Contents

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

5 Results

The data collected as part of the evaluation of this work is presented and discussed in this section. The hardware and software setup used are given in Section 5.1, as to allow for reproducibility of our results. Sections 5.1 and 5.2 discuss some implementation choices based on collected data. Finally, the benchmark results for each scenario are presented in Section 5.4.

5.1 Experimental Setup

The measurements collected for analysis in this chapter were obtained on the CoolMUC4 Linux-Cluster of the Leibniz-Rechenzentrum[1]. The nodes in the `cm4` cluster consist of processors in the Sapphire Rapids family (Intel® Xeon® Platinum 8480+) with 2.1 GiB of memory per logical CPU and 488 GiB per node [1]. For benchmarking purposes, the AutoPas library and `md-flexible` were compiled with Spack GCC 13.2.0 and Intel MPI 2021.12.0 on commit `46bb925c8c5827faee8691afefd9f714630f20f1`.

The scripts used to generate the Slurm jobs and configuration files can be found in the repository of this thesis[2].

5.2 Choice of Simulation Statistics

As referred to before in Section 3.1.2, there are several simulation statistics available upon which trigger strategies could be based. Even the iteration runtimes themselves are differentiated into multiple parameters: the time spent on computing interactions, traversing remainders and rebuilding neighbor lists.

In this thesis, we will consider the sum of these times with the exception of the rebuilding measurements. This choice can be justified by inspecting runtime data we collected: As shown in Figure 5.1, the rebuild times remain constant over all iterations simulated with a particular configuration. There inclusion therefore does not provide any new information, but rather smooths out the overall measurements and thus decreases the effectivity at which a scenario change can be detected.

[1] https://www.lrz.de/
[2] https://github.com/ladnik/bachelor-thesis

Additionally, in scenarios with a low average number of neighbors, the rebuilding of neighbor lists takes longer than the interaction computations. Considering that the rebuilding only happens in iterations that are a multiple of `rebuildFrequency`, this leads to stability problems in trigger strategies with a low number of samples, as the few rebuild iterations greatly outweigh all non-rebuild iterations.

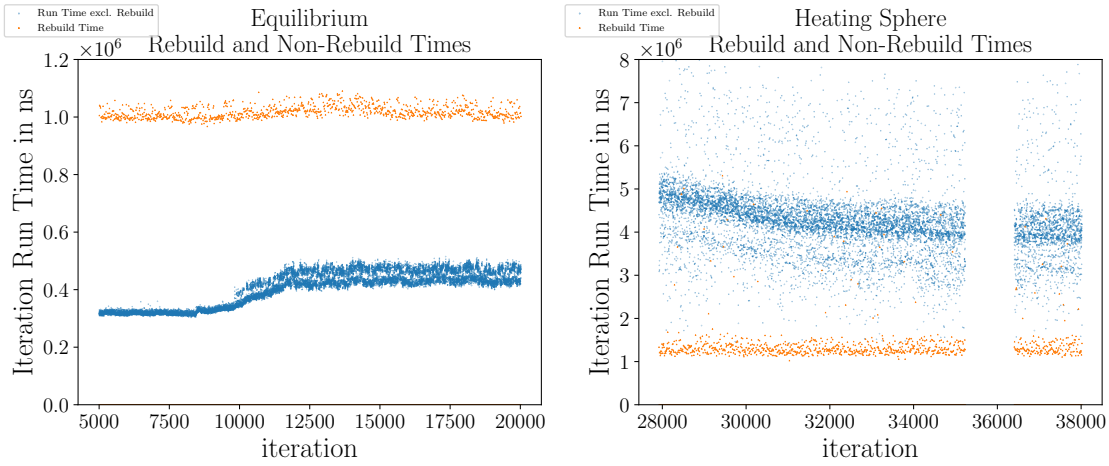


Figure 5.1: Rebuild and non-rebuild times in the equilibrium (left) and heating-sphere (right) scenario. [TODO]

5.3 Computational Overhead

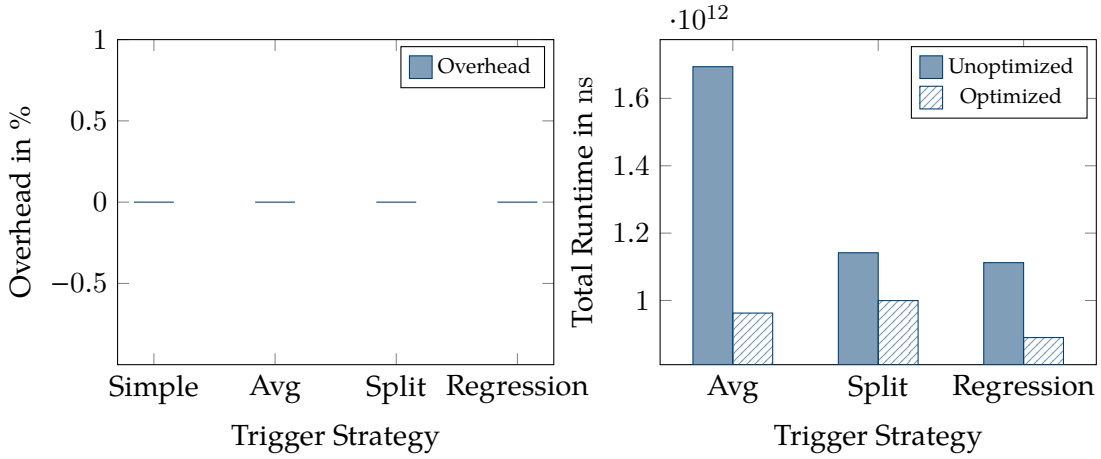
Our strategies analyze data at runtime and therefore ...

To quantify the overhead our strategies introduce, we compare runs with a single initial tuning phase. That way, we remove any changes in runtime that may occur due to different tuning phase initiation points of our strategies. To achieve this, we use a factor high enough, such that none of our strategies initiate a new tuning phase. For the baseline run, we use static tuning intervals with a `tuning-interval` set longer than the total number of iterations to simulate, which results in a single tuning phase starting in iteration 0. Figure 5.2a shows the overhead obtained that way for the equilibrium scenario with $\lambda = 50$ and $n = 500$.

To exemplify the importance of optimizing the trigger routines, Figure 5.2b illustrates the runtime differences between naive and optimized triggers in the equilibrium scenario. The naive version recalculates the average over all samples each iteration, whereas the optimized version uses a ring buffer and running summation to reduce computational cost. The speedup experienced is not only due to a lowering of computational overhead, but also due to less tuning iterations. This fact can be explained by the aforementioned self influence of the triggers: higher overhead might lead to higher fluctuation in iteration runtime which in turn leads to unstable trigger behavior, especially in the averaging trigger.

5.4 Benchmarking Results

The speedups presented in the following plots was computed by the formula given in (5.1), where t_{baseline} represents the runtime with tuning phases at fixed intervals and



(a) Overhead of the various trigger strategies. [TODO]

(b) Average Speedup between unoptimized and optimized runs.

Figure 5.2: TODO

t_{dynamic} the runtime of our implementation.

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

The blue bars in the graphs represent the runtime of that particular iteration. In the configuration plots, the colored background identifies the used configuration: same configurations map to the same color. The gaps in the plot are where tuning iterations have been logged – as their runtime is not relevant for the scenario change and would distort the actual runtime plot, they are not reported here. The red vertical lines indicate the start of a tuning phase.

5.4.1 Equilibrium

Speedup and Default Parameters

As can be seen in Figure 5.3, a trigger factor of $\lambda = 1.5$ leads to increased speedup compared to $\lambda = 1.25$ in nearly all triggering strategies. This is however mainly due to the nature of the equilibrium scenario: after the initial configuration selection, the optimal configuration is not expected to change. Therefore, not initiating any new tuning phases will lead to a decrease in total simulation runtime. That the speedup is indeed a result of the decreased number of tuning iterations can be verified by looking at the right-hand side plots; for the simple and averaging trigger it is most noticeable. Additionally, triggers with a larger sample size will typically trigger less frequently, as more of the variability in iteration runtime is smoothed out. For a too large number of samples, the speedup decreases however, as the computational overhead is directly proportional to the number of samples.

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

Selected Runs

Figure 5.4 shows two of the experimental runs in detail. On the left hand side, a `TimeBasedRegressionTrigger` with $\lambda = 1.25$, $n = 1000$ detects scenario change reliably. The

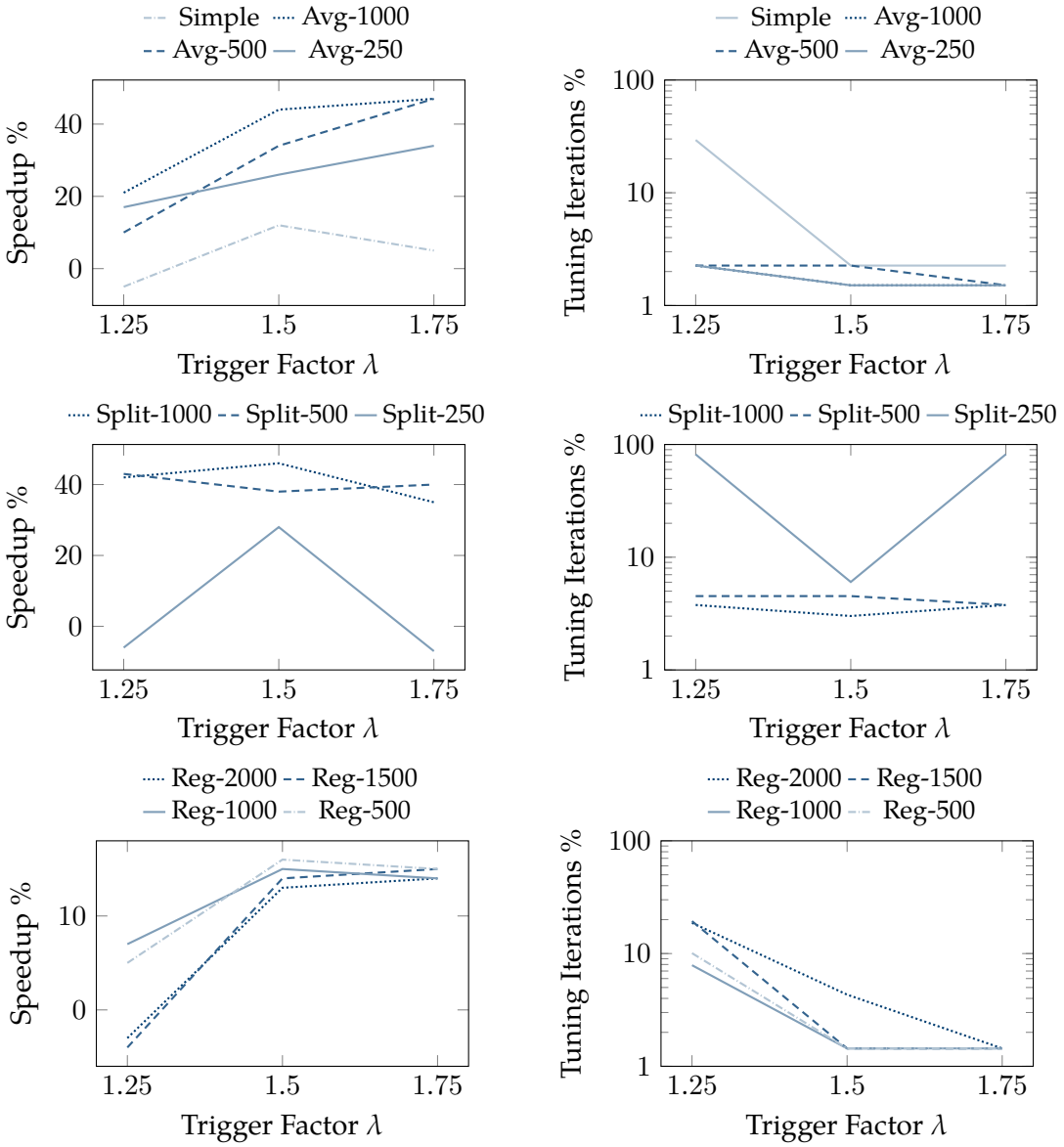


Figure 5.3: Trigger behavior in the equilibrium scenario, the numbers in the legends refer to the number of samples n considered. Note the logarithmic scale in the plots on the right hand side.

Trigger	Trigger factor λ	Number of samples n
`TimeBasedSimple`	1.5	–
`TimeBasedAverage`	1.75	500
`TimeBasedSplit`	1.5	1000
`TimeBasedRegression`	1.5	500

Table 5.1: Suggested default parameters for the equilibrium scenario.

(a) Good scenario change detection with the regression-based trigger.

Figure 5.4: Examples of trigger behavior in the equilibrium scenario.

first dynamically triggered tuning phase appears in iteration 9798, which is due to the sharp increase in iteration runtime. Afterwards, the iteration runtimes do not increase fast enough to warrant any new trigger phase. The next tuning phase is therefore only triggered because of outliers around iteration 80 000, particularly iteration 86 097 immediately before the trigger fires. Interestingly, even tough the scenario change wasn't detectible in the runtime itself, the next chosen configuration (`VLC-C01-NoN3L-AoS-CSF0.5`) performs better. The remaining tuning phases are again triggered by outliers.

On the right hand side...

Additional plots showing the behavior of various trigger strategies are included in the appendix.

Optimality

5.4.2 Exploding Liquid

Speedup and Default Parameters

Selected Runs

Optimality

5.4.3 Heating Sphere

Speedup and Default Parameters

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

Trigger	Trigger factor λ	Number of samples n
`TimeBasedSimple`	–	–
`TimeBasedAverage`	1.75	500
`TimeBasedSplit`	1.5	500
`TimeBasedRegression`	–	–

Table 5.2: Suggested default parameters for the heating sphere scenario.

Selected Runs

Optimality

5.4.4 Comparison

In the previous sections, the various trigger strategies were analyzed within each scenario. To better visualize the results of our benchmarks, **??** lays out the best speedup achieved, grouped by scenario.

5.5 Hybrid Triggers

As was seen by the benchmark results, time-based approaches are not suitable for all scenarios. In these scenarios, iteration runtimes alone might not be a good enough indicator for scenario change. As referred to before in Section 3.1.2, AutoPas provides additional live simulation statistics through its `liveinfo` interface. These could be used

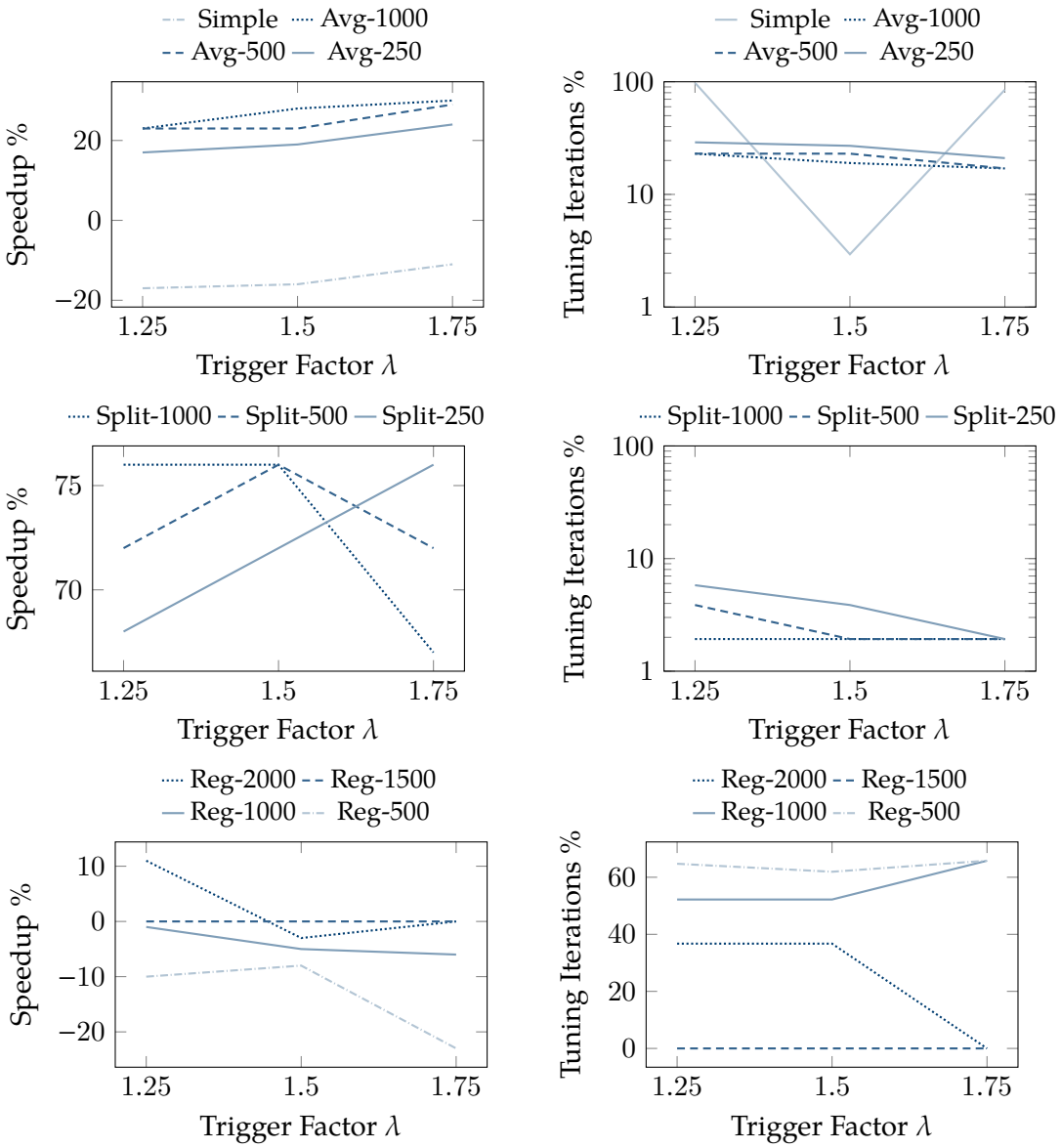


Figure 5.5: Trigger behavior in the heating-sphere scenario, the numbers in the legends refer to the number of samples n considered. Note the logarithmic scale in the plots on the right hand side.

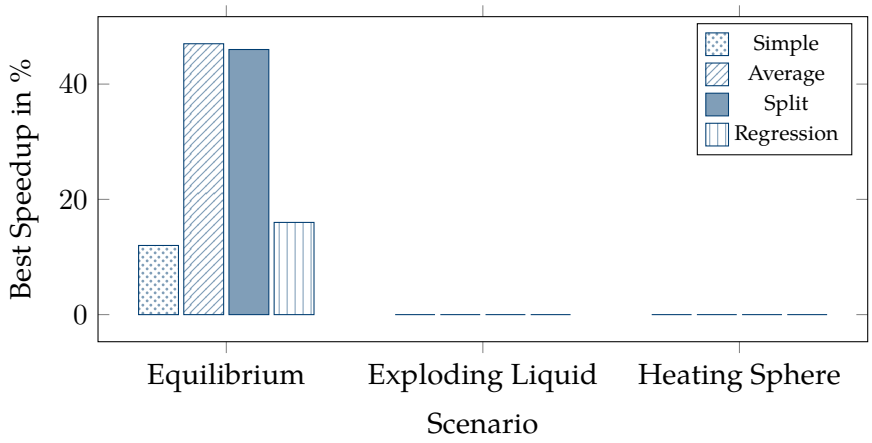


Figure 5.6: Average Speedup between unoptimized and optimized runs for the `TimeBasedAverage`, `TimeBasedSplit` and `TimeBasedRegression` strategies.

in combination with iteration runtimes to find better strategies in detecting scenario change. As a motivation of this approach, Figure 5.7 shows an exemplary run in the heating-sphere scenario. The initial optimal configuration is `VL-List_Iter-NoN3L-AoS`, later on it changes to `LC-C04-N3L-AoS-CSF1`. [2] The iteration runtime does not change, even tough a better configuration is available; the `maxDensity` statistics however, show clear indication of the shift towards a different simulation state.

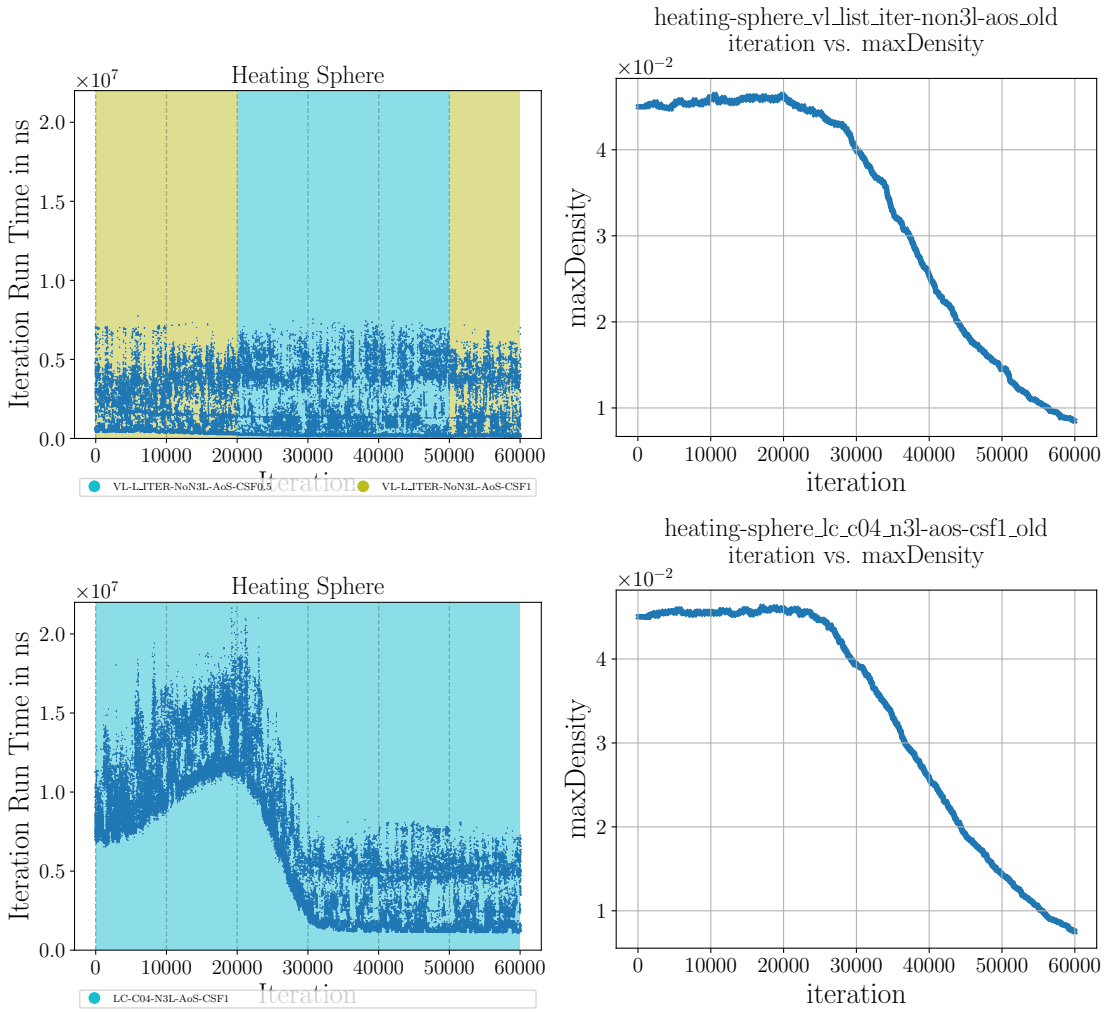


Figure 5.7: Iteration runtime (left) and the maximum densities of particles per cell (right). The top row shows a run with the single configuration `VL-List_Iter-NoN3L-AoS`, the bottom row shows `LC-C04-N3L-AoS-CSF1`

6 Conclusion

In this thesis, four new methods of dynamically initiating new tuning phases in AutoPas were introduced. ...

As shown in Chapter 5, our approach is successful in specific scenarios. Especially in settings in which the optimal configuration does not change rapidly, our method reduces the amount of spent in tuning phases without any significant decrease in the optimality of the chosen configurations.

As we focused on runtime based strategies, scenarios with high variability in iteration runtimes are ...

However, as new tuning strategies are introduced, the dynamic initiation of tuning intervals will likely become irrelevant for single-node applications. For example, the use of tuning strategies based on machine learning leads to very cheap tuning phases [2], which in turn significantly diminishes the speedups observed when comparing to tuning with `full-search`. One application in which a dynamic approach as ours might still be useful is in shared memory setups: As each node runs their own autotuner instance, they can trigger tuning phases independent from each other. In scenarios like exploding-liquid this could still be advantageous.

Possible future work could explore the analysis of `LiveInfo` parameters, the introduction of novel trigger strategies providing more stability, ...

This page is intentionally left blank.

Bibliography

[1] Leibniz Supercomputing Centre. *Job Processing on the Linux-Cluster*. Accessed: 2025-09-01. 2025. URL: https://doku.lrz.de/job-processing-on-the-linux-cluster-10745970.html.

[2] Samuel James Newcome et al. "Algorithm Selection in Short-Range Molecular Dynamics Simulations". In: (May 2025). DOI: 10.48550/ARXIV.2505.03438. arXiv: 2505.03438 [cs.CE].

This page is intentionally left blank.