SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Exploring Fuzzy Tuning Technique for Molecular Dynamics Simulations in AutoPas

Manuel Lerchner

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Remove all TODOS

Exploring Fuzzy Tuning Technique for Molecular Dynamics Simulations in AutoPas

Untersuchung von Fuzzy Tuning Verfahren für Molekulardynamik-Simulationen in AutoPas

Author:	Manuel Lerchner
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisors:	Manish Kumar Mishra, M.Sc. &
	Samuel Newcome, M.Sc.
Date:	10.08.2024

Contents

3 Implementation

This chapter describes the implementation of the Fuzzy Tuning technique in AutoPas. The implementation is divided into three main parts: the generic Fuzzy Logic framework, the Rule Parser, and the Fuzzy Tuning Strategy. The Fuzzy Logic framework is the core of this implementation and implements the mathematical foundation of this technique. The Rule Parser is responsible for parsing the rule base supplied by the user and converting it into the internal representation used by the Fuzzy Tuning framework. Finally, the Fuzzy Tuning Strategy is the interface between the Fuzzy Logic framework and the AutoPas simulation. It is responsible for updating the configuration queue to select configurations to be tested next.

3.1 Fuzzy Tuning Framework

The Fuzzy Tuning framework implements the mathematical foundation of the Fuzzy Tuning technique. It consists of several components that work together to fully describe the workflow of the fuzzy-rule evaluations. The components of the Fuzzy Tuning framework are as follows:

- **Crisp Set**
  The Crisp Set models classical sets using k-cells and is used to define the underlying classical sets over which all fuzzy sets are defined. A k-cell is a hyperrectangle in the k-dimensional space constructed from the Cartesian product of k intervals $I = I_1 \times I_2 \times \ldots \times I_k$ where $I_i = [x_{low}, x_{high}] \subset \mathbb{R}$ is an interval in the real numbers. This cell defines the parameter space of the input variables, and its boundaries are later used for the numerical defuzzification step.
- **Fuzzy Set**
  As mentioned previously, fuzzy sets consist of a membership function assigning a degree of membership to each element of the Crisp Set. For the implementation in C++, we distinguish between two types of membership functions: The `BaseMembershipFunction` and the `CompositeMembershipFunction`. The `BaseMembershipFunction` acts as the standard membership function and directly assigns the degree of membership values to all continuous input values. It has the following signature: $f : \mathbb{R} \to [0, 1]$. Examples of this type of membership function are the trapezoid, sigmoid, and Gaussian functions described in the previous chapter.
  The `CompositeMembershipFunctions` implement the logical operations. They act as links between existing fuzzy sets to create more complex ones. This helps to split up the complex fuzzy sets of the rule base into smaller, more manageable parts. Internally, this way of combining fuzzy sets builds a tree structure where the leave nodes can directly calculate membership value based on `BaseMembershipFunctions`, and the inner nodes combine those membership values and pass them up to their parents. This way of defining fuzzy sets is natural and allows for efficient membership evaluation.

Figure 3.1 shows how complex fuzzy sets can be constructed from simpler fuzzy sets using the `CompositeMembershipFunctions`. Each fuzzy set additionally provides a method to calculate the defuzzified value.

- **Linguistic Variable**
  Linguistic variables act as a container for the fuzzy sets. They group linguistic terms of the same concept and provide ways to access the fuzzy sets.
- **Fuzzy Rule**
  Fuzzy rules store an antecedent and a consequent fuzzy set. Additionally, they provide a method to evaluate the fuzzy set resulting from the rule activation based on the input variables.
- **Fuzzy Control System:** The Fuzzy Control System combines all the concepts described above to create a system that can evaluate a set of fuzzy rules and generate an output based on the input variables. Such a control system acts like a black box $f : \mathbb{R}^n \to \mathbb{R}$ that maps crisp input values to a crisp output value. Multiple such systems are used in later sections to implement the tuning strategy.

A simplified class diagram of the Fuzzy Tuning strategy can be seen in Figure 3.2.

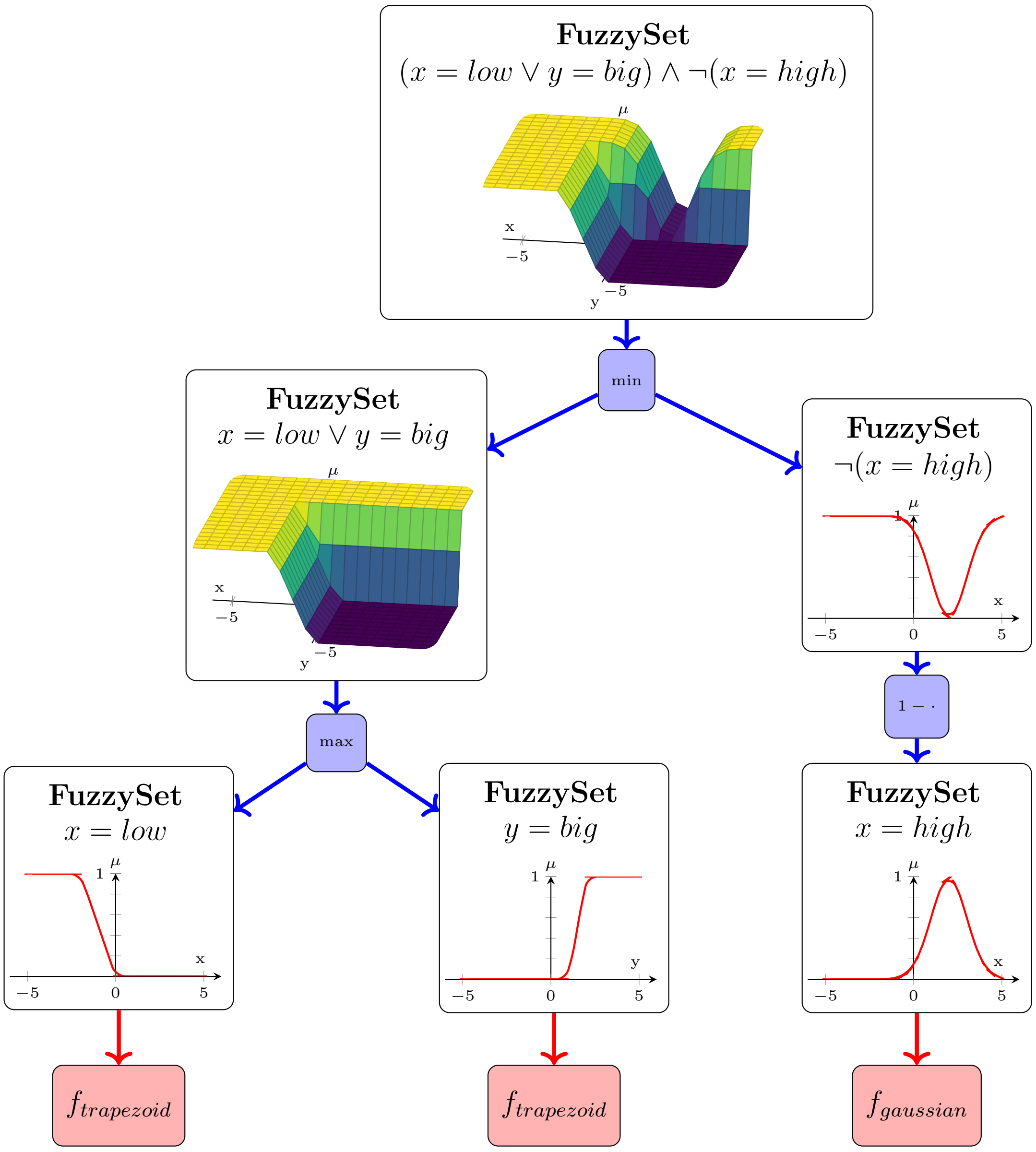

Figure 3.1: Recursive construction of a complex fuzzy set from simpler fuzzy sets. Using the linguistic variables x with the terms $\{low, high\}$ and y with the terms $\{big, small\}$ we can construct the fuzzy set $(x = low \lor y = big) \land \neg(x = high)$ by combining the fuzzy sets $x = low \lor y = big$ and $\neg(x = high)$. Those fuzzy sets are again constructed from the simpler fuzzy sets $x = low$, $y = big$ and $x = high$.
The fuzzy sets at the leaf level can be directly constructed using predefined `BaseMembershipFunctions` (e.g., trapezoid, sigmoid, gaussian ...) and provide the foundation for the more complex fuzzy sets All other fuzzy sets are created by combining other fuzzy sets using `CompositeMembershipFunctions`. The logical operators min, max, and $1 - \cdot$ are implemented this way, as they directly act on top of other fuzzy sets.

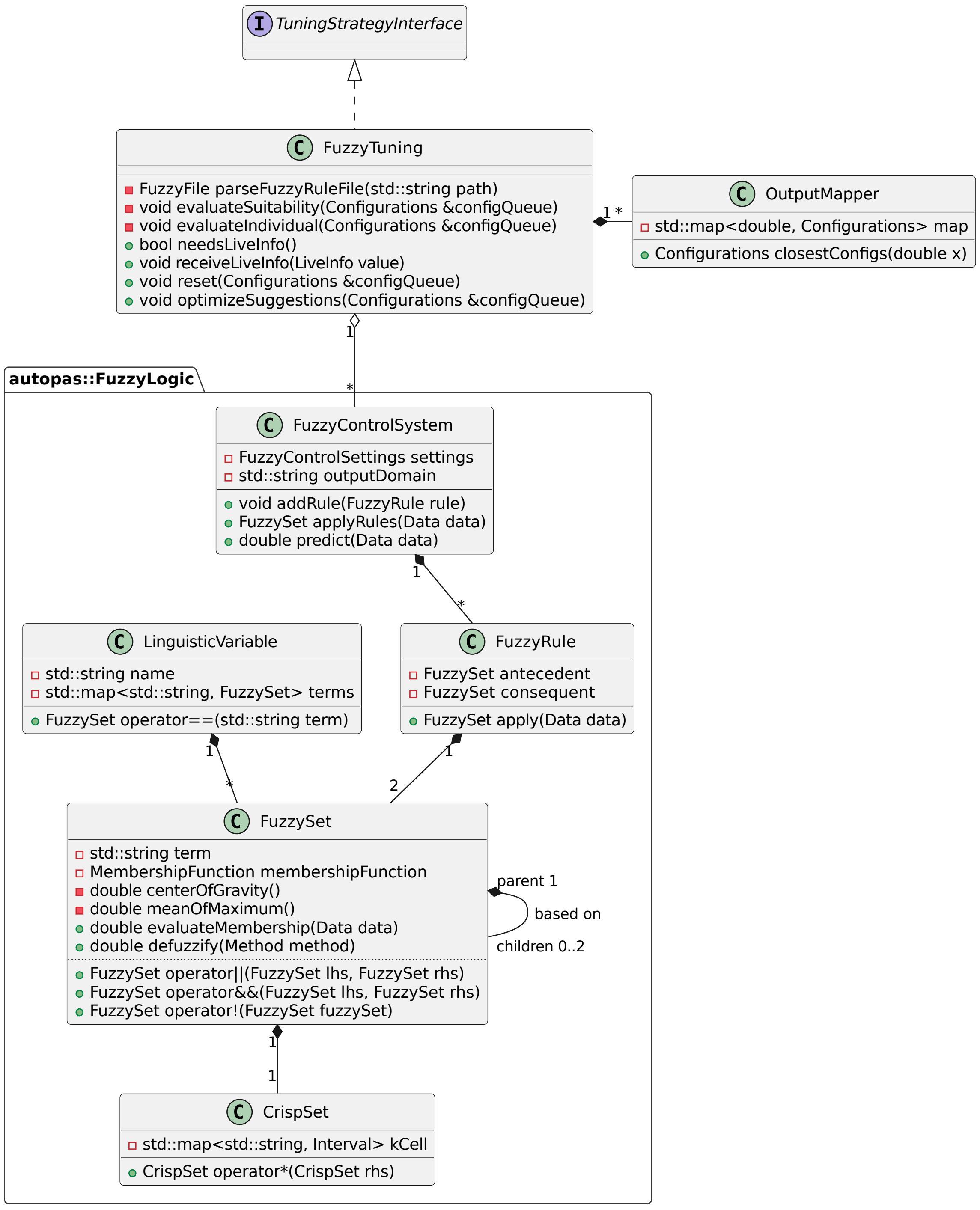
TuningStrategyInterface
FuzzyTuning
FuzzyFile parseFuzzyRuleFile(std::string path)
void evaluateSuitability(Configurations &configQueue)
void evaluateIndividual(Configurations &configQueue)
bool needsLiveInfo()
void receiveLiveInfo(LiveInfo value)
void reset(Configurations &configQueue)
void optimizeSuggestions(Configurations &configQueue)
OutputMapper
std::map<double, Configurations> map
Configurations closestConfigs(double x)
1
*
autopas::FuzzyLogic
FuzzyControlSystem
FuzzyControlSettings settings
std::string outputDomain
void addRule(FuzzyRule rule)
FuzzySet applyRules(Data data)
double predict(Data data)
LinguisticVariable
std::string name
std::map<std::string, FuzzySet> terms
FuzzySet operator==(std::string term)
FuzzyRule
FuzzySet antecedent
FuzzySet consequent
FuzzySet apply(Data data)
2
FuzzySet
std::string term
MembershipFunction membershipFunction
double centerOfGravity()
double meanOfMaximum()
double evaluateMembership(Data data)
double defuzzify(Method method)
FuzzySet operator||(FuzzySet lhs, FuzzySet rhs)
FuzzySet operator&&(FuzzySet lhs, FuzzySet rhs)
FuzzySet operator!(FuzzySet fuzzySet)
parent 1
based on
children 0..2
CrispSet
std::map<std::string, Interval> kCell
CrispSet operator*(CrispSet rhs)

Figure 3.2: Simplified class diagram of the Fuzzy Tuning strategy. There is a clear separation between implementing the Fuzzy Logic framework and the tuning strategy. This allows for an easy reuse of the Fuzzy Logic framework in other parts of AutoPas if desired.

3.2 Rule Parser

The Rule Parser is responsible for parsing the rule base supplied by the user and converting it into the internal representation used by the Fuzzy Tuning framework. It uses the ANTLR4[1] parser generator to create a parser for a domain-specific language tailored to the needs of the Fuzzy Tuning. The language is inspired by common standards such as the Fuzzy Control Language (FCL)[2] but is designed to be more lightweight and directly incorporate aspects of the AutoPas simulation. The parsed rules are transformed into the internal representation by a visitor pattern that traverses the parse tree generated by ANTLR4 and internally builds the corresponding object hierarchy.

3.3 Tuning Strategy

The Tuning Strategy implements the interface between the Fuzzy Tuning framework and the AutoPas simulation and is responsible for updating the configuration queue of configurations to be tested next.

It does this by evaluating all fuzzy systems present in the rule file with the current information on the simulation. It primarily used *LiveInformation* data, containing summary statistics about various aspects of the current simulation state. Possible parameters include the total number of particles, the average density, or the average homogeneity of the particle distribution. Each evaluation of a Fuzzy Control System yields a single numeric value, which is then passed on to the `OutputMapper`, which determines the resulting configuration. Such a mapping is necessary because the output of the Fuzzy Control System is a continuous value, while the configuration space of AutoPas is discrete, and we need to define mappings between the two.

This method of assigning concrete configurations to the continuous output space of the Fuzzy Control Systems is inspired by Mohammed et al. [MKEC22]'s work on scheduling algorithms. Internally, the `OutputMapper` stores an ideal location for each available configuration and determines the configuration closest to the predicted value.

The resulting list of configurations predicted by the Fuzzy Tuning Strategy is then used to update AutoPas's configuration queue. In the following iterations, all of those configurations are used to run a few steps of the simulation, and the configuration with the best performance is then used for the following simulation phase.

Currently, two different modes of interpreting the rules are implemented:

3.3.1 Component Tuning Approach

The component Tuning Approach expects a single Fuzzy Control System for each tunable parameter. All those Fuzz Systems should then attempt to predict the best value of their parameter independent of the other parameters. This approach requires the rule file to only define $\#Parameters$ different Fuzzy Control Systems, which makes it easier to maintain and understand. An obvious drawback of this method is the independence assumption between the parameters, which might not hold in practice.

[1]https://www.antlr.org/
[2]https://www.fuzzylite.com/

Another problem of this approach lies in the defuzzification step. As this method relies on defining a single system for all parameter values, there must be a *ranking* of all those values on a single numeric scale. Such a ranking is problematic, as most tunable values are nominal and do not have a natural order. To circumvent this problem, we can use a defuzzification method that does not perform interpolations between the values. Using such a method, the placement of the linguistic terms can be arbitrary. One such method is MOM. It selects the mean of all x-values for which the membership function is maximal. When using Gaussian-shaped membership functions, this method will only ever return the mean of the Gaussian with the highest activation. The OutputMapper can directly take these values and assign them to the corresponding nominal value of the tunable parameter. All predicted values are then used to filter the configuration queue, excluding all configurations that do not match the predicted values.

Figure 3.3 shows a schematic of the Component Tuning Approach.

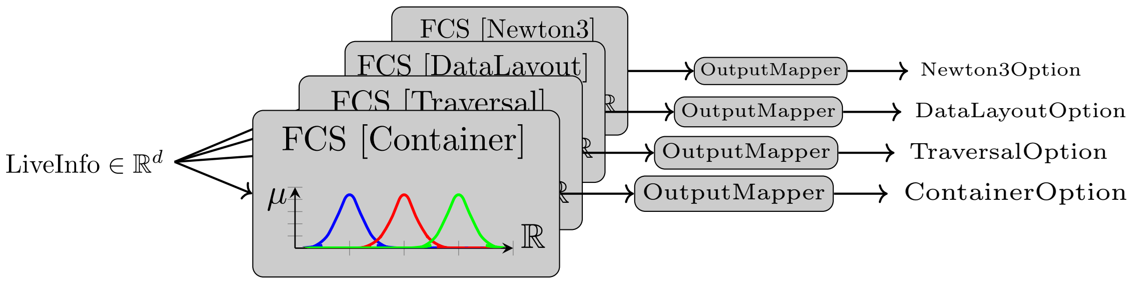

Figure 3.3: Example Visualization of the fuzzy control systems for the Component Tuning Approach. The parameters `Container`, `Traversal`, `DataLayout`, and `Newton3` are tuned independently. The OutputMapper assigns the predicted values to the corresponding nominal values of the parameters.

3.3.2 Suitability Tuning Approach

The Suitability Approach differs from the Component Tuning Approach in that it utilizes $\#Container\_options \cdot \#Traversal\_options \cdot \#DataLayout\_options \cdot \#Newton3\_options$ different Fuzzy Control Systems, one for each possible combination of those parameters. Consequently, there are way more Fuzzy Control Systems to evaluate. The advantage of this approach is that there is no need to rank the output values, and one can utilize the power of interpolation between different predictions by using the COG defuzzification method as the predicted suitabilities have a natural order. Furthermore, dependencies and incompatibilities between the parameters can be modeled accurately. The downside is the increased complexity of the rule file, which quickly becomes infeasible to maintain by hand. Surprisingly, the cost of evaluating all those Fuzzy Control Systems is negligible compared to the overhead of other tuning strategies.

Each System is responsible for predicting the suitability of a specific configuration, ranging from 0 (not suitable) to 100% (perfectly suitable). After applying the OutputMapping, the Strategy selects all configurations performing within a certain threshold of the best configuration and rewrites the configuration queue accordingly. Figure 3.4 shows a schematic of the Suitability Tuning Approach.

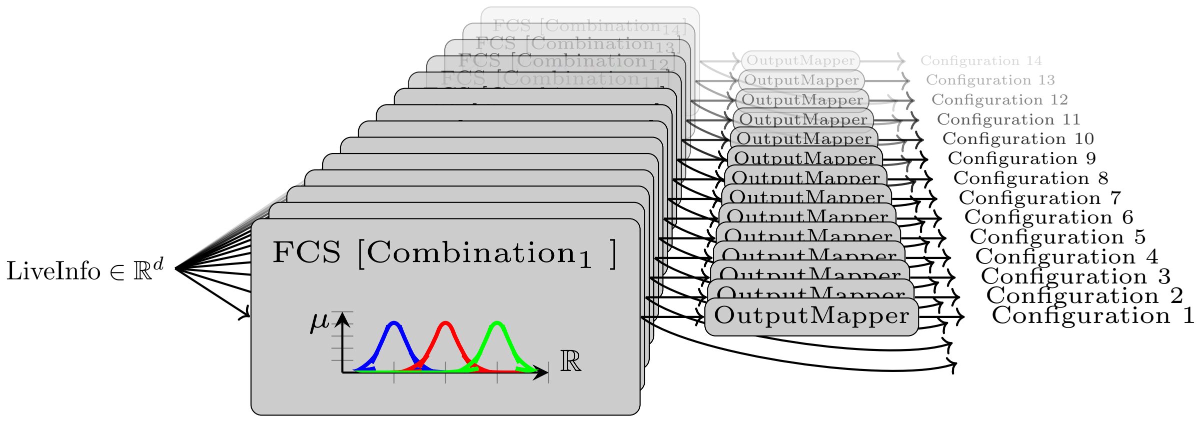

Figure 3.4: Example Visualization of the fuzzy control systems for the Suitability Tuning Approach. Each Fuzzy Control System is responsible for predicting the suitability of a specific configuration. The suitability and the mapped configuration are passed on to the Strategy, which only considers highly suitable configurations for the next simulation steps.

List of Figures

List of Tables

Listings

Bibliography

[BMK96] Bernadette Bouchon-Meunier and Vladik Kreinovich. Axiomatic description of implication leads to a classical formula with logical modifiers: (in particular, mamdani's choice of "and" as implication is not so weird after all). 1996.

[BPR+16] Tobias Brink, Martin Peterlechner, Harald Rösner, Karsten Albe, and Gerhard Wilde. Influence of crystalline nanoprecipitates on shear-band propagation in cu-zr-based metallic glasses. *Phys. Rev. Appl.*, 5:054005, May 2016.

[CBMO06] Keeley Crockett, Zuhair Bandar, David Mclean, and James O'Shea. On constructing a fuzzy inference framework using crisp decision trees. *Fuzzy Sets and Systems*, 157(21):2809–2832, 2006.

[GSBN21] Fabio Alexander Gratl, Steffen Seckler, Hans-Joachim Bungartz, and Philipp Neumann. N ways to simulate short-range particle systems: Automated algorithm selection with the node-level library autopas. *Computer Physics Communications*, 273:108262, 2021.

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

[LM15] Benedict Leimkuhler and Charles Matthews. *Molecular Dynamics: With Deterministic and Stochastic Numerical Methods*. Interdisciplinary Applied Mathematics. Springer, May 2015.

[MKEC22] Ali Mohammed, Jonas H. Müller Korndörfer, Ahmed Eleliemy, and Florina M. Ciorba. Automated scheduling algorithm selection and chunk parameter calculation in openmp. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4383–4394, 2022.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[NGNB23] Samuel James Newcome, Fabio Alexander Gratl, Philipp Neumann, and Hans-Joachim Bungartz. Towards auto-tuning multi-site molecular dynamics simulations with autopas. *Journal of Computational and Applied Mathematics*, 433:115278, 2023.

[PS17] Juan R. Perilla and Klaus Schulten. Physical properties of the hiv-1 capsid from all-atom molecular dynamics simulations. *Nature Communications*, 8(1):15959, 2017.

[SGH$^+$21] Steffen Seckler, Fabio Gratl, Matthias Heinen, Jadran Vrabec, Hans-Joachim Bungartz, and Philipp Neumann. Autopas in ls1 mardyn: Massively parallel particle simulations with node-level auto-tuning. *Journal of Computational Science*, 50:101296, 2021.

[VBC08] G. Viccione, V. Bovolin, and E. Pugliese Carratelli. Defining and optimizing algorithms for neighbouring particle identification in sph fluid simulations. *International Journal for Numerical Methods in Fluids*, 58(6):625–638, 2008.