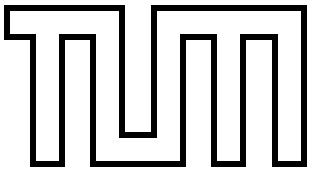


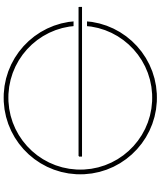
FAKULTÄT FÜR INFORMATIK

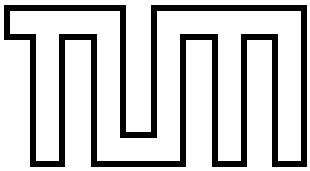
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Information Systems

Implementation and Analysis of Parallelization Algorithms for Molecular Dynamics Simulations

Sabrina Krallmann





FAKULTÄT FÜR INFORMATIK

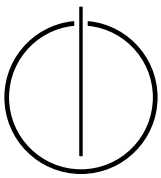
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Information Systems

Implementation and Analysis of Parallelization Algorithms for Molecular Dynamics Simulations

Implementierung und Analyse von Parallelisierungsalgorithmen für Molekulardynamiksimulationen

Author:	Sabrina Krallmann
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Fabio Alexander Gratl, M.Sc.
Date:	September 11, 2020



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

Munich, September 11, 2020 Sabrina Krallmann

Acknowledgements

At this point I want to express my gratitude to the people supporting me during this thesis. I would like to thank my advisor, Fabio Gratl, for creating an ideal working environment, as well as for his continuous support in the form of practical explanations and valuable feedback. Further, I would like to thank Marian for providing his computer for testing purposes and for his untiring support. Also I would like to thank Niklas for giving me valuable physical insights and Akbar for his help with 3D graphics. Finally I want to thank Stephanie and David for proofreading my thesis.

Abstract

Within the field of molecular dynamics, force and distance calculations are computationally heavy and the main cause of a long run-time. To decrease this run-time, there are several neighborhood-based particle search algorithms and parallelization strategies. The library AutoPas addresses this challenge and uses auto-tuning to dynamically choose the best strategy during run-time. The parallelization strategies are implemented in so-called traversals. Those are based on containers which originated from neighborhood-based particle search algorithms. To choose the best option, all traversals have to be tested by linearly scanning all options. In order to optimize this search, one would need to know which parameters are influencing the outcome of searching the currently optimal traversal.

The present thesis analyzes the traversals regarding their behavior depending on different factors. Those factors are a high number of particles, a large domain size, different densities, as well as in-homogeneous and homogeneous scenarios. The scenarios were tested on a Coffee Lake platform and a Haswell platform. Both platforms were also compared.

For a high density, an increasing number of particles leads to the traversals sorting by container, which leads to the assumption that for those scenarios the particle search algorithm is more important than the parallelization strategy. Overall, a high number of particles favored a Verlet Lists approach, while scenarios with a large domain size performed best for Verlet Cluster Lists based traversals. In general, an increasing standard deviation of the homogeneity showed an increasing run-time. This is especially disadvantageous for traversals whose parallelization is sliced-based.

Zusammenfassung

Im Feld der Molekulardynamik sind Kraft- und Distanzberechnungen sehr rechenintensiv. Sie sind damit die Hauptursache für eine lange Laufzeit. Um diese Laufzeit zu verkürzen, gibt es mehrere nachbarschaftsbasierte Partikel-Suchalgorithmen und Parallelisierungsstrategien. Die Library AutoPas adressiert dieses Problem. Mit Hilfe von Auto-Tuning wird während der Laufzeit dynamisch die beste Strategie ausgewählt. Die dort angewandten Parallelisierungsstrategien werden als sogenannte Traversal implementiert. Diese basieren auf Containern, welche auf Grundlage der nachbarschaftsbasierten Partikel-Suchalgorithmen aufbauen. Um die zum aktuellen Zeitpunkt beste Option zu wählen, müssen alle Traversel getestet werden. Dazu werden alle erwägbaren Optionen linear durchsucht. Um diese Suche zu optimieren, bedarf es der Information, welche Parameter für das Ergebnis der Suche nach dem aktuell optimalen Traversals, entscheidend sind.

In der vorliegenden Arbeit werden die Traversal hinsichtlich ihres Verhaltens, in Abhängigkeit von verschiedenen Faktoren analysiert. Diese Faktoren sind eine hohe Anzahl von Partikeln, die Domänengröße, unterschiedliche Dichten, sowie die Homogeniät eines Szenarios. Die Szenarien wurden auf einer Coffee Lake-Plattform und einer Haswell-Plattform getestet. Beide Plattformen wurden zudem miteinander verglichen.

Für eine hohe Dichte führt eine zunehmende Anzahl von Partikeln zu einer Sortierung der Traversen nach Containern. Dies führt zu der Annahme, dass für diese Szenarien der Partikel-Suchalgorithmus relevanter ist, als die Parallelisierungsstrategie. Insgesamt ist der Verlet Lists Algorithmus für eine hohe Anzahl an Partikeln zu bevorzugen. Währenddessen schneiden auf Verlet Cluster Listen basierende Traversal für Szenarien mit einer großen Domänengröße am besten ab. Im Allgemeinen bewirkte eine zunehmende Standardabweichung der Homogenität eine wachsende Laufzeit. Besonders davon betroffen, waren Traversal, deren Parallelisierungsstrategie sliced-basiert ist.

Contents

Part I.

Introduction and Background

1. Introduction

Molecular dynamics simulations, which are a form of N-body problem [Rap97], have a wide range of application fields. Such as Alzheimer's research [BTH05] or drug design [OFF+09] within medicine, or in chemistry for simulating fluids [PEQ82]. The main challenge of molecular dynamics simulations is to increase the performance of a computation. In this thesis the performance was measured by the time to solution. Especially pairwise force and distance calculations are computationally intensive and cause a growing run-time [Eck14]. To scale down the amount of distance calculations, several algorithms for neighborhood-based particle search were developed. These are described in detail in Section 2.2. To further increase the performance, there are multiple parallelization strategies and other parameters from which benefits can be drawn.

The open-source library AutoPas[1] addresses this challenge. By applying auto-tuning, it identifies the currently best configuration during run-time. Part of this configuration are traversals which are based on a certain neighborhood-based particle search algorithm, as well as a specific parallelization strategy. As an example for those traversals, the c04HCP traversal will be explained in this work, since it was implemented in the context of this thesis.

At the moment, all available options have to be tested linearly to identify the optimal configuration. To optimize scanning the search space, one would need to sort the options within, depending on the structure of the scenario that is tested. It might even make sense to test only certain options for specific scenarios depending on its physical structure. In order to make these decisions, more information about the dependencies between the physical structure of a scenario and its impact on the performance of a certain configuration is needed.

The presented thesis aims to analyze those configurations regarding their performance for the following parameters: An increasing number of particles, a changing domain size and different densities. In advance, an upper bound for the density of a scenario was identified by the close-packing of equal spheres problem. Furthermore, the homogeneity of a scenario, which was measured by its standard deviation, was taken into account.

All tests were performed on a Coffee Lake platform, usually with twelve threads, as well as on a Haswell platform with 28 and 56 threads. The two computational platforms and the different amount of threads were also compared. This comparison was done in order to examine which configurations perform differently or better on a certain hardware or with a different amount of threads available. Additionally, tests were performed with two functors, which have different approaches for vectorization.

[1] `https://github.com/AutoPas/AutoPas`

2. Theoretical Background

In this chapter an introduction to the field of molecular dynamics will be given and the physical background will be described. The algorithms for neighborhood-based particle search discussed in this thesis will be explained in detail. Furthermore, an upper bound for the close packing of the spheres will be defined.

2.1. Molecular Dynamics

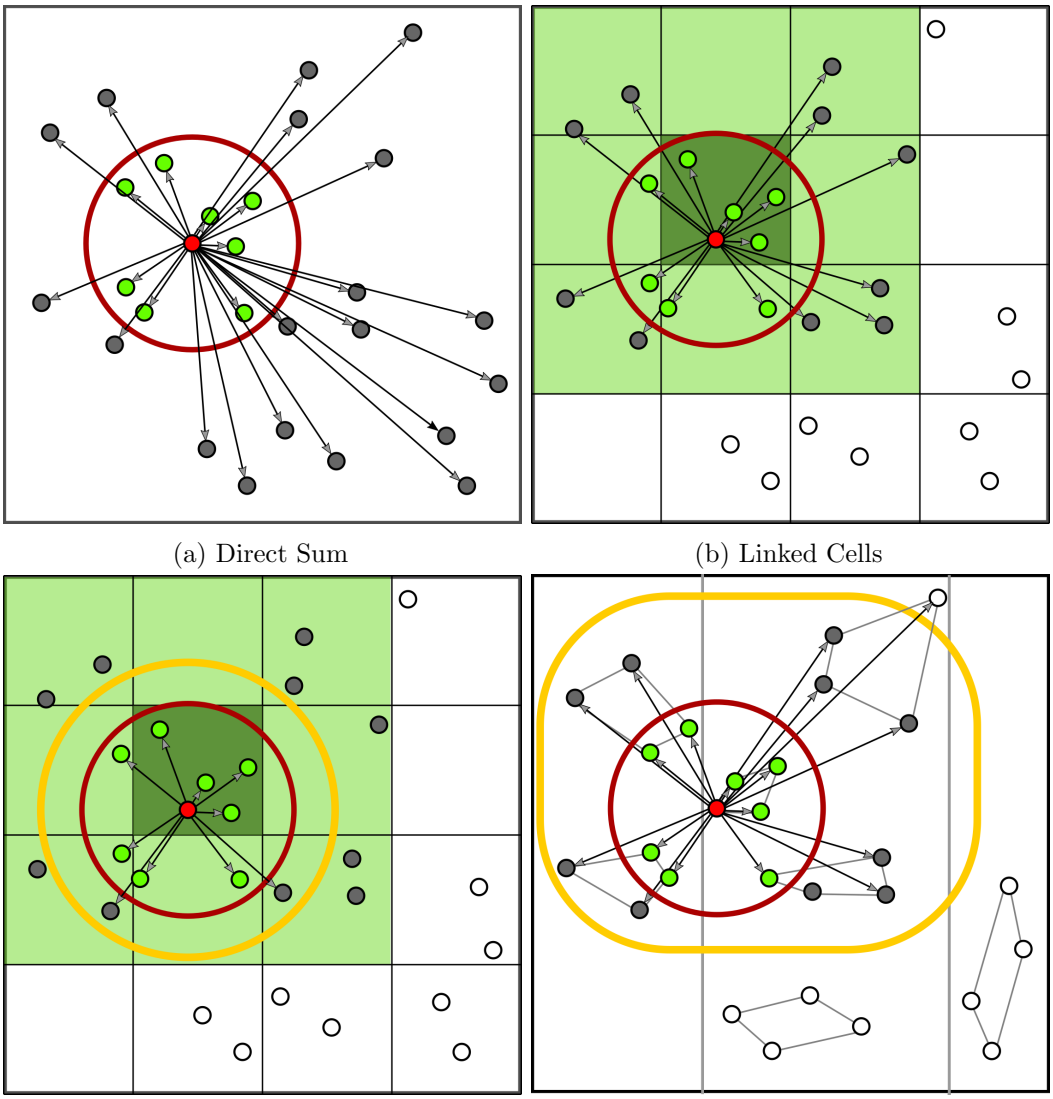
In general, molecular dynamics are part of the N-body problem family [Rap97]. Its main challenge is the efficient computation of positions and velocities at different times for N bodies in a domain. Those bodies have a given position and velocity at starting time and move according to the applied forces and Newton's Laws of motion [Tch20]. The presented thesis, as well as the described library AutoPas focuses on pairwise force calculations between two particles at a time. An often used example for the forces applied is the Lennard-Jones potential. It can be seen in the following equation [GKZ07]:

$$U(r_{ij}) = 4 \cdot \epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right] \tag{2.1}$$

The equation is based on the distance r between two molecules, the parameter ϵ that denotes the depth of the potential well and the parameter σ which determines the zero-crossing of the potential. The Lennard-Jones potential converges to zero. This allows simulations to only consider pairwise force calculations within a certain cutoff radius. Thus unnecessary calculations can be disregarded.

2.2. Algorithms for Neighborhood-Based Particle Search

Since the short range force calculations are the computationally heaviest part of a simulation [Eck14], molecular dynamics simulations aim to reduce those. By evaluating only those particle pairs whose distance is smaller than or equal to the cutoff radius, introduced in Section 2.1, unnecessary calculations can be avoided. To find the particles which are still to be paired, there exist different algorithms. Those will be explained in this chapter.



(a) Direct Sum (b) Linked Cells

(c) Verlet Lists (d) Verlet Cluster Lists

Figure 2.1.: All figures are showing neighborhood-based particle search algorithms. Figure 2.1a is visualizing the Direct Sum algorithm where the distance between all particle pairs needs to be calculated in order to find those that are close enough to each other to calculate the interacting forces. In red, one can see the particle whose distances and interactions with other particles are currently calculated. The cutoff radius is visualized in dark red. In light green, the particles close enough for a pairwise force calculation are marked. Figure 2.1b shows the Linked Cells algorithm. The whole domain is split into cells which are as large as or larger than the cutoff radius. The particles which are not considered for distance calculations are colored in white. The Verlet Lists algorithm that stores neighbor particles in a list can be observed in Figure 2.1c. The verlet skin can be seen as the yellow circle around the cutoff radius. Figure 2.1d displays the Verlet Cluster Lists approach. It can be seen how the domain is split into towers where the particles are grouped into clusters of the size four. The verlet skin is here also marked in yellow.

2.2.1. Direct Sum

The straight-forward way to get all neighboring particles within a certain cutoff radius for a certain particle, is to calculate the distances between all possible particle pairs. As it can be seen in Figure 2.1a for the Direct Sum approach, the distances between the red particle and each other particle are calculated. Then, only the forces between the red particle and the ones located inside the red marked cutoff radius are computed. The particles within the cutoff radius are marked in light green in Figure 2.1a. Direct Sum is thus reducing the overall amount of force calculations to those necessary.

On the upside, the Direct Sum approach is easy to implement and can be used without complex data structures. However, it still needs N distance calculations per particle and hence comes with a complexity of $O(N^2)$. [GST+19]

2.2.2. Linked Cells

One way to minimize the amount of distance calculations is splitting the domain into a 3D grid with cells that are equal to or larger than the cutoff radius. For all particles within one cell, only the particles within the current cell and those within the neighbor cells have to be considered, instead of all particles in the domain. As it can be seen in Figure 2.1b for the red particle, only the green cells are considered. The cutoff radius is shown by the red circle around the red particle. After calculating all distances between the red particle and the particles within the green cells, the pairwise force calculation is applied between the red particle and the light green particles.

If the number of cells is in proportion to the number of particles, the time complexity is reduced to $O(N)$. Since the Linked Cells algorithm needs a lot of pairwise distance calculations between particles, it also creates a lot of overhead. The probability of finding a particle that lays within the cutoff radius is about 16%. In advantage, it provides a good memory usage, by the fact that particles which can be found in the same cell can also be placed next to each other in memory. Storing all particles for one cell in the same object allows better vectorization. [GST+19] [Tch20]

It is also possible to choose a cell size smaller than the cutoff radius. As a disadvantage, it increases the complexity of the implementation due to a larger number of cells that needs to be considered for each particle. A benefit of this approach is less unnecessary calculations at the cost of even more memory redundancy. The smaller the rectangular cells, the better the approximation of a circle. Smaller cells mean less space that needs to be searched. This is due to the fact that only those cells which approximate the form of the cutoff radius are considered. Therefore, the particles within the cells outside of the radius are not included in the amount of distance calculations. As all the particles within one cell are stored in the same object, smaller cells need more objects. This is creating more memory overhead and therefore another disadvantage. Also, each cell needs to memorize its neighboring cells relevant for distance calculations and hence need to store more cells when working with a smaller cell size. [MR99]

2.2.3. Verlet Lists

The idea of storing particles and their positions in a table or list was already introduced by Loup Verlet in 1967 in [Ver67]. The Verlet Lists algorithm displays another option to scale

down the amount of distance calculations. The main idea is to store neighboring particles which are within a certain radius, also called skin, in a list. This skin is slightly larger than the cutoff radius to include particles that might move into the cutoff radius in one of the following iterations. The Verlet skin is marked in yellow and can be seen in Figure 2.1c around the cutoff radius marked in red. This approach is beneficial for reusing the data of the Verlet Lists more often. Since the initialization of the Verlet Lists is complex, as well as computationally heavy, a reuse of them is advantageous [YWLC04] [CD90].

Distance calculations are still needed to check whether a particle is located inside the cutoff radius, but they are only computed on the Verlet List. The probability that a pair, whose distance was evaluated, has a distance smaller than the cutoff radius is depending on the skin, but for certain skin sizes it offers a better hit rate. For example for a skin that is 20% larger than the cutoff radius, the probability of a small enough distance would be 58%. Hence, the Verlet Lists algorithm has much less unnecessary distance calculations than the Linked Cells algorithm. On the other hand, it needs a lot of updating regarding the neighbor lists in case the particles are moving fast. The algorithm is also challenging for the memory usage, since the Verlet Lists are losing the benefit of vectorization, which the Linked Cells approach offered, as explained in Subsection 2.2.2. [GST+19]

For the initialization of all Verlet Lists at the start of a simulation and each time the Verlet Lists have to be rebuilt, it would be necessary to calculate the distances between all particles. This would lead to a complexity of $O(N^2)$ like the Direct Sum approach, described in Subsection 2.2.1. To decrease the amount of distance calculations, the Verlet Lists can be build on top of the Linked Cells algorithm by initially only calculating distances for particles within the neighboring cells. [GST+19] [YWLC04]

2.2.4. Verlet Cluster Lists

An approach to optimize the Verlet Lists algorithm is to store a certain number of particles within a fixed cluster rather than a list. This cluster allows to access several particles at once, since they are each other's neighbors. It is thus more efficient to iterate over those clusters than iterating over each list per particle. The Verlet Cluster Lists algorithm uses the fact that particles close to one another have overlapping Verlet Lists and aims on reducing redundancy and thereby the memory heaviness of the classic Verlet Lists approach. [PH13]

As it can be seen in Figure 2.1d, those clusters are all formed with the same number of particles. To select particles building the same cluster, a 2D grid is applied on the 3D domain, forming towers on the z-axis. Those towers can be seen in Figure 2.1d, separated by the thin lines. Within those towers the particles are sorted along the z-axis. In the resulting order, the particles are iterated and a fixed number N is grouped into a cluster.

One particle, in our case marked in red, can interact with several clusters. To get all interacting clusters, a skin for each particle within the cluster is calculated. The skins are then merged to one large skin and thus no longer have the form of a radius. The large skin surrounding all clusters can be seen in yellow in Figure 2.1d. Within that skin the distance between the red particle and all other particles is computed. Force calculations will then only be applied to those light green particles within the red cutoff radius. While optimizing the memory usage, the Verlet Cluster Lists algorithm comes at the cost of a much more complex implementation.

2.3. Close-Packing of Equal Spheres

To guarantee the physical correctness of the examined scenarios, the upper bound for the density of a scenario needed to be considered. Therefore, the closest possible packing of equal solid spheres will be explained in this chapter. Originally based on the idea of packing cannon balls as closely as possible [Ang90], different approaches have been made to calculate the closest solution for ordering equal spheres.

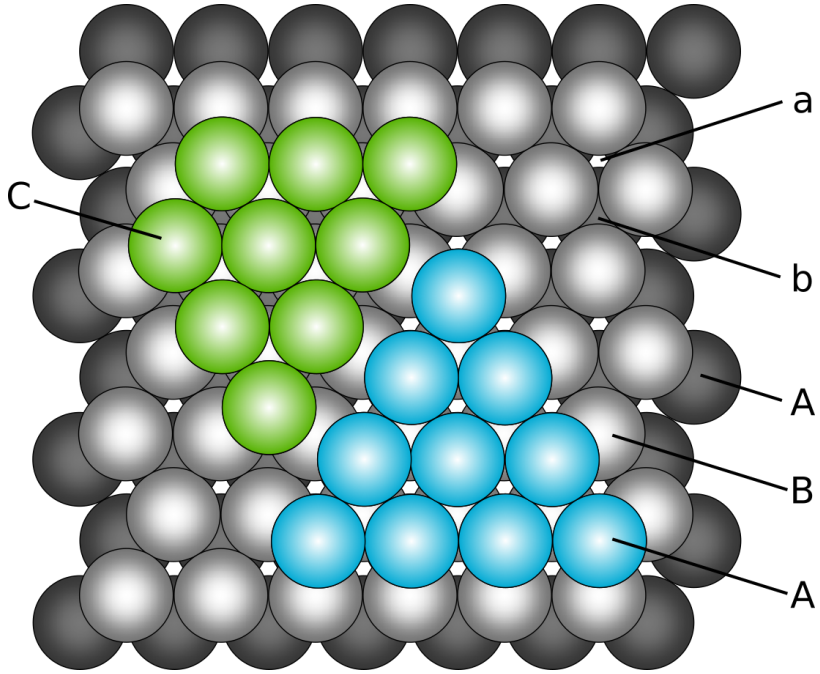
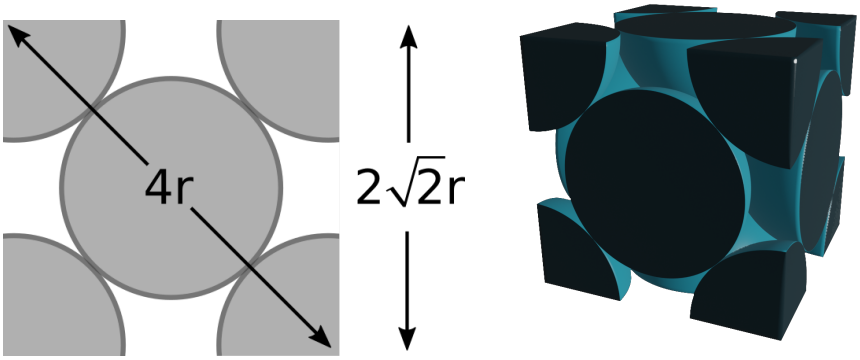


Figure 2.2.: The figure is based on [Hun09] and shows the two options for the closest packing of equal spheres without any overlapping bodies. In light green the face centered cuboid (fcc) approach is visualized. The hexagonal close packed (hcp) can be seen in light blue. The first layer A in dark grey is equal for both approaches, as well as the second layer B in light grey. The third layer of fcc, which is marked by C, places its light green spheres in the gaps shown by a. On top of that it would again continue with a layer of A, creating the pattern $ABCABC$. Hcp creates the pattern $ABAB$, by placing the light blue spheres of the third layer in the gaps marked by b.

In the present thesis the focus lies on the structures of the face centered cuboid (fcc) and hexagonal close packed (hcp). Both allowing the closest possible packing of equal spheres. In the later examined scenarios this concept is always applied within a domain that has a cuboid form. As seen in Figure 2.2, both fcc and hcp start with layer A for placing the spheres. Then fcc and hcp equally continue with layer B, placing the spheres into the gaps originated by layer A. Now, they differentiate regarding the third layer. The third layer for fcc is visualized in light green and the third layer for hcp is colored in light blue. Depending into which gaps one chooses to place the spheres, the third layer is shifted in proportion to both layer A and B. If it is shifted, the spheres of the third layer are placed in the gaps

marked by a in Figure 2.2. They either create a new layer C and are placed in the gaps marked by a or the structure continues with layer A again. If the third layer is equal to A, its spheres are placed in the gaps marked by b in Figure 2.2. Therefore, fcc creates a pattern of $ABCABC$ which continues in this order, while hcp sticks to the pattern of $ABAB$. [Hun09]



(a) 2D Unit Cell (b) 3D Unit Cell

Figure 2.3.: Both figures are showing the unit cell of the closest packing of an equal spheres structure. The unit cell is needed to calculate the density for such a structure. In Figure 2.3a one can see the side of such a unit cell, which is used to calculate its volume. A three-dimensional representation of the unit cell can be observed in Figure 2.3b.

Both described packing structures allow a maximum packing of spheres which are equal to the bodies described in this thesis. It is therefore not of importance which of the structures are applied, but rather to acknowledge that both maximize the density for the placing of solid spheres. The unit cells of both structures contain four spheres [Hun09]. The packing density can be evaluated by using the formula for calculating the density within a unit cell. A side-view of this cell can be observed in Figure 2.3a. The unit cell contains eight $1/8$-spheres and six $1/2$-spheres as it can be seen in Figure 2.3b. The $1/8$-spheres are placed on the corners of the unit cell while the $1/2$-spheres are on each side of the cell. Therefore, the volume of those spheres and the volume of the unit cell need to be calculated in order to calculate the density:

$$V_{spheres} = \left(8 \cdot \frac{1}{8} + 6 \cdot \frac{1}{2}\right) \frac{4\pi}{3} r^3 \tag{2.2}$$

$$V_{unitcell} = \left(2\sqrt{2}r\right)^3 \tag{2.3}$$

$$\varrho = \frac{V_{spheres}}{V_{unitcell}} = \frac{\pi}{3\sqrt{2}} = 0.74 \tag{2.4}$$

The calculated maximum density is 0.74 or 74% [Hun09] [MG12]. This density will be used as the maximum density for testing purposes.

Part II.

Technical Fundamentals

3. Code Base and Implementation

In this section an overview of the library AutoPas, on which this thesis is based on, will be given. Also an introduction into pairwise cell traversals of the domain, which name the different strategies to optimize the molecular dynamics simulations, is provided. The structure of a traversal will be explained on the basis of the c04HCP traversal. Furthermore, its implementation will shortly be described.

3.1. AutoPas

AutoPas is a C++ node-level performance library used for large scale simulations in the field of molecular dynamics. Its purpose is to minimize the amount of calculations needed for those simulations and optimize the algorithms framing them. In particular, it focuses on particle simulations, a type of N-body simulations. Those particles are placed in a three-dimensional area, where pairwise force calculations, described in Section 2.1, are applied on the examined particles. The library focuses on force calculations rather than time integration and boundary conditions, which have to be set by the user.

The open-source software[1] uses different data layouts to especially optimize the force calculations. Since they are the most compute intense [Eck14] part of a simulation, they are also a good entering point for optimizations. By differentiating between Array-of-Structures (AoS) and Structure-of-Arrays (SoA) it is able to choose the data layout that performs best for a certain scenario.

For optimizing these simulations, another option provided by AutoPas is to apply Newton's third law of motion (Newton 3), stating that for every force applied by a body to another one there is an equally strong counter force, applied by the second body to the first [NMC66]. Thus, half of the calculations are saved by updating the counter-particles accordingly when calculating the forces of one of the participating particles in a pairwise force calculation [GST+19]. Whereas Newton's third law can contribute hugely to optimizing the performance, it has a limited range of usage. First, it has to be supported by the potential that is about to be calculated. Second, it is not applicable for all parallelization techniques, causing race conditions within the data. Those race conditions occur when forces of a particle are being updated, while using the same particle for pairwise force calculations with another particle.

Additionally to the described options, an optimal parallelization strategy is chosen at run-time. These strategies will be discussed in detail in Section 3.2. By testing all possible strategies, the optimal strategy is applied for the next fixed amount of iterations, after which the strategies will be tested again. It includes testing the usage of Newton's third law or not and the preferred data layout during run-time. This procedure of finding the current optimal parallelization strategy is called auto-tuning. [GST+19]

[1] `https://github.com/AutoPas/AutoPas`

AutoPas uses a modular approach, where parallelization strategies are implemented based on containers. Each container is representing a particle search algorithm, of those explained in Section 2.2, or a modification of these algorithms. It is keeping iterators for cells and particles together and separating molecular dynamics calculations, such as the Lennard Jones Potential, described in Section 2.1, from other parts of the software. For the presented thesis especially the containers implementing different parallelization strategies are of relevance. The containers can be subdivided in those based on the Linked Cells algorithm, explained in Subsection 2.2.2, those based on Verlet Lists, seen in Subsection 2.2.3 and also VerletClusterLists, which are described in Subsection 2.2.4, as well as combinations of those algorithms.

3.2. c04HCP Traversal as an Example

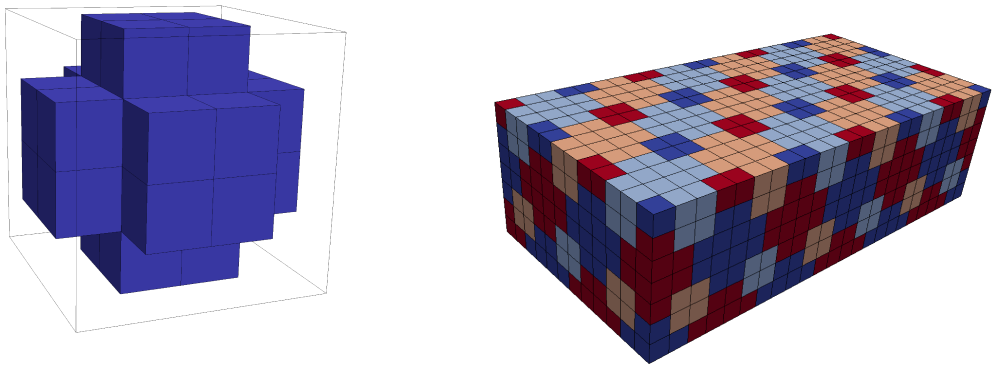
All containers described above support different traversals, which are various implementations of strategies for parallelization. They are applying the particle search algorithms, which were presented in Section 2.2. Traversals can be chosen automatically during run-time with the auto-tuning process described in 3.1. In this section, the structure of traversals will be explained by the example of the c04HCP traversal, which is applicable to the Linked Cells container.

For parallelizing within any container, one needs to prevent race conditions when using Newton 3 for optimization. Therefore, Newton 3 is turned off or one of the following two approaches to prevent race conditions is needed. This can either be a sliced-based or a color-scheme-based approach. Using the sliced approach means subdividing the three-dimensional space into equally large slices of cells where each thread gets to work on one slice at a time. The outer cells of a slice whose particles interact with the next slice of cells are locked temporarily, to prevent race conditions. In contrast, for the color-scheme-based approach, a fixed pattern of cells is assigned a certain color. The colors are processed sequentially while calculations are parallelized within one color. This approach prevents race conditions by avoiding overlapping blocks of the same color. Blocks of the same color are not overlapping due to barriers of other colors between them. [GST+19]

One approach which applies this coloring-scheme is the c08 traversal, using eight colors to prevent race conditions. The idea is to assign blocks that are one cell large or larger but always have the form of a cuboid. Then, each color is computed separately but in parallel. Since cells of the same color never overlap, it is possible to process all of them in parallel. To improve memory reuse, one would need to decrease the number of colors by increasing the blocks per color [Tch20].

In [Tch20] a four color scheme called c04 is described. This optimizes the concept of c08 needing eight colors. It has a better memory reuse, since the packages processed are larger. This could also be done by enlarging the blocks of c08 and is therefore not the main benefit of the c04 approach. It also needs to lock less threads, due to less barriers between the colors. Therefore, the coloring-scheme needs less synchronization and shorter times of waiting to unlock a thread, hence it is more efficient. On the downside, it comes at the cost of being more coarse-grained, since the cuboids are larger. The first approach to a four coloring scheme was to use a form looking like a three dimensional plus sign for each color, containing 32 cells per block, which can be seen in Figure 3.1a. These blocks are building

two Cartesian grids. The first grid is built out of two colors, marked as light blue and light red in Figure 3.1b. The first grid is leaving out spaces where the other two colors, in this case dark blue and dark red, would fit in. While this approach enables a high reuse of cell data and is able to work with at most four memory buffers, it also displays a very complex traversal and the shape of the blocks are posing a challenge. Thus, the suggestion of a form with less complexity was made. [Tch20]



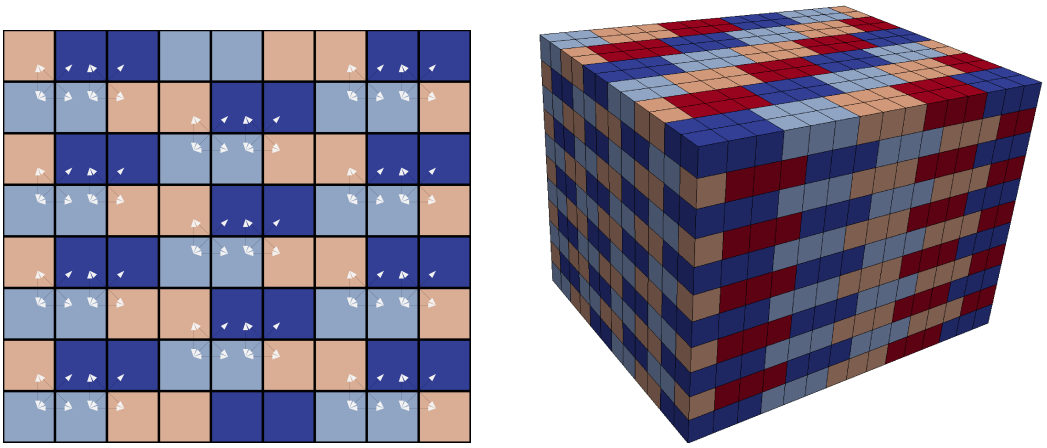
(a) Single c04 Block (b) Whole c04 Domain

Figure 3.1.: The c04 parallelization strategy splits the domain into plus-shaped blocks. The blocks are aggregated by 32 cells and can be seen in Figure 3.1a. In Figure 3.1b the whole domain is shown. It consists of two Cartesian grids, one is build by the dark blue and dark red blocks and the other by the light blue and the light red blocks. The light colored grid is then shifted, so both grids fit into one another. Thus the strategy applies four colors for parallelization purposes. Both figures can be found in [Tch20].

By using a simple cuboid, a traversal has a much easier form to process. This approach is called c04HCP due to the centers of the boxes forming a hexagonal close packing pattern (HCP). It may not be confused with the hcp structure explained in Section 2.3, even though the base idea of placing forms remains the same. For the HCP approach, D+1 colors are needed with D being the number of dimensions used. In a three-dimensional area each block consists of six cells. Those six cells are composed by $1 \times 2 \times 3$ cells, referencing to the x-axis, the y-axis and the z-axis and can be seen in Figure 3.2b. Since the blocks are very small, the c04HCP approach is less coarse-grained than the previously explained four-color approach c04. On the downside, it comes with a more complex implementation because of the constant need of shifting the blocks, of which is less needed in the classic c04 approach. In c04 one had to simply shift one of the grids, while the blocks in c04HCP do not simply repeat their pattern, but need a different pattern in each dimension. [Tch20]

To make sure no blocks of the same color are touching, the following approach is applied: Along the first dimension, the two colors of light red and light blue are simply alternating. This is visualized in Figure 3.2a. Along the second dimension, three colors are alternating. As it can be seen, a dark blue is added as the third color. There, one can also observe how

the light blue boxes interact with the other colors and do not touch other light blue boxes. Along the third dimension four colors are alternating, as shown in Figure 3.2b, where the dark red is added. In each additional dimension, the additional color is needed to avoid diagonal links. In other words: "The scheme should extend to D dimensions by taking blocks of size $D \times (D-1) \times ... \times 1$ and periodic tilings with periods of $D+1, D, ..., 2$." [Tch20]



(a) c03HCP in 2D (b) c04HCP in 3D

Figure 3.2.: In Figure 3.2a a two-dimensional version of the c04HCP strategy can be seen. In a two-dimensional domain only three colors are needed. It can be seen that on the x-axis two colors are needed, so one color never touches cells of the same color. On the y-axis three colors are alternating for the same purpose. The figure is based on [Tch20]. Figure 3.2b, which can be found in [Tch20], visualizes a three-dimensional domain. In a three-dimensional domain four colors are needed to prevent each color from having blocks of the same color as a neighbor.

For each color the pattern of the first dimension is then shifted in the second dimension and the pattern of the second dimension is shifted in the third dimension, so blocks of the same color may never touch. To be precise, every second row the pattern of the dark blue boxes, for example, is shifted by three positions and then shifted back. One can take a closer look at this pattern in Figure 3.2b. In the third dimension the pattern of a color is once placed at its original coordinates, it is then back-shifted outside the boundary by four in proportion to its original x-axis coordinate. When the color is shifted a third time, it is back-shifted by two in proportion to its original x-axis coordinate. For example: one can see that the dark blue box which can be found at the start of the z-axis, is also placed at the end of the domain again, when the pattern starts to repeat itself. In a three-dimensional domain, also a fourth color is needed. As shown in Figure 3.2b, the dark red boxes start to appear in the second row along the z-axis. In this way partial blocks are considered for calculation. To make sure that all partially used blocks are included, the starting points of each additional color are shifted outside the boundaries, too, as explained earlier.

The described differences between c08, c04 and c04HCP display a classic consideration with most traversals: some are easier to implement but may have a worse performance, some are better for memory-reuse but often come at the cost of being more coarse-grained.

Thus, one has to depict what is most important for the scenario being simulated. In general, c04HCP eliminates some restrictions of the c04 traversal, since it is applicable to all sizes and reduces the overhead of computing the cutouts of the cubes surrounding the actual c04-blocks. It also has the upside of a better memory reuse than c08, still coming at the cost of being a bit more coarse-grained. Since the c04HCP traversal is build on several previous traversals it displays a good example of how new traversals are conceived and how they are originally based on one of the main algorithms. In this case, the traversal is based on the Linked Cells approach and also applying previous testing and concepts of other traversals without replacing them.

4. Methodology and Measurements

In this chapter, the computational platforms used for testing are described. Also the methodology for the measurements of run-time, which are the base for the plots, will be presented. For the later explained experiments, a measurement of homogeneity was needed and thus added. The calculation of the homogeneity will also be presented. For the measurements, fixed configurations are used. Those configurations differentiate in overall domain size, the number of particles, the density - which depends on both domain size and the number of particles - and the homogeneity of a scenario. In general, it would also possible to configure more parameters, but as a starting point, those four variables were modified and tested.

4.1. Computational Platforms

All scenarios were run both on a Coffee Lake platform and on the CooLMUC2, a Haswell platform, of the Linux Cluster by the Leibniz Supercomputing Center. The Coffee Lake platform is using an overclocked Intel i7 8700k with 5043 MHz and a Coffee Lake-S architecture. It is running on six cores, twelve with enabled hyper threading and 32GB RAM. For the tests, twelve threads were chosen, if not stated otherwise. The CooLMUC2 has an Haswell Architecture with 28 cores, two hyper threads per core and 64GB RAM[1]. The scenarios were each tested once with 28 and once with 56 threads on the Haswell platform, except for tests examining increasing numbers of threads. All tests were processed with the GCC 7.5.

4.2. Run-Time and Performance Measurement

The performance in this thesis is measured by the time to solution of an experiment, here equivalently used to run-time. The run-time is measured in nanoseconds by collecting the time spent with the pairwise iteration and force calculation of all particles. In the run-time also the building of neighbor lists for one iteration for a certain configuration is included. The rebuilding of the neighbor lists is especially relevant for the Verlet Lists based traversals. Those collected times are then summed up per scenario configuration and used for calculating a mean value for this certain configuration. All scenarios where tested five times per traversal and with 50 iterations per test. The rebuild frequency was 20. The number of particles is either set fixed in advance or calculated by multiplying the numbers of particles per dimension in case a grid structure is used. The domain size, which is the volume of the whole domain, is given by multiplying the length of each dimension. Both are used to calculate the density, which is the domain size divided by the overall number of particles.

[1] `https://doku.lrz.de/display/PUBLIC/CoolMUC-2`

In the following chapters, the time to solution is linked to the different described factors. The smaller the run-time is, the better does the traversal perform in the present experiments. Therefore, a decreasing run-time is equal to an increasing performance. An additional scale for performance is the number of million force updates per second (MFUPS). It will be applied to evaluate the performance dependent of the amount of threads. Thus, both computational platforms can be compared. While there are other possible benchmarks, such as the needed memory, the main focus in this thesis is on the run-time. This benchmark will also be used to compare a certain traversal on different types of computational platforms and by the number of threads.

4.3. Measurement of Homogeneity

Homogeneity is a factor that was examined regarding its influence on the performance of the traversals. While in most tested scenarios the particles were equally distributed, some were set up with particles grouping in a certain corner or generated by a Gauss distribution, leading to different densities over the whole domain. To measure the homogeneity, the domain is subdivided into cells. The size of the cells are dependent on the overall amount of particles. For a perfectly homogeneous scenario, each cell would contain ten particles. As a measurement of homogeneity the standard distribution was chosen.

$$\sigma = \sqrt{\sigma^2} \tag{4.1}$$

$$\sigma^2 = \frac{\sum_{i=1}^{N} (\varrho_i - \bar{\varrho})^2}{N} \tag{4.2}$$

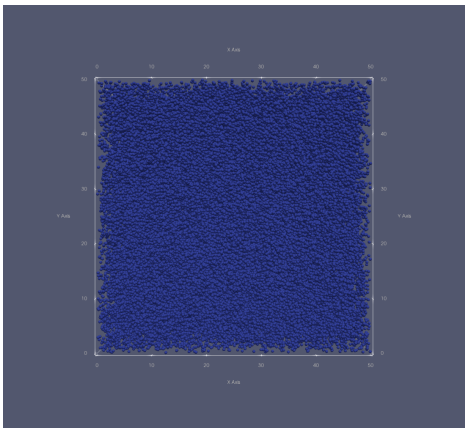
In Equation 4.1 one can see that the standard deviation is calculated by using the variance, by the Equation 4.2. The density of each cell ϱ_i is calculated by the amount of particles in it divided by the size of its cell. The mean density $\bar{\varrho}$ is calculated by dividing the overall amount of particles by the domain size. The densities of all cells, as well as the mean density are then inserted in the formula to compute the variance. From here, the standard deviation is derived.

Part III.

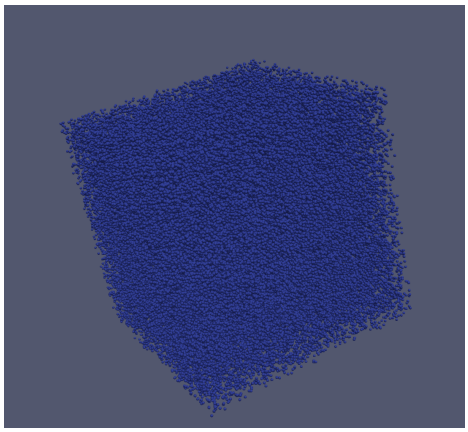
Analysis of Different Factors on the Performance of Traversals

5. Impact of a Scenario's Physical Structure

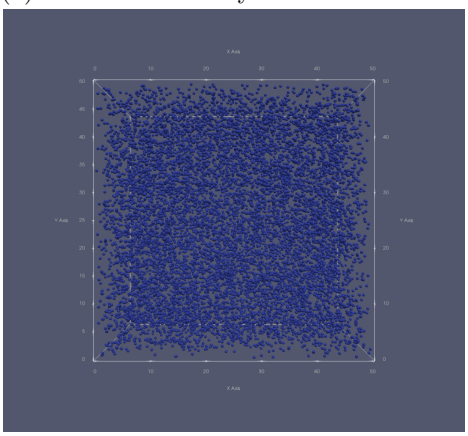
In this chapter, different scenarios with a changing number of particles, density and domain size will be examined regarding how their structure influences the run-time of certain traversals. The tests will be used to conclude which traversal is to be preferred for which structure.



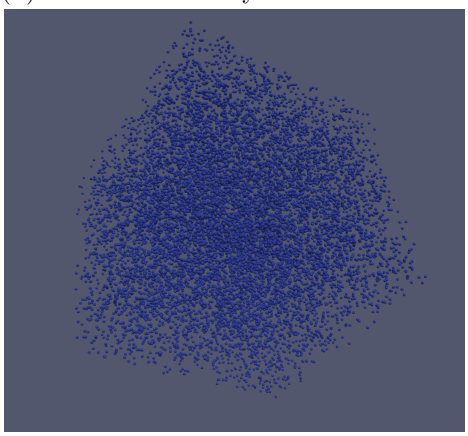
(a) Maximum Density of 0.74 Front View



(b) Maximum Density of 0.74 Side View



(c) Low Density of 0.1 Front View



(d) Low Density of 0.1 Side View

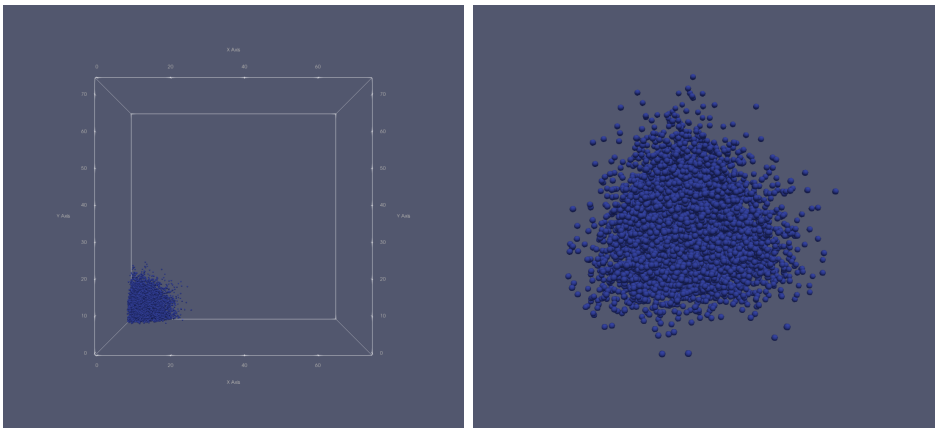
Figure 5.1.: Scenarios with different densities and amounts of particles. The domain is shown as the white frame surrounding the particles. The measurements for the domain of all seen scenarios are $50 \times 50 \times 50$. Figure 5.1a and Figure 5.1b are showing a scenario with 92.500 particles with a density of 0.74. Figure 5.1c and Figure 5.1d visualize 12.500 particles, while the scenario has a density of 0.1.

As a starting point a scenario with a maximum density will be presented. The upper bound for the density was calculated in Section 2.3. The scenario is tested with 92.500 up to 740.000 particles and a domain size increasing from $50 \times 50 \times 50$ up to $100 \times 100 \times 100$ with a constant density of 0.74. It is visualized in Figure 5.1a and Figure 5.1b. To compare the findings, a scenario with the same domain size but a number of particles increasing from 31.250 up to 250.000 was run. This second testing scenario with a lower density of 0.25 can be seen in Figure 5.1c and Figure 5.1d.

Furthermore to investigate the influence of a larger domain size, a scenario with a constant number of particles, in particular 500.000, and an increasing domain size was tested. To test an approximately maximum density of 0.74, the number of particles was divided by 0.74. By applying the cube root, this led to a maximum domain length of 87.75 in each direction. To ensure a large enough domain, the length was rounded up to 88 and then counted upwards in steps of twenty. Therefore, the domain measurements grew from $88 \times 88 \times 88$ to $188 \times 188 \times 188$. Thereby, the density varied from 0.73 to 0.08, which means starting with a high density and ending with a very low one.

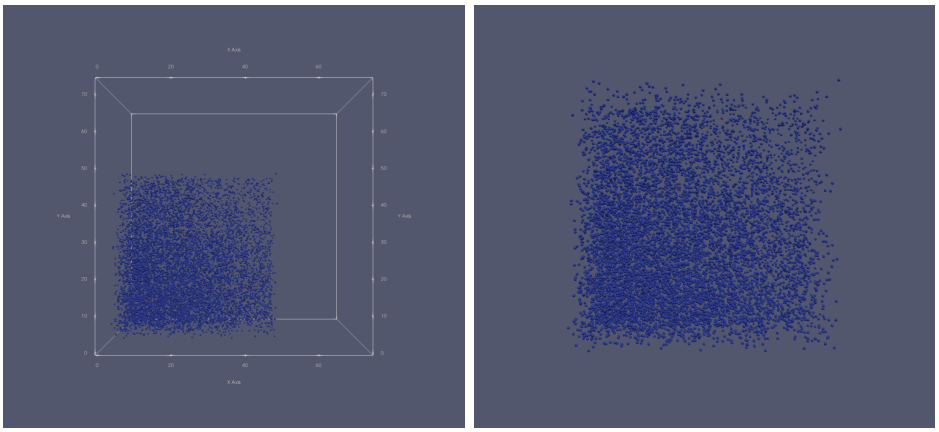
All above described scenarios are homogeneous due to a low standard deviation of homogeneity. To test whether the homogeneity of a scenario has an impact on the runtime and on how the traversals perform, scenarios with a different standard deviation of homogeneity were run. The measurement of homogeneity was explained in Section 4.3. For those experiments, a scenario with one million particles was chosen. The particles were distributed in the form of a cuboid within a domain with a constant size. The place of the cuboid is alternating, as well as the spacing between the particles. The standard deviation of the homogeneity reached from 0.048 to 0.957.

Moreover, in-homogeneous scenarios with a small number of particles, 10.000 to be precise, were tested. For these, a different form, which is generated based on the Gauss distribution, was chosen. The tests that ran within a domain size of $75 \times 75 \times 75$, investigated a standard deviation of homogeneity from 0.044 to 0.734. An in-homogeneous scenario with a medium standard deviation of 0.404 can be observed in Figure 5.2a and in Figure 5.2b. In Figure 5.2c and Figure 5.2d, a homogeneous scenario with a standard deviation of 0.044 is visualized. Here one can see the particles wider spread through the domain.



(a) High Standard Deviation of Homogeneity (0.404) Whole Domain

(b) High Standard Deviation of Homogeneity (0.404) Close View



(c) Low Standard Deviation of Homogeneity (0.044) Whole Domain

(d) Low Standard Deviation of Homogeneity (0.044) Close View

Figure 5.2.: Homogeneous and in-homogeneous scenarios within the same domain size generated with a Gauss distribution. Since the domain size remains the same, the wider spread the particles are, the more homogeneous is the scenario. The amount of 10.000 particles also remains the same. The white frame shows the border of the domain measurements of $75 \times 75 \times 75$. Figure 5.2a and Figure 5.2b both show a scenario where all particles are mostly situated in one corner of the domain. As Figure 5.2c and Figure 5.2d are showing, the second scenario is wider spread and thus more homogeneously distributed within the domain.

5.1. A High Number of Particles Favors Verlet Lists

In general, the traversals start to group by container with an increasing number of particles. The gaps between the containers widen when the amount of particles grows, which is visualized in Figure 5.3a and Figure 5.3c. In the figure each container is plotted in a different color. For a high number of particles, to be precise 30.000 or more particles, and a density of at least 0.25, the verletListsCells container performs best on all hardware variations. The container is shown in red. The variations depending on the computational platform will be discussed in detail in Chapter 6.

Overall, the higher the amount of particles, the greater the gap between verletListsCells container and the others. The scenarios were tested with a constantly high density of 0.74, a constantly lower density set to 0.25 and a very low one of 0.1. Since the finding is stable for a low and a high density, it is to be linked to a high number of particles.

The explicit traversal, which performs better, is dependent on the number of particles, the density and the hardware. On the Haswell platform using 28 threads, the verlet-sliced starts as the slowest within its container for a low density of 0.25 and is the fastest in the end, as seen in Figure 5.3c. The verlet-sliced traversal is shown by the dot-markers in red. It is also the best performing traversal overall when tested with an even lower density of 0.1, which can be observed in Figure 5.3d. For a high density of 0.74 on the same computational platform, the verlet-c18 traversal always performs best within its container even for a smaller amount of particles. It can be seen by the red star-markers in Figure 5.3a. Therefore, one

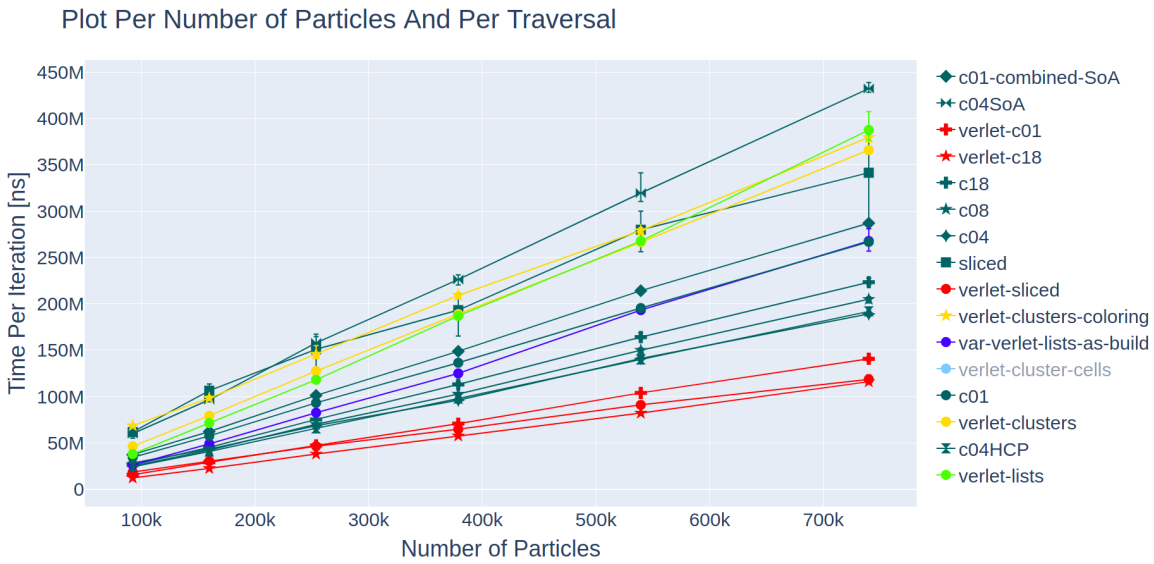
could assume that for a certain number of particles the verlet-c18 traversal outperforms the verlet-sliced traversal. However, if the 250.000 particles with a low density are compared with 253.000 and a high density, the findings show a different conclusion. The verlet-c18 traversal performs better for a high density, whereas for a low density the verlet-sliced traversal is still favorable. This leads to the hypothesis that a color-scheme-based parallelization approach is suited better for a high density, while a sliced-based traversal is favorable for a low density.

Those findings slightly differentiate when the hardware is changed. For a low density of 0.25 the Haswell platform using 56 threads favors the verlet-c01 traversal, which is color-scheme-based. Tested with a density of 0.1, the verlet-sliced traversal is again the best performing, supporting the theory of a lower density being beneficial of a sliced-based approach. In contrast, the Coffee Lake platform still favors the verlet-sliced approach for a high density, which one can observe in Figure 5.3b. Those deviations will be further discussed in Chapter 6, but do in general not contradict the conclusion of a color-scheme-based approach being favored by a high density and a sliced-based approach being favored by a low density scenario.

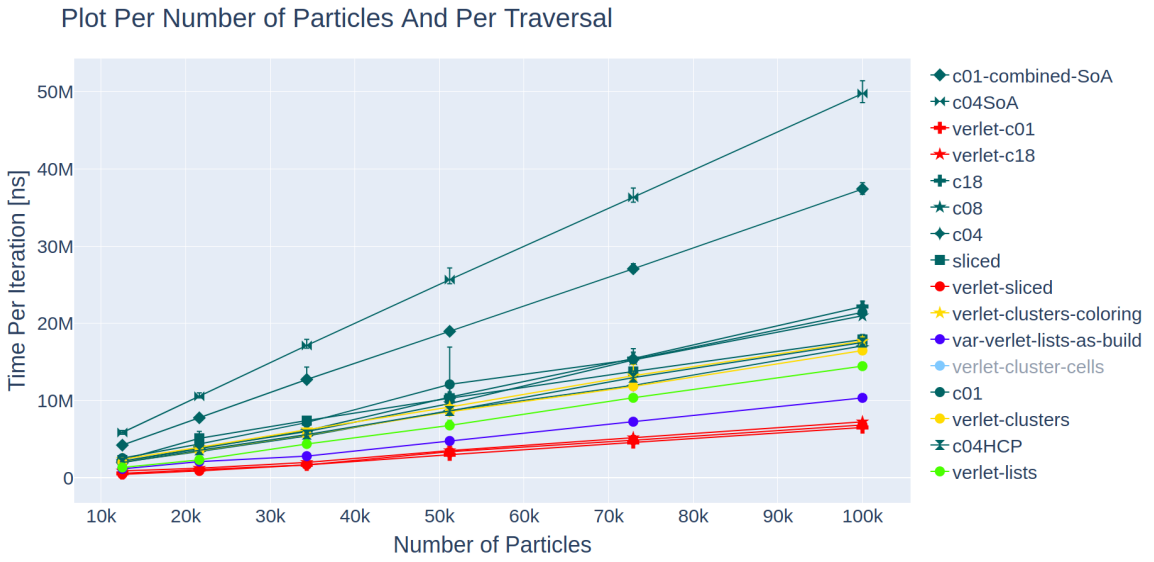
The described observations are partially supported by the linkedCells container, which can be seen in dark green in Figure 5.3a, for instance. While the order of the color-schemed traversals within this container remains when decreasing the density, the sliced traversal changes its proportional performance. It can be seen in Figure 5.3d that for a lower density of 0.1 the sliced traversal performs better than for a high density, which is shown in Figure 5.3a. Nevertheless, there are still color-schemed traversals within this container, which have a lower run-time. Within the linkedCells container the c04 traversal performs best, closely followed by the c04HCP traversal, independent of density and hardware. At least on the Coffee Lake platform, the sliced traversal comes third for a low density of 0.1, right after the c04 and the c04HCP. This will be further discussed in Chapter 6.

The worse performance of sliced-based traversals when looked at a high density can be explained by the greater amount of locks needed. Pairwise calculations on the barriers of two slices need to wait for the next slice to be unlocked. When using a stable amount of slices the number of particles per slice increases and therefore the waiting time. Hence, the increased time to solution for a high density and a sliced-based traversal.

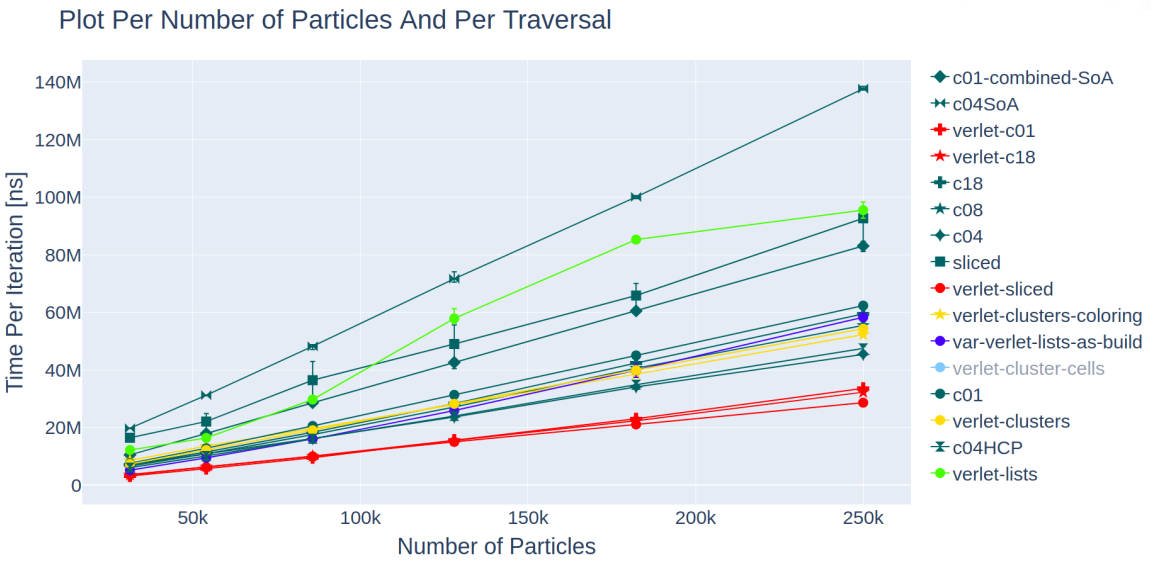
The decreasing performance of color-schemed traversals for scenarios with a low density can be explained by the overhead of empty cells, which need to be processed. This means in detail: For sliced-based traversals, each thread gets to process one slice at a time, while for color-schemed traversals a thread gets assigned one block of cells within a color. Since the slices are more coarse-grained, they are more likely to contain particles at all in comparison to the smaller blocks of the color-schemed approach. This creates a disadvantage for the color-schemed traversals since threads are assigned unnecessarily to blocks without particles in it.



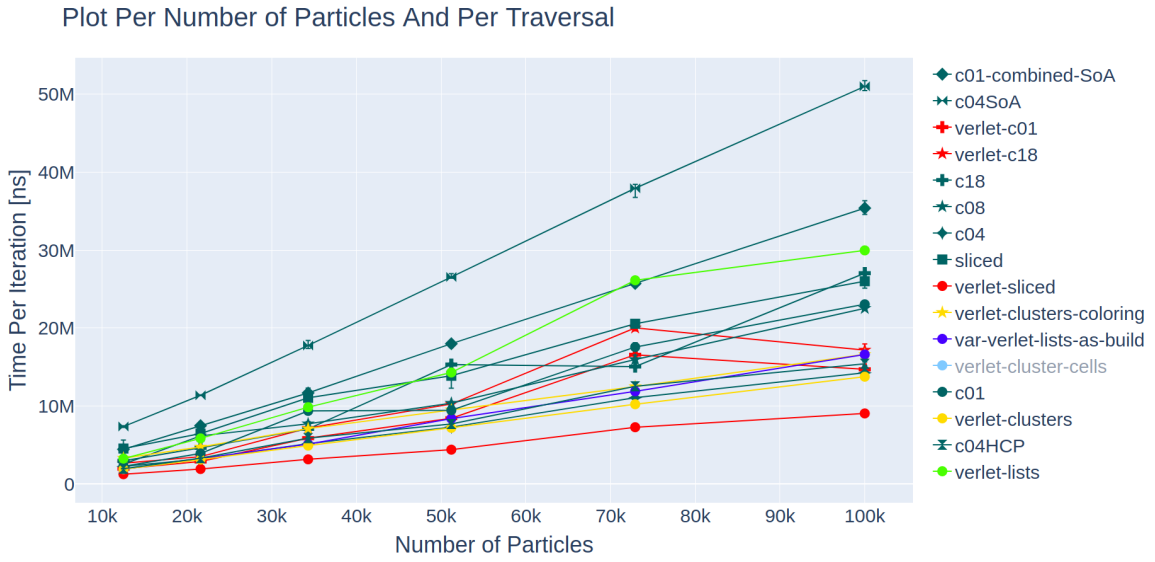
(a) High Number of Particles With a 0.74 Density on Haswell Platform



(b) High Number of Particles With a 0.74 Density on Coffee Lake Platform



(c) High Number of Particles With a 0.25 Density on Haswell Platform



(d) High Number of Particles With a 0.1 Density on Haswell Platform

Figure 5.3.: The figures show the development of an increasing and overall high number of particles. The domain size is increasing from $50 \times 50 \times 50$ to $100 \times 100 \times 100$ in proportion to the number of particles. Figure 5.3a shows experiments run on the Haswell platform, using 28 threads. For the maximum density seen there, the number of particles is increasing from 92.500 to 740.000 with a density of 0.74 which stays constant. The same scenario tested on the Coffee Lake platform with twelve threads is shown in Figure 5.3b. The visualization of a 0.25 density again on the Haswell platform with 28 threads can be seen in Figure 5.3c and increases the number of particles from 31.250 to 250.000. The number of particles is increasing from 12.500 to 100.000 for the tests having a density of 0.1, which is shown in Figure 5.3d.

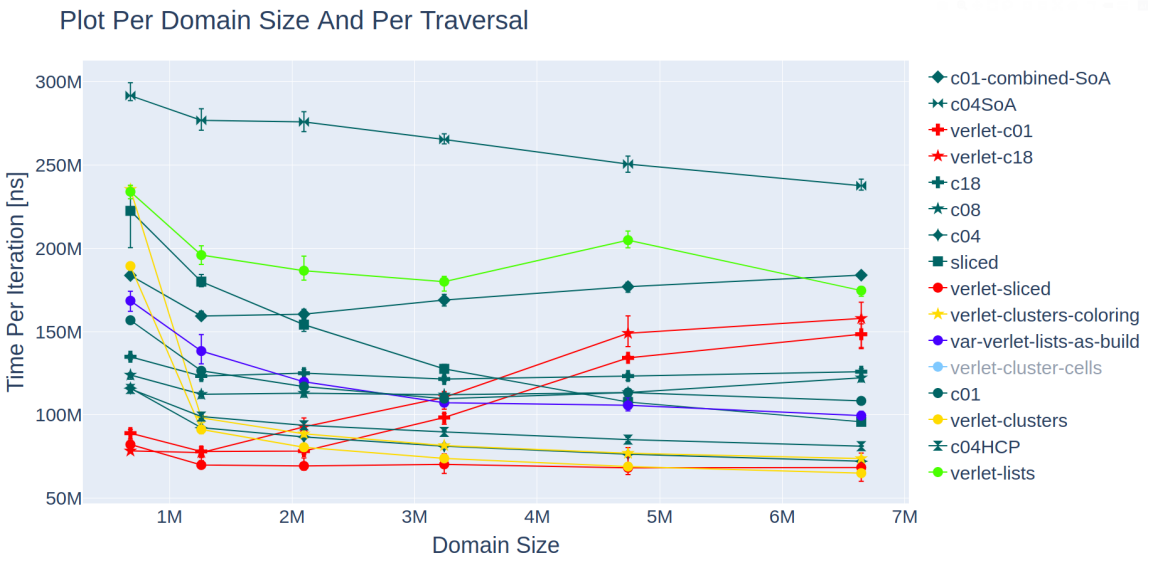
5.2. A Large Domain Size Prefers a Verlet Cluster Lists Approach

For an amount of 500.000 particles with an increasing domain size and therefore a decreasing density, the run-time no longer depends on whole containers but rather on single traversals, as it can be seen in Figure 5.4a. The size of the domain increased from $88 \times 88 \times 88$ to $188 \times 188 \times 188$. Those measurements were, as described before in Chapter 5, calculated by the amount of particles and the aspired density. Regarding the verletClusterLists container which is based on the the Verlet Cluster Lists approach, explained in Subsection 2.2.4, a decreasing time to solution can be observed. Leading to a point where the verlet-clusters traversal is the fastest on the Haswell platform using 28 threads. Its run-time is visualized as the lower yellow graph in Figure 5.4a. This leads to the assumption that a larger domain size favors the Verlet Cluster Lists algorithm. Similar findings can be seen on the Haswell platform when testing with 56 threads. For the Coffee Lake platform the observations again differentiate and will be discussed in Chapter 6.

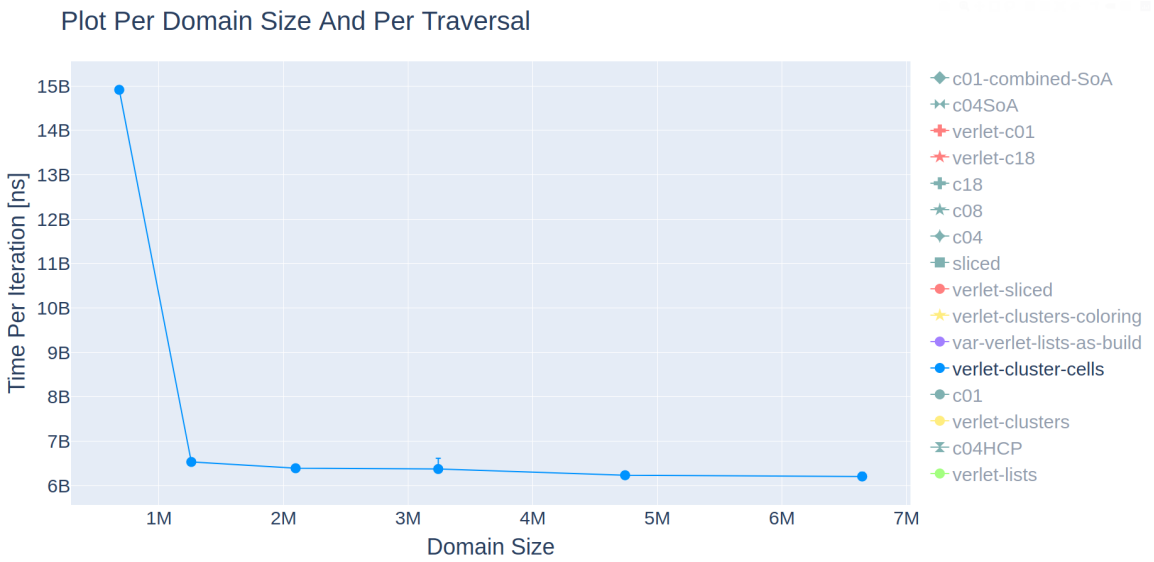
It is also noticeable that the verlet-cluster-cells traversal, which is the only tested traversal within the verletClusterCells container, improves with a larger domain size accompanied by a lower density. As it can be seen in Figure 5.4b, visualized with the neon green line, a larger domain size decreases the time to solution. As visualized in Figure 5.4c, for a density below 0.4, the run-time is kept steady. It is considerably higher for of a density of 0.734, close to the maximum density of 0.74. This result supports the hypothesis of a larger domain size with a low density favoring a Verlet Cluster Lists based approach.

As for a high density and a domain with the measurements of $88 \times 88 \times 88$ the verlet-c18 traversal, based on the verletListsCells container, is the fastest on the Haswell platform using 28 threads, but its performance changes with a larger domain size. Within the verletListsCells container, which was the ideal container prior to increasing the domain size, only the verlet-sliced traversal remains stable on all computational platforms. It quickly outperforms the color-scheme-based traversal on the Haswell platform using 28 or 56 threads and keeps its run-time on the same level independently of the domain size. While for a high density the verletListsCells container is overall favorable, both color-schemed traversals within this container are therefore increasing their run-time with a higher domain size and a thus lower density.

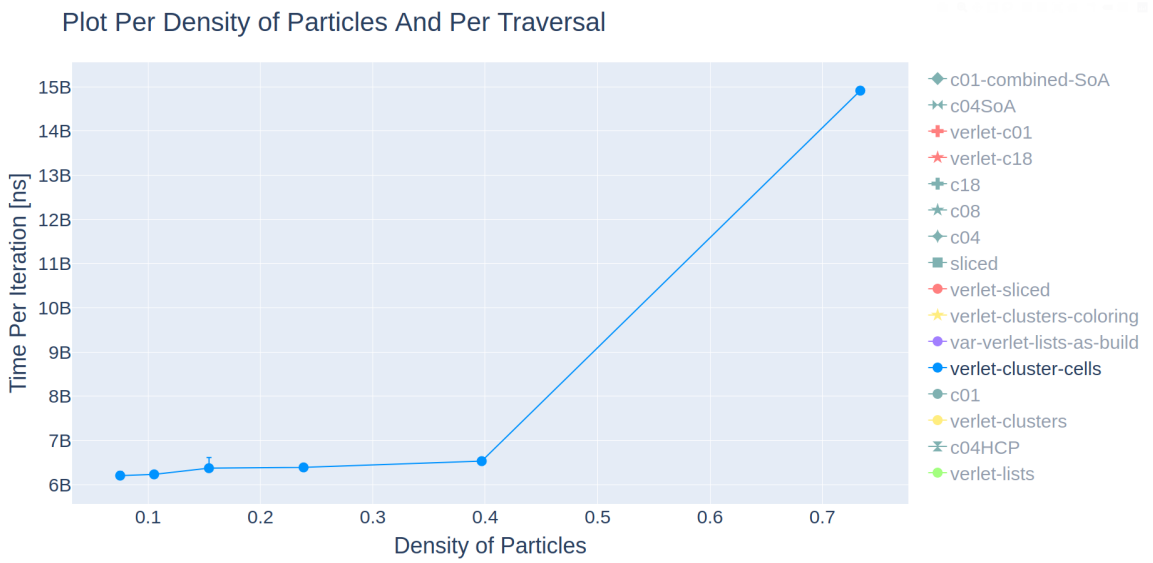
Only on the Coffee Lake platform this container remains stable as it can be seen in Figure 5.4d. The verletListsCells container remains stable and the run-time for the verlet-sliced traversal as well as the verlet-c01 traversal are even improving with a higher domain size. Nevertheless, one can see a decreased run-time for the verletCluserLists container seen in yellow in Figure 5.4d. The findings for the verlet-cluster-cells traversal are similar to those seen on the Haswell platform plots. Therefore, even if the Verlet Lists based traversals are still performing better than all other traversals on the Coffee Lake platform, a huge improvement for the Verlet Cluster Lists based traversals can be derived from the tests. It is thus overall to assume that the Verlet Cluster Lists based approaches are performing better for larger domain sizes combined with a low density. A reason for this behavior could be that wider spread particles, which belong to other clusters, are not considered and particles close to the currently computed particle already belong to its cluster.



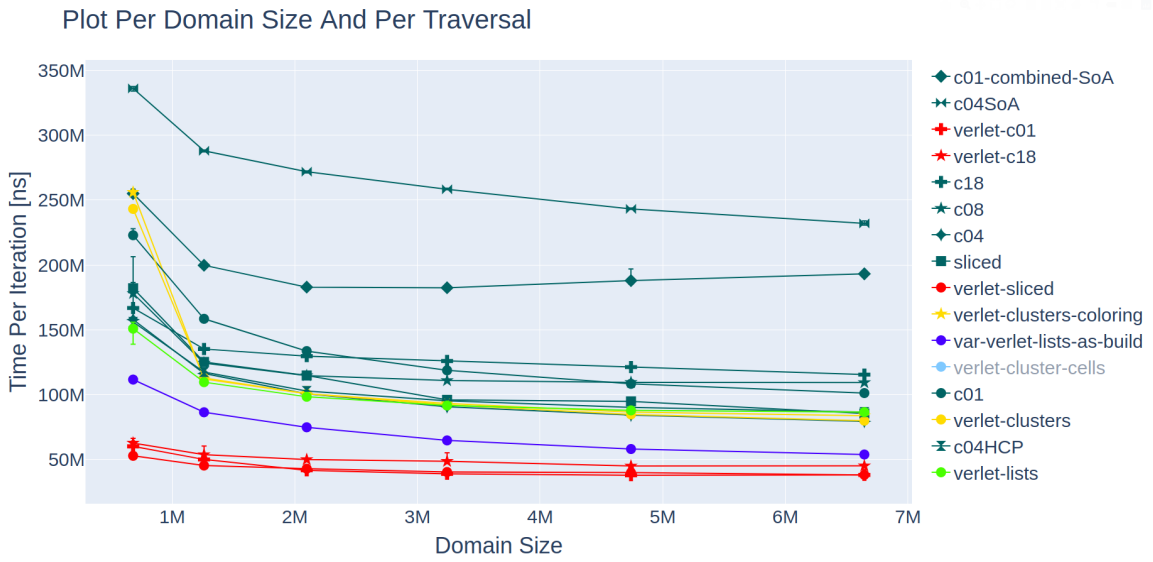
(a) Development of an Increasing Domain Size on the Haswell Platform



(b) Development of verlet-cluster-cells Traversal With Increasing Domain Size on the Haswell Platform



(c) Development of verlet-cluster-cells Traversal With Increasing Density on the Haswell Platform



(d) Development of an Increasing Domain Size on the Coffee Lake Platform

Figure 5.4.: Development of an increasing domain size with a constant number of particles and a therefore decreasing density. The density varies from 0.734 to 0.075 proportionally to the growing domain, while the amount of particles is 500.000 for all tests. In Figure 5.4a the development depending on the domain size on the Haswell platform tested with 28 threads can be observed. There, all traversals are shown, except for the verlet-cluster-cells traversal. This can be seen in Figure 5.4b where its performance in relation to the domain size is visualized. Its development for an increasing density is also shown in Figure 5.4c. The development of the same scenario on the Coffee Lake platform, tested with twelve threads can be seen in Figure 5.4d. Here also the growing domain size is shown as the influencing variable.

5.3. Homogeneity Influences the Performance of Parallelization Strategies

To test the influence of homogeneity on the behavior of all traversals, scenarios with an increasing standard deviation of homogeneity were performed. All tested scenarios simulated one million particles within a domain of $200 \times 200 \times 200$. Starting from a very homogeneous scenario with a standard deviation from 0.048 to 0.957 and thus a rather in-homogeneous scenario. For most traversals, the run-time increases with an increasing in-homogeneity, for a low standard deviation of homogeneity those findings may differentiate but once the standard deviation is exceeding 0.5, the time to solution increases for all traversals in comparison to a lower standard deviation. This can be observed in Figure 5.5a.

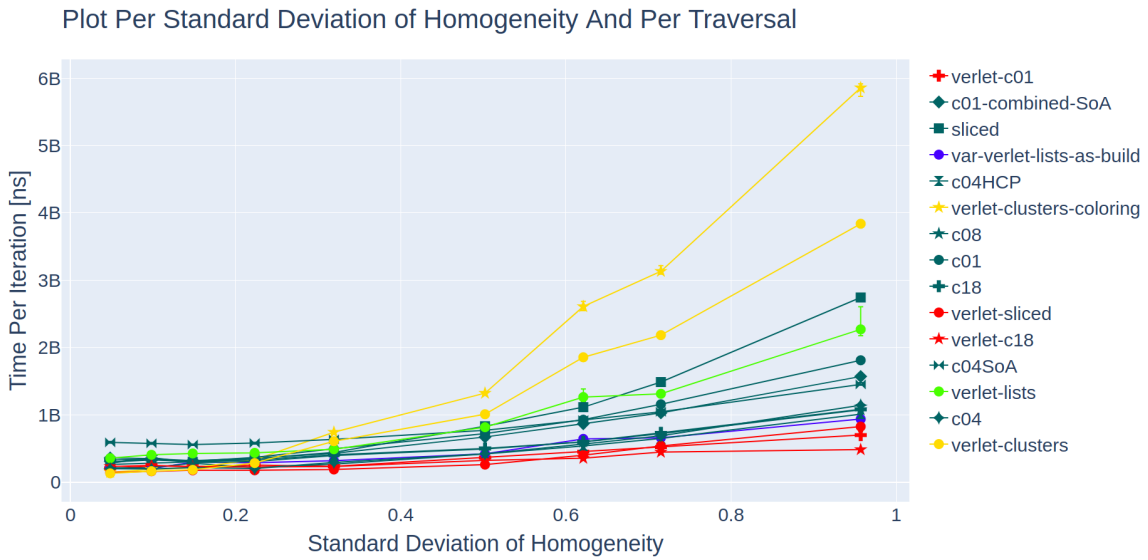
Especially noticeably is the behavior of the verletClusterLists container, which can be observed as the yellow lines in Figure 5.5a. Both the verlet-clusters-coloring, plotted with star-markers, and the verlet-clusters traversal, marked with dots, start off similar to other traversals while increasing their run-time over-proportionally. Other containers remain stable in comparison to the verletClusterLists container, even though they also perform worse with an increasing in-homogeneity. A reason for this finding could be the inefficiency of building Verlet Cluster Lists. Since particles may be very close together in one part of the domain, for each particle a great amount of clusters needs to be considered for pairwise distance calculations. Those clusters are containing particles which are close but not close enough to each other, to be considered for pairwise force calculations. Those unnecessary distance calculations are increasing the time to solution without using the benefits of a Verlet Cluster Lists based approach.

While the gap between especially the verletClusterLists container and the other containers gets bigger, the already observed sorting of traversals by container with an increasing run-time can again be confirmed. This was already examined in Section 5.1, where a growing amount of particles was tested. With an increasing run-time and a larger in-homogeneity the traversals are again grouped by container.

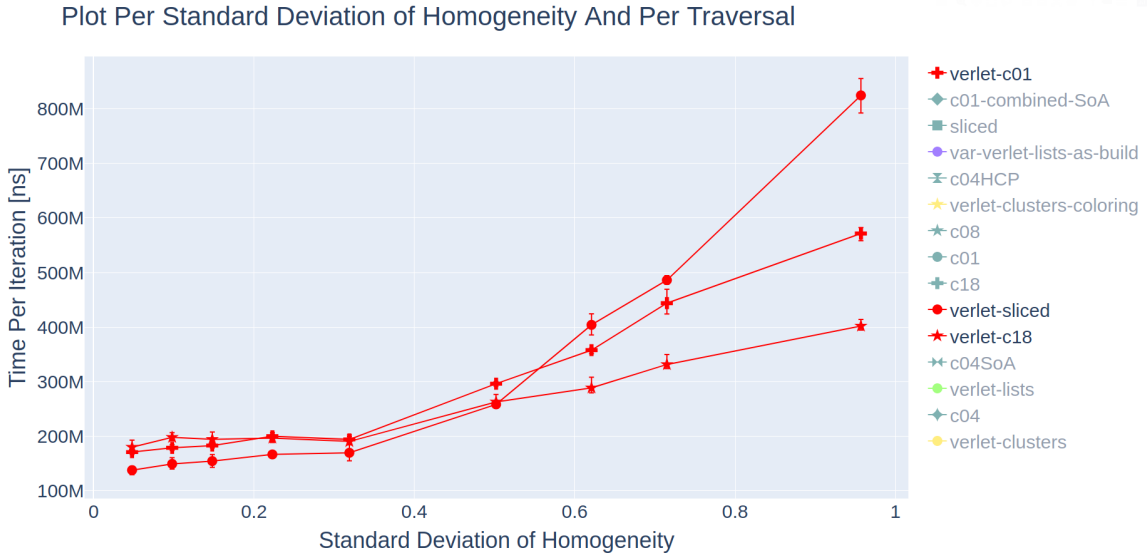
In general, a higher standard deviation of homogeneity is disadvantageous for sliced-based traversals. This can be seen in Figure 5.5b, tested with 56 threads on the Haswell platform. Within the verletListsCells container, the sliced-based traversal, called verlet-sliced, whose data points are visualized with dots, starts as the best performing traversal and ends as the slowest. On the other hand, both color-schemed traversals which start off slower, are faster in comparison to the verlet-sliced traversal for a high standard deviation of homogeneity. The color-schemed traversals verlet-c18 and verlet-c01 can be seen as the star-markers and the cross-markers in Figure 5.5b. Nevertheless, the in-homogeneity is disadvantageous for the time to solution of all traversals. The especially unfavorable impact of in-homogeneity on the sliced-based approaches is supported by Figure 5.5c. There, the scenario was tested with 28 threads, also on the Haswell platform. Within the linkedCells container also the color-schemed traversals perform better than the sliced-based traversal. As an example for a sliced-based approach within the linkedCells container, the traversal called sliced can be observed in Figure 5.5c. To show the performance of a color-schemed traversal within this container, the c04SoA was chosen. Similar findings for both containers can be observed with 28 and 56 threads each, on the Haswell platform. Therefore, one can conclude that for homogeneous scenarios, a sliced-based approach is favored, while for in-homogeneous

scenarios the color-schemed traversals are better performing. This is due to the currently missing load-balancing mechanisms of the sliced-based solutions.

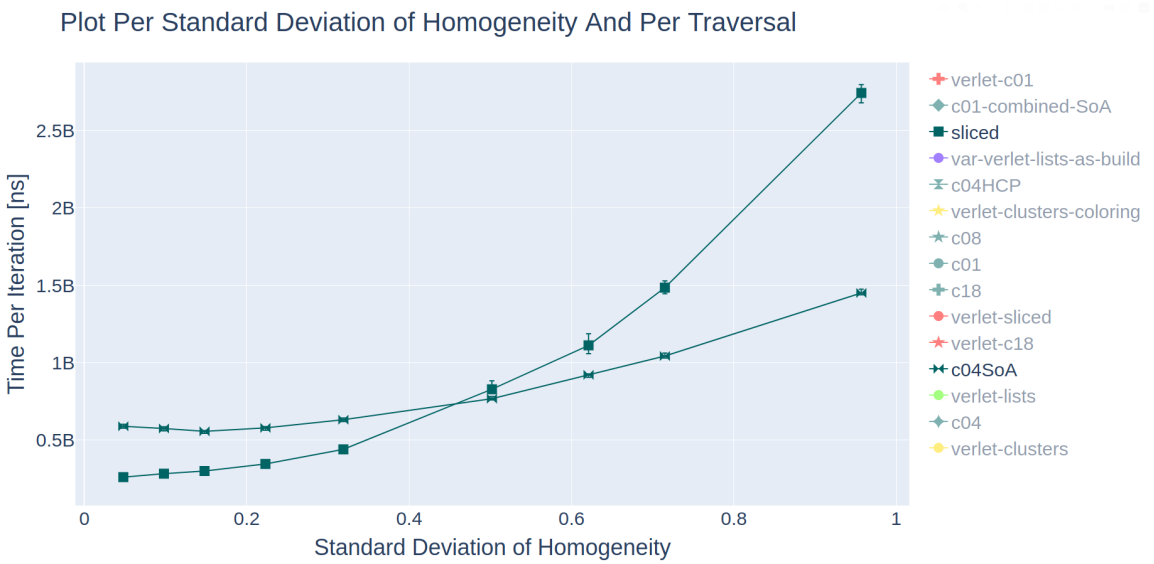
The tests ran on the Coffee Lake platform support the hypothesis of an increasing run-time due to an increased standard deviation of homogeneity. This can be observed in Figure 5.5d. Apart from this, the theory of in-homogeneity having a negative impact on the performance of sliced-based traversals can only partially be supported. The verlet-sliced traversal is best performing for a large number of 740.000 particles in a homogeneous scenario in Section 5.1 on the Coffee Lake platform. Its decrease in performance on this platform is similar to those of the color-schemed traversals, but it is no longer performing best within its container.



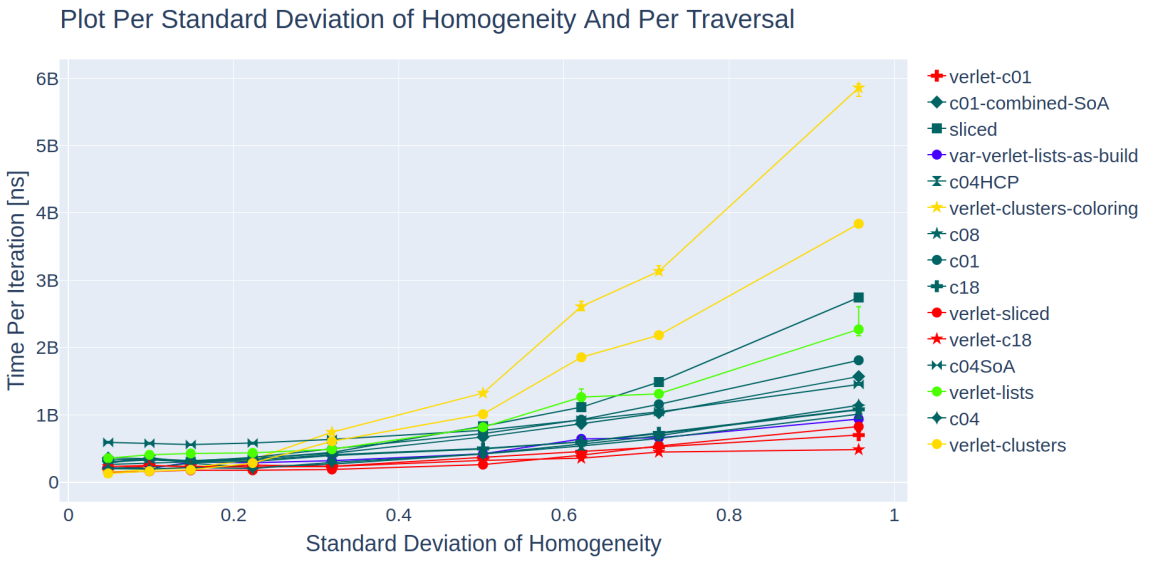
(a) Increasing Standard Deviation of Homogeneity on Haswell Platform With 28 Threads



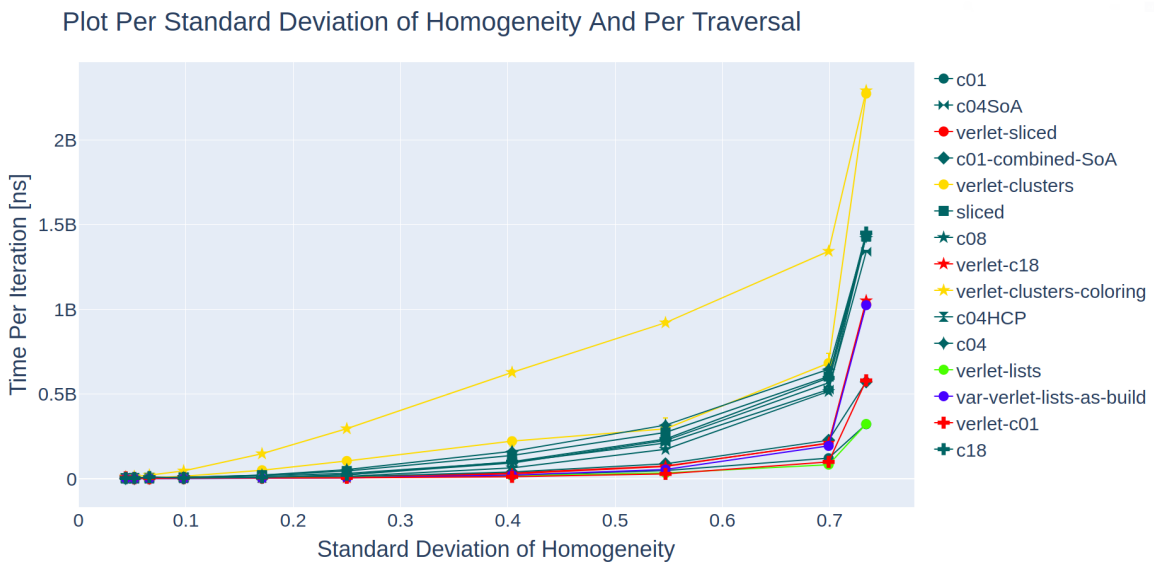
(b) Development of verletListsCells Container With Increasing In-Homogeneity on the Haswell Platform With 56 Threads



(c) Increasing Standard Deviation of Homogeneity Comparison of sliced and c04SoA Traversal on the Haswell Platform Tested With 28 Threads



(d) Development of Increasing In-Homogeneity on the Coffee Lake Platform With Twelve Threads



(e) Gauss Distribution With an Increasing Standard Deviation of Homogeneity on the Haswell Platform With 28 Threads

Figure 5.5.: All figures in this chapter are showing an increasing standard deviation of homogeneity. Figure 5.5d visualizes the performance for one million particles and a standard deviation of homogeneity between 0.048 and 0.957. The domain size remains constant at $200 \times 200 \times 200$. For the same scenarios also tested with 28 threads on the Haswell platform, one can take a closer look at the performance of the sliced and the c04SoA traversal in Figure 5.5c. The traversals within the verletListsCells container are compared in Figure 5.5b, also on the Haswell platform but with 56 threads. The development of the described scenarios on the Coffee Lake platform using twelve threads can be observed in Figure 5.5d. Another test where particles are placed by the Gauss distribution can be seen in Figure 5.5e. The plot shows an increasing standard deviation of homogeneity from 0.044 to 0.734, tested on the Haswell platform with 28 threads. The scenarios were run with 10.000 particles within a domain of $75 \times 75 \times 75$.

In addition to the described large scenario, a small one with only 10.000 particles was examined. As it can be observed in Figure 5.5e, also for a smaller number of particles a less homogeneous scenario needs a longer time to solution. Also the theory of a high standard deviation of homogeneity being disadvantageous for sliced-based traversals can be supported. Both the sliced traversal within the linkedCells container, as well as the verlet-sliced traversal within the verletListsCells container are starting off as one of the best and the best performing traversal within their container for a homogeneous scenario. With a growing in-homogeneity, their run-time increases over-proportionally and both are within the worst performing traversals within their container. Those observations can be made both for 28 and 56 threads on the Haswell platform. The findings are also supported by the tests run on the Coffee Lake platform with twelve threads. The differences between the traversals are yet not as distinct as for the large example with one million particles. It was earlier seen in Section 5.1 that differences between traversals are overall increasing with a

growing number of particles.

An explanation for the seen disadvantage of sliced-based traversals when computing in-homogeneous scenarios could be the fact that most of the particles are located in one part of the domain. Therefore, for a part of the domain an increased waiting time due to the locked areas is needed for particles which are located next to the barriers of the slices. Meanwhile, other parts of the domain are empty and need no calculation at all. Since each thread gets assigned one slice at a time, some threads are computing a main part of the force calculations, while others are not needed at all. Hence, the amount of unnecessary assignments of threads creates an overhead and increases the run-time.

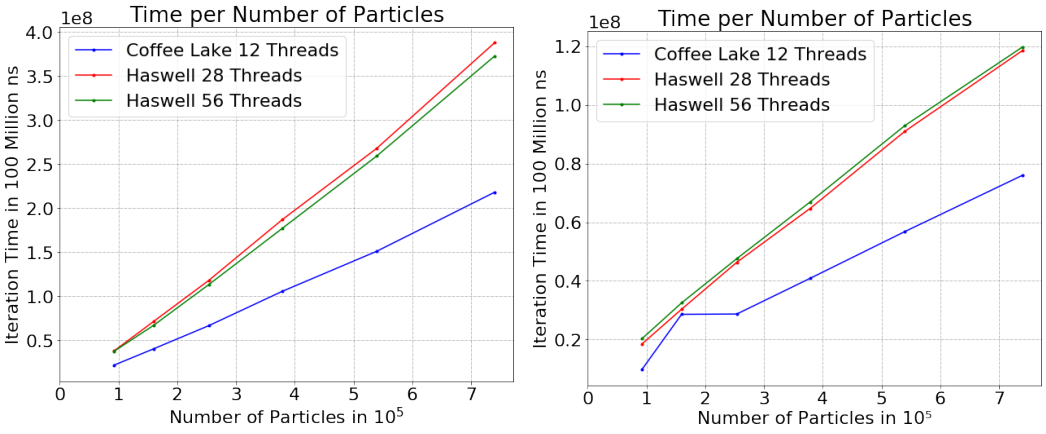
6. Comparison of Computational Platforms

In this chapter, the Haswell and the Coffee Lake platform are compared. These computational platforms were described in Section 4.1. The findings of Chapter 5 will be re-evaluated in consideration of the used hardware. Overall, the Coffee Lake platform with twelve threads performed better in most cases, especially regarding a high number of particles. This finding differentiates for some traversals but when comparing the better performing ones it can be supported.

6.1. Performance for a High Number of Particles

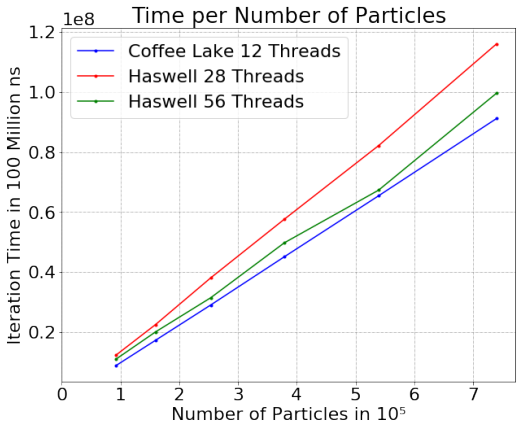
For the comparison of the different computational platforms regarding a high number of particles, the traversals verlet-lists, verlet-sliced and verlet-c18 were chosen. In Section 5.1 it could be seen that the verlet-lists traversal performed very differently on the Coffee Lake platform compared to the Haswell platform. Also, it is considerable that the verlet-c18 traversal always performed best for a high density on the Haswell platform, whereas the verlet-sliced traversal always performed best on the Coffee Lake platform. Therefore, these three traversals are examined in detail regarding their performance on a different hardware.

The verlet-lists traversal is based on the Verlet Lists algorithm. The verlet-sliced and the verlet-c18 traversal are both implemented within the verletListsCells container and also based on the Verlet Lists algorithm. For parallelization purposes, the verlet-sliced traversal is based on the idea of slicing the domain, which was further discussed in Section 3.2. The verlet-c18 traversal on the other hand is using 18 colors which are processed sequentially and parallelized per color. The main idea of a color-scheme-based approach is also described in Section 3.2.



(a) verlet-lists Traversal

(b) verlet-sliced Traversal



(c) verlet-c18 Traversal

Figure 6.1.: The figures show a run-time comparison for several traversals within the verletListsCells container. They were tested on different computational platforms with an increasing number of particles. Starting with 92.500 and ending with 740.000 within a proportionally large domain size to keep the density of 0.74 constant. Figure 6.1a shows the development of the verlet-lists Traversal on the Coffee Lake platform using twelve threads, as well as on the Haswell platform once with 28 and once with 56 threads. Figure 6.1b represents the development of the verlet-sliced traversal which is sliced-based on the same hardware. The run-time of a color-schemed traversal, the verlet-c18 traversal, can be observed in Figure 6.1c.

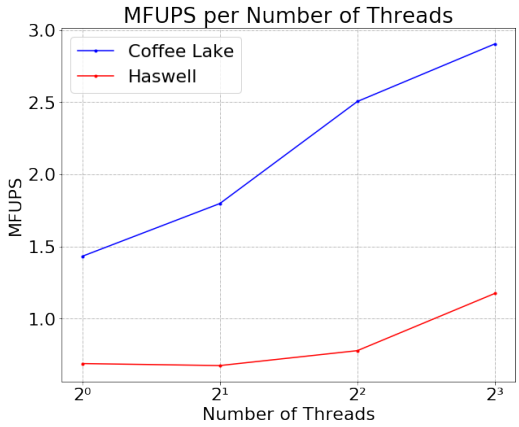


Figure 6.2.: The figure shows a strong scaling for the verlet-lists traversal with increasing threads from 2^0 to 2^3. It is comparing the Coffe Lake platform and the Haswell platform. The number of particles in the scenario was 740.000 with a domain size of $100 \times 100 \times 100$ and therefore a density of 0.74.

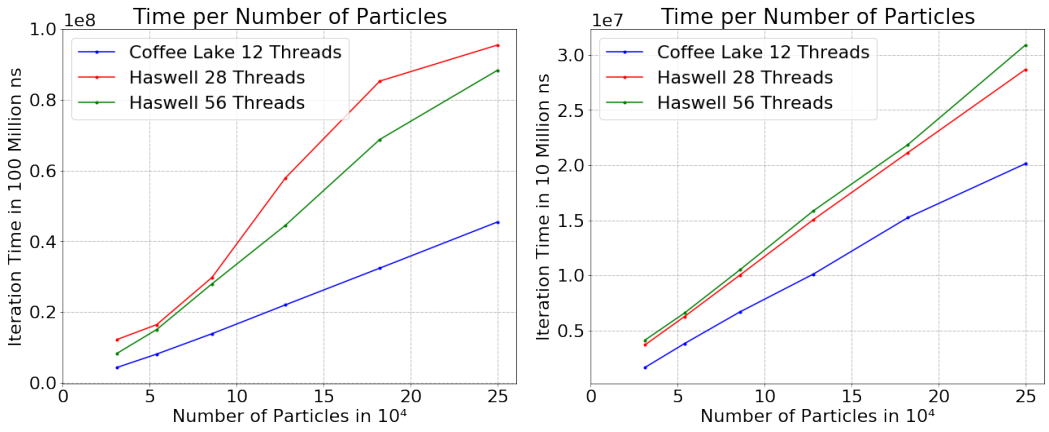
For all three traversals the Coffee Lake platform performs best, regardless of the density, as one can see in Figure 6.1a or Figure 6.3a, for example. For a density as high as 0.74, the largest difference between the tested computational platforms can be depicted for the

verlet-lists traversal, as shown in Figure 6.1a. A reason for the better performance of the verlet-lists traversal on the Coffee Lake platform could be its dependence on the single thread performance. This is confirmed by Figure 6.2. There, it can be seen that even for the same amount of threads the MFUPS are much higher for the Coffee Lake platform, than for the Haswell platform. The scenario was tested with a maximum density of 0.74 and 740.000 particles. The threads increase from 2^0 to 2^3.

When looking at the run-time of the verlet-sliced traversal, there still is a recognizable gap between both graphs for the Haswell platform and the Coffee Lake platform. This gap is visualized in Figure 6.1b. The difference between the performance is the smallest for the verlet-c18 traversal, as shown in Figure 6.1c. It is also considerable that for both, the verlet-lists and the verlet-sliced, the Haswell platform performs similarly regardless of the number of threads, while for the verlet-c18 traversal the Haswell platform using 56 threads is closer to the Coffee Lake platform.

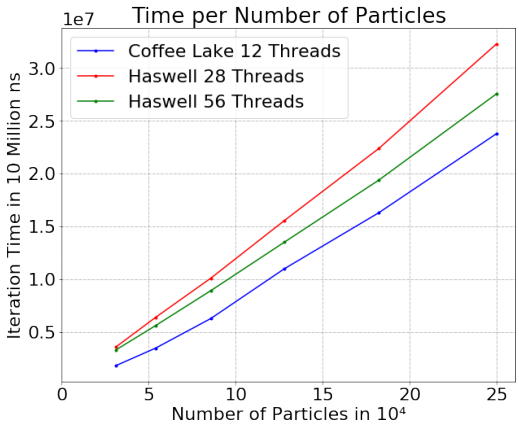
The tests were also performed with a low density of 0.25, which confirm the findings of the tests with a higher density. The gap between the Haswell platform and Coffee Lake platform is the largest for the verlet-lists traversal again. This can be observed in Figure 6.3a. Also, a difference for the Haswell platform and the Coffee Lake platform when testing the verlet-sliced traversal can be seen in Figure 6.3b. It is again the smallest for the verlet-c18 traversal, as shown in Figure 6.3c. These findings lead to the hypothesis that for a coloring-scheme-based traversal the traversals perform similar on different hardware, while for a sliced-based approach a higher number of threads is disadvantageous. This can be explained with the waiting time needed when locking the barrier parts of a slice. The more threads are used, the more are the slices distributed and therefore more dependencies are needed. While the calculations of the slices should still be finished faster, there might also be more often the case of threads waiting for one another.

Furthermore, it can be recognized in Figure 6.3a that the run-time of the Haswell platform is highly increasing, when the amount of particles grows, especially when using 28 threads. Hence, it is decreasing its performance with a higher number of particles. Its run-time is also less steady increasing than the Coffee Lake platform. The latter has a linear progression for the verlet-lists traversal.



(a) verlet-lists Traversal

(b) verlet-sliced Traversal



(c) verlet-c18 Traversal

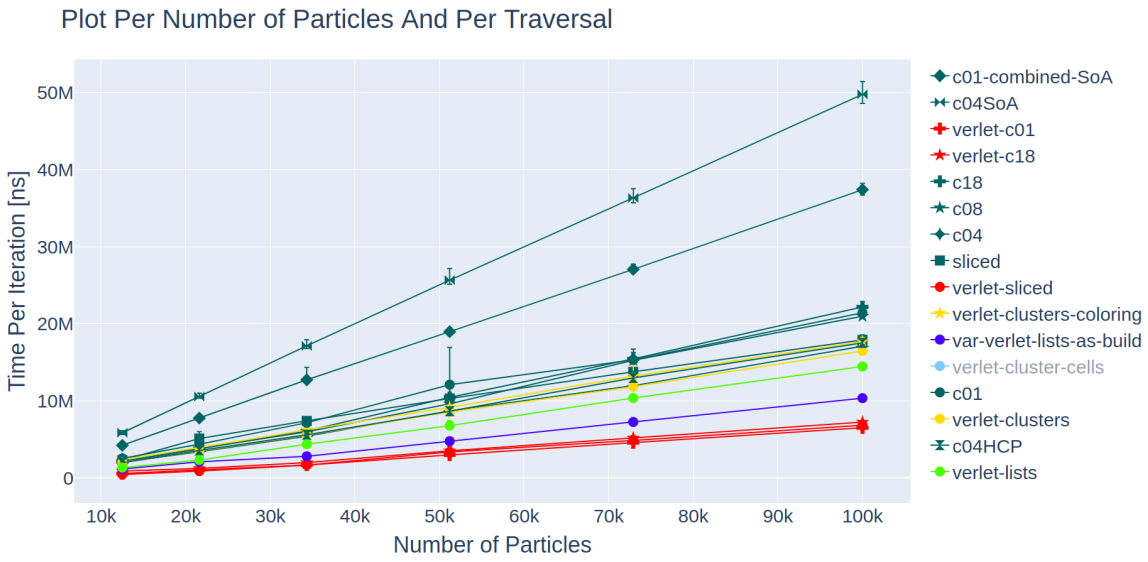
Figure 6.3.: The figures show the development of the run-time on both computational platforms. The Coffee Lake platform was tested with twelve threads and the Haswell platform once with 28 and once with 56 threads. Figure 6.3a visualizes the run-time of the verlet-lists traversal, while Figure 6.3b shows the verlet-sliced traversal and Figure 6.3c the verlet-c18 traversal. The tests were performed with an increasing number of particles from 31.250 up to 250.000 within a proportionally increasing domain size, keeping the density stable at 0.25. All shown traversals are within the verletListsCells container.

6.2. Performance for a Low Density

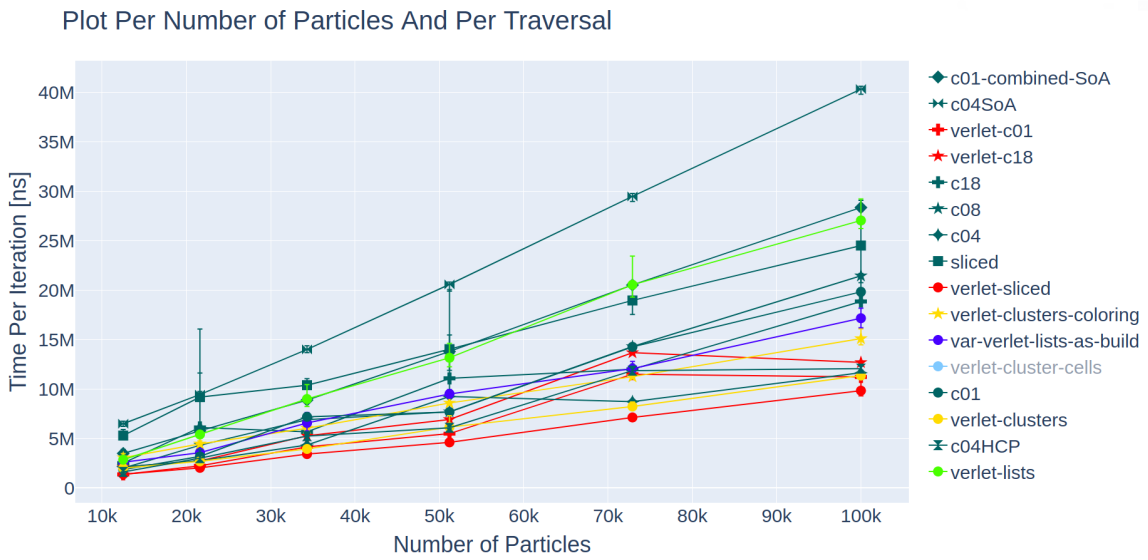
The findings of Section 6.1 differentiate for a very low density of 0.1, which will be examined in this chapter. For a higher density, the performances of the traversals are grouping by container. This grouping remains when the traversals are tested on the Coffee Lake platform, which can be seen in Figure 6.3d. In contrast to this, their performance is mixed on the Haswell platform. This is visualized in Figure 6.3e. Also, the overall performance remains mostly stable on the Coffee Lake platform while increasing and dropping on the Haswell platform.

It is noticeable that for most of the color-schemed traversals their time to solution is increasing in an unstable manner on the Haswell platform, regardless of the container. In contrast is the performance of the sliced-based traversals increasing constant on both platforms. Since especially the run-time of the c04, c01 and the c18 traversal within the linkedCells container varies on the Haswell platform, while increasing stable on the Coffee Lake platform, those three will be examined in detail.

As it can be seen in Figure 6.4a, both the run-time for the c04 traversal on the Coffee Lake platform running with twelve threads and on the Haswell platform running with 28 threads are constantly increasing. Meanwhile, the time to solution for 56 threads on the Haswell platform is jumping from a worse performance, compared to both other platforms, to a better performance. For the c01 traversal, plotted in Figure 6.4b and the c18 traversal, seen in Figure 6.4c, the run-time of the Haswell platform increases in an unstable line, regardless of the number of threads.

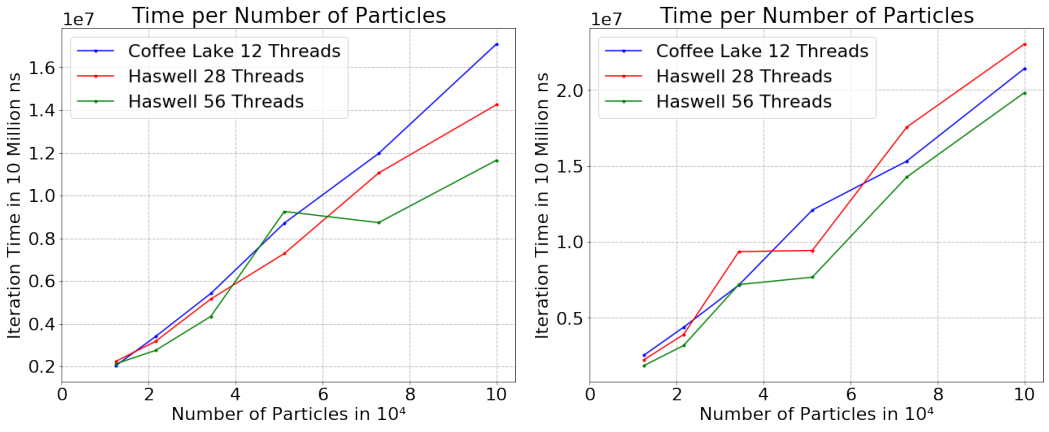


(d) Coffee Lake Platform Twelve Threads



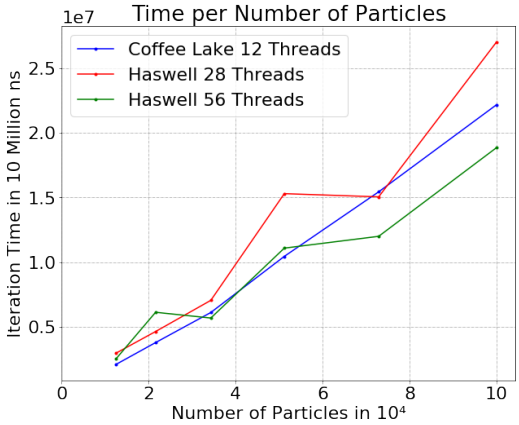
(e) Haswell Platform 56 Threads

Figure 6.3.: The figures show the run-time for an increasing number of particles from 10.000 to 100.000 within a proportionally growing domain size. Thereby, the density is kept stable at 0.1. Figure 6.3d is visualizing the run-time on the Coffee Lake platform with twelve threads. On Figure 6.3e the development on the Haswell platform with 56 threads can be seen.



(a) c04 Traversal

(b) c01 Traversal



(c) c18 Traversal

Figure 6.4.: All figures show how the traversals perform depending on an increasing number of particles from 12.500 to 100.000 within a proportionally increasing domain size. The density of 0.1 was thereby kept constant. The run-time of the traversals is compared by the computational platforms: the Coffee Lake platform with twelve threads and the Haswell platform tested with 28 and 56 threads. Figure 6.4a shows the c04 traversal which is color-schemed, as well as the other two shown traversals, the c01 in Figure 6.4b and the c18 in Figure 6.4c.

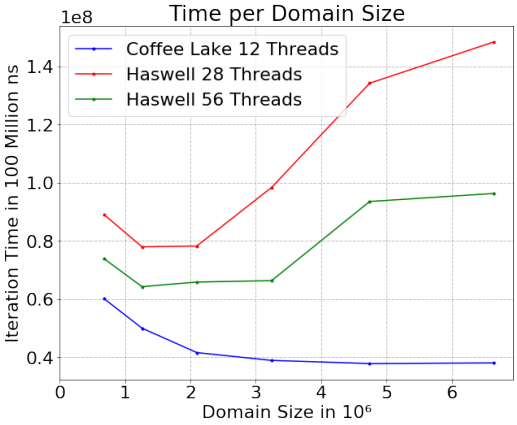
6.3. Performance for an Increasing Domain Size

When looking at an increasing domain size with a fixed amount of 500.000 particles, the development of the run-time for each traversal on the Coffee Lake platform remains stable. Meanwhile, the run-time for some traversals on the Haswell platform increases and for some of them decreases, regardless of the number of threads. This behavior can be observed in Figure 5.4d for the Coffee Lake platform and in Figure 5.4a for the Haswell platform. Especially the traversals within the verletListsCells container, to be precise the traversals verlet-c01 and verlet-c18, again differentiate regarding the used hardware and will therefore be examined in detail. Also, the verlet-lists traversals, the c01-combined-SoA and the

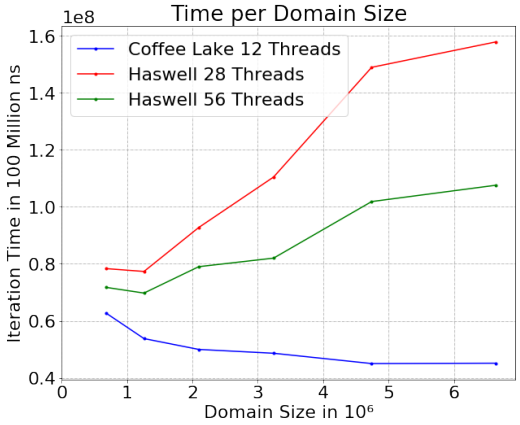
sliced traversal will be further examined, due to their changes in run-time coming with an increasing domain size.

As it can be observed in both Figure 6.5a and Figure 6.5b, the color-scheme-based traversals within the verletListsCells container are even improving their performance on the Coffee Lake platform with a larger domain size and a thus lower density. On the other hand, both traversals have an increasing run-time when tested on the Haswell platform. The performance is worse when using 28 instead of 56 threads, but both favor a higher density. Also considerable is that especially the time to solution measured for the color-scheme-based verlet-c01 traversal drops before increasing again. This leads to the assumption that a maximum density, as well as a too low density, are not favorable for the color-schemed traversals running on the Haswell platform.

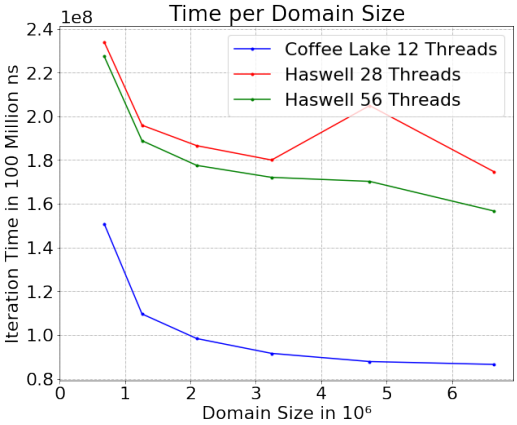
A different observation can be made for the verlet-lists traversal, seen in Figure 6.5c: The progression of all three graphs are quite similar. Meanwhile, the overall gap between the Coffee Lake platform and both Haswell platform test runs is the largest for this traversal.



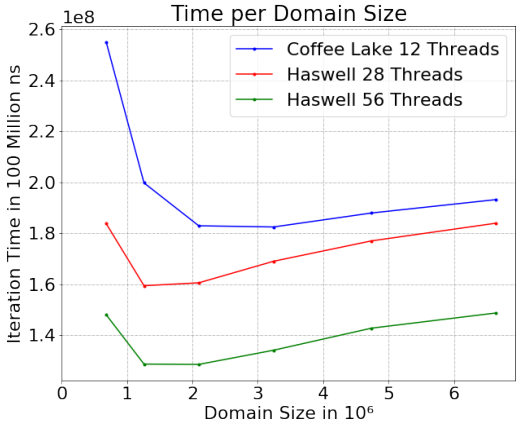
(a) verlet-c01 Traversal



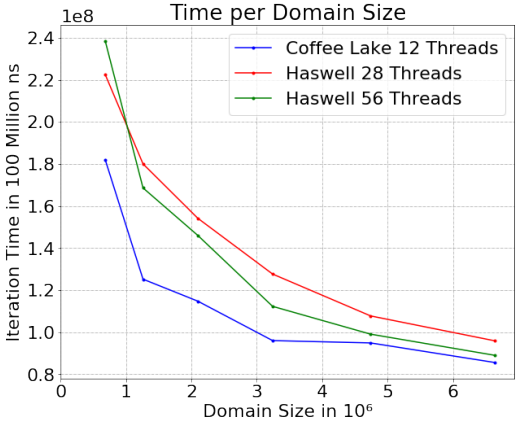
(b) verlet-c18 Traversal



(c) verlet-lists



(d) c01-combined-SoA Traversal



(e) sliced Traversal

Figure 6.5.: The run-time depending on the domain size compared by computational platforms can be seen in the figures above. All scenarios were tested on the Coffee Lake platform with twelve threads and on the Haswell platform with 28 and 56 threads, respectively. The scenario's domain size increased from $88 \times 88 \times 88$ to $188 \times 188 \times 188$ while having a constant amount of 500.000 particles. Therefore, the density decreased. Figure 6.5a shows the run-time of the verlet-c01 traversal, a color-schemed traversal within the verletListsCells container. In Figure 6.5b and Figure 6.5c one can also observe the development of traversals within the verletListsCells container: the verlet-c18 and the verlet-lists traversal. In Figure 6.5d a color-schemed traversal within the linkedCells container can be seen. Figure 6.5e represents a sliced-based approach in the same container.

Out of the five closely examined traversals, the only traversal performing better on the Haswell platform, than on the Coffee Lake platform, is the c01-combined-SoA traversal. Its development is visualized in Figure 6.5d. The traversal is located in the linkedCells container and thus based on the Linked Cells algorithm. It is also a color-scheme-based traversal. Combined with other traversals based on the Linked Cells algorithm, which also perform better on the Haswell platform, than on the Coffee Lake platform, one can draw the conclusion that the Coffee Lake platform can make better use of the Verlet Lists approach.

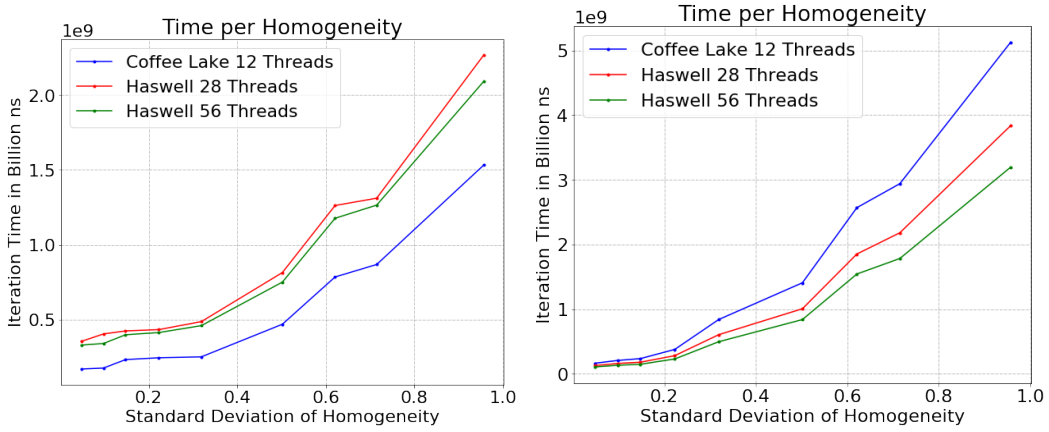
The sliced traversal is compared regarding its run-time on different computational platforms in Figure 6.5e. The traversal is based on the concept of cutting the domain into slices which then get assigned by threads. It can be seen that the larger the domain gets, not only the run-rime decreases, but also the graphs, visualizing the different hardware that was used, converge. It can therefore be assumed that the larger the domain size gets, the less does the choice of hardware matter when executing the sliced traversal. This is a counter example for the verlet-c01 and verlet-c18 traversal. For both traversals, the run-time development for different computational platforms diverges with an increasing domain size. A reason for this could be that for a lower density less waiting for particles at barrier regions of slices is needed. The lower density is due to the larger domain size. It can be assumed that a sliced-based approach is benefiting from the low density, regardless of the number of threads. When looking into the color-schemed traversals with more threads, an overhead occurs, due to the blocks of cells which are assigned to a certain thread without many particles in it to

compute. Thus, the re-assigning of threads is surpassing the benefits of a high number of threads.

6.4. Performance for Homogeneous and In-Homogeneous Scenarios

When examining the tests with an increasing in-homogeneity, especially the verlet-lists traversal and the verlet-clusters traversal differentiate regarding their performance on the two tested computational platforms. The verlet-clusters traversal is based on the verletClusterLists container. Its time to solution is visualized in Figure 6.6b. For more homogeneous scenarios, its run-time is similar on all computational platforms and regardless of the number of threads. With a growing standard deviation of homogeneity, the gaps between the measured run-times are also increasing. Especially the performance of the Coffee Lake platform is decreasing over-proportionally. This leads to the assumption that the negative effect of a growing in-homogeneity is supported by the lower number of threads executed on the Coffe Lake platform.

When looking at the verlet-lists traversal in Figure 6.6a, a different observation can be made. For the traversal that is based on the verletLists container, the gap between the Haswell platform and the Coffee Lake platform remains rather stable. It only slightly grows with an increasing in-homogeneity. Those findings are supporting the hypothesis of traversals based on the Verlet Lists algorithm preferring the Coffee Lake platform.



(a) verlet-lists Traversal (b) verlet-clusters Traversal

Figure 6.6.: The plots show the development of the verlet-lists traversal in Figure 6.6a and the verlet-clusters traversal in Figure 6.6b. They also show the different run-time on the Coffee Lake platform with twelve threads, as well as on the Haswell platform with 28 and 56 threads, respectively. The tested scenarios had an increasing standard deviation of homogeneity. They were tested with a constant number of one million particles within a domain of the size $200 \times 200 \times 200$.

7. Comparison of Different Functors

The above described scenarios were all tested with the functor provided by the AutoPas library as an example. A functor defines the force calculation applied for a simulation. In this case, the Lennard Jones potential, which was described in Section 2.1. Functors are also representing the form of vectorization used for computing the simulation. The previously tested functor was applying automatic-vectorization by the compiler. The automatic functor will be compared to a new one applying hand crafted AVX intrinsic vectorization for SoA based approaches.

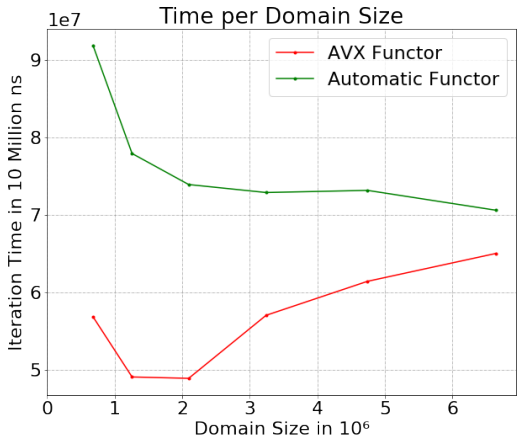
For comparison purposes, the scenario with an increasing domain size, a fixed number of particles and a thereby decreasing density, that was already seen in Section 5.2, was chosen. The number of particles was 500.000 and the measurements of the domain increased from $88 \times 88 \times 88$ to $188 \times 188 \times 188$. A closer look will be taken at the verlet-c18 and the verlet-sliced traversal, since the verletListsCells container previously had an overall good performance. Also, the c04 and the c04HCP traversal based on the linkedCells container, will be examined, because they performed best within their container when tested with the automatized functor. Both Linked Cells based approaches were described in detail in Section 3.2.

When investigating the linkedCells container, the differences in run-time are quite large, especially for a smaller domain size. As it can be seen in Figure 7.1a, the color-schemed c04HCP traversal has an overall lower run-time when tested with the AVX functor. The gap gets smaller as the domain size grows. The run-time development of the c04HCP traversal is shown for a test with 56 threads on the Haswell platform. The gap for a larger domain size is not as small when tested with 28 threads. Since the c04HCP traversal and the c04 are performing similar, this larger gap is shown by the c04 traversal tested with 28 threads in Figure 7.1b. Both figures show that the run-time of the automatized functor has a downwards trend for a growing domain size. Meanwhile, the AVX functor performs best for a domain size between $108 \times 108 \times 108$ and $128 \times 128 \times 128$ and has a declining performance for a larger growing domain size. The findings are again supported by the results of tests run on the Coffee Lake platform with twelve threads. The development of the c04HCP traversal can be seen in Figure 7.1e, where the automatized functor again has an increasing performance as the domain size grows and the AVX functor has a decreasing performance for domain measurements of $148 \times 148 \times 148$ and larger.

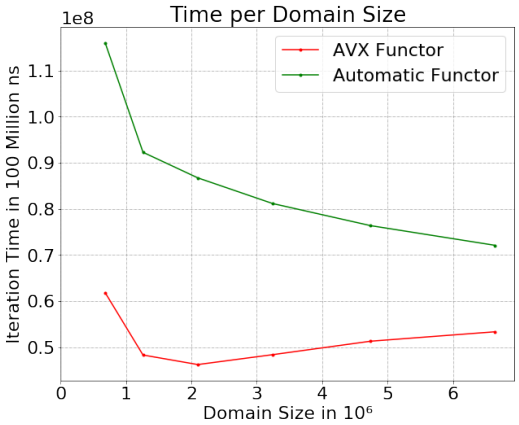
As it can be seen in Figure 7.1c, the performance of the verlet-sliced traversal is similar for both functors when tested with 56 threads on the Haswell platform. Their performance for the sliced-based traversal aligns similarly when tested with 28 threads on the same computational platform. As it is shown in Figure 7.1d, the trend of the run-time is also similar for the verlet-c18 traversal, here tested with 56 threads on the Haswell platform. The differences can be explained by expectable fluctuations of the measurements. The fluctuation is smaller when observing the verlet-c18 traversal on the Coffee Lake platform with twelve threads. As visualized in Figure 7.1f, the developments of the run-time of both functors

nearly align. It should be noted that the verlet-c18 overall has a decreasing run-time on the Coffee Lake platform, while it was overall increasing with a growing domain size on the Haswell platform.

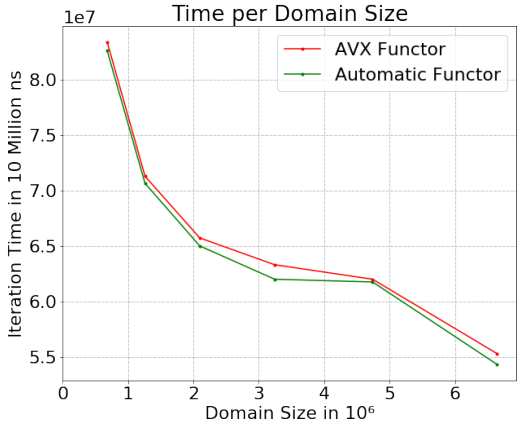
Overall, especially for the linkedCells container, the performance is better when applying the AVX functor for a large domain size and a high number of particles. The gap is increasing when the time to solution increased and vice versa. Future work should further examine the differences between the two functors regarding other physical parameters.



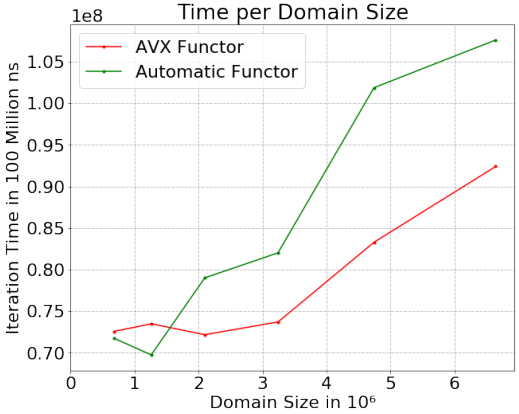
(a) c04HCP Traversal on Haswell



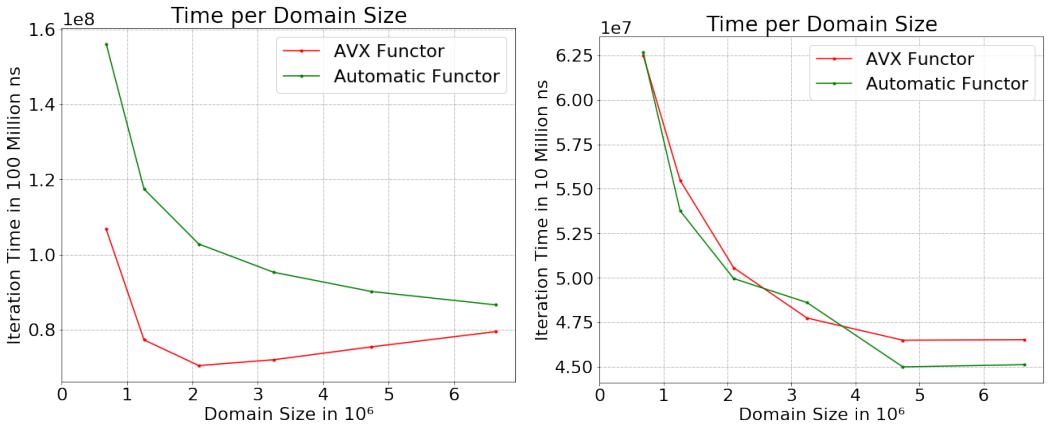
(b) c04 Traversal on Haswell



(c) verlet-sliced Traversal on Haswell



(d) verlet-c18 Traversal on Haswell



(e) c04HCP Traversal on Coffee Lake (f) verlet-c18 Traversal on Coffee Lake

Figure 7.1.: The figures show a run-time comparison for two different functors. One applying automatic-vectorization by the compiler and one applying hand crafted AVX intrinsic vectorization. The tests were performed with 500.000 particles and the measurements of the domain increased from $88 \times 88 \times 88$ to $188 \times 188 \times 188$. Figure 7.1a shows the development of both functors for the c04HCP traversal on the Haswell platform with 56 threads. In Figure 7.1b the tests were run with 28 threads. There, the functors are compared by the c04 traversal. The performance of the verlet-sliced traversal, tested with 56 threads, also on the Haswell platform can be observed in Figure 7.1c. The same tests were also run for the verlet-c18 traversal, which can be seen in Figure 7.1d. The results of the tests run on the Coffee Lake platform with twelve threads can be observed in Figure 7.1f for the verlet-c18 traversal. The same computational platform was tested with the c04HCP traversal in Figure 7.1e.

Part IV.

Conclusion

8. Summary

In the presented thesis, different parameters of simulated scenarios were investigated regarding their influence on the performance of traversals. Overall it is noticeable that with an increasing number of particles the performance of traversals started to group by container. They remained grouped on the Coffee Lake platform when running on twelve threads, even when the homogeneity or the density of a scenario was changing. When tested on the Haswell platform with 28 or 56 threads, they started ungrouping for an increasing standard deviation of homogeneity, as well as for a decreasing density.

The verletListsCells container performed best for scenarios with an increasing number of particles and a constantly high density 0.74. A density of 0.74 was identified by applying the close-packing of equal spheres problem in Section 2.3. For the verletListsCells container, which is based on the Verlet Lists algorithm, described in Subsection 2.2.3, those findings are also supported for lower densities of 0.25 and 0.1. These observations were made regardless of the computational platform. The only exception is the result for a low density of 0.1, where the two color-schemed traversals within this container had a decreasing performance in comparison to the other traversals. Only one of the three verletListsCells traversals, the verlet-sliced traversal still performed best on the Haswell platform. On the Coffee Lake platform this container is still overall favored.

For an increasing domain size and a thereby decreasing density the verletClusterLists container had the lowest run-time. It is based on the Verlet Cluster Lists algorithm seen in Subsection 2.2.4. It is overall observed that the more the domain size increased and therefore the density decreased, the better did the container perform absolute and in comparison to other containers. Also the verletClusterCells container showed an increasing performance as the domain size enlarged. Regardless of that improvement, it hast to be stated that, overall, the verletClusterCells container is performing worst in comparison to all other containers. However, does this finding supporting the theory that the Verlet Cluster Lists algorithm is beneficial for a large domain size and a low density.

When comparing the performance on different computational platforms, the results showed that Linked Cells based and Verlet Cluster Lists based approaches are performing better on the Haswell platform. The Verlet Lists based traversals favored the Coffee Lake platform. Nevertheless did the Verlet Lists based approaches perform better than the other named algorithms on the Haswell platform for certain configurations.

The comparison of two functors with different approaches for vectorization showed that the AVX functor changed only little for Verlet Lists based traversals, especially for the sliced-based. In contrast, the Linked Cells based traversals had a considerably better time to solution when they were tested with the AVX functor. This is most of all important when the density, the number of particles and the domain size is large and gets less important for a lower density.

Overall, it was seen that color-schemed approaches are performing better in case of a high density, while for a low density sliced-based parallelization strategies should be preferred.

A high standard deviation of homogeneity is in general decreasing the performance, but especially the sliced-based approaches showed an increasing run-time and a therefore worse performance. While the color-schemed traversals are predominantly performing better on the Coffee Lake platform, the run-time of the sliced-based approaches is more depending on the other examined factors.

In addition to the described tests, a new traversal was added to the AutoPas library. The c04HCP traversal based on the linkedCells container enables the use of a color-schemed approach similar to the c04 traversal. The c04 traversal was developed in the context of [Tch20]. The new traversal has the benefit of also only needing four colors by having a much simpler form, than the c04 traversal to process. While the c04 blocks where plus shaped and aggregated by at least 32 cells, the c04HCP traversal works with simple cuboids of six or more cells. Since less cells are needed, the blocks can easily be scaled up or down in size. The c04HCP traversal performs similar to the c04 traversal and notably well with the AVX functor. Its time to solution is especially low for a large domain size and a low density.

9. Future Work

In future tests, non-static particles should be included, since for all the evaluated scenarios the particles were not moving. Thereby, a changing homogeneity or density during run-time should be observed. Making the tuning dependent on a scenario's physical parameters has the chance to decrease the time to solution and opens the possibility of learning based tuning. A step further would then be to auto-tune depending on the parameter observations made during run-time.

So far, the cells needed for the calculation of homogeneity were dependent on a fixed number of particles, based on the idea of a perfectly homogeneous scenario. Future work should test the influence of homogeneity again with a different ideal amount of particles and compare the results to those seen in this thesis. It should also aim to analyze the changing load-balancing due to an increasing in-homogeneity, since particles are spread throughout the domain differently, than for a homogeneous scenario. Also, the amount of calculations needed should be investigated. Since particles are closer to one another in an in-homogeneous scenario, for each particle there are more particles located in its cutoff radius and therefore most likely more distance and also more force calculations.

Additionally, future work should run more tests applying the AVX functor. Those tests should include homogeneous and in-homogeneous scenarios, as well as an increasing number of particles. Thereby, the potential of especially the linkedCells container for scenarios with a large amount of particles needs to be examined in detail.

Furthermore, the c04 traversal and the c04HCP traversal could be compared with different sizes of blocks used for the c04HCP traversal. Scaling up the, now six cells large, blocks of the c04HCP traversal would lead to a more coarse-grained approach. Comparing both traversals with similar block sizes could give information about how the run-time for different physical parameters is depending on the size of blocks which are processed. This is especially interesting for scenarios where sliced-based approaches performed better, such as scenarios with a low density.

Part V.

Appendix

List of Figures

Bibliography

[Ang90] WS Anglin. The square pyramid puzzle. *The American Mathematical Monthly*, 97(2):120–124, 1990.

[BTH05] Nicolae-Viorel Buchete, Robert Tycko, and Gerhard Hummer. Molecular dynamics simulations of alzheimer's β-amyloid protofilaments. *Journal of molecular biology*, 353(4):804–821, 2005.

[CD90] Ariel A Chialvo and Pablo G Debenedetti. On the use of the verlet neighbor list in molecular dynamics. *Computer physics communications*, 60(2):215–224, 1990.

[Eck14] Wolfgang Florian Eckhardt. *Efficient hpc implementations for large-scale molecular simulation in process engineering*. PhD thesis, Technische Universität München, 2014.

[GKZ07] Michael Griebel, Stephan Knapek, and Gerhard Zumbusch. Numerical simulation in molecular dynamics, vol. 5 of texts in computational science and engineering, 2007.

[GST+19] Fabio Alexander Gratl, Steffen Seckler, Nikola Tchipev, Hans-Joachim Bungartz, and Philipp Neumann. Autopas: Auto-tuning for particle simulations. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 748–757. IEEE, 2019.

[Hun09] Siegfried Hunklinger. *Festkörperphysik*. Oldenbourg Verlag, 2009.

[MG12] Achim Marx and Rudolf Gross. *Festkörperphysik*. Oldenbourg Wissenschaftsverlag, 2012.

[MR99] William Mattson and Betsy M Rice. Near-neighbor calculations using a modified cell-linked list method. *Computer Physics Communications*, 119(2-3):135–148, 1999.

[NMC66] Isaac Newton, Andrew Motte, and Florian Cajori. The mathematical principles of natural philosophy and his system of the world. *mpnp*, 1966.

[OFF+09] Noriaki Okimoto, Noriyuki Futatsugi, Hideyoshi Fuji, Atsushi Suenaga, Gentaro Morimoto, Ryoko Yanai, Yousuke Ohno, Tetsu Narumi, and Makoto Taiji. High-performance drug discovery: computational screening by combining docking and molecular dynamics simulations. *PLoS Comput Biol*, 5(10):e1000528, 2009.

[PEQ82] JG Powles, WAB Evans, and N Quirke. Non-destructive molecular-dynamics simulation of the chemical potential of a fluid. *Molecular Physics*, 46(6):1347–1370, 1982.

[PH13] Szilárd Páll and Berk Hess. A flexible algorithm for calculating pair interactions on simd architectures. *Computer Physics Communications*, 184(12):2641–2650, 2013.

[Rap97] Dennis C Rapaport. *The art of molecular dynamics simulation*. Cambridge university press, 1997.

[Tch20] Nikola Plamenov Tchipev. *Algorithmic and Implementational Optimizations of Molecular Dynamics Simulations for Process Engineering*. PhD thesis, Technische Universität München, 2020.

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

[YWLC04] Zhenhua Yao, Jian-Sheng Wang, Gui-Rong Liu, and Min Cheng. Improved neighbor list algorithm in molecular simulations using cell decomposition and data sorting method. *Computer physics communications*, 161(1-2):27–35, 2004.