RUSPH

Building a SPH astrophysical simulation code in the Rust programming language

Jacksen Narvaez

Supervisor: Dr. Terrence Tricco

February 15th, 2024

5th Phantom+MCFOST users workshop 2024

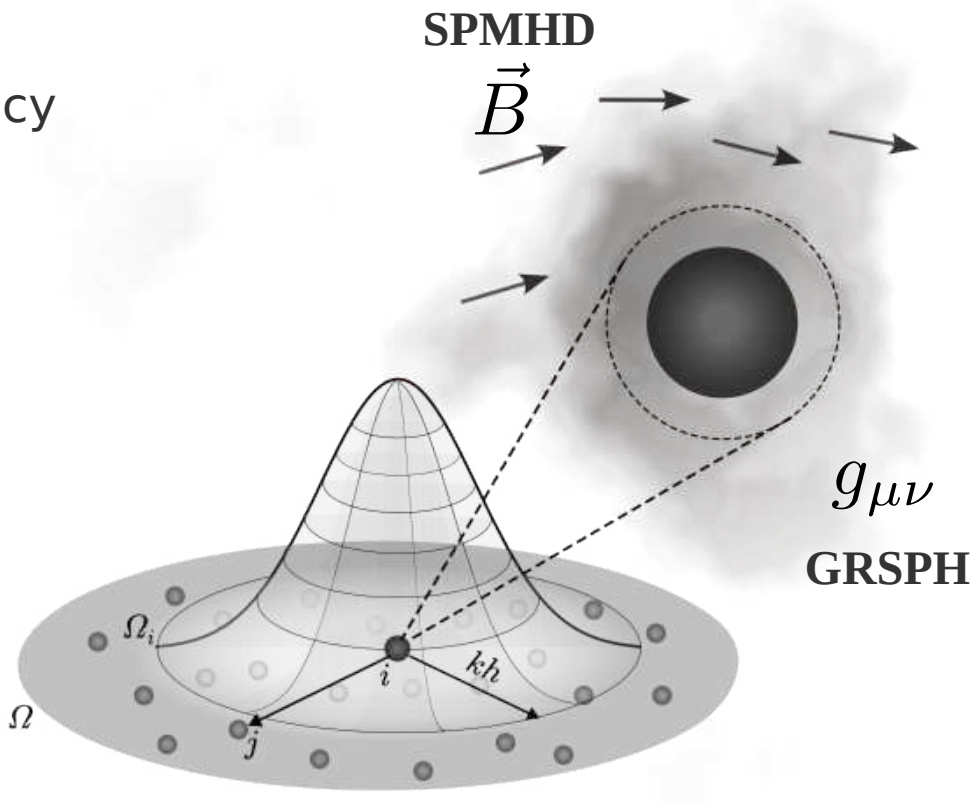
Simulation, Modeling and Synthetic Data Lab

01 – Motivation

- (1977) Gingold & Monaghan / Lucy
  ~ 10^2 Particles
- Currently,
  ~ 10^7 Particles

High **Performance** Codes

... Not only speed, but **safety**.



02 – Rust

2006 - A personal side project of Graydon Hoare

2010 - Mozilla Sponsorship

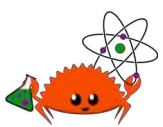
2015 - The first stable release, *Rust 1.0*

The main focus is (memory) Safety

- Null pointers
- Memory Leaks
- Race conditions

Efficient in performance.

Not yet widely used in scientific computing...
But it is emerging!



02 – Rust

Ownership:

Set of rules that the compiler checks.

- Every value in Rust has an owner.
- There can only be one owner at a time.
- When the owner goes out of the scope of the application, the value is removed.

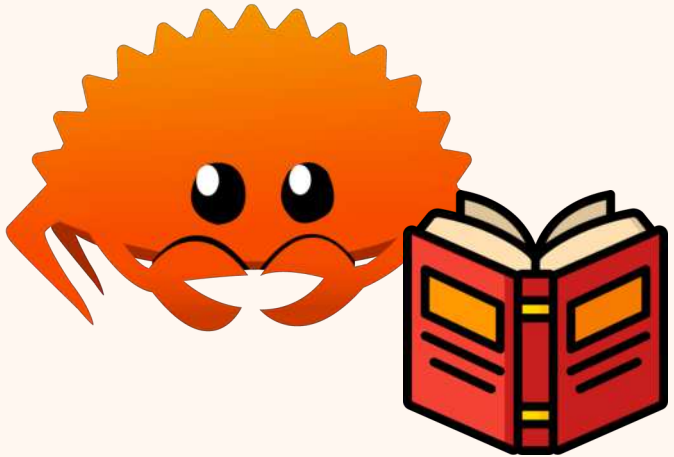
Borrowing:

The action of creating a **reference** - obtaining some value without taking ownership.

- You can have one mutable reference or any number of immutable references.
- References must always be valid.

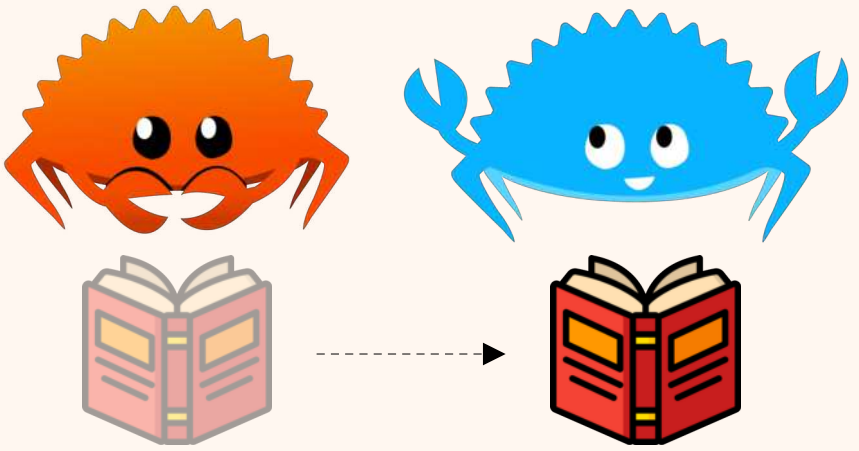
02 – Rust

Owning a <book>:
You can do what you want with it.



B

You can show your <book> to a friend.

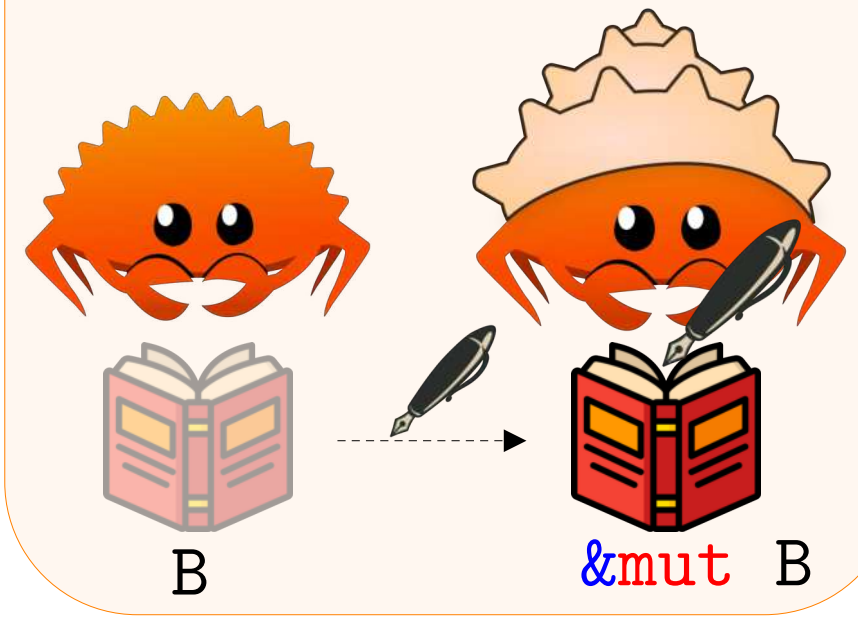


B

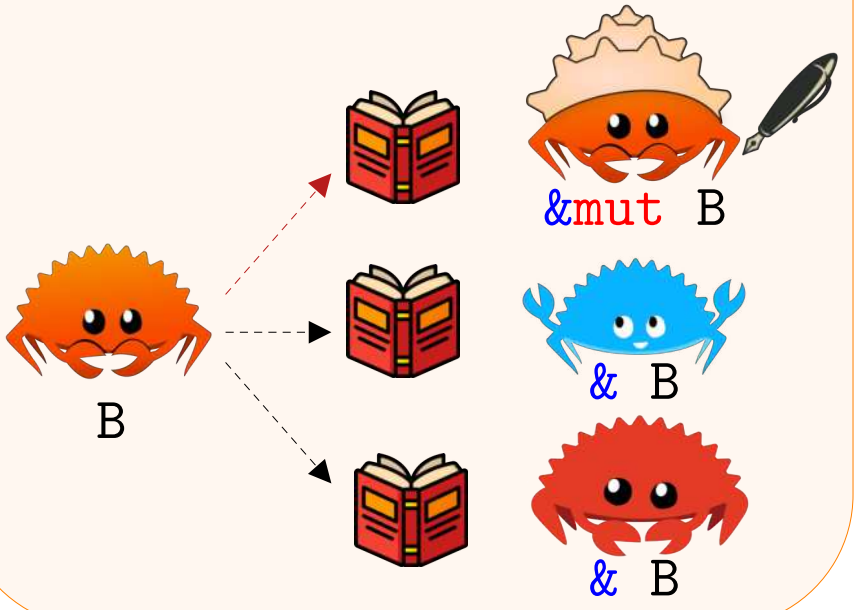
& B

02 – Rust

You can offer to a friend a pen to write in the <book>.

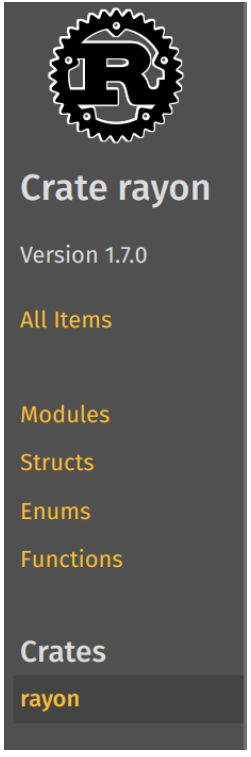


Only one mutable reference



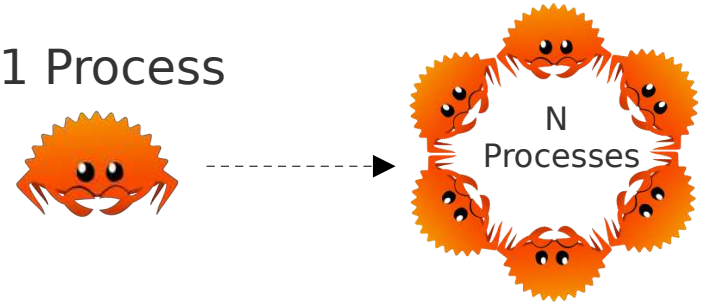
02 – Rust: Parallel computing

Rayon



Data-parallelism library that converts sequential computations into parallel.

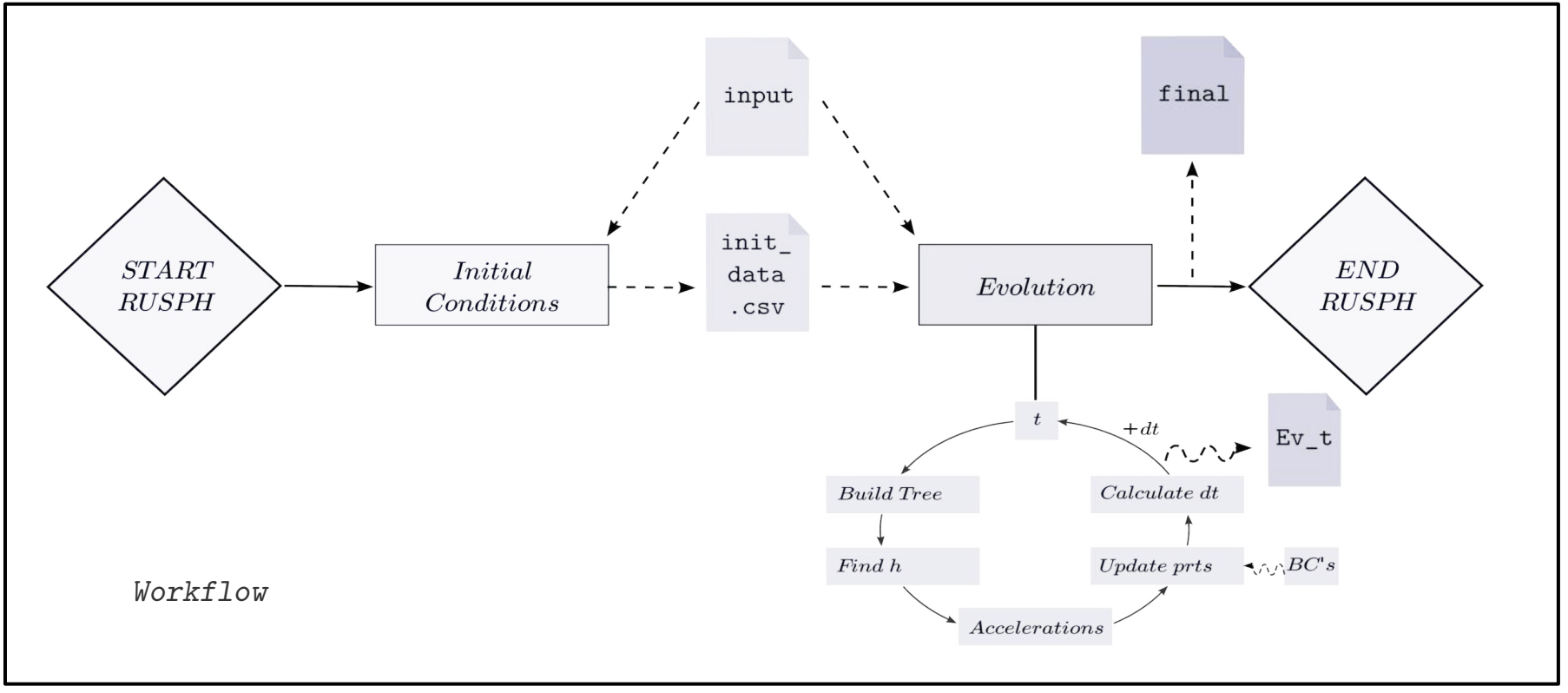
Memory Safety




```rust
use rayon::prelude::*;

fn main() {
    let s1: [i32; 10]  = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

    let sum: i32 = sum_of_squares(& s1);

    println!("The sum is '{}'.", sum);
}

fn sum_of_squares(input: &[i32]) -> i32 {
    input.par_iter() // <-- just change that!
         .map(|&i| i * i)
         .sum()
}
```

03 – The Code: Rusph

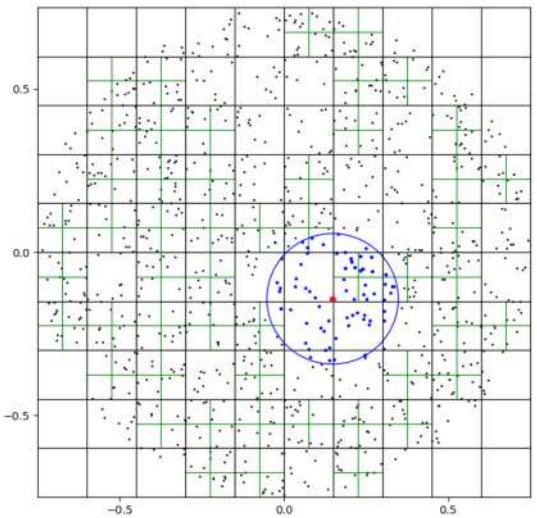
Code https://github.com/JackNarvaez/Rusph.git



03 – The Code: Rusph

1. The B-Tree Algorithm

$\mathcal{O}(N \log N)$



Finding Neighbors

	Worst Case	Best Case
b-tree	$O(nd)$	$O(n)$
octree	$O(nd)$	$O(n \log n)$

2. Artificial Viscosity

$$\frac{d\mathbf{v}_a}{dt} = -\sum_b \frac{m_b}{\bar{\rho}_{ab}} v_{sig} \mathbf{v}_{ab} \cdot \hat{\mathbf{r}}_{ab} \nabla_a W_{ab}$$

$$v_{sig} = \begin{cases} \frac{1}{2}\alpha \left[c_{s,a} + c_{s,b} - \beta \mathbf{v}_{ab} \cdot \hat{\mathbf{r}}_{ab} \right]; & \mathbf{v}_{ab} \cdot \hat{\mathbf{r}}_{ab} \leq 0 \\ 0; & \mathbf{v}_{ab} \cdot \hat{\mathbf{r}}_{ab} > 0 \end{cases}$$

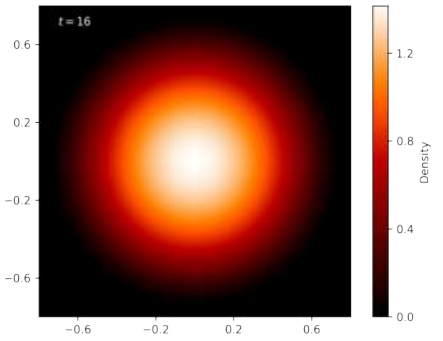
3. Artificial Conductivity

$$\frac{du_a}{dt} = \sum_b \frac{m_b}{\bar{\rho}_{ab}} \alpha_u v^u_{sig} \left(u_a - u_b \right) \hat{\mathbf{r}}_{ab} \cdot \nabla_a W_{ab}(h_a)$$

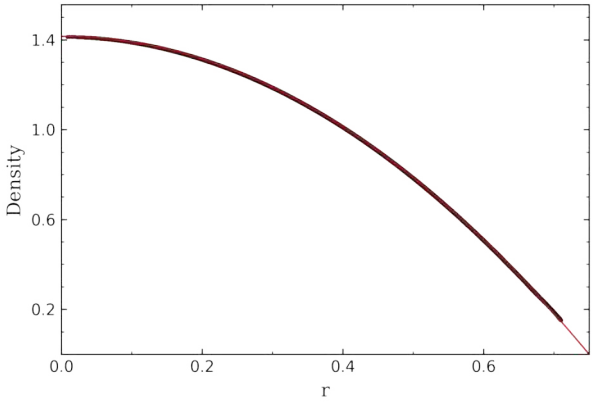
Cavelan et al. (2020)

04 – A look at our results

The toy star model

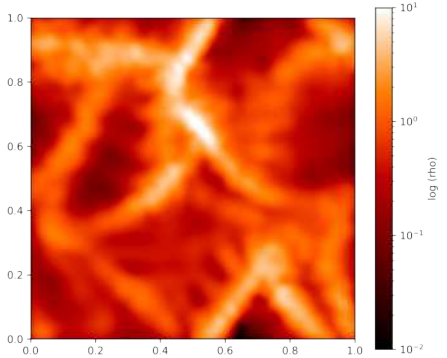


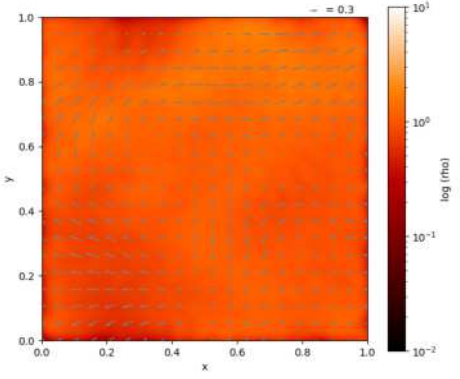
Monaghan & Price, (2004)



~190K Particles

Turbulent density

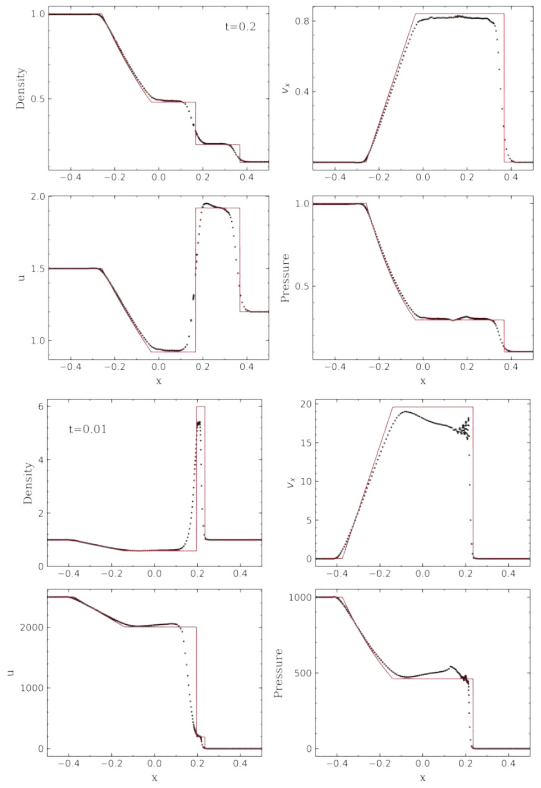




~32K Particles

04 – A look at our results

Shock tests

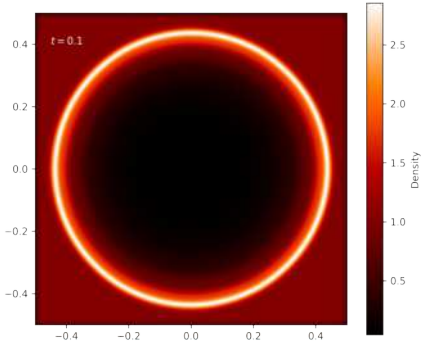


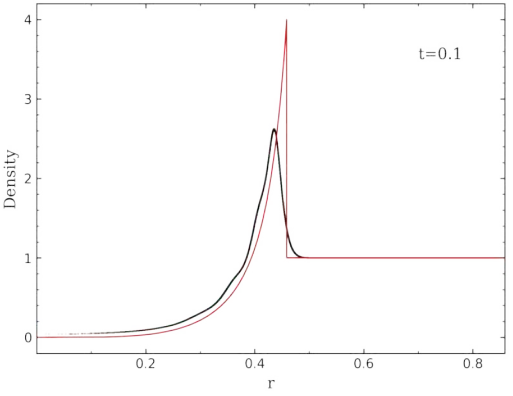
~170K Particles

~170K Particles

The Sedov blast wave

Sedov, (1959)



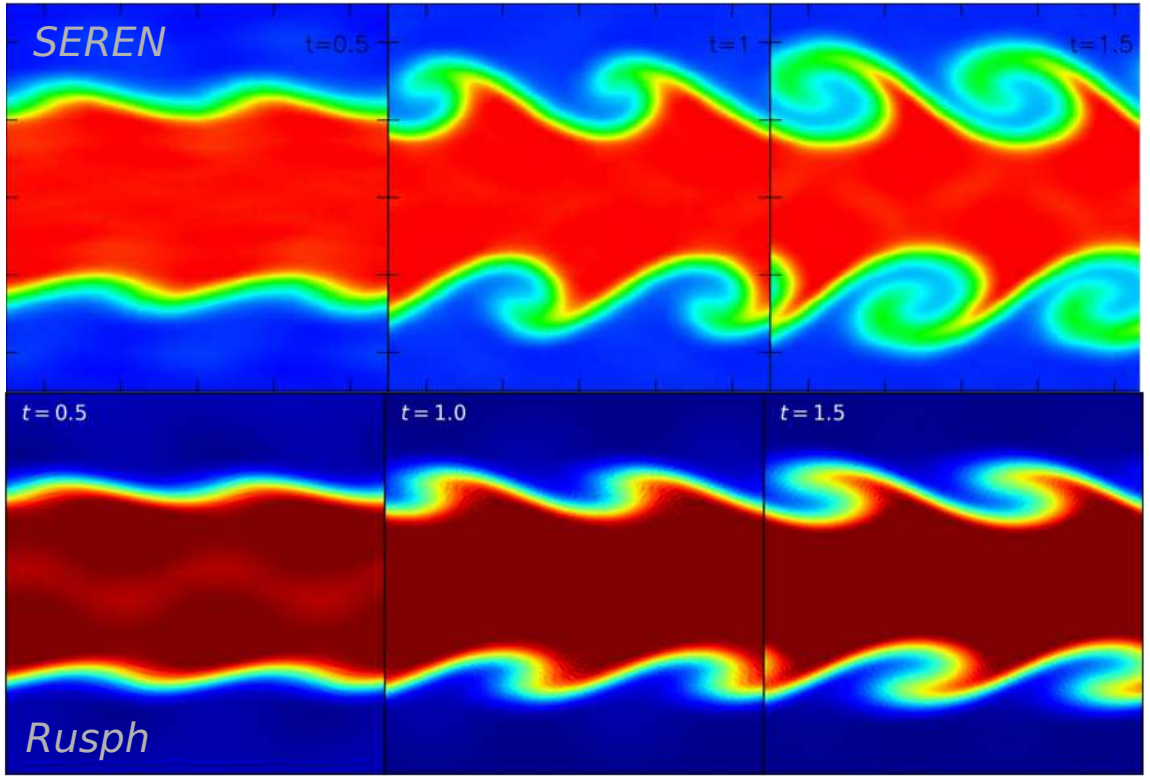


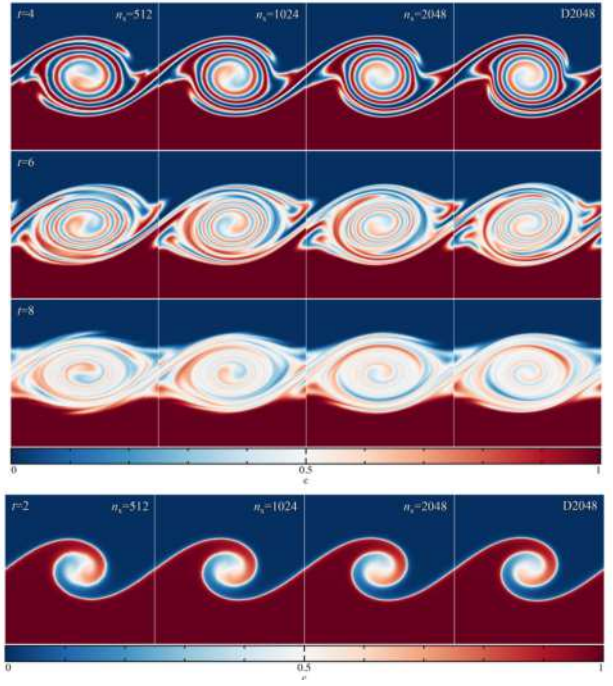
~350K Particles

04 – A look at our results

Kelvin Helmholtz Instabilities

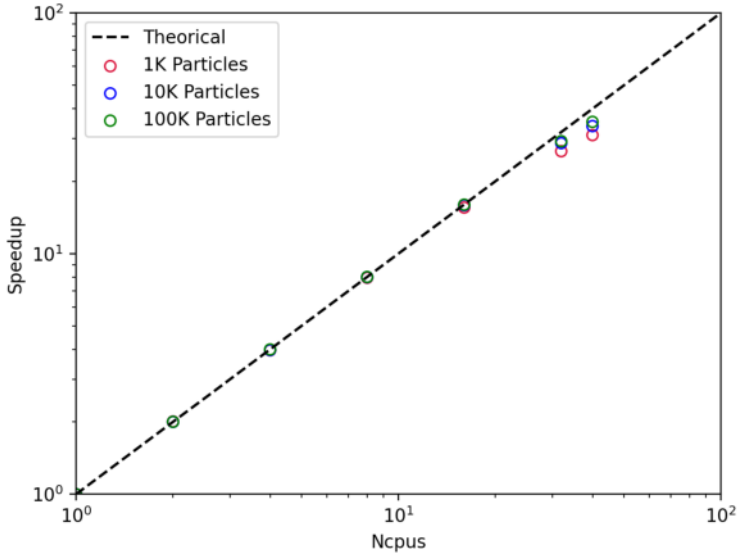
Tricco, (2019) MNRAS 488, 5210–5224



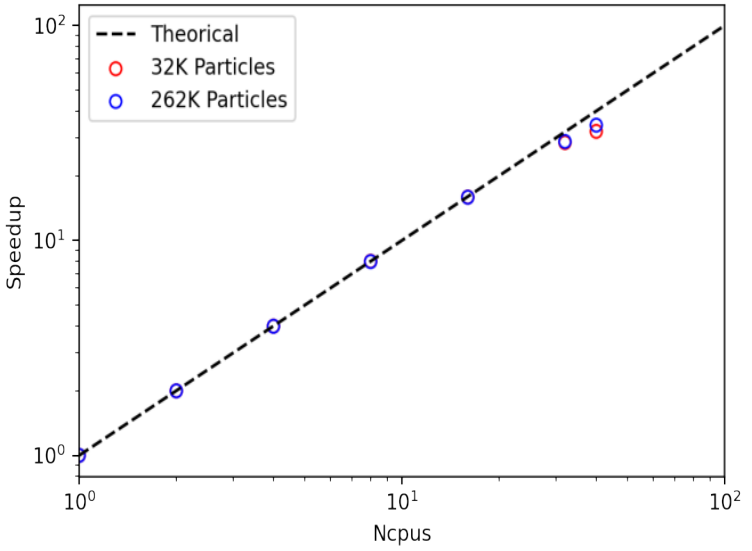


04 – A look at our results

Speedup: Strong scaling results for:



The Sedov Blast Wave Problem

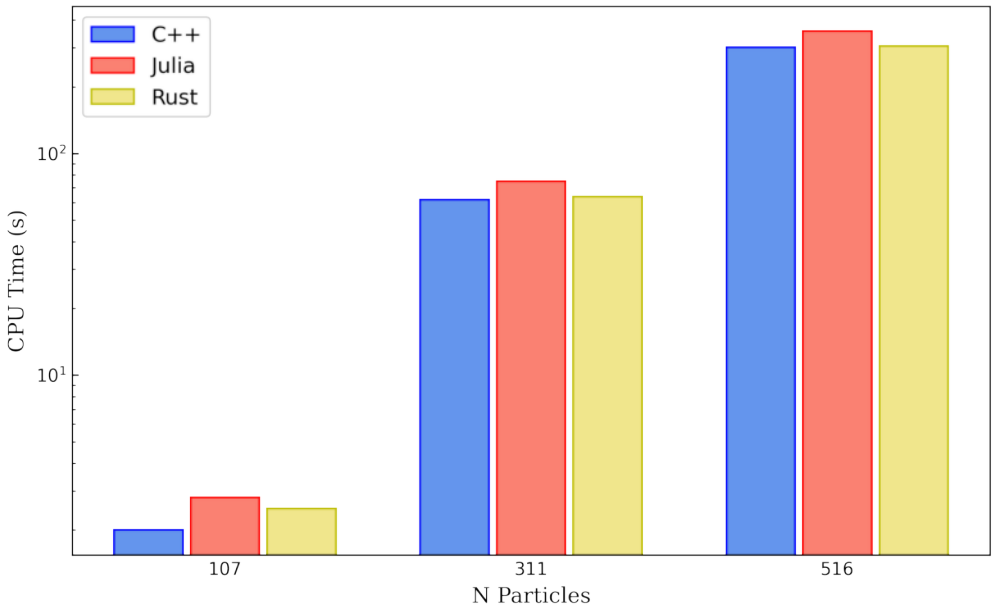


The Turbulent Gas Problem

* Tests performed on **CAIR**(IBM Power9 architecture)

04 – A look at our results

What about performance?



Basic SPH toy star model

05 – Summary

- **Rust**
  - Prevents most memory errors.
  - Performance advantages.
- Strong scaling results.
- Growing scientific ecosystem.
- **Rusph future:**
  - Include Gravity.
  - Improve performance.

Thank you!

Code is there for humans, not computers, to understand.

Rust Community.