Overview of AutoPas Experiments II

Niklas Ladurner

1 Dynamic tuning intervals II

1.1 Time-based dynamic tuning intervals

As discussed before, the `TimeBasedSimple` and `TimeBasedAverage` triggers are suboptimal, which is why I want to focus on the new triggers I've implemented and evaluated. I might however come back to these simple triggers if we can find a suitable scenario in which they would perform well.

As a reminder on how the new dynamic tuning strategies work, a short description:

1. `TimeBasedSplit`: This strategy splits 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 compares whether $\text{avg}(B) \geq \lambda \cdot \text{avg}(A)$.
2. `TimeBasedRegression`: We fit a simple linear regression, adapted to our use case. In the following, n is the number of samples, t_k the runtime at iteration k, i the current iteration and $\bar{t}$, $\bar{x}$ the average runtime and iteration respectively.

$$\hat{\beta}_1 = \frac{\sum_{k=i-n-1}^{i}(k-\bar{x})(t_k-\bar{t})}{\sum_{k=i-n-1}^{i}(k-\bar{x})^2}$$

$$\bar{t} = \frac{1}{n}\sum_{k=i-n-1}^{i} t_k, \quad \bar{x} = \frac{1}{n}\sum_{k=i-n-1}^{i} k$$

Since we do not care about the offset ($\hat{\beta}_0$), nor the values along x (iterations), but only about the factor itself, we can transform this to:

$$\hat{\beta}_1' = \frac{\sum_{k=0}^{n-1}\left(k - \frac{n(n-1)}{2n}\right)(t_{i-n-1+k} - \bar{t})}{\sum_{k=0}^{n-1}\left(k - \frac{n(n-1)}{2n}\right)^2} = \frac{1}{B}\sum_{k=0}^{n-1}(k-A)(t_{i-n-1+k} - \bar{t})$$

Where

$$A = \frac{n-1}{2}, \quad B = \sum_{k=0}^{n-1}(k-A)^2$$

can be precomputed at initialization.

Then, we compare $\hat{\beta}_1'$ to a positive threshold value, after which we assume a significant increase in runtime and start a tuning phase. $\hat{\beta}_1'$ can be interpreted as "in each iteration, the runtime is projected to increase $\hat{\beta}_1'$ nanoseconds". This however, is not a good metric to compare to a user-set triggerFactor, as this heavily depends on the scenario and would require to know rough iteration runtime estimates beforehand. Therefore, I suggest some form of normalization, such

that a factor of 1.0 is roughly equal to "no runtime increase", as to make it somewhat consistent to the other triggers. The normalization currently implemented is as follows:

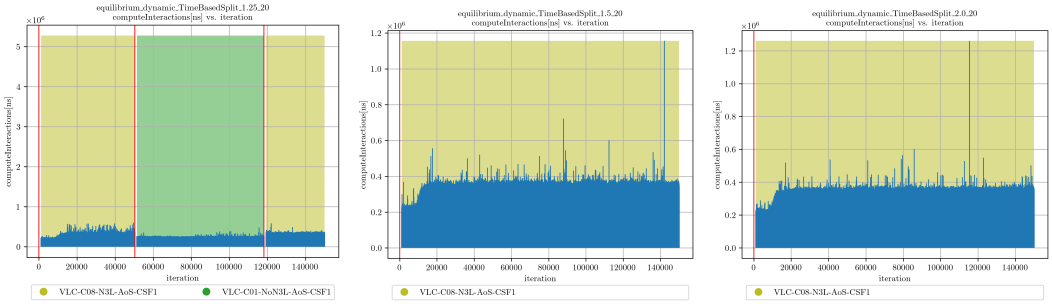
$$\hat{\beta}_{\text{norm}} = \frac{n \cdot \hat{\beta}'_1}{\bar{t}}$$

By this, $\hat{\beta}_{\text{norm}}$ can be used with triggerFactors comparable to the other trigger strategies. We have

- $\hat{\beta}_{\text{norm}} = 1$ if there is no projected change in iteration runtime
- $\hat{\beta}_{\text{norm}} > 1$ if there is a projected increase in iteration runtime
- $\hat{\beta}_{\text{norm}} < 1$ if there is a projected decrease in iteration runtime
- $\hat{\beta}_{\text{norm}} = 2$ if there the runtime of the next interval is projected to be double the current interval's runtime.

Again, 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. I will revisit that topic in future.

1.1.1 Results – `TimeBasedSplit`

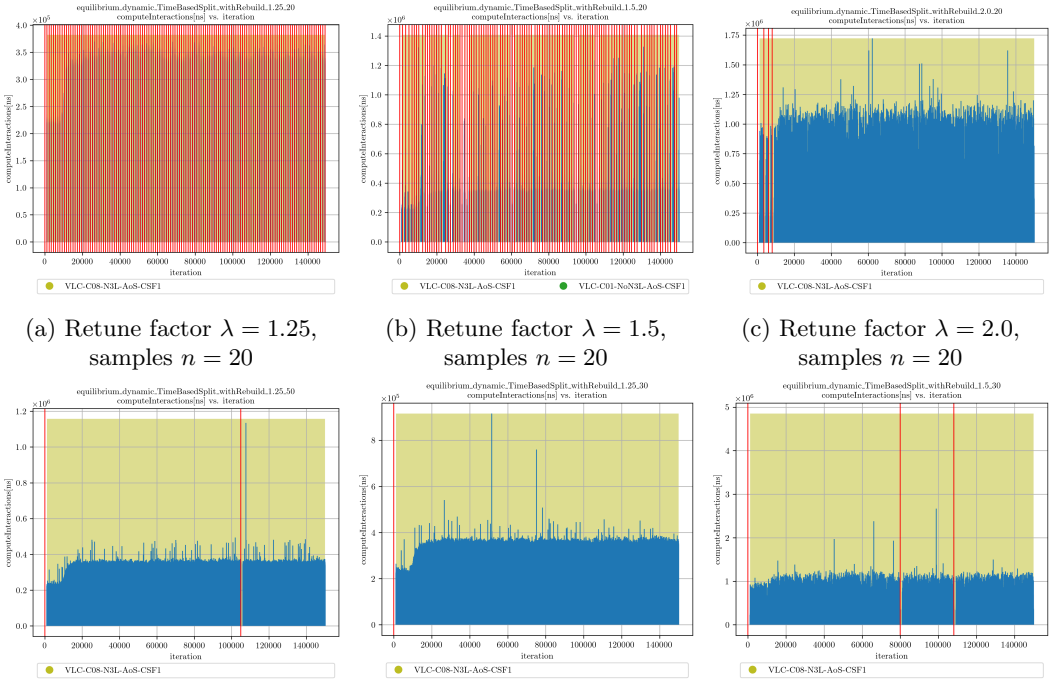


(a) Retune factor $\lambda = 1.25$, samples $n = 20$

(b) Retune factor $\lambda = 1.5$, samples $n = 20$

(c) Retune factor $\lambda = 2.0$, samples $n = 20$

Figure 1: Selected configurations for the equilibrium scenario with `TimeBasedSplit` trigger. In these runs, rebuild times were not tracked.



(a) Retune factor $\lambda = 1.25$, samples $n = 20$

(b) Retune factor $\lambda = 1.5$, samples $n = 20$

(c) Retune factor $\lambda = 2.0$, samples $n = 20$

(d) Retune factor $\lambda = 1.25$, samples $n = 50$

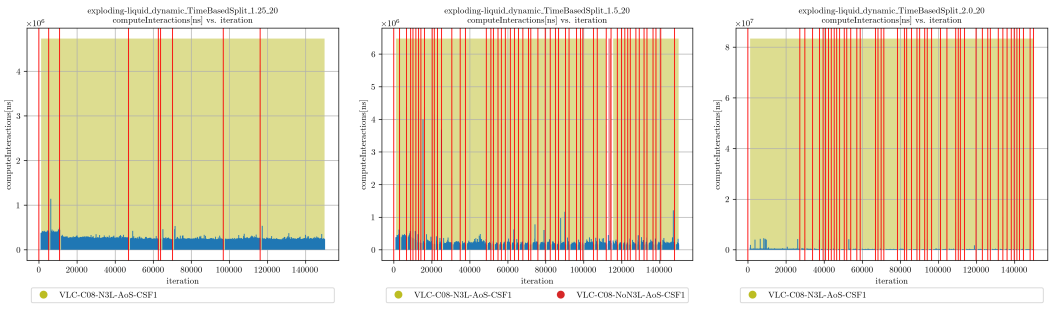
(e) Retune factor $\lambda = 1.25$, samples $n = 30$

(f) Retune factor $\lambda = 1.5$, samples $n = 30$

Figure 2: Selected configurations for the equilibrium scenario with `TimeBasedSplit` trigger.

As can be seen by the difference in Figure 1 and Figure 2, the retuning strategy is unstable if rebuild times are tracked and the number of samples is to low. In that case, the rebuild iterations have too big of an impact, which results in too many triggers being fired. Considering too many samples however, changes in runtime may get smoothed out too much, depending on the frequency at which the change happens. Also, if the number of samples is set such that one interval contains more rebuild iterations than the other, this too leads to wrong behaviour.

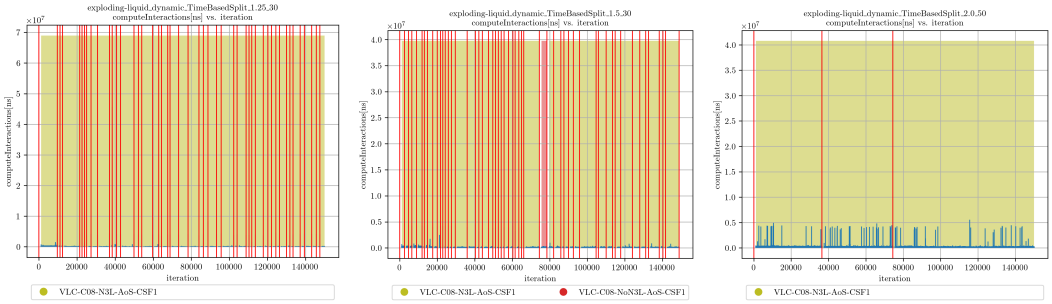
Because of that, I have decided to not include rebuild iteration times for now, however I will revisit this problem in future. A possible idea to reduce the effect of such rebuild iterations would be to track them separately or to filter out high frequencies, i.e. fast changes in runtime, by using low-pass filters. As we do not expect a scenario change to happen from one iteration to the other, we might not need these high frequencies anyways.



(a) Retune factor $\lambda = 1.25$, samples $n = 20$

(b) Retune factor $\lambda = 1.5$, samples $n = 20$

(c) Retune factor $\lambda = 2.0$, samples $n = 20$

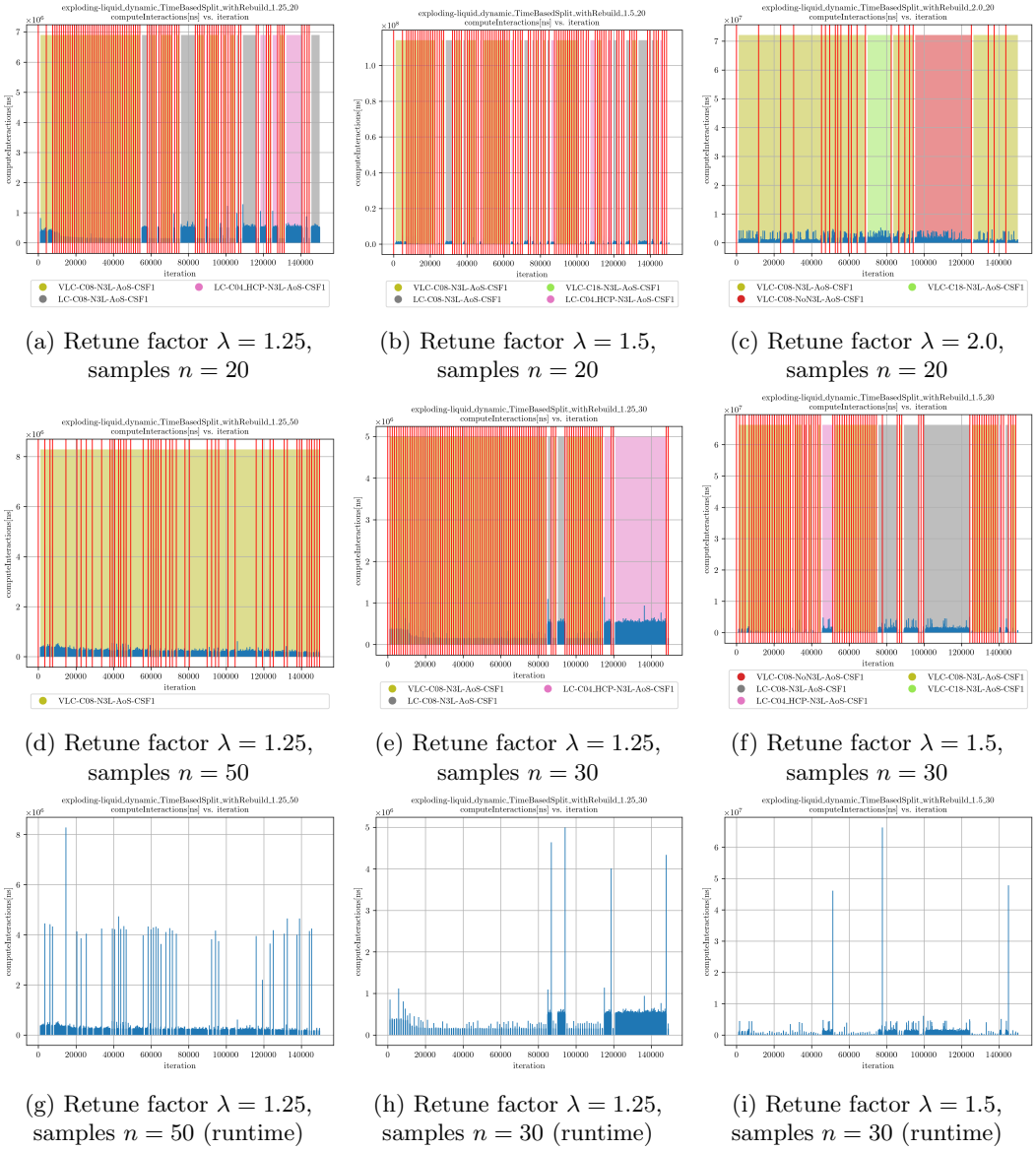


(d) Retune factor $\lambda = 1.25$, samples $n = 30$

(e) Retune factor $\lambda = 1.5$, samples $n = 30$

(f) Retune factor $\lambda = 2.0$, samples $n = 50$

Figure 3: Selected configurations for the exploding liquid scenario with `TimeBasedSplit` trigger. In these runs, rebuild times were not tracked.



(a) Retune factor $\lambda = 1.25$, samples $n = 20$

(b) Retune factor $\lambda = 1.5$, samples $n = 20$

(c) Retune factor $\lambda = 2.0$, samples $n = 20$

(d) Retune factor $\lambda = 1.25$, samples $n = 50$

(e) Retune factor $\lambda = 1.25$, samples $n = 30$

(f) Retune factor $\lambda = 1.5$, samples $n = 30$

(g) Retune factor $\lambda = 1.25$, samples $n = 50$ (runtime)

(h) Retune factor $\lambda = 1.25$, samples $n = 30$ (runtime)

(i) Retune factor $\lambda = 1.5$, samples $n = 30$ (runtime)

Figure 4: Selected configurations for the equilibrium scenario with `TimeBasedSplit` trigger.

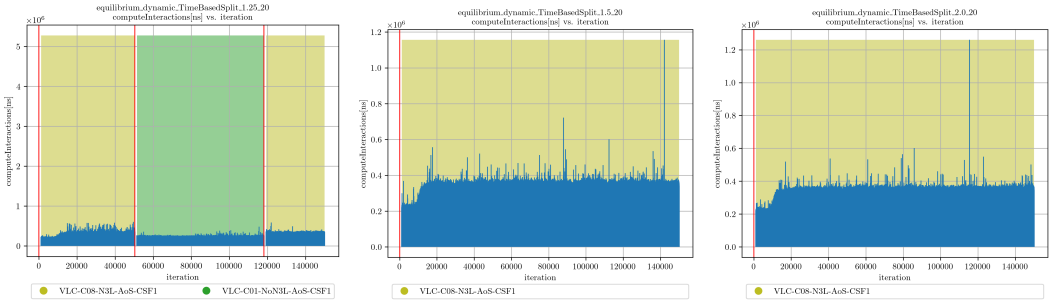
In Figures 3 and 4 we can see a similar behaviour as in the equilibrium case, where rebuild iterations lead to unstable behavior. However, even if we do not track the rebuild times, we can observe too many triggers being fired.

1.1.2 Results – `TimeBasedRegression`

Even tough we exclude rebuild times for now, we still have quite some variance in iteration runtimes. The simple linear regression is very susceptible to outliers, skewing our β_{norm} estimates to values

≥ 2.0. The effect of these outliers can be somewhat mitigated by choosing a higher sampling interval (n), but that incurs a higher performance impact.

It seems there exists a more robust algorithm called the "Theil–Sen estimator", which might be worth investigating. Alternatively, we could filter/average out subintervals (e.g. averaging every sample with its predecessor) and fit a regression line on these.

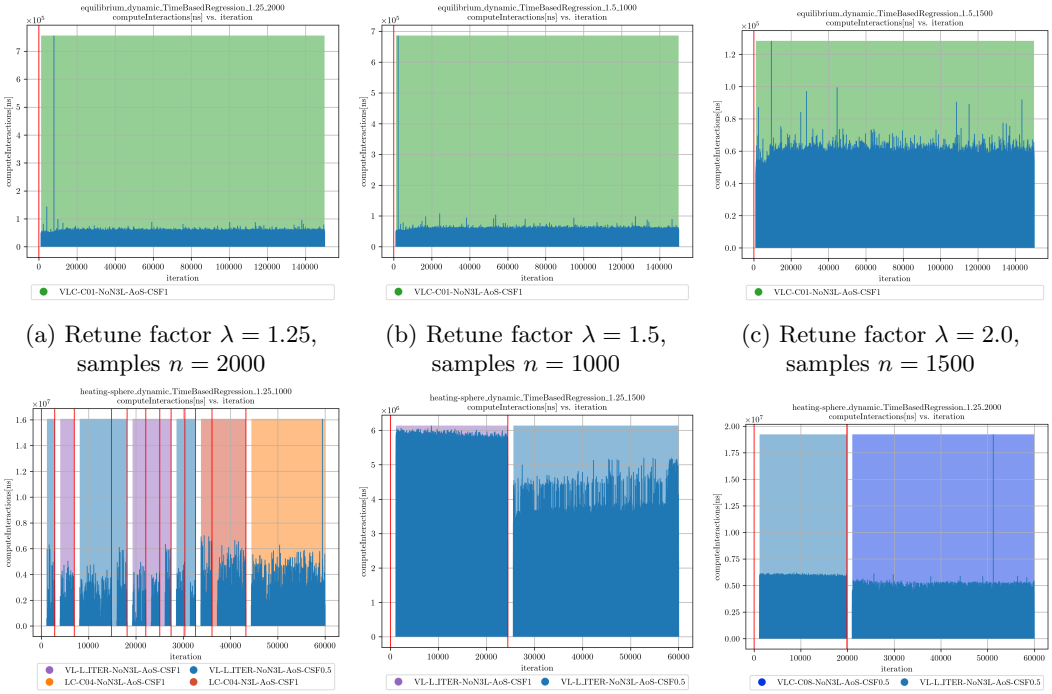


(a) Retune factor $\lambda = 1.25$, samples $n = 20$

(b) Retune factor $\lambda = 1.5$, samples $n = 20$

(c) Retune factor $\lambda = 2.0$, samples $n = 20$

Figure 5: Selected configurations for the equilibrium scenario with `TimeBasedRegression` trigger. In these runs, rebuild times were not tracked.



(a) Retune factor $\lambda = 1.25$, samples $n = 2000$

(b) Retune factor $\lambda = 1.5$, samples $n = 1000$

(c) Retune factor $\lambda = 2.0$, samples $n = 1500$

(d) Retune factor $\lambda = 1.25$, samples $n = 1000$

(e) Retune factor $\lambda = 1.25$, samples $n = 1500$

(f) Retune factor $\lambda = 1.25$, samples $n = 2000$

Figure 6: Selected configurations for the heating sphere scenario with `TimeBasedRegression` trigger. In these runs, rebuild times were not tracked.

2 Evaluation of dynamic tuning intervals

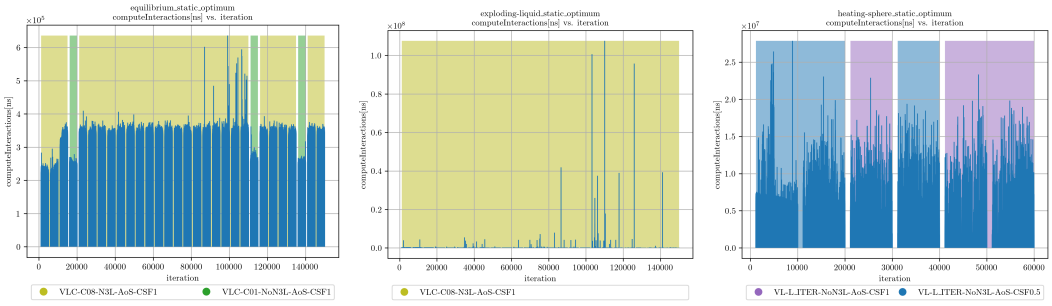
2.1 Increased number of tuning samples and predictive Tuning

With an increased number of tuning samples we expect better configuration choices by the autotuner. Therefore, I ran the equilibrium scenario with `tuning-samples` set to 10. Nothing changed compared to a lower number of `tuning-samples`. `vlc_c08_AoS` is optimal for most intervals, for some intervals we chose `vlc_c01_AoS`.

For the exploding liquid and heating sphere scenarios, `tuning-samples` was already set to 10, but I switched the tuning strategy from predictive tuning to full search. At least in the heating sphere scenario, as discussed in our last meeting, predictive tuning blacklists the optimal configuration for later iterations early on, so by using full search we prevent that.

For the exploding liquid scenario, instead of the many different configurations chosen with a lower number of samples, (see my first overview, 1.3.), there is now only one optimal configuration: `vlc_c08_AoS`, which was also the dominant configuration (80%) in the run with fewer samples. However, this seems to lead to some big spikes in runtime for some iterations.

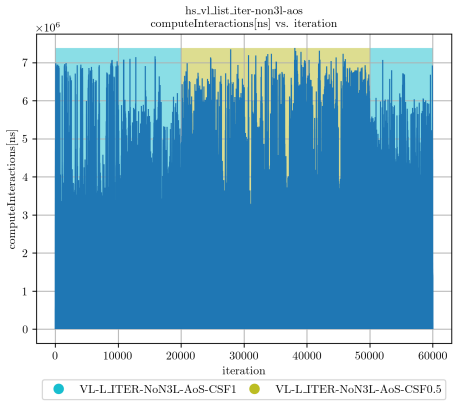
Interestingly, in the heating sphere scenario, we do not switch to `LC-C04-N3L-AoS-CSF1` even with full search.

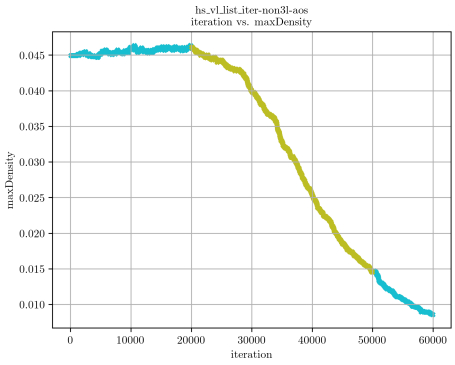


(a) Equilibrium (b) Exploding liquid (c) Heating sphere

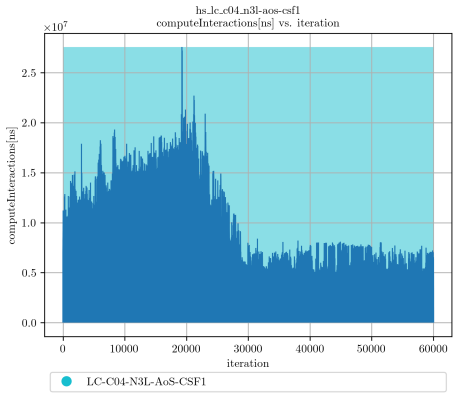
2.2 Single configuration runs

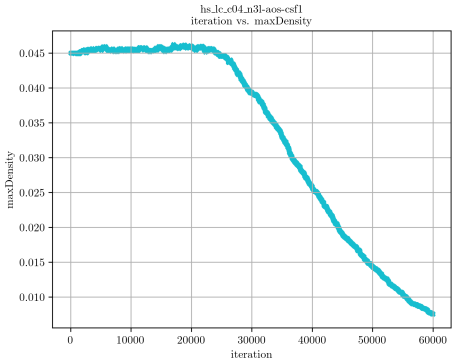
I've run some single configurations for the heating sphere scenario to better investigate wether some liveinfo parameters might indication scenario change. We know that for this scenario, we should start with `VL-List_Iter-NoN3L-AoS` and later on switch to `LC-C04-N3L-AoS-CSF1`. Note that I forgot to specify the CSF as 1, which is why it changes between 0.5 and 1 in the LC case. It seems that neither runtime-wise nor in changes of liveinfo, we can see any point that indicates that we should change configuration. Maybe this will get better once I redo the run with CSF 1 only.





(a) Optimal configuration for the first part of the simulation.





(b) Optimal configuration for the second half of the simulation.