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.**

conclusion

***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 .

Fig. 1: AutoPas System Architecture

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. 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.

III. AutoPas Auto-Tuning Framework

A. System Architecture

The AutoPas framework implements a sophisticated auto-tuning system based on several key components:

- Container concepts for particle management
- Flexible traversal options
- Dynamic parameter space exploration

B. Auto-Tuning Implementation

The auto-tuning mechanism in AutoPas operates through:

1) Runtime performance measurement
2) Search space exploration
3) Adaptive decision-making

C. Analysis of Tuning Strategies

AutoPas employs several auto-tuning strategies:

- Static tuning for initial parameter selection
- Dynamic tuning for runtime adaptation
- Performance profiling for optimization

IV. Early Stopping Optimization

As identified by [2] [3] [4], 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 [3] 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 subsection IV-B.

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. Algorithm 1 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

procedure CHECKEARLYSTOPPING(performance)
  $fastestTime \leftarrow \min(fastestTime, performance)$
  $slowdownFactor \leftarrow \frac{performance}{fastestTime}$
  if $slowdownFactor > maxAllowedSlowdown$ **then**
    $abort \leftarrow true$
  end if
end procedure
procedure GETNEXTCONFIGURATION
  if not $inTuningPhase$ **then**
    return $(currentConfig, false)$
  else if $numSamples < maxSamples$ **and** not $abort$ **then**
    return $(currentConfig, true)$
  else
    $stillTuning \leftarrow$ TUNECONFIGURATION( )
    return $(newConfig, stillTuning)$
  end if
end procedure

B. Evaluation

This section evaluates the performance of the early stopping mechanism described in Algorithm 1. 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 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.

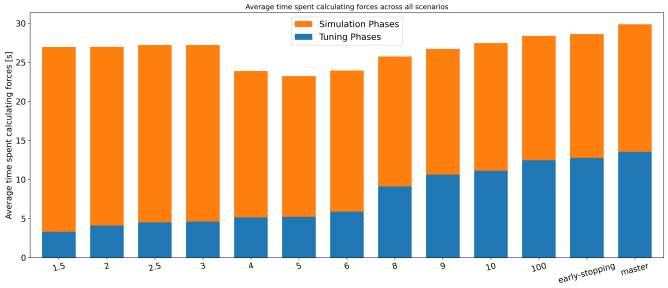

Fig. 2: Total Simulation Time for Exploding Liquid Simulation with Early Stopping divided in tuning and simulation phases. The total simulation time is minimal at a maximum allowed slowdown factor of ≈ 5.

2) Trade-offs and Considerations: While early stopping provides significant benefits, several factors require careful consideration:

- **Threshold Selection:** The performance threshold must balance between aggressive pruning and the risk of missing optimal configurations.
- **Workload Sensitivity:** Different simulation scenarios may require different early stopping strategies. AutoPas adapts its thresholds based on workload characteristics.
- **Cold-Start Handling:** Special considerations are needed during the initial tuning phase when limited performance data is available.

Early stopping represents a crucial optimization in AutoPas's tuning strategy, effectively addressing the overhead challenges inherent in auto-tuning systems while maintaining robust performance optimization capabilities.

TABLE I: Feature Comparison of MD Engines

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

V. Analysis and Discussion

A. Feature Comparison of MD Engines

B. Strengths and Limitations

Comparative analysis reveals:

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

C. Use Case Scenarios

Different engines excel in various scenarios:

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

D. Demonstration of Benefits of AutoTuning

- Performance improvement
- Scalability
- Adaptability

E. 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

[1]

References

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

[2] 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.

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

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

CONTENTS