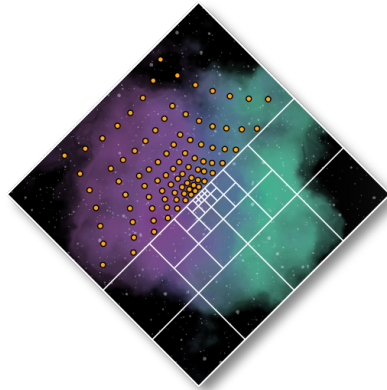




Github public repo



Shamrock



A multi-method & Multi-GPU scalable framework

Timothée David--Cléris (Postdoc IPAG)
& G. Laibe, Y. Lapeyre, L. Sewanou, B. Commerçon



No. 101053020 (Dust2Planets)



RAMSES SNO - 2025

Today's menu

What is Shamrock ?

Recap 11

3 main numerical methods in astro :

Finite elements

Finite Volumes

SPH

$\gamma(\cdot, \cdot)$

&

$\mathbb{U}^{t+1} = F\left(\{\mathbb{U}_i^t\}_{i \in \mathbb{I}}\right)$

Interaction criterion

Numerical scheme

Shamrock & features

Modern supercomputers

Modern compute nodes (LUMI)

Modern supercomputers 4

CPU-only node

4 Tflops | 0.9 TB/s | 520 W

Heterogeneous CPU/GPU node

127 Tflops | 13.4 TB/s | 2380W

NIC

South bridge

Network cards attached directly to GPUs

Much more powerful & efficient

Aurora & exascale

On the road to Exascale

CAUTION WORK IN PROGRESS 41

Up to 1024 nodes

SPH particles (3.2 · 10⁷ per GPU)

Rate (particles/s)

GPU tiles = MPI ranks

FLOPS (aurora)

CRASH !!!

Why ?

Max MPI tag = 524.287

MPI_Probe broken (mpich issue #7427)

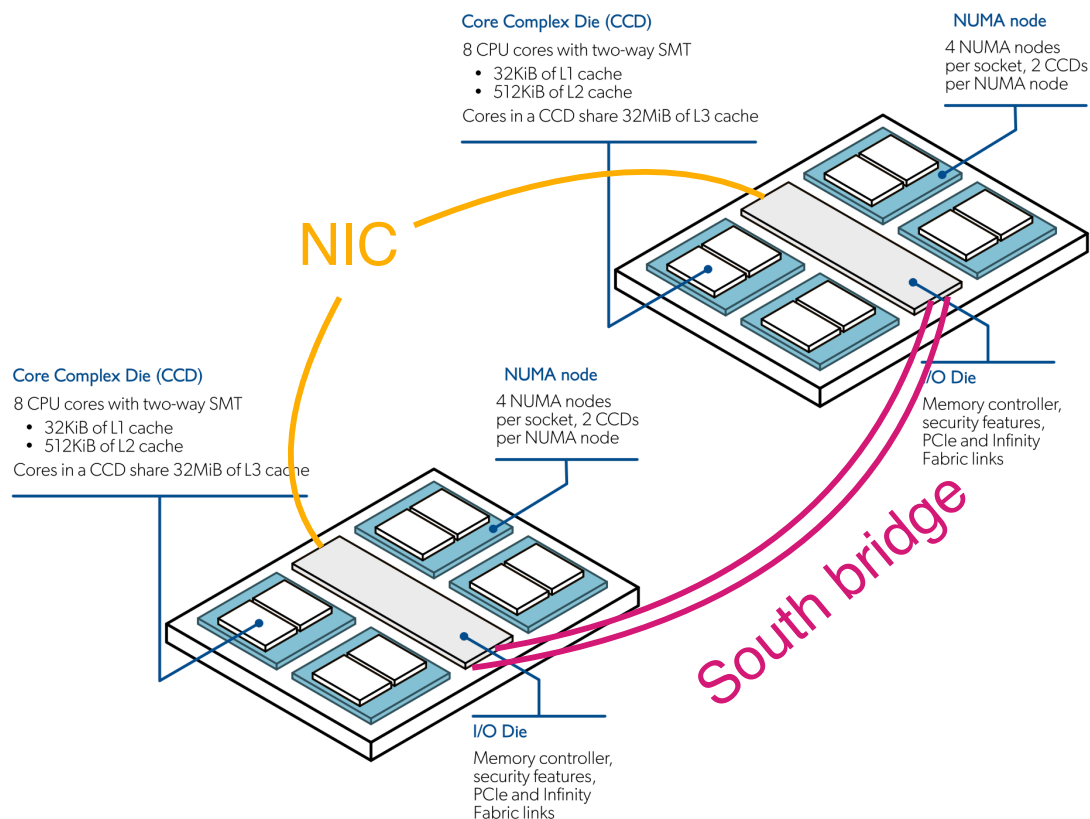
Messages lost (Libfabric overflow)

Fix: Replace probe by a all gather & MPI_Test spin lock

Modern supercomputers

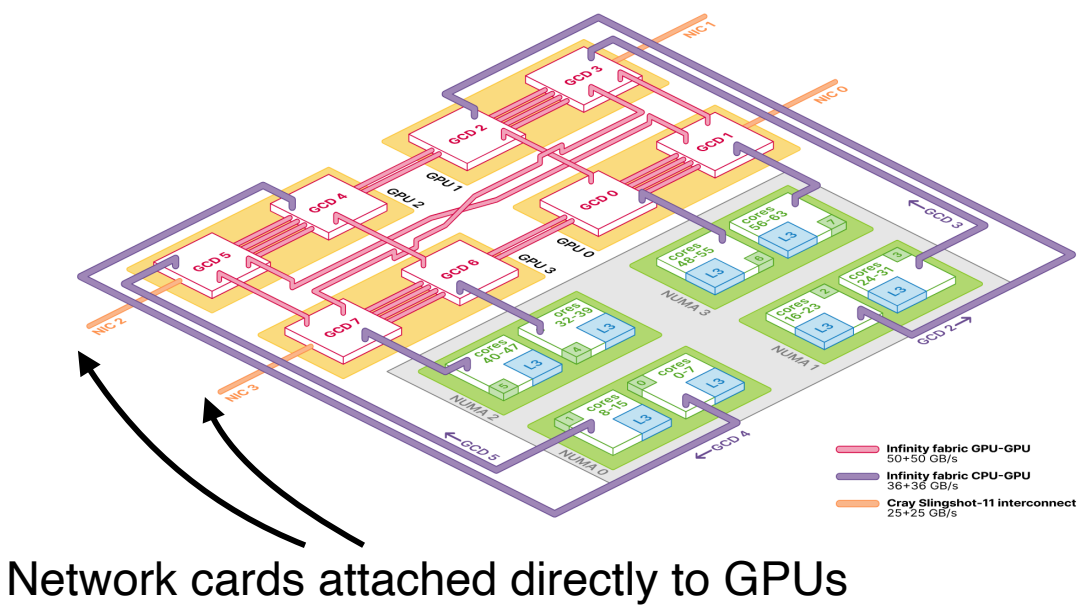
CPU-only node

4 Tflops | 0.9 TB/s | 520 W



Heterogeneous CPU/GPU node

127 Tflops | 13.4 TB/s | 2380W



Much more powerful & efficient

Images adapted from LUMI documentation



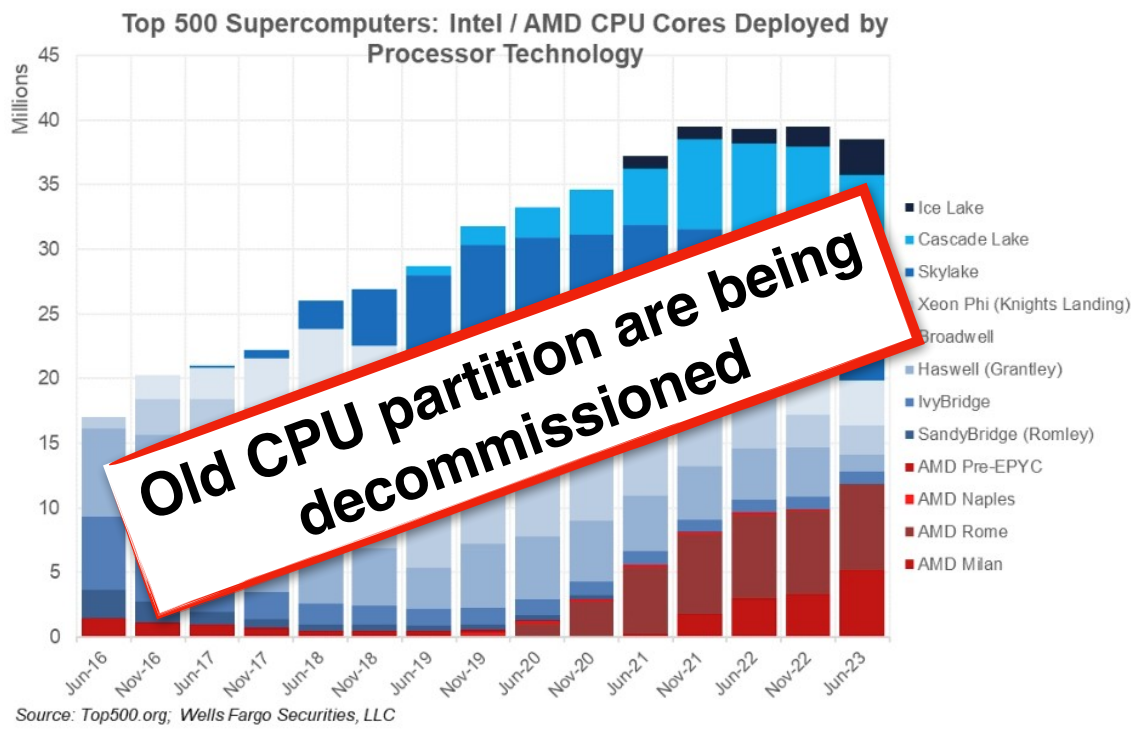
1 Exaflop
= 10^{18} float operations per second

- 10624 nodes = **127 488** GPU tiles
- Bandwidth: **2.21 PB/s** (Slingshot)
- Memory: **8.2 PB** (VRAM)
- Power consumption: **39MW**

From CPUs to GPUs

Top 500 list :

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	El Capitan - HPE Cray EX255a, AMD Instinct MI300A, Slingshot-11, TOSS, HPE United States GPU AMD	11,039,616	1,742.00	2,746.38	29,581
2	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GH, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States GPU AMD	9,060,176	1,353.00	2,055.72	24,607
3	Aurora - HPE Cray EX - Intel Exascale Compute Blades, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States GPU intel	12,640,128	1,012.00	1,980.01	38,698
4	Foghorn - Microsoft Azure, AMD Instinct MI250X, NVIDIA H100, NVIDIA InfiniBand NDR, Microsoft Azure United States GPU nvidia	2,073,600	561.20	846.84	
5	HPC6 - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GH, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy GPU AMD	3,143,520	477.90	606.97	8,461



⇒ Supercomputers move to GPUs
(Efficiency, density, AI, ...)
(But all vendors involved)

⇒ We have to move & can do more

The CPU does not perform the bulk of the compute

Could also happen with NPUs soon-ish

More Resolution

Ex: Global instabilities

Longer integration

Ex: From collapse to planets



1 Exaflop
= 10^{18} float operations per second

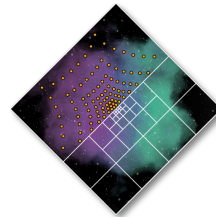
Ex: 1000s of collapses per sim

More Statistics

Ex: A collapse + full nebula around it

Larger Environment

Let's build



Shamrock

Why starting from scratch ?



Single loop approach
& bad multi-node scaling

Good scaling but
1 rank = 1 core
& Fortran



Gadget-4

Locally essential octree (too many comm.)
& CPU tree building / search

Requirements: All vendors + close to native performance

Only two choices:   **kokkos** (Both C++ based)

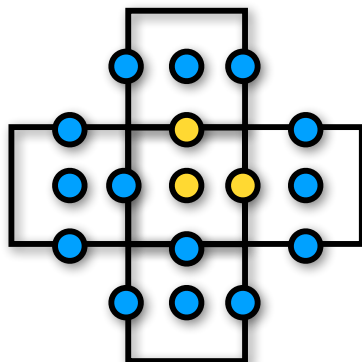
Our choice :  

- C++17 extension
- Programming **standard** (multiple implementations, not only US)
- **Directly** compiled through CUDA, Hip, OpenMP, ...
- Supports **all major** CPU & GPUs (AMD, Nvidia, Intel, ARM)
- Close to **native performance** (SYCL-bench 2020, Gromacs on LUMI)
- Being **upstreamed** in LLVM currently

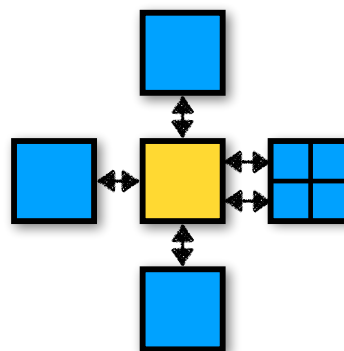
Note: Kokkos can have performance issue on intel GPUs (see Arndt, Trott 2024)

3 main numerical methods in astro :

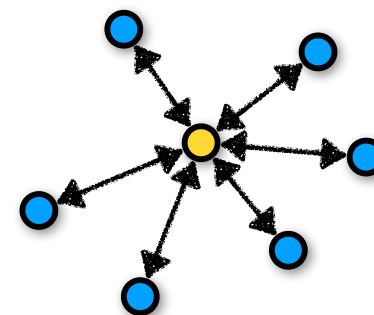
Finite elements



Finite Volumes



SPH



$$\gamma(\text{yellow circle}, \text{blue circle})$$

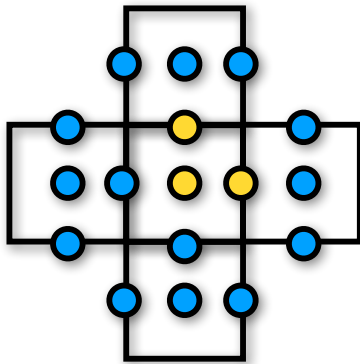
&

$$\mathbb{U}^{t+1} = F \left(\{\mathbb{U}_i^t\}_{i \in \{\text{blue circle}\}} \right)$$

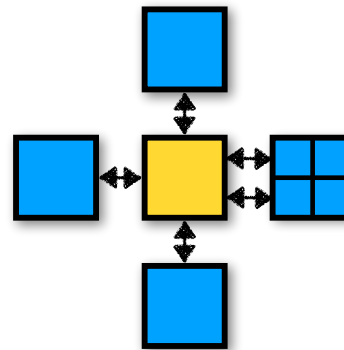
Interaction criterion

Numerical scheme

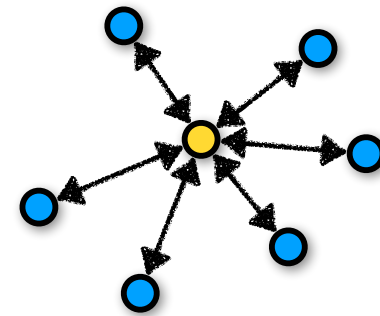
Finite elements



Finite Volumes



SPH



$$\gamma(\text{yellow circle}, \text{blue circle})$$

&

$$\mathbb{U}^{t+1} = F \left(\{\mathbb{U}_i^t\}_{i \in \{\text{blue circle}\}} \right)$$

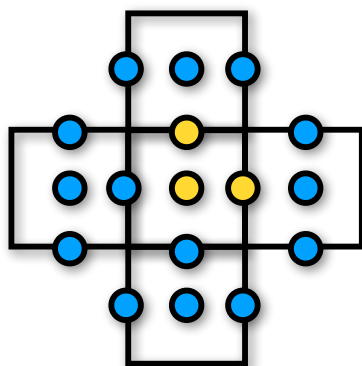
Interaction criterion

Numerical scheme

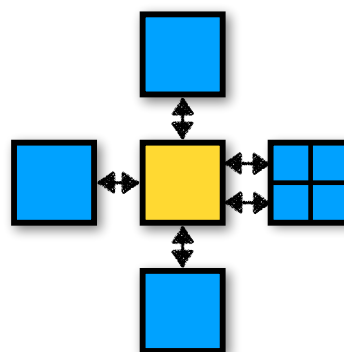
Criterion γ
Scheme F

Abstraction

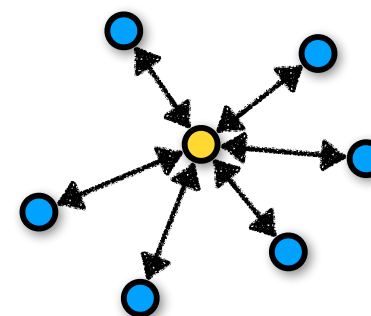
Finite elements



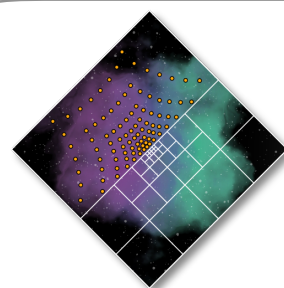
Finite Volumes



SPH



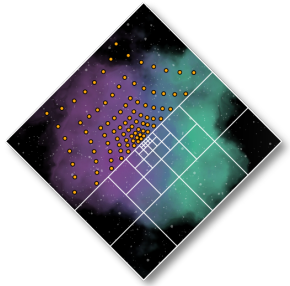
Criterion γ
Scheme F



Shamrock

Generalized sparse communications
Generalized neighbor finding
Generalized load balancing

Solver
+ Scalability

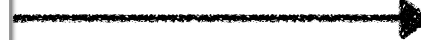


Shamrock

Generalized sparse communications

Generalized neighbor finding

Generalized load balancing



Solver
+ Scalability

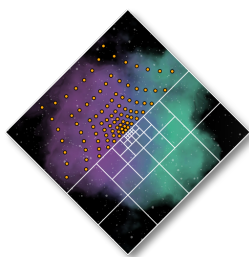
Generic modules



Optimizing SPH $\Leftrightarrow$ Optimizing AMR

Generic modules are coded **once** for all schemes

What is Shamrock ?



Shamrock

Goal:

- Multi method,
- Multi-GPU
- General framework

How:

- Abstractions
- Fast tree building/search
- Sparse communications
- Abstract LB & Domain scheduler

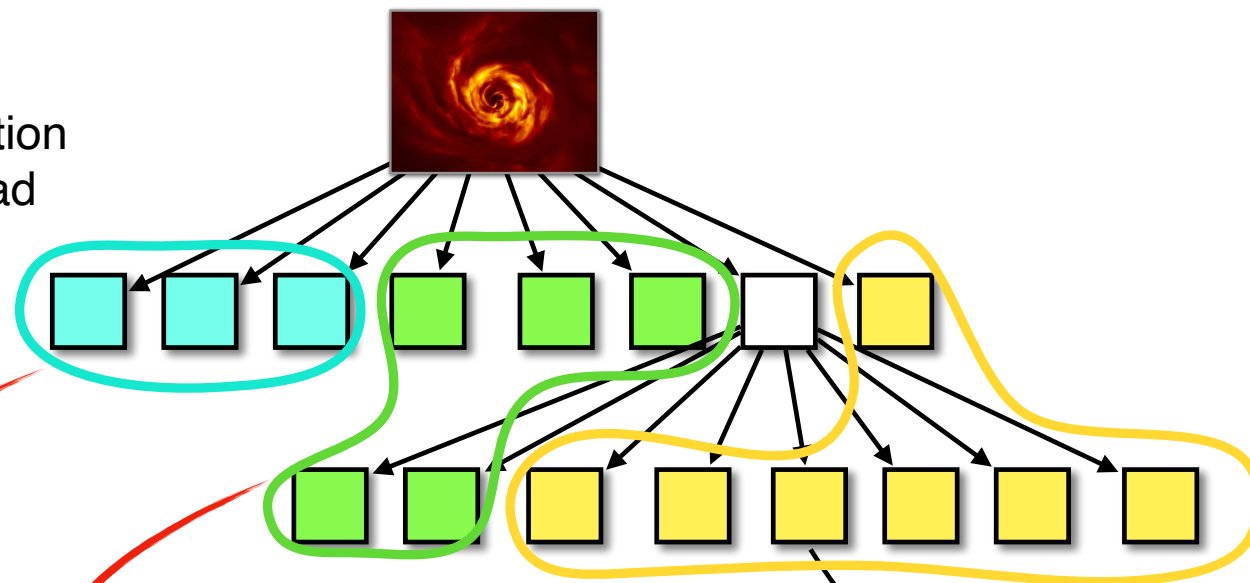


David-Clérès, Laibe, Lapeyre 2025

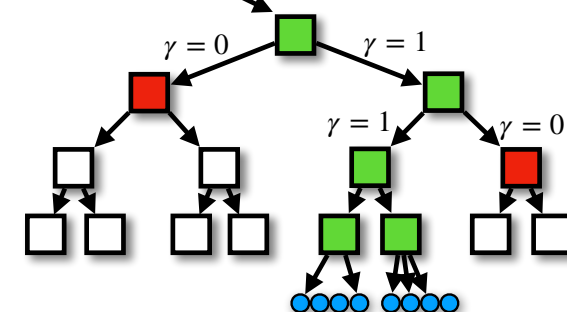
MPI Patch Octree :

Adaptive domain decomposition
according to an abstract load

Load balance
(Hilbert order)



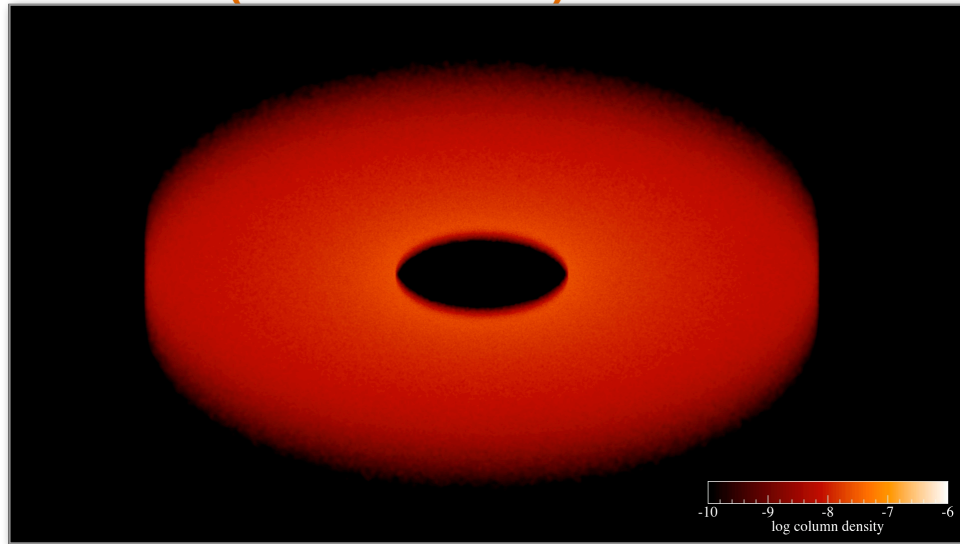
SYCL radix tree :
Compressed BVH



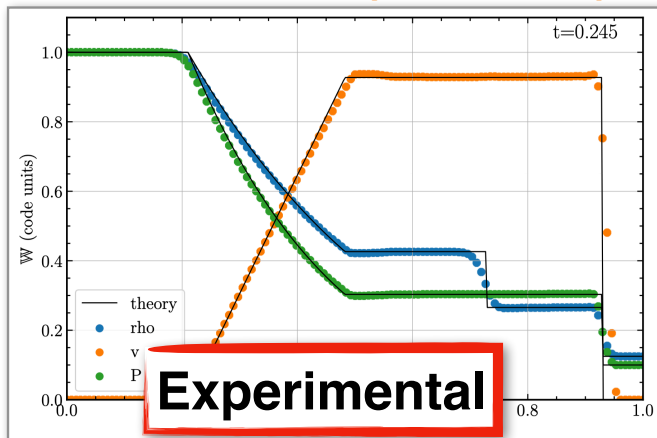
MPI

Ghost zones + metadata sync

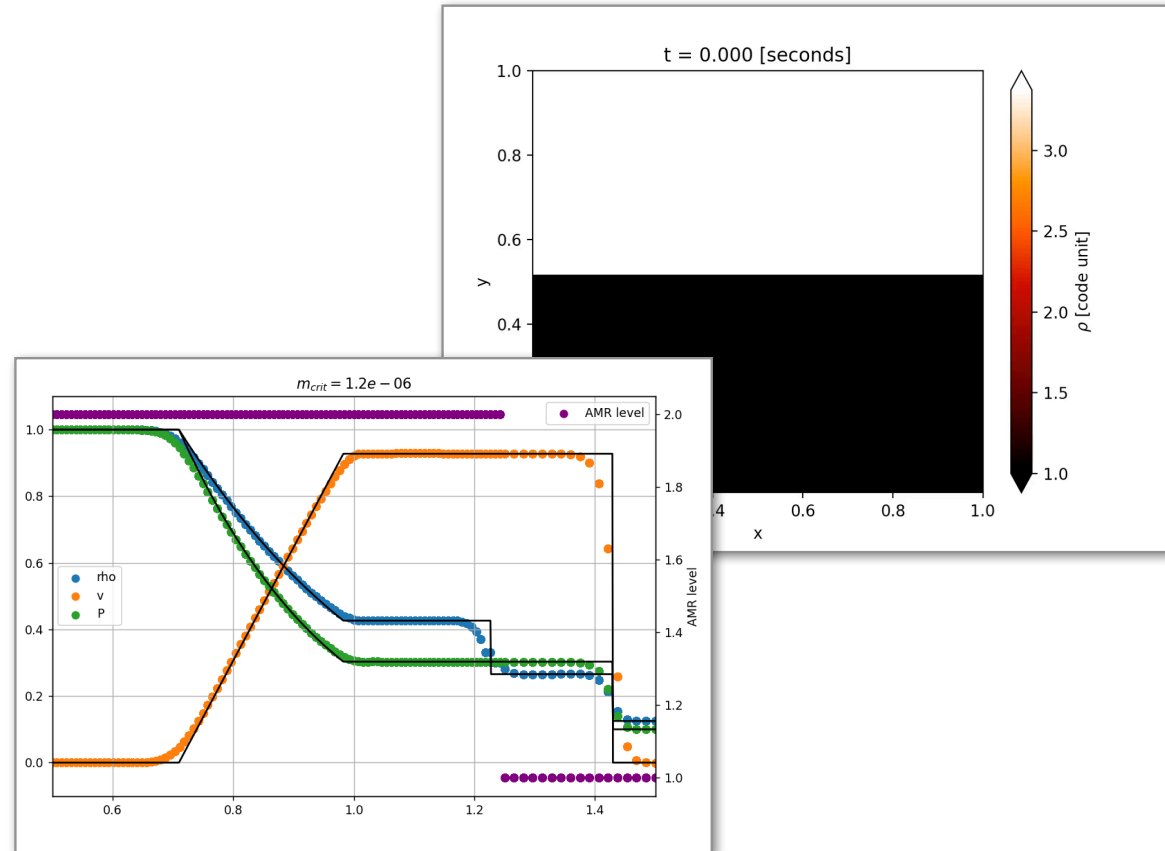
SPH (Phantom kind)



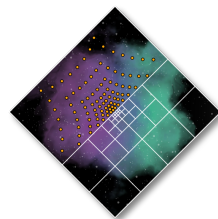
Finite elements (Zeus kind)



Finite volume (RAMSES kind)



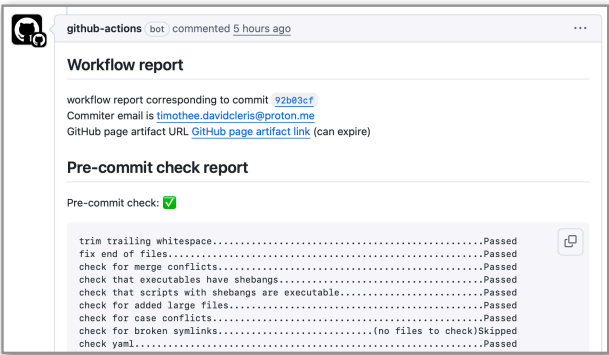
Enable multi-method studies with a single code !



Shamrock **features**



Github public repo
CECILL License



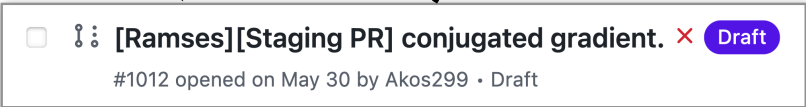
Squash PR



Update docker image

Update documentation page
(+ contributors)

Open PR/draft



CI/CD
+ review

+ Credit contributors (automatically)

```
/**
 * @file UpdateViscosity.hpp
 * @author Timothée David--Cléris (timothee.david--cleris@ens-lyon.fr)
 * @author Yona Lapeyre (yona.lapeyre@ens-lyon.fr)
```

Feature list				
Physical				
SPH model				
Core features				
Feature	Status	Contributor / Maintainer	Paper to cite	Details
Gas solver	Production ready	Timothée David-Cléris	nasa ADS Shamrock1	
Sink particles	Production ready	Timothée David-Cléris	Wait for the paper !	
Pseudo-Newtonian corrections	Production ready	Timothée David-Cléris	Wait for the paper !	PR #319
MHD solver	WIP	Yona Lapeyre	Wait for the paper !	PR #707

Feature	Status	Contributor / Maintainer	Paper to cite	Details
On the fly-plots	Production ready	Timothée David-Cléris	Wait for the paper !	PR #623
Conformance with Phantom	Production ready	Timothée David-Cléris & Yona Lapeyre	nasa ADS Shamrock1	
Saturatn graph	Production ready	Timothée David-Cléris	Wait for the paper !	PR #503

- **Interoperable with Python**
(Simple user interface)

```
[1]: import shamrock

[2]: shamrock.change_loglevel(1)

-> modified loglevel to 0 enabled log types :
log status :
- Loglevel: 1, enabled log types :
[xxx] Info: xxx ( logger::info )
[xxx] : xxx ( logger::normal )
[xxx] Warning: xxx ( logger::warn )
[xxx] Error: xxx ( logger::err )

[4]: shamrock.sys.init("0:0")

[5]: shamrock.sys.world_size()
```

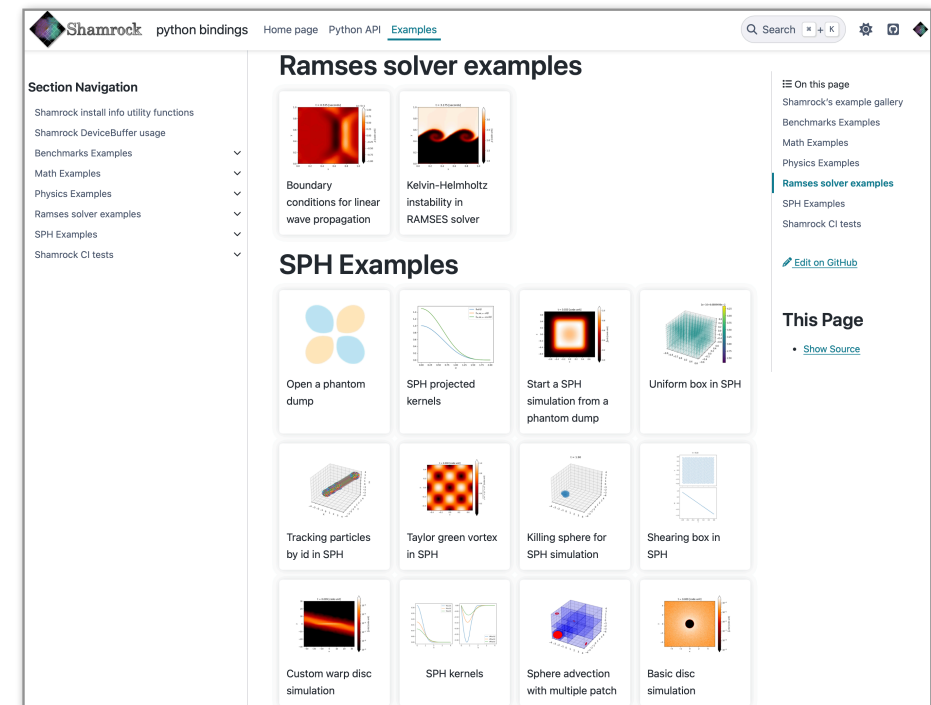
```
[5]: 1

[1]: import shamrock

[ ]: shamrock.get_M

f get_Model_Ramdes function
f get_Model_SPH function
f get_Model_Zeus function
```

Documentation:



Matplotlib inspired doc, [Link](#)

No `#ifdef(s)` “There is only one Shamrock”
(physical) (All solvers & physics in a single binary)

brew install shamrock

Shamrock python package
& shamrock executable

Packaging status

Homebrew 2025.05.0

Spack 2025.05.0

Shamrock packages !!!

⇒ **All solvers accessible in a single shamrock distribution**

```
[1]: import shamrock
```

```
[ ]: shamrock.get_M
```

f	get_Model_Ramses	function
f	get_Model_SPH	function
f	get_Model_Zeus	function

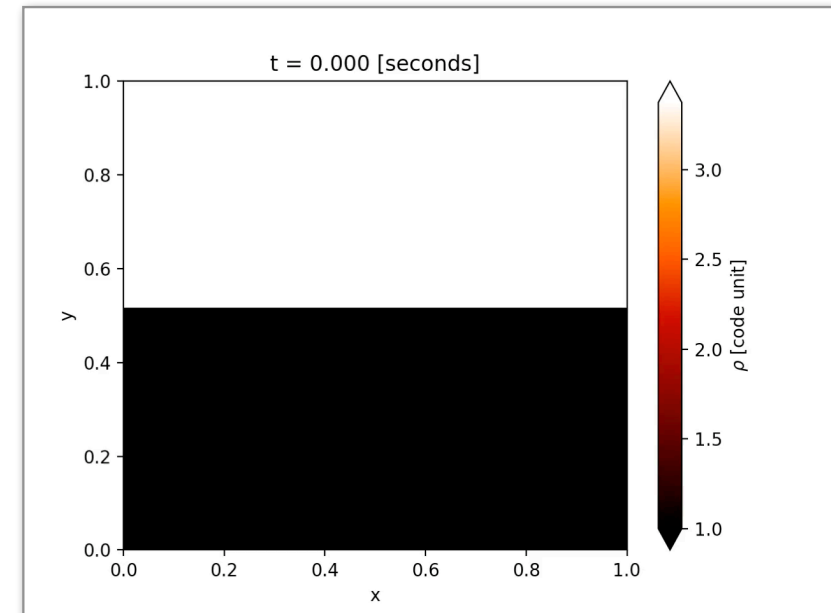
On the fly/in situ analysis:

```
tmax = 0.127 * fact
all_t = np.linspace(0, tmax, fact)

def plot(t, iplot):
    metadata = {"extent": [0, 1, 0, 1], "time": t}
    arr_rho_pos = model.render_slice("rho", "f64", positions)
    plot_rho_slice_cartesian(metadata, arr_rho_pos, iplot, case_name)

current_time = 0.0
for i, t in enumerate(all_t):
    model.dump_vtk(os.path.join(sim_folder, f"{case_name}_{i:04d}.vtk"))
    model.evolve_until(t)
    current_time = t
    plot(current_time, i)

plot(current_time, len(all_t))
```



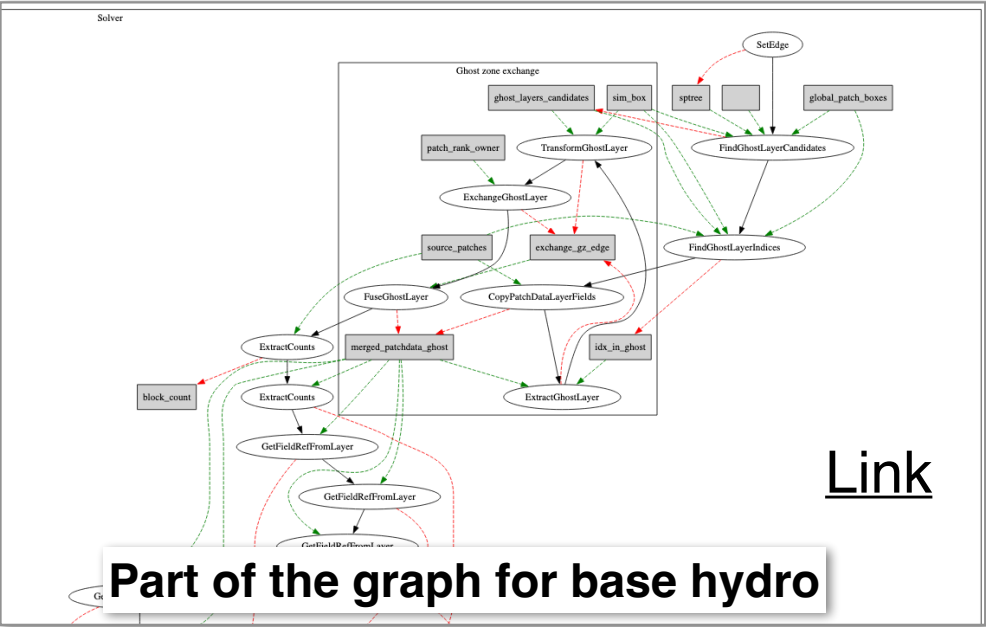
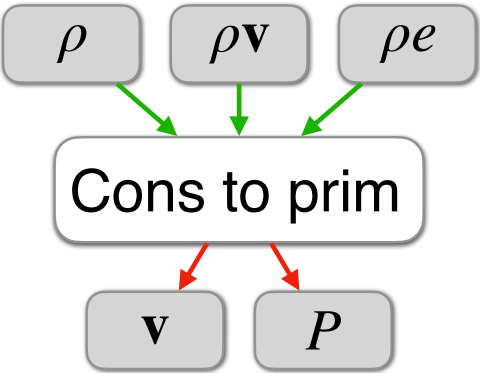
Best way to analyse exascale simulations (>10Tb per dump)

No need to dump outside of restarts

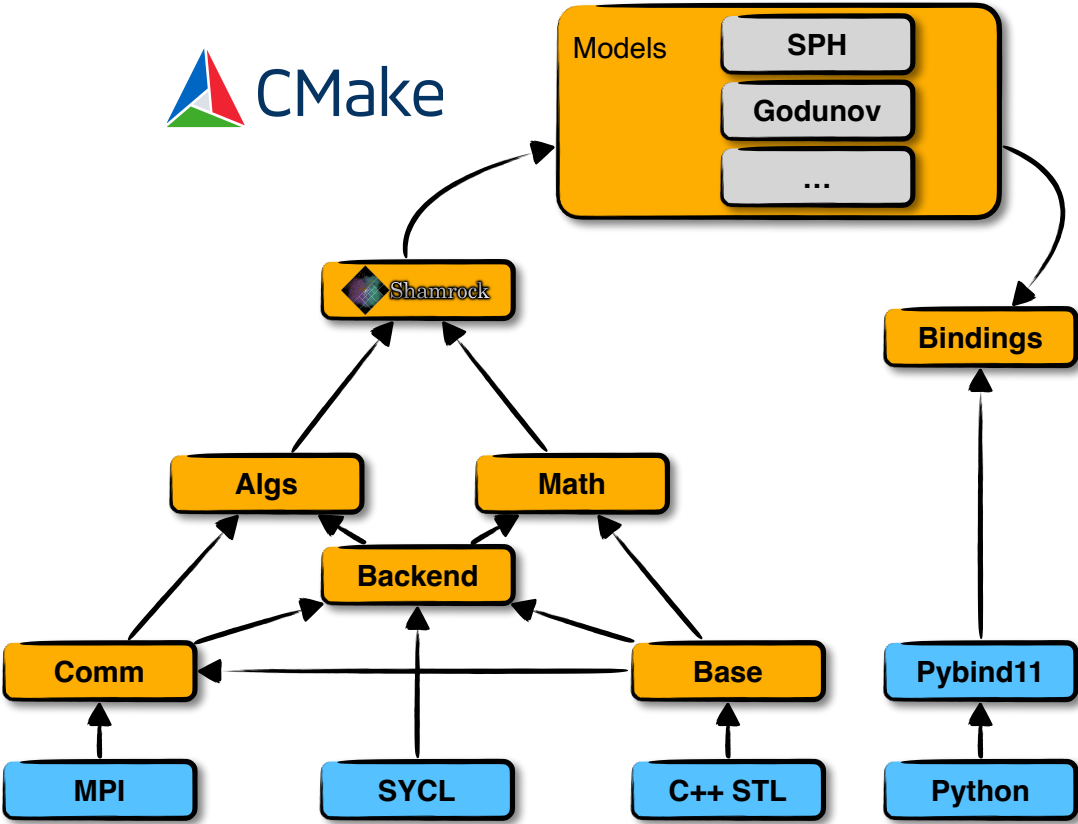
Modularity

Solvergraph:

Generalized modules with abstracted inputs & outputs



Multiple internal libraries:





PODCAST (PI: G. Laibe)

Dust2Planets (PI: F. Ménard)

PEPR Origins (PI: A. Morbidelli)



⇒ 1 of the 3 Exascale codes identified in the project

EuroHPC CoE Space project



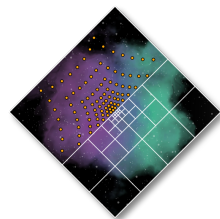
⇒ port RAMSES AMR solver in the Shamrock framework



⇒ Investigate/optimize Shamrock
Exascale scalability
& optimize aurora MPI tuning

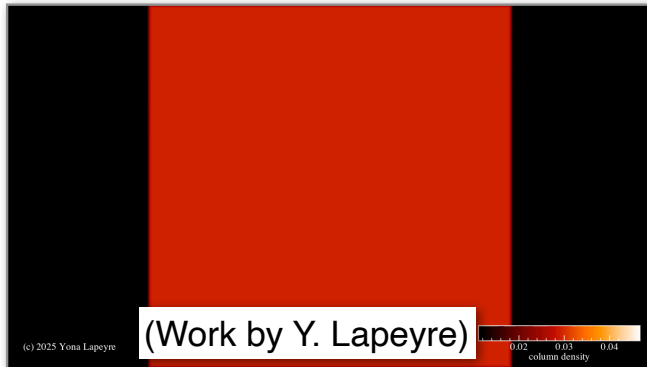
AIRBUS

⇒ Using Shamrock for aerodynamic design optimisation

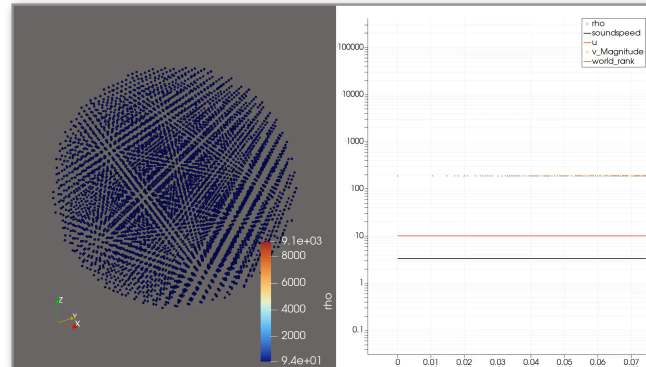


Shamrock **Solvers**

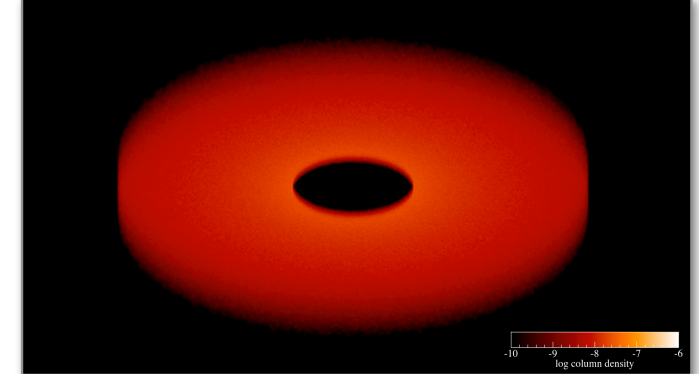
IMHD (div. cleaning)



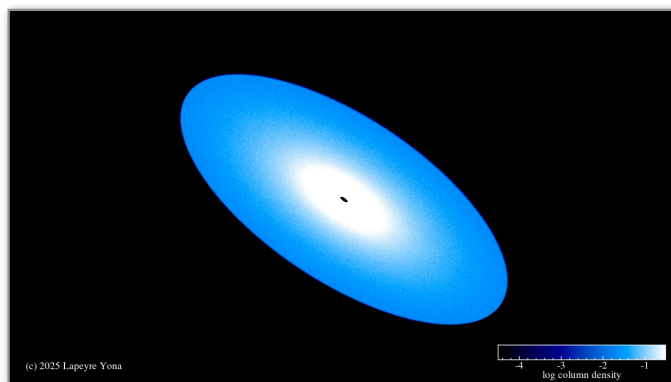
Self-gravity



Sink particles

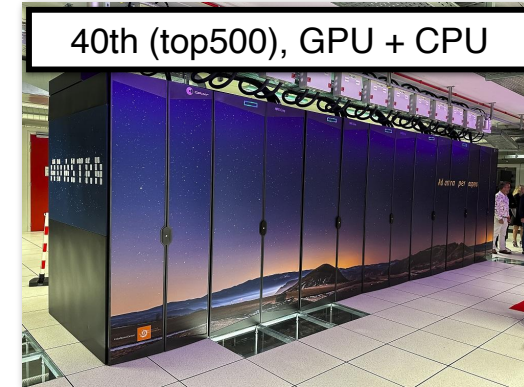
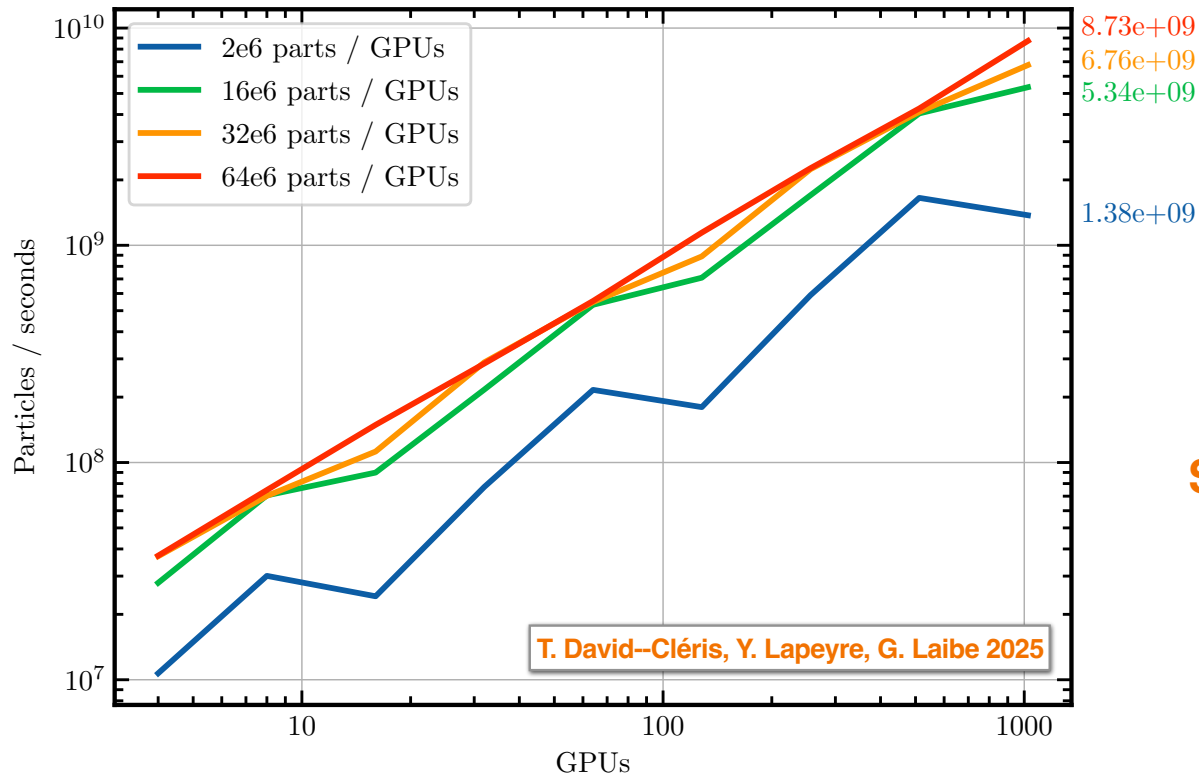


PN corrections



- **Time integration:** Global leapfrog, sub stepping (WIP)
- **Artif. viscosity:** MM97, CD10, α -like
- **MHD:** Ideal (div.cleaning)
- **Dust:** (WIP one-fluid)
- **Self-gravity:** MM, FMM, SFMM, Direct (single GPU)
- **Sink particles**
- **Shearing box**
-

■ : work by Y.Lapeyre



Shamrock :

- 65G particles (4000^3)
- 9G part / seconds
- 7 sec / iterations
- 1024 GPU (MI250x)
- 92% parallel efficiency

Reference :

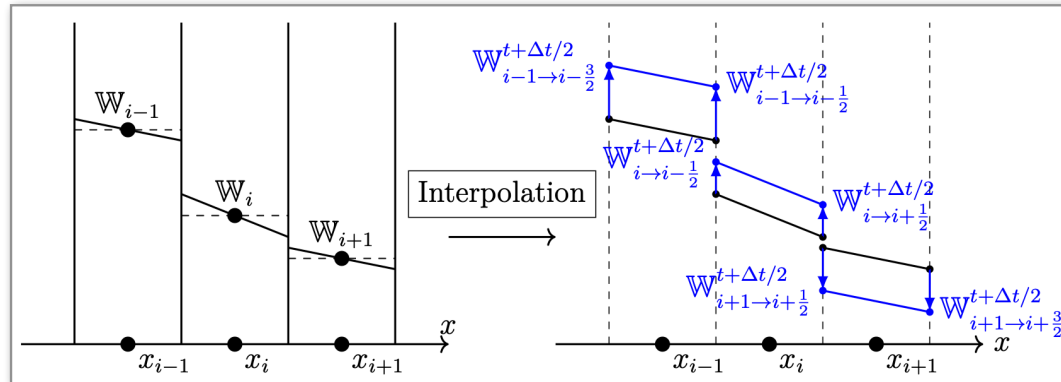
Same test with Phantom SPH = 2e6 part / seconds

→ **x4500 speedup !**

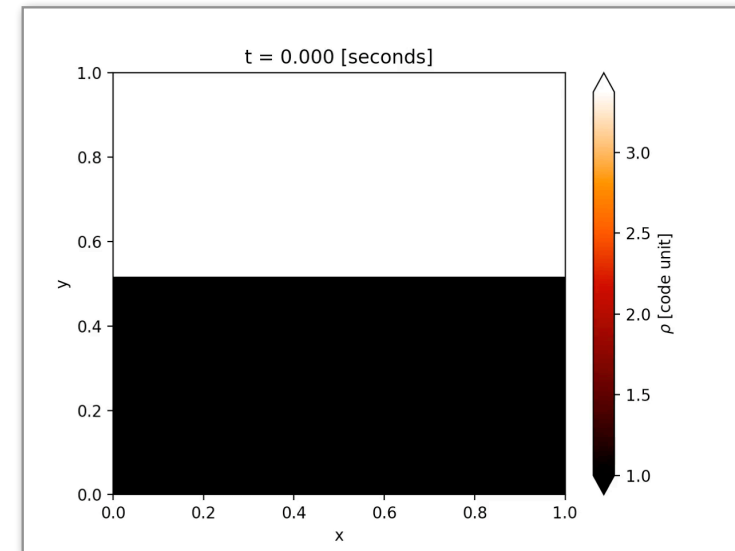
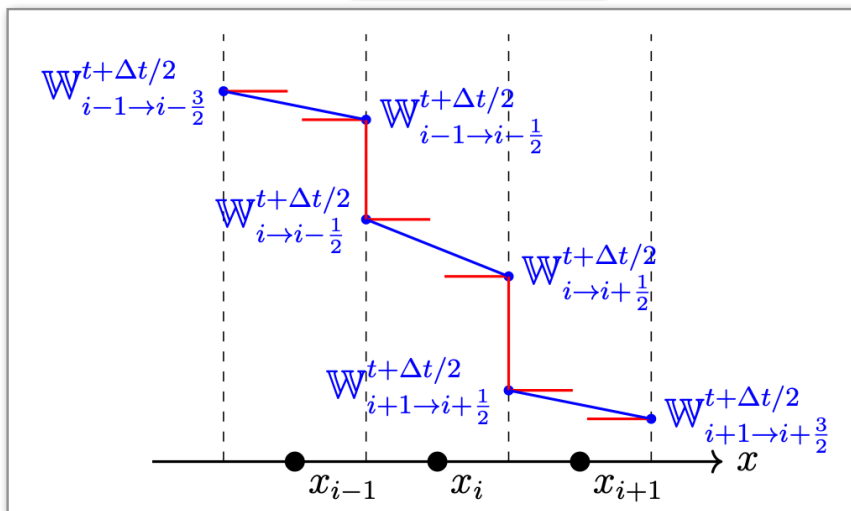


1G particles





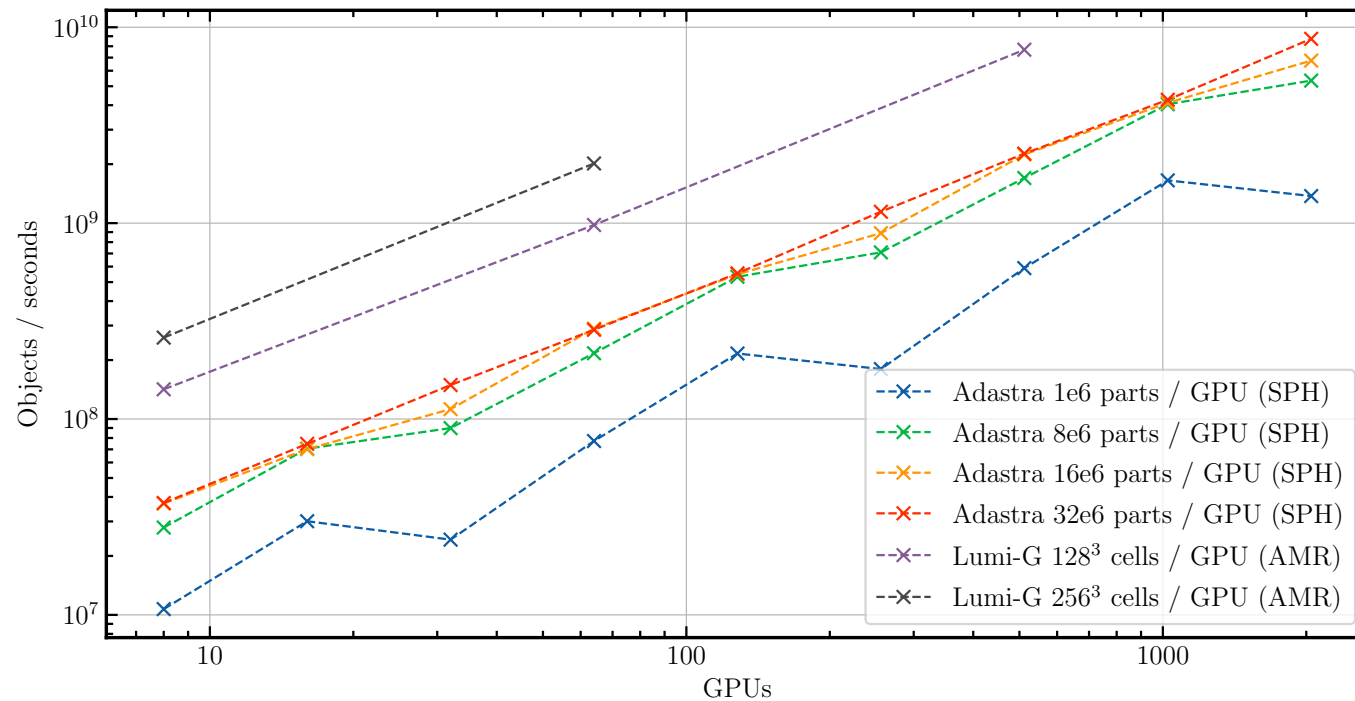
PHD T.David-Cl  ris



- **Time integration:** Global Euler /w face time interp (2nd order)
- **Riemann solvers:** Rusanov, HLL, HLLC I dust: HB, DHLL
- **Slope limiter:** Minmod, Van Leer, Van Leer (sym.)
- **Dust:** IRK1, Exp drag
- **Self-gravity:** CG multi-GPU
- **Refinement:** Mass based, pseudo-gradient (WIP)

■ : work by T. Guillet

■ : work by L. Sewanou



84% parallel efficiency on LUMI



Latest results:
(alloc pressure reduction)

```
[amr::RAMSES][rank=0]
```

rank	rate (N.s ⁻¹)	Nobj	t compute (s)	interf	alloc	mem (max)
0	1.0673e+08	16777216	1.572e-01	4 %	21 %	23.84 GB

On the road to exascale

On the road to Exascale

33



Top 500 list :

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	El Capitan - HPE Cray EX255a, AMD 4th Gen EPYC 24C 1.8GHz, AMD Instinct MI300A, Slingshot-11, TOSS, HPE DOE/NNSA/LLNL United States	11,039,616	1,742.00	2,746.38	29,581
2	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE Cray OS, HPE DOE/SC/Oak Ridge National Laboratory United States	9,066,176	1,353.00	2,055.72	24,607
3	Aurora - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, Intel DOE/SC/Argonne National Laboratory United States	9,264,128	1,012.00	1,980.01	38,698
4	Eagle - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, Microsoft Azure Microsoft Azure United States	2,073,600	561.20	846.84	
5	HPC6 - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, RHEL 8.9, HPE Eni S.p.A. Italy	3,143,520	477.90	606.97	8,461

We have access to this one



120 000 GPUs

1 Exaflop

T. David--Cl  ris, T. Applencourt, M. Wilkins (in prep.)

On the road to Exascale

34



Before going on, a few words ...

This is with SPH, but the RAMSES solver use the same communication pipeline and scheduler.
⇒ **Same comm overhead & Load balancing**

This is only on Aurora, but almost everything mentioned is related to MPIch & libfabric which is also on the other exascale supercomputers.
⇒ **Should be also true on Frontier/EI Captain**



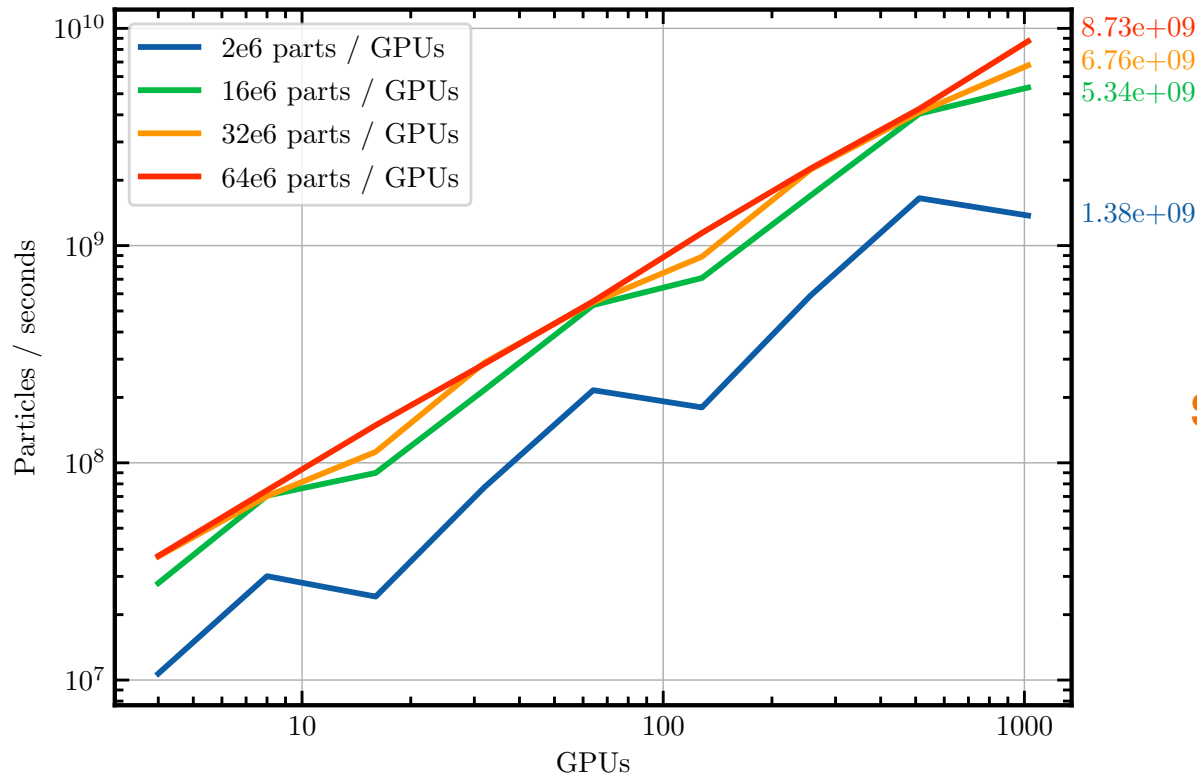
While running on pre-exascale is $\simeq$ easier exascale is HARD !
The following took 6 months

On the road to Exascale

CAUTION
WORK
IN PROGRESS

Same test as on Adastral

Sedov-Taylor blast weak scaling



Shamrock :

- 65.000.000.000 particles (4000^3)
- 9G part / seconds
- 7 sec / iterations
- 2048 GPU tiles (MI250x)



Step 1: Compile and run

```
module use /soft/modulefiles
module load cmake
module load python
module load ninja

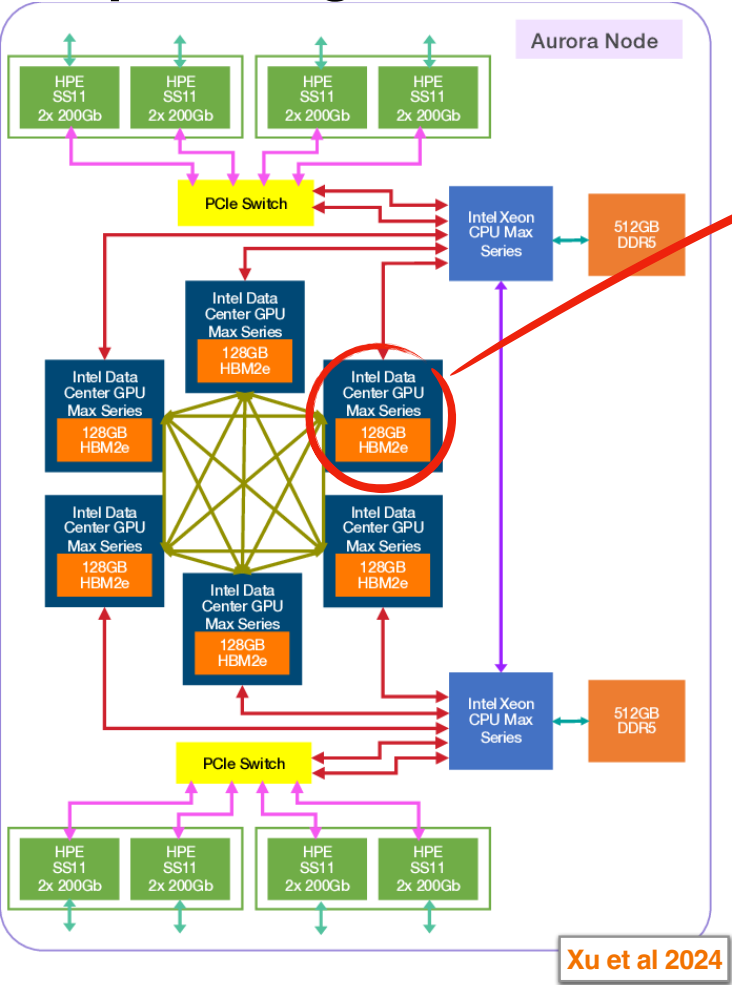
function shamconfigure {
    cmake \
        -S $SHAMROCK_DIR \
        -B $BUILD_DIR \
        -DSHAMROCK_ENABLE_BACKEND=SYCL \
        -DSYCL_IMPLEMENTATION=IntelLLVM \
        -DINTEL_LLVM_PATH=$(dirname $(which icpx))/.. \
        -DCMAKE_CXX_COMPILER=$(which icpx) \
        -DCMAKE_C_COMPILER=$(which icx) \
        -DCMAKE_CXX_FLAGS="-fsycl -fp-model=precise" \
        -DCMAKE_EXE_LINKER_FLAGS="-Wl,--copy-dt-needed-entries" \
        -DCMAKE_BUILD_TYPE="${SHAMROCK_BUILD_TYPE}" \
        -DBUILD_TEST=Yes \
        "${CMAKE_OPT[@]}"
}

function shammake {
    (cd $BUILD_DIR && $MAKE_EXEC "${MAKE_OPT[@]}" "${@}")
}
```

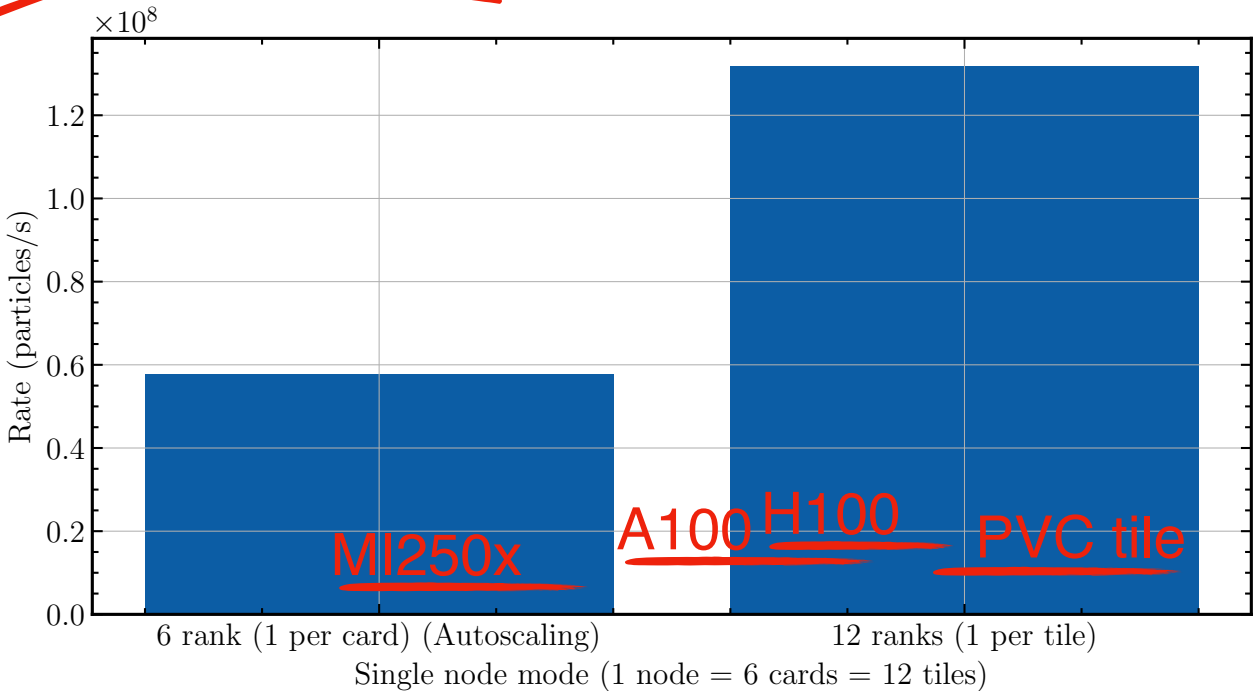
Thanks to SYCL almost no changes !



Step 2: Single node

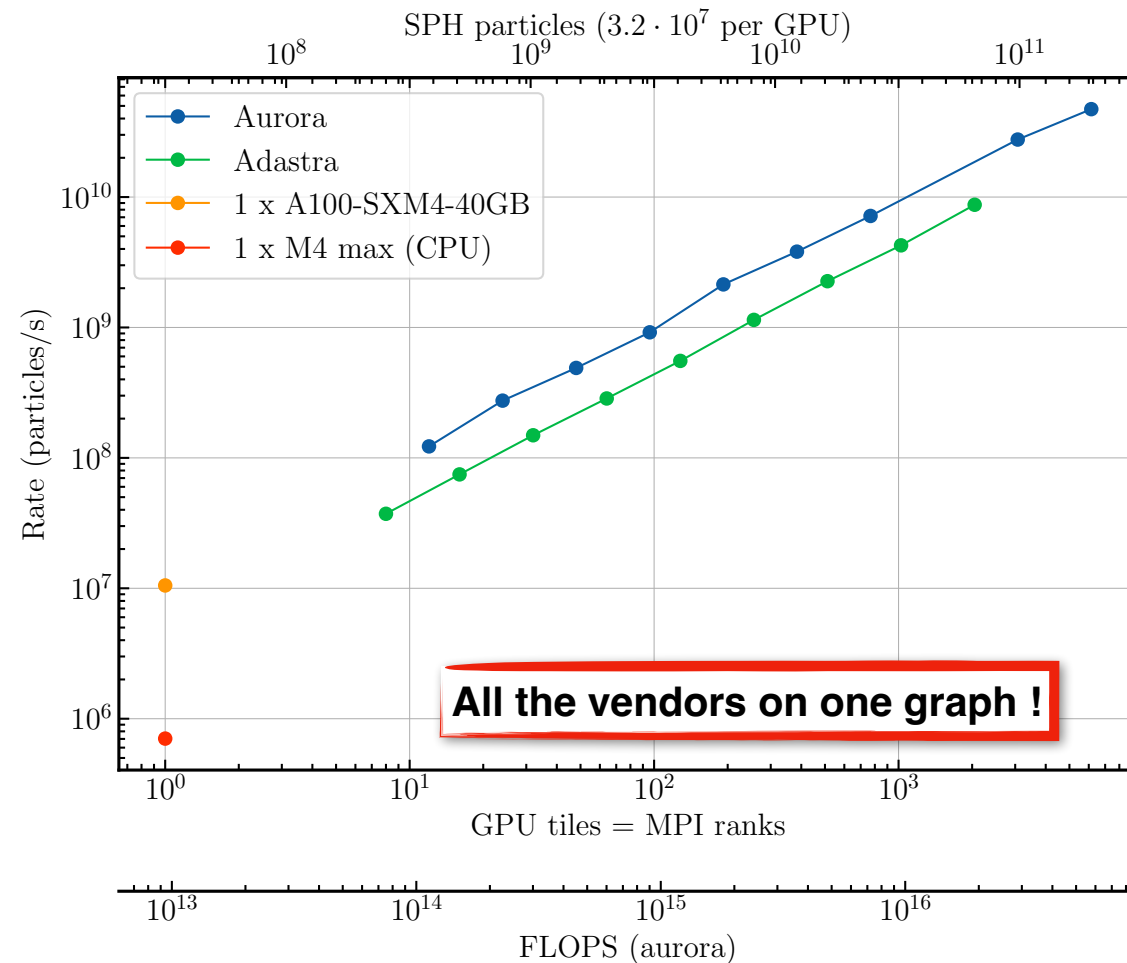


1 PVC GPU = 2 tiles



$\Rightarrow$ 12 ranks per node $\simeq$ 9 A100 (in perf.)

Up to 512 nodes



List of fixes:

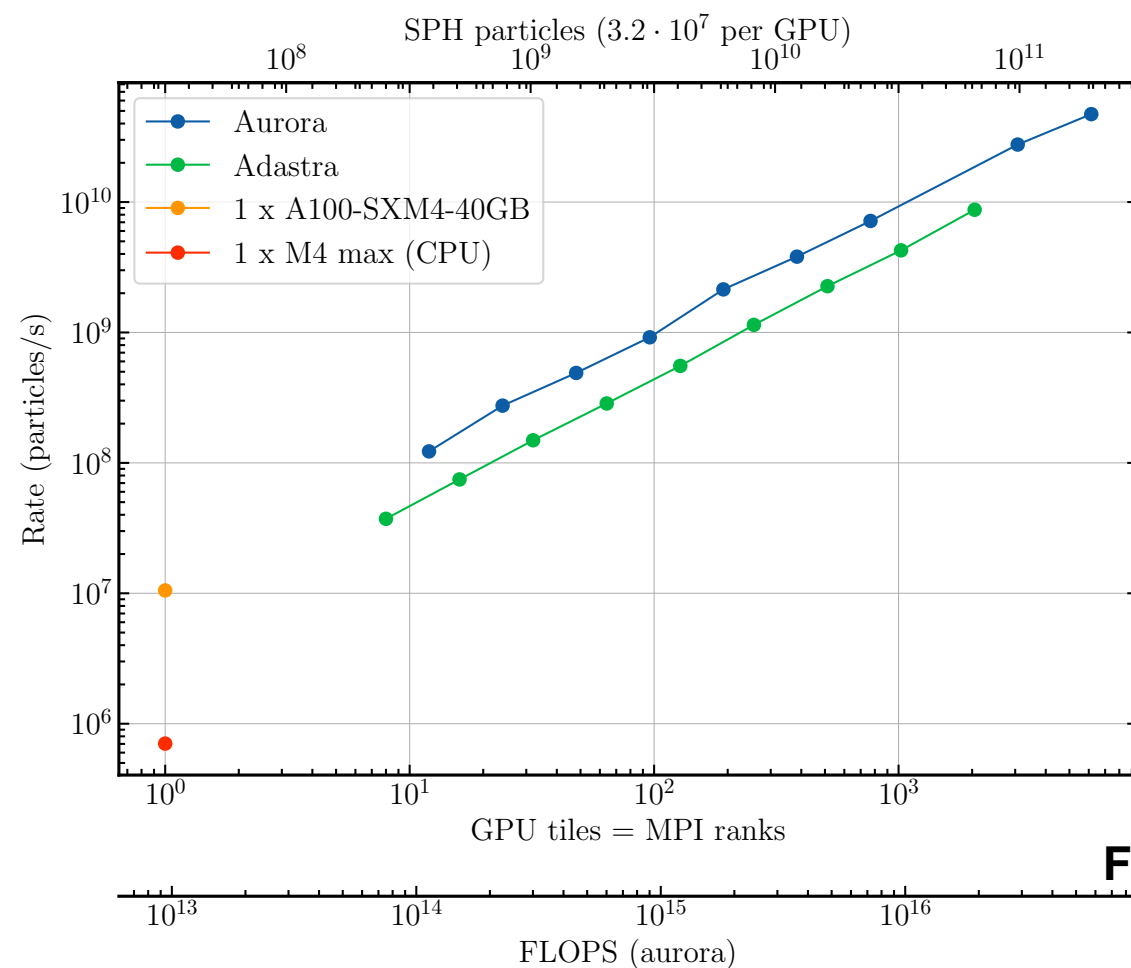
- ⚠ Disable direct GPU comm because of mpich memory leak

Shamrock :

- 196.845.690.624 particles
- 47G part / seconds
- 3 sec / iterations
- 6144 GPU tiles
- 196 TB of VRAM



Up to 1024 nodes



That's a Sedov blast btw ...

CRASH !!!

Why ?

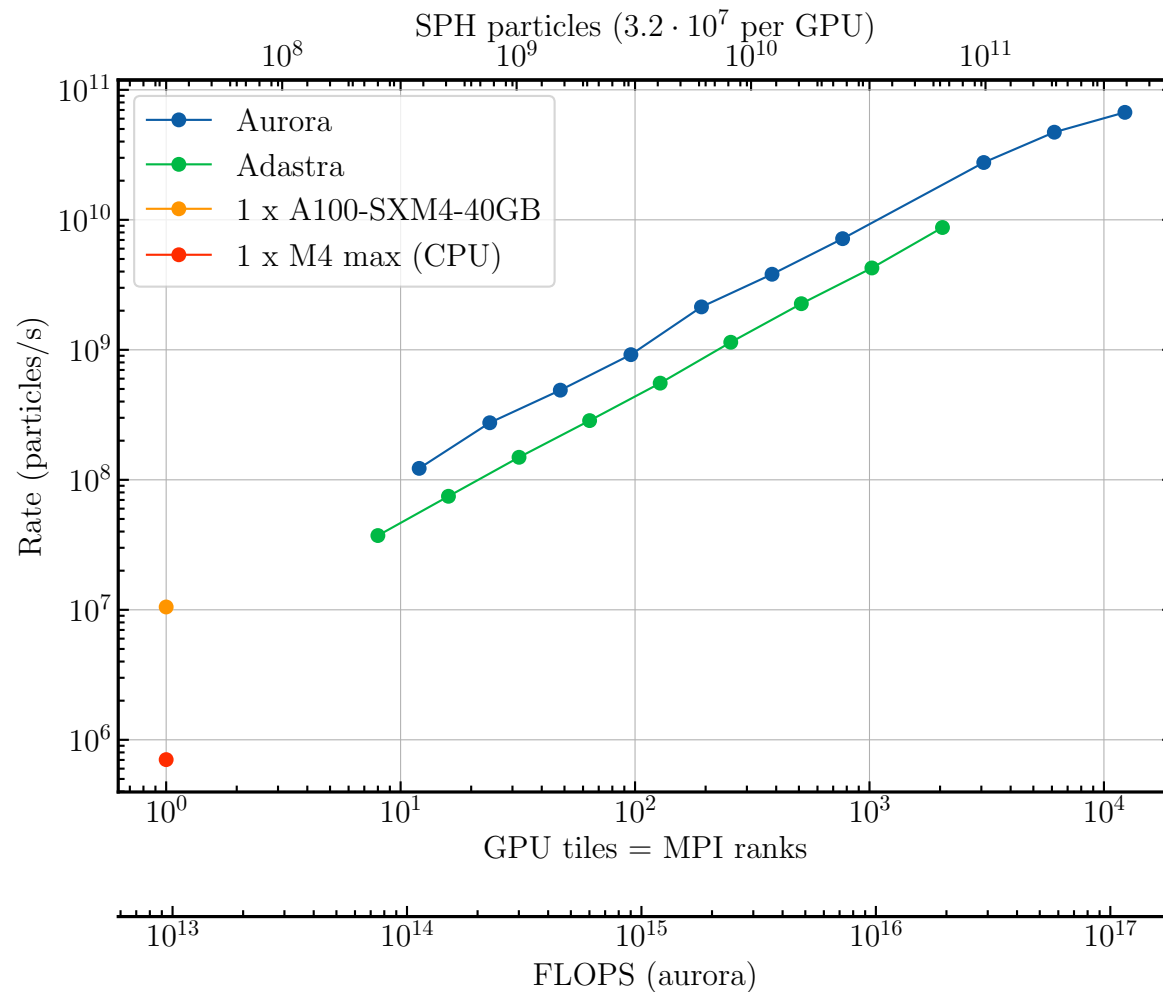
Max MPI tag = 524.287

MPI_Probe broken (mpich issue #7427)

Messages lost (Libfabric overflow)

Fix: Replace probe by a all gather & MPI_Test spin lock

Up to 1024 nodes



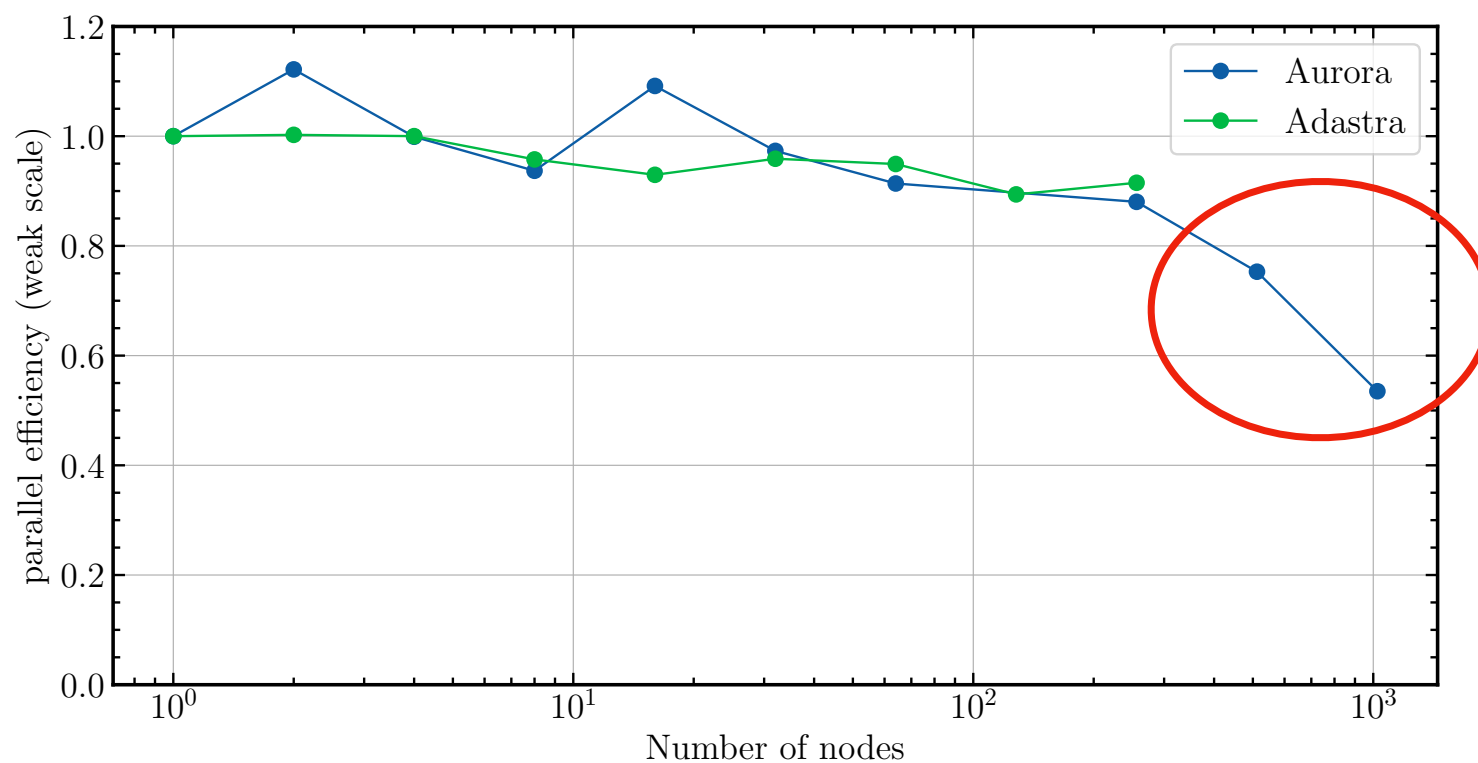
List of fixes:

- ⚠ Disable direct GPU comm because of **mpich** memory leak
- Comm spin lock
- Hybrid match mode

Shamrock :

- **393.692.655.200 particles**
- **67G part / seconds**
- **3 sec / iterations**
- **12288 GPU tiles**
- **393 TB of VRAM**

Up to 1024 nodes



Why the slowdown ?

MPI_Allgather

MPI allgather algorithms

Collab with M. Wilkins
Using ACCLAIM



Ring exchange

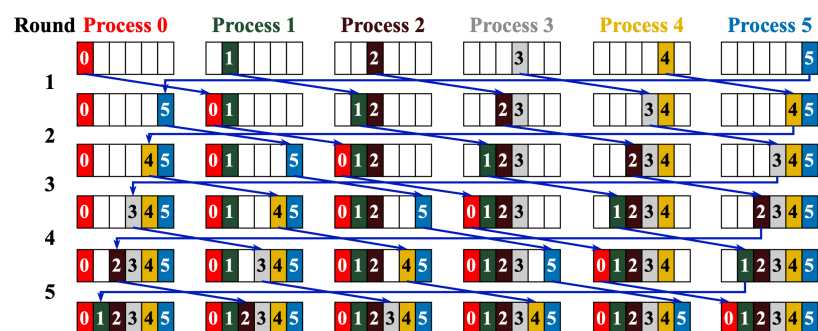
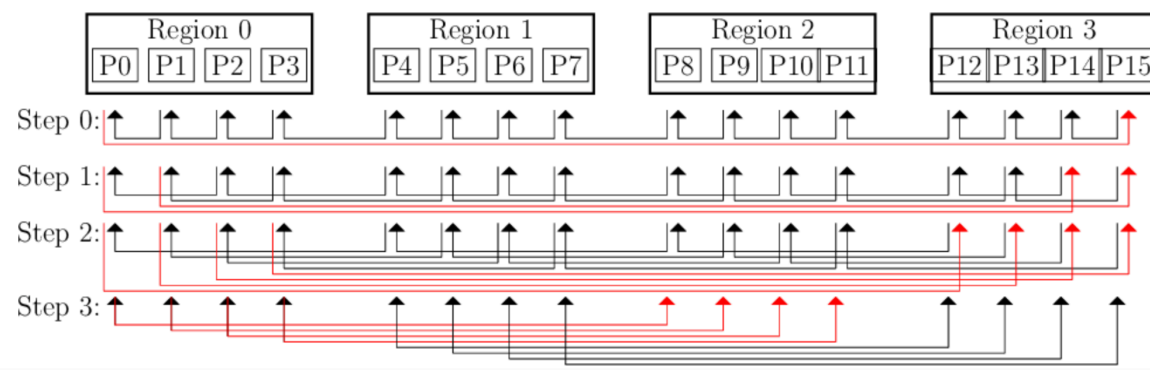


Fig. 5: Ring algorithm for allgather. In each round, processes forward data to their cyclic adjacent neighbors.

Best bandwidth

$$\text{Latency} = \mathcal{O}(N_{\text{ranks}})$$

Bruck's



Lower bandwidth

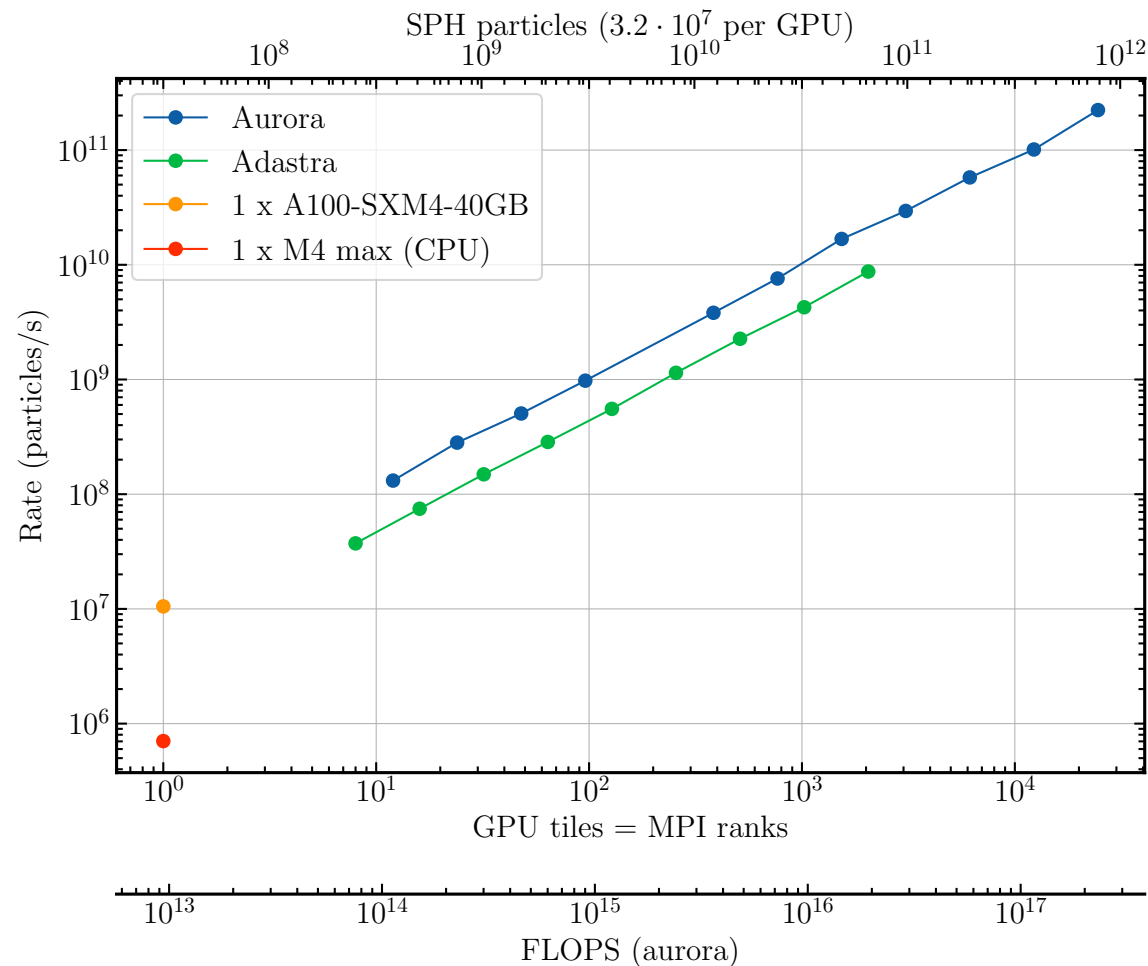
$$\text{Latency} = \mathcal{O}(\ln N_{\text{ranks}})$$

```
export MPIR_CVAR_ALLGATHER_INTRA_ALGORITHM=brucks
export MPIR_CVAR_ALLGATHERV_INTRA_ALGORITHM=brucks
```

Shamrock MPI tuning is now default on Aurora !!!

Up to 2048 nodes

T. David-Cl  ris, T. Applencourt, M. Wilkins (in prep.)



List of fixes:

- ⚠ Disable direct GPU comm because of **mpich** memory leak
- Comm spin lock
- Brucks allgather
- Hybrid match mode
- Work duplication in vector all gather

Shamrock :

- 787.087.140.552 particles 🤖
- 223G part / seconds
- 3 sec / iterations
- 24576 GPU tiles
- 786 TB of VRAM

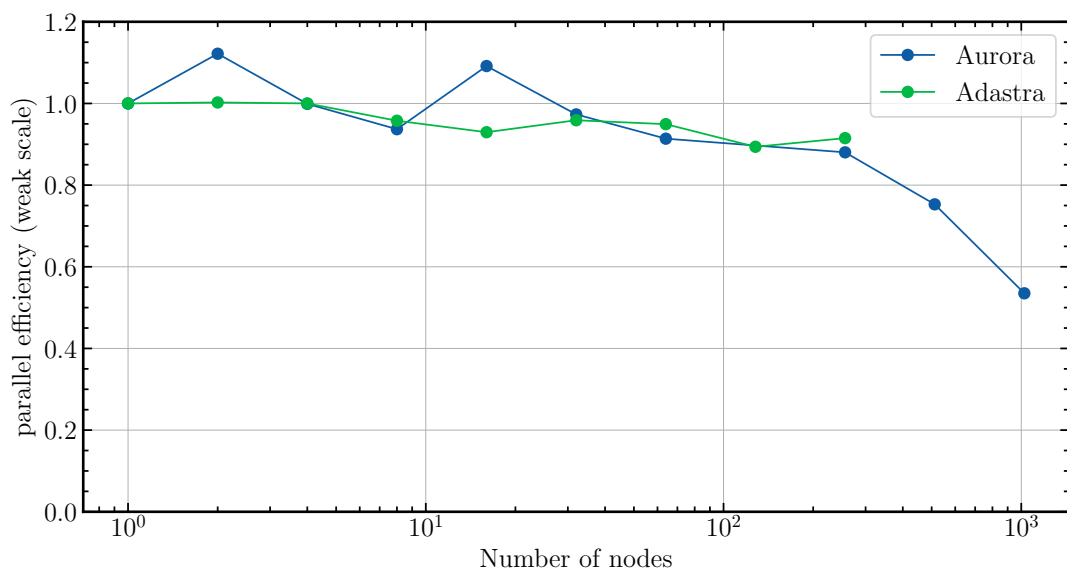
On the road to Exascale



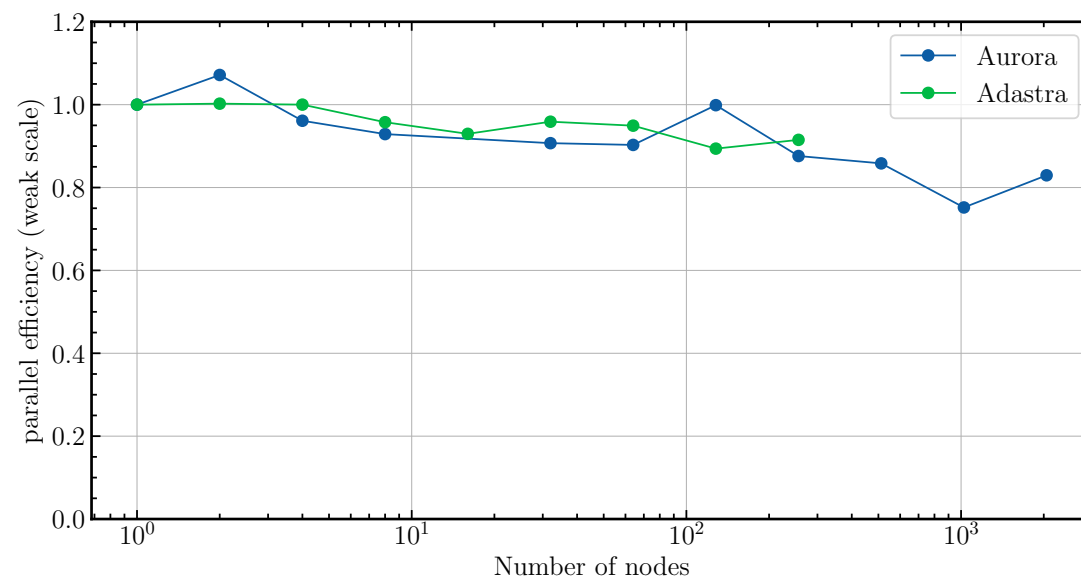
44

Up to 2048 nodes

Ring allgather



Bruck's allgather + Tunning

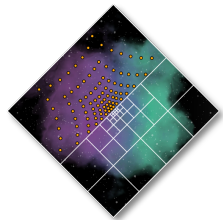


Above 80% at 400Pflop (20% of Aurora) !

Next :

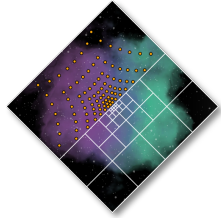


- Granted extension of 40.000 node hours to go beyond 2048 nodes
- Shamrock will be added to Aurora CI as we uncovered many MPI issues



Shamrock

- Re-work (again) of the setups to go beyond 2048 (out of memory currently)
- If remaining compute time attempt physical runs at exascale
- Apply for ALCC/INCITE



Shamrock

- Multi-method
- Multi-GPU
- Many user oriented features
- Scalability was already robust & even better now



- Exascale require a lot of tweaking
- Shamrock's MPI tuning is now default on Aurora (/w Mike Wilkins)
- Weak scaling efficiency > 80% at 24.000 GPU tiles

Thanks !



(c) 2024

