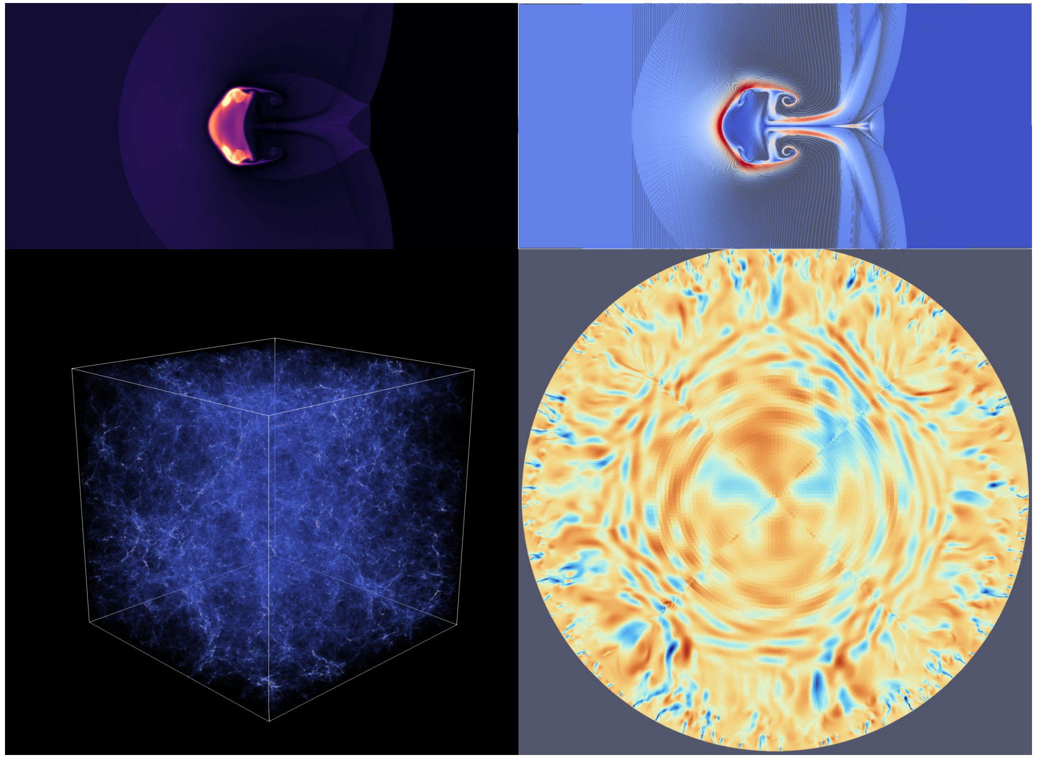
Dyablo:
Performance portable AMR for astrophysics



Ramses SNO Days

24/11/2025

Maxime.Delorme@cea.fr

Arnaud.Durocher@cea.fr

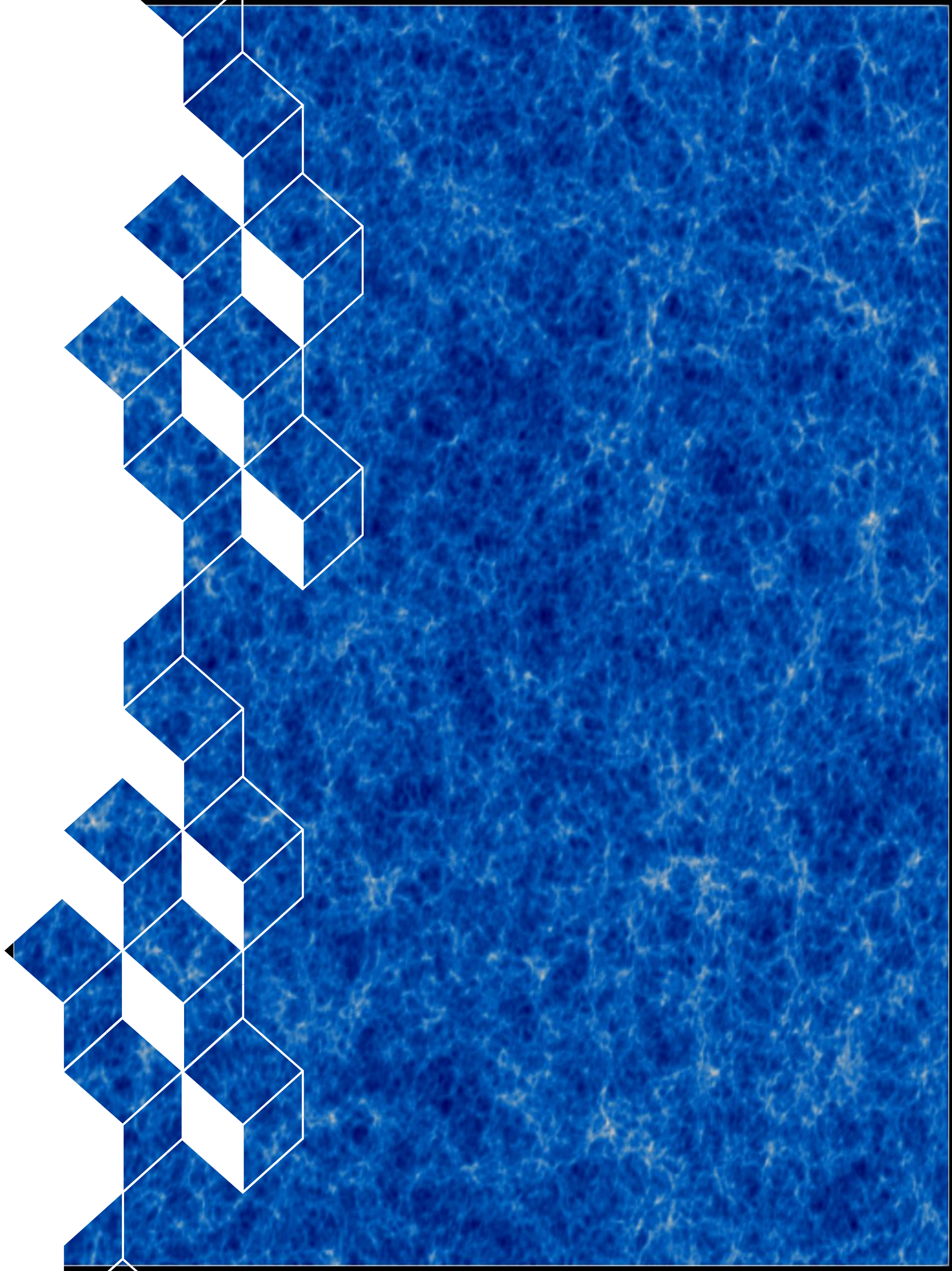
Commit authors: Dominique Aubert (ObAS), Lucas Barbier-Goy (CEA), Catherine Blume (CU Boulder), Michel-Andrès Breton (CEA), Corentin Cadiou (IAP), Grégoire Doebele (CEA), Adam J. Finley (ESTEC), San Han (IAP), Olivier Marchal (ObAS), Mike Petrault (CRIStAL), Leodasce Sewanou (CRAL), Guillaume Tcherniatinsky (IAP)

What is Dyablo ?

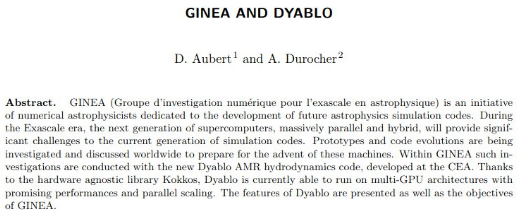
The outline:

- Modern code for the modeling of multi-scale/multi-physics astrophysical fluids
- Written in modern C++ (20 in a couple of days !)
- Relying on Kokkos for performance portability
- Open-source and community-centered
- Benefit from common needs between various astrophysics fields
- Let’s start from scratch and question everything “we do as we always did”

kokkos MPI



Community code built from collaborations



GINEA AND DYABLO

D. Aubert[1] and A. Durocher[2]

Abstract. GINEA (Groupe d'investigation numérique pour l'exascale en astrophysique) is an initiative of numerical astrophysicists dedicated to the development of future astrophysics simulation codes. During the Exascale era, the next generation of supercomputers, massively parallel and hybrid, will provide significant challenges to the current generation of simulation codes. Prototypes and code evolutions are being investigated and discussed worldwide to prepare for the advent of these machines. Within GINEA such investigations are conducted with the new Dyablo AMR hydrodynamics code, developed at the CEA. Thanks to the hardware agnostic library Kokkos, Dyablo is currently able to run on multi-GPU architectures with promising performances and parallel scaling. The features of Dyablo are presented as well as the objectives of GINEA.

WHOLE

NumPEx

FRANCE 2030 PROGRAMME DE RECHERCHE ORIGINES

cexa
cexa-project.org

Software engineering concepts

Separation of concerns

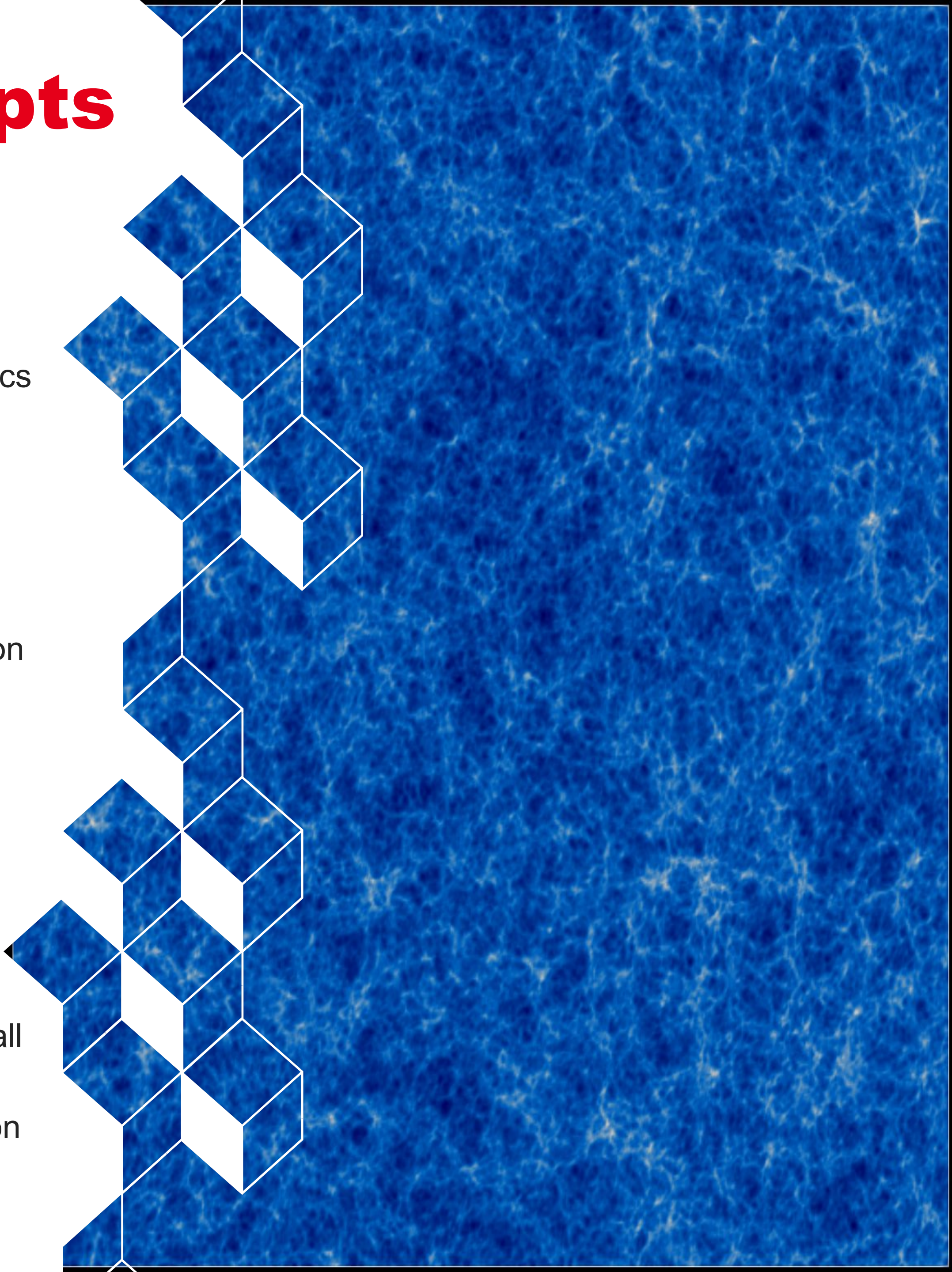
- Physicists implementations should not interfere with HPC
- Back-end implementation should be modifiable without rewriting the physics
- Interface between back-end and physics kernels should stay as stable as possible

High-level abstractions

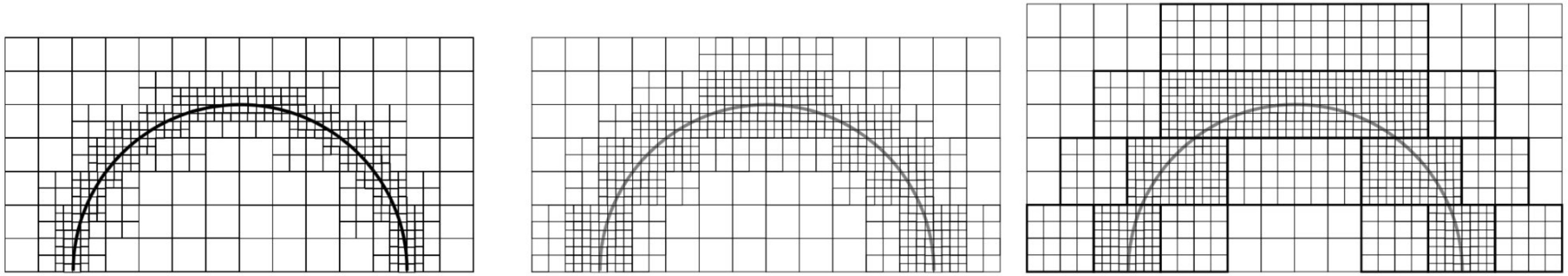
- Having access to high-level classes hides the complexity of implementation
- Easier implementation of physics without having to bother about memory layout/management
- Example: The `CellIndex` class manages neighbor finding for AMR

Modularity

- C++ templating + performance portability = long compilation times
- To avoid having to recompile the code (de)activating modules à la Pluto, all major features of Dyablo are *"plugins"*
- Plugins are compiled once and can be activated/deactivated at initialization instead of compilation.



Block-based AMR



Cell-based AMR with 382 cells

Block-based AMR with 596 cells

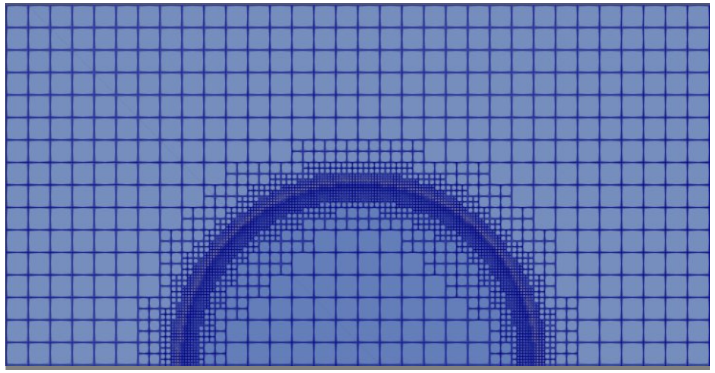
Patch-based AMR with 836 cells

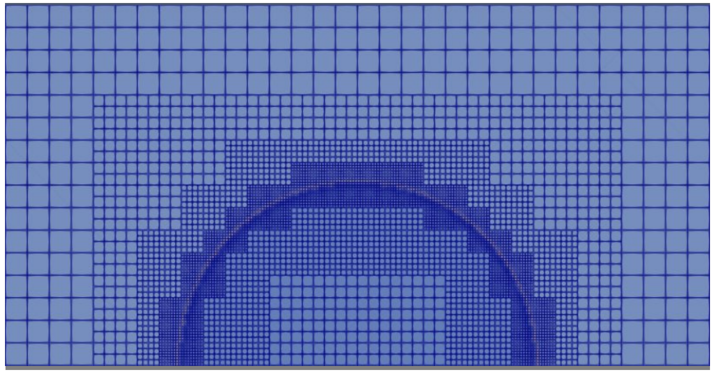
Source: Dunning et al. 2019; "Adaptive Mesh Refinement in the Fast Lane."

Advantages of block-based AMR:

- **Smaller tree:** AMR cycle is faster
- **Increased regularity**: More conformal faces = easier streamlining for the GPU
- Cell-based still possible by taking blocks of size 1

Cell-based
600k octants
600k cells



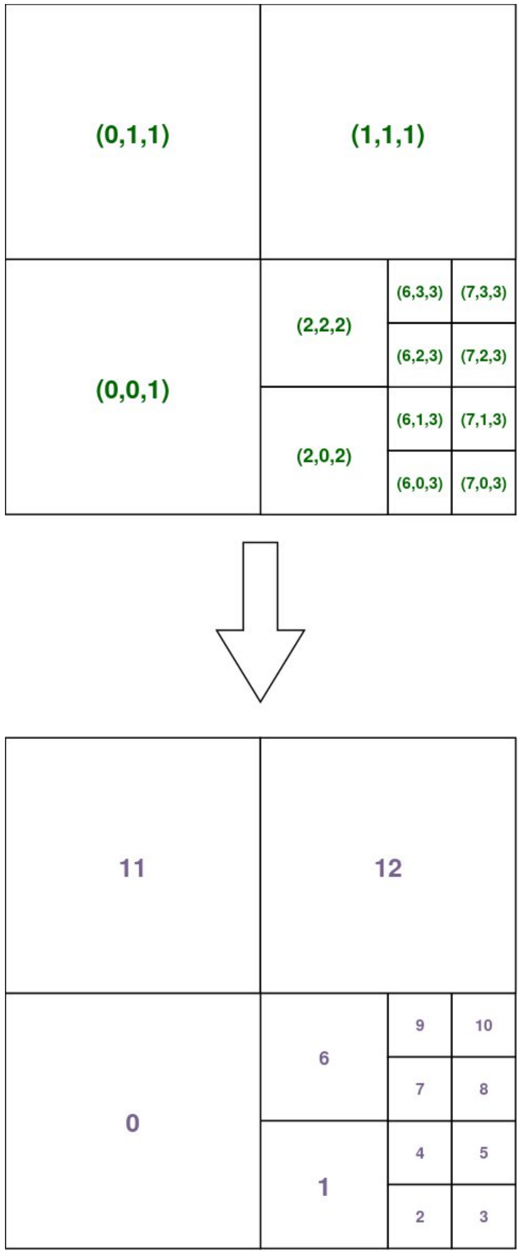


Block-based
18k octants
4x4x4 blocks
1100k cells

Hashmap based neighbor search

Memory layout:

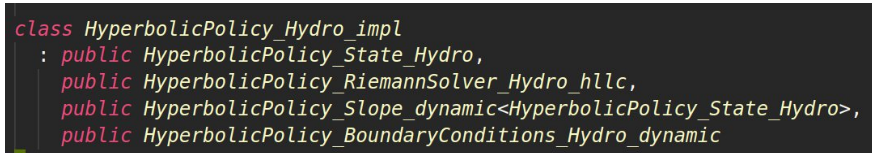
- Linear octree: only the leaves (ie blocks) are stored
- Space-filling curve: octants are ordered along the Morton Z-curve
- All octants are referenced in a hashmap: (i, j, k, level) -> octant_id
- Looking for a neighboring octant is simply looking for matching keys in hashmap:
  - Construct the neighbor key, if it exists return the stored id
  - If not, construct the neighbor key at level-1
  - If still nothing, construct the neighbor key at level+1
- Allows the data to live at all times on GPU memory for computation



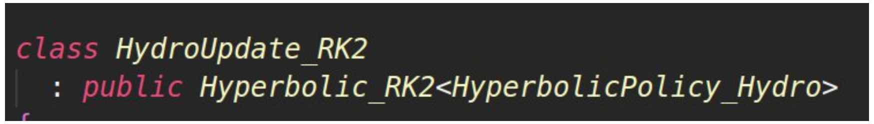
Plugins

Plugins are feature “bricks”

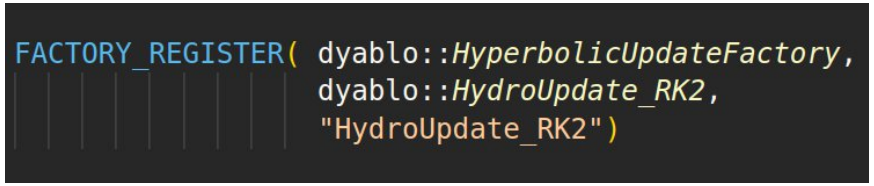
- Every aspect of the code that should be modular is made as a plugin
- Static inheritance and a factory concept to instantiate the plugins at runtime
- Plugins can simple simple objects:
  - Dt calculation, refinement criterion, IO backends, etc.
- Or more complex, multi-layered classes
  - HydroState + HLLC + Slope limiters + BoundaryConditions = Hydro policy
  - Hydro policy + RK2 scheme = Hydro_RK2 plugin
- Plugins are registered in generic factories at compile time
- And selected in .ini file at runtime



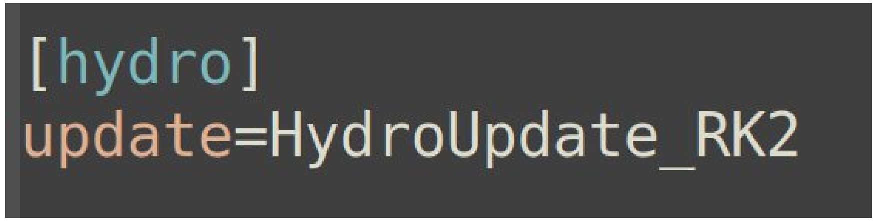
```
class HyperbolicPolicy_Hydro_impl
  : public HyperbolicPolicy_State_Hydro,
    public HyperbolicPolicy_RiemannSolver_Hydro_hllc,
    public HyperbolicPolicy_Slope_dynamic<HyperbolicPolicy_State_Hydro>,
    public HyperbolicPolicy_BoundaryConditions_Hydro_dynamic
```



```
class HydroUpdate_RK2
  : public Hyperbolic_RK2<HyperbolicPolicy_Hydro>
```



```
FACTORY_REGISTER( dyablo::HyperbolicUpdateFactory,
                  dyablo::HydroUpdate_RK2,
                  "HydroUpdate_RK2")
```

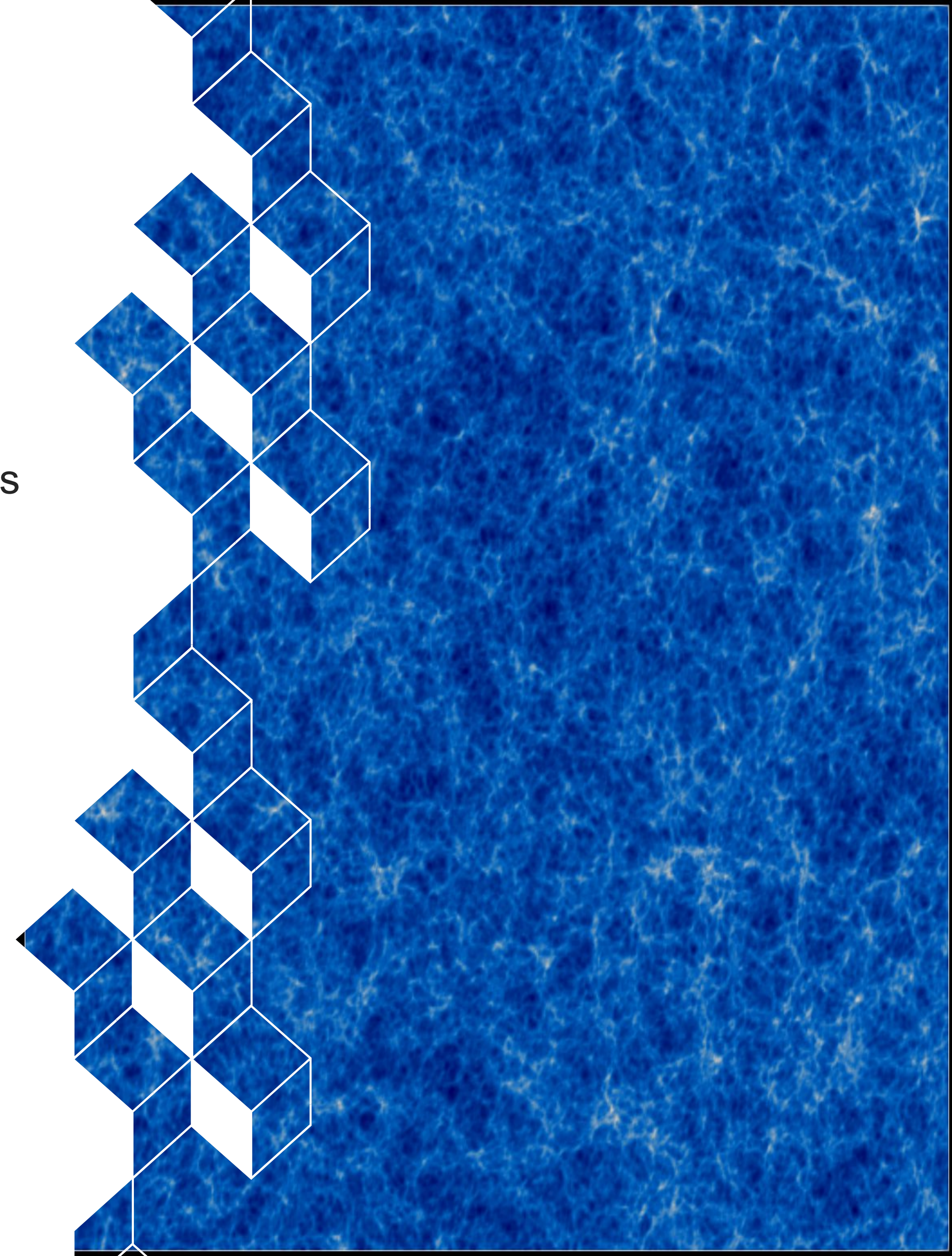


```
[hydro]
update=HydroUpdate_RK2
```

What is in Dyablo ?

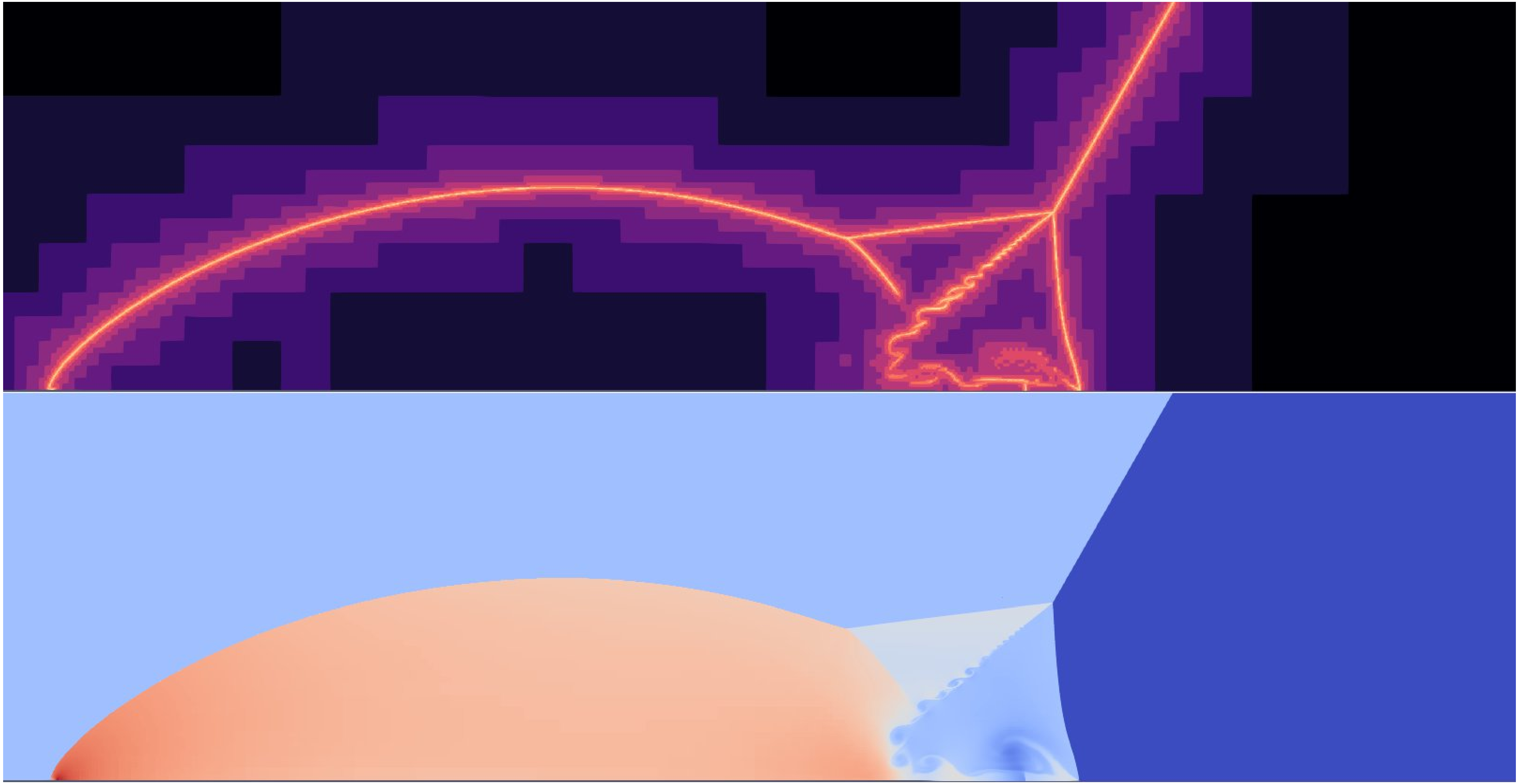
Multiple branches:

- Dev branch with common useful plugins:
  - Hydro/MHD(div cleaning)/RHD(M1 explicit) solvers
  - Thermal Conduction/Viscosity explicit parabolic solvers
  - (self) Gravity update with CG
  - Particles CIC, NGP, tracers
  - Source terms: basic cooling,
- Community branches:
  - Branches led by community applications
  - Two main community branches for now
- Feature branches:
  - Sub-branches made for implementing features

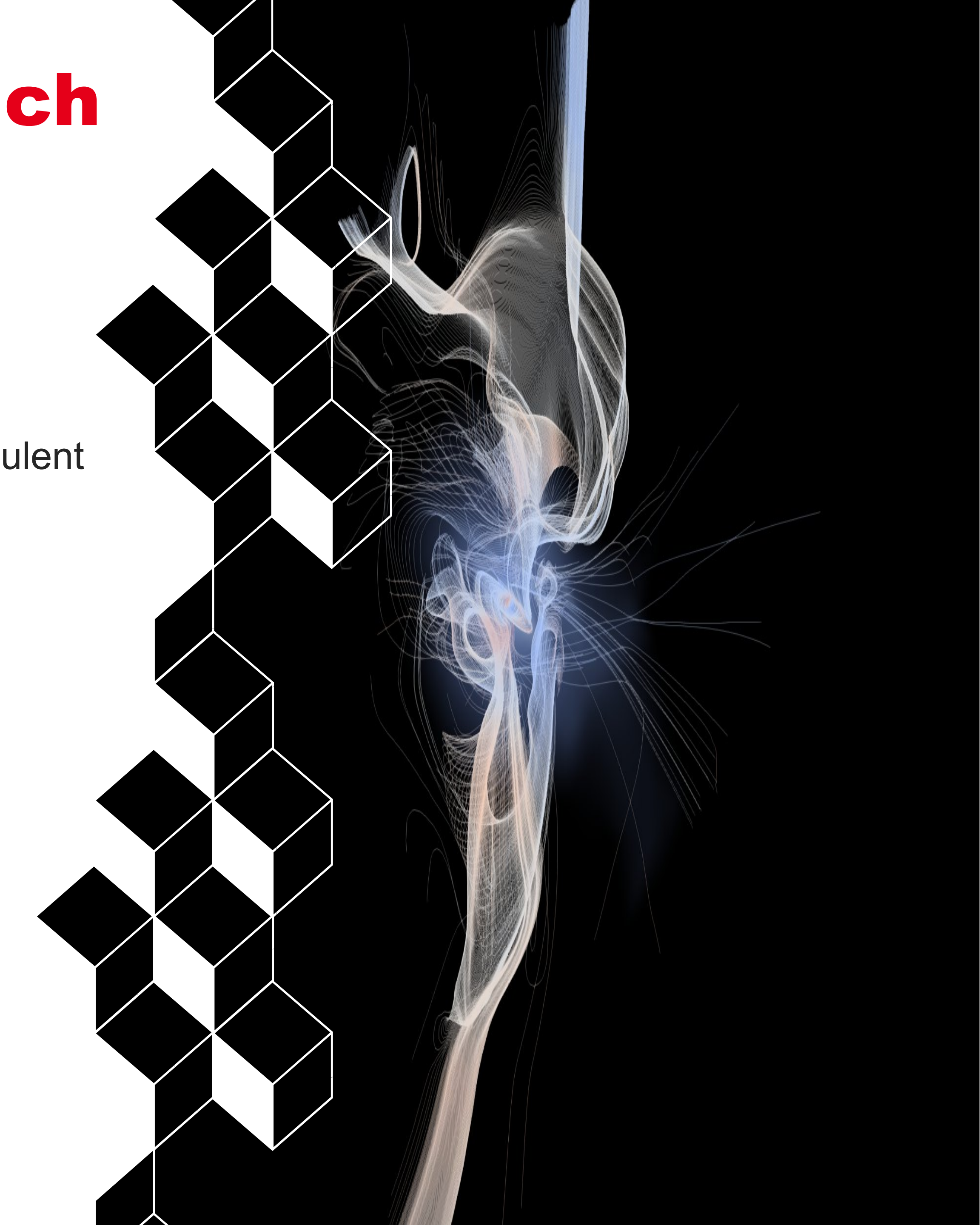


Double Mach reflection

Double Mach reflection



Developments: `cosmo` branch



Exclusive features:

- **Hydro:** Pressure fix (not finalized)
- **Source Terms:** Cooling using Grackle public tables, turbulent forcing term (not finalized)
- **Particles:** Stellar feedback, Star formation
- **IOs:** Reading from Gadget snapshot (not finalized)

Contributors:

- Dominique Aubert – ObAS
- Michel-Andrès Breton – CEA Saclay
- Corentin Cadiou – IAP
- Olivier Marchal – ObAS

A first galaxy with Dyablo

Video courtesy of: Dominique Aubert, Michel-Andrès Breton, Corentin Cadiou, Olivier Marchal,

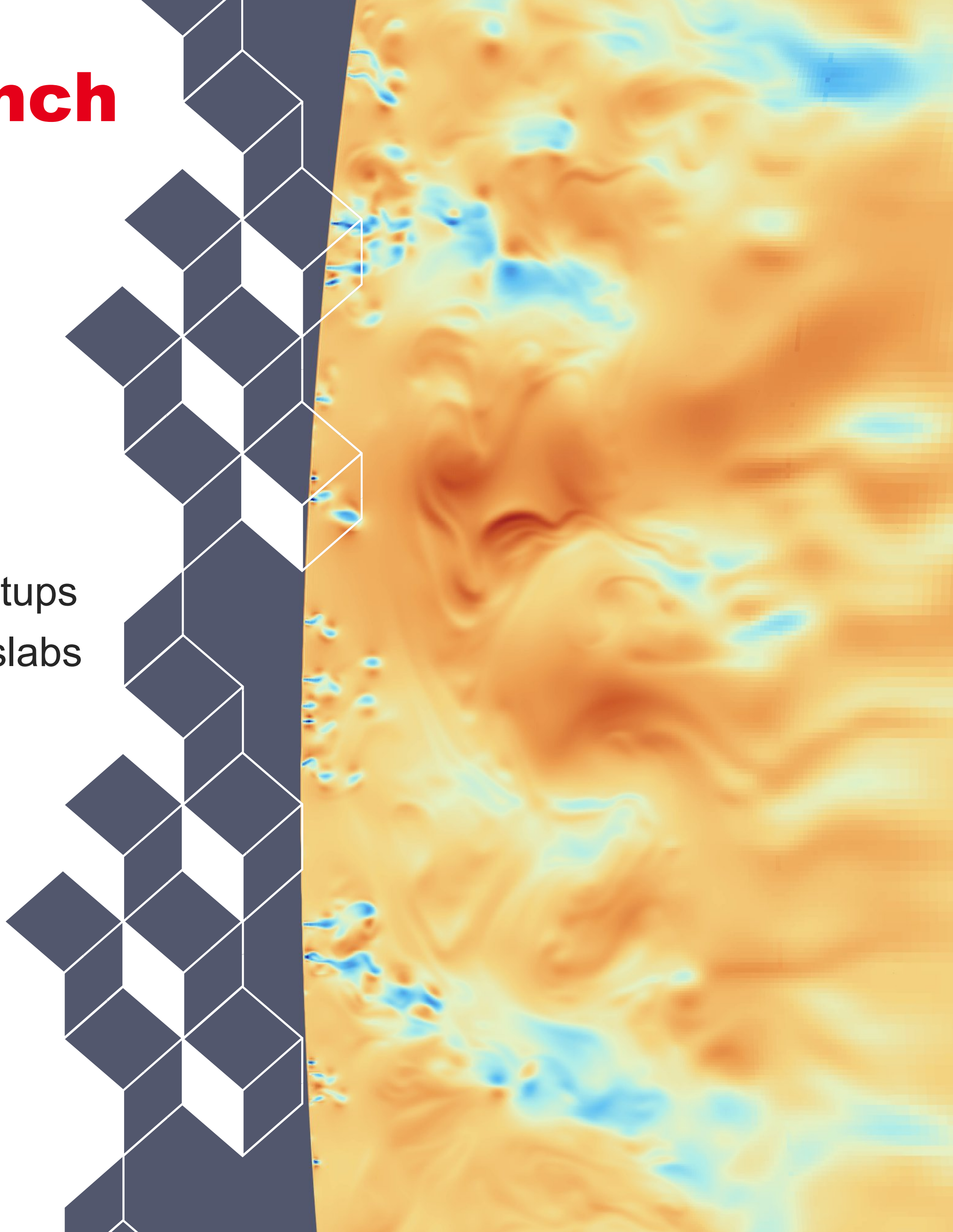
Developments: `wholesun` branch

Exclusive features:

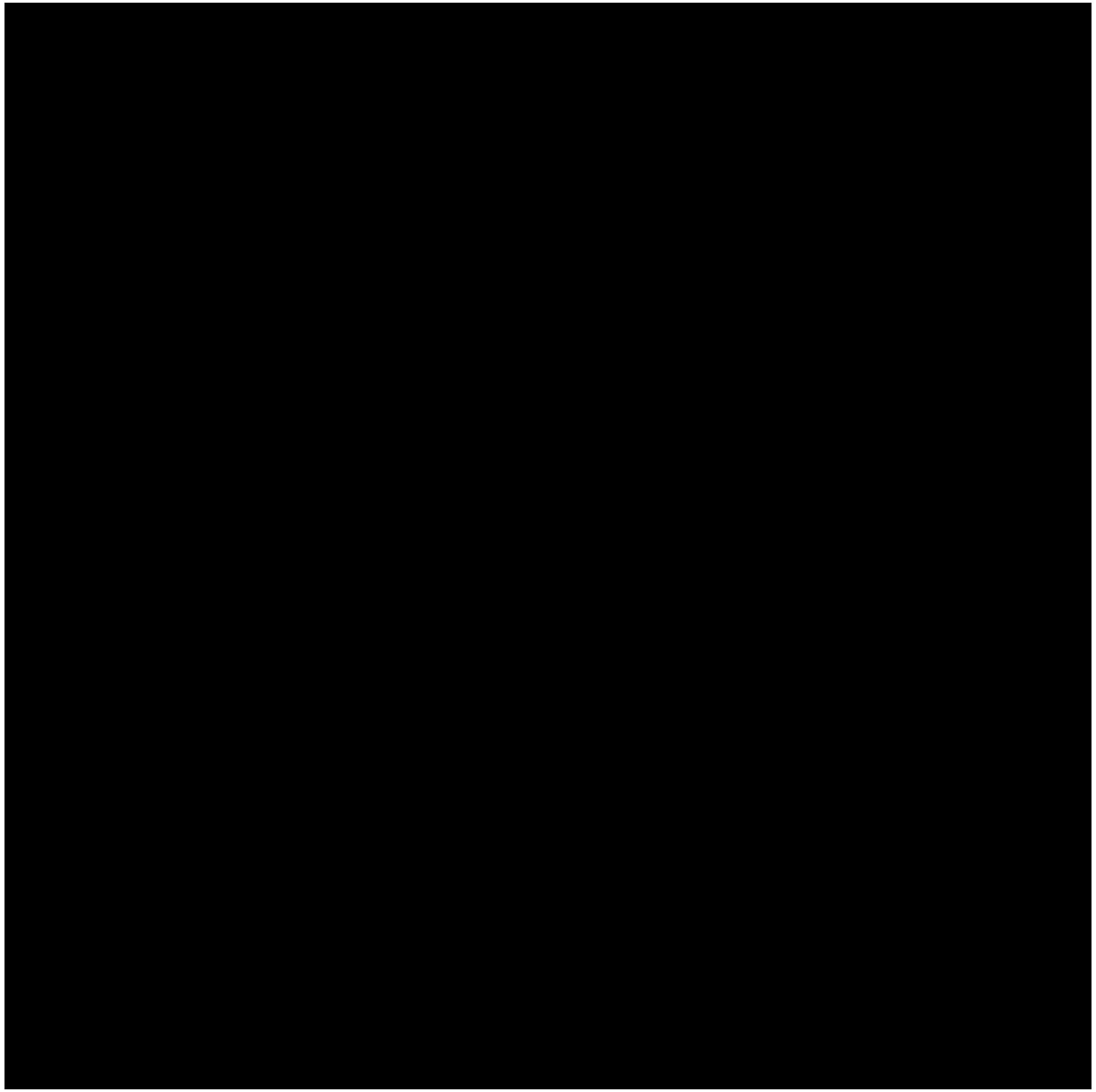
- **Geometry:** Isometric mappings
- **MHD:** Five-Waves solver from MDLS
- **Source Terms:** Isothermal atmosphere cooling
- **Well-balancing:** Alpha-beta well balancing for radial setups
- **Setups:** Various setups for solar physics ranging from slabs to radial profiles

Contributors:

- Lucas Barbier – CEA Saclay
- Catherine Blume – University of Colorado Boulder
- Grégoire Doebele – CEA Saclay
- Adam Finley – ESTEC Leiden

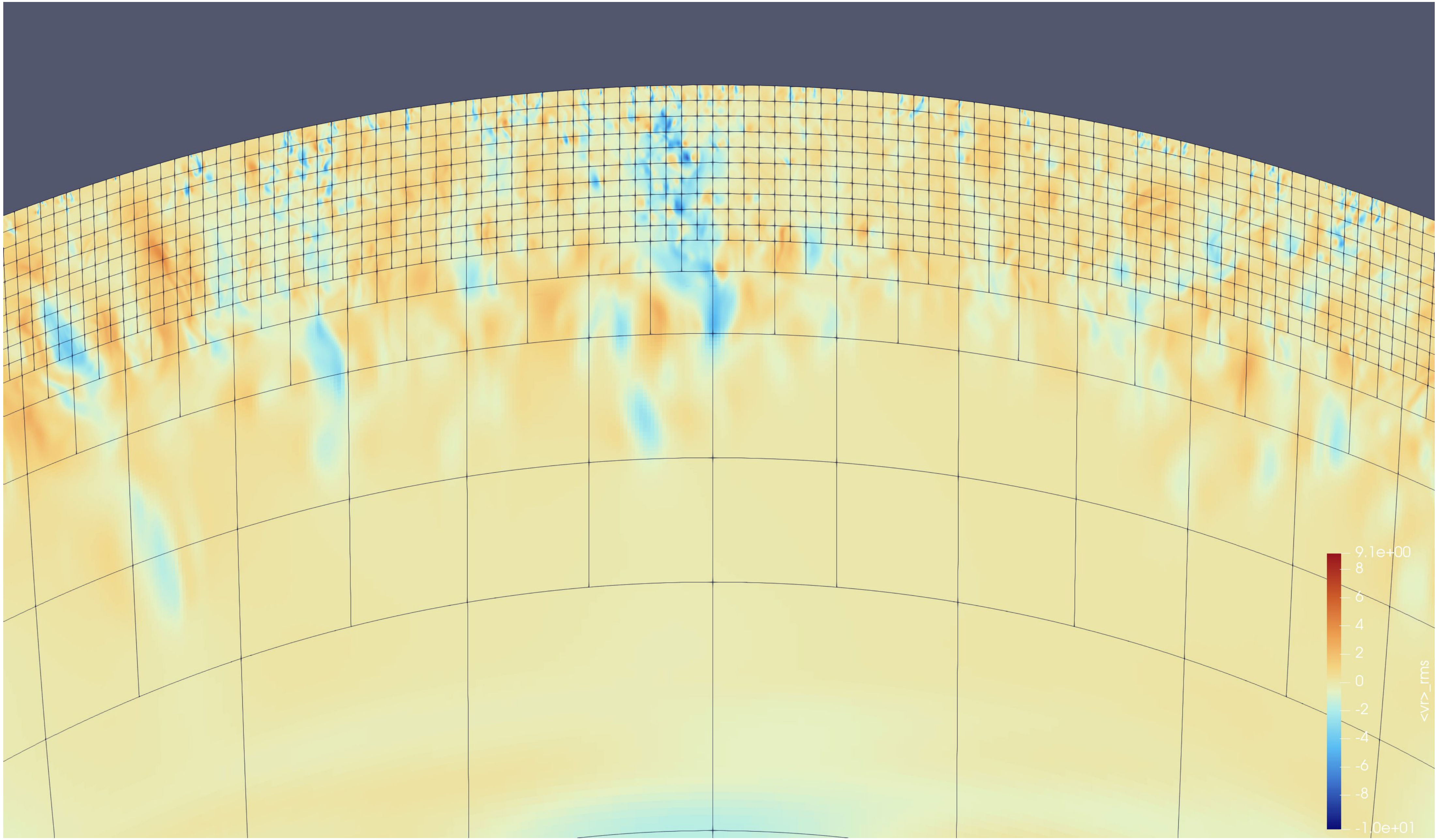


Solar like setup with geometry module



Video provided by Grégoire Doebele (CEA Saclay)

Solar-like setup with geometric module simulation (and AMR)



Edges indicate blocks, not cells !

Developments: other branches

Other branches (development or future community):

- **Multigrid and TSC scheme:** Michel-Andrès Breton
- **Dust/ISM:** Benoît Commerçon, Leodasce Sewanou
  - Base for a future community branch (see maybe Benoît’s talk on Wednesday)
- **Subgrid physics/comparison with Ramses:** Florent Bréhard, Mike Petrault, Jenny Sorce
- **Cosmic rays:** Yohan Dubois, San Han, Guillaume Tcherniatinsky
- **Core features** - Passive scalars, hierarchical timestepping, intermediate level storages: Maxime Delorme, Arnaud Durocher

```
get("GravitySolver_multigrid").start();

reach_cell = pdata->foreach_cell;
nk = foreach_cell.get_amr_mesh().getRank();
reach_cell.get_amr_mesh();

s.get( epare Multigrid").start();
level in ctree
ls = mes ils::find_max_levels(foreach_cell);
evel_m und = max_levels.global;
level_t ocal_leve max_found = pdata->local_level_max_found = max_levels.local;

inter te octs
ermedi amr_mesh);
tOctree& esh = amr_mesh.getLightOctree();
t level_coarse = pdata->level_coarse = lmesh.get_level_min();
KOKKOS G( pdata->first_mpi_multigrid_level <= level_coarse, "Full co

"field", olution", "rhs",  "res", "mask", "phix", "phiy", "phiz" });
diate_fields( {"rho","gphi", "field", "solution", "rhs", "res", "mask"}
vector<UserData_fields::FieldAccessor_FieldInfo> fields_info = {
Irho},
 Iphi},
  Ifield},
    Isolution},

s", Ires dual},
ask", Imas },

td::vec r<UserDat fields::FieldAccessor_FieldInfo> fields_info_leaves(fields_info);
fields_in leaves.push_back({"gx", Igx});
lds_inf aves.push_back({"gy", Igy});
ds_in ves.push_back({"gz", Igz});
ds_info_l ves.push_back({"phix", Iphix});
s_info_lea es.push_back({"phiy", Iphiy});
nfo_lea push_back({"phiz", Iphiz});

FieldAcce or U = U_.getAccessor({fields_info_leaves});
:FieldAccessor Uintermediate = U_.getAccessor_intermediate({fields_info});
>U = U;
a->Uintermediate = Uintermediate;
```

Roadmap for Dyablo v1.0

MULTIGRID

- Integration of the multigrid branch
- Adaptation of the general structure for intermediate level storage/communication

PASSIVE SCALARS

- Passive scalar advection regardless of scheme used

HIERARCHICAL TIMESTEPPING

- Partial integration of levels
- AMR cycle in the middle of an integration cycle

DOCUMENTATION

- Finalizing contributor guide
- Doxygen/Breathe
- CI integration

BENCHMARKING

- Re-run of all benchmarks: hydro, MHD, cosmo, solar.
- Comparison with state of the art

DYABLO V1.0

- Public tag
- Method paper

Current and future work

Peripheral developments/activities:

- **Numpex:**
  - ExaDOST (PC3): Sylvain Joube (postdoc) is working on a new solution for compressed AMD data-formats, visualization and analysis.
  - ExaDI (PC5): Dyablo and Samurai (Polytechnique/CMAP) are leading a working group on a series of benchmarks for modern AMR codes to pinpoint common difficulties
- CExA/PTC: Jean-François David (postdoc) is building analysis tools for the profiling and optimization of large kernels in Kokkos applications

Future work for the core team:

- Python front-end:
  - Being able to fully control the time-loop and let C++ handle the computations
- In-situ analysis:
  - Use the supercomputers to calculate diagnostics and first step analysis while the code is running

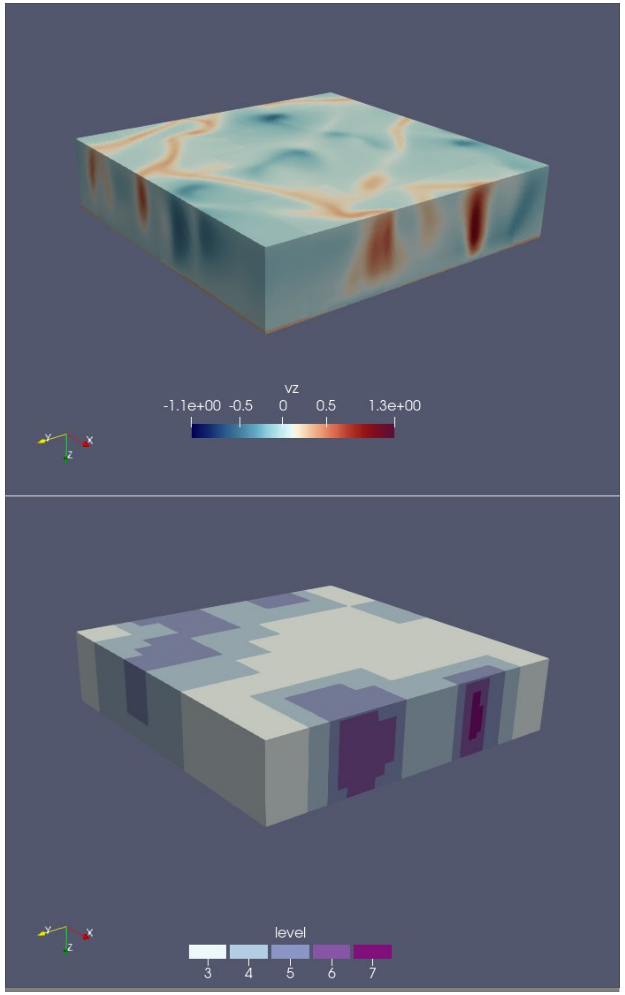
Thank you

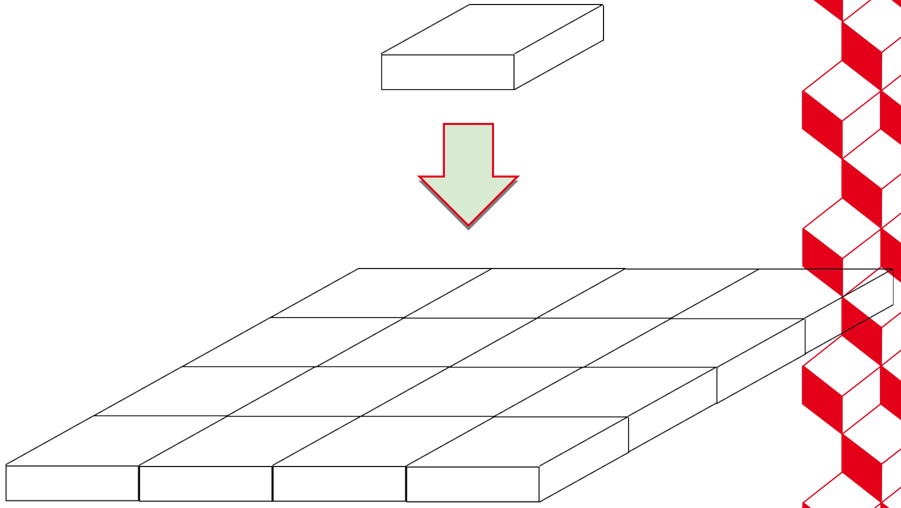
Weak scaling benchmarks

Use case

Solar convection slab :

- 3-7 refinement levels
  - Base resolution 128x128x32
  - Max resolution 2048x2048x512
  - 30.6M cells per domain
- Horizontal tiling per MPI process
- 100 iterations
- 1 AMR cycle per iteration
- No Load-balancing
- Scalability tested on Jean-Zay and Ad-Astra [Tier 1 french SC]
  - CPU : Intel CSL, AMD Genoa
  - GPU : v100, a100, MI250X
  - Tested up to 2048 GPUs ~62 billion cells

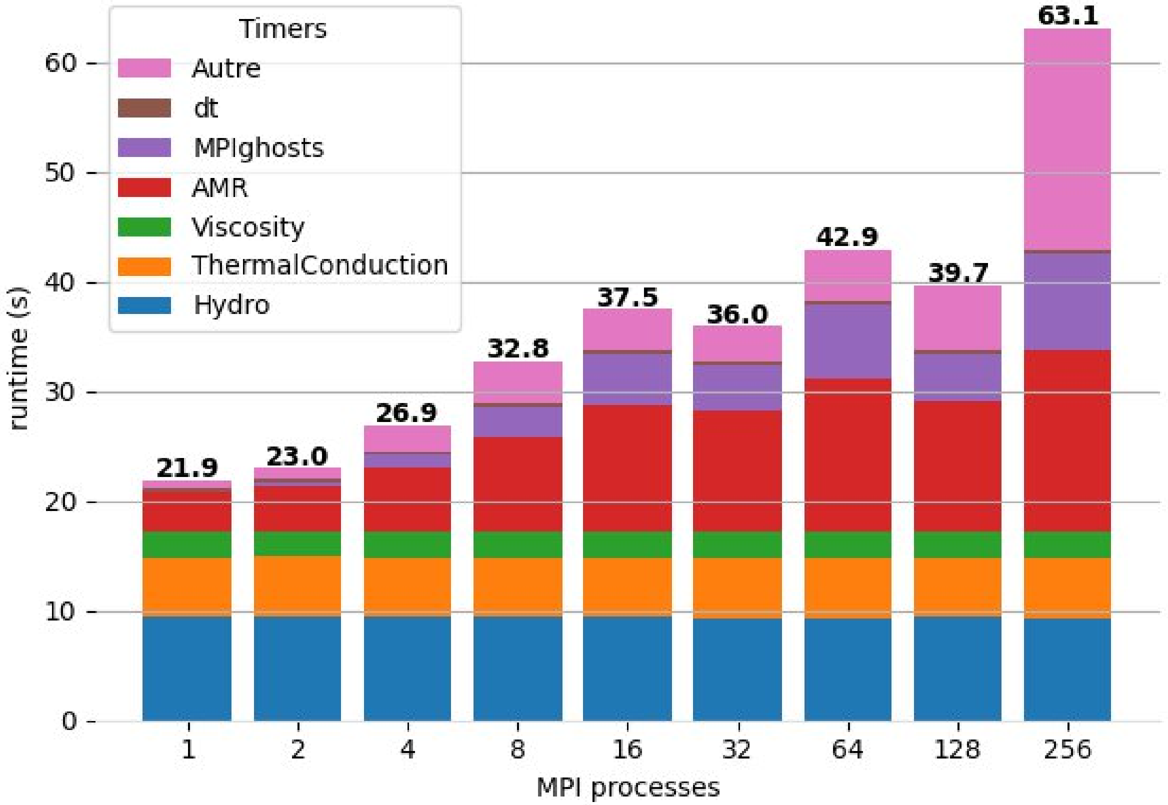




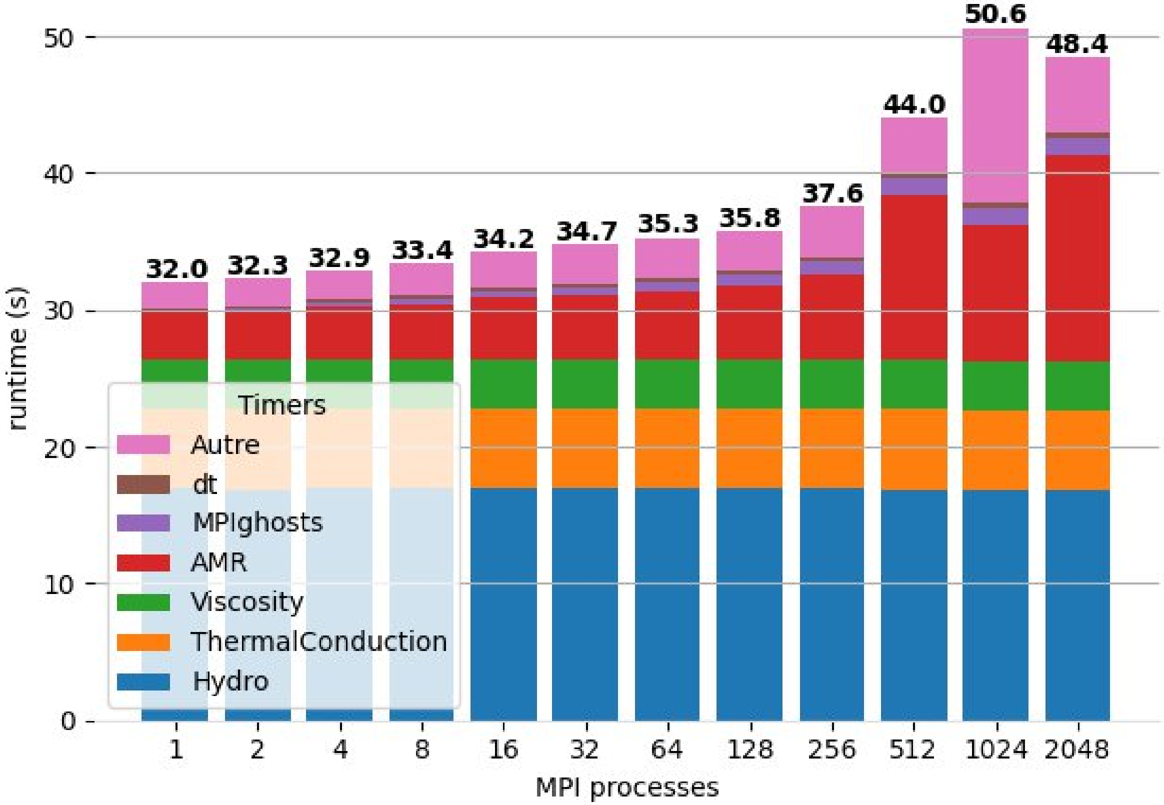
Replication on N MPI processes

Weak scalability for convection runs

Jean Zay GPU *(8 x Nvidia A100)*



Ad Astra GPU *(8 x AMD MI250x)*

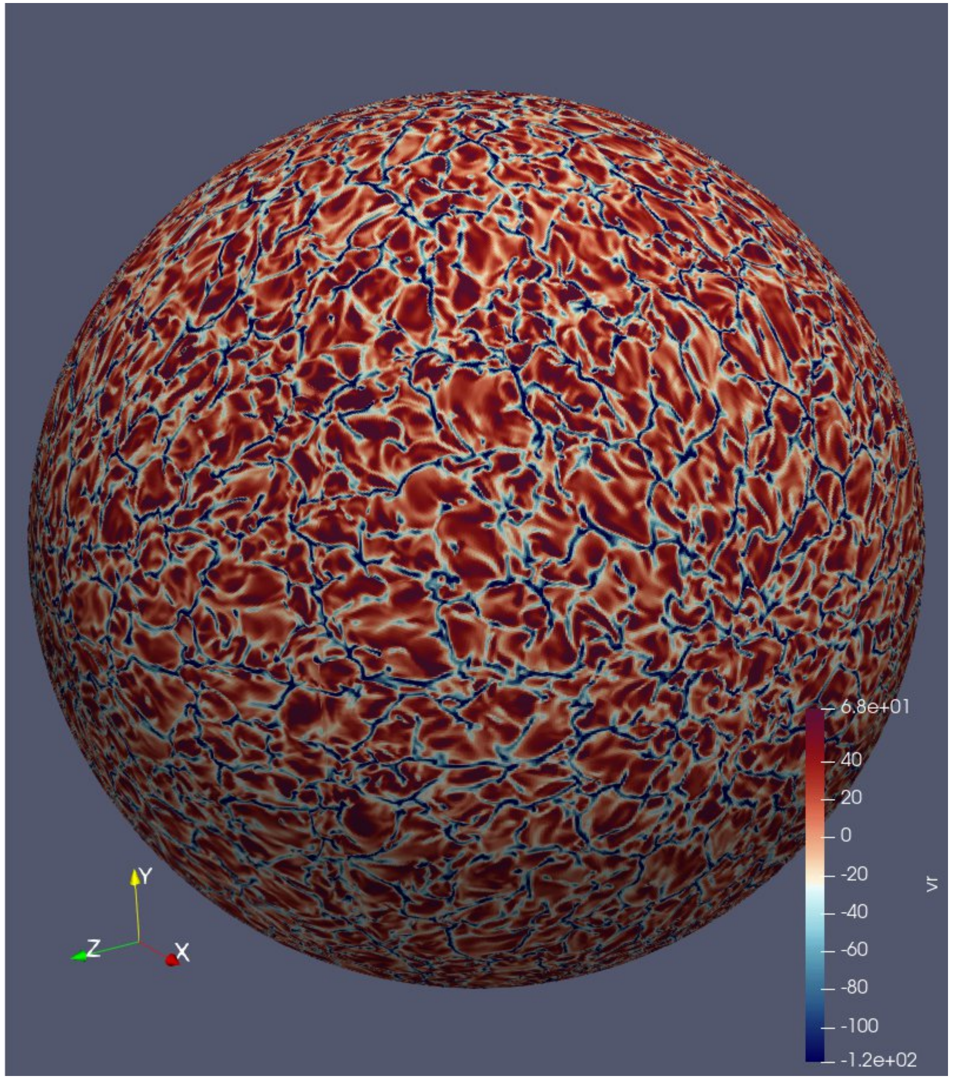


Strong scalability benchmark

Use case

Convection on a sphere:

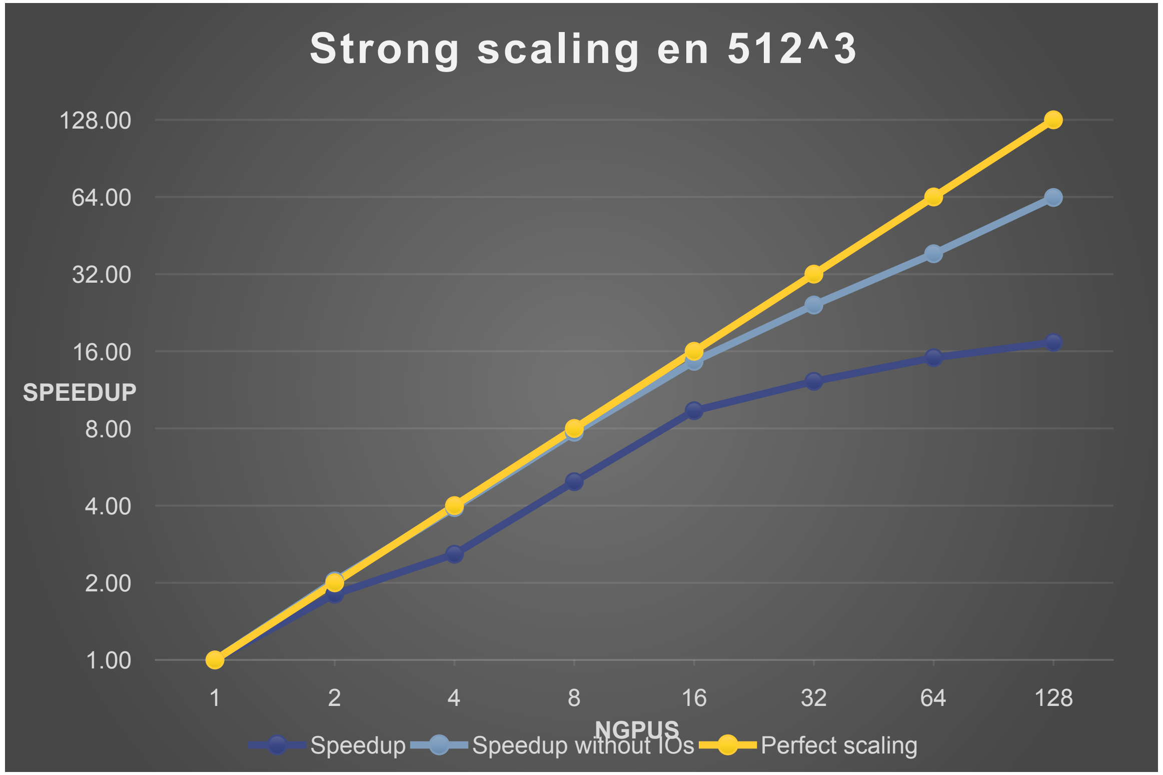
1. Equations:
    1. Navier-stokes
    2. Thermal conduction
    3. Gravity
    4. Heating at the center
2. Boundary conditions: Fixed energy flux at surface
3. 512^3 fixed resolution (130Mcells), solved on a radial mapping

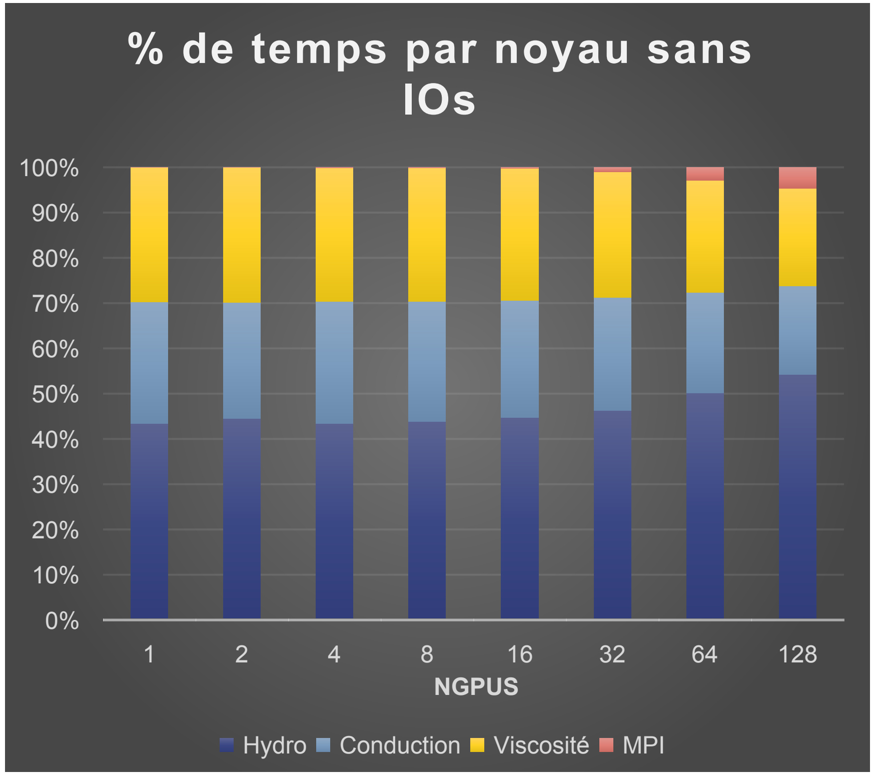


Warning: Geometry module is still very experimental and numbers should be taken with a pinch of salt.

Strong Scalability

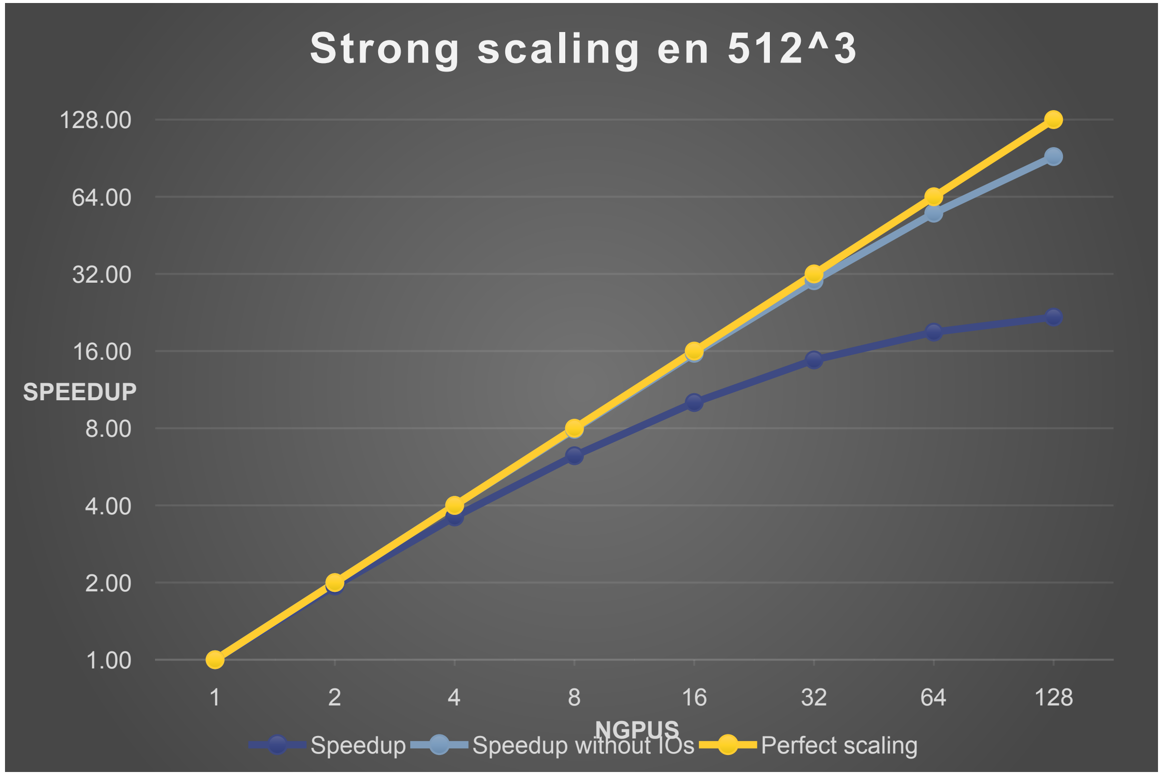
Jean-Zay H100

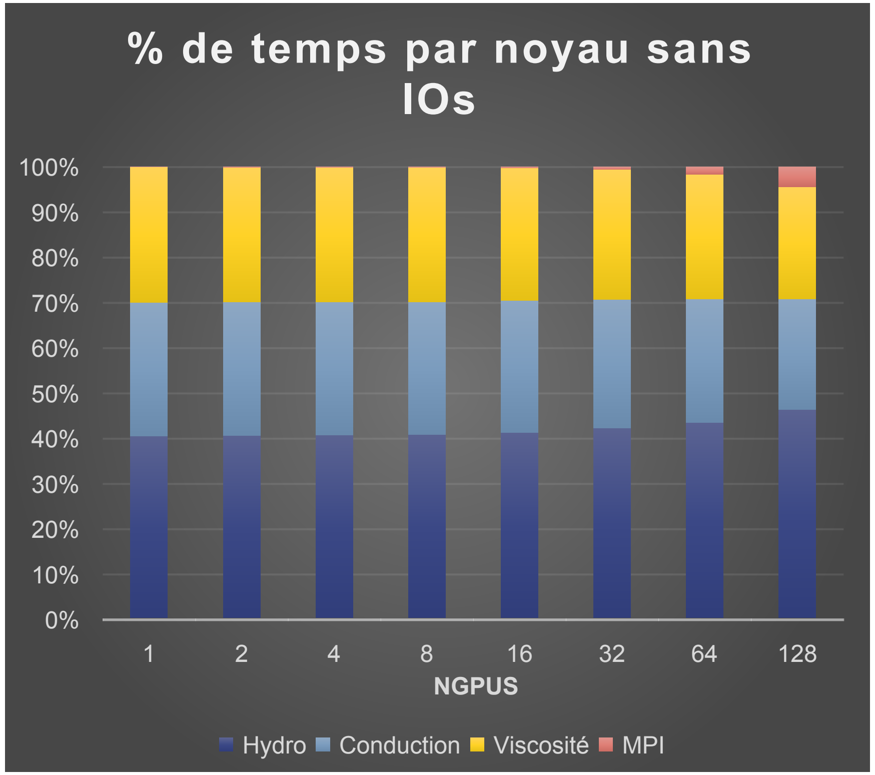




Strong Scalability

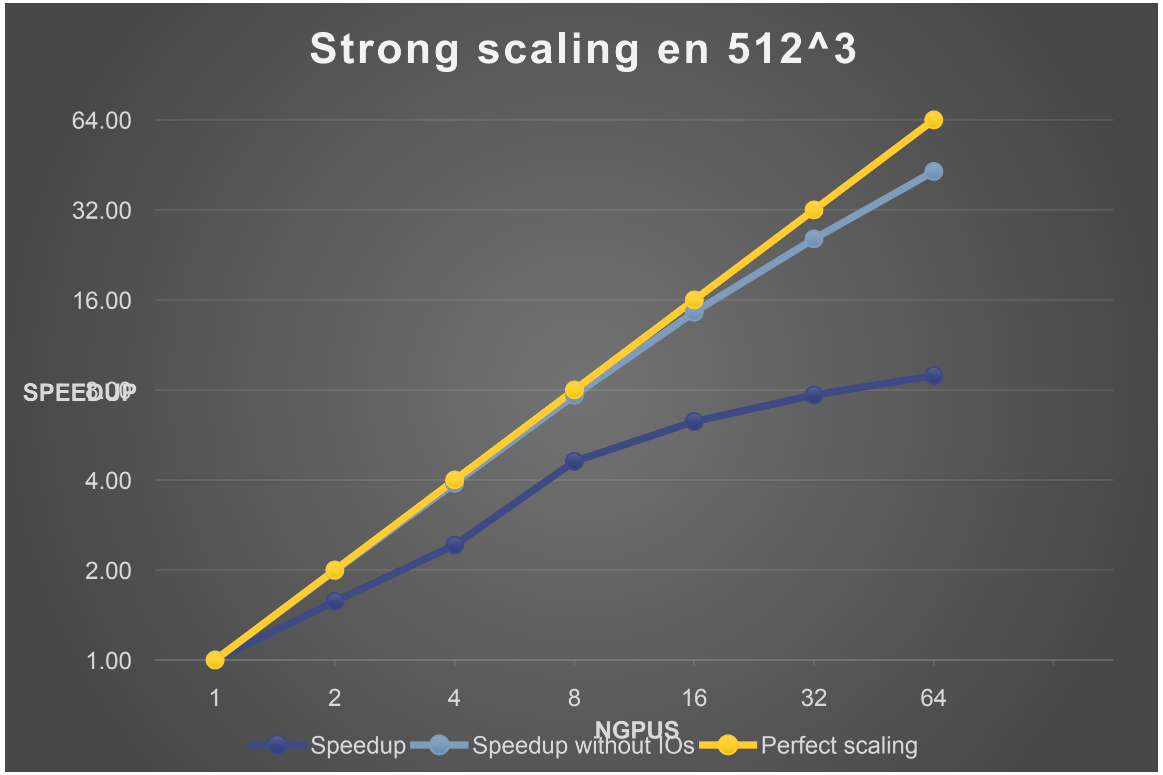
AD-Astra MI250X

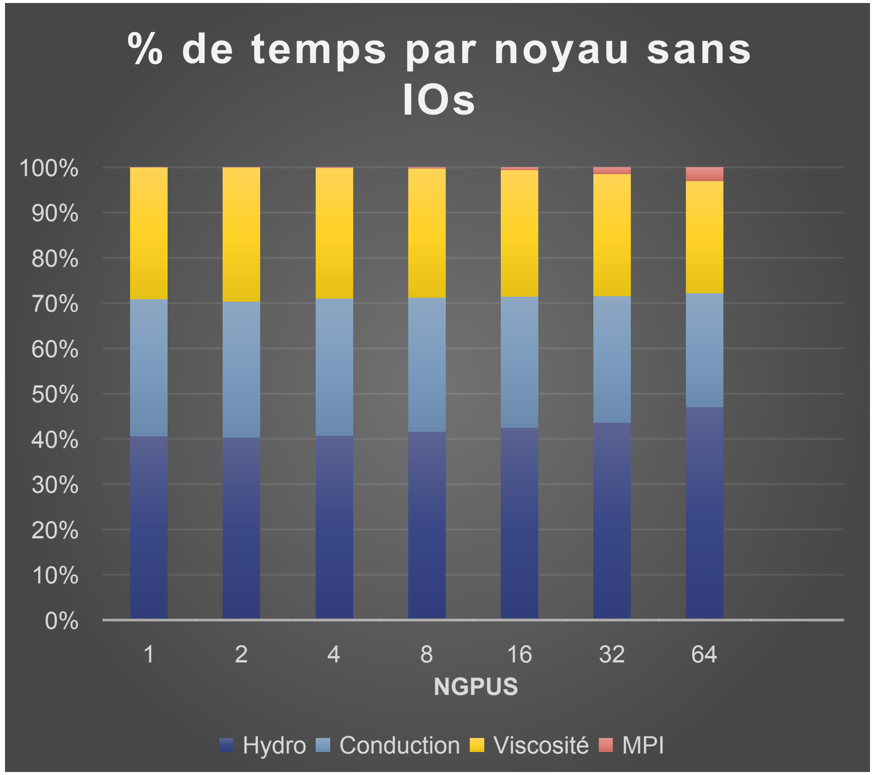




Strong Scalability

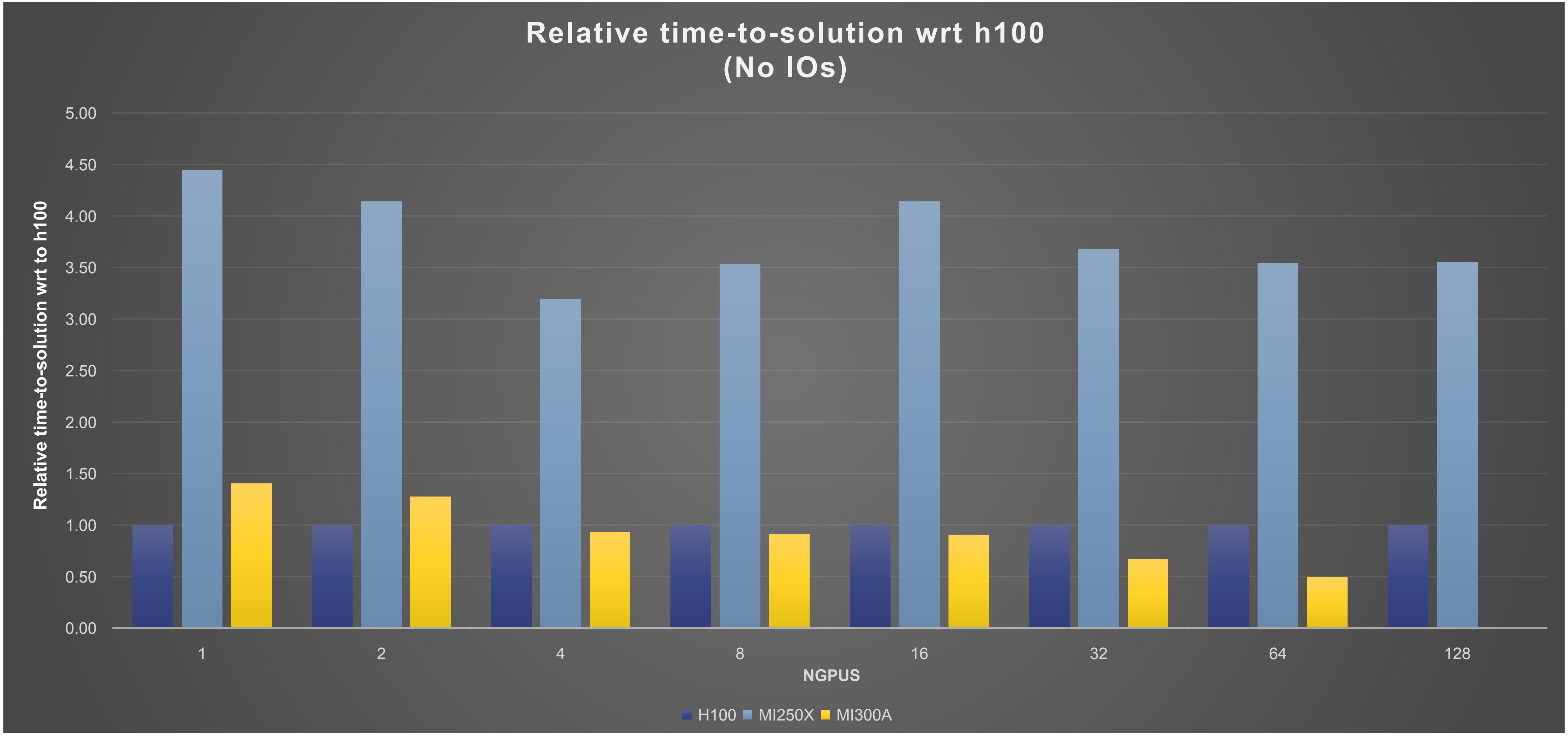
AD-Astra MI300A





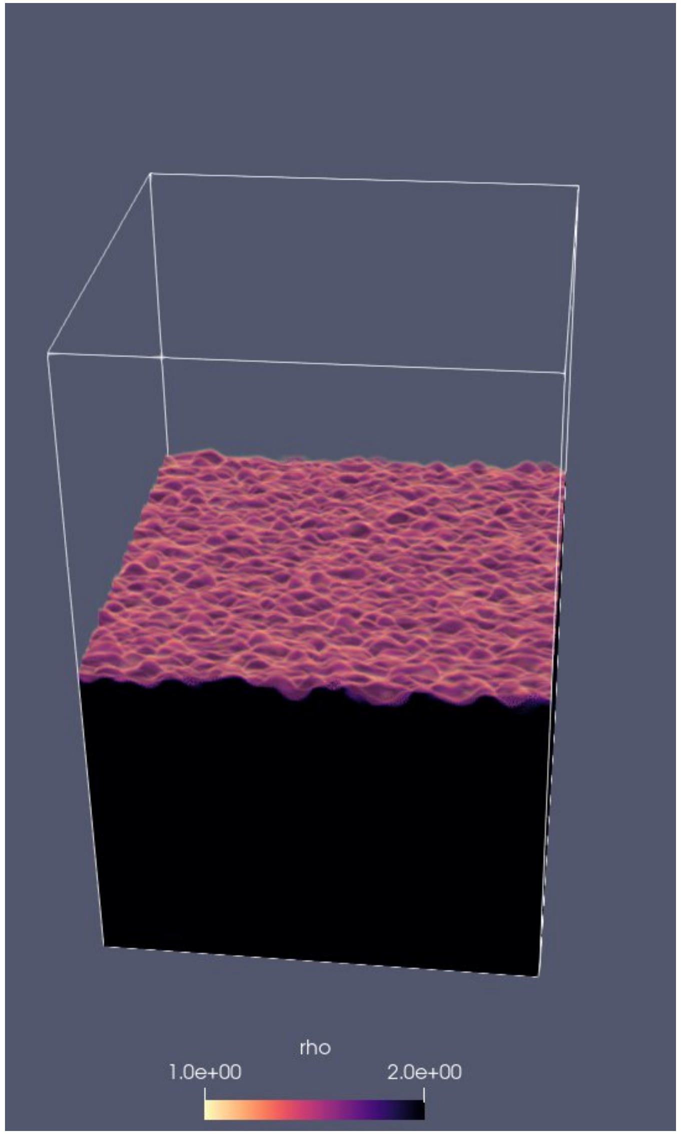
Strong Scalability

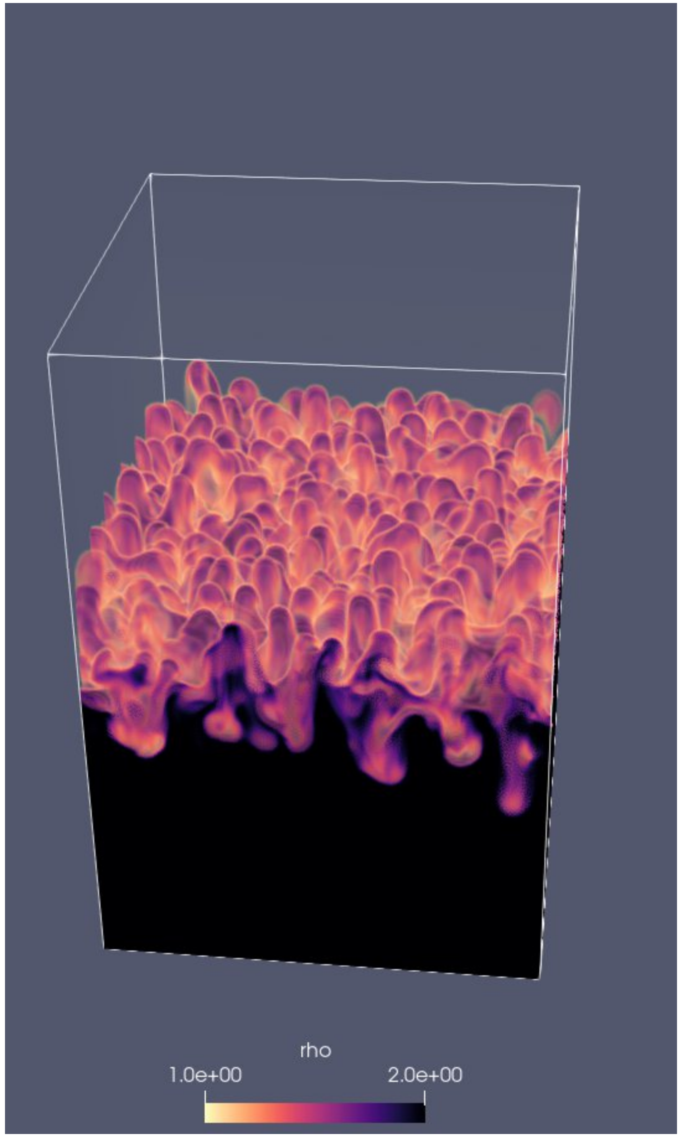
Architecture comparison



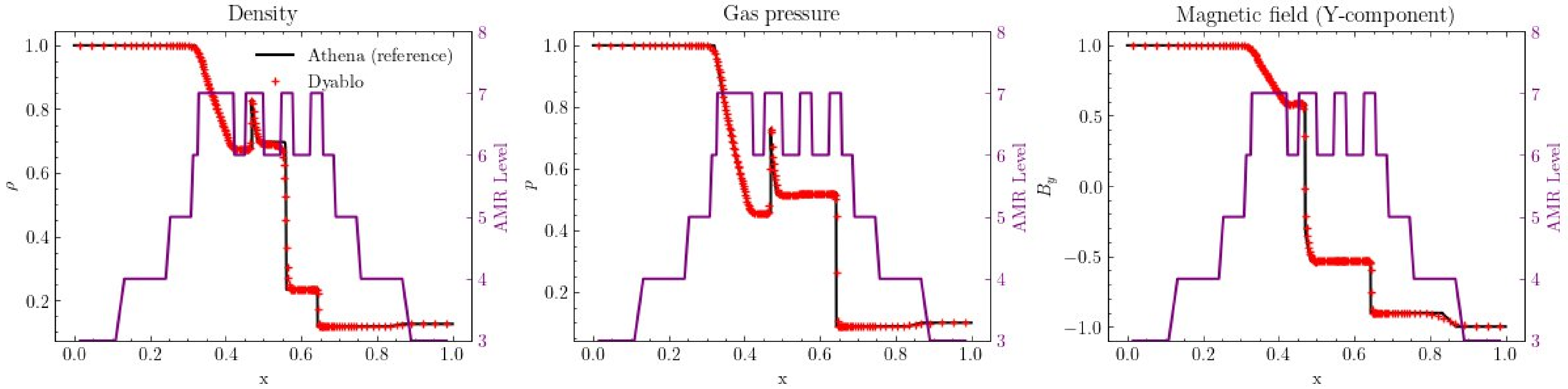
Cloud-shock interaction

Rayleigh-Taylor 3D

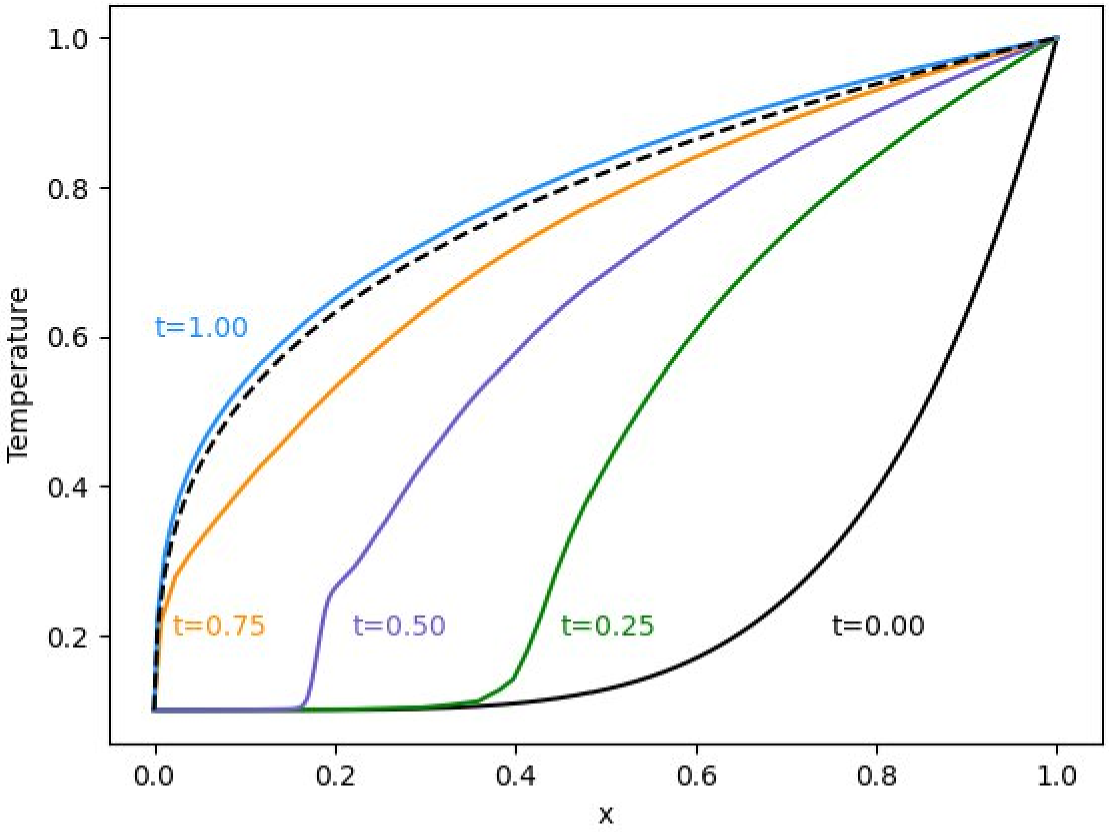




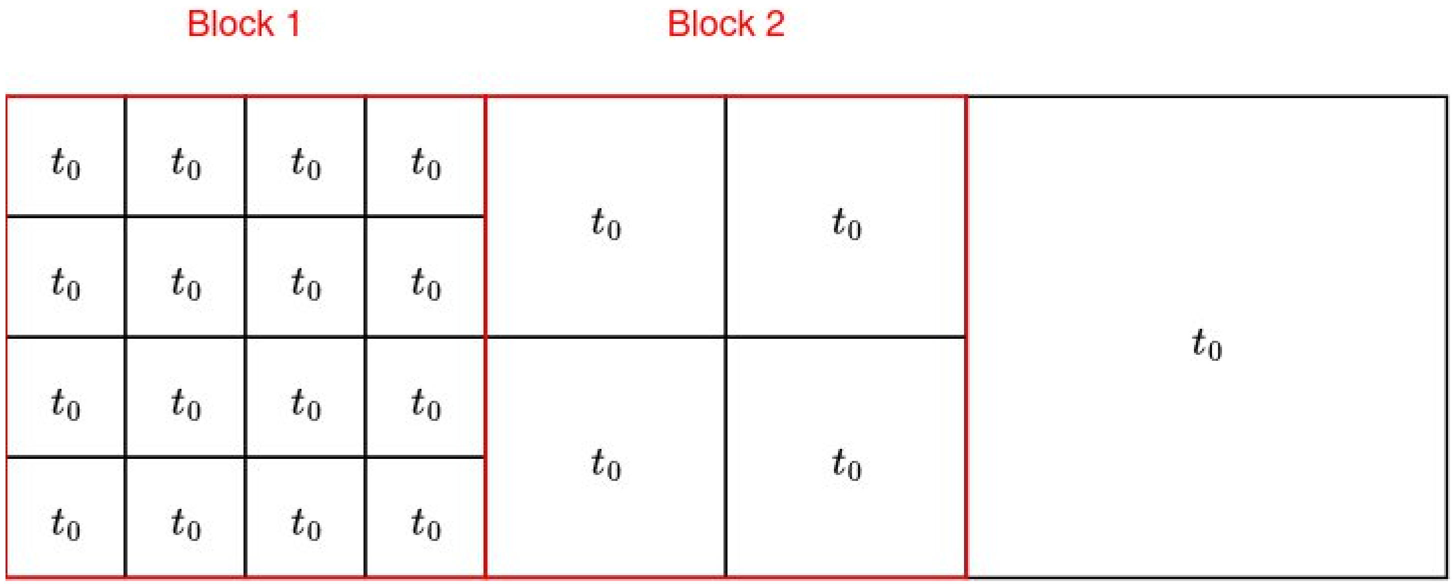
Brio-Wu


Density
Gas pressure
Magnetic field (Y-component)
Athena (reference)
Dyablo
ρ
p
B_y
AMR Level
x
0.0
0.2
0.4
0.6
0.8
1.0
0.5
0.0
−0.5
−1.0
3
4
5
6
7
8

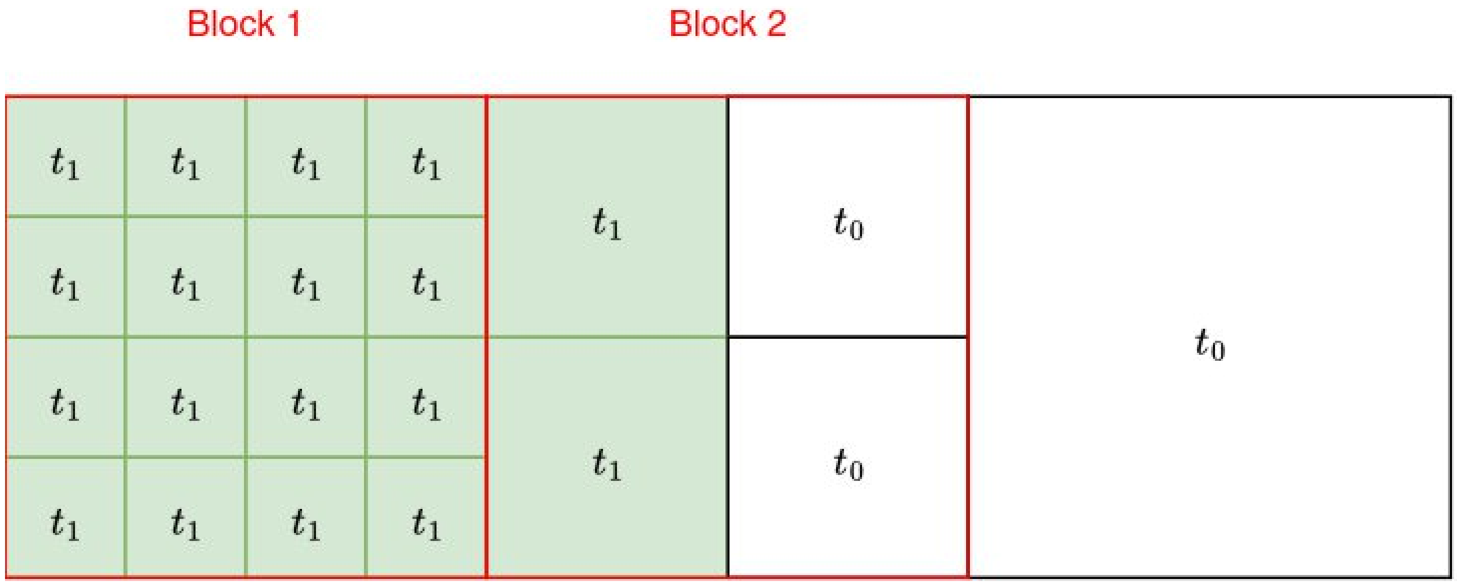
Heat conduction – Rempel et al 2016



Hierarchical Timestepping



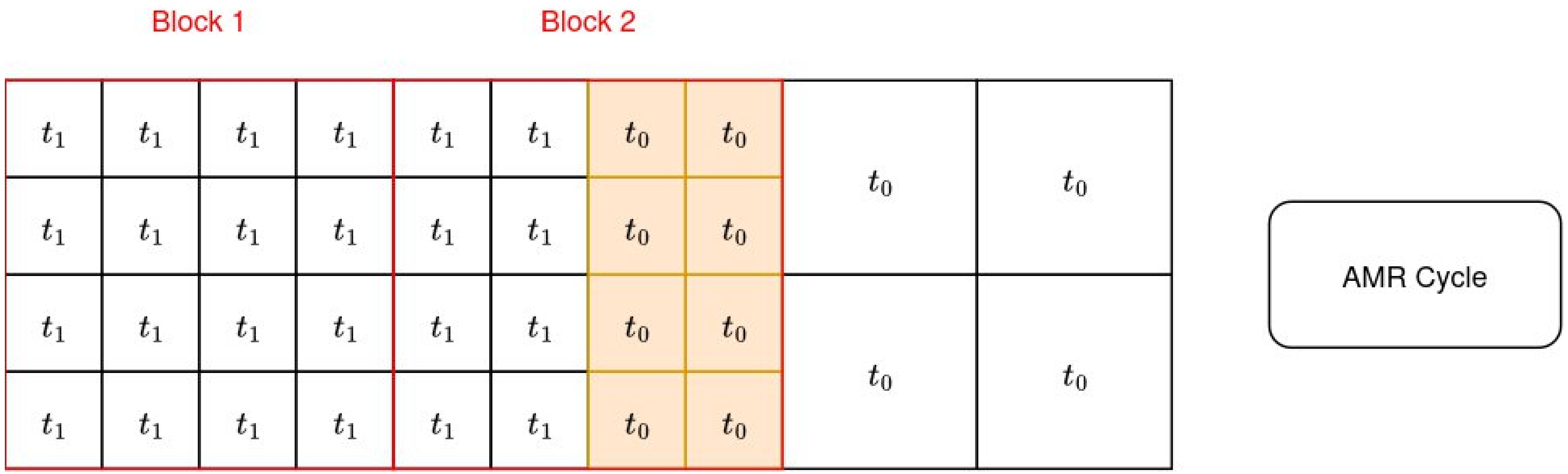
Hierarchical Timestepping



Advancing time

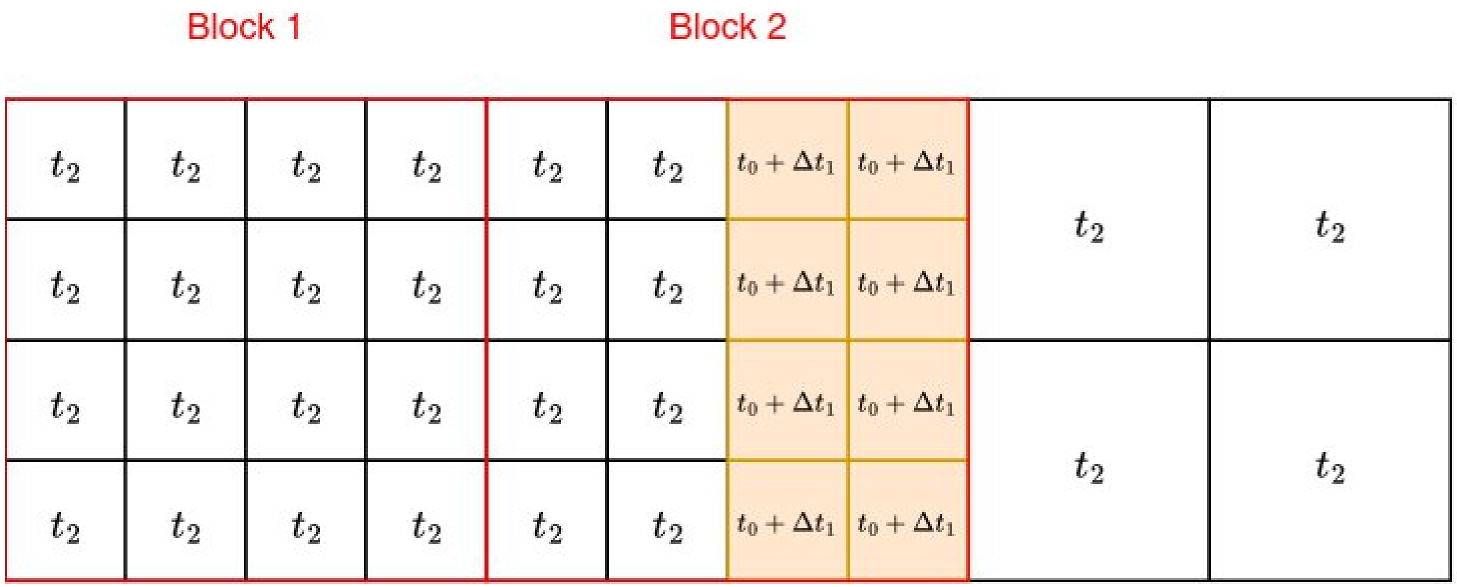
$t_1 = t_0 + \Delta t_0$

Hierarchical Timestepping



t_1	t_1	t_1	t_1	t_1	t_1	t_0	t_0	t_0	t_0
t_1	t_1	t_1	t_1	t_1	t_1	t_0	t_0		
t_1	t_1	t_1	t_1	t_1	t_1	t_0	t_0	t_0	t_0
t_1	t_1	t_1	t_1	t_1	t_1	t_0	t_0		

Hierarchical Timestepping


Block 1
Block 2
t_2 t_2 t_2 t_2 t_2 t_2 $t_0 + \Delta t_1$ $t_0 + \Delta t_1$ t_2 t_2
t_2 t_2 t_2 t_2 t_2 t_2 $t_0 + \Delta t_1$ $t_0 + \Delta t_1$
t_2 t_2 t_2 t_2 t_2 t_2 $t_0 + \Delta t_1$ $t_0 + \Delta t_1$ t_2 t_2
t_2 t_2 t_2 t_2 t_2 t_2 $t_0 + \Delta t_1$ $t_0 + \Delta t_1$

Advancing time

$$t_2 = t_1 + \Delta t_1$$