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.

TODO

This page is intentionally left blank.

Zusammenfassung

TODO

This page is intentionally left blank.

Contents

List of Figures x
List of Tables xiii

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Molecular Dynamics . . . 2
1.2.1 Newton's Laws of Motion . . . 2
1.2.2 Lennard-Jones Potential . . . 3
1.2.3 Störmer-Verlet Algorithm . . . 3

2 AutoPas **5**
2.1 Background . . . 5
2.2 Configuration Parameters . . . 6
2.2.1 Containers . . . 6
2.2.2 Traversals . . . 7
2.2.3 Additional Parameters . . . 7
2.3 Tuning Strategies . . . 8

3 Implementation **10**
3.1 Considerations . . . 10
3.1.1 Computational Overhead . . . 10
3.1.2 Available Simulation Statistics . . . 10
3.1.3 Detecting Scenario Change . . . 11
3.1.4 Interaction with Tuning Strategies . . . 11
3.2 Time-based Triggers . . . 11
3.2.1 Simple Trigger . . . 12
3.2.2 Single-iteration averaging Trigger . . . 12
3.2.3 Interval averaging Trigger . . . 12
3.2.4 Linear Regression Trigger . . . 13

4 Evaluation **15**
4.1 Benchmarking Scenarios . . . 15
4.1.1 Equilibrium . . . 15
4.1.2 Exploding Liquid . . . 15
4.1.3 Heating Sphere . . . 16
4.2 Evaluation Metrics . . . 17
4.3 Default Trigger Parameters . . . 17

5 Results **19**

5.1 Experimental Setup 19
5.2 Choice of Simulation Statistics 19
5.3 Computational Overhead 20
5.4 Benchmarking Results 21
5.4.1 Equilibrium 22
5.4.2 Exploding Liquid 24
5.4.3 Heating Sphere 25
5.4.4 Overview 27
5.5 Hybrid Triggers 27

6 Conclusion **30**

Bibliography 32

Appendices

7 Additional Data **38**

This page is intentionally left blank.

List of Figures

1.1 Various real-world applications of particle simulations. [TODO] 1
1.2 An illustration of the 12-6 LJ potential well, 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] 4

2.1 A comparison of the different Containers implemented in AutoPas [TODO] 7
2.2 An illustration of traversals in AutoPas. [TODO] 7
2.3 Impact of the cell size factor on the number of distance calculations. [TODO] 8
2.4 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. 8

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}. 12
3.2 Comparison for $\lambda = 1.5$ and $n = 11$ between the `TimeBasedSplitTrigger` (left) and `TimeBasedRegressionTrigger` (right) strategies. [TODO] 14

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. . 16
4.2 Evolution of the simulation state in the exploding liquid scenario. 16
4.3 Evolution of the simulation state in the heating sphere scenario. 17

5.1 Rebuild and non-rebuild times in the equilibrium (left) and heating-sphere (right) scenario. [TODO] 20
5.2 Performance comparisons between the various trigger strategies in the equilibrium scenario. [TODO] 21
5.3 Examples of trigger behavior in the equilibrium scenario. [TODO] 22
5.4 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. 23
5.5 Ranking of configurations selected by the best run in the equilibrium scenario for each trigger strategy, compared to baseline static interval run. Only non-tuning iterations are considered. [TODO] 24
5.6 Examples of trigger behavior in the heating-sphere scenario. [TODO] . . . 25
5.7 Trigger behavior in the heating-sphere scenario, the numbers in the legends refer to the number of samples n considered. [TODO: Scales] 26

5.8 Ranking of configurations selected by the best run in the heating-sphere scenario for each trigger strategy, compared to baseline static interval run. Only non-tuning iterations are considered. [TODO] 27
5.9 Average Speedup between unoptimized and optimized runs for the `TimeBasedAverage`, `TimeBasedSplit` and `TimeBasedRegression` strategies. . . . 28
5.10 Iteration runtime (left) and the maximum densities of particles per cell (right). The top row shows a run with the single configuration `VL-List_Iter-NoN3L-AoS`, the bottom row shows `LC-C04-N3L-AoS-CSF1` 29

This page is intentionally left blank.

List of Tables

5.1 Suggested default parameters for the equilibrium scenario. 24
5.2 Suggested default parameters for the heating sphere scenario. 27

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 equations of particle trajectories, the Lennard-Jones potential as a model for pairwise interactions, and the Störmer-Verlet algorithms as numerical schemes 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, for example, 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 finding such numerical solutions to a 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. In 2017, for instance, the TianNu project simulated 2.97×10^{12} particles 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

Figure 1.1: Various real-world applications of particle simulations. [TODO]

optimal algorithmic 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 as 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. One iteration of said loop consists of calculating the forces between particles and integrating the equations of motion. These two steps are repeated until an equilibrium is reached, at which point any 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 latter exerts force $F_b = -F_a$ on the first body.

The second law is particularly significant, as it allows to compute the trajectories of particles based on the forces acting upon them. The third law, while secondary in dynamics, is useful especially regarding optimizations: for pairwise interactions, any 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 r_c can be introduced, beyond which the forces can be assumed negligible. This significantly reduces the computational complexity, as only the interactions between close neighbors have to be computed. Under optimal conditions, the number of interaction computations can be reduced to $O(n)$. [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 provides a sufficiently good approximation, such that macroscopic effects can be derived from simulating interactions at a molecular 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 1.2. [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).

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

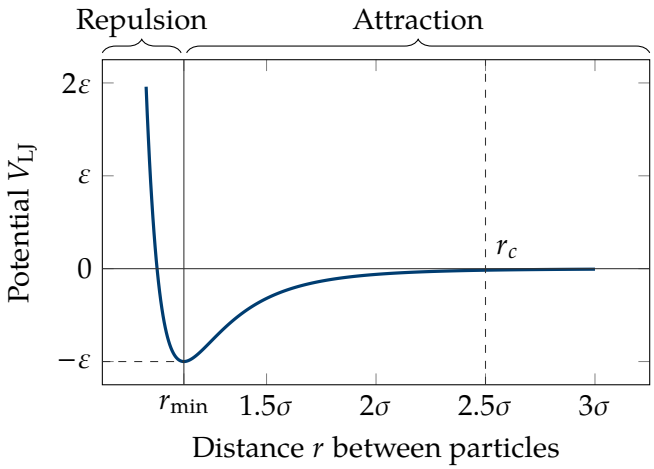


Figure 1.2: An illustration of the 12-6 LJ potential well, with the minimum of $-\varepsilon$ at $r_{\min} = \sigma\sqrt[6]{2}$, zero-crossing at σ and cutoff radius r_c. The figure is based on Lenhard et al. [16]

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

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, the concept of autotuning is introduced, as well as the `md-flexible` application. The various algorithmic configuration parameters available 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 short-range 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 environments. AutoPas, on the other hand, tries to provide optimal simulation conditions for a broader range of scenarios by implementing a wide selection of algorithms and switching between them at runtime. This removes the need for expert knowledge in simulation setup and allows for a simple interface through which the AutoPas library can be viewed as a black-box.

As referred to earlier, this autotuning approach is currently implemented as follows: In each iteration that is a multiple of a predefined `tuning-interval`, a tuning phase is initiated. In these tuning phases, a number of configurations are selected to be sampled. These configurations each represent a distinct combination of algorithmic settings (c.f. Section 2.2). Each configuration is sampled, i.e. executed for a set number of iterations, after which the measurements are condensed into a single value, referred to as "evidence". This evidence is used to rank the sampled configurations based on a tuning

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

metric; either runtime or energy consumption. The best configuration is selected to compute the subsequent iterations until the beginning of the next tuning phase.

As AutoPas is only a library, we require an application that interacts with it and provides a front end to allow for our particle simulations. In this work, we will use the `md-flexible` application, provided together with AutoPas. Based on the LJ potential, `md-flexible` facilitates MD simulations with integrated parallelization, distributed memory computation, load balancing, and highly configurable scenario generators.

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. 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 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 $O\left(n^2\right)$, it is only suitable for small scenarios.

Linked Cells The linked cell approach divides the simulation space up into cells along a regular grid. Each particle is then assigned to the 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 simulation space are close in memory, which results in cache-friendly 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.

Figure 2.1: A comparison of the different Containers implemented in AutoPas [TODO]

Figure 2.2: An illustration of traversals in AutoPas. [TODO]

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), forms several 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. Thus, Verlet cluster lists not 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

Containers provide an efficient way to identify neighbors of a particle — to efficiently compute the interactions themselves however, the traversal, i.e. the order in which particles are iterated over, is equally important. Traversals are relevant for the performance mostly due to memory and cache access patterns. Different container types require traversals, tailored to the data structures used in storing the neighbors. A very limited selection of these traversals will be explained hereafter.

C01 Base Step The c01 base step is not a traversal strategy on its own, but defines the cells in which interaction computations must be performed for any given particle. It is implemented for the Linked Cells and Verlet List Cells containers as `lc_c01` and `vlc_c01` respectively. As the simplest base step, it computes all interactions with the neighboring cells of the particle's base cell.

C18 Base Step Similarly, c18 computes interactions only on its forward neighbors, potentially halving the number of calculations that have to be performed. This base step is enabled by Newton's third law (cf. Section 1.2.1), where only one of the interacting particles needs to compute the interaction force. However, some form of synchronization must be employed, as to avoid race conditions on force updates. This limits the extent to which parallelization is possible.

VL List Iteration The VL List Iteration is the only traversal strategy available for Verlet lists. All lists are processed in parallel, within a given list the particles are traversed in sequential order.

2.2.3 Additional Parameters

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

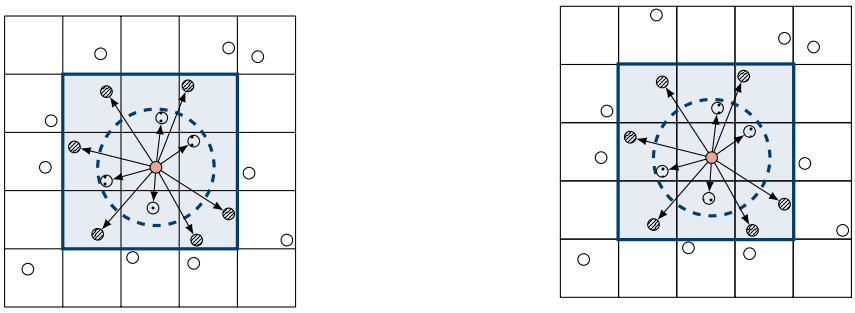


Figure 2.3: Impact of the cell size factor on the number of distance calculations. [TODO]

Data Layout The data layout option concerns the layout of the particle structures in memory. 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 stored after each other; in the SoA layout, each attribute type is stored in a separate array, with each entry holding the value for a specific particle. Figure 2.4 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 2.3.

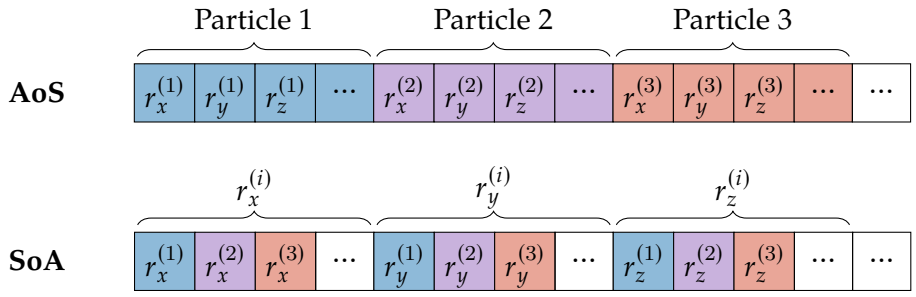


Figure 2.4: Comparison between the Array of Structures (AoS) and Structure of Arrays (SoA) memory layouts. The $r^{(i)}$'s correspond to the position vector of the ith particle.

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 possi-

ble configurations, i.e. all combinations of parameters. The optimal 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 predict which configurations might be optimal, the results from previous tuning iterations are used to extrapolate performance in the current tuning phase. The strategy allows to specify the degree of accuracy, 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).

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

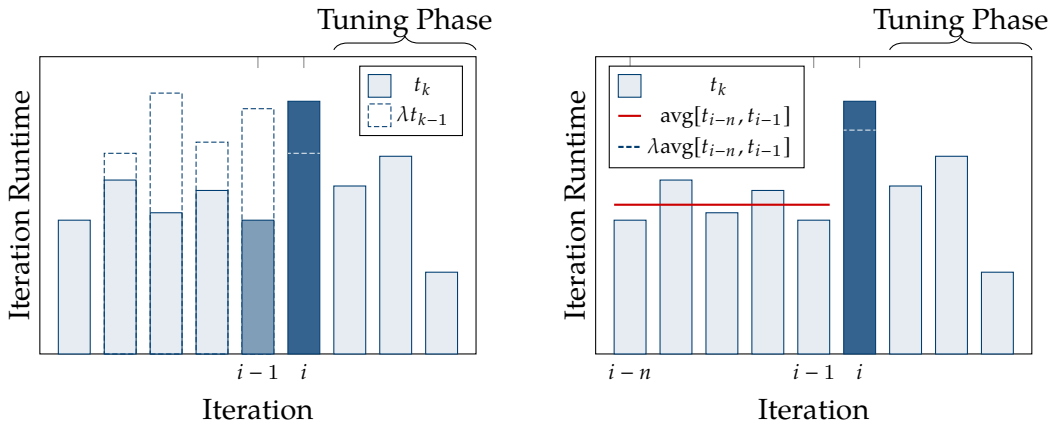


Figure 3.1: Comparison for $\lambda = 1.5$ and $n = 5$ between the `TimeBasedSimpleTrigger` (left) and `TimeBasedAverageTrigger` (right) strategies. A new tuning phase is initiated in both cases, however the `TimeBasedAverageTrigger` is less susceptible to the dip in t_{i-1}.

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.

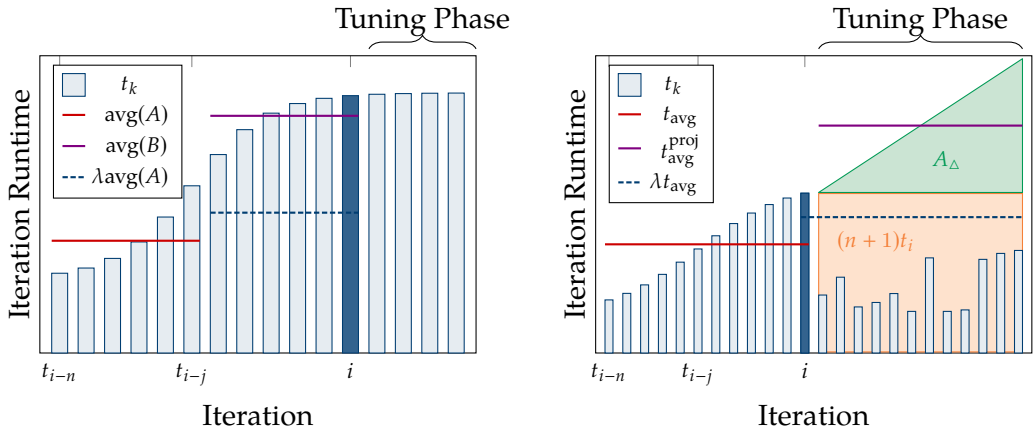


Figure 3.2: Comparison for $\lambda = 1.5$ and $n = 11$ between the `TimeBasedSplitTrigger` (left) and `TimeBasedRegressionTrigger` (right) strategies. [TODO]

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 that may appear in real-world applications. Subsequently, Section 4.2 defines the evaluation metrics applied to the benchmarks. These metrics are intended to provide comparability between simulation runs with dynamically initiated tuning intervals and to the baseline runs with tuning at fixed frequency. 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 that represent different basic particle structures. The heating-sphere and exploding-liquid scenarios are based on the configuration files given by Newcome et al., adapted and parametrized for use in this thesis [18]. The other scenarios are are taken from the AutoPas `md-flexible` application[1].

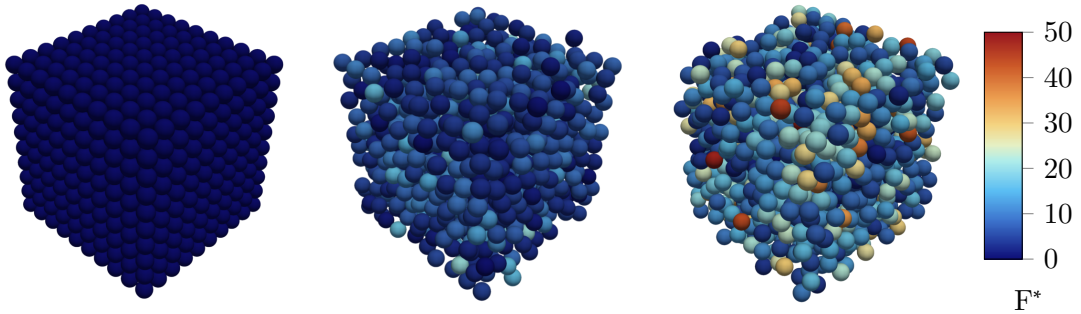
4.1.1 Equilibrium

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

4.1.2 Exploding Liquid

Similarly to the equilibrium scenario, the exploding liquid scenario (Figure 4.2) starts off with the particles packed into a cuboid, with periodic boundaries imposed on the

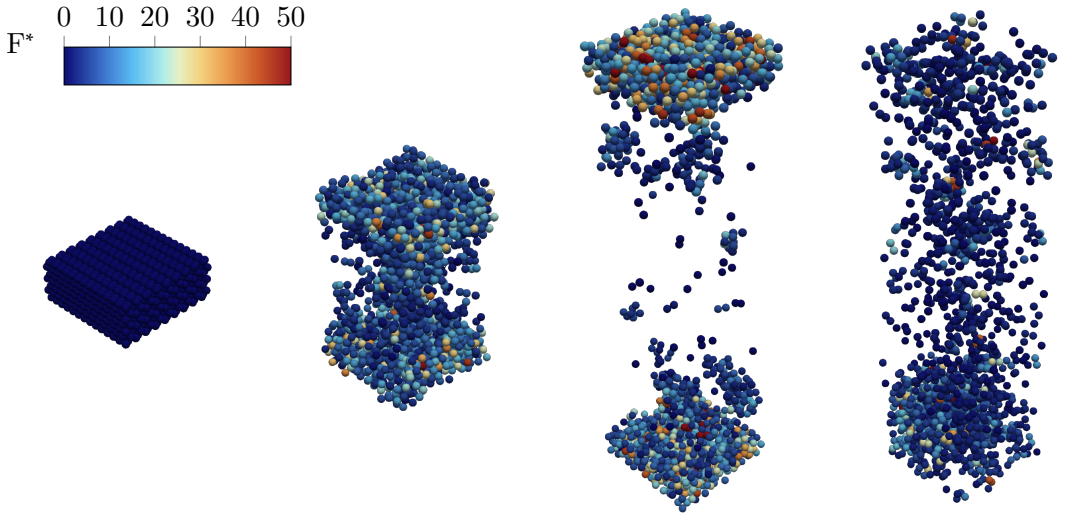
[1] https://github.com/AutoPas/AutoPas/tree/master/examples/md-flexible/input



(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.

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.



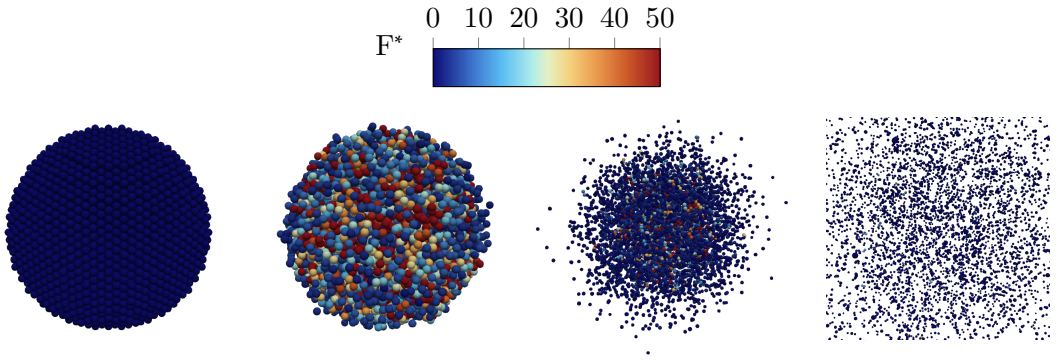
(a) Iteration 0 (b) Iteration 3000 (c) Iteration 11 000 (d) Iteration 31 000

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 change of $\Delta T^* = 0.1$ every 100 iterations. Additionally, Brownian motion is

applied, i.e. random fluctuations in particle positions [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 our dynamic initiation of tuning phases and the currently implemented static approach, we use multiple metrics. The primary goal is to reduce the total simulation runtime for a range of typical scenarios; it is therefore our first 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 computing too many iterations using a suboptimal configuration.

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 implementation 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 user-set 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. Reasonable default values can be

found by comparing a range of combinations (λ_i, n_j) for any given scenario and trigger strategy.

5 Results

The data collected as part of the evaluation of the introduced trigger strategies is presented and discussed in this section. The hardware and software setup are given in Section 5.1, as to allow for reproducibility of our results. Sections 5.2 and 5.3 discuss some implementation choices based on collected data. The main benchmark results including achieved speedups and default trigger parameters are presented in Section 5.4, grouped by scenario. Finally, Section 5.5 shows statistics collected through the `LiveInfo` system, as to motivate hybrid triggers.

5.1 Experimental Setup

The measurements collected for analysis in this chapter were obtained on the CoolMUC4 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 trigger strategies could be based. Even the iteration runtimes themselves are differentiated into multiple parameters: 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 remain constant over all iterations simulated with a particular configuration. Their inclusion therefore does not provide any new

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

information, but rather smooths out the overall measurements and thus decreases the effectivity at which a scenario change can be detected.

Additionally, in scenarios with a low average number of neighbors, the rebuilding of neighbor lists takes longer than the interaction computations. Considering that the rebuilding only happens in iterations that are a multiple of `rebuildFrequency`, this leads to stability problems in trigger strategies with a low number of samples, as the few rebuild iterations greatly outweigh all non-rebuild iterations.

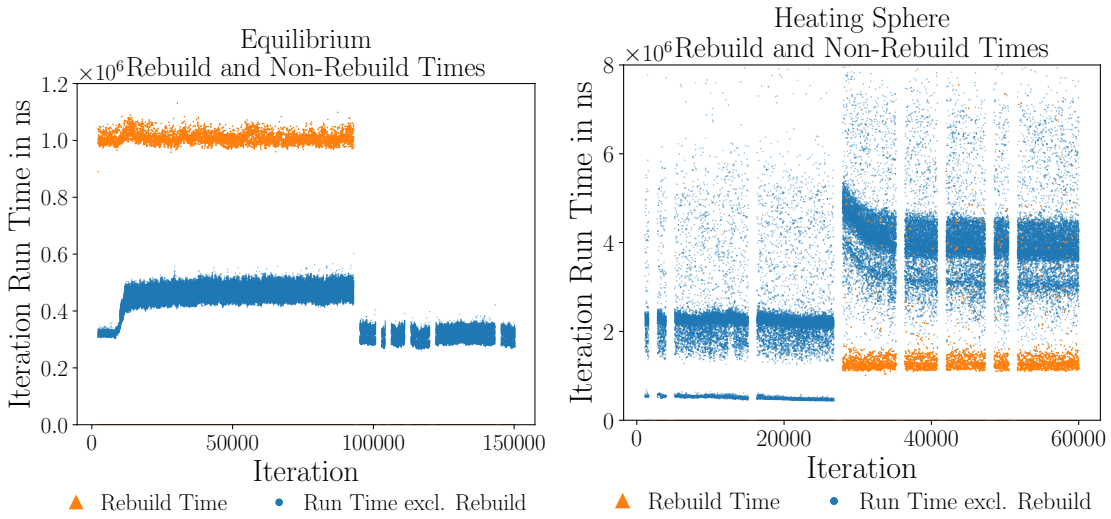


Figure 5.1: Rebuild and non-rebuild times in the equilibrium (left) and heating-sphere (right) scenario. [TODO]

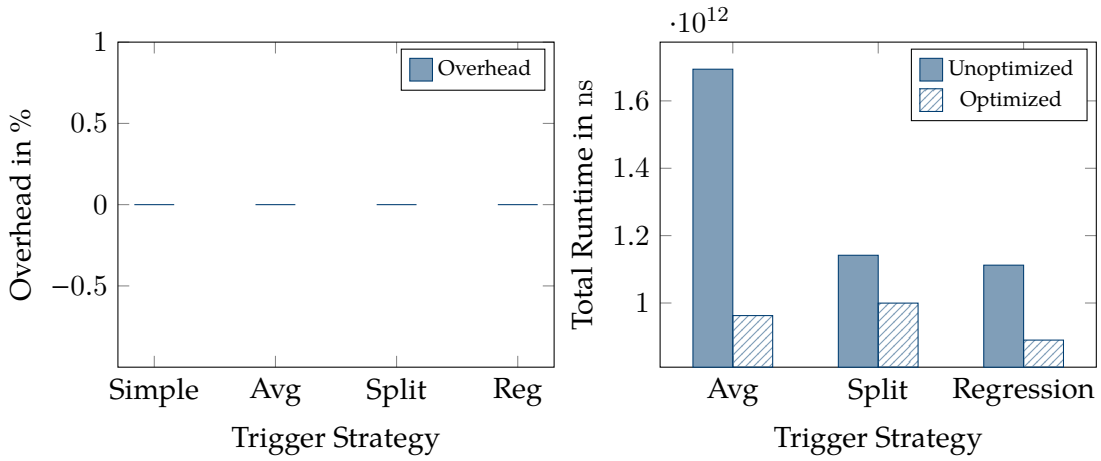
5.3 Computational Overhead

Our strategies analyze data at runtime and therefore need additional computations in each iteration. The performance impact of these should be negligible in comparison to the simulation steps, as otherwise any performance gains due to fewer tuning phases are nullified.

To quantify the overhead our strategies introduce, we compare runs with a single initial tuning phase. That way, we remove any changes in runtime that may occur due to different tuning phase initiation points of our strategies. To achieve this, we use a factor high enough, such that none of our strategies initiate a new tuning phase. For the baseline run, we use static tuning intervals with a `tuning-interval` set longer than the total number of iterations to simulate, which results in a single tuning phase starting in iteration 0. Figure 5.2a shows the overhead obtained that way for the equilibrium scenario with $\lambda = 50$ and $n = 500$.

To exemplify the importance of optimizing the trigger routines, Figure 5.2b 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 a lower number of tuning iterations. That 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.



(a) Overhead of the various trigger strategies. [TODO]

(b) Average Speedup between unoptimized and optimized runs.

Figure 5.2: Performance comparisons between the various trigger strategies in the equilibrium scenario. [TODO]

5.4 Benchmarking Results

The relative speedups presented in the following plots were computed by the formula given in (5.1), where t_{baseline} represents the runtime with tuning phases at fixed intervals and t_{dynamic} the runtime of our implementation.

$$S = \frac{t_{\text{baseline}}}{t_{\text{dynamic}}} - 1 \tag{5.1}$$

In the plots showing the selected configurations for a given run, the blue scatter dots represent the runtime of that particular iteration. The colored background identifies the used configuration: same configurations map to the same color in a given plot. The gaps are where tuning iterations have been logged — as their runtime is not relevant for our purposes and would distort the actual runtime plot, they are not reported here. The dashed gray lines indicate the start of a new tuning phase.

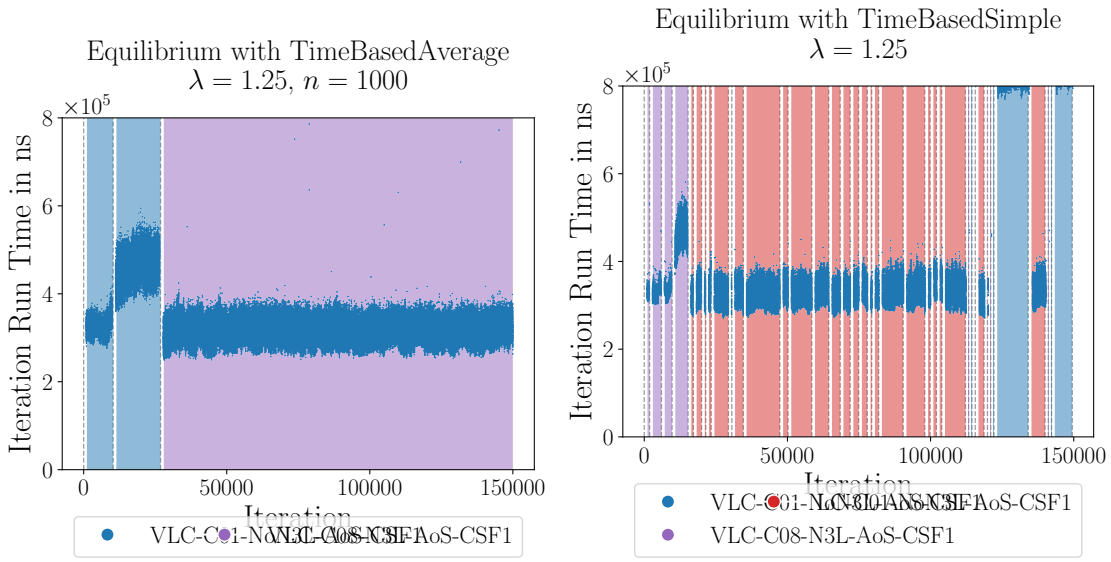
Additional plots and data can be found in the Appendix, Chapter 7. [TODO]

5.4.1 Equilibrium

Selected Runs

Figure 5.3 shows two of the experimental runs in detail. On the left hand side, a `TimeBasedAverageTrigger` with $\lambda = 1.25$, $n = 1000$ reliably detects scenario change. Two tuning phases are started in the initial phase, where overall iteration runtime increases. After the scecond tuning phase, a better configuration is found. As there is no further indication that the simulation state changes, the remaining iterations are performed using this configuration. As was explained in Section 4.1.1, it is indeed the case that no further configuration change is needed. In this run therefore, the presented trigger was beneficial.

On the right hand side, a worst-case outcome is shown. The `TimeBasedSimpleTrigger` used in that run triggered too many new tuning phases, which lead to an increase in total simulation runtime compared to the baseline run. The main reason for the overreaction lies in the implementation of the trigger: as long as the runtime of one iteration is greater than λ times that of its predecessor, a tuning phase is initiated. In combination with the high variance shown in the runtime samples, it leads to clearly suboptimal performance.



(a) Good tuning results. (b) Too many tuning phases triggered.

Figure 5.3: Examples of trigger behavior in the equilibrium scenario. [TODO]

Speedup and Default Parameters

As can be seen in Figure 5.4, 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.

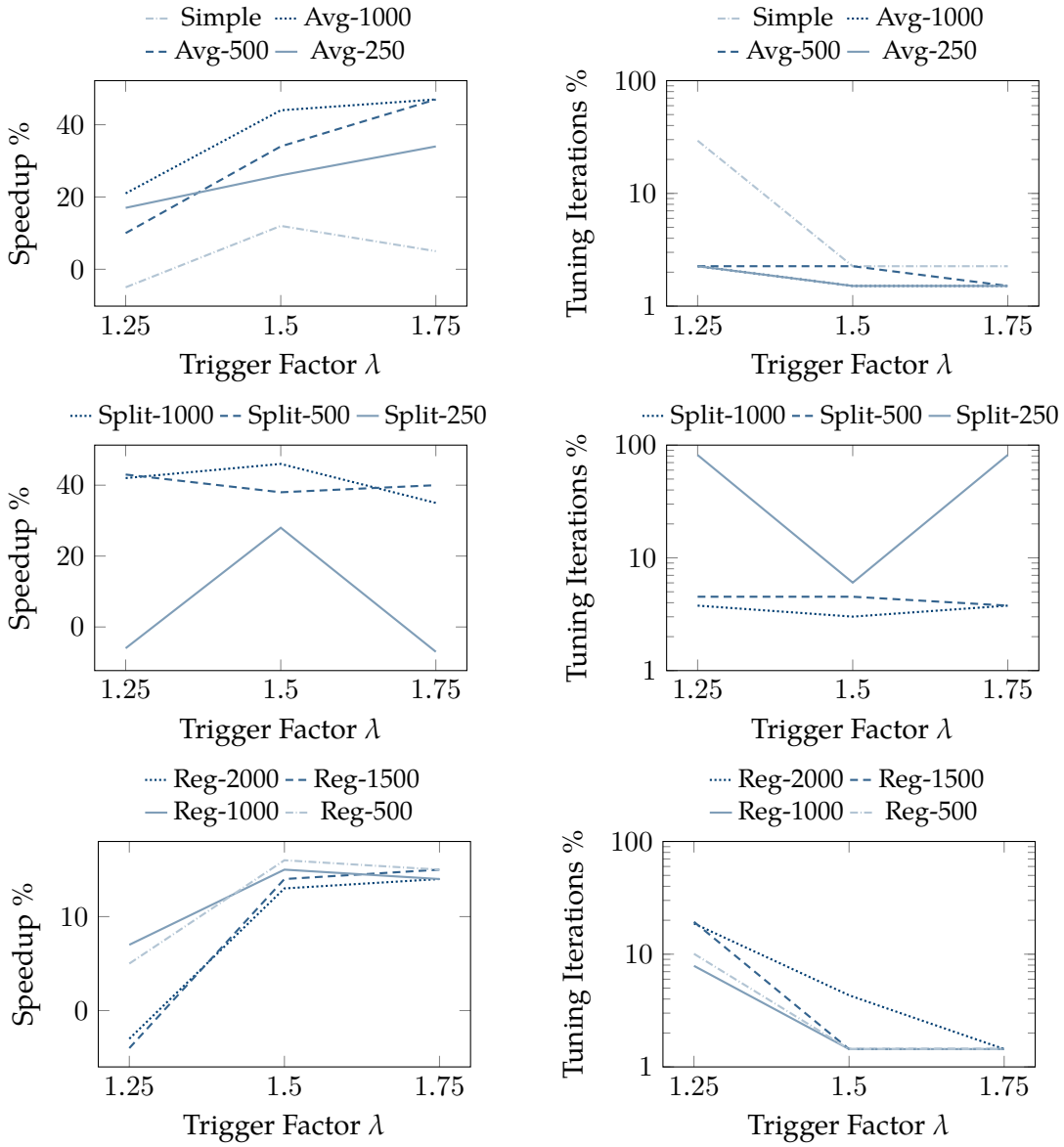


Figure 5.4: Trigger behavior in the equilibrium scenario, the numbers in the legends refer to the number of samples n considered. Note the logarithmic scale in the plots on the right hand side.

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

Optimality

Our second evaluation metric as stated in Section 4.2 concerns the quality of the chosen configurations. For efficient computation, we expect the configuration at any iteration to be one of the best choices. As can be seen in Figure 5.5, this is achieved across all strategies, with all configurations being one of the top three choices for that specific iteration. In fact, the simple and averaging triggers achieve up to 98 % of all non-tuning iterations using the optimum configuration.

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.

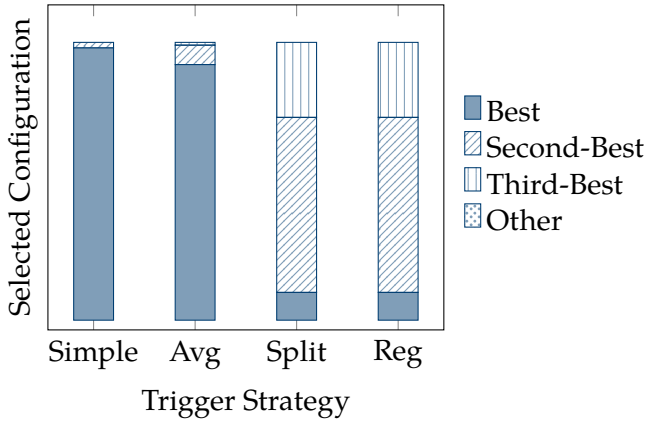


Figure 5.5: Ranking of configurations selected by the best run in the equilibrium scenario for each trigger strategy, compared to baseline static interval run. Only non-tuning iterations are considered. [TODO]

5.4.2 Exploding Liquid

Speedup and Default Parameters

Selected Runs

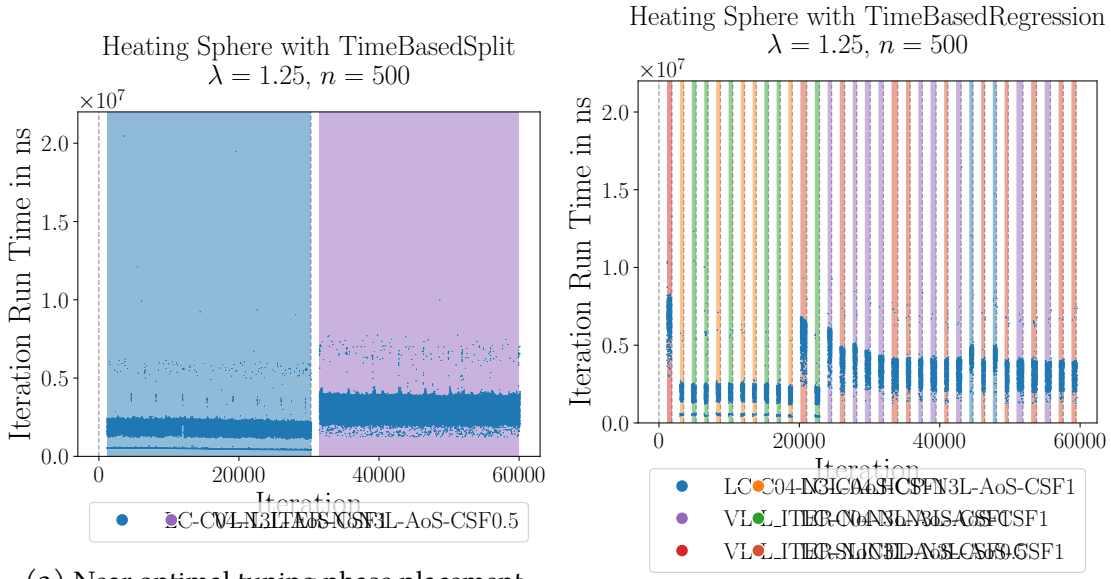
Optimality

5.4.3 Heating Sphere

Selected Runs

Figure 5.6 again presents two sample runs as to illustrate optimal and unsatisfactory results. The left figure shows the near optimal initiation of tuning phases by the `TimeBasedSplitTrigger`. The first tuning phase aligns with the sphere in its compact state, slowly expanding; the latter half corresponds to the particles spread out over the domain. For these two states, one configuration is optimal for each, the displayed trigger response is therefore appropriate However it seems that there is no clear trend in the data itself, therefore ...

Worse results can be seen in the `TimeBasedRegressionTrigger`, pictured on the right hand side.



(a) Near optimal tuning phase placement.

(b) Too many tuning phases triggered.

Figure 5.6: Examples of trigger behavior in the heating-sphere scenario. [TODO]

Speedup and Default Parameters

The heating sphere scenario experiences high variance in iteration runtimes, which directly influences the behavior of our trigger strategies, as shown in Figure 5.7. The triggers which are unstable due to these variations are the `TimeBasedAverageTrigger`, `TimeBasedSimpleTrigger` and `TimeBasedRegressionTrigger`, with the latter two being affected the most. In some cases, this results in our strategies under-performing the static approach by up to $-23\,\%$, as in the regression case. There, the instability of the simple linear regression approach to outliers becomes apparent and is reflected in the number of tuning phases triggered. This suggests, that the regression based trigger may not be used in scenarios that behave similarly. To address this issue, the linear least squares estimation in the regression trigger could be replaced with a more robust approach like the Theil-Sen estimator [26].

On the other hand, the `TimeBasedSplitTrigger` performs exceptionally well, with speedups of up to $76\,\%$. The averaging of the runtime samples appears to filter out the noise enough such that the trigger behaves as intended.

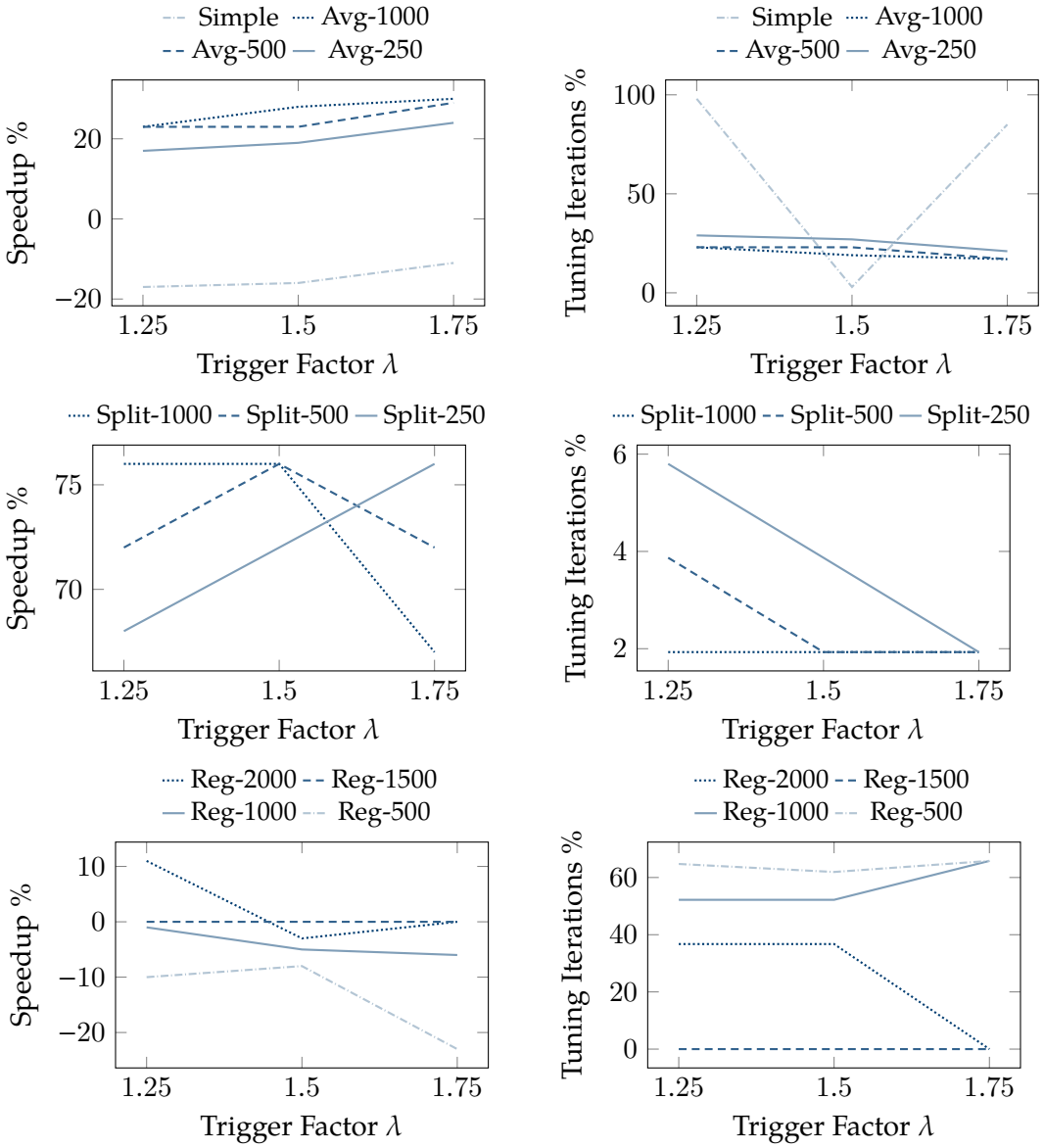


Figure 5.7: Trigger behavior in the heating-sphere scenario, the numbers in the legends refer to the number of samples n considered. [TODO: Scales]

The collected data suggests default parameters as presented in Table 5.2.

Trigger	Trigger factor λ	Number of samples n
`TimeBasedSimple`	not recommended	–
`TimeBasedAverage`	1.75	500
`TimeBasedSplit`	1.5	500
`TimeBasedRegression`	not recommended	–

Table 5.2: Suggested default parameters for the heating sphere scenario.

Optimality

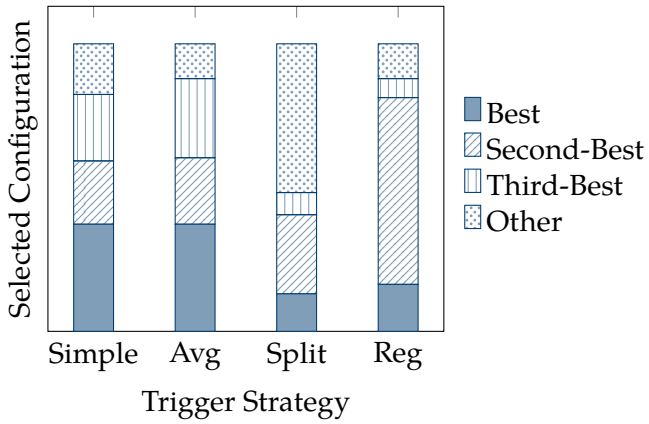


Figure 5.8: Ranking of configurations selected by the best run in the heating-sphere scenario for each trigger strategy, compared to baseline static interval run. Only non-tuning iterations are considered. [TODO]

5.4.4 Overview

In the previous sections, the various trigger strategies were analyzed within each scenario. To better visualize the results of our benchmarks, Figure 5.9 lays out the best speedup achieved, grouped by scenario. The equilibrium scenario provides overall good conditions for application of our methods, as can seen by the speedup in all trigger strategies. In the heating sphere scenario, all strategies perform worse, except for the `TimeBasedSplitTrigger` which seems to be perfectly suited, with speedups of up to 76 %.

5.5 Hybrid Triggers

For some scenarios, time-based approaches may not be suitable for all scenarios. In these scenarios, iteration runtimes alone might not be a good enough indicator for scenario change. As referred to before in Section 3.1.2, AutoPas provides additional live simulation statistics through its `liveinfo` interface. These could be used in combination with iteration runtimes to find better strategies in detecting scenario change. As a motivation of this approach, Figure 5.10 shows an exemplary run in the heating-sphere scenario. Using static containers, the initial optimal configuration is `VL-List_Iter-NoN3L-AoS`, later on it changes to `LC-C04-N3L-AoS-CSF1`. [18] The iteration runtime

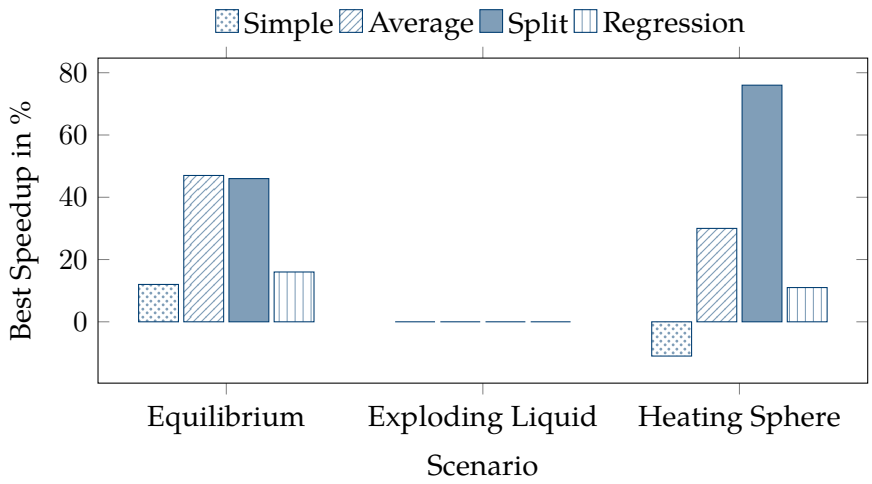


Figure 5.9: Average Speedup between unoptimized and optimized runs for the `TimeBasedAverage`, `TimeBasedSplit` and `TimeBasedRegression` strategies.

does not change, even tough a better configuration is available; the `maxDensity` statistics however, show clear indication of the shift towards a different simulation state.

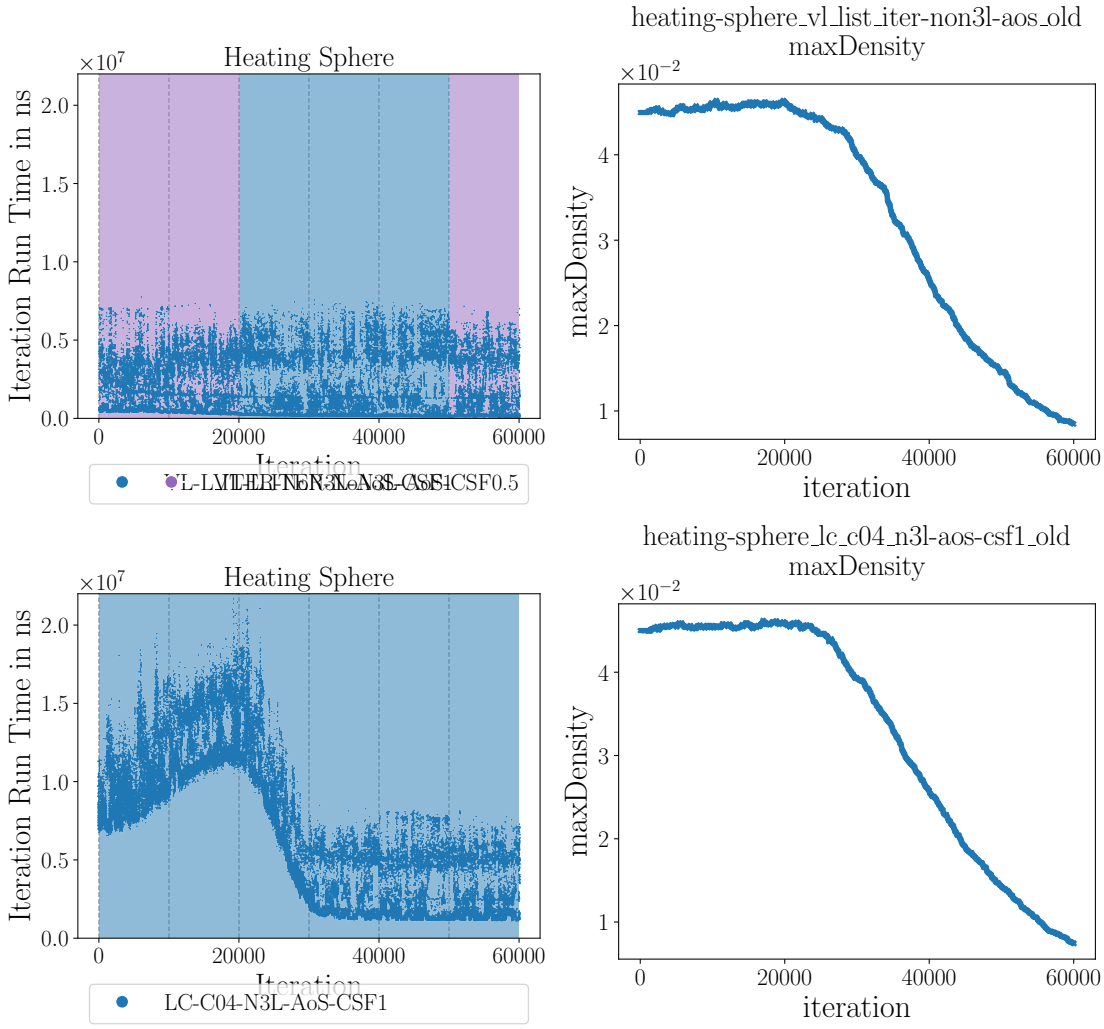


Figure 5.10: Iteration runtime (left) and the maximum densities of particles per cell (right). The top row shows a run with the single configuration `VL-List_Iter-NoN3L-AoS`, the bottom row shows `LC-C04-N3L-AoS-CSF1`

6 Conclusion

In this thesis, four new methods to dynamically initiate new tuning phases in AutoPas were introduced. Additionally, reasonable default values for the corresponding control parameters were derived empirically. It was shown that our trigger strategies decrease simulation runtime across almost all tested scenarios when using `full-search` as the tuning strategy. Especially in settings with low variance in iteration runtime, all strategies reduce the number of iterations spent in tuning phases without any significant decrease in the optimality of the used configurations. The most promising candidate was shown to be the comparison of averaged intervals, due to its resilience to noise in the live input data, leading to speedups of up to $76\,\%$ under optimal conditions. The other strategies investigated are more susceptible to the aforementioned fluctuations in the input data. Nonetheless, these triggers still perform well in scenarios with low variance between iterations.

However, as new tuning strategies are introduced, the dynamic initiation of tuning intervals will likely become less relevant for single-node applications. In particular, tuning strategies based on machine learning lead to cheap tuning phases [18], which in turn significantly diminishes the achievable speedups. One application in which a dynamic approach as ours might still be of use, is in distributed memory setups: As each node runs its own AutoPas instance, it can trigger tuning phases independently from other nodes. In scenarios with heterogeneous particle distribution over the whole domain, the best configuration on a specific node is likely to change separate from other parts of the domain. Thus, our proposed method may still be advantageous.

Possible future work may explore the analysis of additional live simulation statistics, either as single parameter or hybrid strategies. The introduction of novel triggering algorithms providing more stability, such as linear regression using the Theil-Sen estimator, should be considered. More advanced methods such as digital filters have to be implemented efficiently, as not to inflate per-iteration overhead. Another interesting subject for further research could be the combination of static and dynamic tuning intervals; at fixed, but much shorter intervals, a dynamic trigger would evaluate whether to start a new tuning phase or not.

In summary, the dynamic initiation of tuning phases has been shown to be lightweight method of decreasing unnecessary tuning phases, whilst still ensuring good configuration fit. Using more efficient tuning strategies reduces the performance gains in single-node applications, but some benefits remain in multi-node settings.

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.

[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". In: *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, May 2019. ISBN: 9781728135106. DOI: 10.1109/ipdpsw.2019.00125.

[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.

[26] Rand Wilcox. “Chapter 10 - Robust Regression”. In: *Introduction to Robust Estimation and Hypothesis Testing* (*Third Edition*). Ed. by Rand Wilcox. Third Edition. Statistical Modeling and Decision Science. Boston: Academic Press, 2012, pp. 471–532. ISBN: 978-0-12-386983-8. DOI: https://doi.org/10.1016/B978-0-12-386983-8.00010-X.

This page is intentionally left blank.

Appendices

This page is intentionally left blank.

7
Additional Data

This page is intentionally left blank.