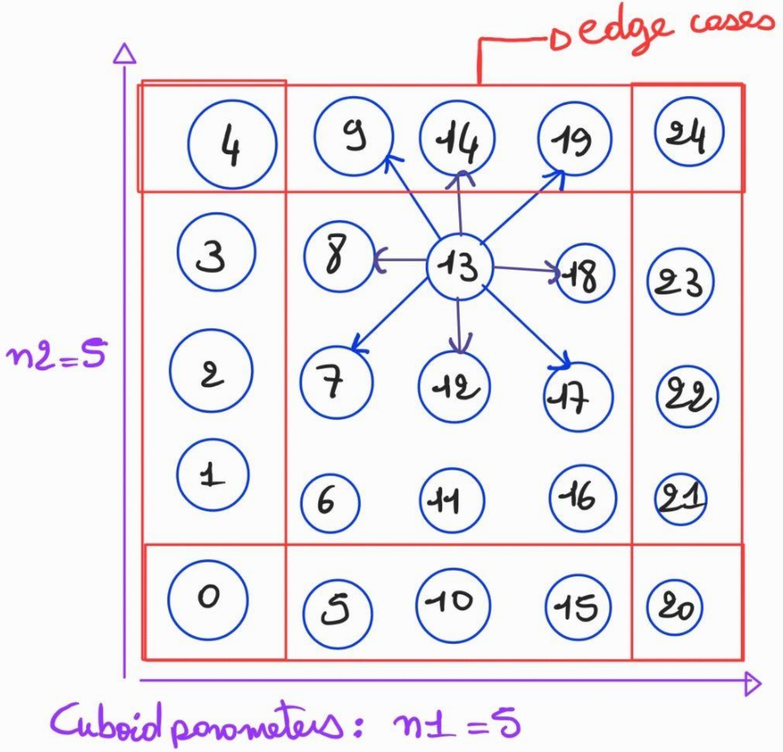
Worksheet 5

Membranes, Parallelization with OpenMP and Flow Simulation

Who are my neighbours ?



- particle id in membrane/cuboid generation

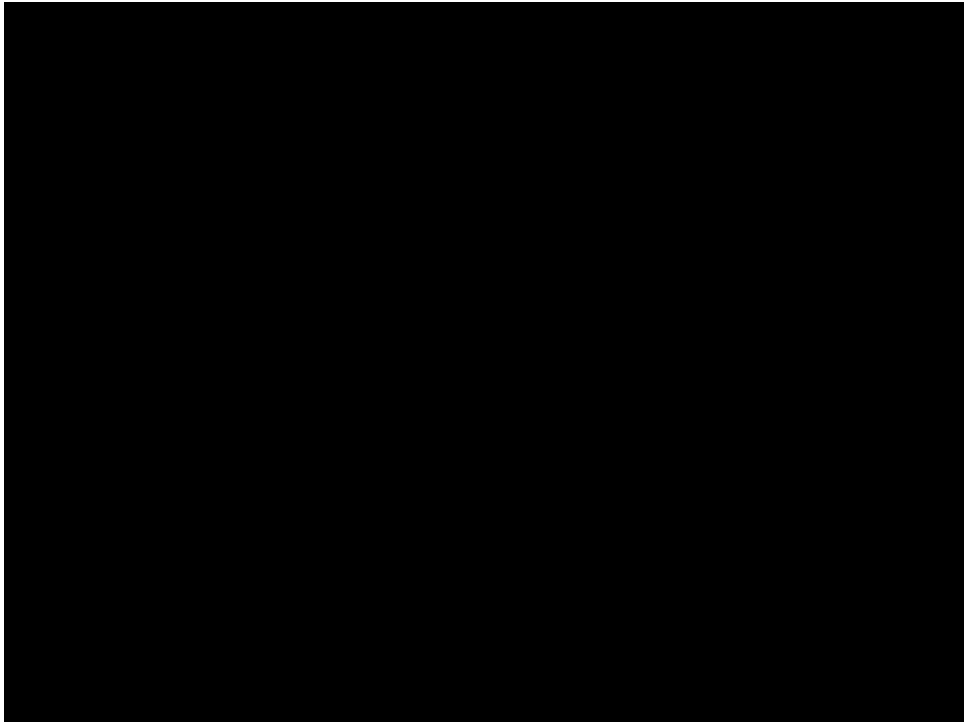
id=13 → diagonal neighbours:

$id+n2+1 = 19$
$id+n2-1 = 17$
$id-n2+1 = 9$
$id-n2-1 = 7$

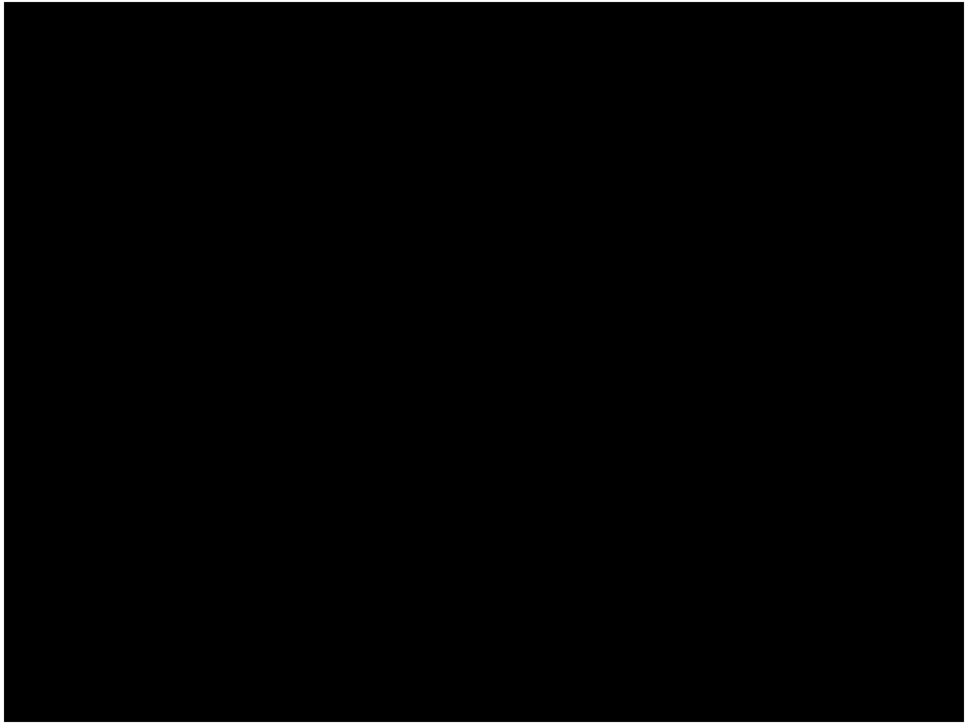
→ parallel neighbours:

$id+1 = 14$
$id-1 = 12$
$id+n2 = 18$
$id-n2 = 8$

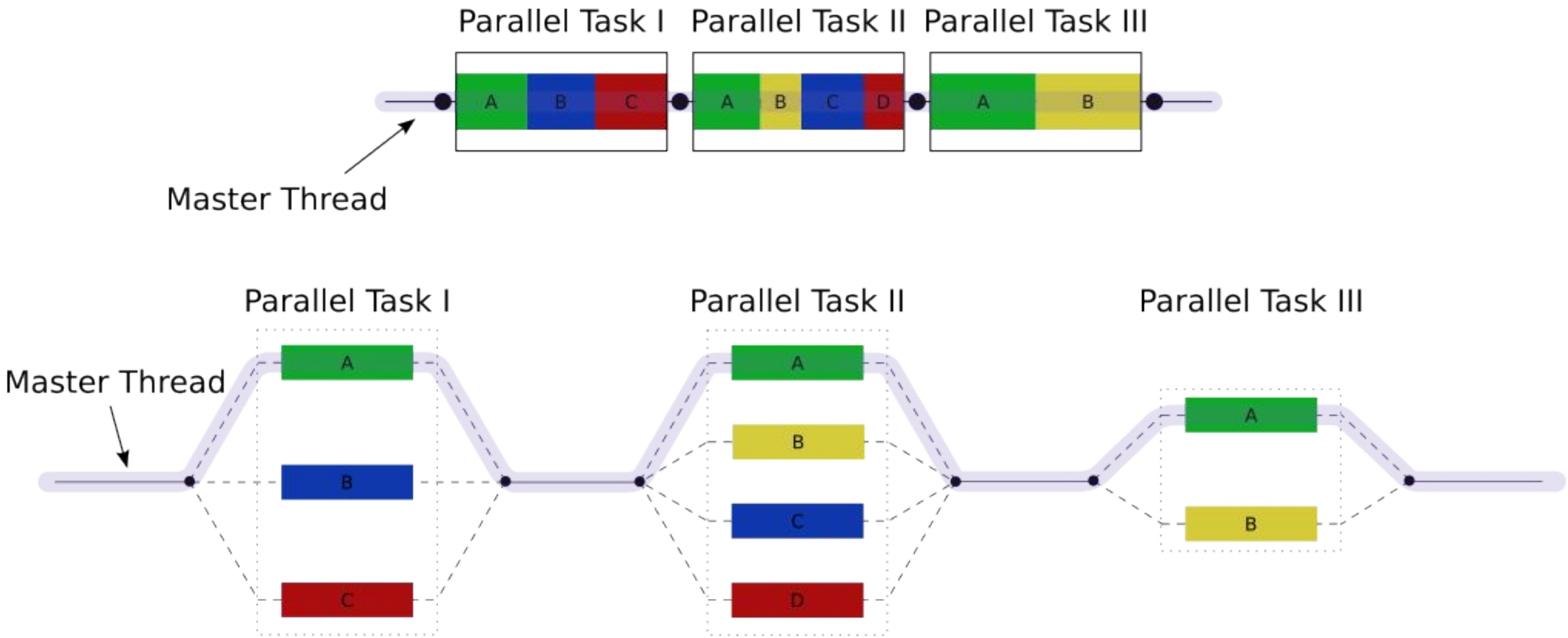
Membrane Flying



Membrane Folding



OpenMP - An introduction



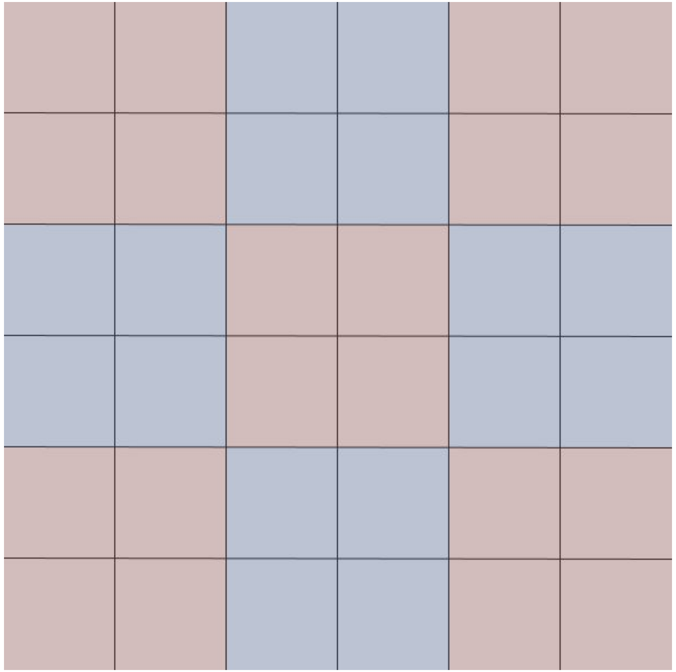
OpenMP - On code

OpenMP is based around defining parallel blocks and then assigning some work to the threads spawned by them

```c
#pragma omp parallel
{
   printf("This is a parallel region");
};
```

```c
#pragma omp parallel
{
   #pragma omp for
   for (int i = 0; i < size_of_arr; i++) {
       arr[i] += 1;
   }
};
```

OpenMP - The playing field



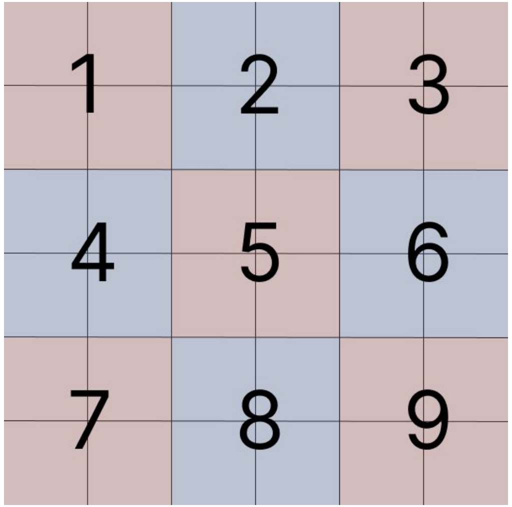
Our domain can be thought of as a grid of disjoint cells, which contains particles that might interact over cell boundaries.

Questions that arise:

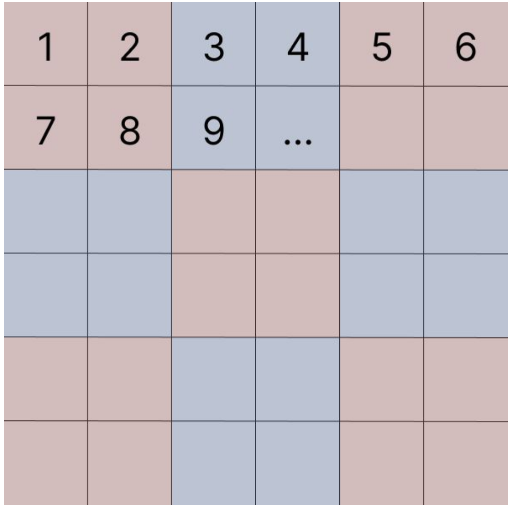
- How do we synchronize interactions on cells?
- Memory-Boundness?

OpenMP - Strategies for pairwise interactions

Subdomain



Over cells



OpenMP - But what about synchronization?

If we do pairwise interactions across cells, it might be that we have unordered reads and writes onto the same particle. This will case data inconsistency!

OpenMP - But what about synchronization?

On the other hand, if we synchronize our accesses too tightly, we don't get parallelism.

OpenMP - But what about synchronization?

And even if the synchronization ensures data consistency and good performance, it might deadlock!

OpenMP - Cell-wide Locking + Sort’n’Lock

Cell-wide Locking

This refers to using a index list of OpenMP locks to prevent individual cells from being written/read over while a thread is working

Sort’n’Lock

This is a common pattern in parallel applications where we ensure consistency in the order of lock acquisition by means of a total order. This way we let go of the infamous *Hold and Wait* condition, which is one of the 4 required conditions for a deadlock.

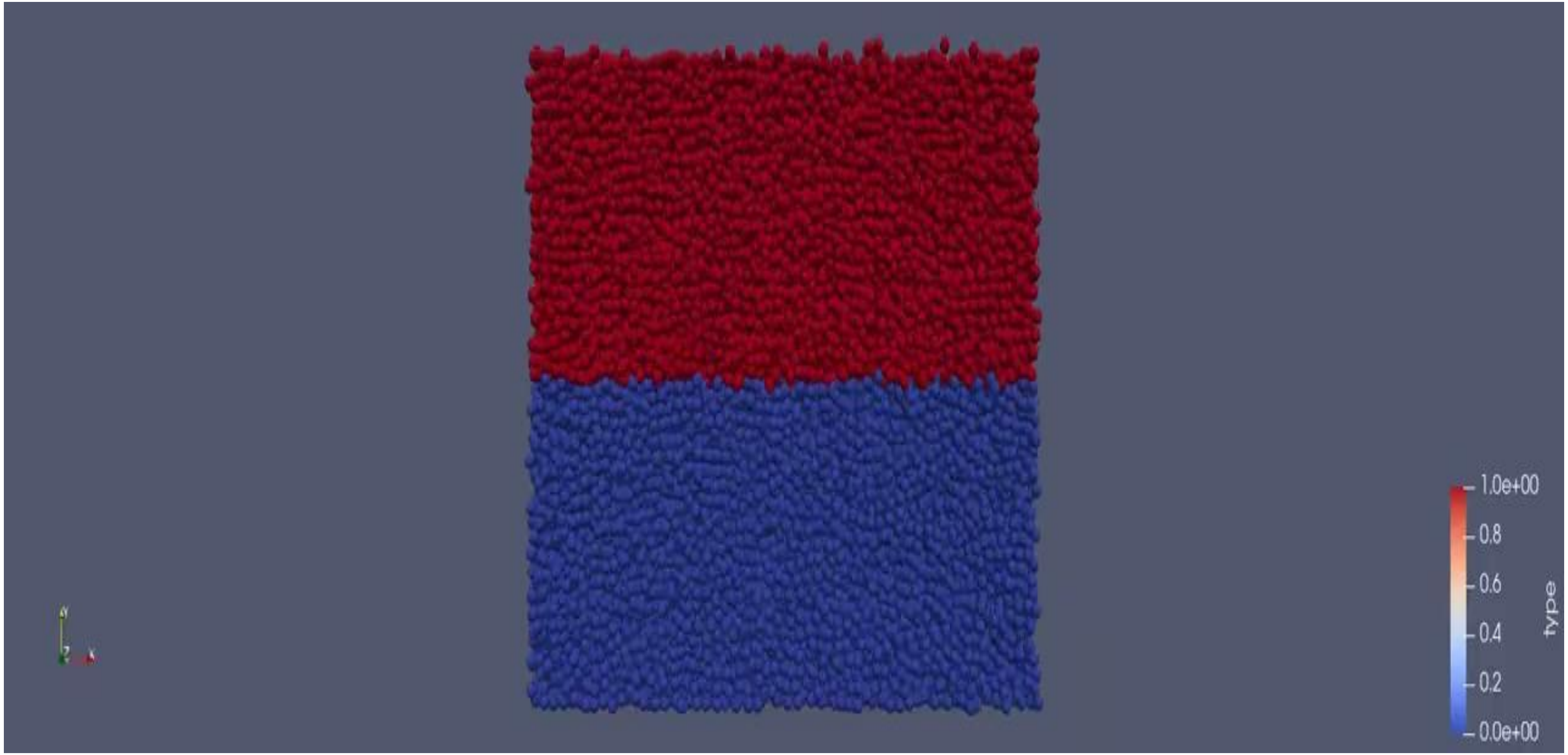
OpenMP - PAPI

- Highly recommended!

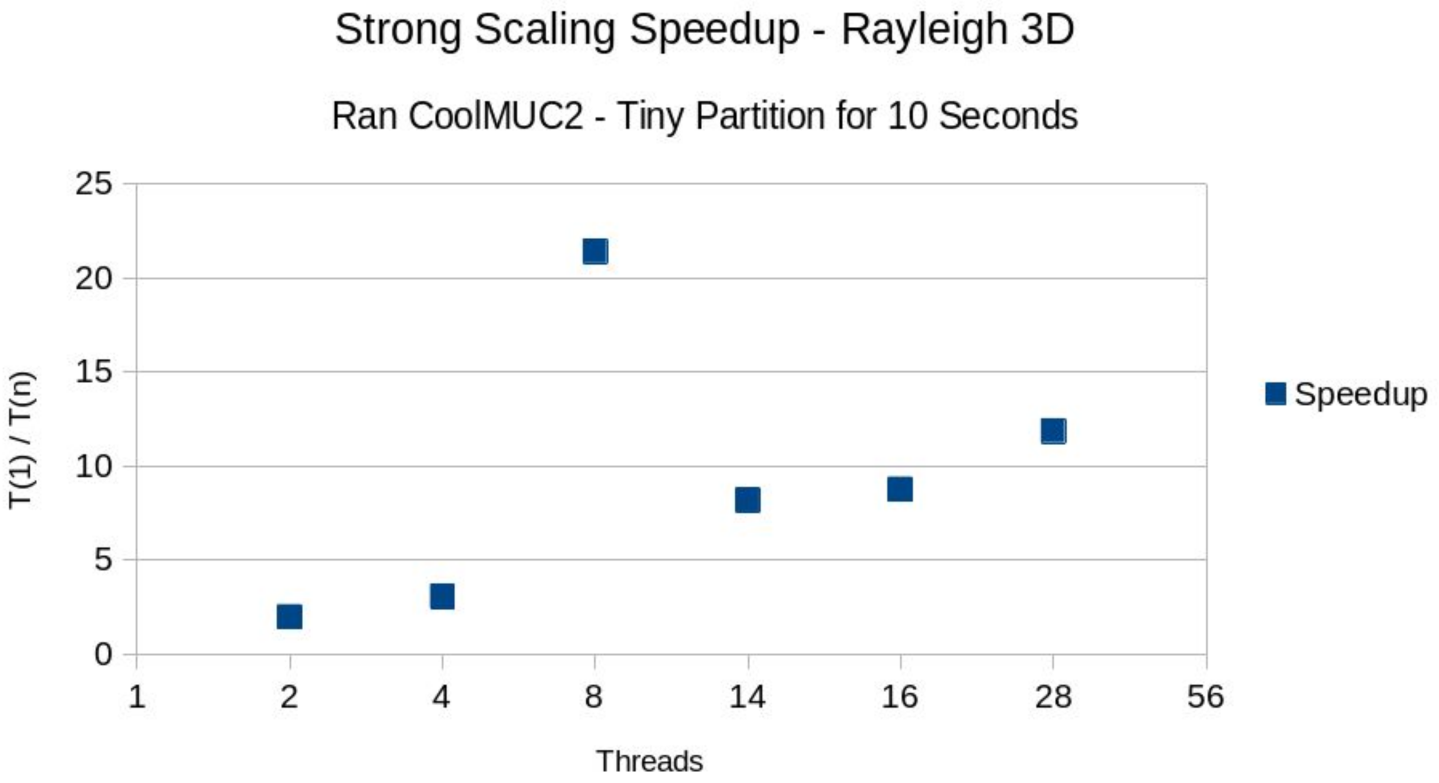
With this performance analysis tool, we were able to

1. Evaluate lock contention on individual functions
2. Evaluate cache miss rates for individual functions
3. Identify bottlenecks within the program flow

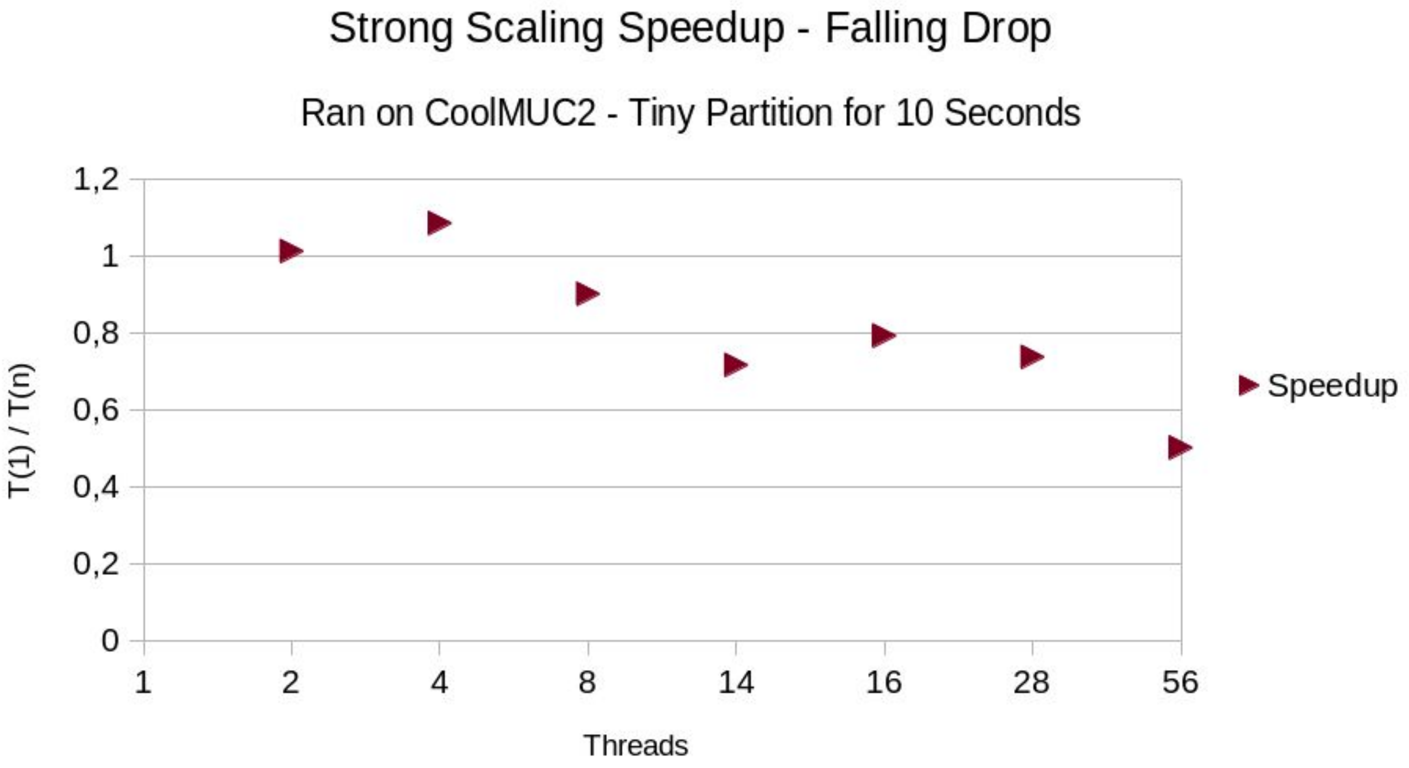
OpenMP - Rayleigh 3D Sim.



OpenMP - Types of job and parallelism



OpenMP - Types of job and parallelism



Task 4 - Implementation

- New subclass of Thermostat class
- Implementation similar to simple thermostat
- ParticleStatistics class - calculates values and outputs them to a csv file

```
iteration_0-bin_1-velocity:0.494889|-5.860183|-0.005233,
iteration_0-bin_1-density:0.472222,
```

Task 4 - Various influences

- **Mass:** 0.1 vs 3.0 -> no big change
- **Spheres in liquid:** effect depends heavily on the size of the sphere
- **Thermostats:** new thermostat vs old -> only small differences
- **Sigma:** 0.1 vs 3.0 -> low sigma - liquid goes through walls and low velocities, high sigma - extremely high velocities
- **Epsilon:** no change, visually or in values
- **Removing walls:** liquid expands to fit domain, lower density

Lessons Learned and Project Reflections

- Importance of communication
- Design comes before implementation
- Adaptable project plans
- Feedback driven development
- Documentation for knowledge transfer

References and Bibliography

1. https://en.wikipedia.org/wiki/Fork%E2%80%93join_model#/media/File:Fork_join.svg