Eiron: Code Camp 2024 presentation

Oskar Lappi

November 2024

Research Pipeline

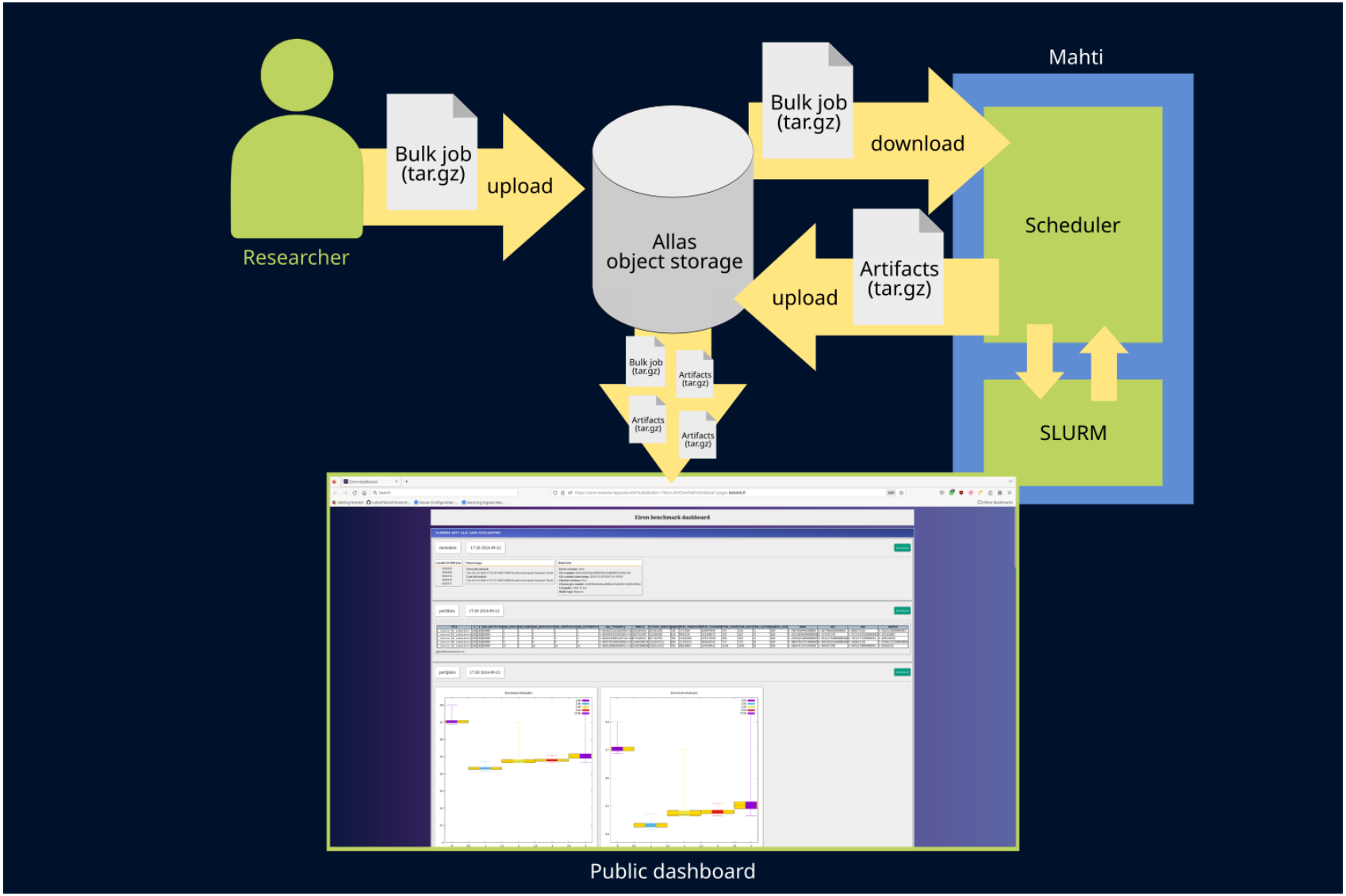
Research pipeline

To make life easier for myself, I've set up a rough research pipeline to run these experiments.

- I upload an experiment to an object storage bucket
- A scheduler picks it up and schedules SLURM jobs
- The same scheduler uploads the results back to the bucket after completion
- A public web dashboard displays the results

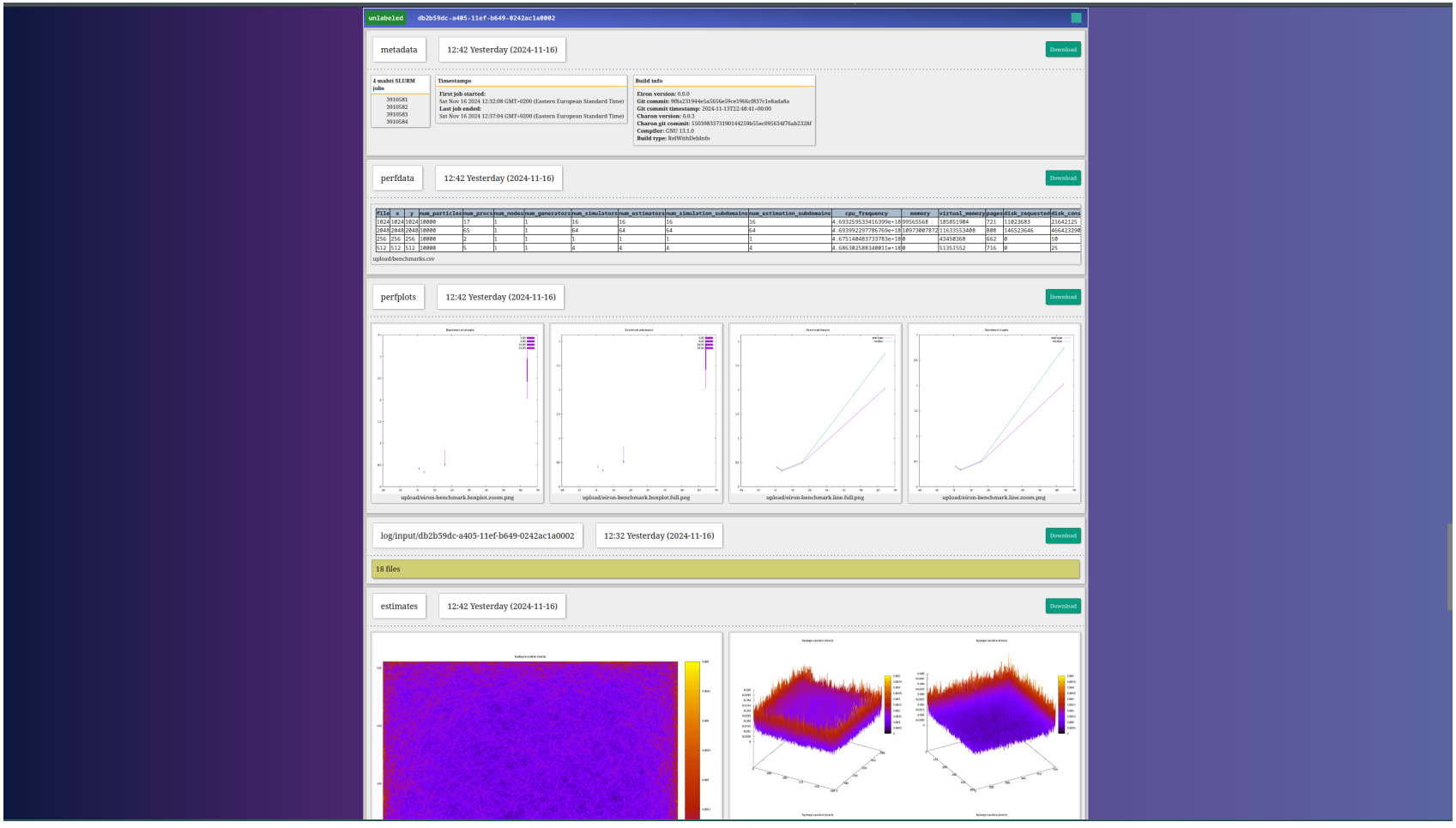
This makes it easier to run multiple jobs and check on them, and also makes it possible to share results quickly with collaborators.

Research pipeline architecture



Research pipeline dashboard

The dashboard is a simple web app that downloads the outputs of the research pipeline and displayes them.
It's a very rough prototype, but it works for my current purposes.



Use-cases for research pipeline

Performance

- *Tuning:* Run the same benchmark while varying performance parameters (queue sizes, etc.) to find the best configuration
- *Scalability:* Run the same benchmark while varying the number of processes and/or the problem size

Testing

- Run the same simulation problem with different execution scheme, verify that results are the same

Computational experiments

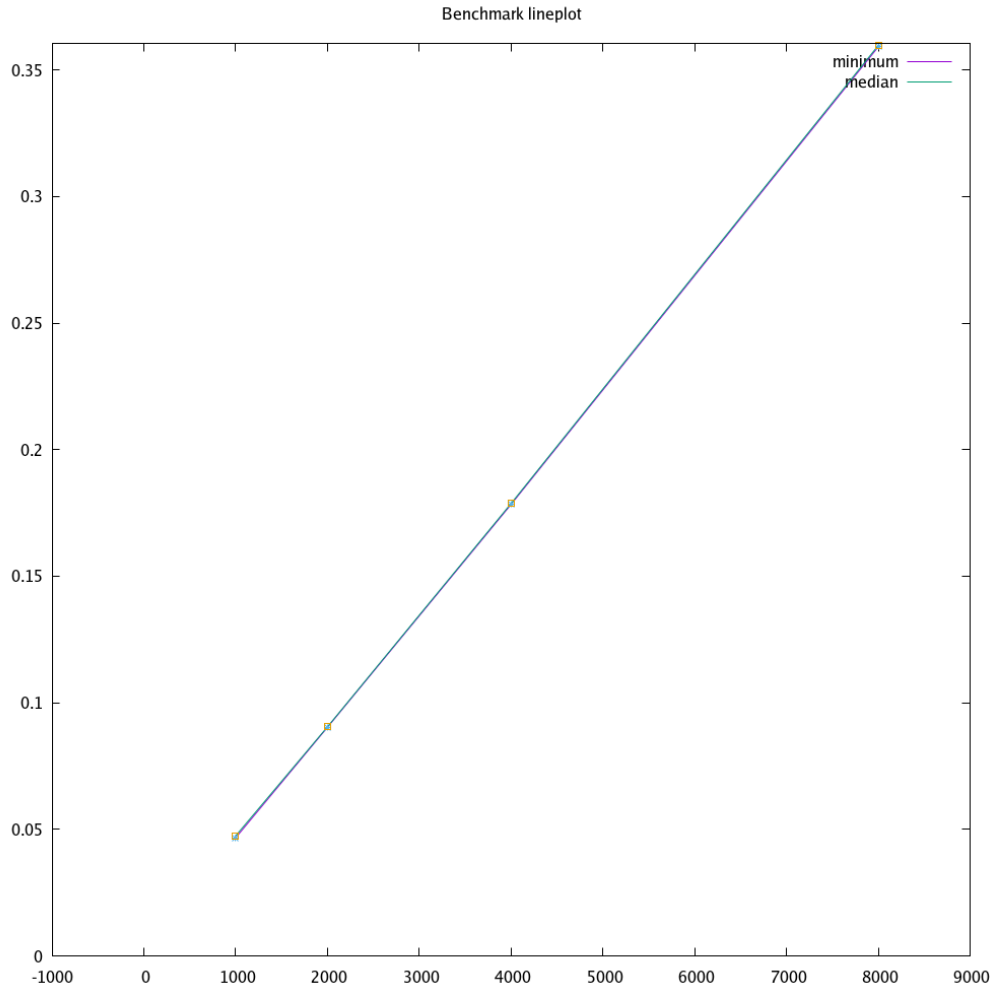
- *Simulation algorithms:* Compare kinetic and KDMC algorithms
- *Estimation methods:* E.g. probable events vs. actual events
- *Ensembles:* Not very relevant at the moment, but possibly in future

Performance plots

(Y-Axis runtime)

Serial performance: particle count effect

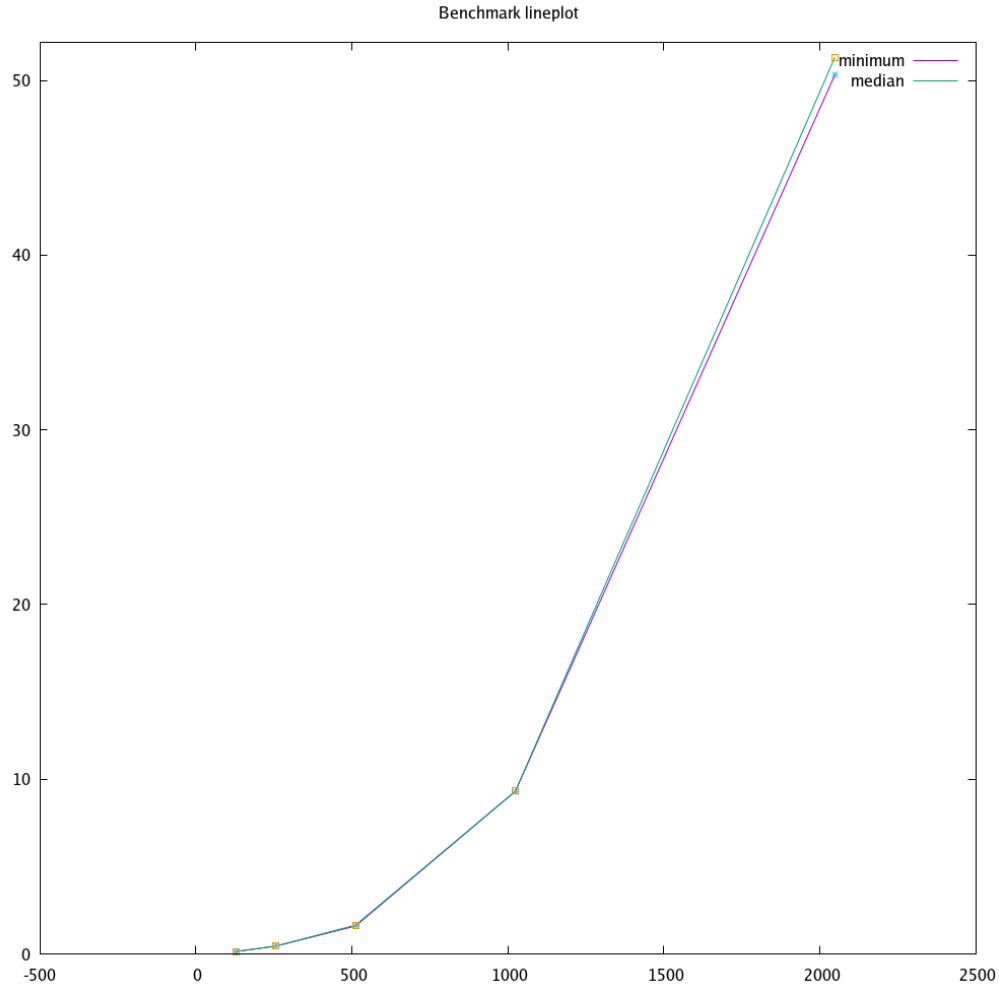
Warmup: increase the particle count on a single process run.



No surprises, the runtime scales linearly with particle count.

Serial performance: box size effect

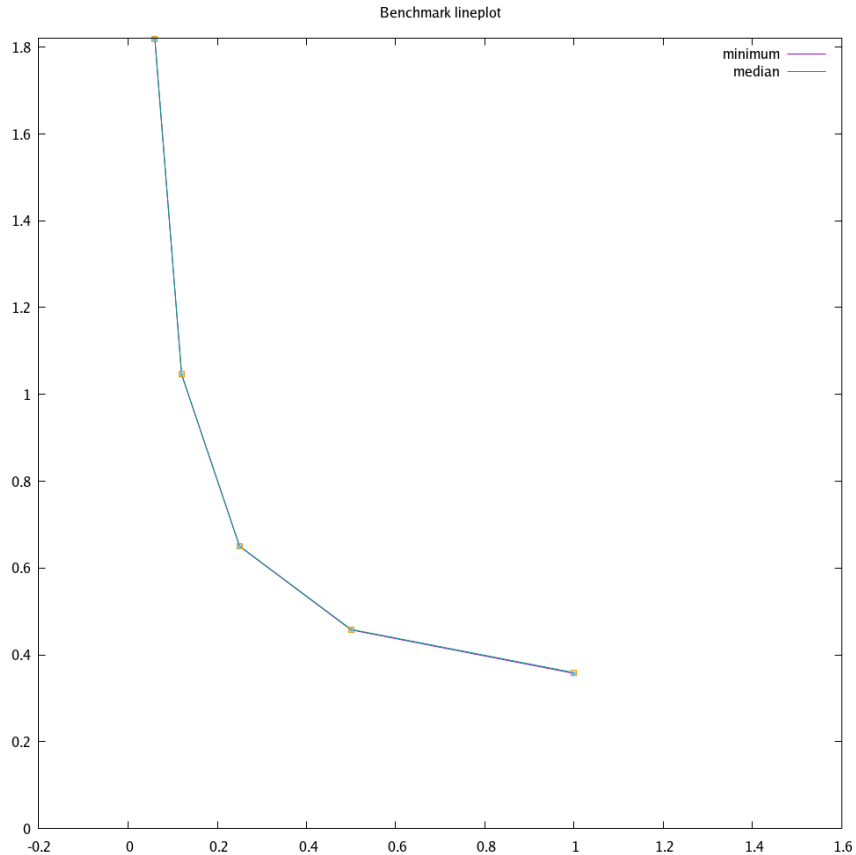
Increase box size in steps of 4x: 128^2, 256^2, 512^2, 1024^2, 2048^2,



The runtime increase is $\sim$3.5x for the first two steps, $\sim$5.5x for the last two.

Performance study: mean free path effect

Start with a mean free path of 0.625 box lengths, and double it until we get MFP=1 box length.



Remember: scattering processes only, the particles only exit when they hit the box wall. Lower mfp let's a particle live longer. MFP is fixed at 0.5 box lengths for the other benchmarks.

Domain decomposition effect

(single process)

Performance study: domain decomposition effect

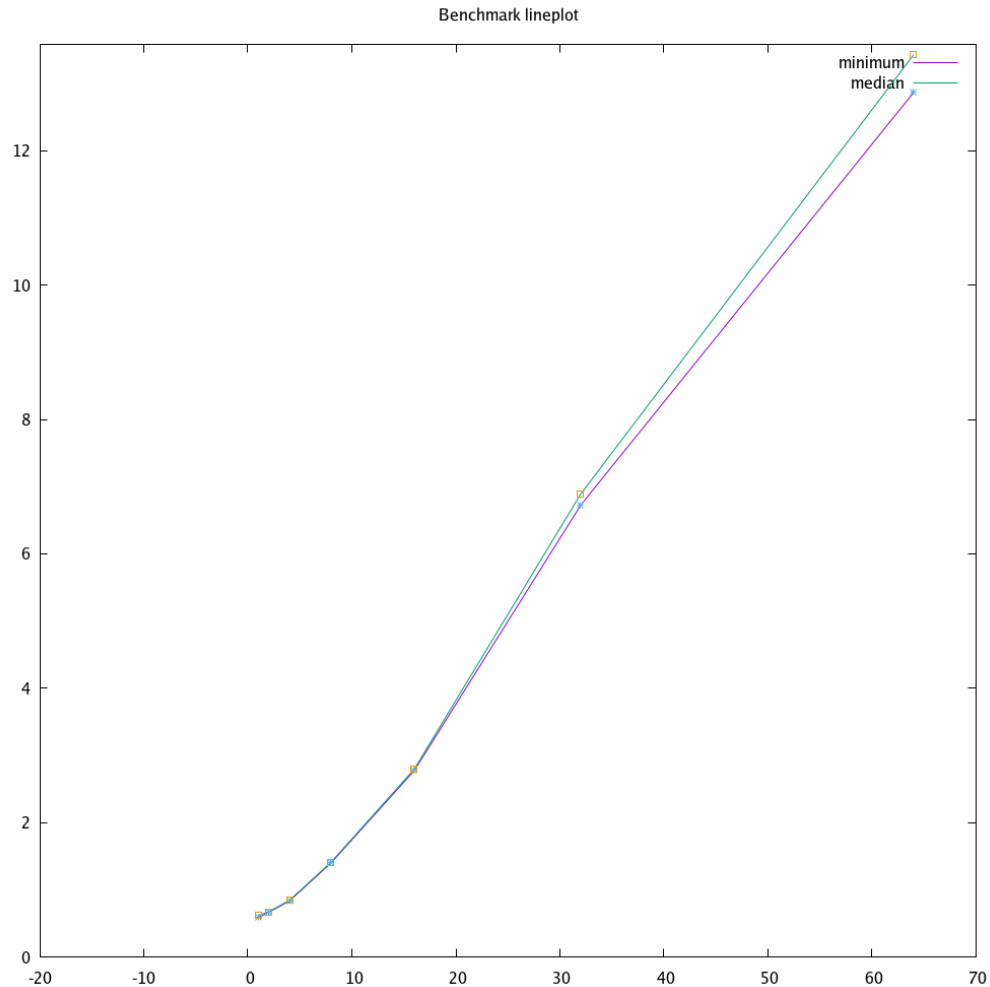
For this experiment, we use two processes, one generating particles, the other simulating and estimating them. If there's more than one subdomain, the simulating process is sending messages to itself.

The grid size stays constant, but we're varying the number of subdomains.

Every subdomain we add should increase the work done by the simulator. However, there are memory efficiency effects...

256x256 grid: domain decomposition effect

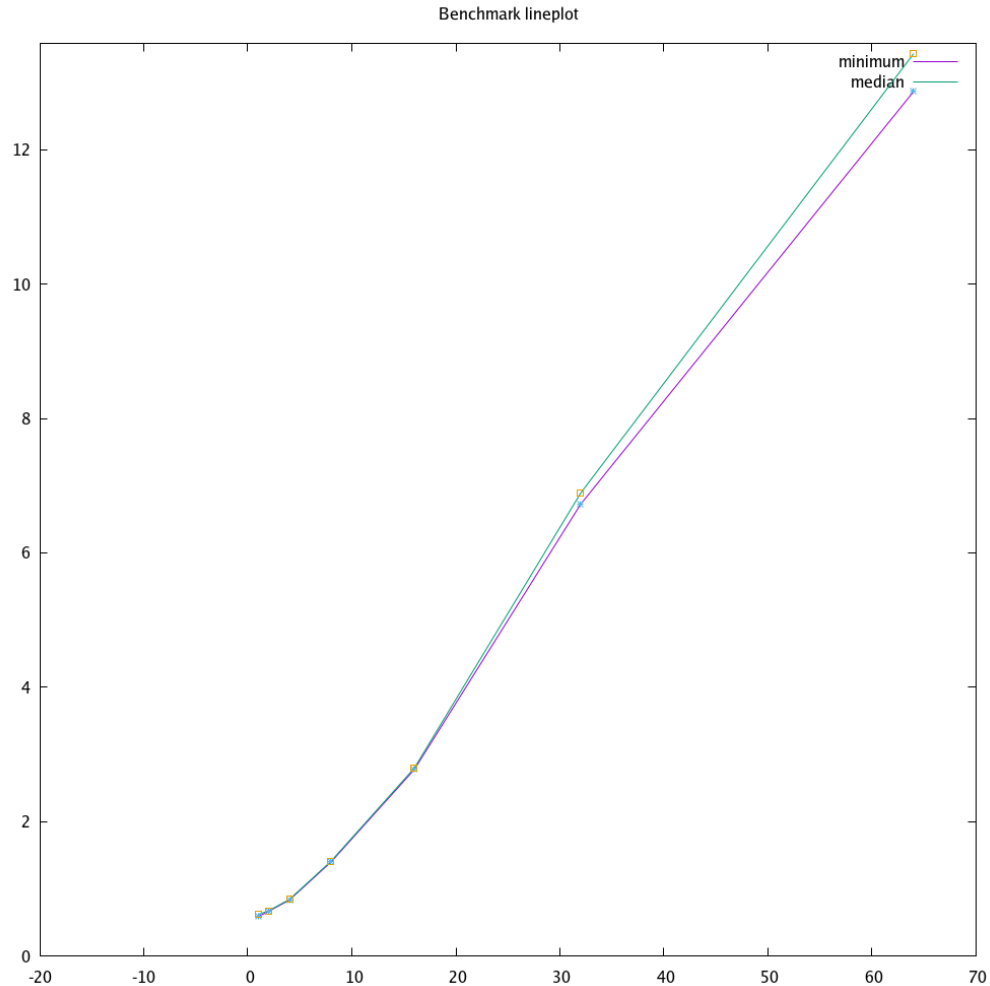
For a 256x256 grid, more subdomains is indeed just more work.



X-axis is number of subdomains, Y-axis is runtime in seconds.

256x256 grid: domain decomposition effect

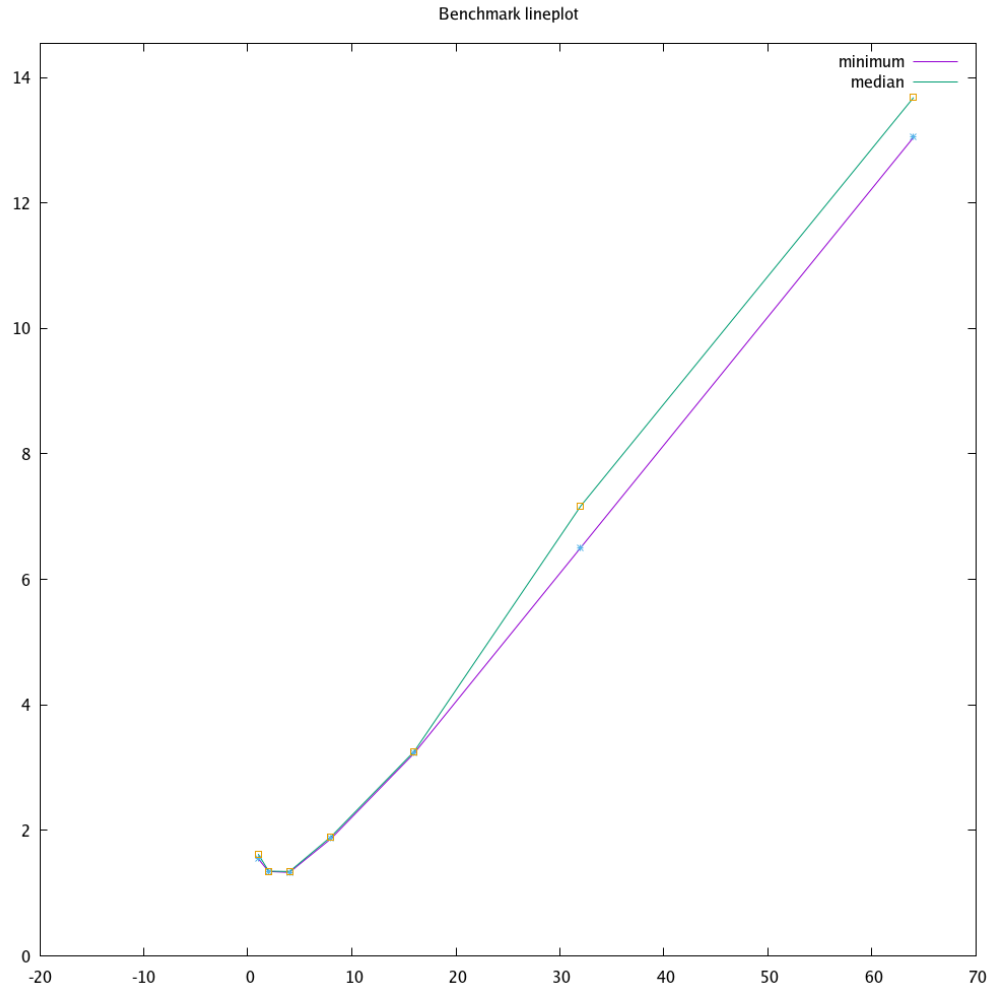
For a 256x256 grid, more subdomains is indeed just more work.



Runtime seems to be roughly linear with the subdomain count.

512x512 grid: domain decomposition effect

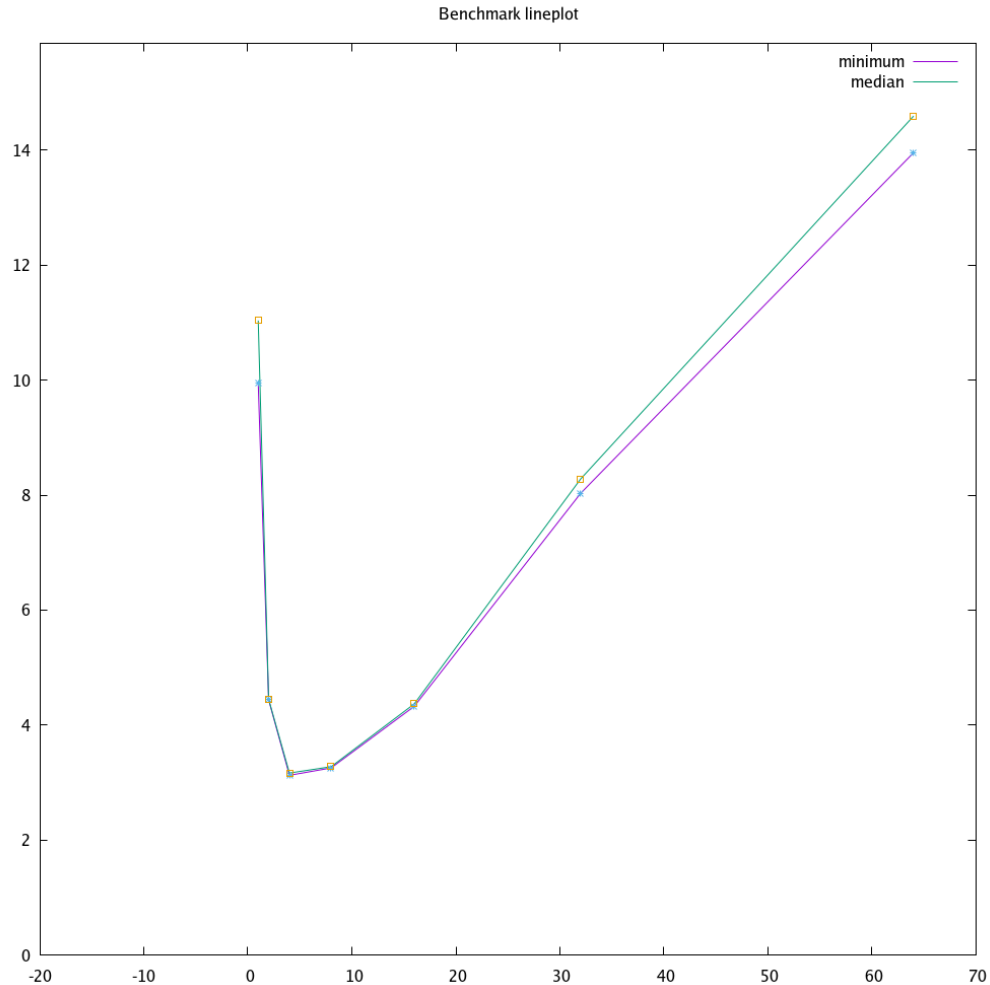
For a 512^2 grid, 4 subdomains or a subdomain size of 256^2 is best.



We start to see an optimum valley forming.

1024x1024 grid: domain decomposition effect

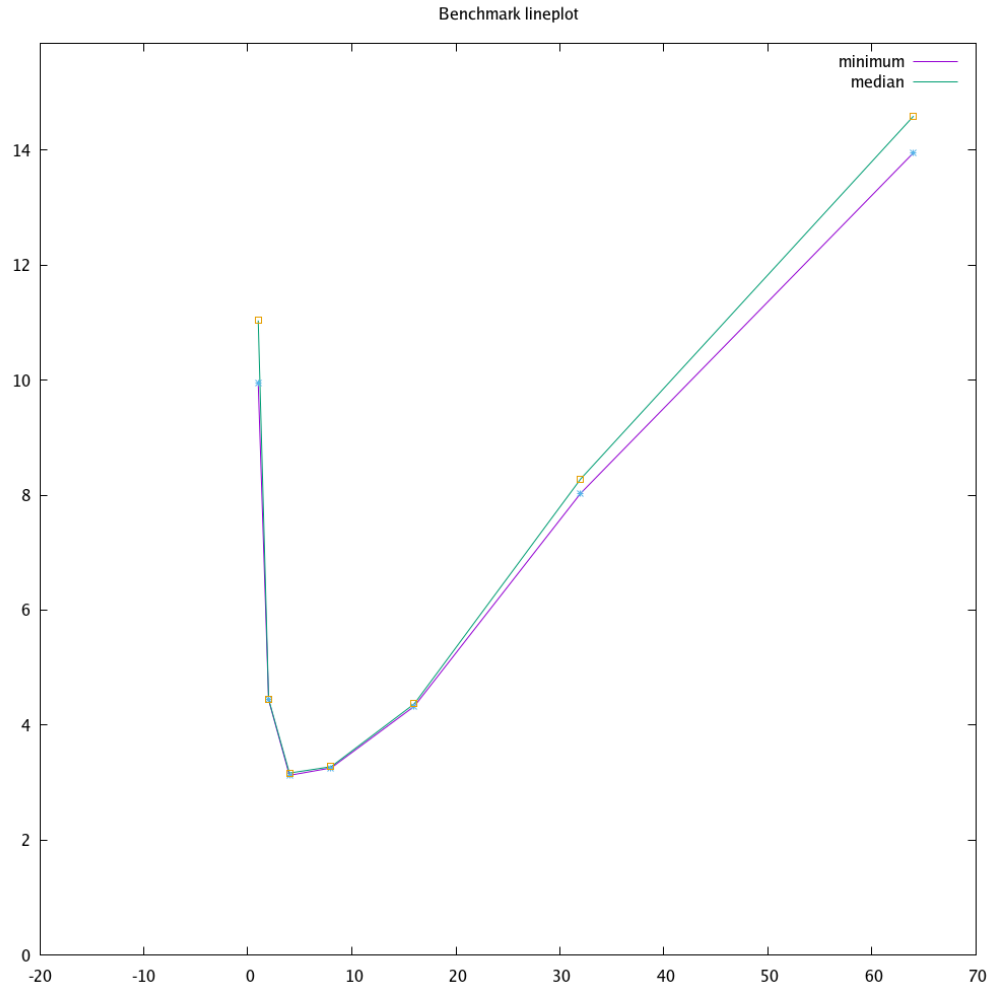
For a 1024^2 grid, 4 subdomains or a subdomain size of 512^2 is best.



Notice the more than 2x difference between 1 subdomain, 11 seconds; and 2,4,8,16 subdomains, all around 3-4 seconds.

1024x1024 grid: domain decomposition effect

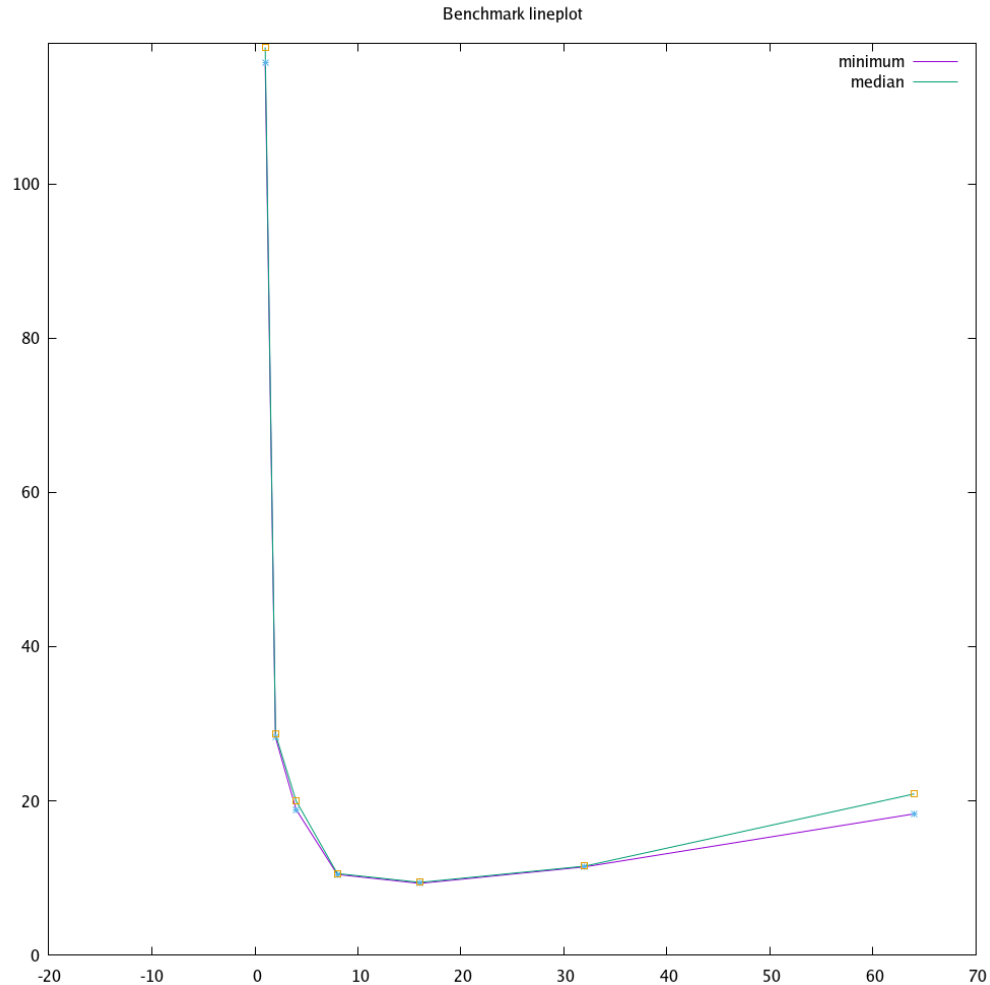
For a 1024^2 grid, 4 subdomains or a subdomain size of 512^2 is best.



On the right side of the valley, the difference is also $\sim$2x, or at least on the same order as the left side of the valley.

2048x2048 grid: domain decomposition effect

For a 2048^2 grid, 16 subdoms or a subdomain size of 512^2 is best.



Notice the huge gradient on the left side of the valley, 1 subdomain is 12x slower than 16.

Domain decomposition effect: interpretation

The single simulator process still holds an equal amount of memory (or more) when we subdivide the grid, and it still does the same computational work + some extra communication work.

So what's going on here, why is there a certain subdomain size that seems to be best for performance?

The reason is a difference in memory layout. We've essentially created a blocked memory layout by subdividing the grid, where points that are closer together in space are closer together in memory.

The cache system prefers that you process spatially local data in subsequent instructions, we've improved spatial locality, which explains the difference.

Domain decomposition effect: interpretation

This 12x difference in the 2048 case is quite bad though, and we could try to improve the situation by separately implementing a blocked indexing of the grid to replace the row-wise scan index we've now used.

Ideas like space-filling curves might also be possible, but they often only permit square or 1-by-2 grids with dimensions of power two.

For a code like EIRENE, with unstructured grids of polygons, a quadtree index or a smart sorting of the polygons that maximizes spatial locality could be a similar improvement.

I think this could be a good problem to give a PhD or MSc student.

Scaling

Benchmarks from last week

I started studying the scalability last friday, and have made some progress.

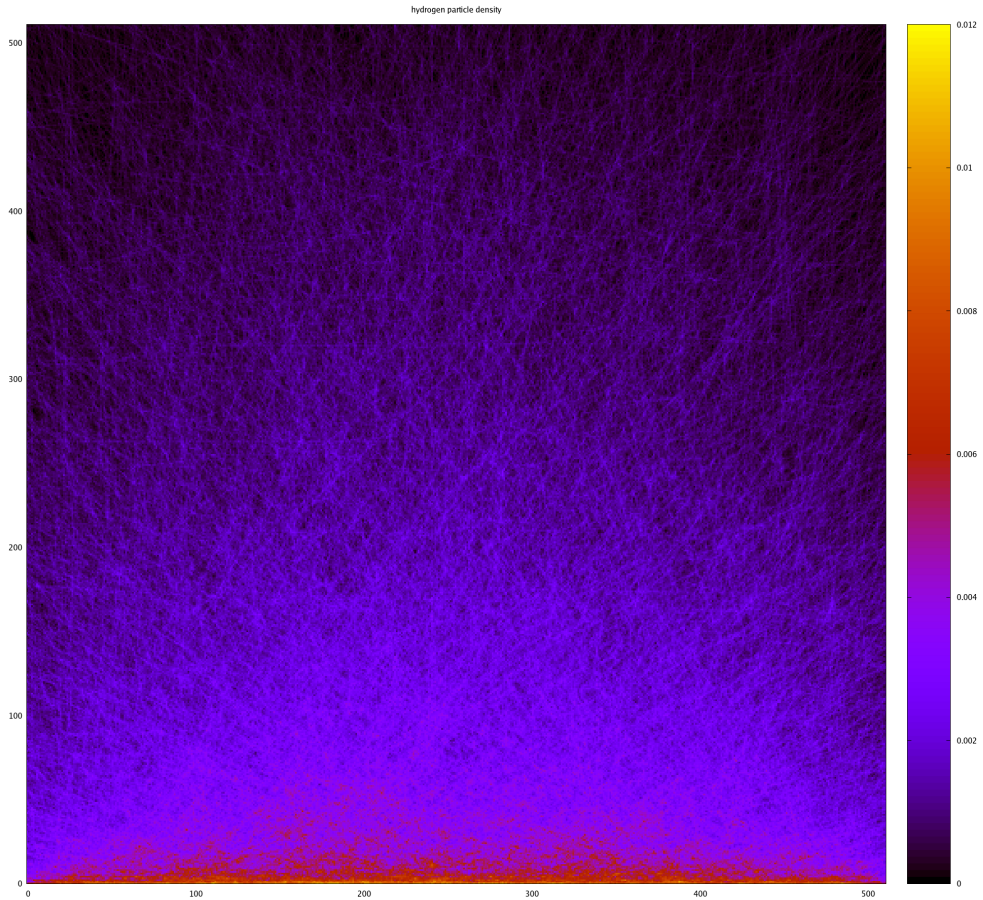
There are some unresolved issues, I still need some time to tune parameters before settling on which experiments to publish.

Still, I think the results so far are interesting, and I would appreciate any comments or thoughts on them.

Scaling case 1

Performance study: weak scaling

Let's create a simple simulation case: one source at the bottom boundary, a MFP of 0.5 box lengths, and 10 thousand particles.



Performance study: weak scaling

For this experiment, we use one generator process, and vary both the number of simulator processes, the domain size, and the domain decomposition.

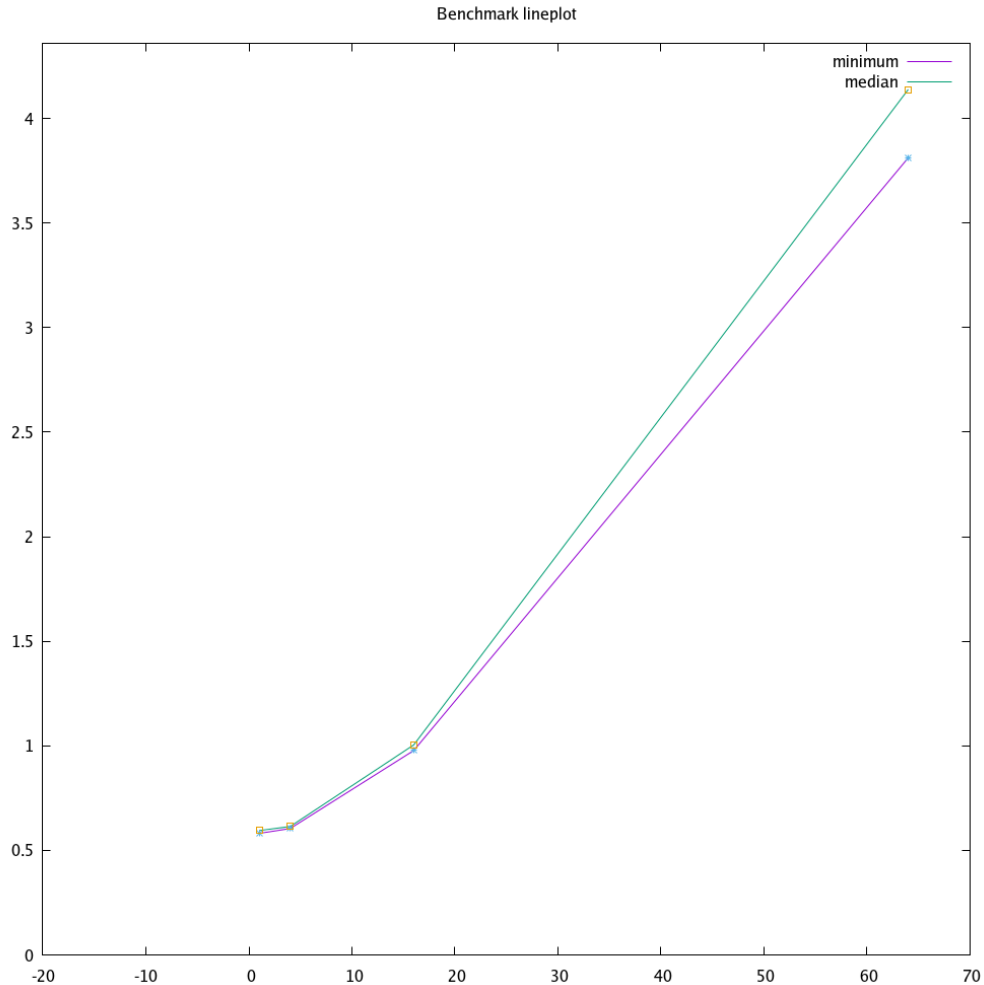
Each simulator process is in charge of exactly one subdmomain. The subdomain size stays constant, but we vary the global grid size and the number of simulators.

The number of processes is $1 + \#simulators$, because of the generator process.

The ideal case scenario is a straight horizontal line. However, this turns out to be tricky to achieve.

Performance study: weak scaling

Let's run the same simulations we were running before and see how they scale when we increase the resolution.

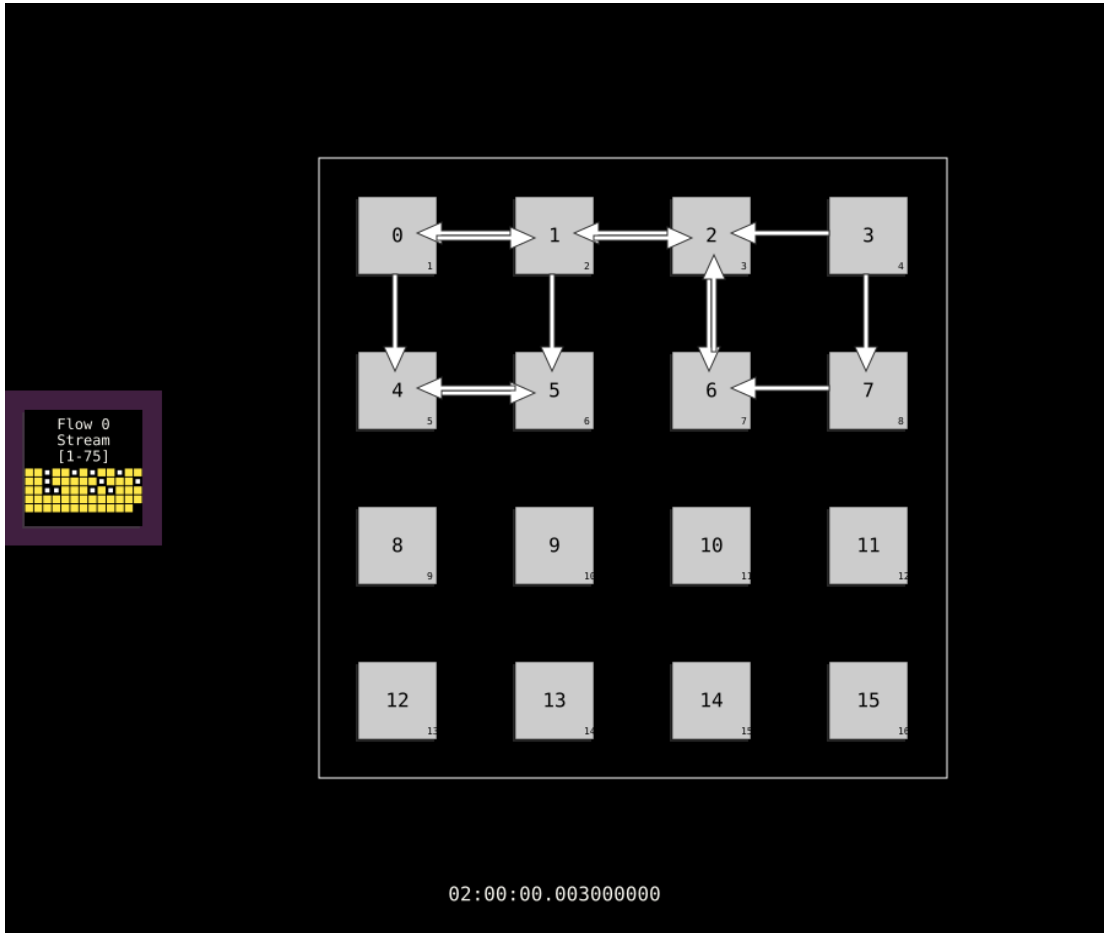


As we see, we're nowhere near the best case scenario.

Investigating scaling problems

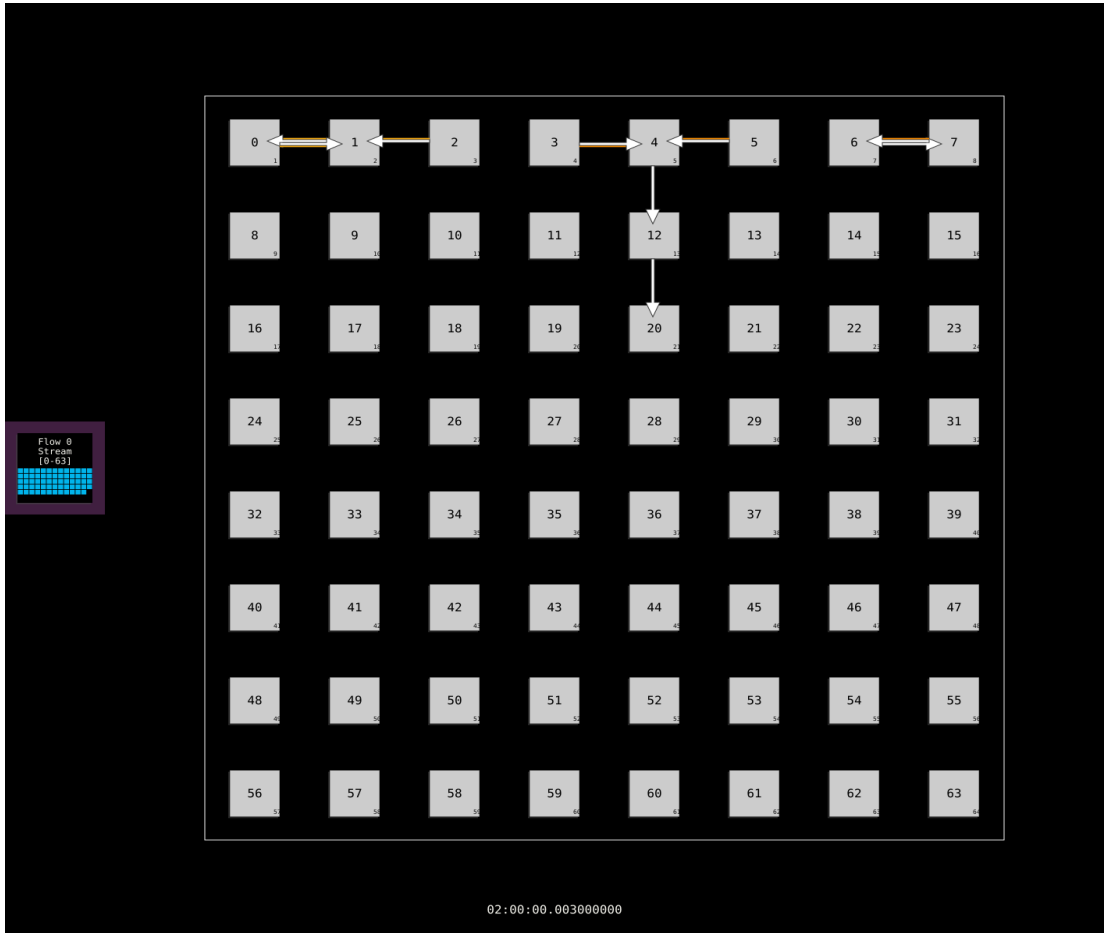
Traces

Let's start by looking at traces, this is the 16 process trace, which is still doing quite fine.



Traces

This is the 64 process trace, which is roughly 4x slower.



We can clearly see a load balancing issue, the processes closer to the particle source are doing more work than the rest.

Messages

Let's also look at the number of messages sent, if that is causing an issue.

We can get the number of messages sent (from trace file)

- 4 processes: 28875
- 16 processes: 47111
- 64 processes: 83600

The runtime difference from 16 to 64 processes is a 4x, while the number of messages is 2x. However, the number of messages sent might have nonlinear effects on the runtime.

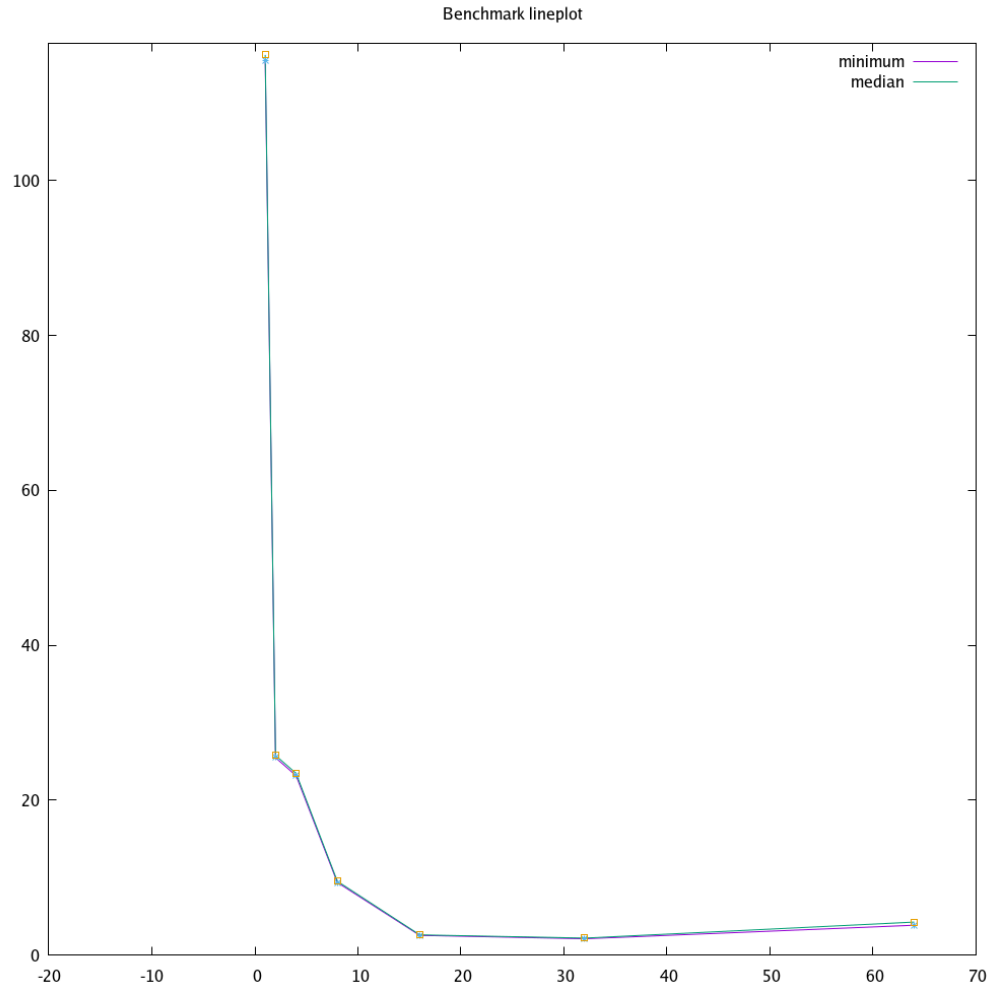
More processes = more messages

It's possible that we're simply sending more messages and that is killing performance. It's also possible that at 64+1 processes, the program has to run on two CPU sockets, which means that MPI has to send data along a slower route.

To check this, let's take the last point on the previous graph and run a strong scaling experiment. Start with one simulator and double the number of simulators and subdomains until we get to 64, which matches the final point on the previous slide.

Performance study: strong scaling

Strong scaling from 1 simulator to 64 simulators.



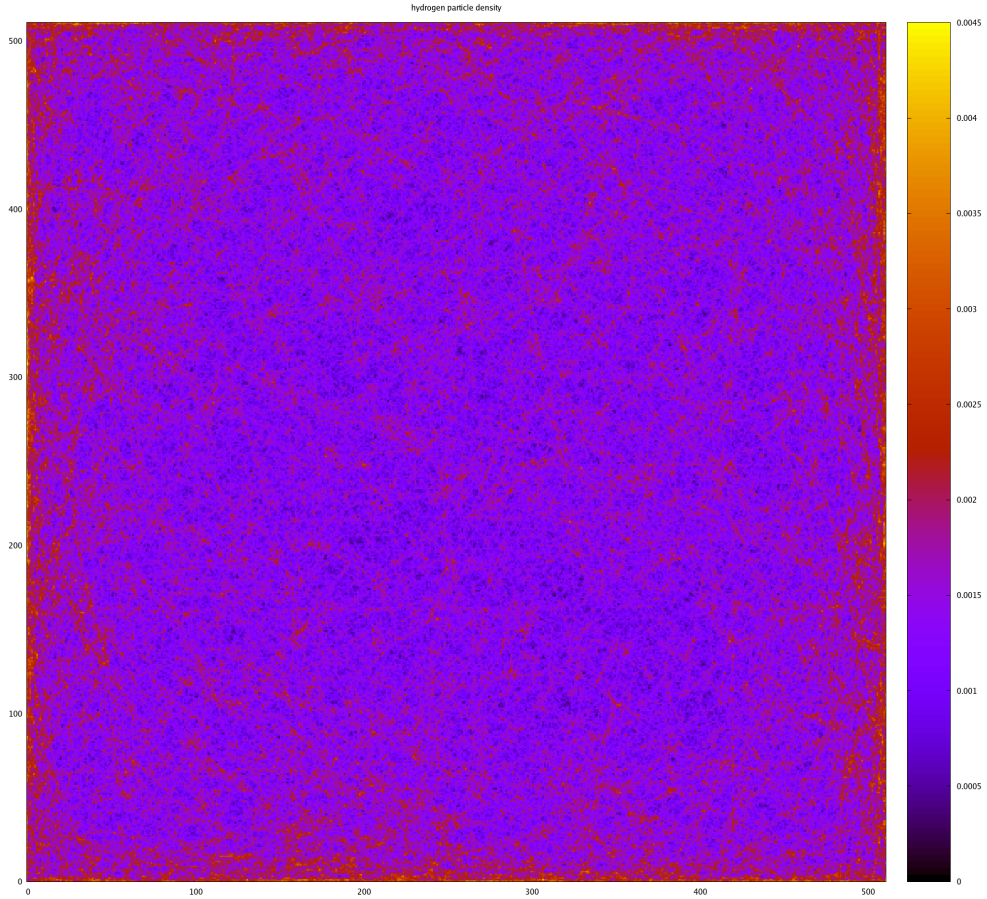
Indeed, 16 or 32 processess is much better (almost 2x faster).

Maybe the problem is communication?

Scaling case 2

Performance study: weak scaling

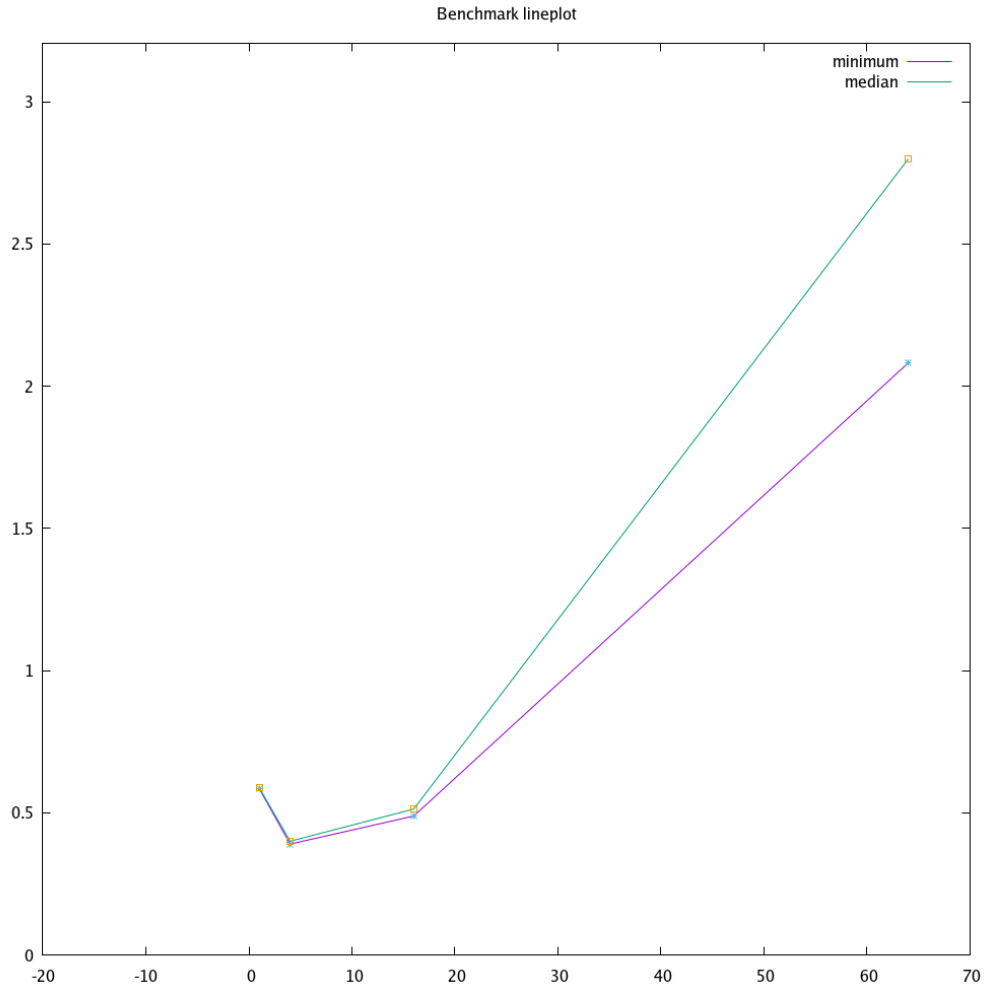
We can investigate the load imbalance effect from the traces. Let's create a more balanced simulation case.



Here we're putting a source on every boundary, instead of just at the bottom. How does this look?

Performance study: weak scaling

This is indeed better.



Up to 8 or 16 procs, runtime is flattish, but then we slow down.

Clearly some other effect is slowing us down.

Scaling: interpretation

After discussion during the code camp, it's clear that the original weak scaling case was not well formed:

- The MFP was too low
- The number of particles also has to scale with the resolution

I will run a new weak scaling test taking these facts into account, which I believe should look much better.

Clearly load balance is one big factor.

The number of subdomains can also have a big effect.

Finally, there are parameters in charon that can't currently be adjusted at runtime, like message queue sizes. The effect of these parameters should also be investigated.

Thank you

Questions/Comments