FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Implementation of Linked-Cells Traversals for 3-Body Interactions in AutoPas

Nanxing Nick Deng

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Implementation of Linked-Cells Traversals for 3-Body Interactions in AutoPas

Author:	Nanxing Nick Deng
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Markus Mühlhäußer, M.Sc.
Date:	15.02.2024

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

Munich, 15.02.2024 Nanxing Nick Deng

Acknowledgements

First, I would like to express my sincere gratitude to the advisors of this thesis, Markus Mühlhäußer and Fabio Gratl, for their invaluable guidance and support throughout this research. My thanks also go to the Chair of Scientific Computing and Prof. Dr. Hans-Joachim Bungartz for enabling this cooperation and allowing me to work on my thesis in this interesting area of research. I am grateful to the Leibniz Supercomputing Centre (LRZ) for providing access to the CoolMUC-2 Linux cluster, which was instrumental in the performance analysis conducted in this work. Lastly, I thank my family and friends for their encouragement during my studies.

Abstract

Research efforts in molecular dynamics are dedicated to accurately determining equilibrium states of molecular systems, which are vital for predicting material properties such as melting points and thermal conductivity. Studies have evaluated that incorporating the three-body Axilrod-Teller potential into a two-body model increases simulation accuracy. AutoPas is an open-source library that automatically selects the best-fitting strategies for any given particle simulation scenario, aiming to deliver optimal node-level performance. The linked cells algorithm, used to efficiently identify neighboring particles, is part of this selection pool. There are multiple cell traversal schemes in AutoPas for pairwise interactions. This thesis implemented three-body linked cell traversals with the C08 base step for three-body interactions at the core, and we extended the implementational idea of the C08 base step to N-body interactions. A performance analysis of three-body traversals across simulations of different density and cell size configurations showed the dominance of C08 among all other traversals, including C08-based traversals and the C01 traversal. Cell Size Factor of 0.5 achieved the best performance in most simulations. Besides, the study highlights linked cell limitations due to the low hit rate for three-body interaction. The evaluation examines the balance point of performance gains by hit rate increase and the price of administrative overhead of cells for trying to achieve higher hit rates. This overhead constraint limits the use case of linked cells for three-body interactions to hybrid algorithms that use spatial information, e.g., Verlet lists, and to highly dense systems, where the proportion of force computation dilutes the cell overhead.

Zusammenfassung

Forschungsbemühungen in der Molekulardynamik sind darauf ausgerichtet, Gleichgewichtszustände molekularer Systeme genau zu bestimmen, die für die Vorhersage von Materialeigenschaften wie Schmelzpunkten und Wärmeleitfähigkeit von entscheidender Bedeutung sind. Studien zeigen, dass die Einbeziehung des dreikörperigen Axilrod-Teller-Potentials in ein Zweikörpermodell die Genauigkeit der Simulation erhöht. AutoPas ist eine Open-Source-Bibliothek, die automatisch die besten Rechenstrategien für ein beliebiges Partikelsimulationsszenario auswählt und darauf abzielt, eine optimale Leistung auf lokaler Computerknotenebene zu liefern. Der Linked-Cells-Algorithmus, der zur effizienten Identifizierung benachbarter Partikel verwendet wird, ist Teil des automatischen Auswahlmechanismus. In AutoPas sind mehrere Zelltraversierungs-Schemata für paarweise Wechselwirkungen implementiert. Diese Arbeit implementiert Linked-Cells-Traversierungen für Dreikörper-Wechselwirkungen in AutoPas. Dies wird durch die Implementierung eines Traversierungsschritts für Dreikörper-Interaktionen realisiert. Darüber hinaus präsentiert diese Arbeit eine Verallgemeinerung des Traversierungsschritts für N-Körper-Interaktionen. Eine Performanzanalyse von Drei-Körper-Traversierungen über Simulationen verschiedener Dichte- und Zellgrößenkonfigurationen zeigte die Dominanz von C08 unter allen anderen Traversierungen, einschließlich C08-basierter Traversierungen und der C01-Traversierung. Ein Zellgrößenfaktor von 0,5 erzielte in den meisten Simulationen die beste Leistung. Darüber hinaus hebt die Studie die Einschränkungen des Linked-Cells-Algorithmus' aufgrund der niedrigen Trefferrate für Drei-Körper-Interaktionen hervor. Die Bewertung untersucht den Gleichgewichtspunkt von Perfomanzgewinnen durch Erhöhung der Trefferrate und den Preis des administrativen Overheads von Zellen, um höhere Trefferraten zu erreichen. Diese Overhead-Beschränkung begrenzt den Anwendungsfall verketteter Zellen für Drei-Körper-Interaktionen auf Hybridalgorithmen, die räumliche Informationen verwenden, z. B. Verlet-Listen, und auf hochdichte Systeme, wo der Anteil der Kraftberechnung den Zellen-Overhead verdünnt.

Contents

1. Introduction

Molecular dynamics (MD) simulations have become indispensable for scientific domains. Their applications span various fields, from material science to biological processes, showing growing relevance as computational power increases. Despite different use cases, all simulations follow the same principle of approximating the behavior of particles defined by some form of interaction[GSBN22]. Modeling these interactions over time can provide insights into understanding reactions, phase changes, and mechanical properties at an atomic level.

Current MD research focuses on refining techniques to enhance accuracy and efficiency. A key goal in ongoing research is to understand and accurately determine the states of a molecular system in which macroscopic properties like temperature and pressure of elements are stable, also called equilibrium, which is essential to predict material properties. In particular, the equilibrium between solid and liquid is focused on since it exhibits phase behaviors to understand properties like melting point, strength, and thermal conductivity.

For over a decade, it has been known that the distribution of multi-pole moments between atoms determines different types of interactions in molecular systems. Studies have evaluated the contributions from third-order interactions involving dipoles and quadrupoles and fourth-order triple dipole interactions. These evaluations reveal a significant cancellation among multi-pole terms, indicating that the third-order triple-dipole Axilrod-Teller potential effectively represents three-body dispersion interactions. Extending a two-body model, which governs most interactions, by a three-body term can increase the simulation accuracy, especially in dense systems where higher-order interactions become more significant. [WS06, MS99, MTS01]

Developing algorithms and optimizations comes inherently with extensive simulations and complex multi-body potentials. A well-known approach is the linked cells algorithm (LC), a spatial segmentation of the simulation domain into a grid of cells, allowing efficient identification of neighboring particles. This thesis aims to implement Linked-Cells traversals for three-body interactions in the open-source C++ node-level library AutoPas.

Firstly, Chapter 2 and Chapter 3 introduce the relevant theoretical and technical backgrounds of molecular dynamics simulations and linked cell traversals. Chapter 4 describes the tools used for this work. Afterward, Chapter 5 details implementing the three-body base step for the C08 traversal and extending the implementational idea to N-body potentials. Furthermore, we optimized a function that generates cell triplets for three-body interactions in AutoPas in terms of complexity. Lastly, the performance analysis in Chapter 6 focuses on evaluating scaling for various cell sizes and particle densities, comparing traversal methods through simulations on the CoolMUC-2 Linux cluster for homogeneous particle distributions. The study also includes an analysis of an inhomogeneous example of a falling drop.

2. Theoretical Background

In a Molecular Dynamics (MD) simulation, the microscopic evolution of a multi-body system over time is determined by numerically solving the classical equations of motion, applying boundary conditions that correspond to the system's geometric or symmetrical characteristics.[TM00]

2.1. Verlet Integration

Ordinary Differential Equations (ODE) are essential for describing the time evolution of physical systems. In MD, they model how particles' positions and velocities change over time under the influence of forces, adhering to Newton's laws of motion. Given the particles' amount and non-linear behavior, finding closed-form analytical solutions for the ODEs governing the interactions is impossible. Numerical approximation methods circumvent this issue, allowing for tracking system states' discrete step-wise advancement over time.

Verlet integration is preferred in molecular dynamics simulations for its numerical stability. It's a symplectic integrator, effectively conserving the system's total energy over time, a beneficial aspect in long-term simulations prone to numerical errors.

In the context of Newton's second law

$$\frac{dv}{dt} = \frac{d^2x}{dt^2} = a = \frac{F}{m} \tag{2.1}$$

where x is position, m is mass, and F is force, the Verlet method updates the position as:

$$x_{n+1} = 2x_n - x_{n-1} + h^2 \frac{F(x_n)}{m} \tag{2.2}$$

The method requires the last two positions and the force at position x_n and depends on the key parameter h, which is the time step of the integration.[Ver67] The force is derived from multi-body-potentials, which will be introduced in Section 2.2.

2.2. Force Computation

Particle Distance

The Euclidean distance formula calculates the distances r_{ij} between particles i and j in three-dimensional space:

$$r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \tag{2.3}$$

2.2.1. Lennard-Jones Potential

The Lennard-Jones (LJ) potential dominates molecular dynamics simulations due to its computational efficiency and simplicity. Its straightforward mathematical form enables fast, resource-efficient simulations, particularly advantageous for less-dense systems where complex many-body effects are less likely to influence interactions. Combining the LJ potential with other potential models can enhance simulation accuracy in more complex systems.

The potential function takes the distance between two particles r_{ij}. It includes two other essential parameters, ϵ and σ, which can be adjusted to model interactions between molecules and atoms. ϵ determines the strength of the attraction, known as potential well, while σ represents the distance at which the potential between particles is zero.

$$U_{ij}(r_{ij}) = \begin{cases} 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right], & \text{if } 0 < r_{ij} < r_{cutoff}. \\ 0, & \text{otherwise.} \end{cases} \tag{2.4}$$

The function comprises a repulsive part, which rapidly increases as particles approach each other (modeled by the r^{-12} term), representing the Pauli exclusion principle, and an attractive part (modeled by the r^{-6} term), which describes the Van der Waals forces that occur at longer distances. Figure 2.1 illustrates the potential function for $\epsilon = 1.0$ and $\sigma = 1.0$.

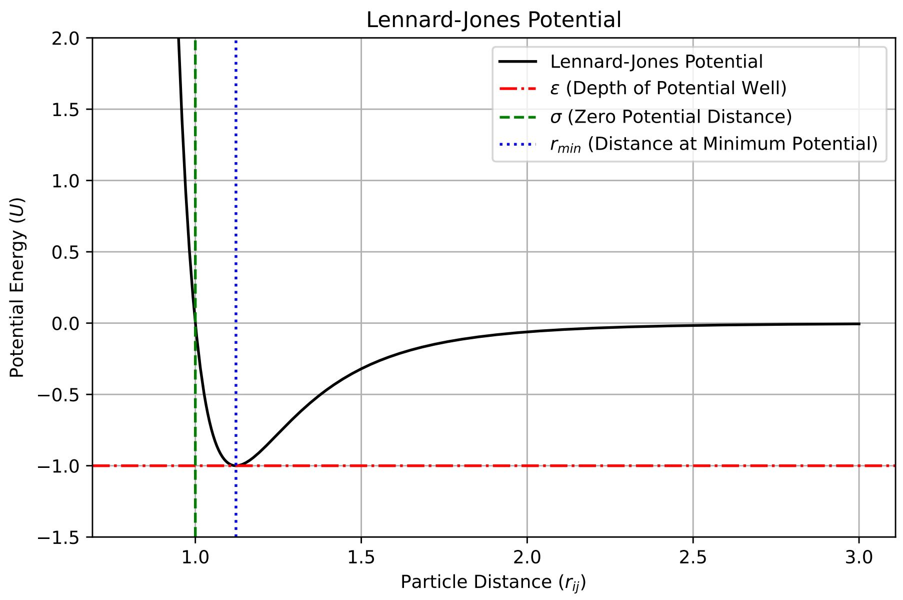


Figure 2.1.: Lennard-Jones potential with $\epsilon = 1.0$ and $\sigma = 1.0$.

2.2.2. Axilrod-Teller Potential

As mentioned in Chapter 1, including more complex interaction potentials is desirable to enable higher accuracy at the solid-liquid interface. [WS06, MTS01] showed that the three-body Axilrod-Teller (AT) potential effectively represents third-order triple-dipole dispersion interactions and can extend a two-body model. The AT potential is non-additive, which refers to being unable to derive forces from the sum of pairwise interactions among them, meaning that the potential requires triplets of particles for calculation. The resulting complexity is $O(N^3)$ with N particles.

The potential function accounts for key parameter v and the angles θ_i, θ_j, θ_k between the three observed particles i, j, k. The non-additive coefficient v quantifies the strength of interaction and is typically derived from the properties of the simulated element. The AT potential function in [AT43] is given as:

$$U_{ijk}(r_{ij}; r_{ik}; r_{jk}; \theta_i; \theta_j; \theta_k) \quad = \quad v \left(\frac{1}{r_{ij} r_{ik} r_{jk}} \right)^3 (1 + 3 \cos\theta_i \cos\theta_j \cos\theta_k) \tag{2.5}$$

The cosine law helps to express the angles in terms of the lengths of the sides, which are the distances between particles:

$$r_{ij}^2 = r_{ik}^2 + r_{jk}^2 - 2 r_{ik} r_{jk} \cos\theta_k \quad \Rightarrow \quad \cos\theta_k = \frac{-r_{ij}^2 + r_{ik}^2 + r_{jk}^2}{2 r_{ik} r_{jk}} \tag{2.6}$$

By applying the cosine law on Equation 2.5, the complexity of the cosine function, including angles, is resolved, and only dependencies on the relative inter-particle distances r_{ij}, r_{ik}, r_{jk} and the coefficient v remain:

$$U_{ijk} \quad = \quad v \left[\left(\frac{1}{r_{ij} r_{ik} r_{jk}} \right)^3 + \left(\frac{3(-r_{ij}^2 + r_{ik}^2 + r_{jk}^2)(r_{ij}^2 - r_{ik}^2 + r_{jk}^2)(r_{ij}^2 + r_{ik}^2 - r_{jk}^2)}{8 r_{ij}^5 r_{ik}^5 r_{jk}^5} \right) \right] \tag{2.7}$$

2.2.3. Force Derivation

Deriving forces from a potential energy function is done by calculating the negative gradient of the potential energy function with respect to the positions of the particles involved. The force F_i on particle i derived from potential U equals a vector consisting of one component for each spatial direction, x, y, and z:

$$F_i = -\nabla_i U = \left(-\frac{\delta U}{\delta x_i}, -\frac{\delta U}{\delta y_i}, -\frac{\delta U}{\delta z_i} \right) \tag{2.8}$$

Newton's Third Law Of Motion

Newton's third law states that when one body exerts a force on a second body, the second body simultaneously exerts a force equal magnitude and opposite direction on the first body [New33].

In the context of particle simulations, this implies that the force between particles i and j, denoted as F_{ij}, is equal in magnitude but opposite in direction to the force between j and i, i.e., $F_{ij} = -F_{ji}$. This symmetry allows for optimization by halving the number of force evaluations in a two-body model.

Marcelli presents in [Mar01] how Newton's third law can be applied to a three-body model using the derivative of the potential with respect to spatial direction.

In the following $F^x_{i(jk)}$ shall be the total force exerted in x-direction on particle i in the three-body relationship with particles j and k, and $F^x_{i(j)k}$ and $F^x_{i(k)j}$ being the contribution from only particle j and k respectively.

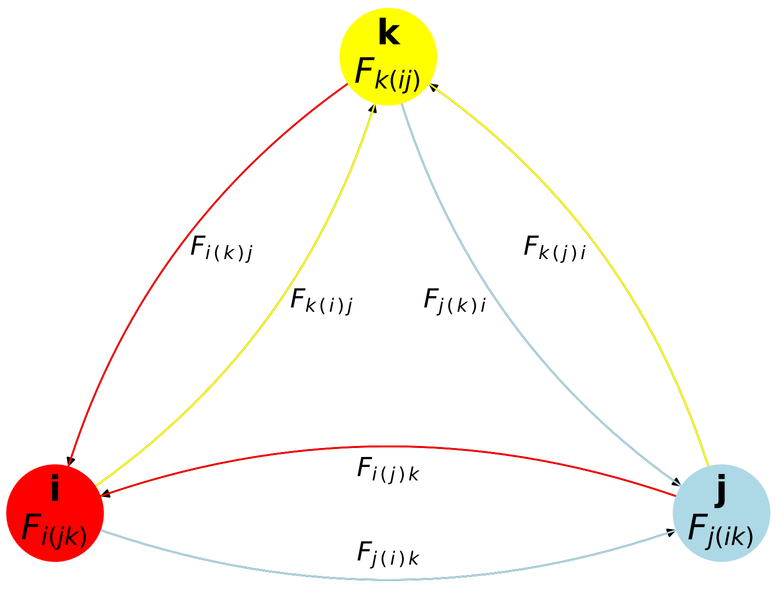


Figure 2.2.: Nodes represent particles annotated with the total force exerted on them in a triplet. Arrows represent force contributions by one particle. The same colored arrows combine to the total force on a particle. Arrows connecting the same two particles are equal in magnitude and opposite in direction.

Since the potential function depends on the particle distance function, which itself requires particle coordinates, applying the chain rule resolves the distance dependencies with coordinate dependencies. Then, identifying force components with opposite directions but equal magnitude helps to interpret the force relationship in the three-body interaction model (Figure 2.2):

$$F^x_{i(jk)} = -\frac{\delta U_{ijk}}{\delta x_i} = -\left[\frac{\delta x_{ij}}{\delta x_i}\frac{\delta U_{ijk}}{\delta x_{ij}} + \frac{\delta x_{ik}}{\delta x_i}\frac{\delta U_{ijk}}{\delta x_{ik}}\right] = -\left[\frac{\delta U_{ijk}}{\delta x_{ij}} + \frac{\delta U_{ijk}}{\delta x_{ik}}\right] = F^x_{i(j)k} + F^x_{i(k)j} \tag{2.9}$$

$$F^x_{j(ik)} = -\frac{\delta U_{ijk}}{\delta x_j} = -\left[\frac{\delta x_{ij}}{\delta x_j}\frac{\delta U_{ijk}}{\delta x_{ij}} + \frac{\delta x_{jk}}{\delta x_j}\frac{\delta U_{ijk}}{\delta x_{jk}}\right] = -\left[-\frac{\delta U_{ijk}}{\delta x_{ij}} + \frac{\delta U_{ijk}}{\delta x_{jk}}\right] = -F^x_{i(j)k} + F^x_{j(k)i} \tag{2.10}$$

$$F^x_{k(ij)} = -\frac{\delta U_{ijk}}{\delta x_k} = -\left[\frac{\delta x_{ik}}{\delta x_k}\frac{\delta U_{ijk}}{\delta x_{ik}} + \frac{\delta x_{ik}}{\delta x_k}\frac{\delta U_{ijk}}{\delta x_{jk}}\right] = -\left[-\frac{\delta U_{ijk}}{\delta x_{ik}} - \frac{\delta U_{ijk}}{\delta x_{jk}}\right] = -F^x_{i(k)j} - F^x_{j(k)i} \tag{2.11}$$

Combining Equation 2.9, Equation 2.10 and Equation 2.11 shows that the total forces exerted on particle i and j are equal in magnitude and opposite in direction to the total force exerted on particle k:

$$\begin{aligned} F^x_{i(jk)} + F^x_{j(ik)} &= F^x_{i(j)k} + F^x_{i(k)j} - F^x_{i(j)k} + F^x_{j(k)i} \\ &= F^x_{i(k)j} + F^x_{j(k)i} \\ &= -\,F^x_{k(ij)} \end{aligned} \tag{2.12}$$

3. Technical Background

The following chapter presents the technical background for different neighbor identification algorithms, including the linked cell algorithm. It introduces linked cell traversals with base steps and domain coloring schemes.

The core idea of neighbor identification algorithms is to minimize the total force computations. Optimally, only pairs and triplets of particles with particle distances less than the cutoff radius (neighbors) are identified and considered for expensive force computations. The algorithm choice depends on the simulation demands, like system density and homogeneity, with no optimal strategy for each scenario. Each algorithm comes with different data structures and processing techniques and has its strengths in performance.

Cutoff Radius

The cutoff radius r_{cutoff} defines the maximum distance within which particle interactions are considered. It sets an interaction sphere (IS) around each particle, beyond which the forces exerted by other particles are deemed negligible and thus ignored in the computation. This concept is important for efficiently calculating non-bonded short-range interactions, such as van der Waals forces or electrostatic interactions, which decrease rapidly with distance. Implementing a cutoff radius reduces the computational load strongly, limiting the number of particle combinations significant for force calculations. The choice of the cutoff radius is a balance between computational efficiency and accuracy of the simulation.

For a pairwise potential model, the complexity of the naive approach of $O(N^2)$ with N representing the number of particles can be reduced to $O(N)$ by introducing a cutoff radius without significant errors. The red line in Figure 3.1 symbolizes the cutoff radius. [GSBN22]

In a three-body model, there are different possibilities for applying the cutoff radius. This thesis uses the version where all three particles i, j, and k must be within each other's interaction zone. This means the three particle-distances r_{ij}, r_{ik} and r_{jk} need to be less than r_{cutoff}.

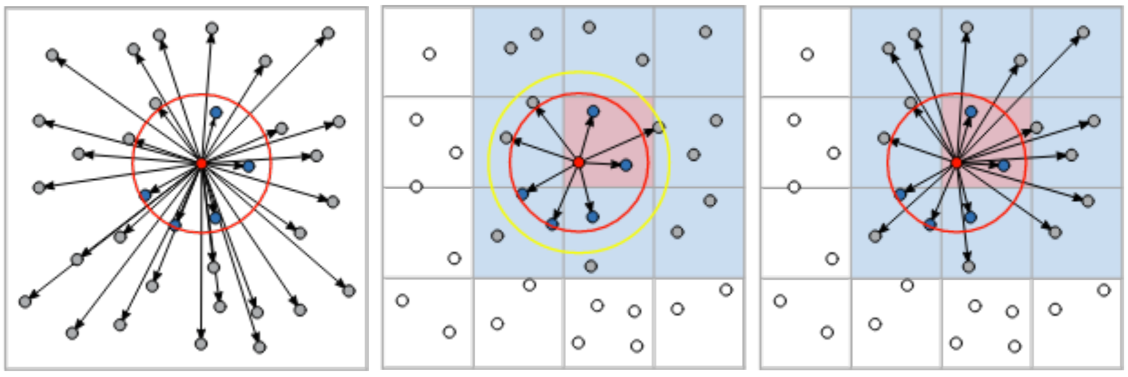


Figure 3.1.: Neighbor identification algorithms in pairwise force calculation. The red the circle symbolizes the cutoff radius. Particle colors indicate interaction intensity: Red: Particle for which interactions are calculated. Blue: Particles inside the red-colored particle's cutoff radius. Gray: Potentially in range of the red particle; therefore, calculating the distance. White: particles that are out of range and which are, thus, omitted.[GSBN22]

3.1. Direct Sum

Direct sum (Figure 3.1a) is the most straightforward approach to identifying relevant neighbor combinations and requires no additional memory or computational overhead. The algorithm iterates the entirety of N^P particle combinations in a domain with N particles and combinations with P particles to check the validity of distances in terms of the cutoff radius, resulting in $O(N^P)$ distance checks. In the context of three-body interactions, this algorithm would be in the order of $O(N^3)$. Therefore, it is only usable for small systems regarding particle number.

3.2. Verlet Lists

The Verlet list algorithm (Figure 3.1b) is an approach for particle interaction modeling that does not rely on spatial information but instead on storing interaction partners of particles in Verlet or neighbor lists. These lists include references to all particles within a specified cutoff range. However, generating these neighbor lists is computationally expensive, so the reuse across multiple simulation iterations enhances efficiency. The lists include particles slightly outside this range, covered by an extended radius called the Verlet skin, to sustain simulation accuracy between the list rebuilds. The skin thickness is positively correlated to the number of iterations till the rebuild of the lists. This method reduces the number of particle distance computations required, as one only needs to calculate distances for particles within the Verlet skin, offering a significant computational advantage over evaluating all particle combinations in neighboring cells, as done in the linked cells algorithm (Section 3.3). The algorithm uses the spatial information of linked cells to build the neighbor lists. Otherwise, the Verlet list would fall back to the direct sum approach.

Despite its advantages in reducing computational load, the Verlet Lists approach has notable drawbacks. It has a significant memory footprint, as it requires storing a list of particle references for each particle. Moreover, there is a lack of data locality, meaning the spatial information in the lists is insufficient to determine if two particles are close in memory. This lack of information can lead to inefficient memory access patterns, negatively affecting cache performance and challenging efficient vectorization.[GSBN22] This algorithm is not the focus of this work, but it is still relevant for performance analysis (Subsection 6.1.1).

3.3. Linked Cells

The linked cells (LC) algorithm, shown in Figure 3.1c, divides the simulation domain into a Cartesian grid, with each cell's dimensions equal to or greater than the specified cutoff radius. Particles are assigned to these cells, and the algorithm identifies neighboring particles by evaluating those within a specific cell and its adjacent cells.

A benefit of iteration and vectorization is that particles of the same cell are stored in the same data structure and end up close in memory. In a two-body model, this process notably reduces the computational complexity to $O(N)$ in scenarios with homogeneous particle distributions. A deficit of LC is the loss of particle relations. Compared to Verlet Lists, which initiate computations per particle, LC iterates per cell, meaning constant overhead loss is induced even for empty subdomain cells. [GSBN22]

Cell Indexing

When using the LC algorithm, using a 1D index representation as cell identifiers instead of coordinates can be beneficial. The purpose is to efficiently store and access elements in a multi-dimensional grid using a one-dimensional data structure like an array.

Having a 3D Cartesian grid with dimensions $dim_x \times dim_y \times dim_z$, where dim_x, dim_y, dim_z represent the number of cells along the x, y, and z-axis, respectively, the bijective mapping f from the 3D grid coordinates x, y, z of a cell to a unique 1D index is given by:

$$f = x * (dim_x^2) + y * dim_y + z \tag{3.1}$$

Additionally, the concept of a cell being *forward visible* is introduced, defined by the requirement that a cell possess a lexicographically linearized cell index greater than that of the current cell.

Cell Distance

We focus on cells within the cutoff radius to optimize force computation, excluding more distant cells. The critical measure is the distance between the nearest corners of different cells, representing the closest possible particle separation from separate cells. Assuming the cells are cubes with cell length l, the inter-cell distance function d is:

$$\begin{aligned} d(x_1, y_1, z_1, x_2, y_2, z_2) = &(\max(0, |x_1 - x_2| - 1) * l)^2 + \\ &(\max(0, |y_1 - y_2| - 1) * l)^2 + \\ &(\max(0, |z_1 - z_2| - 1) * l)^2 \end{aligned} \tag{3.2}$$

3.3.1. Cell Size Factor

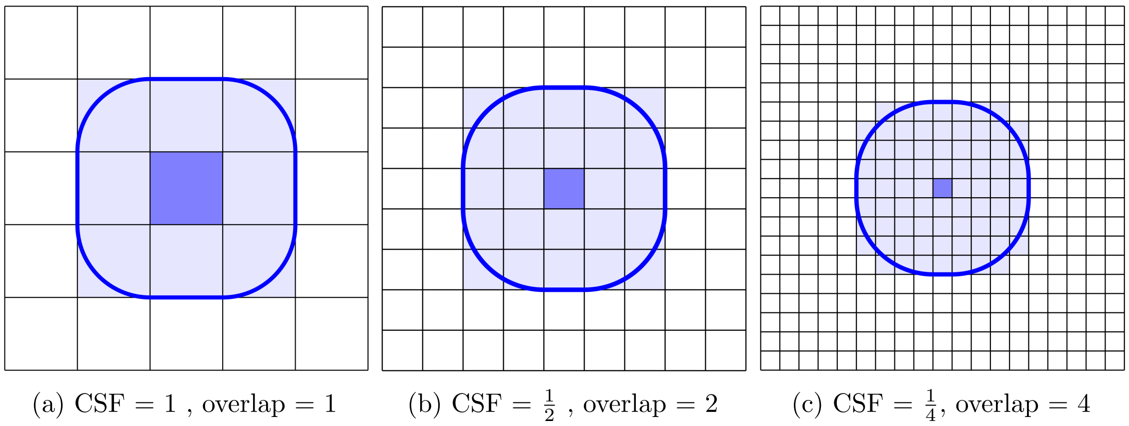

(a) CSF = 1 , overlap = 1 (b) CSF = $\frac{1}{2}$, overlap = 2 (c) CSF = $\frac{1}{4}$, overlap = 4

Figure 3.2.: Increase in overlap leads to a better approximation of the interaction circle. [Men19]

Menges [Men19] shows that smaller cell sizes than the cutoff radius can shift the interaction sphere (IS), approximated by cells, from a square shape to a more circular form. This approach impacts the searching space for neighboring particles since cells outside the sphere can be left out entirely. The cell size factor (CSF) governs this optimization, and it determines the number of cells in each direction within the cutoff radius, called overlap:

$$overlap = \left\lceil \frac{1}{CSF} \right\rceil \tag{3.3}$$

Here, the product of the CSF and the cutoff radius, $CSF * r_{cutoff}$ determines the absolute cell length. An optimal approximation of the IS theoretically requires an infinite number of cells. This fact raises the question of at what point the overhead maintaining the cubically growing number of cells would outweigh the force computations saved. [Men19]

Hit Rate by Volume

The hit rate is a critical metric for the LC algorithm, denoting the likelihood of a particle within the search space being a neighboring particle. This rate is instrumental in identifying the relative share of insignificant search space, which would cause distance checks, data fetch, and iteration overhead.

Since the cells within the cutoff radius (Figure 3.2) determine the search space, the ratio of volumes of the actual interaction sphere V_{real} to the volume approximated by the grid cells V_{approx} is the hit rate R of pairwise force computations:

$$R = \frac{V_{\text{real}}}{V_{\text{approx}}} \tag{3.4}$$

The ratio over different overlaps is plotted in Figure 3.3 as the red line, approaching one as the approximation gets more accurate and spherical as the overlap increases.

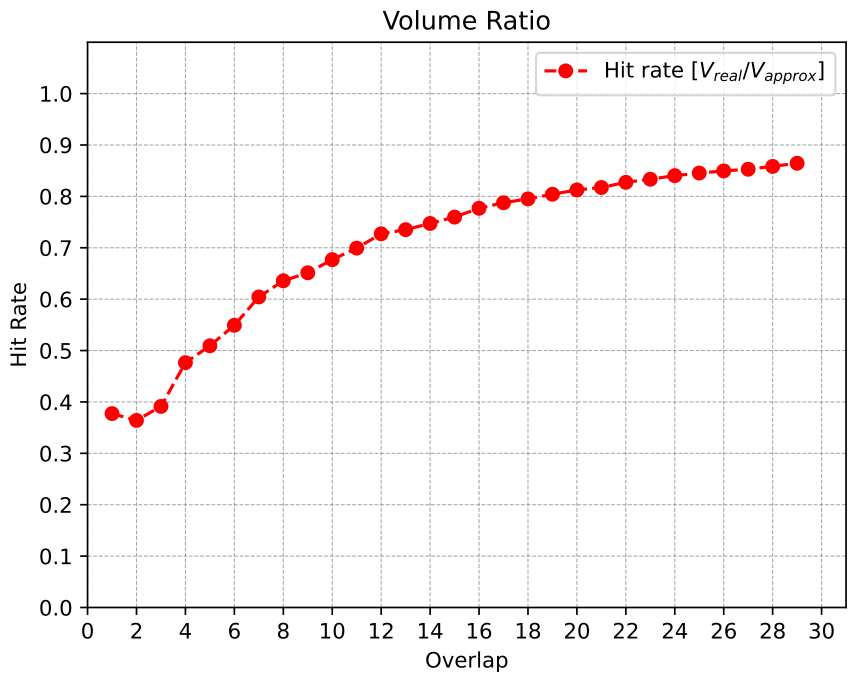


Figure 3.3.: Hit rate by cell volume for different overlaps. An increase in overlap leads to a better approximation of the interaction sphere.

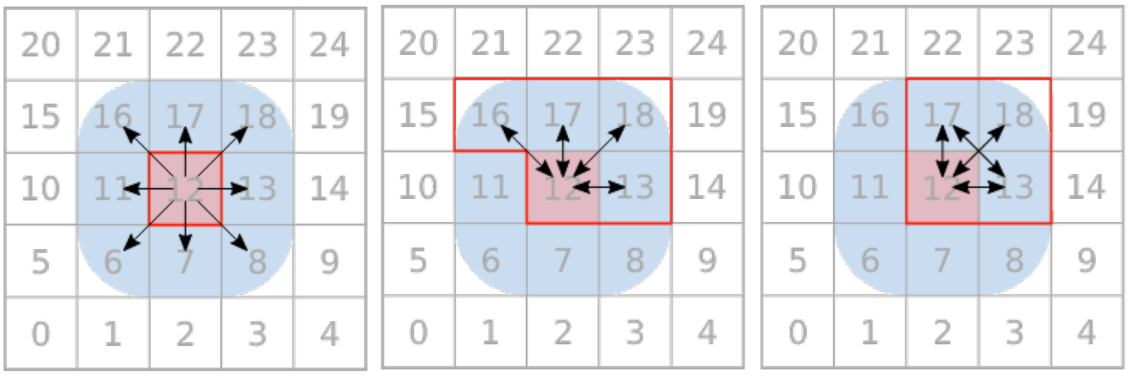
3.4. Traversals

Next, this section discusses traversal methods for the linked cells (LC) algorithm implemented in AutoPas. It includes an overview of the foundational base steps and the various traversal techniques built upon them. These traversal strategies will be either newly implemented or modified to be able to compute three-body interactions, which Chapter 5 presents.

The isolation of groups of cells in the LC domain into subdomains allows parallel computation, in which a single thread updates the forces of particles inside the subdomain to prevent update conflicts. A thread reserves a cell (base cell), and its adjacent cells are locking them from other threads. With this domain partitioning, the degree of parallelism achievable depends on the number of simultaneous processable base cells.

A traversal defines the scheme for dividing the domain into subdomains and the iteration over these while ensuring thread safety. A traversal iterates every cell once and performs a base step on the cell (base cell). A base step calculates forces for particles in a base cell, which affects particles in adjacent cells with distances less than the cutoff radius to the base cell. A base step defines the pattern of neighboring cells for force calculations and whether the adjacent cell's particles are only read from or updated using Newton's third law of motion.

3.4.1. Base Steps



(a) C01 (b) C18 (c) C08

Figure 3.4.: Base steps. The red cell is the base cell. The blue area is the interaction sphere (cutoff radius). The zone surrounded by the red line is isolated from other threads.[GSBN22]

C01 Base Step

The C01 base step (Figure 3.4a) calculates two-body interactions between particle pairs inside of the current base cell and between particles of the base cell and its adjacent cells. Due to the base step's symmetry regarding interacting directions, it does not use the Newton3 optimization. It only reads from the neighboring cells, calculates the forces, and updates only particles in the base cell. Otherwise, forces would be updated repetitively for a cell every time an adjacent cell is a base cell. Limiting the writing operations to only the base cell has the benefit of theoretically being able to scale up to as many threads as there are cells.[GSBN22]

C18 Base Step

The C18 base step (Figure 3.4b) is a product of applying the Newton3 optimization. This optimization reduces the computational load of pairwise interactions by half, which is accomplished by restricting force calculations to cells in the forward-visible direction from the current base cell, forming a $2 \times 2 \times 3$ block in 3D space.[GSBN22]

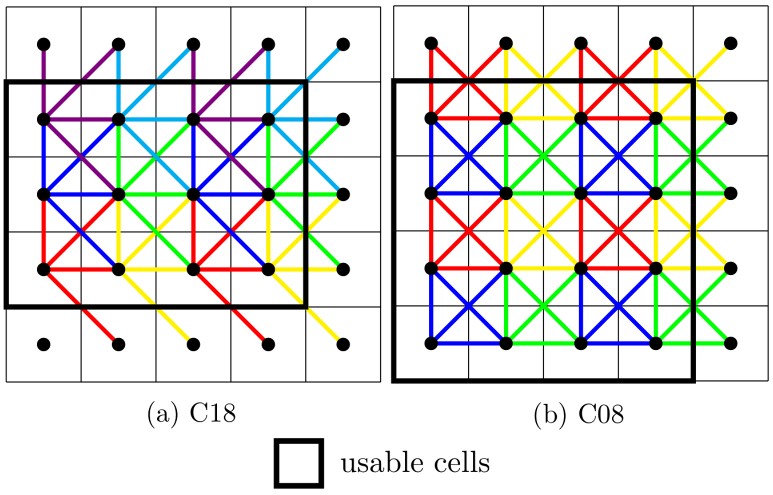
Since these forward-visible cells are updated using the Newton3 method, they need to be safeguarded against other operations performed on the base cell, as illustrated in Figure 3.4b. This protection leads to a lower level of parallelism compared to C01. Reducing the total number of cells required for one base cell increases the likelihood of maintaining all relevant particles in the lower-level cache throughout the base step. [GSBN22]

C08 Base Step

The C08 base step enhances efficiency by building on the concept introduced in the C18 base step. It substitutes the forward diagonal interaction computation (as exemplified in Figure 3.4c with cells 12 and 16) with the forward computation involving the subsequent cells (in this case, cells 13 and 17). This modification effectively reduces the area of local interaction, leading to increased parallelism and more effective cache reuse. [GSBN22]

3.4.2. Domain Coloring

Domain coloring models, which packages of cells can be worked on in parallel by coloring them the same.



[Men19]

Figure 3.5.: Domain coloring of C18 and C08.

C18

Figure 3.5 shows that the C18 base step stacks two more base steps on top and one more base step to the side to reuse the same color. This results in six colors in 2D and 18 in 3D. [GSBN22]

C08

The C08 traversal needs eight colors for a $2 \times 2 \times 2$ base step cubes since all neighboring cells in this package can not be processed in parallel (Figure 3.5). One Base step pattern needs to be stacked in each forward base vector direction, meaning the number of colors C of dimensions D is:

$$C = 2^D \tag{3.5}$$

C18 and C08 can improve cache efficiency since the cell packages for a base step are fetched freshly every time without synchronization between steps, even though operating on the same cells. The lack of synchronization is often acceptable because the main computation calculates forces. [TSH$^+$19]

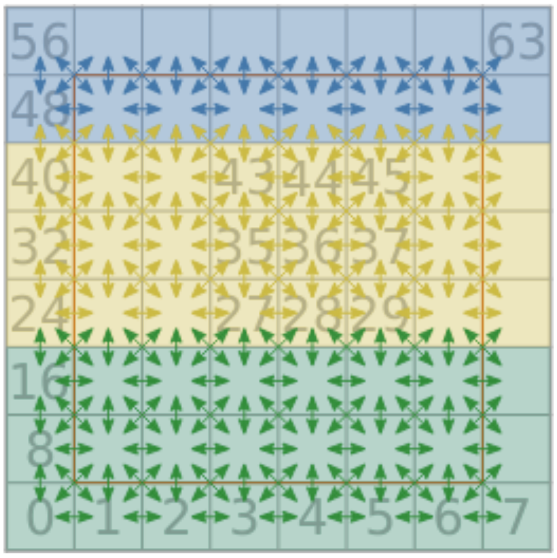


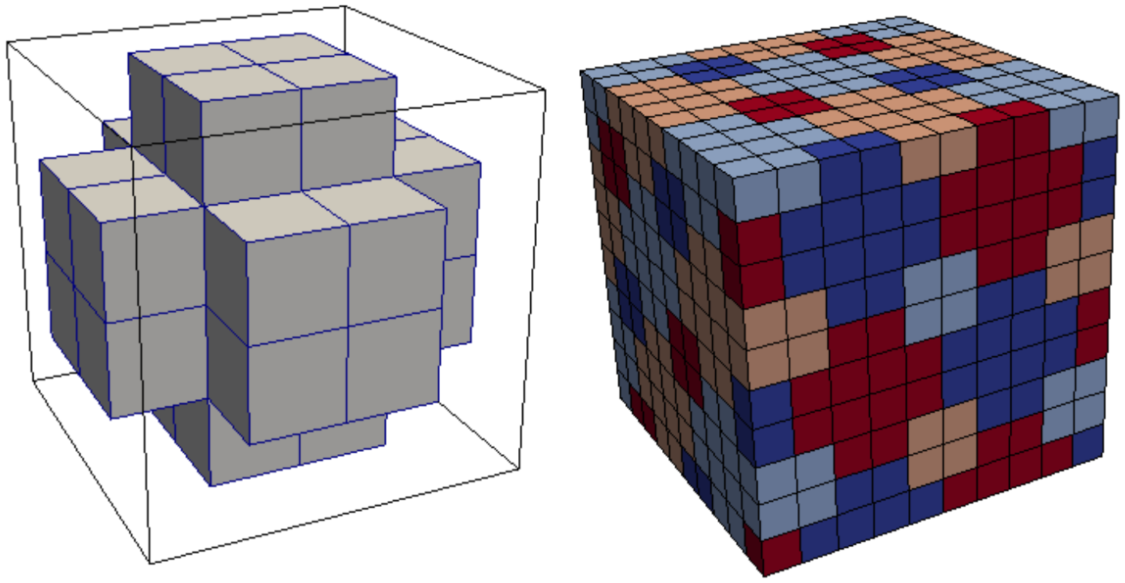
Figure 3.6.: Sliced traversal 2D with 3 colors. The third slice gets fewer cells due to domain size.[GSBN22]

The sliced traversal approach in AutoPas focuses on reducing scheduling overhead by statically assigning cells to threads. This method, illustrated in Figure 3.6, involves dividing the simulation domain into equal-sized slices corresponding to the number of threads, with at least two layers of cells in the sliced dimension. The C08 base step processes every cell of a slice. Threads lock the first layer of cells in their slice and release the lock once all cells in that plane are processed. This process is repeated for each layer, minimizing lock contention and synchronization costs. Cells are processed first in directions perpendicular to the slicing dimension. This strategy enables quicker release of locks and reduces waiting times between threads. While this traversal lacks load balancing mechanisms, it performs well in homogeneous scenarios with sufficient cell layers, thanks to minimal scheduling overhead. Specifically, having many halo cells causes load imbalance since the interactions with them are skipped. [GSBN22, TSH^{+}19]

Sliced With Two Colors

An approach in [Fis20] introduces the exchange of the static scheduling of relatively thick slices by the dynamical scheduling of slices with the thinnest possible thickness, respecting the interaction length. This dynamic method smooths potential load imbalances with a smaller granularity of slices.

C04



(a) 32 cells pattern of C04 (b) Domain coloring of C04.

Figure 3.7.: Cell pattern and domain coloring used in the C04 traversal.

The C04 traversal requires an intermediate amount of synchronization overhead compared to C08 and Sliced. It offers lower dynamic scheduling overhead than C08, given by its pattern of 32 cell packages in 3D (Figure 3.7). Furthermore, the C04 introduces a color-to-dimension dependency of $D + 1$ for dimensions greater than two, which could offer more significant gains over c08 in higher-dimensional applications. [TSH+19]

4. Description Of Tools

4.1. AutoPas

AutoPas, part of the TalPas project, is an open-source C++ library that introduces algorithmic autotuning to molecular dynamics (MD) simulations. It addresses two main challenges: adapting to dynamic changes in simulation behavior and simplifying usage for researchers without extensive computer science expertise. AutoPas automatically adjusts simulation parameters for optimal performance and offers an easy-to-use interface. Designed for parallel execution and optimized for node-level performance, AutoPas uses statistical methods to select the best configurations for efficient simulations. While not built for multi-node operations, it can integrate with larger simulation frameworks like ls1 mardyn.

4.2. MD-Flexible

MD-Flexible is a particle simulation tool built upon the AutoPas library. It features object generators for creating 3D shapes filled with particles and a velocity-scaling thermostat to manage the system's total kinetic energy. A simulation configuration is done through a YAML file, which details the objects to be simulated and various options passed to AutoPas. These options include settings for domain size, interaction potential, cell size, density, iterations, containers and traversal methods, and the quantity and regularity of tuning iterations.

4.3. CoolMUC-2

The CoolMUC-2 system at the Leibniz Supercomputing Center (LRZ) is a Linux cluster with 812 nodes connected through FDR14 Infiniband. Each node is equipped with Intel Haswell Xeon E5-2697 v3 processors, which support AVX2 for 256-bit vector operations and have a Level 3 cache of 2.5 MB per core.

The system uses a 64-bit address space with 28 cores per node, each capable of two threads through hyperthreading. OpenMP 4.5 is used for shared memory tasks for parallel computing, and Intel MPI 2019.12 for distributed memory tasks. The development environment for all benchmarks of this thesis uses CMake 3.21.4 for generating Makefiles and GCC 11.2.0 for program compilation.

5. Implementation

This chapter presents the implementational details for three-body linked cell traversals. We first review the core implementation of the adapted C08 base step, including some optimizations. In Section 5.2, we will extend the implementational idea to n-body base steps. Finally, Section 5.3 will detail the integration of the base step into three-body traversal schemes.

5.1. Three-Body C08 Base Step

As introduced in subsubsection 3.4.1, the C08 base step results from integrating the Newton3 optimization while maintaining cache locality. Since three-body interactions require finding up to three cells, the complexity of a naive approach is in the realm of $O(M^3)$, with $M = (overlap + 1)^3$ cells in a C08 base step cube. The C08 base step aims to cover all cell triplets with particle distances less than the interaction length after a domain traversal. The first approach is to recognize critical traits of the C08 base step and apply them to three-cell combinations.

A requirement for the C08 base step triplets is that they do not contain cell combinations covered by another base step. An intuitive approach is to find these combinations and subtract them from all $\binom{M}{3}$ triplets possible in the current base step cube. The remaining set of triplets is relevant for C08.

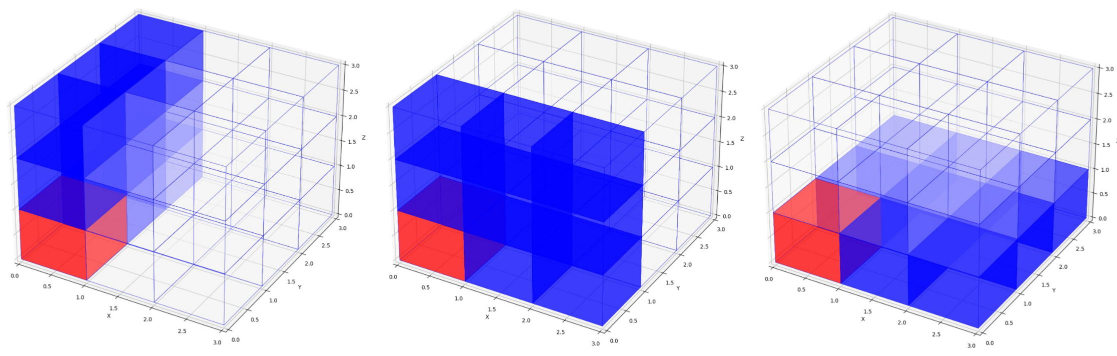
The combinations to be left out are all triplets in smaller rectangular cuboids, not containing the base cell, fitting into the current base step cube. Other base steps cover cell triplets within these cuboids. Since all triplets of a cuboid have all triplets of the cuboids inside of it, we can unify the triplets of the three biggest possible cuboids to get the set of triplets to remove. These three biggest cuboids can be interpreted as the base step zones initiated by the direct neighbors in positive x, y, and z direction, intersecting into the current base step zone. Each of the neighbor cuboids is the current base step cube missing either $(x, y, 0)$-plane, $(x, 0, z)$-plane or $(0, y, z)$-plane, illustrated in Figure 5.1. This geometrical knowledge translates to filtering all triplets with either xs, ys, or zs all non-zero.

To define this formally, let $C = \{(x, y, z) \mid x, y, z \in [overlap + 1]\}$ be the set of cell coordinates and C^3 all triplet cell combinations in coordinate format. The set of triplets $X \subset C^3$ to remove to get the C08 triplets is defined as:

$$\begin{aligned} X = \{((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)) \mid &(x_1 \neq 0 \ \wedge \ x_2 \neq 0 \ \wedge \ x_3 \neq 0) \ \vee \\ &(y_1 \neq 0 \ \wedge \ y_2 \neq 0 \ \wedge \ y_3 \neq 0) \ \vee \\ &(z_1 \neq 0 \ \wedge \ z_2 \neq 0 \ \wedge \ z_3 \neq 0)\} \end{aligned} \tag{5.1}$$

Finding the complementary set to the C08 triplets becomes inefficient due to the cubic growth of the total amount and the triplets to extract. By defining a condition for the complementary set X, De Morgan's second law $\neg(P \vee Q) = \neg P \wedge \neg Q$ can be applied to get the condition for the C08 triplets:

$$\begin{aligned} C^3 \setminus X = \overline{X} &= \{((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)) \mid \neg\,((x_1 \neq 0 \;\wedge\; x_2 \neq 0 \;\wedge\; x_3 \neq 0) \vee \\ &\qquad (y_1 \neq 0 \;\wedge\; y_2 \neq 0 \;\wedge\; y_3 \neq 0) \vee \\ &\qquad (z_1 \neq 0 \;\wedge\; z_2 \neq 0 \;\wedge\; z_3 \neq 0))\} \\ &= \{((x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)) \mid (x_1 = 0 \;\vee\; x_2 = 0 \;\vee\; x_3 = 0) \wedge \\ &\qquad (y_1 = 0 \;\vee\; y_2 = 0 \;\vee\; y_3 = 0) \wedge \\ &\qquad (z_1 = 0 \;\vee\; z_2 = 0 \;\vee\; z_3 = 0)\} \end{aligned} \tag{5.2}$$



(a) $(0, y, z)$-plane (b) $(x, 0, z)$-plane (c) $(x, y, 0)$-plane

Figure 5.1.: C08 base step cube at CSF=0.5 or overlap=2. The red cell is the base cell. Valid C08 cell triplets require one cell from each plane.

The negated condition means a C08 triplet has to have cells on all base cell planes. It's not necessary for all cells to be on the planes; for example, the base cell alone is the intersection of all three planes and suffices for a valid combination.

5.1.1. computeOffsets()-Function

In AutoPas, the `computeOffsets()`-function implements the triplets generation for a given overlap and stores the relative triplet offsets, which are then used at a later point to calculate the actual offsets for each base cell. It is within the `LCC08CellHandler3B`-class, which is the basis of all traversals introduced in Subsection 3.4.2, except for C18. For a complete force calculation, we must consider that particles may be distributed over one to three cells. The triplet, which consists of three times the base cell, represents the scenario of three particles being in one cell. Cases where the particles are distributed over two cells c_1, c_2 are added as triplet (c_1, c_1, c_2) or (c_2, c_2, c_1). The $CellFunctor$ handles such a triplet bidirectionally. We can apply the logic in section Section 5.1 for combinations with all particles in different cells.

To effectively apply the CSF optimization parameter as outlined in Subsection 3.3.1, it's necessary to evaluate the distances between cells. When dealing with triplets, this involves calculating three separate inter-cell distances (subsubsection 3.3). Should any of these distances surpass the cutoff radius, the corresponding triplet is excluded from the force computation process. This rule implies that the search for the third cell in a triplet is confined to the overlapping area of the interaction spheres from the first two cells, illustrated in Figure 5.2.

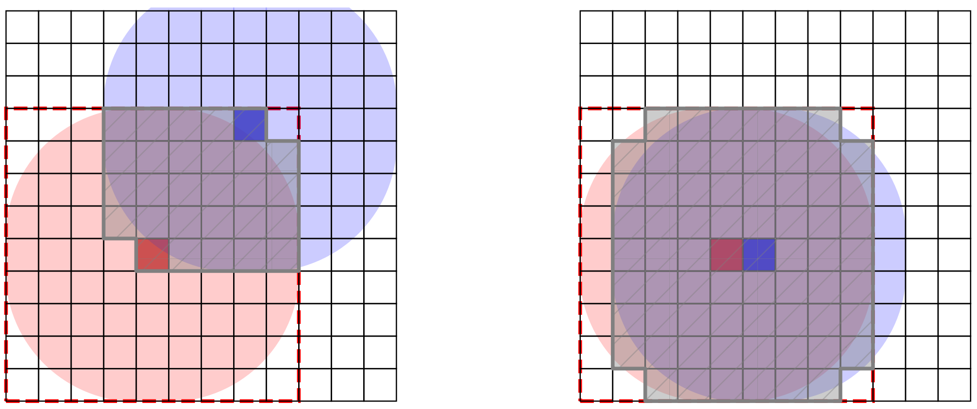


Figure 5.2.: Example for the smallest and largest area for the third cell, depending on the first two cells. The grey-shaded area is the approximated intersection of two interaction zones (red and blue). The red dashed line shows the current base step zone. Implemented by checking that all three cell distances are less than the cutoff radius.

For simplicity, distance checks for whether the distance between the two closest corners of two cells is within the interaction length are omitted. Besides, in this approach, *overlap* and *dimension* have the same length in x, y, and z direction. Algorithm 1 iterates over all possible triplets and stores the triplets, which fulfill the condition in Section 5.1 and which have ascending indices to ensure having no permutations.

Algorithm 1: C08 Triplets

```
Input:   overlap, dimension as ov, dim
Output:  C08 base step triplets

// Map coordinates to index
Function threeDToOneD($x$, $y$, $z$):
    return $(x * dim + y) * dim + z$

// C08 triplet condition
Function is_base_plane_triplet($x_1$, $y_1$, $z_1$, $x_2$, $y_2$, $z_2$, $x_3$, $y_3$, $z_3$):
    return $(!x_1 \mid\mid !x_2 \mid\mid !x_3)$ && $(!y_1 \mid\mid !y_2 \mid\mid !y_3)$ && $(!z_1 \mid\mid !z_2 \mid\mid !z_3))$

// Compute C08 triplet offsets
Function computeOffsets($ov$):
    $offsets = [0, 0, 0]$
    for $x_1 \leftarrow 0$ to $ov$ do
        for $y_1 \leftarrow 0$ to $ov$ do
            for $z_1 \leftarrow 0$ to $ov$ do
                for $x_2 \leftarrow 0$ to $ov$ do
                    for $y_2 \leftarrow 0$ to $ov$ do
                        for $z_2 \leftarrow 0$ to $ov$ do
                            for $x_3 \leftarrow 0$ to $ov$ do
                                for $y_3 \leftarrow 0$ to $ov$ do
                                    for $z_3 \leftarrow 0$ to $ov$ do
                                        $i_1$ = threeDToOneD($x_1$, $y_1$, $z_1$)
                                        $i_2$ = threeDToOneD($x_2$, $y_2$, $z_2$)
                                        $i_3$ = threeDToOneD($x_3$, $y_3$, $z_3$)
                                        if $i_1 <= i_2 < i_3$ then
                                            if is_base_plane_triplet($x_1$, $y_1$, $z_1$, $x_2$,
                                             $y_2$, $z_2$, $x_3$, $y_3$, $z_3$) then
                                                $offsets$.append($i_1$, $i_2$, $i_3$)
    return offsets
```

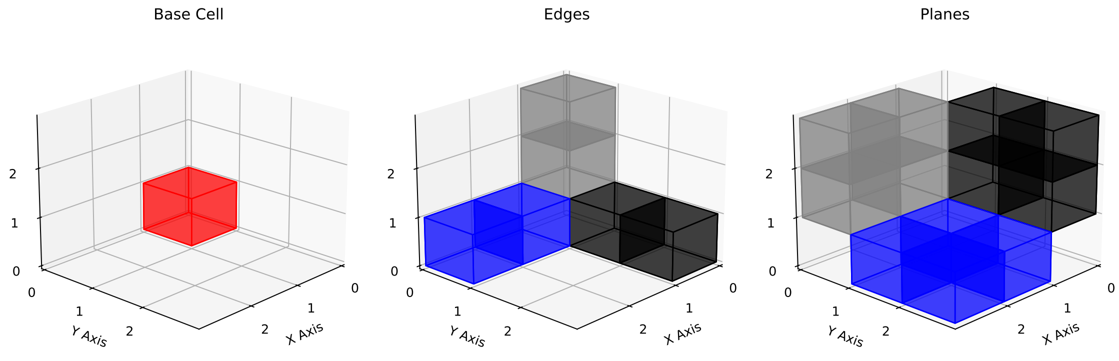
Figure 5.3.: computeOffsets() for triplets in base step cube with $(overlap + 1)^3$ cells.

5.1.2. Complexity Optimization for computeOffsets()-Function

Although in the more extensive picture of a three-body simulation, the overhead for offset generation takes only a tiny part, still the naive approach in Algorithm 1 has a complexity of $O((overlap + 1)^9)$. In AutoPas, every simulation iteration for the C08 traversal calls the function, which causes a constant accumulation in computation time.

Next, the goal is to optimize the $computeOffsets$-function by making use of the condition that all triplets have to be part of the three base planes. This idea means that instead of iterating all triplets in the cube and filtering, we can combine triplets from groups of cells, which together fulfill the C08 condition. Cells are grouped by the base planes they are part of:

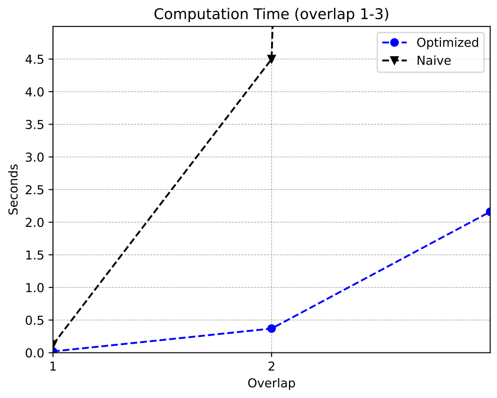
1. The intersection of all three base planes is the base cell, which can form triplets with any two other cells (Figure 5.4a).
2. The intersection of two planes, excluding the base cell, is on the edge between the respective planes. These cells require at least one cell on the missing plane for a valid triplet (Figure 5.4b).
3. The cells only part of one base plane, can form valid triplets with two cells, which also are only part one of the two other base planes, respectively, or these cells can be in triplets covered in the cases mentioned above (Figure 5.1).
4. Cells, which are not on any base plane, need the other two cells of the triplet to be on all three base planes, also covered in the combinations described in points 1 and 2.



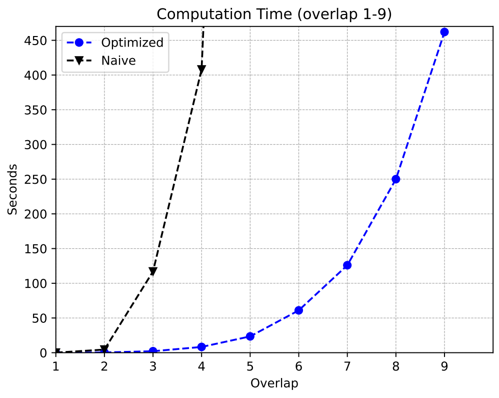
(a) Cells part of all base planes (b) Cells part of two base planes (c) Cells part of one base plane

Figure 5.4.: Cells grouped by the number of base planes they are part of. C08 triplets can be built bottom-up by combining cells into triplets that touch all three base planes.

With this geometrical knowledge, the loop structure in Algorithm 1 with a depth of nine loops can be enrolled to more shallow structures with six loops at the deepest point.



(a) Overlap 1-3



(b) Overlap 1-9

Figure 5.5.: Time to compute offsets with one thread on CoolMUC-2 in 1000 AutoPas iterations for different overlaps.

Figure 5.5 illustrates the accumulated times measured in 1000 C08 traversal iterations for both approaches with one thread on CoolMUC-2. The approaches confirm the expected differences in complexity, with the optimized times for overlaps up to four being almost negligible.

5.2. N-Body C08 Base Step

Since the necessary condition for a combination is to have cells on all three base cell planes, this logic extends to any N-cell combination, with N being the number of cells in a combination. Trivially, N has to be equal to or less than the number of cells M in the current base step cube. This algorithm does not currently have a use case in AutoPas but can easily be added using the following approach.

We can define the set of C08 tuples C_{C08}^N for the combination space of C^N with $C = \{(x,y,z) \mid x,y,z \in [overlap+1]\}$ as:

$$\begin{aligned} C_{C08}^N = \{((x_1,y_1,z_1),\dots,(x_n,y_n,z_n)) \mid\ & (x_1 = 0\ \vee\ \dots\ \vee\ x_n = 0) \wedge \\ & (y_1 = 0\ \vee\ \dots\ \vee\ y_n = 0) \wedge \\ & (z_1 = 0\ \vee\ \dots\ \vee\ z_n = 0)\} \end{aligned} \tag{5.3}$$

To account for situations where particles are distributed across fewer than N cells, one can add the relevant combination by duplicating any cell within that combination to make up the difference, which is identical to the approach used in the triplet method described in section 5.1.1. To maintain order, in the following, the smallest cell index will be repeated to fill up the combination.

This logic (Equation 5.3) can be formalized recursively with memory and compute complexity of $O(M^N)$ as:

Algorithm 2: C08 N-Body Cell Combinations

```
Input:  $overlap$, $dimension$ as $ov$, $dim$
Output: C08 base step combinations

// Map coordinates to index
Function threeDToOneD($x$, $y$, $z$):
    return $(x * dim + y) * dim + z$

// C08 condition
Function is_base_plane_combination($xs$, $ys$, $zs$):
    return
     $any$([!$x$ for $x$ in $xs$]) and $any$([!$y$ for $y$ in $ys$]) and $any$([!$z$ for $z$ in $zs$])

// Recursive part
Function n_cell_combinations($n$, $xs$, $ys$, $zs$):
    $offsets$ = []
    for $x \leftarrow 0$ to $ov$ do
        for $y \leftarrow 0$ to $ov$ do
            for $z \leftarrow 0$ to $ov$ do
                // distance check (omitted)
                $indices$ = []
                for $i \leftarrow 0$ to $xs$.size() $- 1$ do
                    $indices$.append(threeDToOneD($xs[i]$, $ys[i]$, $zs[i]$))
                $index$ = threeDToOneD($x$, $y$, $z$)
                // avoid same combination in different orders
                if not is_strictly_ascending([$index$].concat($indices$)) then
                    continue
                $xs$.append($x$)
                $ys$.append($y$)
                $xs$.append($z$)
                if is_base_plane_combination($xs$, $ys$, $zs$) then
                    // fill current index $N$ times
                    $fillingIndices$ = []
                    for $i \leftarrow 0$ to $n - 1$ do
                        $fillingIndices$.append($index$)
                    $combination$ = $fillingIndices$.concat($indices$)
                    $offsets$.append($combination$)
                $offsets$.concat(n_cell_combinations($n - 1$, $xs$, $ys$, $zs$))
    return offsets

// Call recursive function
Function computeOffsets():
    return n_cell_combinations($n$, [], [], [])
```

Figure 5.6.: computeOffsets() for combinations for N-Body interactions in base step cube with $(overlap + 1)^3$ cells.

5.3. Three-Body Traversals

In Subsection 3.4.2, C08-based traversals with traversal schemes were introduced, which differ in coloring, cache reuse between steps, and scheduling. These traversals for three-body interactions, C08, C04, Sliced, and SlicedC02, were implemented based on code for two-body interactions. The implemented classes are named: `LCC08Traversal3B`, `LCC04Traversal3B`, `LCSlicedTraversal3B`, `LCSlicedC02Traversal3B`. The logic of the traversals, which they have in common, is the iteration of the domain in their respective strides. Within the strides, these traversals all process the cells by applying the C08 base step. The underlying class `LCC08CellHandler3B`, which is at the core of each of the mentioned traversal classes, performs the base step. Cell triplets indices are computed for each base step and processed by the `CellFunctor3B`. `CellFunctor3B` iterates all significant particle triplets for a given cell triplet and invokes the suiting call of `AxilrodTellerFunctor` under consideration of the Newton3 optimization, data layout, and more. At last, the `AxilrodTellerFunctor` checks the particle distances and calculates the forces.

6. Performance

This chapter evaluates the performance of three-body traversals on a node level, considering homogeneous particle distributions with variable density and cell size factors. It also includes an analysis of an inhomogeneous simulation scenario involving a falling drop. The study compares the C01 and C08 traversal methods and variations within C08-based traversals. The performance metrics under evaluation encompass time per iteration, giga-FLOPs per second (GFLOPs/s), and hit rate.

6.1. Homogeneous Particle Distribution

6.1.1. Simulation Setup

Establishing a setup where each method can perform to its full potential is essential to ensure a fair and compelling comparison of traversal methods within a simulation. The performance reaches its full potential when evenly distributed among the available threads. This analysis aims to equalize the workload for each thread while employing up to 56 threads on the cm2_tiny-cluster of CoolMUC-2, which corresponds to the total capacity of a node of 28 cores, including its hyperthreading capabilities. This approach guarantees sufficient subdomains or base steps, precisely at least 56, to utilize all threads fully.

A cutoff value of 2.5 is selected to mirror real-world applications. In AutoPas, the Verlet list algorithm (Section 3.2), based on Linked Cells, allows for configuring a Verlet skin around the cells. This configuration consists of a verlet-skin-per-timestep of 0.02 and a verlet-rebuild-frequency of 10. This feature also enhances the performance of the LC algorithm, as it decreases the frequency of updates for particles changing cells to match the rebuild frequency. Consequently, for a CSF of 1.0, the absolute cell length is calculated as $2.5 + (0.02 * 10) = 2.7$.

Using a box size of $13.5 \times 13.5 \times (13.5 \times 56) = 13.5 \times 13.5 \times 756$, which corresponds to $5 \times 5 \times 280 = 7000$ inner cells plus 6818 halo cells at a CSF of 1.0, ensures that the `Sliced` traversal method divides the domain along its longest axis into $(280 + 2)/2 = 141$ slices, with the number two as number of halo cells and slice thickness respectively. The traversals `Sliced02` and the `C04` produce 282 slices and 142 packs of 32 cells for each color, respectively. The choice of this domain size allows analysis of weak scaling just by enlarging the domain into one direction by a $5 \times 5 \times 5$ cell block. This 125-cell block per thread ensures that the `C04` traversal gets four packs for each color, translating to 52 cells for two colors and 56 cells for the other two colors.

To provide context for the particle spacing parameter, the following figure 6.1 illustrates the calculated number of particles per non-halo cell within this setup. At a spacing of one, there are 180375 particles in the domain, with 25.76 per cell.

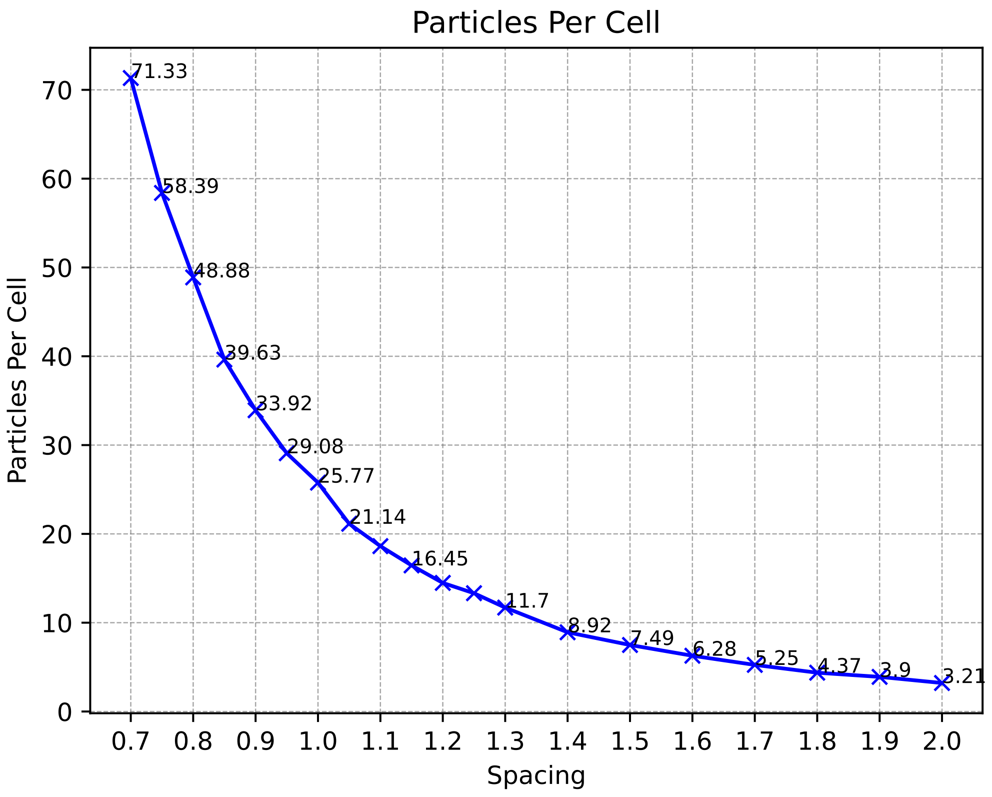
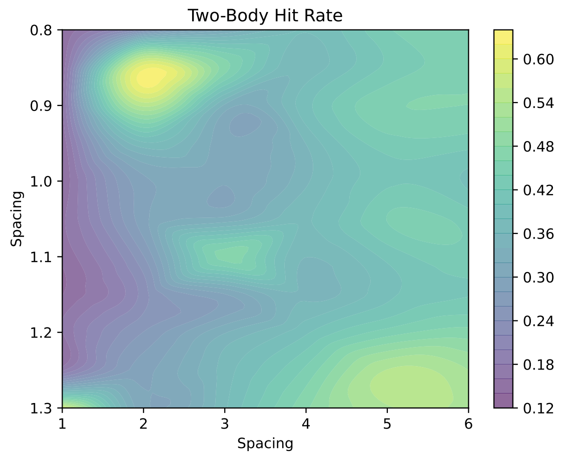


Figure 6.1.: Particles per cell over spacings 0.65 to 1.65.

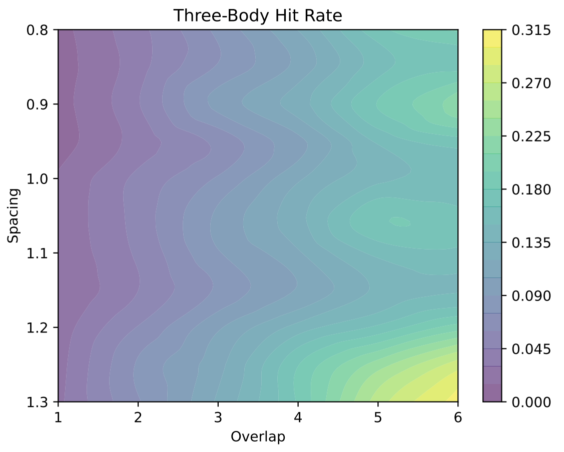
6.1.2. Particle Hit Rate

An empirical approach to determine the hit rate involves measuring two key aspects: the total number of particle triplets undergoing distance checks and the subset of these triplets relevant for force calculations. All traversal methods show an identical hit rate, attributed to their shared use of the Linked Cells (LC) data structure and identical distance check procedures.

The comparison of hit rates shown in figure 6.2, which represent the ratio of valid distance checks to the total number of distance checks for two-body and three-body interactions, reveals a consistent trend of increase in hit rates as the overlap increases. For two-body interactions, the hit rate in Figure 6.2a rises from 10% to 60%, displaying few unexpected spikes to the upside. In contrast, the three-body interaction hit rate spans a narrower range, from nearly 0% up to 30%, maintaining a steady, smooth progression across the entire plot, which can be seen in Figure 6.2b.



(a) Two-body hit rate

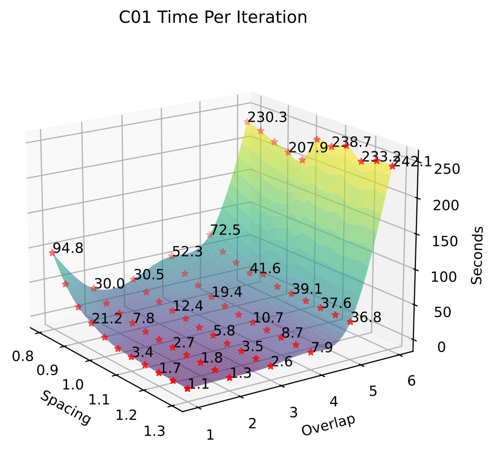


(b) Three-body hit rate

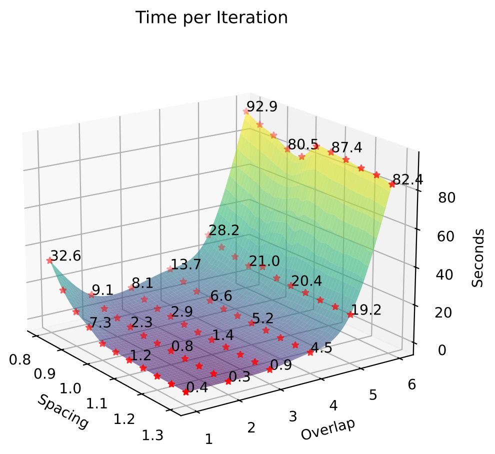
Figure 6.2.: Comparison of hit rates, the ratio of valid distance checks to the total distance checks for two-body and three-body interactions in AutoPas. Both plots show an uptrend of hit rates with an increase of overlap. The two-body hit rate ranges from 10% to 65%, whereas the three-body hit rate only ranges from close to 0% to 30%.

6.1.3. Comparison of C01 and C08

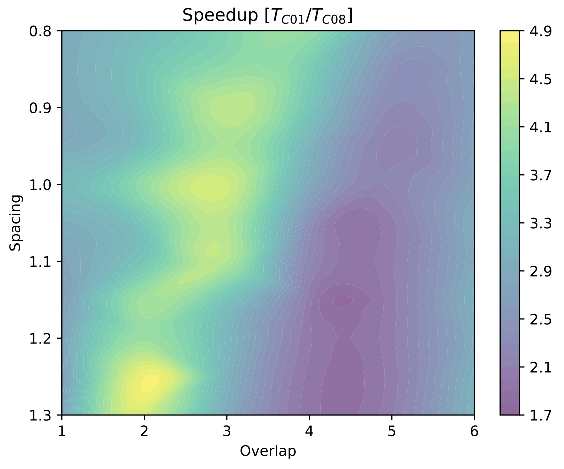
In the performance analysis between the C01 and C08 traversal methods, several key observations were made based on the provided plots, which show how metrics perform on multiple configurations of overlap and particle spacing. The figures 6.3 depicts the wall clock times for each iteration using 56 threads for both traversal methods, along with the speedup $\frac{T_{C01}}{T_{C08}}$ across various spacing and overlap settings.



(a) C01 time per iteration



(b) C08 time per iteration



(c) Speedup [T_{C01}/T_{C08}]

Figure 6.3.: C01 and C08 times per iteration. C08 shows speedup compared to C01, ranging from 1.8 to 4.8 across all overlap-spacing configurations. The greatest speedups form a diagonal pattern of configurations (highlighted yellow), offering the highest share of computation spent on force computations, which C08 leverages well with Newton3.

As spacing values rise, there is a noticeable decline in iteration time for both methods (Figure 6.3b and Figure 6.3a), which matches expectations due to the decreasing amount of particles in the domain and less force computations.

The plots displaying time per iteration for the C01 and C08 traversal methods highlight that the time required for each iteration proliferates as overlap surpasses four. Such a rise in computation time is anticipated due to the cubic increase in the number of cells for an increase in overlap. The higher amount of cells results in equally many base steps. The amount of cell triplet combinations processed in each base step increases in a manner shown in Figure 6.4.

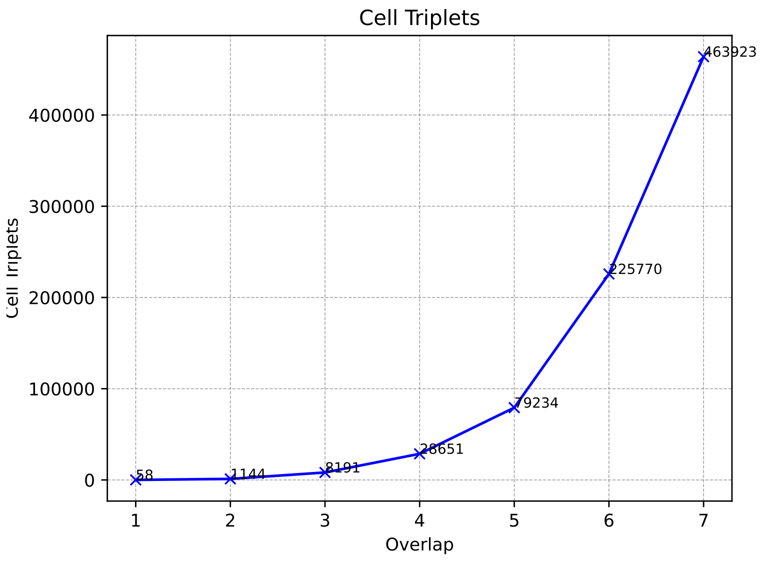


Figure 6.4.: Triplets per Overlap

Together, these factors considerably magnify the influence of the overlap value on computation time due to the administrative overhead of the cells. For overlap=6, the extra work needed to manage cells starts to outpace the part of force computation in the total computational tasks. At high overlaps, the spacing between particles has a diminishing impact on time per iteration, showing that changes in particle spacing don't much influence performance due to the high overhead from managing cells. Figure 6.3a and Figure 6.3b confirm this with roughly equal values for overlap=6. Assuming overlap is heading towards infinity, the time per iteration would mirror the time to traverse the whole domain of almost empty cells, with the share of force computation approaching 0%.

Furthermore, the observation that an overlap of one when paired with high-density configurations significantly increases the time per iteration since the high amount of particles amplifies the low hit rate at overlap=1 (Figure 6.2b), resulting in a lot of invalid distance checks.

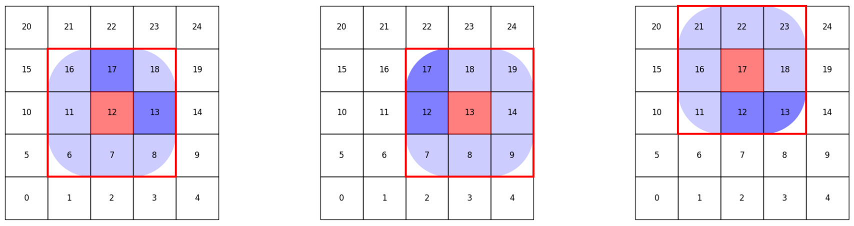
The described rises at both ends of overlap lead to the observation that there is an optimal overlap for each spacing, at which the gains through better hit rate outweigh the cell managing overhead the most. This optimal point in overlap shifts higher as spacing decreases, representing a high enough particle amount per cell thinning down the overhead while ensuring gains on hit rate, forming a *corridor* of optimal particles per cell, which is a diagonal pattern from low density and low overlap to those with high density and high overlap. Along this diagonal, the share of computation spent effectively on force computation is the highest since it reflects the maximum of saved distance checks while trying to keep the overhead little. This pattern can also be observed in Figure 6.3c and in the latter scaling analysis.

Upon comparing the two methods directly, the C08 traversal consistently exhibits a lower time per iteration and outperforms C01 across all tested overlap and spacing parameters. Figure 6.3c represents the ratio of the time per iteration of C01 to that of C08 at 28 thread and highlights that there are configurations where C01's performance significantly lags behind C08, as evidenced by the peaks in the plot. The speedup ranges from around 1.8 to 4.8, forming a slightly diagonal mountain pattern across the plot, matching the above-described pattern. Figure 6.3c confirms the assumption of the share of computation spent effectively on force computation being the highest on this diagonal since C08 outperforms C01 over these configurations the most, being able to leverage the Newton3 optimization.

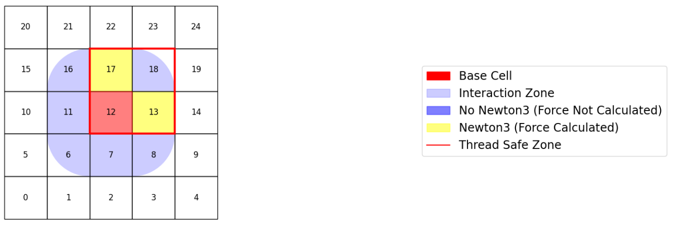
Comparison of FLOPs

In AutoPas, a separate force functor `FlopCounterFunctor3B` counts performed floating point operations (FLOPs) on particle distance calculations and force calculations. One distance calculation between two particles has the value of 8 FLOPs. A force calculation by the `AxilrodTellerFunctor` to compute the force for a single particle in a triplet costs 57 FLOPs, resulting in 171 FLOPs without distance checks a triplet. By applying Newton's third law of motion, the FLOPs count for only force calculation of a triplet is reduced to 97, resulting in a speedup of approximately 1.82 purely for force calculations.

Given that the C01 traversal method updates and locks particles only within the current base cell, it requires invoking the force calculation for one particle triplet thrice, each time with a different cell serving as the base cell, shown in figure 6.5. As a result, C01 causes recalculating distances three times as frequently, corresponding to each of these separate invocations. On the contrary, C08 does not revisit cell combinations in the same iteration from different base cells, and distances are checked once for each triplet in cases where all particles are in separate cells. For C01, this difference in traversal logic causes a surplus in FLOPs for distance check and invocation overhead incurred by the `CellFunctor3B`. The workload for force computations and distance checks in GFLOPs is compared in figure 6.6.



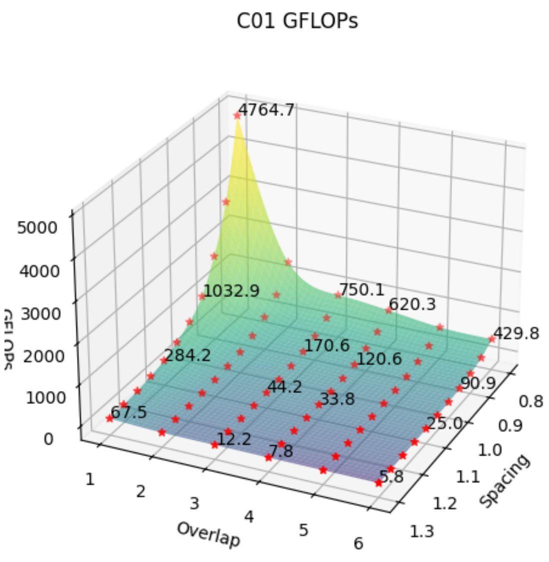
(a) C01 base steps to cover one triplet



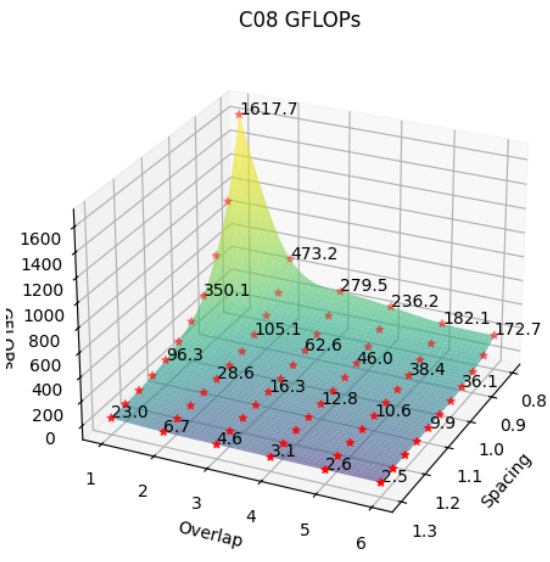
(b) C08 base step to cover one triplet with Newton3

Figure 6.5.: Difference of C01 and C08 with Newton3 in cell triplet processing. C01 passes a triplet thrice to the `CellFunctor3B`, re-checking particle distances, whereas C08 with Newton3 covers a triplet with one invocation by locking and writing to neighboring cells. C08 with Newton3 optimization saves FLOPs on `AxilrodTellerFunctor` and distance checks.

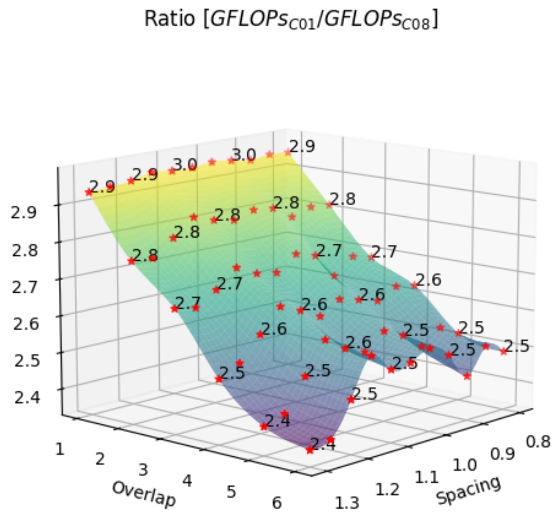
Figure 6.6 shows the comparison of C01 and C08 in terms of GFLOPs per iteration, and the ratio $\frac{GFLOPs_{C01}}{GFLOPs_{C08}}$ on each configuration.



(a) C01 GFLOPs



(b) C08 GFLOPs



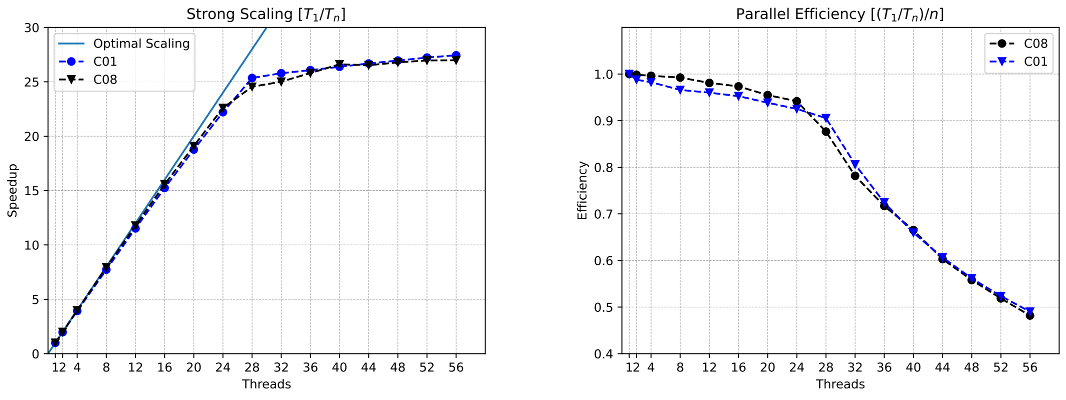
(c) Ratio of GFLOPs

Figure 6.6.: Comparison of total GFLOPs of C01 and C08 across spacing-overlap configurations. The ratio of GFLOPs [$C01/C08$] ranges from 2.4 to 2.9, with a decrease in the ratio for increasing overlap. The impact of GFLOPs is the most significant, with either overlap being fixed to 1 or spacing fixed to 0.8. Combining them returns the peaks. C01 shows the property of amplifying bad hit rates compared to C08 (Figure 6.5, which causes the uptrend in the ratio of GFLOPs.

The GFLOPs for both C01 and C08 traversal methods tend to rise quickly, with either overlap being fixed to 1 or spacing fixed to 0.8. Combining them returns the peaks of Figure 6.6a and Figure 6.6b. Conversely, this growth in GFLOPs is tempered when the overlap between cells is larger, effectively constraining the increase of GFLOPs with higher densities. These observations highlight the impact of adjusting cell sizes and the subsequent reduction of insignificant distance checks, as shown in figure 6.2. It should be noted here that the cell managing overhead is not included in the GFLOPs analysis, which only counts force computations distance checks.

The ratio plot reveals that not only do GFLOPs remain consistently higher for C01 across all settings, but they also accelerate more rapidly as the CSF increases. This acceleration is due to C01's methodology of recalculating distances multiple times, which amplifies the distance check workload, especially at low overlaps, which leads to low hit rates for three-body interactions Figure 6.2b.

Strong Scaling

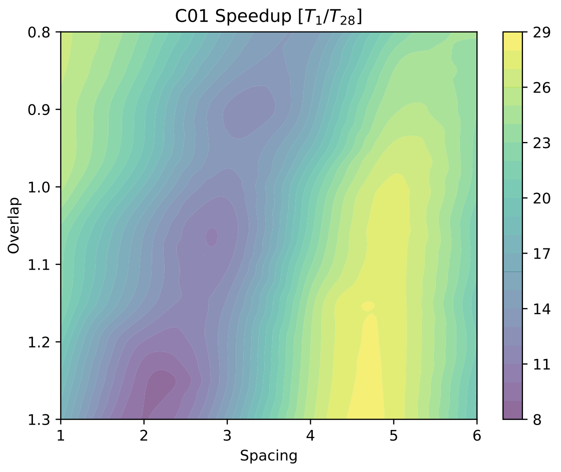


(a) Strong Scaling (b) Parallel Efficiency

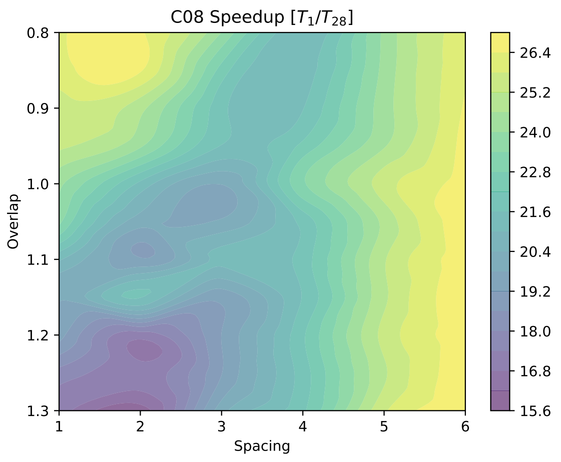
Figure 6.7.: Strong scaling of C01 and C08. Almost optimal scaling till 28 threads. Accelerated decrease in parallel efficiency beyond 28 threads due to hyperthreading.

The two figures provided show the strong scaling performance and efficiency of two methods, C01 and C08. The speedup factor evaluates strong scaling, $\frac{T_1}{T_n}$, which measures how the execution time decreases as more cores are added for a constant problem size. Efficiency is calculated as the speedup divided by the number of cores, $\frac{(T_1/T_n)}{n}$, representing the utilization of computational resources.

In figure 6.7a, we observe that both C01 and C08 exhibit an increase in speedup as more cores are used, indicating a reduction in execution time. The speedup increases almost linearly up to 28 threads, after which it begins to plateau. Figure 6.7b displays the strong scaling efficiency for both methods. Initially, the efficiency remains high and close to 1, indicating good core utilization. However, above 28 threads, the efficiency for both methods declines, corresponding to the plateauing of the speedup. This flattening aligns with the provided knowledge of CoolMUC-2 having 28 physical cores, and hyperthreading is used for larger threads.



(a) C01 Speedup [T_1/T_{28}]



(b) C08 Speedup [T_1/T_{28}]

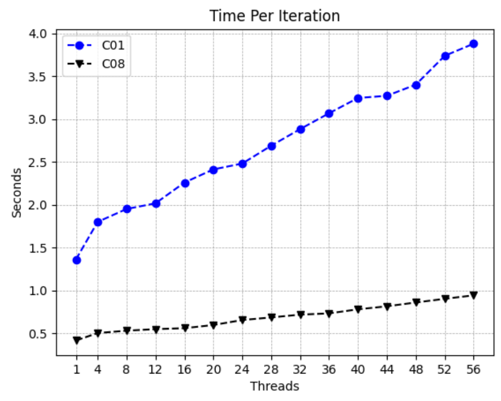
Figure 6.8.: Strong scaling of C01 and C08, $\frac{T_1}{T_{28}}$ across spacing-overlap configurations. Both traversals show almost optimal speedups at low overlap with low spacing and at overlap=5, highlighting the pattern of Figure 6.3c in an inverse version. The pattern's accuracy is substantiated by the observation that configurations with diminished parallel efficiency are characterized by increased locking and synchronization among threads. This correlation matches the assumption of these configurations of the diagonal pattern having the highest share of computation spent on force computations.

Contour Figure 6.8 illustrates the speedup of the C08 traversal method across overlap-spacing configurations, with the speedup factor $\frac{T_1}{T_{28}}$ on the z-axis. The speedup factor measures performance improvement when using 28 threads compared to a single thread. The x-axis represents the overlap between computational cells, and the y-axis shows the spacing of the particles.

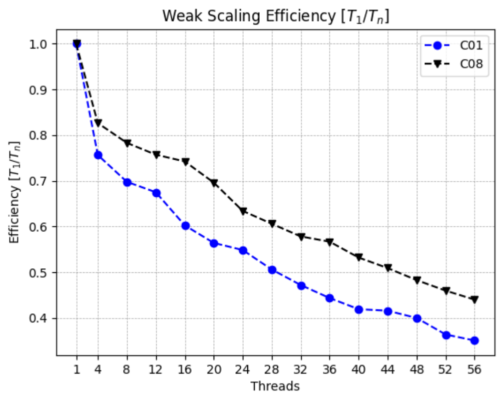
Between overlaps of one to three, particle spacing significantly influences speedup. The plot shows that within this overlap range, the strong scaling speedup tends to increase with a higher density of particles. As overlap grows beyond three, the positive correlation of density on speedup diminishes. The reason is the increasing administrative overhead for handling a more significant number of cells, which expands cubically with greater overlap, and the cubically decreasing particles per cell with growing overlap. As a result, the overhead for cell management begins to overshadow the relative contribution of force computation to the total computational work. At overlaps between four and six, spacing exerts minimal influence on speedup, indicating that performance is less affected by adjustments in particle spacing because of the overwhelming cell management overhead. In this high overlap regime, the speedup reflects the ratio of domain iteration times using one thread versus 28 threads. As the number of cells tends towards infinity, this speedup ratio approaches the ideal of 28.

Additionally, the diagonal pattern of Figure 6.3c matches the inverted pattern in Figure 6.8a and Figure 6.8b visually. Since the configurations with lowest parallel efficiency involve more locking and synchronization between threads, both Figure 6.8a and Figure 6.8b supports the initial assumption that a considerable amount of computational effort is directed towards force computations in these cases.

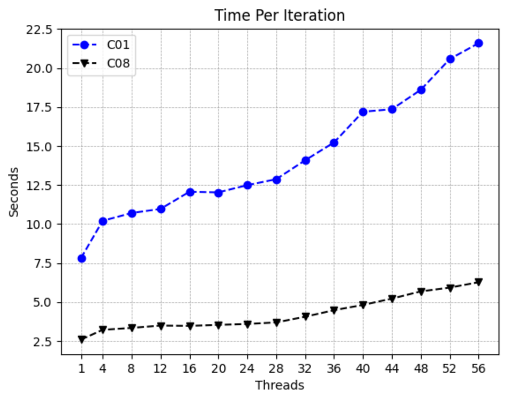
Weak Scaling



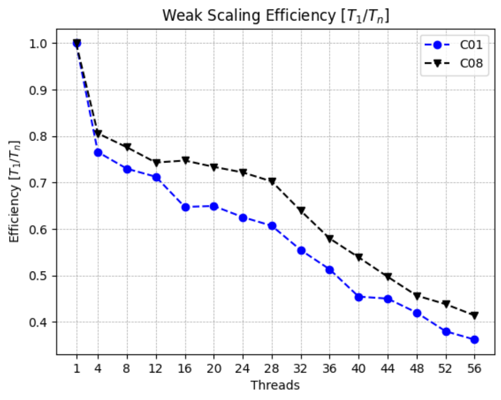
(a) Time per iteration for spacing=1.0 and overlap=2



(b) Weak scaling efficiency for spacing=1.0 and overlap=2



(c) Time per iteration for spacing=0.8 and overlap=2



(d) Weak scaling efficiency for spacing=0.8 and overlap=2

Figure 6.9.: Comparison of weak scaling of C01 and C08 across spacing=1 and spacing=0.8 with overlap=2. A steep drop in efficiency for threads 1 to 4 indicates overhead or inefficiencies, which is pronounced at the initial thread increase. At spacing=1, synchronization overhead is still a significant part of the simulation time, causing constant drag to the downside in efficiency. At spacing=1 the overhead dilutes the parallel portion of the program, which can be observed by the missing impact of the switch to hyperthreading at 28 threads. At 28 threads efficiencies of spacing=0.8 are 10% higher than at spacing=1.0, because force computation has a bigger proportion, outweighing synchronization overhead.

The following compares the weak scaling behavior of C01 and C08. Weak scaling measures how the execution time or efficiency changes as the size of the problem increases proportionally to the number of threads. The size of the problem primarily depends on the number of particles. The homogeneous distribution of particles gives an acceptable division of workload by cells. As explained in section 6.1.1, the increase of cells per thread is a cube of 125 cells being appended in the longest direction of the domain. Each of these cubes contains approximately 3200 particles.

Figure 6.9a and Figure 6.9c detail the time taken per iteration for both methods with spacing=0.8 taking 5 times longer, while Figure 6.9b and Figure 6.9d present the efficiency. Efficiency is determined by the ratio of the ideal time, which assumes perfect scaling, to the actual time taken when employing n threads. An efficiency $\frac{T_1}{T_n}$ of 1 indicates optimal scaling.

C08 outperforms C01 in both configurations. Figure 6.9b and Figure 6.9d confirm this by showing less efficiency of C01 across all threads. The efficiencies for both configurations drops significantly as the number of threads increases from 1 to 4. This steep decline suggests that both methods encounter overhead or inefficiencies when transitioning from single-threaded to multi-threaded execution, which is more pronounced in this initial thread increase. As the number of threads continues to rise from 4 to 28, Figure 6.9d of spacing=0.8 depicts the traversals maintaining a low rate of decrease of the efficiency level, while Figure 6.9d shows a faster downtrend of efficiencies. At 28 threads both traversals maintain 10% more efficiency with spacing=0.8.

Notably, Figure 6.9d presents a more pronounced drop in gradient of the efficiency lines beyond 28 threads for both traversal methods. However, for the configuration of spacing=1.0 with overlap=2(Figure 6.9b), the trend of efficiency decrease from 4 to 28 threads does not significantly change beyond 28 threads.

For spacing=1.0 and overlap=2, there are $180375/(7000 * 8) = 3.22$ particles per cell, while spacing=0.8 at overlap=2 holds $342176/(7000 * 8) = 6.11$ particles per cell. The share of force computation is more significant for spacing=0.8 due to more particles per cell. Conversely, the proportion of synchronization and cell managing overhead for spacing=1.0 is higher than at spacing=0.8, meaning the force computations are less significant.

The weak scaling of both traversals becomes more efficient as spacing decreases, because force computation fill a bigger share of simulation time, outweighing administrative overhead. This is confirmed by Figure 6.10, which is a plot of weak scaling efficiency for C08 at 28 threads across spacing-overlap configurations. With the increase in overlap, a parallel can be drawn to the effects previously observed in Figure 6.3c and Figure 6.8. Here, the influence of administrative overhead becomes more pronounced. The weak scaling efficiency in high overlap domain reflects the ratio of iteration time of 1 thread against iteration time of 28 threads over 28 times larger domain with few to non particles per cell, which approaches 1.

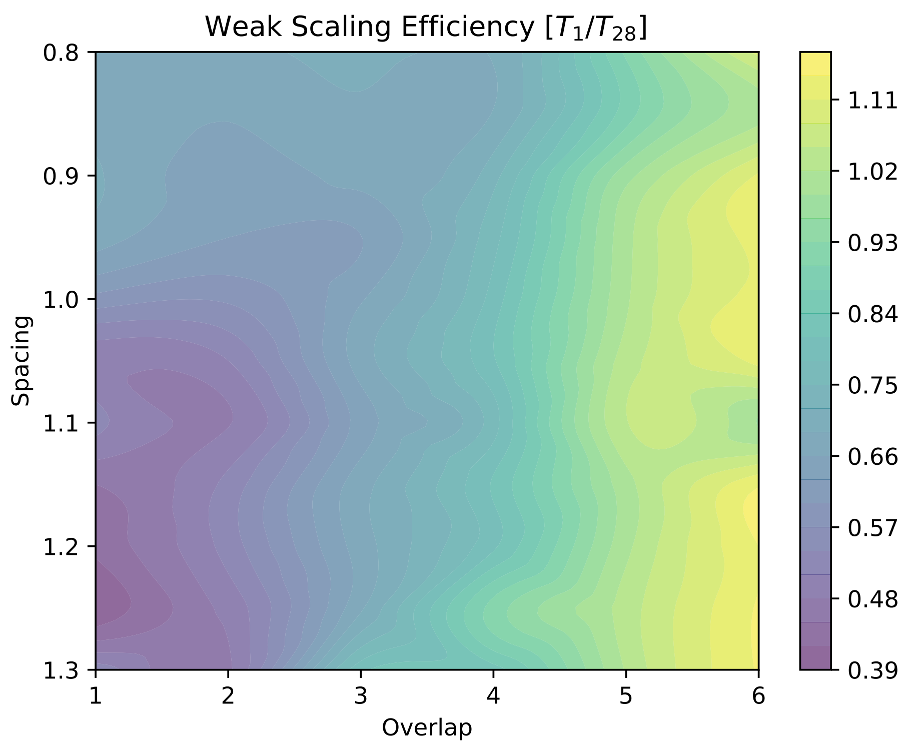
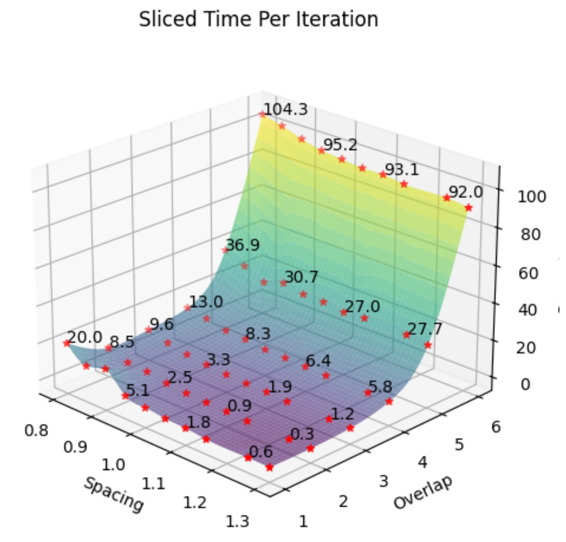


Figure 6.10.: C08 weak scaling efficiency $[T_1/T_{28}]$ over spacing and overlap configurations. From overlap 1 to 3, the efficiency increases when spacing decreases, confirming Figure 6.9. For growing overlap, the same effect of overwhelming overhead as in Figure 6.3c and Figure 6.8 appears, resulting in effectively dividing the iteration times of 1 thread against iteration times of 28 threads over 28 times larger domain, which approaches 1.

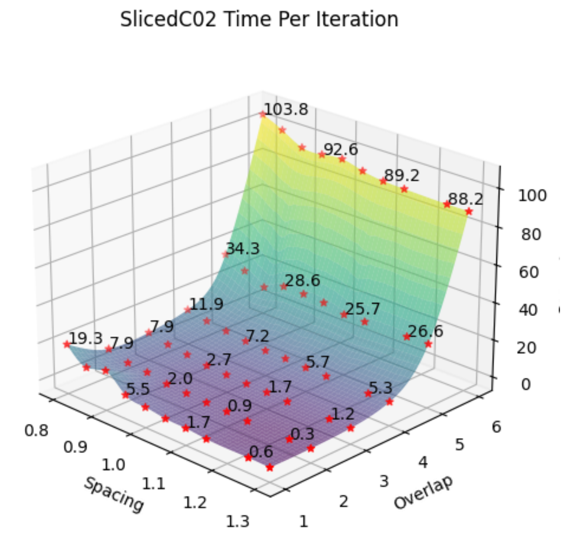
6.1.4. Comparison Of C08-Based Traversals

The following C08-based traversals are compared and analyzed regarding strong and weak scaling across spacing-overlap configurations.

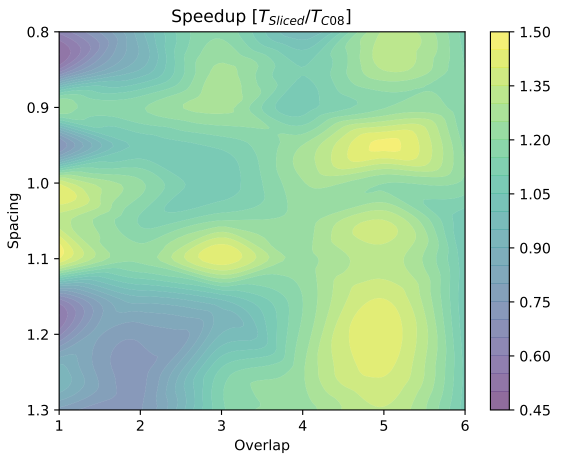
Firstly, iteration times of the traversals are shown in Figure 6.11a and Figure 6.11b, and speedups compared to C08 across different configurations are illustrated in Figure 6.11c and Figure 6.11d.



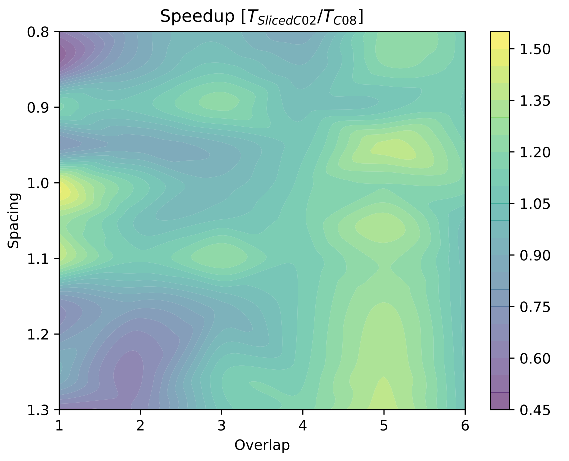
(a) Sliced time per iteration



(b) SlicedC02 time per iteration



(c) Speedup $[T_{Sliced}/T_{C08}]$

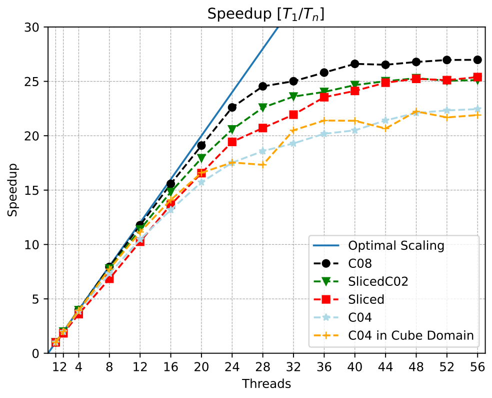


(d) Speedup $[T_{SlicedC02}/T_{C08}]$

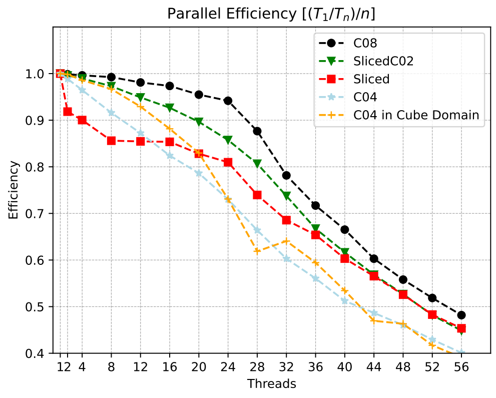
Figure 6.11.: The simulation times of both plots are very similar. The ratios of $[T_{Sliced}/T_{C08}]$ and $[T_{SlicedC02}/T_{C08}]$ range from around 40% to around 150%. Both sliced variants outperform C08 in certain low-overlap area configurations. Both variants exhibit slower times in high overlap areas.

Strong Scaling

The strong scaling performance in Figure 6.12a is assessed through the speedup factor $\frac{T_1}{T_n}$, which indicates the decrease in execution time as additional cores are utilized while maintaining a constant problem size of 7000 cells with 130375 particles. Parallel efficiency is represented in Figure 6.12b by the ratio of the speedup to the number of cores, $\frac{(T_1/T_n)}{n}$, reflecting the efficiency of resource utilization.



(a) Speedup $[T_1/T_n]$



(b) Parallel efficiency $[(T_1/T_n)/n]$

Figure 6.12.: Strong scaling of C08-based traversals for spacing=1.0 and overlap=1. C08 performs best with almost linear speedup until 28 threads after using hyperthreading. C04 performs roughly the same in the chosen long cuboid domain and a cube-shaped domain with an equal amount of cells.

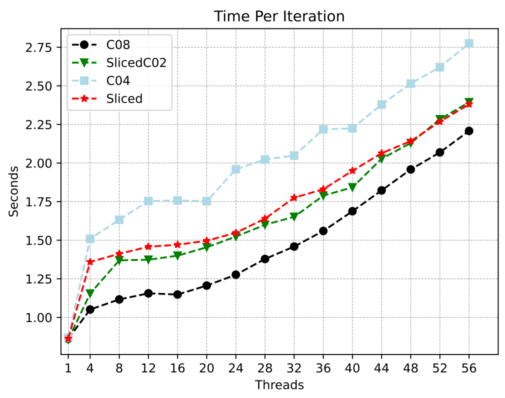
From 1 to 28 threads, the scaling efficiency of C08, C04, SlicedC02, and Sliced traversals demonstrates a relatively linear increase in speedup, indicative of good strong scaling. However, the speedup plateaus once hyperthreading comes into play beyond 28 threads. C08 leads in performance, with SlicedC02 following closely. Though starting slower, the Sliced traversal gradually reaches the performance of the others, which is reflected by a slight initial dip in parallel efficiency before stabilizing between 85% to 80% efficiency up to 24 threads.

At 56 threads, the speedup for C08 peaks at approximately 27, while C04 reaches a speedup of around 22. A separate measurement was taken with C04 in a cube-shaped domain with dimensions $51.3 \times 51.3 \times 51.3$, resulting in $19^3 = 6859$ cells. The performance of C04 in the cube-shaped domain did not exhibit any notable improvements compared to the original long cuboid domain. The parallel efficiency for C04 in the cube domain also aligns with the efficiency observed in the original domain.

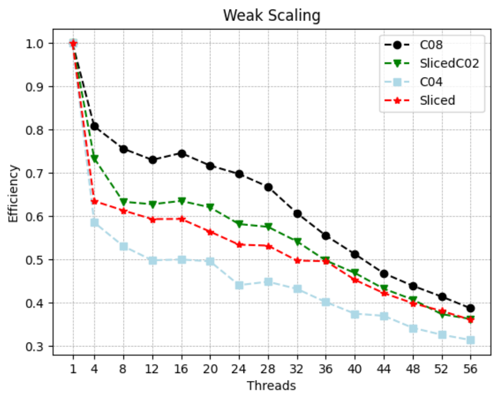
Weak Scaling

The following compares the weak scaling behavior among C08-based traversals. The simulation setup is analog to the one used in subsubsection 6.1.3. For each thread, a cube of 125 cells is added. The only examined configuration here is spacing=1 with overlap=1. Figure 6.13 depicts the time per iteration, whereas Figure 6.13b shows the efficiency $\frac{T_1}{T_n}$.

All traversal times in Figure 6.13a show similar shapes and are on top of each other in a parallel manner. The fastest to slowest traversal orders are C08, SlicedC02, Sliced, and C04. Similarly to the analysis in subsubsection 6.1.3, the efficiency experiences an initial drop, then maintains a slow decrease rate till 28 threads and accelerates in decrease as hyper-threading employs. The plots indicate that for three-body interactions, the synchronization mechanisms of different traversal schemes can end up being more of a slowdown than a performance benefit.



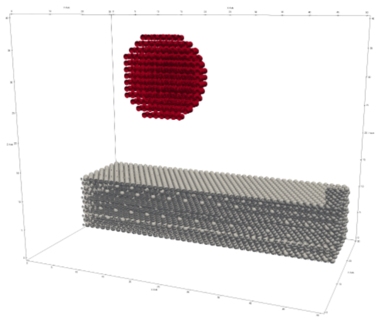
(a) Time per iteration

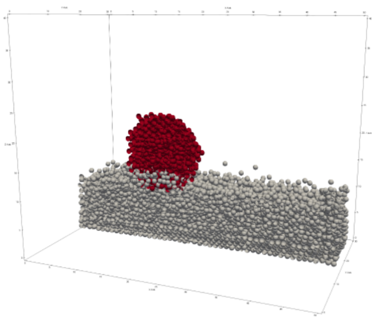


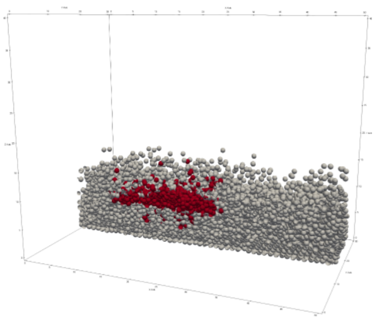
(b) Weak scaling efficiency $[T_1/T_n]$

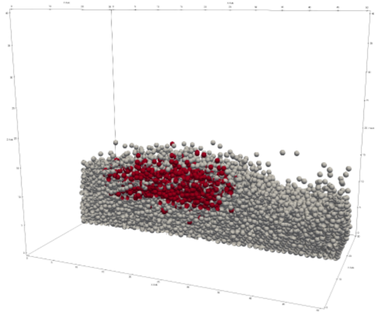
Figure 6.13.: Comparison of weak scaling of C08-based traversals for spacing=1.0 and overlap=1. The initial drop of efficiency till four threads. More steady and slow decrease in efficiency from threads 4 to 28. Faster decrease in efficiency after 28 threads. The benefits of traversal schemes built on top of C08 are less pronounced for three-body interactions than for two-body interactions.

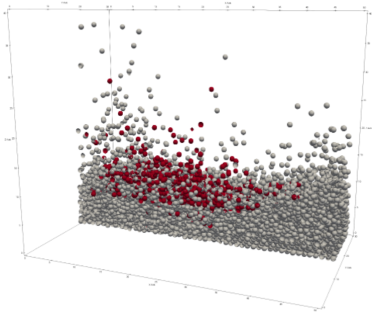
6.2. Inhomogeneous Simulation: Falling Drop


(a) Time step 0


(b) Time step 3000


(c) Time step 4500


(d) Time step 5500


(e) Time step 9500

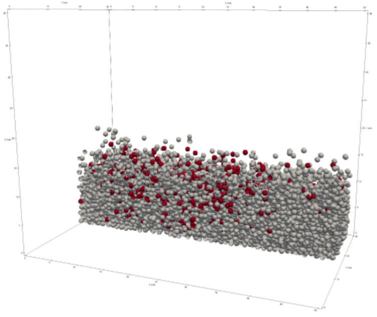

(f) Time step 35000

Figure 6.14.: The distribution of the drop material (red) after the impact is highlighted in a cross-section of the falling drop scenario, colored by particle type.[GSBN22]

In the falling drop simulation, commonly used for educational purposes and illustrated in Figure 6.14, a sphere of particles is influenced by gravity to fall into a stationary particle layer. The simulation domain is delineated by boundaries that reflect particles into the domain from every wall—the collision results in the rebound of the shock wave from the domain's base, causing a mingling of particles from the falling drop and the stationary layer. As time progresses, the force of gravity causes the particles to settle at the bottom of the domain. This simulation, consisting of 15094 particles, was carried out on a single node of CoolMUC-2 with 28 physical cores and two hyper threads per core, resulting in 56 threads. In the following, the iteration log was evaluated to analyze the performance of two-body and three-body linked cell traversals in combination (Table 6.1) and the auto-tuning results.

Functor	Functor Time (s)	Percentage of Total Functor Time (%)
Axilrod-Teller	3253.42	87.34%
Lennard-Jones	465.18	12.66%

Table 6.1.: Total computation times and percentages for Axilrod-Teller and Lennard-Jones functors. Three-body interactions cause 87% of the combined force computation times due to a lower hit rate and more expensive force computation than two-body interactions.

Table 6.2 outlines the results from two-body force computations, while Table 6.3 details the outcomes for three-body force computations. In the context of two-body interactions, the traversals C01, C08, and C18, with a Cell Size Factor (CSF) of 1, account for 94.49% of the iterations and 67.16% of the overall computation time for two-body forces, contributing 35.5%, 26.56%, and 5.1% respectively.

Conversely, for three-body force computations, C08 at a CSF of 0.5 dominates, occupying 96.65% of all iterations and 72.52% of the computation time, highlighting its dominance in terms of performance. The tuning iterations, which are represented by the non-bold rows, have a maximum of 72 per row due to the configuration parameters Newton3 and data-layout. For pairwise interactions, two data-layouts exist: Structure of Arrays (SoA) and Array of Structures (AoS), while for three-body interactions, only AoS is currently implemented. This distinction in data-layouts halves the number of tuning iterations for three-body interactions. The reason C01 has fewer tuning iterations is its incompatibility with Newton3.

Two significant insights emerge from this analysis: first, the competitive nature of two-body traversals contrasts with C08's singular dominance in three-body traversals. Secondly, the selection of CSF=1 for two-body interactions implies efficient hit rates. In contrast, the choice of CSF=0.5 for three-body interactions indicates a less efficient hit rate at CSF=1 and a generally poorer hit rate, as depicted in Figure 6.2.

Besides, the underperformance of other C08-based traversals can be attributed to the limited size of the simulation domain, which restricts the ability of larger traversal patterns to fully utilize all 56 threads.

Traversal	CSF	TotalTime (s)	Percentage	Iterations	TimePerIteration (s)
lc_c01	0.25	3.16	0.68%	36	0.09
lc_c01	0.33	2.61	0.56%	36	0.07
lc_c01	0.50	4.60	0.99%	36	0.13
lc_c01	**1.00**	**165.19**	**35.50%**	**6529**	**0.03**
lc_c08	0.25	17.26	3.71%	72	0.24
lc_c08	0.33	10.61	2.28%	72	0.15
lc_c08	0.50	6.57	1.41%	72	0.09
lc_c08	**1.00**	**123.59**	**26.56%**	**5072**	**0.02**
lc_c18	0.25	38.36	8.24%	72	0.53
lc_c18	0.33	22.61	4.86%	72	0.31
lc_c18	0.50	4.03	0.87%	72	0.06
lc_c18	**1.00**	**23.75**	**5.10%**	**2572**	**0.01**
lc_sliced_c02	0.25	22.80	4.90%	72	0.32
lc_sliced_c02	0.33	13.71	2.95%	72	0.19
lc_sliced_c02	0.50	4.37	0.94%	72	0.06
lc_sliced_c02	1.00	2.11	0.45%	72	0.03

Table 6.2.: Auto-tuning chose three traversals with CSF=1 for non-tuning iterations, indicating efficiencies of the traversals become effective across various phases and a rather competitive distribution with C01 being preferred over C08.

Traversal	CSF	TotalTime (s)	Percentage	Iterations	TimePerIteration (s)
lc_c01_3b	0.25	30.50	0.94%	18	1.69
lc_c01_3b	0.33	9.54	0.29%	18	0.53
lc_c01_3b	0.50	6.07	0.19%	18	0.34
lc_c01_3b	1.00	14.92	0.46%	18	0.83
lc_c04_3b	1.00	163.37	5.01%	36	4.54
lc_c08_3b	0.25	28.91	0.89%	36	0.80
lc_c08_3b	0.33	10.59	0.32%	36	0.29
lc_c08_3b	**0.50**	**2363.85**	**72.52%**	**14497**	**0.16**
lc_c08_3b	1.00	43.24	1.33%	36	1.20
lc_sliced_3b	0.25	89.26	2.74%	36	2.48
lc_sliced_3b	0.33	33.20	1.02%	36	0.92
lc_sliced_3b	0.50	32.40	0.99%	36	0.90
lc_sliced_3b	1.00	106.96	3.28%	36	2.97
lc_sliced_c02_3b	0.25	129.59	3.98%	36	3.60
lc_sliced_c02_3b	0.33	47.36	1.45%	36	1.32
lc_sliced_c02_3b	0.50	42.36	1.30%	36	1.18
lc_sliced_c02_3b	1.00	107.51	3.30%	36	2.99

Table 6.3.: C08 with CSF=0.5 was chosen for all non-tuning iterations. Sliced traversals and C04 traversal perform worse due to the small domain and more threads than cells are accessible in parallel.

7. Conclusion

This Bachelor's thesis presented the implementation and performance of linked-cells traversals for three-body interactions within AutoPas. It begins with a summary of the theoretical physics concepts of MD and the technical background of the strategies for efficiently identifying neighboring particles and parallelization, which includes traversal schemes and base steps. The implementation chapter details the C08 traversal base step developed for three-body interactions and a generalization of this traversal step to N-body interactions. Performance analysis is a critical component of the thesis, thoroughly comparing the C01 and C08 traversal methods using simulations run on the CoolMUC-2 Linux cluster. The analysis extends to variations within C08-based traversals, assessing metrics such as time per iteration, gigaflops per second, hit rate, and scaling across simulation setups with different cell size factors and particle densities. C08 outperforms C01 via Newton3 optimization across all scenarios by a factor ranging from 1.8 to 4.8 and also shows dominance among the C08-based traversals. The evaluation highlights the limitation of linked cells due to the low particle hit rate of below 2% at cell size factor 1 for three-body interactions. Achieving a higher share of force computations of the total simulation time or increasing the hit rate using cell size optimization loses effectiveness as cell size decreases due to the significant administrative overhead of the cubically rising number of cells. This observation is shown in Figure 6.3 by a pattern across spacing-overlap configurations from low density and low overlap to those with high density and high overlap, reflecting a growing optimal cell size for increasing density. This pattern concludes that linked cell traversals are effective when both conditions, high hit rate and low proportion of cell overhead, are only given in highly dense systems. Notably, the pure C08 traversal typically performs best over most conducted simulations, particularly when the cell size factor is set to 0.5. This was confirmed in the analysis of an inhomogeneous simulation of a falling drop. Looking ahead, the thesis proposes areas for future research and development. The observation of a low hit rate for three-body linked cell traversals suggests that Verlet lists offer a better solution for three-body interactions due to their property of storing neighboring particles for each particle. However, linked cell traversals retain utility as they are essential for constructing Verlet lists. This insight opens avenues for further investigation, potentially leading to more sophisticated algorithms that leverage the strengths of both linked cell traversals and Verlet lists for high performance in molecular dynamics simulations.

A. Appendix

A.1. Optimized computeOffsets()-Function Python Code

Below is the Python code for the optimization of the complexity of the computeOffsets()-function for three-body interactions, which Subsection 5.1.2 presented.

```

def pair_to_triple(pair):
    [a, b] = sorted(pair)
    return [a, a, b]


def append_inner_offsets(o1, o2, all, offsets):
    for x in range(1, all.__len__()):
        for y in range(1, all[0].__len__()):
            for z in range(1, all[0][0].__len__()):
                offsets.append([o1, o2, all[x][y][z]])


def append_plane_offsets(o1, o2, plane, offsets):
    for x in range(1, plane.__len__()):
        for y in range(1, plane[0].__len__()):
            offsets.append([o1, o2, plane[x][y]])

def offsets_3_edges_2_same(edge1, edge2, offsets):
    for a in range(1, edge1.__len__()):
        for b in range(1, edge2.__len__()):
            for a1 in range(a + 1, edge1.__len__()):
                offsets.append([edge1[a], edge1[a1], edge2[b]])


def offsets_2_edges_1_unknown(edge1, edge2, planes, all, offsets):
    ov_1 = edge1.__len__()
    ov_2 = edge2.__len__()
    for a in range(1, ov_1):
        # 2 edges 1 wildcard
        for b in range(1, ov_2):
            # 2 edges (1 double)
            offsets.append(pair_to_triple([edge1[a], edge2[b]]))
            # 2 edges 1 plane
            append_plane_offsets(edge1[a], edge2[b], planes[0], offsets)
            append_plane_offsets(edge1[a], edge2[b], planes[1], offsets)
            append_plane_offsets(edge1[a], edge2[b], planes[2], offsets)
            # 2 edges 1 inner
            append_inner_offsets(edge1[a], edge2[b], all, offsets)
    # 2 edges (2 different cells same edge)
    offsets_3_edges_2_same(edge1, edge2, offsets)
    offsets_3_edges_2_same(edge2, edge1, offsets)

```

```python

def offsets_1_edge_2_unknown(edge, plane1, plane2, plane3, all, offsets):
    for a1 in range(1, edge.__len__()):
        for x3 in range(1, plane3.__len__()):
            for y3 in range(1, plane3[0].__len__()):
                # 1 edge 1 plane (edge double)
                offsets.append(pair_to_triple([edge[a1], plane3[x3][y3]]))
                # 1 edge 2 planes
                append_plane_offsets(edge[a1], plane3[x3][y3], plane1, offsets)
                append_plane_offsets(edge[a1], plane3[x3][y3], plane2, offsets)
                # 1 edge 1 plane (2 different cells same plane)
                for x32 in range(1, plane3.__len__()):
                    for y32 in range(1, plane3[0].__len__()):
                        if plane3[x3][y3] < plane3[x32][y32]:
                            offsets.append([edge[a1], plane3[x3][y3], plane3[x32][y32]])
                # 1 edge 1 plane 1 inner
                append_inner_offsets(edge[a1], plane3[x3][y3], all, offsets)
                # 1 edge 1 plane (2 different cells same edge)
                for a2 in range(a1 + 1, edge.__len__()):
                    offsets.append([edge[a1], edge[a2], plane3[x3][y3]])


def lcc08_offsets(overlap):
    dim = overlap + 1
    dim_x = dim
    dim_y = dim
    ov_x = dim
    ov_y = dim
    ov_z = dim
    all = [[[((dim_x ** 2) * x) + (y * dim_y) + z for z in range(ov_z)] for y in range(ov_y)] for x in range(ov_x)]
    plane_xy = [[((dim_x ** 2) * x) + (y * dim_y) for y in range(ov_y)] for x in range(ov_x)]
    plane_xz = [[((dim_x ** 2) * x) + z for z in range(ov_z)] for x in range(ov_x)]
    plane_yz = [[(y * dim_y) + z for z in range(ov_z)] for y in range(ov_y)]
    edge_x = [(dim_x ** 2) * e for e in range(ov_x)]
    edge_y = [dim_y * e for e in range(ov_y)]
    edge_z = [e for e in range(ov_z)]

    planes = [plane_xy, plane_xz, plane_yz]
    # base cell combos
    offsets = [[0, 0, 0]]
    for a in range(0, ov_x * ov_y * ov_z):
        for b in range(a + 1, ov_x * ov_y * ov_z):
            offsets.append([0, a, b])

    # 3 planes
    for x in range(1, ov_x):
        for y in range(1, ov_y):
            for x1 in range(1, ov_x):
                for z in range(1, ov_z):
                    append_plane_offsets(plane_xy[x][y], plane_xz[x1][z],
                              plane_yz, offsets)
    # 3 edges
    for x in range(1, ov_x):
        for y in range(1, ov_y):
            for z in range(1, ov_z):
                offsets.append([edge_x[x], edge_y[y], edge_z[z]])
    # 1 edge 2 wildcard
    # 2 edges 1 wildcard
    offsets_2_edges_1_unknown(edge_x, edge_y, planes, all, offsets)
    offsets_2_edges_1_unknown(edge_x, edge_z, planes, all, offsets)
    offsets_2_edges_1_unknown(edge_y, edge_z, planes, all, offsets)

    # 1 edge 1 plane 1 wildcard
    offsets_1_edge_2_unknown(edge_x, plane_xy, plane_xz, plane_yz, all,
        offsets)
    offsets_1_edge_2_unknown(edge_y, plane_xy, plane_yz, plane_xz, all,
        offsets)
    offsets_1_edge_2_unknown(edge_z, plane_yz, plane_xz, plane_xy, all,
        offsets)

    return offsets
```

Listing A.1: General form of a typical runner() function.

List of Figures

List of Tables

Bibliography

[AT43] BM Axilrod and Ei Teller. Interaction of the van der waals type between three atoms. *The Journal of Chemical Physics*, 11(6):299–300, 1943.

[Fis20] Vincent Fischer. Implementation and analysis of load balancing options for autopas' sliced traversal. Master's thesis, Technical University of Munich, Sep 2020.

[GSBN22] Fabio Alexander Gratl, Steffen Seckler, Hans-Joachim Bungartz, and Philipp Neumann. N ways to simulate short-range particle systems: Automated algorithm selection with the node-level library autopas. *Computer Physics Communications*, 273:108262, 2022.

[Mar01] Gianluca Marcelli. *The role of three-body interactions on the equilibrium and non-equilibrium properties of fluids from molecular simulation.* University of Kent (United Kingdom), 2001.

[Men19] Christian Menges. Optimization and evaluation of the linked-cell algorithm. Master's thesis, Technical University of Munich, Aug 2019.

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

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

[New33] Isaac Newton. *Philosophiae naturalis principia mathematica*, volume 1. G. Brookman, 1833.

[TM00] Mark E Tuckerman and Glenn J Martyna. Understanding modern molecular dynamics: Techniques and applications, 2000.

[TSH$^+$19] Nikola Tchipev, Steffen Seckler, Matthias Heinen, Jadran Vrabec, Fabio Gratl, Martin Horsch, Martin Bernreuther, Colin W Glass, Christoph Niethammer, Nicolay Hammer, et al. Twetris: Twenty trillion-atom simulation. *The International Journal of High Performance Computing Applications*, 33(5):838–854, 2019.

[Ver67] Loup Verlet. Computer" experiments" on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical review*, 159(1):98, 1967.

[WS06] Liping Wang and Richard J Sadus. Three-body interactions and solid-liquid phase equilibria: Application of a molecular dynamics algorithm. *Physical Review E*, 74(3):031203, 2006.