Dynamic Particle Simulations using GPUs

3 Simulations.

What is a scheduler?

- Launching kernels
- Work distribution
- Parameters
- Coordination

Tests and analysis of schedulers.

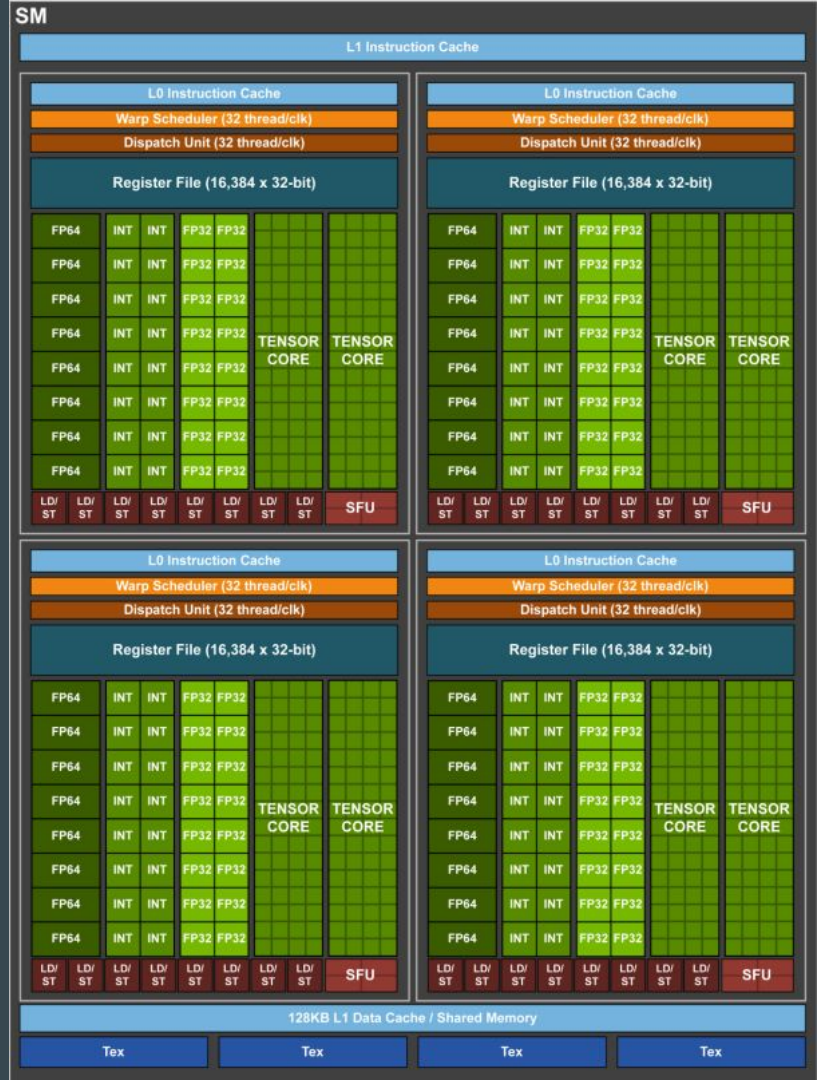
Architecture

Specifications of GPU:

- No. Streaming Multiprocessors (SMs): 80
- No. Processing Blocks pr. SM: 4
- Max Warps pr. Processing Blocks: 16
- No. Threads pr. Warp: 32

Maximum No. Threads pr. SM: 4*16*32=2048

Maximum No. Threads: 2048*80=163.840



Scheduler Tests

Simple simulation to analyse the performance of various scheduling techniques

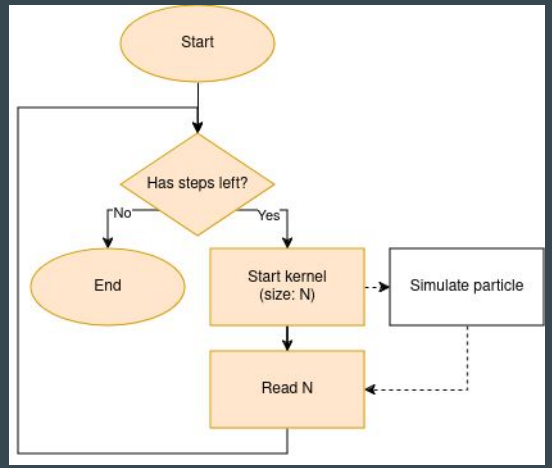
What does the simulation do?

- Moves in a 2D space
- Particles are affected by gravity
- Split when they hit the bottom

Scheduler Tests

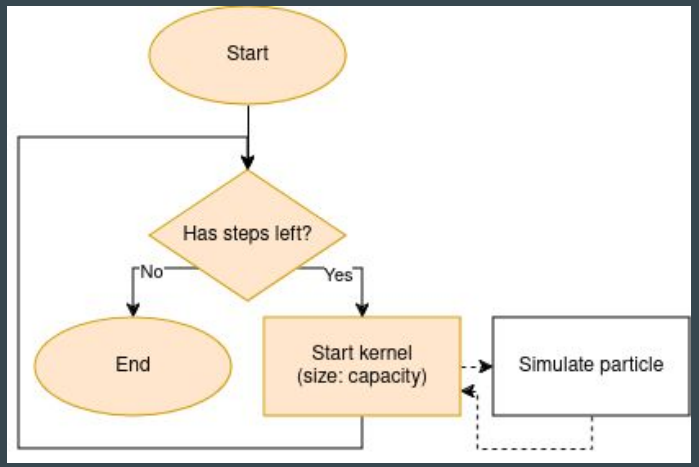
- For all schedulers: Each thread handles one particle

CPU Sync Iterate



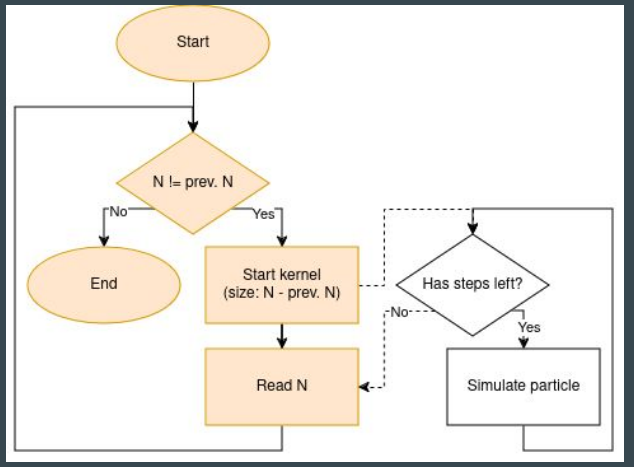
1 thread pr. particle currently being simulated

Huge Iterate



1 thread pr. possible particle

CPU Sync Full

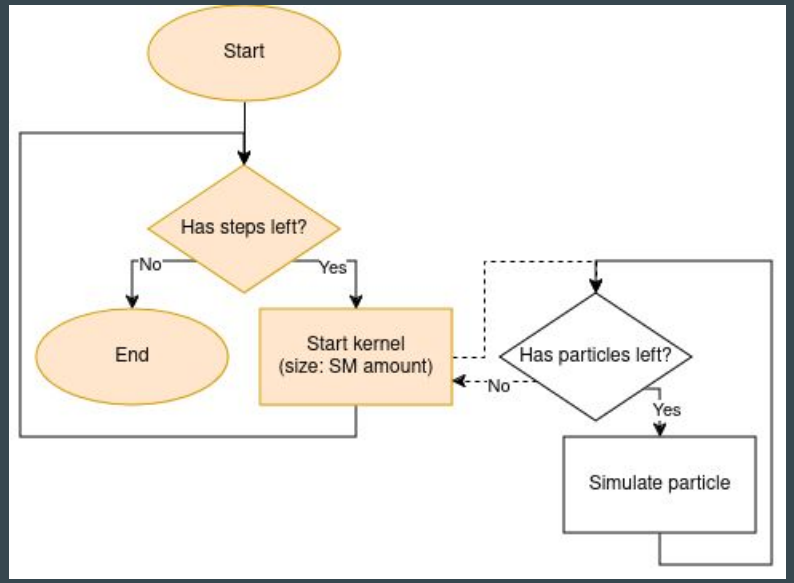


Each thread simulates particle fully

Scheduler Tests

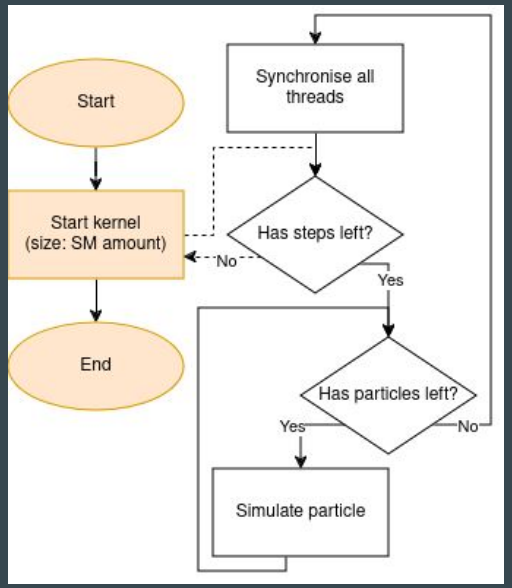
- For all schedulers: Each thread handles multiple particles

Static Simple Iterate



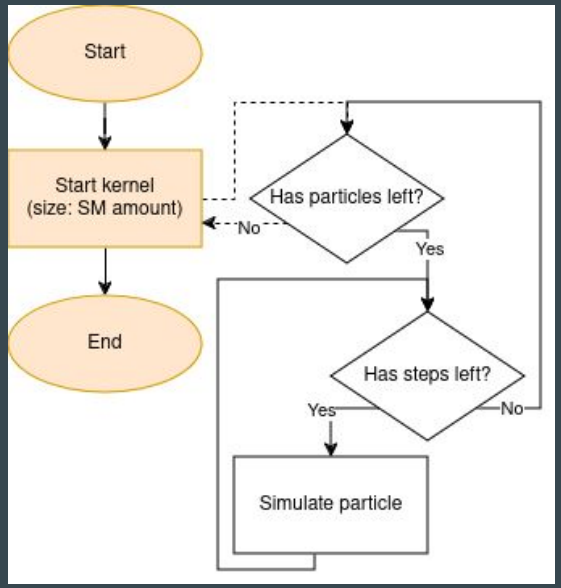
Extension of CPU Sync Iterate

Static GPU Iterate X



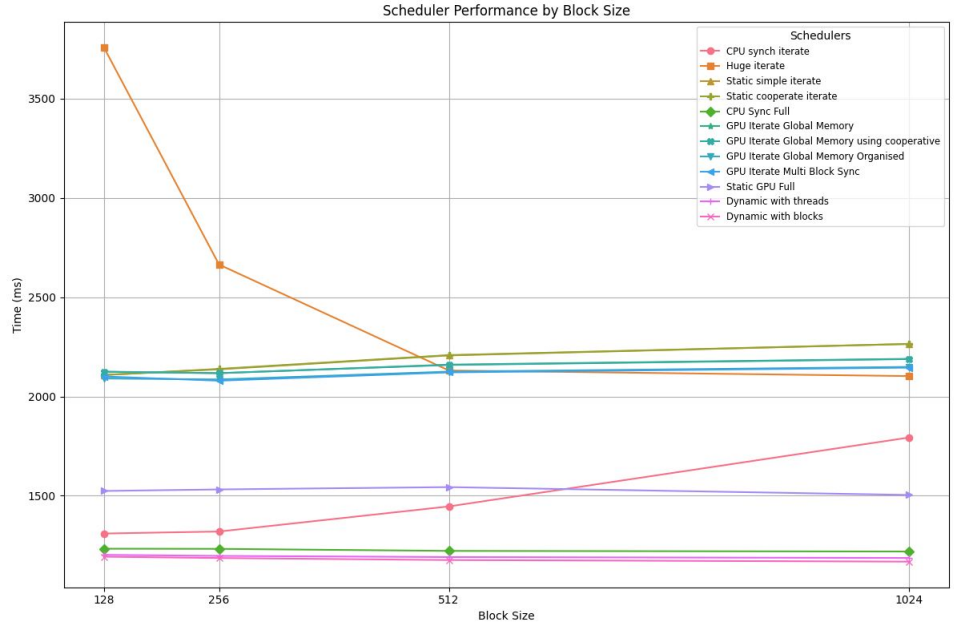
Syncs on GPU

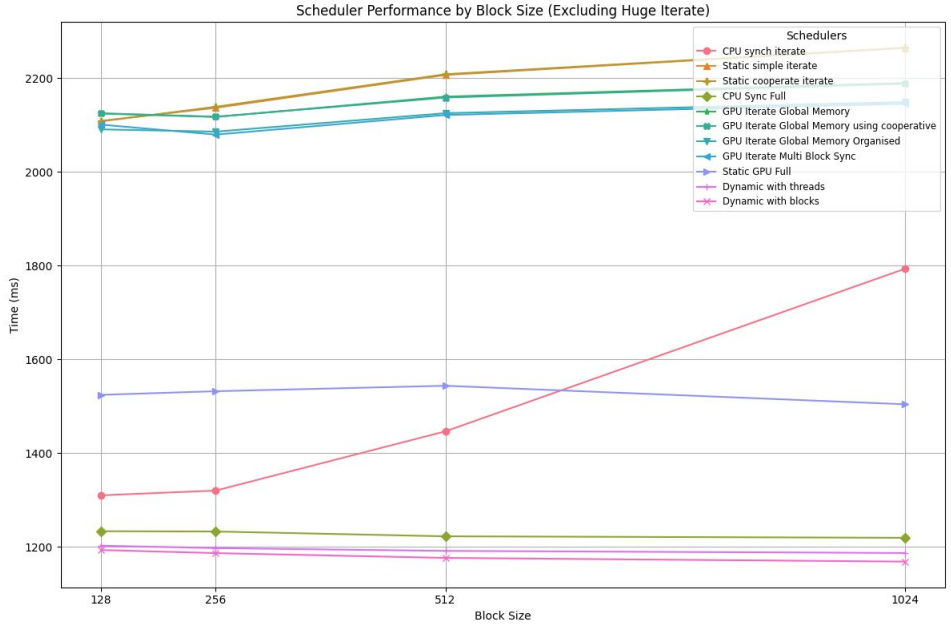
Static GPU Full + Dynamic



Each thread simulates particle fully

Scheduler Tests: Block Size





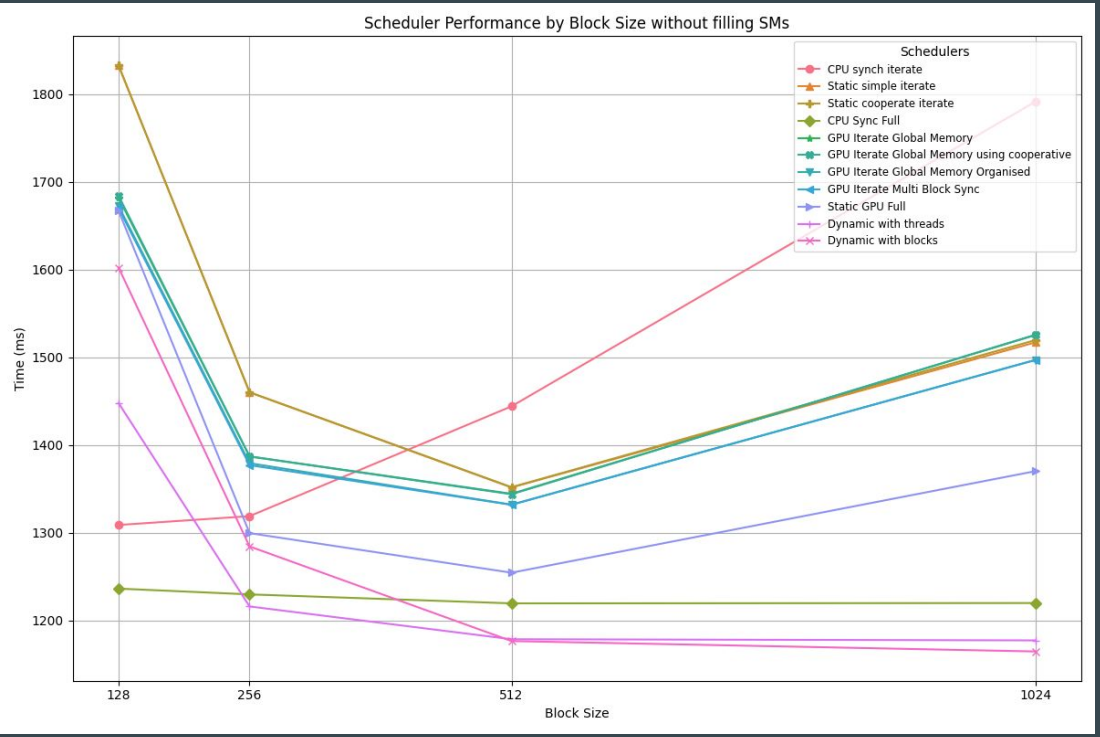
Huge Iterate:

- As many threads as capacity
- More blocks result in more scheduling work

CPU Sync Iterate:

- As many threads as current particles
- Slow threads cause a more severe bottleneck

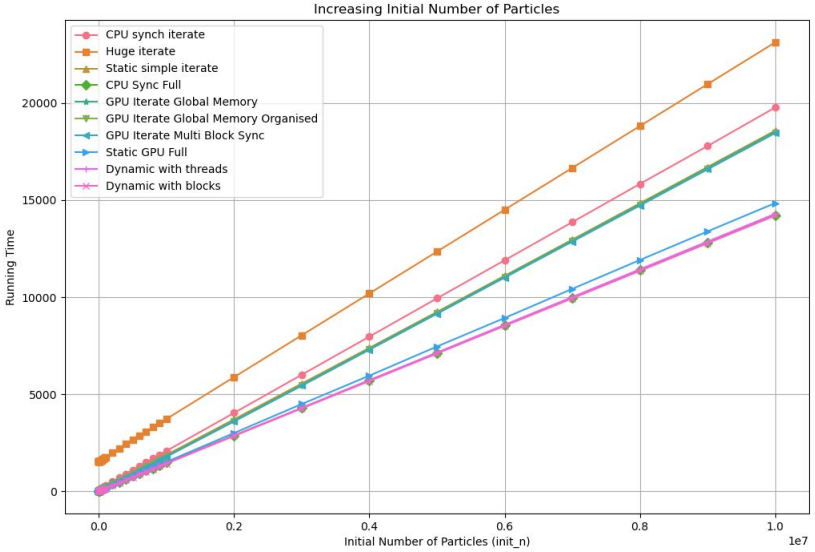
Scheduler Tests: Block Size without filling SMs

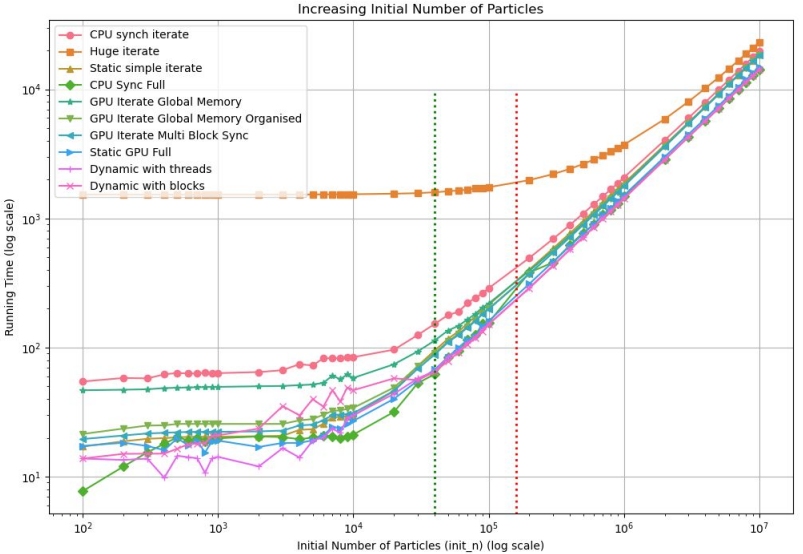


Improvement of the Statically Implemented Iterate Schedulers:

- Fewer threads with more work > more threads with less work
- All threads synchronise after each time step
- Extra scheduling work

Scheduler Tests: Initial Number of Particles

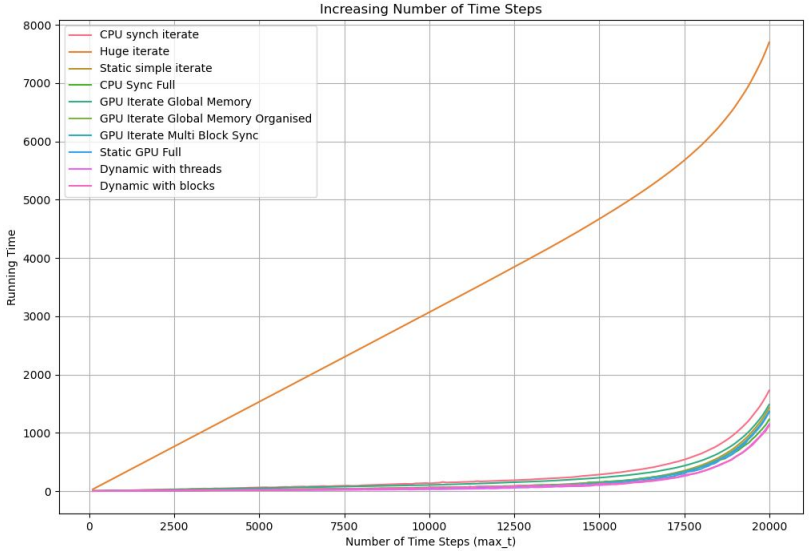


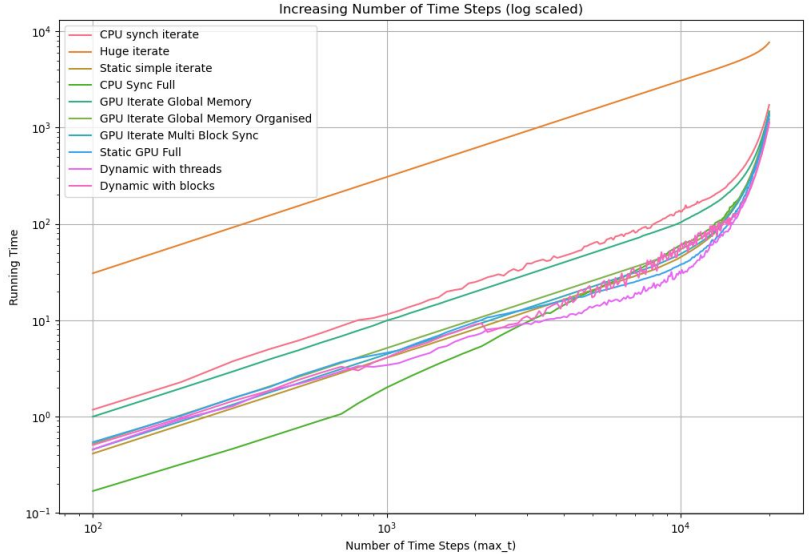


Linear Growth

Constant until SMs are filled

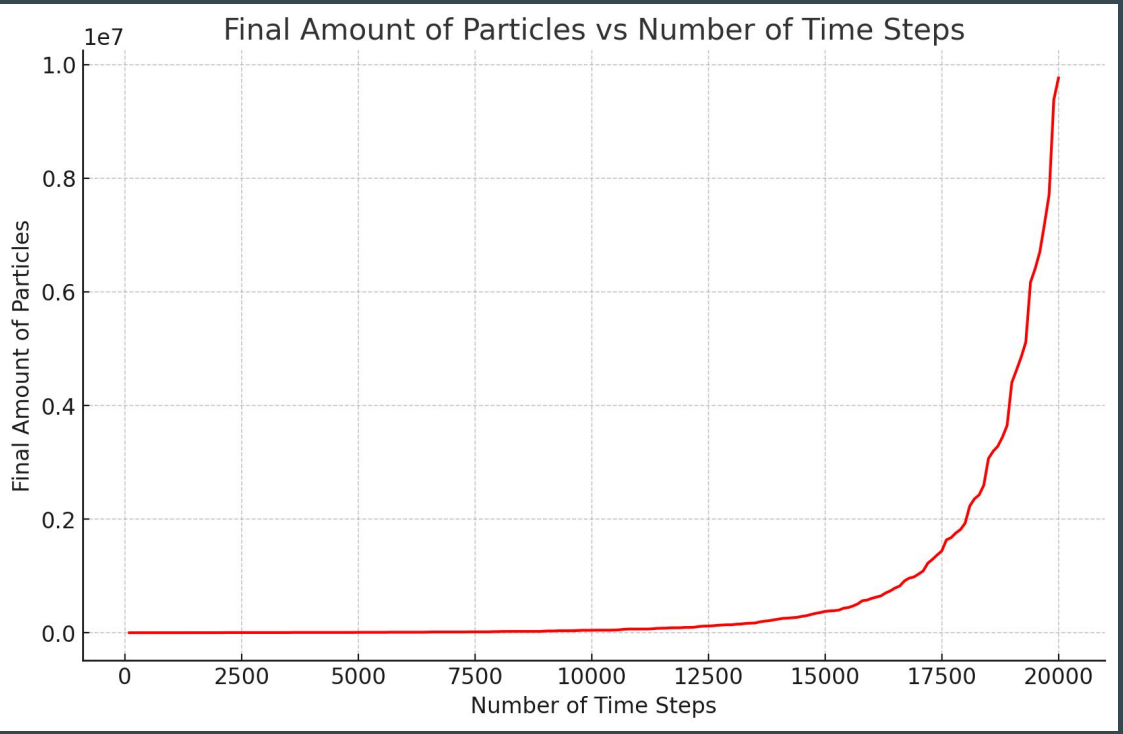
Scheduler Tests: Number of Time Steps





Linear until exponential
Exponential when SMs are filled

Scheduler Tests: Number of Time Steps



Exponential growth in number of particles

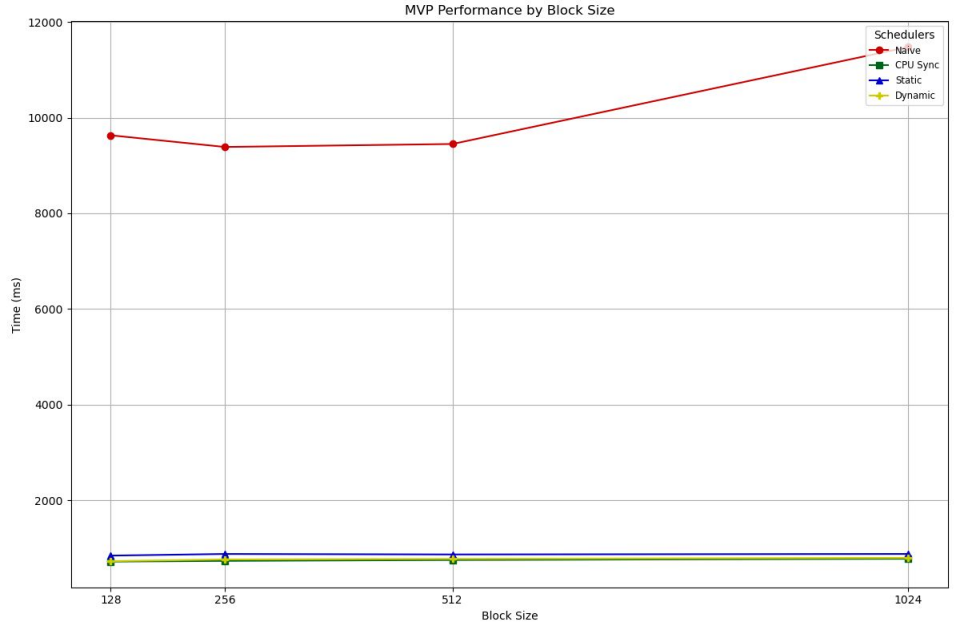
Minimal Viable Product (MVP)

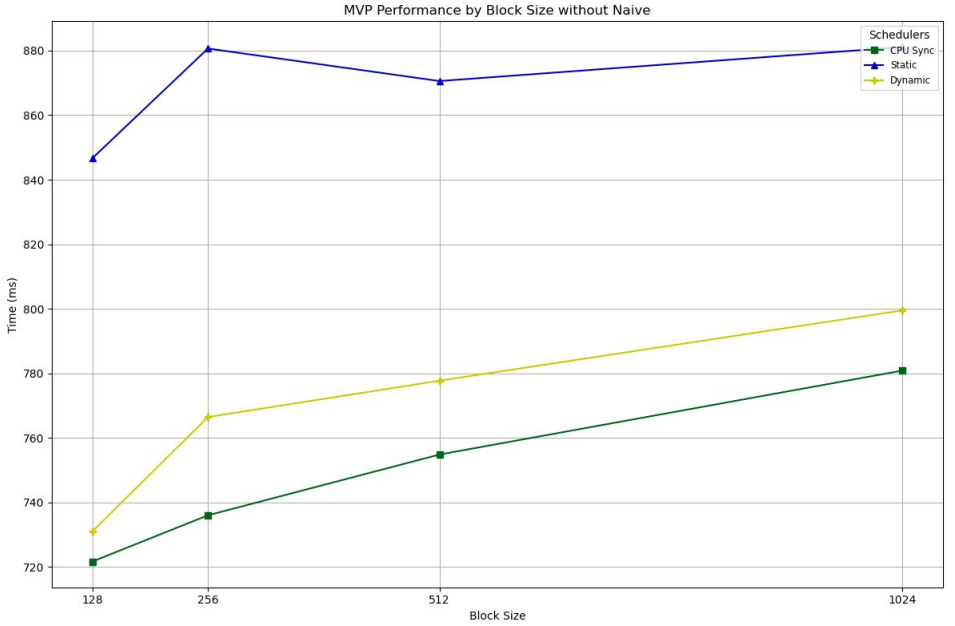
Naive	Based on CPU Sync Iterate. Optimisations: Shared memory and memory organising.
CPU Sync	Same as CPU Sync Full.
Static	Based on Static GPU Full. Optimisations: fewer atomicAdd-operations.
Dynamic	Based on Dynamic With Blocks. Optimisations: Shared memory

Simulation changes:

- Chance to split every step instead of on bounce.
- Changes to particle movement.

MVP: Block Size



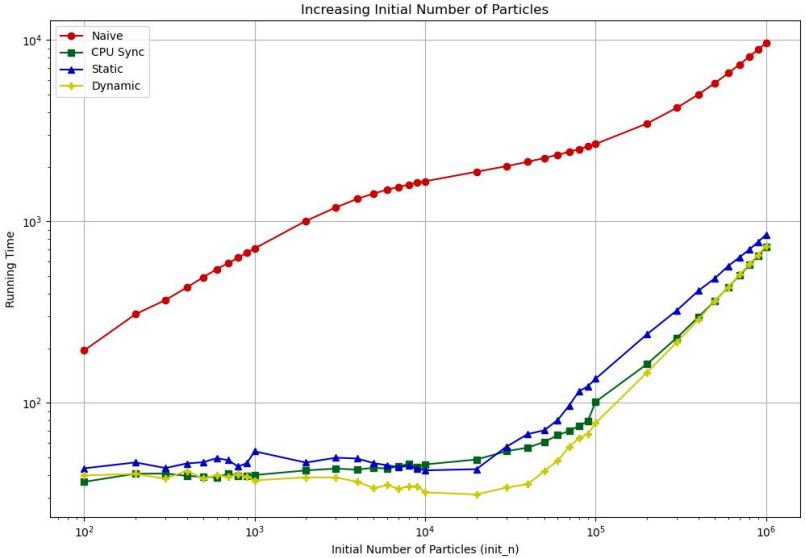


Same as before, but memory optimisation skewed the optimal block size

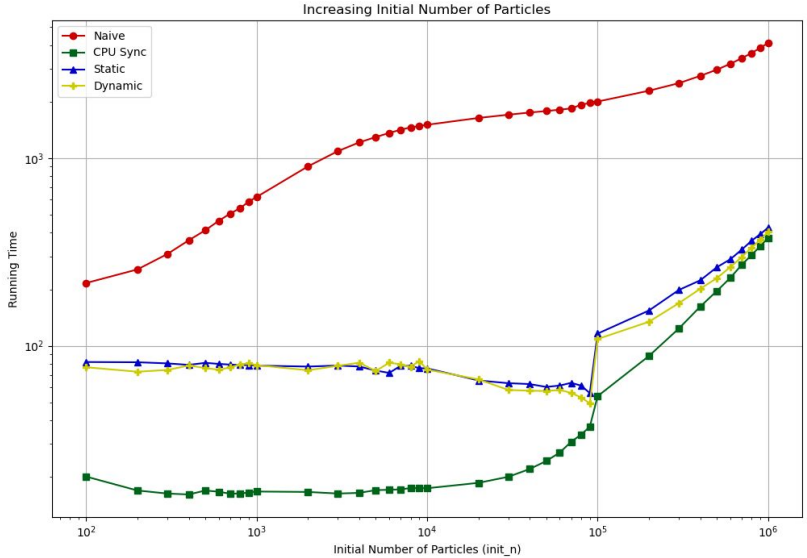
Small block size results in less bottleneck from slow threads

MVP: Initial Number of Particles

Simulating newly spawned



Without simulating newly spawned

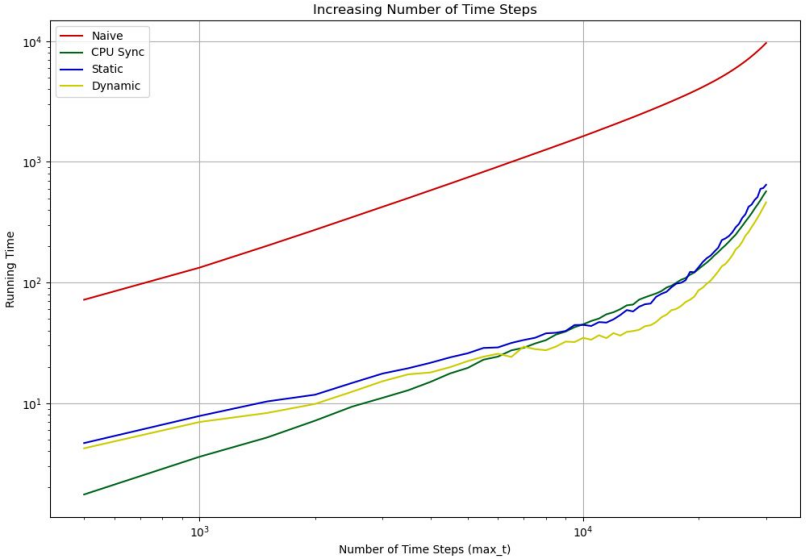


for init_n > 10.000

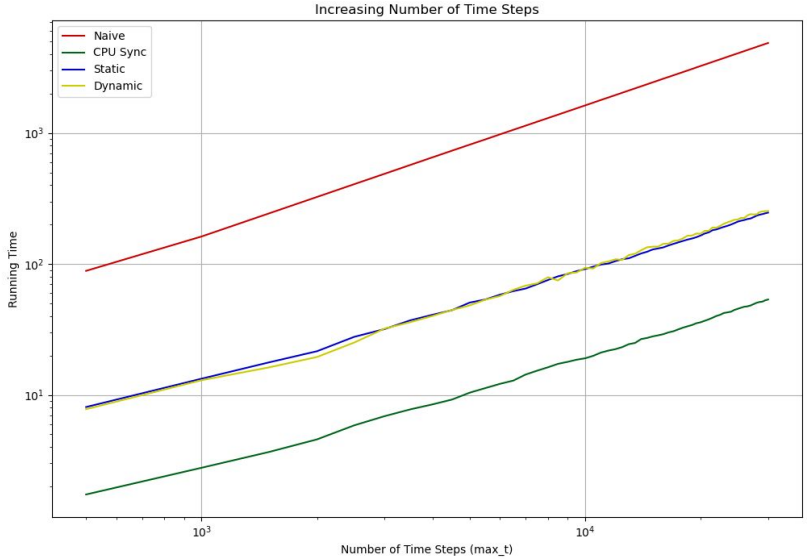
init_n · 0,05 % > 0,5

MVP: Number of Time Steps

Simulating newly spawned



Without simulating newly spawned



Exponential growth is caused by increased no. particles

Scheduler For Further Work

Dynamic:

- For the most part performed the best
  - Close behind when not
- Room for optimisations

Static:

- Has many of the same same quality
- But overall worse

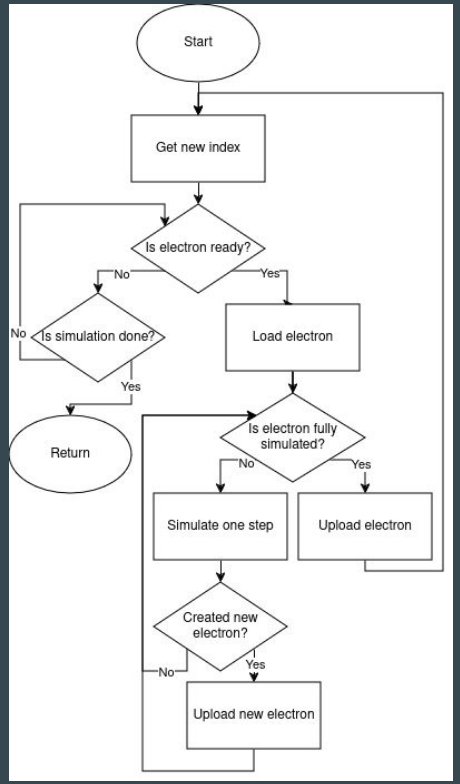
Particle-In-Cell (PIC)

- The final simulation
- Close to physically correct
  - Electrons divided into cells on a grid
  - Poisson steps, each containing many mobility steps
  - Chance for removal of particles
  - Cross sections for collision chances depending on velocity
- Optimises Dynamic

Dynamic Optimisation: Implementation

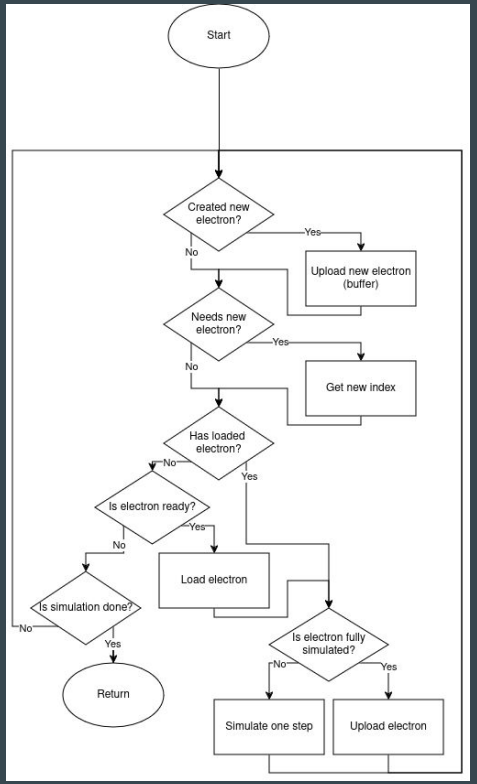
Old:

- Nested loops
- Simple



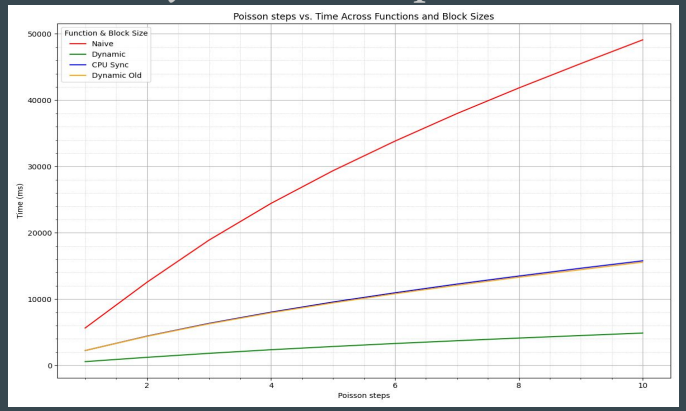
New:

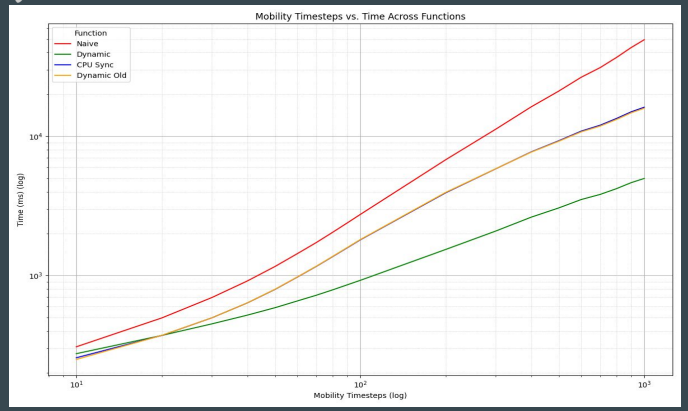
- Only one loop
- More steps
- Allows electron buffer upload
- Warp cooperation

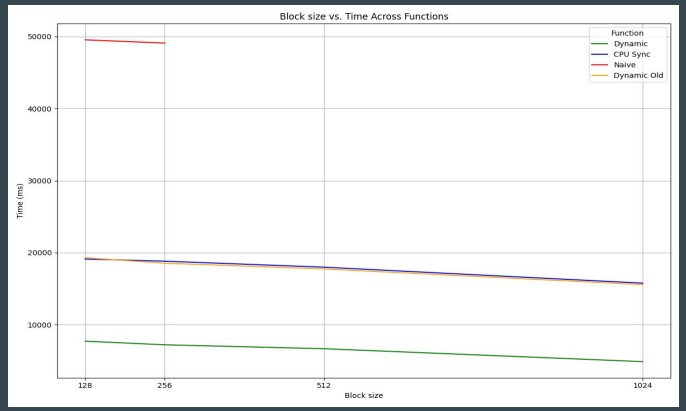


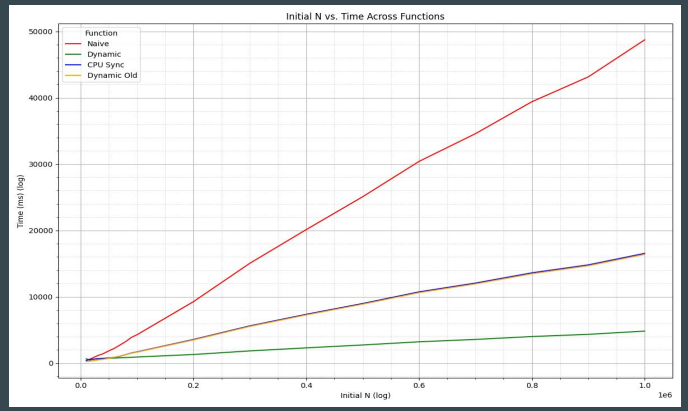
Dynamic Optimisation: Performance

New Dynamic is superior in nearly all cases.









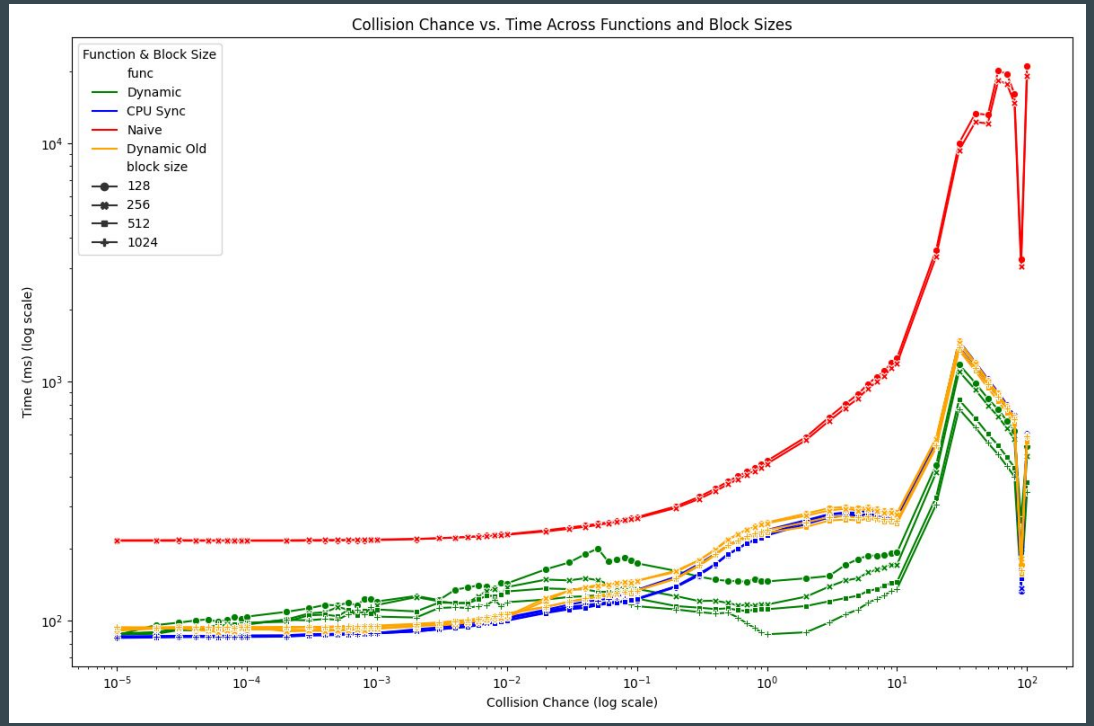
Except...

Dynamic Optimised: Performance for Constant CC

When collision chance is constant:

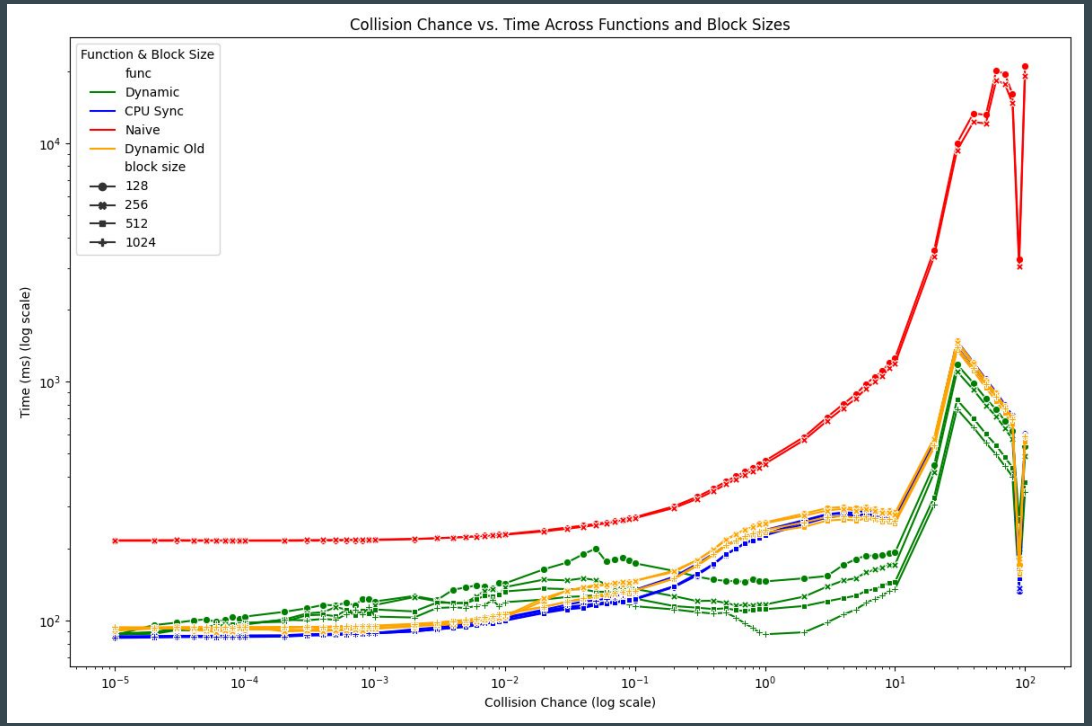
- If collisions are rare, the additions are not worth it.
- If collisions are very common, the additions make little difference.

But something here is weird.



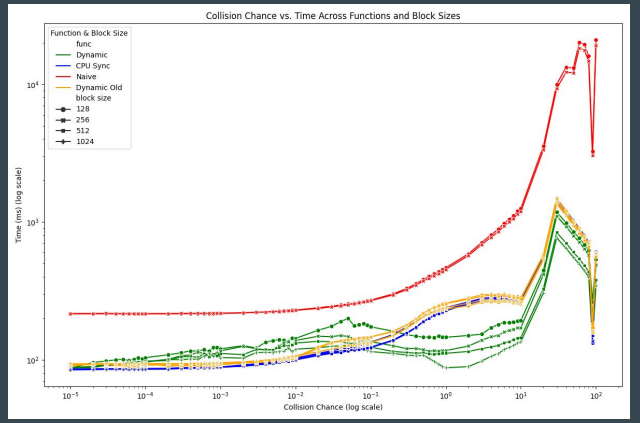
Random Numbers: Weird Behaviour

- Decline after ~30%
- Dip at 90%

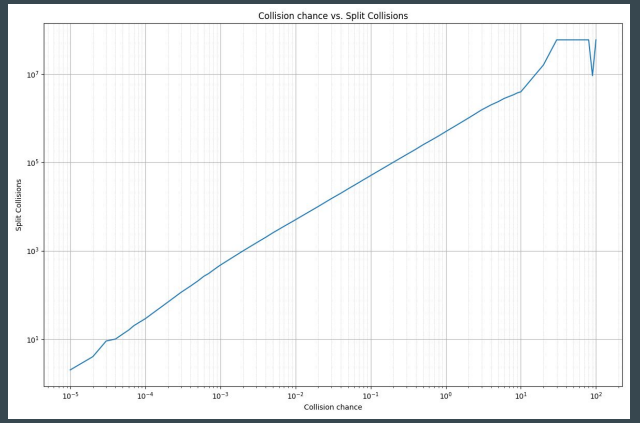


Random Numbers: Non-Linear Collision Counts

- The same tendency can be seen in collision counts.
- Caused by lack of uniformity in random numbers.



- Too fast increase after 10% (hits capacity).
  - This explains steep rise in processing time.
- Dip at 90%.



Random Numbers: Use Sequence Parameter

Submitted code:

- Varying seed
- Constant sequence

```
__device__ void newRandState(curandState* rand_state, int seed){
    curand_init(39587 + seed, 0, 0, rand_state);
}
```

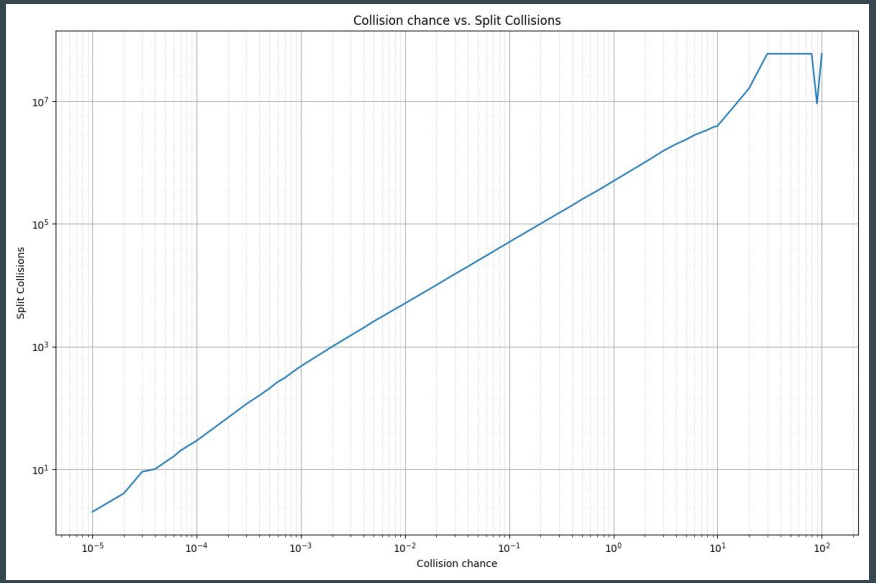
Fixed code:

- Constant seed
- Varying sequence

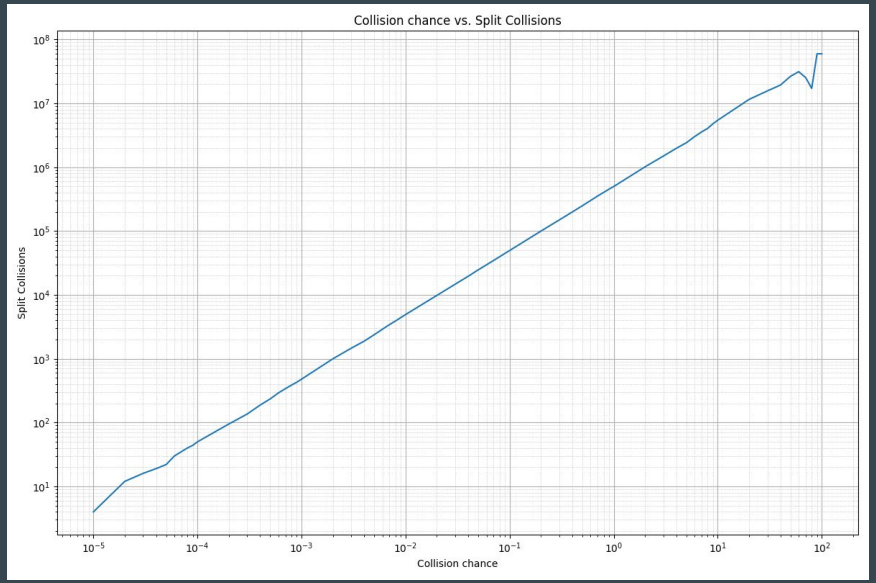
```
__device__ void newRandState(curandState* rand_state, int sequence){
    curand_init(39587, sequence, 0, rand_state);
}
```

Random Numbers: Sequence Fix Results

Before



After



Random Numbers: No Determinism

Submitted code:

- Each particle has a state
- New states generated with new particles

```
d_electrons[new_i] = new_electron;
d_rand_states[new_i] = new_rand_state;
```

```
newRandState(new_rand_state, randInt(rand_state, 0, 2000000000)+100000000);
```

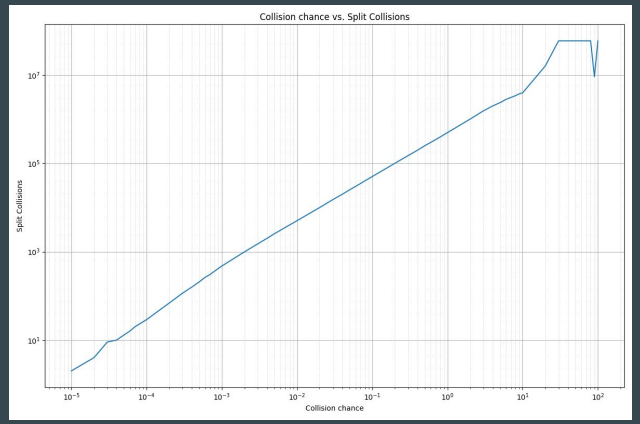
Fixed code:

- States initialised at start
- No control of which electron uses which state

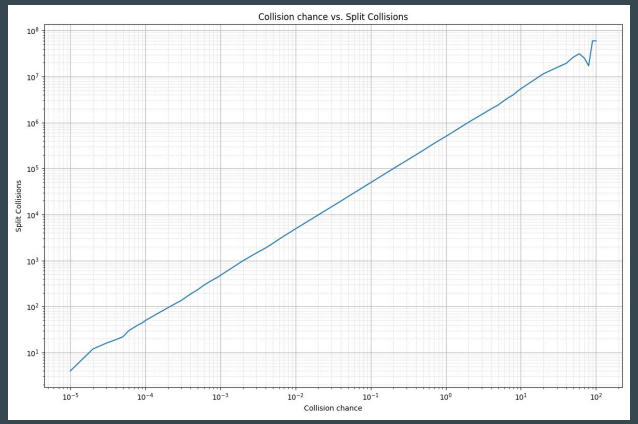
```
__global__ void setup_rand(curandState* d_rand_states) {
    int i = threadIdx.x+blockDim.x*blockIdx.x;
    newRandState(&d_rand_states[i], i);
}
```

Random Numbers: No Determinism Results (Collisions)

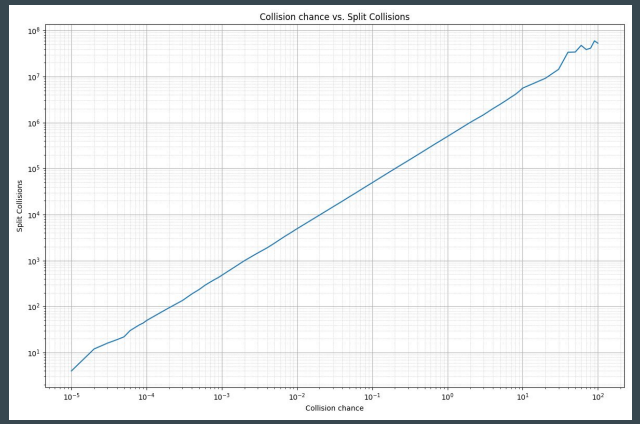
Before



Varying sequence



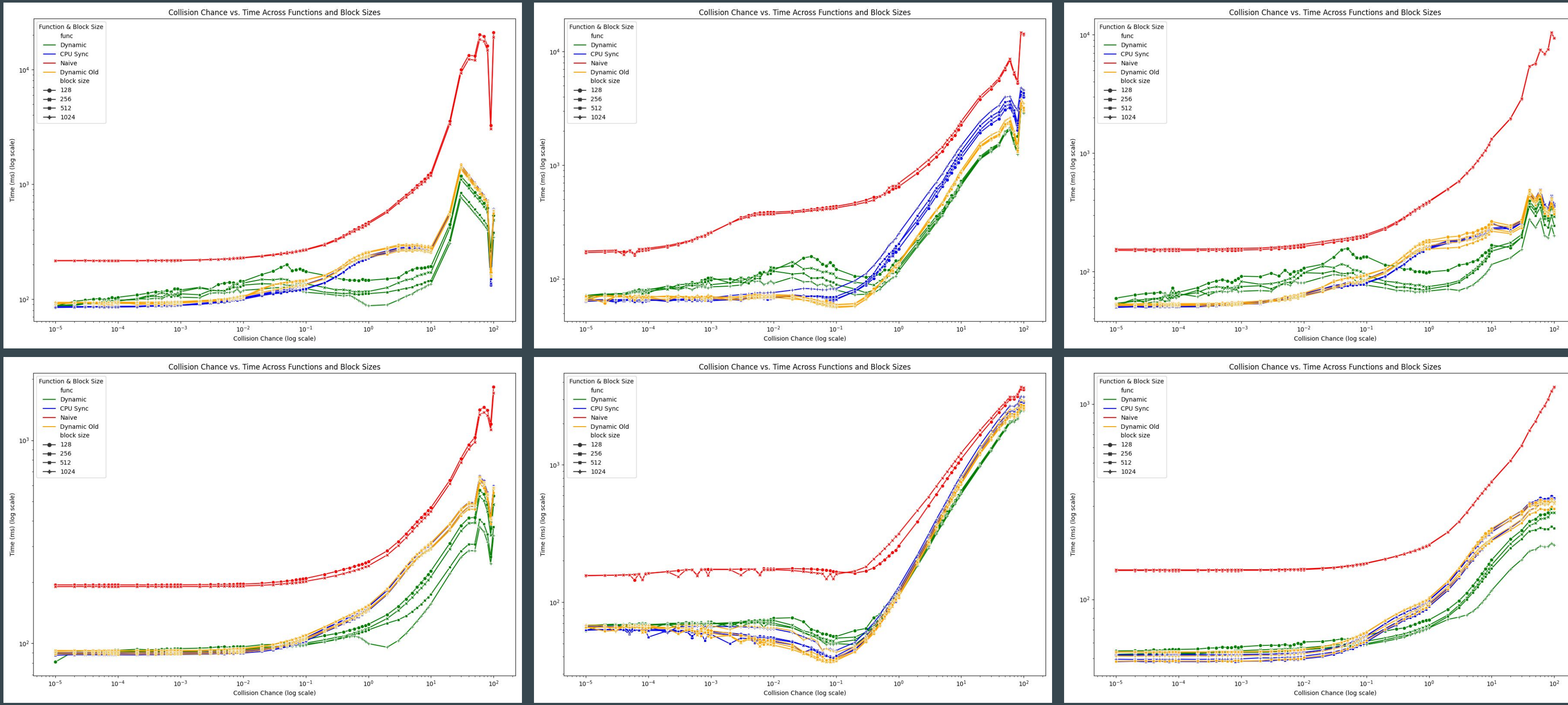
No determinism



Random Numbers: Time Results

L:

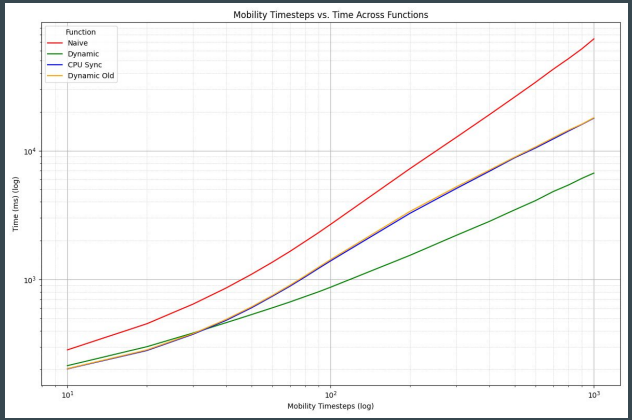
S:



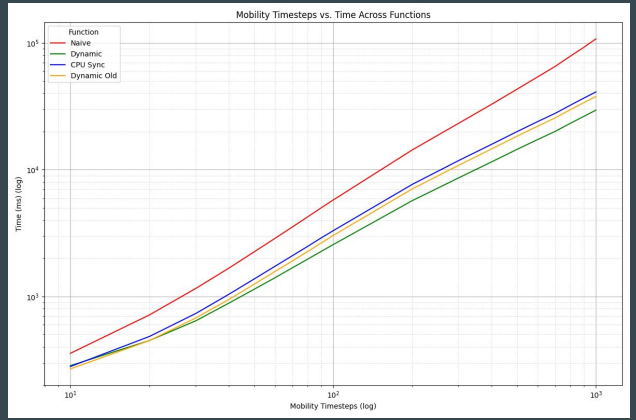
Before | Varying sequence | No determinism

Random Numbers: Consequences for Mobility Steps

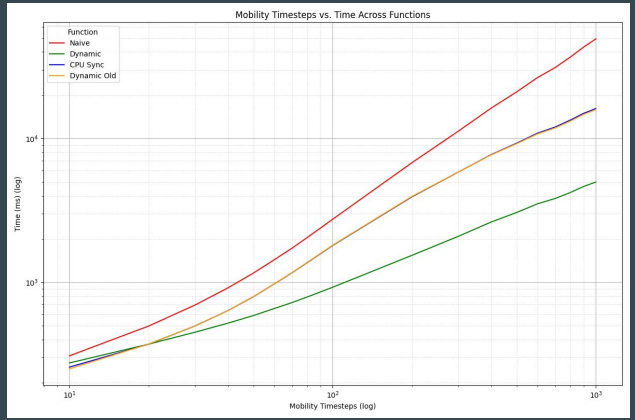
Before



Varying sequence



No determinism



Conclusion

- We have come to understand various scheduling paradigms.
  - How they influence performance.
  - Why they work as they do.
  - How the details of the simulation affects the performance of various schedulers.
  - Persistent dynamic schedulers have a lot of potential and are often superior.
- We have created a near-realistic simulation and optimised a scheduler for it.
- Random number generation matters.
  - Can affect outcome.
  - Can affect benchmarks.
- Beyond schedulers, impact of changes can be hard to predict.
  - Always test different block sizes.
  - Micro-benchmarks are helpful for small changes.