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

Adaptive Initiation of AutoPas Tuning Phases for Efficient Particle Simulations

Adaptive Einleitung 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 Kumar Mishra, M.Sc. (hons)
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

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

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`

Abstract

Particle simulations have established themselves as an indispensable tool in scientific research, and are used across a wide range of applications ranging from biophysics to materials science. The efficiency of such simulations is strongly influenced by the choice of computational strategies and their corresponding parameters. However, determining the optimal algorithmic configuration often requires expert knowledge and is highly dependent on the specific scenario studied.

AutoPas is a particle simulation library that addresses this issue by dynamically adjusting the algorithms employed in the simulation at runtime, based on live simulation data. This enables researchers to interact with AutoPas through a simple black-box interface, without requiring extensive knowledge in setting up optimal simulation parameters. To achieve this dynamic reconfiguration, AutoPas reevaluates the optimal algorithmic settings in tuning phases, initiated at fixed intervals. This thesis proposes four novel methods for determining ideal points for the dynamic initiation of tuning phases, based on data collected during the simulation itself.

Experimental results demonstrate that our approach reduces simulation runtime by up to $47\,\%$ compared to fixed tuning intervals with exhaustive search, whilst still selecting suitable configurations. Moreover, our implementation is lightweight, introducing on average only $0.9\,\%$ overhead per iteration. The newly introduced methods are shown to be particularly well suited for MPI-parallel applications.

Zusammenfassung

Partikelsimulationen haben sich längst als unverzichtbare Hilfsmittel in der Wissenschaft etabliert und kommen in zahlreichen Bereichen, von der Biophysik bis hin zu den Materialwissenschaften, zum Einsatz. Die Effizienz solcher Simulationen wird stark von der Wahl der verwendeten Simulationsmethoden und den entsprechenden Parametern beeinflusst. Die Bestimmung der optimalen Konfiguration erfordert jedoch oft Expertenwissen und hängt vom jeweiligen untersuchten Szenario ab.

AutoPas ist eine Programmbibliothek für Partikelsimulationen, die dieses Problem zu lösen versucht, indem die in der Simulation verwendeten Algorithmen zur Laufzeit auf Grundlage von Live-Simulationsdaten dynamisch angepasst werden. Dadurch können Endanwender über eine einfache Black-Box-Schnittstelle mit AutoPas interagieren, ohne über umfangreiche Kenntnisse zur Einrichtung optimaler Simulationsparameter verfügen zu müssen. Um diese dynamische Neukonfiguration zu erzielen, ermittelt AutoPas in festgelegten Intervallen die beste algorithmische Konfiguration erneut. Die vorliegende Arbeit stellt vier neue Methoden vor, um basierend auf im Verlauf der Simulation gesammelten Daten ideale Auslösungszeitpunkte dieser Tuning-Phasen bestimmen.

Die gesammelten Performanzergebnisse zeigen, dass unser Ansatz die Simulationslaufzeit im Vergleich zu statischen Tuning-Intervallen mit vollständiger Suche des Parameterraums um bis zu $47\,\%$ reduziert, wobei dennoch geeignete Konfigurationen ausgewählt werden. Darüber hinaus verursacht unsere Implementierung nur einen zusätzlichen Laufzeit-Overhead von durchschnittlich $0.9\,\%$ pro Iteration. Die neu entwickelten Strategien zeigen sich als besonders effektiv in MPI-parallelisierten Anwendungen.

Contents

List of Figures

List of Tables

1 Introduction

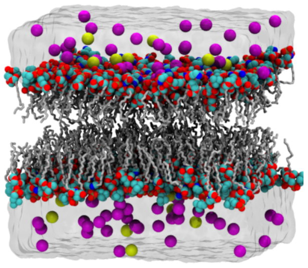
This chapter introduces the fundamental concepts necessary to understand Molecular Dynamics (MD) simulations. We begin with a discussion of the motivation behind the general n-body problem and the goals of this thesis in Section 1.1. Afterwards, the components of a simple MD simulation loop are presented 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 [6]. These days, particle simulations have established themselves as indispensable tools across a wide range of scientific fields. Applications reach from drug design [14] to plasma physics [25] and materials science [21]. Two such applications are illustrated in Figure 1.1.

One example of a software framework enabling n-body simulations is the AutoPas library [9]. 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 algorithmic configurations without requiring expert knowledge during setup ("autotuning").


(a) Simulation of a lipid bilayer, representative for cell membranes. [24]

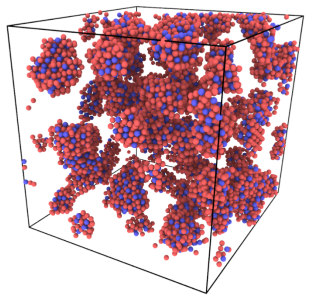

(b) Simulation of crystal growth in metallic glass during annealing. [5]

Figure 1.1: Real-world applications of MD simulations.

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 solve 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. On that scale, 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 as it is 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, 12, 26]

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

The simplest interatomic potentials one could apply here describe the interactions between only two particles, such as the Gravitational, Lennard-Jones or Coulomb potentials [12]. 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. [7]

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. [20]

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. [10]

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 $O(n)$. [10]

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). [17]

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

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). The parameters ε and 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. [17, 27]

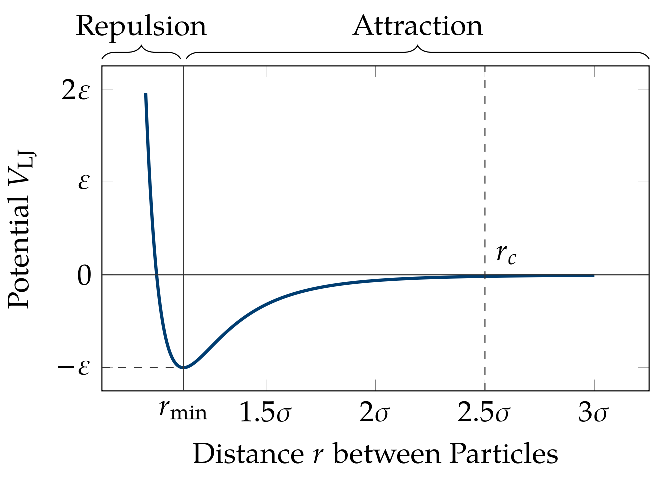


Figure 1.2: An illustration of the 12-6 LJ potential well, with the minimum of $-\varepsilon$ at $r_{\min} = \sigma\sqrt[6]{2}$, zero-crossing at σ and cutoff radius r_c. The figure is based on Lenhard et al. [17].

1.2.3 Störmer-Verlet Algorithm

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

The Störmer-Verlet algorithm is one such numeric method for solving these systems. 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).

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

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

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, $\mathbf{a}_i(t) = \frac{\mathbf{F}_i(t)}{m_i}$.

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

Using this result, we could already calculate the velocities of the individual particles. However, there are some drawbacks, e.g., high error propagation [8]. 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. [8, 13, 16]

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, together with 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. [9, 10, 11] and Seckler et al. [23], 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 ls1-mardyn[1], LAMMPS[2], or GROMACS[3], is the autotuning algorithm.

The mentioned simulation programs are highly specialized on specific applications and therefore focus on optimizing the algorithms used in these environments. In contrast, AutoPas aims to provide optimal simulation conditions across a wider range of scenarios by implementing a broad set of algorithms and dynamically switching between them at runtime. This removes the need for expert knowledge in simulation setup and allows for a simple interface by 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). All configurations are sampled, i.e., executed for a set number of iterations, after which the measurements are condensed into a single value, referred to as

[1] `https://www.ls1-mardyn.de/`

[2] `https://www.lammps.org/`

[3] `https://www.gromacs.org/`

"evidence." This evidence is used to rank the sampled configurations based on a tuning 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 12-6 LJ potential and Störmer-Verlet time integration, `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.

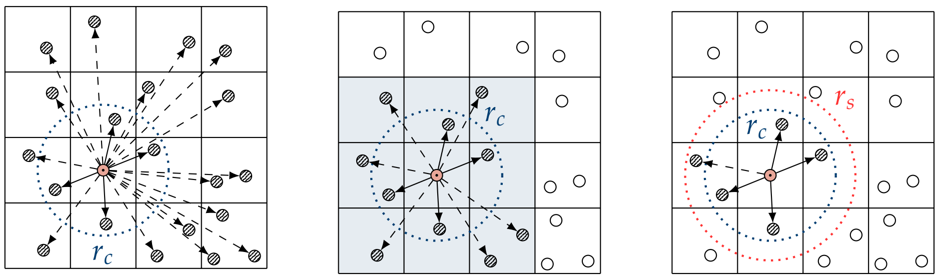
2.2.1 Containers

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 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 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 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 move 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 par-



(a) Direct Sum (b) Linked Cells (c) Verlet Lists

Figure 2.1: An illustration of selected neighbor identification algorithms used for containers in AutoPas. Particles for which distance calculations are performed are marked with a diagonal line pattern, dashed arrows lead to particles outside the cutoff radius. This figure is based on Gratl et al. [10].

ticle) 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), 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 also 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 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.

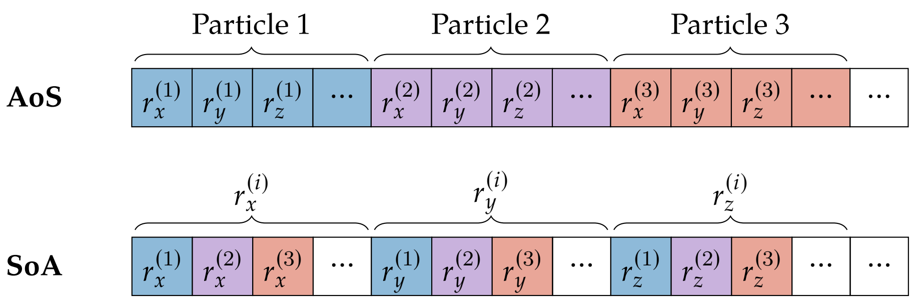
2.2.3 Additional Parameters

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

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 2.2 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 2.3.

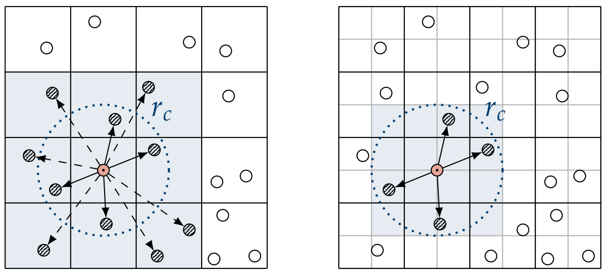


Particle 1
Particle 2
Particle 3
AoS
$r_x^{(1)}$ $r_y^{(1)}$ $r_z^{(1)}$ ··· $r_x^{(2)}$ $r_y^{(2)}$ $r_z^{(2)}$ ··· $r_x^{(3)}$ $r_y^{(3)}$ $r_z^{(3)}$ ··· ···
$r_x^{(i)}$
$r_y^{(i)}$
$r_z^{(i)}$
SoA
$r_x^{(1)}$ $r_x^{(2)}$ $r_x^{(3)}$ ··· $r_y^{(1)}$ $r_y^{(2)}$ $r_y^{(3)}$ ··· $r_z^{(1)}$ $r_z^{(2)}$ $r_z^{(3)}$ ··· ···

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.

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 thesis, only selected strategies are presented.



(a) Cell Size Factor 1.0 (b) Cell Size Factor 0.5

Figure 2.3: Impact of the cell size factor on the number of distance calculations. The dotted line represents the cutoff radius, superfluous distance calculations are marked by dashed arrows.

FullSearch The default tuning strategy is an exhaustive search over all possible 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. [22]

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 statistics provided by the simulation, different methods of finding trigger points may be optimal. In this chapter we therefore present the 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 outweighed by the additional computations in each iteration. Furthermore, it may 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 runtime measurements of the individual iterations. 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. [19]

The iteration runtimes themselves are again differentiated into multiple parameters: the time spent on computing interactions, traversing remainders and rebuilding neighbor lists. In this work, we will consider the sum of all these times with the exception of the rebuilding measurements. This choice will be justified by collected data in Section 5.2.

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 or in a combination of multiple parameters (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 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. Therefore, to keep results comparable between scenarios, 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`.

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 `rebuild-frequency`, this would lead to the initiation of a new tuning phase in each rebuild iteration, as the rebuild iteration greatly outweighs the non-rebuild iteration. This is one of the reasons why we do not consider rebuild times in the input of our trigger strategies.

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). Figure 3.1 illustrates a comparison between the `TimeBasedSimpleTrigger` and `TimeBasedAverageTrigger`.

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

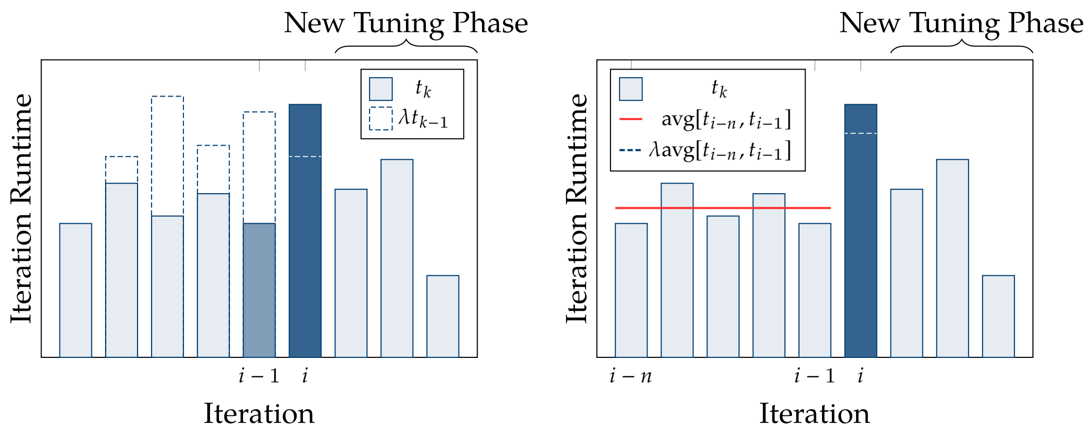


Figure 3.1: Comparison for $\lambda = 1.5$ and $n = 5$ between the `TimeBasedSimpleTrigger` (left) and `TimeBasedAverageTrigger` (right) strategies. A new tuning phase is initiated in both cases, however the `TimeBasedAverageTrigger` is less susceptible to the dip in t_{i-1}.

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. This difference is shown in Figure 3.2.

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.

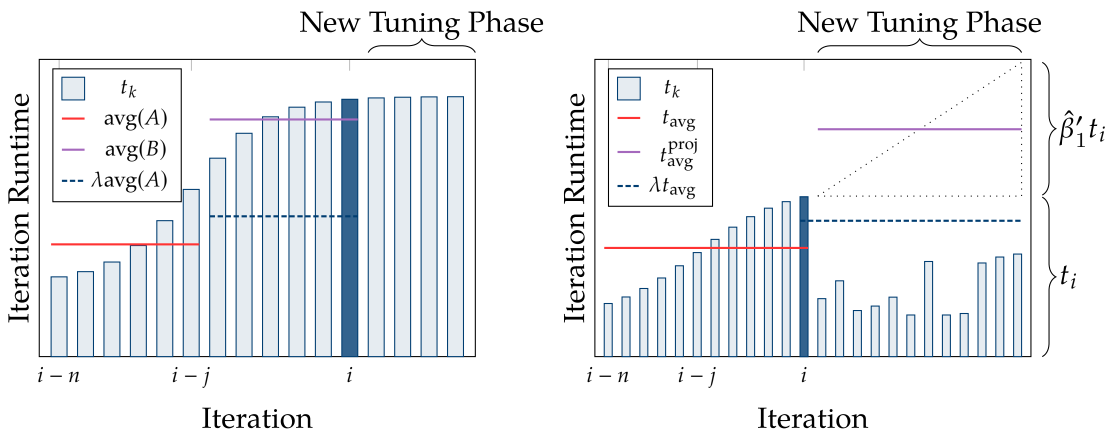


Figure 3.2: Comparison for $\lambda = 1.5$ and $n = 11$ between the `TimeBasedSplitTrigger` (left) and `TimeBasedRegressionTrigger` (right) strategies.

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 obtained.

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. [19], adapted and parametrized for use in this thesis. 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`

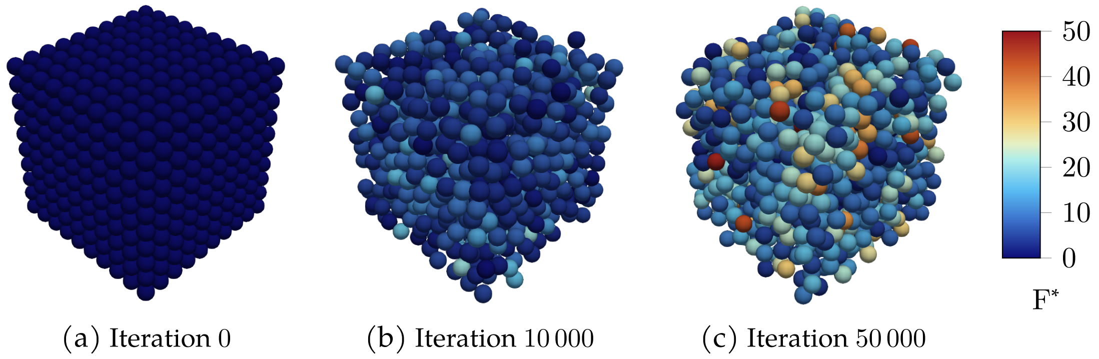


Figure 4.1: Evolution of the simulation state in the equilibrium scenario. The coloring indicates the forces acting upon a particle, and is given in reduced units. Note that the overall forces decrease as the equilibrium is reached, even though the specific timestamps depicted might suggest otherwise.

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.

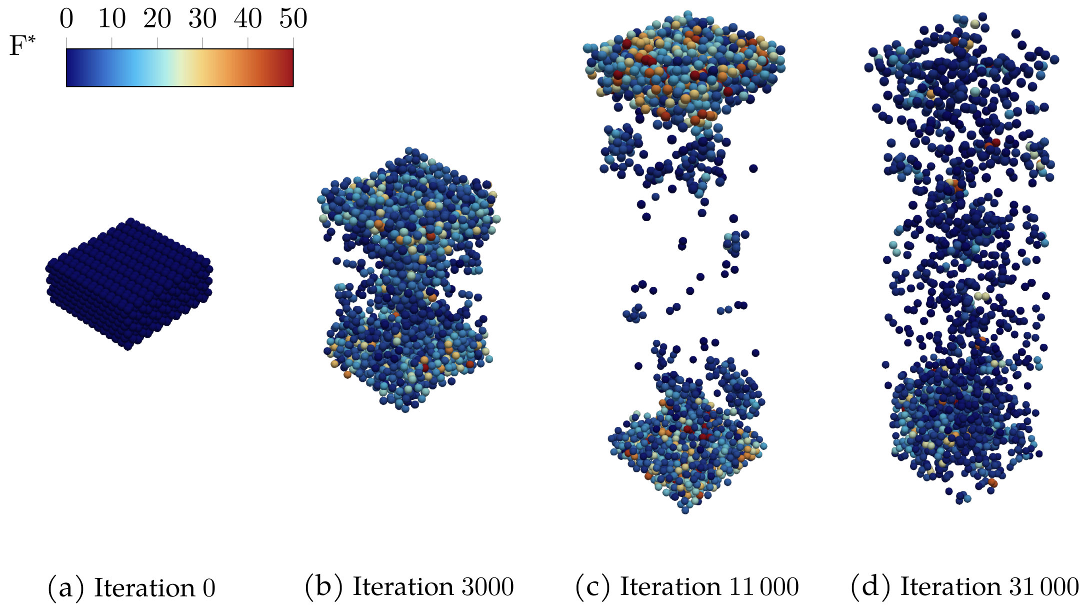
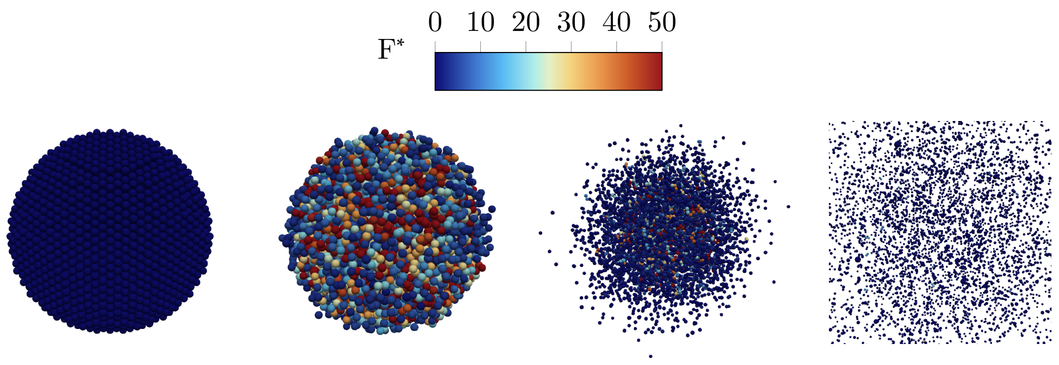


Figure 4.2: Evolution of the simulation state in the exploding-liquid scenario.

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 [18]. 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 using the optimal configuration. As an approximate baseline, we use simulation runs with short, fixed 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 whether 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 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 [15]. 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 `bc47d1ea7e8598afcf58bd35fc531439aa0c7dda`.

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, all trigger strategies are based on the iteration runtimes excluding rebuild times. This choice can be justified by inspecting runtime data we collected: As shown in Figure 5.1, the rebuild times change little over all iterations simulated with a particular configuration. Their inclusion therefore does not provide any new information, but rather smooths out the overall measurements and thus decreases the effectivity at which a scenario change can be detected.

Additionally, rebuild iterations only happen at `rebuild-frequency`. This can lead to stability problems in trigger strategies used with a low number of samples, as the few

[1] `https://www.lrz.de/`

[2] `https://github.com/ladnik/bachelor-thesis`

rebuild iterations greatly outweigh all non-rebuild iterations.

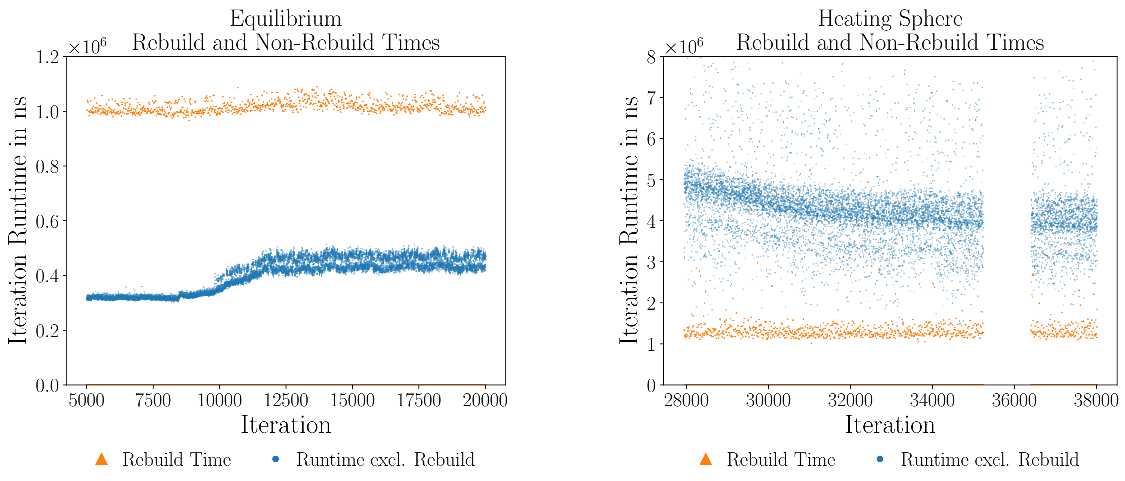


Figure 5.1: Rebuild and non-rebuild times in the equilibrium (left) and heating-sphere (right) scenario. The configurations used were `VLC-C08-N3L-AoS-CSF1` and `LC-C04-NoN3L-AoS-CSF1` respectively. The rebuild times do not contribute any new information regarding scenario change.

5.3 Computational Overhead

The implemented tuning 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, runs without any tuning iterations are compared, such that any changes in runtime that may occur due to different tuning phase initiation points are removed. This is achieved by using a single predefined configuration, such that no tuning takes place.

Figure 5.2 (left) shows the overhead obtained in that manner for the heating-sphere scenario using `LC-C04-N3L-AoS-CSF1` and $n = 500$. To better illustrate the measurements, all values are given as relative and absolute increase in average baseline runtime per iteration. At most, an overhead of 1.9 % per iteration is seen, which can be considered insignificant. Note however, that the overhead is an absolute increase; depending on the scenario, time spent computing interactions varies, therefore the relative values change. It should also be considered, that the absolute difference between the static baseline and the dynamic runs lay in the range of 1 to 7 s over the complete run. Hardware heterogeneity might make up a significant part of this difference — which would directly influence the measured relative and absolute overhead. Nonetheless, the results may give some indication on which strategies are more compute-intensive than others. In particular, usage of the `TimeBasedSimpleTrigger` does incur nearly no runtime penalty (0.084 %), whereas the more complex trigger strategies have a correspondingly larger impact.

To exemplify the importance of optimizing the trigger routines, Figure 5.2 (right) 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. Note that, particularly in the averaging trigger, the speedup experienced is not only due to a lowering of computational overhead, but also due to a lower number of tuning iterations. This can be explained by the feedback of the trigger strategies, as they also influence iteration runtime.

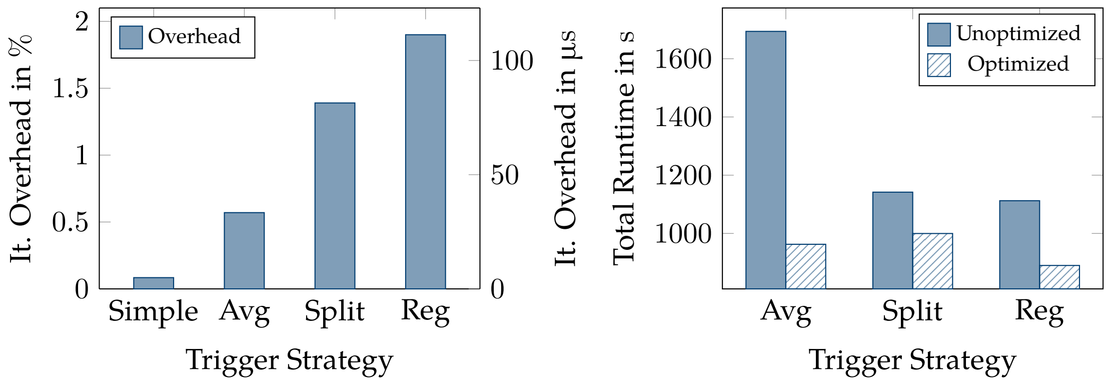


Figure 5.2: Performance comparisons between the various trigger strategies: Relative and absolute iteration overhead in the heating-sphere scenario (left) and average runtime decrease obtained through optimizations in the equilibrium scenario (right).

5.4 Benchmarking Results

The relative speedups presented in the following line charts 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.

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

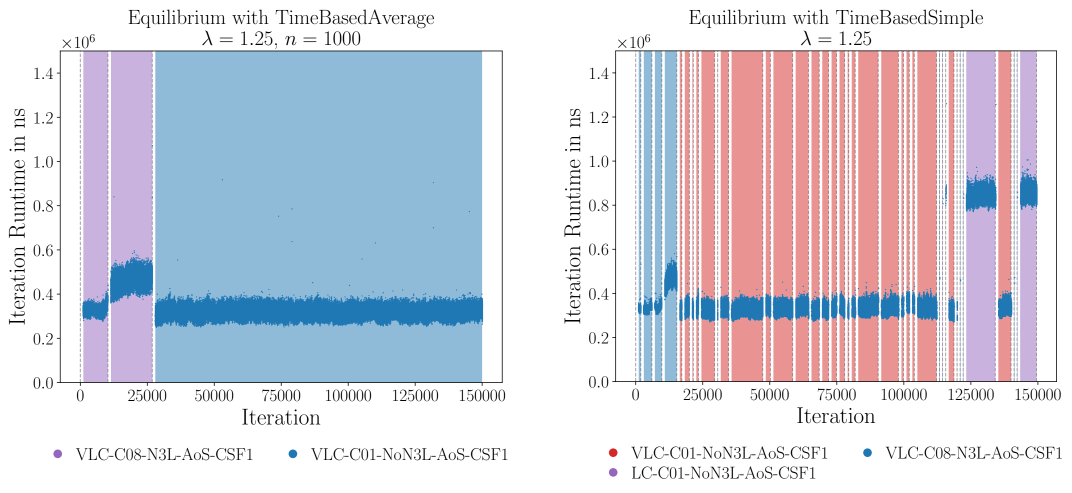
For all plots showing the selected configurations for a given run, the blue scatter dots represent the runtime of that particular iteration. The colored background identifies the used configuration: same configurations map to the same color in a given plot. Gaps along the x-axis occur where tuning iterations 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 gray dashed lines indicate the start of a new tuning phase.

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 second tuning phase, a better configuration is found. As there is no further indication that the simulation state changes, the remaining iterations are performed using the same 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 the one of its predecessor by a factor of λ or more, a tuning phase is initiated. The few outliers in the equilibrium scenario that were averaged out in the previous example, are detrimental to this trigger strategy.



(a) Good scenario change detection. (b) Too many unnecessary tuning phases.

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

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 the majority of trigger strategies. This is however mainly due to the nature of the equilibrium scenario: After the initial relaxation, the optimal configuration is not expected to change. Therefore, not initiating any new tuning phases will lead to a decrease in total simulation runtime, which is why larger values for λ perform better, as they reduce trigger sensitivity. A factor chosen too large however, diminishes this effect again, as the changes in the initial phase are not accounted for. That the speedup is indeed a result of the decreased number of tuning iterations can be verified in the right-hand side plots; for the baseline run with static tuning intervals, 22.6 % of iterations were spent in tuning phases. Conversely, for strategies initiating too many new tuning intervals, i.e., with more tuning iterations than the baseline run, the simulation runtime increases.

Additionally, triggers with a larger sample size will typically trigger less frequently, as more of the variability in iteration runtime is smoothed out. However, for a too large number of samples such as $n > 1000$, the speedup may decrease, as the computational overhead is directly proportional to the number of samples. This is particularly noteworthy for scenarios with fewer particles, where the computation of interactions takes shorter time. Therefore, the relative overhead of our strategies is larger, which would explain the lower speedup seen in the regression triggers: Compared to the `TimeBasedAverageTrigger`, at roughly the same number of tuning iterations, the speedup achieved is significantly lower.

Interestingly, the `TimeBasedSplitTrigger` with $n = 250$ triggered significantly more tuning iterations for $\lambda = 1.75$ than for $\lambda = 1.5$. The rather small sample size increases the sensitivity towards noise in the input data — which is why outside influences by the underlying hardware could explain the difference between the two runs. This holds particularly if one considers that our equilibrium scenario consists of few particles with which interactions must be computed, resulting in low iteration runtimes.

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

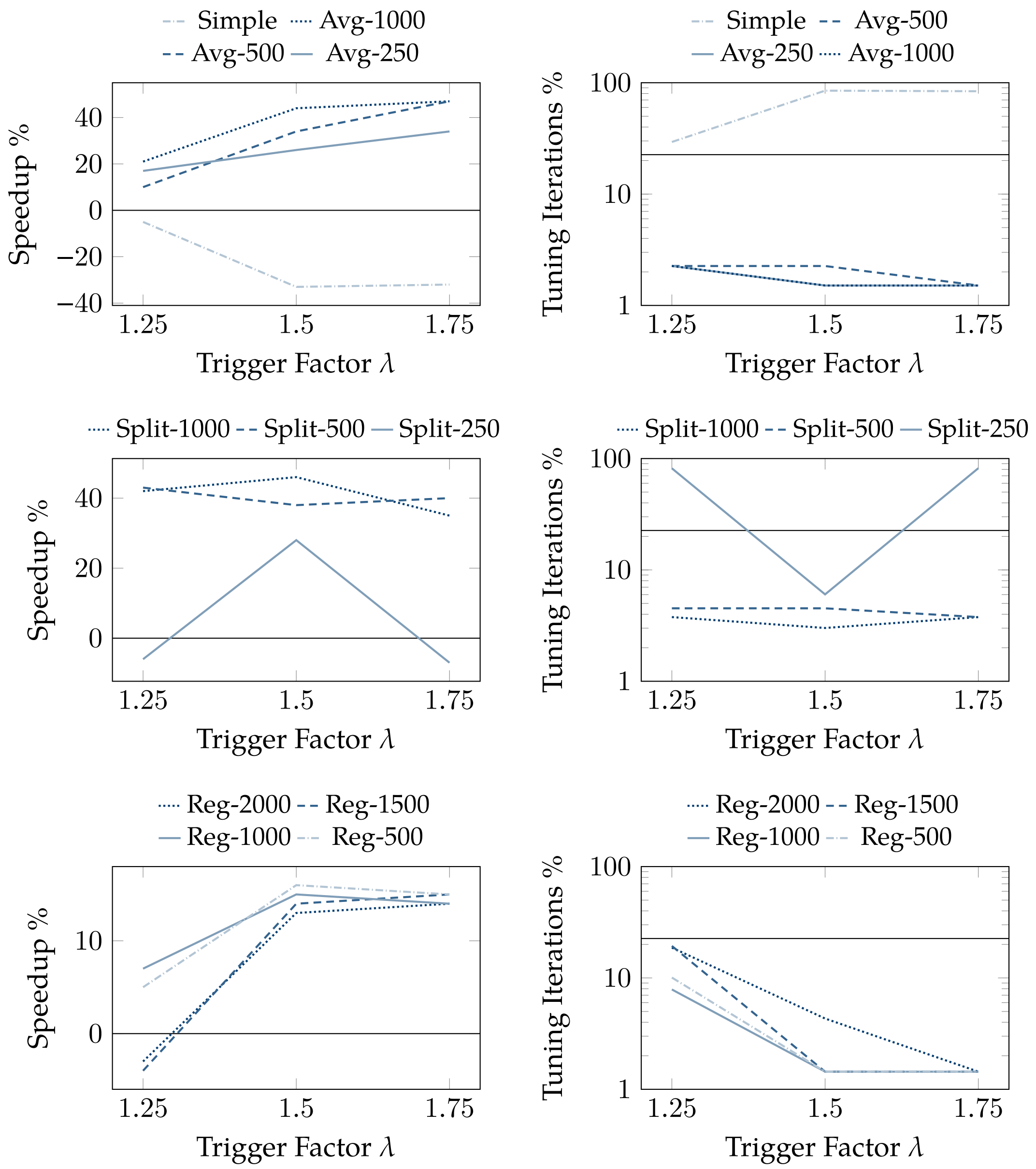


Figure 5.4: Trigger behavior in the equilibrium scenario, the numbers in the legends refer to the number of samples n considered. The line in the background represents the baseline run. Note the logarithmic scale in the plots on the right hand side.

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 non-tuning iteration to be one of the best choices. As can be seen in Figure 5.5, this is achieved across all strategies except the simple trigger, with all configurations being one of the top three choices for that specific iteration. It should be noted, that the ranking of optimal configurations is only an approximation, as it is based on the baseline

Trigger	Trigger factor λ	Number of samples n
`TimeBasedSimple`	not recommended	–
`TimeBasedAverage`	1.75	500
`TimeBasedSplit`	1.5	1000
`TimeBasedRegression`	1.5	500

Table 5.1: Suggested default parameters for the equilibrium scenario.

run and therefore restricted to the resolution of that run's `tuning-interval`. The best performing strategy regarding optimality appears to be the `TimeBasedAverageTrigger`, for which 92 % of all non-tuning iterations were computed using the same configuration as in the baseline run. Except for the `TimeBasedSimpleTrigger`, all strategies computed the full simulation length with configurations among the top 3 choices of the static run.

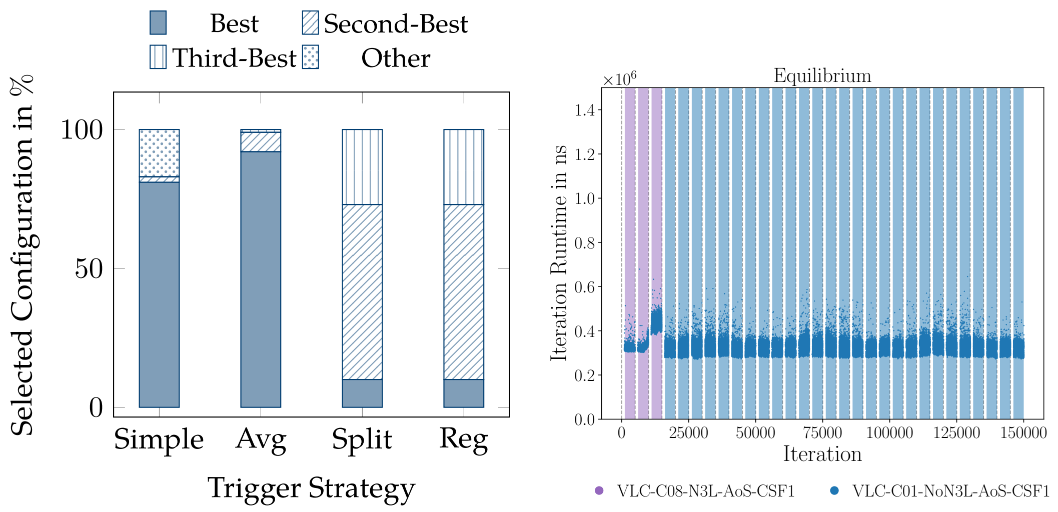


Figure 5.5: Ranking of configurations selected by the best run in the equilibrium scenario for each trigger strategy (left) and selected configurations in the baseline run (right).

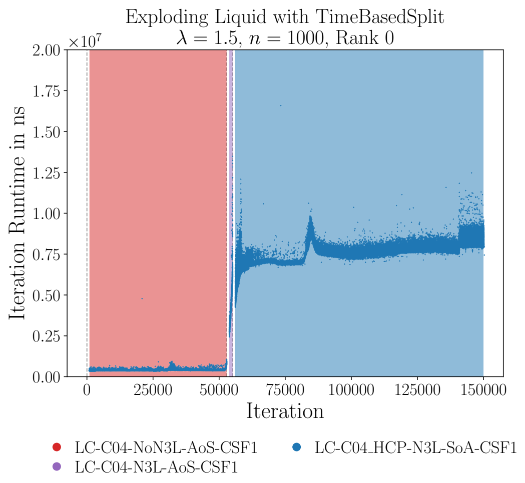
5.4.2 Exploding Liquid

Selected Runs

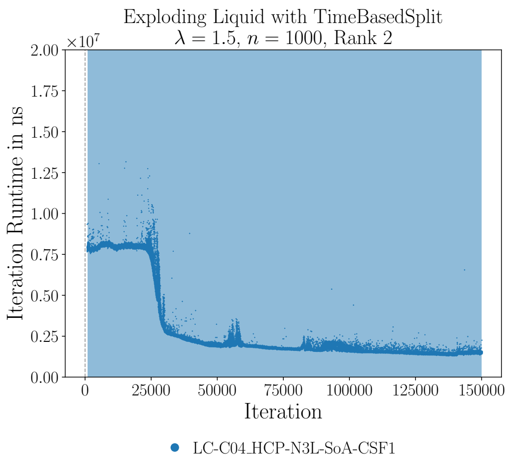
The exploding-liquid scenario was executed on 6 MPI processes, with the simulation domain subdivided along the y-axis. As a result, different ranks encounter the "particle wave" at different points in time, as it spreads outward from the center (cf. Section 4.1.2). This effect can be seen in Figure 5.6, where the plot showing rank 2 (in the center of the domain, plot on the right hand side) experiences high iteration runtimes at the beginning of the simulation. Rank 0 (edge of the domain, plot on the left hand side), however, shows this influx of particles not until iteration 50 000.

The figure on the left shows an optimal response to the peak in iteration runtime by the `TimeBasedSplitTrigger`. The initial configuration remains optimal until the particles enter this part of the domain, after which two tuning phases are triggered. The first trigger at iteration 52 814 is not needed, as due to the rapid increase in iteration runtime, a second one at iteration 55 088 is tripped. This additional tuning phase could be prevented by using a bigger factor λ, the effect on total simulation runtime would however be limited. The second half of the simulation again has one optimal configuration, which is well reflected in the trigger's behavior.

The second plot on the right does not necessarily illustrate bad trigger behavior, but rather a limitation of our approach. After the initial expansion, the subdomain of rank 2 significantly transforms at iteration 25 000, with a different configuration possibly being better suited. However, our triggers only consider increases in input data. A more appropriate method in such cases would be to consider changes in magnitude (cf. Section 3.1.3), i.e., any deviation from the current average.



(a) Good scenario change detection.



(b) No reconfiguration after peak.

Figure 5.6: Examples of trigger behavior in the exploding-liquid scenario.

Speedup and Default Parameters

Although the exploding-liquid scenario is more complex than the equilibrium scenario, there is still a performance gain in all trigger strategies: The highest speedup is reached by the `TimeBasedAverageTrigger`, with a reduction in total simulation time of 20 %. This is less than was reached in the equilibrium scenario, but still significant. In general, the results look very similar to those of the previous scenario, with some notable exceptions. For example, the `TimeBasedSimpleTrigger` reaches positive speedup for larger values of λ. Considering that in most ranks, the configuration change happens at the initial inflow of particles into the subdomain, as simple trigger is sufficient. With $\lambda = 1.25$ however, the trigger appears to be too sensitive, which again leads to an excessive number of tuning phases.

The `TimeBasedSplitTrigger` displays opposite behavior; for larger values of λ, the speedup is greater, as triggering too few tuning phases leads to a large number of iterations being computed using a suboptimal configuration. Similarly, triggers with a larger sample size tend to smooth out the changes in iteration runtime and thus have a delayed reaction to the rapid transformation of the domain. The `TimeBasedRegressionTrigger` again leads to lower speedups than averaging or split triggers at optimal configuration. However, the difference is minor and can thus be attributed to the higher overhead of this strategy.

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

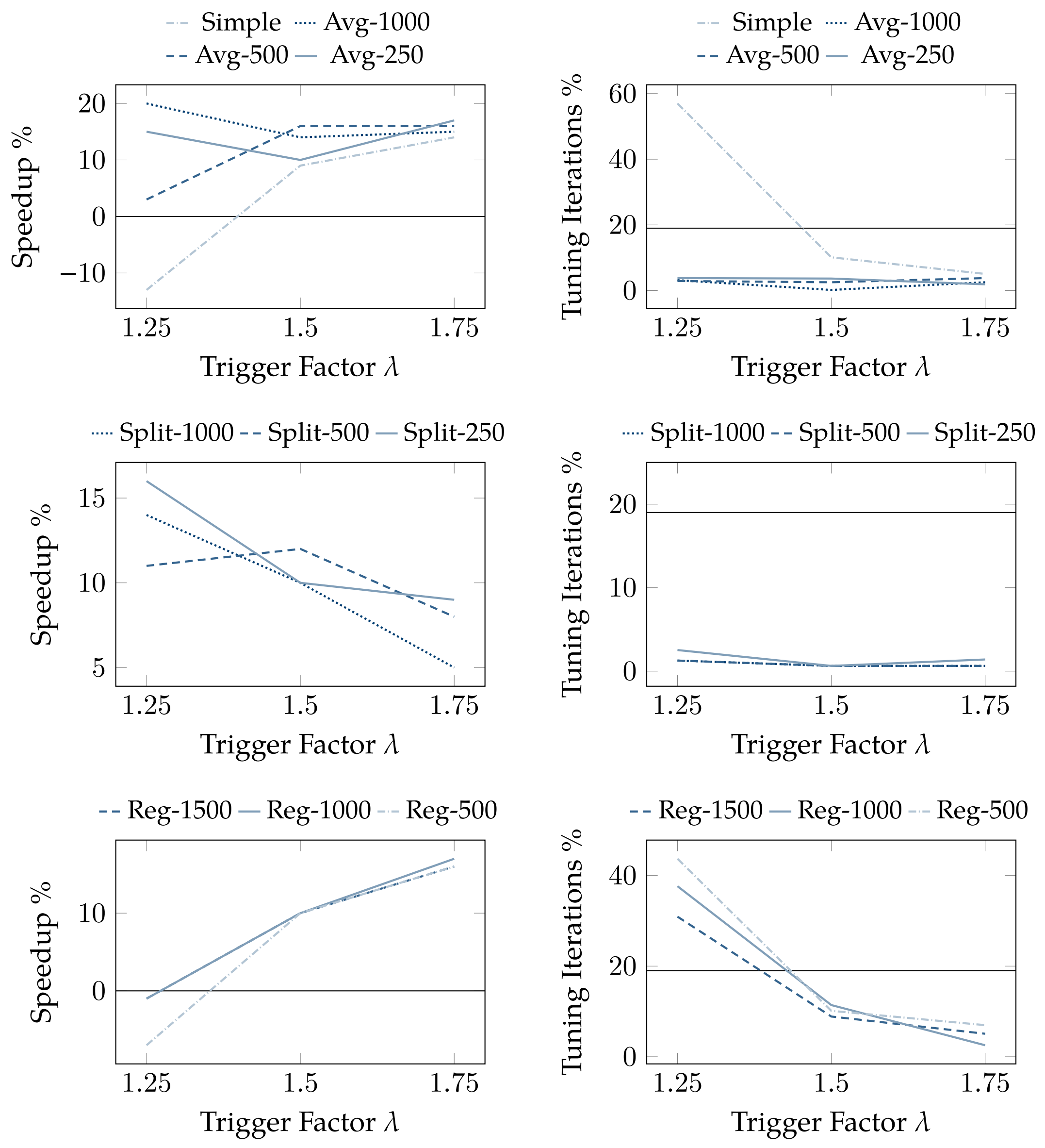


Figure 5.7: Trigger behavior in the exploding-liquid scenario, the numbers in the legends refer to the number of samples n considered. The line in the background represents the baseline run.

Trigger	Trigger factor λ	Number of samples n
`TimeBasedSimple`	1.75	–
`TimeBasedAverage`	1.25	1000
`TimeBasedSplit`	1.25	250
`TimeBasedRegression`	1.75	1000

Table 5.2: Suggested default parameters for the exploding-liquid scenario.

Optimality

The configuration fit is best for the `TimeBasedSimpleTrigger`, `TimeBasedAverageTrigger` and `TimeBasedRegressionTrigger`: As shown in Figure 5.8, these strategies select, on average, the best configuration for 95 % of all non-tuning iterations. The strategy that appears to have the worst fit is the `TimeBasedSplitTrigger` — note, however, that it used the fourth best configuration in 81 % of iterations.

It should also be mentioned, that these statistics are based on rank 0 across all runs. Therefore, the problem of missing reconfiguration after a drop in iteration runtime, as discussed previously, is not reflected in the results.

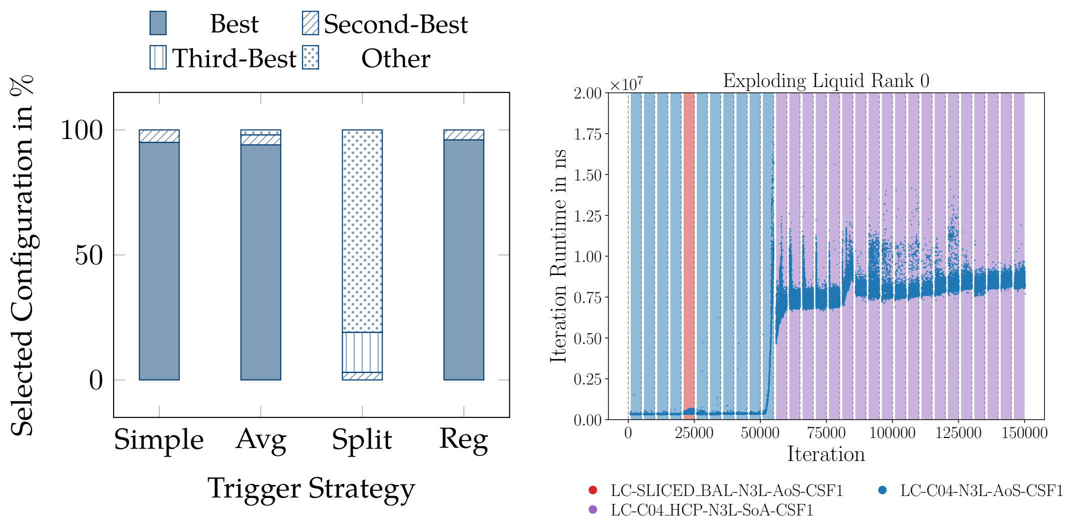


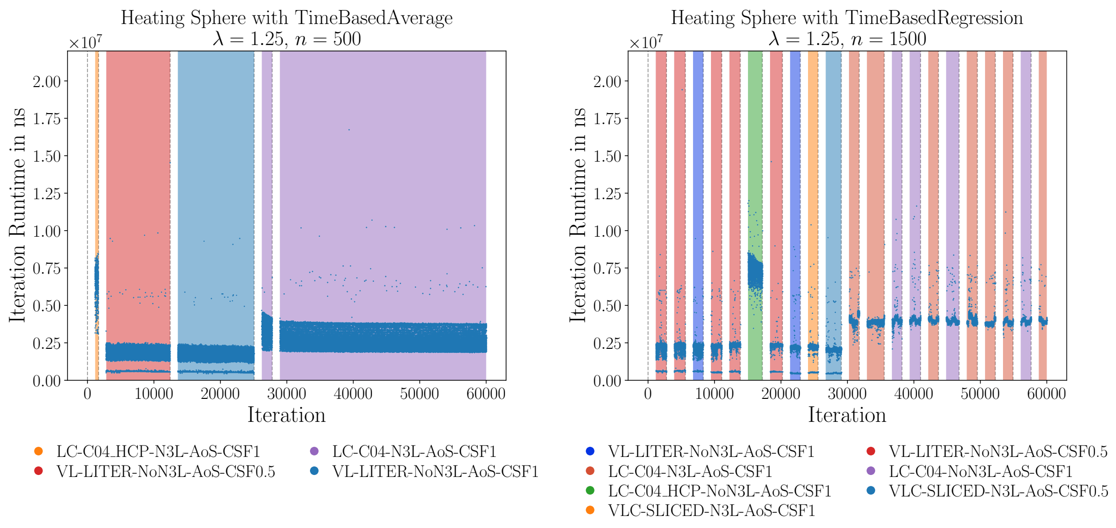
Figure 5.8: Ranking of configurations selected by the best run in the exploding-liquid scenario for each trigger strategy (left) and selected configurations in the baseline run (right).

5.4.3 Heating Sphere

Selected Runs

Figure 5.9 again displays two sample runs to illustrate optimal and unsatisfactory results. The left figure shows a reduction in the number of tuning phases initiated by the `TimeBasedAverageTrigger`. After the last tuning phase, the configuration does not change from the previous one — that tuning phase was therefore unnecessary. Compared to the equilibrium scenario, the iteration runtimes form a broad band, which indicates a high variance. The absolute ranges of these variations lie in the range of 1×10^6 to 5×10^6 ns, in contrast to a spread of 1×10^5 ns in the equilibrium scenario. Additionally, more outliers are seen, which might worsen the performance of the strategies susceptible to oscillations; one of them is the `TimeBasedAverageTrigger` depicted. Without any clear indication of an increase in runtime, new tuning phases are initiated, since a single outlier larger than the last n samples by a factor of λ can trigger the strategy. Interestingly, most outliers do not have a significant impact due to the averaging approach. As shown in the plot, the use of the averaging trigger results in fewer tuning phases than in the baseline run, which in turn explains the speedup measured.

Worse results can be seen in the `TimeBasedRegressionTrigger`, pictured on the right hand side. After multiple tuning phases, the same configuration is selected, which again indicates unnecessary triggering. The regression approach is suboptimal in the heating-sphere scenario, due to the high variance in iteration runtimes described before: Outliers can skew the slope derived by the least-squares method, which leads to a incorrect prediction of the runtime in the next interval, triggering our strategy. Hence, a slowdown compared to the baseline run is observed. 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 approach could be replaced with a more robust method like the Theil-Sen estimator [28].



(a) Reduced number of tuning phases. (b) Too many unnecessary tuning phases.

Figure 5.9: Examples of trigger behavior in the heating-sphere scenario.

Speedup and Default Parameters

The high variance in iteration runtimes directly influences the behavior of our trigger strategies, as shown in Figure 5.10. The strategies which should be unstable due to these variations are the `TimeBasedAverageTrigger`, `TimeBasedSimpleTrigger` and `TimeBasedRegressionTrigger`. However, results only show poor performance of the latter two. The simple trigger under-performs the static approach by up to $-41\,\%$ with a highly unfavorable $99\,\%$ of all iterations being tuning iterations. Similarly, the regression approach leads to runtime increases across almost all tested combinations of (λ, n).

The `TimeBasedAverageTrigger`, however, performs exceptionally well, with speedups of up to $40\,\%$, proportional to the decreased number of tuning iterations. Rather counterintuitively, this outcome could be accounted for precisely by the prevalence of outliers: If there occur enough outliers within the trigger's sample interval, they raise the average runtime, which in turn reduces the impact of any current outlier.

In summary, it can be said that our approach is not suitable for the heating-sphere scenario. Considering that there is no clear indication of scenario change in runtime (cf. Section 5.5), this was expected.

The collected data suggests default parameters as presented in Table 5.3.

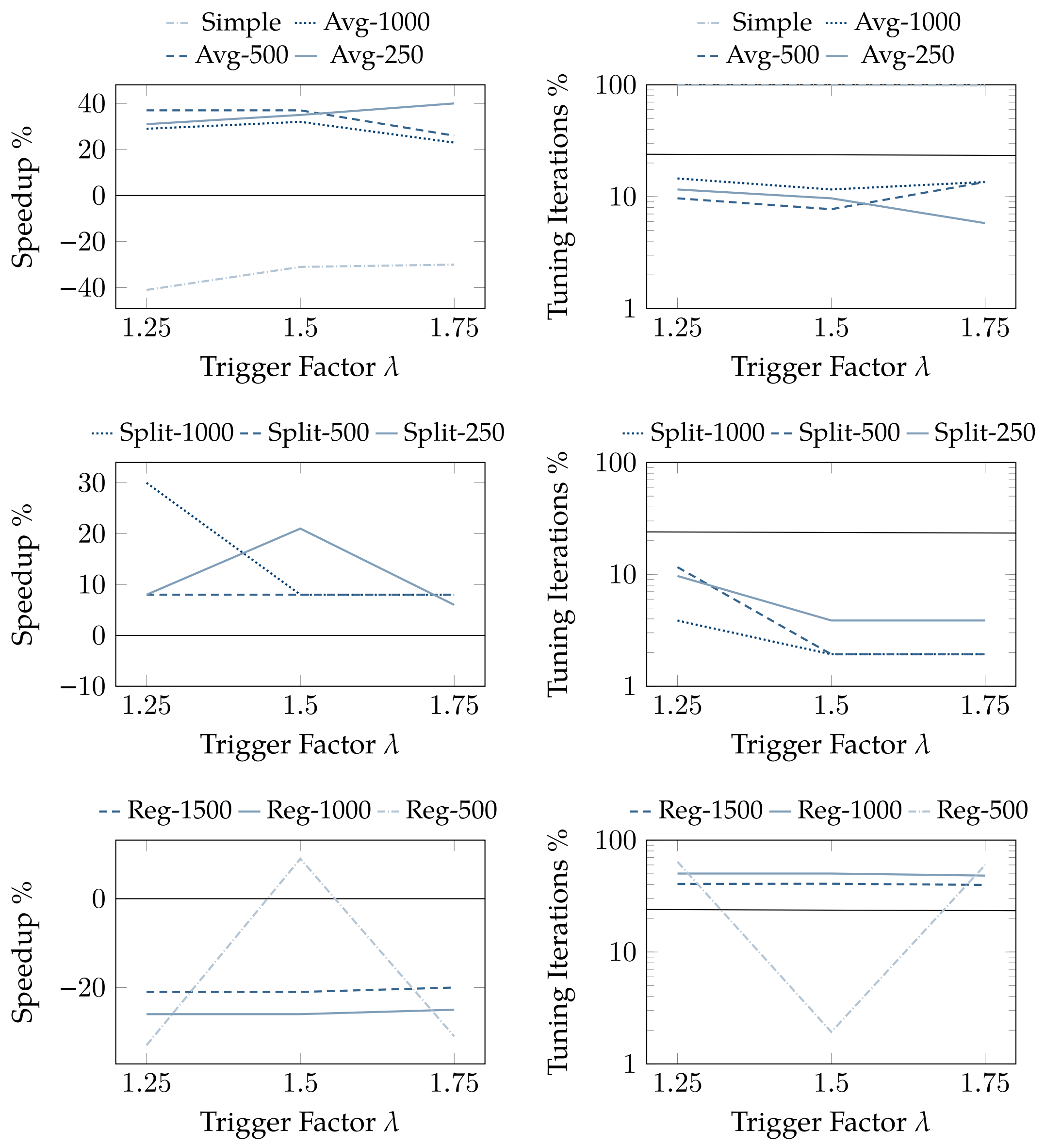


Figure 5.10: Trigger behavior in the heating-sphere scenario, the numbers in the legends refer to the number of samples n considered. The line in the background represents the baseline run. Note the logarithmic scale in the plots on the right hand side.

Trigger	**Trigger factor λ**	**Number of samples n**
`TimeBasedSimple`	not recommended	–
`TimeBasedAverage`	1.5	500
`TimeBasedSplit`	1.5	250
`TimeBasedRegression`	not recommended	–

Table 5.3: Suggested default parameters for the heating-sphere scenario.

Optimality

Figure 5.11 shows the configuration selected by the best performing run for each trigger. The values again are given for all non-tuning iterations, compared to the configurations selected in the baseline run. Note that, e.g., the simple trigger displays better configuration fit than the averaging trigger; this is primarily due to tuning iterations being ignored. Additionally, not triggering a new tuning phase has a higher runtime impact than computing iterations with suboptimal configuration fit.

It can be seen that, particularly for the strategies triggering fewer tuning phases, i.e., the `TimeBasedAverageTrigger` and `TimeBasedSplitTrigger`, the configuration fit is suboptimal. For both, on average 25 % of all non-tuning iterations were computed using a configuration that was not in the top 3 choices of the baseline run.

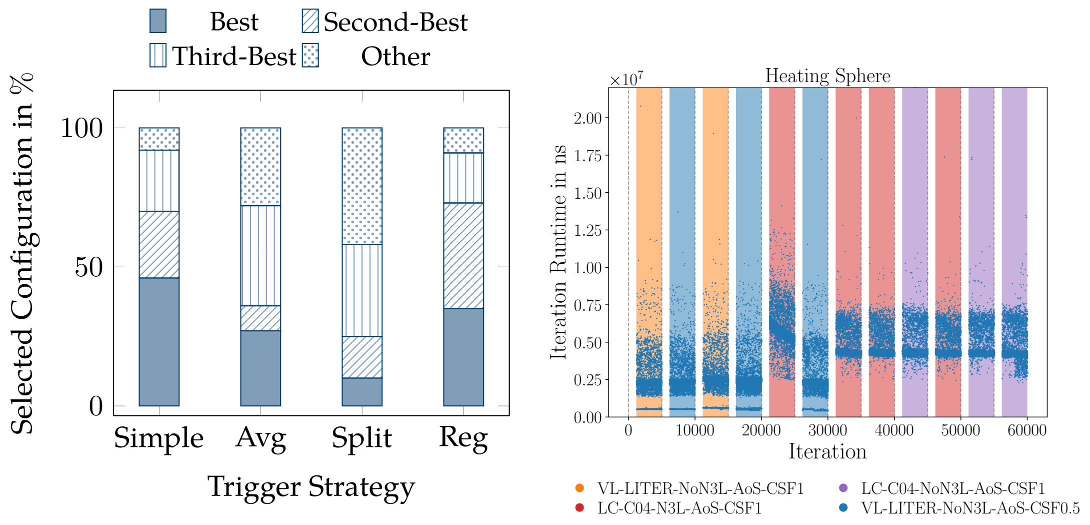


Figure 5.11: Ranking of configurations selected by the best run in the heating-sphere scenario for each trigger strategy (left) and selected configurations in the baseline run (right).

5.5 Hybrid Triggers

Time-based approaches may not be suitable for scenarios in which iteration runtime alone is not good enough indicator for scenario change. This was seen, e.g., in the heating-sphere scenario. Since 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 5.12 shows an exemplary run for the heating-sphere scenario. Using static containers, the initial optimal configuration is `VL-List_Iter-NoN3L-AoS`, and changes to `LC-C04-N3L-AoS-CSF1` later on [19]. Although a better configuration is available, the iteration runtimes do not change. However, the `maxDensity` statistic indicates the shift towards a different simulation state; considering that the particles are packed closely in the initial phase and expand outwards, this decrease in particle density is consistent with expectations. The `maxDensity` statistic would therefore be a reasonable trigger input.

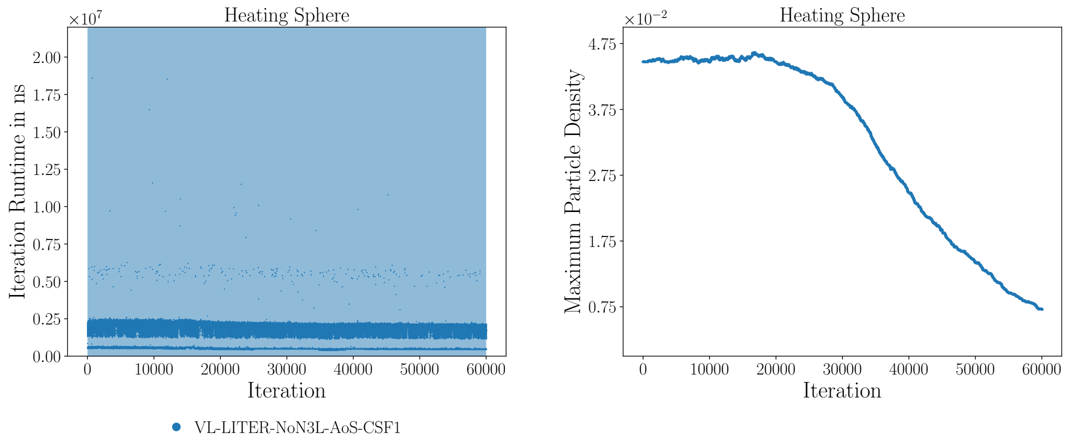


Figure 5.12: Iteration runtime (left) and the maximum particle density (right) for the heating-sphere scenario with single configuration `VL-List_Iter-NoN3L-AoS`. The iteration runtime does not indicate scenario change, but the `maxDensity` statistic shows the transformation of the simulation state.

6 Conclusion

In this thesis, four novel methods for the dynamic initiation of 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 can decrease simulation runtime across most typical scenarios when using `full-search` as the tuning strategy. Especially in settings with low variance in iteration runtime, all strategies except the naive approach reduce the number of tuning phases without any significant decrease in optimality of the selected configurations. The most promising candidates were shown to be the strategies based on sample averages, due to their resilience to noise in the live input data. These strategies lead to speedups of up to $47\,\%$ under optimal conditions. The regression strategy investigated was more susceptible to the aforementioned fluctuations in the input data, but nonetheless outperformed static tuning.

As new tuning strategies are introduced, however, the dynamic initiation of tuning intervals will likely become less relevant for single-process applications. In particular, tuning strategies based on machine learning lead to cheap tuning phases [19], which in turn significantly diminishes the achievable speedups. One application in which a dynamic approach as ours might still be of use, is in MPI-parallel setups: As each rank runs its own AutoPas instance, it can trigger tuning phases independently from other ranks. In scenarios with heterogeneous particle distribution over the whole domain, the best configuration for a specific rank 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 new 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-process applications, but some benefits remain in MPI-parallel settings.

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. 2nd printing of the 2nd ed. 1993". Trans. by A. Iacob. 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] Tobias Brink. "Heterogeneities in Metallic Glasses: Atomistic Computer Simulations on the Structure and Mechanical Properties of Copper-Zirconium Alloys and Composites". PhD thesis. Darmstadt: Technische Universität Darmstadt, 2017.

[6] 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`.

[7] 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`.

[8] 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`.

[9] 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`.

[10] 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`.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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].

[20] 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.

[21] 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.

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

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

[24] Sergey Vassiliev. *Running Molecular Dynamics on Alliance clusters with AMBER*. https://computecanada.github.io/molmodsim-amber-md-lesson/figures/. Workshop material developed and delivered by Compute Canada. Accessed: 2025-09-08. 2025.

[25] 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.

[26] 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.

[27] 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.

[28] 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.