GPUs in Space: Extending GPU Support for SPH

NASA: James Webb Space Telescope, NVIDIA: RTX 2080 Technical Photography

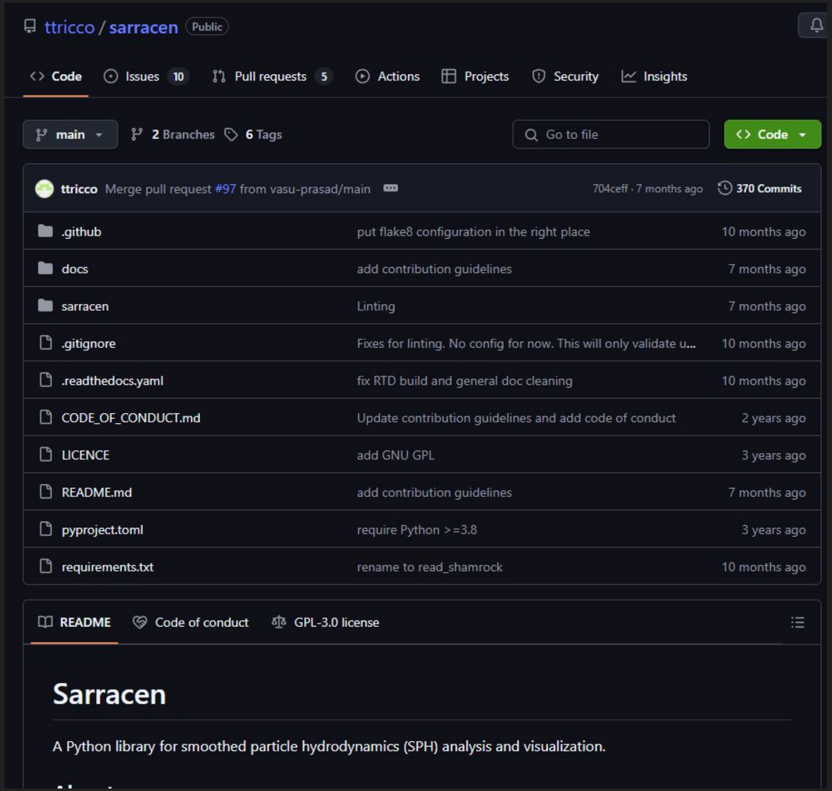
Student: Andrew Harris
Supervisor: Dr. Terrence Tricco

MEMORIAL UNIVERSITY


MEMORIAL
UNIVERSITY

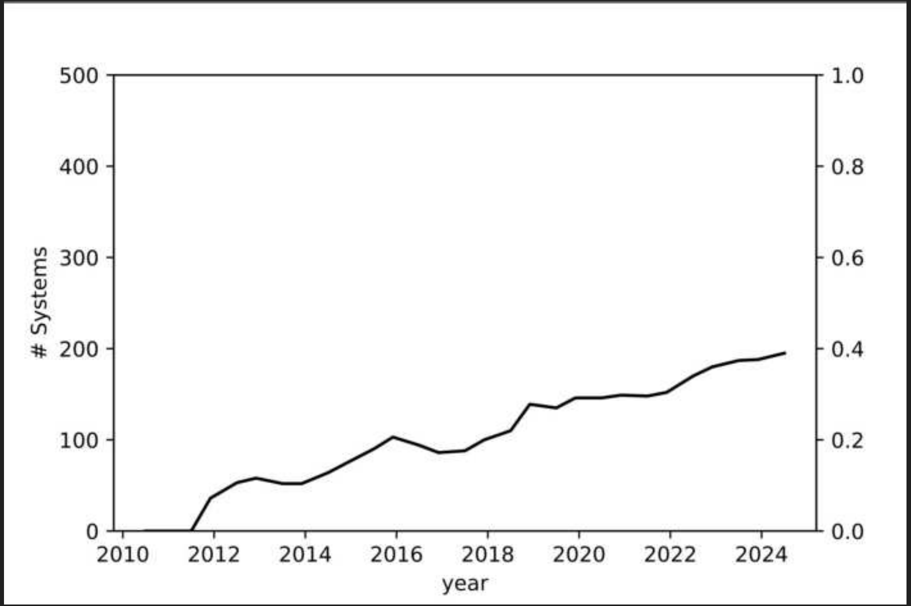
Sarracen

- Sarracen is a Python-based tool performing analysis and visualization of SPH data.
  - Built in collaboration with Dr. Tricco
- Open-source, and available for installation on PyPI!



The Computational Cost of SPH

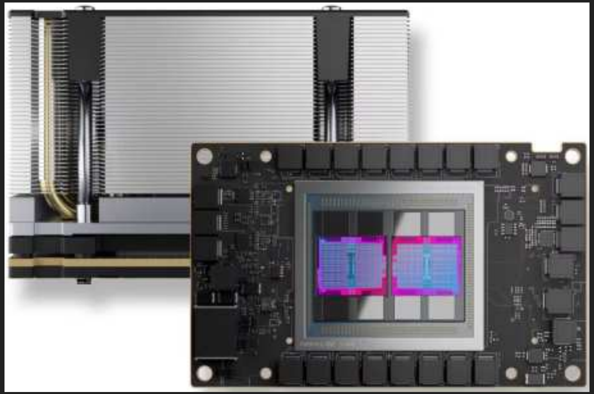
- The number of GPUs available on large-scale computing systems is increasing year-over-year.
- Astrophysical SPH codes should take advantage of these resources, to enable larger and more efficient simulations.



top500.org: % of top 500 supercomputers with GPUs. Graph credit: Dr. Tricco

The Challenge with GPUs

On Adastra (CINES, accelerated nodes):



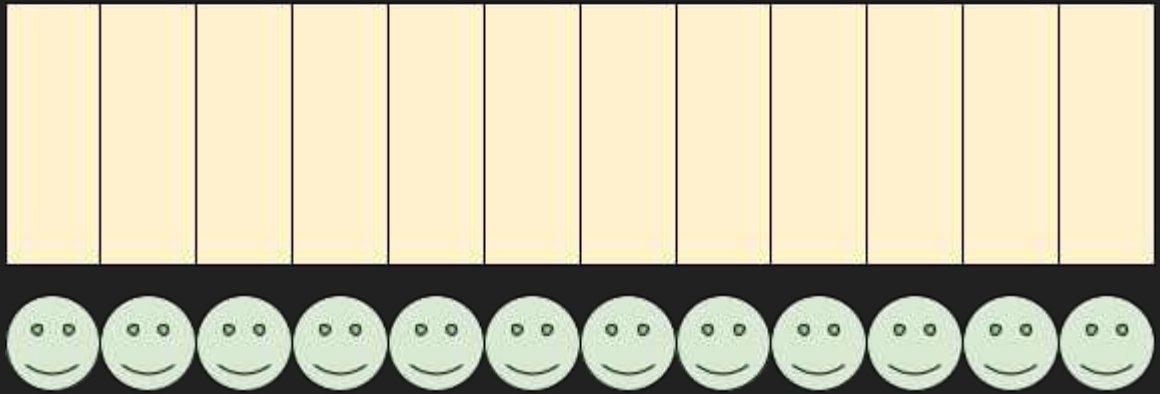
AMD 7A53: 64 cores

AMD MI250X: 4x 14080 cores

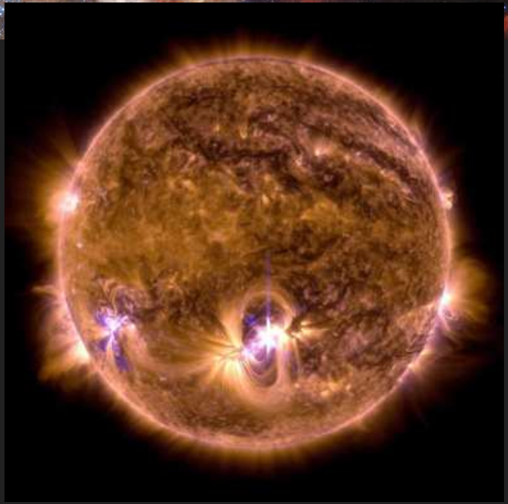
Image Credits: AMD

The Challenge with GPUs

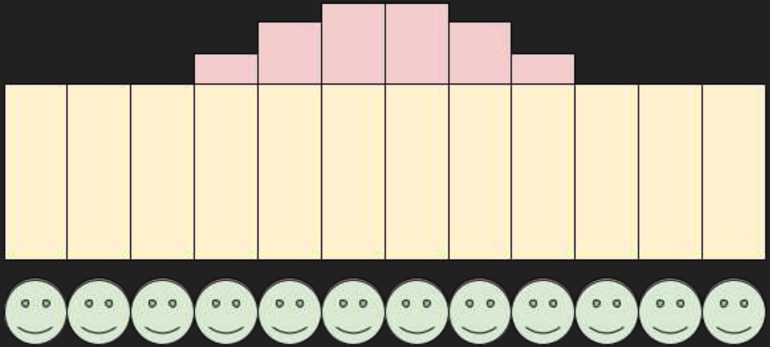
- SPH computations must be distributed amongst the available cores.
- In an ideal world, all cores have the **same amount** of work.



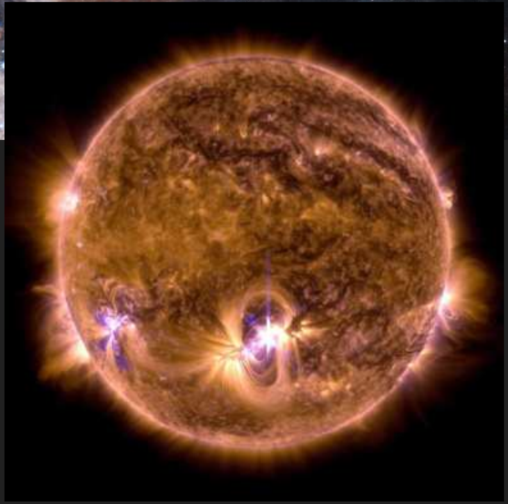
Density Changes



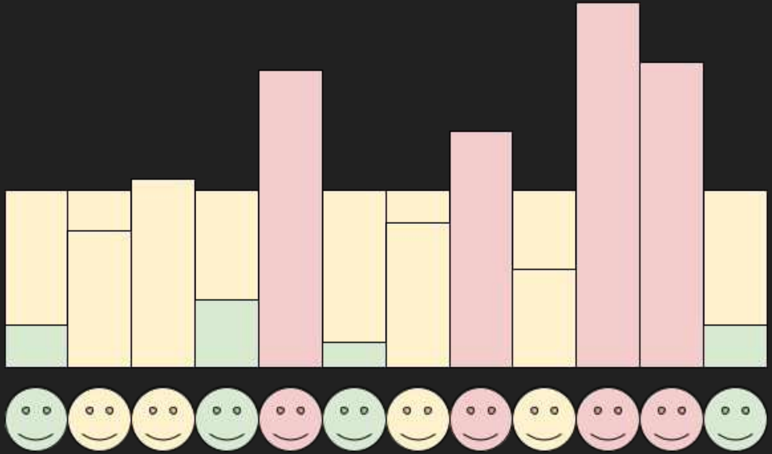
- If we divide computations **by region,** we end up with an uneven workload, since particle density can **vary greatly** throughout the simulation.
- Dividing work by **number of particles** does not work either, since particles in **dense regions** have **more neighbours** to consider.



Moving Particles



- A particle’s **neighbours are always changing**. This makes the amount of work variable and hard to predict.
- The workload must be **rebalanced** across cores as the simulation evolves.

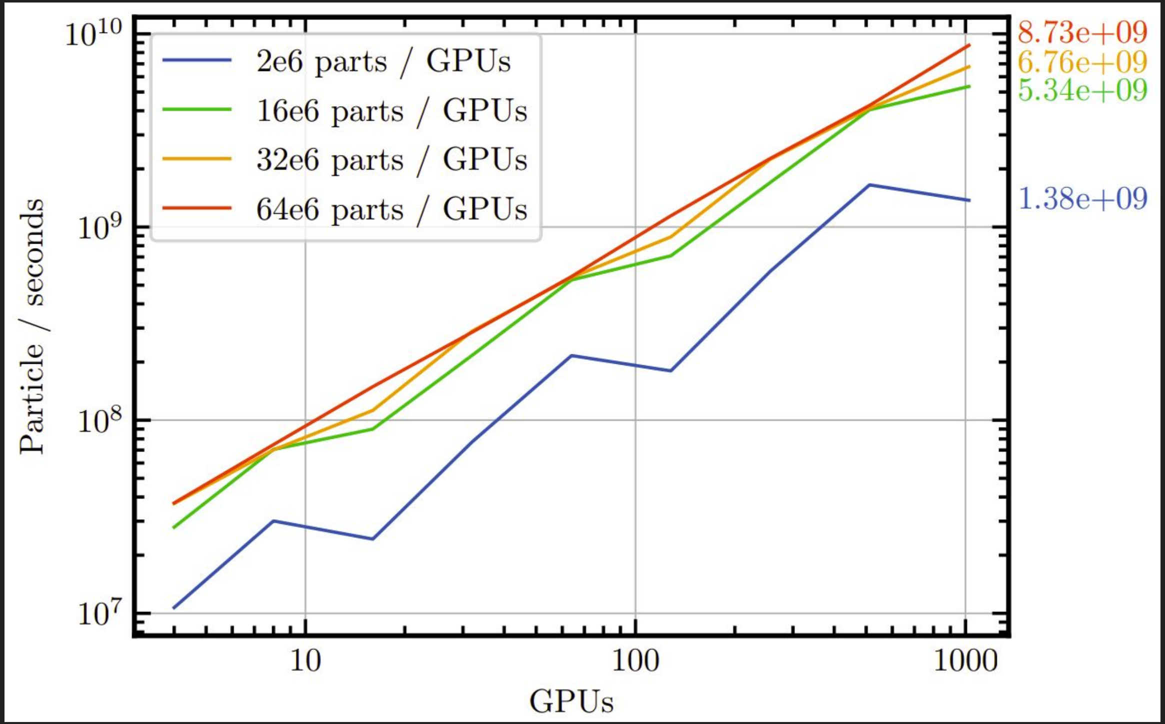


Shamrock

- A modern astrophysical code that operates on clusters of **GPUs**.
- Highly optimized for **speed**, with major algorithmic optimizations.

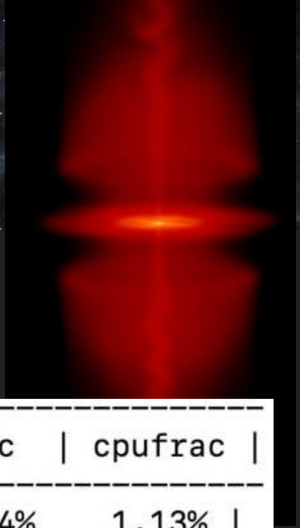
Logo from: https://shamrock-code.github.io/

Shamrock Performance



Performance scaling of Shamrock on the CINES Adastra supercomputer

Variable Timesteps



- Across a simulation, some particles may need to update **more frequently** than other particles.
    - For example: Dense vs. Sparse regions
- We can give each particle an **individual timestep**, to only update particles as required.

bin	dt	npart	frac	cpufrac
0	1.111E-02	950054	62.14%	1.13%
1	5.554E-03	110009	7.20%	0.64%
2	2.777E-03	164209	10.74%	1.36%
3	1.388E-03	148126	9.69%	2.54%
4	6.942E-04	86629	5.67%	5.60%
5	3.471E-04	39425	2.58%	9.90%
6	1.736E-04	22797	1.49%	17.10%
7	8.678E-05	7285	0.48%	24.88%
8	4.339E-05	417	0.03%	36.84%

Breakdown of particle timestep bins in Phantom

Potential Issues

- Particles with a more frequent timestep may be difficult to split over the cores of a GPU.
- The more frequent timesteps could significantly **bottleneck** the performance.

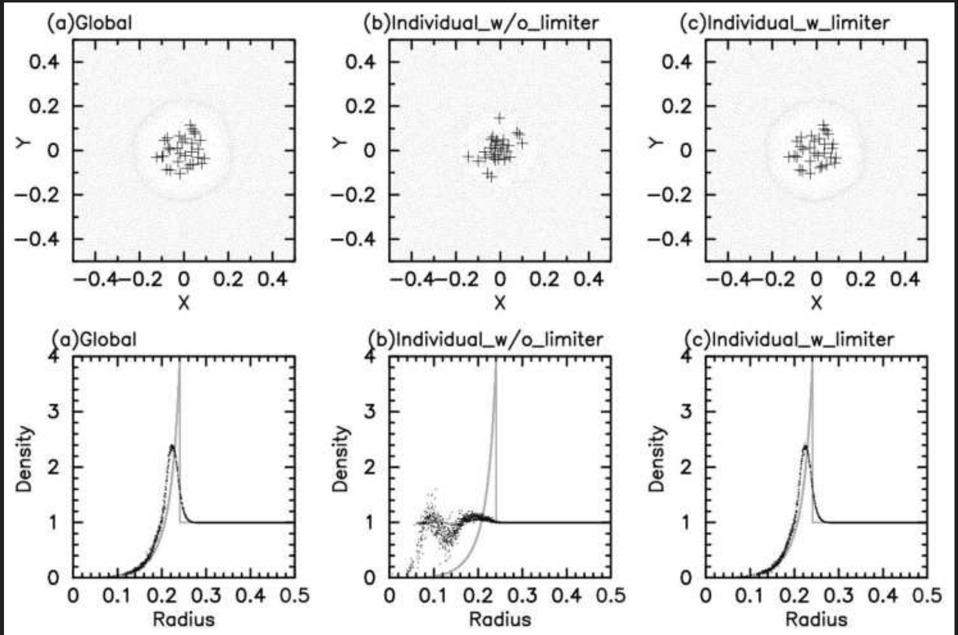
AMD MI250X: 14080 cores

?

7	8.678E-05	7285	0.48%	24.88%
8	4.339E-05	417	0.03%	36.84%

Potential Issues

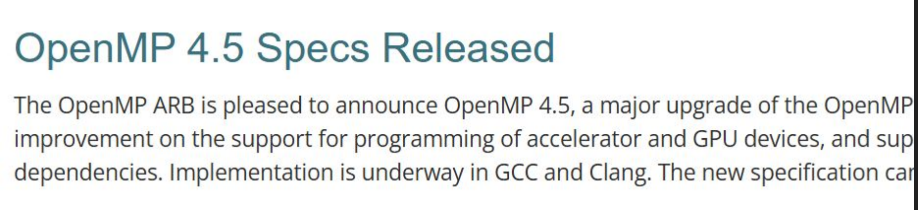
- Neighbouring particles must be within a specific number of timesteps for numerical accuracy (adjustable parameter)
- Therefore, the computational load of any particle can rapidly change.
- In Phantom, this is known as a particle “waking up” its neighbours.



From: (**Saitoh** & Makino 2009)

Next Steps

- Create a test bench single-GPU code to test variable timesteps.
- The code will use **OpenMP** for parallelism, so the result can be easily ported back to Phantom.


OpenMP 4.5 Specs Released

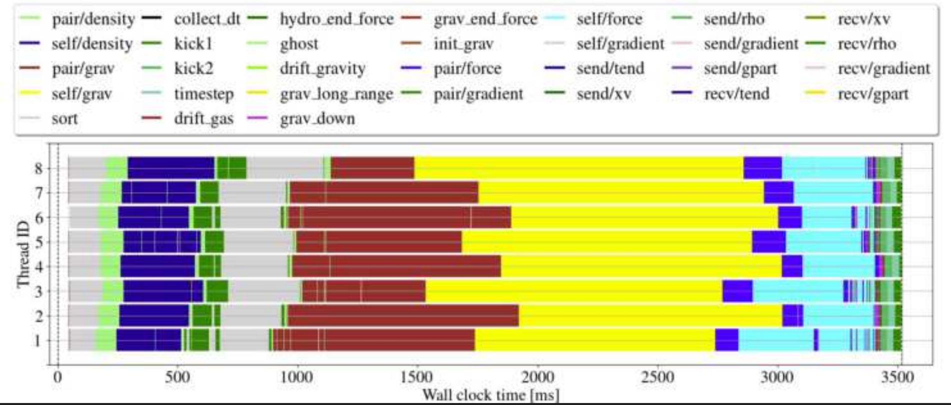
The OpenMP ARB is pleased to announce OpenMP 4.5, a major upgrade of the OpenMP
improvement on the support for programming of accelerator and GPU devices, and sup
dependencies. Implementation is underway in GCC and Clang. The new specification ca


Significantly improved support for devices. OpenMP
also runtime routines for device memory management.

Screenshots from: https://www.openmp.org/uncategorized/openmp-45-specs-released/

Bonus Challenge: Hybrid CPU + GPU Code

- The codes we have discussed only work on either the CPU **or** GPU.
- It is possible that certain types of physics could be **deferred** to the CPU on a GPU code.
- The major challenge here would be the **memory transfer overhead** between the CPU and GPU.



Task-based parallelism breakdown in Swift [7]

Thank you!