Parallelizing Monte Carlo Neutral Transport

Oskar Lappi

May 2025

Issue: performance & scalability

EIRENE was written in the early 1980s, when JET was being built.
ITER is 10× the volume of JET, and the plasma is more collisional.
EIRENE doesn't implement domain decomposition (DD).

More collisions in ITER sims	$\Longrightarrow$	More work for the kinetic model
ITER is 10× bigger	$\Longrightarrow$	Higher resolution needed
Higher res. grids & no DD	$\Longrightarrow$	Growing process memory footprint
	$\Longrightarrow$	EIRENE is the bottleneck in ITER sims

Legacy software is tricky to work with

- EIRENE is tightly coupled with a number of plasma solvers
- EIRENE supports a rich set of parameters and grid types
- EIRENE simulates a single particle history from start to finish in one go — which is an assumption that will be present in much of the code base

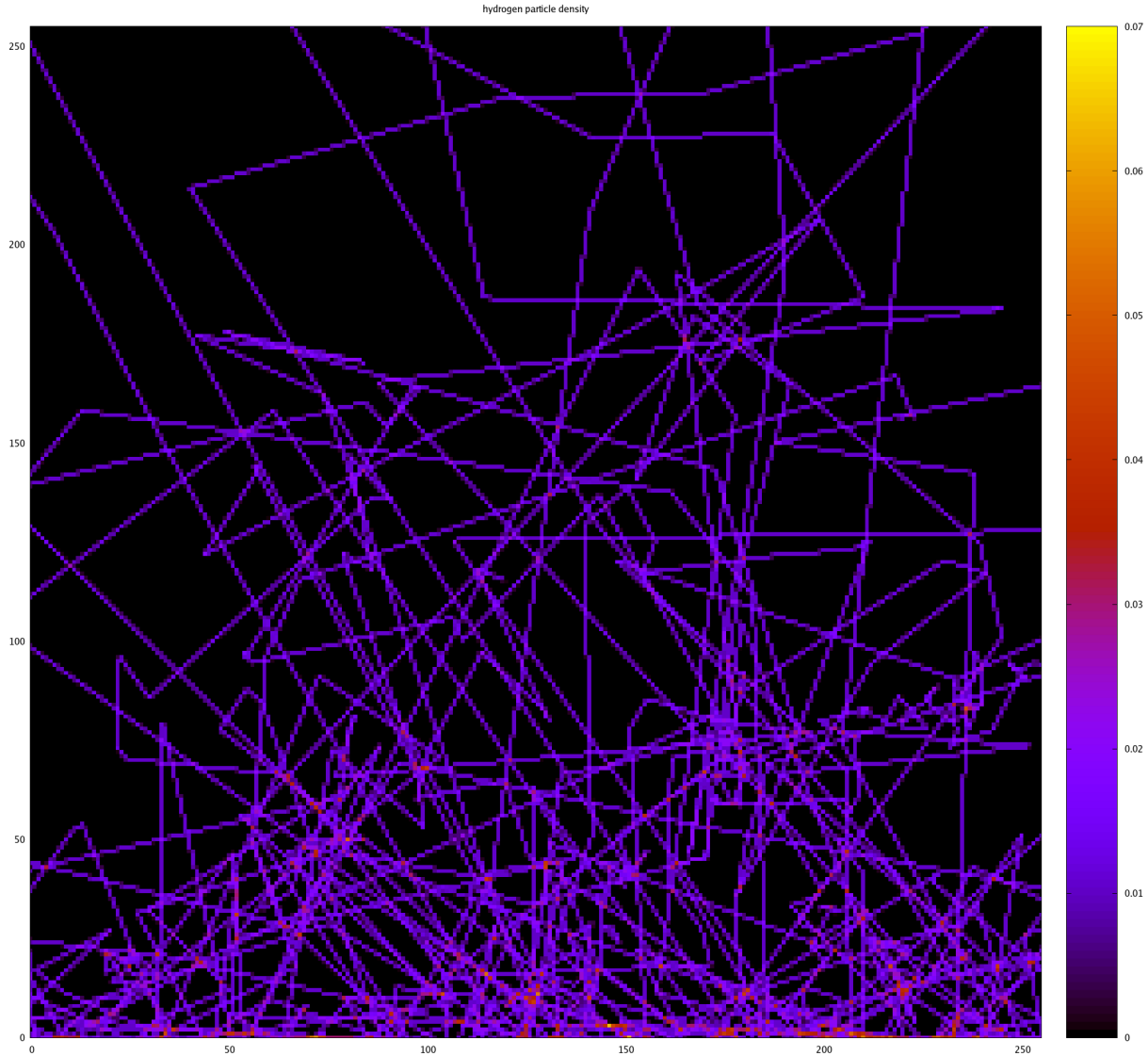
It's tricky to implement a big feature like domain decomposition while making sure not to break any of this, so I've done it in a new code: Eiron.

High-level algorithm for simulating one particle

1. Sample particle from a source distribution
2. Follow the trajectory of a particle, integrating over the cross section until the sum equals an RNG exponential variable
3. Estimate particle density, etc., along the trajectory
4. Sample an outcome of a collision event at that point
5. Repeat 2,3,4 until particle is terminated

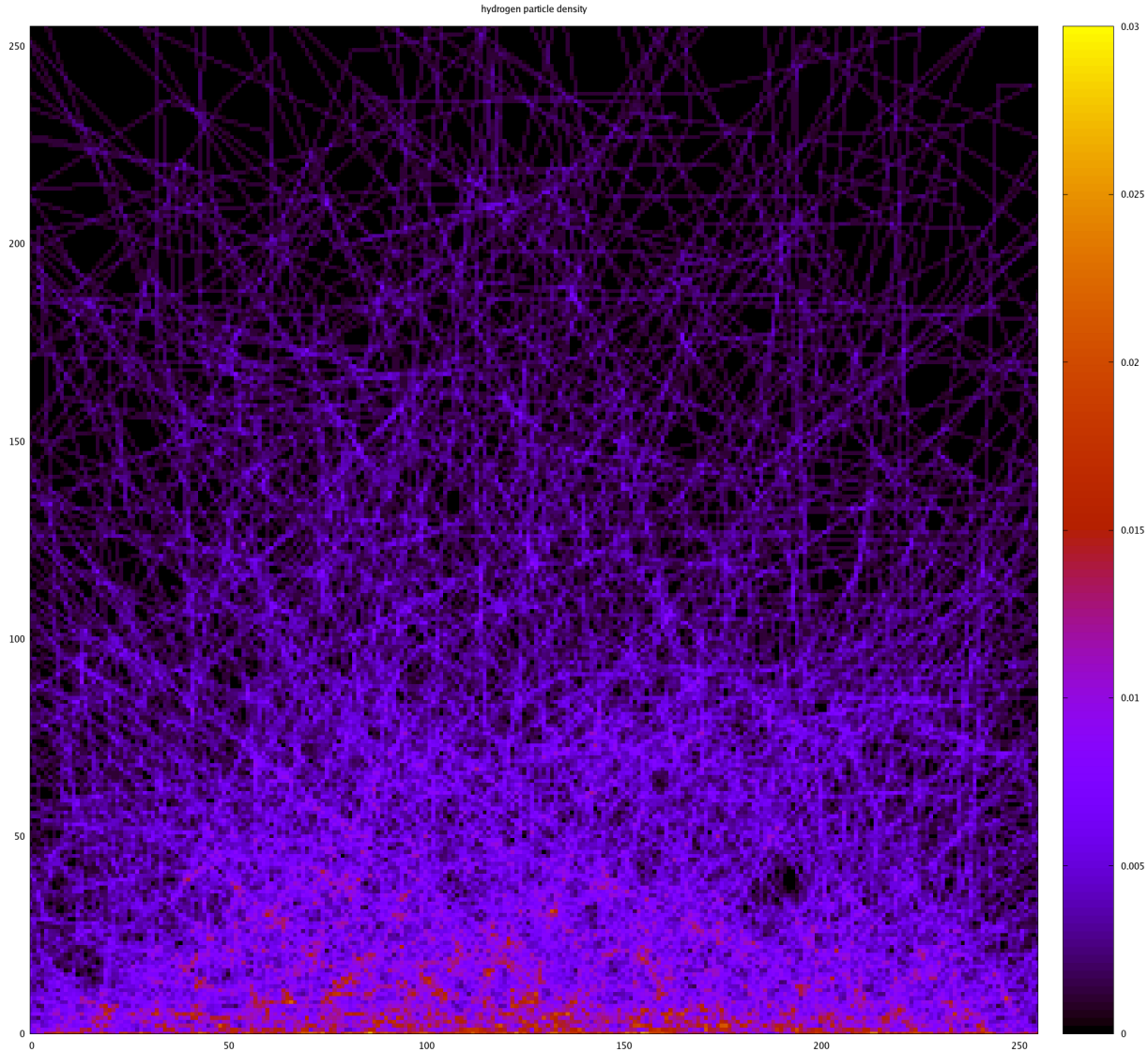
Monte Carlo transport

Particle density, 256^2 grid, 100 samples (17 ms, 128 cores)



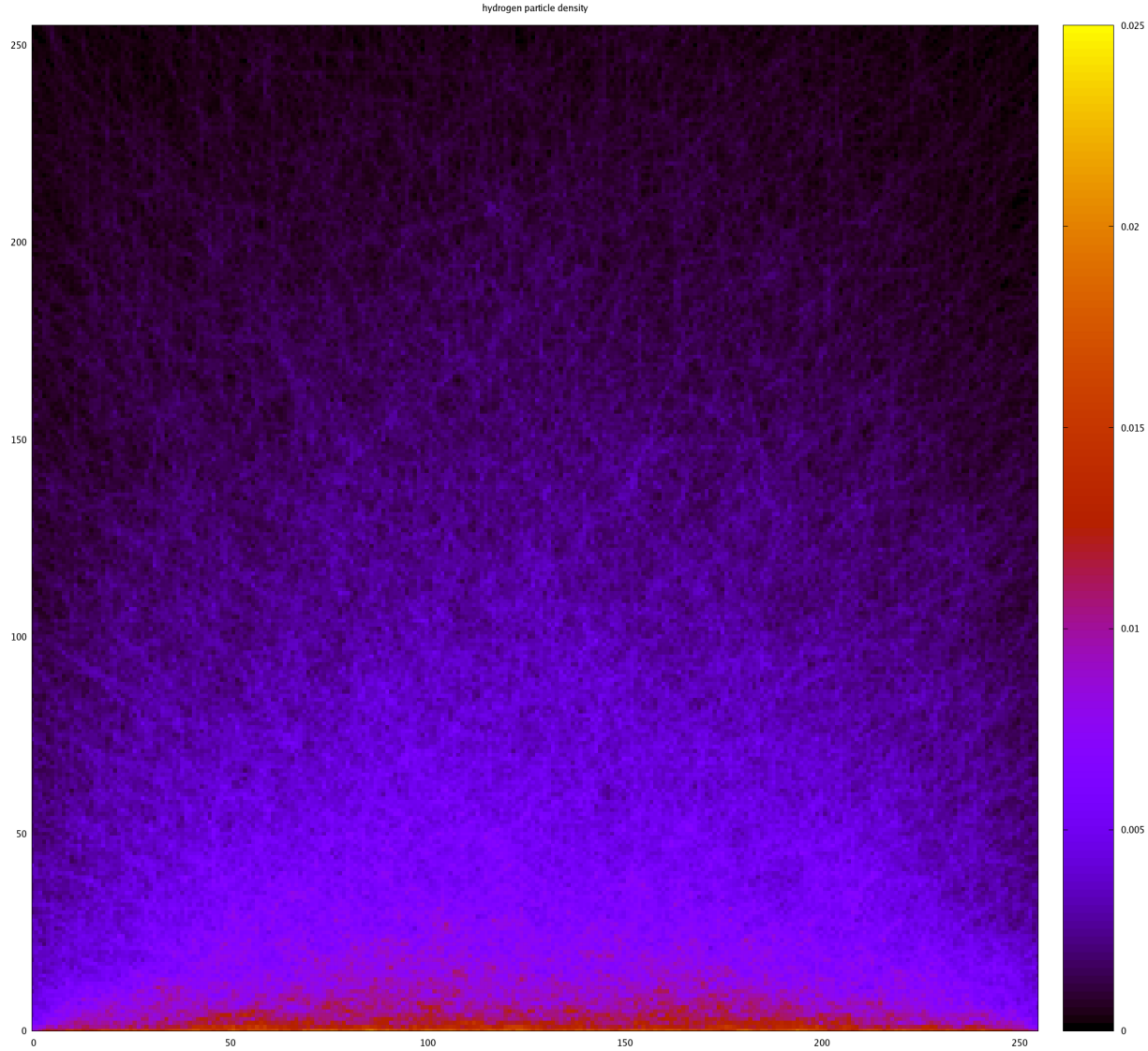
Monte Carlo transport

Particle density, 256^2 grid, 1K samples (23ms, 128 cores)



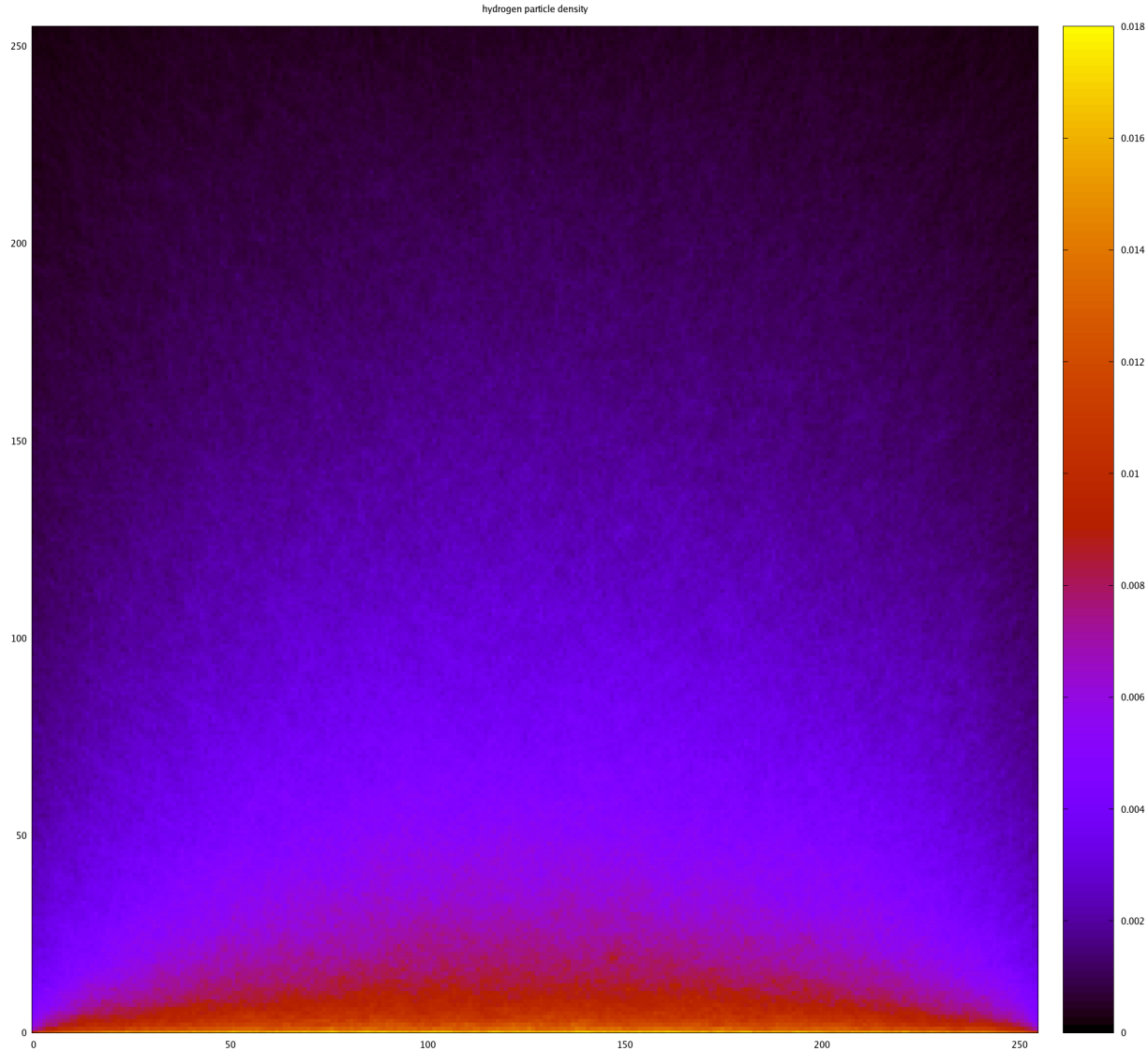
Monte Carlo transport

Particle density, 256^2 grid, 10K samples (42 ms, 128 cores)



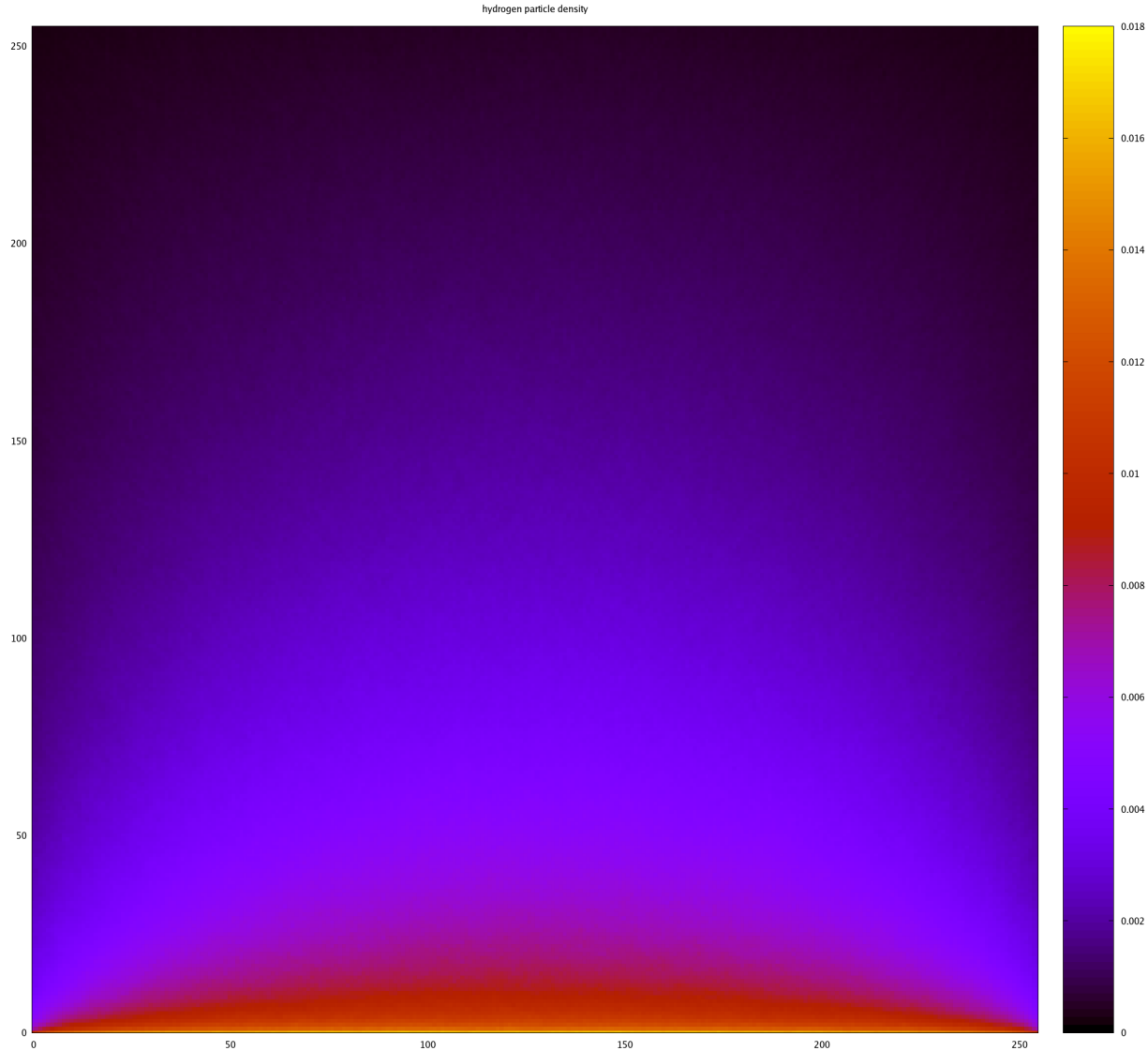
Monte Carlo transport

Particle density, 256^2 grid, 100K samples (220 ms, 128 cores)



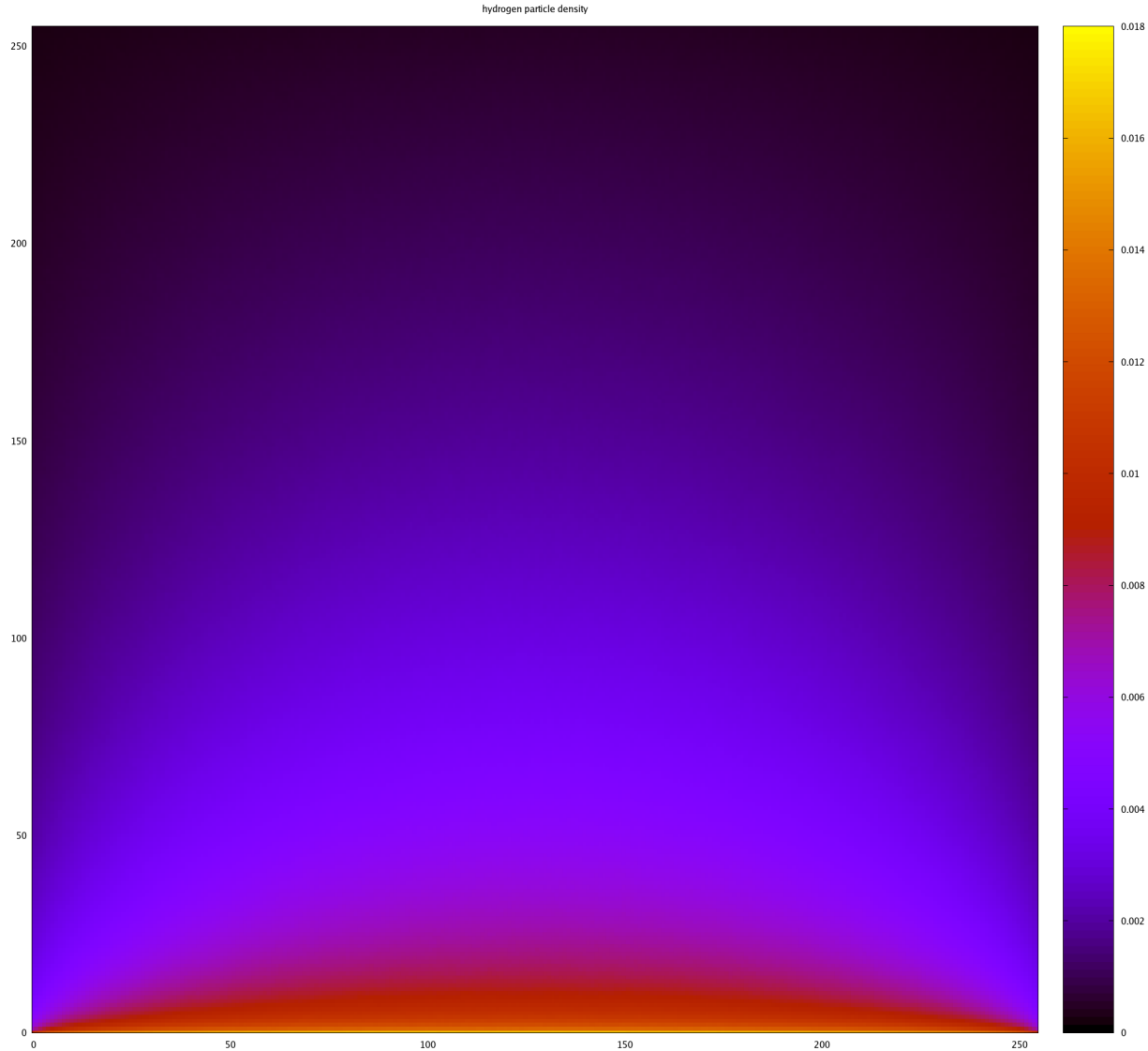
Monte Carlo transport

Particle density, 256^2 grid, 1M samples (1.9 s, 128 cores)



Monte Carlo transport

Particle density, 256^2 grid, 10M samples (18 s, 128 cores)



Technologies used

Eiron is written using C++, OpenMP, and MPI.

OpenMP is a preprocessor-based framework, preprocessor pragmas are used to specify parallel regions where many threads execute simultaneously, typically you divide work in a big loop among many threads.

MPI is a high-performance message passing standard where, typically, a single program is executed by multiple processes with the same communication context for passing messages.

Parallel algorithms in Eiron

OpenMP-shared

Particle ranges are assigned to OpenMP threads.
OpenMP threads share one estimation grid.

OpenMP-private

Particle ranges are assigned to OpenMP threads.
Each OpenMP-thread has a private estimation grid.

MPI-domain-decomposed

MPI ranks are assigned subdomains.
Particles crossing subdomain boundaries are sent asynchronously to neighboring ranks.

Algorithm OpenMP-shared

for each source in sources **do**
    for particles in source, in parallel over threads **do**
        Simulate particle, estimate into shared estimation grid

This is how EIRENE does OpenMP. It requires all writes to the estimation grid to be done with atomic increments.

There's also cache communication overhead/*cache thrashing*: when thread/core B tallies to a cache line that thread/core A has previously tallied to, the cache line has to be moved:

core A cache $\rightarrow$ main memory $\rightarrow$ core B cache

Because B then writes to the cache line, the cache line instance in core A is invalidated; if A goes to tally to the cache line, the same happens in reverse.

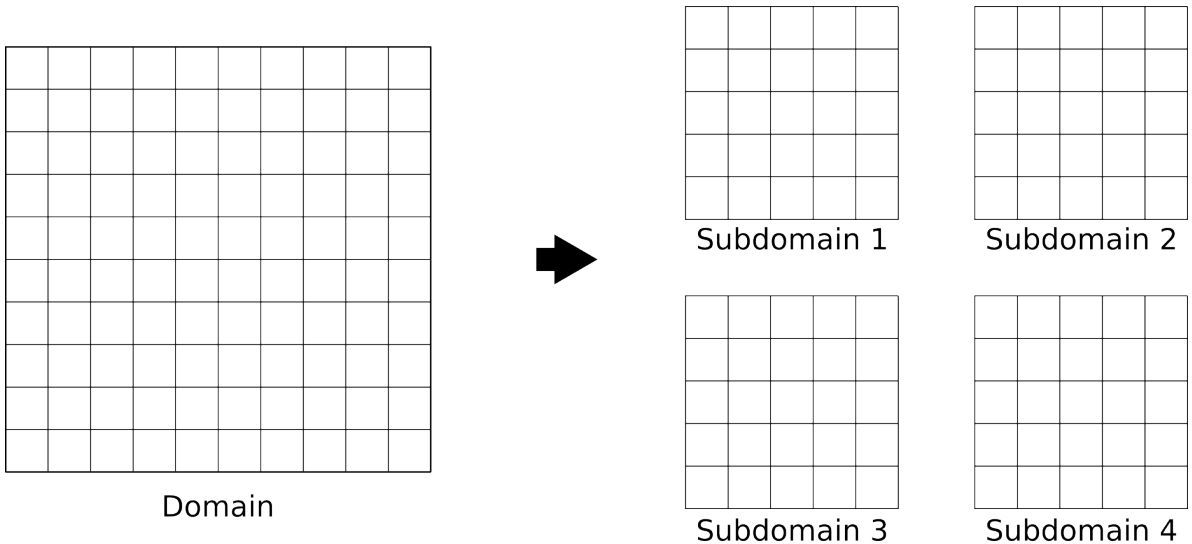
Algorithm OpenMP-private

```
In an OpenMP parallel region
Create thread-private copy of estimation grid
for each source in sources do
    for particles in source, in parallel over threads do
        Simulate particle, estimate into private estimation grid
Reduce estimation grids
```

This one does not have the issues of the shared memory tallies, but the memory use does scale linearly with the number of threads.

Domain decomposition

Domain decomposition is what you reach for when your simulation space is too big to efficiently simulate in a single process.



Domain-decomposed Monte Carlo has been done before, but, in all papers I've read, communication is done using synchronous or semi-asynchronous multi-particle buffer messages.

Eiron sends asynchronous single-particle messages using queues of message buffers and MPI request objects.

Algorithm MPI-domain-decomposed

```
Partition grid and assign subdomains to MPI ranks
Find local particle sources (overlapping a local subdomain)
Mark all ranks as active
while there are active particles and active ranks do
    Try to generate particle from local sources
    if no particle generated then
        Try to receive particle
    Simulate particle (if there is one)
        if particle crossed subdomain boundary then
            Send particle forward
            Mark particle as active (if locally generated )
        if particle terminated then
            send "particle.id done" to generating rank
    if local sources depleted and no active particles then
        broadcast "local rank done" to all ranks
    Receive "particle done" and "rank done" messages
    Mark those particles and ranks as inactive
```

Strong scaling experiments

- 6 grid resolutions: $128^2, 256^2, 512^2, 1024^2, 2048^2, 4096^2$
- 2 mean free paths: 0.05 grid lengths, 0.25 grid lengths
- 3 scattering fractions: 0.01, 0.5, 0.99
  - the scattering fraction $= \frac{\#\textit{Scattering events}}{\#\textit{All bg collision events}}$

The core count is doubled 7 times & there are 3 parallel algorithm $\implies$ there are $6 \times 2 \times 3 \times (1 + 3 \times 7) = 792$ configurations.

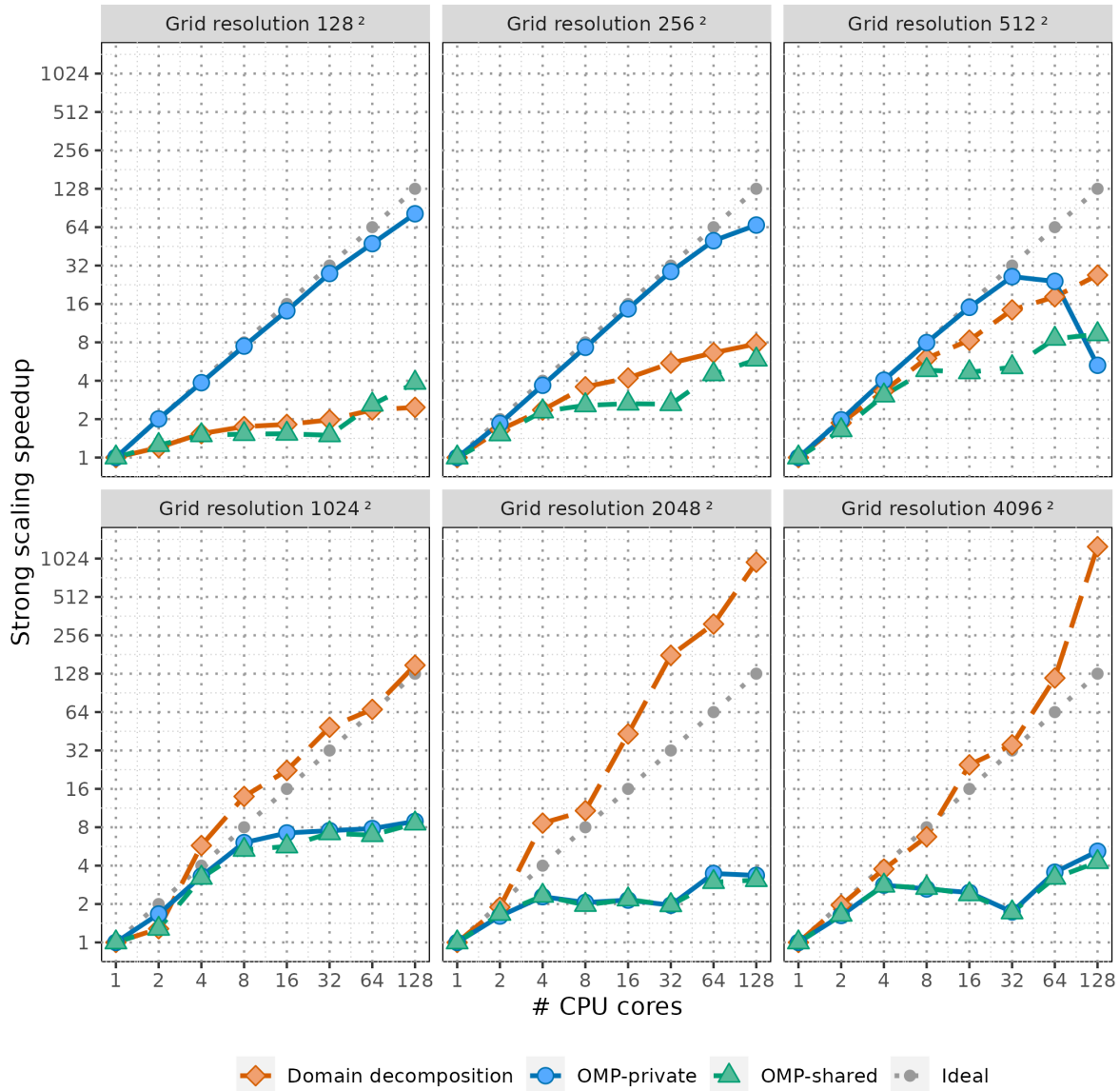
We run each config 5 times and report the minimum.

The strong scaling experiments were run on CSC's Mahti, where one node contains two 64-core AMD Rome 7H12 CPUs.

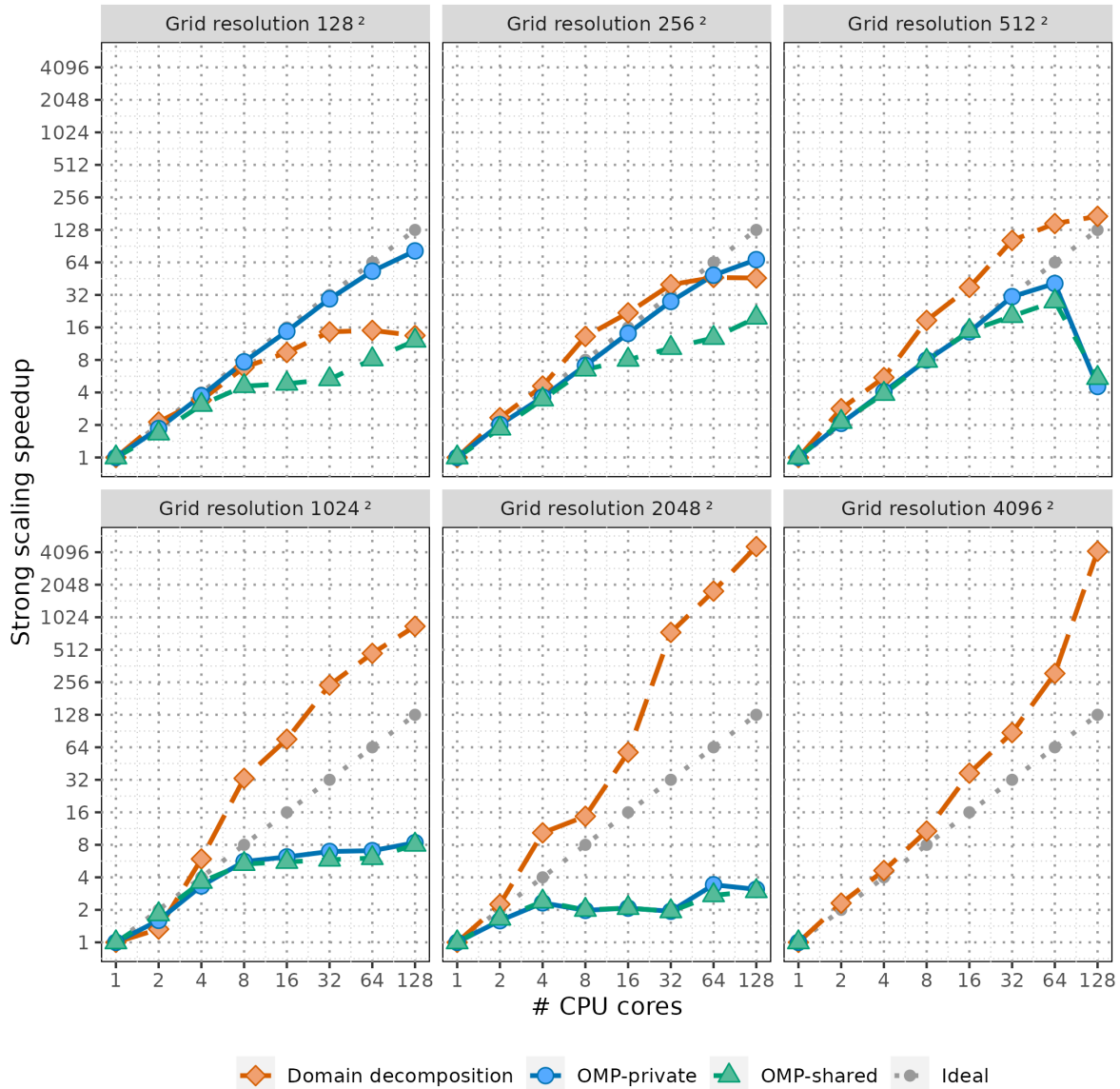
In the interest of time, let's focus on two extremes:

- $\lambda = 0.25$, *scattering fraction* $= 0.01$
- $\lambda = 0.05$, *scattering fraction* $= 0.99$

$\lambda = 0.25 \cdot \mathit{grid\ length},\ \mathit{scattering} = 1\%$



$\lambda = 0.05 \cdot \mathit{grid\ length},\ \mathit{scattering} = 99\%$



Superlinear scaling

Why is domain-decomposition scaling superlinearly? L3 cache. In AMD's Zen 2 architecture, four cores share one L3 cache with 16 MiB.

Velten et al., 2022, measured the L3 cache latency of a Zen2 CPU to be 20 ns, and main memory latency to be 110 ns.

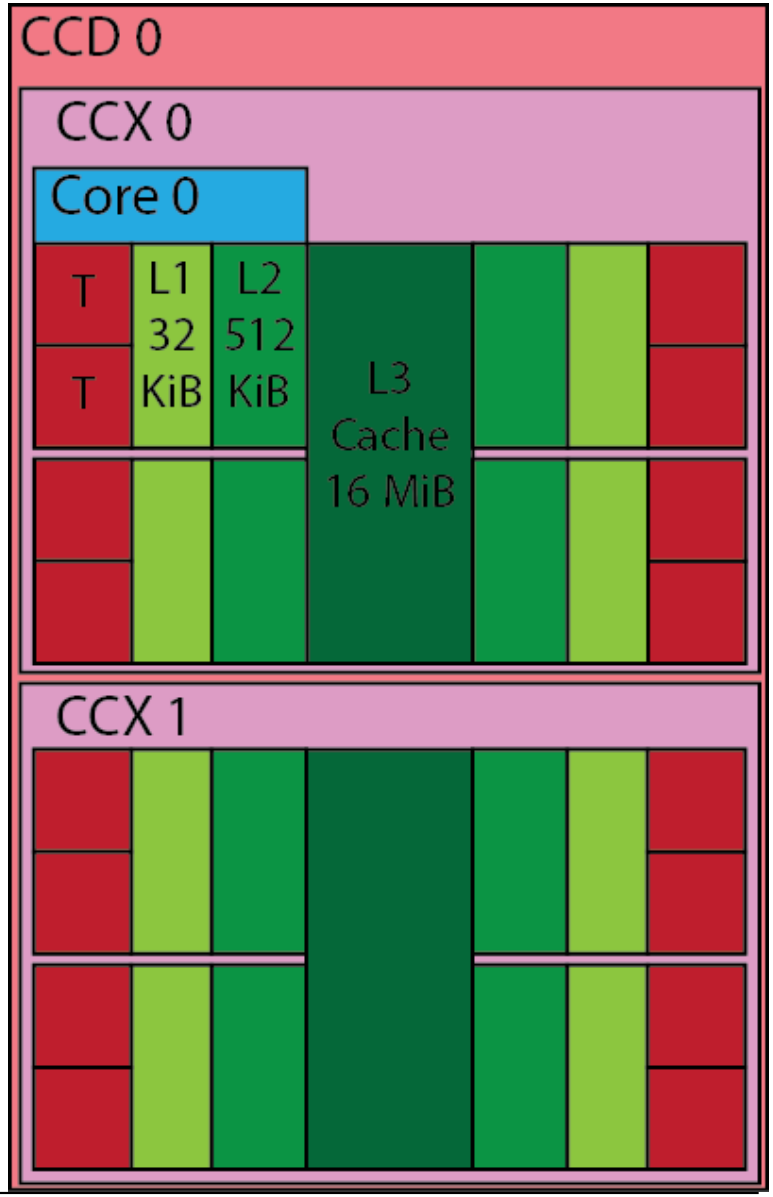


Image from: Docs CSC, `https://docs.csc.fi/computing/systems-mahti/`

Superlinear scaling

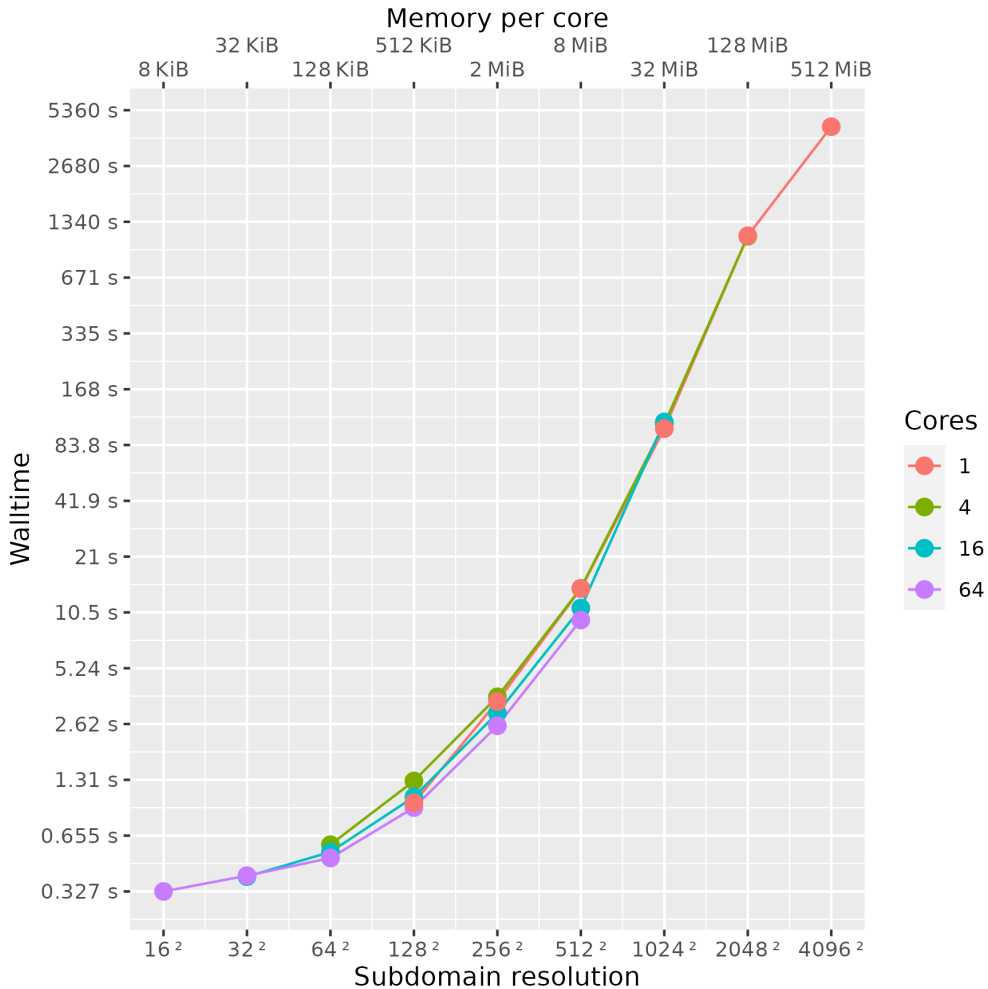
I did not have the foresight to record cache misses or hits in my benchmarks, but we can model cache miss ratios of Monte Carlo grid access on this CPU architecture, and we can see that 512^2 grids fit into L3 completely, removing the cost of cache misses.

Grid resolution	Mem. footprint	shared L3 miss ratio	L3 slice miss ratio	L2 miss ratio
128^2	512 KiB	0%	0%	0%
256^2	2 MiB	0%	0%	50%
512^2	8 MiB	0%	50%	87.5%
1024^2	32 MiB	50%	87.5%	87.5%
2048^2	128 MiB	87.5%	98.4%	99.6%
4096^2	512 MiB	98.4%	99.6%	99.9%

512^2 is also the largest grid resolution that scales linearly, with no (large) superlinear improvement.

Process memory footprint is main performance factor

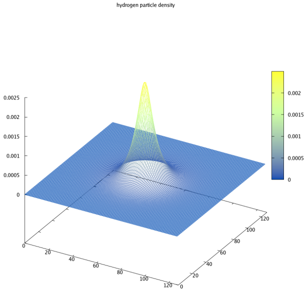
- This is a log-log plot
- Bigger effect than any simulation parameters
- Bigger effect than the number of cores used

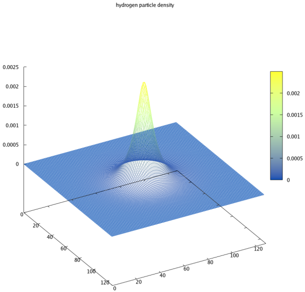


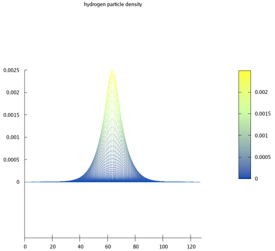
KDMC work

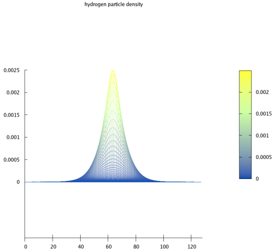
- Collab with Emil Løvbak, Thijs Steel, Giovanni Samaey
- 2D KDMC in Eiron
- Compared it to regular, kinetic, Monte Carlo
- Writing a paper for PET
- Will now show KDMC experiments in high collision regime
- No ionization, cutoff after t=50

Kinetic, 10^{10} particles, $\lambda = 0.035$

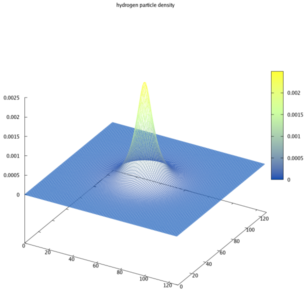


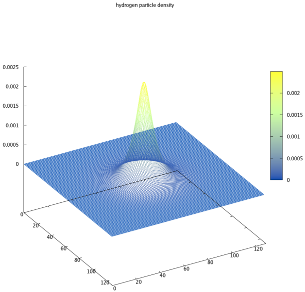


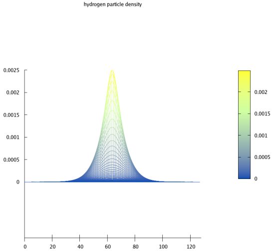


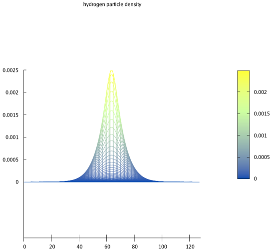


KDMC, 10^{10} particles, $\lambda = 0.035$

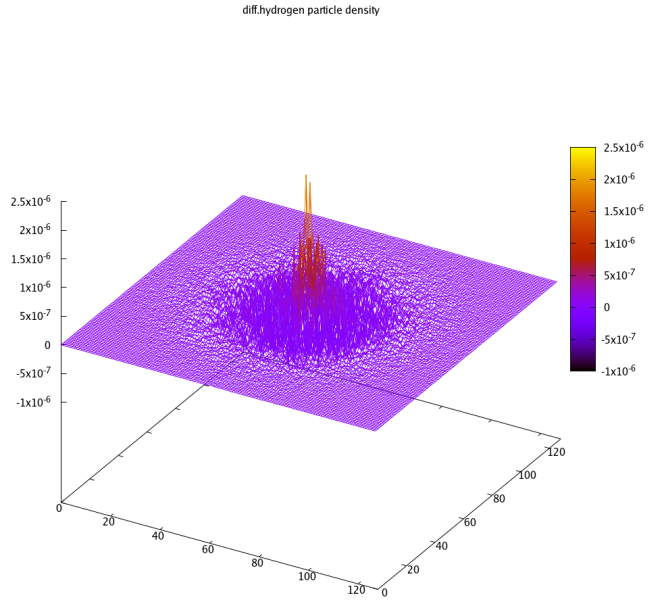


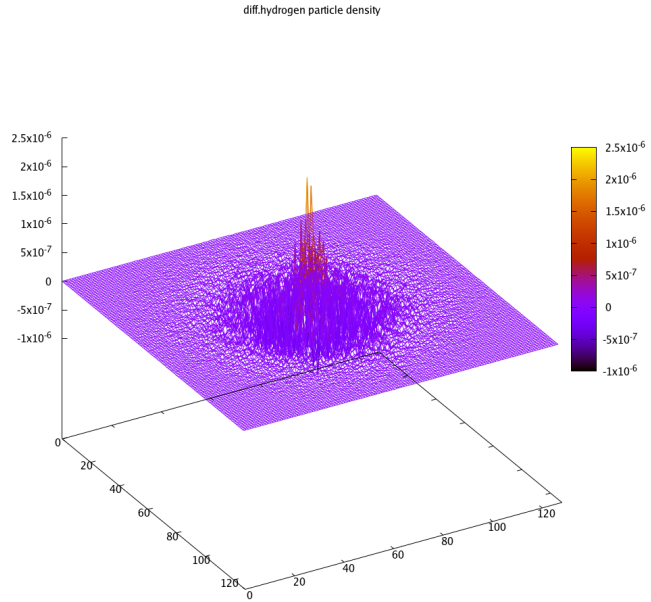


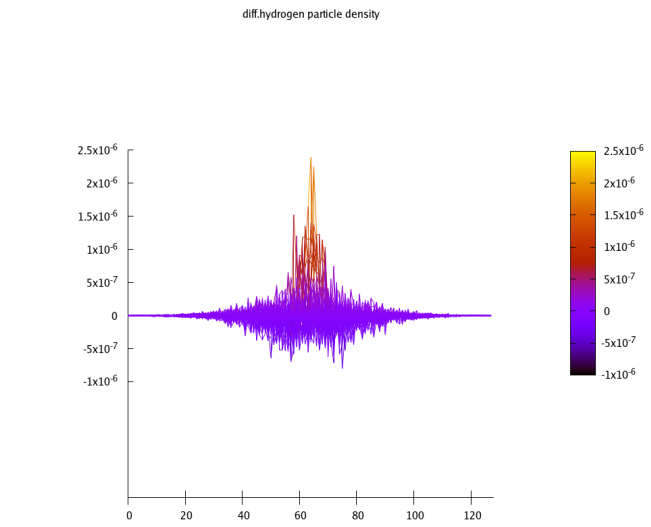


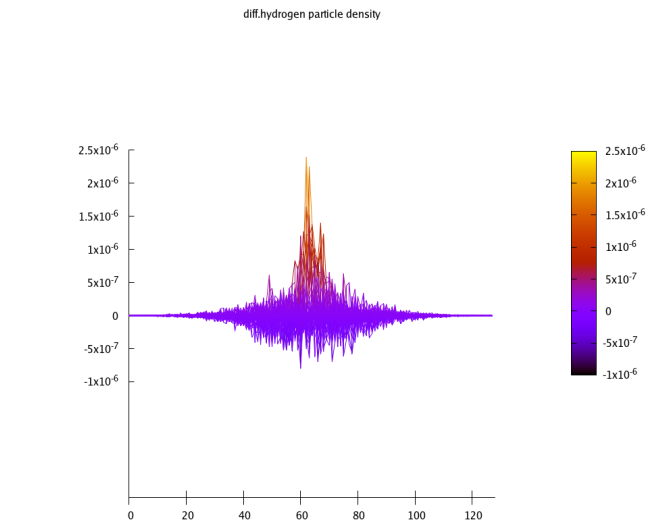


Diff, 10^{10} particles, $\lambda = 0.035$

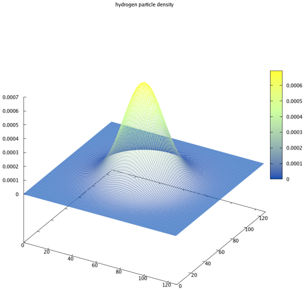


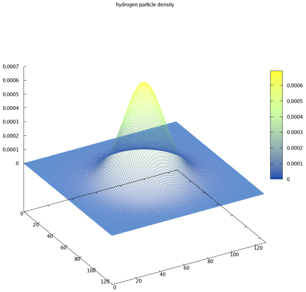


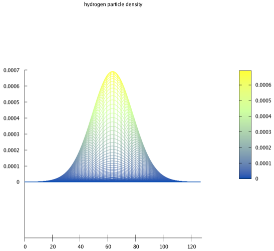


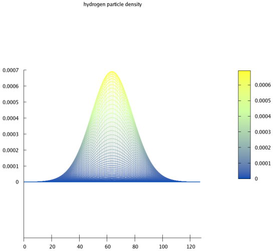


Kinetic, 10^{10} particles, $\lambda = 0.0035$

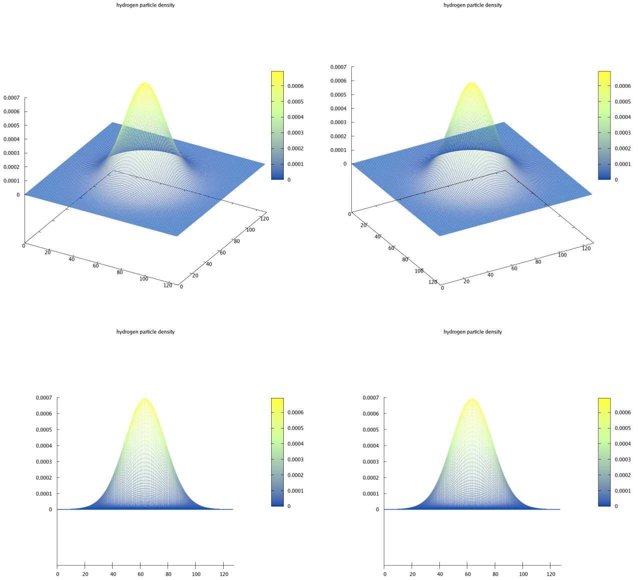




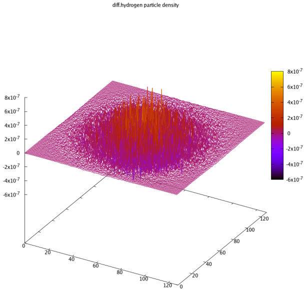


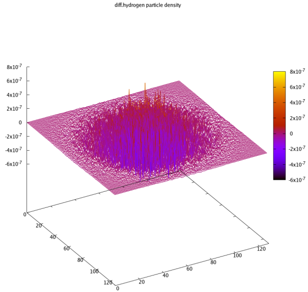


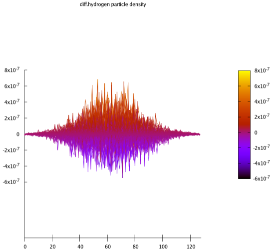
KDMC, 10^{10} particles, $\lambda = 0.0035$

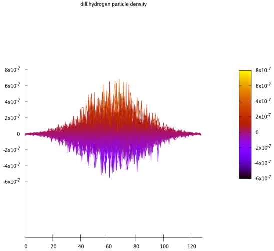


Diff, 10^{10} particles, $\lambda = 0.0035$









Further work

We're writing the main Eiron paper currently, still working on weak scaling experiments and reviewing the literature. Also writing a paper on KDMC with Emil, Thijs, and Giovanni.

I believe this to be a novel domain decomposition algorithm.

The findings comparing different approaches should be valuable for existing Monte Carlo projects as well as new developments.

Future work would look at hybrid OMP/MPI or GPU implementations, also context switching subdomains within a single process.

Where to get it:

Git repo: `https://version.helsinki.fi/lapposka/eiron`
Contact: oskar.lappi@helsinki.fi

Note: there is no user manual, and the physics is very bare bones. Please contact me if you want to use Eiron, I'm interested in working together to develop Eiron into a useful code.

Thank you

Questions/Comments