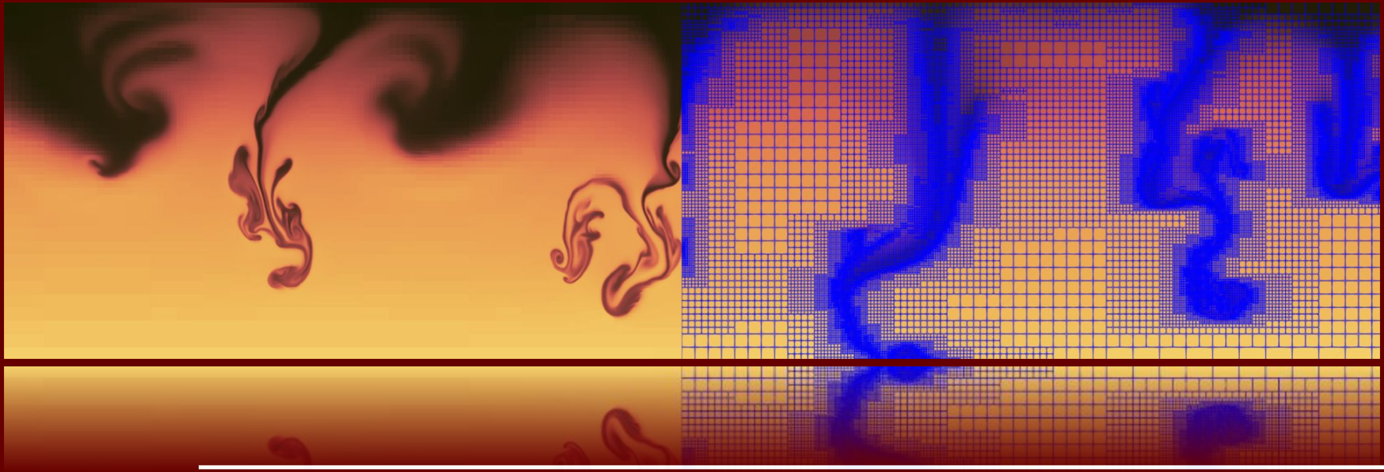
Dyablo

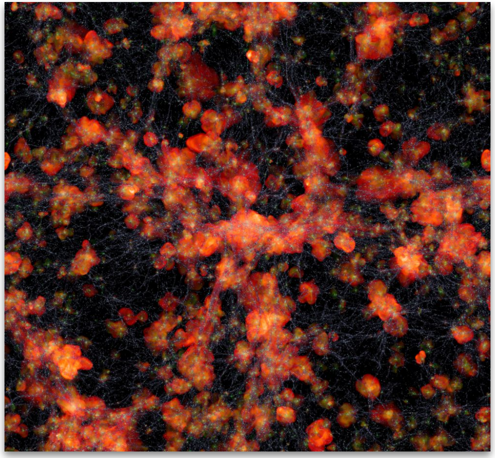
A new hardware-agnostic AMR code for Exascale

Arnaud Durocher (arnaud.durocher@cea.fr)
Maxime Delorme (maxime.delorme@cea.fr) - *CEA DRF/IRFU/DEDIP/LILAS*
Numpex - AMR Workshop - 06/02/2024



HPC needs for Astrophysics

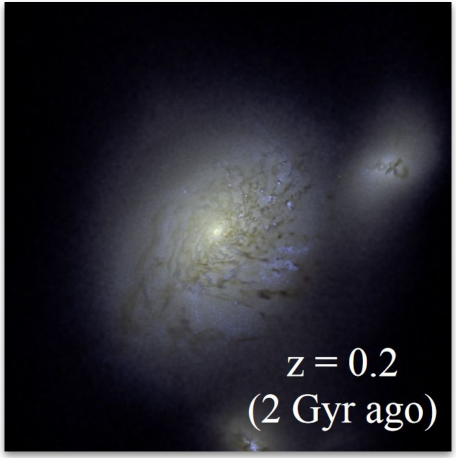
Simulate physical phenomena at every scale



Cosmology

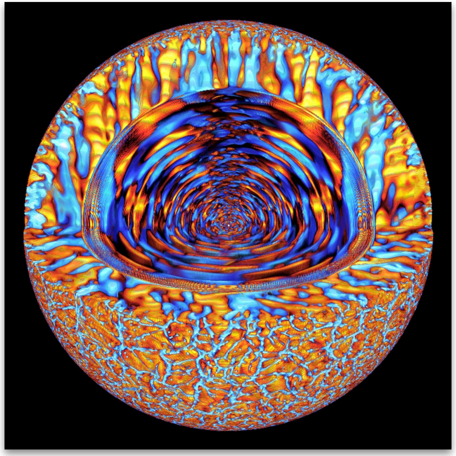
Extreme Horizon

(RAMSES)



Galaxies

(RAMSES)



Solar/Stellar

(ASH)

An ever-growing need for computing power to better understand the universe

Towards Exascale

A diversity of new supercomputer architectures

Older CPU architectures
x86, Intel, AMD, …

- Low energy efficiency, Low power density

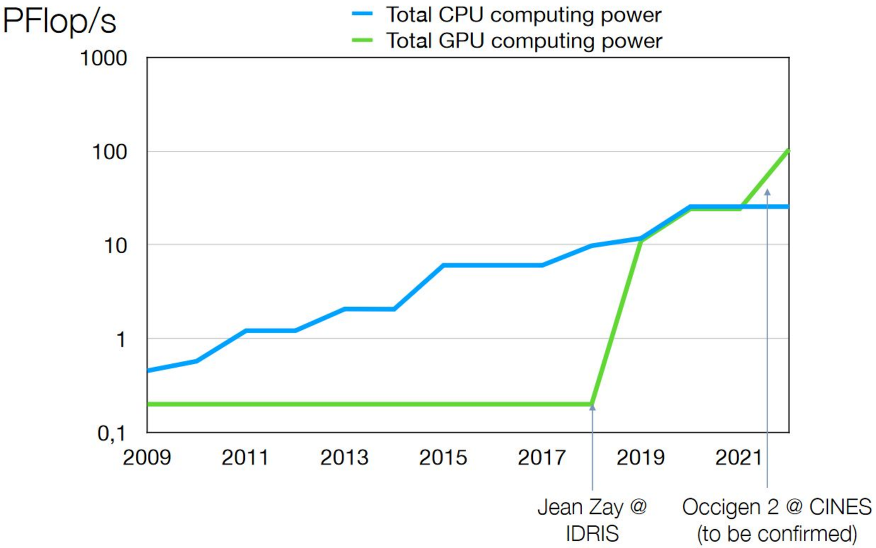
➡ Need a lot of compute nodes

Newer GPU architectures (Exascale)
FR : *Jean-Zay, Irene* (Nvidia), *Ad-Astra* (AMD)
EU: *LUMI* (Finland, AMD), *Leonardo* (Italy, Nvidia)
US : *Frontier* (AMD), *Summit, Sierra* (Nvidia)

- Better energy efficiency, More power per node
- Massively parallel shared memory architectures

➡ ***More efficient but harder to code***

And other new vector architectures : ARM (A64FX, EPI), RISC-V, …



Computing power in French national centers (GENCI 2021)

New architecture for Exascale are harder to program and need new software stacks

Dyablo

Replacing the software stack for Exascale

Older applications and Exascale

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

- Older languages (Fortran) and prog. models
- No shared-memory parallelism (MPI only)
- Sequential algorithms

➡ **Need new software stack and algorithms**

Dyablo's software stack

- Written in *C++*, uses external libraries *(HDF5, PABLO, ...)*
- *Kokkos + MPI* parallelism
- New parallel algorithms

➡ **Supports Exascale Hardware**

Dyablo

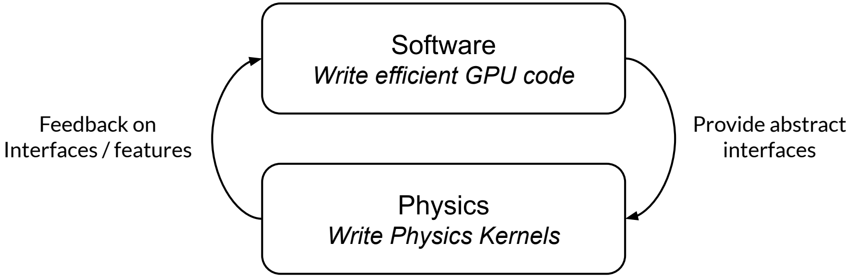
Leverage current software development methods

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


Software
Write efficient GPU code
Provide abstract interfaces
Physics
Write Physics Kernels
Feedback on Interfaces / features

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

Features in Dyablo

Address simulation needs for the astrophysical community

Multi-physics simulations

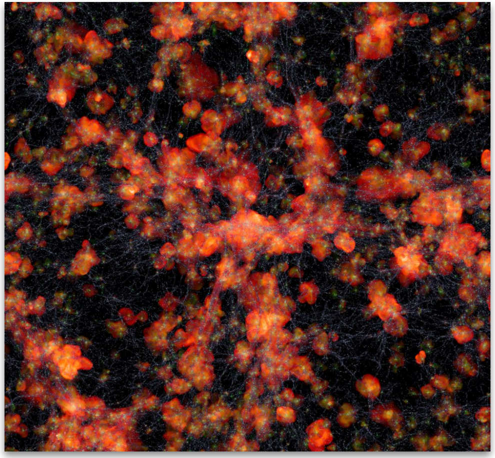
- Hydrodynamics / MHD
- Self-Gravity
- Particles
- …

Adaptive Mesh Refinement (AMR)

- Wide range of time/space scales in same simulation

Massively parallel simulations :

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



Extreme Horizon
(RAMSES)

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

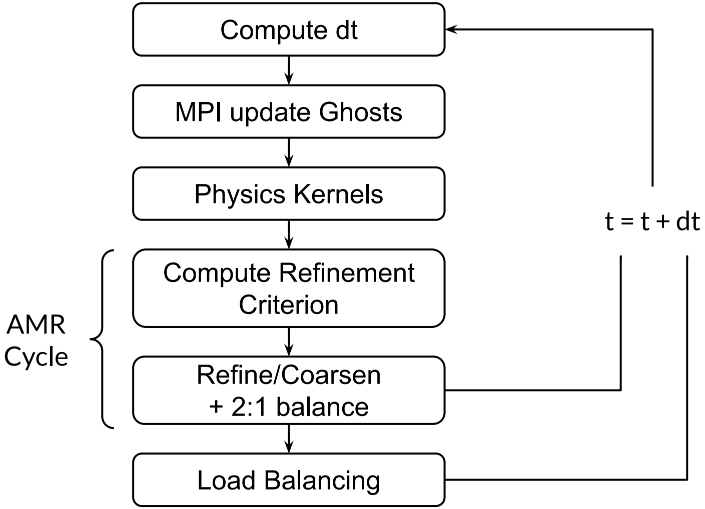
- *RAMSES Community (DAp, …)* Same needs as RAMSES, but at Exascale : dark matter self-gravity, star or galaxy formation, …
- Whole Sun (DAp) - Solar simulation : Convection, radiative transfer, spherical geometry, …

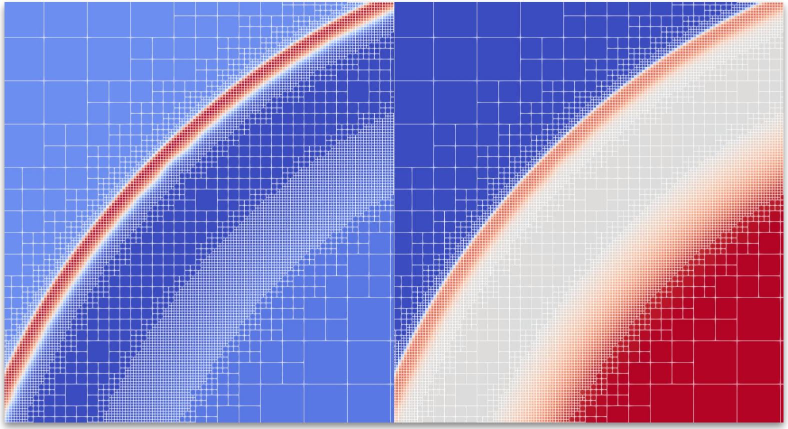
AMR in Dyablo

Adaptive Mesh Refinement (as in RAMSES)

- 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

AMR on GPU in Dyablo

GPU Data Structures for the AMR Octree

Finite Volume Scheme

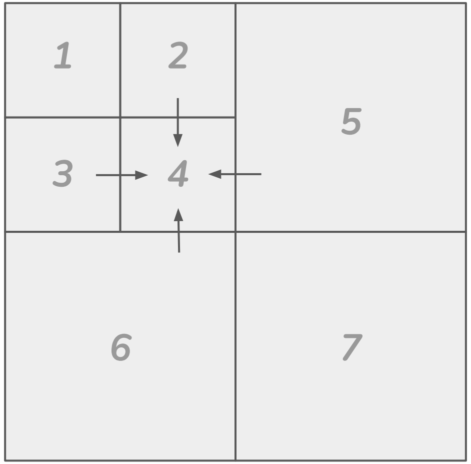
For each cell :

1. Compute Gradients/Reconstruction
2. Flux computation *(Riemann solver)*
3. Update Cell

=> Need neighborhood (*stencil*)

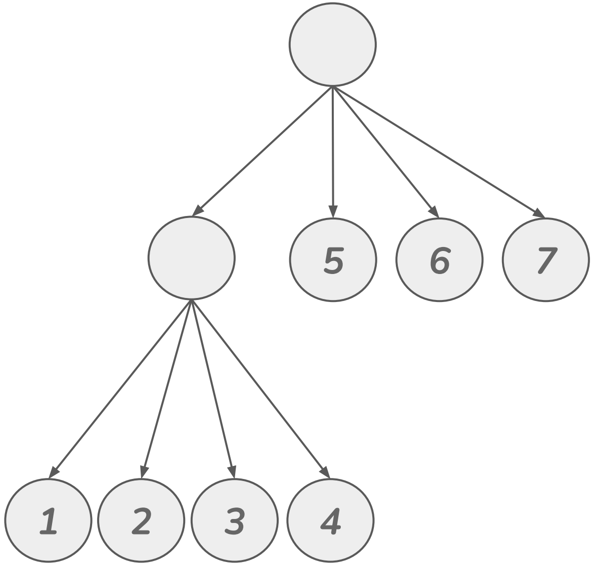
AMR mesh :

- How to store physical fields?



Octree associated with mesh :

- How to iterate on cells?
- How to get neighbors?



AMR on GPU in Dyablo

GPU Data Structures for the AMR 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)

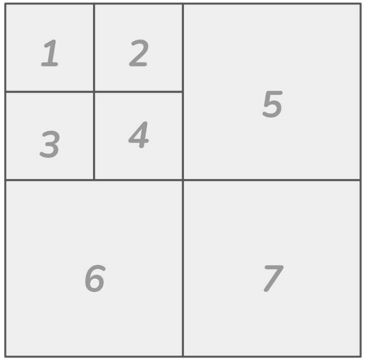
Accessing neighborhood

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

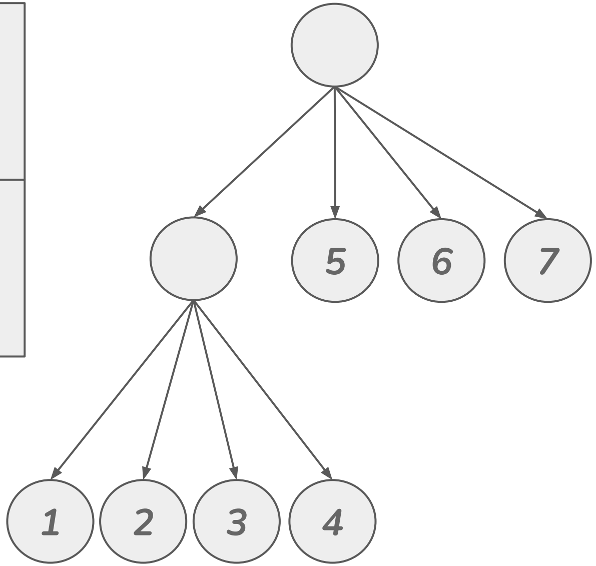
Modularity : 2 AMR backends

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

AMR mesh



Associated Octree

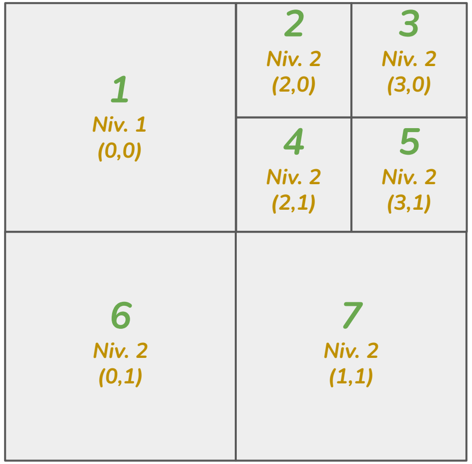


AMR on GPU in Dyablo

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



Maillage AMR

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**

À gauche de 4 :

1. ***4* -> *Niv. 2 : (2,1)***
2. À droite : ***Niv. 2 : (2-1,1)***
3. ***Niv. 2 : (1,1) n'existe pas :***

   On cherche au niveau 1 : ***Niv. 1 : (0,0)***
4. ***Niv. 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 :

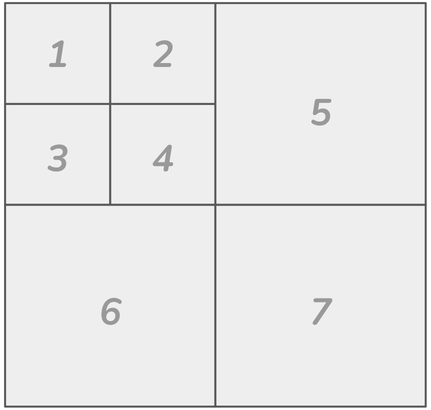
`foreach_cell()`

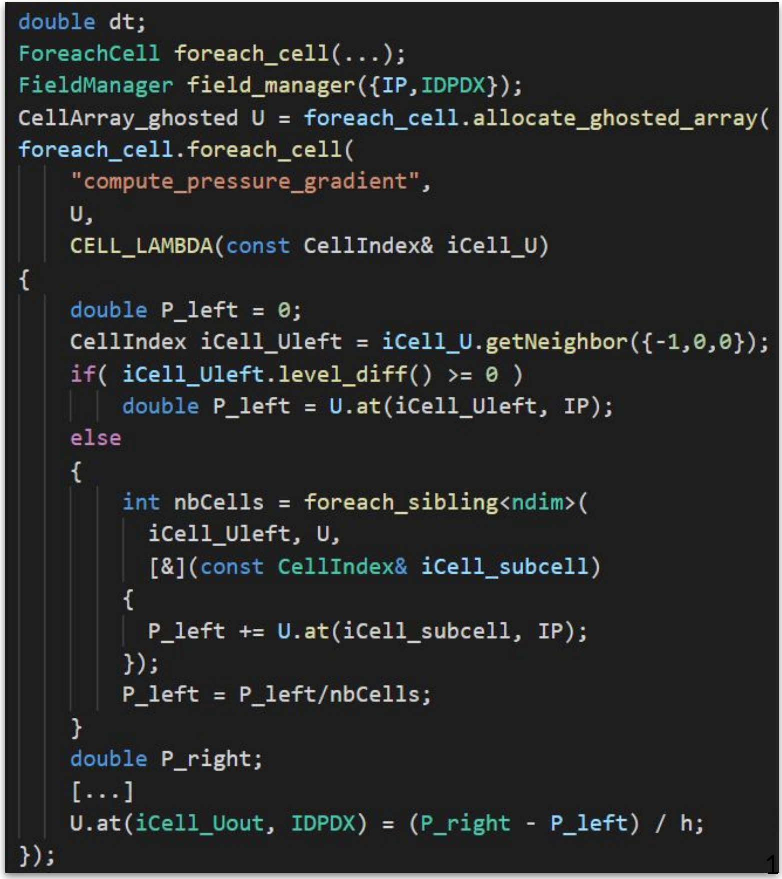
- Lambda-based loop
- Hide Kokkos

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 :

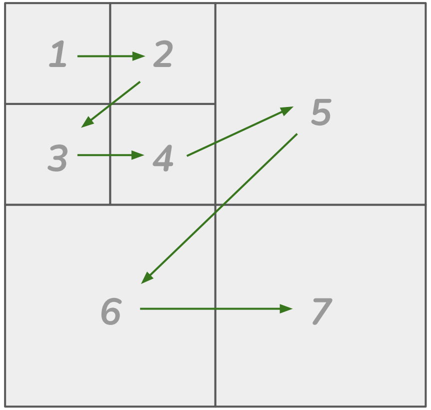
`foreach_cell()`

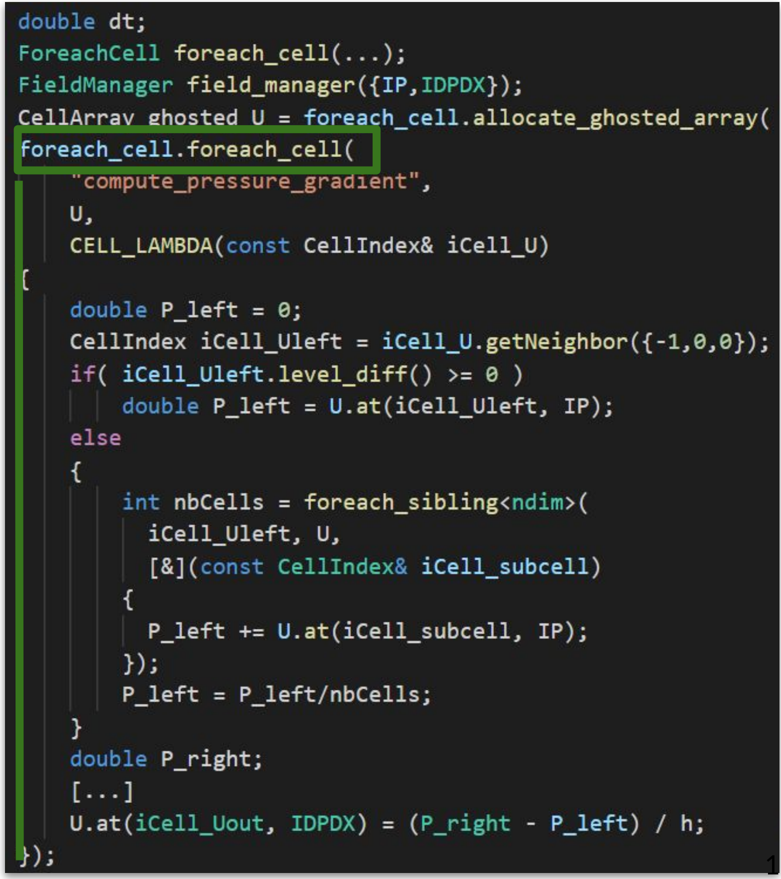
- Lambda-based loop
- Hide Kokkos

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 :

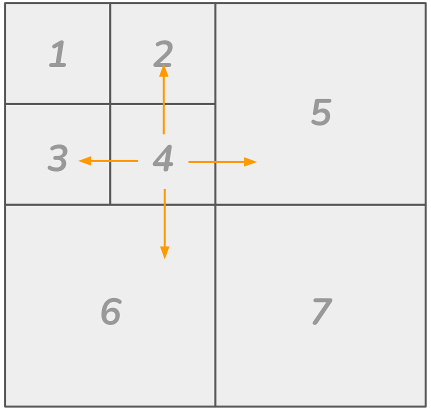
`foreach_cell()`

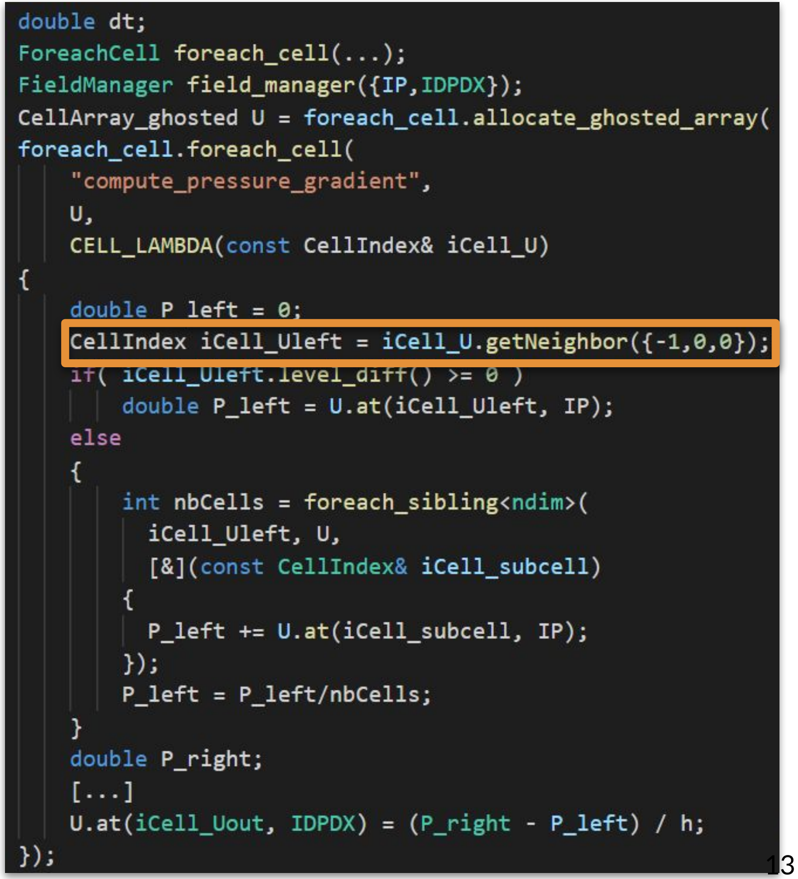
- Lambda-based loop
- Hide Kokkos

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;
});
```

AMR on GPU in Dyablo

Other differences compared to RAMSES

Block-based AMR

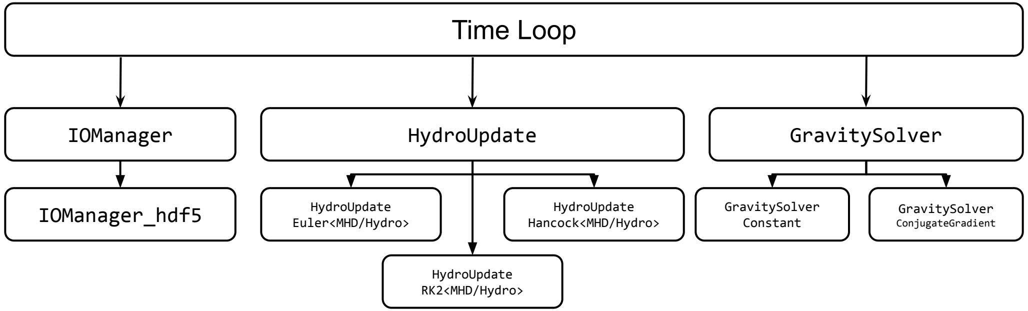
- Store cartesian blocs of cells at leaves of the Octree

➡ Cartesian grids better for GPU

➡ Octree is smaller : AMR cycle is faster

Modularity : plug-ins

- Modern software architecture and patterns
- New Kernels can be added as plug-ins
- Choose at execution time : 2D/3D, numerical scheme, IOs, …



Time Loop

IOManager

IOManager_hdf5

HydroUpdate

HydroUpdate Euler<MHD/Hydro>

HydroUpdate Hancock<MHD/Hydro>

HydroUpdate RK2<MHD/Hydro>

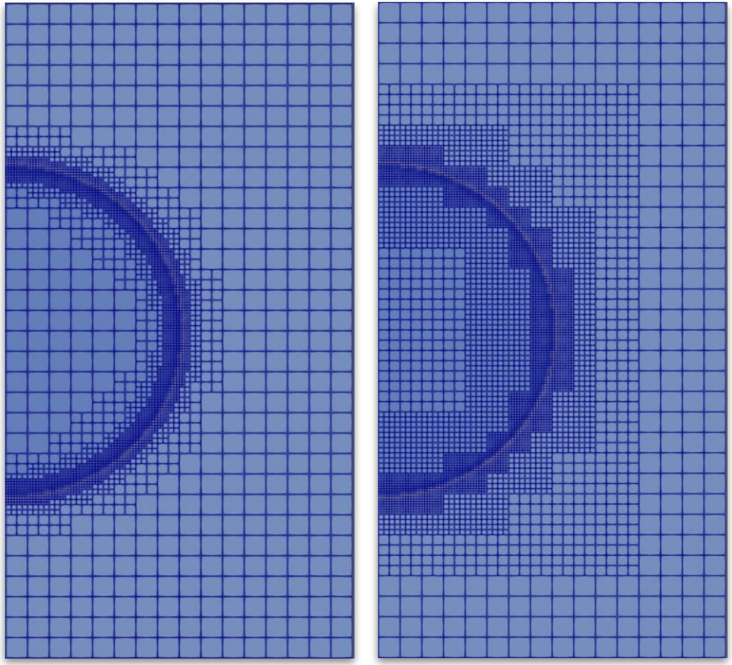
GravitySolver

GravitySolver Constant

GravitySolver ConjugateGradient

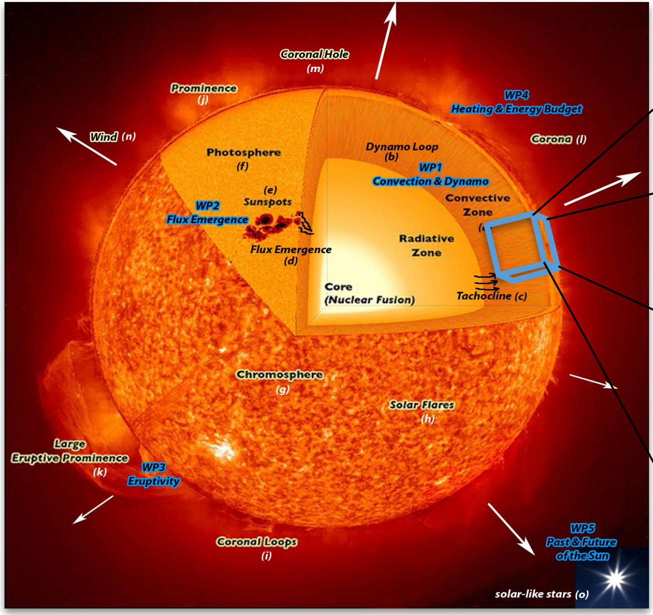
Cell Based
600k Octants
600k Cells

Block Based
18k Octants
1100k Cells

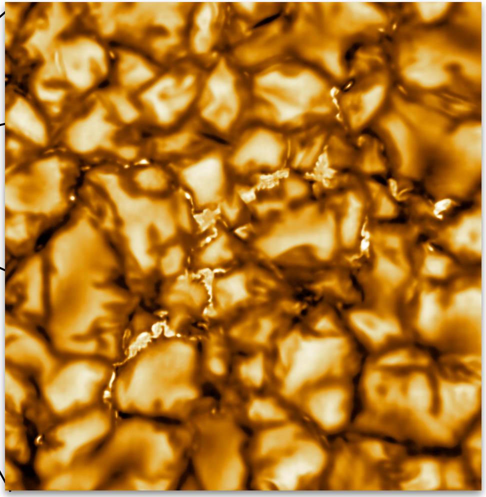


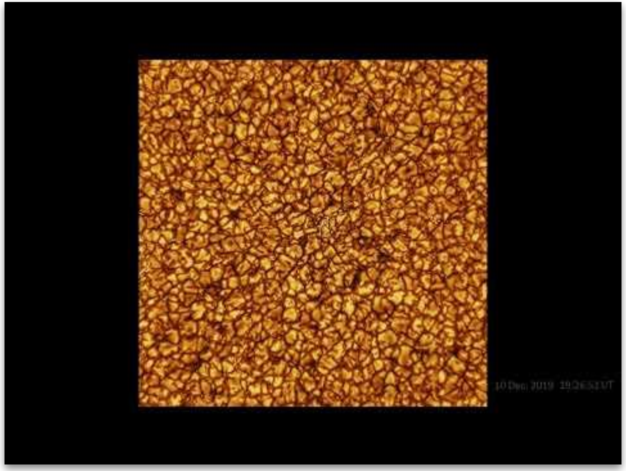
Application #1

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



Credit: [Whole Sun website]





Credit: NSO/NSF/AURA/DKIST

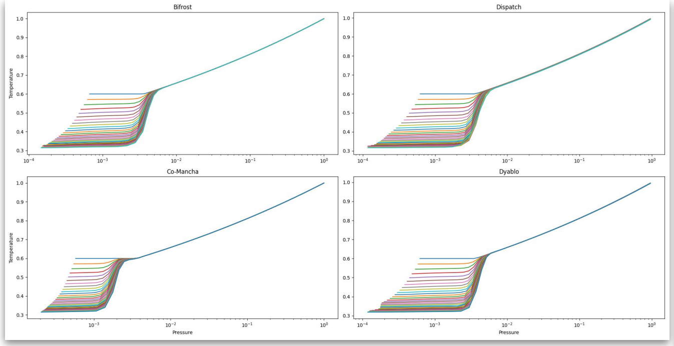
Solar convection

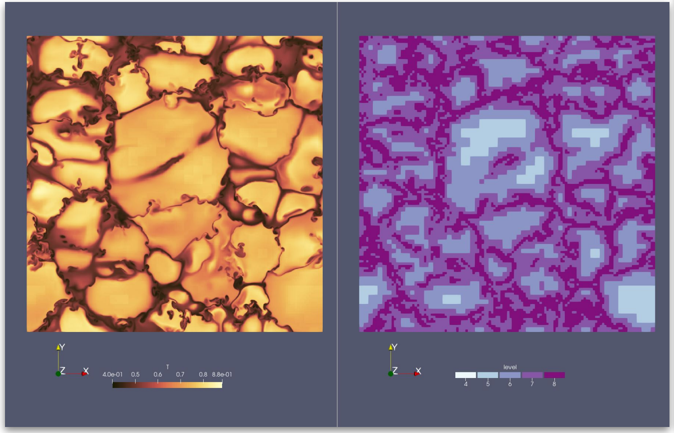
Two convection benchmarks

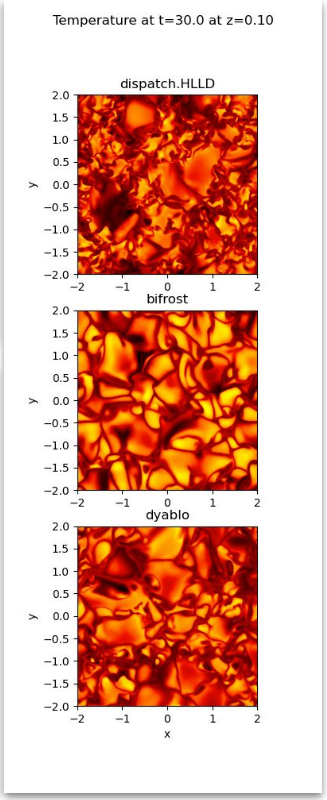
Setup #1

- Open boundary setup
- Surface cooling to simulate Solar atmosphere
- Code comparison with state of the art
  - Bifrost
  - Dispatch
  - Mancha
- Solving for :
  - Hydro
  - Gravity
  - Thermal conduction
  - Viscosity
  - Newtonian cooling

Code has been validated wrt to state of the art







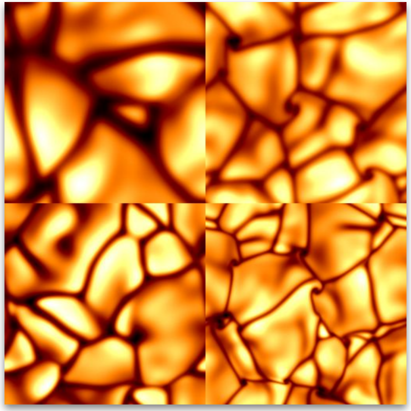
Solar convection

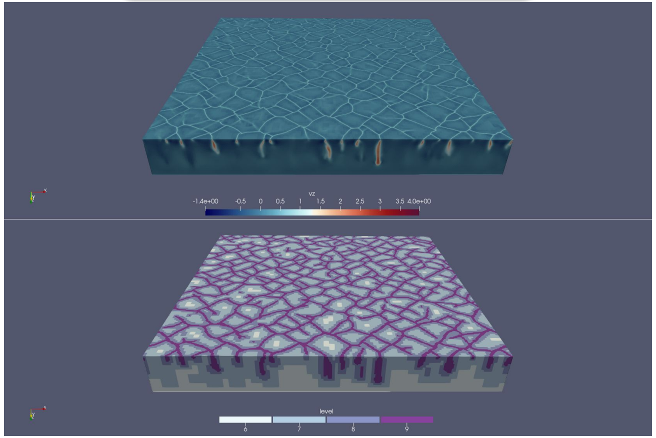
Two convection benchmarks

Setup #2

- Closed boundary setup
- Based on 90’s and 00’s classical setups
- Better control on the experiment
- Parameter study on Prandlt number and stratification
- Solving for :
  - Hydro
  - Gravity
  - Thermal conduction
  - Viscosity
  - (MHD)

Works on CPUs and GPUs
Works with AMR



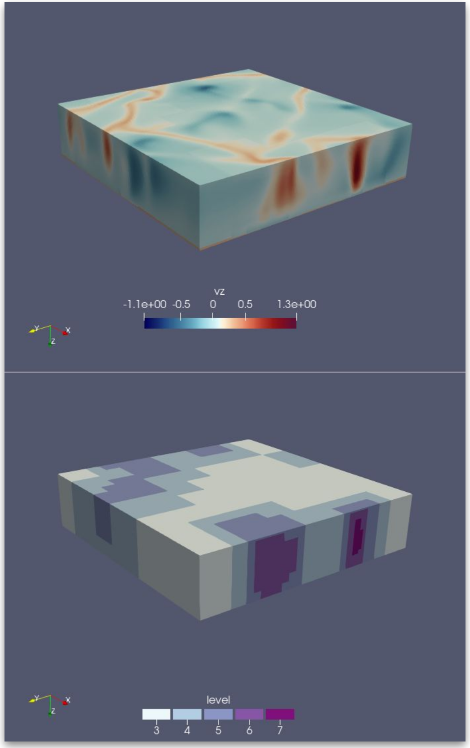


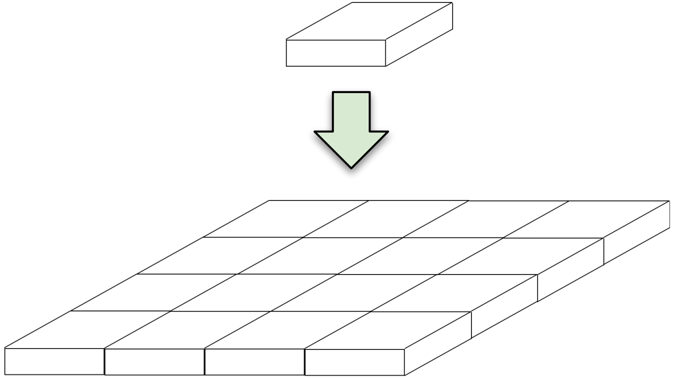
Weak scaling benchmarks

Use case

Solar convection slab :

- Convection slab:
  - Hydro + TC + viscosity + cooling
- 3-7 refinement levels
  - Base resolution 128x128x32
  - Max resolution 2048x2048x512
  - 30.6M cells per domain
- Horizontal tiling per MPI process
  - Load-balancing is ensured
- 100 iterations, 1 AMR cycle per iteration
- 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

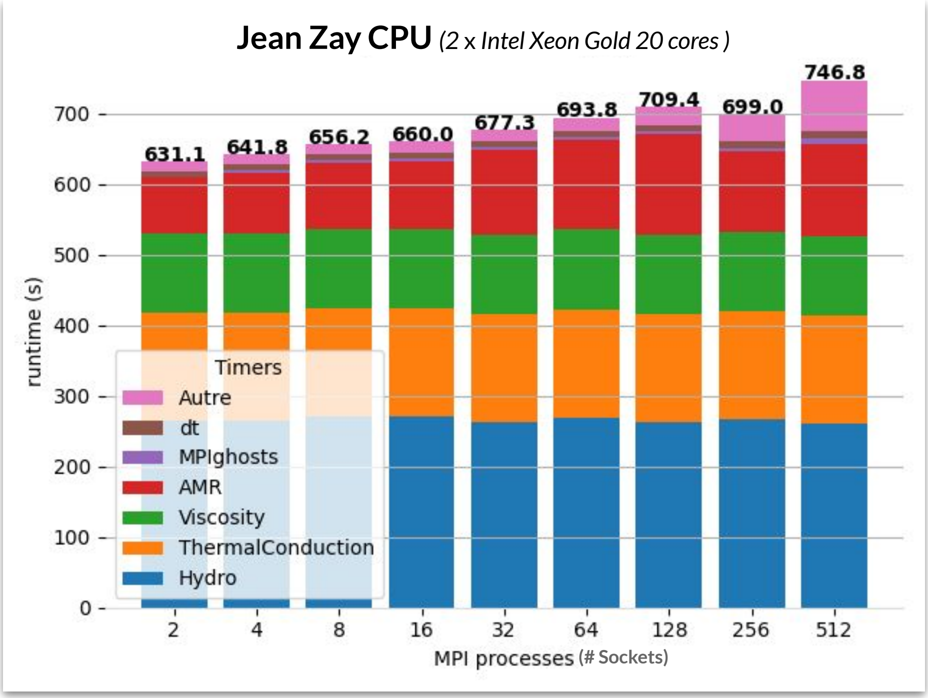


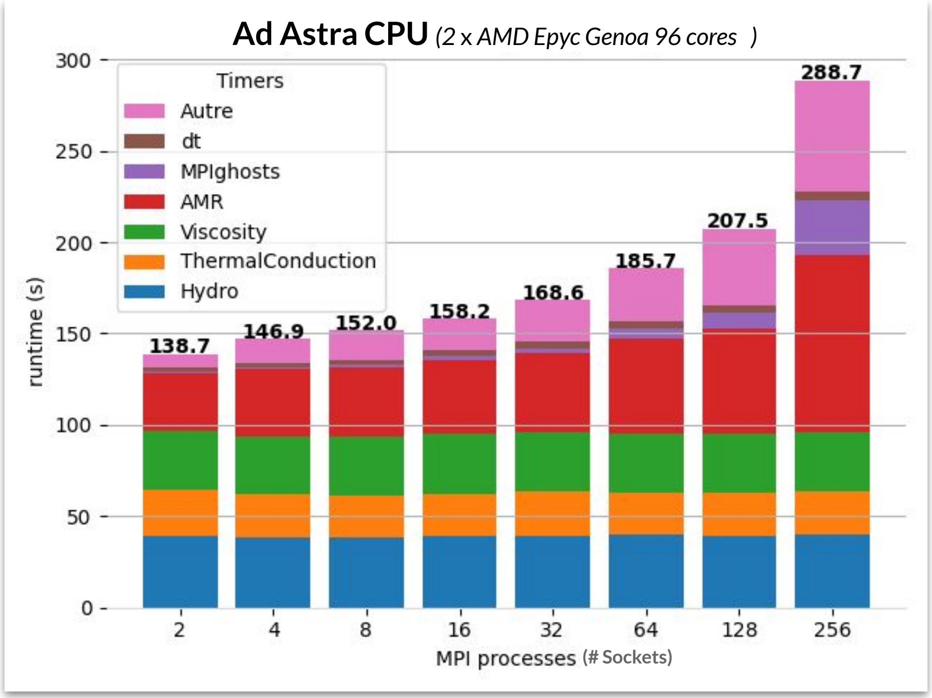


Replication on N MPI processes

Weak scaling benchmarks

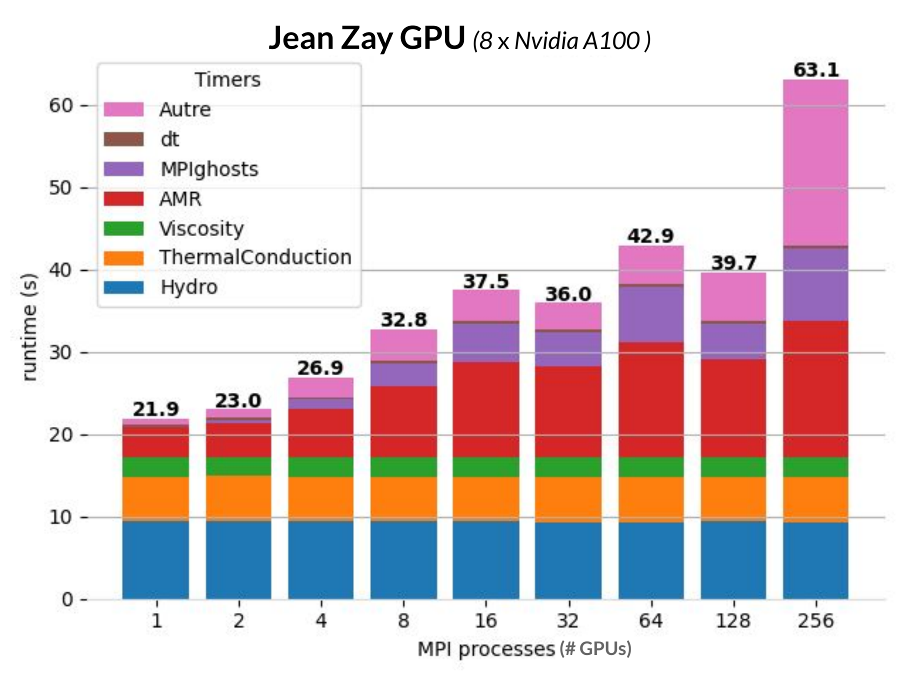
CPU results

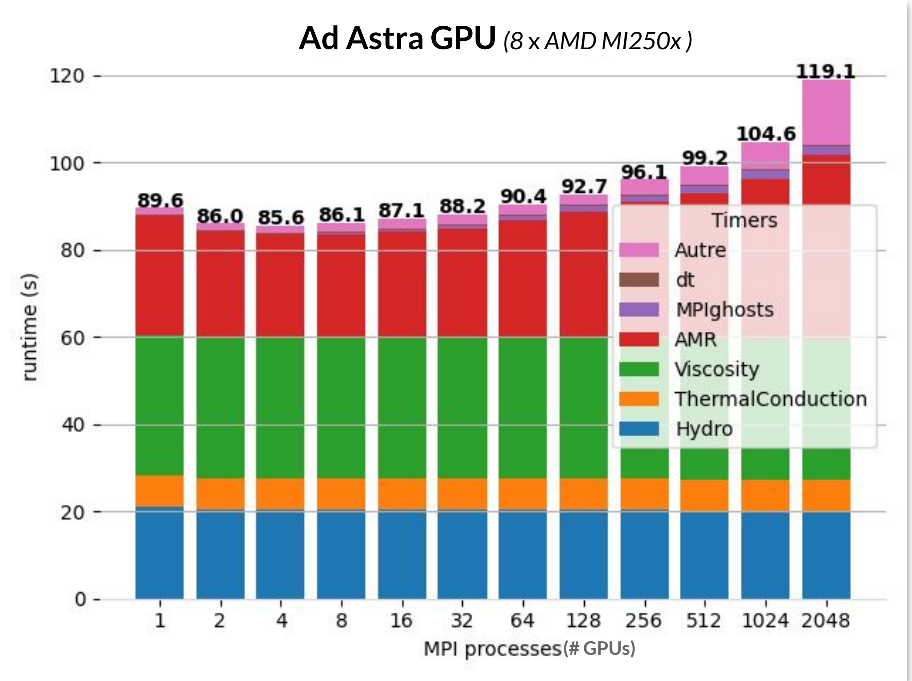




Weak scaling benchmarks

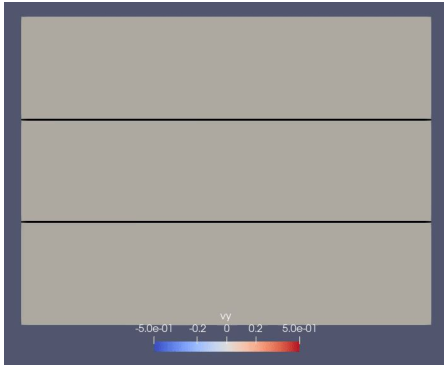
GPU results

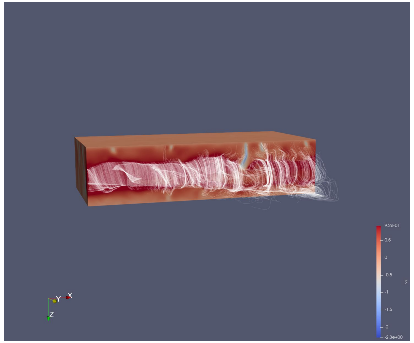

Jean Zay GPU *(8 x Nvidia A100)*


Ad Astra GPU *(8 x AMD MI250x)*

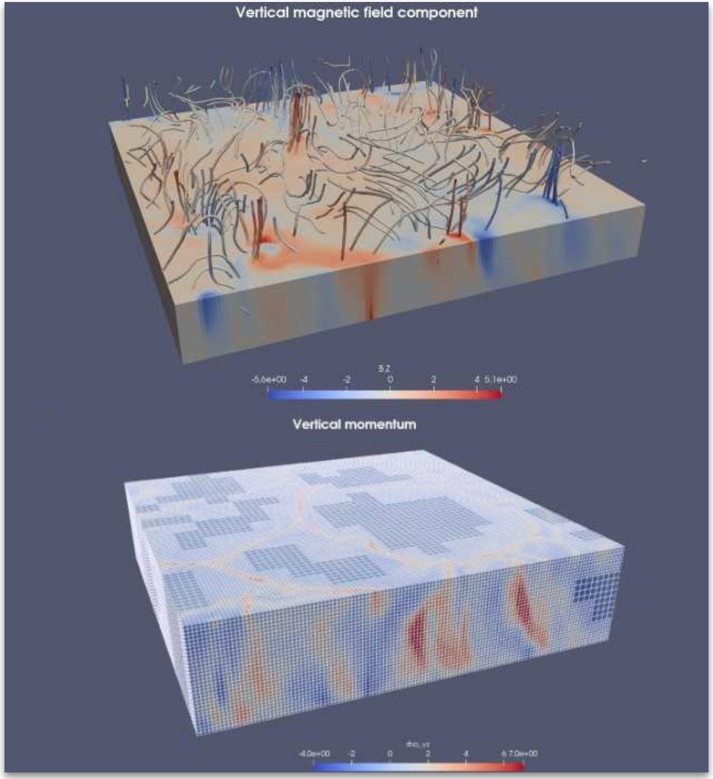
Ongoing work - Whole Sun project

Tri-layer setup (A. Finley)





Flux tube experiment (C. Blume)

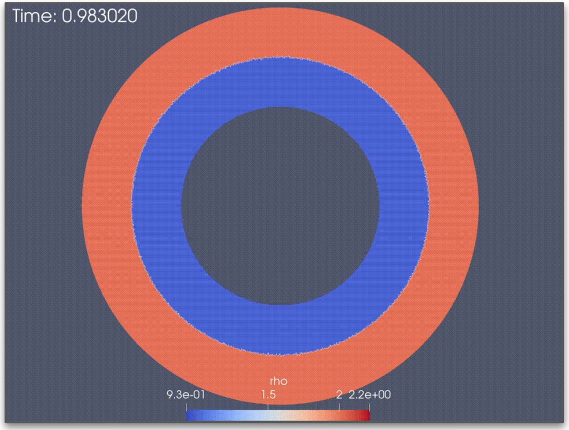


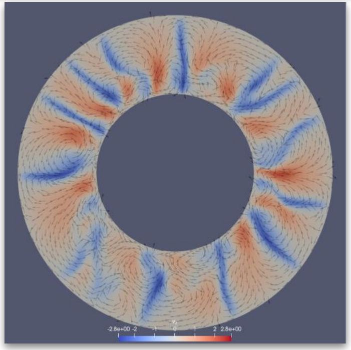
Vertical magnetic field component

Vertical momentum

Convective Dynamo (A. S. Brun)

Geometry module
(G. Doebele)



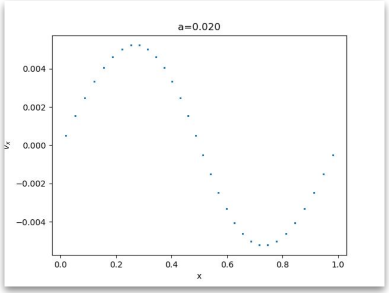


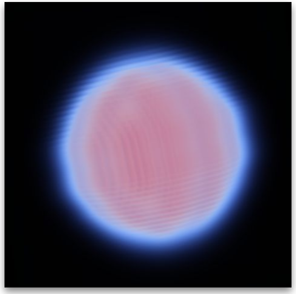
Application #2

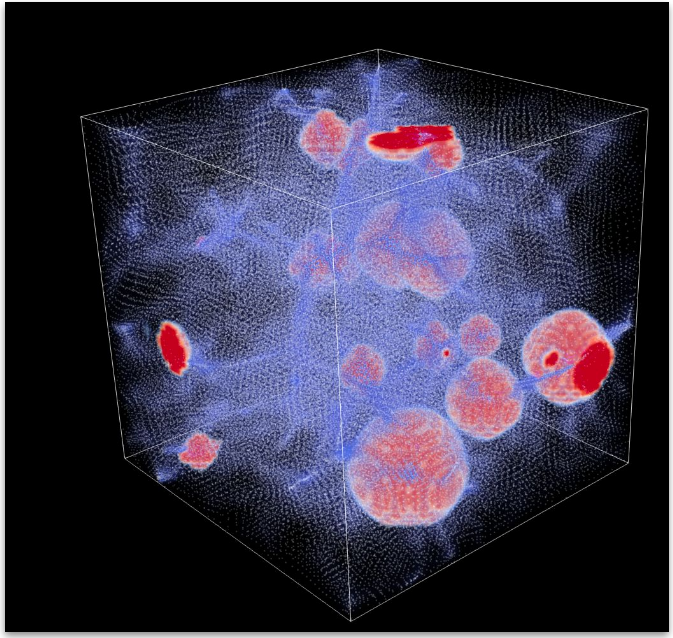
Cosmological radiative transfer

Setup :

- Collaboration with Observatoire de Strasbourg (D. Aubert, O. Marchal)
- Periodic expanding box (super-comoving coordinates)
- Solving for :
  - Hydro (mesh + particles)
  - Self-Gravity
  - Radiative transfer (M1)
- Allows for the simulation of large structure formation and ionization
- Validation tests :
  - Zeldovitch pancake
  - Stromgren sphere







A box of 64^3 points of width 4 Mpc
In blue : DM particles
In red: Ionized regions

Leveraging Exascale architectures with Kokkos

Performance Portability with Kokkos

- Dyablo runs on CPUs (Intel, AMD, ARM*) and GPUs (Nvidia, AMD*, Intel**) with one codebase
- Performance and Scalability
- Hide *some* of the GPU code complexity

Adapted AMR algorithms for GPUs

- Hashmap AMR Octree
- Block-based AMR
- Hide complexity behind abstract interfaces

➡ *Proof that Separation of concerns works for AMR at Exascale in Dyablo*

Two applications showcased

Solar physics

- Cartesian slabs over a few AMR levels
- MHD + diffusion operators
- Comparison with state of the art codes
- Weak scaling benchmark up to 2048 GPUs and ~50k CPU cores

Cosmology

- Expanding coordinates
- Particle-Mesh integration
- Hydro + Gravity + Radiation

More to come from collaborations at CEA, CNRS and the various partnerships in astrophysics labs : dust, stellar formation, galaxy formation, star-planet interaction, stellar physics, etc.

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” -> 3 postdocs/PhD students potentially working on dyablo for the implementation of new physics.
  - Two confirmed working on dust and gravitational solvers.

HPC/Computer science projects

- **EUPEX :** European Pilot for Exascale -> Porting the code to ARM architectures
- **CExA :** Exascale at CEA -> Working jointly with kokkos to improve the code and the performances
- **PEPR Numpex**

Challenges for AMR at Exascale

IOs and visualization formats and tools

Build data format suited for AMR

- To store efficiently :
    - Use hierarchical properties for efficient storage/compression
    - Take into account block-based approaches
- For efficient post-processing (ex : Level of details, partial load…)

Our experience : Unstructured data is too heavy

(30.6Mcells ~ 3.83Gb of geometry + 1.2Gb of physics data for a pure hydro run)

Build post-processing tools

- User friendly tools :
    - Features : (Extract regions, statistics, slices, … and many more)
    - Fast and light enough to run on post processing machines

Our experience: paraview is not light enough (needs to load whole mesh at once)

Leads for possible solutions : (both need adaptation for block-based)

- Custom format and tools : e.g. LightAMR
- HypertreeGrid in paraview

Challenges for AMR at Exascale

Implicit methods for diffusion operators

Time-stepping issues with diffusion solvers

- Hyperbolic CFL scales as Δx
  - 20 active AMR levels -> dtmax/dtmin ~ 10^6
- Parabolic CFL scales as $Δx^2$
  - 20 active AMR levels -> dtmax/dtmin ~ 10^{12}
- Explicit integration is impossible.
- **Possible solution :**
  - IMEX methods where diffusive steps are solved implicitly
  - **BUT** : IMEX methods mixed with LTS are notoriously difficult to implement and to parallelize
  - **BUT (2)** : Astrophysics simulation tend to be heavy on memory and cannot afford the cost of matrix based linear solvers -> Matrix-free methods should be privileged
- **Possible solution #2 :**
  - Using STS or RKL methods.
  - **BUT** : Can subcycling make up for the orders of magnitudes differences in dt in the extreme cases ?

Challenges for AMR at Exascale
Load balancing

Load balancing is notoriously hard of AMR codes:

- Multi-scale runs will require large numbers of active levels (> 15) and a large number of particles
    - Cell distribution different from particles distribution ?
    - Load distribution between levels?

Our experience :

- Without local timestepping we don’t have the issue yet.
- We hope hybrid parallelism helps (more work per process)

Challenges for AMR at Exascale

Kernel performance for all architectures

Kokkos as a performance portability layer

- Runs on every target architecture so far
- **But** performance portability isn't perfect

=> CPU and GPUs have different bottlenecks that should be handled differently

- Store vs recompute, data layout, local memories …

We need Kokkos optimization expertise to optimize kernels

Optimize kernels

- Efficiently use Kokkos performance portability tools

  (View layout, hierarchical parallelism, scratch memory)
- Write different versions for different backends

  (Store vs recompute, …)

We would like to build expertise, tools and methods to optimize Kokkos kernels

Roadmap 2024

Core developments :

- Local time-stepping
- Units
- Geometry
- 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

EUPEX :

- Profiling SVE and HBM
- Profiling on Grace Hopper
- Porting to Rhea [if available in 2024]

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