Algorithm Selection and Auto-Tuning in AutoPas

Manuel Lerchner
Technical University of Munich
Munich, Germany

***Abstract*—Molecular dynamics (MD) simulations face significant computational challenges that require highly optimized simulation engines to deal with the enormous number of particles present in modern simulations. Naturally researchers have put a lot of effort into developing algorithms and frameworks that can efficiently simulate these systems. This paper examines the auto-tuning capabilities of AutoPas, a modern MD framework, and provides a comparative analysis with other prominent MD engines such as GROMACS and LAMMPS. We analyze the approaches to static and dynamic optimization and evaluate their effectiveness in various simulation scenarios. Furthermore, we investigate a possible improvement to the auto-tuning capabilities of AutoPas by introducing an early stopping mechanism to reduce the overhead the parameter space exploration.**

conclution

***Index Terms*—molecular dynamics, auto-tuning, algorithm selection, performance optimization, GROMACS, LAMMPS**

I. Introduction

Molecular dynamics simulations represent a computational cornerstone in various scientific fields, from materials science to biochemistry. To deliver accurate results, these simulations typically make use of complex, and computationally intensive interaction-models acting on enourmous number of particles. For simulation engines to be practical, they must be highly optimized to handle the computational load efficiently and utilize available resources effectively.

Prominent optimization techniques used in modern molecular dynamics (MD) engines fall into two main categories: static and dynamic optimization. Static optimizations rely on predefined configurations and performance models, often fine-tuned for specific hardware architectures. These optimizations include strategies like memory layout optimization, vectorization, and architecture-specific instruction use (e.g., SIMD) .

Efficient Computation and Optimization Techniques for Molecular Dynamics Simulations

Modern compiler frameworks such as Kokkos and SYCL further abstract hardware-specific optimizations, enabling more portable code across different hardware platforms (e.g., GPUs, CPUs). Kokkos provides a performance-portable parallel programming model that supports diverse high-performance computing environments, while SYCL, a standard by the Khronos Group, facilitates single-source C++ for heterogeneous platform .

In contrast, dynamic optimizations adjust parameters based on the current simulation state and the actual hardware performance. Unlike static optimizations, which are set before the simulation begins, dynamic optimizations allow for adjustments throughout the simulation. This approach enables MD engines to periodically measure and respond to actual performance, optimizing parameters like load balancing, cache locality, and communication patterns to improve efficiency under complex and possibly changing conditions .

In particular, we will focus on the auto-tuning capabilities of AutoPas, a modern MD framework that focuses on dynamic optimization techniques to achieve high performance in complex and possibly changing simulation scenarios. We will compare AutoPas's auto-tuning capabilities with other prominent MD engines, such as GROMACS and LAMMPS, to evaluate their effectiveness in various simulation scenarios. We will also investigate a possible improvement to the auto-tuning capabilities of AutoPas by introducing an early stopping mechanism to reduce the overhead of parameter space exploration.

CITE: Studies on dynamic tuning in MD simulations or load balancing techniques

II. AutoPas

AutoPas was developed on the basis of creating an efficient particle / N-Body simulation enginge applicable to a wide range of applications [**?**]. To support these various simulations, not just limited to molecular dynamics, AutoPas is build on a modular software architecture that allows for different algorithms and data structures to be used (mostly) interchangeably in the underlying simulation engine. AutoPas acts as a middleware between the simulation code provided by the user and various implementations of algorithms and data structures, which are chosen dynamically based on performance criteria.

A. Algorithm Library

All different algorithmic implementations for solving N-Body problems are part of the so called *Algorithm Library* of AutoPas. The *Algorithm Library* contains different implementations for certain key aspects of the simulation, such as neighbor identification, traversal patterns, and memory layouts.

Together they form a so called *Configuration* fully describing the internal implementation of the engine. Such a configuration is a 6-tuple consisting of implementations for: *Container*, *Traversal*, *Load Estimator*, *Data Layout*, *Newton 3*, and *Cell Size Factor*.

An obvious benefit of this modular approach is the ability to easily swap out implementations for certain aspects of the simulation, without having to change the entire codebase. This allows for easy experimentation with different implementations and the ability to quickly adapt to new hardware or simulation scenarios.

Another benefit of this modular approach is presented with the ever growing configuration library of AutoPas. As the

library is constantly updated and new implementations targeting specific hardware or simulation scenarios are added, it is possible to easily test the feasibility of older implementations under new hardware [**?**].

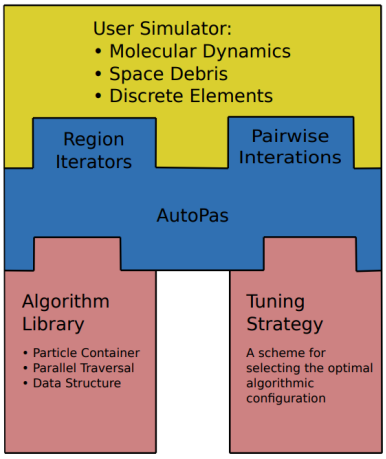

Fig. 1: AutoPas Library Structure [**?**]

B. Auto-Tuning Framework

Manually selecting the best implementations for each tunable parameter is a daunting task, and would require extensive domain knowledge difficult to acquire, and maintain under the constantly changing software and hardware landscape. To address this issue, AutoPas performs automated algorithm selection to maximize certain performance metrics, such as simulation speed or energy efficiency [**?**]. The auto-tuning framework of AutoPas periodically initiates so called *tuning-phases* in which it measures and evaluates promising[1] configurations in order to determine the best configuration for the current simulation state [**?**]. The winning configuration is then used until the next tuning phase is initiated.

The key to efficient tuning phases is the ability to efficiently determine promising configurations. Just using the naive approach of evaluating all possible configurations turns out to be infeasible in practice, as many of the naively evaluated configurations turn out to be orders of magnitude slower than the best known configuration, thus causing a drastic increasing of the total simulation time [**?**] [**?**]. As AutoPas is developed further, and new implementations are added to the algorithm portfolio, the number of possible configurations will steadily increase, further exacerbating the problem of naively evaluating all configurations.

AutoPas attempts to mitigate this problem by using so called *TuningStrategies* which prune the search space of possible configurations based on certain criteria [**?**]. A tuning strategy is tasked with the difficult job pruning enough configurations to make the tuning phase feasible, but not too many, as to not miss the optimal configuration. Tuning strategies need to balance the trade-off between potentially finding a better configuration and the cost of potentially encountering worse configurations during exploration [**?**].

[1]Promising configurations are suggested by so called TuningStragies

C. Static Optimization Techniques

Appart from the static optimization already performed by modern compilers, AutoPas is capable of performing basic tuning at compile time. As AutoPas hower primarily aims at dynamic optimization, the static optimization capabilities are limited to a set of compile-time flags that can be used to statically enable or disable certain features of the simulation engine, such as SIMD instructions, MPI support, or Auto-Vectorization [**?**].

D. Kokkos Integration

There are however attemps to integrate more advanced static optimization techniques into the AutoPas framework. One such attempt is the integration of the Kokkos library into AutoPas aiming to provide performance portability across different hardware platforms, and be able to run existing algorithms on GPGPUs [**?**].

Kokkos is a performance-portable parallel programming model supporting diverse high-performance computing environments [**?**]. Kokkos provides high-level C++ abstractions for parallel execution and memory management, enabling developers to write performance-portable code that can be compiled for different hardware architectures, including GPUs, Intel Xeon Phis, or many-core CPUs [**?**].

The integration is still in an experimental phase and not yet capable of targeting systems other than conventional multi-core CPUs [**?**]. A full integration of Kokkos into AutoPas could provide a significant performance boost, especially as classical MD Simulation engines such as LAMMPS show benefits from using Kokkos, particularly on large simulations on GPUs [**?**].

III. Early Stopping Optimization

As identified by [**?**] [**?**] [**?**], overhead caused by evaluating subobtimal configurations during the tuning phase can be a significant bottleneck in the performance of the AutoPas framework. Even though the tuning strategies employed by AutoPas are highly efficient, they still tend to suggest a large number of configurations that are not optimal. Rule driven tuning strategies such as *RuleBasedTuning* and *FuzzyTuning* can mitigate this problem to some extent, but due to the complexity of particle simulations, those rule-bases are expected to be highly incomplete.

All mentioned sources suggest that some form of *early stopping* mechanism could be beneficial for the AutoPas framework. The primarily goal of such a mechanism would be to detect tuning-iterations that take much longer than the currently best known configuration and stop the evaluation of those configurations early. There are two approaches to this problem:

- **Stopping Further Samples:** Currently AutoPas supports testing a certain parameter configuration multiple times to get a more accurate and stable performance measurement. A simple way to implement early stopping would be to stop the evaluation of further samples of a configuration if the performance of a sample is significantly worse than the best known configuration. The implementation of this approach would be relatively simple, but it is fairly course grained as all started samples would still be evaluated fully.
- **Interrupting the Evaluation:** A more fine grained approach as proposed in [**?**] would be to interrupt the evaluation of a configuration as soon as it is clear that the performance is significantly worse than the best known configuration. This is way more difficult to implement, as it would require the ability to interrupt the evaluation of a configuration at any point in time. Especially in a MPI environment with multiple nodes, aborting and resetting the simulation to a consistent state would require a lot of synchronization and communication work.

Both mentioned approaches require a user defined threshold for the maximum allowed slowdown of a configuration before it should be stopped. This threshold will be determined empirically in **??**.

To get a first impression of the potential benefits of an early stopping mechanism, we implemented the first approach in the AutoPas framework. The changes to the existing codebase are minimal and the early stopping mechanism can be implemented using existing functionality. **??** shows the implementation of the early stopping mechanism in the AutoPas framework.

A. Implementation

The early stopping mechanism is triggered by the new `CheckEarlyStopping` function, which is called after the performance of a configuration has been measured. The function compares the performance of the current configuration to the best known performance encountered in the current tuning phase. If the performance of the current configuration is significantly worse than the best known performance, the `abort` flag is set to `true`. The existing `GetNextConfiguration` function is modified slightky to trigger a re-tuning of the configuration if the `abort` flag is set. The `abort` flag is reset during re-tuning.

Algorithm 1 Early Stopping Algorithm in AutoPas

```
1:  procedure CheckEarlyStopping(performance)
2:      fastestTime ← min(fastestTime, performance)
3:      slowdownFactor ← performance / fastestTime
4:      if slowdownFactor > maxAllowedSlowdown then
5:          abort ← true
6:      end if
7:  end procedure
8:  procedure GetNextConfiguration
9:      if not inTuningPhase then
10:         return (currentConfig, false)
11:     else if numSamples < maxSamples and not abort then
12:         return (currentConfig, true)
13:     else
14:         stillTuning ← tuneConfiguration( )
15:         return (newConfig, stillTuning)
16:     end if
17: end procedure
```

B. Evaluation

This section evaluates the performance of the early stopping mechanism described in **??**. The performance of the early stopping mechanism is evaluated for different values of the maximum allowed slowdown factor, in order to determine the optimal threshold for the early stopping mechanism.

All benchmarks are performed on the CoolMUC2 supercomputer and are repeated 3 times to account for statistical reference variance.

1) Exploding Liquid Simulation: The first benchmark is performed with the *Exploding Liquid* scenario present in the `md-flexible` framework. The simulation consists of 1764 initially close-packed particles that are simulated with a Lennard-Jones potential. During the simulation, the particles rapidly expand outwards and eventually hitting the simulation boundaries. The simulation is run with a single thread on a single node of the CoolMUC2 supercomputer.

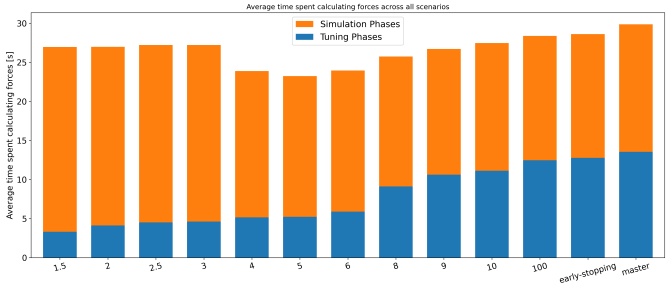

Fig. 2: Total Simulation Time for Exploding Liquid Simulation with Early Stopping divided in tuning and simulation phases. The total simulation time is minimal at a maximum allowed slowdown factor of ≈ 5.

Compared to FullSearch without early stopping, the early stopping mechanism can reduce the total simulation time from to seconds. This is a reduction of %.

X

2) Spinodal Decomposition Simulation: The second benchmark is performed with the *Spinodal Decomposition* scenario present in the `md-flexible` framework. The simulation consists of 100.000 particles that are simulated with a Lennard-Jones potential. The simulation is run with 14 threads on a single node of the CoolMUC2 supercomputer.

Y

Compared to FullSearch without early stopping, the early stopping mechanism can reduce the total simulation time from to seconds. This is a reduction of %.

Z

3) Early Stopping combined with Tuning Strategies:

Add Spinodal Decomposition Benchmark Results

IV. Analysis and Discussion

All evaluated benchmarks show that there exists just a slim range of optimal thresholds for the early stopping mechanism which actually reduce the total simulation time. This is expected, the two corner cases $maxAllowedSlowdown \to 1$ and $maxAllowedSlowdown \to \infty$ are expected to perform poorly. $maxAllowedSlowdown \to 1$ essentially results in tuning phases with just one sample per configuration, which is not enough to get a good estimate of the performance of a configuration and causes simulation phases with suboptimal configurations. On the other hand, $maxAllowedSlowdown \to \infty$ results in the early stopping mechanism never aborting a configuration, which is equivalent to always evaluating all samples of a configuration, even if the configuration is known to be suboptimal.

X

Consequently, the optimal threshold for the early stopping mechanism is a trade-off between the overhead of evaluating suboptimal configurations and the risk of missing optimal configurations due to noise in the performance measurements.

Y

From the executed benchmarks, we deduce that the optimal threshold for the early stopping mechanism is around for the *Exploding Liquid* scenario and around for the *Spinodal Decomposition* scenario, but it is expected that the optimal threshold is highly dependent on the specific simulation scenario and the environment the simulation is run in.

Z

todo

It is howerver noteworthy that the early stopping mechanism never caused a significant slowdown of the simulation, even if poor thresholds were chosen.

X

Y

We conclude that the (naive) early stopping mechanism is a valuable addition to the AutoPas framework, as it can reduce the total simulation time significantly without causing any significant slowdowns.

A. Future Work

As the current implementation is only capable of stopping further samples of a configuration, the next step would be to extend the early stopping mechanism to dynamically blacklist certain implementations.

A crude way to implement this would be to create a set of parameters most influential to the performance of the simulation (for example: {*Particle Container*, *Data Layout*}) and blacklist configurations that use a certain values if there is enough evidence that a configuration with those values is not a good choice for the current simulation scenario (potentialy if more than $X\%$ of samples using certain values are stopped early).

The parameter *Particle Container* should be included in the set of parameters to blacklist, as it is known to be highly influential to the performance of the simulation [**?**].

B. Feature Comparison of MD Engines

TABLE I: Feature Comparison of MD Engines

Feature	AutoPas	GROMACS	LAMMPS
Auto-tuning	✓	Partial	Partial
GPU Support	✓	✓	✓
Dynamic Load Balancing	✓	✓	✓

C. Strengths and Limitations

Comparative analysis reveals:

- Performance impact of different approaches
- Overhead considerations
- Scalability characteristics

D. Use Case Scenarios

Different engines excel in various scenarios:

- Large-scale simulations
- GPU-accelerated computations
- Memory-constrained environments

E. Demonstration of Benefits of AutoTuning

Even though AutoPas is designed to perform periodic auto-tuning, it often sufficient to just perform a single tuning phase at the beginning of the simulation, as many scenarios tend to behave fairly stable over time. Simulating an inhomogeneous scenario can sometimes benefit from re-tuning the configuration after a number of simulation steps, especially when using a MPI environment with multiple nodes. Inhomogeneous scenarios can cause the load to be distributed unevenly both across the nodes and across time steps, further increasing the potential benefits of periodically re-tuning the configuration on each node.

The currently provided example runs from `md-flexible` are not sufficient to demonstrate the benefits of periodic

re-tuning . More complex scenarios, most likely involving multiple nodes and a high number of particles, are required to demonstrate the benefits of periodic re-tuning.

V. State of the Art in MD Simulations

A. GROMACS

1) Static Optimization Techniques:

2) Dynamic Optimization Strategies:

B. LAMMPS

1) Static Optimization Techniques:

2) Dynamic Optimization Strategies:

Limitations of those Approaches

As mentioned earlier, both GROMACS and LAMMPS focus primarily on static optimization. While these techniques are heavily optimized, both engines are not capable of performing datastructure and algorithm changes at runtime. This limitation can lead to suboptimal performance in situations where the chosen implementation is not optimal. Those situations can also arise during the simulation, when the simulation state changes such that other datastructures or algorithms would be more efficient.

C. Performance Comparison

- AutoPas demonstrates superior performance in various scenarios.
- GROMACS and LAMMPS show competitive performance in specific use cases.
- Auto-tuning plays a crucial role in optimizing performance across different engines.

VI. Conclusion

A. Summary of Findings

This study provides a comprehensive comparison of auto-tuning approaches in modern MD engines, highlighting the unique advantages of AutoPas's implementation while acknowledging the strengths of established frameworks like GROMACS and LAMMPS.

B. Future Directions

Future research directions include:

- Further optimization of auto-tuning strategies
- Integration of machine learning techniques for performance prediction
- Collaboration between MD engine developers to share optimization strategies

[**?**]

cite data from ba

References

[1] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202–3216, 2014. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing.

[2] 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, 2022.

[3] 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.

[4] Ludwig Gärtner. Integrating kokkos into autopas for hardware agnostic particle simulations. Master's thesis, Technical University of Munich, Feb 2022.

[5] HobbyProgrammer. Skip (or even timeout) extremely long running iterations of configurations during tuning. https://github.com/AutoPas/AutoPas/issues/673, 2022. Accessed: 2024-11-06.

[6] Tobias Humig. Project report: Exploring performance modeling in autopas. Project report, Technical University of Munich, Oct 2023.

[7] LAMMPS. Speeding up lammps with kokkos. *LAMMPS Documentation*, 2024.

[8] Manuel Lerchner. Exploring fuzzy tuning technique for molecular dynamics simulations in autopas. Bachelor's thesis, Technical University of Munich, Aug 2024.

[9] Samuel James Newcome, Fabio Alexander Gratl, Philipp Neumann, and Hans-Joachim Bungartz. Towards the smarter tuning of molecular dynamics simulations. In *SIAM Conference on Computational Science and Engineering (CSE23)*. SIAM, Feb 2023.

[10] Nikola Plamenov Tchipev. *Algorithmic and Implementational Optimizations of Molecular Dynamics Simulations for Process Engineering*. PhD thesis, Technische Universität München, 2020.

CONTENTS