Adaptive Initiation of AutoPas Tuning Phases for Efficient Particle Simulations

Bachelor's Thesis in Informatics

Niklas Ladurner

Technische Universität München

School of Computation, Information and Technology - Informatics

Chair of Scientific Computing in Computer Science

Munich, September 15th, 2025

This page is intentionally left blank.

Adaptive Initiation of AutoPas Tuning Phases for Efficient Particle Simulations

Adaptives Auslösen von Tuning-Phasen für effiziente Partikelsimulation in AutoPas

Bachelor's Thesis in Informatics

Niklas Ladurner

Supervisor: Univ.-Prof. Dr. Hans-Joachim Bungartz
Chair of Scientific Computing in Computer Science

Advisor: Manish Mishra, M.Sc.
Chair of Scientific Computing in Computer Science

Date: 15.09.2025

Technische Universität München

School of Computation, Information and Technology - Informatics

Chair of Scientific Computing in Computer Science

Munich, September 15th, 2025

This page is intentionally left blank.

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

Munich, September 15th, 2025 Niklas Ladurner

This page is intentionally left blank.

Acknowledgements

I would like to express my deepest gratitude to my advisor Manish Mishra. His guidance, patience, and insightful explanations during our meetings were invaluable in helping me understand AutoPas and its wide array of complex features. His continuous encouragement and his in-depth analysis of preliminary results pushed me to refine my ideas and improve the quality of this thesis.

Additionally, I would like to thank all my friends who kindly took the time to proofread parts of this work and provide thoughtful feedback. Their constructive criticism and suggestions helped shape this thesis into what it is now.

The evaluation of our proposed mechanisms could not have been performed without sizeable processing capabilities. Thus, for supporting this thesis by providing computing time on its Linux-Cluster, I gratefully acknowledge the Leibniz Supercomputing Centre.

Lastly, I want to thank José Areia and all contributors to the *IPLeiria-Thesis*[1] template, which was adapted and used in this work.

[1] https://github.com/joseareia/ipleiria-thesis

This page is intentionally left blank.

Abstract

Particle simulations have become an indispensable tool in research and are used across a wide range of applications. Depending on the specific scenario, different simulation configurations may be more suitable. AutoPas is a particle simulation library that offers a simple black-box interface for researchers. To achieve this, AutoPas reevaluates the optimal configuration at fixed intervals. This thesis proposes a dynamic approach to determine ideal points for the initiation of new tuning phases.

This page is intentionally left blank.

Zusammenfassung

TODO

This page is intentionally left blank.

Contents

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

1 Introduction

This chapter introduces some fundamental concepts necessary to understand Molecular Dynamics (MD) simulations. We begin with a discussion of the motivation behind the general n-body problem and the goals of this thesis in Section 1.1. Afterwards, the components of a simple MD simulation loop are presented in Section 1.2. These include Newton's laws of motion for providing the governing equations of particle trajectories, the Lennard-Jones potential as a model for pairwise interactions, and the Störmer-Verlet algorithms as a numerical scheme for integrating the equations of motion.

1.1 Motivation

The n-body problem is a foundational challenge of classical physics. It concerns the interaction and movement of bodies, like the trajectories of masses in the solar system. At such astronomic scales, general relativity additionally introduces a high degree of complexity. Yet, even in classical Newtonian physics, the systems of equations tend to no longer be solvable by analytic means if $n > 2$ bodies are involved, except for certain special cases. Hence, numerical algorithms have become essential in finding approximate solutions. [2]

With the advent of computer-based simulation, the feasibility of numerical solutions to the n-body problem has increased significantly. Over the past few decades, advances in high performance computing (HPC) have further increased both scale and efficiency of such simulations, allowing for the modeling of large systems at unprecedented resolution. For instance, the TianNu project simulated 2.97×10^{12} particles in 2017 on the Tianhe-2 supercomputer [5]. These days, particle simulations have established themselves as indispensable tools across a wide range of scientific fields. Applications reach from drug discovery [13] to plasma physics [23] and materials science [20].

One example of a software framework enabling n-body simulations is the AutoPas library [8]. Its internal mechanisms will be discussed in detail in Chapter 2; for the motivation of this thesis it suffices to know, that AutoPas seeks to dynamically select optimal simulation configurations without requiring expert knowledge during setup ("autotuning").

To achieve this, so-called tuning phases are initiated at fixed intervals. During each tuning phase, different configurations are sampled for a predetermined number of iterations, after which the best performing configuration is selected to simulate the remaining iterations until the next tuning phase. Naturally, these static intervals do not necessarily align with the points at which it would be most advantageous to switch configurations. Consider a scenario, in which the optimal configuration changes rapidly in the beginning, but stabilizes and settles into an equilibrium later on. Having one uniform static interval, it would be either too short — resulting in unnecessary tuning phases during equilibrium, or too long — resulting in suboptimal performance in the early phase. This thesis proposes a method to resolve this problem by dynamically initiating tuning phases based on live simulation data.

1.2 Molecular Dynamics

Molecular Dynamics (MD) simulation is one method of solving the classical n-body problem on the molecular level. At that level, the interactions between atoms are subject to the of laws quantum mechanics, in particular the Schrödinger equation. That equation however is unsuitable for the simulation of larger systems due to its complexity by nature of being a partial differential equation.

Therefore, simplifications such as the Born-Oppenheimer approximation have to be employed. This approximation is based on the fact that the nuclei of atoms have much greater mass than the electrons surrounding them. Under the additional assumption, that the nuclei can be considered static relative to the movements of the electrons, we can separate the Schrödinger equation into two parts coupled by an interaction potential. Using further simplifications, we obtain (1.1), which directly corresponds to the classical laws of motion as stated by Newton (cf. Section 1.2.1). In this equation, $\mathbf{p}_i(t), \mathbf{a}_i(t), m_i, V(\mathbf{p}_i(t))$ are the position, acceleration, mass and potential acting on a particle i at time t. [4, 11, 24]

$$m_i \mathbf{a}_i(t) = -\nabla_{\mathbf{p}_i} V(\mathbf{p}(t)) \tag{1.1}$$

The simplest interatomic potentials one could apply here describe the interactions between only two particles, such as the Gravitational, Lennard-Jones or Coloumb potentials. [11]

Considering the formula presented in (1.1), the main MD simulation loop is rather simple. It consists of calculating the forces between particles or atoms and integrating the equations of motion. These two steps are repeated until an equilibrium is reached, at which point the desired measurements can be taken. [6]

1.2.1 Newton's Laws of Motion

As referred to before, Newton's laws of motion can be applied to MD simulation in approximating particle behavior. These well-known laws of classical mechanics are as follows. [19]

I. *Every object perseveres in its state of rest, or of uniform motion in a right line, unless it is compelled to change that state by forces impressed thereon.*
In other words, if the net force on any body is zero, its velocity is constant.

II. *The alteration of motion is ever proportional to the motive force impressed; and is made in the direction of the right line in which that force is impressed.*
In other words, $F = m \cdot a$.

III. *To every action there is always opposed an equal reaction: or, the mutual actions of two bodies upon each other are always equal, and directed to contrary parts.*
In other words, if one body exerts force F_a on another body, than the second body exerts force $F_b = -F_a$ on the first body.

The second law is of utmost importance, as it allows to compute trajectories of particles based on the force acting upon them. The third law, while secondary in dynamics, is useful especially regarding optimizations: for pairwise interactions, the force needs to be evaluated only once, since the second particle experiences a force of the same magnitude in opposed direction. [9]

1.2.2 Lennard-Jones Potential

Simulating all pairwise interactions between atoms has complexity $O(n^2)$. To reduce this complexity, most MD simulations restrict themselves to short-range interactions. As the forces of these interactions are negligible if the interacting particles are far apart, a cutoff-radius can be introduced after which the forces can be assumed to be close to zero. This significantly reduces the computational complexity, as only the interactions between close neighbors have to be computed. [9]

The Lennard-Jones (LJ) potential is one such short-range interaction potential that acts on pairs of particles. It is based on empirical data and a sufficiently good approximation such that macroscopic effects can be derived from simulating the interactions at an atomic level. It is most frequently used in the form of the 12-6 potential as defined in (1.2).

$$V_{\mathrm{LJ}}(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{1.2}$$

In this equation, r is the distance between the two particles, ε the interaction strength and σ the distance at which the potential signs change (zero-crossing). Parameters ε, σ are dependent on the simulation context, e.g. the material which ought to be simulated. The potential function is illustrated in Figure 1.1. [16, 25]

1.2.3 Störmer-Verlet Algorithm

Using LJ potentials and Newton's laws of motion, we can construct a system of ordinary differential equations. To solve them analytically is practically infeasible for large systems, therefore numeric solvers have to be used in approximating a solution, as stated before.

The Störmer-Verlet algorithm is one such numeric technique for solving the systems constructed as mentioned. With $\mathbf{p}_i(t), \mathbf{v}_i(t), \mathbf{a}_i(t), m_i, \mathbf{F}_i(t)$ as the position, velocity, acceleration, mass and force acting on a particle i at time t, we can derive the algorithm by the summation of Taylor expansions. First, we deduce the position of particle i at time $t + \delta t$, as in (1.3). Secondly, we take a backwards step to $t - \delta t$, as in (1.4).

$$\mathbf{p}_i(t + \delta t) = \mathbf{p}_i(t) + \delta t\dot{\mathbf{p}}_i(t) + \frac{1}{2}\delta t^2\ddot{\mathbf{p}}_i(t) + \frac{1}{6}\delta t^3\dddot{\mathbf{p}}_i(t) + O(\delta t^4) \tag{1.3}$$

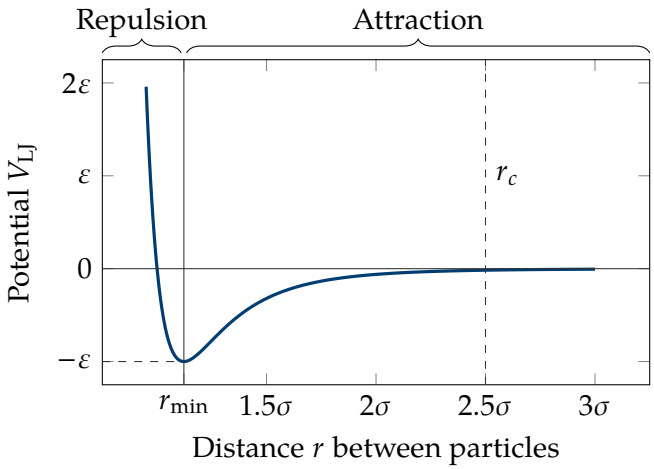


Figure 1.1: An illustration of the potential well of the 12-6 LJ potential, with the minimum of $-\varepsilon$ at $r_{\min} = \sigma\sqrt[6]{2}$, zero-crossing at σ and cutoff radius r_c. The figure is based on Lenhard et al. [16]

$$\mathbf{p}_i(t-\delta t) = \mathbf{p}_i(t) - \delta t\dot{\mathbf{p}}_i(t) + \frac{1}{2}\delta t^2\ddot{\mathbf{p}}_i(t) - \frac{1}{6}\delta t^3\dddot{\mathbf{p}}_i(t) + O(\delta t^4) \tag{1.4}$$

By adding both (1.3) and (1.4) and reordering terms, we conclude (1.5).

$$\mathbf{p}_i(t+\delta t) = 2\mathbf{p}_i(t) - \mathbf{p}_i(t-\delta t) + \delta t^2\ddot{\mathbf{p}}_i(t) + O(\delta t^4) \tag{1.5}$$

As the second derivative of the position $\mathbf{p}_i(t)$ is the acceleration $\mathbf{a}_i(t)$, we can express (1.5) as (1.6). Where, by Newton's second law, $\mathbf{a}_i(t) = \frac{\mathbf{F}_i(t)}{m_i}$.

$$\mathbf{p}_i(t+\delta t) = 2\mathbf{p}_i(t) - \mathbf{p}_i(t-\delta t) + \delta t^2\mathbf{a}_i(t) + O(\delta t^4) \tag{1.6}$$

By this formulation we could already calculate the velocities of the particles, however, there are some drawbacks – e.g. high error propagation [7]. A more exact and efficient approach, sometimes referred to as the Velocity-Verlet algorithm, can be derived similarly. For that, considering (1.7), we can rearrange and substitute into (1.5) to conclude (1.8) and finally (1.9).

$$\mathbf{v}_i(t) = \frac{\mathbf{p}_i(t+\delta t) - \mathbf{p}_i(t-\delta t)}{2\delta t} \rightsquigarrow \mathbf{p}_i(t-\delta t) = \mathbf{p}_i(t+\delta t) - 2\delta t\mathbf{v}_i(t) \tag{1.7}$$

$$\mathbf{p}_i(t+\delta t) = \mathbf{p}_i(t) + \delta t\mathbf{v}_i(t) + \frac{\delta t^2}{2}\mathbf{a}_i(t) + O(\delta t^4) \tag{1.8}$$

$$\mathbf{v}_i(t+\delta t) = \mathbf{v}_i(t) + \frac{\delta t}{2}[\mathbf{a}_i(t) + \mathbf{a}_i(t+\delta t)] + O(\delta t^4) \tag{1.9}$$

Because of the aforementioned improved properties of this method, it is often preferred in MD simulations. [7, 12, 15]

2 AutoPas

This chapter examines the particle simulation library AutoPas and provides an overview of its architecture and features. In Section 2.1, AutoPas' concept of autotuning is introduced, together with the `md-flexible` application. The various configuration parameters available for the simulation are introduced thereafter in Section 2.2. Additionally, Section 2.3 shortly outlines the different tuning strategies for the selection of the optimal combination of these parameters.

The contents of this chapter are mainly drawn from the works introducing AutoPas, specifically the publications by Gratl et al. [8, 9, 10] and Seckler et al. [22], as well as the AutoPas documentation [3].

2.1 Background

AutoPas is an open-source C++ library that facilitates MD simulations. The feature that sets AutoPas apart from other particle simulation software such as LAMMPS[1], ls1-mardyn[2] or GROMACS[3], is the autotuning algorithm.

The aforementioned programs are highly specialized on specific applications and therefore focus on optimizing the algorithms used in these contexts. AutoPas on the other hand tries to provide optimal algorithms for a broader range of scenarios by dynamically switching between different configurations at runtime.

As AutoPas is only a library, we require an application that interacts with AutoPas and provides a front end to allow for our particle simulations. In this work, we will use the `md-flexible` application provided by AutoPas as an example. The program facilitates molecular dynamics simulations based on the LJ potential.

2.2 Configuration Parameters

As outlined in Section 2.1, AutoPas is designed to allow for dynamic adaptation of the algorithmic configuration used in computing the actual simulation steps. Such a

[1] https://www.lammps.org/
[2] https://www.ls1-mardyn.de/
[3] https://www.gromacs.org/

configuration refers to a distinct combination of different algorithmic parameters. The relevant parameters are categorized and described in the following.

2.2.1 Containers

Containers in AutoPas are classes responsible for particle management and neighbor identification. They store the actual particle data in a specific memory layout and allow for the efficient lookup of neighbors, i.e. particles inside the cutoff radius r_c. Grouped by the neighbor identification algorithm, there are currently four different types of containers.

Direct Sum The simplest algorithm is the direct sum: It calculates the distances between the current particle and all other particles, discards those which lie outside of r_c and proceeds with the force calculations on the remaining particles. As this method has complexity $O(n^2)$, it is only suitable for small scenarios.

Linked Cells The linked cell approach divides the simulation space up into a regular grid, consisting of so-called cells. Each particle is then assigned to a cell corresponding to its location in space. Considering a cell size greater or equal to r_c, only neighboring cells have to be considered in the force calculations. For homogeneous particle distributions, this reduces the complexity to $O(n)$. Additionally, particles close to each other in the simulation space are close in memory, which results in good caching behavior and allows for vectorization.

Verlet Lists One drawback of the linked cells algorithm is the high number of particles that lie inside neighboring cells but outside the cutoff radius. They are discarded in the force computation, but still require the distance computations. The Verlet list algorithm solves this issue by introducing a neighbor list of interaction partners for each particle. As rebuilding these lists is expensive, not only neighbors inside the cutoff radius are stored, but also particles that might come into interaction range. This is achieved by extending the radius by a so-called Verlet skin. As particles move, these neighbor lists have to be rebuilt periodically, relying on other neighbor identification algorithms such as linked cells. Furthermore, Verlet lists have a large memory footprint (one list per particle) and do not provide the advantageous memory properties of linked cells.

Verlet Cluster Lists To reduce the overall number of lists in the Verlet list approach, multiple particles can be clustered together, effectively combining their individual neighbor lists to a single one for the whole cluster. This is possible due to the fact that neighboring particles are likely to share multiple of their neighbors. The clustering is based on a subdivision of the simulation domain into a Cartesian grid (x/y) which, extruded along the third dimension (z), form towers. Inside each of these towers, particles are grouped into clusters of size M, ordered by their position along the z-axis. Instead of an exact combination of all cutoff radii, a simple bounding box is constructed around each cluster. Verlet cluster lists not

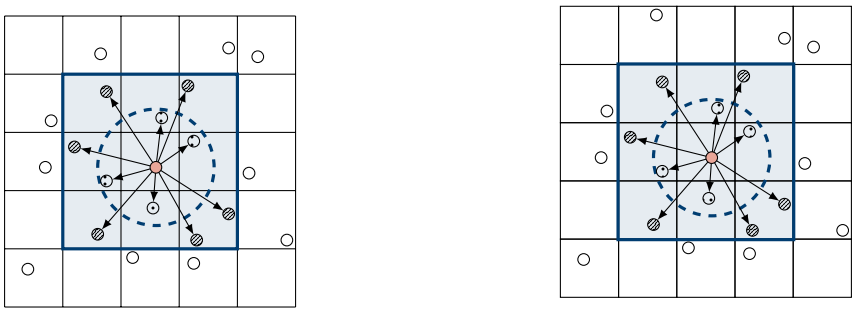


Figure 2.1: asdf

only reduce the total number of neighbor lists, but also allow for vectorization, as clusters are groupings of spatially close particles. On the other hand, the number of particles for which distance calculations have to be performed increases.

2.2.2 Traversals

2.2.3 Additional Parameters

Additionally, there are a number of additional configuration parameters that do not fall into the aforementioned classes. They are given below.

Data Layout This option concerns the memory layout of the particle structures. As each particle has multiple attributes associated to it, all particles together can be laid out either as an Array of Structures (AoS) or a Structure of Arrays (SoA). In the AoS layout, all particles are laid out after each other, in the SoA layout, each attribute type is grouped into an array, where each entry corresponds to the value of this attribute for a specific particle. Figure 2.2 illustrates both principles.

Newton3 As mentioned in Section 1.2.1, Newton's third law states that $F_a = -F_b$ for two bodies a, b exerting forces on each other. This allows for optimizing pairwise interactions, as only one force has to be computed. However, this approach is not always beneficial as it may limit parallelization – once the force is evaluated, both particles must be updated at once.

Cell Size Factor The cell size factor (CSF) parameter specifies the side length of the cells in relation to the interaction cutoff radius (r_c). It can reduce the number of particles for which distances have to be calculated, as a smaller cell side length better approximates the spherical nature of the cutoff radius. This behavior is illustrated in Figure 2.1.

2.3 Tuning Strategies

AutoPas provides a variety of different tuning strategies. These are used in sampling and selecting the new optimal configuration in the tuning phases of the autotuner. As they are not particularly relevant to the topics discussed in this paper, only selected strategies are presented.

FullSearch The default tuning strategy is an exhaustive search over all possible configurations, i.e. all combinations of parameters. The opti-

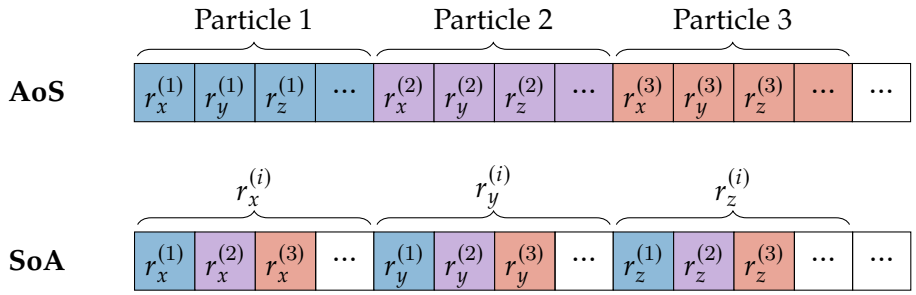


Figure 2.2: Comparison between the Array of Structures (AoS) and Structure of Arrays (SoA) memory layouts. The $r^{(i)}$'s correspond to the position vector of the ith particle.

mal configuration is thereby guaranteed to be trialed at some point. However, the space of all possible configurations grows exponentially in the number of parameters, of which many configurations may be highly suboptimal. Other tuning strategies are therefore more suitable for most scenarios.

RandomSearch The random search tuning strategy randomly selects a given number of configurations which are then sampled. This can greatly reduce the number of configurations to test, but may not select the optimal configuration.

PredictiveTuning The predictive tuning strategy reduces the number of configurations that are sampled during tuning phases by only testing configurations that are expected to perform well. To select which configurations might perform well, the results from previous tuning iterations are used to extrapolate a configurations performance in the current tuning phase. The strategy allows to specify the degree of the interpolation, i.e. how many `full-search` tuning phases are required before the extrapolation takes place. Predictive tuning is typically used with the `slow-config-filter` which blocks configurations that show extremely poor performance from all successive tuning phases. [21]

3 Implementation

To decide on when a new tuning phase should be initiated, we analyze simulation data gathered at runtime. The decision is then made by an algorithm we will refer to as a "trigger strategy". Depending on the scenario and available statistics provided by the simulation, different methods of finding trigger points may be optimal. In this chapter we therefore present the various strategies we investigated. Section 3.1 lays out some key points to consider, independent of any specific tuning strategy. In Section 3.2 we subsequently introduce the strategies we will evaluate in this thesis and their respective mathematical background.

3.1 Considerations

When developing trigger strategies, several aspects must be taken into account. These include the additional computational costs introduced, the types of simulation statistics used, and the criteria by which relevant changes in the simulation scenario are detected. Moreover, the chosen trigger mechanisms do not operate in isolation but interact with tuning strategies. This section outlines these considerations in more detail.

3.1.1 Computational Overhead

Our trigger strategies introduce additional computations, as we have to make decisions based on data that can only be collected at runtime. Therefore, the overhead must be kept as small as possible, otherwise gains made by triggering less tuning phases might easily be dwarfed by the additional computations in each iteration. Furthermore, this can lead to feedback of our method to itself, as our strategies may affect iteration runtime which in turn alters the trigger behavior.

3.1.2 Available Simulation Statistics

AutoPas tracks a number of live simulation statistics. This thesis primarily focuses on runtimes of the current iteration, which include e.g. time spent computing interactions, tuning or rebuilding neighbor lists. In addition to these runtime statistics, the `liveInfo` system reports parameters such as the estimated number of neighbor interactions, the

number of empty cells or the standard deviation of the number of particles in cells. [18]

3.1.3 Detecting Scenario Change

After deciding on which simulation statistic to base the triggering strategies on, one needs to define a notion of "scenario change". We consider two categories in which to classify this change:

Parameter Space Change can occur either in a single parameter or combination of multiple ones (hybrid). The hybrid approach has higher complexity and computational cost, but could be better in scenarios which do not indicate change in the single observed parameter. E.g., a configuration might become suboptimal without any increase in iteration runtime — but a different configuration might be even better suited after e.g. a change in density of the particle distribution.

Type of Variation Depending on the parameters used, change can be indicated by two forms of variation. The first is an increase in the parameter value, the second a change in magnitude: I.e., if the parameter value deviates to much from its starting point in either direction.

In this work, we will investigate strategies which are based on a single parameter (iteration runtime) and trigger at parameter increase.

3.1.4 Interaction with Tuning Strategies

As introduced in Section 2.3, AutoPas offers various tuning strategies. Depending on the specific simulation scenario, one strategy might be more efficient — e.g. the `slow-config-filter` setting in the heating-sphere example. To keep results comparable between scenarios therefore, all experiments were executed using the `full-search` strategy. As this strategy is expected to sample more suboptimal configurations than others, the effect of tuning iterations on the whole simulation runtime is higher. Using more tailored tuning strategies, the improvements as presented in this thesis might not be as visible.

3.2 Time-based Triggers

The simplest approach in detecting whether the current configuration might have become suboptimal, is to observe changes in iteration runtime. As a specific configuration becomes less suitable due to changes in simulation state, one would expect the runtime to increase, as e.g. suboptimal containers lead to unfavorable access patterns. Therefore, the primary focus of this thesis lies on runtime-based strategies in finding trigger points.

The frequency at which new tuning phases are initiated, is indirectly determined by the user through the `trigger-factor` configuration parameter; hereafter denoted as λ. For triggers based on a larger sample set, the parameter `trigger-n-samples`, denoted as n, is additionally used.

3.2.1 Simple Trigger

The most simplistic implementation of a time-based trigger considers only the runtimes of the current and immediately preceding iteration. In other words, if $t_i \geq \lambda \cdot t_{i-1}$, a new tuning phase is triggered. This trigger is implemented as the `TimeBasedSimpleTrigger`.

3.2.2 Single-iteration averaging Trigger

The simple strategy described in Section 3.2.1 is quite unstable. Because of external factors such as hardware heterogeneity, the iteration runtimes are subject to noise. This leads to variability between two successive iterations that is not due to any transformation in the scenario, which is detrimental to the idea of runtime-based detection of scenario change. To diminish the effects of random noise, we extend our sampling interval and average the runtime over multiple samples. This is implemented as the `TimeBasedAverageTrigger`, which differs from the `TimeBasedSimpleTrigger` in that the comparison is performed with respect to the moving average of the last n runtime samples, as in (3.1).

$$t_i \geq \frac{\lambda}{n} \cdot \sum_{k=i-n}^{i-1} t_k \tag{3.1}$$

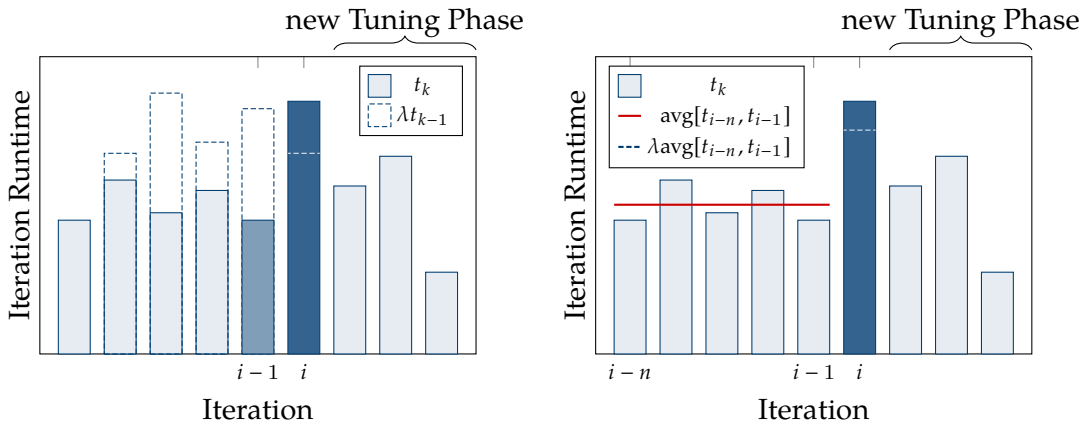


Figure 3.1: Comparison for $\lambda = 1.5$ and $n = 5$ between the `TimeBasedSimpleTrigger` (left) and `TimeBasedAverageTrigger` (right) strategies. A new tuning phase is initiated in both cases, however the `TimeBasedAverageTrigger` is less susceptible to the dip in t_{i-1}.

3.2.3 Interval averaging Trigger

Considering that we expect scenario changes to happen gradually, the runtime might not increase drastically in a single iteration, but rather across a series of subsequent iterations. As the previous two triggers only compare to the current iteration's runtime, they are suboptimal under such circumstances. Taking this effect into account, the `TimeBasedSplitTrigger` splits the measurements of the last n iterations and the current iteration into two intervals A, B as defined in (3.2). A new tuning phase is then initiated if $\text{avg}(B) \geq \lambda \cdot \text{avg}(A)$.

$$A := \left[t_{i-n}, t_{i-j}\right], \quad B := \left[t_{i-j+1}, t_i\right], \quad j = \left\lceil \frac{n}{2} \right\rceil \tag{3.2}$$

3.2.4 Linear Regression Trigger

The `TimeBasedRegressionTrigger` is conceptually similar to the `TimeBasedSplitTrigger`, although with one major difference. Instead of comparing the current interval of runtimes to a previous one, the comparison is based on an estimate of the future runtime based on data of the current interval.

The general idea is to fit a simple linear regression, adapted to our use case, on the last n runtime samples and the current iteration's runtime. Using simple linear regression we obtain a slope estimator $\hat{\beta}_1$, by which we can predict the runtime of the next interval.

In the following, t_k is the runtime at iteration k, i the current iteration and t_{avg}, k_{avg} the average runtime and iteration respectively. The slope estimator $\hat{\beta}_1$ in the standard simple linear regression model is defined as (3.3) [1].

$$\hat{\beta}_1 = \frac{\sum_{k=i-n}^{i}(k - k_{\text{avg}})(t_k - t_{\text{avg}})}{\sum_{k=i-n}^{i}(k - k_{\text{avg}})^2} \tag{3.3}$$

where

$$t_{\text{avg}} = \frac{1}{n+1}\sum_{k=i-n}^{i} t_k, \quad k_{\text{avg}} = \frac{1}{n+1}\sum_{k=i-n}^{i} k \tag{3.4}$$

The value of the estimator $\hat{\beta}_0$, i.e. the intercept at $y = 0$, is not of interest. Similarly, as the samples are taken in discrete steps of one iteration, the values of k can be shifted to the interval $[0, n+1]$. Considering this, the model can be transformed to (3.5).

$$\hat{\beta}_1 \propto \hat{\beta}_1' = \frac{\sum_{k=0}^{n}\left(k - \frac{n(n+1)}{2(n+1)}\right)(t_{i-n+k} - t_{\text{avg}})}{\sum_{k=0}^{n}\left(k - \frac{n(n+1)}{2(n+1)}\right)^2} = \frac{1}{C_2}\sum_{k=0}^{n}(k - C_1)(t_{i-n+k} - t_{\text{avg}}) \tag{3.5}$$

where

$$C_1 = \frac{n}{2}, \quad C_2 = \sum_{k=0}^{n}(k - C_1)^2 = \frac{n(n+1)(n+2)}{12} \tag{3.6}$$

The transformed estimator $\hat{\beta}_1'$ can thus be interpreted as the projected increase in runtime per iteration. This, however, is not a practical metric to compare with a user-set configuration parameter, as it heavily depends on the scenario and would require advance knowledge of the range of iteration runtimes. Therefore, we use a normalization function, such that a slope of $\hat{\beta}_{\text{norm}} = 1.0$ is equal to "no runtime increase". Additionally, the normalization should ensure that $\hat{\beta}_{\text{norm}}$ can be compared to a factor λ that matches the other triggering methods. Given these restrictions, we can derive one such normalization in the following manner: Starting off t_i, we extrapolate the iteration runtimes for the next interval based on $\hat{\beta}_1'$. With that, we compute the area of the triangle representing the additional runtime we expect in the next interval (3.7).

$$A_{\triangle} = \frac{(n+1)^2}{2}\hat{\beta}_1' \tag{3.7}$$

Then, we use t_i as the baseline and add $A_{\triangle}$ for the comparison to the current interval, which results in (3.8).

$$(n+1)t_i + A_\triangle \geq (n+1)\lambda t_{\text{avg}} \tag{3.8}$$

Which can be reordered to the final normalized value (3.9).

$$\hat{\beta}_{\text{norm}} = \frac{2t_i + (n+1)\hat{\beta}'_1}{2t_{\text{avg}}} \tag{3.9}$$

In particular, we have:

(i) $\hat{\beta}_{\text{norm}} = 1$ if there is no projected change in iteration runtime.
(ii) $\hat{\beta}_{\text{norm}} > 1$ if there is a projected increase in iteration runtime.
(iii) $\hat{\beta}_{\text{norm}} < 1$ if there is a projected decrease in iteration runtime.
(iv) $\hat{\beta}_{\text{norm}} = 2$ if there the runtime of the next interval is projected to be double the current interval's runtime.

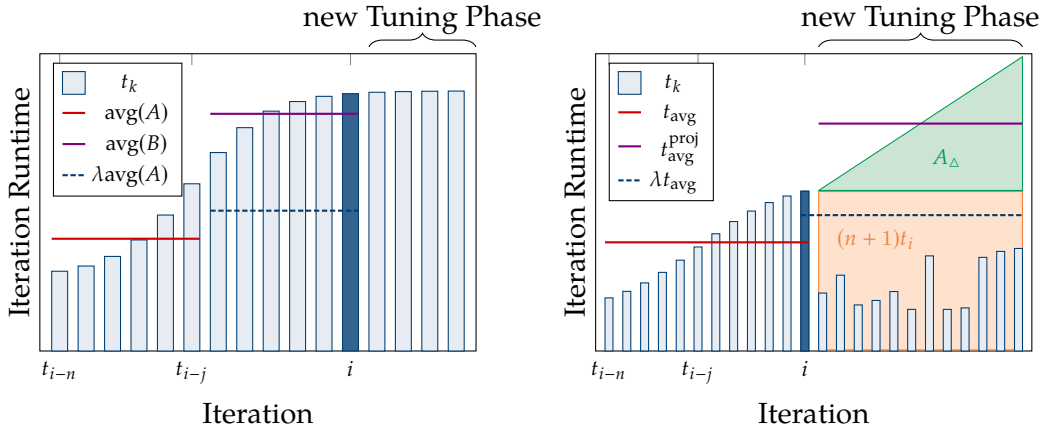


Figure 3.2: Comparison for $\lambda = 1.5$ and $n = 11$ between the `TimeBasedSplitTrigger` (left) and `TimeBasedRegressionTrigger` (right) strategies.

3.3 Hybrid Triggers

As will be discussed later, time-based approaches are not suitable for all scenarios. In these scenarios, iteration runtimes alone might not be a good enough indicator for scenario change. As AutoPas provides additional live simulation statistics through its `liveinfo` interface, these can be used in combination with iteration runtimes to find better strategies in detecting scenario change.

4 Evaluation

This chapter presents the scenarios and criteria employed in the evaluation of our implementation. Section 4.1 introduces a series of benchmarking scenarios, which have been chosen to reflect distinct simulation characteristics appearing in real-world applications. Subsequently, Section 4.2 defines the evaluation metrics applied to these benchmarks. These metrics are intended to provide comparability between simulation runs with dynamically initiated tuning intervals and to the baseline runs with static tuning. Finally, Section 4.3 will shortly outline how default values for the newly introduced trigger parameters can be found.

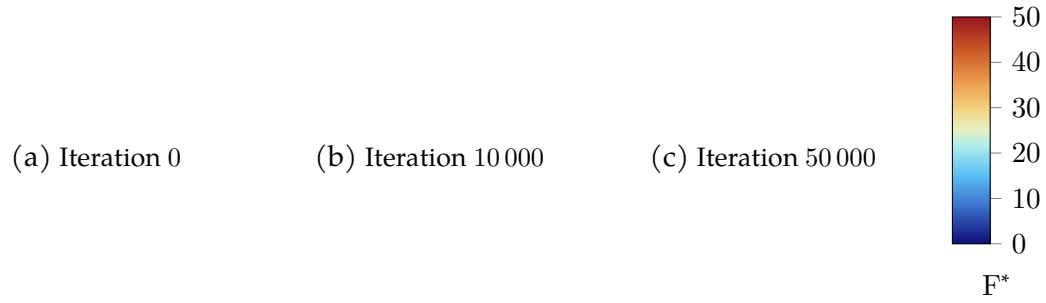
4.1 Benchmarking Scenarios

As to not limit our analysis to one specific simulation setting, we use a selection of benchmarking scenarios. These represent different structures as they may appear in real-world applications. The heating-sphere and exploding-liquid scenarios are based on the ones given by Newcome et al., the configuration files have been adapted and parametrized for use in this thesis [18]. The other scenarios are are taken from the AutoPas `md-flexible` application[1].

4.1.1 Equilibrium

In the equilibrium scenario (Figure 4.1), particles are packed tightly into a cube, periodic boundary conditions are imposed on the simulation space. Periodic boundary conditions ensure that particles exiting the simulation domain are reinserted on the opposite side. The particles interact with each other and the grid structure loosens up, but ultimately an equilibrium is reached in which no rapid changes in velocity occur anymore. After that initial relaxation of the grid structure, no further scenario change expected. Therefore, no additional tuning phases should be needed, as the optimal configuration is not expected to change.

[1] https://github.com/AutoPas/AutoPas/tree/master/examples/md-flexible

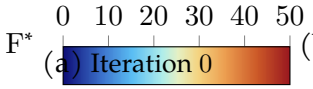


(a) Iteration 0 (b) Iteration 10 000 (c) Iteration 50 000

Figure 4.1: Evolution of the simulation state in the equilibrium scenario. The coloring indicates the forces acting upon a particle, and is given in reduced units.

4.1.2 Exploding Liquid

Similarly to the equilibrium scenario, the exploding liquid scenario (Figure 4.2) starts of with the particles packed into a cuboid, with periodic boundaries applied to the simulation space. The cuboid explodes in y-direction and collides with the boundary. This leads to multiple waves of particles with decreasing intensity, until the simulation finally settles into an equilibrium state, with particles spread out over the whole domain. If a single autotuning instance is used for the whole domain, the rapid changes in particle positions and heterogeneous particle distribution make finding an optimal configuration very hard. However, if the domain is split up into multiple independent AutoPas instances on different MPI nodes, each AutoTuning instance can independently find an optimal configuration for its part of the domain. Using this, the simulation domain can be split up into regions with high particle density and velocities, and regions with little to no particles.

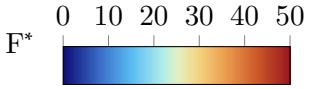


(a) Iteration 0 (b) Iteration 3000 (c) Iteration 11 000 (d) Iteration 31 000

Figure 4.2: Evolution of the simulation state in the exploding liquid scenario.

4.1.3 Heating Sphere

The heating sphere scenario (Figure 4.3) starts off with a dense, small sphere of particles. In contrast to the previously introduced scenarios, reflective boundary conditions are applied. Over the course of the simulation, the temperature rises from 0.1 to 100 with a $\Delta T^* = 0.1$ every 100 iterations. Additionaly, brownian motion, i.e. random fluctuations in particle positions, is applied [17]. The sphere expands with the increasing temperature and particles slowly radiate outwards. In the late phase of the simulation, particles are spread out across the whole domain. Between the initial, compacted state and the equilibrated state, the optimal configuration changes.



(a) Iteration 0 (b) Iteration 4000 (c) Iteration 23 000 (d) Iteration 60 000

Figure 4.3: Evolution of the simulation state in the heating sphere scenario.

4.2 Evaluation Metrics

To compare results between dynamic and static tuning intervals, we use multiple metrics. The primary goal is to reduce the total simulation runtime for a range of typical scenarios; it is therefore our primary metric. As tuning phases spend time in quite suboptimal configurations, a reduction in total runtime is the expected result if our approach reduces the number of tuning phases without spending too many iterations using a suboptimal configuration outside tuning phases.

The metric of total runtime is not particularly fine-grained however, as it only takes into account entire simulation runs. To achieve a more detailed benchmark, we also consider the number of iterations that were computed under the optimal configuration. As an approximation to the optimal configuration in each iteration, we use simulation runs with tuning phases at fixed frequency, a high number of tuning samples and short tuning intervals. Based on this approximation we can then rank the configuration our dynamic run chose in terms of optimality.

Finally, we also consider the number of tuning phases initiated or, more precisely, the number of tuning iterations over the course of the whole simulation. Otherwise, we could not differentiate wether any achieved speedup is due to our trigger strategies or the fact that not triggering any tuning phases at all was more efficient for a given scenario.

4.3 Default Trigger Parameters

All presented trigger strategies are based on a `trigger-factor` λ. The averaging, split and regression triggers additionally take into account the number of samples to inspect, denoted as n. For any dynamic tuning trigger to be useful, reasonable default values for these parameters are needed, as the performance of the whole simulation is dependent on the trigger's behavior. Furthermore, optimal values for these parameters may depend on the scenario, trigger strategy or both. These default values can be found by comparing a range of combinations (λ_i, n_j) for any given scenario and trigger strategy.

5 Results

...

5.1 Experimental Setup

The data analyzed in this chapter was collected on the Linux-Cluster of the Leibniz-Rechenzentrum[1]. The nodes in the `cm4` cluster consist of processors in the Sapphire Rapids family (Intel® Xeon® Platinum 8480+) with 2.1 GiB of memory per logical CPU and 488 GiB per node [14]. For benchmarking purposes, the AutoPas library and `md-flexible` were compiled with Spack GCC 13.2.0 and Intel MPI 2021.12.0 on commit `46bb925c8c5827faee8691afefd9f714630f20f1`.

The scripts used to generate the Slurm jobs and configuration files can be found in the repository of this thesis[2].

5.2 Choice of Simulation Statistics

As referred to before in Section 3.1.2 there are several simulation statistics available upon which we could base our trigger strategies. Even iteration runtimes themselves are subdivided into the time spent on computing interactions, traversing remainders and rebuilding neighbor lists. In this thesis, we will consider the sum of these times with the exception of the rebuilding measurements. This choice can be justified by inspecting runtime data we collected. As shown in Figure 5.1, the rebuild times smooth out the overall measurements and thus decrease the effectivity at which a scenario change can be detected. Moreover, the rebuilding of neighbor lists happens at a fixed `rebuildFrequency`, which leads to problems in trigger strategies with a low number of samples, as the rebuild iterations greatly outweigh all non-rebuild iterations.

Figure 5.1: asdf

[1] https://www.lrz.de/
[2] https://github.com/ladnik/bachelor-thesis

5.3 Computational Overhead

To exemplify the importance of optimizing the trigger routines, Figure 5.2 illustrates the runtime differences between naive and optimized triggers in the equilibrium scenario. The naive version recalculates the average over all samples each iteration, whereas the optimized version uses a ring buffer and running summation to reduce computational cost. The speedup experienced is not only due to a lowering of computational overhead, but also due to less tuning iterations. This fact can be explained by the aforementioned self influence of the triggers: higher overhead might lead to higher fluctuation in iteration runtime which in turn leads to unstable trigger behavior, especially in the averaging trigger.

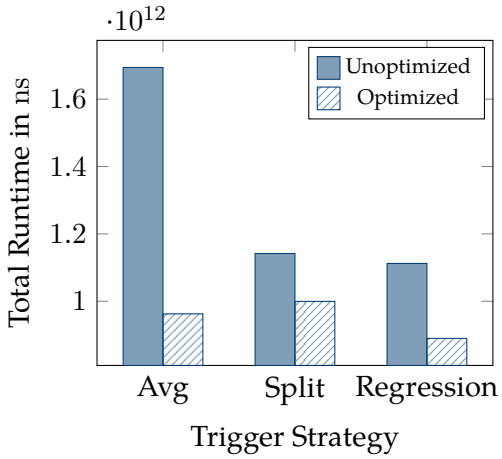


Figure 5.2: Average Speedup between unoptimized and optimized runs for the `TimeBasedAverage`, `TimeBasedSplit` and `TimeBasedRegression` strategies.

5.4 Benchmarking Results

The blue bars in the graphs represent the runtime of that particular iteration. In the configuration plots, the colored background identifies the used configuration: same configurations map to the same color. The gaps in the plot are where tuning iterations have been logged – as their runtime is not relevant for the scenario change and would distort the actual runtime plot, they are not reported here. The red vertical lines indicate the start of a tuning phase.

5.4.1 Equilibrium

As can be seen in Figure 5.3, a trigger factor of $\lambda = 1.5$ leads to increased speedup compared to $\lambda = 1.25$ in nearly all triggering strategies. This is however mainly due to the nature of the equilibrium scenario: after the initial configuration selection, the optimal configuration is not expected to change. Therefore, not initiating any new tuning phases will lead to a decrease in total simulation runtime. That the speedup is indeed a result of the decreased number of tuning iterations can be verified by looking at the right-hand side plots; for the simple and averaging trigger it is most noticeable. Additionally, triggers with a larger sample size will typically trigger less frequently, as more of the variability in iteration runtime is smoothed out. For a too large number of samples, the speedup decreases however, as the computational overhead is directly proportional to the number of samples.

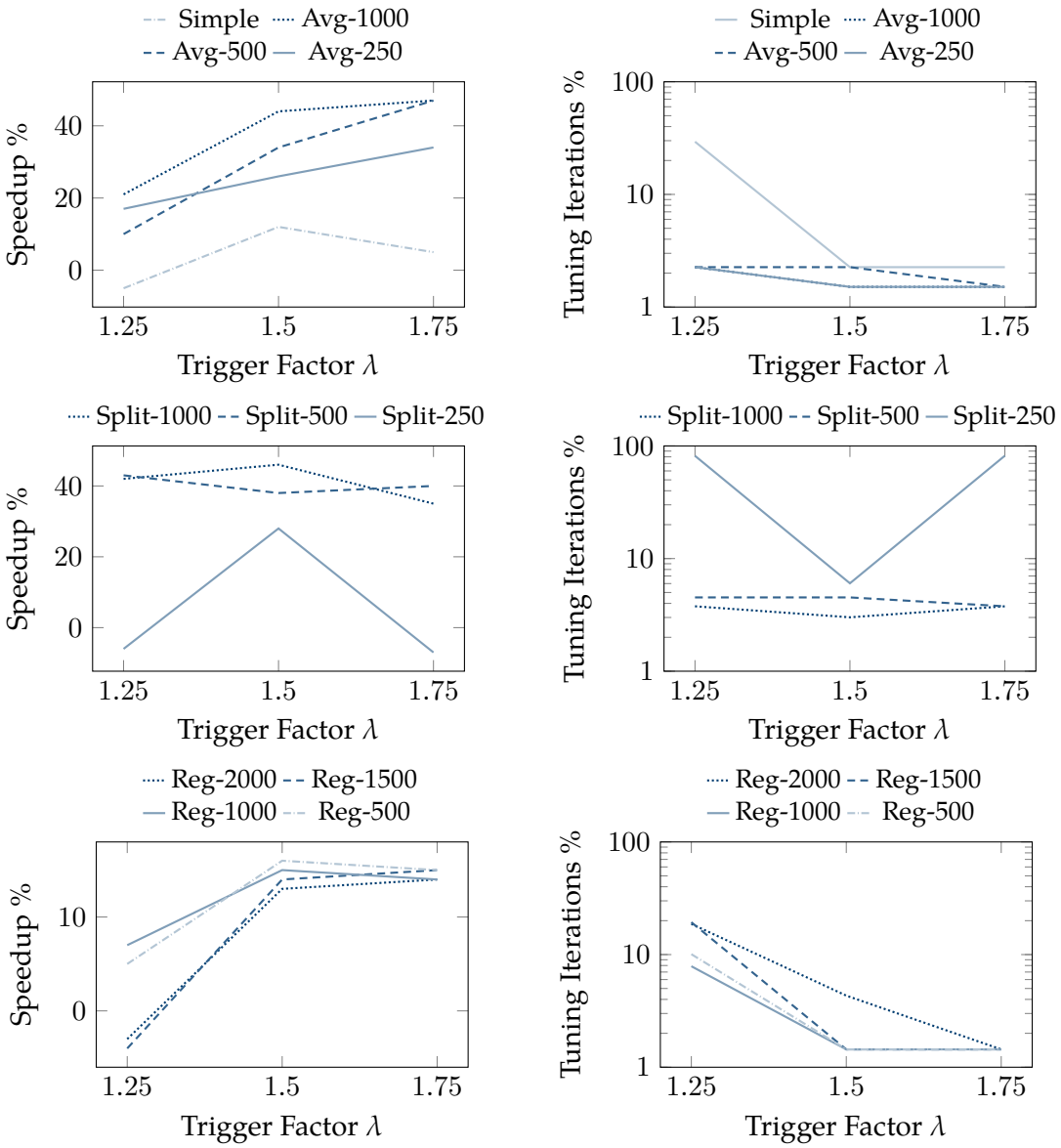


Figure 5.3: Trigger behavior in the equilibrium scenario, the numbers in the legends refer to the number of samples n considered. Note the logarithmic scale in the plots on the right hand side.

The collected data suggests default parameters as presented in Table 5.1.

Trigger	Trigger factor λ	Number of samples n
`TimeBasedSimple`	1.5	–
`TimeBasedAverage`	1.75	500
`TimeBasedSplit`	1.5	1000
`TimeBasedRegression`	1.5	500

Table 5.1: Suggested default parameters for the equilibrium scenario.

5.4.2 Exploding Liquid

5.4.3 Heating Sphere

5.4.4 Runtime and Number of Tuning Iterations

5.4.5 Optimality

6 Conclusion

This chapter will ...

6.1 Dynamic Initiation of Tuning Intervals

6.2 Future Work

This page is intentionally left blank.

Bibliography

[1] Bovas Abraham and Johannes Ledolter. *Introduction to Regression Modeling*. Belmont, CA: Duxbury Press, 2006.

[2] V. Arnold, V. Kozlov, and A. Neishtadt. "Mathematical aspects of classical and celestial mechanics. Transl. from the Russian by A. Iacob. 2nd printing of the 2nd ed. 1993". In: *Itogi Nauki i Tekhniki Seriia Sovremennye Problemy Matematiki* (Jan. 1985).

[3] AutoPas. *AutoPas Doxygen Documentation*. Accessed: 2025-08-28. 2025. URL: https://autopas.github.io/doxygen_documentation/git-master/.

[4] M. Born and R. Oppenheimer. "Zur Quantentheorie der Molekeln". In: *Annalen der Physik* 389.20 (1927), pp. 457–484. DOI: https://doi.org/10.1002/andp.19273892002.

[5] J. D. Emberson et al. "Cosmological neutrino simulations at extreme scale". In: *Research in Astronomy and Astrophysics* 17.8 (Aug. 2017), p. 085. ISSN: 1674-4527. DOI: 10.1088/1674-4527/17/8/85. URL: http://dx.doi.org/10.1088/1674-4527/17/8/85.

[6] Daan Frenkel and Berend Smit. "Chapter 4 - Molecular Dynamics Simulations". In: *Understanding Molecular Simulation (Second Edition)*. Ed. by Daan Frenkel and Berend Smit. Second Edition. San Diego: Academic Press, 2002, pp. 63–107. ISBN: 978-0-12-267351-1. DOI: https://doi.org/10.1016/B978-012267351-1/50006-7.

[7] Alexander Fulst and Christian Schwermann. *Molekulardynamiksimulation*. Accessed: 2025-08-23. 2013. URL: https://www.uni-muenster.de/Physik.TP/archive/fileadmin/lehre/TheorieAKkM/ws13/Fulst-Schwermann.pdf.

[8] Fabio Alexander Gratl et al. "AutoPas: Auto-Tuning for Particle Simulations". en. In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, May 2019. ISBN: 9781728135106. DOI: 10.1109/ipdpsw.2019.00125. URL: https://ieeexplore.ieee.org/document/8778280.

[9] Fabio Alexander Gratl et al. "N Ways to Simulate Short-Range Particle Systems: Automated Algorithm Selection with the Node-Level Library AutoPas". In: *Computer Physics Communications* 273 (2021), p. 108262. DOI: 10.1016/j.cpc.2021.108262.

[10] Fabio Alexander Gratl-Gaßner. "AutoPas: Automated Dynamic Algorithm Selection for HPC Particle Simulations". PhD thesis. Technische Universität München, 2025. URL: https://mediatum.ub.tum.de/1765326.

[11] Michael Griebel, Gerhard Zumbusch, and Stephan Knapek. *Numerical Simulation in Molecular Dynamics: Numerics, Algorithms, Parallelization, Applications*. Vol. 5. Texts in Computational Science and Engineering. Springer Berlin Heidelberg, 2007. ISBN: 978-3-540-68094-9. DOI: 10.1007/978-3-540-68095-6.

[12] Ernst Hairer, Christian Lubich, and Gerhard Wanner. "Geometric numerical integration illustrated by the Störmer–Verlet method". In: *Acta Numerica* 12 (2003), pp. 399–450. DOI: 10.1017/S0962492902000144.

[13] Scott A. Hollingsworth and Ron O. Dror. "Molecular Dynamics Simulation for All". In: *Neuron* 99.6 (2018), pp. 1129–1143. DOI: 10.1016/j.neuron.2018.08.011.

[14] Leibniz Supercomputing Centre. *Job Processing on the Linux-Cluster*. Accessed: 2025-09-01. 2025. URL: https://doku.lrz.de/job-processing-on-the-linux-cluster-10745970.html.

[15] Benedict Leimkuhler and Sebastian Reich. "Geometric integrators". In: *Simulating Hamiltonian Dynamics*. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2005, pp. 70–104.

[16] Johannes Lenhard, Simon Stephan, and Hans Hasse. "On the History of the Lennard-Jones Potential". In: *Annalen der Physik* 536.6 (2024), p. 2400115. DOI: https://doi.org/10.1002/andp.202400115.

[17] Peter Mörters and Yuval Peres. *Brownian motion*. Vol. 30. Cambridge University Press, 2010.

[18] Samuel James Newcome et al. "Algorithm Selection in Short-Range Molecular Dynamics Simulations". In: (May 2025). DOI: 10.48550/ARXIV.2505.03438. arXiv: 2505.03438 [cs.CE].

[19] Isaac Newton. *Mathematical Principles of Natural Philosophy*. Ed. by Florian Cajori. Trans. by Andrew Motte. First English translation 1729; revised edition. Berkeley: University of California Press, 1934.

[20] Eric J.R. Parteli and Thorsten Pöschel. "Particle-based simulation of powder application in additive manufacturing". In: *Powder Technology* 288 (2016), pp. 96–102. ISSN: 0032-5910. DOI: https://doi.org/10.1016/j.powtec.2015.10.035.

[21] Julian Mark Pelloth. "Implementing a predictive tuning strategy in AutoPas using extrapolation". MA thesis. Technical University of Munich, Sept. 2020.

[22] Steffen Seckler et al. "AutoPas in ls1 mardyn: Massively parallel particle simulations with node-level auto-tuning". en. In: *Journal of Computational Science* 50 (2021), p. 101296. DOI: 10.1016/j.jocs.2020.101296.

[23] J P Verboncoeur. "Particle simulation of plasmas: review and advances". In: *Plasma Physics and Controlled Fusion* 47.5A (Apr. 2005), A231. DOI: 10.1088/0741-3335/47/5A/017.

[24] Troy Van Voorhis. *XII. The Born–Oppenheimer Approximation*. MIT OpenCourseWare, *Introductory Quantum Mechanics I (5.73), Fall 2005*. Lecture notes, Section XII (The Born–Oppenheimer Approximation). 2005. URL: https://ocw.mit.edu/courses/5-73-introductory-quantum-mechanics-i-fall-2005/bf19f723f60f6baeba12abcb6b97f6f5_sec12.pdf.

[25] Xipeng Wang et al. "The Lennard-Jones potential: when (not) to use it". In: *Phys. Chem. Chem. Phys.* 22 (19 2020), pp. 10624–10633. DOI: 10.1039/C9CP05445F.

This page is intentionally left blank.