Fachhochschule Aachen

Campus Jülich

Faculty: Medical Engineering and Technomathematics
Course of study: Applied Mathematics and Informatics

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Acceleration of MDMC simulations by introducing a multipole coefficient based energy prediction algorithm

Masterthesis by Jonas Kroschewski
Matrikelnummer: 3273652

Jülich, August 22, 2025

Jülich Supercomputing Centre,
Forschungszentrum Jülich GmbH

This work was supported by:

Prof. Dr. Andreas Kleefeld
Prof. Dr. Godehard Sutmann

Declaration of Authorship

Ich, Jonas Kroschewski, versichere hiermit an Eides Statt, dass ich die vorliegende Masterarbeit mit dem Titel 'Acceleration of MDMC simulations by introducing a multipole coefficient based energy prediction algorithm' selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Jülich, den 22. August 2025

Name des Studenten: Jonas Kroschewski

Unterschrift des Studenten:

Belehrung

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Jülich, den 22. August 2025

Name des Studenten: Jonas Kroschewski

Unterschrift des Studenten:

Abstract

This work focuses on improving the coupled MD-MC simulation framework by enhancing the MC++ code by H. Ganesan that is used to model strain aging processes in e.g. steel. Key improvements include the addition of a LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) interface to replace the discontinued IMD (ITAP Molecular Dynamics) library, the conversion of the EAM-potential (Embedded Atom Method), and the translation of the concept of 'virtual atoms' to LAMMPS which represent dislocation sites in body-centered cubic (bcc) ferrite. To further accelerate simulations, a novel multi-pole coefficient based algorithm for predicting the potential energy of atom configurations is proposed, based on database searches for similar configurations. This approximation aims to reduce the computational cost of energy calculations, although its accuracy and reliability are critically evaluated. Additionally, to address the assumed increased data throughput required by the new algorithm, a modification of a parallel replica method is introduced to the MC++ code, aimed at reducing communication overhead while maintaining the accuracy of simulation results. These advancements are expected to significantly enhance the efficiency and scalability of the MDMC simulation framework for modeling strain aging processes in steel. Steel has long been a vital material in engineering, with recent advancements in high-strength steels addressing the automotive industry's need for enhanced fuel efficiency and reduced environmental impact. One key strengthening mechanism in steel, particularly low-carbon steel, is strain aging, where carbon atoms segregate at dislocation sites, impeding dislocation motion and enhancing yield strength. However, simulating strain aging processes—such as carbon diffusion and dislocation evolution—presents significant challenges due to the disparity in timescales involved. Molecular Dynamics (MD) and Monte Carlo (MC) simulations offer complementary approaches to model these processes, but limitations in computational efficiency persist. Thus the coupled MDMC framework was developed which is further improved and accelerated in the course of this work.

Contents

1 Introduction

Since the Industrial Revolution, steel has become a fundamental engineering material with widespread applications in infrastructure, machinery and consumer goods. Recent decades have seen significant advancements in high strength steels, particularly to address the demands of the automotive industry for improved fuel efficiency in response to increasing environmental concerns.
One critical strengthening mechanism in steels is strain aging, theoretically described by Cottrell and Bilby in the year 1949 [2],[1]. They proposed that in low carbon steels, carbon atoms in solid solution migrate to dislocation sites, forming 'Cottrell atmospheres', which impede dislocation motion. In low carbon steels ($\approx 0.002\%$ C), strain aging is exploited through a process known as bake hardening. Automotive components formed from thin steel sheets undergo plastic deformation during press forming, followed by thermal treatment during the paint-baking cycle ($\approx 170°C$ for 20 min), which results in an increase in yield strength due to the immobilization of dislocations by segregated carbon atoms.
The mechanical properties of alloys such as steel are often studied through atomistic simulations, with classical Molecular Dynamics (MD) and Monte Carlo (MC) methods being widely employed for this purpose [5]. Molecular Dynamics simulations calculate atomic trajectories by solving Newton's equations of motion and utilizing empirical force fields like the Embedded Atom Method (EAM). While MD offers accurate modeling of defects and their evolution in crystalline materials, its time step is typically on the order of femtoseconds, limiting the simulation duration to nanoseconds. This makes simulating rare events, such as carbon diffusion, particularly challenging, as these processes occur over much longer timescales (seconds to hours).
In contrast, Monte Carlo (MC) methods adopt a stochastic approach, exploring system configurations via random perturbations. The Metropolis algorithm [20] is commonly used in Monte Carlo applications, which is not constrained by physical time, making it ideal for simulating diffusion processes like carbon migration. By coupling MD and MC, researchers can model both dislocation evolution and carbon segregation in a unified framework. This coupling is achieved using 'virtual atoms' [10, p.27f], which serve as placeholders for all octahedral sites in body-centered cubic (bcc) iron. However, the efficiency of this coupling requires parallelization, as simulating large volumes of material and their deformation or segregation mechanisms would be computationally prohibitive without it.
This work is mainly dedicated to the improvement of the MC++ code by H.Ganesan, which implements such an MDMC framework [10],[11],[12] for the simulation of strain aging processes. These improvements include not only technical aspects such as the implementation of a LAMMPS [16] interface in addition to the existing IMD [14] interface for MD simulations, but also methodological additions to accelerate the execution of MDMC simulations. The need to add a LAMMPS interface is due to the fact that active support for IMD has been discontinued and therefore an alternative is needed in the long term. In addition to adaptations in the code, this changeover also requires the

conversion of the EAM potential and the implementation of the concept of virtual atoms in LAMMPS.
To accelerate the MDMC simulations, an algorithm will be introduced that can approximately predict the potential energy of local atom configurations without the need for a full computation of the atom interactions. This will be realized by a similarity search in a database [3] in which all local configurations that have occurred in the course of a Monte Carlo atom exchange are stored. The approximation of the potential energy of a new configuration is done by searching for a similar configuration in this database. If such a configuration can be identified, its energy is used as a prediction for that of the current configuration. In order to significantly reduce the amount of data stored in the database and to simplify the determination of similarity, multipole coefficients [6] of several orders and moments are used as descriptors for these configurations. This also introduces a certain degree of uncertainty when comparing two configurations, as the information content is also drastically reduced by the alternative representation. It is therefore also necessary to investigate the reliability and accuracy with which the potential energy can be predicted. If this can be guaranteed, the introduction of the energy prediction can be expected to significantly speed up the MDMC simulation, as this can also eliminate a large proportion of the time-consuming MD simulations.
Since the search in the database is an important part of the prediction algorithm, two other models are considered in this thesis in addition to the simplest model and their search performance is compared.
As already mentioned, parallelization is an essential component for increasing the performance of MDMC simulation. Since the introduction of the energy prediction algorithm is expected to require a significantly increased throughput of atom data, it is assumed that the current master-worker approach may reach its limits for many processors, or at least that the communication overhead will increase significantly. For this reason, a modification of the parallel replica method [7] will be implemented in the course of this work. This should reduce the necessary communication, especially for long simulation runtimes, while achieving comparable results in terms of system energy as with the master-worker scheme. It is to be determined if this method represents a valid alternative, or at least addition, for the master-worker approach.

2 Theoretical Preliminaries

2.1 Particle Simulations

Particle simulations are applied to explore the properties of varied materials like mechanical, thermal, magnetic, optical and electric behavior. They are most commonly found in the field of solid state physics, theoretical chemistry and material science. While methods like e.g. Density Functional Theory (DFT) are accurate, they are computationally expensive [21]. Therefore other methods like Molecular Mechanics and Monte Carlo are often preferred as they can handle systems of bigger size.

2.1.1 Molecular Mechanics

Molecular Mechanics focus on the classical laws of mechanics. It considers the interaction of the many-body system as well as different external and environmental conditions like added temperature or pressure. To perform a simulation with Molecular Mechanics there are specific requirements that can be divided into three categories:

1. **Initial Configuration:**
   An initial system must be given which is in a physically correct state in order to be able to carry out the simulation from there. In this work, iron-carbon systems that form a homogeneous lattice structure are considered in particular. The correctness of material parameters such as the crystal structure, lattice constant and potential energy of the atoms must therefore match as well as the elementary properties of the individual atoms such as mass.

2. **Potential:**
   If there is an initial system for the material under investigation, atomic movements and thus structural changes are caused by an imbalance of forces between individual particles. To calculate these forces, a potential function is used which, depending on e.g. distance, density and atomic species, provides the resulting force for each atom and thus not only specifies a direction of movement, but can also provide a total energy of the system.
   One of the simplest potentials is the pair potential. Here, the force $F_{i,j}$ between two atoms i and j depends solely on the distance $r_{i,j}$ between those two. Another parameter varies the distance at which the atoms exert neutral force on each other. If two particles are closer to each other, they are repelled, otherwise attracted. In order to calculate the resulting force for a particle, the sum of the individual forces to all particles must be calculated. Since this step would have a complexity of $O(\frac{n\cdot(n-1)}{2})$ under the assumption of symmetrical forces, only all particles within a

cutoff radius r_c are usually considered, since the forces are very small for larger distances.
As such a pair potential is not sufficient to mimic the properties of a metal lattice, more complex potentials are necessary. One such potential is the Embedded Atom Model (EAM) of Daw and Baskes, which is based on DFT [23]. The total energy of a system can be calculated as follows:

$$E_{tot} = \sum_{i}^{n} F_i(\rho_{h,i}) + \frac{1}{2} \sum_{i,j;i\neq j}^{n} \Phi_{ij}(R_{ij}) \tag{1}$$

$$\rho_{h,i} = \sum_{j;j\neq i}^{n} \rho_j^a(R_{ij}) \tag{2}$$

Thereby Φ_{ij} is the pair potential between atoms i and j with euclidean distance R_{ij}. The local electron density $\rho_{h,i}$ is the combined sum of densities of the neighboring atoms ρ_j^a with distances R_{ij} which is then evaluated by the cohesive function F_i. In practice this function is determined empirically based on the type of material used and its properties (e.g. lattice-type, lattice constant,...). By adding the electron density this many-body potential is able to handle crystal structures even if they have cracks or impurities [9, p.14f, p.30f]. Therefore an EAM potential for FeC interactions was chosen to compute the necessary energies in this work.

3. **Evolution algorithm:**
The evolution of atom coordinates in a Molecular Mechanics simulation depends on the choice of the potential but also on the applied evolution algorithm. Those can be classified in two categories: Molecular Statics (MS) and Molecular Dynamics (MD).
Molecular Statics focuses on finding the minimal energy state of an initial configuration. The system is therefore observed at a temperature of $T = 0K$ eliminating particle velocities and thus momentum. The minimization consists solely of structural changes of the configuration. Applying the potential-function yields a force along the potential gradient for each particle. This can be used for common minimization algorithms like Steepest Descent, Conjugate Gradient or FIRE [13]. Molecular Statics algorithms are often applied to generate initial configurations for Molecular Dynamics [9, p.16].
The main difference of Molecular Dynamics to Statics is that each particle may have a non zero velocity in addition to its position. For each time step δt the forces on all atoms are calculated. Applying Newtons equation of motion $F = m \cdot a$ yields the acceleration on each atom and thus the change in velocity that is to be applied. Multiple integration schemes exist to achieve the position update. Those provide different benefits like accuracy, reversibility, stability, speed or conservation of energy. Usually the time step has to be chosen quite small ($1fs$) for the Molecular Dynamics algorithm to work. This is also a huge limiting factor as it is not feasible to run Molecular Dynamics on greater timescales [9, p.18].

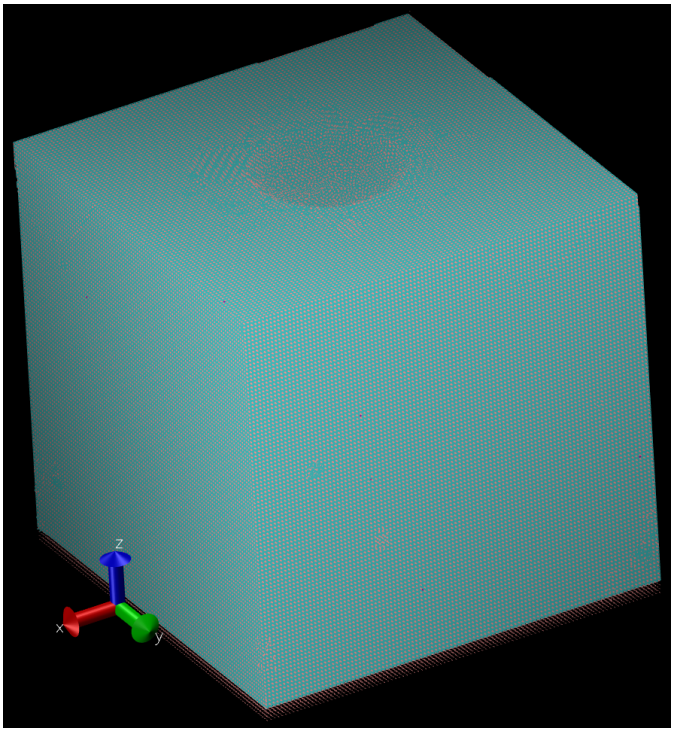

Figure 1: Example for an initial configuration for an iron-carbon system that the MDMC simulation was exerted on. The system has a nano-indentation on the top which should provide enough lattice distortions so that Cottrell atmospheres can form.

2.1.2 Monte Carlo

Monte Carlo (MC) methods represent a stochastic approach to exploring the phase space of a system. In this work the Metropolis algorithm [20] was employed. Instead of evolving the system continuously like MD, the method undergoes a sequence of trial moves where particles are randomly displaced. For each such move the energy change ΔE of the system is observed which leads to acceptance or rejection of it. One possible probabilistic criterion for acceptance is given by the following equation where κ_B is the Boltzmann constant and T the temperature.

$$P_{accept} = \min\left(1, e^{-\frac{\Delta E}{\kappa_B T}}\right) \tag{3}$$

If the systems temperature is $T = 0K$ the configuration with the lower energy is always accepted. This mimics the Molecular Statics behavior. While Monte Carlo offers the advantage of potentially faster and less computational expensive convergence for certain properties, its brute-force nature may lead to high rejection rates in the long run. To

address this, biased Monte Carlo methods can be applied which weight the sampling towards the regions that where more rewarding in the past.

2.2 Introduction to MDMC-Simulations

One possible application of MDMC Simulations is to model the formation of Cottrell atmospheres [2], which result from the segregation of interstitial solutes like carbon in metallic alloys (e.g. iron-grid). In this context, segregation refers to the inhomogeneous distribution of solute atoms in a crystalline solid, often driven by the presence of lattice defects that modify local energies and promote solute accumulation in specific regions.
As the name implies MDMC Simulations combine the benefits of the Monte Carlo Method with the accuracy of Molecular Dynamics/Statics to accomplish this task. The Cottrell atmospheres form over a long time period from several minutes up to multiple hours which makes a Molecular Dynamics based approach not feasible concerning the computational cost. As the diffusion process is thermally activated a Molecular Statics simulation would usually not work.
These problems can be overcome by the Monte Carlo method, since it is not dependent on simulating the whole diffusion process for each carbon atom. In the course of an MC trial move a random carbon atom is determined whose position is changed locally or globally. If the repositioned carbon atom is located close to a lattice defect or a forming Cottrell atmosphere, there is a higher probability that the configuration resulting from the swap will lead to a lower system energy. To increase the accuracy of this process, a local relaxation is performed before the system energy is evaluated so that the carbon atom is in the most favorable position and the iron grid is aligned. In combination with the Monte Carlo method a relaxation with Molecular Statics becomes feasible again as the atoms are able to overcome the thermal barrier as they simply change positions.
Such a local relaxation in comparison to simulating the whole configuration has several advantages. Firstly, it greatly reduces the number of atoms to be relaxed, which significantly shortens the determination of the position and energy changes of the atoms. Meanwhile, the final result is not very different from that of a global relaxation, since the pair interaction forces are subject to a cutoff distance that sets all forces on particles beyond this barrier to zero and the distortion of the lattice is strongly damped due to the strong iron-iron interaction. On the other hand, this allows parallel processing of several trial moves, provided that their relaxation regions do not overlap.

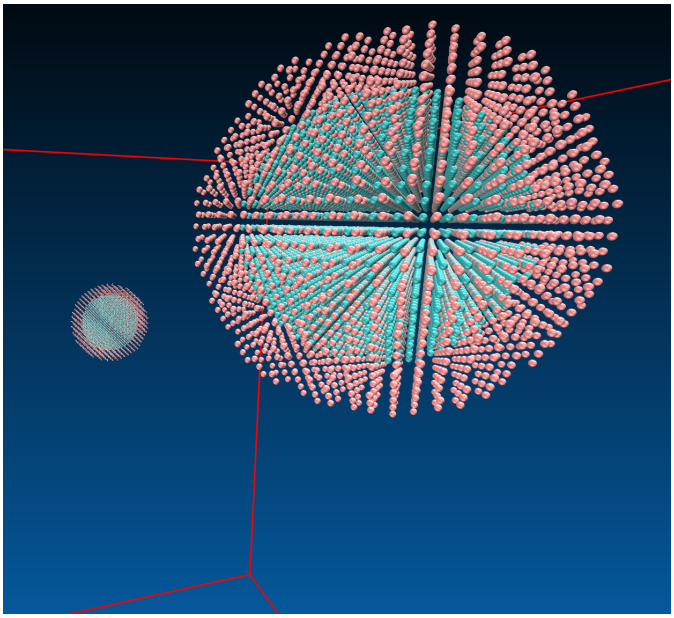

Figure 2: The figure shows the local spheres of particles around the current and future positions of a carbon atom. The future position may be anywhere in the system and so the local relaxation areas are usually independent of each other, as shown in the picture. These local spherical configurations are extracted from the whole particle configuration. The inner zone contains the virtual atoms (blue) in addition to the iron (red) and carbon (purple) atoms. As the outer zone only serves as stable mantle around the inner one that is relaxed, it does not need to contain the virtual atoms.

To allow a seamless integration of a locally relaxed configuration back in the system, the relaxation space is made up of three zones. In the core zone, which makes up the largest part, all particles are relaxed and the potential energies are calculated. The particles can move freely according to the EAM potential. In the surrounding transition zone, the potential energies are also determined, but all particles are held in place so that this zone limits the internal particles in their expansion. In the outermost zone, the mantle zone, no energy determination takes place and all particles are also fixed. The sole purpose of this zone is to ensure that the energies of the particles in the transition zone are calculated correctly and that these particles experience both pressure from the inside and a counter-pressure from the outer particles.
In bcc iron lattices, the carbon atoms preferentially occupy octahedral interstitial sites due to their small size compared to the host iron atoms. One problem that arises from this for the Monte Carlo method is the effective localization of these interstitial sites. Especially if the lattice gets distorted during the simulation, the interstitial sites also move which makes finding valid sampling sites more complex.

2.2.1 Virtual Atoms

To overcome these limitations, the concept of virtual atoms was introduced. These pseudo-atoms are positioned at all octahedral sites in a defect-free bcc iron lattice. The role of virtual atoms is to represent potential sampling sites for the carbon atoms without introducing physical interference with the simulated material. This is accomplished by restricting the interacting forces between virtual and non-virtual atoms to be asymmetrical so that the virtual atoms exert no forces on non-virtual ones. Thereby the virtual atoms behave like carbon atoms as they share the same mass and potential function. After minimizing the system energy, the virtual atoms are present in their local minimum without affecting the system itself. If the position of a carbon atom is to be varied in the course of a Monte Carlo trial move, the position of a random virtual atom is determined instead of a random position. If the carbon atom is then swapped with this virtual atom, it is already in an almost optimal position, so that the effort for the subsequent relaxation of the MDMC simulation is also reduced. Another benefit from using virtual atoms as sampling sites is that under normal circumstances sampling sites never lie within other existing atoms which would make them invalid.

2.2.2 Improved Sampling Algorithms

The sampling of interstitial sites can be further improved than just with virtual atoms. The simulated systems is therefore divided into multiple cells. For each of those the rate of accepted and rejected trial moves is tracked. Based on this data a most promising cell is determined, where the next virtual atom for the trial move is to be picked from. This procedure is especially valuable to simulate the formation of Cottrell atmospheres as this sampling algorithm is able to actively react to changes in the system. If for example such an Atmosphere gets saturated, the algorithm is able to choose another, perhaps formerly less favorable, location to form a second Cottrell atmosphere in. A more detailed description of this biased sampling algorithm and its various parameters can be found in the work of M. Longsworth [18].

2.3 Multipole Coefficients

2.3.1 Legendre Polynomials

Legendre polynomials are a special kind of function that have the property to be orthogonal to each other. Furthermore integrating the polynomial in the interval $[-1, 1]$

should yield 1 as solution. These restrictions lead to the following definition:

$$\int_{-1}^{1} P_m(x)\, P_n(x)\, \mathrm{d}x = 0 \quad \forall n, m \in \mathbf{N} \quad n \neq m \tag{4}$$

Together with the additional condition $P_n(1) = 1$ all Legendre polynomials of the first kind can be determined. An alternative definition is given by the solutions to the Legendre differential equation [24, 14.2(i)].

$$\left(1 - x^2\right) \frac{\mathrm{d}^2}{\mathrm{d}x^2} P_n(x) - 2x \frac{\mathrm{d}}{\mathrm{d}x} P_n(x) + n(n+1) P_n(x) = 0 \tag{5}$$

This equation arises naturally in a physical context, if the Laplace equation is solved in spherical coordinates. The polynomial solution is not the only solution to this second-order differential equation as there is also a non-polynomial one called Legendre functions of second kind.
As the given approaches to retrieve the polynomials can be tedious for higher orders of n an explicit way to get those is given by Rodrigues formula [17, p.334].

$$P_n(x) = \frac{1}{2^n n!} \frac{\mathrm{d}^n}{\mathrm{d}x^n} \left(x^2 - 1\right)^n \tag{6}$$

2.3.2 Associated Legendre Polynomials

The associated Legendre polynomials are the solution to the general Legendre equation which is related to the Legendre differential equation in Eq. (5). The general Legendre equation is a two parameter generalization of this formula which is given by [24, 14.2(ii)]:

$$\left(1 - x^2\right) \frac{\mathrm{d}^2}{\mathrm{d}x^2} P_\ell^m(x) - 2x \frac{\mathrm{d}}{\mathrm{d}x} P_\ell^m(x) + \left(\ell(\ell+1) - \frac{m^2}{1-x^2}\right) P_\ell^m(x) = 0 \tag{7}$$

The two indices ℓ and m stand for the degree and order of the polynomial, respectively. This equation only has solutions on $[-1, 1]$ if ℓ and m are integers and $0 \geq m \geq \ell$ or $-\ell \geq m \geq 0$. For $m = 0$ this equation describes the ordinary Legendre polynomials. For $m \geq 0$ the associated Legendre polynomials can also be written as derivatives of ordinary Legendre polynomials $P_\ell(x)$ [24, 14.6(i)].

$$P_\ell^m(x) = (-1)^m \left(1 - x^2\right)^{\frac{m}{2}} \frac{\mathrm{d}^m}{\mathrm{d}x^m} P_\ell(x) \tag{8}$$

Using Rodrigues formula from Eq. (6) $P_\ell^m(x)$ can also be expressed in the form [24, p. 14.7.10]:

$$P_\ell^m(x) = \frac{(-1)^m}{2^\ell \ell!} \left(1 - x^2\right)^{\frac{m}{2}} \frac{\mathrm{d}^{\ell+m}}{\mathrm{d}x^{\ell+m}} \left(1 - x^2\right)^\ell \tag{9}$$

This way of expressing the formula allows m to be $-\ell \geq m \geq \ell$. It can be shown that P_ℓ^m and P_ℓ^{-m} are proportional to each other with proportionality factor c_{lm} which results in the following representation.

$$P_\ell^{-m}(x) = (-1)^m \frac{(\ell - m)!}{(\ell + m)!} P_\ell^m(x) \tag{10}$$

The associated Legendre polynomials can be calculated recursively which is advantageous if they are used in context of computational science. The most helpful recurrence formulas are given by:

$$P_{\ell+1}^{\ell+1}(x) = -(2\ell + 1)\sqrt{1 - x^2} P_\ell^\ell(x) \tag{11}$$

$$P_\ell^\ell(x) = (-1)^\ell (2\ell - 1)!! \left(1 - x^2\right)^{\frac{\ell}{2}} \tag{12}$$

$$P_{\ell+1}^\ell(x) = x(2\ell + 1) P_\ell^\ell(x) \tag{13}$$

Thereby the double factorial $n!!$ is not equal to $(n!)!$ but:

$$n!! = \prod_{k=0}^{\lceil \frac{n}{2} \rceil - 1} (n - 2k) \tag{14}$$

2.3.3 Reparameterization

The associated Legendre polynomials can be reparameterized with $x = \cos(\theta)$ resulting in the following representation of the formula:

$$P_\ell^m(\cos(\theta)) = (-1)^m (\sin(\theta))^m \frac{\mathrm{d}^m}{\mathrm{d}(\cos(\theta))^m} P_\ell(\cos(\theta)) \tag{15}$$

Given this alternative representation the definitions of orthogonality also have to be adjusted. For fixed m two polynomials are orthogonal if they satisfy [24, p. 14.17]:

$$\int_0^\pi P_k^m(\cos(\theta)) P_\ell^m(\cos(\theta)) \sin(\theta) \, \mathrm{d}\theta = \frac{2(\ell + m)!}{(2\ell + 1)(\ell - m)!} \delta_{k,\ell} \tag{16}$$

where $\delta_{k,\ell}$ is the Kronecker delta function. For fixed ℓ two polynomials have to meet the following equations to be orthogonal.

$$\int_0^\pi P_\ell^m(\cos(\theta)) P_\ell^n(\cos(\theta)) \csc(\theta) \mathrm{d}\theta = \begin{cases} 0 & m \neq n \\ \frac{(\ell+m)!}{m(\ell-m)!} & m = n \neq 0 \\ \infty & m = n = 0 \end{cases} \tag{17}$$

The underlying differential equation that is solved by this reparameterized function can be written as:

$$\frac{\mathrm{d}^2 y}{\mathrm{d}\theta^2} + \cot(\theta) \frac{\mathrm{d}y}{\mathrm{d}\theta} + \left(\ell(\ell + 1) \frac{m^2}{\sin(\theta)^2} \right) y = 0 \tag{18}$$

2.3.4 Spherical Harmonics

Spherical harmonics play an important role in many theoretical and practical applications. One of those is the spherical multipole expansion where the basic Laplace spherical harmonics $Y_\ell^m(\theta, \phi)$ are used. Those functions are closely connected to the associated Legendre polynomials as they also form an orthogonal system. When the Laplace equation is solved in spherical domains, the obtained solution is called a spherical harmonic. Given θ as colatitude and ϕ as longitude the Laplacian in spherical coordinates is:

$$\nabla^2 f = \frac{\mathrm{d}^2 f}{\mathrm{d}\theta^2} + \cot\theta \frac{\mathrm{d}f}{\mathrm{d}\theta} + \csc\theta^2 \frac{\mathrm{d}^2 f}{\mathrm{d}\phi^2} \tag{19}$$

Solving the partial differential equation $\nabla^2 f + \lambda f = 0$ yields a ϕ-dependent part $\sin(m\phi)$ or $\cos(m\phi)$ for $m \geq 0$ and a θ-dependent part:

$$\frac{\mathrm{d}^2 y}{\mathrm{d}\theta^2} + \cot\theta \frac{\mathrm{d}y}{\mathrm{d}\theta} + \left(\lambda - \frac{m^2}{\sin(\theta)^2}\right) y = 0 \tag{20}$$

The solution for this equation is already known from Eq. (18) and is given by the associated Legendre polynomial $P_\ell^m(\cos(\theta))$ for $\ell \geq m$ and $\lambda = \ell(\ell+1)$. It can be shown that the equation $\nabla^2 f + \lambda f = 0$ only has solutions if $\lambda = \ell(\ell+1)$ and those solutions are proportional to the following two functions:

$$\begin{aligned} P_\ell^m(\cos(\theta))\cos(m \cdot \phi) \quad & 0 \leq m \leq \ell \\ P_\ell^m(\cos(\theta))\sin(m \cdot \phi) \quad & 0 < m \leq \ell \end{aligned}$$

Therefore, the spherical harmonics $Y_\ell^m(\theta, \phi)$ are often expressed as complex exponentials.

$$Y_\ell^m(\theta, \phi) = N \cdot e^{im\phi} P_\ell^m(\cos(\theta)) \quad -\ell \leq m \leq \ell \tag{21}$$

Thereby N is a normalization factor which is chosen based on the application of the spherical harmonics. In the field of quantum mechanics for example this is often chosen as

$$N = (-1)^m \sqrt{\frac{(2\ell+1)(\ell-m)!}{4\pi(\ell+m)!}} \tag{22}$$

2.3.5 Multipole Expansion

Multipole Expansions are often used to describe potential fields in three dimensional space. This is done by constructing a mathematical series based on the charges q_i and positions of the atoms. In this work those positions are denoted in spherical coordinates instead of carthesian to simplify the expansion for higher orders. This leads to an atom

been described by the angles θ for colatitude and ϕ for longitude and r for the distance to the origin $\overrightarrow{p}$ of the evolution. The expansion series Φ_s can be written as sum of spherical harmonics O_ℓ^m [8].

$$\Phi_s\left(\overrightarrow{p}\right) = \sum_{\ell=0}^{s} \sum_{m=-\ell}^{\ell} \omega_\ell^m\left(\overrightarrow{p}\right) \tag{23}$$

$$\omega_\ell^m\left(\overrightarrow{p}\right) = \sum_{i=1}^{N} q_i O_\ell^m\left(r_i\right) \quad r_i = |\overrightarrow{p_i} - \overrightarrow{p}|_2 \tag{24}$$

Thereby s is the maximum order of the expansion. For each order ℓ and moment m the spherical harmonic O_ℓ^m is evaluated and multiplied by the charge of the atom at the position p_i. The single results are then summed to the series Φ_s which is getting more detailed the higher s is chosen.

$$O_\ell^m\left(r_i\right) = \frac{r_i^\ell}{(\ell + |m|)!} P_\ell^m\left(\cos\left(\theta_{r_i}\right)\right) e^{-im\phi_{r_i}} \tag{25}$$

The spherical harmonic has the same base form as in Eq. (21). The prefactor N is chosen so that multipole moments of higher order have less impact on the final value of the series. Further the atoms in greater distance to the origin p of the expansion have higher impact. The associated Legendre polynomial P_ℓ^m is once recursively constructed for the current m and ℓ and then evaluated for all the θ_{r_i} of the atoms.
In this work the multipole expansion is used to get a less complex descriptor for a configuration of atoms. Usually for each atom three values have to be stored to describe its position which gets computationally expensive if two groups of atoms have to be compared in terms of position differences. As this is the main idea behind the performance increase the multipole expansion is used to lower the number of computations while still describing the atom distribution in acceptable detail. This is done by calculating the different multipole moments apart from each other for all atoms to get e.g. 15 values for $s = 5$ which are usually summed but instead can be compared to the mutipole coefficients of other atom groups.
In this context one change was done to the formula for the spherical harmonics in Eq. (25). As a change in atom position close to the origin of the expansion should be of more importance than one that lies further out the parameter r_i^ℓ in the prefactor N was swapped from the nominator to the denominator. Further the exponent ℓ vanishes to assure that the low order multipole coefficients give the same weight to r_i as the high order ones. This prevents the coefficients to change non-linearly for rising orders ℓ for the same change Δr in distance to the origin of expansion.
These changes lead to the following version of the formula to compute the multipole coefficients.

$$O_\ell^m\left(r_i\right) = \frac{1}{r_i \cdot (\ell + |m|)!} P_\ell^m\left(\cos\left(\theta_{r_i}\right)\right) e^{-im\phi_{r_i}} \tag{26}$$

2.4 Parallel Replica Method

The Parallel Replica Method is a technique that can be applied to speed up finding rare events, energy minimization or Molecular Dynamics simulations of systems that very rarely change their state [7], [22], [25]. It significantly improves the performance of those systems by parallelizing the wait time for new state transitions. The discussed variation of the Method can be divided into three main steps that are continuously repeated [7].

- Decorrelation:
  A single process runs a regular MD simulation starting from the current system state until the correlation time s_c is reached. Afterwards a copy of the current configuration is broadcast to all other processes.

- Dephasing:
  Each receiving processor also executes an MD simulation of the received configuration. Thereby the velocities are randomized to ensure that the replicas are statistically independent. If there is no state transfer happening over a time span on s_c the replica is considered 'dephased' else the MD simulation is repeated until this condition is met.

- Parallel search:
  If all replicas are 'dephased' they initiate yet another MD run. The first replica running into a state transfer during this simulation stops all other replicas. The current configuration is then accepted as new state and the total simulation time is advances by the time the replica took. Afterwards the cycle is repeated starting from the updated state.

There are various ways to determine the correlation time s_c. One of the simpler methods monitors statistics like residence time and transition rates. If the results are inconsistent, because the state transitions happen in quick succession before the configuration has settled, s_c is too short. On the other hand if very few transitions occur or the dephasing step has to be repeated multiple times the correlation time might be to long.
In this work the idea of the Parallel Replica Method is applied in a slightly deviated form to measure its performance gain compared to the former master-worker approach. With increasing number of Monte Carlo sampling steps the probability to accept a random trial move decreases as the diffusion sites get more saturated. This leads to fewer state changes and longer wait times in between accepted states. The idea of introducing the Parallel Replica Method is that instead of the master generating and distributing the trial moves to the workers each worker has an exact replica of the whole configuration and performs trial moves by itself. After one worker has accepted N valid states, it informs all other processes that are then synchronizing their systems to the one with the lowest total energy. Afterwards the workers continue performing trial moves on the updated configuration.

3 Implementation

3.1 MC++ Lammps interface

In the course of this work, the existing MC++ code was extended by a LAMMPS interface. Since the currently used program for the execution of MD simulations (IMD) is no longer maintained, a replacement had to be found so that the MC++ code remains usable in the future. The choice fell on LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) because it is not only an alternative to IMD, but also offers many additional functions and an active community. In addition, it is possible to integrate LAMMPS as a library, which opens up many more possibilities. The following section explains some of the obstacles encountered during implementation and gives a brief overview of the functionality of the implemented LAMMPS interface.

3.1.1 Mimic Virtual Atoms

The use of virtual atoms in combination with the Monte Carlo method offers an enormous advantage when finding sampling sites. Since the virtual atoms are included in the relaxation process just like all others, the MD program must support asymmetric forces. This modification was subsequently added to IMD. In LAMMPS however, the basic assumption is that all forces between particles act symmetrically. The loops for e.g. determining the neighbor lists or the pair interactions are adapted accordingly in order to increase the performance. An integration of asymmetric particle interactions in LAMMPS would therefore require a complete restructuring of the code. Since this would not only have meant an immense amount of work, but would also have required subsequent maintenance of the custom LAMMPS code in the event of version changes to the main version, this step was not taken in the course of this work.
Instead, a workaround was found that does not require any changes to the LAMMPS implementation and delivers correct results as long as the simulated system is not too volatile, such as liquids or gases. As a first step the EAM potential file containing all the sampled pair-interaction- and density-functions was translated into a LAMMPS readable format. For this purpose, many of the existing sampling points of the functions had to be interpolated, and half of the pair interaction functions were omitted since all interactions are implicitly considered symmetric. A few of those functions are shown in the following figure.

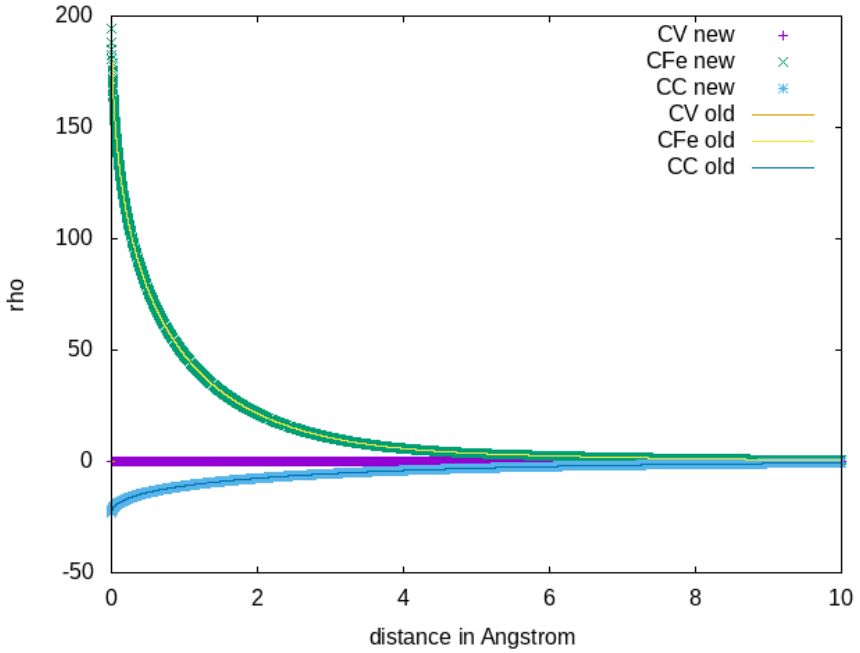

Figure 3: To be compatible with LAMMPS, the EAM-potential file had to be rewritten in a different format. This format requires, that the sampling starts at a distance of $R_{ij} = 0$Å and applies the same sampling range and interval to all functions. The plot shows three of the electron density functions $\rho(R_{ij})$. Thereby the 'new' resampled functions are laid behind the 'old' ones.

To carry out the relaxation step of a trial move, the previous and future positions of the carbon atom to be swapped are first determined. All particles within a sphere around these two points are extracted from the whole configuration and assigned to the central, transition or mantle zone depending on their distance from the center. The carbon atom is replaced by a virtual atom and inserted at the new position. These atoms are then added to the LAMMPS simulation with the command 'lammps_create_atoms(...)' and assigned to a corresponding group according to their zone. The particles of the transition and mantle zones are fixed at their respective positions with the help of the LAMMPS command 'fix setforce 0 0 0'. The subsequent relaxation can be divided into three sub-steps.
The first step involves canceling the pair interactions between the virtual and non-virtual atoms. This is done by modifying the neighbor lists by deleting the other particles from these lists. In addition, the virtual atoms are also kept in place. Subsequently, an energy minimization with 'minimize' takes place, which uses the 'FIRE' algorithm as the minimization procedure.
Afterwards the iron and carbon atoms should be in the local energy minimum. Therefore, in the second step, they are held in position and the virtual atoms can move freely

again. The neighbor lists are also completed so that the iron and carbon atoms interact with the virtual ones. The energy is then minimized a second time. This time only the virtual atoms move to their energetic minimum.
In the third and final step, the potential energy of the extracted particles is determined. To do this, the virtual particles are removed from the neighbor lists. Then the energy is calculated for each atom using the compute command 'compute pe/atom'. These energies can then be summed up to determine whether the performed trial move is accepted if the resulting energy is lower than that in the beginning.
If this is the case, the particles of the base configuration are replaced by the relaxed ones. This seamless integration is possible because neither the particles from the transition zone nor the mantle zone have changed their position and can therefore be exchanged without creating stresses in the surrounding system as the zone sizes are chosen accordingly to the cutoff radii.

3.1.2 Acceleration Techniques

The MDMC algorithm can be accelerated in several ways. With the present master-worker approach, several trial moves can be processed simultaneously. This includes the relaxation step, since the extracted spheres are mostly independent of each other. In case of an overlap of these volumes, the master must resolve this conflict and discard all but the best configuration. These problems as well as the resulting job queue management is further discussed in the work of H. Ganesan [9]. The expected speedup corresponds to the maximum number of simultaneously active worker processes. Since the master generates all trial moves and distributes them to the workers, including the extraction of the particles in the spherical areas, it reaches its limits with high processor numbers. For this reason, a parallel replica method was introduced to the MC++ code in addition to the master worker scheme. Its functionality and further details are covered in a later section.
Together with the implementation of the LAMMPS interface, the possibility of multiple processes working on the same trial move has also been added. This has the advantage that the relaxation step is accelerated for very dense particle packings or larger extracted spheres, as this is the most computationally intensive step.

3.2 Energy Prediction

Due to the relaxation step being so computationally expensive, this work addresses the question of whether it is possible to completely omit the relaxation step in some cases. The underlying idea is to use a method that can predict the expected energy changes of a trial move with a high degree of confidence. If the prediction is that a positive energy delta is to be expected, the relaxation step can be omitted and the next trial move can be

processed. The predicted energy for a configuration is based on the similarity to previous configurations. If two configurations are identical except for minimal differences, a similar potential energy can usually be expected. This comparison with past configurations requires the use of a database and an efficient similarity search to keep the overhead as low as possible. Another aspect that helps to minimize the additional overhead is that instead of the actual configurations, their multipole coefficients are compared, refer to 2.3.5. With a sphere radius of 20 for the extracted configurations, this means a difference of about 15,000 particles with 3 coordinates each to a total of 15 coefficients used for describing the configuration. In the following sections, the operation of the prediction algorithm and the sampling for the coefficients database will be discussed in more detail. Furthermore, different database models are discussed to make adding and searching for coefficients more efficient.

3.2.1 Prediction Algorithm

The process of predicting an energy difference for a given trial move can be divided into multiple steps:

1. After the worker has obtained the trial move configuration containing the two spheres of particles, consisting of center, transition and mantle particles, a further extraction step is performed. All IDs of the atoms within a distance r_{extr} from one of the centers are determined. This is done in order to further reduce the computational effort for the prediction step. Since the prediction algorithm relies solely on positional criteria and the displacement of the particles is greatest near the carbon insertion or extraction site, the region near these points is the most important. The extraction radius r_{extr} has to be chosen as high as necessary to still describe the configuration in detail but also as low as possible to reduce computing cost and therefore overhead as well as noise induced by describing too many particles with too few coefficients.

2. In the second step, the multipole coefficients of the extracted particles are determined. This is done by calculating the distance to the corresponding center of the sphere as well as the longitudinal and latitudinal angles for each particle. Then, as described in Section 2.3.5, the multipole coefficients are determined for each sphere center, but not summed up and kept separate from each other instead.

3. This is followed by a search for a similar sample in the coefficient bank. During this process, all samples with a deviation of less than 1% per coefficient are identified. However, it is essential that the first coefficient, the number of particles, matches exactly. If no corresponding sample can be found, the sample with the highest match is selected. In this context, “highest match” implies that, starting with the first coefficient, as many successive coefficients as possible must match thus

in the worst case only the first two match. If a larger number of similar samples are found, the one with the highest weight is prioritized. The weighting of the individual coefficients is discussed in a later section. If a sample with the highest match is identified at the end of the procedure, a predicted energy is also stored with it. Dedicated databases exist for the insertion and removal of carbon atoms. If no matching coefficients can be found in at least one of these databases, no energy prediction can be carried out.

4. The energy difference is determined by adding the energy predictions for the insertion and removal of the carbon atom and comparing it to the current potential energy of the trial move configuration. The quality of the prediction depends on a large number of parameters. It is essential that the database has been adequately trained. Furthermore, it should be noted that the required matching accuracy of the coefficients and the extraction radius r_{extr} have a significant influence on the result. In addition, it is possible to regulate the level of detail of the configuration descriptions by means of the order ℓ of the multipole expansion (see Section 2.3.5).

3.2.2 Configuration Sampling

The previous section described how the energy prediction works. What happens to the result of this prediction will now be explained in more detail.
If no similar sample could be found in the database for the current configuration, the usual relaxation step is carried out. The potential energy of the particles located within the corresponding sphere with radius r_{extr} is then determined. This energy is used in combination with the determined multipole coefficients to add new samples to both databases.
If a lower resulting energy is predicted for the current trial move, the relaxation step is carried out in each case as this is the desired result of a trial move. An additional sample is then added to each database. As part of the evaluation, the validity of the prediction is analyzed and the extent to which it deviates from the actual result. The credits of the corresponding sample in the database are adjusted depending on the correctness of the prediction. A detailed explanation of these sample-credits is provided in a later chapter.
If the algorithm predicts an increase in potential energy, the relaxation can be omitted and the trial move rejected. One problem that arises from this is that in this case no new samples can be generated for the database, as no energy is calculated. As a result, the configurations with a positive energy delta are also less frequently represented in the database in the long term. Since it is necessary that these cases can be reliably recognized and thus must be represented sufficiently often, a criterion should be found according to which a relaxation nevertheless takes place.
One such criterion is the probability p_{corr} with which correct energy predictions are made. At the beginning, this probability is 0% and increases with each correctly estimated energy change. If a higher energy is predicted as the result of a trial move, a cross-

check is initiated with a probability of $1 - p_{corr}$ by carrying out a relaxation anyway. This procedure ensures that, with sufficient reliability, more and more relaxations are omitted, but still a sufficient number are carried out to fill the database accordingly and make predictions.

3.3 Coefficient Database model

As previously mentioned, the choice of database structure plays an important role in minimizing the overhead required for energy prediction. The search for similar coefficients in the database takes the most time, but the insertion of new samples into the database should not be neglected either. In the following, three database structures are examined in more detail and their advantages, disadvantages and implementation are briefly discussed.

3.3.1 Linear Database

The linear database is the first and simplest implementation of such a structure. All samples are stored one after the other in a large list. The coefficients are not sorted. Assuming the default parameters, a sample has 21 individual values. Of these, 15 are multipole coefficients, one contains the predicted energy and the other 5 contain the weight and credits of the sample as well as 3 placeholders for future parameters. Adding an entry to this list has a negligible overhead of $O(1)$, since only one more entry is appended to the end. Searching for similar coefficients, on the other hand, requires considerably more effort. All entries in the database list have to be iterated. In addition, the coefficients of each entry must be iterated again to determine the degree of similarity, as described in Section 3.2.1. The total effort to find a similar coefficient therefore should have a complexity of $O(n)$, where n is the number of samples in the database.
In order to confirm the expected scaling behavior, several test runs were carried out. For this purpose, random multipole coefficients were generated, each assuming equally distributed values between -10 and 10. A total of $40,000$ of these randomly generated coefficients were entered into a linear database. After each insertion process, a random sample was searched for from the database and the time required for this was determined. These times are shown in figure 4 together with the function $f(x) = x$ and confirm the assumed scaling behavior of $O(n)$.

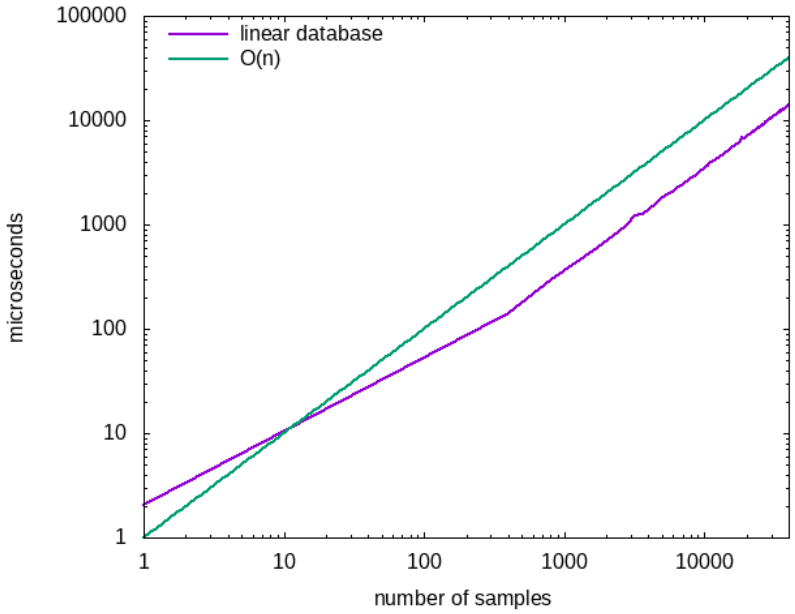

Figure 4: Necessary time in microseconds to search a random sample in a linear database with size N. The samples consist of randomly generated multipole coefficients, where each value is drawn from an equal distribution from -10 to 10.

3.3.2 Sorted Database

The first approach to speed up the database search is to sort the coefficients. For this purpose, a new database structure was created that maintains a separate list for each of the 21 entries of a sample. The first 15 lists containing the coefficients are sorted by size. The individual values are stored together with an index to enable the individual coefficients to be clearly assigned to a common database entry.
The sorting of the coefficients makes it possible to search for similar samples using binary search. For each of the 15 searched coefficients, the values in the 15 lists that are closest to it are searched for. Based on these values, all other values that lie within the tolerance are determined. Due to the sorting, these are immediately before or after the most similar value in the list. For the next step, the indices of these entries determined in this way are summarized in a list, so that in this case there are 15 separate index lists. Then, starting from the index list for the first coefficient, the largest possible intersection of all indices is determined. If the same index is represented in k lists, the first k coefficients of this entry are within the similarity tolerance. To subsequently obtain the predicted energy, the list of energies is searched for the entry with the corresponding index.
One problem with this database model is the insertion of new samples. This requires a re-sorting of all coefficient lists, which has a complexity of $O(n \cdot \log(n))$. Although the addition of new samples can be buffered so that the lists are sorted every n added entries,

the scaling behavior is still significantly worse than that of the linear database model. Further the Overhead introduced by the required additional indices and the algorithm for finding the biggest intersection make this approach undesirable for larger Databases which also can be seen in figure 5.

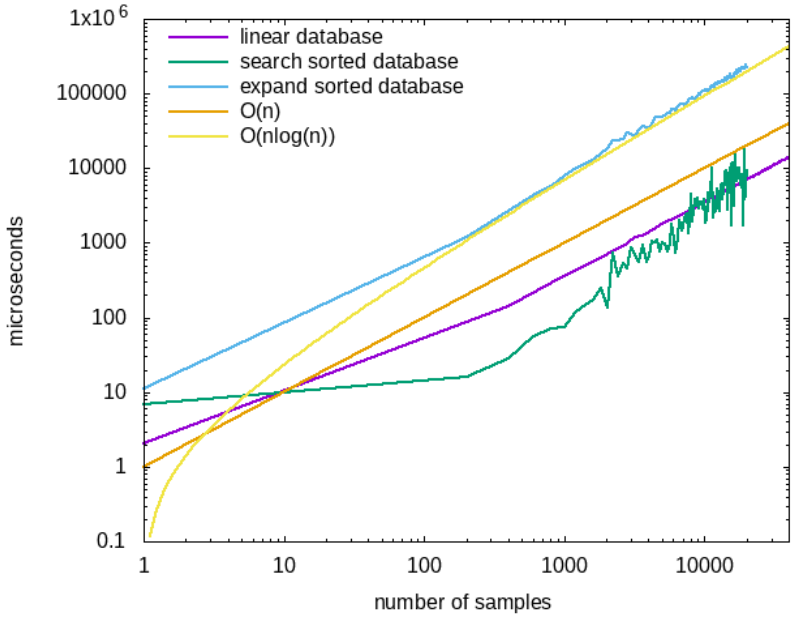

Figure 5: Necessary time in microseconds to search a similar sample in a sorted database of size N using binary search and index-intersection. While faster for database sizes of about $N = 1000$ the Sorted-Database model has worse scaling than the linear approach. Furthermore the addition of samples to the sorted database requires a resorting of the coefficient lists. The necessary time for sorting is much higher than the search times as it scales with $O(n \log(n))$

3.3.3 Multidimensional Database

The multidimensional database is a modification of the linear database model. The underlying idea is to replace the single large sample list with several smaller ones in order to speed up the search by reducing the length of each list. For this purpose, a binning procedure is applied, which roughly sorts the multipole coefficients so that similar coefficients are located next to each other in the database.
Assuming a three-dimensional database with $h = 5$ bins in each dimension is to be generated, the first three coefficients c_0, c_1 and c_2 of each sample c in the database are used for binning. The bin sizes b_d for each dimension d are calculated from the smallest

and largest occurring values for the d-th coefficient across all samples as follows:

$$b_d = \frac{\| \max(c_d) \| + \| \min(c_d) \|}{h - 2} \tag{27}$$

Thereby two bins in each dimension are exclusive to those samples whose coefficients lie outside the former lowest or highest value and could not be into an existing bin. Therefore the discussed example database effectively has only three bins in each dimension. In three dimensions, the database structure can be imagined as a cube consisting of h^3 small cubes. Each of these small cubes represents a list of samples. If a new sample is to be added to the database, the spatial coordinates of the corresponding cube or list is determined by assigning the first three coefficients to a bin. The sample is then added to the corresponding list. If a similar entry is to be found in the database for given coefficients, the list index is determined on the same way. Then it is iterated just over all samples of this list and all $3^3 - 1 = 26$ neighboring lists in the database. This saves a considerable number of comparisons that would have had to be made using the linear database model.

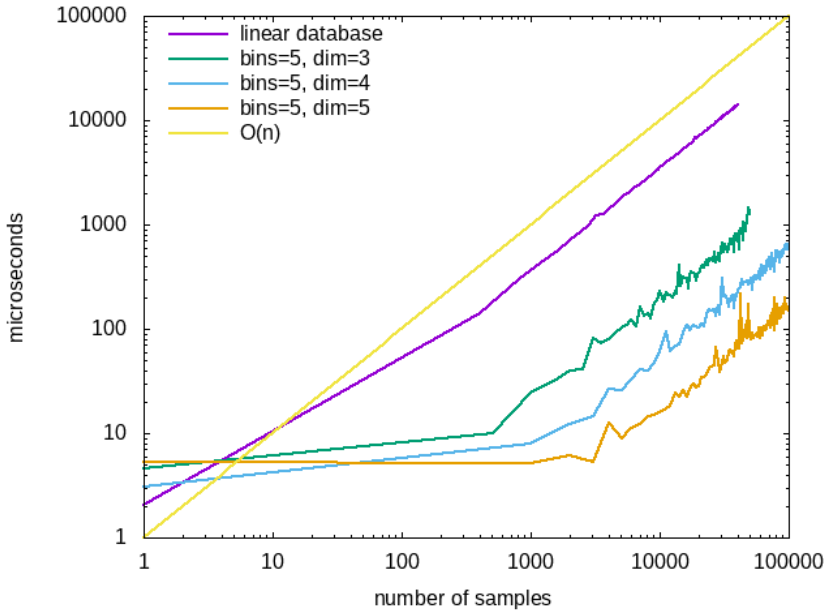

Figure 6: Search times for the linear database model compared to the multidimensional database. The basic three-dimensional database with five bins is about ten times faster than the linear one. An increase in dimension reduces the necessary search time further. Meanwhile the scaling behavior of $O(n)$ stays the same for high number of samples. Also the multidimensional database model introduces an overhead for determining the bin indices, which can be seen well for low database sizes.

As shown in figure 6, the database with three dimensions and five bins used as an exam-

ple already shows a significant reduction in search times by a factor of about 10 compared to the linear database model. It can also be seen that for large amounts of data the multidimensional database has the same scaling behavior of $O(n)$ as the linear database. This can be explained by the fact that no changes were made to the search procedure itself, but only the list length was significantly reduced by binning. The binning of the coefficients also introduces an overhead, which can be seen in the plot for small amounts of data.
Furthermore, an increase in the dimension of the database has a positive effect on the search times. For the four- and five-dimensional database, correspondingly more coefficients are used for binning, so that the five-dimensional database has $5^5 = 3125$ lists instead of $5^3 = 125$. As can be seen from the plot, this additionally shortens the search time by a factor of up to 10.

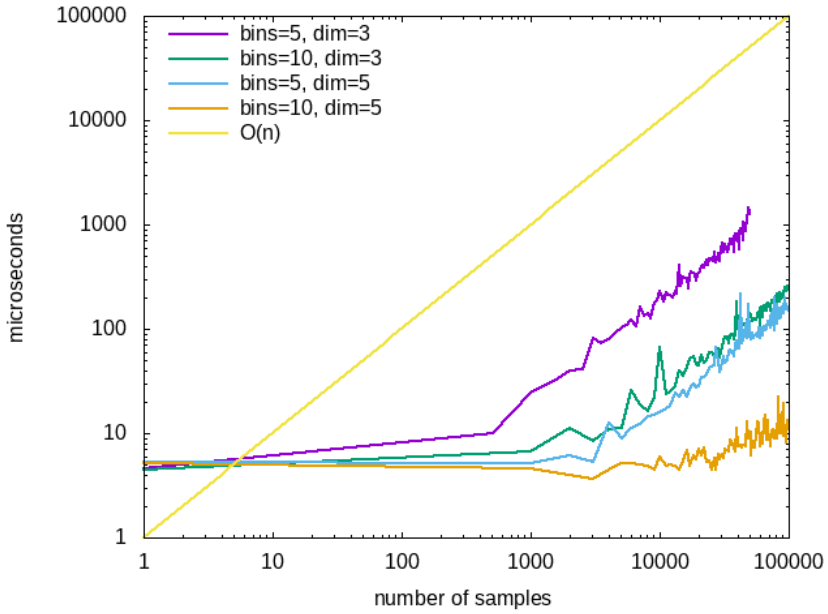

Figure 7: Increasing the bin count while keeping the dimension of the database the same shortens the search times as well. As the green and blue curve show similar times, the universal measure to estimate performance seems to be the number of lists the database contains that can be calculated by $N_l = h^d$ where h is the number of bins and d the number of dimensions.

Increasing the number of bins also shortens the search time required. The figure above shows the search times for the three- and five-dimensional database with five and ten bins in each dimension. In addition to the obvious time savings, it is also noticeable that the search times for the three-dimensional database with ten bins and the five-dimensional database with five bins are similar. A comparison of the number of lists shows that the three-dimensional database with $10^3 = 3000$ has almost as many lists as the five-

dimensional database with $5^5 = 3125$. This therefore appears to be a valid comparison criterion for estimating the performance for different combinations of dimension and bin numbers. The number of lists in such a database can be calculated by $N_l = h^d$ where h is the number of bins and d the number of dimensions.

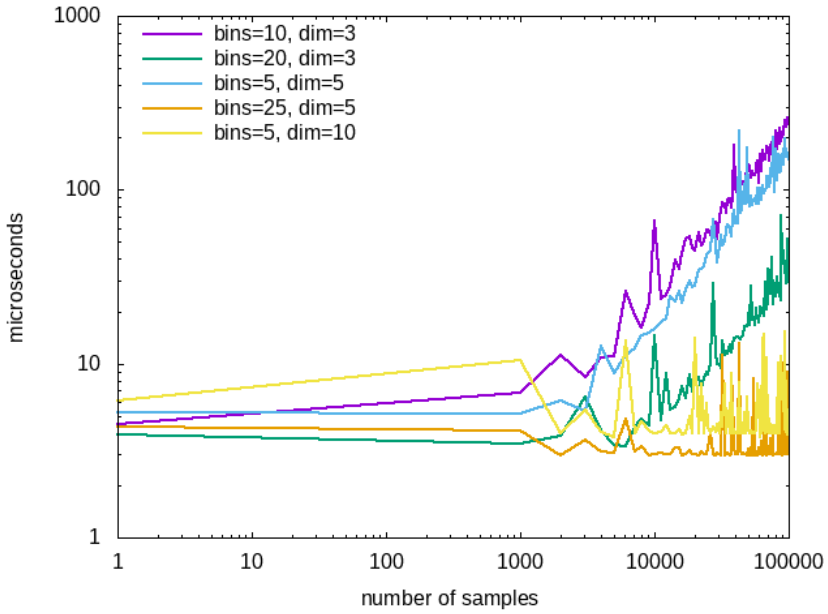

Figure 8: If the dimensions or number of bins are increased further for the multidimensional databases the search time can be reduced to be nearly constant in theory. As long as the number of samples in the database is low enough this holds true.

If the number of bins for the five-dimensional database is further increased to 25, the typical $O(n)$ scaling behavior is no longer recognizable for up to 10^5 entries. Provided that all lists are filled equally, this database has to contain $9,765,625$ entries before one list got more then one sample. The determination of the list index is therefore sufficient in theory to find the most similar coefficient in the database, which guarantees an almost constant search time.

One problem with this division of the bins is that the bins could become too narrow, so that it is also necessary to search the second-nearest neighbors of the particular list, as they are also within the similarity tolerance. Another problem occurs when many new coefficients with values beyond the bin boundaries are added to the database. As already mentioned, these coefficients are stored in the bins at the edge. As there are only two edge bins per dimension, there are $2^5 = 32$ lists in this database to accommodate these outlier coefficients.

An alternative is therefore to use a database with $d = 10$ dimensions and $h = 5$ bins each. It contains the identical number of lists but has $2^{10} = 1024$ bins to accommodate the mentioned outlier coefficients. In addition, the bins are five times wider, so the first

problem mentioned should be less of an issue. As one can see in figure 8 the search times for this database are also comparable except the greater overhead introduced by the increased amount of dimensions.
The storage space required for such a high-dimensional database is also limited. Assuming space is reserved for one entry per list and such an entry contains 21 values. If these values are stored as floats, the required memory is $9765625 \cdot 21 \cdot 4\text{byte} \approx 820\text{Mb}$. It should be noted that 100,000 samples are often more than sufficient for energy prediction in practice.

3.3.4 Synchronization

In order to relieve the master process of additional work, the procedure for predicting the energy was outsourced to the worker processes. This makes it necessary for each worker to have a copy of the coefficient database, as no shared memory can usually be assumed between the processes. One problem that arises from this is that the workers add new samples to the database independently of each other and these therefore continue to develop independently of each other. In order to improve the quality of the energy prediction, the databases should be synchronized. The workflow should not be disturbed and therefore synchronization should not take place with every change in the database. Thus the worker waits until a defined number of new entries are added. A synchronization request is then sent to the master process, which forwards it to all other workers.
The synchronization itself can be carried out in two different ways. In the first variant, only the changes or new entries are communicated. This approach is particularly suitable for the linear database model, as the new entries are bundled at the end of the list. All changes communicated in this way are collected and merged by a worker. This worker then sends the merged changes back to the other workers, which include then in their databases.
The second approach is to exchange the entire database between the workers. When using a multidimensional database, this procedure is in fact not necessary, but it makes sense to use the synchronization to reselect the bin boundaries. The aim is to ensure that the lists in the database are filled as evenly as possible and that there are no more entries in the marginal bins. This requires a re-binning of all entries. If one worker has rebuilt the entire database this way, it is send back to all other workers, who exchange their database to the new one. This step seems costly in terms of communication. In practice though not more than $40,000$ samples with 21 values each are usually send. In comparison to that a common trial move contains $20,000$ atoms that require 7 values each.

3.3.5 Optimization

To improve the prediction quality, the database is optimized at regular intervals, usually in the course of synchronization. This means a reduction of the database content while maintaining the information content. In the course of optimization, duplicates or quite similar entries are searched for. These duplicate entries are removed from the database as they do not contain any additional information and fill the corresponding bin unnecessarily. It is also possible, for example, to remove the oldest samples from the database as they may be irrelevant to the current state of the simulation. In addition, the entries in the database that have often led to incorrect predictions can be determined. As these are most likely to be outliers, they will also be removed.

3.3.6 Sample Weight

The sample weight was introduced in order to recognize that one coefficient configuration is sampled more frequently than others. When searching for duplicates in the database, their sample weights are averaged and 1 is added afterwards. This weight is then assigned to the remaining entry. If several coefficients within the tolerance are found during the similarity search, the one with the highest sample weight is selected for the prediction.

3.3.7 Sample Credits

The sample credits are a measure of the predictive reliability of an individual sample. They are made up of two values. One is the number of energy predictions for which this sample was used. The second is a running counter, which is increased by one for each correct prediction and subtracted by one for each incorrect prediction. If this counter is 0 after N predictions, an incorrect prediction was made in 50% of the cases. When optimizing the database, a threshold can be set below which all samples with a lower value are removed from the database. As a prerequisite for this, a minimum number of predictions can be defined so that individual samples are not removed due to a few outliers.

3.4 Parallel Replica Method

As already mentioned in Section 2.4, the Parallel Replica method was implemented in the MC++ code to increase parallel performance. This requires each process to have a copy of the entire particle configuration. In addition, some steps such as the generation and evaluation of a trial move also had to be relocated from the master to each worker.

A further adjustment was made so that the replicas are no longer compared at regular intervals, but as soon as a worker has accepted a certain number of trial moves. This number can be adjusted during the course of the simulation so that not too many system comparisons are made at the beginning and not too few as the number of iterations increases.

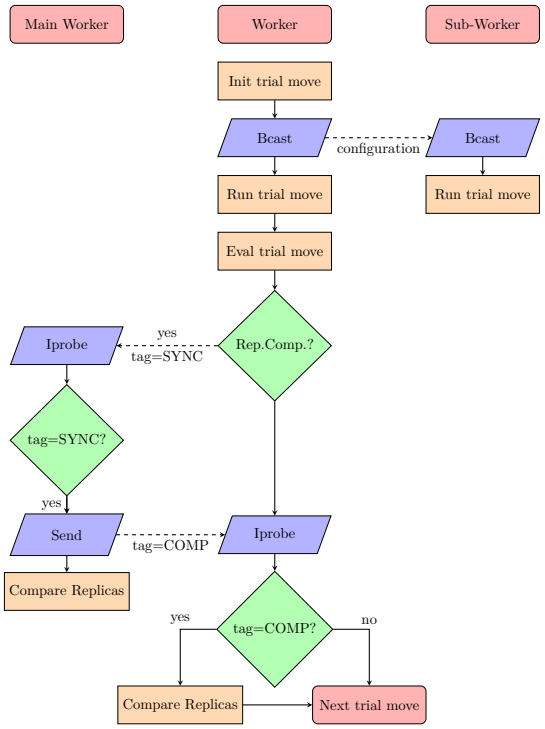

Figure 9: The flow-chart shows the crucial steps of the implemented parallel replica method. Thereby each worker fulfills the task of generating and evaluatingtrial moves while also providing its 'Sub-Workers' with the particle configurations necessary for the relaxation. Those 'Sub-Workers' are only present, if more than one processor should handle the LAMMPS relaxation step. The 'Main Worker' has an additional task as it should handle the interruptions for comparing replicas. If any worker accepted a specified number of trial moves, an interruption message is send to the coordinating worker, who then forwards this message which leads to all workers invoke a replica compare. Afterwards the independent evaluation of trial moves is continued.

The workers can be divided into three groups (see figure 9). The first group consists of the 'Workers' who are handling the important steps for the simulation. Those workers are closely connected with their 'Sub Workers', whose only purpose it is to run the LAMMPS relaxation step in parallel after being provided with the particle configuration

from their respective worker. The third group consists of just one 'Main Worker' who has the additional task to handle the interruptions for the comparison of the replicas.
The simulation cycle starts by each worker initializing the next trial move independent of each other. This includes choosing a carbon and a virtual atom by random or biased sampling. After that the two spherical configurations are extracted from the main system. Based on this trial move configuration the energy prediction algorithm is performed by the worker. Depending on the result the current trial move is either dismissed or the relaxation is executed if a lower energy tendency is predicted, a crosscheck is performed or no prediction could be made. In the case of execution the trial move configuration is broadcast to the sub-workers if there are any present. Afterwards the relaxation step with LAMMPS is executed in parallel with the sub-workers and the results are evaluated. In case of a successful trial move the base system is updated through integration of the trial move configuration. If no relaxation is to be performed a flag is broadcast signaling the sub-worker to not wait for a trial move configuration within this iteration step. Given that a worker has accepted the specified number of trial moves, it sends a message containing its rank together with the tag SYNC = 2 to the main worker at rank 0. As the send operation in non-blocking the worker continues to evaluate further trial moves while a flag prevents multiple interruption requests from this worker. Termination requests due to reaching the total amount of evaluated trial moves are also handled within this section of the code. Those messages use the tag TERM = 4 to be distinguishable from other messages.
The code section following the evaluation and the interruption invoking handles the forwarding of the worker interruption signal, which is shown in a greatly simplified form in figure 9. The main worker uses an 'Iprobe' to retrieve the next pending message from any source with any tag. If this message has a SYNC tag, it is received via an 'Irecv' followed by a 'MPI_Wait'. Since the main worker participates in the replica compare like any other worker, its 'compareConfigs' flag is set to true. The main worker then forwards the message content to all other workers. To do this, it uses a non-blocking 'Isend' with the tag COMP = 3. However, if the received message has the TERM tag, the 'finished' flag is set to true to end the simulation and a message with the same tag is forwarded to all other workers via a non-blocking send. Afterwards the same tag is forwarded via broadcast to the sub workers if a message was received. In all other cases the NEXT tag is used to notify the sub workers that the next trial move is processed.
All other workers handle this section similarly, with the exception that they wait exclusively for messages from the main worker. Instead of the SYNC tag, the messages are checked for the COMP and TERM tags and the corresponding flags are set when they occur. The sub workers are notified via a broadcast in the same way as it is the case for the main worker.
The sub workers on the other hand wait exclusively for the broadcast signal from their associated workers. Since sending a termination signal at this point in the code was considered essential for the correct termination of the program, this broadcast is also used to inform the sub workers that a replica compare has been initiated, even if they are not participating in it in the current state of the program. For further clarification of the interruption handling an excerpt of the code can be seen in figure 10.

```
while(!finished){
    [...]
    //Handle interrupt
    if(worldRank==0){ //Main Worker
        MPI_Iprobe(MPI_ANY_SOURCE,MPI_ANY_TAG,mTComm,&flagSYNC,&Stat);
        if(syncReqSend||(flagSYNC==1&&Stat.MPI_TAG==SYNC)){
            bcTAG=COMP;
            if(!syncReqSend){
                MPI_Irecv(&rank,1,MPI_INT,Stat.MPI_SOURCE,SYNC,mTComm,&req);
                MPI_Wait(&req,&status);
            }else{rank=0;}
            compareConfigs=true;
            for(auto i=1; i<mTSize; ++i){
                MPI_Send(&rank,1,MPI_INT,i,COMP,mTComm);
            }
        }else if(terminationReqSend||(flagSYNC==1&&Stat.MPI_TAG==TERM)){
          bcTAG=TERM;
          if(!terminationReqSend){
            MPI_Irecv(&rank,1,MPI_INT,Stat.MPI_SOURCE,TERM,mTComm,&req);
                    MPI_Wait(&req,&status);
          }else{rank=0;}
          finished=true;
            for(auto i=1; i<mTSize; ++i){
            MPI_Send(&rank,1,MPI_INT,i,TERM,mTComm);
          }
        }else{bcTAG=NEXT;}
        MPI_Bcast(&bcTAG,1,MPI_INT,0,tComm);
        flagSYNC=0;
    }else if(threadRank==0){ //Worker
        MPI_Iprobe(0,MPI_ANY_TAG,mTComm,&flagCOMPARE,&Stat);
        if(flagCOMPARE==1&&Stat.MPI_TAG==COMP){
            bcTAG=COMP;
            MPI_Recv(&rank,1,MPI_INT,0,COMP,mTComm,&status);
            compareConfigs=true;
        }else if(flagCOMPARE==1&&Stat.MPI_TAG==TERM){
            bcTAG=TERM;
            MPI_Recv(&rank,1,MPI_INT,0,TERM,mTComm,&status);
            finished=true;
        }else{bcTAG=NEXT;}
        MPI_Bcast(&bcTAG,1,MPI_INT,0,tComm);
        flagCOMPARE=0;
    }else{ //Sub Worker
        MPI_Bcast(&bcTAG,1,MPI_INT,0,tComm);
        if(bcTAG==COMP){
            compareConfigs=true;
        }else if(bcTAG==TERM){finished=true;}
    }
    //Replica compare
    if(compareConfigs){MPI_Barrier(sComm_); [...]}
}

```

Figure 10: Shortened version of the code section that is responsible to forward interrupts and coordinate the initiation of the replica compare.

Within the replica compare procedure the worker with the lowest potential energy in its system is determined by executing an 'Allreduce'. Afterwards the worker containing the minimum energy state transfers its whole particle configuration into a buffer, which is then broadcast to all other workers but not sub-workers. The broadcasting procedure is then repeated for several sampling parameters needed for the biased sampling as well as for various tracking variables and counters. After comparing and exchanging replicas each worker continues to process trial move independently.
The main advantage of the parallel replica method over the master-worker approach is that it does not require at least two communication steps to process each trial move. Without the energy prediction algorithm, this does not pose a problem, as each relaxation takes an average of 6 seconds. This way, the master process is able to supply more workers with jobs at the same throughput of sent trial move configurations. The introduction of energy prediction has drastically reduced this time, as only the trial move configurations that lead to a lower system energy are relaxed. Assuming an optimal prediction algorithm, the proportion of relaxations performed approximates the average acceptance probability of the trial moves. Depending on the particle configuration under investigation, this is initially around 1% and decreases sharply at the beginning of the simulation and then more slowly thereafter. Compared to the relaxation step, the energy prediction takes only a few milliseconds. Due to this fact, the master-worker approach quickly reaches its limits, as the master is fully occupied with sending new trial moves, most of which are rejected after a very short time.
Apart from the replica compare, the parallel replica method has virtually no communication overhead. Since the processors do not interact with each other apart from this, this method should offer a near-optimal speedup. However, for high processor counts, another phenomenon becomes apparent that appears to reduce the algorithmic efficiency. Although there is only a low probability of acceptance for the trial moves, most of the execution time is spent relaxing the few supposedly accepted trial move configurations. When a processor invokes a replica compare, it takes some time for this message to reach all other processors, as the current trial move is processed to completion before the 'Iprobe' is reached. Due to the distribution of execution time, there is a high probability that other processes will also have performed a relaxation and accepted a trial move during the same time period. This means that at the time of the replica compare, there are several accepted trial moves, but only one of these systems is retained for the continuation of the simulation and all others are discarded. Due to the fact that more accepted trial moves are found than are adopted into the main configuration, the effective acceptance probability for the parallel replica method with many processors is expected to be lower than for the serial variant.

4 Evaluation

4.1 Energy Prediction

This section focuses on the energy prediction algorithm that has been introduced to the MC++ code. First there are discussed some basics regarding the behavior of multipole coefficients. Afterwards a closer look is taken at how well and how fast the configuration space can be sampled and stored in the database. Besides evaluating the reliability of predicting the correct energy trend it is also analyzed how accurate the energy amount can be predicted and which parameters can be changed for further improvement.

4.1.1 Sensitivity of Multipole Coefficients

In order to enable a quicker comparison of two particle configurations, these are transferred to an alternative representation. For this purpose, the first 15 multipole coefficients are calculated based on approximately 200 particle coordinates. These are then used to search the database for similar configurations. Due to the drastic information reduction from about 600 floating point numbers to 15, it is of great importance that both coarse and fine differences between two configurations can be identified using the multipole coefficients.
The following section will therefore take a closer look at how the displacement of individual particles affects the multipole coefficients. Small changes in the configuration of the particles should also cause small changes in the multipole coefficients to ensure better comparability. At the same time, changes in position near the center of the sphere should have a greater effect on the coefficients, since this is where the greatest change occurs due to the insertion or removal of the carbon atom see Eq. (26).
In order to demonstrate the desired properties using an example, a multipole coefficient of the second order, in particular the first moment, is considered in more detail. The formulas for calculating the multipole coefficients of second order for different moments m are listed below.

$$\frac{\left(0.75\cos\left(\theta\right)^2 - 0.25\right)\left(\cos\left(\phi\right) + i \cdot \sin\left(\phi\right)\right)}{r} \quad m = 0 \tag{28}$$

$$\frac{0.5\cos\left(\theta\right)\sqrt{\cos\left(\theta\right) - 1}\sqrt{\cos\left(\theta\right) + 1}\left(\cos\left(\phi\right) + i \cdot \sin\left(\phi\right)\right)}{r} \quad m = 1 \tag{29}$$

$$\frac{-0.125\sin\left(\theta\right)^2\left(\cos\left(\phi\right) + i \cdot \sin\left(\phi\right)\right)}{r} \quad m = 2 \tag{30}$$

To illustrate the behavior of this multipole coefficient with respect to position changes, a configuration with only one particle was generated. This is located at a fixed position with distance $r = 1$ to the center and the angles $\phi = \frac{\pi}{3}$ and $\theta = \frac{\pi}{5}$. These values were

chosen arbitrarily with the exception that none of them sets the coefficient to 0.
In the following figure, the distance r was increased by δr and the resulting change of the multipole coefficient was plotted. The same procedure was repeated for all distances $r = 1 \dots 10$, whereby the particle distance δr to the center was varied between 0 and 10.

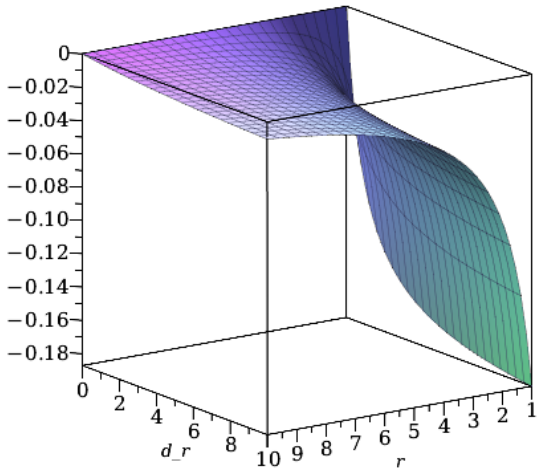

Figure 11: Deviation of the multipole coefficient of 2nd order and 1st moment if varying the distance r of the considered particle to the origin. The angles ϕ and θ are fixed at $\frac{\pi}{3}$ and $\frac{\pi}{5}$ respectively. The starting distance of the particle to the center is r which is then moved outwards by d_r. In that sense the graph shows the derivative in respect to r for the specified point.

As can be seen in figure 11, moving the particle away from the center causes a decrease in the multipole coefficient. The decrease is greater the closer the particle is to the center. For distances from about 8Å a shift only has a marginal effect on the coefficient. Thus it might only be worth it to compute the multipole coefficients for all atoms that lie within this range. As the functions of the multipole coefficients behave similar to $\frac{1}{r}$ close to $r = 0$, during the trial moves that insert a carbon atom at the center, this carbon is excluded from the coefficient computation.
In addition to the variation of the distance, changes in the two angles ϕ and θ were also considered. For this purpose, the particle was fixed at a small distance of $r = 1$ and $\theta = \frac{\pi}{5}$. The angle ϕ was then varied from 0 to 2π and the change in the coefficient was plotted.
As can be seen from figure 12, the decrease or increase of the coefficient depends on the starting position and the direction of rotation. Furthermore, the deviation of the

coefficient is radially symmetrical and has a maximum. Consequently, it is possible for two particles to have the same multipole coefficient at different locations or after an opposite rotation.

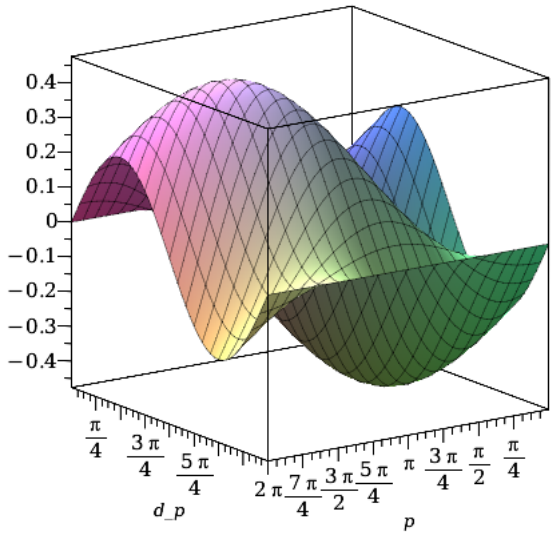

Figure 12: Deviation of the multipole coefficient of 2nd order and 1st moment if varying the angle ϕ of the considered particle to the origin by d_p. The radius r is fixed at 1 the angle θ at $\frac{\pi}{5}$. From its starting position the angle ϕ is varied up to 2π.

For this reason, it is necessary to use several coefficients, including those of higher order, to describe the position of a particle as clearly as possible. These coefficients have a higher frequency of coefficient deviation in the scenario under consideration, which is why they eliminate many of the problematic symmetries in combination with the lower-frequency coefficients. On their own, the higher order multipole coefficients could describe the configuration in greater detail. However, these coefficients have even more symmetries than the lower order ones, which is why they are only advantageous in combination with each other.
The behavior of the multipole coefficient to changes in θ are similar to that for altering ϕ. While ϕ is fixed at $\frac{\pi}{3}$, θ is varied between 0 and π as θ is the elevation angle. This elevation is then changed by up to π so even beyond the usual maximum angle to see if the coefficients also behave symmetrically in θ. The corresponding graph can be seen in figure 13. Note that in practice only the coefficient changes in the further half of the graph occur.

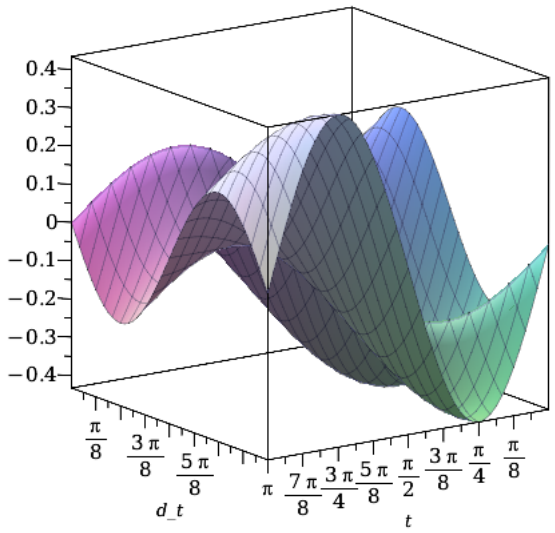

Figure 13: Deviation of the multipole coefficient of 2nd order and 1st moment if varying the angle θ of the considered particle to the origin by d_t. The radius r is fixed at 1 the angle ϕ at $\frac{\pi}{3}$. From its starting position the angle θ is varied up to π.

4.1.2 Sampling configuration space

A possible measure of the coverage of the configuration space by the coefficients database is the prediction chance. It is calculated from the probability that for any given valid trial move configuration multipole coefficients are found in the database that are similar. This probability depends not only on the number of samples in the database or the number of coefficients per sample, but also on the required maximum percentage deviation of the coefficients from each other, as described in Section 3.2.1.
In figure 14 the development of the prediction chance is plotted for the simulation starting from an empty database. The four graphs are distinguished by applying different combinations of high or low deviation and slow or fast learning. Low deviation in this context means that the maximum allowed percentual deviation of two coefficients is lower for them to count as similar. A high deviation is thereby more tolerant and allows a wider range of samples to count as similar which improves the chance to find a fitting sample in the database. It should be noted, that choosing a higher deviation possibly has bad influence on the quality or accuracy of the prediction, which is discussed in a later section.

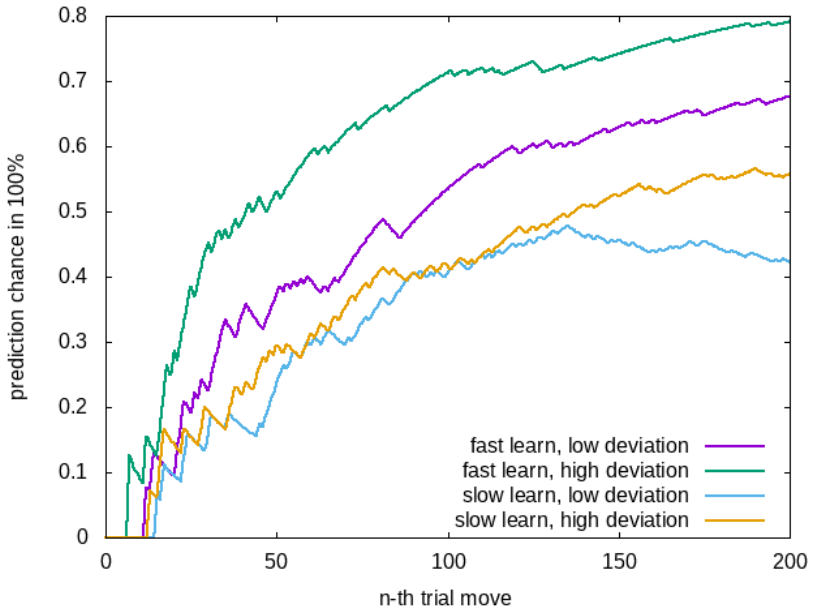

Figure 14: Development of the prediction chance if each of the four workers starts from an empty database. The prediction chance is the probability to find a similar sample in the database for a given trial move configuration. If the maximum allowed deviation for two coefficients to count as similar is lower, the prediction should get more precise, but it takes more samples in the database to fill the gaps. Increasing the allowed deviation accelerates the sampling of configuration space. If 'fast learning' is active the databases of all workers are synchronized, which potentially quadruples the number of samples in each of them.

Each worker has its own instance of a coefficient database, which is filled with samples from the database initialization file. If this file is empty, each worker starts developing its database from zero independent of each other. To improve each database faster, a synchronization method was introduced which merges all scattered databases and provide each worker with the combined version. This procedure was denoted as 'fast learn' in comparison to the 'slow learn' where each worker independently grows its database. For just four workers this already provides a significant acceleration of configuration space sampling as can be seen in figure 14. It even has a bigger impact on the prediction chance than choosing a higher coefficient deviation. Therefore to speed up the prediction chance gain the focus should lie on adding additional workers instead of increasing the allowed deviation, which might also harm the prediction quality itself.
The easiest way to determine the prediction quality is to determine the probability of a correct prediction being made. Correct in this context means that only the trend has to be predicted correctly, i.e. whether the system energy decreases or increases as a result of a trial move. In figure 15 this probability was calculated and plotted for the simu-

lations in figure 14. There are no significant differences in the predictive quality of the four variants. After 200 trial moves, all four have a probability of correctly predicting the energy trend of about 90%. This rapid increase can be explained by the fact that in most cases the system energy increases and only in the rarest cases does a trial move lead to success. During an average MDMC simulation without weighted sampling with 16000 trial moves, only 25 were accepted. This corresponds to a success rate of approximately 0.15%. Since only the coefficients that occurred during the simulation end up in the coefficient database, the configurations that lead to a lower system energy are highly underrepresented. In the 'initial' phase the database and energy prediction are therefore mainly used to recognize the trial moves that lead to a higher system energy.

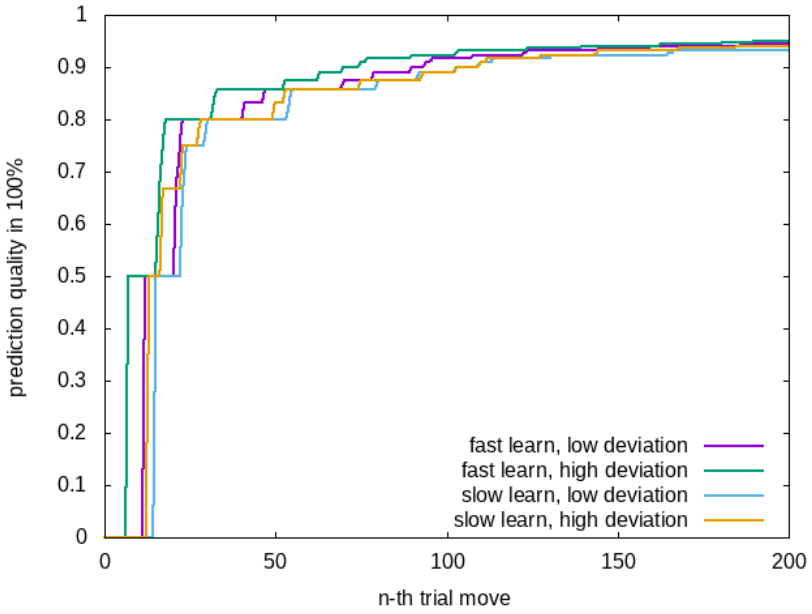

Figure 15: The prediction quality is the probability that the trend of the system energy after a trial move is predicted correctly, i.e. whether the system energy decreases or increases as a result of a trial move. In contrast to the prediction chance there is no significant difference between the four plotted variants. As lower system energies resulting out of a trial move are quiet rare, the prediction quality is mainly tracking the rate at which higher system energies are correctly predicted.

These results seems sobering at first, but there are several ways to identify a successful trial move.

- **Predicting a lower energy**
  If there are enough samples in the database, it is also possible to reliably predict events that occur only rarely. However, this requires a considerable number of

sampled configurations, as these events are not only rare, but their phase space is also larger. This can be explained by the fact that the system under investigation is for the most part almost exclusively homogeneous and significant lattice distortions only occur in the vicinity of the forming Cottrell atmospheres. This makes it even more difficult to increase the prediction chance for these rare cases.

- **Erroneous prediction with crosscheck**
  If a positive energy trend is predicted for a trial move, a cross-check is carried out with a certain probability, as described in Section 3.2.2. The probability with which the relaxation is nevertheless executed is calculated from the counter probability of the prediction quality. In the example above, this is $1 - 0.94 = 6\%$. If a lower system energy is detected when this cross-check is carried out, this leads to acceptance of the trial move. However, the probability of this is very low.

- **No similar sample in the database**
  The most likely case to identify a trial move that leads to a lower system energy is that no similar sample could be found in the database. As already mentioned, the phase space for the unsuccessful trial moves is smaller and can be sampled much faster. If no similar coefficients can be found in the database and the database can recognize higher energy tendencies quite successful, the probability that a lower system energy will be detected during a relaxation is higher, since it is a configuration that has not yet occurred during the current or previous simulations, which makes it to a rare event.

With increasing prediction chance and quality, a correct prediction of the change in system energy can be made for more and more trial move configurations. As a result, an increasing number of relaxations can be omitted. This is also reflected in the time required to perform N trial moves. In figure 16 these times are plotted for the sampling approaches discussed in this section. For comparison, the required time if no energy prediction takes place and therefore all relaxations are carried out is also plotted.
It can be clearly seen that the energy prediction itself, but also an accelerated learning phase of the database, has a strong time-reducing effect on the runtime of the simulation. An important aspect is the extent to which favorable trial moves are overlooked by omitting relaxations. Therefore multiple simulations were run both with and without energy prediction. Each simulation included $16,000$ trial moves that were divided between four workers. For every worker parameters like the number of relaxations and the number of predictions were tracked.
During the simulations without energy prediction all of the $16,000$ relaxations were run which took a bit more than 9 hours to complete. Of all trial moves just 22 to 36 were accepted. In contrast to that the simulations with active energy prediction finished in 1.5 hours. The average prediction chance at the end of the simulation lied at around 85% and the prediction quality at 93%. The database contained about $35,000$ samples in total. Approximately $14,000$-times an energy could be predicted whereby about 40 of them predicted a lower energy. The prediction of a lower energy was wrong approx. 50%

of the time, whereas all of the about 320 crosschecked higher predictions were correct. In total about 2600 relaxations were run and approximately 27 trial moves were accepted during the simulation.
While those findings do not guarantee that no successful trial moves are omitted they strongly suggest, that it might be not a significant part.

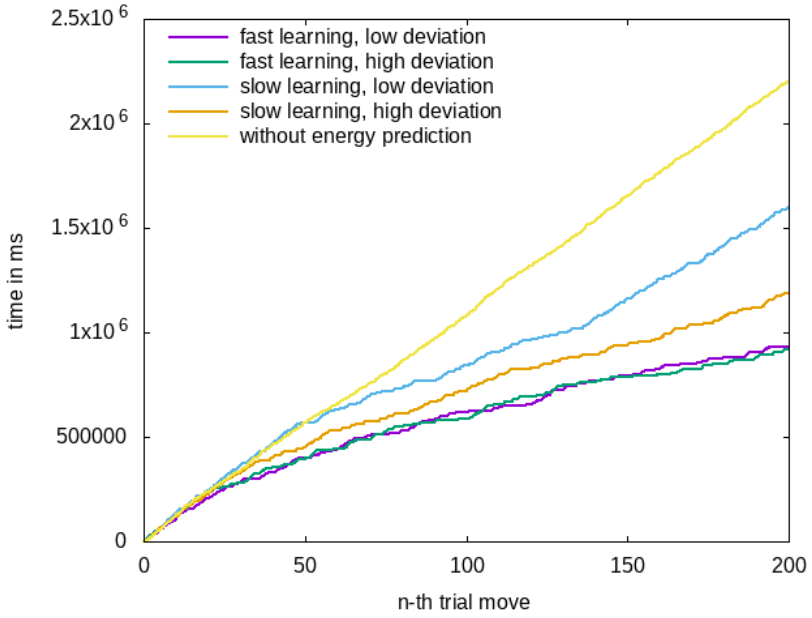

Figure 16: The graphs shows the development of the cumulative runtime for the four prediction based approaches compared to the version without energy prediction. The simulations are initialized with a empty coefficient database. While the former approach runs all relaxation steps, the simulations that apply energy prediction are neglecting more and more relaxations in favor of a significantly shorter runtime. The runtime development is thereby closely connected to the prediction chance as mentioned in Section 3.2.2 and which can be seen in figure 14.

4.1.3 Prediction accuracy

While the correct prediction of the energy trend has been used as a quality feature so far, this section takes a closer look at the accuracy with which the exact amount of energy can be determined. To accomplish this, the percentual deviation from the actual energy is calculated for each prediction made. A deviation of 100% means that the predicted energy is twice as high as the predicted energy. As long as this deviation is positive, the energy tendencies of the underlying trial move match. As soon as it falls below zero, the prediction is incorrect in terms of the previously discussed prediction quality. The aim

of this section is to identify various parameters that have a positive or negative effect on the prediction accuracy and, if necessary, to estimate optimal parameters.

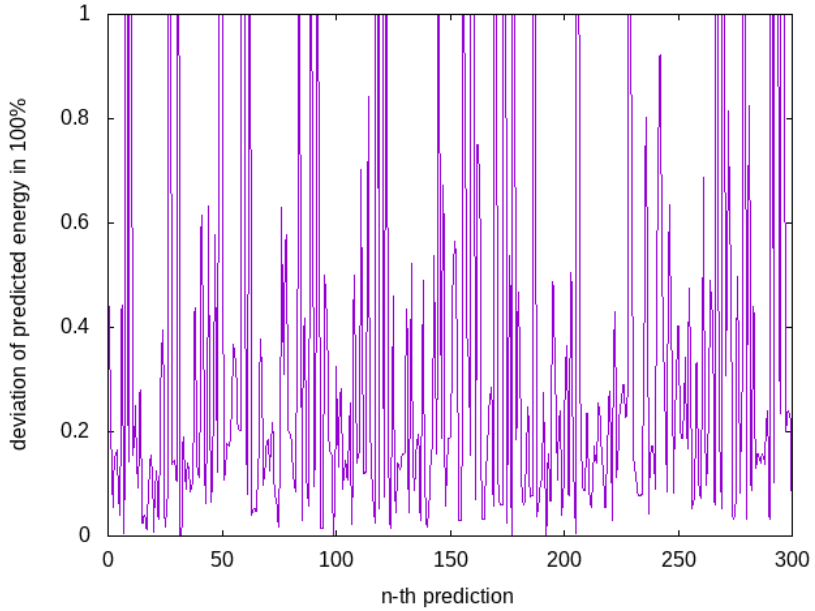

Figure 17: Percentual deviation of the predicted energy based on the multipole coefficients compared to the actual energy of the configuration after the relaxation step. To calculate the multipole coefficients a spherical configuration with radius $r = 8$ was extracted from the system. While a few predictions are far off the actual value the median of the percentual deviation lies at around 17%.

In figure 17 the percentual deviation of the prediction to the actual energy is plotted. The radius of the extracted sphere for calculating the multipole coefficients is 8Å. Apart from a few outliers that predicted a much too high energy, most values are in the range between 0% and 60%. Among the 300 predictions there are only a few incorrect ones with a deviation below zero. The median of the considered values is 17%, which is an unexpectedly low deviation considering that the whole trial move configuration is represented by just 15 multipole coefficients.
It is therefore reasonable to assume that the energies in the database also vary little and that there are only a few deviations. This would mean that the accuracy of the prediction can be almost independent of the prediction parameters and still give results similar to the above. This assumption can be refuted by repeating the above analysis for a smaller sphere for the multipole coefficients. The area in which the energies are calculated remains the same. If there is a different distribution of deviations, the prediction accuracy is not independent of the parameters and optimal ones can be found.

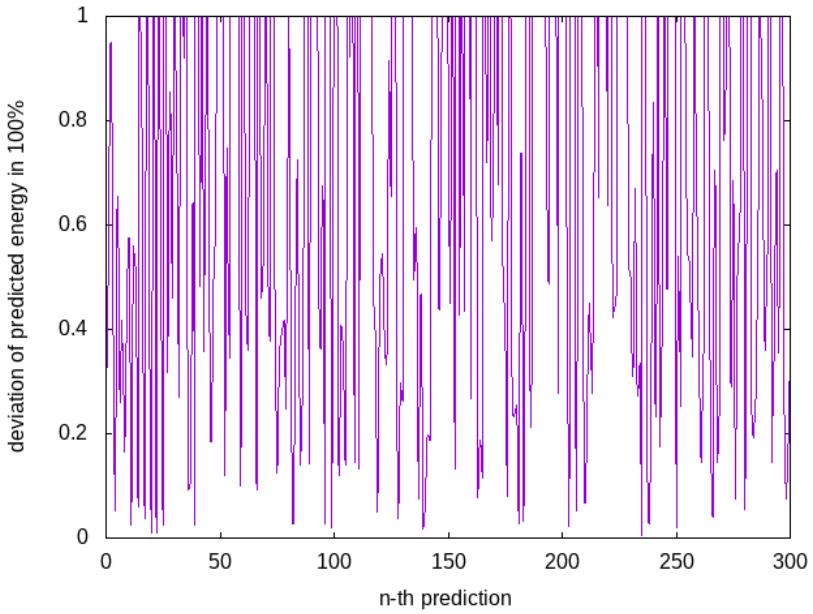

Figure 18: Percentual deviation of the predicted energy based on the multipole coefficients compared to the actual energy of the configuration after the relaxation step. To calculate the multipole coefficients a spherical configuration with radius $r = 4$ was extracted from the system. Compared to the previous figure there are much more outliers. The median of the percentual deviation lies at 65%, which is significantly higher.

In figure 18 the corresponding percentual deviation for a sphere radius of $r = 4$Å is plotted. It can be seen that there are many more outliers compared to the previous example. The overall distribution of the deviations between 0% and 100% seems to have slightly shifted upwards. This observation is supported by the fact that the median deviation of all 300 trial moves increased to 65%.
As the radius of the extracted sphere to compute the multipole coefficients has a measurable impact on the overall prediction accuracy, the experiment is repeated for various radii. For better visibility the measured data is represented as boxplot as can be seen in figure 19. The purple and blue box are the ones discussed beforehand. In addition to that the radii $6, 10$ and 12 were also tested. The worst result is achieved for the small extraction radii. The reason for this is assumed to be that there are not enough particles and thus information in the smaller sphere to be able to sufficiently capture the overall configuration. Since the lattice constant of a regular iron lattice is about 2.8Å, only the two neighboring cells are considered with an extraction radius of $r = 4$.
In contrast, the best result could be achieved for a radius of $r = 8$. The median deviation is 17% and the lower and upper quartiles are at approximately 9% and 41% respectively. If the outliers are not taken into account, the maximum deviation is 82%.

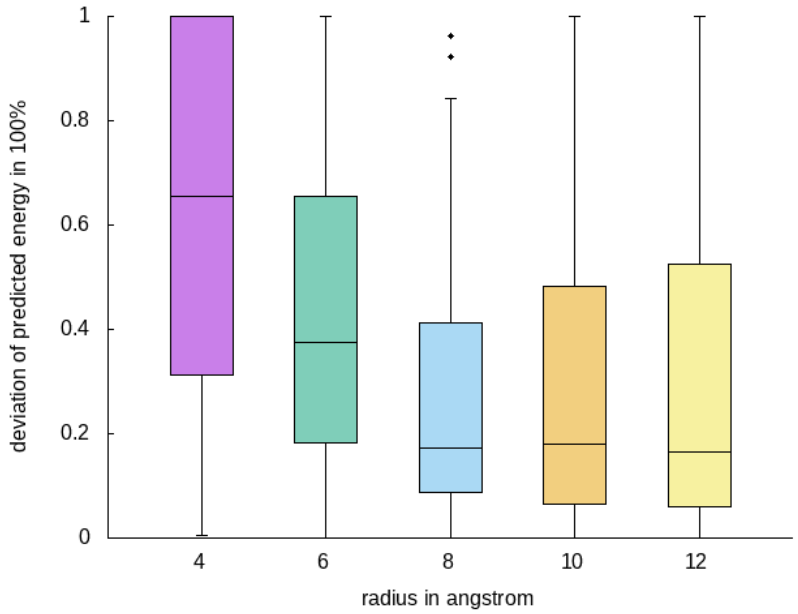

Figure 19: The plot shows the distribution of the percentual deviation of the predicted energy to the actual one. The distributions are represented in the form of boxplots for different radii of the extracted sphere, for which the multipole coefficients are calculated for. On average the most precise predictions are made for a radius of 8Å.

If the extraction radius is increased further the results stay similar except the upper quartile as well as the maximum deviation are increasing again. The medians for radii of 10 and 12 are roughly the same while the lower quartile even decreased a bit to about 7%. This behavior which shows for larger extraction sphere radii might be closely connected to the increasing number of particles. If more particles are used to calculate the multipole coefficients, the absolute value of the individual coefficients increases, as more individual contributions are added up. While maintaining the relative deviation of two coefficients so that they are still considered similar, the absolute deviation that the coefficients may have also increases. Another factor for the reduced accuracy is the blurring of the higher order coefficients. As these react very sensitively to particle displacements and in particular to additional particles, a higher number of particles means that these coefficients are more difficult to assign. Another point is that with a larger radius, more particles are represented by only 15 coefficients. For a more accurate description, more multipole coefficients might be necessary. In addition to an increased computational effort to determine the coefficients, this would also mean more storage space and a longer search time.
Due to these facts, the number of coefficients is not increased and the 'optimal' radius of $r = 8$ for 15 coefficients is retained, as satisfying results could already be achieved using this radius.

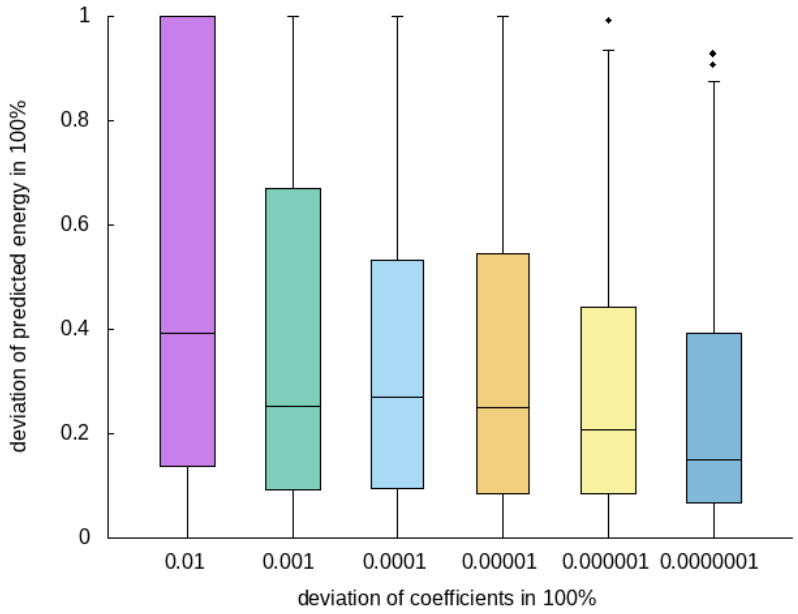

Figure 20: The plot shows the distribution of the percentual deviation of the predicted energy to the actual one. The distributions are represented in the form of boxplots for different similarity thresholds. For two coefficients to count as similar during the database search their difference has to lie within this relative threshold. Thereby the biggest threshold is 1% percentual deviation and the lowest 0.00001%.

Another aspect to consider is the variation of the allowed percentual deviation between two coefficients so that they are still considered similar. In figure 20 the deviation of the predicted energy for different allowed coefficient errors in plotted. The highest relative deviation was chosen with 1%. The deviation was reduced down to 0.00001% in multiples of 10. If can be seen that the lower the relative deviation the more accurate the energy prediction gets. Thereby the median does not change as much. But especially the upper quartile lowers down from above 100% to approximately 40%. Also the lower quartile stays at around 10% for all tested deviations.
If the allowed deviation is chosen too low, another problem arises which is connected to the way similar samples are found inside the database. The prediction algorithm just finds the closest match starting from the first coefficient onward (see Section 3.2.1). This means that if only the first two coefficients can be matched, all coefficients that have matching first two coefficients are considered for the energy prediction. This drastically increases the number of found coefficients in the database and therefore the multitude of predicted energies as well as search times. Furthermore the prediction quality also suffers from this effect in the way that lower energies cannot be predicted as well anymore. Configurations that lead to a lower system energy originate from a bigger phase space and therefore demand more coefficients to be described well. While their coefficients

might also be found in the case where just two matching coefficients are considered for the database search, the chance that the correct coefficient is selected is very low as the sample with the higher sample weight and sample credits is preferred. Because the occurrence of lower energies for a trial move is very rare, their sample weights and credits are low as well if there is any. Therefore, a poorer prediction accuracy should be accepted for an improved prediction quality of low system energies.

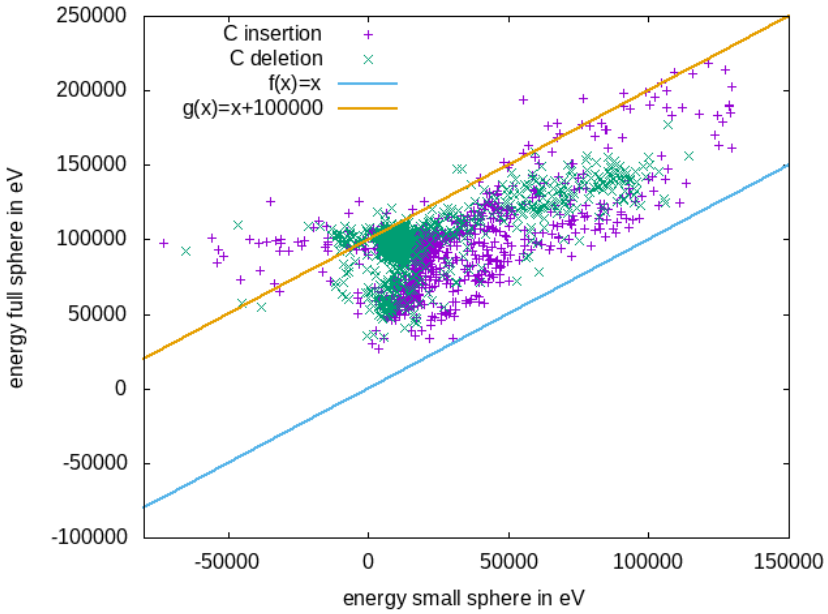

Figure 21: The plot shows the energy, if its calculated for a sphere radius of $r = 8$ compared to the energy of the whole trial move configuration with a radius of about $r = 20$. The points are thereby divided into two groups, one for the insertion and one for the deletion of carbon. Besides the fact that the energies for the full configuration are higher there seems to be a correlation between those two energies. While the energies for the full configuration are always positive it should be noted that some of the energies calculated in the small sphere are negative due to single atoms interacting with others outside the extracted sphere.

An obvious aspect that plays a role in investigating the accuracy of the energy prediction is the determination of the energy itself. In the previous studies, the resulting energy of a trial move was determined by summing the energy of the particles in the sphere with radius $r = 8$ extracted for the multipole coefficients. Also for the measurements in figure 19 this radius was retained. Since for the final energy evaluation of a trial move not only the energies within the smaller sphere with radius $r = 8$ but of the entire configuration with a radius of about $r = 20$ are considered, the extent to which these

two energies correlate with each other is examined below. Furthermore, the extent to which it is advantageous to use the energy of the entire configuration as a prediction is also considered. For the following research the simulation was started with a pretrained database with around $10,000$ samples. This was done to assure that the predictions already are of acceptable quality so that correlations are easier to recognize.
In figure 21 the energies calculated for the smaller sphere with radius $r = 8$ are plotted in comparison to the energy of the entire trial move configuration. The energies were determined after the relaxation step. It can be seen that the energies for the overall configuration are always positive whereas the others can also assume negative values. This has to do with the fact that before the relaxation, all particles located within the extracted sphere are determined in order to have an initial energy for the decision criterion. In the energy evaluation following the relaxation, the identical particles are considered in order to ensure comparability. If this were not the case, it is possible that the energy is only lower because a particle has run out of the extracted sphere during the relaxation. Considering the same particles both before and after relaxation also has similar disadvantages, such as negative energies, but this approach has proven to be more reliable in practice.

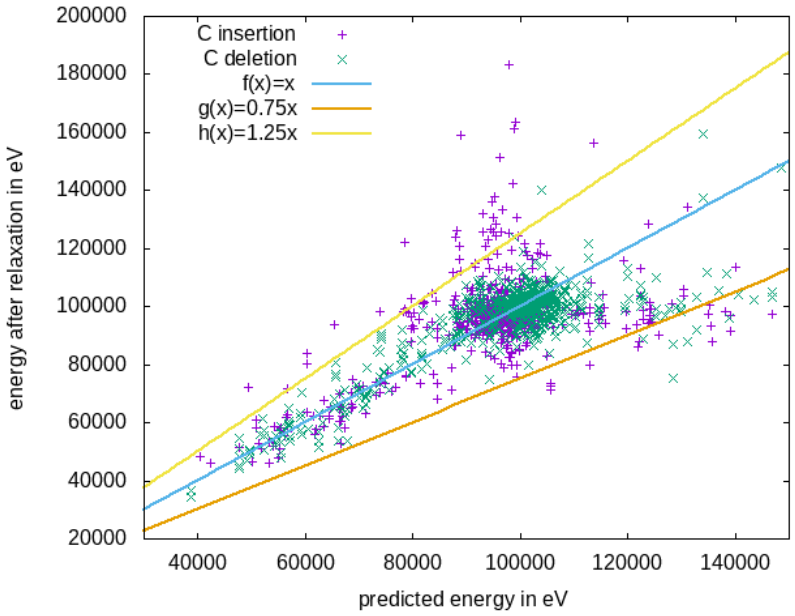

Figure 22: The plot shows the predicted energy in contrast to the actual energy of the configuration after the relaxation. As the energy for the prediction is computed over the whole configuration all points of the graph should lie on the blue straight if there is pinpoint prediction accuracy. Each prediction is divided into two sub predictions, one for the insertion and one for the deletion of the carbon atom each happening in the same trial move configuration but inside different spheres.

Furthermore there seems to be a weak correlation between the two energies. Although the data points are highly scattered, the deviation of most of them can be limited to an interval of about $70,000$eV around the function $h(x) = x + 55,000$. In addition, the majority of the data points cluster in a radius of $10,000$eV around the point $(10,000; 100,000)$. To find out if the prediction accuracy or even quality can be improved by increasing the radius of the sphere that the energy is computed from, the energy of the entire configuration was taken as the predicting energy. The predicted energy was then compared with the actual energy of the configuration as plotted in figure 22.
In most cases, the predicted energy almost matches the actual energy. The deviations are usually only a little over $10,000$eV or for higher energy amounts about 25%, which means a significant increase in accuracy. As in the previous example, the data points accumulate at around $100,000$eV. The only striking point is that the predictions for the insertion of carbon are sometimes much lower than the actual system energy. In other cases, the predictions for the insertion but also the removal of carbon from the configuration are too high. But even then, the relative deviation does not exceed a value of about 60%.

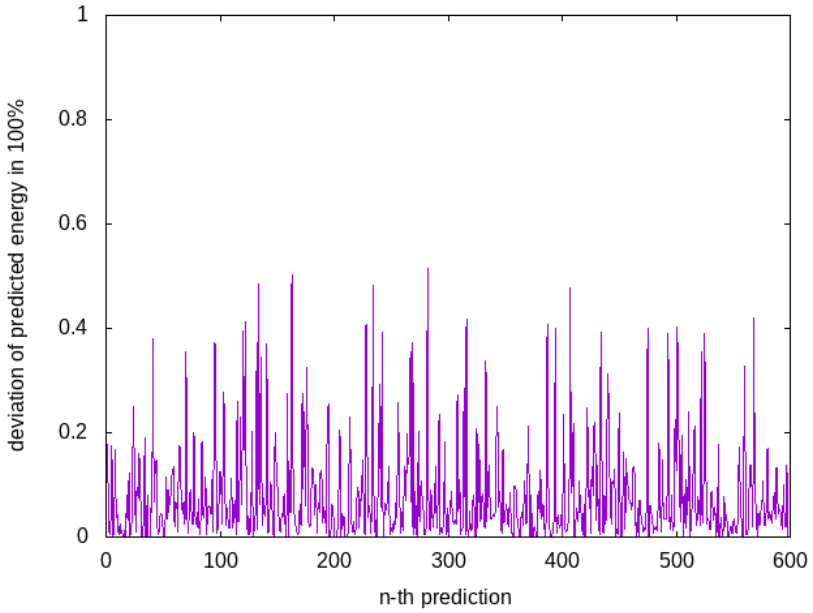

Figure 23: Using the energy of the whole trial move configuration for prediction significantly increases the prediction accuracy. The figure shows the relative deviations of the predictions from the actual energy of the configuration. The allowed percentual error at which two coefficients count as similar lies at 1%, which is quiet high compared to the previous findings in figure 20 for the smaller energy computation sphere.

This drastic increase in the accuracy of the prediction can also be seen in the percentual deviation. In figure 23 this is plotted for 600 predictions. It should be noted that the simulation was not started with an empty database, but that approximately $15,000$ samples were already available. The average deviation is 8% and the median is 4%. The maximum deviation is approximately 52%. The upper and lower quartiles are 10% and 2% respectively. The maximum allowed error between two coefficients to count as similar was set to 1%, which is quiet high compared to the previous suggested values (see figure 20). But as already stated this value should be chosen as low as necessary to achieve accurate predictions but as high as possible to still be able to identify all configurations in higher detail using more of the available coefficients.

4.2 Improving Performance

After analyzing the prediction method and its reliability in detail, the following section focuses on increasing the performance of the MC++ code. Whether it is by parallelization of the relaxation step, comparing the speedup introduced by the energy prediction or a decrease in search time by reorganizing the samples in the database.

4.2.1 Relaxation Parallelization

Several processors can be used to speed up the relaxation process in the event of a successful trial move. Since such a procedure also entails a parallel overhead, the extent to which parallelization of the relaxation is worthwhile is examined below. Since under usual conditions the a relaxation step for a radius of about $r = 20$ takes about 8 seconds, the radius of the trial move configurations extracted from the system was increased to $r = 30$ in order to be able to recognize the scaling behavior more easily. The configurations therefore consist around $50,000$ particles instead of the usual $15,000$. In figure 24 the relaxation times for different numbers of processors are plotted. For $1, 2, 4$ and 8 processors, four runs with 20 relaxations each were performed. The relaxation times were averaged.
A single processor takes between 18 and 22 seconds for a relaxation. The times vary widely, as the time required for relaxation depends heavily on the configuration to be processed. This time reduces to 15 to 18 seconds if two processors run in parallel. Using four processors this can be cut down further to just 12 to 13 second. For even higher numbers of processors there is no performance gain noticeable. This implies that the overhead of the relaxation in combination with the serial part lies at around 10 to 11 seconds. The observed relaxation step includes receiving the particles, generating all particles in an instance of LAMMPS, the relaxation step itself as well as the back-transformation to MC++ atoms and sending them back to the master process. All steps except the relaxation run in serial because due to the lack of shared memory all processors have

to handle the receiving and generating of he particles. As these steps only make up for about 400ms of the time they are negligible. The other part of the time is consumed for the relaxation itself. As the relaxation does not consist of one singular minimization but of a combination of three separate steps, the large serial part might originate from this. If virtual atoms could be handled in LAMMPS the workaround as described in Section 3.1.1 would not be necessary anymore, which could speed up the relaxation step. So apart from a seemingly optimal speedup of the relaxation itself, due to the large serial part the time taken to run the whole relaxation process can just be halved. For further improvement it would be optimal to make use of shared memory so that there is less traffic for sending and receiving the configurations as well as generating the particles in LAMMPS.

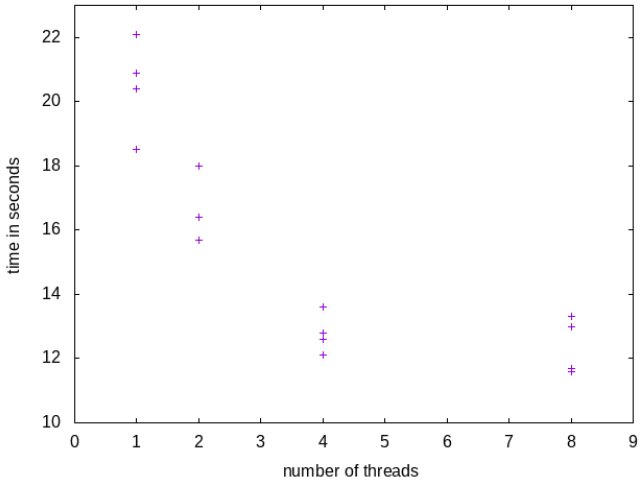

Figure 24: The plot shows the average time needed to relax one trial move. Plotted are 16 distinct runs for different numbers of threads with 20 relaxations each. The tracked time includes receiving, creating and submitting the particle-configuration. The such induced overhead can be estimated to be around 10 seconds. By introducing up to 4 threads the pure relaxation time can be reduced from 12 to about 3 seconds. Further increasing the number of threads does not provide any significant improvement.

4.2.2 Prediction Speedup

Assuming that it was known before running the simulation which trial moves would lead to success, all other relaxations could be omitted, which would drastically reduce the execution time. The algorithm used to predict the energy is an attempt to be able to predict the result of a trial move, at least to a certain degree. This section covers the extent to which the use of the energy prediction algorithm can speed up the execution time of a simulation. The time saving is achieved by not performing

selected relaxations, as the result of the trial move is predicted to be unsuccessful with a high probability. Since with the parameters currently used, including a relaxation radius of around $r = 20$Å, a single relaxation takes an average of about 8 seconds on a single processor, which is the most time-consuming part of processing a trial move. Furthermore, due to the characteristics of the considered iron-carbon system, only very few trial moves overall lead to success and minimize the system energy, which means that a large part of them can be omitted.

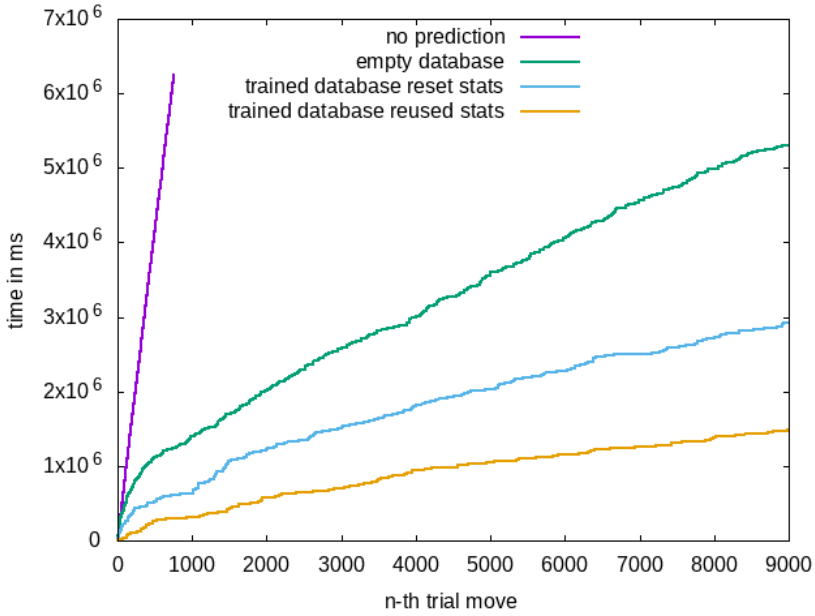

Figure 25: In the plot the cumulative times for N trial moves are plotted. Thereby four variants are compared. The first one is the simulation without using the energy prediction algorithm that executes all relaxations. The other variants apply energy prediction while one starts with an empty database and the others with one already containing $10,000$ samples. The two remaining variants can be differentiated by one reusing the probability statistics from a previous simulation while the other resets those statistics. As the first variant takes significantly longer it was just executed for the first 750 trial moves

In figure 25 the execution times required to process N trial moves is shown. A distinction is made between four variants. The first variant is used solely for comparison with the other variants, as it does not use the energy prediction method. Therefore, all relaxations are carried out, which means a runtime of approximately 8.25 seconds per trial move. The total runtime for 750 trial moves is therefore around 6200 seconds and for 9000 trial moves it would be around 74400 seconds. The second variant applies the prediction algorithm, but the simulation starts with an empty database, which means that the algorithm has to be trained starting from zero. The prediction chance therefore starts

with a probability of 0% and increases depending on the number of samples in the database. Since the majority of all trial move configurations are almost identical, the prediction chance initially increases sharply and flattens out as the simulation progresses. The shape of this curve is shown in figure 26 for further illustration. The number of samples in the database amounts to just under 600 after the completion of 9000 trial moves as only this number of relaxations was performed.

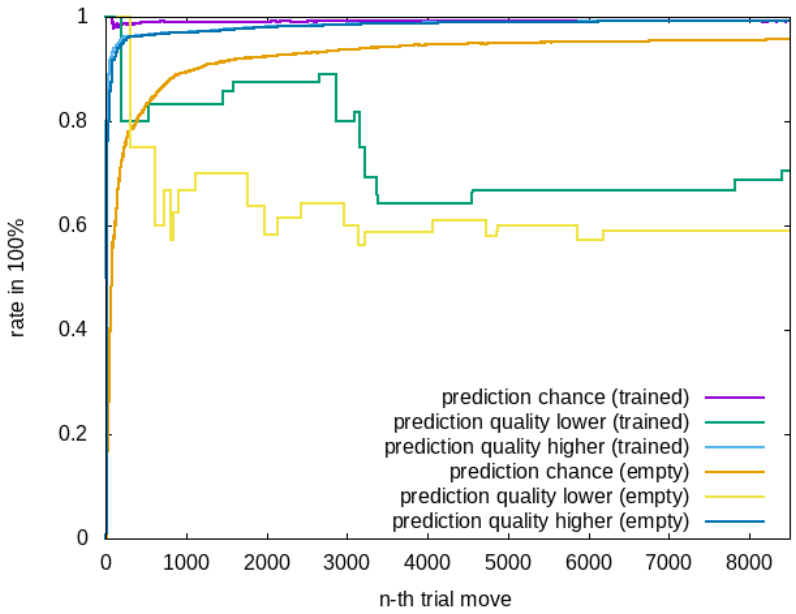

Figure 26: The plot shows the development of the prediction chance and quality for simulations that start with an empty database as well as with one containing about $10,000$ samples. All previously mentioned techniques like fast learning, an 'optimal' extraction radius as well as the improved accuracy for the energy prediction have been applied. Additionally the simulation is carried out over a higher number of trial moves than before.

The reason for this low number of relaxations is, in addition to the increasing prediction chance, the equally rapidly increasing prediction quality for the configurations that lead to a positive trend of system energy. This already amounts to a value of over 95% after around 1000 trial moves. Therefore, when a higher energy is predicted, a cross-check is only initiated in around 5% of cases. In the course of the simulation, a total of 150 of such cross-checks were carried out with the result that each of those predictions was correct. In comparison, relaxations were performed far more frequently due to the fact that no matching sample could be found in the database. This was the case around 450 times, of which on average 4 trial moves were accepted. For all other 8550 trial moves a prediction could be made. A lower system energy was predicted only about 17 times, whereby this prediction was incorrect in about 40% of the cases (see figure 26).

In conclusion on average 14 trial moves of a total of 9000 were found and accepted using this variant, which took 6400 seconds.
The third variant differs from the second in that the database is pre-trained with around $10,000$ samples. However, all runtime statistics, including the prediction chance, are initialized with 0%, as would also be the case with an empty database. Since the configuration space is already well covered by the database, the prediction probability increases almost immediately to over 98%. The prediction quality for higher energy tendencies also rises very quickly to over 95%, whereby this is similar to that of the second variant (see figure 26).
While a comparable number of cross-checks are undertaken, the number of configurations for which no prediction could be made has fallen significantly to a value of around 80. The total number of relaxations is therefore only around 250 of which 20 is due to prediction of a lower energy trend. The prediction quality for these cases is slightly higher than for the second variant at around 70%, which can be attributed to the advanced training of the database. However, the main speedup lies in the reduced number of relaxations, which makes this variant perform 9000 trial moves faster than performing 750 of them without energy prediction (see figure 25). Despite this low relaxation rate, around 14 trial moves were found and accepted during a total execution time of just 2700 seconds.
The fourth variant corresponds to the third with the exception that the prediction statistics are taken from several previous program runs. This means that, among other things, both the prediction chance and the quality are at a high level right from the start. The prediction chance is 99.1% and the prediction quality for higher and lower energy trends is 99.9% and 60% respectively. For this reason, only about 11 cross-checks are performed and only 75 times no prediction was made. This is also reflected in the number of relaxations, which is about 110 The remaining high number of predictions for a lower energy trend is striking. This amounts to around 23 compared to 17 and 20 for the other variants. On average, 14 of these were accepted. The total execution time of the simulation for this variant with 9000 trial moves is around 1480 seconds. With an average relaxation time of 9 seconds, 445 seconds are spent on reading and writing the simulation configuration of about 500Mb, initializing the database and other overhead in addition to communication. Nevertheless the speedup of this fourth variant compared to the one without energy prediction lies at approximately 50. Even the slower variants starting from an empty database shows a speedup of about 11.
The remaining question is whether using the energy prediction method overlooks trial moves that would have been accepted under normal circumstances. In the simulations without energy prediction, an average of 6 trial moves out of a total of 750 were accepted. Using the second option, starting from an empty database, around 5.5 trial moves were accepted out of the first 750 whereby it should be noted that with 20 times this variant was not tested as often as the fourth variant. With the third variant about the same but slightly better results could be achieved. For 750 trial moves the fourth variant took about 900 seconds to perform 100 relaxations on average. With 120 times it was repeated most often out of all variants. During each run about 6 trial moves were accepted which suggests that not a significant number of trial moves is omitted due to

the energy prediction method.

4.2.3 Database sample distribution

The performance of individual database models was compared in Section 3.3.3. Thereby the multidimensional database performed by far the best. A fundamental assumption to ensure short search times was that all lists in the database are populated more or less equally, by the generated coefficients following a uniform distribution. But in the practical use case the coefficients are not uniformly distributed. Some figures already suggested that a large part of all samples are quiet similar coefficient- and energy-wise and in comparison just a few are different. Therefore this section focuses on the actual distribution of the samples in the database and the problems that arise from this circumstance. Additionally the advantages of using different parameters for the number of dimensions and bins is discussed.

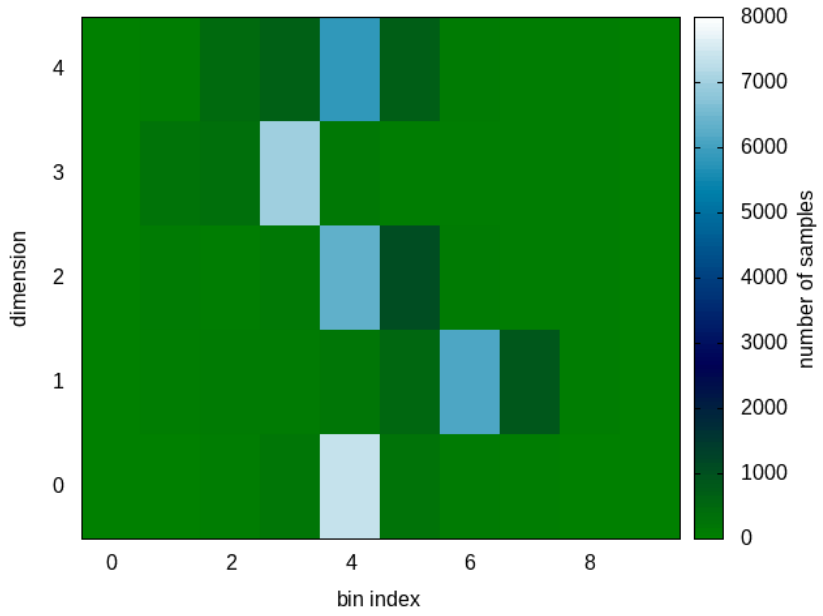

Figure 27: The plot shows the distribution of samples in a 5-dimensional database with 10 bins in each dimension. The heat map can be interpreted as how many samples got assigned the specified bin index in the respective dimension. It can be seen that the sample distribution is not even and a large part of the samples got assigned the same bin indices due to being similar.

The database used for the analysis contains about 8,000 entries. To illustrate the distribution of the samples in the database, heat maps were generated as shown in figure 27. The heat maps were generated by counting the occurrences of bins indices in each dimension for all entries in the database. What is immediately apparent is the very uneven distribution of the entries. In figure 27 about 90% of all samples are assigned to a single bin per dimension. This not only means that coefficients of the same order of magnitude are predominantly searched for in the database, but that the corresponding lists in which the search is carried out contain a very large number of entries. This results in linear scaling behavior, which should initially be avoided or at least postponed by using the multidimensional database. On average a single search in the database takes 1000 microseconds compared to about 1500 microseconds for the linear database model. Even an increase in the number of bins per dimension does not lead to a significant improvement in the sample distribution, which can be seen in figure 28. Although there is a split in the second dimension (index 1), the individual bin indices are still assigned around 2000 and 5000 times.

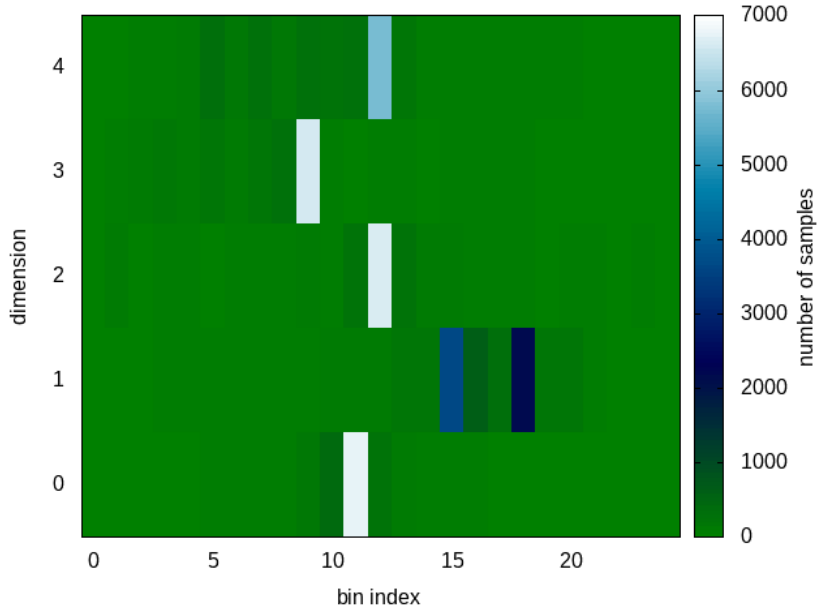

Figure 28: The plot shows the distribution of samples in a 5-dimensional database with 25 bins in each dimension. The heat map can be interpreted as how many samples got assigned the specified bin index in the respective dimension. Through the increase in the number of bins the samples get distributed over a slightly larger range of bins.

In addition to increasing the number of bins per dimension, the number of dimensions was also increased in order to check whether the distribution problem could be equalized.

For this purpose, a database with 10 dimensions and 5 bins per dimension was created and a heat map was also generated (see figure 29). However, this shows a similar picture to the databases before. Although this database might be better suited for changes in the obtained coefficients due to the higher number of edge bins (see 3.3.3) it has no advantage regarding the sample distribution.

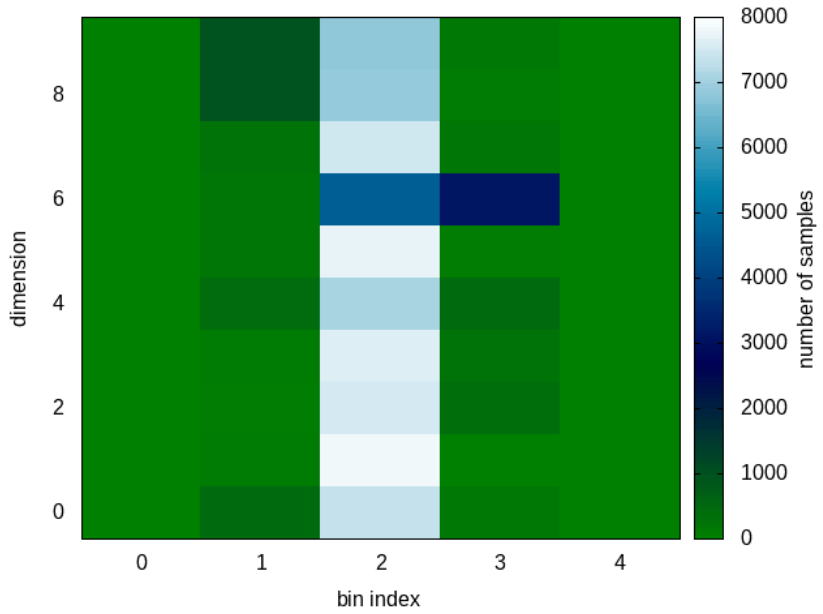

Figure 29: The plot shows the distribution of samples in a 10-dimensional database with 5 bins in each dimension. The heat map can be interpreted as how many samples got assigned the specified bin index in the respective dimension. The samples are quite evenly distributed, but over a smaller number of lists as the number of edge bins is much higher than in the other database variants.

To solve this problem, the concept of the multidimensional database was extended by one aspect. When using equidistant bins, a uniform distribution of the coefficients is necessary in order to have the entries in the database evenly distributed. If this is not guaranteed, the number of entries per list can be equalized by non-equidistant binning. The determination of the bin index then additionally requires the iteration of a list in the length of the number of bins. Furthermore, the database search also searches all neighboring lists of the specified list, since the non-equidistant binning does not guarantee a minimum bin width. The similarity search also works without this adjustment, but in practice it has proven to be beneficial for the hit rate for lower energy tendencies.
In figure 30 the result of these changes to the database model can be seen. On loading the database into the memory the bin sizes were chosen appropriately to that all lists have the same number of entries in them at the start of the simulation. This leads

to the search times being significantly faster with only 700 microseconds compared to around 3000 microseconds previously. In contrast to this, a linear database model would have taken about $15,000$ microseconds to find a similar sample in a database containing $40,000$ samples.

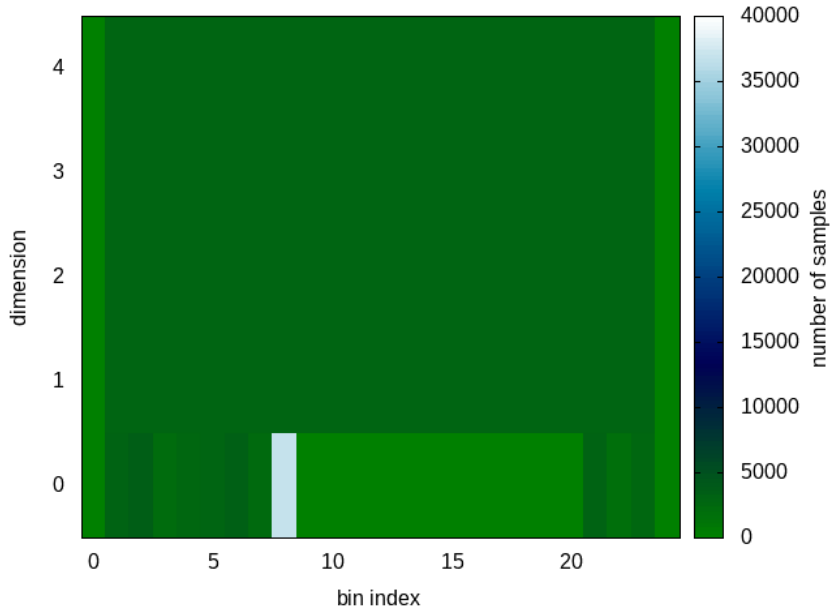

Figure 30: The plot shows the distribution of samples in a 5-dimensional database with 25 bins of variable size in each dimension. The heat map can be interpreted as how many samples got assigned the specified bin index in the respective dimension. Applying non uniform bin sizes leads to the samples in the database being nearly evenly distributed apart from the first coefficient.

4.3 Parallel Replica Method

This chapter takes a closer look at the performance of the implemented parallel replica method. In particular, the extent to which it can accelerate the runtime of the MDMC simulation is examined. A direct comparison is made with the existing master-worker approach. In addition to possible accelerating effects, the development of the potential energy of the system is also examined if it undergoes any change due to variation of the applied method. In addition, the extent to which the choice of a longer replica comparison interval has a positive influence on the simulation and what trade offs had to be accepted, is considered.

4.3.1 Runtime comparison

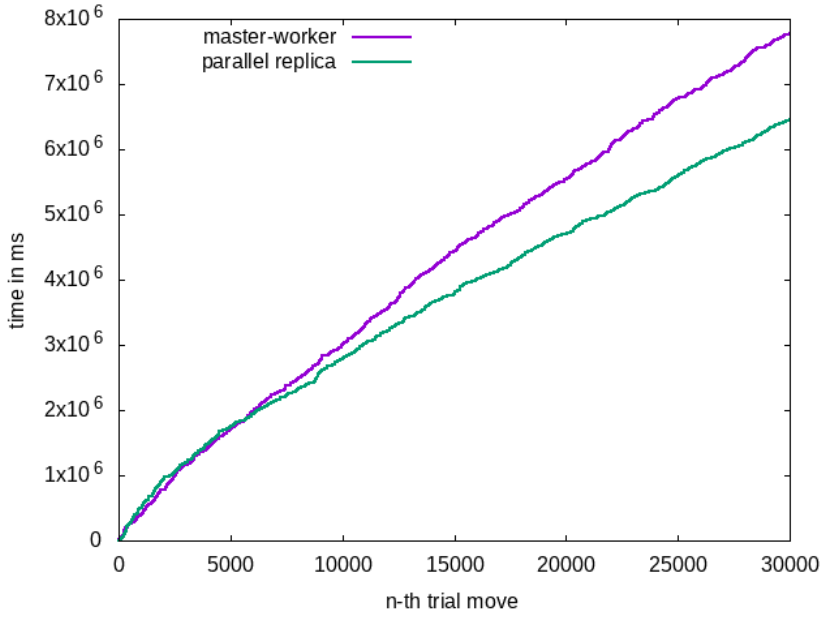

Figure 31: The graph shows the cumulative runtime required to execute N trial moves. In particular, the master-worker approach for $90,000$ trial moves distributed on one master and 3 workers and the parallel replica method for $30,000$ trial moves on each of 3 processors are compared. As the processors run in parallel, the values on the x-axis for the master-worker scheme were scaled to $30,000$ from originally $90,000$. While both methods show a similar scaling behavior in the beginning, the higher the number of trial moves the faster the parallel replica method becomes in comparison.

In order to compare the runtime of the parallel replica method and the master-worker approach, several simulations were run for a larger number of trial moves to ensure easier comparability. In addition, the energy prediction algorithm was applied to speed up the process. Thereby the prediction statistics from previous simulations were taken over, similar as for the fourth variant in figure 25. The multidimensional database consists of approximately $60,000$ samples. Applying the master-worker approach, a simulation for $90,000$ trial moves was performed. Thereby a single master and three worker processes were used. This is compared with the parallel replica method, for which a simulation was carried out on three processes with $30,000$ trial moves each.

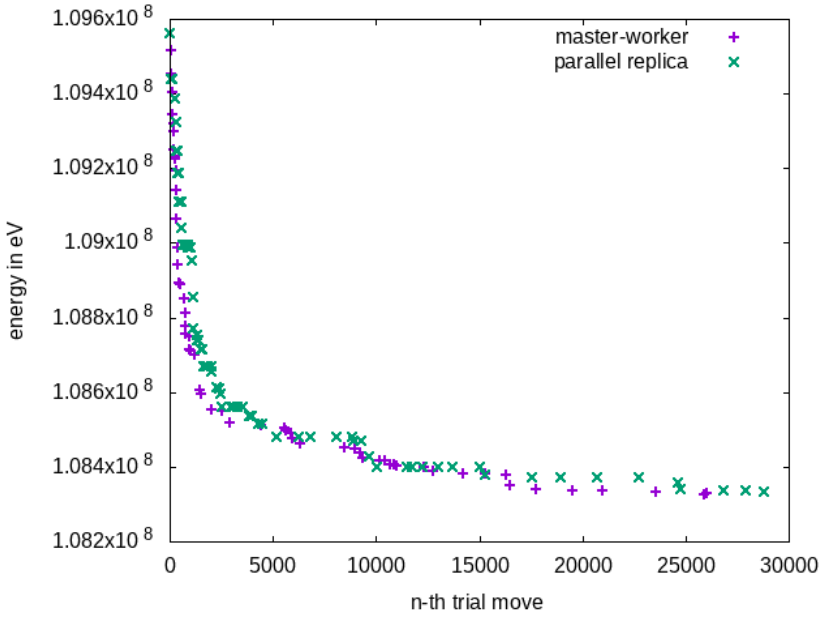

Figure 32: The graph shows the development of the total system energy for the master-worker approach in comparison to the parallel replica method for a total of $90,000$ trial moves. It can be seen that there is not much difference between the curves which supports the assumption that there is no direct disadvantage in using this new method.

In figure 31 the cumulative runtime for those two variants is plotted. It can be seen that while the parallel replica method is slightly slower in the beginning, it performs the simulation in about 6500 seconds and thus around 1000 seconds faster than the master-worker approach. The overhead of the parallel replica approach is closely connected to the rate at which accepted trial moves are found. The less trial moves are accepted the less replica compares have to be performed, which take about 4 seconds on average. In the current example 89 replica compares were performed which took about 356 seconds. More than half of those occurred during the first 7000 trial moves and the rest

during the 23000 remaining ones. This explains why this method took longer than the master-worker approach in the beginning. In comparison the master worker approach just accepted 57 trial moves while taking longer for the whole simulation. Nevertheless the resulting potential energy of the system developed similar for both method, which can be seen in figure 32.
Furthermore the master-worker approach scales less and less well for higher processor counts than 4. The decisive factor is the communication overhead for sending jobs to the workers. If the master is overloaded, individual workers have to wait for a corresponding message, which is detrimental to parallel efficiency. To measure this effect of high processor counts and set it in comparison to the parallel replica method the simulation was performed on the Jusuf [15] supercomputer for various processor counts. Overall the average relaxation time shortened to about 5 seconds as well as the database search time that decreased to 430 microseconds, which sped up the simulation in general. Using the master-worker scheme 6400 trial moves were performed on 64 processors as well as 9600 on 96 processors so in theory 100 trial moves are handled on each processor. The simulations finished in 54 and 73 second respectively whereby the initialization as well as the cleanup step are excluded.

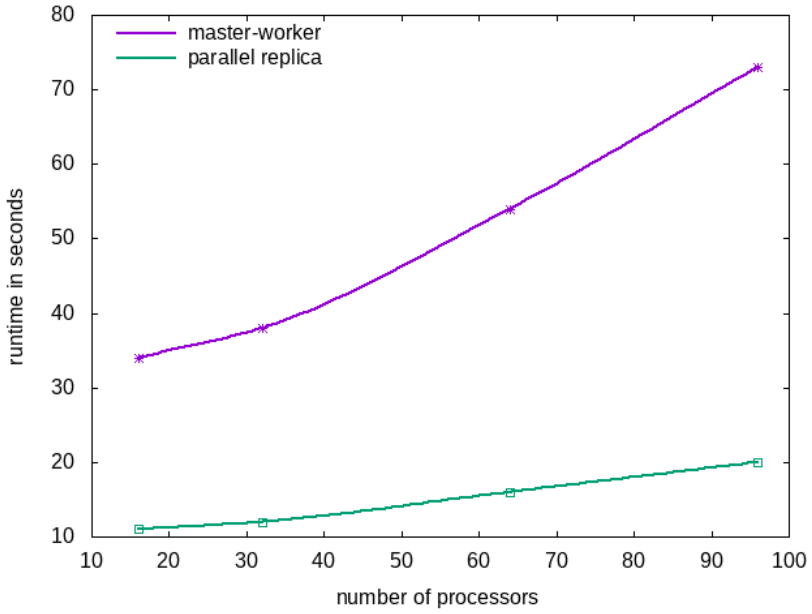

Figure 33: The graph shows the runtime of the simulation for the master-worker approach as well as the parallel replica method for 100 trial moves for each processor. The simulations were performed for processor counts of $16, 32, 64$ and 96. The communication overhead of the master-worker scheme leads to noticeably longer execution times as well as a worse scaling behavior for larger number of processors.

Already with 64 processors the effect of the master overload was noticeable as the average wait time of a processor per trial move was approximately 0.2 seconds. For 96 processors this number was even higher with around 0.5 seconds. Thereby on average 0.71 seconds were necessary to perform a single trial move. This means that per processor a total time of about 50 of 71 seconds was wasted on waiting for messages of the master process. This includes the wait times for receiving the trial move configuration, transmitting the result as well as the relaxed trial move configuration, if the trial move was successful.
The same number of trial moves was performed using the parallel replica method for 64 and 96 processors. Each of those was tasked to run 100 trial moves resulting in a total of 6400 and 9600 respectively. The first configuration finished the simulation in only 16 seconds and the second one in 20 seconds. In figure 33 the discussed configurations as well as ones for 16 and 32 processors are plotted. It can be clearly seen, that the parallel replica method is not only faster but also shows better weak-scaling behavior for high processor counts. It should be noted that due to the weak scaling approach additional trial moves are introduced, that might lead to additional replica compares and therefore an increased amount of time to run the simulation.

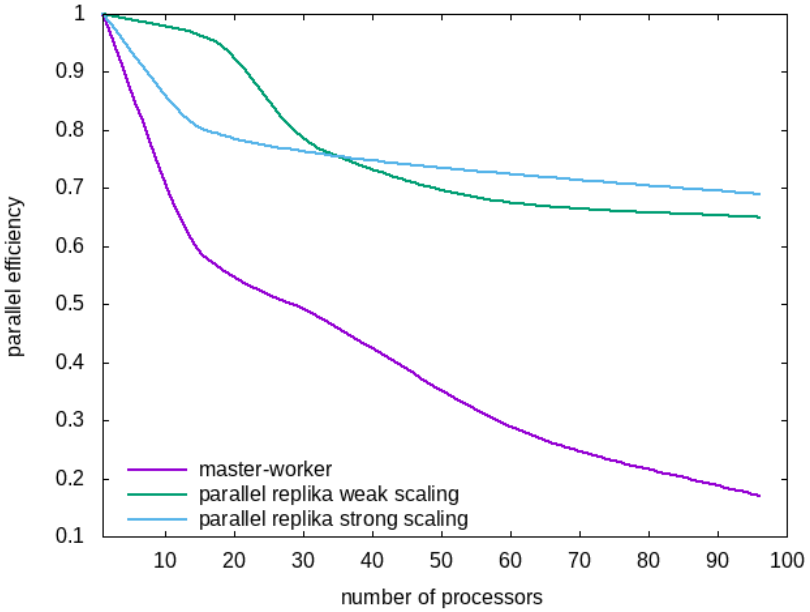

Figure 34: The graph shows the parallel efficiency of the master-worker scheme in comparison to the parallel replica method. While the efficiency of the master-worker approach quickly drops to under 50% the efficiency of the parallel replica method stays above 65% for processor counts up to 96.

Regarding the parallel efficiency the parallel replica method shows a similar advantageous behavior compared to the master-worker approach (see figure 34). While its efficiency quickly drops off to values far below 50% the efficiency of the parallel replica method stays at above 65% for high processor counts applying weak- as well as strong-scaling.

4.3.2 Memory consumption

In addition to the obvious advantages, the parallel replica method also has disadvantages that cannot be ignored. One of these is the significantly higher memory consumption, as shown in figure 35. This is due to the fact that each processor must have its own copy of the entire configuration, as the changes made are initially local and exclusive for the processor until the replica compare is performed.

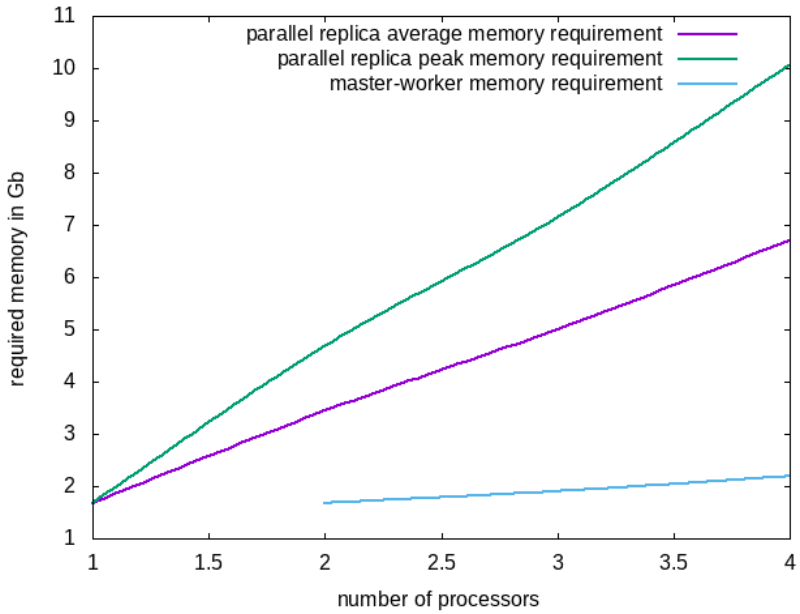

Figure 35: The plot shows the required memory to run the simulation. When using the parallel replica method each process is required to keep its own version of the entire configuration. In addition to that the replica compare itself is quiet memory intensive which leads to the even higher peak memory consumption. In contrast to this the master-worker approach only depends on one copy of the system and communicates the much smaller trial move configurations. As the minimal configuration consists of one master and one worker the graph begins at 2.

The available data was generated for an initial configuration with around $9,000,000$ atoms. At runtime, a single processor requires about 1.7Gb of memory to store this configuration. A similar value can be seen for the master-worker approach with one master and one worker. If the number of processors for the parallel replica method is increased, the average amount of memory required increases linearly. Another aspect that occurs for 2 processors and more is the additional memory requirement for the replica compare, which again results in a significant increase. Due to the memory structure of the atoms, which is not suitable for sending via MPI [19], it is necessary to create a separate buffer for sending and receiving the configuration. In theory, this would mean a doubling of the required memory. By making some adjustments to the way the data is transmitted, this value could be reduced to about 50% increased consumption. Since the generation of the configuration replica on the receiving processor requires the simultaneous presence of all particles, the increased memory requirement cannot fall below this threshold as long as the replicas are generated in parallel via MPI-broadcast and not serially.

4.3.3 Omitted Trial Moves

Another possibly limiting perk of the parallel replica method is the possible omission of accepted trial moves. This occurs especially when the acceptance rate of trial moves is still quite high at the beginning of the simulation. Due to the way in which the replica comparison is initiated (see Section 3.4), where all processors are finishing their currently processed trial move, it can happen that further trial moves are accepted between initialization and execution of the replica compare. Since only the system with the minimum energy is retained, this leads to the rejection of all other systems, including those that already contain accepted trial moves. In figure 36 the cumulative count of all discarded trial moves per processor in plotted. The simulation was run on 96 processors that each processed 1000 trial moves.
On average, each processor accepted 9 trial moves. As can also be seen from the graph most of these trial moves were discarded, so that at the end of the simulation only 20 replica compares were carried out, in each of which one or more trial moves were accepted. For some processors, even all of the 14 accepted trial moves were discarded because another system always had the lower energy. This poses a problem in that it reduces the algorithmic efficiency of the parallel replica method. The goal of minimizing the energy of the simulated system is also not achieved at the desired speed. Since trial moves are discarded simply because they belong to a system with non-minimal energy, possibilities to further minimize the energy in the replicated system are also discarded. This is because many trial moves do not conflict with each other and a combination of these would theoretically be possible.

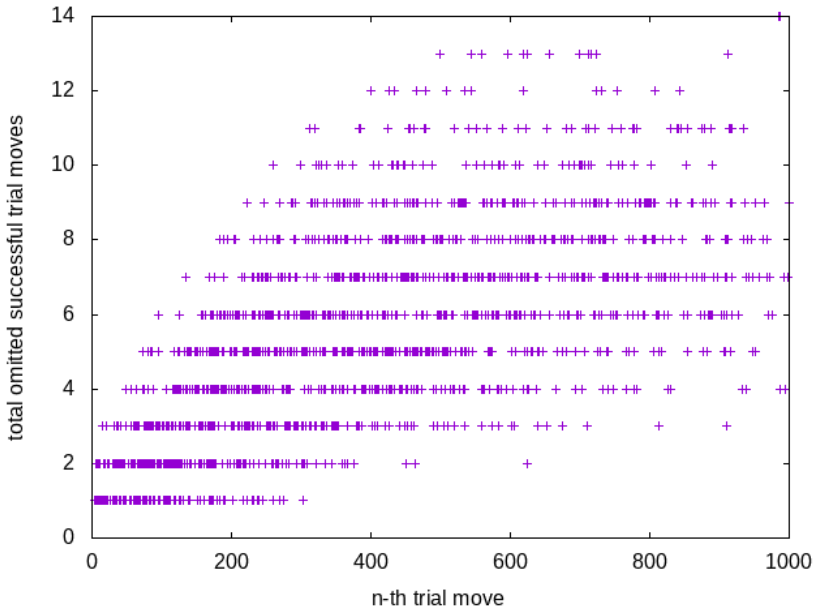

Figure 36: The graph shows the total number of omitted successful trial moves after performing N trial moves for each of the 96 processors for the parallel replica method. The omitting happens due to multiple processors finding an accepted trial move and initializing a replica compare within the same interval between two replica compares. As just the system with the lowest energy remains after the replica compare, all other systems, also those with accepted trial moves, are omitted. In the initial phase of the simulation the probability of multiple processors simultaneously accepting a trial move is high.

At the same time, the graph, which becomes shallower towards the upper right, also shows that the observed problem is not permanent, as the acceptance rate of the trial moves decreases over time. This reduces the probability that several processors will accept a trial move within the same replica compare time-frame and thus also reduces the number of discarded trial moves. This behavior can also be seen in figure 37 were the equivalent graph is shown for the trial moves 1000 to 2000 for each processor. Note that the cumulative sum starts over at 0 for this simulation.

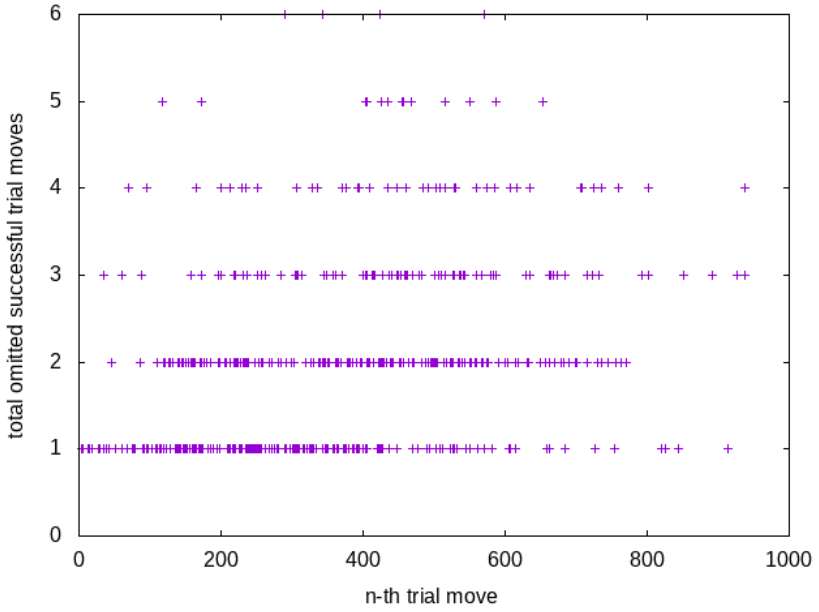

Figure 37: The graph shows the number of omitted successful trial moves after performing N trial moves for each of the 96 processsors for the parallel replica method. Compared to the initial phase of the simulation in figure 36 the probability of multiple processors simultaneously accepting a trial move is much lower for the 1000 trial moves afterwards which are plotted in this graph.

As can be seen from the example, the omission of trial moves due to collision is closely connected to both the acceptance probability and the number of processors. To get a better idea of the collision probability, the effective acceptance probability p_{ae} is introduced. This is calculated from the quotient of the number of replica compares and the total number of all trial moves.

$$p_{ae} = \frac{N_{replicaComapres}}{N_{trialMoves}} \tag{31}$$

For a single processor p_{ae} corresponds to the average acceptance probability for all processed trial moves. For higher processor counts, p_{ae} lies below the actual acceptance probability due to the trial move collisions. For the system under consideration, p_{ae} for the first 600 trial moves with one processor is approximately 1.72%. Increasing the number of trial moves to 1200 or 9600 reduces the average acceptance probability to 1.05% and 0.29%, respectively. For 16 processors with 600 trial moves each, p_{ae} is only 0.1%, which corresponds to approximately $\frac{0.01}{0.0029} = 34\%$ of the corresponding serial acceptance probability. When considering this development for even higher processor counts while maintaining the total number of trial moves, it can be seen that this proportion continues to decrease. For 32 processors, the proportion is around 19%, and for 64 and 96 processors, it is 8% and 5% respectively (see figure 38).

P	trial moves/P	p_{ae}	$\frac{p_{ae}}{0.0029}$	$1-(1-p_{ae})^P$
1	600	0.0172	-	0.0172
1	1200	0.0105	-	0.0105
1	9600	0.0029	1	0.0029
16	600	0.001	0.345	0.015
32	300	0.000555	0.191	0.017
64	150	0.000243	0.0838	0.015
96	100	0.000138	0.0476	0.013
96	600	0.000165	0.0569	0.015
96	1000	0.000167	0.0575	0.015

Figure 38

In addition to this development is can be seen that the effective acceptance probability seems to bottom out when increasing the number of trial moves per processor for 96 processors total. Another observation can be made if p_{ae} is assumed to be constant and as such the same for each processed trial move. In addition to that each trial move is assumed to be processed synchronous on each processor as if there would be an 'MPI_Barrier' for all processes after each trial move. The probability that at least one trial move is accepted on P processors during each iteration can then be calculated by $1-(1-p_{ae})^P$. While this value shows much variation for the serial execution for higher numbers of processors it nearly always lies around 1.5% which is comparable to the true acceptance probability at the beginning of the serial simulation (see figure 38).
A proportion of around 5% for the p_{ae} for 96 processors compared the the serial acceptance probability means that approximately 20 times as many trial moves can theoretically be accepted, provided that the individual trial move configurations of the respective replica compares do not overlap. This would allow merging all accepted trial moves within the same replica compare to a single new configuration which is then replicated.
The collision probability for two independent trial moves can approximately be determined based on the overlapped system volume. Assuming that the two spheres of a single trial move configuration do not overlap, the collision probability p_{col} for inserting the $(n+1)$-th sphere can be calculated using the following equation for $n=2$.

$$p_{col} = \frac{n \cdot \frac{4}{3}\pi(2r)^3}{V} \tag{32}$$

Thereby V is the Volume of the simulation box and r is the radius of the extracted trial move sphere. This radius is doubled as the additional sphere with identical radius has to be placed with a minimal distance to the existing ones to not overlap. In the considered

examples the extracted sphere has a radius of $r = 21,53846$. This leads to a total exclusion volume of $334828,59$ for each trial move sphere with the collision probabilities shown in the following figure 39. Note that this is the maximum collision probability as in practical spheres of the same trial move configuration may overlap as well as whole trial move configurations if they are performed in serial order and not parallel.

N spheres	abs. Volume	p_{col}	cumul. Prob. for no collision
0	0	0	1
1	334828,59	0.0124	0.9876
2	669657,19	0.0248	0.9631
3	1004485,78	0.0372	0.9272
4	1339314,38	0.0496	0.8813
5	1674142,97	0.062	0.8266
6	2008971,56	0.0744	0.7651
7	2343800,16	0.0868	0.6986
8	2678628,76	0.0992	0.629
9	3013457,35	0.1117	0.5591
10	3348285,95	0.124	0.4898

Figure 39

Given five independent trial move configurations whose spheres do not overlap, the probability that the 10 spheres are locally independent of each other is only about 49%. This means that even if there are about 20 times as many accepted trial moves as are currently adopted for the main configuration, the chance to be able to merge all of them into a singular configuration during the replica compare is quiet low as they have to be locally independent.
A possible solution for this problem would be to shorten the time between initiation an execution of the replica compare to be more or less instant while interrupting all running relaxations. A simpler approach to reduce the number of rejected trial moves is to slightly increase the replica compare interval as can be seen in figure 40. This means that a replica compare is not initiated after every accepted trial move, but only after every second one at a compare interval of 2. The basic problem, that several processors accept a trial move between the initiation and execution of a replica compare, remains. However, by increasing the compare interval, significantly fewer replica compares are performed, with only 9 compared to 20 previously. This leads to fewer discarded trial moves overall. This phenomenon only works for very low compare intervals of 2, as the probability of multiple trial moves being accepted in close succession is increasingly low. Thus very few processes reach this number of acceptances in short time which means

fewer processes with more than one accepted trial move to be discarded. The success of this method depends heavily on the timing of the relaxations and the reliability of the energy prediction.

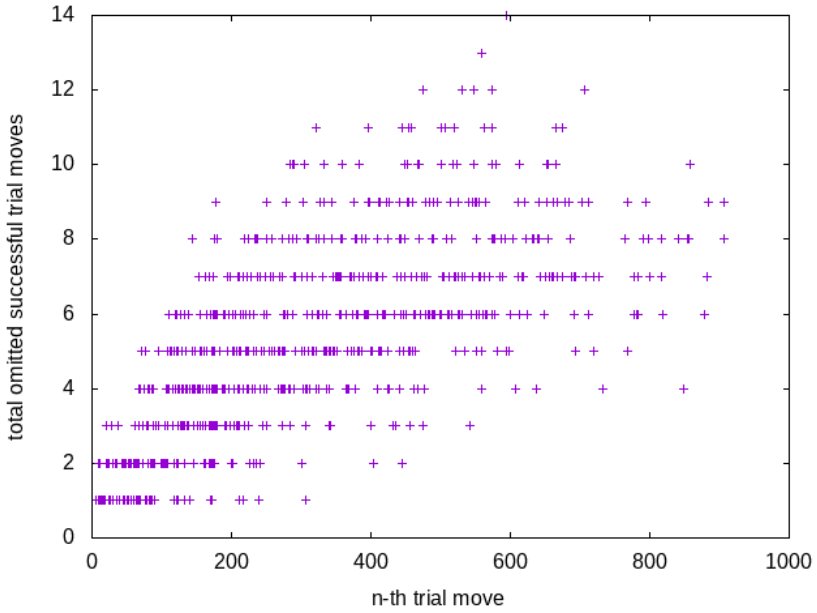

Figure 40: The graph shows the number of omitted successful trial moves after performing N trial moves for each of the 96 processors for the parallel replica method. The simulation starts from the same initial configuration as in figure 36 with the single difference that not after each accepted trial move a replica compare is initiated but the replica compare interval is increased to a minimum of two accepted trial moves per process.

If the compare interval is further increased to values beyond 2, the average of accepted trial moves for each processor between replica compares increases as well and with it the number of omitted trial moves. With a compare interval of just 2 this average lies closely above 1, while two trial move of progress are made at minimum each replica compare. For a compare interval of 3 this average will lean more towards 2 which results in 2 discarded trial moves for a progress of at least 3 on each replica compare.
Further using higher replica intervals also means much less replica compares in total. This tends to become a problem as there are only one or few processes that reach this number of acceptances thus there is less choice for a maximum descend in potential energy. This behavior can already be seen for a compare interval of 2 where on average a less minimal potential energy could be found for the same number of trial moves. However the deviation in the final energy is still small what can be seen in figure 41 where the development of the system energy is plotted for replica compare intervals of 1 and 2.

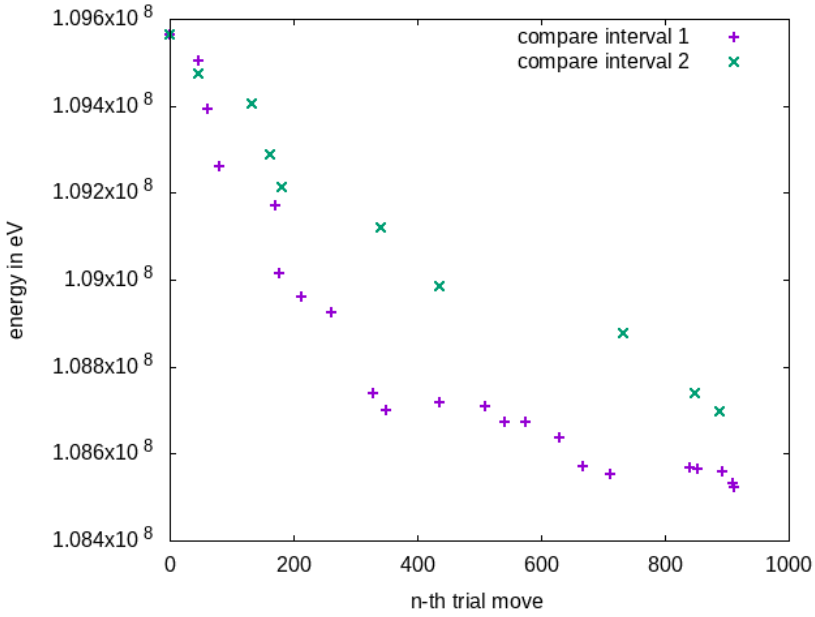

Figure 41: In the graph the development of the potential energy in the simulated system is plotted. Beginning with the starting configuration in total 96,000 trial moves distributed on 96 processes were performed. The fact that the curves show a similar decrease in energy shows, that increasing the comparison interval to 2 has not a great negative impact yet.

To completely avoid this problem of the parallel replica method, the master-worker scheme can be used to process the initial trial moves of a system. If the acceptance rate for the trial moves falls below a certain threshold, the parallel replica method can be used to speed up the search for accepted trial moves while the discarding rate of the trial moves stays low.

5 Conclusion

In the course of this work, the multipole coefficient based algorithm for the prediction of potential energy was successfully integrated into the existing MC++ code and applied. Furthermore, in addition to the IMD, a LAMMPS interface has been realized to perform the necessary relaxations. The concept of virtual atoms was implemented in a way that is compatible with LAMMPS, albeit with a workaround. The performance gain through the use of the prediction algorithm is made up of several components.
On the one hand, the idea is based on a similarity search within previously processed configurations in order to use this information to draw conclusions about the energy of the current configuration. A high-performance database structure for storing this data is therefore essential. In this work, three possible database models were examined in more detail, with the multidimensional database model proving to be the best. This enabled the search time for a data volume of $60,000$ entries in the database to be reduced from around $23,000$ to 3000 microseconds. After a further adaptation of this model, in which variable bin sizes were introduced, the search time was only 700 microseconds or 430 microseconds on the JUSUF supercomputer.
On the other hand, the energy prediction itself requires a high degree of reliability in order to guarantee correct predictions. In the course of analyzing some freely selectable parameters, the effects on the prediction quality could be determined and in some cases optimal parameters could be found. An extension of the zone within which the energy is predicted has yielded the greatest gain in accuracy. After a relatively short training phase of 800 trial moves, higher energy trends could already be predicted correctly with a probability of over 90%. However, the configuration space was not sufficiently sampled so that no prediction could be made after 800 trial moves in 20% of cases.
Furthermore, for the algorithm to be used successfully, it is also necessary to be able to recognize lower energy trends. This could only be achieved quite reliably with a database from around $10,000$ entries with 60% probability. To maximize the gained speedup through using this prediction method, an almost complete coverage of the configuration space is necessary, so that only a handful of configurations have to be relaxed, due to no prediction could be made. Since the configurations that lead to a higher energy tendency occur far more frequently, the hope is that these are almost completely sampled so that an increased proportion of unpredictable configurations will therefore lead to a lower energy tendency.
The maximum speedup that could be achieved by using the energy prediction is approximately 50 for a database with around $60,000$ entries. Thereby the relative deviation of the predicted energy to the actual one was on average about 8%. For this size of database the chance to find a similar configuration and thus be able to predict an energy was around 98.5%. Higher energy tendencies were correctly predicted to nearly 100% whereas lower ones only about 80% of the time. For comparing performance, the variant of the program without energy prediction was used, in which no relaxation is omitted and therefore the effort is significantly increased.
In the existing MC++ code, the parallel processing of trial moves was made possible

by implementing a master-worker scheme. This type of implementation enabled a significant speedup with a minimal increase in memory requirements. The communication overhead introduced by this approach was still negligible, as one relaxation was executed for each trial move, which accounted for the majority of the runtime. The introduction of the energy prediction algorithm in this work changed this ratio of communication overhead to computation time considerably. A possible approach that the master process takes over the energy prediction completely and only the relaxations are executed on the workers was discarded, as this limits the speed at which new configurations can be processed.
For this reason, a modification of the parallel replica method was implemented. This can be used to speed up the search for accepted trial moves, especially in advanced simulations with a lower acceptance rate, as all processors involved search simultaneously applying the prediction algorithm. Especially for high processor numbers of 96 the runtime of the simulation could be further reduced without a loss in quality of the result. However, this method also poses some problems, such as the greatly increased memory requirements, which needs to be addressed. To summarize, the parallel replica method is a valid and necessary replacement for the existing master-worker scheme as it shows much less limitations considering communication overhead for highly parallel simulations.

6 Outlook

Not all aspects could be examined in detail and all problems encountered could be fully resolved in the course of this work. This section therefore takes a closer look at some of these points and provides an outlook on possible future developments.
The current implementation of the multidimensional database was realized with the $C++$ data structure 'std::vector'. A problem which is particularly evident for higher dimensions and bin numbers is the high rate of pre-initialized memory. Since the vector data structure requires that all entries are continuously stored in memory, empty lists must also be initialized in the database and corresponding memory space must be reserved. A further problem arises if a list requires more memory than intended and all subsequent lists in the memory have to be relocated in order to maintain continuity. A possible solution to this problem is that each of the database lists contains only one pointer. This pointer is a null pointer as long as the list contains no entries. To add entries, a $C++$ 'std::deque' [4] is generated, with the pointer in the database pointing to the first element. If further elements are added to this list, the 'std::deque' is extended by a further element. The advantage of the 'std::deque' is that the elements do not have to be stored continuously in memory and can be accessed from both ends of the container, which makes fast initialization possible. However, unlike before, one or more dereferences are required for each search in the database. This can have a negative effect on search times, which needs to be investigated further.
Another aspect with which the database can be further improved is a rebinning procedure. This involves adjusting the bin boundaries based on the available data and re-sorting all entries. This should in turn make them more evenly distributed and, on average, speed up searches in the database. As such a procedure is very time-consuming, it should only be carried out if the samples are very inhomogeneously distributed. It is therefore necessary to develop a measure to determine this inhomogeneity so that automated rebinning can be realized.
The last two aspects both concern the parallel replica method and in particular the replica compare itself. Until now, the system with the minimum energy was always replicated in order to minimize the energy as quickly as possible. This raises the question of whether it might not be beneficial to replicate not just the minimum, but the N minimum systems in order to retain more diversity in the configurations searched.
Another method to further optimize the minimization of the potential energy has already been briefly touched upon, but it intervenes more strongly in the minimization process. The idea is to identify trial moves during the replica compare that are independent of each other, since the distance between the trial move configurations is larger than their radii plus the cutoff radii of the pair interactions between the atoms. Under this condition, these trial moves can all be included in the same configuration without the need for further relaxation. The new energy delta of the composite system corresponds to the sum of the deltas of the individual trial moves. If successfully applied, this procedure has the positive side effect of relativizing the problem of discarded trial moves, as these can be used to further minimize the system energy.

References

[1] Veiga R. G. A. et al. "Atomistic modeling of carbon Cottrell atmospheres in bcc iron". In: *Journal of Physics: Condensed Matter* 25.2 (2012), p. 025401. DOI: 10.1088/0953-8984/25/2/025401. URL: https://dx.doi.org/10.1088/0953-8984/25/2/025401.

[2] Cottrell A.H. and Bilby B.A. "Dislocation Theory of Yielding and Strain Ageing of Iron". In: *Proceedings of the Physical Society. Section A* 62.1 (1949), p. 49. DOI: 10.1088/0370-1298/62/1/308. URL: https://dx.doi.org/10.1088/0370-1298/62/1/308.

[3] Li C. et al. "Clustering for approximate similarity search in high-dimensional spaces". In: *IEEE Transactions on Knowledge and Data Engineering* 14.4 (2002), pp. 792–808. DOI: 10.1109/TKDE.2002.1019214.

[4] *Cppreference: std::deque*. URL: https://en.cppreference.com/w/cpp/container/deque.html.

[5] Frenkel D. and Smit B. *Understanding Molecular Simulation: From Algorithms to Applications*. 3rd ed. Academic Press, 1996.

[6] Jackson J. D. *Classical Electrodynamics*. Third edition. ISBN: 0-471-30932-X. John Wiley & Sons, Inc., 1999.

[7] Perez D., Uberuaga B. P., and Voter A. F. "The parallel replica dynamics method – Coming of age". In: *Computational Materials Science* 100 (2015). Special Issue on Advanced Simulation Methods, pp. 90–103. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2014.12.011. URL: https://www.sciencedirect.com/science/article/pii/S0927025614008489.

[8] Sutmann G. and Steffen B. "A particle–particle particle-multigrid method for long-range interactions in molecular simulations". In: *Computer Physics Communications* 169.1 (2005), pp. 343–346. ISSN: 0010-4655. DOI: https://doi.org/10.1016/j.cpc.2005.03.077.

[9] Ganesan H. "Highly parallel molecular dynamics / Monte Carlo coupling towards solutes segregation modelling". PhD thesis. Ruhr-Universität Bochum, Universitätsbibliothek, 2019. DOI: 10.13154/294-6470.

[10] Ganesan H., Begau C., and Sutmann G. "MC/MD Coupling for Scale Bridging Simulations of Solute Segregation in Solids: An Application Study". In: *Simulation Science*. Springer International Publishing, 2018, pp. 112–127. ISBN: 978-3-319-96271-9.

[11] Ganesan H., Teijeiro C., and Sutmann G. "Parallelization comparison and optimization of a scale-bridging framework to model Cottrell atmospheres". In: *Computational Materials Science* 155 (2018), pp. 439–449. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2018.08.055. URL: https://www.sciencedirect.com/science/article/pii/S0927025618305834.

[12] Ganesan H., Teijeiro C., and Sutmann G. "Parallelization comparison and optimization of a scale-bridging framework to model Cottrell atmospheres". In: *Computational Materials Science* 155 (2018), pp. 439–449. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2018.08.055. URL: https://www.sciencedirect.com/science/article/pii/S0927025618305834.

[13] Guénolé J. et al. "Assessment and optimization of the fast inertial relaxation engine (fire) for energy minimization in atomistic simulations and its implementation in lammps". In: *Computational Materials Science* 175 (2020), p. 109584. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2020.109584. URL: https://www.sciencedirect.com/science/article/pii/S0927025620300756.

[14] Stadler J., Mikulla R., and Trebin H.-R. "IMD: A Software Package for Molecular Dynamics Studies on Parallel Computers". In: *International Journal of Modern Physics C* 08.05 (1997), pp. 1131–1140. DOI: 10.1142/S0129183197000990. URL: https://doi.org/10.1142/S0129183197000990.

[15] *JUSUF Supercomuter at Forschungszentrum Jülich GmbH*. URL: https://www.fz-juelich.de/en/ias/jsc/systems/supercomputers/jusuf.

[16] *LAMMPS Documentation*. 2025. URL: https://docs.lammps.org/Manual.html.

[17] Abramowitz M. and Stegun I. A. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. Vol. 55. ISBN: 0-486-61272-4. US Government printing office, 1968. URL: https://personal.math.ubc.ca/~cbm/aands/toc.htm.

[18] Longsworth M. *Biasing Technique in Parallel Monte Carlo Simulations for Segregation in Solids*. Masterthesis, RWTH Aachen, 2017. 2017. URL: https://juser.fz-juelich.de/record/840660.

[19] *M.P.I. Forum, "MPI: A Message-Passing Interface Standard Version 3.0."* 2012. URL: http://mpi-forum.org/mpi-30/.

[20] Metropolis N. and Ulam S. "The Monte Carlo Method". In: *Journal of the American Statistical Association* 44.247 (1949). PMID: 18139350, pp. 335–341. DOI: 10.1080/01621459.1949.10483310. URL: https://www.tandfonline.com/doi/abs/10.1080/01621459.1949.10483310.

[21] Hohenberg P. and Kohn W. "Inhomogeneous Electron Gas". In: *Phys. Rev.* 136 (3B 1964), B864–B871. DOI: 10.1103/PhysRev.136.B864. URL: https://link.aps.org/doi/10.1103/PhysRev.136.B864.

[22] Shirts M. R. and Pande V. S. "Mathematical Analysis of Coupled Parallel Simulations". In: *Phys. Rev. Lett.* 86 (22 2001), pp. 4983–4987. DOI: 10.1103/PhysRevLett.86.4983. URL: https://link.aps.org/doi/10.1103/PhysRevLett.86.4983.

[23] Daw M. S. and Baskes M. I. "Embedded-atom method: Derivation and application to impurities, surfaces, and other defects in metals". In: *Phys. Rev. B* 29 (1984), pp. 6443–6453. DOI: 10.1103/PhysRevB.29.6443.

[24] Olver F. W. J.; Olde Daalhuis A. B.; Lozier D. W.; Schneider B. I.; Boisvert R. F.; Clark C. W.; Miller B. R.; Saunders B. V.; Cohl H. S. and McClain M. A. *NIST Digital Library of Mathematical Functions*. `https://dlmf.nist.gov/`, Release 1.2.3 of 2024-12-15.

[25] Arthur F. Voter. "Parallel replica method for dynamics of infrequent events". In: *Phys. Rev. B* 57 (22 1998), R13985–R13988. DOI: `10.1103/PhysRevB.57.R13985`. URL: `https://link.aps.org/doi/10.1103/PhysRevB.57.R13985`.

7 Appendix

7.1 Parallel replica worker

```
void Slave::runAsSlaveParRep(Master& master, int pRank, string mcOutputFile,
    double rSphere, int checkInterval)
 {
   \\Initilization[...]
   while(!finished)
   {
     currentMCStep++;
     nIterations++;
     sphereCenters.clear();
     inputSphere.clear();

     //Initialise new trial move
     mcStepTimeStampHist.push_back(chrono::duration_cast<chrono::milliseconds
    >(trialMoveGenTimeStart-trialMoveTimeBegin).count());
     if(threadRank==0)
     {
       currJob = Job(currentMCStep, master, (*master.masterDomain_),
    currentMCStep);
       currJob.myResult.isConverged = false;
       currJob.myResult.result=false;
       currJob.state = FINISHED;
       sphereCenters.push_back(globalList[currJob.swapList[0].ID].getPosition
    ());
       sphereCenters.push_back(globalList[currJob.swapList[1].ID].getPosition
    ());

       //Construct configuration
       master.masterDomain_->createConfiguration(currJob.swapList,
    inputSphere);
       for (const auto& s : currJob.swapList)
       {
         // adding chosen Particle into configuration
         Particle  particle  = globalList[s.ID];
         particle.setType(s.type);
         inputSphere.push_back(particle);
       }
     }
   //Run configuration
     if(threadRank == 0)
     {
       if(predictEnergy_)
       {
         //Predict energy for the swap operation
         vector<long> swapIDList = vector<long>{currJob.swapList[0].ID,
    currJob.swapList[1].ID};
           mpPrediction = predObject.predictEnergy(inputSphere, sphereCenters
    , rSphere, swapIDList);
         energyDiff = predObject.getEnergyDiff(inputSphere, sphereCenters);
```

```
41          eBefore = energyDiff.first + energyDiff.second;
42          ePredFe = mpPrediction[0].back();
43          ePredFeC = mpPrediction[1].back();
44          ePred = ePredFe + ePredFeC;
45
46          if(mpPrediction[0].size() > 16 || mpPrediction[1].size() > 16) //15
    coeff + 1 energy; In case of no prediction is set to be 17 where energy
    is 0 and last one is atom count (see predictEnergy())
47          {
48            nNoPredictions++;
49            nRelaxations++;
50            //no prediction could have been made -> perform normal localMD run
51            int flagRunLocalMD=1;
52            MPI_Bcast(&flagRunLocalMD, 1, MPI_INT, 0, threadComm_);
53
54            //Broadcast new job
55            nParticles = inputSphere.size();
56            long* types = new long[nParticles];
57            double* ePot = new double[nParticles];
58            double* positions = new double[3*nParticles];
59            for(auto i=0; i<inputSphere.size(); ++i)
60            {
61              types[i] = inputSphere[i].getType();
62              ePot[i] = inputSphere[i].getEpot();
63              positions[3*i] = inputSphere[i].getPosition().x;
64              positions[3*i+1] = inputSphere[i].getPosition().y;
65              positions[3*i+2] = inputSphere[i].getPosition().z;
66            }
67            MPI_Bcast(&nParticles, 1, MPI_LONG, 0, threadComm_);
68            MPI_Bcast(types, nParticles, MPI_LONG, 0, threadComm_);
69            MPI_Bcast(ePot, nParticles, MPI_DOUBLE, 0, threadComm_);
70            MPI_Bcast(positions, 3*nParticles, MPI_DOUBLE, 0, threadComm_);
71
72            currJob.myResult.isConverged = mdObject.runLocalMD(inputSphere,
    currJob.myResult.particles,threadComm_, currentMCStep);
73            if(mpPrediction[0].size() > 16){mpPrediction[0].pop_back();}
74            if(mpPrediction[1].size() > 16){mpPrediction[1].pop_back();}
75
76            if(currJob.myResult.isConverged < 0)
77            {
78              nFailedLocalMD++;
79            }else
80            {
81              nFinishedRelaxations++;
82              //replace non predicted energy ("0") with actual energy
83              energyDiff = predObject.getEnergyDiff(currJob.myResult.particles
    , sphereCenters);
84              mpPrediction[0].back() = energyDiff.first;
85              mpPrediction[1].back() = energyDiff.second;
86
87              if(mpPrediction[0].back() + mpPrediction[1].back() < eBefore){
88                nNoPredLower++;
89              }
90              //add new samples to database
```

```
91              predObject.mpCoeffDbFe_.addSample(mpPrediction[0]);
92              predObject.mpCoeffDbFeC_.addSample(mpPrediction[1]);
93            }
94            predObject.noPrediction(0);
95            predObject.noPrediction(1);
96          }else
97          {
98            nPredictions++;
99            //if a lower energy is predicted run normal relaxation step
100           if(ePred < eBefore)
101           {
102             nPredLower++;
103             nRelaxations++;
104             int flagRunLocalMD=1;
105             MPI_Bcast(&flagRunLocalMD, 1, MPI_INT, 0, threadComm_);
106
107             //Broadcast new job
108             nParticles = inputSphere.size();
109             long* types = new long[nParticles];
110             double* ePot = new double[nParticles];
111             double* positions = new double[3*nParticles];
112             for(auto i=0; i<inputSphere.size(); ++i)
113             {
114               types[i] = inputSphere[i].getType();
115               ePot[i] = inputSphere[i].getEpot();
116               positions[3*i] = inputSphere[i].getPosition().x;
117               positions[3*i+1] = inputSphere[i].getPosition().y;
118               positions[3*i+2] = inputSphere[i].getPosition().z;
119             }
120             MPI_Bcast(&nParticles, 1, MPI_LONG, 0, threadComm_);
121             MPI_Bcast(types, nParticles, MPI_LONG, 0, threadComm_);
122             MPI_Bcast(ePot, nParticles, MPI_DOUBLE, 0, threadComm_);
123             MPI_Bcast(positions, 3*nParticles, MPI_DOUBLE, 0, threadComm_);

124             currJob.myResult.isConverged = mdObject.runLocalMD(inputSphere,
    currJob.myResult.particles,threadComm_, currentMCStep);
125
126             if(currJob.myResult.isConverged < 0)
127             {
128               nFailedLocalMD++;
129             }else
130             {
131               nFinishedRelaxations++;
132               //replace predicted energy with actual energy
133               energyDiff = predObject.getEnergyDiff(currJob.myResult.
    particles, sphereCenters);
134               mpPrediction[0].back() = energyDiff.first;
135               mpPrediction[1].back() = energyDiff.second;
136
137               if(mpPrediction[0].back() + mpPrediction[1].back() < eBefore)
138               {
139                 nRightLowerPred++;
140                 predObject.rightPrediction(0);
141                 predObject.rightPrediction(1);
```

```
142               }else
143               {
144                 nWrongLowerPred++;
145                 predObject.wrongPrediction(0);
146                 predObject.wrongPrediction(1);
147               }
148               //add new samples to database
149               predObject.mpCoeffDbFe_.addSample(mpPrediction[0]);
150               predObject.mpCoeffDbFeC_.addSample(mpPrediction[1]);
151             }
152           }else
153           {
154             nPredHigher++;
155             if(alwaysRunRelaxation_ || ((1.0*rand()/RAND_MAX) > (double(
    nRightHigherPred)/nPredHigherRunAnyway)))
156             {
157               nPredHigherRunAnyway++;
158               nRelaxations++;
159               int flagRunLocalMD=1;
160               MPI_Bcast(&flagRunLocalMD, 1, MPI_INT, 0, threadComm_);
161
162               //Broadcast new job
163               nParticles = inputSphere.size();
164               long* types = new long[nParticles];
165               double* ePot = new double[nParticles];
166               double* positions = new double[3*nParticles];
167               for(auto i=0; i<inputSphere.size(); ++i)
168               {
169                 types[i] = inputSphere[i].getType();
170                 ePot[i] = inputSphere[i].getEpot();
171                 positions[3*i] = inputSphere[i].getPosition().x;
172                 positions[3*i+1] = inputSphere[i].getPosition().y;
173                 positions[3*i+2] = inputSphere[i].getPosition().z;
174               }
175               MPI_Bcast(&nParticles, 1, MPI_LONG, 0, threadComm_);
176               MPI_Bcast(types, nParticles, MPI_LONG, 0, threadComm_);
177               MPI_Bcast(ePot, nParticles, MPI_DOUBLE, 0, threadComm_);
178               MPI_Bcast(positions, 3*nParticles, MPI_DOUBLE, 0, threadComm_)
    ;
179               currJob.myResult.isConverged = mdObject.runLocalMD(inputSphere
    ,currJob.myResult.particles,threadComm_, currentMCStep);
180               if(currJob.myResult.isConverged < 0)
181               {
182                 nFailedLocalMD++;
183               }else
184               {
185                 nFinishedRelaxations++;
186                 //replace predicted energy with actual energy
187                 energyDiff = predObject.getEnergyDiff(currJob.myResult.
    particles, sphereCenters);
188                 mpPrediction[0].back() = energyDiff.first;
189                 mpPrediction[1].back() = energyDiff.second;
190                 if(mpPrediction[0].back() + mpPrediction[1].back() > eBefore
    )
```

```
191                 {
192                   nRightHigherPred++;
193                   predObject.rightPrediction(0);
194                   predObject.rightPrediction(1);
195                 }else
196                 {
197                   nWrongHigherPred++;
198                   nPredHigherButLower++;
199                   predObject.wrongPrediction(0);
200                   predObject.wrongPrediction(1);
201                 }
202                 //add new samples to database
203                 predObject.mpCoeffDbFe_.addSample(mpPrediction[0]);
204                 predObject.mpCoeffDbFeC_.addSample(mpPrediction[1]);
205               }
206             }else
207             {
208               int flagRunLocalMD=0;
209               MPI_Bcast(&flagRunLocalMD, 1, MPI_INT, 0, threadComm_);
210 
211               //dont run localMD
212               currJob.myResult.isConverged = false;
213               currJob.myResult.particles = inputSphere;
214             }
215           }
216         }
217       }else
218       {
219         nRelaxations++;
220         int flagRunLocalMD=1;
221         MPI_Bcast(&flagRunLocalMD, 1, MPI_INT, 0, threadComm_);
222 
223         //Broadcast new job
224         nParticles = inputSphere.size();
225         long* types = new long[nParticles];
226         double* ePot = new double[nParticles];
227         double* positions = new double[3*nParticles];
228         for(auto i=0; i<inputSphere.size(); ++i)
229         {
230           types[i] = inputSphere[i].getType();
231           ePot[i] = inputSphere[i].getEpot();
232           positions[3*i] = inputSphere[i].getPosition().x;
233           positions[3*i+1] = inputSphere[i].getPosition().y;
234           positions[3*i+2] = inputSphere[i].getPosition().z;
235         }
236         MPI_Bcast(&nParticles, 1, MPI_LONG, 0, threadComm_);
237         MPI_Bcast(types, nParticles, MPI_LONG, 0, threadComm_);
238         MPI_Bcast(ePot, nParticles, MPI_DOUBLE, 0, threadComm_);
239         MPI_Bcast(positions, 3*nParticles, MPI_DOUBLE, 0, threadComm_);
240         currJob.myResult.isConverged = mdObject.runLocalMD(inputSphere,
    currJob.myResult.particles,threadComm_, currentMCStep);
241         if(currJob.myResult.isConverged==1){nFinishedRelaxations++;}
242       }
243      }else
```

```
244       {
245         int flagRunLocalMD;
246         MPI_Bcast(&flagRunLocalMD, 1, MPI_INT, 0, threadComm_);
247         if(flagRunLocalMD == 1)
248         {
249           //Receive broadcast job
250           MPI_Bcast(&nParticles, 1, MPI_LONG, 0, threadComm_);
251           long* types = new long[nParticles];
252           double* ePot = new double[nParticles];
253           double* positions = new double[3*nParticles];
254           MPI_Bcast(types, nParticles, MPI_LONG, 0, threadComm_);
255           MPI_Bcast(ePot, nParticles, MPI_DOUBLE, 0, threadComm_);
256           MPI_Bcast(positions, 3*nParticles, MPI_DOUBLE, 0, threadComm_);
257           Particle p;
258           for(auto i=0; i<nParticles; ++i)
259           {
260             p.setType(types[i]);
261             p.setEpot(ePot[i]);
262             p.setPosition(positions[3*i], positions[3*i+1], positions[3*i+2]);
263             inputSphere.push_back(p);
264           }
265           vector<Particle> tempParticles;
266           currJob.myResult.isConverged = mdObject.runLocalMD(inputSphere,
    tempParticles,threadComm_, currentMCStep);
267         }
268       }
269
270       //Invoke Sync
271       if(threadRank == 0)
272       {
273         if(currJob.myResult.isConverged == 1)
274         {
275           //Acceptance check
276           master.processJob(currJob);
277           if(currJob.myResult.result==true && currJob.myResult.deltaEnergy <
    0){
278             nAccepted++;
279             nTrialSuccess++;
280           }else{
281             nRejected++;
282           }
283           if(!terminationReqSend && !syncReqSend && nTrialSuccess>=
    checkInterval)
284           {
285             syncReqSend=true;
286             ePotHistGlobal.push_back(master.masterDomain_->getTotalEnergy());
287             if(worldRank!=0){MPI_Isend(&worldRank, 1, MPI_INT, 0, TAGSYNC,
    mainThreadComm_, &request);}
288           }
289         }
290
291         if(!terminationReqSend && currentMCStep>=totalMCSteps)
292         {
293           terminationReqSend=true;
```

```
294         if(worldRank!=0){MPI_Isend(&worldRank, 1, MPI_INT, 0, TAGTERM,
    mainThreadComm_,  &request);}
295       }
296     }
297     //Dynamic Sync
298     if(worldRank==0) // MasterSlave
299     {
300       MPI_Iprobe(MPI_ANY_SOURCE, MPI_ANY_TAG, mainThreadComm_, &flagSYNC, &
    probeStatus);
301       if(syncReqSend || (flagSYNC==1 && probeStatus.MPI_TAG==TAGSYNC))
302       {
303         bcTAG=TAGCOMPARE;
304         if(!syncReqSend){
305           MPI_Irecv(&srcRank, 1, MPI_INT, probeStatus.MPI_SOURCE, TAGSYNC,
    mainThreadComm_, &request);
306           MPI_Wait(&request, &status);
307         }else{
308           srcRank=0;
309         }
310         compareConfigs=true;
311         for(auto i=1; i<mainThreadSize; ++i)
312         {
313           MPI_Send(&srcRank, 1, MPI_INT, i, TAGCOMPARE, mainThreadComm_);
314         }
315       }else if(terminationReqSend || (flagSYNC==1 && probeStatus.MPI_TAG==
    TAGTERM)){
316         bcTAG=TAGTERM;
317         if(!terminationReqSend){
318           MPI_Irecv(&srcRank, 1, MPI_INT, probeStatus.MPI_SOURCE, TAGTERM,
    mainThreadComm_, &request);
319           MPI_Wait(&request, &status);
320         }else{
321           srcRank=0;
322         }
323         finished=true;
324         for(auto i=1; i<mainThreadSize; ++i)
325         {
326           MPI_Send(&srcRank, 1, MPI_INT, i, TAGTERM, mainThreadComm_);
327         }
328       }else{
329         bcTAG=TAGSWAPLIST;
330       }
331       MPI_Bcast(&bcTAG, 1, MPI_INT, 0, threadComm_);
332       flagSYNC=0;
333     }else if(threadRank==0) //MainSlave
334     {
335       MPI_Iprobe(0, MPI_ANY_TAG, mainThreadComm_, &flagCOMPARE, &probeStatus
    );
336       if(flagCOMPARE==1 && probeStatus.MPI_TAG==TAGCOMPARE)
337       {
338         bcTAG=TAGCOMPARE;
339         MPI_Recv(&srcRank, 1, MPI_INT, 0, TAGCOMPARE, mainThreadComm_, &
    status);
340         compareConfigs=true;
```

```
341       }else if(flagCOMPARE==1 && probeStatus.MPI_TAG==TAGTERM){
342         bcTAG=TAGTERM;
343         MPI_Recv(&srcRank, 1, MPI_INT, 0, TAGTERM, mainThreadComm_, &status)
      ;
344         finished=true;
345       }else{
346         bcTAG=TAGSWAPLIST;
347       }
348       MPI_Bcast(&bcTAG, 1, MPI_INT, 0, threadComm_);
349       flagCOMPARE=0;
350     }else //ThreadSlave
351     {
352       MPI_Bcast(&bcTAG, 1, MPI_INT, 0, threadComm_);
353       if(bcTAG==TAGCOMPARE)
354       {
355         compareConfigs=true;
356       }else if(bcTAG==TAGTERM){
357         finished=true;
358       }//else No compare was initialised and mainThread broadcasts new Job
359     }
360
361     //Compare Configs
362     if(compareConfigs)
363     {
364       nCompares++;
365       MPI_Barrier(slaveComm_);
366       if(threadRank == 0)
367       {
368         //Get process with minimum energy
369         double currEpot = ePotHistGlobal.back();
370         double* allEpot = new double[mainThreadSize];
371         MPI_Allgather(&currEpot, 1, MPI_DOUBLE, allEpot, 1, MPI_DOUBLE,
      mainThreadComm_);
372
373         int idx_min_1=0, idx_min_2=0;
374         double eMin=allEpot[0];
375         for(auto i=1; i<mainThreadSize; ++i)
376         {
377           if(allEpot[i] < eMin)
378           {
379             idx_min_2 = idx_min_1;
380             idx_min_1 = i;
381           }
382         }
383              currentMCStep = replicate(master, idx_min_1, currentMCStep);
384         delete[] allEpot;
385
386         if(syncOnCompareCounter == syncEveryNthCompare){
387           syncOnCompareCounter=0;
388           MPI_Barrier(mainThreadComm_);
389           predObject.mpCoeffDbFe_.sync(mainThreadComm_, mainThreadRank,
      mainThreadSize, TAGDBSYNC, true);
390           MPI_Barrier(mainThreadComm_);
391           predObject.mpCoeffDbFeC_.sync(mainThreadComm_, mainThreadRank,
```

```
    mainThreadSize, TAGDBSYNC, true);
392         }else{
393           syncOnCompareCounter++;
394         }
395
396         if(mainThreadRank == idx_min_1){
397           nTrialDeletedOnCompareHist.push_back(0);
398         }else{
399           nTrialDeletedOnCompareHist.push_back(nTrialSuccess);
400           nTrialDeletedOnCompare += nTrialSuccess;
401         }
402         nTrialSuccess=0;
403         if(!syncReqSend){
404           ePotHistGlobal.push_back(master.masterDomain_->getTotalEnergy());
405         }
406         compareOnIterationHist.push_back(nIterations);
407       }else
408       {
409         currentMCStep = replicate(master, -1, currentMCStep);
410       }
411
412       //MasterSlave has to drain all SYNC calls
413       if(worldRank==0){
414         MPI_Iprobe(MPI_ANY_SOURCE, TAGSYNC, mainThreadComm_, &flagSYNC, &
    probeStatus);
415         while(flagSYNC==1)
416         {
417           MPI_Irecv(&srcRank, 1, MPI_INT, probeStatus.MPI_SOURCE, TAGSYNC,
    mainThreadComm_, &request);
418           MPI_Wait(&request, &status);
419           MPI_Iprobe(MPI_ANY_SOURCE, TAGSYNC, mainThreadComm_, &flagSYNC, &
    probeStatus);
420         }
421       }
422
423       compareConfigs=false;
424       syncReqSend=false;
425       MPI_Barrier(slaveComm_);
426     }
427   }
428  \\Evaluation[...]
429 }
430
```

7.2 Calculate multipole coefficients

```
1 vector<double> MultipolePrediction::getMpCoeff(vector<Vector3d>& atoms,
    double rSphere)
2 {
3     using namespace complex_literals;
4     vector<complex<double>> mp_coeff;
```

```
    vector<double> mp_coeff_crop;
    vector<double> pm;
    mp_coeff.assign(lMax_*lMax_, 0.0);
    double r_s_x, r_s_y, r_s_z, r_s, r_s_l, cos_phi, sin_phi, phi, cos_theta,
     sin_theta;
    long lm;

    for(int s = 0; s < atoms.size(); ++s)
    {
        r_s_x = atoms[s].x;
        r_s_y = atoms[s].y;
        r_s_z = atoms[s].z;
        r_s = sqrt(r_s_x*r_s_x + r_s_y*r_s_y + r_s_z*r_s_z);

        if(r_s_x == 0.0 && r_s_y == 0.0)
        {
            cos_phi = 1.0;
            sin_phi = 0.0;
        }else
        {
            cos_phi = r_s_x / sqrt(r_s_x*r_s_x + r_s_y*r_s_y);
            sin_phi = r_s_y / sqrt(r_s_x*r_s_x + r_s_y*r_s_y);
        }
        phi = evalPhi(cos_phi, sin_phi);

        cos_theta = r_s_z / r_s;
        sin_theta = sqrt(r_s_x*r_s_x + r_s_y*r_s_y) / r_s;

        pm = assocLegendre(lMax_, lMax_, cos_theta);

        mp_coeff[0] += 1.0;
        for(int l = 1; l < lMax_; ++l)
        {
            r_s_l = r_s;
            lm = factorial(l);
            mp_coeff[l*lMax_ + 0] += (1.0 / (r_s_l * lm)) * pm[0*lMax_+l] *
(1.0 + 1i * 0.0);
            for(int m = 1; m <= l; ++m)
            {
                lm = factorial(l + std::abs(m));
                mp_coeff[l*lMax_ + m] += (1.0 / (r_s_l * lm)) * pm[m*lMax_+l]
 * (cos(m*phi) - 1i * sin(m*phi));
            }
        }
    }

    for(int l = 0; l < lMax_; ++l)
    {
        for(int m = 0; m <= l; ++m)
        {
            mp_coeff_crop.emplace_back(mp_coeff[l*lMax_+m].real());
        }
    }

```

```
56      return mp_coeff_crop;
57  }
58
```

7.3 Multidimensional database bin index

```
1  long nonuniformbinnedDatabase::getBinIdx(vector<double>& sample)
2  {
3      long idx = 0;
4      for(auto i=0; i<nDims_; ++i)
5      {
6          if(sample[i] < bin_lo_[i])
7          {
8              idx+=0;
9          }else if(sample[i] > bin_hi_[i])
10         {
11             idx+=(nBins_-1)*dimOfs_[i];
12         }else
13         {
14             idx += int(1 + (lower_bound(bin_size_[i].begin(), bin_size_[i].
       end(), sample[i]) - bin_size_[i].begin())) * dimOfs_[i];
15         }
16     }
17     return idx;
18 }
19
```

7.4 Database similarity search

```
1  pair<vector<long>,std::vector<double>> nonuniformbinnedDatabase::getBestFit(
       vector<double>& sample, vector<double>& coeff_deviation)
2  {
3      vector<long> binIdxList;
4      long binIdx = getBinIdx(sample);
5      binIdxList.push_back(binIdx);
6      for(auto i=0; i<nDims_; ++i)
7      {
8          if(binIdx+pow(nBins_,i) < data_.size()){binIdxList.push_back(binIdx+
       pow(nBins_,i));}
9          if(binIdx-pow(nBins_,i) >= 0){binIdxList.push_back(binIdx-pow(nBins_,
       i));}
10     }
11
12     std::vector<double> mp_diff;
13     mp_diff.assign(sample.size(), 0.0);
14     std::pair<vector<long>,vector<double>> best_fit;
15     best_fit = make_pair(vector<long>{-1,-1}, vector<double>{-1});
16     long kMax=0;
17     int highest_weight = -1;
```

```
    int current_weight = 0;

    for(auto& i: binIdxList)
    {
        for(auto j=0; j<data_[i].size(); ++j)
        {
            getMpDiff(sample, data_[i][j], mp_diff);
            for(auto k=0; k<mp_diff.size(); ++k)
            {
                if(mp_diff[k] <= fabs(sample[k]*coeff_deviation[k]))
                {
                    if(k > kMax)
                    {
                        best_fit.first[0] = i;
                        best_fit.first[1] = j;
                        best_fit.second[0] = data_[i][j][15];
                        kMax = k;
                        highest_weight = data_[i][j][16];
                    }else if(k == kMax)
                    {
                        current_weight = data_[i][j][16];
                        if(current_weight > highest_weight)
                        {
                            best_fit.first[0] = i;
                            best_fit.first[1] = j;
                            best_fit.second[0] = data_[i][j][15];
                            highest_weight = current_weight;
                        }
                    }
                }else
                {
                    break;
                }
            }
        }
    }
    return best_fit;
}

```

7.5 LAMMPS relaxation step

```
int interfaceLAMMPS::runLocalMD(vector<Particle>& inConfiguration, vector<
    Particle>& outConfiguration, MPI_Comm subGroup, long jobCount)
{
    int pRank;
    int pSize;
    MPI_Comm_rank(subGroup, &pRank);
    MPI_Comm_size(subGroup, &pSize);
  long nParticles = inConfiguration.size();
    double eSumBefore=0; double eSumAfter=0;
```

```
9
10      //Initialise LAMMPS
11      const char *lmpargv[] {"liblammps", "-echo", "log", "-screen", "out.
        lammps", "-log", "log.lammps", "-in", "none"};
12      int lmpargc = sizeof(lmpargv)/sizeof(const char *);
13      LAMMPS_NS::LAMMPS *lmp = new LAMMPS_NS::LAMMPS(lmpargc, (char **)lmpargv,
         subGroup);
14      lammps_file(lmp, inputFile_);
15
16    Particle sphereParticle;
17      long nUnmatchedParticles = 0;
18      long nLammpsParticles = 0;
19      int nCoreZone = 0;
20      int nMiddleZone = 0;
21      int nOuterZone = 0;
22      int nEmbedZone = 0;
23      std::vector<long> outConfigurationMask;
24      outConfigurationMask.assign(nParticles, 0);
25
26      vector<int> lmpid_to_inConfidx; //index i is lammps particle index and
        value[i] is index of lammps particle in inConfiguration[value[i]]
27      vector<int> core_id;vector<int> core_type;vector<double> core_pos;
28      vector<int> middle_id;vector<int> middle_type;vector<double> middle_pos;
29      vector<int> outer_id;vector<int> outer_type;vector<double> outer_pos;
30      vector<int> embed_id;vector<int> embed_type;vector<double> embed_pos;
31      timeBegin = chrono::high_resolution_clock::now();
32      for(auto i=0; i<nParticles; i++)//types 0,1,2
33      {
34          eSumBefore += inConfiguration[i].getEpot();
35          if(inConfiguration[i].getType() >= 0 && inConfiguration[i].getType()
        < 3)
36          {
37              outConfigurationMask[i] = 1;
38              core_id.push_back(i);
39              core_type.push_back((inConfiguration[i].getType()%9)+1);
40              core_pos.push_back(inConfiguration[i].getPosition().x);
41              core_pos.push_back(inConfiguration[i].getPosition().y);
42              core_pos.push_back(inConfiguration[i].getPosition().z);
43          }else if(inConfiguration[i].getType() >= 9 && inConfiguration[i].
        getType() < 12)
44          {
45              outConfigurationMask[i] = 2;
46              middle_id.push_back(i);
47              middle_type.push_back((inConfiguration[i].getType()%9)+1);
48              middle_pos.push_back(inConfiguration[i].getPosition().x);
49              middle_pos.push_back(inConfiguration[i].getPosition().y);
50              middle_pos.push_back(inConfiguration[i].getPosition().z);
51          }else if(inConfiguration[i].getType() >= 18 && inConfiguration[i].
        getType() < 21)
52          {
53              outConfigurationMask[i] = 3;
54              outer_id.push_back(i);
55              outer_type.push_back((inConfiguration[i].getType()%9)+1);
56              outer_pos.push_back(inConfiguration[i].getPosition().x);
```

```
57              outer_pos.push_back(inConfiguration[i].getPosition().y);
58              outer_pos.push_back(inConfiguration[i].getPosition().z);
59          }else if(inConfiguration[i].getType() % 9 == 3)
60          {
61              outConfigurationMask[i] = 4;
62              embed_id.push_back(i);
63              embed_type.push_back(1); // handle embed atoms as Fe atoms
64              embed_pos.push_back(inConfiguration[i].getPosition().x);
65              embed_pos.push_back(inConfiguration[i].getPosition().y);
66              embed_pos.push_back(inConfiguration[i].getPosition().z);
67          }else
68          {
69              outConfigurationMask[i] = 0;
70              nUnmatchedParticles++;
71          }
72      }
73
74      int ncreated1, ncreated2, ncreated3, ncreated4;
75      nCoreZone = core_id.size();
76      int* c_type = core_type.data();
77      double* c_pos = core_pos.data();
78      ncreated1 = lammps_create_atoms(lmp, nCoreZone, NULL, c_type, c_pos, NULL
        , NULL, 0);
79
80      lmp->input->one("group core type 1 2 3");
81      nMiddleZone = middle_id.size();
82      int* m_type = middle_type.data();
83      double* m_pos = middle_pos.data();
84      ncreated2 = lammps_create_atoms(lmp, nMiddleZone, NULL, m_type, m_pos,
       NULL, NULL, 0);
85
86      lmp->input->one("group middle subtract all core");
87      nOuterZone = outer_id.size();
88      int* o_type = outer_type.data();
89      double* o_pos = outer_pos.data();
90      ncreated3 = lammps_create_atoms(lmp, nOuterZone, NULL, o_type, o_pos,
       NULL, NULL, 0);
91
92      lmp->input->one("group outer subtract all core middle");
93      nEmbedZone = embed_id.size();
94      int* e_type = embed_type.data();
95      double* e_pos = embed_pos.data();
96      ncreated4 = lammps_create_atoms(lmp, nEmbedZone, NULL, e_type, e_pos,
       NULL, NULL, 0);
97      lmp->input->one("group embed subtract all core middle outer");
98
99      nLammpsParticles = lammps_get_natoms(lmp);
100     timeEnd = chrono::high_resolution_clock::now();
101   avgTimeMeasurements[2] += chrono::duration_cast<chrono::milliseconds>(
      timeEnd-timeBegin).count();
102
103     //Make sure that all thread procs created atoms successfully
104     int atomCreationSuccess, equalAtomCount;
105     if(nLammpsParticles == nParticles){
```

```
106         equalAtomCount=1;
107     }else{
108         equalAtomCount=0;
109     }
110     MPI_Allreduce(&equalAtomCount, &atomCreationSuccess, 1, MPI_INT, MPI_SUM,
         subGroup);
111     if(atomCreationSuccess!=pSize){
112         outConfiguration = inConfiguration;
113         delete lmp;
114         return 0;
115     }
116     int* atom_ids_before = new int[nLammpsParticles];
117     lammps_gather_atoms(lmp,(char *) "id",0,1,atom_ids_before);
118 
119     for(auto& val : core_id){lmpid_to_inConfidx.push_back(val);}
120     for(auto& val : middle_id){lmpid_to_inConfidx.push_back(val);}
121     for(auto& val : outer_id){lmpid_to_inConfidx.push_back(val);}
122     for(auto& val : embed_id){lmpid_to_inConfidx.push_back(val);}
123 
124     lmp->input->one("group Fe type 1");
125     lmp->input->one("group C type 2");
126     lmp->input->one("group V type 3");
127     lmp->input->one("group FeC type 1 2");
128     lmp->input->one("group Fe_core intersect Fe core");
129     lmp->input->one("group C_core intersect C core");
130     lmp->input->one("group V_core intersect V core");
131     lmp->input->one("group FeC_core_middle subtract FeC embed outer");
132 
133     //Zone fixes
134     lmp->input->one("fix middle_fix_force middle setforce 0.0 0.0 0.0 region
    simulation_box");
135     lmp->input->one("fix outer_fix_force outer setforce 0.0 0.0 0.0 region
    simulation_box");
136     lmp->input->one("fix embed_fix_force embed setforce 0.0 0.0 0.0 region
    simulation_box");
137     //Relax all but Fe and C separate to V
138     lmp->input->one("neigh_modify exclude type 1 3");
139     lmp->input->one("neigh_modify exclude type 2 3");
140     lmp->input->one("fix v_fix_force V setforce 0.0 0.0 0.0 region
    simulation_box");
141     try{
142         lmp->input->one("minimize 1.0e-4 1.0e-6 200 2001");
143     }catch(const exception &e)
144     {
145         cout << e.what() << endl;
146         outConfiguration = inConfiguration;
147         delete lmp;
148         return 0;
149     }
150     lmp->input->one("neigh_modify exclude none");
151     lmp->input->one("unfix v_fix_force");
152     //Relax only V
153     lmp->input->one("fix fe_fix_force Fe setforce 0.0 0.0 0.0 region
    simulation_box");
```

```
154     lmp->input->one("fix c_fix_force C setforce 0.0 0.0 0.0 region
        simulation_box");
155     lmp->input->one("velocity Fe set 0 0 0");
156     lmp->input->one("velocity C set 0 0 0");
157     try{
158         lmp->input->one("minimize 1.0e-4 1.0e-6 200 2002");
159     }catch(const exception &e)
160     {
161         cout << e.what() << endl;
162         outConfiguration = inConfiguration;
163         delete lmp;
164         return 0;
165     }
166     lmp->input->one("unfix fe_fix_force");
167     lmp->input->one("unfix c_fix_force");
168     //Perform one update cycle to get all potential energies of atoms right
169     lmp->input->one("compute atom_pe_compute all pe/atom");
170     lmp->input->one("neigh_modify exclude type 1 3");
171     lmp->input->one("neigh_modify exclude type 2 3");
172     lmp->input->one("neigh_modify exclude type 3 3");
173     lmp->input->one("fix nve_fix all nve");
174     try{
175         lmp->input->one("run 0");
176     }catch(const exception &e)
177     {
178         cout << e.what() << endl;
179         outConfiguration = inConfiguration;
180         delete lmp;
181         return 0;
182     }
183     //Get updated trajectories back
184     int* atom_ids_after = new int[nLammpsParticles];
185     int* atom_types = new int[nLammpsParticles];
186     double* atom_mass = new double[nLammpsParticles];
187     double* atom_coords = new double[3*nLammpsParticles];
188     lammps_gather_atoms(lmp,(char *) "id",0,1,atom_ids_after);
189     lammps_gather_atoms(lmp,(char *) "type",0,1,atom_types);
190     lammps_gather_atoms(lmp,(char *) "mass",1,1,atom_mass);
191     lammps_gather_atoms(lmp,(char *) "x",1,3,atom_coords);
192     double* atom_pe = lmp->force->pair->eatom;
193     if(pRank == 0)
194     {
195         long coreCount = 0;
196         long middleCount = 0;
197         long outerCount = 0;
198         long unmatchedCount = 0;
199
200         int nOrderChanges=0;
201         int particleIdx;
202         double atomMassTable[4] = {55.845, 12.0107, 12.0107, 55.845};
203         for(auto i=0; i<nParticles; ++i)
204         {
205             if(outConfigurationMask[i] == 0)
206             {
```

```
207                 cout << "ERROR in LAMMPS::localMD there should be no
    configmask[i]=0" << endl;
208                 outConfiguration.push_back(inConfiguration[i]);
209                 unmatchedCount++;
210                 delete lmp;
211                 outConfiguration = inConfiguration;
212                 return 0;
213             }else
214             {
215                 particleIdx = lmpid_to_inConfidx[atom_ids_after[i-
    unmatchedCount]-1];
216                 if(particleIdx<0 || particleIdx >= inConfiguration.size())
217                 {
218                     cout << "LAMMPS " << myrank_ << " ERROR: particleIdx=" <<
     particleIdx << " inConfig.size()=" << inConfiguration.size() << "
    atomIdAft:" << atom_ids_after[i-unmatchedCount]-1 << " at i=" << i << "
    with nLammpsPart=" << nLammpsParticles << " and nPart=" << nParticles <<
    endl;
219                 }
220                 sphereParticle.setmcID(inConfiguration[particleIdx].getmcID()
    );
221                 sphereParticle.setType(inConfiguration[particleIdx].getType()
    );
222                 sphereParticle.setMass(atomMassTable[sphereParticle.getType()
    %9]);
223                 sphereParticle.setPosition(atom_coords[3*(i-unmatchedCount)],
     atom_coords[3*(i-unmatchedCount)+1], atom_coords[3*(i-unmatchedCount)
    +2]);
224                 sphereParticle.setEpotBeforeLammps(inConfiguration[
    particleIdx].getEpot());
225                 outConfiguration.push_back(sphereParticle);
226                 eSumAfter += outConfiguration.back().getEpot();
227             }
228         }
229         delete[] atom_ids_after;
230         delete[] atom_types;
231         delete[] atom_mass;
232         delete[] atom_coords;
233 
234         if(inConfiguration.size() != outConfiguration.size())
235         {
236             cout << "LAMMPS " << myrank_ << " ERROR: Configuration sizes
    after Lammps relaxation dont match." << endl;
237         }
238     }
239     delete lmp;
240     return 1;
241 }
242 
```