ν, which is called `v` in the code, the squared cutoff radius and the instance of the `ExternalForceService`. The methods and their functionality are listed below:

handleCell: The method passes all molecules in the linked cell to the `ExternalForceService`, iterates over all triplets of molecules within this cell, and initiates the force calculation by the method `getAxilrodTellerForce` for each triplet. It subtracts the reversed forces, which are returned by `getAxilrodTellerForce`, from the molecules' force buffers.

handleCellPair: First, all molecule-triplets, that contain two molecules of the first passed cell and one of the second cell are iterated over. Then, the triplets, which contain one molecule of the first cell and two of the second are iterated over. The force computations are handled as described above.

handleCellTriplet: This method iterates over all molecule-triplets, which contain one molecule of each of the three cells, and initiates the force calculations like mentioned before.

getAxilrodTellerForce: The method receives the positions of all three molecules and calculates the reversed force for each of the molecules if the distance between all three pairs of molecules respectively is not bigger than the cutoff radius. Otherwise, all returned forces are $\mathbf{0}$. The three computed forces, which are all instances of the utility class `Vector`, are returned in a `std::vector`. The algorithm of the force computation is shown in figure 4.2.

```
Algorithm 1: getAxilrodTellerForce
  Input:   positions of three molecules
  Output:  vector containing the reversed force vectors
Function getAxilrodTellerForce($\mathbf{x}_i$, $\mathbf{x}_j$, $\mathbf{x}_k$):
    // Distances and their powers
    $\mathbf{r}_{ij} \leftarrow \mathbf{x}_j - \mathbf{x}_i$
    $r_{ij}^2 \leftarrow \langle \mathbf{r}_{ij}, \mathbf{r}_{ij} \rangle$ // dot product
    if $r_{ij}^2 \leq r_{cutoff}^2$ and $r_{ik}^2 \leq r_{cutoff}^2$ and $r_{jk}^2 \leq r_{cutoff}^2$ then
        $r_{ij} \leftarrow \sqrt{r_{ij}^2}$
        $r_{ij}^3 \leftarrow r_{ij}^2 \cdot r_{ij}$
        $r_{ij}^4 \leftarrow r_{ij}^2 \cdot r_{ij}^2$
        $r_{ij}^5 \leftarrow r_{ij}^4 \cdot r_{ij}$
        $r_{ij}^6 \leftarrow r_{ij}^4 \cdot r_{ij}^2$
        // Gradients
        $dVdRij \leftarrow \frac{3 \cdot \nu}{8 \cdot r_{ij}} \cdot (-\frac{8}{r_{ij}^4 \cdot r_{ik}^3 \cdot r_{jk}^3} - \frac{1}{r_{ik}^5 \cdot r_{jk}^5} + \frac{5 \cdot r_{ik}}{r_{ij}^6 \cdot r_{jk}^5} + \frac{5 \cdot r_{jk}}{r_{ij}^6 \cdot r_{ik}^5} - \frac{1}{r_{ij}^2 \cdot r_{ik}^3 \cdot r_{jk}^5} - \frac{1}{r_{ij}^2 \cdot r_{ik}^5 \cdot r_{jk}^3} - \frac{3}{r_{ij}^4 \cdot r_{ik} \cdot r_{jk}^5} - \frac{3}{r_{ij}^4 \cdot r_{ik}^5 \cdot r_{jk}} - \frac{5}{r_{ij}^6 \cdot r_{ik} \cdot r_{jk}^3} - \frac{5}{r_{ij}^6 \cdot r_{ik}^3 \cdot r_{jk}} + \frac{6}{r_{ij}^4 \cdot r_{ik}^3 \cdot r_{jk}^3})$
        // Forces
        $\mathbf{F}_i \leftarrow \mathbf{r}_{ij} \cdot dVdRij + \mathbf{r}_{ik} \cdot dVdRik$
        $\mathbf{F}_j \leftarrow \mathbf{r}_{ij} \cdot (-dVdRij) + \mathbf{r}_{ik} \cdot dVdRik$
        $\mathbf{F}_k \leftarrow \mathbf{r}_{ij} \cdot (-dVdRij) + \mathbf{r}_{ik} \cdot (-dVdRik)$
        return [ $\mathbf{F}_i$, $\mathbf{F}_j$, $\mathbf{F}_k$]
    else
        return [ $\mathbf{0}$, $\mathbf{0}$, $\mathbf{0}$]
```

Figure 4.2.: The computation of the negative Axilrod-Teller forces of a molecule-triplet. The allocations of the variables for the distances and their powers are only shown for the molecule-pair ij. The other allocations work analogously. The calculation of the gradients is only shown for $\frac{1}{r_{ij}} \frac{\partial V}{\partial r_{ij}}$ $(dVdRij)$. The two other gradients are calculated analogously. In the formula, the indices ij and ik, or ij and jk have to be exchanged. Like in the code, $\mathbf{r}_{ij}$ is the vector from molecule i to j. In chapter 2 it was called $\mathbf{x}_{ij}$. The variable $\mathbf{F}_i$ actually stores the value $-\mathbf{F}_i$. The variable is called $\mathbf{F}_i$ to be consistent with the code. The same holds for $\mathbf{F}_j$ and $\mathbf{F}_k$.
Source: [Mar01]

Potential Energy Calculation

The class `AxilrodTellerPotentialEnergyMapping` is structured and works similarly to the force mapping. The potential energy is calculated in the method `getAxilrodTellerPotentialEnergy` according to the formula resulting after the application of the cosine rule, mentioned in chapter 2.3.4. In contrast to the force computations, the potential energy is added to the molecules' respective variables. The mapping is never used in SimpleMD, similar to the `LennardJonesPotentialEnergyMapping`. Nevertheless, it is still included for completeness.

4.1.2. Including the Axilrod-Teller Parameter ν

The three-body forces and potential energy calculations in the extension of SimpleMD are based on the Axilrod-Teller potential. Therefore, the coefficient ν has to be included in the calculations. This section shows which changes were necessary that the parameter, which is part of the XML input file, is parsed and included in the new mappings described in chapter 4.1.1.

MoleculeConfiguration

The configuration class responsible for parsing ν is `MoleculeConfiguration`. Similar to the Lennard-Jones parameters, ν is retrieved from the XML file, where it is stored with the key `v`. It replaces the parsing of the Lennard-Jones parameters. The value is stored in the new attribute `v`. Additionally, a getter method is created for this attribute.

MolecularPropertiesService and MolecularProperties

The `MolecularDynamicsSimulation` passes all molecular properties, including ν, to the service at initialization. The service then uses all properties to call the constructor of `MolecularProperties` with these values as input arguments. The implementation of three-body interactions requires the new attribute `v` of `MolecularProperties`, storing the value of ν, a getter method returning it, and a new constructor for `MolecularProperties` and `MolecularPropertiesService` respectively, which replaces the Lennard-Jones parameters with the Axilrod-Teller coefficient and, ultimately, stores it in the new attribute.

The `MolecularPropertiesService` is passed to the `AxilrodTellerForceMapping` at initialization. The mapping gets the value of ν by accessing `MolecularProperties` via the instance of the service.

4.1.3. Linked Cell Approach for Three-Body Interactions

Force Calculations

In case of three-body interactions, SimpleMD still uses the existent linked cell approach. Since all three molecules have to be within each other's cutoff radius for the forces to be different from 0, the three molecules have to be within the same cell, two neighboring cells, or three cells which all neighbor each other. For molecules in the same cell or in two neighboring cells, the approach described in chapter 3.4 can be reused.

The same two guidelines to the initiation of cell-triplet calculations as to the initiation of pair calculations (see chapter 3.4) apply. Therefore, each cell can only initiate the cell-triplet iterations for most of the triplets including the handled cell and its neighbors in the 2×2 square or $2 \times 2 \times 2$ cube with the handled cell as bottom-left(-front) corner.

Two-Dimensional Force Calculations

In the two dimensional case, a cell needs to initiate the triplet calculations for all 4 combinations of 3 of the 4 cells of this square because in 3 of the cases, the handled cell is part of the triplet and, thus, has the lowest coordinate in both directions, and in the fourth case, all three other cells are involved but the right and the top neighbor cannot see each other and the top-right neighbor can see neither of those.

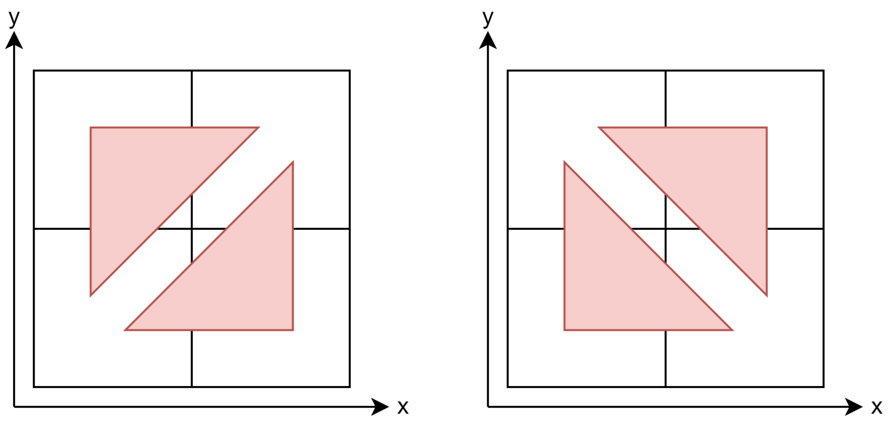


Figure 4.3.: Force calculations that need to be initiated by the bottom-left cell

Since the same behavior applies to all cells, the remaining 9 triplet calculations involving a cell are initiated by its left, bottom, and bottom-left neighbor. Each of these cells triggers 3 computations.

Three-Dimensional Force Calculations

In case of a three-dimensional simulation, there are a total of 56 cell-triplets, which include only the handled cell and/or its visible neighbors. However, the computations only have to be initiated for 44 of these 56 triplets because all cell-triplets within the yz-plane including the right neighbor, within the xz-plane including the top neighbor, and within the xy-plane including the back neighbor can be triggered by the right, back, and top neighbor respectively. Each of these three cell-squares include 4 triplets as explained in the section above, resulting in 12 interactions that do not need to be initiated. These triplets and the respective cell initiating the force computations are listed in table 4.1.

Neighbor 1	Neighbor 2	Neighbor 3	Initiating neighbor
right	top-right	right-back	right
right	top-right	top-right-back	right
right	right-back	top-right-back	right
top-right	right-back	top-right-back	right
top	top-right	top-back	top
top	top-right	top-right-back	top
top	top-back	top-right-back	top
top-right	top-back	top-right-back	top
back	right-back	top-back	back
back	right-back	top-right-back	back
back	top-back	top-right-back	back
right-back	top-back	top-right-back	back

Table 4.1.: Three-body interactions that do not need to be initiated by a cell, although all cells are in the $2 \times 2 \times 2$ cube with the handled cell as bottom-left-front corner.

The 44 triplet interactions that need to be initiated by a cell can be split into two groups. There are 21 triplets which all include the handled cell, and there are 23 triplets not including the handled cell. The interactions of the latter triplets need to be triggered by this cell due to mutual invisibility. The two neighboring cells of the 21 triplets which include the handled cell are listed in table 4.2.

Neighbor 1	Neighbor 2
right	top
right	top-right
top	top-right
right	back
right	right-back
right	top-back
right	top-right-back
top	back
top	right-back
top	top-back
top	top-right-back
top-right	back
top-right	right-back
top-right	top-back
top-right	top-right-back
back	right-back
back	top-back
back	top-right-back
right-back	top-back
right-back	top-right-back
top-back	top-right-back

Table 4.2.: Three-body interactions that need to be initiated by a cell and include this cell.

The remaining 23 triplet interactions a cell has to initiate are listed below.

Neighbor 1	Neighbor 2	Neighbor 3
right	top	top-right
right	top	back
right	top	right-back
right	top	top-back
right	top	top-right-back
right	top-right	back
right	top-right	top-back
top	top-right	back
top	top-right	right-back
right	back	right-back
top	back	right-back
top-right	back	right-back
right	back	top-back
top	back	top-back
top-right	back	top-back
right	back	top-right-back
top	back	top-right-back
top-right	back	top-right-back
right	right-back	top-back
top	right-back	top-back
top-right	right-back	top-back
top	right-back	top-right-back
right	top-back	top-right-back

Table 4.3.: Three-body interactions that need to be initiated by a cell and not include this cell.

LinkedCellService

The `LinkedCellService` only had the methods `iterateCells` and `iterateCellPairs`. Neither of these implemented the handling of cell-triplets. Thus, the new method `handleCellTriplets` was created. The input arguments are the cell mapping that should be applied, the coordinates of the subdomain, the iteration should be executed on, and a boolean value which specifies whether to use OpenMP for parallelization purposes or not.

First, multiple vectors are initialized and filled with the indices of the two-cells of all cell-pairs and the three cells of all cell-triplets whose handling needs to be initiated, in relation to the index of the currently handled cell. Then, the method `beginCellIteration` is executed and the iteration over all non-ghost cells, according to increasing indices, commences. If a cell is handled, the mapping is applied to the cell, before application to the cell-pairs and cell-triplets. The cells for the pair and triplet application are picked according to the previously created vectors. Finally, the method `endCellIteration` is called.

If a shared memory parallelization, using OpenMP, should be applied, the approach described in chapter 3.5.2 can be used. The same red-black traversal works in this method as well because all computations only involve a cell and its direct neighbors.

MolecularDynamicsSimulation

The simulation only works with three-body interactions if the `AxilrodTellerForceMapping` gets passed to the `LinkedCellServices`'s method `iterateCellTriplets`. Therefore, the `MolecularDynamicsSimulation` needs to initialize the mapping in its method `initServices`. It calls the constructor of the mapping and passes the instances of the `ExternalForceService` and the `MolecularPropertiesService` as input arguments. The force mapping uses these objects to get the values to store in all attributes.

All calls of `iterateCellPairs` with the `LennardJonesForceMapping` as input need to be replaced by calls of the new method `iterateCellTriplets` and the `AxilrodTellerForceMapping` as input argument. The same applies for the calls in the `BoundaryTreatment` as well.

4.1.4. Compile Option and Preprocessor Directives

Since CMake is used to generate Makefiles, all compile options are implemented in the file `CMakeLists.txt`. This file is located in the directory MaMiCo, one directory level above SimpleMD.

In this file the option `MDBody` has been added, which could be set to either `MDBody2`, for two-body interactions, or `MDBody3`, for three-body interactions. The value of this parameter is passed to SimpleMD as compile definition. Additionally, the two new mappings are added in the list of source files to compile.

In the file `MolecularDynamicsUserIput.h`, depending on the value defined for `MDBody`, the global constant `MD_BODY` is either set to 2 or to 3. If neither of the two allowed values for `MDBody` is defined (which should not happen) the compilation immediately fails.

This constant is used in preprocessor directives to only compile the code needed for the three-body computations if the option is set accordingly. In the previous chapters, the word *replace* was used multiple times. However, this did not refer to deleting the two-body code and implementing the three-body code instead. It meant that both versions of the code are there but, according to `MD_BODY`, one version of the code gets ignored by the compiler. The general use of the directives is shown in listing 4.1.

```
#if (MD_BODY == 2)
  // two-body code
#else
  // three-body code
#endif
```

Listing 4.1: Use of preprocessor directives for three-body interactions.

4.2. Adaptive Resolution

In contrast to the two-body interactions and three-body interactions, the implementation of the adaptive resolution requires both types of interactions, not just one. New mappings and changes to existing structures are needed, to store the two types of forces separately, to weight them for each molecule, and to sum up the weighted values. Backwards compatibility is, again, guaranteed by a constant set according to a compile option, and preprocessor directives.

4.2.1. Functionality of Adaptive Resolution in SimpleMD

The adaptive resolution in SimpleMD is constructed for use with reflecting boundaries. If any other type of boundaries (especially periodic ones) is used, a correct calculation of forces is very unlikely. The reason for this is that between the molecules on opposite sides of the domain only one of the force types would be calculated, resulting in some molecules missing out on partial forces. The scheme is applied in one dimension. In this dimension, the area from the beginning[2] of the domain to the beginning of the interface layer is used for two-body interactions, both types of forces are applied in a relation depending on each molecule's position inside the interface layer, and only three-body interactions are applied between the end[3] of the interface layer and the end of the domain. It may occur, that one type of force between molecules has to be calculated, but the force is not applied to all molecules because some of them are outside of the interface layer and others are not. This circumstance, however, does not cause any problems because the calculated force of this type is assigned the weight 0 if it is not needed and, thus, it has no impact on the total force.

Calculations of Partial Forces

Since force calculations are the most expensive part of each time step, each type of force is only calculated for those cells, where it might impact the total force of the molecules inside the cell. All molecules in the interface layer need both types of forces. Due to the linked cell approach, the force calculations involve all neighbors. Therefore, both types of forces need to be calculated for all interactions involving at least one cell in the interface layer. These interactions can include one or two cells outside the interface layer.

Force computations are still initiated in the way described in chapter 3.4. As a result the initiation of three-body force calculations has to begin in the plane of cells, whose coordinates in the dimension adaptive resolution is applied in is lower by one than the according coordinates of the cells which the interface layer starts in. If the interface layer starts in the non-ghost cells with the lowest coordinates, the three-body computations begin in these cells.

On the other hand, the last plane of cells, which need to initiate two-body force computations, is the one the interface layer ends in. All two-body computations between the

[2]the boundary with the lower coordinate
[3]the boundary with the higher coordinate

cells in this plane and the cells in the neighboring plane, whose coordinates in the according direction are higher by one, are initiated by the lats plane of cells, which is still (partially or completely) in the interface layer.

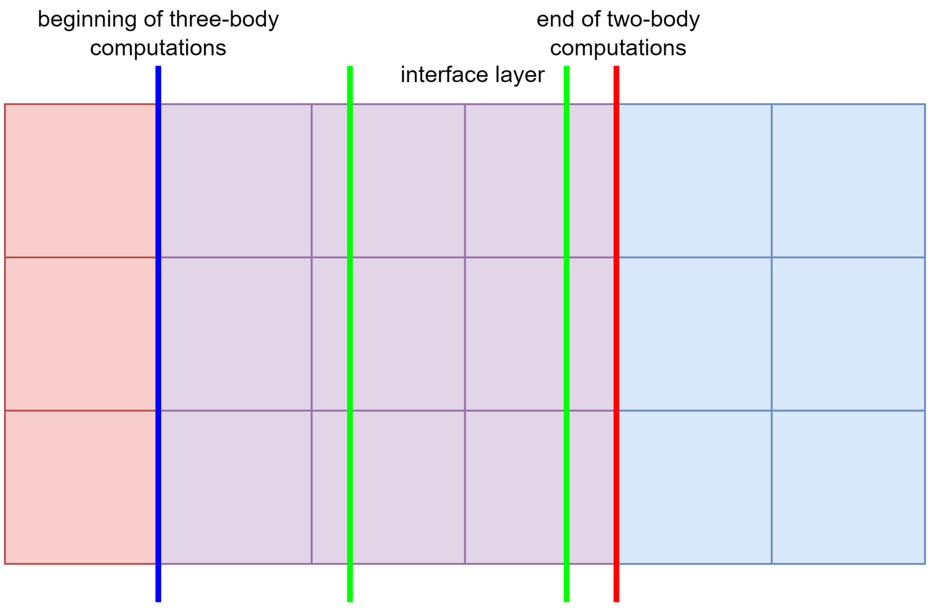

beginning of three-body computations
end of two-body computations
interface layer

Figure 4.4.: Force computations that are initiated by the cells for adaptive resolution. Red cells only initiate two-body computations, blue cells only initiate three-body computations, and purple cells initiate both.

Weights, Total Force, and Total Potential Energy

The weights $w(x_d) \in [0, 1]$ for the molecules depend on:

- The coordinate x_d of each molecule, in the dimension d adaptive resolution is applied in
- The coordinate $x_{interface}$ of the boundary of the interface layer with the lower coordinate
- The length of the interface region $d_{interface}$

They are calculated according to the formula below, which fulfills all criteria described in chapter 2.4.

$$w(x_d) = \begin{cases} 0 & x_d \leq x_{interface} \\ sin^2(\frac{\pi}{2 \cdot d_{interface}} \cdot (x_d - x_{interface})) & x_{interface} < x_d < x_{interface} + d_{interface} \\ 1 & x_d \geq x_{interface} + d_{interface} \end{cases}$$

Based on [PDSK05]

If $d_{interface} = 0$, the formula transforms to:

$$w(x_d) = \begin{cases} 0 & x_d \leq x_{interface} \\ 1 & x_d > x_{interface} \end{cases}$$

The weight determines the impact of three-body forces on the total force and the impact of the three-body potentials on the total potential energy. These are calculated according to the formulas shown in chapter 2.4, but the indices need to be reversed.

4.2.2. Compile Option and Preprocessor Directives

In the file `CMakeLists.txt`, the option `ADRES` has been added. This value can either be set to `OFF` or `ON`, and decides whether adaptive resolution is activated or not.

If the option's value is `ON`, the string `AdRes` is passed to SimpleMD as compile definition and the two new mappings, which will be described in chapter 4.2.4, are added to the source files to compile. Otherwise, neither of this happens and the compilation proceeds as chosen in `MDBody`.

In the `MolecularDynamicsUserInput.h`, the new global constant `AD_RES` is set to `MD_YES`, which is defined as `1` or `true`, if the compile definition exists. Else, the constant is set to `MD_NO`, which is defined as `0` or `false`.

This constant is used in preprocessor directives in a way, that the impact of the constant `MD_BODY` is undone if adaptive resolution is activated. In that case both, the two-body and three-body code, is compiled and used in the program. There are also some parts of the code which should only be used if adaptive resolution is activated. Preprocessor directives are used in this context as well. The use of directives in this case is shown in listing 4.2 and can be compared to the use for three-body interactions shown in listing 4.1.

```
#if (MD_BODY == 2 || AD_RES == MD_YES)
  // two-body code
#endif
#if (MD_BODY == 3 || AD_RES == MD_YES)
  // three-body code
#endif

#if (AD_RES == MD_YES)
  // adaptive resolution code
#endif
```

Listing 4.2: Use of preprocessor directives for adaptive resolution.

4.2.3. Changes in Existing Classes

In this chapter, all necessary changes for adaptive resolution in the respective classes are described. Preprocessor directives guarantee that all these changes only apply if the compile option is set to `ON`.

Molecule: The class `Molecule` was extended by the attributes `twoBodyForce`, `threeBodyForce`, `twoBodyPotentialEnergy`, and `threeBodyPotentialEnergy`. They are used to store the partial forces and energies. The existing fields are filled with the total forces after their calculation. The latter two are not used in SimpleMD. Additionally, all getter and setter methods for the new attributes have been added.

LennardJonesForceMapping, LennardJonesPotentialEnergyMapping, AxilrodTellerForceMapping, and AxilrodTellerPotentialEnergyMapping: These classes store the computed forces and energies in the respective new attributes of `Molecule` and no longer in `force` or `potentialEnergy`.

DomainConfiguration: The three new attributes `adResDimension`, `interfaceStart`, and `interfaceLength`, are created and getter methods are added. All these values are retrieved during parsing of the subtag *domain-configuration* of the XML file.

MolecularDynamicsSimulation and BoundaryTreatment: In all places where force calculations occur, first the Lennard-Jones forces are calculated, using the `LinkedCellService`'s method `iterateCellPairs`, then the Axilrod-Teller forces are calculated, using `iterateCellTriplets`.

MoleculeConfiguration: This class now parses both the Lennard-Jones parameters and the Axilrod-Teller parameter, along with the remaining values in the subtag *molecule-configuration*.

MolecularProperties and MolecularPropertiesService: Yet another constructor has been added to the classes. The new constructor receives the values of all three parameters of the Lennard-Jones and Axilrod-Teller potential (ϵ, σ, and ν) as input arguments.

LinkedCellService: Three new attributes have been added. The field `adResDimension` stores the dimension for the adaptive resolution, `threeBodyStart` stores the coordinate in this direction of the first cells which have to initiate three-body computations, and `twoBodyEnd` stores the coordinate of the last cells that have to initiate two-body computations. The fields are filled with the values during the initialization. The values for the latter two are calculated in a way such that they correspond to the cells described in chapter 4.2.1. The methods `iterateCells` and `iterateCellTriplets` now skip the handling of a cell if the mapping, that should be applied is the respective force mapping, and the coordinate in the adaptive resolution dimension is lower than `threeBodyStart` in `iterateCellTriplets`, or higher than `twoBodyEnd` in `iterateCellPairs`.

4.2.4. New Cell Mappings

The simulation requires a new mapping, which weights the partial forces and sums them up accordingly. A new mapping that computes the total potential energy from the partial energies is added for completeness, although it is never used in SimpleMD. Both mappings work on single molecules and thus could be cell mappings and molecule mappings likewise. We decided to implement them as cell mappings which are applied by the `LinkedCellService`'s method `iterateCells`.

Computation of the Total Force

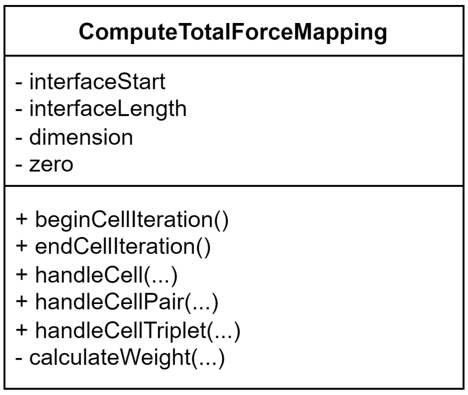


Figure 4.5.: Simplified class diagram of the ComputeTotalForceMapping.

The class `ComputeTotalForceMapping` has 4 attributes. The 3 fields, defining the interface layer, are used for the calculation of the weight, which is described in chapter 4.2.1 and executed by the method `calculateWeight`. This method receives the coordinate of the handled molecule in the direction adaptive resolution is applied in, and returns the weight $w(x_d)$. The vector zero contains, as the name suggests, the zero vector. It is used to reset the partial forces after the computation of the total force by copying its values.

The only method with a function, apart from `calculateWeight`, is `handleCell`. It iterates over all molecules, calculates their weights, and sums up the total force as described in chapter 4.2.1. The total force is stored in the molecule's field `force` and the fields for the partial forces are reset in the end of the method execution.

Computation of the Total Potential Energy

The class `ComputeTotalPotentialEnergy` works similarly to the `ComputeTotalForceMapping`. It calculates the weight according to the molecule's position, computes the total potential energy as described in chapter 4.2.1, and stores the total energy in the attribute `potentialEnergy` before resetting the partial energies.

Both new classes require the fields for partial forces or potential energies in the class `Molecule`. Since these attributes only exist if adaptive resolution is used, these two classes, unlike the Lennard-Jones and Axilrod-Teller mappings, cannot be compiled if adaptive resolution is switched off.

5. Performance

In this chapter the performances of the extensions described in chapter 4 are analyzed. In chapter 5.1, we investigate the scalability of the program for the shared memory parallelization with OpenMP[1] and for the distributed memory parallelization with MPI. For the message-passing interface, Intel's implementation[2] is used. In chapter 5.2, the performances of two-body interactions, three-body interactions, and three different cases of adaptive resolution are compared.

All measurements were performed on CoolMUC-2[3], a Linux cluster segment of the Leibniz Supercomputing Centre (LRZ)[4], and on HSUper[5], a HPC cluster hosted by the Container-based High Performance Computing Center (CBRZ)[6] at Helmut Schmidt University[7].

CoolMUC-2 has 812 nodes, connected by a FDR14 Infiniband interconnect. Each node consists of 28 Haswell-based cores with 2 hyperthreads. The system has a 64 bit address space. The shared memory parallelization uses the OpenMP version 4.5 and the distribute memory parallelization uses the Intel MPI 2019.12. The Makefile is created by CMake 3.21.4 and the program is compiled with gcc 11.2.0.

HSUper consists of 571 regular nodes, connected by a non-blocking NVIDIA InfiniBand HDR100. Each node contains 2 Intel Icelake sockets, featuring 36 cores each. This yields a total of 72 cores per node. Additionally, there are 5 fat memory nodes and 5 GPU nodes, which were not used for the measurements. The parallelization used OpenMP 4.5 or Intel MPI 2021.6, respectively. For compilation purposes, the CMake version 3.23.1 and the gcc version 12.1.0 are used.

The measurement, based on the chrono library[8], uses the std::chrono::high_resolution_clock. The majority of the initialization and shutdown phases cannot be parallelized and might affect the measurement result of short simulations. Hence, the measurement starts before the first time step starts and is stopped after the last one is finished.

The measured simulations were based on a randomly generated checkpoint file. A Python script creates the checkpoint file based on the XML it receives as input. It reads the information about the domain offset, the domain size, and the number of molecules per

[1]`https://www.openmp.org`
[2]`https://www.intel.com/content/www/us/en/developer/tools/oneapi/mpi-library.html`
[3]`https://doku.lrz.de/coolmuc-2-11484376.html`
[4]`https://www.lrz.de`
[5]`https://www.hsu-hh.de/hpc/en/hsuper`
[6]`https://www.hsu-hh.de/hpc/en/cbrz`
[7]`https://www.hsu-hh.de`
[8]`https://en.cppreference.com/w/cpp/chrono`

direction. Based on this data, the molecules are placed randomly, to some extent, in the domain. The script and its functionality is described in appendix B.

The majority of the measured cases yielded very consistent results among multiple measurement executions. In appendix C, the consistency among the measured times is analyzed and the method for extracting the wall times, used in the following chapters, is described.

5.1. Scalability

All parameters that had the same values throughout all simulations are listed in table 5.1. Most values, stored in the XML file and used for computations, have been normalized according to the method described in [GZK07].

Parameter	Normalized value	Actual value
m	1	$6.69 \cdot 10^{-26}$kg $= 39.948$u
ϵ	1	$1.65 \cdot 10^{-21}$J $= 120$K$\cdot k_B$
σ	1	$3.4 \cdot 10^{-10}$m $= 3.4$Å
ν	1	$1.00 \cdot 10^{-16}$JÅ^9
δt	0.00005	$1.085 \cdot 10^{-16}$s
number of time steps	200	200
r_{cutoff}	2.2	7.48Å
size of a linked cell	2.5 in each direction	8.5Å in each direction

Table 5.1.: Parameters with the same values shared in all simulations. The parameters m, ϵ, and σ are based on the molecular properties of argon. Source: [GZK07]

For convenience reasons, the value $\nu = 1$ ($1.00 \cdot 10^{-16}$JÅ^9) has been chosen. However, according to multiple sources, like [AS22] and [AL22], the value $7.35 \cdot 10^{-18}$JÅ^9 is more accurate for argon. After normalization, the value would be 0.0734. This discrepancy is accepted because the purpose of the measurements is to assess the performance of the program and not to run a realistic MD simulation. The relatively low value of 200 time steps is acceptable because the execution times of the single time steps are consistent and relatively equal.

In all adaptive resolution measurements, the scheme was applied in the z-dimension. The size of the interface layer in all three versions of the adaptive resolution scheme, that were measured, is 10% of the domain size. The interface regions are centered at 25%, 50%, and 75% of the domain, respectively. The legends of the diagrams show the percentages of the sizes of the three regions of the domain in relation to the total domain size. The order of the values is the percentage of the three-body area, followed by the interface layer and the two-body region[9], e.g. 70/10/20 means that the interface layer's size is 10% of the domain size and that it's centered at 25% of the domain. However, these percentages only show in which shares in the domain the respective partial forces are considered in the calcula-

[9]In the program, the order is the other way around.

tion of the total force. It does not show, which shares are part of each type of force calculation.

Additionally, the VaryCheckpointMapping was not used in the measured simulations because it randomly changes the molecules' properties in the first time step and hence, would make multiple results of the same measurement incomparable.

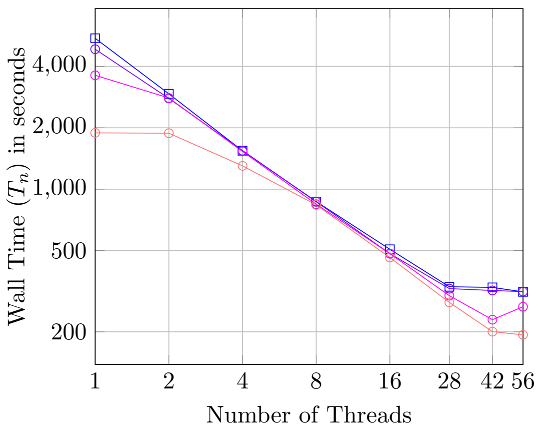
5.1.1. Shared Memory Parallelization

The initial setups and checkpoint files, especially the domain sizes and numbers of molecules, differed between the measurements made on CoolMUC-2 and HSUper. Hence, in the measurements on HSUper the start and end of the adaptive resolution interface layer is aligned with the borders between the linked cells, whereas this is not the case for the measurements on CoolMUC-2, which leads to a larger number of unnecessary force computations. Further details on the exact numbers can be found in the respective parts.

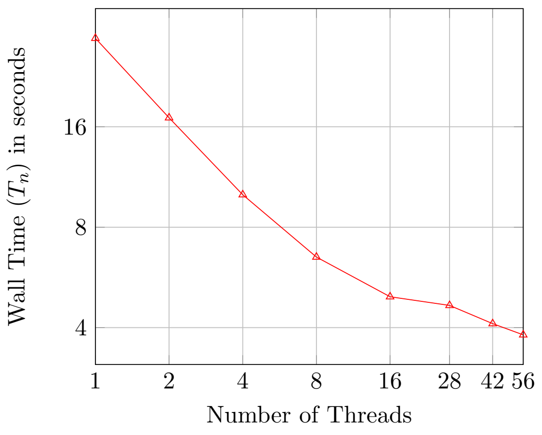
Another difference between the clusters is the number of cores per node. All measurements of the shared memory parallelization were performed on one node and used up to all of its cores. The measurements were performed with 1, 2, 4, 8, 16, 28, 42, and 56 threads on CoolMUC-2, and with 1, 2, 4, 8, 16, 32, 36, 54, and 72 threads on HSUper.

Strong Scaling on CoolMUC-2

For the strong scaling measurements on CoolMUC-2, the simulations had a domain size of 47 (159.8Å), including 44 molecules, in each direction. This results in a total of 85,184 molecules, spread over 6,859 linked cells. All simulations worked with the same checkpoint file and thus, the same distribution of molecules. Figure 5.1 shows the wall times of the runs and figure 5.2 shows the speedup in comparison to the sequential execution time, as well as the legend of all three figures.



(a) Three-body interactions and adaptive resolution



(b) Two-body interactions

Figure 5.1.: Strong scaling wall times of shared memory parallelization on CoolMUC-2.

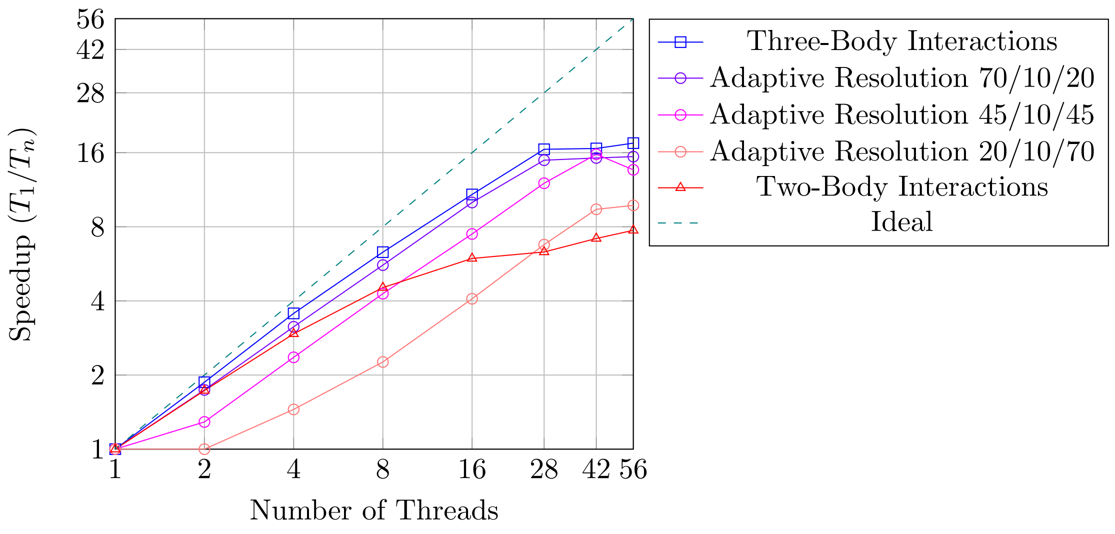


Figure 5.2.: Strong scaling speedup for shared memory parallelized three-body interactions, two-body interactions and adaptive resolution cases on CoolMUC-2.

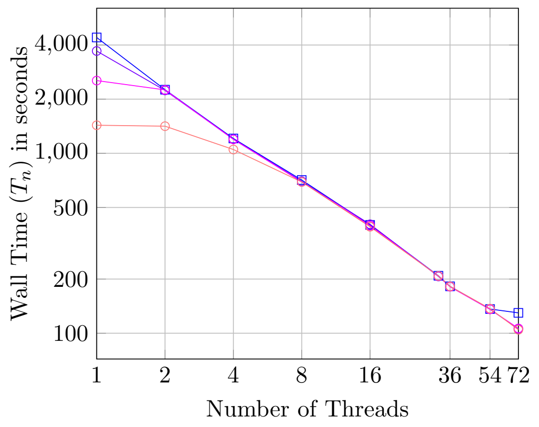
As can be seen, the three-body interactions scale best. Although they do not improve ideally, their improvements up to 28 threads are consistent and acceptable. For more threads, there are still improvements but these are significantly worse. The reason for this behavior is that up to 28 threads, each thread can work on its own physical core. If there are more threads, pairs of two threads have to share one core, which leads to less improvement than if both threads had their own cores.

The speedup of the two-body interactions is lower and the drop in improvements happened earlier because the domain of the measured simulation was too small. The additional performance overhead for logistical purposes caused by the increasing number of threads is noticeable because the total improvement is rather low due to the already low execution times. In further measurements with larger domains and more molecules, higher improvements were detected. These can be seen in appendix D. Running such larger simulations with three-body interactions, however, is not feasible.

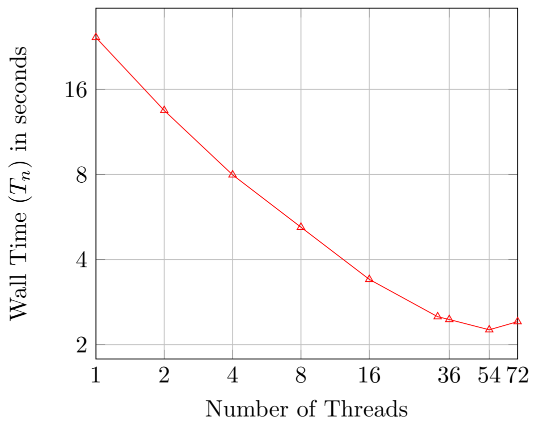
The adaptive resolution speedups get worse as the two-body volume of the domain gets bigger. The reason for this can be seen in figure 5.1a. The more threads used, the closer the execution times for adaptive resolution and three-body interactions get. This convergence can be seen, especially, in the curve for the 20/10/70 case of adaptive resolutions. There is no noticeable time improvement from increasing the number of threads from 1 to 2. This phenomenon results from some threads having similar workloads as their counterparts in three-body simulations. This is explained in more detail in chapter 5.2, as well as the reason for the time improvement of adaptive resolution over three-body interactions when hyperthreading is used.

Strong Scaling on HSUper

The strong scaling measurements on HSUper were performed with simulations consisting 47 molecules, spread over a domain size of 50 (170Å), in each direction. Thus, the total number of molecules is 103,823 and the number of linked cells is 8,000. Like on CoolMUC-2, all simulations used the same checkpoint file. The wall times are shown in figure 5.3 and the speedups in figure 5.4.



(a) Three-body interactions and adaptive resolution



(b) Two-body interactions

Figure 5.3.: Strong scaling wall times of shared memory parallelization on HSUper.

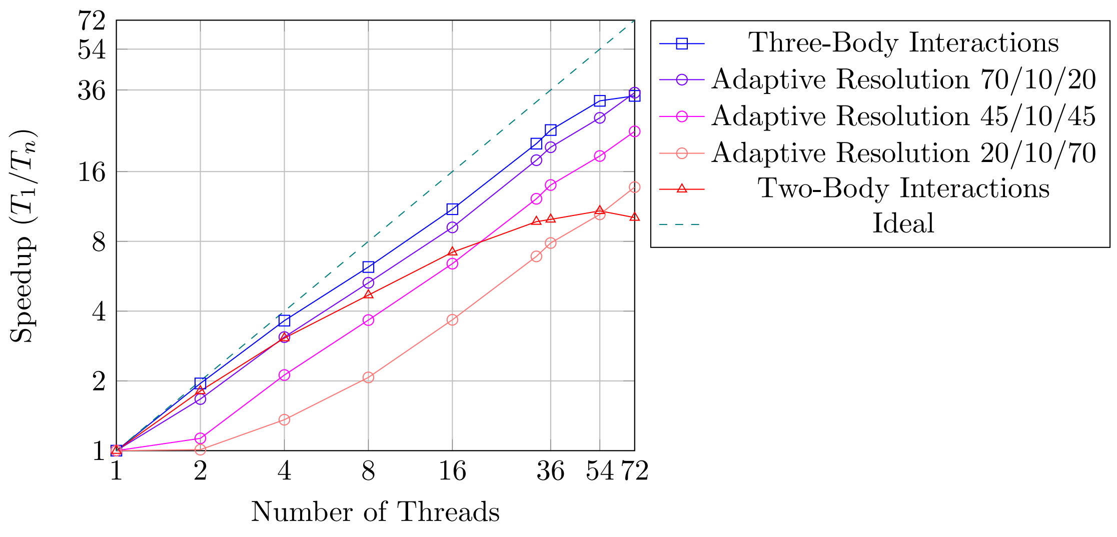


Figure 5.4.: Strong scaling speedup for shared memory parallelized three-body interactions, two-body interactions and adaptive resolution cases on HSUper.

The graphs show similar results like for CoolMUC-2. Three-body interactions scale the best here too. However, this time, there is no drop in improvements at a certain point because no hyperthreading is performed on HSUper. Like previously described, two-body interactions perform so poorly, even to the extent that the run time with 72 threads was higher than the time with 54 threads, because the domain and number of molecules is too small. In further measurements with larger domains and a bigger number of molecules, higher improvements were detected. The results of these measurements can be seen in appendix D. Running these larger simulations with three-body interactions, however, is not feasible. The convergence of the execution times of the adaptive resolution and the three-body measurements can still be seen. Additionally, the performance improvement of the three-body interactions reduces for the step from 54 to 72 threads. An explanation for this behavior is the imprecision of the measured wall times for this step, which can be seen in appendix C.

Weak Scaling

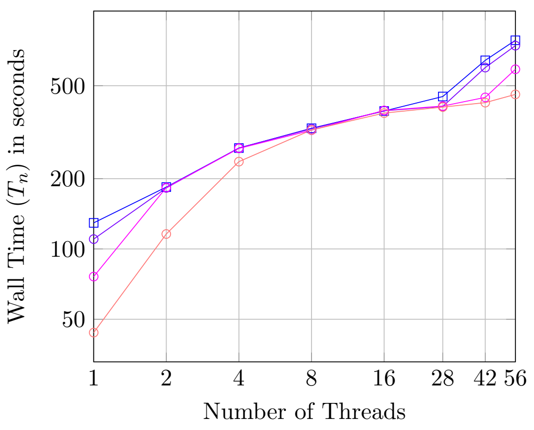
The weak scaling measurements on both clusters were performed with 4,700 molecules per thread. The domain size and number of molecules of the z-dimension never changed to guarantee an equal share of unnecessary force computations for all adaptive resolution cases and all numbers of threads. The size in this direction is 50 (170Å) and the number of molecules is 47. For 1 thread, the size in both of the remaining two directions is 11.25 (38.25Å), including 10 molecules.

When doubling the number of threads, the domain size and number of molecules in the x-direction and y-direction doubles alternately as well. In the first two doubling steps, the domain size doubles from 11.25 to 22.5. However, the number of linked cells only increases from 5 to 9. This might result in faster execution times than in cases where the number of linked cells also doubles.

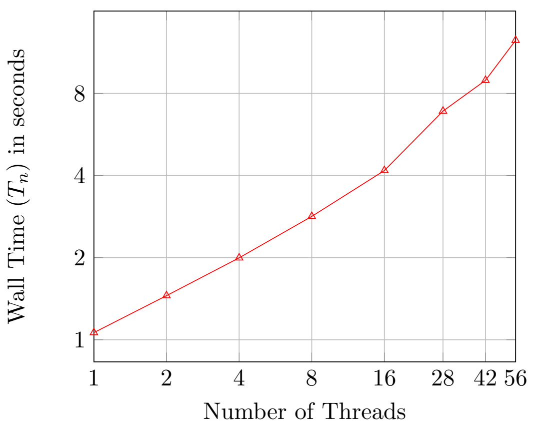
On CoolMUC-2, when increasing the number of threads from 16 to 28, the number of molecules is increased, accordingly, from 40 to 70. However, the domain size is increased from 45 to 77.5 instead of 78.75. This happens to align the boundary and the linked cell border. The effect of this discrepancy is that the mean distance between the molecule in this direction is now approximately 1.1 instead of the prior value 1.25. This could lead to worse performances due to additional force calculations because the neighbor's neighbor-molecule is now averagely 2.2 units away, and no longer 2.5, and, therefore, has a higher probability of being within the cutoff radius of 2.2 units. The shortened mean distance in one direction is kept for 42 and 56 threads.

The same applies to the measurements performed on HSUper with 36 or more threads. In these cases the new mean distance is approximately 1.1, as well.

Figure 5.5 shows the wall times of the runs on CoolMUC-2 and figure 5.6 those of the runs on HSUper. The weak scaling efficiencies can be seen in figures 5.7 and 5.8.

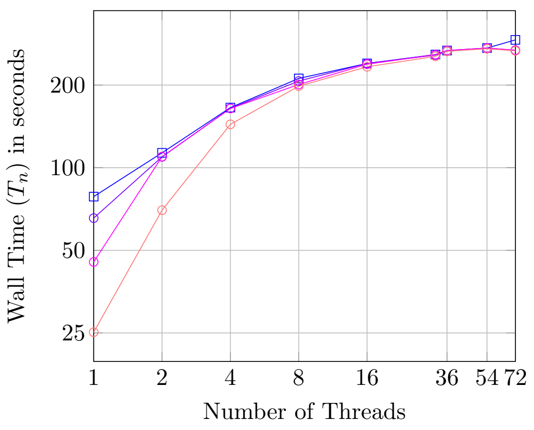


(a) Three-body interactions and adaptive resolution

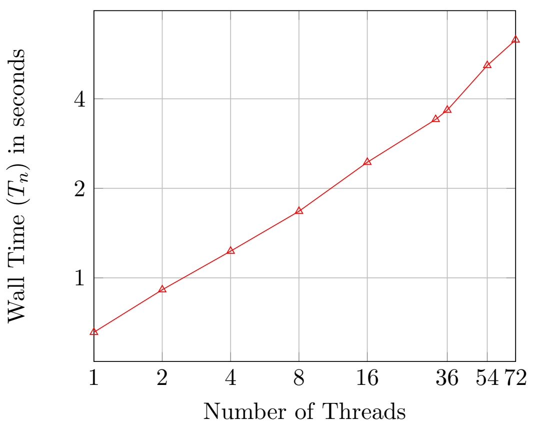


(b) Two-body interactions

Figure 5.5.: Weak scaling wall times of shared memory parallelization on CoolMUC-2.



(a) Three-body interactions and adaptive resolution



(b) Two-body interactions

Figure 5.6.: Weak scaling wall times of shared memory parallelization on HSUper.

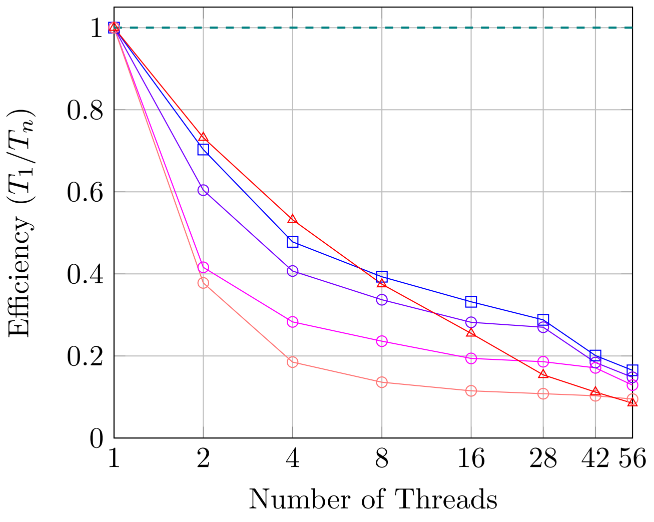


Figure 5.7.: Weak scaling efficiency for shared memory parallelized three-body interactions, two-body interactions and adaptive resolution cases on CoolMUC-2.

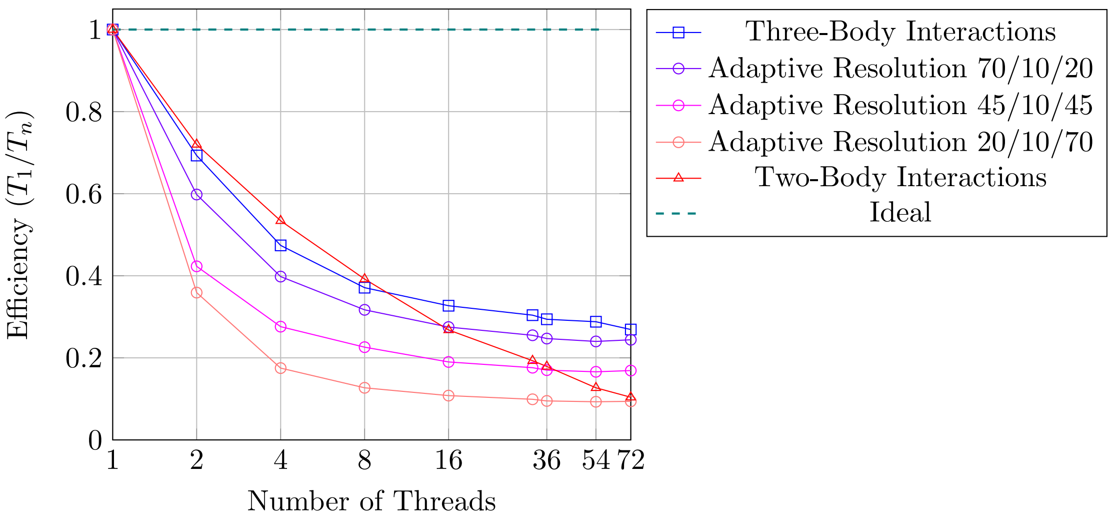


Figure 5.8.: Weak scaling efficiency for shared memory parallelized three-body interactions, two-body interactions and adaptive resolution cases on HSUper.

In figures 5.5a and 5.6a, we can observe logarithmic increases in wall time for three-body interactions and because of the convergence, for adaptive resolution as well. This results in an exponential decrease in efficiency, which can be seen in figures 5.7 and 5.8. Due to the convergence, the efficiency of the adaptive resolution cases is the lower, the larger the two-body area is. The performance improvements on the CoolMUC-2 cluster, again, significantly reduce for increases beyond 28 threads. This leads to a stronger increase in wall time and a stronger decrease in efficiency.

The wall times for two-body interactions on both clusters increase logarithmically as well, although it does not appear that way from the graphs because of the shifted y-axes. However, the increase is almost linearly. Thus, the efficiency continuously decreases and falls below the efficiency of all other cases at some point, although the rate of decline is lower than for the rest, for a low number of threads. The same behavior also occurs during measurements with bigger domains and more molecules per thread.

5.1.2. Distributed Memory Parallelization

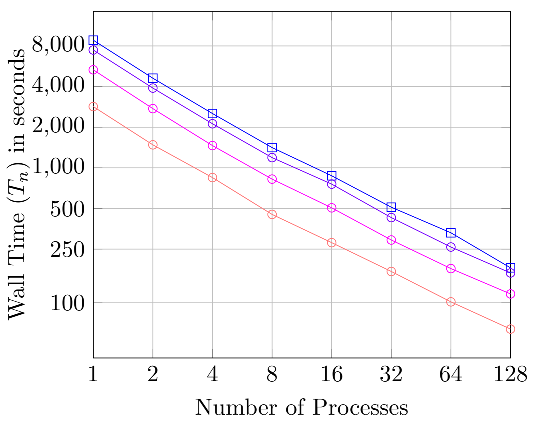
In contrast to the shared memory parallelization measurements, the wall times of distributed parallelization were measured with the same domain sizes, numbers of molecules, and numbers of processes on both clusters. The measurements were performed with 1, 2, 4, 8, 16, 32, 64, and 128 processes, distributed over up to 8 nodes on CoolMUC-2, to avoid hyperthreading, and over up to 4 nodes on HSUper, although only 2 would have been necessary.

SimpleMD allows for the specification on how the domain should be split among the processes. The information about the number of processes per direction is passed in the XML file. However, it is required that the number of linked cells in each direction is divisible by the number of processes in this direction. Hence, the domain sizes and numbers of molecules differ from the values used for shared memory parallelization.

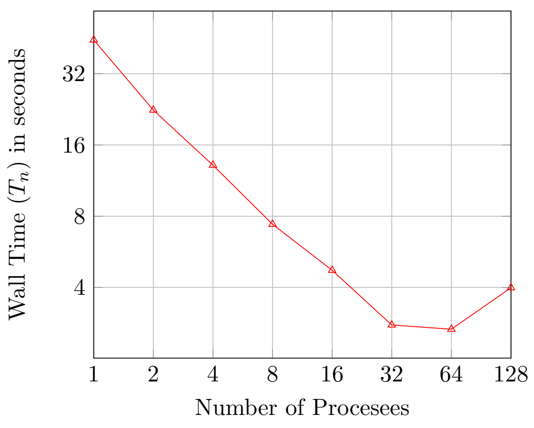
The effect that the adaptive resolution times and those of the three-body interactions converge, is avoided because in all simulations the domain was not divided in the dimension the adaptive resolution scheme is applied in. Therefore, the number of processes is doubled in the x-direction and y-direction alternately, up to 16 processes in the x-direction and 8 processes in the y-direction.

Strong Scaling

For the strong scaling measurements, the domain size was $(80, 40, 50)$ (corresponding to $(272\text{Å}, 136\text{Å}, 170\text{Å})$), resulting in $(32, 16, 20)$ linked cells per direction, including $(75, 37, 47)$ molecules per direction. This results in a total of 130,425 molecules, spread over 10,240 linked cells. Figure 5.9 shows the wall times on CoolMUC-2, figure 5.10 those on HSUper, and figures 5.11 and 5.12 the speedups on the respective cluster.

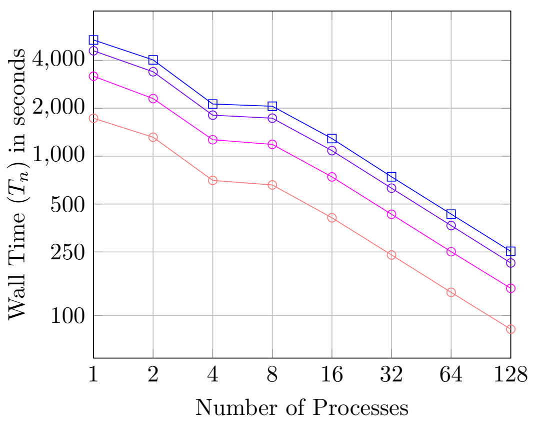


(a) Three-body interactions and adaptive resolution

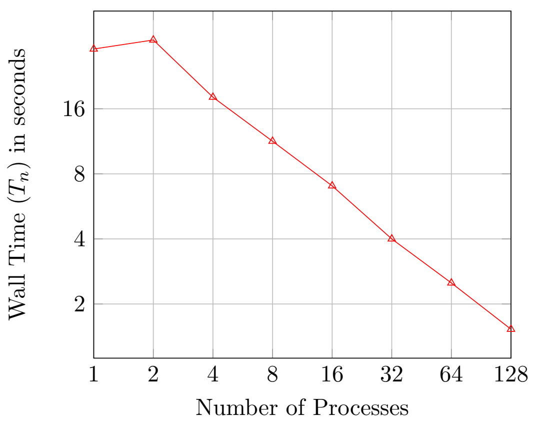


(b) Two-body interactions

Figure 5.9.: Strong scaling wall times of distributed memory parallelization on CoolMUC-2.



(a) Three-body interactions and adaptive resolution



(b) Two-body interactions

Figure 5.10.: Strong scaling wall times of distributed memory parallelization on HSUper.

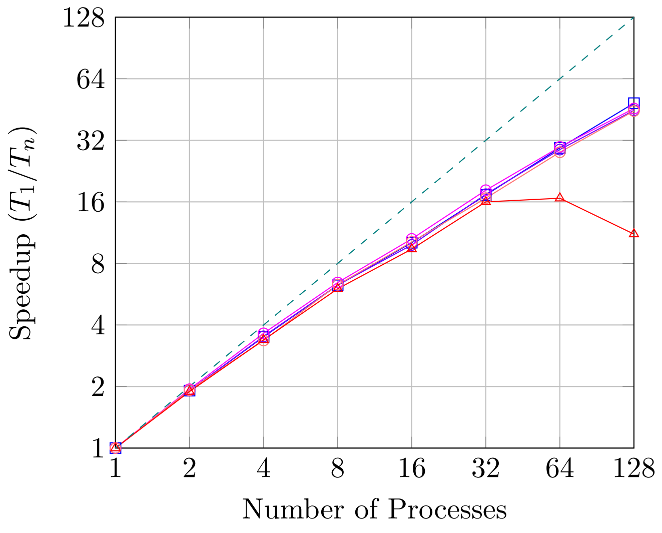


Figure 5.11.: Strong scaling speedup for distributed memory parallelized three-body interactions, two-body interactions and adaptive resolution cases on CoolMUC-2.

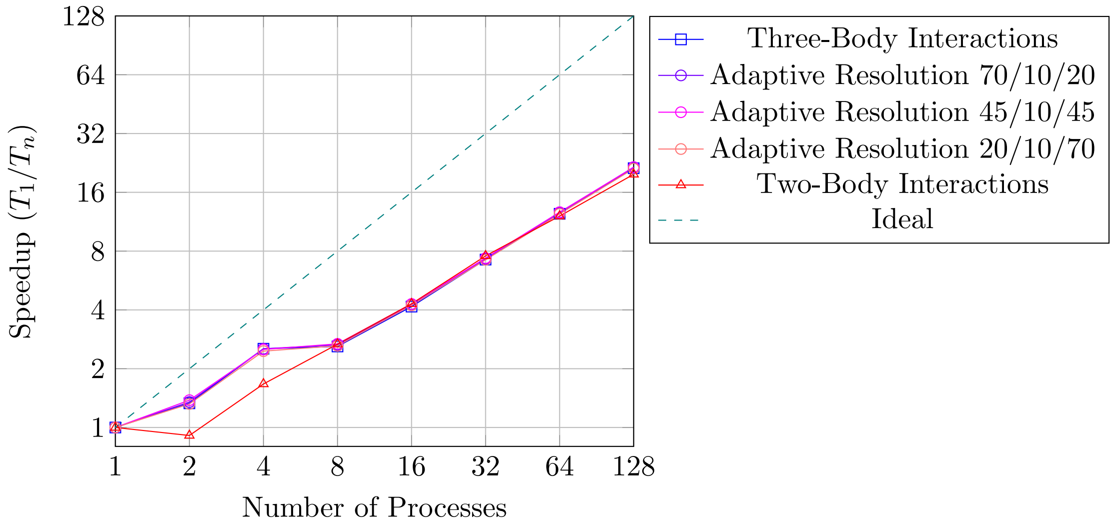


Figure 5.12.: Strong scaling speedup for distributed memory parallelized three-body interactions, two-body interactions and adaptive resolution cases on HSUper.

As can be seen, on CoolMUC-2 consistent speedups can be achieved with distributed memory parallelization. The performance deficits caused by the convergence and hyper-threading do not occur here for the reasons stated before. Similar to the values of shared memory parallelization, the performance drastically reduces for two-body interactions and a bigger number of processes because the domain is too small and the communication overhead is bigger than the performance improvement. For drastically larger domains two-body interactions, have significantly higher speedups because the larger the domain, the lower the difference between the perfect speedup and the actual speedup. Figures showing the improved performance with larger domains are in appendix D.

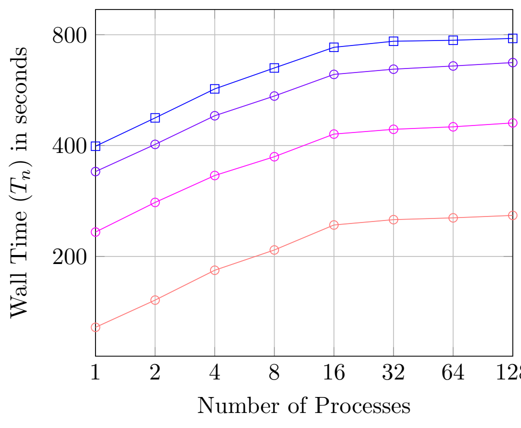
However, on HSUper, the total speedup is significantly lower than on CoolMUC-2, most likely due to higher relative communication times. Additionally, the three-body interactions and all adaptive resolution cases overperform in the measurements with 4 processes, and the two-body interactions underperform with 2 processes. Unfortunately, the reason for these under- and overperformances could not be determined.

Weak Scaling

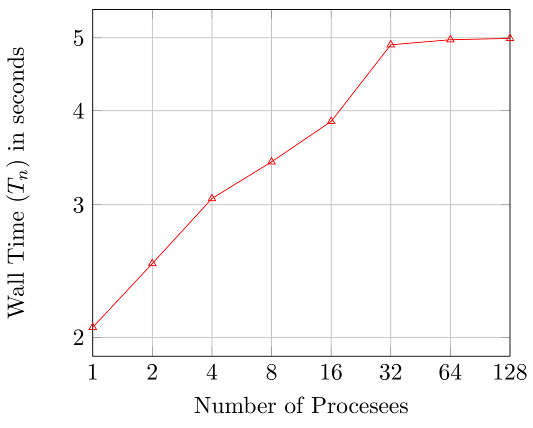
The weak scaling measurements were performed with 6,768 molecules per process. Like for shared memory parallelization, the number of molecules and the domain size of the z-dimension never changed, as well as the number of processes in this direction. Like before, the size in this direction is 50 (170Å) and the number of molecules is 47. The remaining two dimensions start with a domain size of 12.5 (42.5Å), including 12 molecules, and 1 process in each direction.

The number of processes are doubled alternately in the x-direction and y-direction. When doubling the number of processes in one direction, the domain size and number of molecules, in this direction, are doubled as well. Thus, each process works on a subdomain with the size $(12.5, 12.5, 50)$, containing initially an average of $(12, 12, 47)$ molecules per direction. This is a total of 500 linked cells per thread.

The first pair of figures below shows the wall times measured on CoolMUC-2, the second pair shows the times on HSUper, and the remaining two figures portray the efficiencies on the respective clusters.

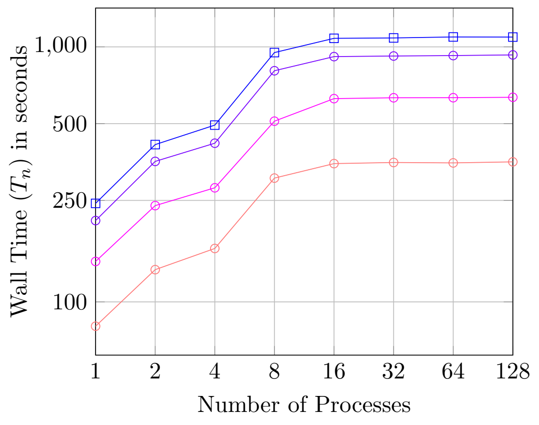


(a) Three-body interactions and adaptive resolution

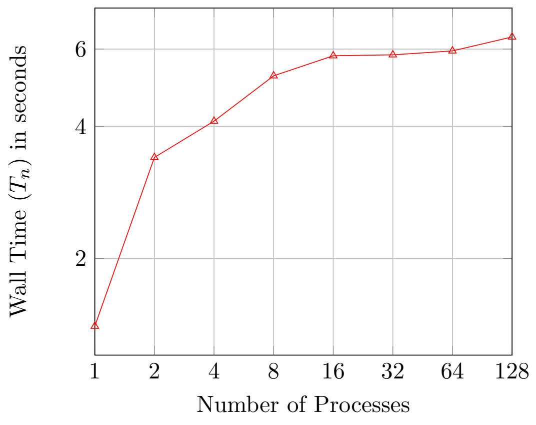


(b) Two-body interactions

Figure 5.13.: Weak scaling wall times of distributed memory parallelization on CoolMUC-2.



(a) Three-body interactions and adaptive resolution



(b) Two-body interactions

Figure 5.14.: Weak scaling wall times of distributed memory parallelization on HSUper.

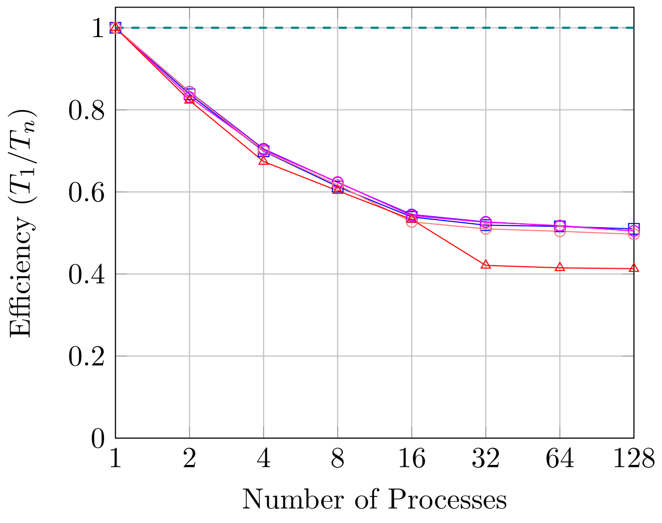


Figure 5.15.: Weak scaling efficiency for distributed memory parallelized three-body interactions, two-body interactions and adaptive resolution cases on CoolMUC-2.

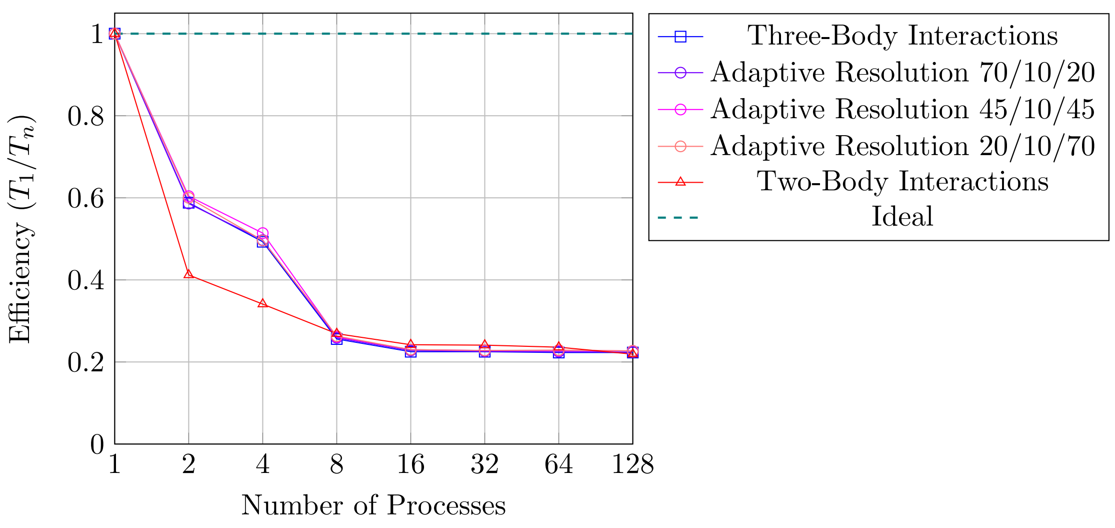


Figure 5.16.: Weak scaling efficiency for distributed memory parallelized three-body interactions, two-body interactions and adaptive resolution cases on HSUper.

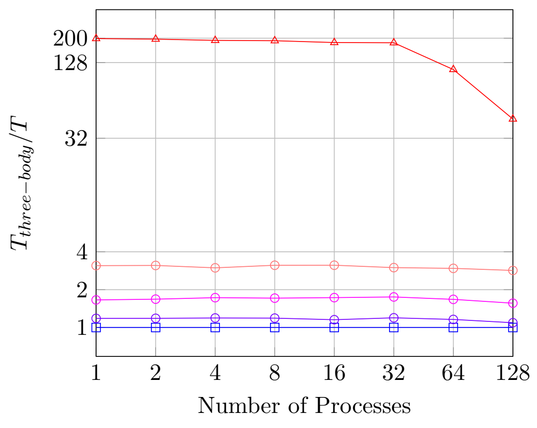
In the figures, we can see logarithmic increases in all wall times. These lead to the exponential decrease in efficiencies. Like in the strong scaling graphs, the overperformance of the execution with three-body interactions and adaptive resolution, using 4 processes, and the underperformance of two-body interactions with 2 processes on HSUper is detectable. This results in a worse efficiency for the two-body case with 2 and 4 processes in comparison to the respective values for three-body interactions and adaptive resolution. Otherwise, the efficiency for three-body interactions, two-body interactions, and adaptive resolution are similar. On CoolMUC-2, there is an excessive step in wall time, and therefore also in efficiency, from 16 to 32 processes. This happens most likely due to the low execution times. The majority of this time increase is caused by the increase in communication which is necessary because, beginning at 32 processes, the processes are distributed across multiple nodes, each containing 16 processes. However, this effect does not happen on HSUper. Possible reasons are the higher execution times and the significantly lower efficiencies, and most likely a significantly faster communication between the nodes.

5.2. Comparison of Three-Body Interactions, Two-Body Interactions, and Adaptive Resolution

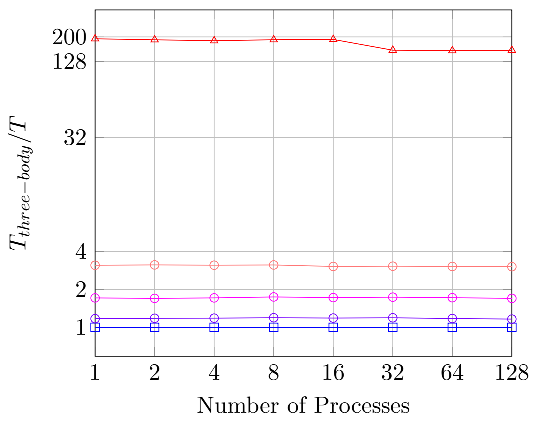
All results presented in this chapter are based on the measurements performed for the analysis of scalability. For the respective setups and wall times refer to the descriptions and figures in chapter 5.1.

In the following chapters, the execution times of two-body interactions, three-body interactions, and the 3 adaptive resolution cases are compared and analyzed. Chapter 5.2.1 focuses on the measured times when distributed memory was applied and chapter 5.2.2 on the results of shared memory parallelization.

5.2.1. Distributed Memory Parallelization

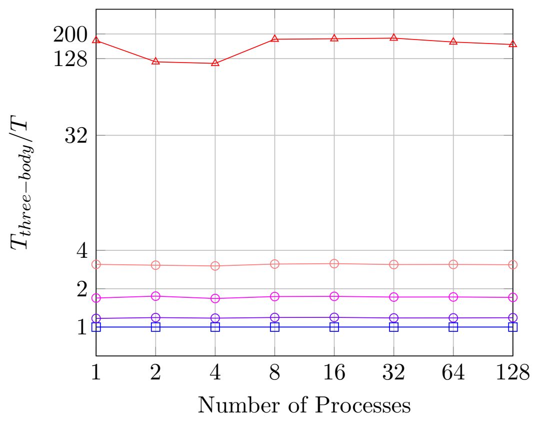


(a) Strong scaling

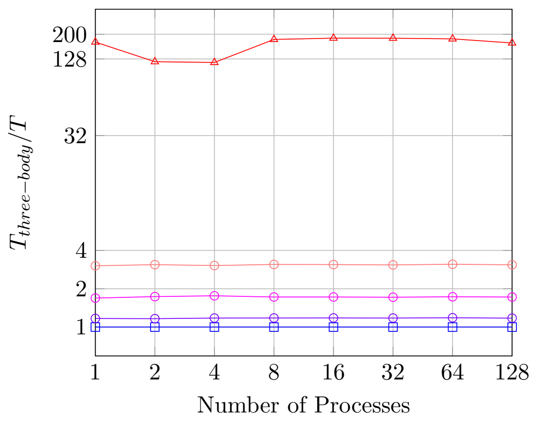


(b) Weak scaling

Figure 5.17.: Comparison of wall times for distributed memory parallelized three-body interactions, two-body interactions, and adaptive resolution cases on CoolMUC-2.



(a) Strong scaling



(b) Weak scaling

Figure 5.18.: Comparison of wall times for distributed memory parallelized three-body interactions, two-body interactions, and adaptive resolution cases on HSUper.

All figures above show the same behavior. In all cases the respective adaptive resolution cases are approximately 1.2, 1.7, and 3 times faster than the three-body interactions. The two-body interactions are approximately 190 times faster on CoolMUC-2 and roughly 180 times faster on HSUper.

Additionally, some phenomena, that have already been mentioned prior, can be seen here as well. These are the lack of improvement for a high number of processes due to a smaller domain (see figure 5.17a), the step from 16 to 32 processes (see figure 5.17b), and the overperformance and underperformance on HSUper (see figure 5.18). Due to the step in weak scaling wall times on CoolMUC-2, the improvement of two-body interactions in comparison to their three-body counterpart is reduced to a factor of approximately 155. The reduced factor of improvement on HSUper, because of the overperformance of 4 processes in all cases except the two-body interactions and the under performance of the two-body interactions with 2 processes, is roughly 120.

The numbers show, that the run time of two-body force computations is really low in comparison to the time three-body force calculations take. Thus, the time improvements of the adaptive resolution cases approximately match the ratio of total cells to cells which need to initiate three-body force computations. All adaptive resolution versions contain 20 planes of cells in the z-direction, in which the scheme is applied in. Out of those 20 cell-planes 17, 12, and 7 planes respectively need to initiate the computations. The ratios $\frac{20}{17} \approx 1.18$, $\frac{20}{12} \approx 1.67$, and $\frac{20}{7} \approx 2.86$ approximately match the speedups, with respect to slight measurement inaccuracies.

5.2.2. Shared Memory Parallelization

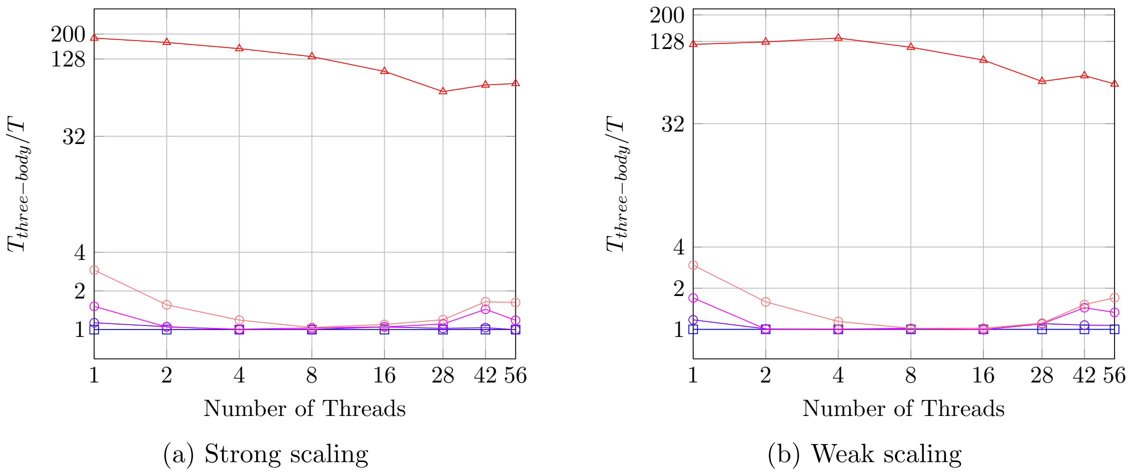
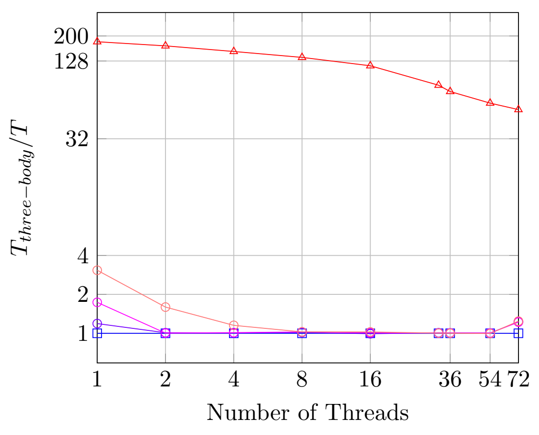
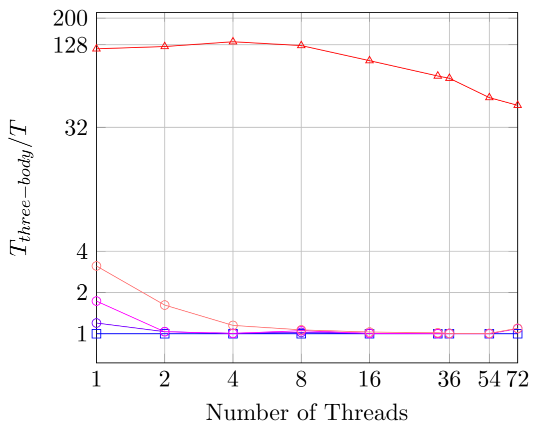


Figure 5.19.: Comparison of wall times for shared memory parallelized three-body interactions, two-body interactions, and adaptive resolution cases on CoolMUC-2.



(a) Strong scaling



(b) Weak scaling

Figure 5.20.: Comparison of wall times for shared memory parallelized three-body interactions, two-body interactions, and adaptive resolution cases on HSUper.

As can be seen in all figures above, the performances of all adaptive resolution cases and three-body interactions converge for an increasing number of threads. The reason is that OpenMP, when run on either of the clusters, by default uses static scheduling to distribute the iterations among the threads. The loop executions get split into chunks of $\frac{loop\ executions}{threads}$ iterations and each thread gets assigned to one of these. [Boa15] However, the iteration in the z-direction, in which the adaptive resolution scheme is applied in, is the top level iteration. Thus, for a big number of threads, the chunks contain only cells in a smaller number of different xy-planes along the z-axis. The larger the share of the domain that needs to initiate three-body force calculations, the lower the minimum number of threads such that there is at least one thread which has to initiate three-body interactions for all cells in its chunk. If there is at least one such thread in each of the red-black traversal groups, this threads works as long as any thread in the simulation with three-body interactions. Any other thread finishing earlier has to wait for the three-body-like threads to finish. Thus, the adaptive resolution simulation takes as long as a three-body simulation if there are enough threads.

The two-body performances in the strong scaling measurements significantly reduces in comparison to the three-body performances. The reason are the low execution times and the relatively higher performance overhead for maintaining the threads. The performances in comparison to three-body interactions in the weak scaling measurements first improve and then drastically reduce. The reason, which has already been mentioned in chapter 5.1.1, is that the efficiency of three-body interactions first drops steeper but then levels off, whereas the efficiency of the two-body interactions proceeds to drop even further.

Figure 5.19 shows, that the simulations using only three-body interactions suffered most from the performance deficits caused by hyperthreading. Two-body simulations achieve short-term improvements against three-body interactions, the moment hyperthreads are used, before following the old trend when the number of threads increases. The adaptive resolution simulations can perform consistently better when hyperthreading is used because some threads, as described before, may finish significantly faster than others. In that case they do not require the computational power of a physical core and each of the threads still computing can ideally work on one physical core alone.

Part III.

Future Work and Summary

6. Future Work

This work focused on the extension for SimpleMD introducing Axilrod-Teller based three-body interactions, and a simple adaptive resolution scheme involving these and the Lennard-Jones based two-body interactions and analyzed the performance of these implementations. Therefore, this chapter introduces further enhancements which would be subsequent additions based on the extensions, and shows options to further investigate some phenomena we could observe during the performance analysis.

Currently, adaptive resolution can only be applied in one dimension. In this direction the order of areas is defined as the two-body area followed by the interface layer and the three-body region. The next enhancement step would be to grant the user the freedom to switch the placement of the two-body and three-body areas in this direction. Additionally, an implementation of the scheme for the remaining boundary types will be useful. Another potential addition could be allowing the interface layer to be skew in the domain or even to have any other form than a plane. However, such ideas, especially the latter one, are complicated to implement.

Further possible extensions for SimpleMD are the implementations of new potentials. These can cover long range forces and forces involving more than three particles. If there are more potentials available, a further extension of the adaptive resolution scheme, which includes more than two different potentials or lets the user chose the potentials, is possible.

In chapter 5.1.2, we determined the overperformance of three-body interactions and all adaptive resolution cases with 4 processes, and the underperformance of two-body interactions with 2 processes on HSUper. A deeper investigation is required to determine why this phenomenon happens.

In the same chapter, we saw a step in execution times from 16 to 32 processes on CoolMUC-2. We assume that the reason for this step is an increase in communication times due to the use of multiple nodes. However, this assumption still needs to be proven or refuted.

In chapter 5.2.2, we saw and explained the convergence of three-body and adaptive resolution wall times. There is a possibility for further analysis on how the execution times behave when the scheme is applied in the other dimensions and how different OpenMD scheduling modes affect the results.

In general, all measurements of the adaptive resolution times were performed with interface layers of a length of 10% of the domain size. Further investigation is possible on how the performance changes when the size of the interface layer differs.

7. Summary

The aim of this thesis was to give an overview of the basic structure of molecular dynamics simulations and to explain how SimpleMD is used to run MD simulations. Furthermore, we presented the extensions for three-body interactions and adaptive resolution made in SimpleMD, and analyzed their scalability and how they perform compared to each other.

We have presented the approach that SimpleMD uses to run a two-body simulation, based on Lennard-Jones potentials, a linked cell method and a velocity Verlet time integration. The simulation is executed in one central class, which uses configurations to parse the input and set up the system. It operates via services that provide necessary functionality, especially the ability to apply mappings to molecules and cells.

Additionally, we have explained how the three-body extension, based on the Axilrod-Teller potential, uses the structures that were already present to integrate the additions. The implementation was based on one new mapping, or rather two of which one is not used. It required additional functions in existing classes to be able to utilize the new mapping and transport the coefficient to the object.

The implementation of the simple adaptive resolution case made use of both the prior existing two-body implementation and the three-body extension, to allow for simulations which make use of both types of forces in distinct areas of the domain and offer a smooth transition between these regions. The implementation was centered around another new mapping, or, similar to the three-body extension, two of which one is not used. We showed that each type of force only needs to be calculated in certain shares of the domain, which leads to performance improvements. This extension, again, needed changes in existing structures to utilize the new mapping and, especially, to store partial forces.

Besides, we discussed how preprocessor directives enabled the usage of only certain parts of the code, depending on the choice of compile options. We showed that it is possible to choose either of the two potentials as a basis for the force calculations, and to choose to ignore the choice about the potential if adaptive resolution is enabled.

We examined that the parallelization of the simulation, in a shared and distributed memory manner, resulted in consistent and satisfying performance improvements.

Moreover, we determined that the performance improvement of adaptive resolution over three-body simulations is consistent and directly related to the share of the domain which has to perform three-body force calculations. However, the performance improvement decreases for an increasing number of threads when a shared memory parallelization is used because the workload of single threads increasingly resembles the workload of its three-body counterpart.

Finally, we gave a brief outlook on possible further extensions of SimpleMD and future research that can be conducted based on the extensions.

Part IV.

Appendix

A. Detailed Description of Simple MD

A.1. Structure

This chapter contains more detailed information about the structure and classes of SimpleMD. Most of the aspects described in this chapter are copied from chapter 3.5, all pictures are omitted. Since there would not be any further information about the subdirectories *configurations* and *services*, these sections are omitted as well.

A.1.1. Main Directory

Main Methods: The main directory contains two files providing `main` methods. The main method of `main.cpp` sets up, runs, and shuts down one MD simulation, whereas the main method of `main_multi.cpp` does the same with multiple simulations at the same time.

Constants: Global constants are defined in the files `MolecularDynamicsDefinitions.h` and `MolecularDynamicsUserInput.h`. The latter file generates constants based on the CMake compile options. These are used in preprocessor directives in entire SimpleMD. The former file defines the number of neighbors of each linked cell depending on the number of dimensions defined in the user input, the number π, a tolerance of 10^{-13}, and the boundary types, which all are used outside of preprocessor directives.

BoundaryTreatment: The class provides functions for filling and emptying ghost cells. These are mainly used for periodic and parallel boundaries, and in distributed memory parallelization. It enables sending molecules from one subdomain to another, and for sending them from one side of the domain to the other if periodic boundaries are used. However, the functionality of the `BoundaryTreatment` will not be described any further because the focus of this thesis is set on reflecting boundaries. Furthermore, the implementation of three-body interactions and the simple adaptive resolution scheme happened on a cell-level and molecule-level and not on a (sub-)domain-level, where the `BoundaryTreatment` operates.

MolecularProperties: The class stores the properties that all molecules have in common. It contains the mass of the molecules, the Lennard-Jones parameters, the cutoff radius, and Boltzmann's constant.

Molecule: The class contains all properties that differ from molecule to molecule. These are the position, the velocity, the forces of the current and previous time step, the potential energy, and an id.

LinkedCell: This class provides the functionality to store molecules in a list, access this list and iterate over it.

MolecularDynamicsSimulation: This class runs the MD simulation. Its methods allow to initialize the system, simulate one time step, run the entire simulation, and shut down the system. During the initialization, it uses the functions of the directory configurations and during each time step, it operates by using the services. The control flow of SimpleMD is described in chapter 3.6.

A.1.2. Subdirectories

Cell Mappings

All cell mapping classes are structured similarly. They feature the methods `beginCellIteration` and `endCellIteration`, setting up local attributes before the first cell is handled and winding up everything after the last cell has been handled. However, most of the times these methods serve no function. If they serve a function, it is often in the context of file I/O for plotting or writing checkpoint files. Additionally, there are the two methods `handleCell` and `handleCellPair`. They receive one or two cells respectively and their indices, and apply the mapping to the molecules in these cells. The cell mappings are:

ComputeMeanVelocityMapping: This mapping computes the mean velocity of all molecules in the local domain and writes it to a file if defined.

ComputeTemperatureMapping: This mapping uses the mean velocity to calculate the temperature of the subdomain. Both this and the `ComputeMeanVelocityMapping` are not relevant for the functionality of the simulation, instead they create statistical values.

DeleteMoleculesMapping: Molecules that left the local domain during one time step are no longer used because they moved to another subdomain or left the simulation. This mapping, therefore, deletes all molecules from both, the passed cell and the `MoleculeService`. It is used for ghost cells after the force calculations in distributed memory parallelization simulations.

EmptyLinkedListMapping: After the last cell mapping has been applied in a time step, this mapping is used to remove all molecules from each cell because, after time integration, they might belong in another cell.

LennardJonesForceMapping: This mapping computes the Lennard-Jones forces, as described in chapter 3.3.1 for all molecule-pairs in one cell if `handleCell` is called, and for all molecule-pairs with one molecule being located in each of the two cells if `handleCellPair` is executed. The forces are added to or subtracted from the force buffer of the molecules. Additionally, if `handleCell` is called, constant external forces are added to the force buffer of each molecule. The external forces are added by the `ExternalForceService`, which is mentioned in chapter 3.5.2.

LennardJonesPotentialEnergyMapping: The mapping works like the force mapping. It calculates the potentials and adds them to the attribute or subtracts them from it.

ResetPotentialEnergyMapping: The potential energy attributes of all molecules in the cell are set to 0. Both this and the `LennardJonesPotentialEnergyMapping` are never used in a SimpleMD simulation.

VaryCheckpointMapping: This mapping is used every 500 time steps. It calculates the mean velocity of all molecules in a cell and sets the velocities of all molecules to a random value close to the mean velocity.

The remaining mappings are the `CopyMoleculesMapping`, copying all molecules to a list, the `ProfilePlotterMapping` and `RDFMapping`, used for statistical purposes, and the `CollectMoleculesMapping`, `ParallelBoundaryEmptyCellsMapping`, `PeriodicAndParallelBoundaryFillCellsMapping`, and `PeriodicBoundaryEmptyCellsMapping`, used in distributed memory parallelization and simulations with parallel or periodic boundaries.

Molecule Mappings

Like the cell mappings, all molecule mappings have a similar structure. This structure contains the methods `beginMoleculeIteration` and `endMoleculeIteration`, which serve the same purpose as their equivalents in the cell mappings, and `handleMolecule` which applies the mapping to a passed molecule. The molecule mappings are:

ComputeMeanVelocityMapping: This mapping has the same purpose as its equivalent in the cell mappings.

SetMeanVelocityMapping: Forces the molecules in the subdomain to have a chosen mean velocity by adding the difference between the old average velocity and the desired one to all molecules. Both mean velocity molecule mappings are never used in a SimpleMD simulation.

InitialPositionAndForceUpdate: The mapping replaces the time integration in the initial time step. Its functionality is described in chapter 3.3.2. After the calculation it stores the force calculated in the initial time step in the buffer for the force of the previous time step and resets the force buffer to 0.

UpdateLinkedCellListMapping: This mapping sorts the molecules into the linked cells according to the coordinates. It is used at the end of every time step.

VelocityStoermerVerletMapping: This mapping applies the time integration as described in chapters 2.2 and 3.3.2. Similar to the `InitialPositionAndForceUpdate`, the force of the current time step is stored in `forceOld` and the buffer `force` is reset to $\mathbf{0}$.

The remaining molecule mappings are the `Adios2Writer`, the `VTKMoleculeWriter`, and the `WriteCheckPointMapping`. As the name suggests, all these mappings write different types of checkpoint files.

A.2. Control Flow

A.2.1. Initialization

The main method receives one input argument, the XML file. First, an object of the class `MolecularDynamicsConfiguration` is instantiated and, thus, objects of all other configuration classes as well. This object parses the XML tag *molecular-dynamics*. This tag contains all subtags and key-value-pairs needed for SimpleMD. Then, an instance of `MolecularDynamicsSimulation` is created. This object received the instance of the configuration. Finally, the method `initServices` of the `MolecularDynamicsSimulation` object is called.

During the execution of `initServices`, firstly all objects of the service classes are created by passing them the values extracted from the XML file by the configuration objects. The `MoleculeService` can either get the information about all molecules from a checkpoint file if one has been specified in the subtag *domain-configuration*, or otherwise it generates the molecules according to the values of *molecules-per-direction*. After that, all mappings are created. Necessary values, again, have been extracted from the input file.

In the final step of `initServices`, the molecules are sorted into linked cells and the force computation happens. After the forces of all molecules have been calculated, the molecules are removed from their cells, the special version of the time integration in the initial step is executed, and the molecules are put back into (potentially new) cells.

A.2.2. Time Step

After `initServices` is finished, the main method calls `simulateOneTimestep` of the `MolecularDynamicsSimulation` as many times as the number of time steps, defined in the XML file, requires. The method `runSimulation` can be executed as well, as it also executes the same loop.

During one time step, first, the `BoundaryTreatment` prepares the ghost cells, prior to the force calculation. For the force computations, the simulation object calls `iterateCellPairs` of the `LinkedCellService` and passes the instance of the `LennardJonesForceMapping`, created in the initialization. Then, mappings for statistical purposes, plotting, and writing checkpoint files are applied. In certain intervals the order of the molecules in the `MoleculeService` gets changed, to improve the efficiency. In usually much larger intervals, the `VaryCheckpointMapping` gets applied. In the end, all molecules are removed from the linked cells, before executing the time integration and putting them back into the linked cells. For all molecule mappings, including the time integration, the method `handleMolecules` of the `MoleculeService` is called and the instance of the mapping is passed as input argument. After the conclusion of one time step, the `main` method calls the method again, until all time steps have been simulated.

A.2.3. Shutdown

In the end the method `shutdownServices` is called on the instance of `MolecularDynamicsSimulation`. This method, first, calls `shutdown` on all services which store objects in a data structure, like the `LinkedCellService` and the `MoleculeService`. This method deletes all stored objects and sets up an empty version of the used data structure. Finally, the destructors of all by the `MolecularDynamicsSimulation` initialized objects are executed with the delete operator. The `MolecularDynamicsSimulation` object and all instances of the configuration classes are deleted when the program terminates.

B. Creation of Checkpoint Files

```
Algorithm 2: createCheckpointFile
   Input:   XML file
   Output:  none, checkpoint file is created
1  Function createCheckpointFile($\mathbf{x}_{domain}$, $\mathbf{d}_{domain}$, $\mathbf{n}$):
       // Create checkpoint file, write total number of molecules and
          number of dimensions in it
2      chunkSize ← $\begin{pmatrix} \frac{d_{domain,x}}{n_x} \\ \frac{d_{domain,y}}{n_y} \\ \frac{d_{domain,z}}{n_z} \end{pmatrix}$
3      for $k \leftarrow x_{domain,z}$ to $x_{domain,z} + d_{domain,z}$ stepsize $chunkSize_z$ do
4          for $j \leftarrow x_{domain,y}$ to $x_{domain,y} + d_{domain,y}$ stepsize $chunkSize_y$ do
5              for $i \leftarrow x_{domain,x}$ to $x_{domain,x} + d_{domain,x}$ stepsize $chunkSize_x$ do
                   // The chunk starts at $\begin{pmatrix} i \\ j \\ k \end{pmatrix}$ and ends at
                      $\begin{pmatrix} i \\ j \\ k \end{pmatrix}$ + chunkSize.
6                  $\mathbf{x}_{molecule}$ ← random, uniformly distributed point, with each
                    coordinate in the inner 40% of the chunk in the respective direction
7                  $\mathbf{v}_{molecule}$ ← random velocity in each dimension, distributed according
                    to Gaussian distribution with $\mu = 0$ and $\sigma = 1$
                   // Write $\mathbf{x}_{molecule}$ and $\mathbf{v}_{molecule}$ (and $\mathbf{0}$ as force) to the
                      checkpoint file
       // Close checkpoint file
```

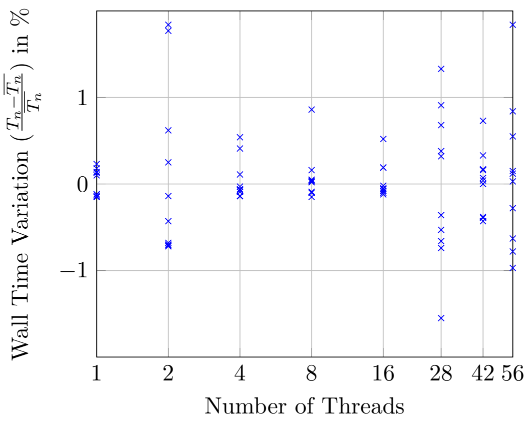
Figure B.1.: The algorithm used for the creation of checkpoint files in the corresponding Python script. Reading the XML file is omitted. Instead the domain offset $\mathbf{x}_{domain}$, domain size $\mathbf{d}_{domain}$, and number of molecules per direction $\mathbf{n}$ are treated as input arguments. The checkpoint file is created and given the name, defined in the XML file. The current force of each molecule is set to $\mathbf{0}$ because it is never used.

The molecules are distributed in a grid-like structure and within the center of their chunk to guarantee a minimal distance between them. If two molecules were too close, the forces of these molecules get very big. This leads to too high velocities and possibly a position, that is so far outside of the domain that after one reflection the molecule is still outside of the domain but on the opposite side. This case is not expected by SimpleMD and would lead to a termination of the program.

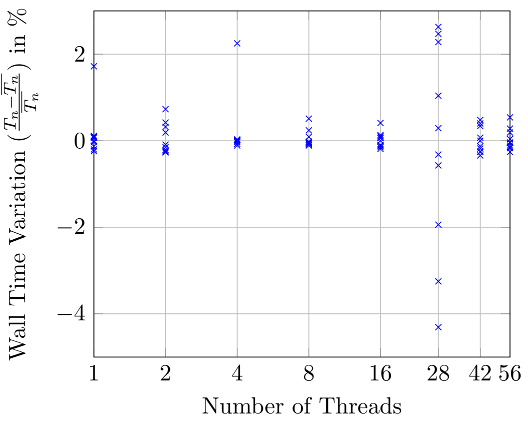
C. Consistency of the Measured Times

In this chapter, the consistency of the measured wall times is analyzed. Each of the wall times of all three-body interaction cases presented in chapter 5.1 (shared and distributed memory parallelization, all different numbers of processes, and on both clusters) has been measured 10 times. The arithmetic mean of all values except the lowest and highest has been used for the analysis in the chapters 5.1 and 5.2. All values' deviation from the mean value are shown and discussed below. The values for all adaptive resolution and two-body interaction cases have been determined similarly but with less measurements.

CoolMUC-2

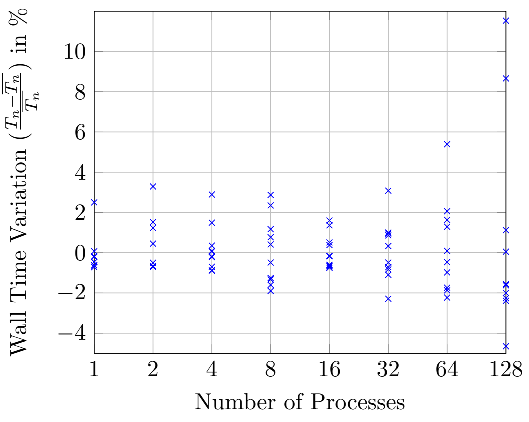


(a) Strong scaling

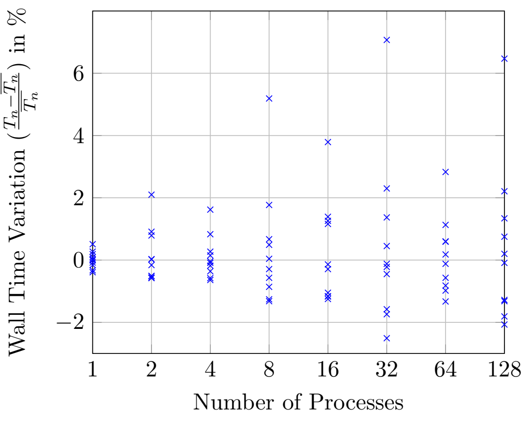


(b) Weak scaling

Figure C.1.: Wall time deviations from the mean value of three-body interactions, performed with shared memory parallelization on CoolMUC-2.



(a) Strong scaling



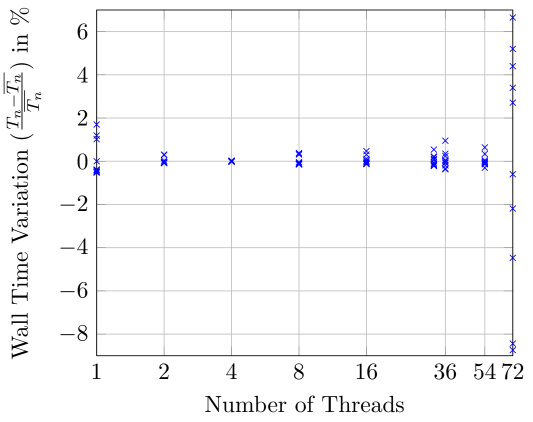
(b) Weak scaling

Figure C.2.: Wall time deviations from the mean value of three-body interactions, performed with distributed memory parallelization on CoolMUC-2. For strong scaling with 1 process, the value +99.25% does not fit in the figure.

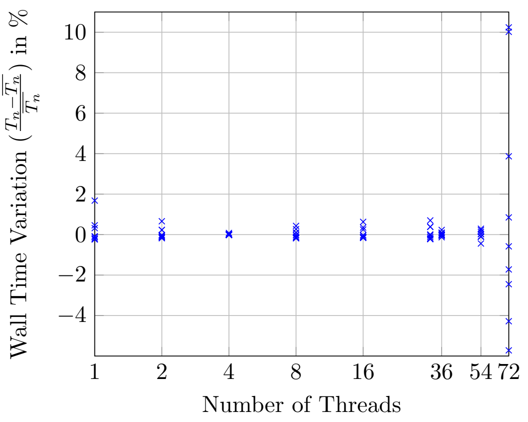
The figures show, that measured times of shared memory parallelization are very precise. Almost all measured times are within a deviation of 1% from the mean value. Bigger deviations occur quite rarely. However, if 28 threads are used, the measured times do not concentrate around the mean value, like seen in all other cases. Fortunately, the deviations in that case are relatively low, although they are larger than with any other number of threads.

On the other hand, most times measured using distributed memory parallelization are within a deviation of 3% from the mean value. However, measurement results outside this 3% margin occur quite frequently. The more processes are used, the more inconsistent are the measured times.

HSUper

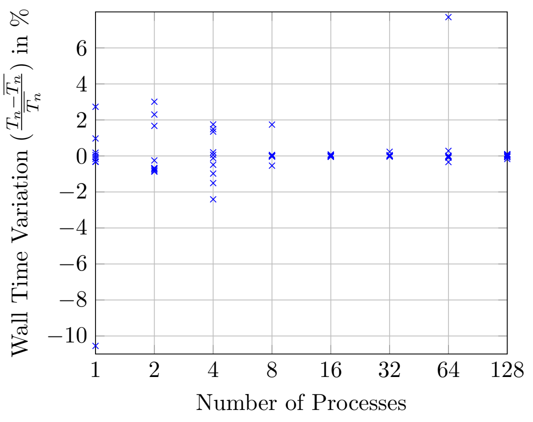


(a) Strong scaling

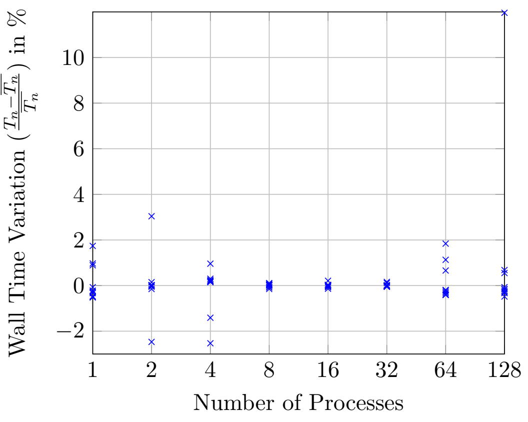


(b) Weak scaling

Figure C.3.: Wall time deviations from the mean value of three-body interactions, performed with shared memory parallelization on HSUper. For weak scaling with 72 threads, the value −36.58% does not fit in the figure.



(a) Strong scaling



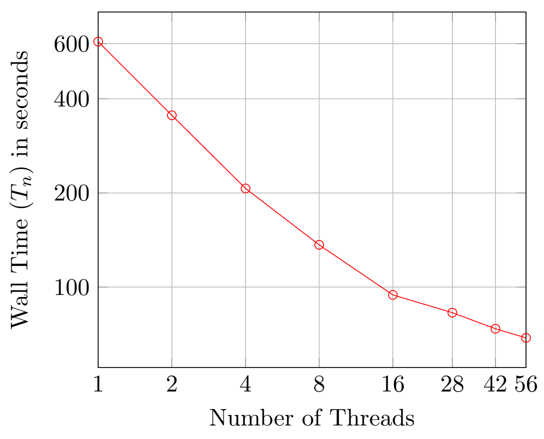
(b) Weak scaling

Figure C.4.: Wall time deviations from the mean value of three-body interactions, performed with distributed memory parallelization on HSUper.

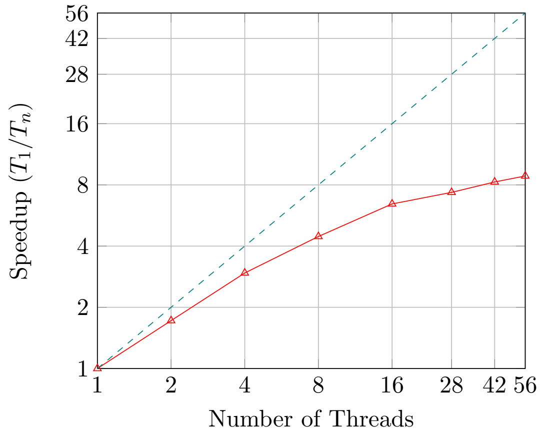
As can be seen, the measurements of the shared memory parallelization are very precise except for 72 threads. Most values are really close to the mean wall time and the deviation of the actual times is normally within 2%, and for the vast majority even within 1%. The probability of high deviations is quite low. However, if the simulation is measured with 72 threads, similar to measurements on CoolMUC-2 with 28 threads, no concentration of measurement results can be detected. The measured times are distributed among a pretty large time interval.

The measurements of the distributed memory parallelization are not quite as precise as the ones for shared memory parallelization but they are nevertheless acceptable. Concentrations of measurement results around the mean value are present for all numbers of processes. The deviation of the majority of the times from the mean value is below 3% for low numbers of processes and gets lower when the number of processes increases. Single results with high deviations seem to occur for all numbers of processes sometimes, but still significantly less frequently than on CoolMUC-2.

D. Additional Measurements with Larger Domains

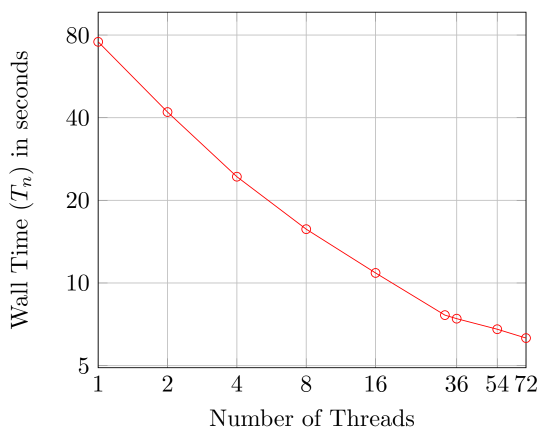


(a) Wall time

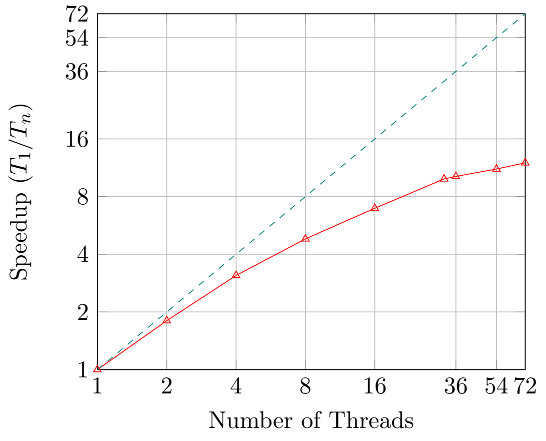


(b) Speedup

Figure D.1.: Strong scaling wall time and speedup of shared memory parallelization on CoolMUC-2 with two-body interactions. The domain size was (200, 200, 50) and the number of molecules was (188, 188, 47).

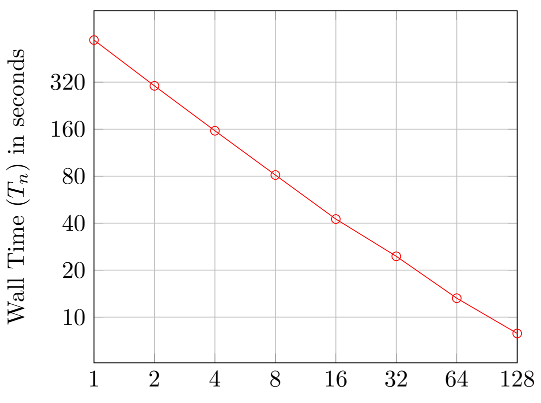


(a) Wall time

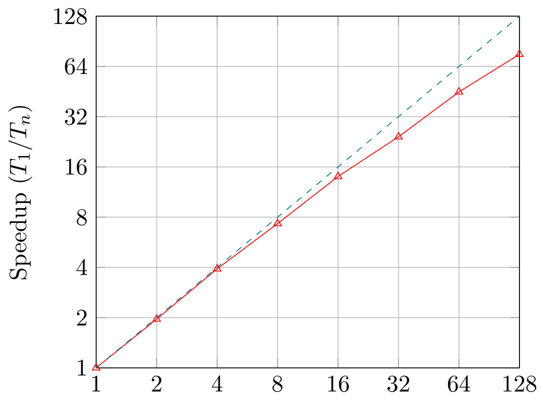


(b) Speedup

Figure D.2.: Strong scaling wall time and speedup of shared memory parallelization on HSUper with two-body interactions. The domain size was (90, 100, 50) and the number of molecules was (80, 90, 47).

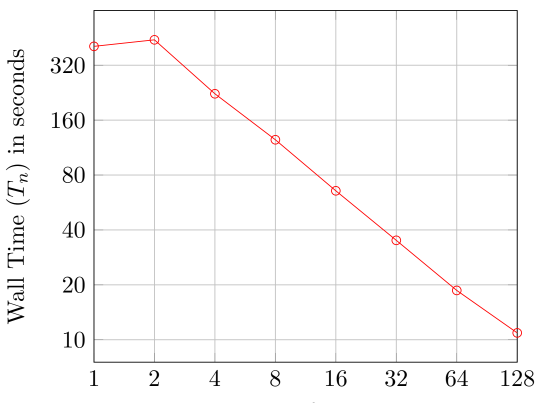


(a) Wall time

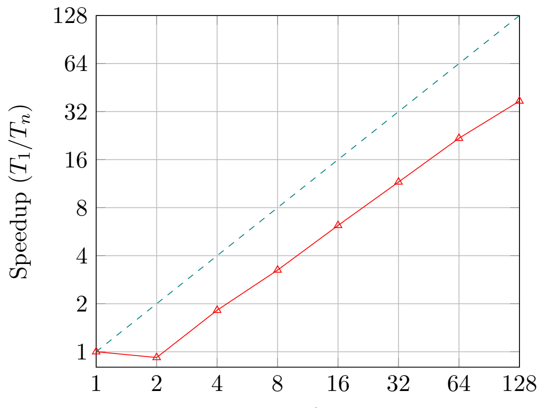


(b) Speedup

Figure D.3.: Strong scaling wall time and speedup of distributed memory parallelization on CoolMUC-2 with two-body interactions. The domain size was (200, 200, 50) and the number of molecules was (188, 188, 47).



(a) Wall time



(b) Speedup

Figure D.4.: Strong scaling wall time and speedup of distributed memory parallelization on HSUper with two-body interactions. The domain size was (200, 200, 50) and the number of molecules was (188, 188, 47).

List of Figures

List of Tables

Bibliography

[AL22] Lyuba Alboul and Sergey V. Lishchuk. Bulk viscosity of gaseous argon from molecular dynamics simulations. *Phys. Rev. E*, 105:054135, May 2022.

[Amd67] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), page 483–485, New York, NY, USA, 1967. Association for Computing Machinery.

[AS22] B.P. Akhouri and J.R. Solana. Thermodynamic properties of ar, kr and xe from a monte carlo-based perturbation theory with an effective two-body lennard-jones potential. *Physica A: Statistical Mechanics and its Applications*, 608:128280, 2022.

[AT04] B. M. Axilrod and E. Teller. Interaction of the van der Waals Type Between Three Atoms. *The Journal of Chemical Physics*, 11(6):299–300, 12 2004.

[Boa15] OpenMP Architecture Review Board. OpenMP Application Programming Interface. `https://www.openmp.org/wp-content/uploads/openmp-4.5.pdf`, November 2015. Accessed 30.07.2023.

[CW00] C.F. Cornwell and L.T. Wille. Parallel molecular dynamics simulations for short-ranged many-body potentials. *Computer Physics Communications*, 128(1):477–491, 2000.

[GSBN21] 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, 2021.

[Gus88] John L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31(5):532–533, may 1988.

[GZK07] Michael Griebel, Gerhard Zumbusch, and Stephan Knapek. *Numerical Simulation in Molecular Dynamics: Numerics, Algorithms, Parallelization, Applications.* Springer Berlin Heidelberg, 2007.

[JN19] Piet Jarmatz and Philipp Neumann. Mamico: Parallel noise reduction for multi-instance molecular-continuum flow simulation. In *Computational Science – ICCS 2019*, pages 451–464, Cham, 2019. Springer International Publishing.

[KKN$^+$13] Manaschai Kunaseth, Rajiv K. Kalia, Aiichiro Nakano, Ken-ichi Nomura, and Priya Vashishta. A scalable parallel algorithm for dynamic range-limited n-tuple computation in many-body molecular dynamics simulation. In *SC '13:*

Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pages 1–12, 2013.

[KY14] Penporn Koanantakool and Katherine Yelick. A computation- and communication-optimal parallel direct 3-body algorithm. In *SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 363–374, 2014.

[Li18] Xin Li. Scalability: strong and weak scaling. `https://www.kth.se/blogs/pdc/2018/11/scalability-strong-and-weak-scaling`, November 2018. Accessed 02.08.2023.

[LMV16] Y. S. Lavrinenko, I.V. Morozov, and I. A. Valuev. Reflecting boundary conditions for classical and quantum molecular dynamics simulations of nonideal plasmas. *Contributions to Plasma Physics*, 56(5):448–458, 2016.

[Mar01] Gianluca Marcelli. The role of three-body interactions on the equilibrium and non-equilibrium properties of fluids from molecular simulation. 2001.

[MS99] Gianluca Marcelli and Richard J. Sadus. Molecular simulation of the phase behavior of noble gases using accurate two-body and three-body intermolecular potentials. *The Journal of Chemical Physics*, 111(4):1533–1540, 07 1999.

[MTS01] Gianluca Marcelli, B. D. Todd, and Richard J. Sadus. On the relationship between two-body and three-body interactions from nonequilibrium molecular dynamics simulation. *The Journal of Chemical Physics*, 115(20):9410–9413, 11 2001.

[New50] Sir Isaac Newton. *Newton's Principia: The Mathematical Principles of Natural Philosophy.* 1850. Translated by Andrew Motte.

[NFA$^+$16] Philipp Neumann, Hanno Flohr, Rahul Arora, Piet Jarmatz, Nikola Tchipev, and Hans-Joachim Bungartz. Mamico: Software design for parallel molecular-continuum flow simulations. *Computer Physics Communications*, 200:324–335, 2016.

[NW20] Philipp Neumann and Niklas Wittmer. Open Boundary Modeling in Molecular Dynamics with Machine Learning. In *Computational Science – ICCS 2020*, pages 334–347, Cham, 2020. Springer International Publishing.

[O'S13] L. D. O'Suilleabhain. Three Body Approximation to the Condensed Phase of Water. 2013.

[PDSK05] Matej Praprotnik, Luigi Delle Site, and Kurt Kremer. Adaptive resolution molecular-dynamics simulation: Changing the degrees of freedom on the fly. *The Journal of Chemical Physics*, 123(22):224106, 12 2005.

[POM$^+$23] Sergey Pozdnyakov, Artem R. Oganov, Efim Mazhnik, Arslan Mazitov, and Ivan Kruglov. Fast general two- and three-body interatomic potential. *Phys. Rev. B*, 107:125160, Mar 2023.

[SABW82] William C. Swope, Hans C. Andersen, Peter H. Berens, and Kent R. Wilson. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics*, 76(1):637–649, 01 1982.

[Ver67] Loup Verlet. Computer ”experiments” on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159:98–103, Jul 1967.