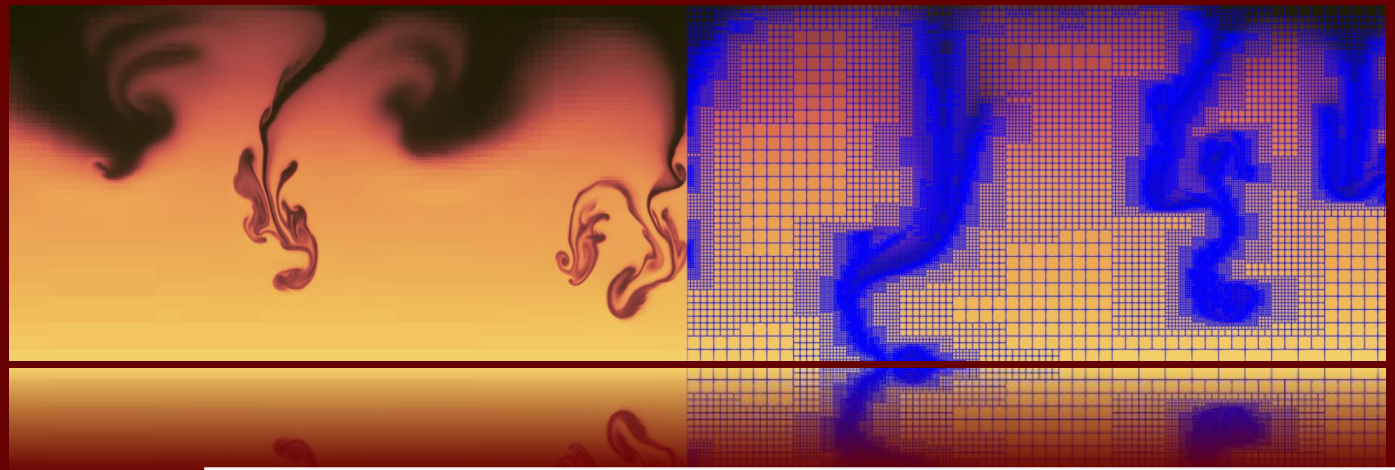
Dyablo

A simulation code for astrophysics fluids on adaptive meshes

Maxime Delorme (maxime.delorme@cea.fr) - CEA DRF/IRFU/DEDIP/LILAS
Arnaud Durocher

Scientific collaborators: Dominique Aubert (ObAS), Catherine Blume (CU-Boulder), Allan-Sacha Brun (CEA), Grégoire Doebele (CEA), Adam Finley (CEA), Olivier Marchal (ObAS)

AMRex at Exascale - Cambridge - 25/06/2024



cea irfu

CEA Saclay - IRFU

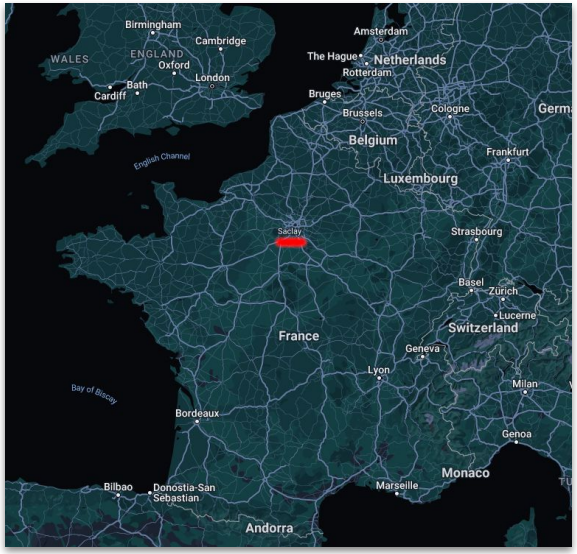
Institute of Research on the Fundamental Laws of the Universe

Fundamental physics departments

- **Astrophysics department (DAp)**
- **Nuclear physics department**
- **Particle physics department**

Engineering departments

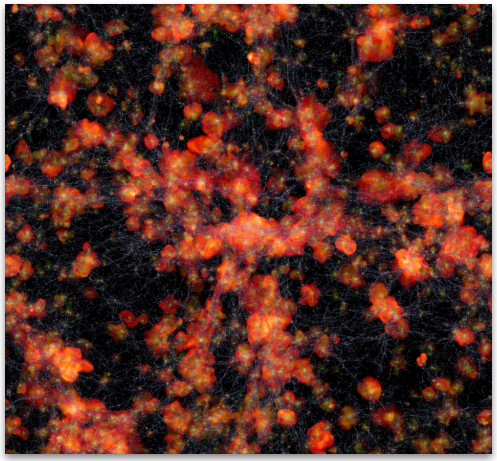
- **Department of Electronics, Detectors and Computer Science for Physics (DEDIP)**
- **Systems Engineering Department**
- **Accelerators, Cryogenics and Magnetism Department**
- **National Large Heavy Ion Accelerator**



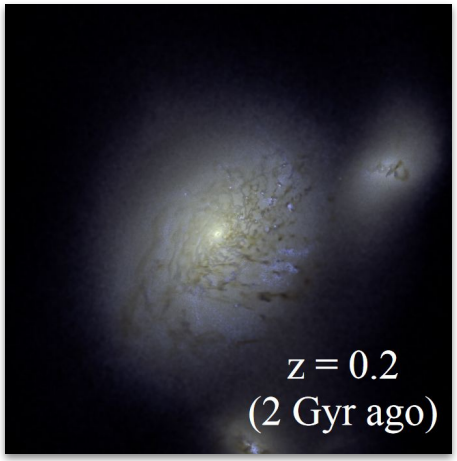
IRFU is ~1000 employees, ~100 PhD students, ~50 post-doctorates

HPC needs for Astrophysics at IRFU

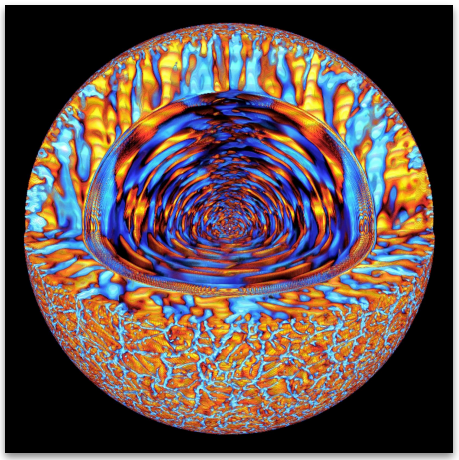
Simulate physical phenomena at every scale



Cosmology
Extreme Horizon
(RAMSES)



Galaxies
(RAMSES)

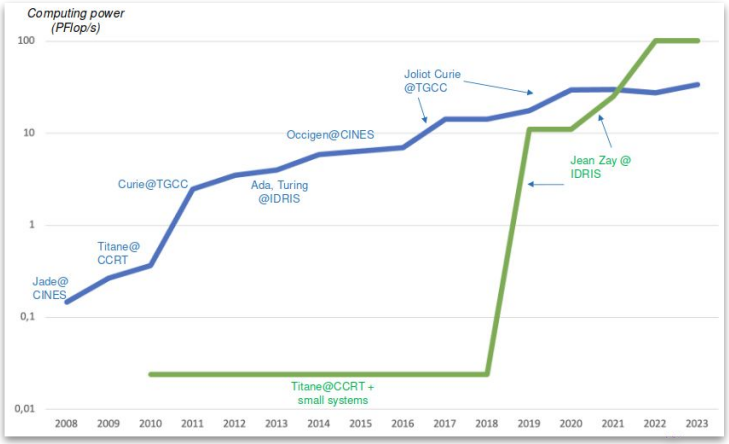


Solar/Stellar
(ASH)

An ever-growing need for computing power to resolve better temporal and spatial scales

Towards Exascale

A diversity of new supercomputer architectures



CPU vs GPU power available in French computing centers on the 2008-2023 period

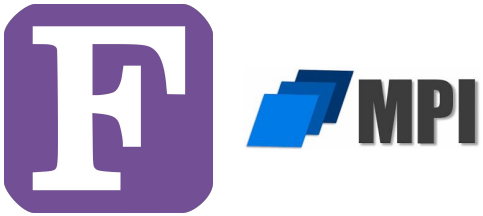
Rank	System	CPU	GPU
1	**Frontier - USA**	AMD	AMD
2	**Aurora - USA**	Intel	Intel
3	**Eagle - USA**	Intel	NVidia
4	**Fugaku - Japon**	Fujitsu	-
5	**LUMI - Finland**	AMD	AMD
6	**Alps - Switzerland**	NVidia	NVidia
7	**Leonardo - Italy**	Intel	NVidia
8	**MareNostrum - Spain**	Intel	NVidia
9	**Summit - United States**	IBM	NVidia
10	**Eos - United States**	Intel	NVidia

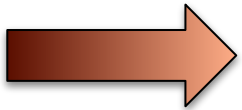
Top 10 world supercomputers (Top 500 / June 2024)

Many different architectures available. New architecture for Exascale are harder to program and require new software stacks.

Dyablo

Replacing the software stack for Exascale at IRFU





Older applications and Exascale

Ex : RAMSES - Failed to port to GPU *(contrat de progres - Idris - 2019)*

- Older languages (Fortran) and programming models
- No shared-memory parallelism (MPI only)
- No software engineering concepts

Needs new software stack and algorithms

Dyablo's software stack

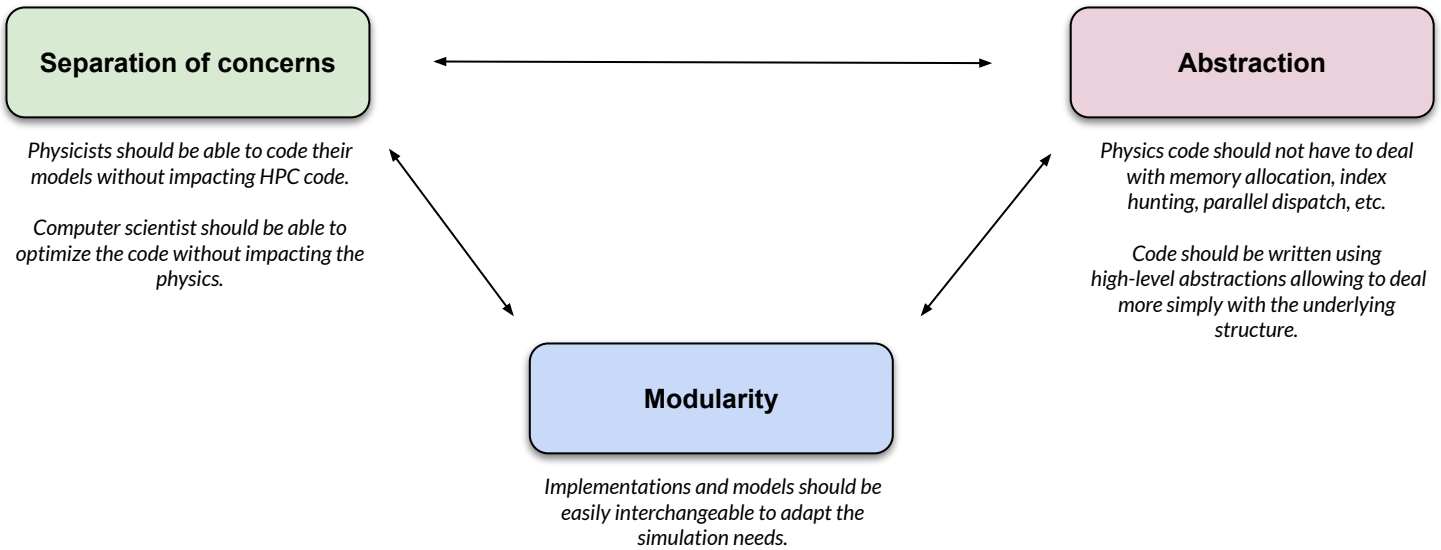
- Written in *C++*, uses external libraries
- *Kokkos + MPI* parallelism
- Built on modern paradigms

Supports Exascale Hardware

Dyablo

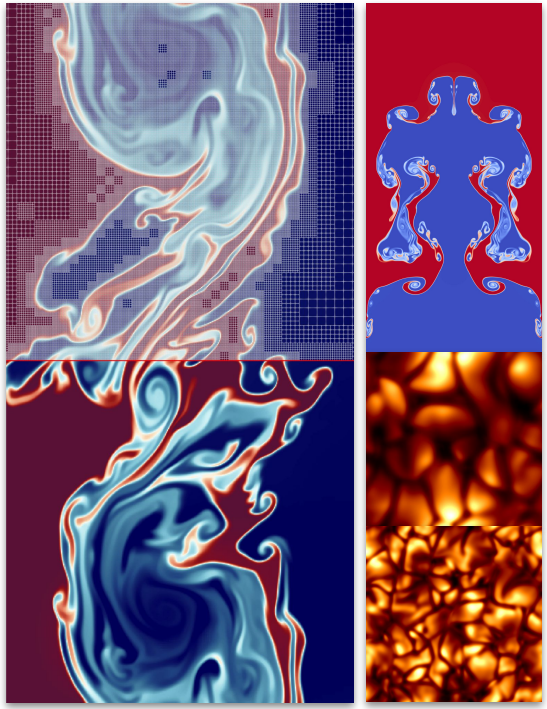
Leverage current software development methods

The three pillars of Dyablo



Features in Dyablo

Address simulation needs for the astrophysical community



Multi-physics simulations

- Hydrodynamics / MHD
- (Self-)Gravity
- Particles
- Radiation
- Diffusion terms: Thermal conduction, Viscosity
- Cosmology: Comoving coordinates

Adaptive Mesh Refinement (AMR)

- Wide range of time/space scales in same simulation
- Block based approach to AMR

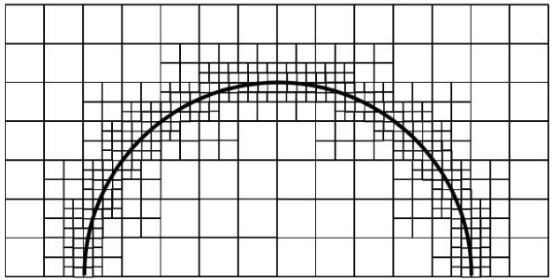
Massively parallel simulations :

- Shared-memory parallelism with *Kokkos* (CPU, GPU, ...)
- Distributed parallelism with MPI

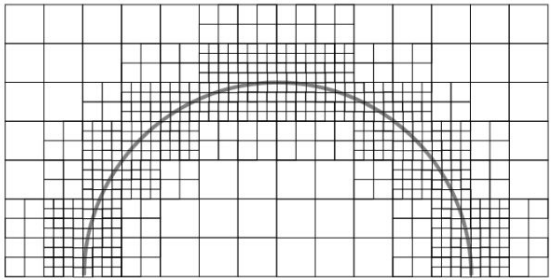
Features in Dyablo will evolve with the specific needs of the involved laboratories

Block-based AMR ?

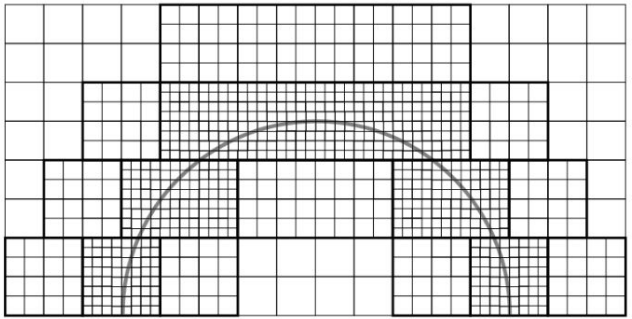
Source: Dunning et al. 2019; "Adaptive Mesh Refinement in the Fast Lane."



Cell-based AMR with 382 cells



Block-based AMR with 596 cells



Patch-based AMR with 836 cells

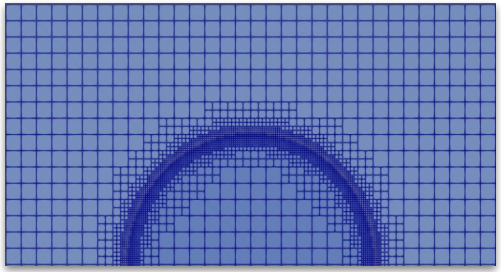
Block-based AMR

- Store cartesian blocks of cells at leaves of the Octree

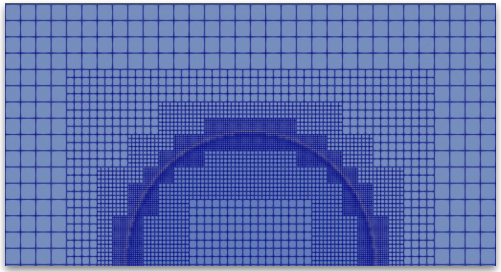
➡ Cartesian grids better for GPU

➡ Octree is smaller : AMR cycle is faster

Cell Based
600k Octants
600k Cells



Block Based
18k Octants
1100k Cells

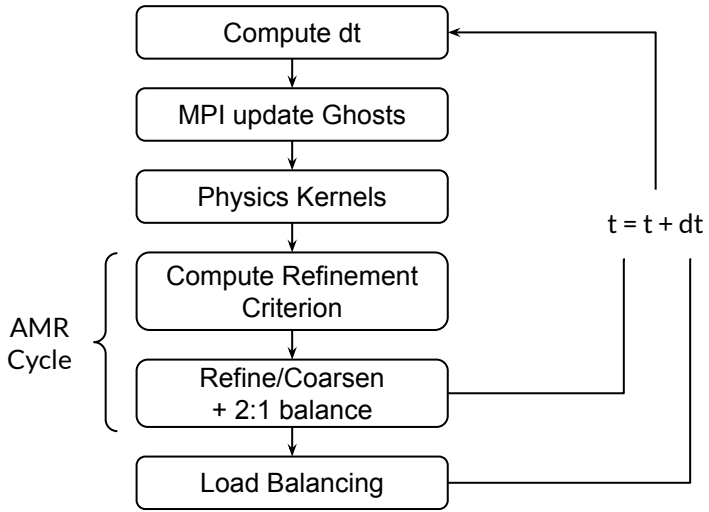


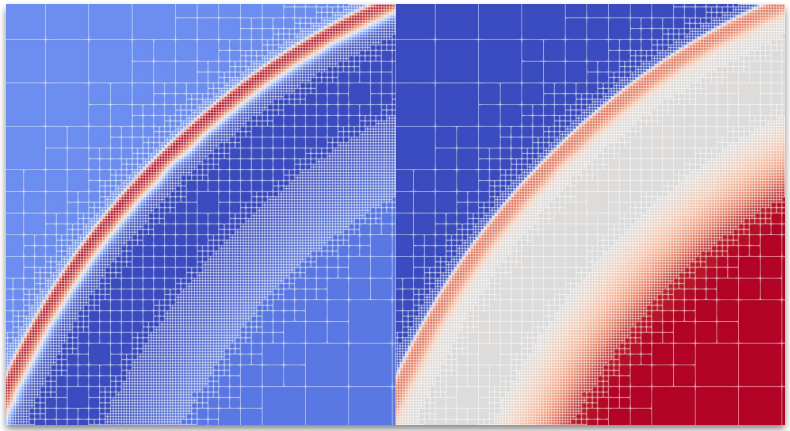
AMR in Dyablo

Adaptive Mesh Refinement

- More resolution in regions of interest
- Octree-based AMR mesh *(cartesian AMR)*
- *Dynamic mesh* changing at every timestep

➡ *AMR cycle may be costly, access patterns are random*





Refined mesh for a Sedov Blast in Dyablo

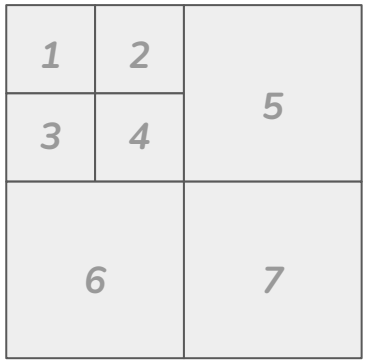
Block-based AMR :

- Every octant is refined in 2 in each direction
- The leaves of the tree are regular cartesian subgrids
- Typical block-sizes 4 - 8 - 16

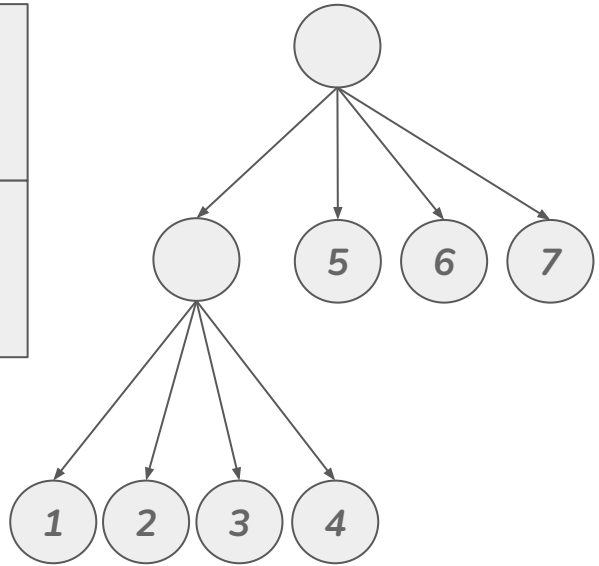
AMR on GPU in Dyablo

GPU Data Structures for the AMR Octree

AMR mesh



Associated Octree



Storing and updating the AMR Octree

- Chained structures not efficient on GPU
- Neighbors must be close in memory

➡ Fields are stored in arrays (`Kokkos::View`)

➡ Cells are stored in Morton Order (Z-curve)

Accessing neighborhood

- "Linear octree"
- Using hashmap to find neighbors (`Kokkos::UnorderedMap`)

Modularity : 2 AMR backends

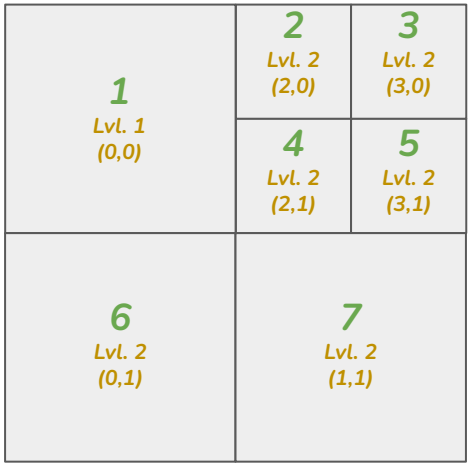
- PABLO: 3rd party CPU only library
  - 2 Octree representations for CPU/GPU (+translations)
- Dyablo: our own backend based on Kokkos
  - GPU compatible, more flexibility
- *More possible (AMReX backend ???)*

AMR on GPU in Dyablo

Finding neighbors

Unstructured linear tree

- **index** : of the cell in **Morton** order (Z-curve)
- **position** : refinement level and position on the regular grid at this level
- **Convert : index -> position** : array of positions
- **Convert : position** -> **index** : hashmap



AMR mesh

Hashmap

*Key/value container that able to "quickly" (O(1)) a value (**index**) associated to the key (**position**)*

- `Kokkos::UnorderedMap`
- Key : **position**; Value : **index**

Request a neighbor from an index:

1. **index -> position** (Array)
2. Arithmetics on **position**
   (neighbor could be at a different level)
3. Neighbor's **position** -> Neighbor's **index**

Left of *4* :

1. *4* **->** ***Lvl. 2 : (2,1)***
2. On the right : ***Lvl. 2 : (2-1,1)***
3. ***Lvl. 2 : (1,1)* does not exist:**
   We look at level 1 : ***Lvl. 1 : (0,0)***
4. ***Lvl. 1 : (0,0)*** -> **1**

AMR on GPU in Dyablo

Write AMR Kernels

Kernels are written using abstract interfaces :

- User friendly and readable by *normal* humans
- Optimization possible without changing kernel code

Apply function on each cell :

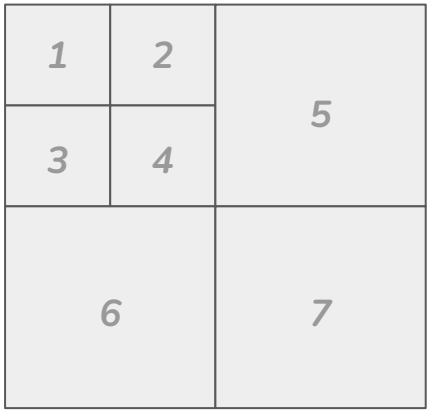
`foreach_cell()`

- Lambda-based loop
- Hide Kokkos

Access data :

`CellArray, CellIndex`

- Hide mem. Layout
- Hide index computation
- AMR neighbor access




```
double dt;
ForeachCell foreach_cell(...);
FieldManager field_manager({IP,IDPDX});
CellArray_ghosted U = foreach_cell.allocate_ghosted_array(
foreach_cell.foreach_cell(
    "compute_pressure_gradient",
    U,
    CELL_LAMBDA(const CellIndex& iCell_U)
{
    double P_left = 0;
    CellIndex iCell_Uleft = iCell_U.getNeighbor({-1,0,0});
    if( iCell_Uleft.level_diff() >= 0 )
        double P_left = U.at(iCell_Uleft, IP);
    else
    {
        int nbCells = foreach_sibling<ndim>(
          iCell_Uleft, U,
          [&](const CellIndex& iCell_subcell)
        {
          P_left += U.at(iCell_subcell, IP);
        });
        P_left = P_left/nbCells;
    }
    double P_right;
    [...]
    U.at(iCell_Uout, IDPDX) = (P_right - P_left) / h;
});
```

AMR on GPU in Dyablo

Write AMR Kernels

Kernels are written using abstract interfaces :

- User friendly and readable by *normal* humans
- Optimization possible without changing kernel code

Apply function on each cell :

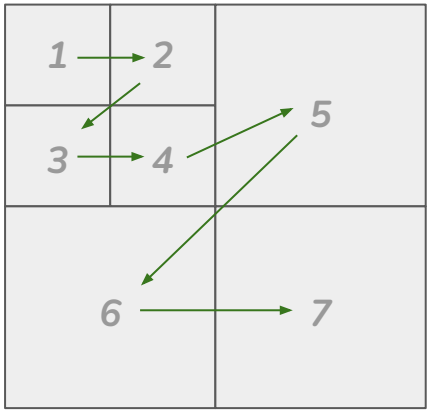
`foreach_cell()`

- Lambda-based loop
- Hide Kokkos

Access data :

`CellArray, CellIndex`

- Hide mem. Layout
- Hide index computation
- AMR neighbor access




```
double dt;
ForeachCell foreach_cell(...);
FieldManager field_manager({IP,IDPDX});
CellArray ghosted_U = foreach_cell.allocate_ghosted_array(
foreach_cell.foreach_cell(
    "compute_pressure_gradient",
    U,
    CELL_LAMBDA(const CellIndex& iCell_U)
{
    double P_left = 0;
    CellIndex iCell_Uleft = iCell_U.getNeighbor({-1,0,0});
    if( iCell_Uleft.level_diff() >= 0 )
        double P_left = U.at(iCell_Uleft, IP);
    else
    {
        int nbCells = foreach_sibling<ndim>(
          iCell_Uleft, U,
          [&](const CellIndex& iCell_subcell)
        {
          P_left += U.at(iCell_subcell, IP);
        });
        P_left = P_left/nbCells;
    }
    double P_right;
    [...]
    U.at(iCell_Uout, IDPDX) = (P_right - P_left) / h;
});
```

AMR on GPU in Dyablo

Write AMR Kernels

Kernels are written using abstract interfaces :

- User friendly and readable by *normal* humans
- Optimization possible without changing kernel code

Apply function on each cell :

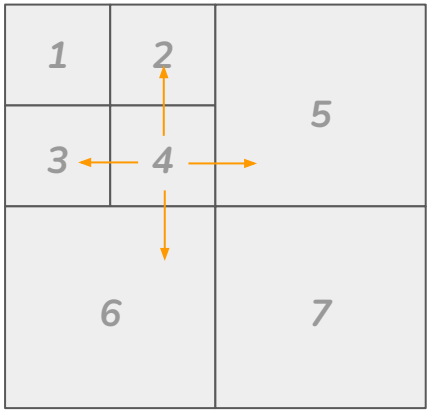
`foreach_cell()`

- Lambda-based loop
- Hide Kokkos

Access data :

`CellArray, CellIndex`

- Hide mem. Layout
- Hide index computation
- AMR neighbor access




```
double dt;
ForeachCell foreach_cell(...);
FieldManager field_manager({IP,IDPDX});
CellArray_ghosted U = foreach_cell.allocate_ghosted_array(
foreach_cell.foreach_cell(
    "compute_pressure_gradient",
    U,
    CELL_LAMBDA(const CellIndex& iCell_U)
{
    double P_left = 0;
    CellIndex iCell_Uleft = iCell_U.getNeighbor({-1,0,0});
    if( iCell_Uleft.level_diff() >= 0 )
        double P_left = U.at(iCell_Uleft, IP);
    else
    {
        int nbCells = foreach_sibling<ndim>(
          iCell_Uleft, U,
          [&](const CellIndex& iCell_subcell)
        {
          P_left += U.at(iCell_subcell, IP);
        });
        P_left = P_left/nbCells;
    }
    double P_right;
    [...]
    U.at(iCell_Uout, IDPDX) = (P_right - P_left) / h;
});
```

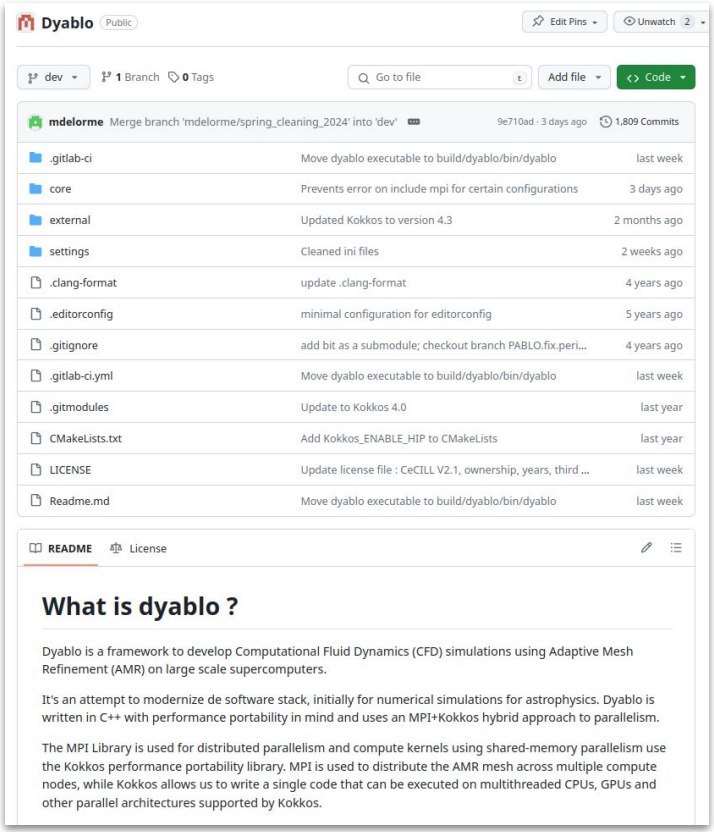
Development process

Versioning
- Code is versioned using git, on a dedicated gitlab server at CEA (accessible upon request)

Testing/CI
- Unit-tests written with *gtest*
- Validation tests produce artifacts
- Pipeline run automatically for every merge-request proposal

Development process with physicists
- Physicists try to implement their modules in dyablo
- Feedback on the interfaces, API, and capabilities of dyablo are sent back to us. Either in person or vis slack
- New interfaces are proposed/modified
- When convergence is reached, MR is proposed, code is reviewed and integrated into *dev* branch

We are now live on github !!!

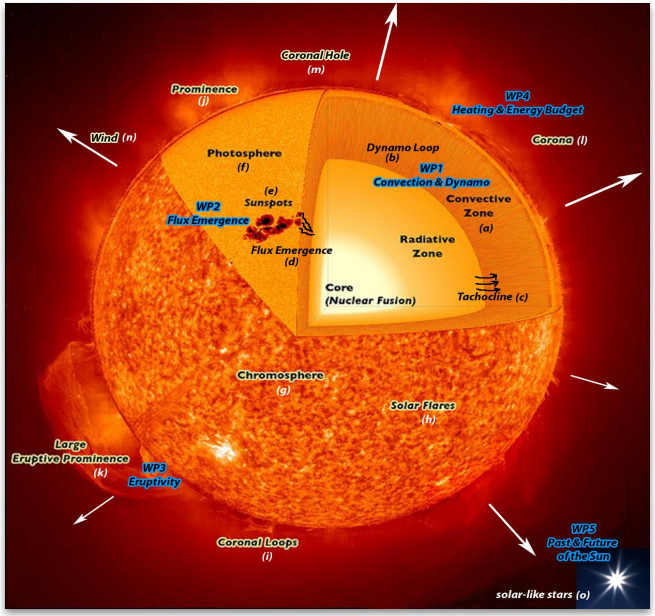


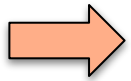
Mirror of the development branch + tagged versions

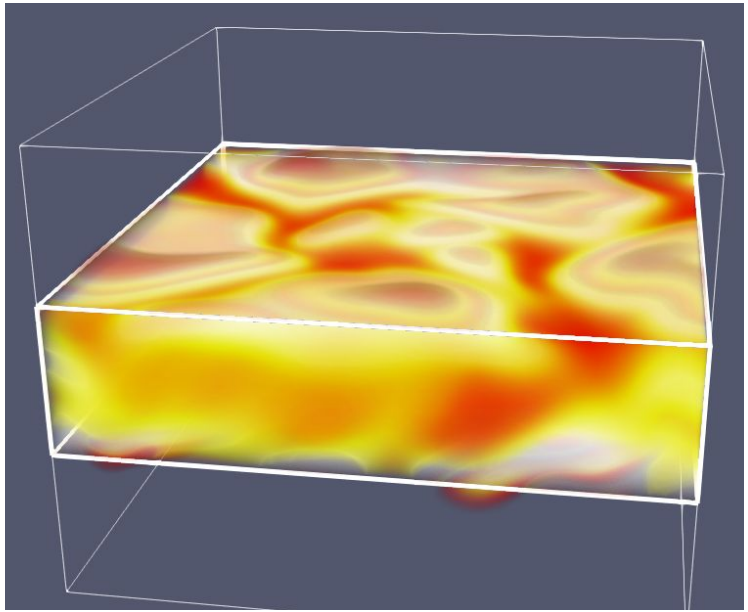
https://github.com/Dyablo-HPC/Dyablo

Application #1

Solar convection benchmark towards whole sun simulations *(DAp ERC-Synergy Whole Sun)*





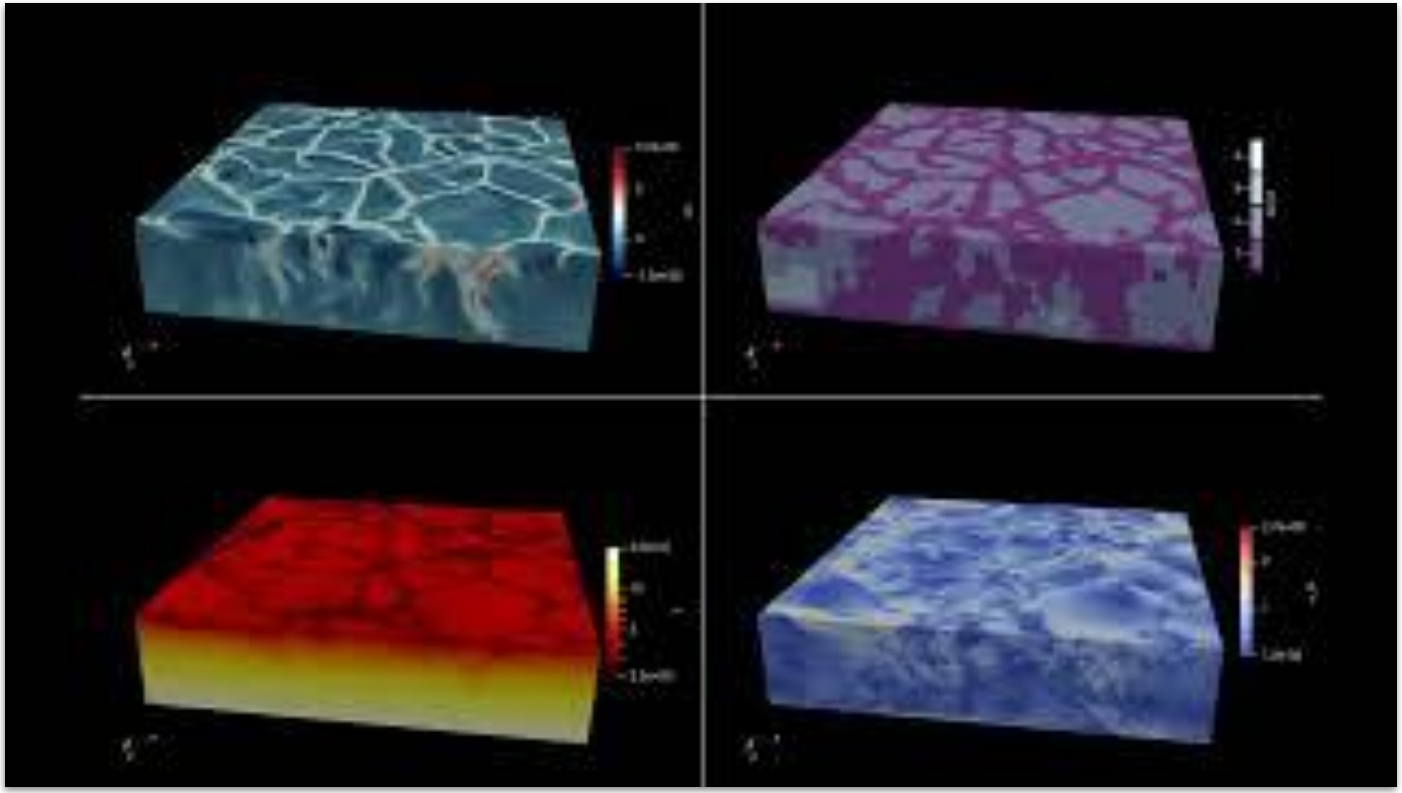


Credit: [Whole Sun website]

Solar convection

Without stable layers

Ran on 4 Nvidia a100

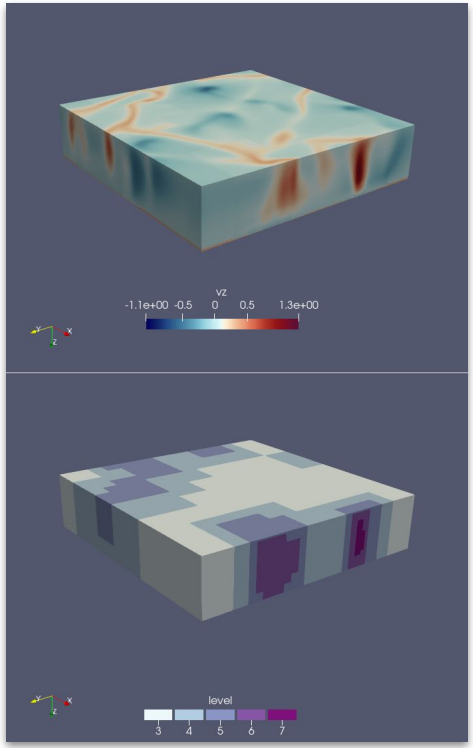


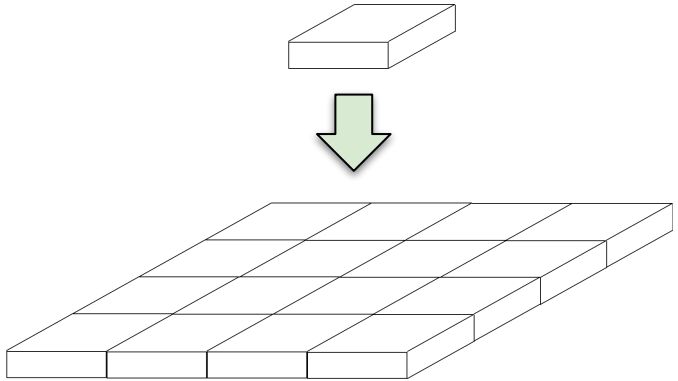
Weak scaling benchmarks

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
  - CPU : CSL (JZ), Genoa (AA)
  - GPU : v100 (JZ), a100 (JZ), MI250X (AA)
  - Tested up to 2048 GPUs ~62 billion cells

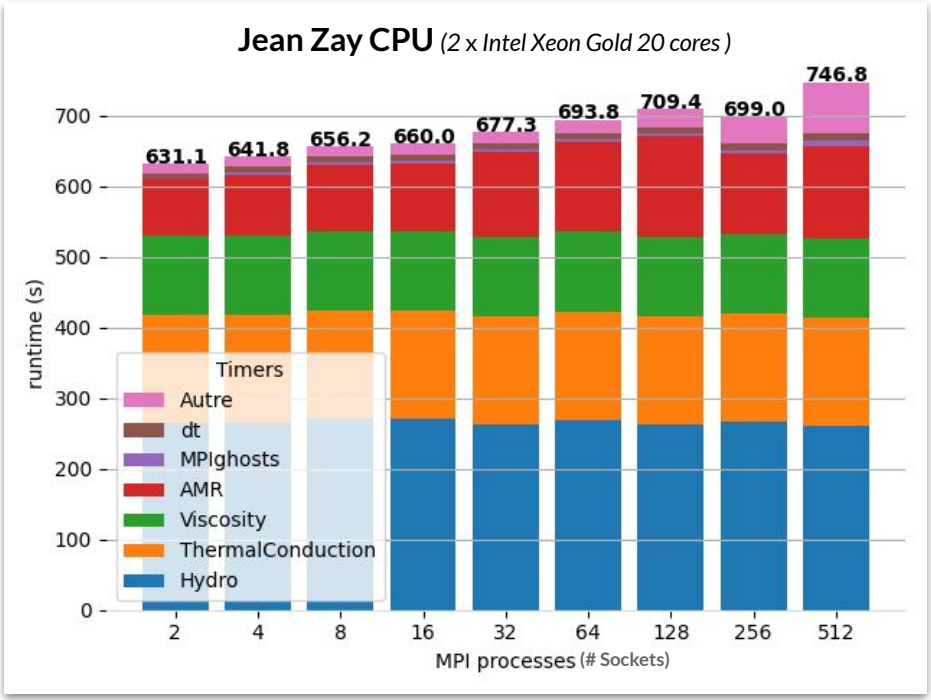


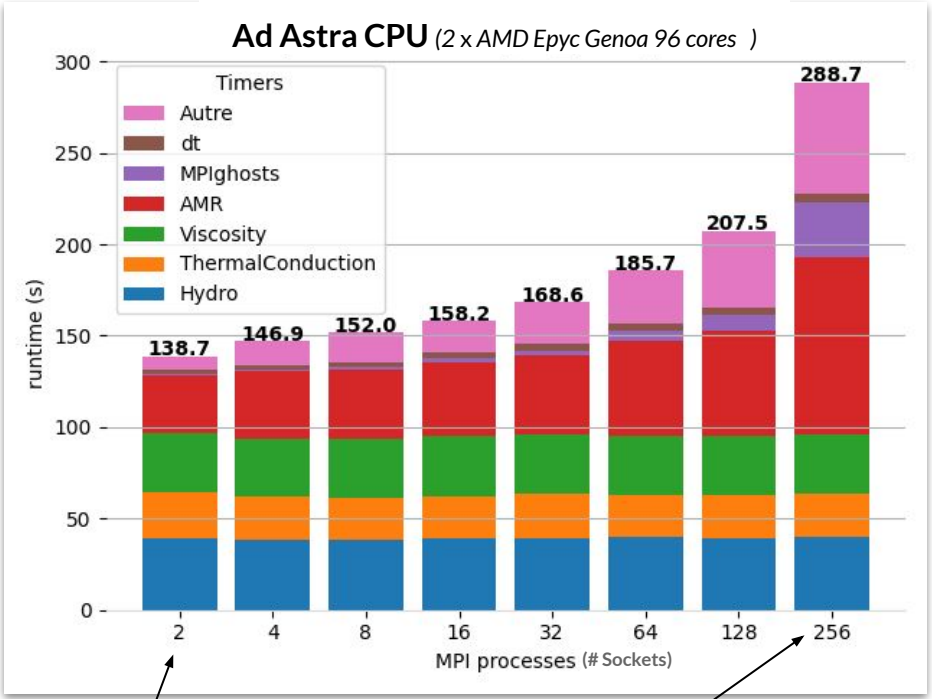


Replication on N MPI processes

Weak scaling benchmarks (11/23)

CPU results



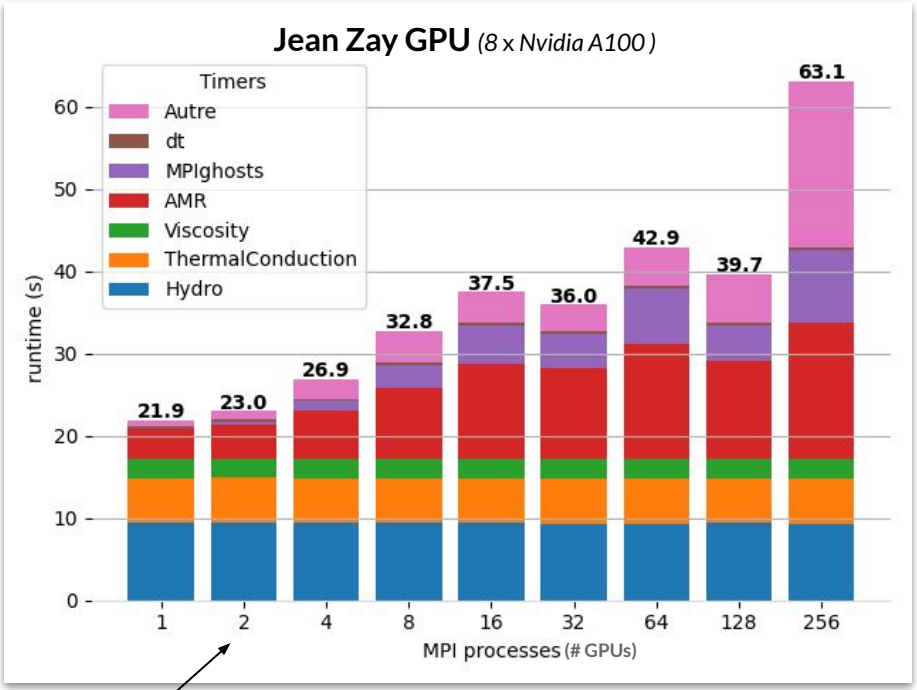


X4.5 compared to JZ CPU

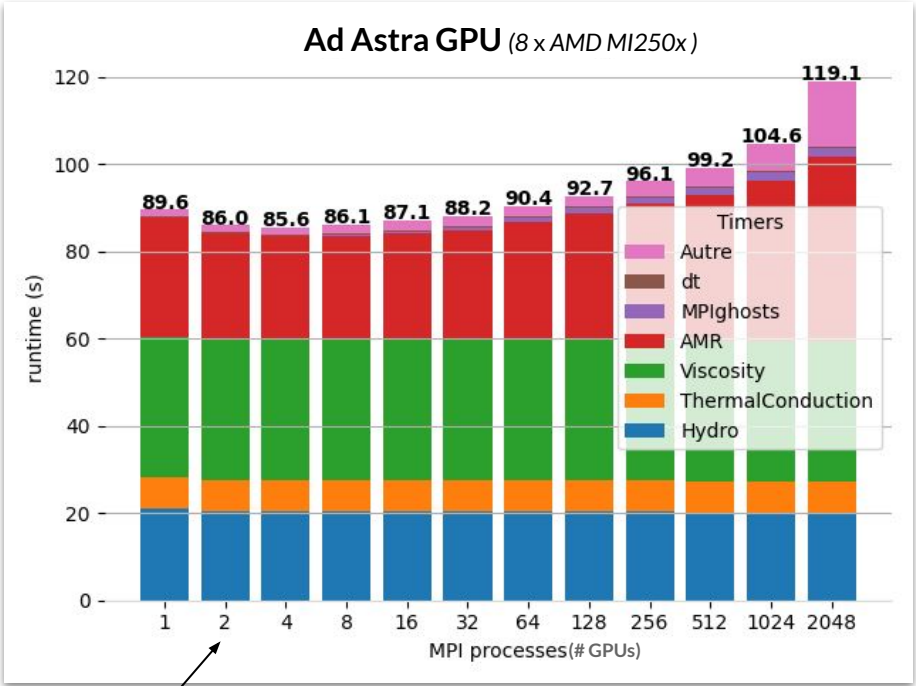
X2.3 compared to JZ CPU

Weak scaling benchmarks (11/23)

GPU results

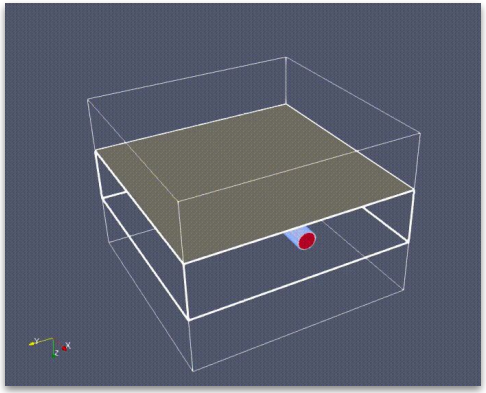


X27.4 compared to JZ CPU

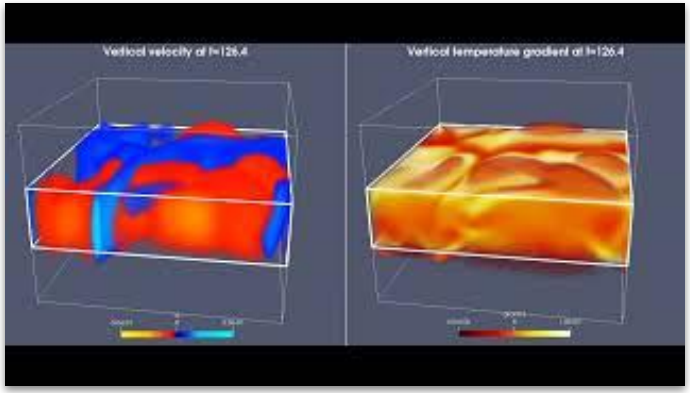


X7.3 compared to JZ CPU

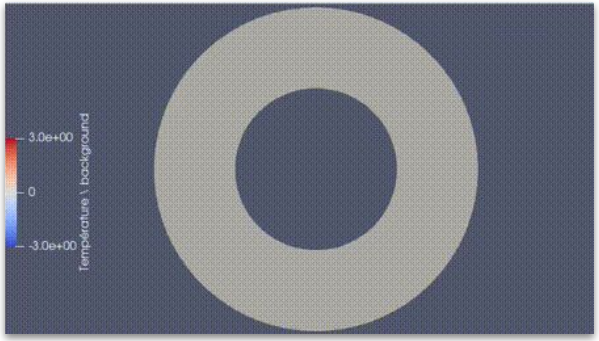
More convection runs !



Flux tube emergence
(with C. Blume)



Three-layer convection
(with A.S. Brun and A. Finley)



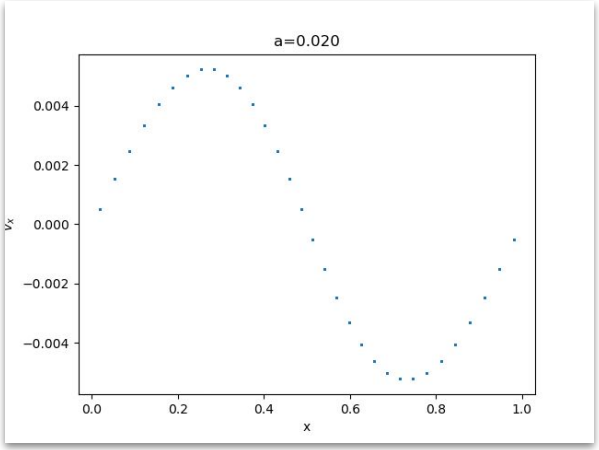
Convection in a ring
(Grégoire Doebele's PhD)

Observatoire astronomique de Strasbourg

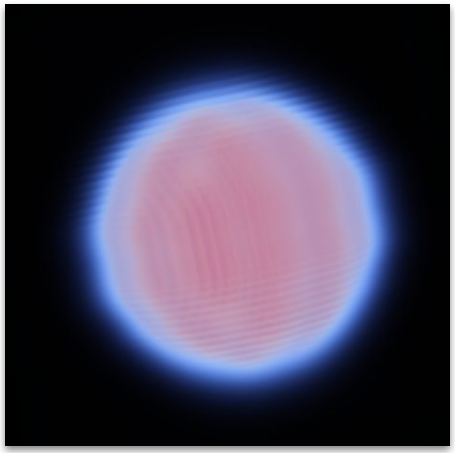
Application #2

Cosmological radiative transfer *(ObAS - Dominique Aubert, Olivier Marchal)*

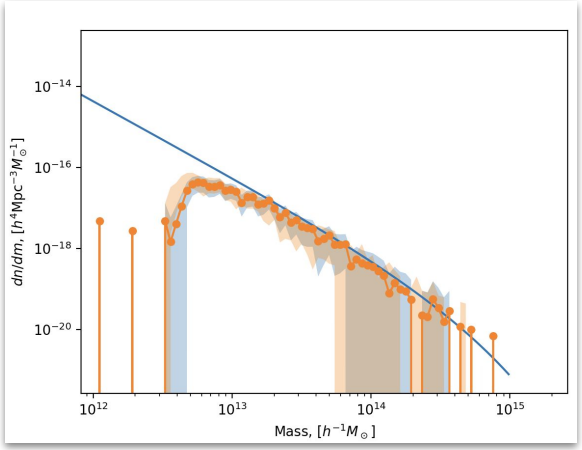
Validation runs



Zeldovitch' Pancake



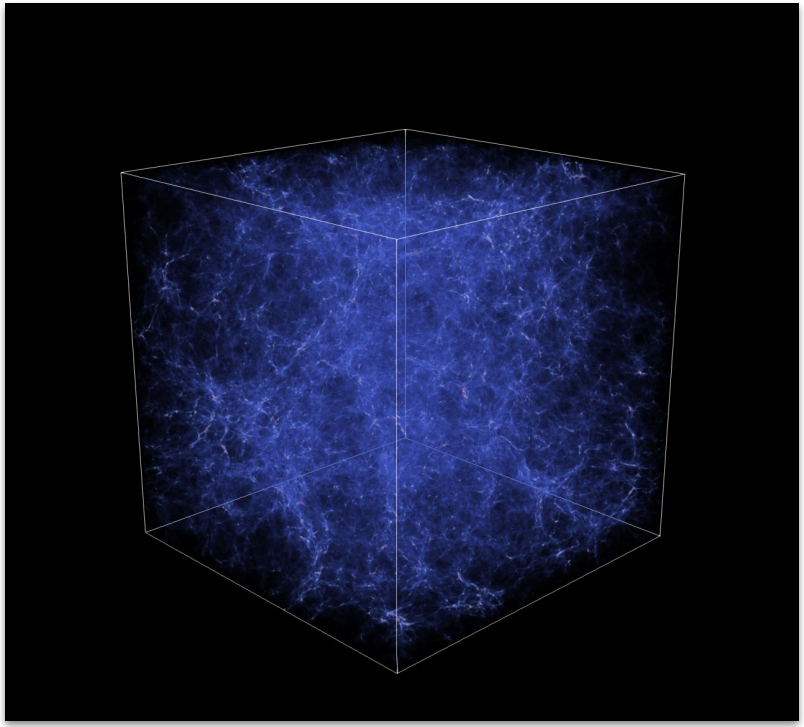
Stromgren's sphere



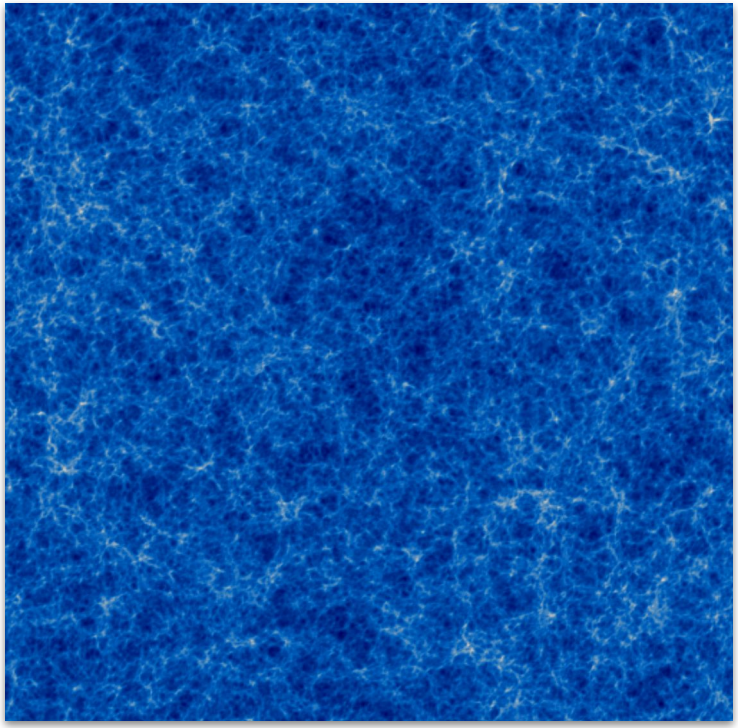
Halo mass function on small test run

Observatoire astronomique de Strasbourg

Cosmological simulations with RT



Box of 512^3 points of width 128 Mpc at z=5.5; 3 active refinement levels
Ran on Irene, 16 MPI processes, on AMD Rome with 64 cores each.



Slice of a run of 1024^3 points of width 128 Mpc at z=9.5; 3 active refinement levels
Ran on Irene, 128 MPI processes on AMD Rome with 64 cores each.

Dyablo - Projects, community and collaborations

Physics/Applicative projects :

- **Whole Sun :** ERC Synergy on solar physics
- **GINEA :** Groupement d'Instrumentation Numérique pour l'Exascale en Astrophysique (GT CNRS) -> Cosmology, Galaxy formation
- **PEPR Origins :** "From the formation of planets to life" -> 4 postdocs/PhD students potentially working on dyablo for the implementation of new physics/methods.
  - One PhD working on dust, two PhDs planned on cosmic-rays and visualization
  - One postdoc planned on gravity solvers

HPC/Computer science projects

- **EUPEX :** European Pilot for Exascale -> Porting the code to ARM architectures
- **CExA :** Exascale at CEA -> 1 postdoc confirmed to work on optimization tools and methods for Kokkos
- **PEPR Numpex** *(Demonstrator PC5 Exa-DI)*
  - IO / Visualization tools for AMR *(1 postdoc PC3 Exa-Dost)*
  - Implicit methods / linear solvers on AMR grids
  - Load balancing
  - ...

Roadmap 2024

Core developments :

- **Local time-stepping**
- Units
- **Geometry**
- ***Refactoring of solver policies***
- Logging

Post-treatment and analysis

- Python back-end

Publication and dissemination

- Method paper
- Solar convection paper
- **Open sourcing the code**
- Documentation

Performances

- Small grain CPU and GPU profiling
- Kernel optimization
- Tuning

PEPR Origins :

- Dust
- *Non ideal MHD*
- Constrained transport
- Gravitation

Whole Sun :

- Multi-layered convection
- Flux tube experiments

EUPEX :

- Profiling SVE and HBM
- Profiling on Grace Hopper

CExA :

- **PTC-SN on tuning, automatic kernel extraction and optimization**

PEPR NumPEX (PC3) :

- IO formats for AMR
- Data compression
- *PDI/DASK/DEISA integration*

PEPR NumPEX (PC5) - Propositions

- IOs and visualization formats and tools
- Implicit methods for diffusion operators
- Load balancing
- Kernel performance for all architectures

Additional slides

Dyablo

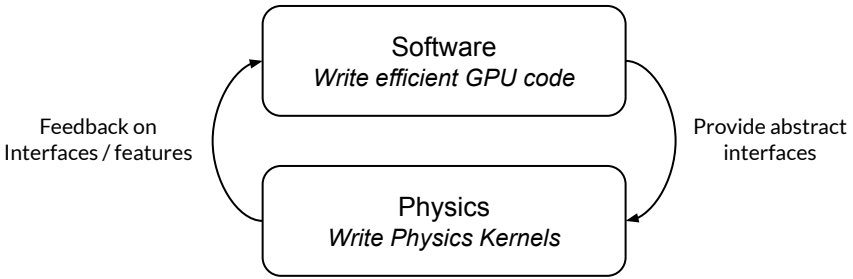
Leverage current software development methods

Development of older simulation codes

- One-man codes : physicists also optimize code
- Code from scratch : not leveraging libraries
- Physical model are becoming more complicated
- Code is harder to optimize (new architectures)

➡ **Need “separation of concerns”**

Code is written by code experts and physics kernels are written by physicists



Dyablo’s development organization

- *Modular* : plugins for kernels, IOs, …
- Uses *abstract interfaces* to separate optimization details from physics kernels

Encourage collaboration :

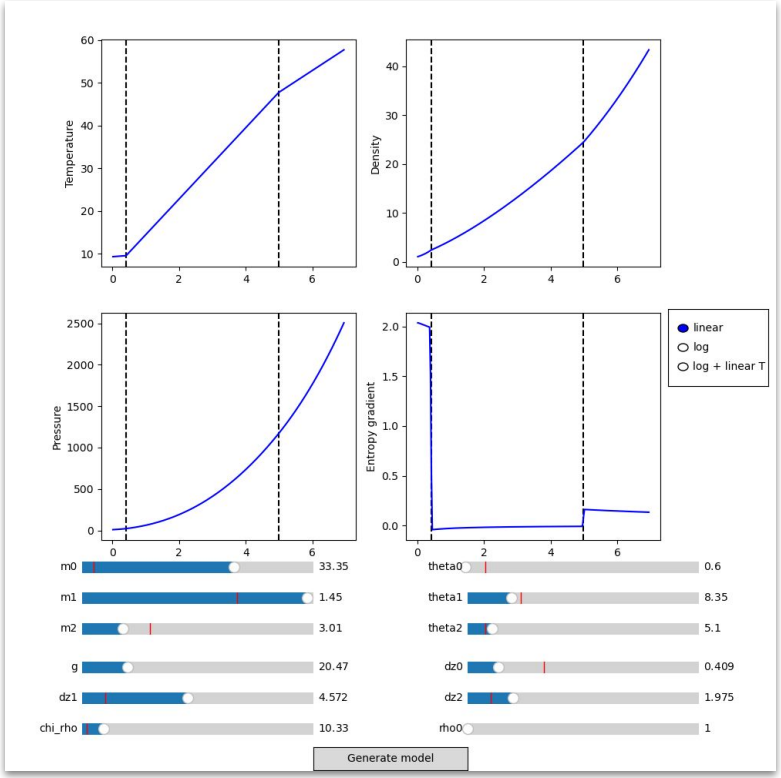
- *Software development / support* *(CEA DEDIP)*
  - Write abstract interfaces perform operations on the AMR mesh
  - Optimize behind the scene algorithms
- *Physics labs :* (ex: CEA DAp Whole-Sun, …)
  - Write physics kernels using this interface
  - Create applications based on dyablo
  - Provide feedback for the software dev. team

Tri-Layer setup

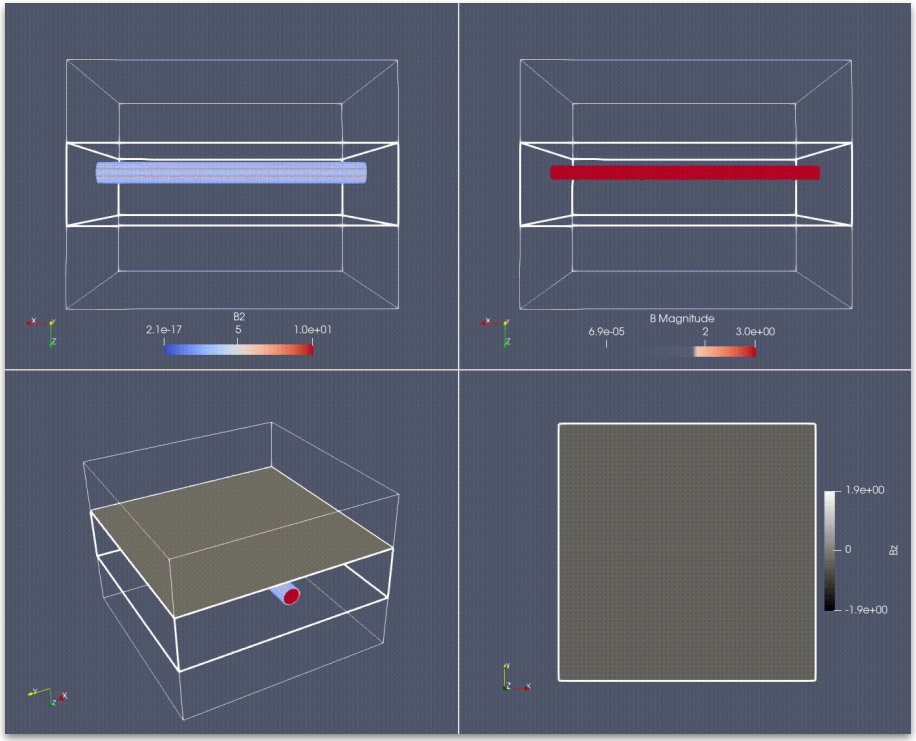
$$T(z) = T_i + \theta_i(z - z_i)$$

$$\rho(z) = \rho_i \left(\frac{T(z)}{T_i} \right)^{m_i}$$

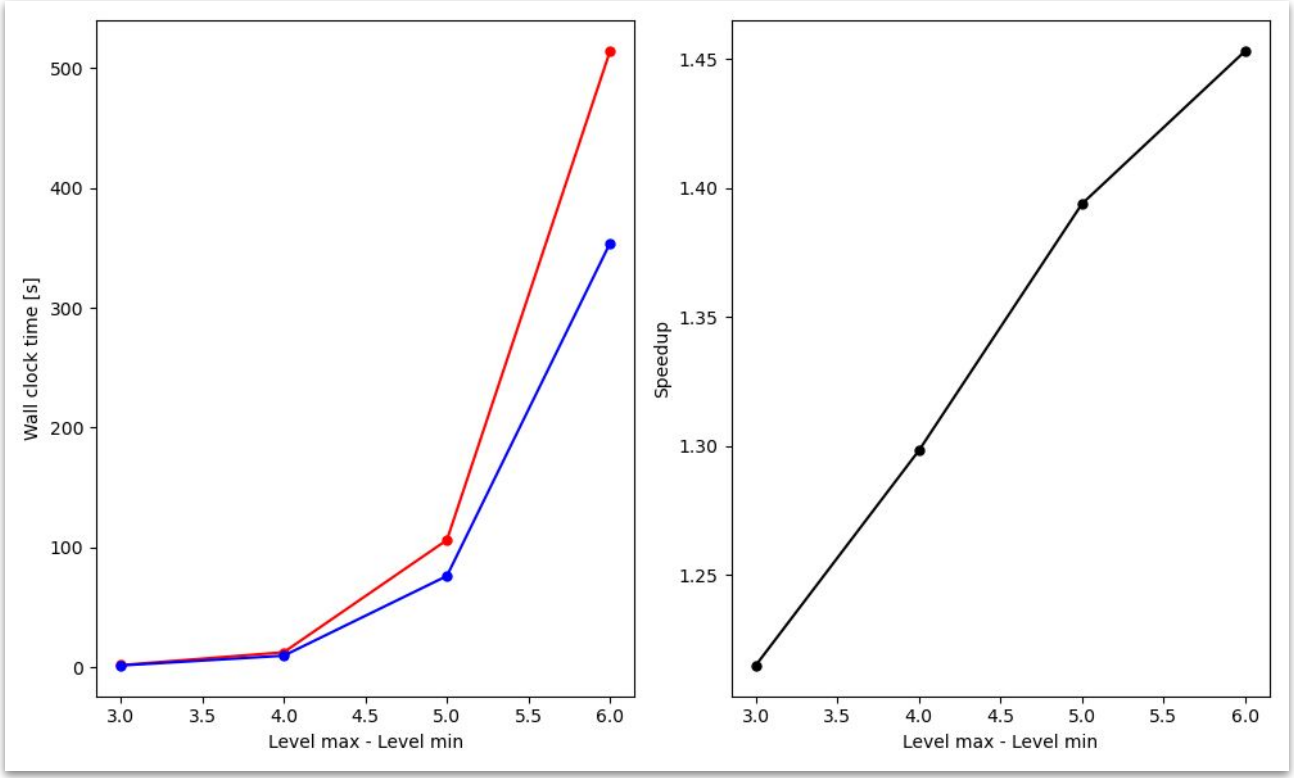
$$P(z) = P_i \left(\frac{T(z)}{T_i} \right)^{m_i+1}$$



Flux tube emergence



Local time stepping - Proof of concept



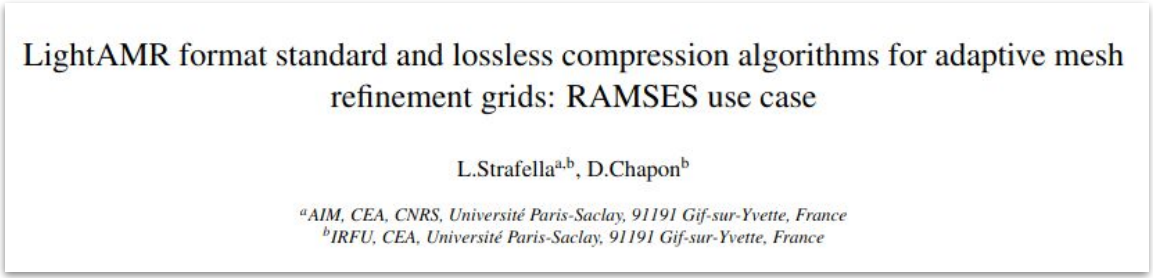
File formats and IOs

The problem

- No actual standard for AMR formats
- Currently Dyablo uses unstructured grid hdf5 + xdmf format
    - Extremely expensive to write, to read and to store
- No compression applies
- Post-treatment is close to nightmarish

The solution

- Implementing LightAMR format in Dyablo
- LightAMR is extremely efficient
- … but dedicated to cell-based AMR and thus requires adaptation
- Interconnexion with HyperTreeGrid necessary as well
- -> PEPR Numpex - PC3 -> One postdoc to work on that.



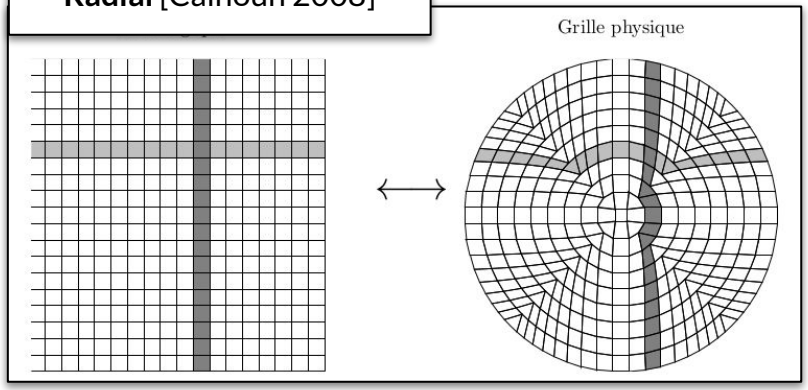
LightAMR format standard and lossless compression algorithms for adaptive mesh refinement grids: RAMSES use case

L.Strafella[a,b], D.Chapon[b]

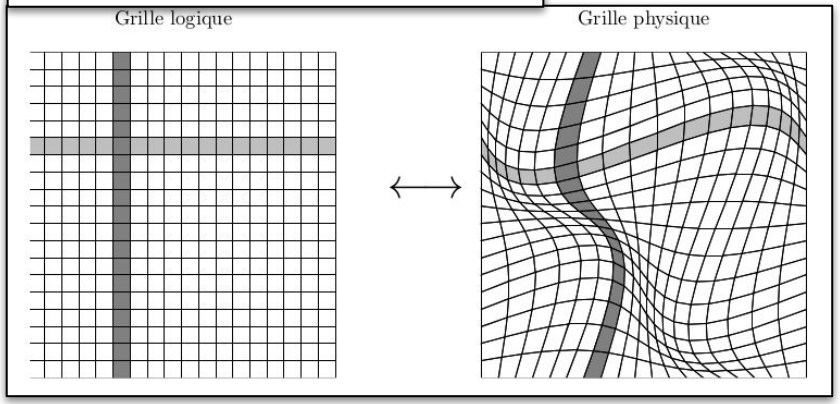
[a]AIM, CEA, CNRS, Université Paris-Saclay, 91191 Gif-sur-Yvette, France
[b]IRFU, CEA, Université Paris-Saclay, 91191 Gif-sur-Yvette, France

Geometry

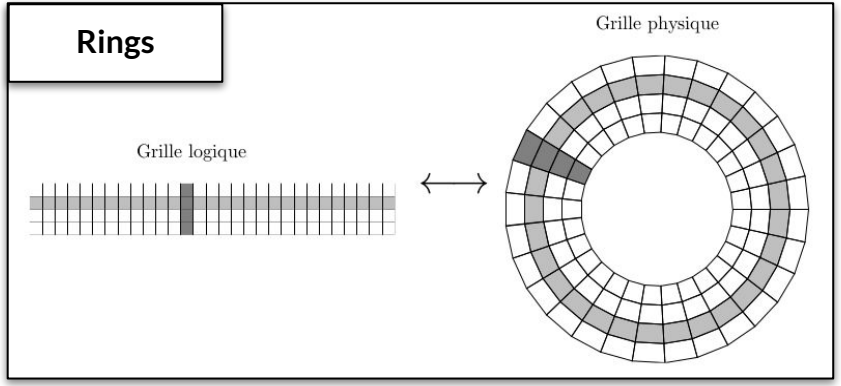
Radial [Calhoun 2008]



Warped cartesian[Persson 2009]



Rings



Geometry

Resolution method

- Data lives in physical space
- To solve Riemann problem :
  - All vector quantities are rotated in logical space
  - Cell properties are recomputed in logical space
  - Riemann problem is solved traditionally
  - Vector quantities are rotated back into physical space