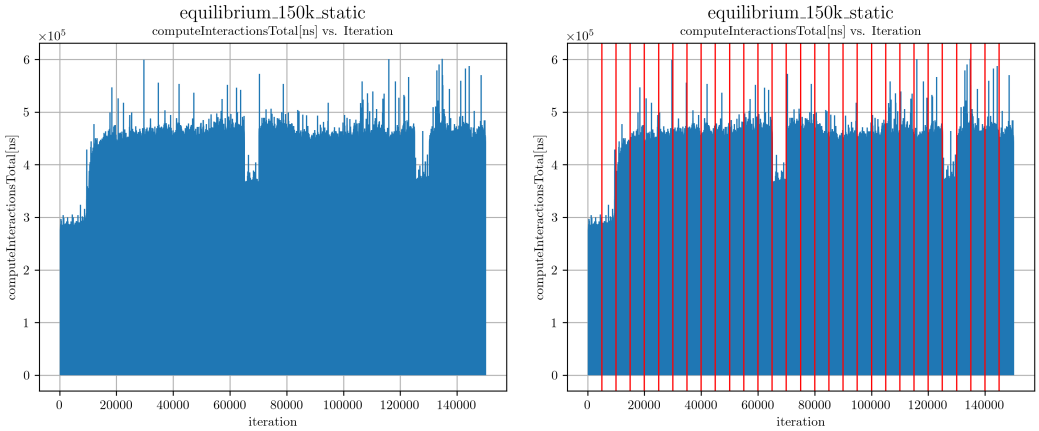
Overview of preliminary AutoPas Experiments

Niklas Ladurner

1 Static tuning intervals

The blue bars in the graph represent the runtime of that particular iteration. In the configuration plots, the colored background identifies the used configuration: same color means same configuration. Keep in mind, that the mapping configuration $\mapsto$ color may change between plots. 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, I left them out. The red vertical lines indicate the start of a tuning phase.

1.1 Equilibrium



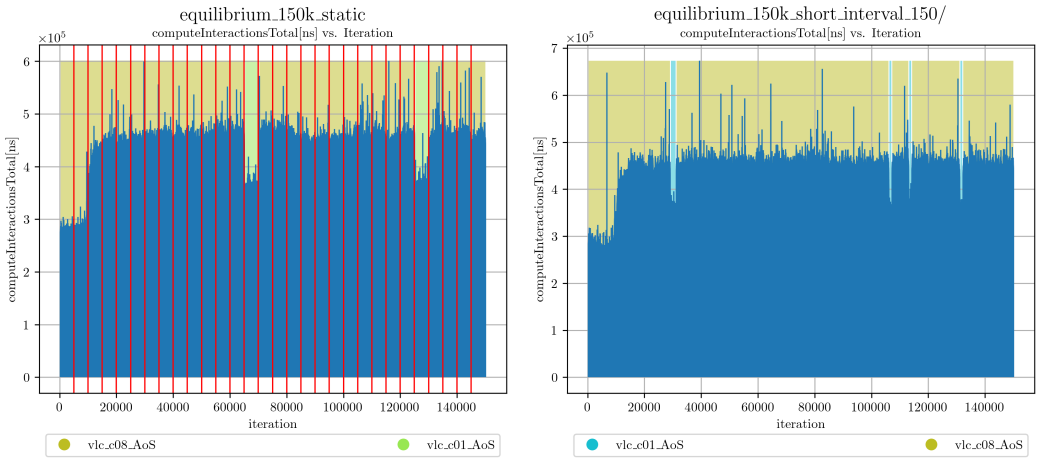
(a) Runtime

(b) Beginning of tuning intervals marked in red

Figure 1: Runtime vs. iteration for the equilibrium scenario

Running the equilibrium scenario on CoolMUC4, the runtime stays fairly consistent, except for two intervals in which we change to a different config, which leads to better runtimes (see 1). Apart from that, some spikes in runtime still happen, although they are not as visible on this scale. These anomalies are quite bad for a naive dynamic tuning triggers, where we would simply compare the current iteration runtime to the last one – as we would trigger tuning phases that may not be necessary. I will go into how we could prevent this with different smoothing approaches later.

Using static tuning intervals we see that, as expected, one singular configuration seems to be optimal for most of the simulation. The retuning in the cases where no change in configuration happens isn't necessary, therefore we can make improvements by just not triggering new tuning phases. When running with a much shorter `tuning-interval` (in the plotted case 1000 instead of 5000 iterations), we see that in some cases a new optimal configuration is selected (see 2). I suspect however, that these changes only have a negligible performance impact, as they are only optimal for a few iterations.



(a) `tuning-interval` set to 5000 iterations (b) `tuning-interval` set to 1000 iterations

Figure 2: Selected configurations for the equilibrium scenario

1.2 Heating Sphere

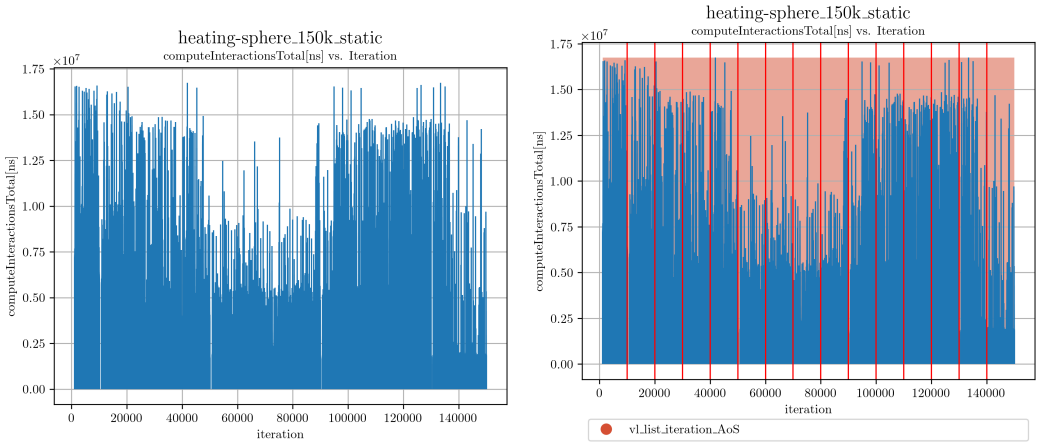


Figure 3: Runtime and selected configurations for the heating sphere scenario

In this scenario, there seems to be small relative variance in iteration runtime (3). Changes in runtime aren't followed by configuration changes. This may be because the configuration file uses predictive-tuning and slow-config-filter, so in my understanding it might happen that we don't select the optimal configuration. On the other hand, it could also just mean that runtime alone isn't a good predictor for change in this scenario.

1.3 Exploding Liquid

In the exploding liquid scenario, we see many spikes in iteration runtime, and different configurations show significant differences in runtime (4). We see that one configurations remains optimal after

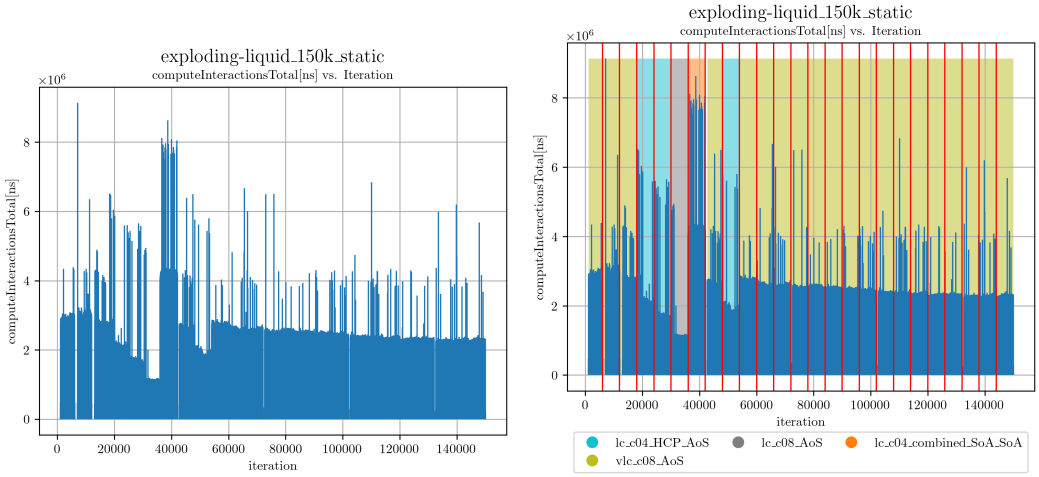


Figure 4: Runtime and selected configurations for the exploding liquid scenario. One extreme outlier has been removed.

multiple tuning phases, so dynamic tuning should be effective here. Again, it seems as if we do not always select the optimal strategy, which could again be because of predictive tuning. This is especially visible in the transition from `lc_c08_AoS` to `lc_c04_combinded_SoA_SoA`.

Runtime alone again doesn't seem to be a perfect predictor for scenario change, as there are intervals with increased runtime that are not followed by a configuration change after the next retune. However, especially in the beginning where we change to a different optimum for multiple tuning phases, it might still be a good approximation.

2 Dynamic tuning intervals

2.1 Time-based dynamic tuning intervals

Here are some time-based retuning trigger strategies I've already implemented or think could be worth investigating:

1. `TimeBasedSimple` (implemented): This strategy compares the current iteration runtime to the previous one and triggers a tuning phase, if $t_i \geq \lambda \cdot t_{i-1}$, where λ is a retune factor set by the user.

2. `TimeBasedAverageN` (implemented): This strategy is similar to `TimeBasedSimple`, but instead of comparing to the last iteration, we compare to the moving average of the last n runtime samples. I.e. a tuning phase is triggered, if

$$t_i \geq \frac{\lambda}{n} \cdot \sum_{k=i-n-1}^{i-1} t_k$$

   This strategy would not prevent tuning phases to be triggered by spikes in runtime, but would prevent it from triggering in the next iteration after a dip in runtime.

3. `TimeBasedLinearRegression`: Although this strategy is probably too compute intensive, fitting a simple linear regression over the last n samples would give us an estimate on wether the runtime is increasing significantly. The estimate for the slope would then be (I hope my maths is correct):

$$\hat{\beta} = \frac{\sum_{k=i-n-1}^{i-1}(x_k - \bar{x})(t_k - \bar{t})}{\sum_{k=i-n-1}^{i-1}(x_k - \bar{x})^2}$$

   Which can be transformed to

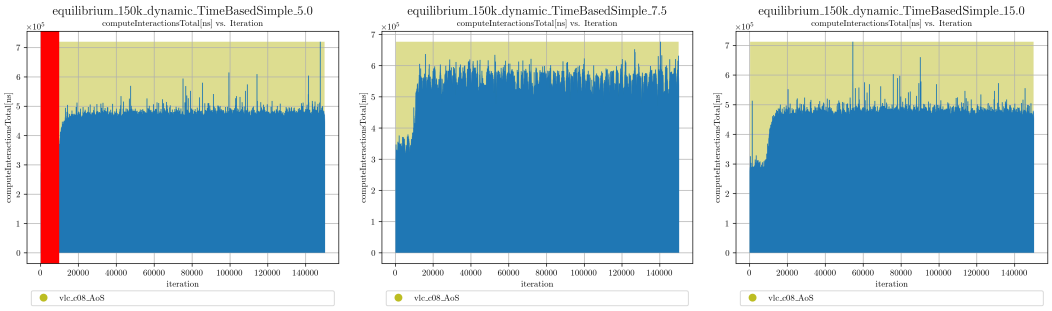
$$\hat{\beta}' = \frac{\sum_{k=0}^{n}\left(k - \frac{n^2+n}{2}\right)(t_{i-n+k-1} - \bar{t})}{\sum_{k=0}^{n}\left(k - \frac{n^2+n}{2}\right)^2}$$

   Then, we could compare $\hat{\beta}'$ to a positive threshold value, after which we assume a significant increase in runtime and start a tuning phase. The terms involving k and n should be constant, so these could be precomputed.

4. `TimeBasedSplit`: Another trigger strategy would be to split the measurements of the last n iterations and current iteration into two intervals $A = [t_{i-n}, t_{i-j}], B = [t_{i-j+1}, t_i]$ (with $j = \lfloor \frac{n}{2} \rfloor$) and then compare whether $\text{avg}(B) \geq \lambda \cdot \text{avg}(A)$.

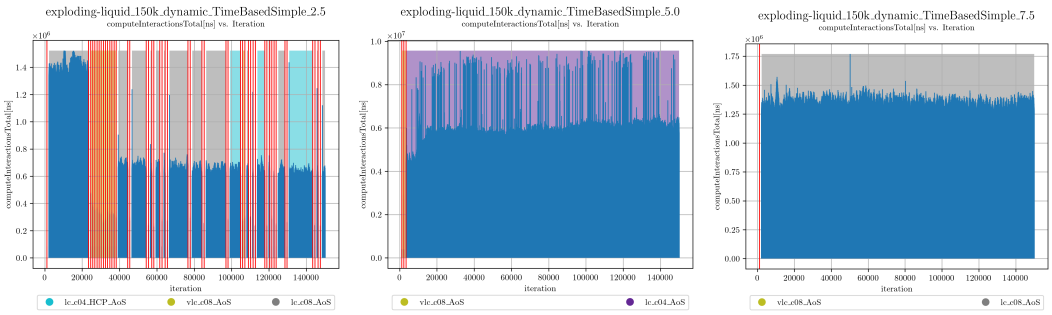
As all these strategies only try to estimate whether a parameter, in this case runtime, increases, they could also be used together with `liveinfo` measurements in parameter based tuning.

2.1.1 Results – `TimeBasedSimple`



(a) Retune factor $\lambda = 5$ (b) Retune factor $\lambda = 7.5$ (c) Retune factor $\lambda = 15$

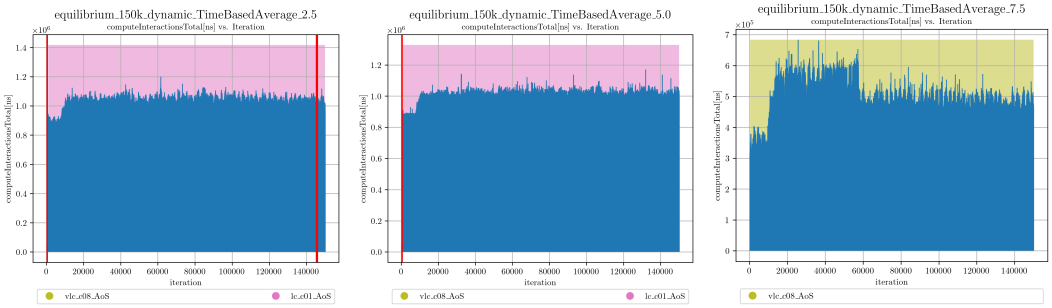
Figure 5: Selected configurations for the equilibrium scenario with `TimeBasedSimple` trigger



(a) Retune factor $\lambda = 2.5$ (b) Retune factor $\lambda = 5$ (c) Retune factor $\lambda = 7.5$

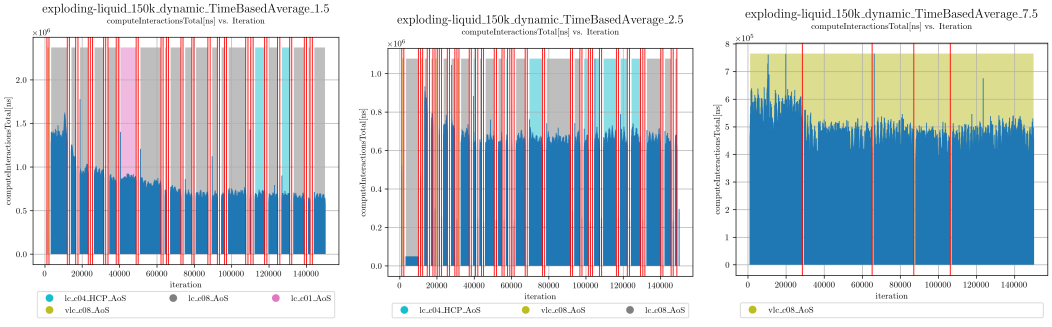
Figure 6: Selected configurations for the exploding liquid scenario with `TimeBasedSimple` trigger

2.1.2 Results – `TimeBasedAverage`



(a) Retune factor $\lambda = 2.5$, samples $n = 10$ (b) Retune factor $\lambda = 5$, samples $n = 10$ (c) Retune factor $\lambda = 7.5$, samples $n = 10$

Figure 7: Selected configurations for the equilibrium scenario with `TimeBasedAverage` trigger



(a) Retune factor $\lambda = 1.5$, samples $n = 10$ (b) Retune factor $\lambda = 2.5$, samples $n = 10$ (c) Retune factor $\lambda = 7.5$, samples $n = 10$

Figure 8: Selected configurations for the exploding liquid scenario with `TimeBasedAverage` trigger

As can be seen by these results, the `TimeBasedSimple` strategy seems suboptimal. Setting the retune factor to low leads to very aggressive triggering, as dips followed by spikes in runtime happen across all investigated scenarios. Setting it too high, we may run suboptimal configurations for too long. If the overhead of the `TimeBasedAverage` strategy to it is negligible, it would be preferable.

In the `TimeBasedAverage` strategy also seems to work better for lower λ, overall triggering less tuning phases than the simple strategy. However, in theory, it should also be susceptible to runtime spikes. That could be prevented by strategy that also smooths out the first operand of the comparison.

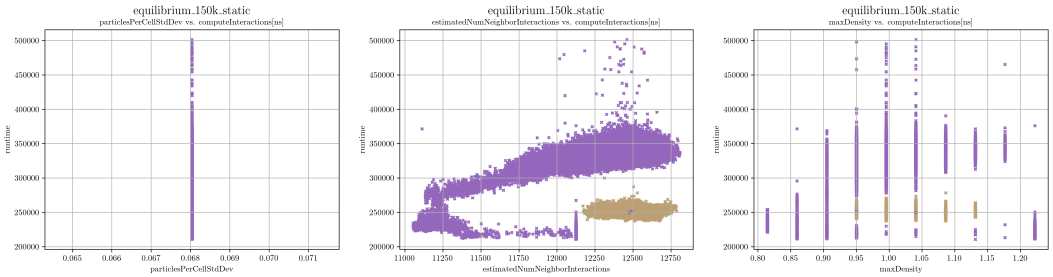
The optimal λ is probably scenario dependent, maybe it should be set dynamically based on the first few iterations or a specific parameter of the simulation. I have not yet found a way to make an educated guess about which λ would be best. I will go more into comparing the actual runtime of the whole simulation once I've figured that out.

2.2 Parameter-based dynamic tuning intervals

To get an idea on which `liveInfo` parameters correlate to iteration runtime, I've plotted some of them in different scenarios using static tuning. Data points of same color belong to the same configuration.

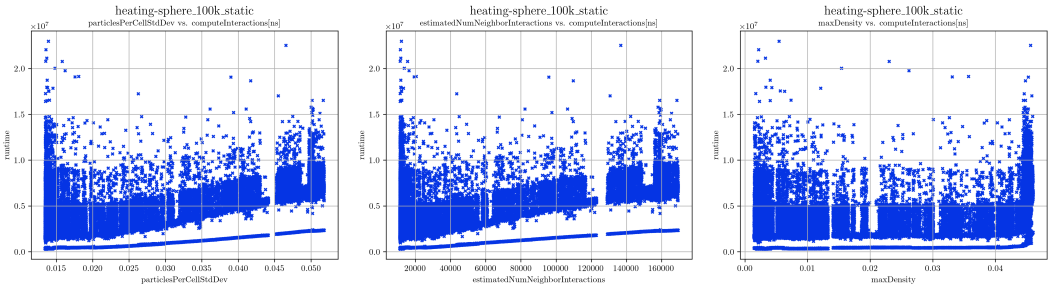
The most promising parameter seems to be `estimatedNumNeighborInteractions`, however there is still high variance within the same configuration, and between different configurations it is even worse.

Of all parameters I plotted, none seem to have a very strong correlation to runtime. Is there another approach on evaluating how optimal a configuration is? Otherwise think it might be better to use a hybrid approach, using runtime and `liveInfo` parameters together. That's why I haven't implemented any parameter-based triggers yet.



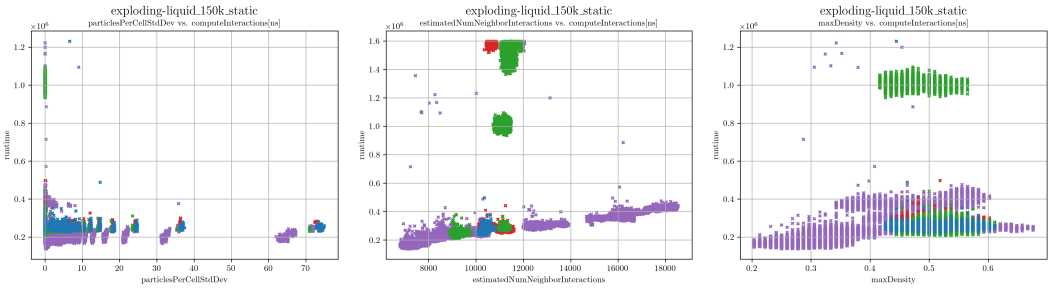
(a) particlesPerCellStdDev (b) estimatedNumNeighborInteractions (c) maxDensity

Figure 9: Selected `liveInfo` parameters for the equilibrium scenario



(a) particlesPerCellStdDev (b) estimatedNumNeighborInteractions (c) maxDensity

Figure 10: Selected `liveInfo` parameters for the heating sphere scenario



(a) particlesPerCellStdDev (b) estimatedNumNeighborInteractions (c) maxDensity

Figure 11: Selected `liveInfo` parameters for the exploding liquid scenario