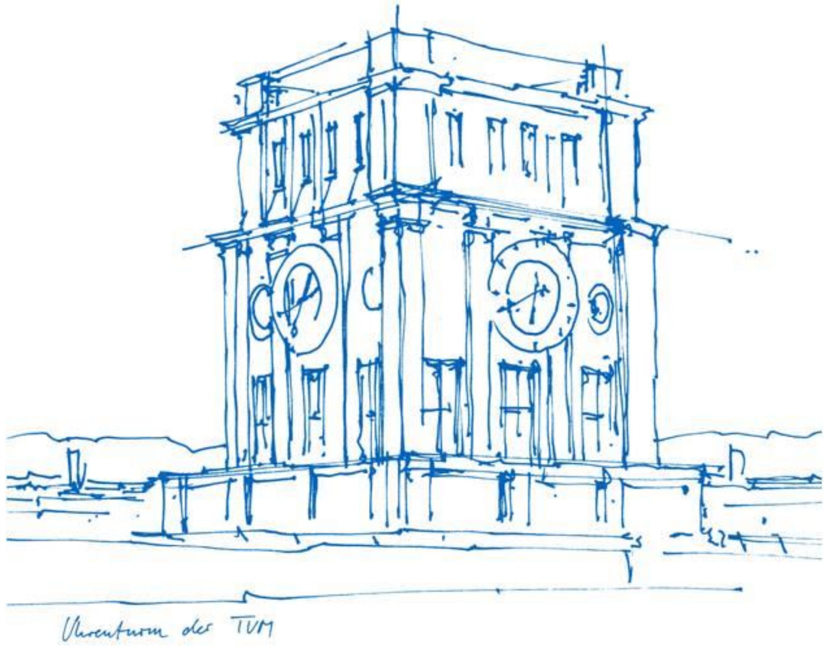
PSE Molecular Dynamics: Worksheet 5

Group C, 31.01.2025

Luca-Dumitru Drîndea
Mara Godeanu
Flavius Schmidt



3D Simulations

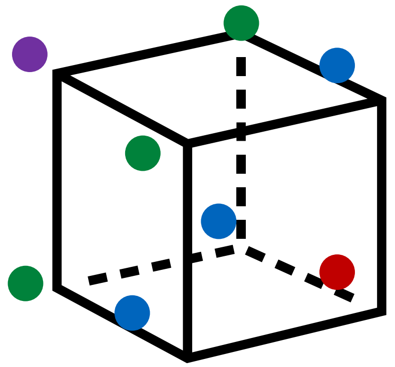
Number of dimensions: 2D or 3D must be given as a parameter now

Many minor tweaks: neighbour cell calculations, reflective borders …

Major tweaks:

1. **Ghost Particles**: power set of Border Locations – works fine in 2D too!

2. **Determining Border Condition**
   - see next slide!



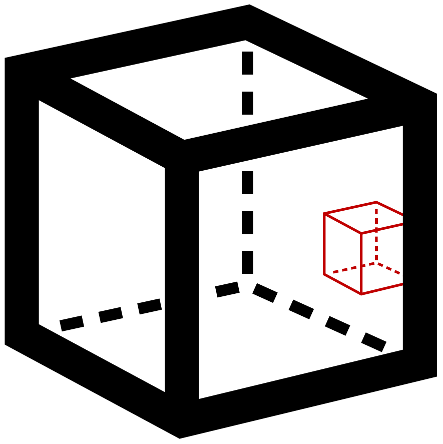
<span style="color:red">Red Particle</span> is mirrored in 3D periodic domain

Border Conditions…

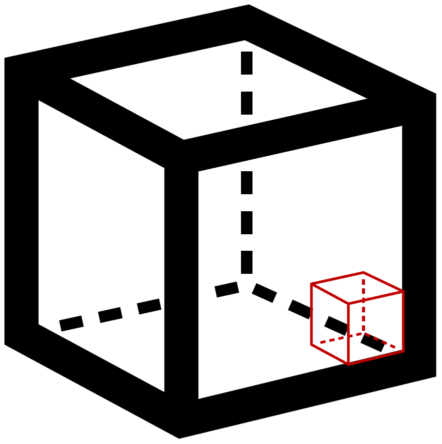
Not a corner: Easy! $\longrightarrow$ *choose the only option*

Corner: Pain :($\longrightarrow$ *20 cases, two systems: double / triple corner*

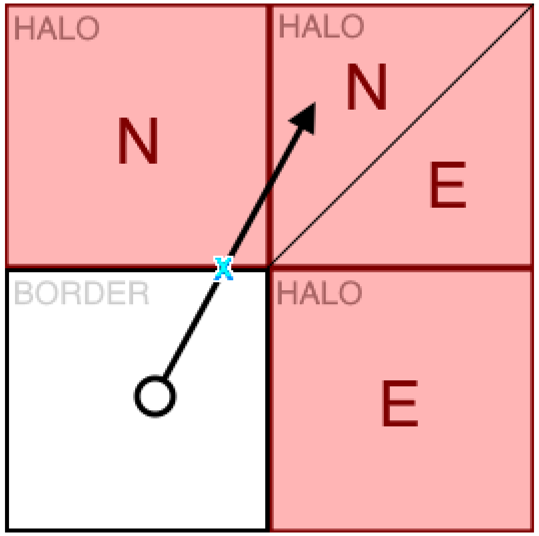
Double Corner

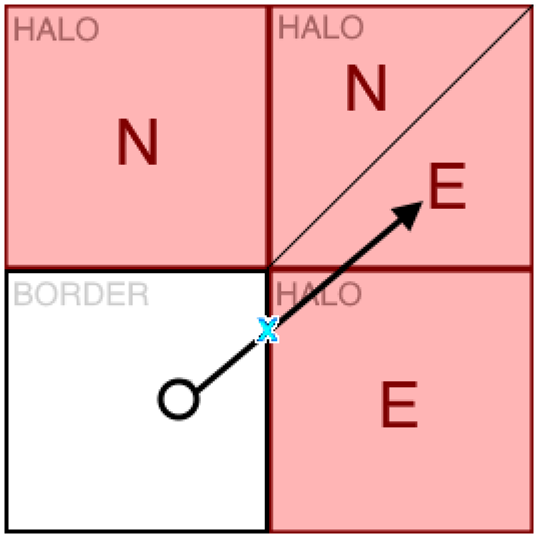


Triple Corner



First a Recap from Sheet 3



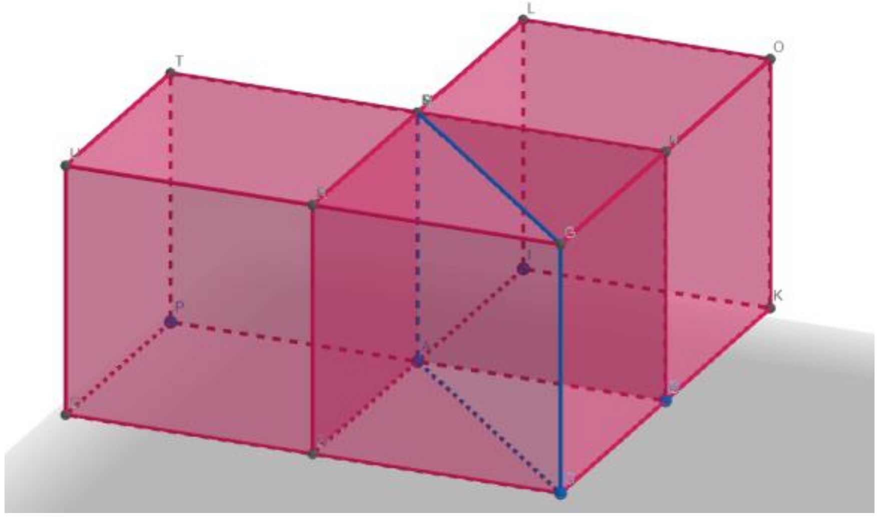


Divide the corner cell in **two**,

then check if the particle is **above or below** the line

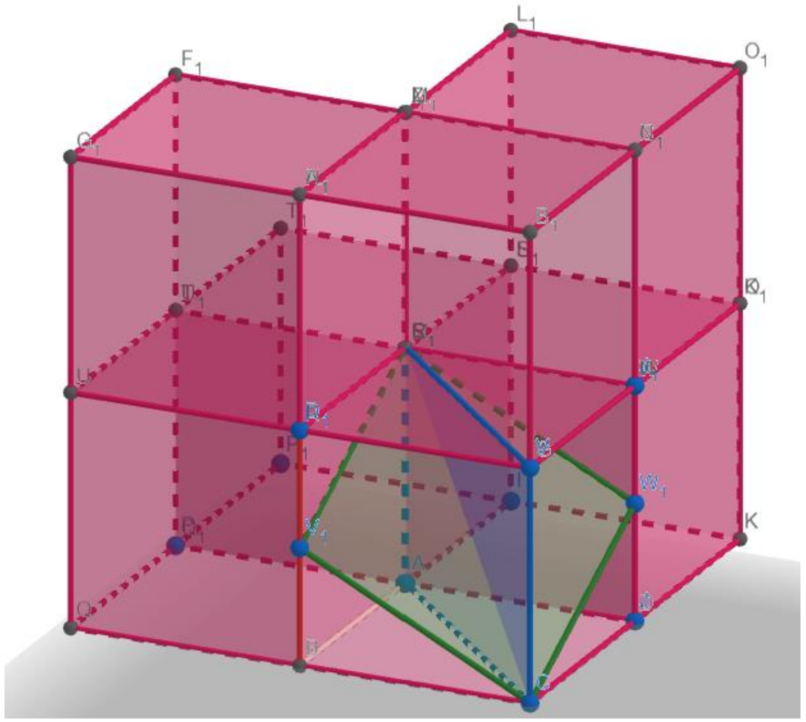
Corners *(12 double cases, 8 triple)*

Double



simple extension from 2D into 3D
(from line to plane)

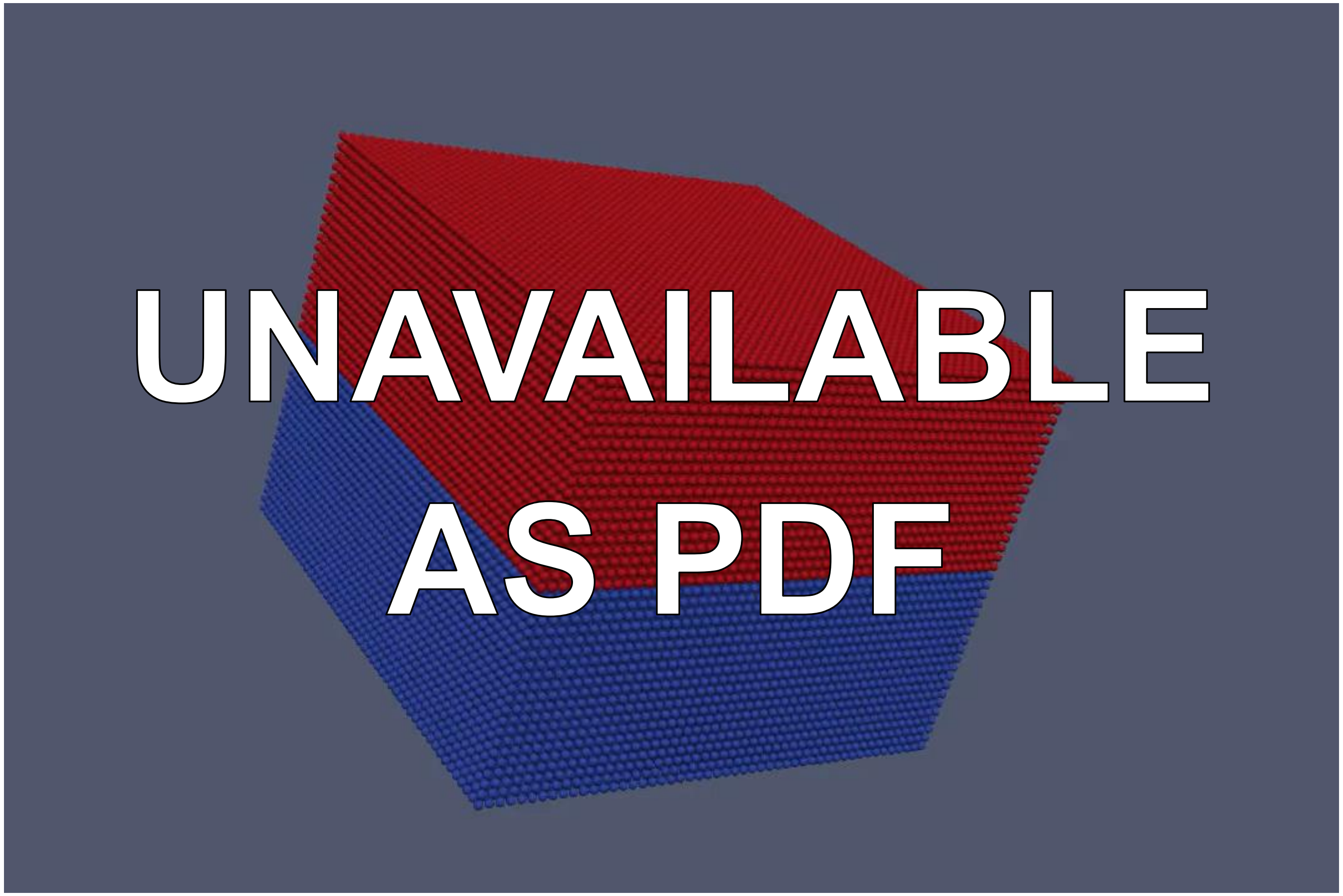
Triple



Two-tiered decision system:
first green for vertical, then blue for left/right

(images generated with GeoGebra)

3D Rayleigh-Taylor Instability (Video)



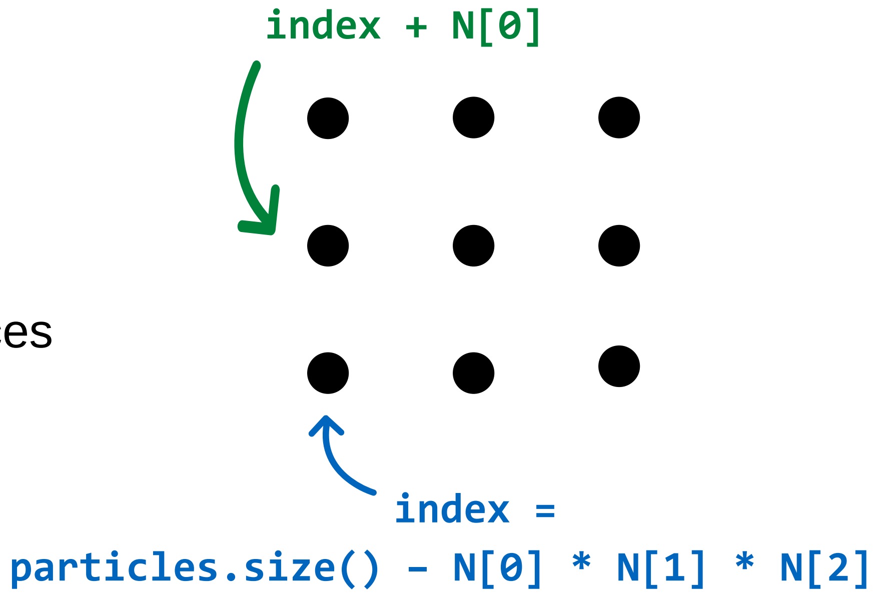
Membrane

New particle attributes: membrane stiffness, average bond length etc.

Particles have neighbour vectors:
based on index in ParticleContainer

Define special particles: they get extra forces

New forces function for membranes:
based on harmonic potential

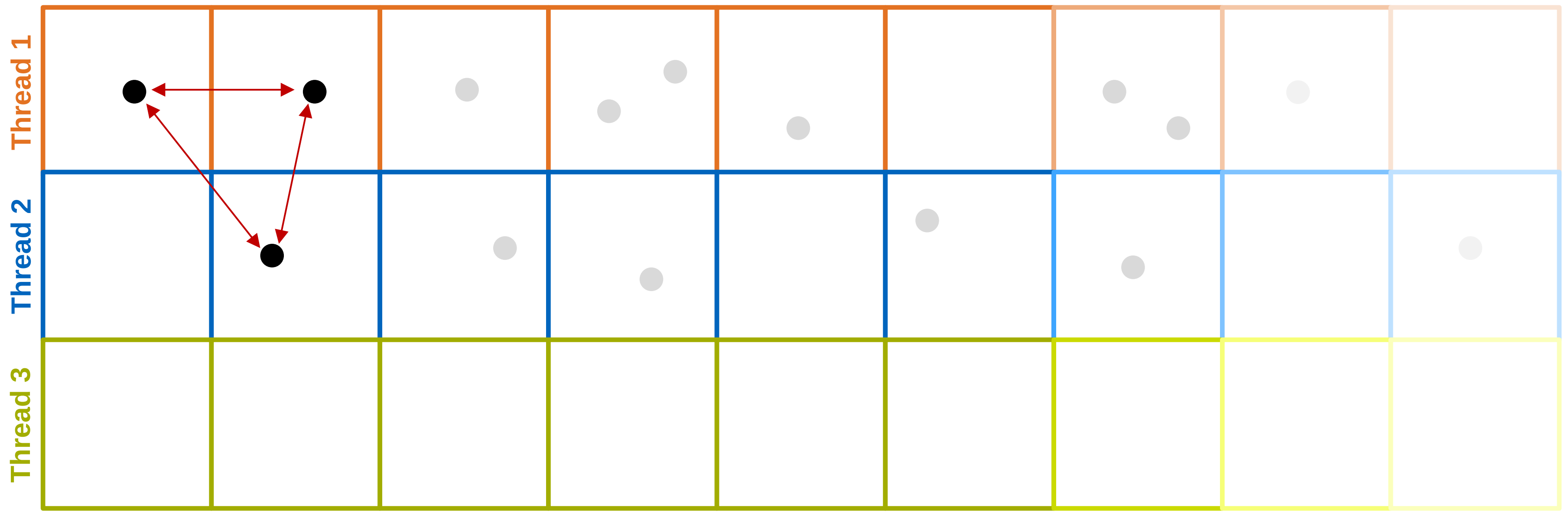


Maxwell-Boltzmann velocity distribution disabled!

Membrane (Video)

Parallelization (force calculation)

Method 1: Standard, Coarse-Grained (`#pragma omp parallel for`)



✓ **Simple and configurable**

`(static, dynamic…)`

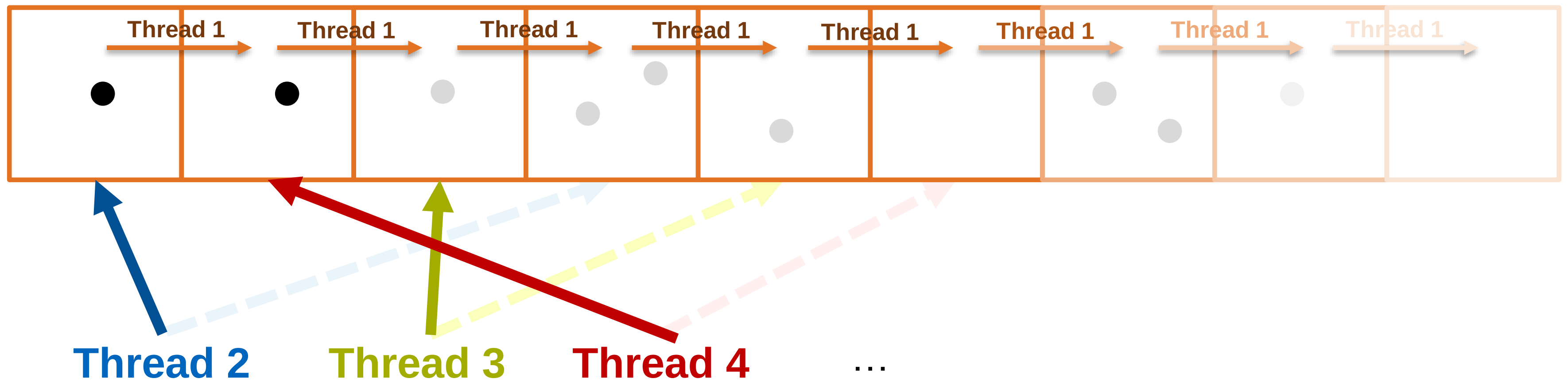
Less overhead

Better for homogenous systems

✗ **Potential for load imbalance**

Parallelization (force calculation)

Method 2: Task-Based, Fine-Grained (`#pragma omp task`)



✓

Better Load Balancing
(esp. inhomogeneous sys.)

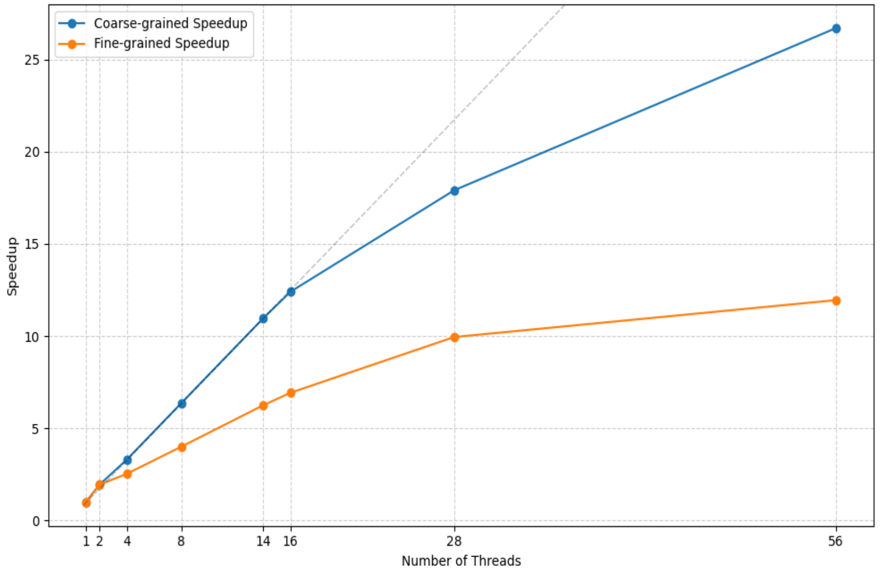
✗

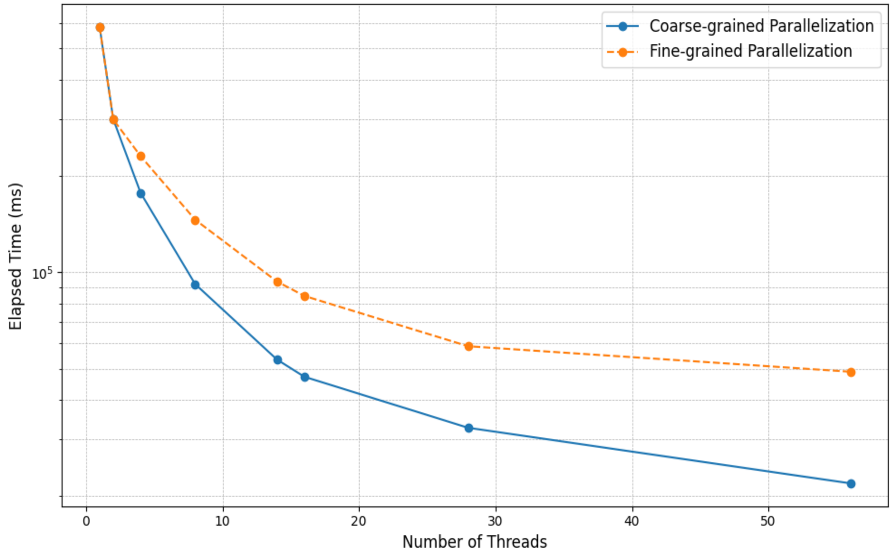
Worse for homogeneous systems
(more overhead)

Parallelization (speedup)

Results on CoolMUC-4 (`cm4tiny`), Task 3 Input, 1000 iterations…

g++ 13.2.0





Speedup for 1 – 56 Threads

28 Threads: 17x

56 Threads: 26x

Time for 1 – 56 Threads

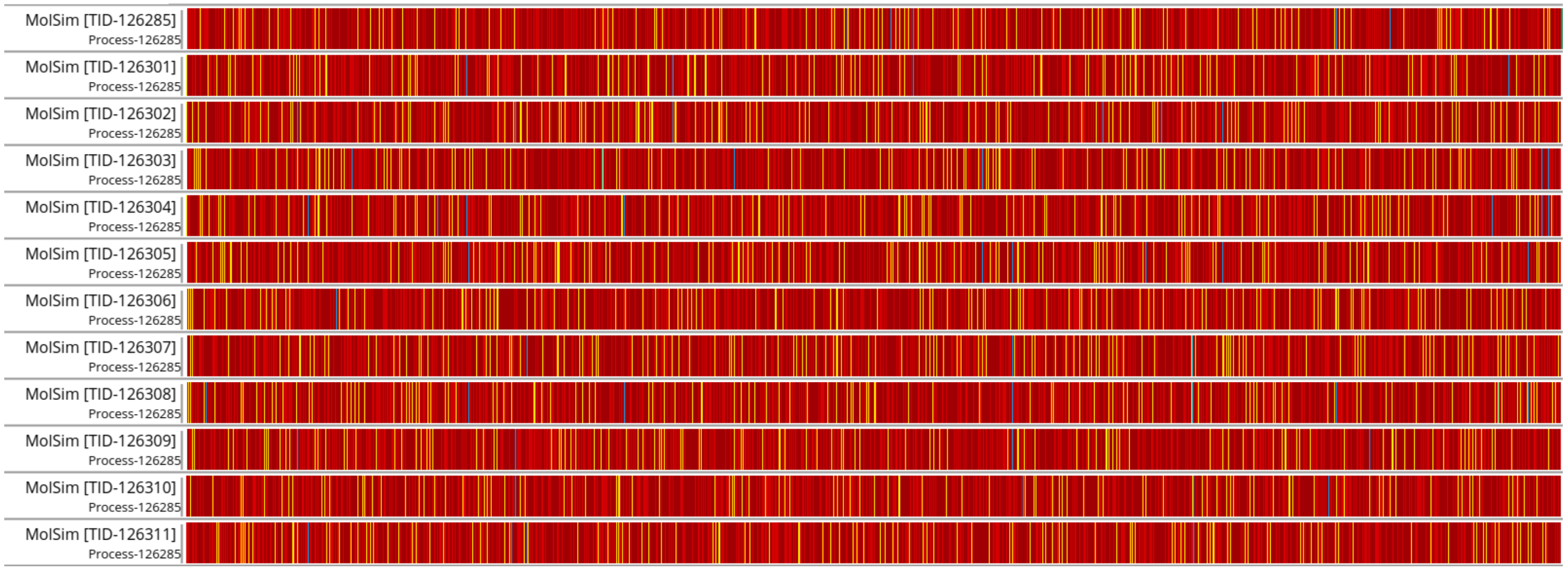
1 Thread: ~580s

56 Threads: ~22s

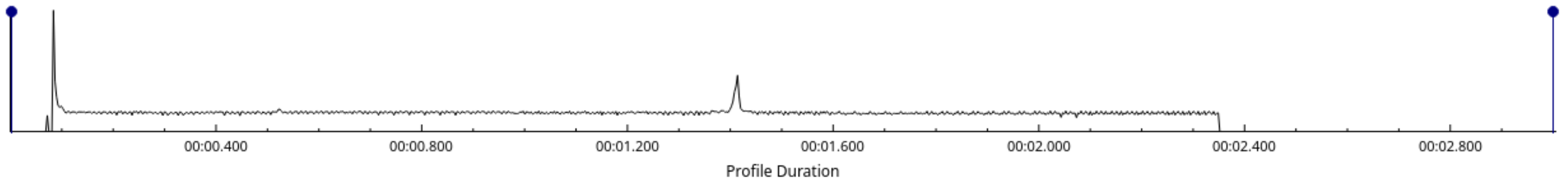
Parallelization (profiling, AMD uProf)

GOOD: Steady Concurrency

red: force, **yellow:** position, **blue:** velocity



MolSim [TID-126285] Process-126285
MolSim [TID-126301] Process-126285
MolSim [TID-126302] Process-126285
MolSim [TID-126303] Process-126285
MolSim [TID-126304] Process-126285
MolSim [TID-126305] Process-126285
MolSim [TID-126306] Process-126285
MolSim [TID-126307] Process-126285
MolSim [TID-126308] Process-126285
MolSim [TID-126309] Process-126285
MolSim [TID-126310] Process-126285
MolSim [TID-126311] Process-126285

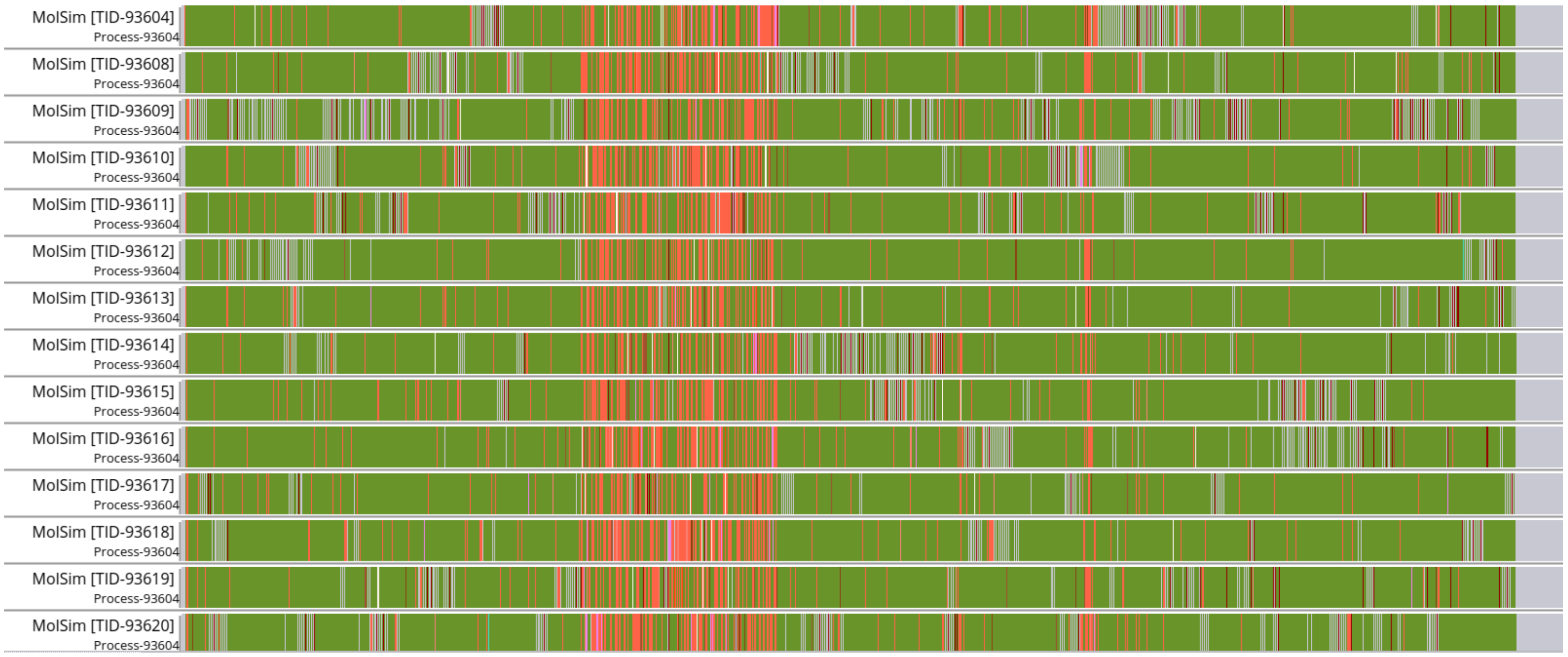


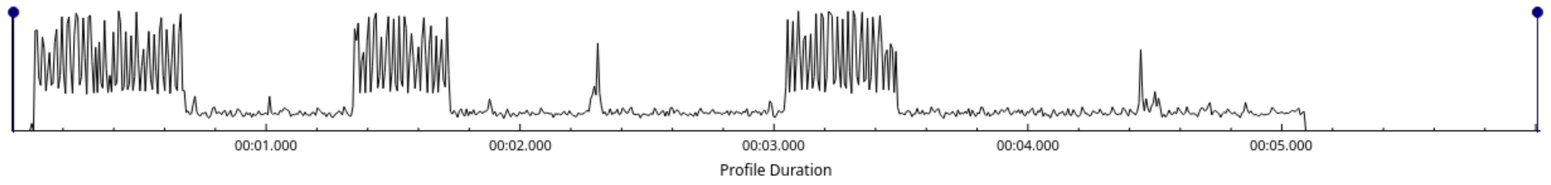
CPI (Cycles Per Instruction)

Parallelization (profiling, AMD uProf)

BAD: CPI Spikes (busy waiting?) → 50% - 150% slower!

green: `libgomp`, **NOT** our code



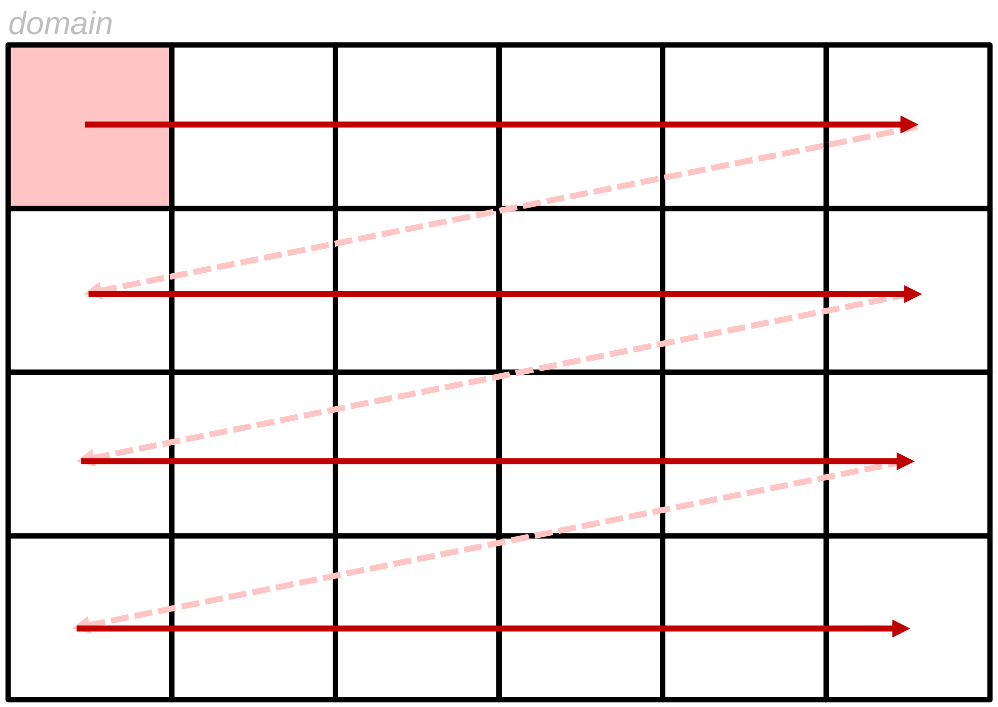


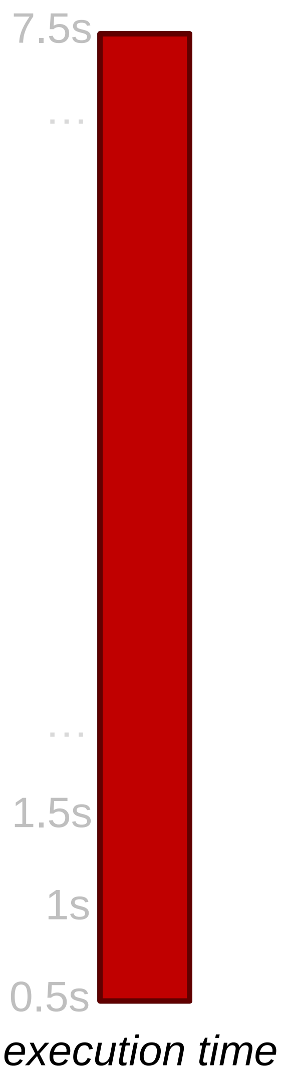
CPI (Cycles Per Instruction)

Optimizations

Results on ZenBook 13, Ubuntu 24.04, Linux 6.8, 16GB RAM, AMD Ryzen 7 5800U (8 cores, 16 threads) @ 4.51 GHz
Contest 1 Input (2D), 1000 iterations…
g++ 13.3.0

- **Base (serial):** ~7.5s

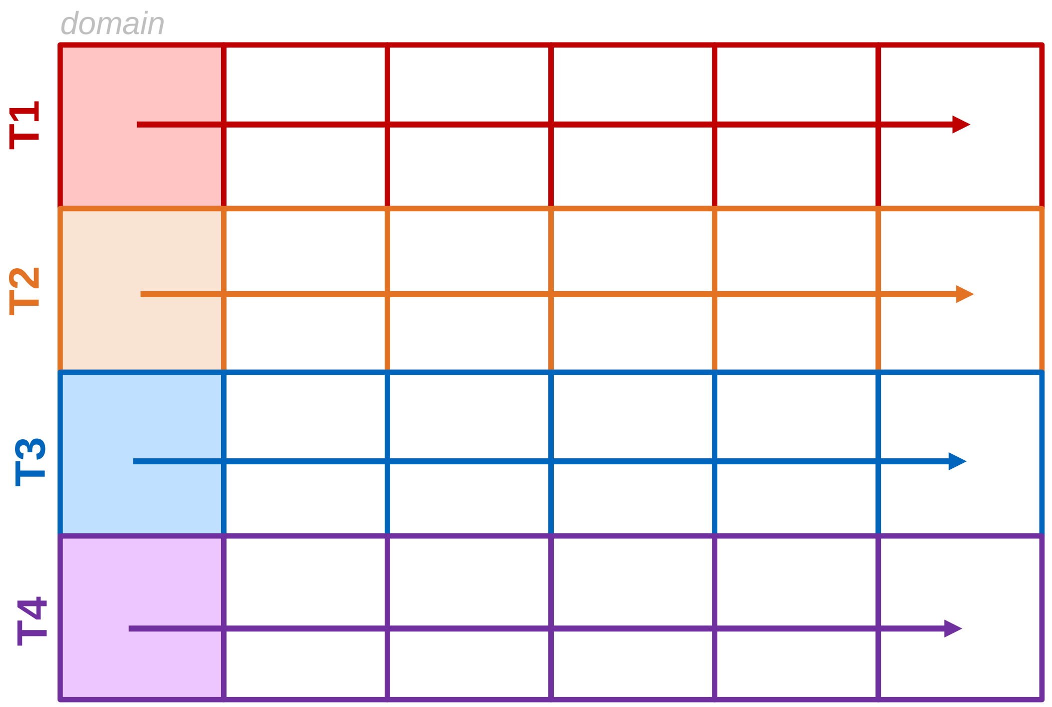


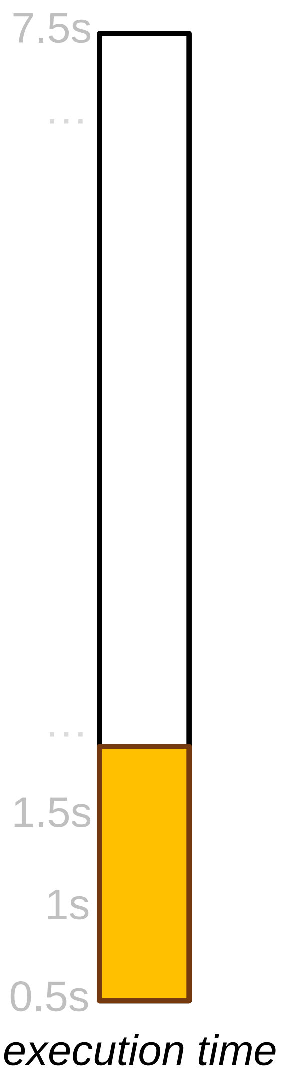


Optimizations

Results on ZenBook 13, Ubuntu 24.04, Linux 6.8, 16GB RAM, AMD Ryzen 7 5800U (8 cores, 16 threads) @ 4.51 GHz
Contest 1 Input (2D), 1000 iterations…
g++ 13.3.0

- **Base (serial):** ~7.5s
- **Base (parallel, 16 threads):** ~1.8s





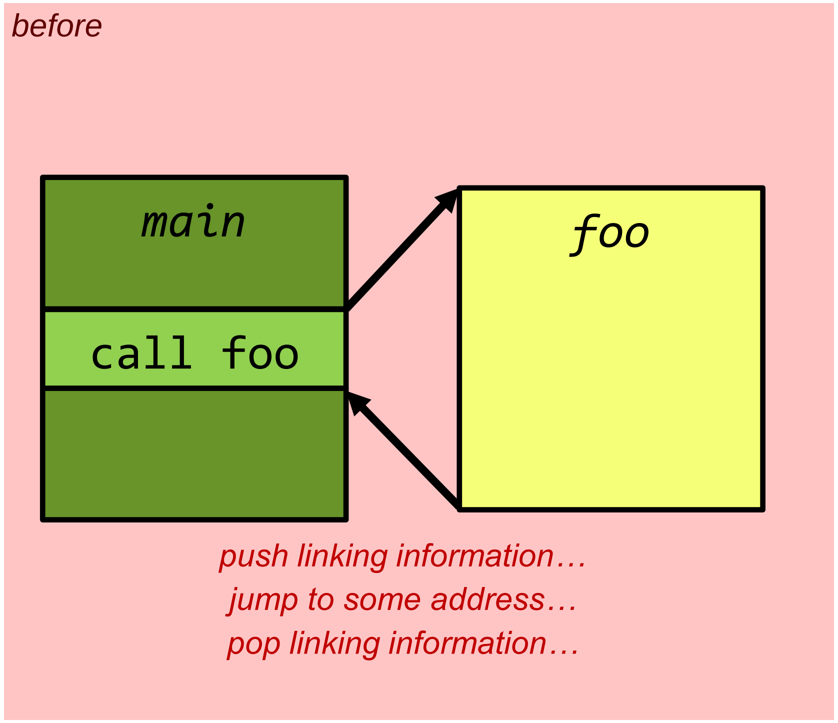
Optimizations

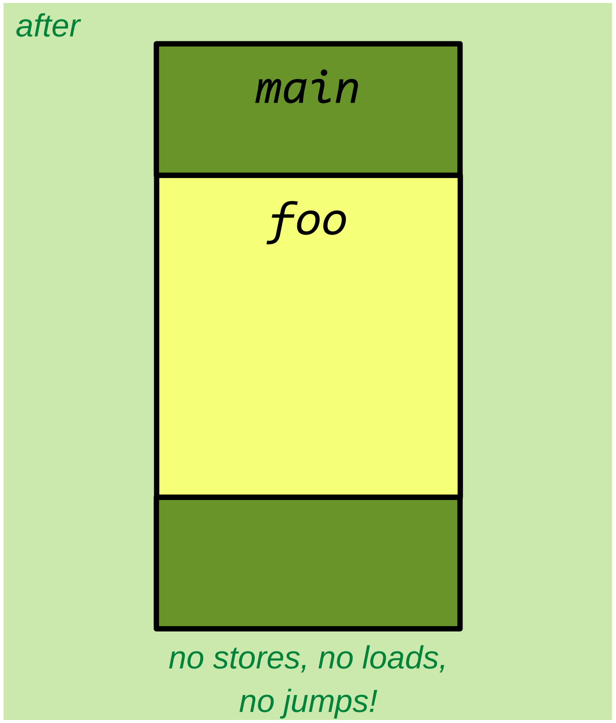
Results on ZenBook 13, Ubuntu 24.04, Linux 6.8, 16GB RAM, AMD Ryzen 7 5800U (8 cores, 16 threads) @ 4.51 GHz

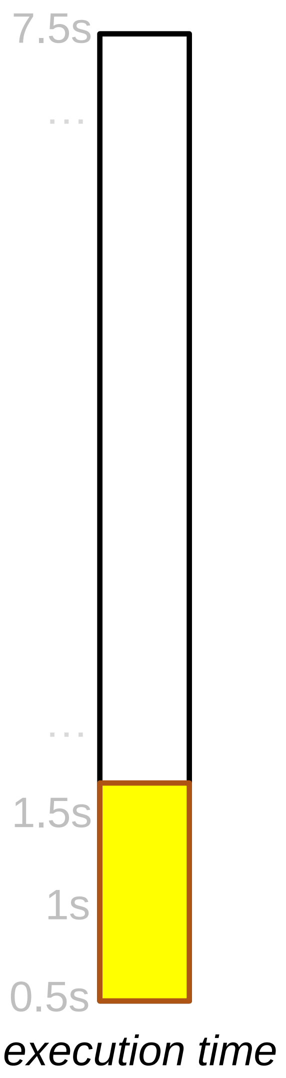
Contest 1 Input (2D), 1000 iterations…

g++ 13.3.0

- **Base (serial):** ~7.5s
- **Base (parallel, 16 threads):** ~1.8s
- **Function Inlining**: ~1.6s







Optimizations

Results on ZenBook 13, Ubuntu 24.04, Linux 6.8, 16GB RAM, AMD Ryzen 7 5800U (8 cores, 16 threads) @ 4.51 GHz
Contest 1 Input (2D), 1000 iterations…
g++ 13.3.0

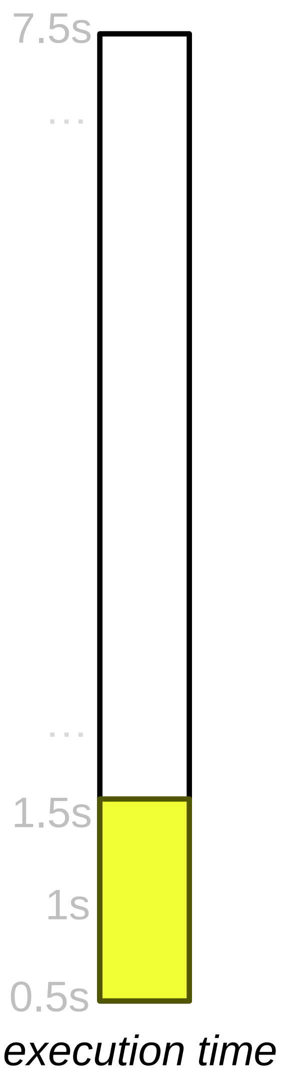
- **Base (serial):** ~7.5s
- **Base (parallel, 16 threads):** ~1.8s
- **Function Inlining**: ~1.6s
- **Periodic Boundary Checks**: ~1,5s

before

North	South	West	East	Above	**Below**
outflow,	outflow,	outflow,	outflow,	outflow,	**periodic**
↑	↑	↑	↑	↑	↑
no	nope	still no	nyet	nuh uh	**yes!**

after

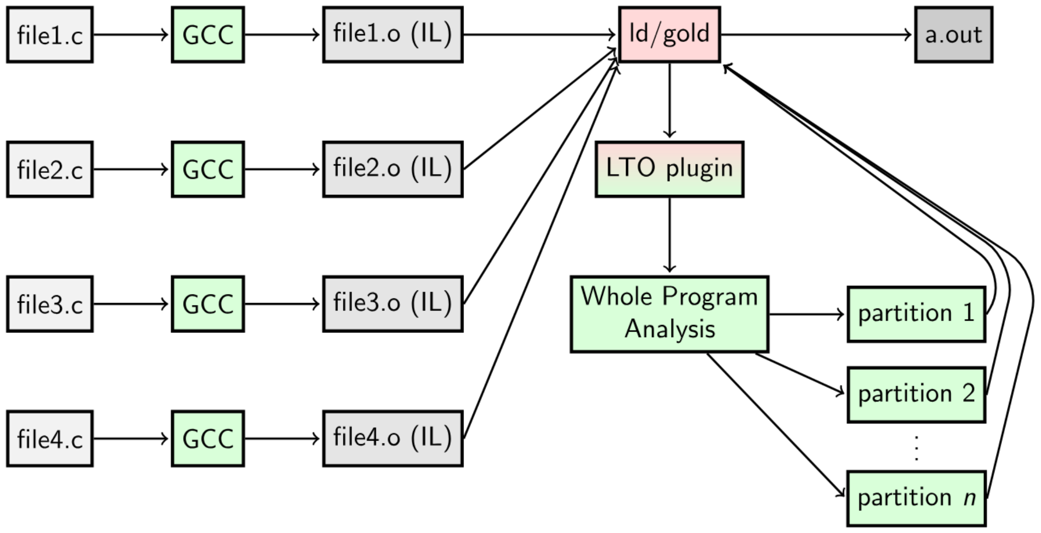
```
bool anyPeriodic = true
```

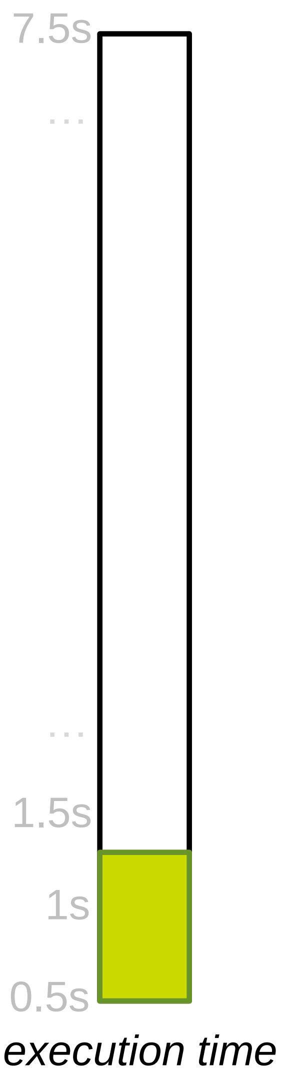


Optimizations

Results on ZenBook 13, Ubuntu 24.04, Linux 6.8, 16GB RAM, AMD Ryzen 7 5800U (8 cores, 16 threads) @ 4.51 GHz
Contest 1 Input (2D), 1000 iterations…
g++ 13.3.0

- **Base (serial):** ~7.5s
- **Base (parallel, 16 threads):** ~1.8s
- **Function Inlining**: ~1.6s
- **Periodic Boundary Checks**: ~1,5s
- **Link Time Optimization**: ~1.3s





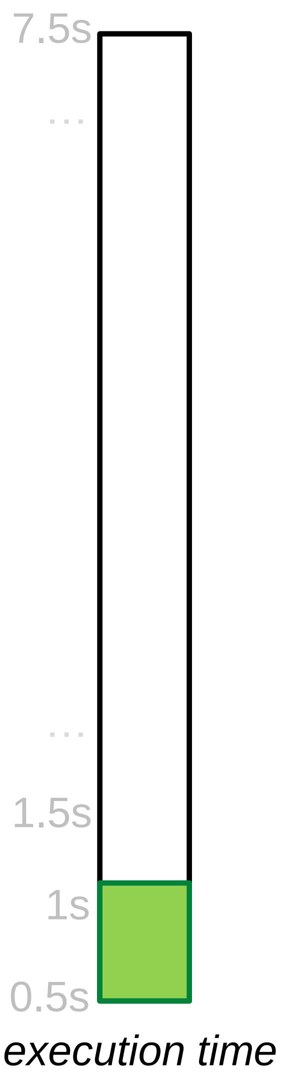
Optimizations

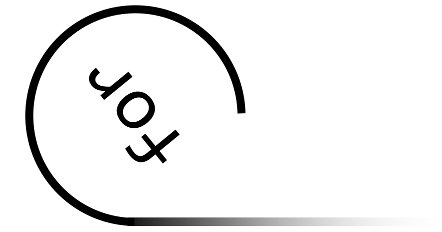
Results on ZenBook 13, Ubuntu 24.04, Linux 6.8, 16GB RAM, AMD Ryzen 7 5800U (8 cores, 16 threads) @ 4.51 GHz

Contest 1 Input (2D), 1000 iterations…

g++ 13.3.0

- **Base (serial):** ~7.5s
- **Base (parallel, 16 threads):** ~1.8s
- **Function Inlining**: ~1.6s
- **Periodic Boundary Checks**: ~1,5s
- **Link Time Optimization**: ~1.3s
- **Compiler Flags**: ~1.1s





```
-march=native
-mtune=native
```

```
-fdata-sections
-ffunction-sections
```

```
-funroll-loops
```

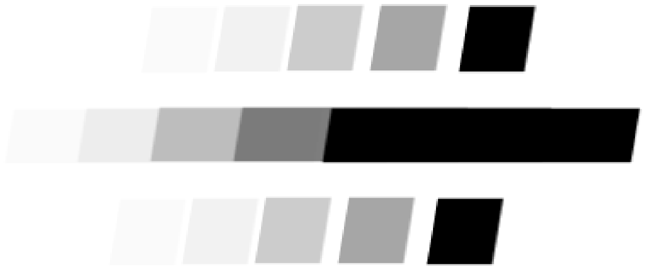
Optimizations

Results on ZenBook 13, Ubuntu 24.04, Linux 6.8, 16GB RAM, AMD Ryzen 7 5800U (8 cores, 16 threads) @ 4.51 GHz

Contest 1 Input (2D), 1000 iterations…

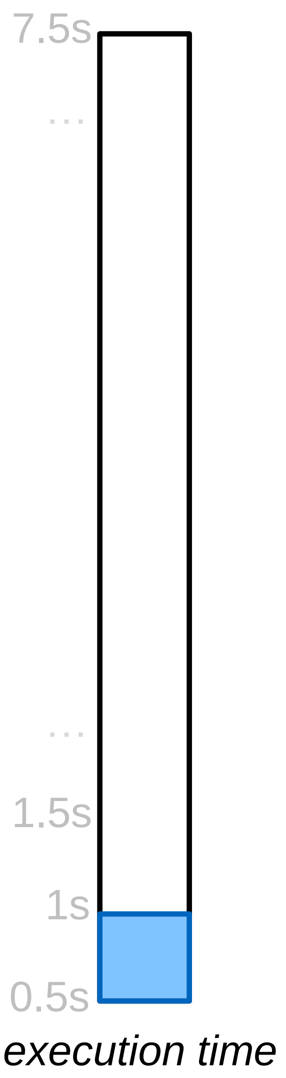
g++ 13.3.0

- **Base (serial):** ~7.5s
- **Base (parallel, 16 threads):** ~1.8s
- **Function Inlining**: ~1.6s
- **Periodic Boundary Checks**: ~1,5s
- **Link Time Optimization**: ~1.3s
- **Compiler Flags**: ~1.1s
- **Fast Math**: ~0.9s



`-ffast-math`

No INFs, no NANs, no problem…?



Optimizations

Results on ZenBook 13, Ubuntu 24.04, Linux 6.8, 16GB RAM, AMD Ryzen 7 5800U (8 cores, 16 threads) @ 4.51 GHz

Contest 1 Input (2D), 1000 iterations…

g++ 13.3.0

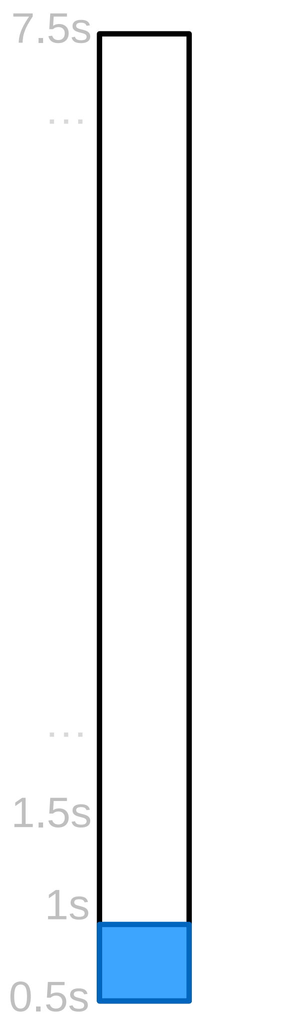
- **Base (serial):** ~7.5s
- **Base (parallel, 16 threads):** ~1.8s
- **Function Inlining**: ~1.6s
- **Periodic Boundary Checks**: ~1,5s
- **Link Time Optimization**: ~1.3s
- **Compiler Flags**: ~1.1s
- **Fast Math**: ~0.9s
- **Skip Outflow Checks**: ~0.85s

before

```
if(!p.isActive()) continue;
```

after

```
(void)0; NOP                    -DNO_OUTFLOW=ON
```

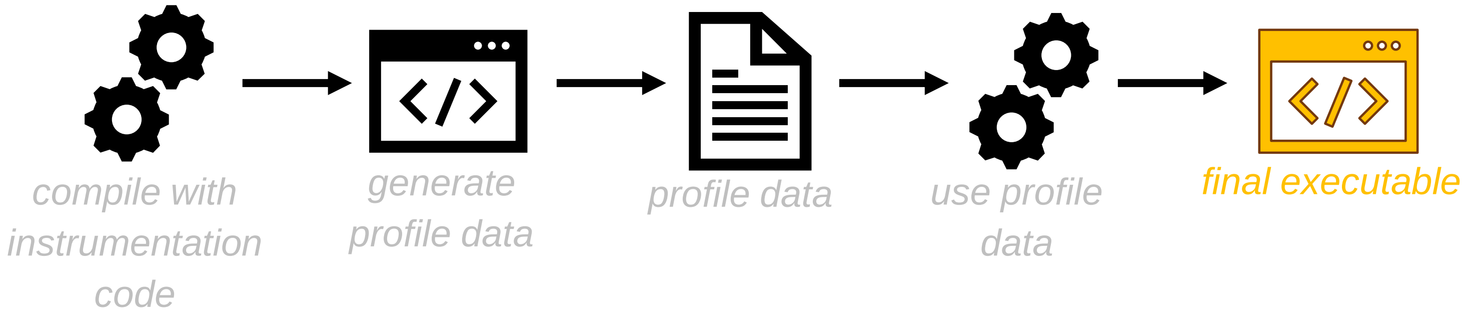


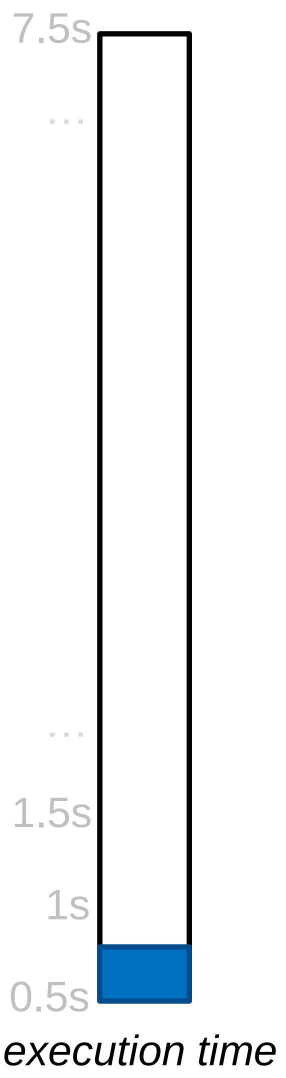
execution time

Optimizations

Results on ZenBook 13, Ubuntu 24.04, Linux 6.8, 16GB RAM, AMD Ryzen 7 5800U (8 cores, 16 threads) @ 4.51 GHz
Contest 1 Input (2D), 1000 iterations…
g++ 13.3.0

- **Base (serial):** ~7.5s
- **Base (parallel, 16 threads):** ~1.8s
- **Function Inlining**: ~1.6s
- **Periodic Boundary Checks**: ~1,5s
- **Link Time Optimization**: ~1.3s
- **Compiler Flags**: ~1.1s
- **Fast Math**: ~0.9s
- **Skip Outflow Checks**: ~0.85s
- **Profile Guided Optimization**: ~0.75s





Optimizations (Contest Results)

Results on CoolMUC-4 (`cm4tiny`)

Input	Compiler	Runtime (s)	MUPS/s	Threads
2D	**`g++ 13.2.0`**	**0,479**	**20.876.826,72**	**112**
2D	`icpx 2023.2.1`	0,570	17.543.859,65	112
3D	**`g++ 13.2.0`**	**11,533**	**8.670.770,83**	**224**
3D	`icpx 2023.2.1`	12,138	8.238.589,55	150

`g++:` PGO enabled, fast math, no outflow, coarse parallelization

`icpx:` PGO disabled, fast math, no outflow, coarse parallelization

Thermostat Update

New Thermostat attribute: *`nanoFlow`*

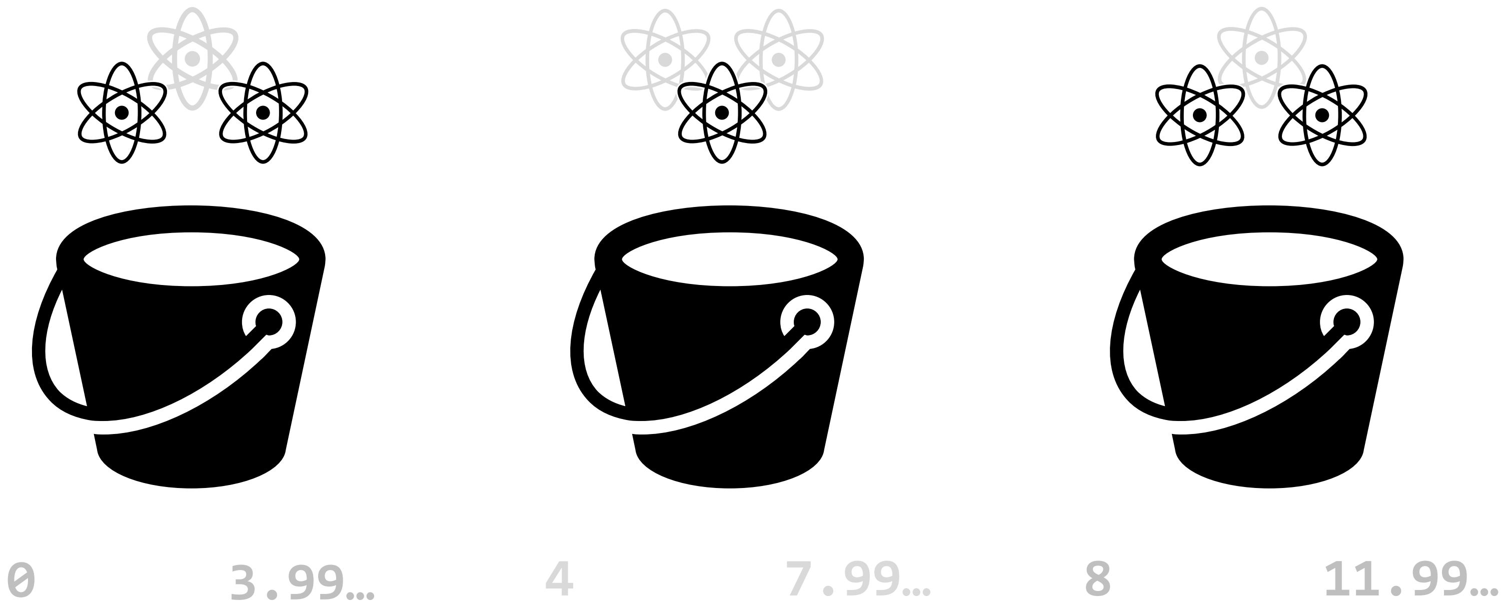
New Functionality: *`calculateThermalMotions()`*

No more **particle velocities**

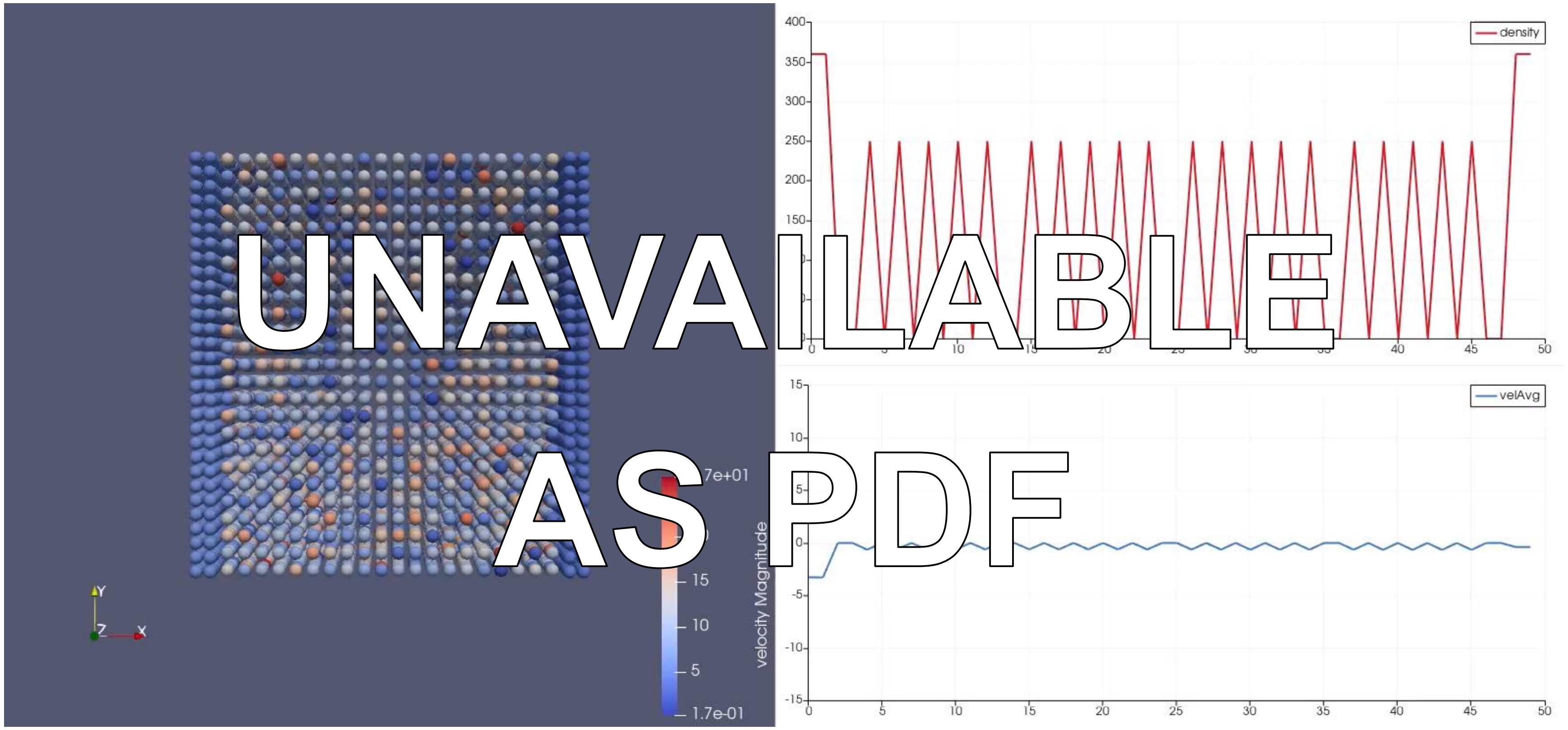
Welcome **thermal motions**!

Analyzer

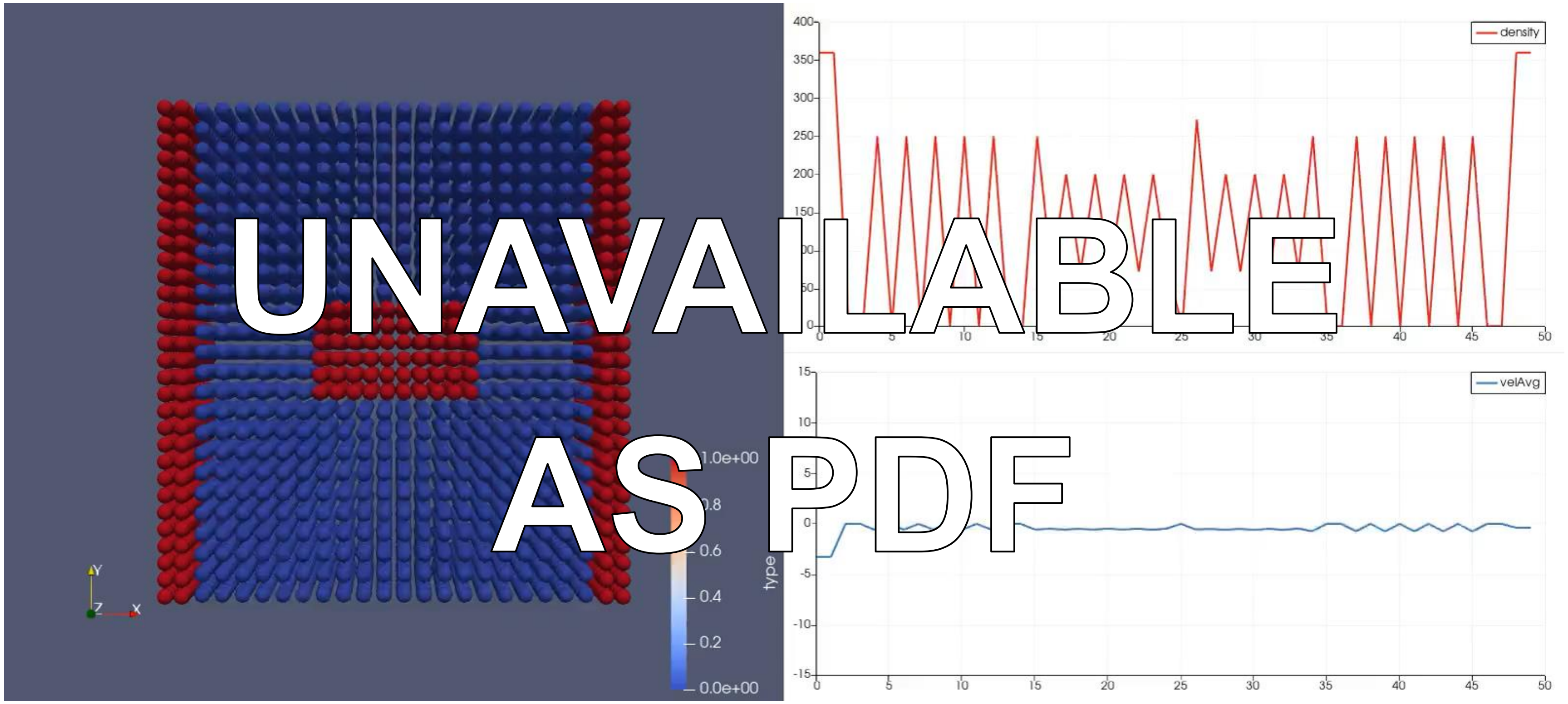
- Sort particles into **density** and **velocity** bins
- **Density vector**: Number of particles in each bin by coordinate on X-axis
- **Velocity vector**: Average velocity on the Y-axis of particles in each bin



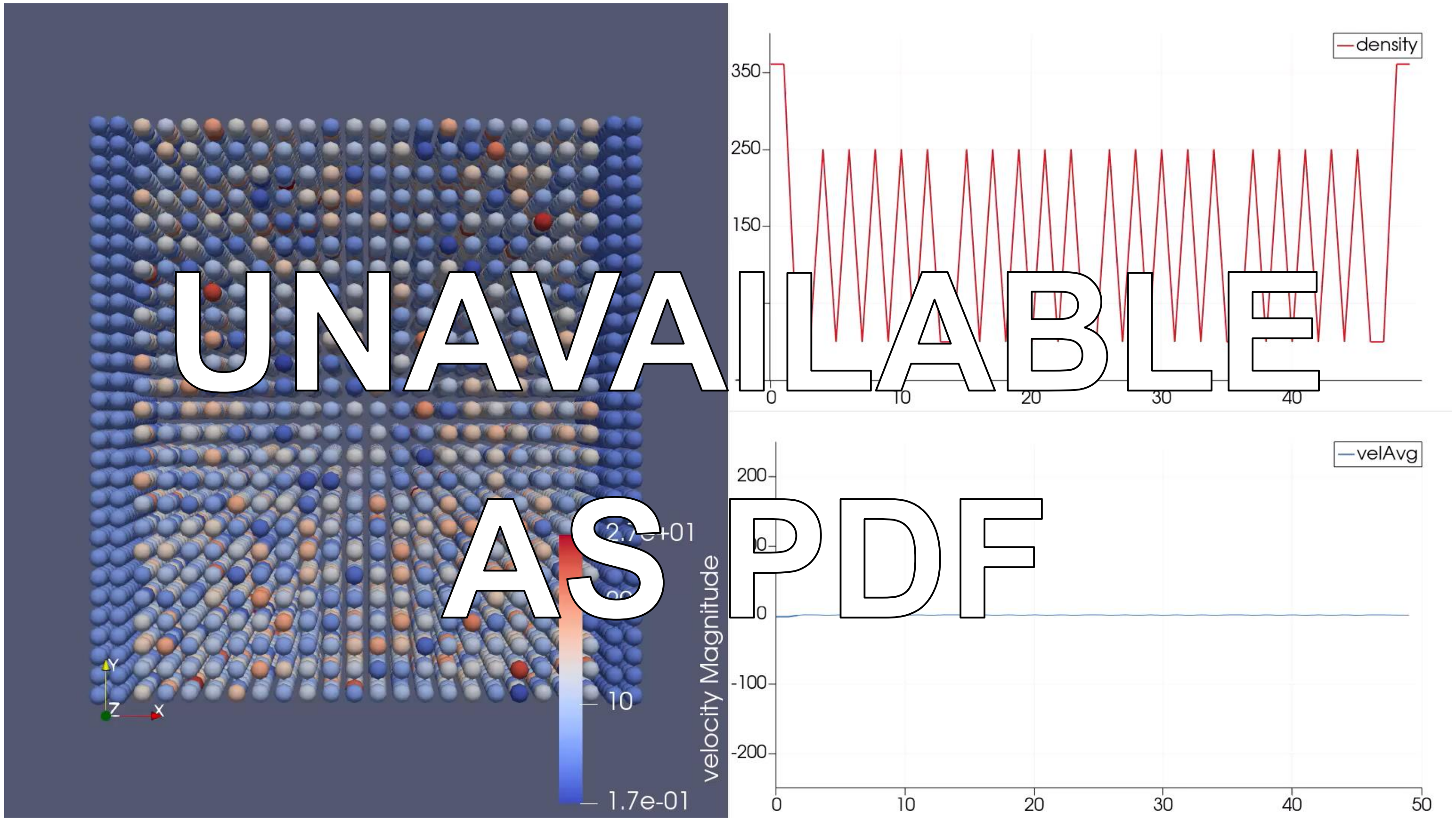
Nanoflow, Regular (Video)



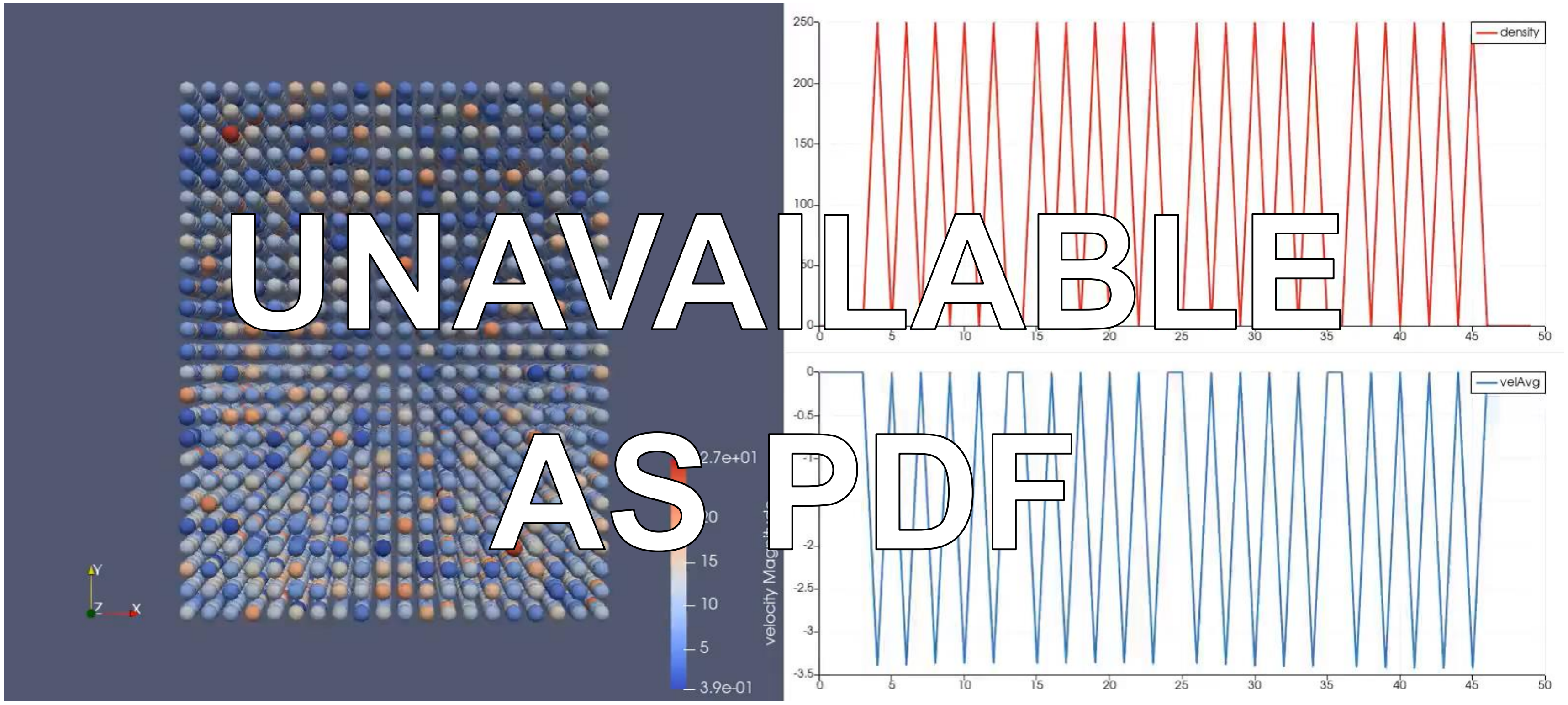
Nanoflow, Inner Block (Video)



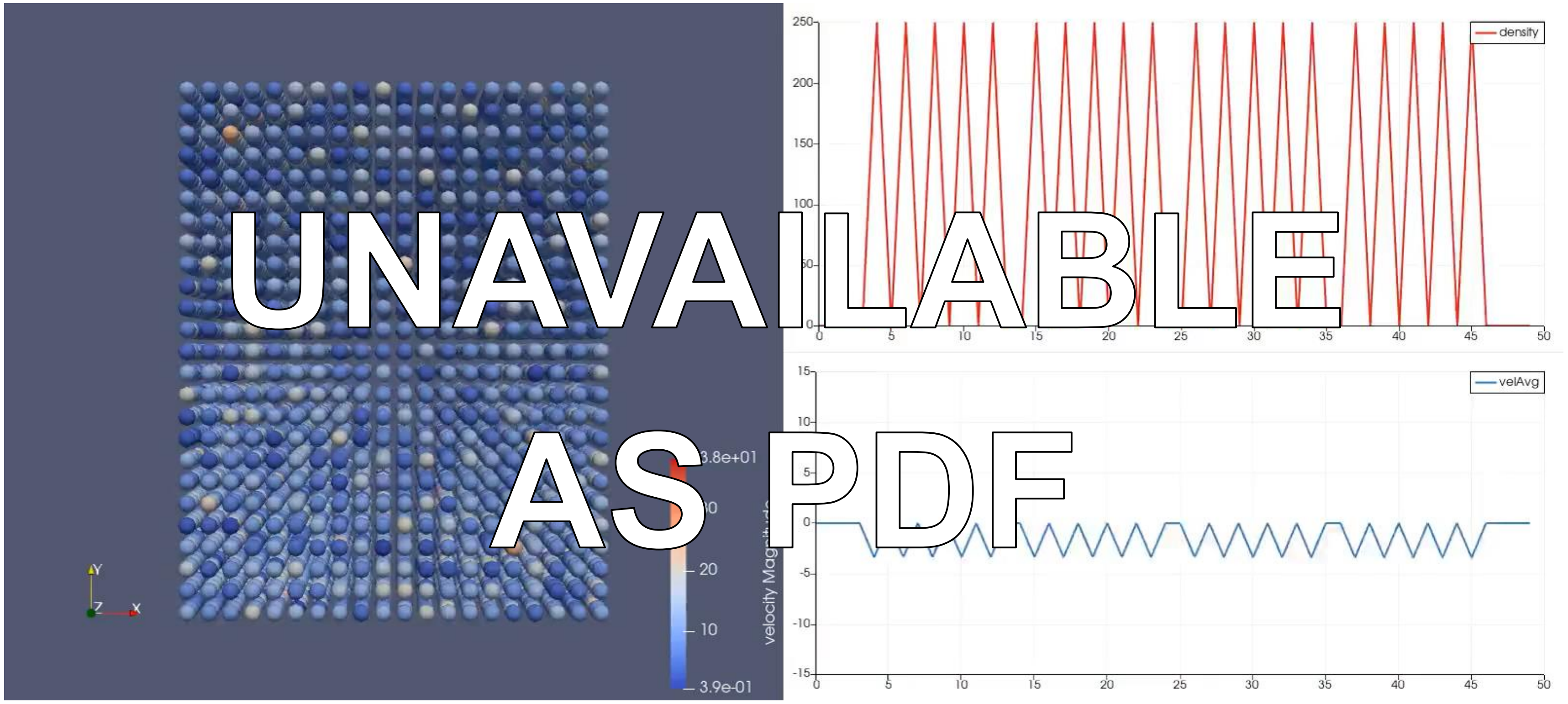
Nanoflow, Increased Gravity (Video)



Nanoflow, No Walls, Outflow (Video)



Nanoflow, No Walls, Reflective (Video)


UNAVAILABLE
AS PDF
3.8e+01
3.9e-01
velocity Magnitude
Y
Z
X
density
velAvg