FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Implementing a predictive tuning strategy in AutoPas using extrapolation

Julian Mark Pelloth

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Implementing a predictive tuning strategy in AutoPas using extrapolation

Implementierung einer Predictive-Tuning Strategie in AutoPas unter der Verwendung von Extrapolation

Author:	Julian Mark Pelloth
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Fabio Alexander Gratl, M.Sc. and Steffen Seckler, M.Sc.
Date:	15.09.2020

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

Munich, 15.09.2020 Julian Mark Pelloth

Abstract

The AutoPas library is capable of simulating computationally intensive molecular dynamic simulations. To reduce the run time, it provides many configurations to select the fastest for the specific scenario. AutoPas tries to achieve this task by testing the hole search space frequently. In this thesis, a new approach is implemented that uses extrapolation to predict the efficiency and only tests the best predictions to avoid inefficient configurations. The predicting strategy can reduce the run time by up to 74%.

Contents

Part I.

Introduction and Background

1. Introduction

Molecular Dynamics is used to simulate liquids, solids, gases, and molecules' properties. This makes it possible to execute experiments that can only be realized with a simulation[HEDM15]. The simulations initialize the N atoms or molecules as particles with a mass, velocity, and force, and the behavior is computed with Newton's equations. The pairwise force calculation for every particle has a time complexity of $\mathcal{O}(N^2)$, which makes it expensive to compute and needs optimization.[Pli95]
AutoPas is a library for arbitrary N-body simulations, which provides different configurations to optimize the force calculation. In order to choose the most efficient configuration, it provides a functionality called *auto-tuning*. This function frequently tries to map the best configuration to the current scenario during the simulation by testing all configurations. Additionally, AutoPas provides different selection strategies to minimize the number of configurations that have to be tested.
This thesis tries to optimize the auto-tuning process by implementing a new selection strategy that uses extrapolation to predict the efficiency of all configurations. The predictions are used to select and test only efficient configurations. The new strategy avoids testing configurations with long run time. This approach aims to reduce the time spent in the tuning-process and the simulation's total run time. There are three extrapolation methods, linear regression [Subsection 2.1.1], Lagrange polynomial [Subsection 2.1.2], and Newton polynomial [Subsection 2.1.3] implemented to cover different cases and to be able to analyze them to determine the best method.

2. Theoretical Background

2.1. Extrapolation Methods

For this thesis, three different extrapolation methods have been implemented to predict the evidence of a configuration. The user can decide through the simulation's input, which method is used by the tuning strategy.

2.1.1. Linear Regression

The first method is the linear regression[Fro18], which puts a line through all input points. For this we need to calculate the gradient $\hat{b}$ and the y-intercept $\hat{a}$ with $\bar{x}$ and $\bar{y}$ being the arithmetical mean.

$$\hat{b} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} \tag{2.1}$$

$$\hat{a} = \bar{y} - \hat{b}\bar{x} \tag{2.2}$$

These two values, shown in Equation 2.1 and Equation 2.2, are used to calculate the regression line.

$$\hat{y} = \hat{a} + \hat{b}x \tag{2.3}$$

Equation 2.3 is then used to extrapolate the evidence for a configuration.
The linear regression is a fast and straightforward method to calculate a prediction. Additionally, it provides the possibility to store the calculated function for a second extrapolation.

2.1.2. Lagrange polynomial

The second method makes use of the Lagrange interpolation[HS02], where input values pairs $(x_0, y_0), \ldots, (x_n, y_n)$ are used to calculate so called Lagrange polynomials

$$\begin{aligned} L_j(x) &= \prod_{i=0, i\neq j}^{n} \frac{x - x_i}{x_j - x_i} \\ &= \frac{(x - x_0)\cdots(x - x_{j-1})(x - x_{j+1})\cdots(x - x_n)}{(x_j - x_0)\cdots(x_j - x_{j-1})(x_j - x_{j+1})\cdots(x_j - x_n)} \end{aligned} \tag{2.4}$$

for $j = 0, 1, \ldots, n$. Thereafter a single polynomial with the degree n can be calculated out of the Lagrange polynomials:

$$p(x) = \sum_{j=0}^{n} y_j L_j(x) \tag{2.5}$$

The method's intended purpose is to use the function to interpolate within the interval of the input values with the polynomial of Equation 2.5. In this thesis, it is used to extrapolate them and calculate a prediction for a configuration.
This method provides the possibility to calculate a polynomial extrapolation function if a linear function can not accurately represent the input points. It can calculate a linear polynomial. However, the linear regression is expected to perform better for linear functions. Therefore, the Lagrange polynomial should only be used to extrapolate a non-linear function. A disadvantage is that the polynomial L_j, as seen in Equation 2.4, is calculated for a specific x value. Therefore, the Lagrange polynomial does not produce a function that can be stored and reused.

2.1.3. Newton polynomial

A similar approach is taken with the third method, the Newton interpolation[FH07][HS02], where a polynomial function of the degree n is defined as

$$\begin{aligned} p(x) =& f[x_0] + (x - x_0)f[x_0, x_1] + (x - x_0)(x - x_1)f[x_0, x_1, x_2] + \\ & \ldots + (x - x_0) \cdots (x - x_{n-1})f[x_0, \ldots, x_n] \end{aligned} \tag{2.6}$$

with the coefficients $f[x_i, \ldots, x_{i+k}]$ that can be calculated as follows:

$$f[x_0] = f(x_0) \tag{2.7}$$

$$f[x_0, x_1] = \frac{f[x_1] - f[x_0]}{x_1 - x_0} \tag{2.8}$$

$$f[x_0, \ldots, x_k] = \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0} \tag{2.9}$$

$$f[x_i, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i} \tag{2.10}$$

This polynomial, calculated with Equation 2.6, is normally used for interpolation. As it is done in the second method, it is used for extrapolating the base pairs to calculate a prediction.
The Newton polynomial has no clear advantage over the Lagrange polynomial as both can potentially have trouble with rounding errors. This method was chosen to have a second extrapolation method for polynomials with a degree higher than one to compare them in efficiency and accuracy. An advantage of the Newton polynomial is that the calculated polynomial can be stored and reused in the next tuning-phase.

2.1.4. Runge's Phenomenon

Carl Runge discovered that the interpolation of the function $\frac{1}{1+x^2}$, in the same interval with a different amount of base points, leads to different accuracy of the interpolation on the edges of the interval. The phenomenon describes that the more base points are used for the interpolation, the more the result for the edges diverges from the function's real value. The cause is that the interpolation oscillates towards the endings of the interval. Therefore having an interpolation function with a higher degree can decrease the accuracy, and fewer base points can lead to a result with a lower interpolation error on the edge of

the interval.[Run01]
The phenomenon describes how it should be beneficial for the accuracy of edge cases of the interpolation to use fewer base points. In this thesis, interpolation methods are used to extrapolate the base points beyond the given interval. The phenomenon could have a significant impact on the extrapolated value. It is crucial to consider it and test how the accuracy is impacted when more input for the extrapolation is used.

2.2. Molecular Dynamics

Molecular dynamic (short: MD) simulations provide the possibility of modeling molecular structures in great detail. This provides a numerical solution for the N-body problem. MD is used to simulate scenarios like fluid dynamics or biomolecules. There are different categories of MD problems. The focus of AutoPas is on short-range MD simulations, where the interactions occur between pairs of particles that are in close range. The function to calculate the pairwise potential is the Lennard-Jones Potential. It is defined by the relative range between two particles r_{ij}, parameter ϵ, governing the strength of the interaction and σ, defining a length scale.[Rap04]

$$U(r_{ij}) = 4\epsilon \left(\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right) \tag{2.11}$$

Applying Equation 2.11 on every particle pair results in the force for each of them. The Lennard-Jones Potential is modified with a *cutoff radius* r_c because it converges to zero when the distance between to particles increases and, therefore, is insignificantly small. This can be seen in Figure 2.1, where the Lennard-Jones Potential is shown for $\epsilon = \sigma = 1$. The cutoff radius reduces the number of potential force calculations because it is only calculated when the distance between two particles is smaller or equal to the cutoff.

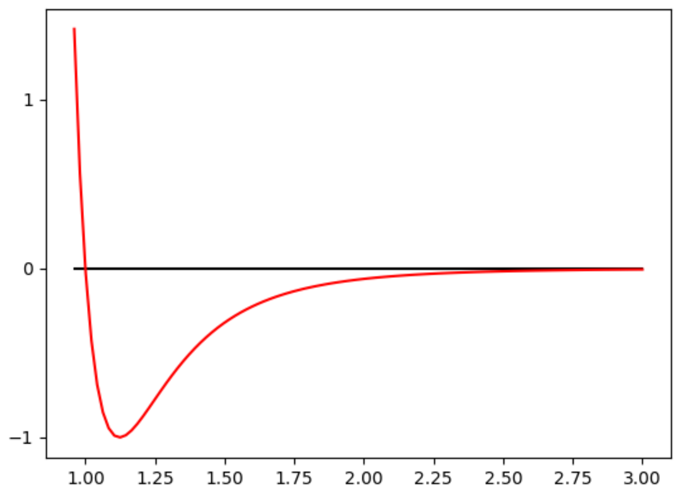


Figure 2.1.: The Lennard-Jones Potential for $\epsilon = \sigma = 1$

3. Introduction to AutoPas

AutoPas[1] is an open-source C++ library to provide a base and optimize the node-level performance of arbitrary N-body simulations. It acts as a black-box for the user that provides different algorithms to apply short-range force calculations to particles in a domain. To optimize the run time, AutoPas uses auto-tuning to select the optimal algorithm. Therefore the user does not need to have a deeper understanding of the library to use it.[GST$^+$19]

3.1. Configurations

A configuration combines a particle container [Subsection 3.1.1], a traversal [Subsection 3.1.2], a data layout [Subsection 3.1.3], and the optimization with Newton's third law of motion (Further named as: Newton3) [Subsection 3.1.4]. This determines how the particles are stored, how they are traversed, and how the interactions between them are calculated. The different parts all have advantages and disadvantages, which makes them better for different scenarios. AutoPas provides many configurations to ensure it chooses the best strategy for the current state of the simulation.[GST$^+$19]

3.1.1. Particle Container

The particle containers are the algorithms for the pairwise force calculation. They identify the particles within a prescribed neighborhood, such as finding all particle pairs with a distance less than a prescribed cutoff radius. There are currently seven different containers implemented, all based on the following four algorithms: DirectSum, LinkedCells, VerletLists, and VerletClusterLists.[GST$^+$19]

DirectSum

The DirectSum is the intuitive way of calculating each particle's force because it calculates every particle's pairwise interaction with each other regardless of the distance. After this, it only considers the particles in the cutoff radius, as seen in Figure 3.1a. This algorithm has a time complexity of $\mathcal{O}(N^2)$. DirectSum causes no memory overhead since no complicated data structure needs to be stored.[GST$^+$19]

LinkedCells

To reduce the number of distance calculations, LinkedCells divides the simulation domain into cells. The width of these cells is bigger or equal to the cutoff radius. On account of this, only the distance between the particle and the particles within the particular and neighboring cells needs to be calculated. Those within the cutoff radius are used to calculate the force. As Figure 3.1b shows, only the red and blue cells need to be evaluated to calculate the force acting on the red particle. The result of storing

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

the particles in cells is that they can continuously be placed into the memory, thus providing a good vectorization opportunity.

$$\frac{\text{Cutoff volume}}{\text{Search volume}_{LC}} = \frac{\frac{4}{3}\pi r_c^3}{(3r_c)^3} \approx 0.155 \tag{3.1}$$

LinkedCells still has flaws. The chance of a particle being within the cutoff radius is less than 16%, which means that 84% of distance calculations are unnecessary.[GST+19]

VerletLists

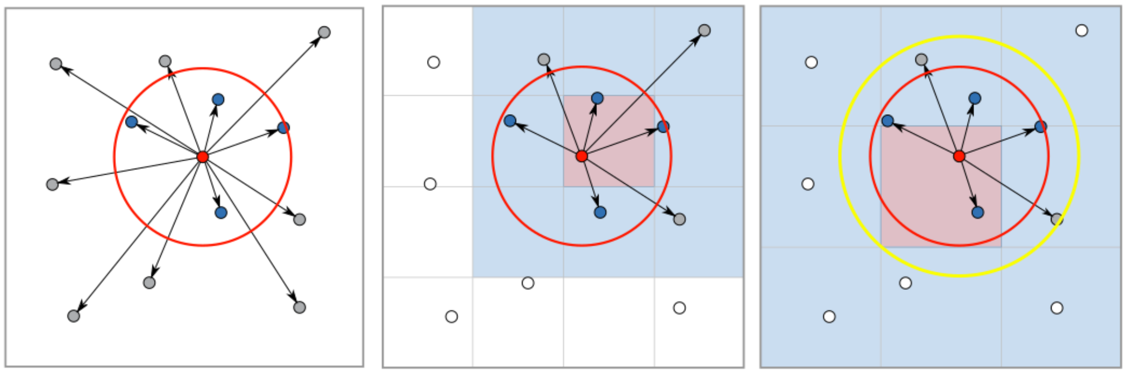
Instead of creating cells, all distances can be pre-computed. The relevant particles for a particle can be stored in a so-called VerletList. These lists need to be rebuilt when a particle moves out of the cutoff radius or comes in range. In order to reduce the rebuild frequency, a so-called Verlet skin is created, which has the radius of the cutoff radius $r_c + s$ with $s > 0$ being the skin summand. The distance between the particles in the skin still needs to be calculated, as shown in Figure 3.1c. The probability of finding a particle with a skin radius increases in comparison to LinkedCells. A drawback of the VerletLists is that every particle pairs' distance needs to be evaluated, which would have a complexity of $\mathcal{O}(N^2)$. This can be optimized by combining the VerletLists with the LinkedCells algorithm.[GST+19]

VerletClusterLists

This algorithm uses particle clusters for VerletLists. VerletClusterLists[AMS+15] does not construct the neighbor lists out of the particle in the Verlet skin. Instead, particles near each other have similar neighbors and build particle clusters. The algorithm divides the domain in a grid along the x- and y-axis, creating towers along the z-axis. The side of the towers is calculated with Equation 3.2.

$$\text{Tower Side Length} = \sqrt[3]{\frac{\text{Cluster Size} \cdot \text{Domain Volume}}{\text{Number of Particles}}} \tag{3.2}$$

The clusters are then built inside each tower. The particles are sorted along the z-axis, and a certain number of particles form a cluster. The last cluster is filled with dummy particles if not enough particles are left. Then neighbor lists are built out of the clusters. The distance calculation is done between two neighboring clusters and the cluster itself. Two clusters are neighbors when one particle has at least one neighbor in the other cluster.



(a) DirectSum (b) LinkedCells (c) VerletLists

Figure 3.1.: Interactions for the calculation of the force on the red particle in different container types. The red circle illustrates the cutoff radius, the yellow one the Verlet-skin radius. The distance of every particle connected with an arrow is analyzed. Only the force with the particles within the cutoff radius is calculated. The white particles are not involved in any interactions. In Figure (c), the blue and red cells are used for the neighbor list construction. Source: [GST+19]

3.1.2. Traversal

Containers only store the particles but do not provide any order in which the particle pairs are processed. This is what the traversals do. Furthermore, they aim to parallel the force calculations in order to utilize modern multi-core processors. The traversals try to reduce the scheduling overhead and address the race conditions when Newton3 is applied. Additionally, some traversals have the purpose of balancing the workload among the available cores.
The traversals can be roughly divided between three different categories that address the race conditions differently. The first category uses domain coloring, which assigns a color to every cell. These colors create barriers in the hole domain to prevent race conditions because they can not occur in cells with the same color. Thus all cells with the same color can be dynamically scheduled without locks. The second category tries to utilize every usable thread by dividing the domain into slices. Each thread is assigned one slice. Race conditions can only occur in cells that are neighboring a cell of another slice. These race conditions are solved with local locks. With the slicing, the threads' usage is better; however, obtaining the locks can cause overhead. The third category disables Newton3 entirely, which has the effect that no race conditions exist and the particles can be scheduled arbitrarily.
A container can have multiple traversals, but every traversal is only applicable to one container. Table 3.1 lists all traversals with their corresponding containers.[GST+19]

Container	Traversal
DirectSum	ds_sequential
LinkedCells	lc_c01 lc_c01_combined_SoA lc_c01_cuda lc_c04 lc_c04_HCP lc_c04_combined_SoA lc_c08 lc_c18 lc_sliced lc_sliced_balanced lc_sliced_c02
VerletLists	vl_list_iteration
VerletListCells	vlc_sliced vlc_c18 vlc_c01 vlc_sliced_balanced vlc_sliced_c02
VerletClusterCells	vcc_cluster_iteration
VerletClusterLists	vcl_cluster_iteration vcl_c06 vcl_c01_balanced
VarVerletListsAsBuild	vvl_as_built

Table 3.1.: All traversals and their corresponding container
Source: `https://www5.in.tum.de/AutoPas/doxygen_doc/master/`

3.1.3. Data Layout

AutoPas uses two different data layouts, Array of Structures (AoS) and Structure of Arrays (SoA), to store the particles. AoS stores the particles as a struct containing all the data in one array. This gives the advantage that the particles can be easily accessed. SoA instead builds a structure out of separate arrays for every parameter where all particles' data are stored. The parameters of any particle i can be accessed in the corresponding parameter arrays at the index i. With SoA, the particles are stored in the same cache line and can be loaded simultaneously. This leads to an efficient force calculation by utilizing vectorization efficiently.[GST$^+$19]

3.1.4. Newton3

The Newton3 option optimizes the force calculations with Newton's third law of motion. The law states that for every force exerted on a body i by a body j, body i exerts a force of equal magnitude but opposing direction on body j [New87]. This means that the calculated force F between two particles i and j hold: $F_{ij} = -F_{ji}$. When the Newton3 optimization is applied, the number of force calculations is reduced by a factor of two. Since they only need to be calculated once and can be applied to both interacting particles. As mentioned in Subsection 3.1.2, the optimization has the drawback that it causes race conditions, which have to be addressed when parallelizing the force calculation.[GST$^+$19]

3.2. Auto Tuning

AutoPas contains multiple configurations that can calculate different simulations and scenarios within a different run time. As stated at the beginning of Chapter 3, AutoPas is a base for arbitrary N-body simulations. Therefore, it can not make any assumptions that could be used to select an optimal configuration for the simulation before it is started. To solve this problem, AutoPas selects a configuration during the simulation. To maximize the performance, it needs to map the simulations' state to the most efficient configuration. This is called the algorithm selection problem[Ric76]. AutoPas resolves this problem with auto-tuning.[GST$^+$19]
Different strategies are implemented to select the configurations, which are then tested in the following so-called *tuning-phase*. The auto-tuning process will test every selected and currently applicable configuration over the next iterations. The test consists of using the configuration for a certain amount of iterations. The measured run time of every iteration is used to calculate the *evidence*, which can be the mean value, or it just selects the fastest. Then the best configuration is determined by selecting the fastest evidence. This configuration is then used for the computation of the simulation until the next tuning-phase. After a certain number of iterations, AutoPas repeats the auto-tuning process, because the scenario in a simulation can change and with it, another configuration could have the optimum run time.[GST$^+$19]
AutoPas provides different tuning strategies for the selection of the configurations that it then tests during the following tuning-phase. The focus in this thesis is on FullSearch, where no specific selection occurs as it selects all configurations and tests them.

Part II.

Implementation of a Predictive Tuning Strategy

4. Refactoring Search Space based Tuning Strategies

AutoPas already has different tuning strategies implemented. They share similarities because they all have the same interface and overall purpose of selecting the best configuration. Additionally, the implemented tuning strategy shares other similarities with FullSearch. They both represent the *search space*, all applicable configurations, in a set. Therefore, other functions and variables are identical in both classes. To make the code more maintainable and avoid code duplication, the created superclass SetSearchBasedTuningStrategy contains the functionality both classes share. As seen in Figure 4.1, PredictiveTuning and FullSearch both inherit the search space and the functions both classes use. It also shows that PredictiveTuning has other similarities to FullSearch, like how the search space is filled and how the Newton3 option is removed. It contains all of FullSearche's functions, adapts them, implements additional variables and functions to optimize the tuning-process.

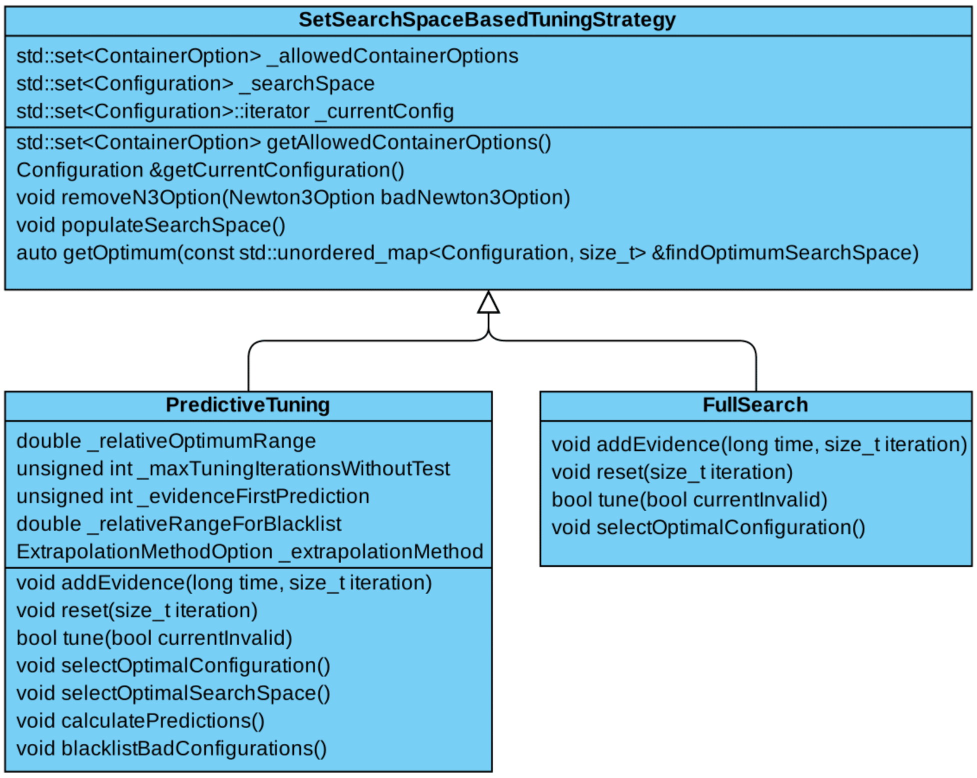

SetSearchSpaceBasedTuningStrategy
std::set<ContainerOption> _allowedContainerOptions
std::set<Configuration> _searchSpace
std::set<Configuration>::iterator _currentConfig
std::set<ContainerOption> getAllowedContainerOptions()
Configuration &getCurrentConfiguration()
void removeN3Option(Newton3Option badNewton3Option)
void populateSearchSpace()
auto getOptimum(const std::unordered_map<Configuration, size_t> &findOptimumSearchSpace)
PredictiveTuning
double _relativeOptimumRange
unsigned int _maxTuningIterationsWithoutTest
unsigned int _evidenceFirstPrediction
double _relativeRangeForBlacklist
ExtrapolationMethodOption _extrapolationMethod
void addEvidence(long time, size_t iteration)
void reset(size_t iteration)
bool tune(bool currentInvalid)
void selectOptimalConfiguration()
void selectOptimalSearchSpace()
void calculatePredictions()
void blacklistBadConfigurations()
FullSearch
void addEvidence(long time, size_t iteration)
void reset(size_t iteration)
bool tune(bool currentInvalid)
void selectOptimalConfiguration()

Figure 4.1.: UML Class Diagram of the tuning strategies that are based on a set to store the configurations

5. Predictive Tuning

The predictive tuning strategy's general idea is to extrapolate the collected evidence of every configuration to create a prediction. Then only the configurations with the best predictions are selected and tested during the following tuning phase. The goal is to prevent inefficient configurations from being tested in the first place and thereby reducing the run time of the simulation.
To gather evidence that PredictiveTuning later uses to extrapolate the predictions, it needs to test every configuration during the first number of tuning phases. This can be seen in Figure 5.1, where the selection strategy is shown in red. In the beginning, PredictiveTuning needs to check if enough tuning-phases have been performed to be able to calculate a prediction. If this is not the case, it selects all configurations for testing. Otherwise, it will proceed with calculating the predictions for each configuration. PredictiveTuning goes through them to check if the respective prediction is in the optimum predictions interval or if it was not tested in a certain number of tuning-phases. If either one of these is the case, the configuration gets selected to be tested in the following tuning-phase. After the selection, the selected configurations get tested, which is represented in blue. If not enough tuning-phases were performed yet, and all configurations were selected, they all get tested. When PredictiveTuning used predictions for the selection. It first tests the configurations with a prediction in the optimum predictions interval. After that, PredictiveTuning checks if at least one valid configuration was tested successfully and produced evidence. If that is not the case, it calculates a new best prediction out of the remaining configurations and selects new configurations with the best predictions to test them. This procedure is done until either a valid configuration is found or there are no configurations left to test. If PredictiveTuning runs out of configurations to test, it throws an exception. Following the testing of a valid configuration, PredictiveTuning tests the configurations that have not been tested in a defined number of tuning-phases, represented in orange. After the testing, PredictiveTuning checks if it produced at least one evidence and then selects the fastest configuration, else it throws an exception.
The number of tuning-phases in which all configurations get tested depends on how many base points should be used by the extrapolation. Additionally, it determines the degree of the polynomial if the Lagrange or Newton polynomial is applied. The goal is to have a sufficiently accurate prediction with the least amount of base points because testing every configuration during a tuning-phase is expensive. However, selecting an inefficient configuration at the end of a tuning-phase could be worse. A good compromise needs to be drawn depending on the degree of the Lagrange or Newton polynomial or how many points the linear regression should use. The goal of PredictiveTuning is to reduce the run time. Thus the accuracy only needs to be sufficient enough to select efficient configurations. Therefore, a less then perfect precision can be accepted if it reduces the run time.
When PredictiveTuning completed enough tuning-phases, it will calculate a prediction for every configuration before each tuning-phase. It will use these predictions to determine

the configurations that the upcoming tuning-phase will test assumed each configuration has been tested enough in the previous tuning-phases. When not enough evidence for a configuration was produced to this point, the prediction is set to the maximum value, and PredictiveTuning selects it automatically. Those will be tested with the configurations that have not been tested in a certain number of tuning-phases. To chose the configurations, the best prediction is used to create an interval containing the best configurations. If a prediction is in the interval, the corresponding configuration is selected. The interval dimension is relative to the optimum prediction because the run time can differ significantly depending on the simulation's scenario. A fixed interval could lead to different problems. It could be too large for small and too little for big simulations. However, the relative interval has the disadvantage that it expands more when the fastest prediction increases. As a result, more configurations with a longer run time from the beginning can be chosen for the next tuning-phase.
As seen in the red part of Figure 5.1, when PredictiveTuning goes through the configuration and selects the configurations for testing, there are configurations selected that have not been tested for a certain number of tuning-phases. PredictiveTuning tests them to prevent an inaccurate extrapolation since it uses the last produced evidence to extrapolate those to the current time value. The time value is further away with every following tuning-phase in which the configuration is not tested. Additionally, the scenario of the simulation can change, and other configurations could get more efficient. Therefore a configuration is guaranteed to be tested every m tuning-phases. PredictiveTuning wants to avoid inaccurate extrapolations, but it also wants to avoid testing inefficient configurations. The described functionality leads to a reasonable compromise because the evidence collected during the previous tuning-phase should lead to the most accurate prediction. Having to test each configuration every m tuning-phases means testing all the inefficient configurations, which takes up a lot of time. This parameter needs to be set with caution. PredictiveTuning should only test the configurations with imprecise predictions as often as it is necessary, to prevent inaccurate prediction, which could be worse, especially with extrapolation polynomials of a higher degree.
The new tuning-strategy needs to change the tuning process. Not every configuration in the search space can always be applied to the current state of the simulation. This does not matter when PredictiveTuning tests all configurations. However, it needs to be considered when only the configurations with the optimal predictions and those that have not been tested for a certain number of tuning-phases are probed. To ensure that an efficient configuration is selected in the end, PredictiveTuning needs to test the optimal predictions first and has to check if an applicable configuration was tuned. If that is not the case, it selects new optimal predictions out of the remaining configurations. As seen in Figure 5.1 following the 'No'-branch of the first blue decision node, PredictiveTuning does this until at least one usable configuration is tested, or if there are no configurations left to test. Then the configurations that were selected because they have not been tested in several tuning-phases are then tested. PredictiveTuning selects the optimal configuration out of all produced evidence. This procedure ensures that an optimal prediction is tested. It prevents that PredictiveTuning only selects the best configuration from the evidence of the configurations that were selected because they had not been tested for a certain number of tuning-phases. This is important as the configuration for the next couple thousand iterations has to be picked out of those. The goal is to select the most efficient configuration.

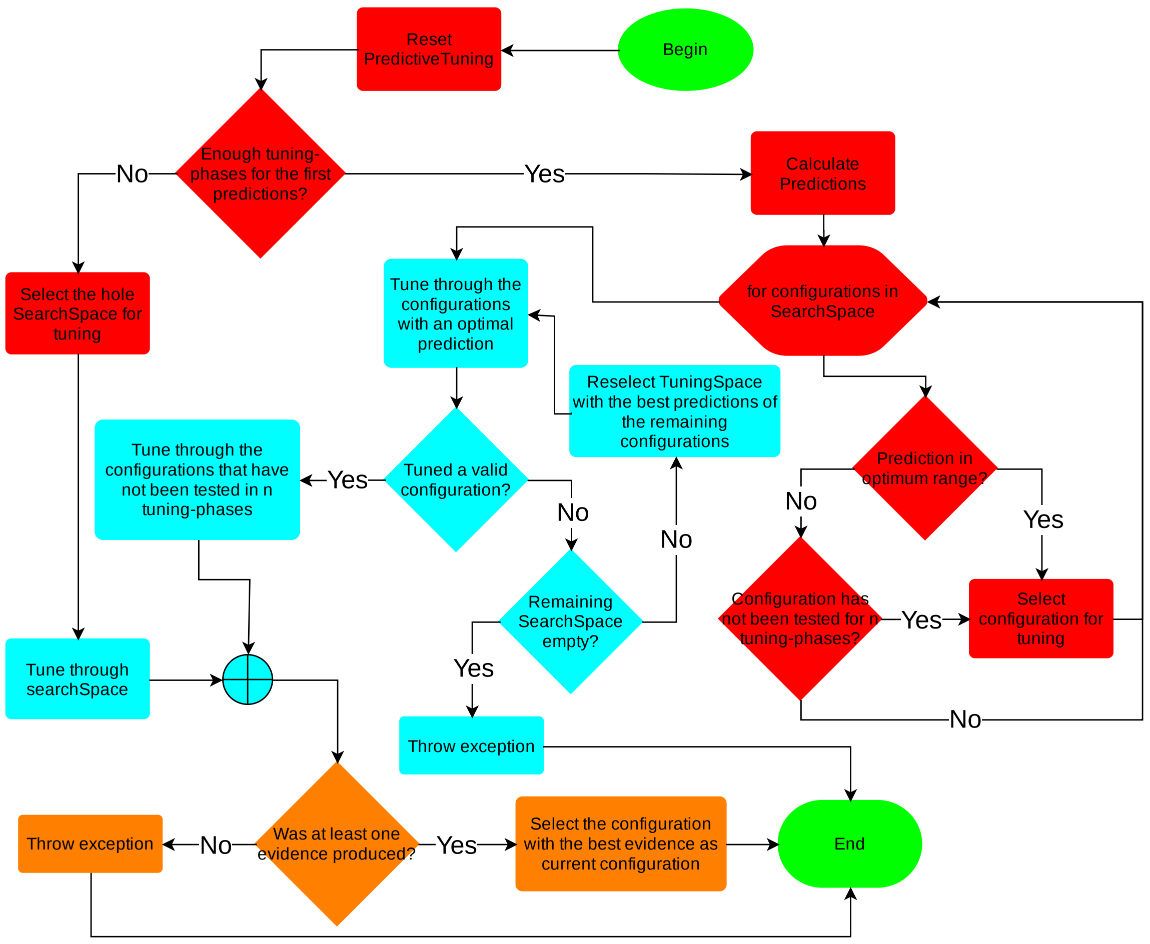

Begin
Reset PredictiveTuning
Enough tuning-phases for the first predictions?
No
Yes
Calculate Predictions
for configurations in SearchSpace
Select the hole SearchSpace for tuning
Tune through the configurations with an optimal prediction
Reselect TuningSpace with the best predictions of the remaining configurations
Prediction in optimum range?
Tune through the configurations that have not been tested in n tuning-phases
Yes
Tuned a valid configuration?
No
No
No
Yes
Remaining SearchSpace empty?
Configuration has not been tested for n tuning-phases?
Yes
Select configuration for tuning
Yes
Tune through searchSpace
Throw exception
No
Throw exception
No
Was at least one evidence produced?
Yes
Select the configuration with the best evidence as current configuration
End

Figure 5.1.: The UML Flow Diagram shows the whole tuning process of PredictiveTuning. The objects in red describe the selection strategy for the configurations that get tested. After the selection, the testing phase is executed, which is shown with the blue objects. In the end, the process of evaluating the tested configurations is described in orange.

5.1. Extrapolation Methods

The extrapolation methods are implemented following the formulas stated in Section 2.1. To prevent redundant calculations, the functions calculated with the linear regression and the Newton polynomial for every configuration are stored to reuse them if no new evidence was added in the previous tuning-phase. The polynomial created by the Lagrange polynomial can not be stored. Therefore, the extrapolation must be calculated again for every new prediction. This does not affect the run time because the number of configurations and the evidence used in the extrapolation, more in Section 6.1, is not high enough to significantly impact the simulation duration.

5.2. Blacklist

The assumption that an inefficient configuration does not drastically increase its efficiency during a simulation opens up new functionality to decrease tuning time. PredictiveTuning uses this assumption to erase all inefficient configurations from the search space. It compares the evidence of every configuration to the evidence of the fastest configuration after the first tuning phase or when tuning produces the first evidence for a configuration. If the evidence of a configuration is greater than the best evidence multiplied by the *relative blacklist range*, it is blacklisted. Then it is not possible to select this configuration again during this simulation. The blacklist will only be applied once for every configuration. When a user wants PredictiveTuning to use the blacklist during a simulation, the user needs to set the relative blacklist range. This value determines the relative range of the blacklist. If the user does not set the variable, PredictiveTuning does not use this function.
The blacklist's goal is to increase the tuning-process efficiency by prohibiting blacklisted configurations from being selected. Those configurations are expected to be inefficient. Therefore, they would only be selected either when the configuration was not tested for too many tuning-phases or when the prediction was inaccurate. This reduces the chance of selecting an inefficient configuration as the best configuration and the number of inefficient configurations used during the simulation. The blacklist should reduce the tuning time significantly and speed up the run time of the simulation. Nevertheless, it could slow the simulation down if the allowed configuration interval is to narrow and configurations get blacklisted that could be the best later in the simulation.

Part III.

Analysis

Every simulation of the following analysis was performed on the CoolMUC-2 system of the LRZ Linux Cluster. Each node consists of 28-core Haswell nodes and has 64 GB RAM available. There were two simulations used for the performance analysis of PredictiveTuning. A static simulation that has particles in-homogeneously distributed in the domain, as seen in Figure B.1, and the Spinodal Decomposition Experiment, which is an example of a simulation that is usually performed with AutoPas.
The phenomenon of the Spinodal Decomposition describes the rapid decomposition of a homogeneous solution, which results in the creation of two different thermodynamic phases[CHK+94]. The experiment is performed in two separate simulations. The first simulation initializes the particle grid. Then all particles get equilibrated by simulating 100.000 thousand iterations with a constant temperature. In the end, a checkpoint of the simulation is created. The second simulation, the actual decomposition, is started by initializing the particles with the first simulation's checkpoint and has the same parameters as the equilibration except that it has half the temperature. The simulation is then run for 80.000 iterations, where the temperature reduction causes the particles to form clusters.
The configurations all have different efficiency, and some take up a lot of time. Therefore, some of the shown plots do not depict the simulation's complete data but rather a selection. This is done for consistency and to make essential parts visible. Additionally, if the behavior of the three implemented extrapolations methods is similar, only one of them is shown as an example.

6. Evaluation of Input Variables

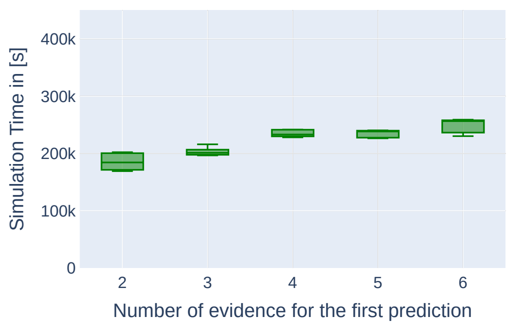
Many variables influence how PredictiveTuning selects the configurations for the following tuning-phase. The analysis of the input variables of PredictiveTuning should lead to reasonable default values for the best performance. However, it can not be claimed that they will be the best for every simulation.
The different input parameters are evaluated by the run time of the simulation and the accuracy of PredictiveTuning. The accuracy is measured by comparing the evidence of the selected configurations of FullSearch and PredictiveTuning in the same tuning-phase. If the time between the two does not deviate within five percent, it is counted as accurate. The sum of the accurate selections divided by the number of tuning-phases is then presented as the overall accuracy. If not stated otherwise, this is meant when talked about the accuracy.

6.1. Number of Evidence until First Prediction

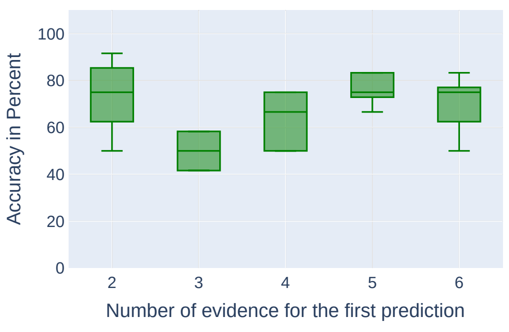
The number of evidence until the first prediction (Further named as *evidenceUntilPrediction*) affects two things. It declares how many tuning phases every configuration is tuned until the first prediction can be calculated. This also determines how much evidence is used to calculate the prediction and the degree of the Lagrange and Newton polynomial. This variable can significantly affect the accuracy of the prediction and the efficiency of PredictiveTuning. The goal is to have the least amount of tuning-phases that test all configuration before the first prediction is calculated while obtaining an accurate prediction.
As mentioned before, this variable affects the degree of the polynomial of Lagrange and Newton. The linear regression is always a linear function. Therefore they need to be analyzed separately.

6.1.1. Linear Regression

Increasing the *evidenceUntilPrediction* with the linear regression only makes sense when the accuracy increases, which leads PredictiveTuning to choose more efficient configurations to reduce the run time of a simulation. If this is not the case, the impreciseness is accepted.
The smallest number possible is two. Figure 6.1a shows that the Spinodal Decomposition Experiment's total run time with two evidence is generally faster than when three or more evidence have to be produced for the first prediction. Additionally, the linear regression is accurate with only two *evidenceUntilPrediction*, and PredictiveTuning is not getting more accurate with more evidence used in the extrapolation, as seen in Figure 6.1b. In conclusion, only having to produce two evidence for the first prediction is faster and has higher accuracy. This is the number chosen as the default value for the linear regression.



(a) Run time

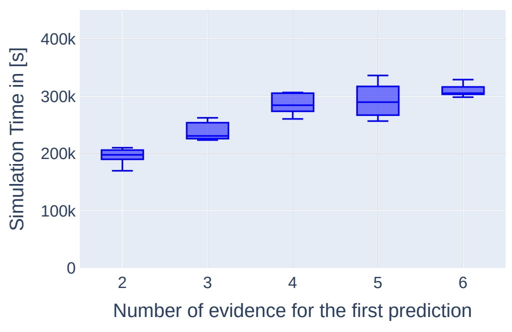


(b) Accuracy

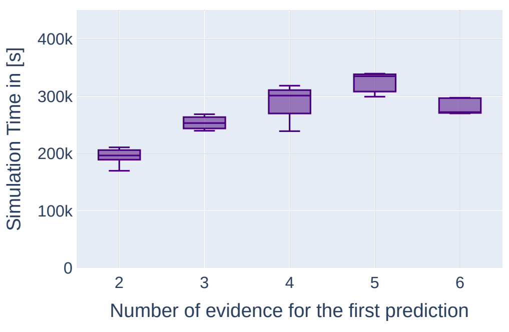
Figure 6.1.: The run time of the Spinodal Decomposition (a) and the accuracy (b) of PredictiveTuning using the linear regression with *evidenceUntilPrediction* set to two and three. The simulation is faster when only two evidence are used in the extrapolation. The accuracy can be more consistent with a higher value. However, it is not more accurate. Colors represent the tuning strategy. For the values, see Figure A.2

6.1.2. Lagrange and Newton polynomial

Figure 6.2 shows the Spinodal Decomposition run time and displays that choosing a higher value for *evidenceUntilPrediction* increases the run time significantly for both extrapolation methods. This was anticipated because there are two obstacles to effectively reduce the run time with a higher number for *evidenceUntilPrediction*. The first one is that more tuning-phases are testing all configurations, as seen in Figure 6.3, which is very time-consuming. An extrapolation method with a higher number of base points needs to calculate more accurate predictions, which leads to the selection of more efficient configurations. The Runge Phenomenon affects the accuracy of the predictions significantly, as seen in Figure 6.4a. It shows that the Lagrange polynomial is most accurate with two base points and that the predictions get more inaccurate the more evidence is used for the extrapolation. However, this does not affect the overall accuracy of the selection of configurations. Figure 6.5 shows that extrapolating with an even number of base points is more accurate than with an uneven number.
Increasing the value for *evidenceUntilPrediction* does not reduce the run time. Therefore a lower number is recommended. The default value is set to two evidence until the first prediction.

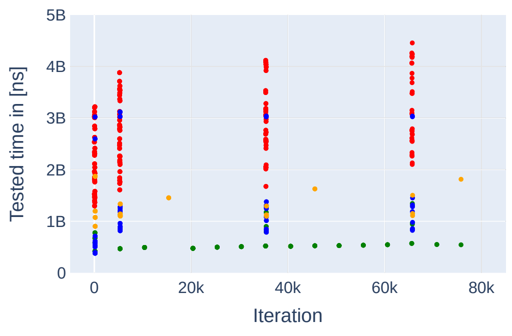


(a) Lagrange polynomial

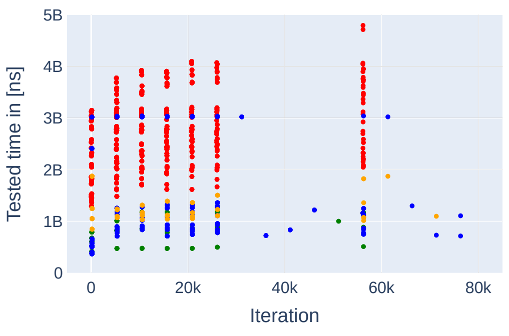


(b) Newton polynomial

Figure 6.2.: The Spinodal Decomposition's run time with PredictiveTuning and different inputs for *evidenceUntilPrediction*. Increasing the input for the *evidenceUntilPrediction* increases the run time for both extrapolation methods. The prediction strategy has to test inefficient configurations for more tuning-phases, and fewer were it avoids them through the predictions. Colors represent the tuning strategy. For the values, see Figure A.2

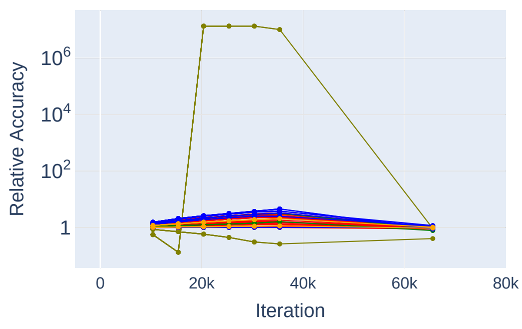


(a) *evidenceUntilPrediction* set to 2

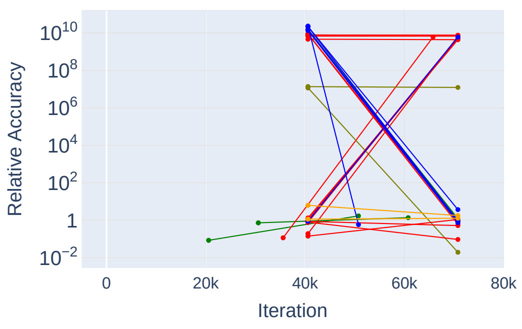


(b) *evidenceUntilPrediction* set to 6

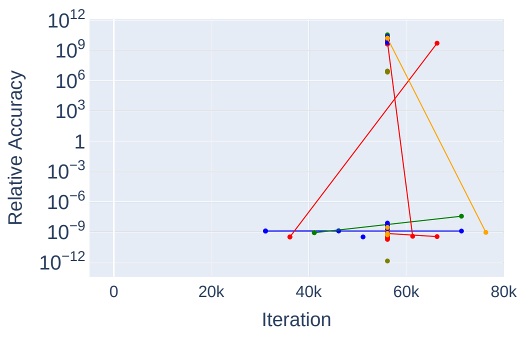
Figure 6.3.: The evidence produced during the Spinodal Decomposition with PredictiveTuning using the Lagrange polynomial for *evidenceUntilPrediction* set to two (a) and six (b). Each point represents one produced evidence. Increasing the *evidenceUntilPrediction* increases the number of tuning-phases that test all configurations and decreases the number of tuning-phases that can avoid testing inefficient configurations. The usage of more base points by the extrapolation does not ensure a more accurate prediction. This can be seen, for example, in (b) in all non-full-search tuning phases where the selected configuration does not produce the fastest evidence. The fact that the prediction quality worsens with an increasing number of base points can be traced back to the Runge Phenomenon. For the values see Figure A.1



(a) *evidenceUntilPrediction* set to 2

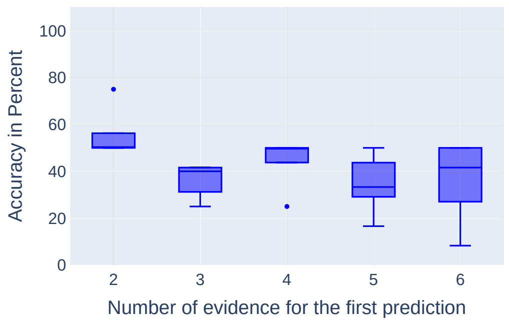


(b) *evidenceUntilPrediction* set to 3

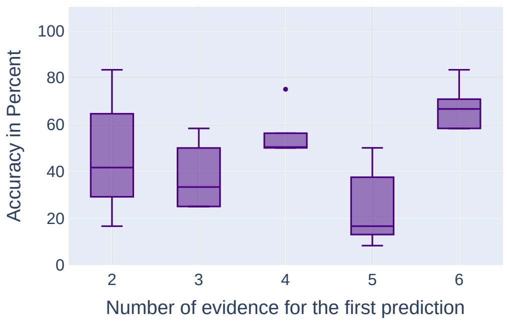


(c) *evidenceUntilPrediction* set to 4

Figure 6.4.: The accuracy for each prediction is calculated by dividing the prediction and the evidence during the same tuning-phase. The best possible value for a prediction is 1. Each dot represents the prediction accuracy, calculated by dividing the prediction through the tested evidence in the same tuning-phase. When two dots are connected, it means that they belong to the same configurations. The worst prediction with two evidence (a) is at about five million times the produced evidence and far less than for three (b) with exceeding the real produced evidence by a factor by around twenty-five billion. The accuracy gets even worse when the extrapolation uses four evidence (c) compared to three. There is no prediction that is near the produced evidence due to the Runge Phenomenon. Colors represent containers. For the values see Figure A.1



(a) Lagrange polynomial

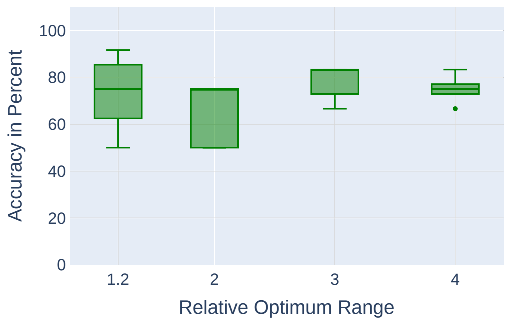


(b) Newton polynomial

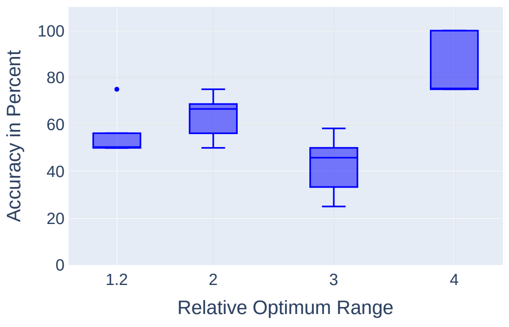
Figure 6.5.: The accuracy of PredictiveTuning and different inputs for *evidenceUntilPrediction* during the Spinodal Decomposition. The accuracy for both extrapolations is higher for even numbers. The Lagrange polynomial has the best accuracy with two base points and the Newton polynomial with six. Colors represent the tuning strategy. For the values, see Figure A.2

6.2. Relative Optimum Range

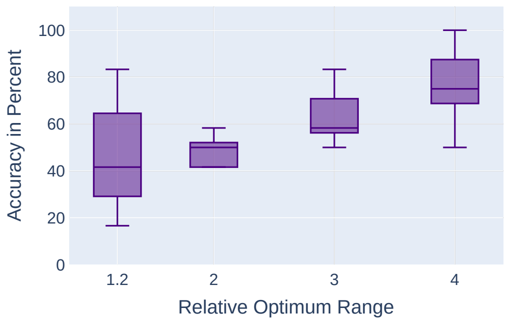
The relative range of the interval for the optimal predictions (Further named as *relative optimum range*) primarily affects how the chance rises for a configuration to get selected for the tuning process. Setting the *relative optimum range* to a higher value means that more configurations get selected. As seen in Figure 6.8b, where the number of configurations selected for every tuning-phase is considerably more with a *relative optimum range* of four, compared to Figure 6.8a, which has it set to 1.2. Testing more configurations does not necessarily translate to higher overall accuracy. Figure 6.6 shows that a higher range of the interval can be more inaccurate. Nevertheless, the highest value for the *relative optimum range* has the best accuracy for the Lagrange and Newton Polynomial.
A better accuracy does not affect the run time of the Spinodal Decomposition significantly, as seen in Figure 6.7. The run time is reduced for a higher value of the *relative optimum range* with the Lagrange and Newton polynomial. The simulation takes a bit longer with the linear regression. Overall the impact is neither significant nor consistent enough. Therefore, the *relative optimum range* should preferably be a value between $[1.1, 2]$ because PredictiveTuning aims to reduce the selection of configurations.



(a) Linear Regression

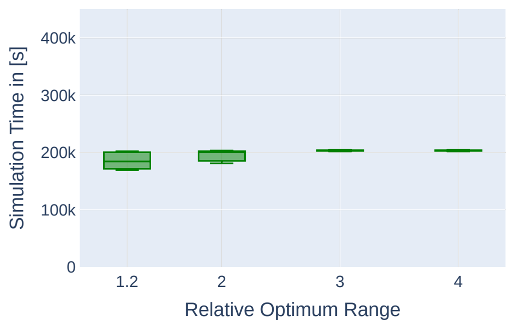


(b) Lagrange polynomial

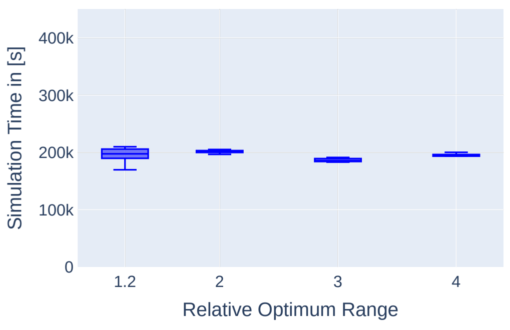


(c) Newton polynomial

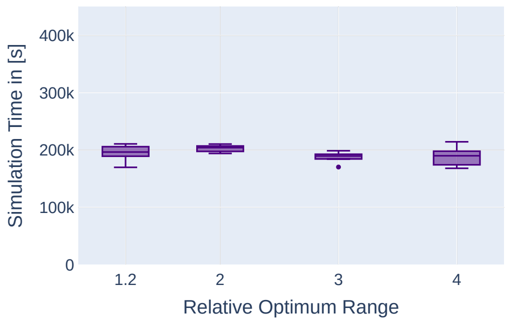
Figure 6.6.: The accuracy of PredictiveTuning and 1.2, 2, 3, and 4 as input for the *relative optimum range* during the Spinodal Decomposition. The accuracy does not always increase when the *relative optimum range* is set with a higher value. Colors represent the tuning strategy. For the values, see Figure A.2



(a) Linear Regression

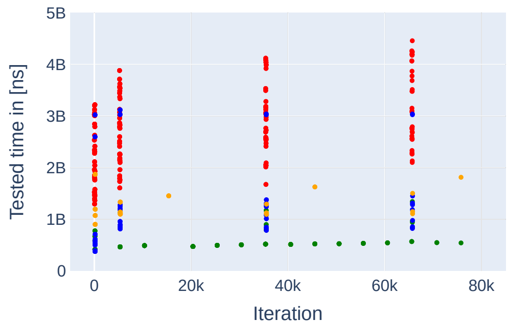


(b) Lagrange polynomial

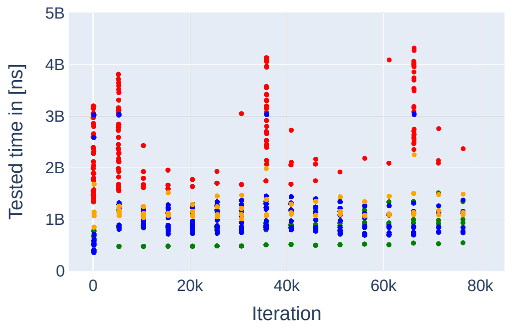


(c) Newton polynomial

Figure 6.7.: The Spinodal Decomposition's total run time with PredictiveTuning and 1.2, 2, 3, and 4 as input for the *relative optimum range*. Increasing the *relative optimum range* increases the number of tested configurations. When the interval is still narrow enough that no inefficient configurations are selected, it can slightly reduce the run time using the Lagrange and Newton polynomial. The linear regression is faster, with a lower *relative optimum range*. Colors represent the tuning strategy. For the values, see Figure A.2



(a) *Relative optimum range* set to 1.2



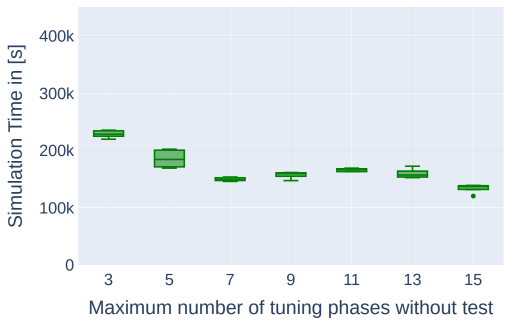
(b) *Relative optimum range* set to 4

Figure 6.8.: The produced evidence during the Spinodal Decomposition with Predictive-Tuning with the Lagrange polynomial and the input for the *relative optimum range* set to 1.2 (a) and 4 (b). Each point represents one produces evidence. Since increasing the number results in a widening of the interval in which the prediction has to be to get selected, setting the *relative optimum range* to four (b) results in the selection and testing of many more configurations, which always contains the most efficient one. When the number is set to 1.2 (a), only a few configurations get selected, and at one point, it is not testing the most efficient configuration. Colors represent containers. For the values see Figure A.1

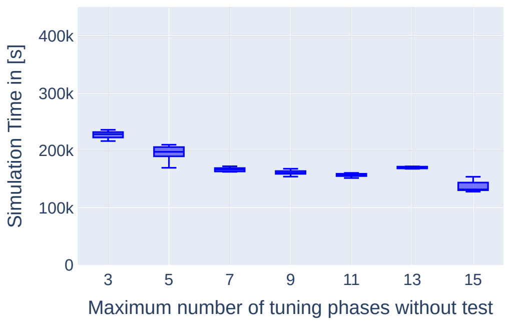
6.3. Maximum Tuning Phases Without Test

The number of maximum tuning phases a configuration might not be tested (Further named as *maxTuningPhasesWithoutTest*) can have a more significant impact on the run time than the *relative optimum range*. As it affects the accuracy of the prediction and how often PredictiveTuning has to test inefficient configurations. The goal is to have this number as high as possible without losing too much accuracy that efficiently enough configurations are still selected.
Figure 6.9 shows that the Spinodal Decomposition's run time decreases when *maxTuningPhasesWithoutTest* is selected higher. It reduces the number of tuning-phases where all inefficient configurations are selected. When the parameter is set between six and thirteen, it means that all configurations could only be tested once again after the first prediction if their prediction never is in the optimum range. A reduction of the run time can not be considered for a general proposition of the effectiveness and can only be used for this specific simulation. When the number is greater than thirteen, PredictiveTuning does not test the inefficient configurations again, which produces the fastest run time. Additionally, the run time of the simulation is more consistent when a higher number is selected for the linear regression and the Lagrange polynomial.
The accuracy of PredictiveTuning does not necessarily decrease with a higher *maxTuningPhasesWithoutTest*. As seen in Figure 6.10, where the average accuracy for three is worse for the linear regression and Newton polynomial than the values greater than six. Only the Lagrange polynomial is always around 60% accuracy and primarily differs in the

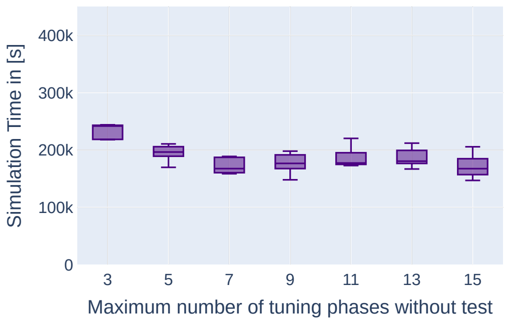
consistency. Newton still has the worst overall accuracy of the three extrapolation methods. The difference between the selected configurations with the Lagrange and Newton polynomial can be seen in Figure 6.11a and Figure 6.11b. The Lagrange polynomial consistently selects configurations with a short run time. In comparison, the selected configurations with the Newton polynomial have differing run time. Around half of them are not the fastest configurations. Those configurations are also selected to calculate the next five thousand iterations. However, the Newton polynomial still produces a faster run time, which means that the time it would take to test all inefficient configurations mostly is higher than selecting some less efficient configurations at the end of a tuning-phase. Because the run time is the deciding factor, the *maxTuningPhasesWithoutTest* should be set to number as high as possible to ensure that all inefficient configurations get tested as few times as possible as long as no significant changes in the configurations are expected.
The Spinodal Decomposition does not change the fastest configuration after the second tuning-phases, as seen in Figure 6.8b, which makes the selection of a higher number so effective. If more changes are expected, it could be beneficial to use the variable to check if some changes have occurred.



(a) Linear Regression

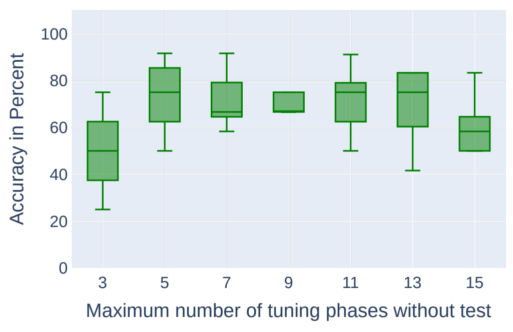


(b) Lagrange polynomial

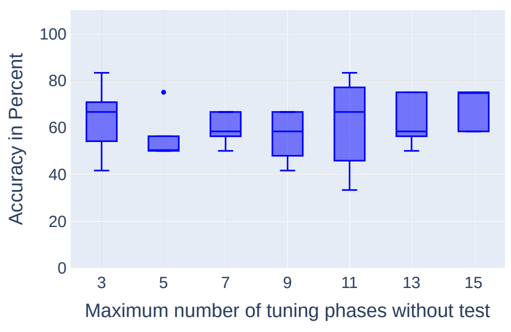


(c) Newton polynomial

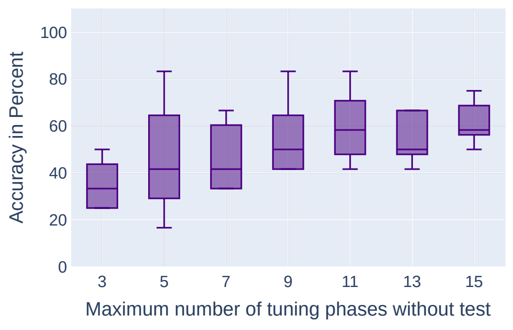
Figure 6.9.: The Spinodal Decomposition's total run time with PredictiveTuning and different inputs for *maxTuningPhasesWithoutTest*. A lower *maxTuningPhasesWithoutTest* leads to an increase in the total run time because inefficient configurations get tested more frequently. With a number between six and thirteen, the inefficient configurations only get tested once again, and a greater number has the effect that those are not tested again. The reason for the difference in the run time is the simulation itself and how the update at a certain point in time affects the predictions. Colors represent the tuning strategy. For the values, see Figure A.2



(a) Linear Regression

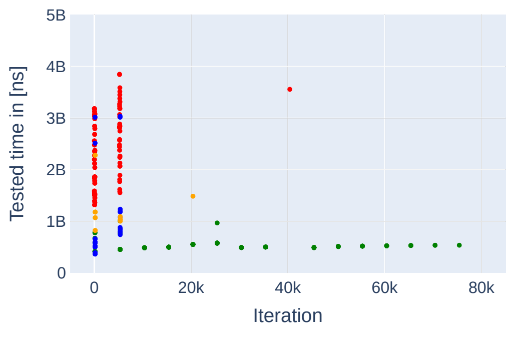


(b) Lagrange polynomial

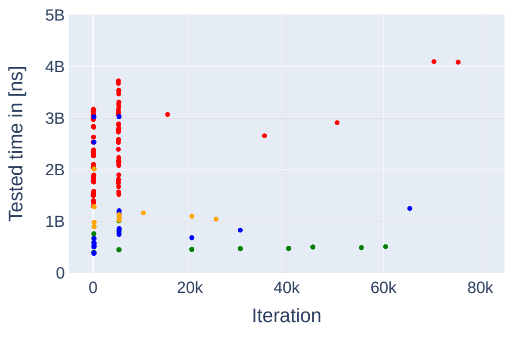


(c) Newton polynomial

Figure 6.10.: The accuracy of PredictiveTuning with different inputs for *maxTuningPhasesWithoutTest*. The accuracy does not increase when the number is lower, and all configurations get tested more often. Testing them more often has a negative effect. Additionally, the accuracy can be increased when the inefficient configurations are less tested. The three extrapolation methods are all affected differently. The linear regression (a) has s better and more consistent accuracy at first. This changes when the *maxTuningPhasesWithoutTest* is greater than nine, and the overall accuracy is worsened. The Newton polynomial is more accurate when the inefficient predictions are tested less. The accuracy of the Lagrange polynomial is always around 60% and only changes in the consistency. It does get more accurate when the inefficient predictions are not tested again. Colors represent the tuning strategy. For the values, see Figure A.2



(a) Lagrange polynomial



(b) Newton polynomial

Figure 6.11.: The produced evidence of the tested configurations with *maxTuningPhasesWithoutTest* set to fifteen for the Lagrange (a) and Newton (b) polynomial. Each point represents one produces evidence. The tested configurations with the Newton polynomial are not always the most efficient. The Lagrange polynomial always selects the fastest configurations and sometimes a less efficient one. Colors represent containers. For the values see Figure A.1

6.4. Blacklist

The blacklist reduces the run time by eliminating inefficient configurations when they are first tested. Figure 6.12 shows every tested evidence. In the first tuning-phase, there are more configurations tested than in the following ones. This illustrates the functionality of the blacklist as it erases the inefficient configurations from the search space.
The main goal is to reduce the tun time of the simulation. As seen in Figure 6.13b, the Spinodal Decomposition Experiment's total run time without the blacklist is significantly higher than with it. The run time can roughly be reduced by 33% up to 45%. Furthermore, in this simulation, the narrowing interval of the blacklist does not have an impact on the run time. It is nearly the same for the relative blacklist range of five and ten as both eliminate the worst predictions. Narrowing the blacklist range further does not reduce the run time significantly for this simulation.
When the *maxTuningPhasesWithoutTest* is set to a more efficient value, as explained in Section 6.3, the effect of the blacklist is reduced, as seen in Figure 6.13a. Comparing the run time of both figures shows that the simulation has the lowest run time when the relative blacklist range and *maxTuningPhasesWithoutTest* both are set to five. The blacklist can reduce the negative effect of the lower setting of *maxTuningPhasesWithoutTest* and can use the up site to reduce the run time.
The key message is that the blacklist is an excellent addition to making the tuning more efficient and reducing the total run time by eliminating hopelessly slow configurations.

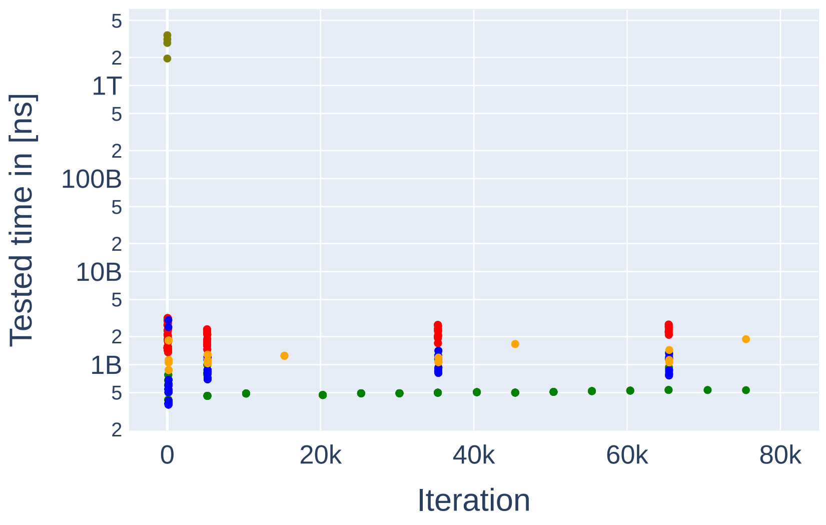
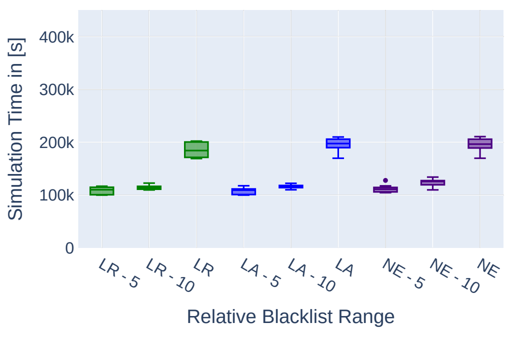
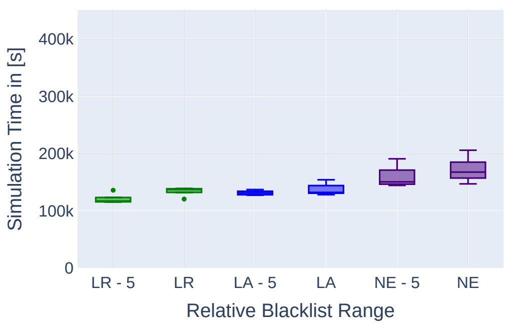


Figure 6.12.: The produced evidence during the Spinodal Decomposition with Predictive-Tuning using linear regression and a relative blacklist range of 5. Each point represents one produced evidence. After the first tuning-phase, the inefficient configurations get blacklisted, which means that in the following ones, there are fewer configurations that can be selected. Colors represent containers. For the values see Figure A.1



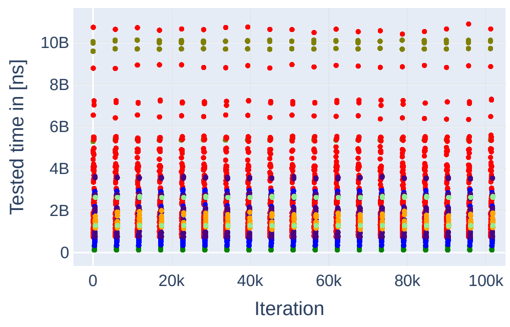
(a) *maxTuningPhasesWithoutTest* set to 5



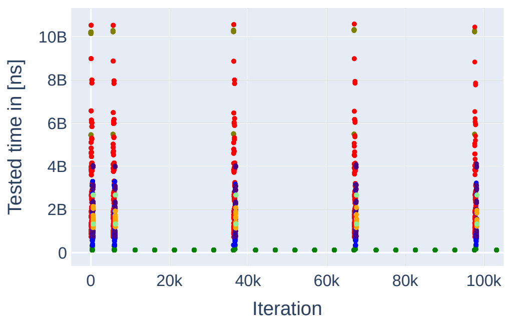
(b) *maxTuningPhasesWithoutTest* set to 15

Figure 6.13.: The run time of the Spinodal Decomposition Simulation with PredictiveTuning using linear regression(LR), Lagrange(LA), and Newton polynomial(NE). The difference between (a) and (b) is the input for *maxTuningPhasesWithoutTest*. Apart from that, the simulations differ in the blacklist range input. When no number is stated, it means that PredictiveTuning does not use the blacklist. It is removing all inefficient configurations after the first tuning-phase speeds up the tuning-process of all extrapolation methods. It is more important to exclude the inefficient configuration because the run time does not significantly improve when the blacklist range is reduced from ten to five, as seen in (a). Additionally, (b) shows that the blacklist always affects the run time, even when PredictiveTuning does not test the inefficient predictions again. Comparing (a) and (b) shows the benefits of the blacklist with the addition of the lower value of *maxTuningPhasesWithoutTest*. The simulation is faster when the variable is set to five. Colors represent the tuning strategy. For the values, see Figure A.2

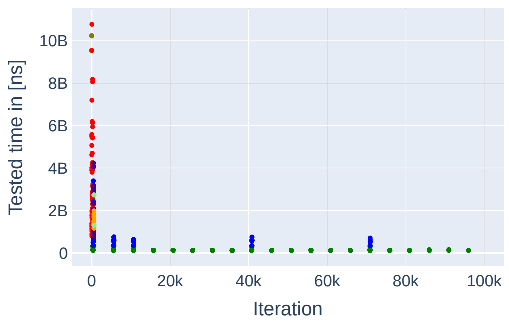
7. Comparing PredictiveTuning to FullSearch



(a) FullSearch



(b) PredictiveTuning using linear regression



(c) PredictiveTuning using linear regression and blacklist

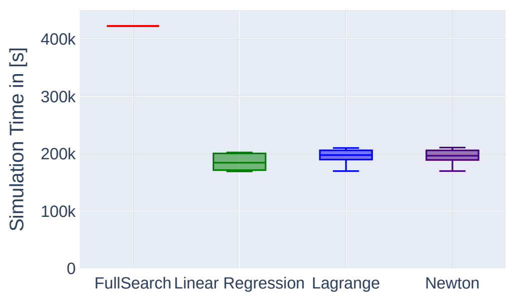
Figure 7.1.: Every produced evidence by all configurations during the inhomogeneous performance test. Each point represents one produced evidence. Since FullSearch (a) tests every configuration frequently, it produces much more evidence. The predictive strategies avoid this by testing only the configurations with the best predictions. This is improved even further with the usage of the blacklist (c). Colors represent containers. For the values see Figure A.1

PredictiveTuning is created to speed up the tuning process and, therefore, to be more efficient than FullSearch. The effectiveness of the new strategy should be seen through a reduction of the run time.
The inefficiency of FullSearch can be seen in Figure 7.1a, as it tests all configurations in

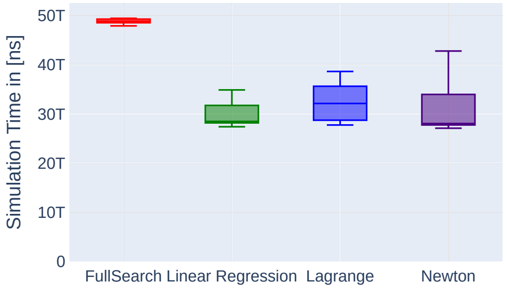
every tuning-phase. In contrast, Figure 7.1b and Figure 7.1c show that PredictiveTuning tests far fewer configurations over the whole simulation, making the tuning process more efficient. As displayed in Figure 7.3, where the testing of the configurations, which only takes around 11.000 of the 100.000 iterations of the simulation, takes up more than 60% of the force calculations. PredictiveTuning can reduce the percentage to around 30% to 25% and down to 10% with the blacklist while simultaneously reducing the run time. Figure 7.2b shows the run time of the inhomogeneous performance test. PredictiveTuning is always faster than FullSearch. The results can differ between a 15% to a 40% reduction, which is not consistent.

The results for the Spinodal Decomposition are more consistent than those of the inhomogeneous performance test. Figure 7.2a displays the Spinodal Decomposition run time when FullSearch or PredictiveTuning with either one of the extrapolations methods is applied as the tuning-strategy. It shows that PredictiveTuning reduces the run time of the Spinodal Decomposition consistently by 50%, even if it does not predict the best configuration every time, as seen in Figure 6.11b.

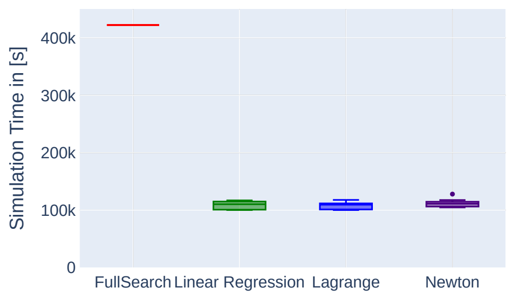
This depicts the new tuning-strategy's effectiveness and that it has a significant impact on the run time. As seen in Figure 7.2c, the blacklist gives the potential to reduce the run time of the Spinodal Decomposition even further. It is subsequently reducing the run time to around 30% to 26% compared to the run time of FullSearch. Considering that four hundred thousand seconds are roughly four and a half days, reducing the duration to a fourth saves a little over three days of computation time.



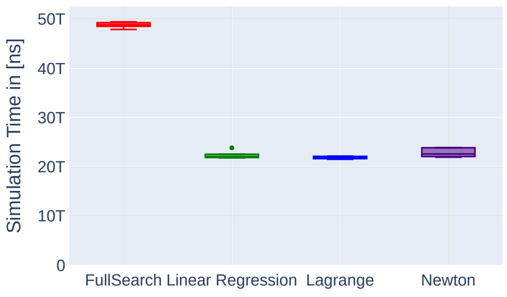
(a) Spinodal Decomposition



(b) Inhomogeneous performance test



(c) Spinodal Decomposition with blacklist



(d) Inhomogeneous performance test with blacklist

Figure 7.2.: The total run time of five runs of the Spinodal Decomposition and ten runs of the inhomogeneous performance test. The simulations are performed using FullSearch and PredictiveTuning using all extrapolation methods with and without the blacklist as the selection strategies. In both simulations, the run time using the Predictive Strategies is lower than the run time of FullSearch, because it avoids testing the inefficient configurations during tuning, which saves much time. The Spinodal Decomposition's run time can be reduced by 50% and even further with the blacklist. The linear regression is the most effective selection strategy. Colors represent the tuning strategy. For the values, see Figure A.2

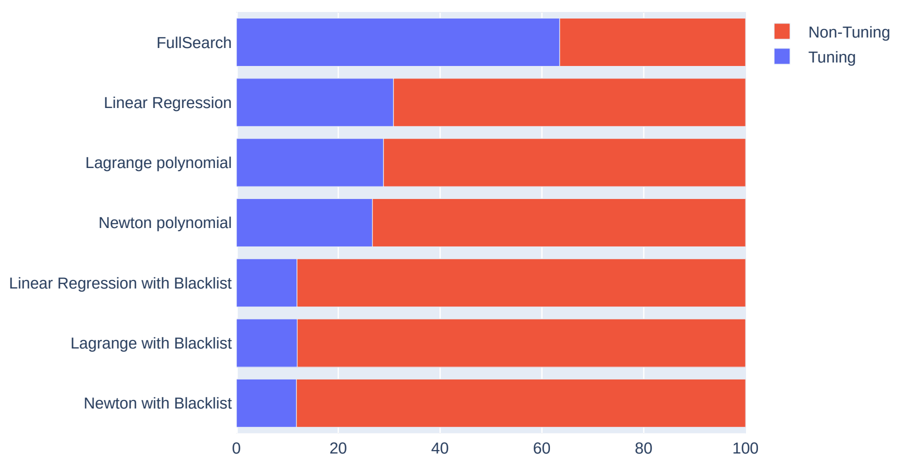


Figure 7.3.: Distribution of the total force calculation time spent during tuning (blue) and non-tuning (red) iterations. Since FullSearch frequently tests many inefficient configurations, it spends a lot of time tuning, while Predictive Strategies avoid this. Blacklisting the inefficient configuration has the effect that those can not be tested twice, which leads to even less time spent in tuning.

Part IV.

Conclusion

8. Conclusion and Future Work

In this thesis, a tuning strategy was implemented that predicts the run time of all configurations using extrapolation. The evaluation with the simulation of the Spinodal Decomposition depicts a significant performance increase when compared to FullSearch. The run time can be reduced by up to 74%. This is a substantial improvement and shows that PredictiveTuning can effectively prohibit inefficient configurations from being tested. Additionally, the analysis displays that linear regression is the best extrapolation method and should be preferred over the others.
It should be possible to improve PredictiveTuning in future work. The extrapolation methods could be improved by implementing an error evaluation. One example could be to calculate the error of the created polynomial for the existing evidence and factoring it into the existing function before the prediction is calculated. Another method would be to compute the prediction error to the produced evidence and use it for the next prediction. This should be an easy way to improve the predictions. Additionally, the optimal prediction range could be calculated dynamically, for example, through the best and worst prediction or evidence. The same could be done for the number of tuning phases, after which configurations have to be tested. One possibility would be to put the configurations into groups with roughly the same behavior and run time. Then it would be possible to test only one of every group. When the prediction is too inaccurate, or the run time has improved to a certain point, the entire group would be tested. Besides, other approaches to predict the configurations like another extrapolation method or another data-driven approach could be implemented. The number of possible improvement shows that there is potential to improve this tuning-strategy and to make it even more efficient.

Part V.

Appendix

A. Legends for the plots

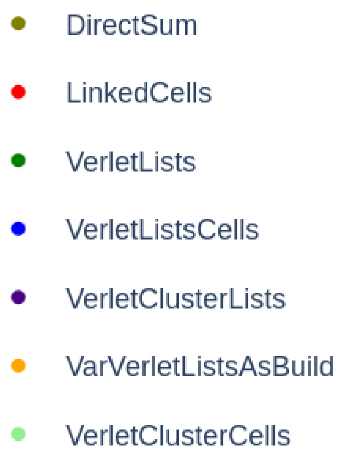


Figure A.1.: Legend for the evidence plots, that distinguishes between the container of the configurations

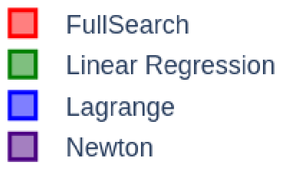


Figure A.2.: Legend for the time and accuracy plots, that distinguished between FullSearch and the extrapolation method used in PredictiveTuning.

B. Simulations

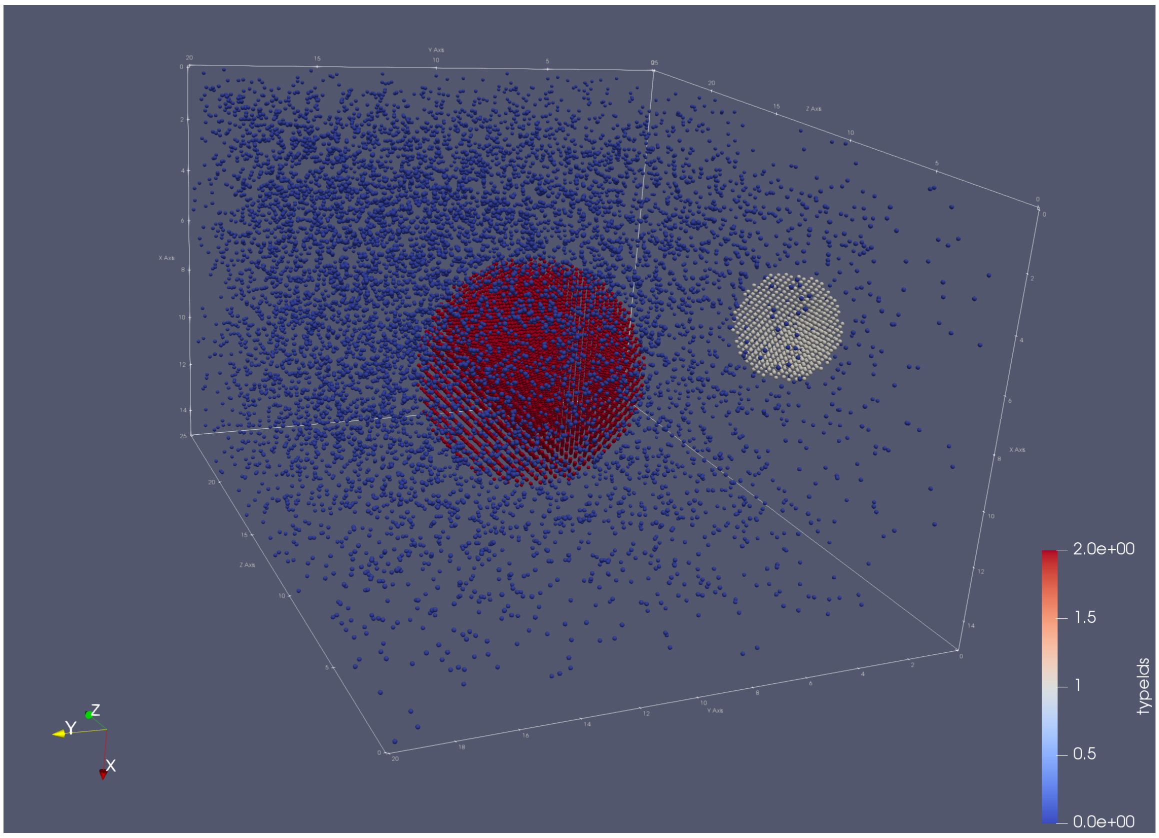


Figure B.1.: Particle Distribution in the inhomogeneous performance test.

List of Figures

List of Tables

Bibliography

[AMS$^+$15] M. J. Abraham, T. Murtola, R. Schulz, S. Páll, J. C. Smith, B. Hess, and E. Lindahl. Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1:19–25, 2015.

[CHK$^+$94] J. B. Clark, J. W. Hastie, L. H. E. Kihlborg, R. Metselaar, and M. M. Thackeray. Definitions of terms relating to phase transitions of the solid state. *Int. Pure Appl. Chem*, 66:577–594, 1994.

[FH07] R. W. Freund and R. W. Hoppe. *Stoer/Bulirsch: Numerische Mathematik 1*. Springer-Verlag, Berlin-Heidelberg, 2007.

[Fro18] I. Frost. *Einfache lineare Regression: die Grundlage für komplexe Regressionsmodelle verstehen*. Springer-Verlag, Wiesbaden, 2018.

[GST$^+$19] F. A. Gratl, S. Seckler, N. Tchipev, H.-J. Bungartz, and P. Neumann. Autopas: Auto-tuning for particle simulations. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 748–757. IEEE, 2019.

[HEDM15] X. Hu, P. Egberts, Y. Dong, and A. Martini. Molecular dynamics simulation of amplitude modulation atomic force microscopy. *Nanotechnology*, 26(23):235705, 2015.

[HS02] T. Huckle and S. Schneider. *Numerik für Informatiker*. Springer-Verlag, Berlin-Heidelberg, 2002.

[New87] I. Newton. *Philosophiae naturalis principia mathematica*, volume 1. Edmond Halley, 1687.

[Pli95] S. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of computational physics*, 117(1):1–19, 1995.

[Rap04] D. C. Rapaport. *The art of molecular dynamics simulation*. Cambridge university press, 2004.

[Ric76] J. R. Rice. The algorithm selection problem. In *Advances in computers*, volume 15, pages 65–118. Elsevier, 1976.

[Run01] C. Runge. Über empirische funktionen und die interpolation zwischen äquidistanten ordinaten. *Zeitschrift für Mathematik und Physik*, 46(224-243):20, 1901.