mini-ramses: status report

Romain Teyssier

PRINCETON UNIVERSITY

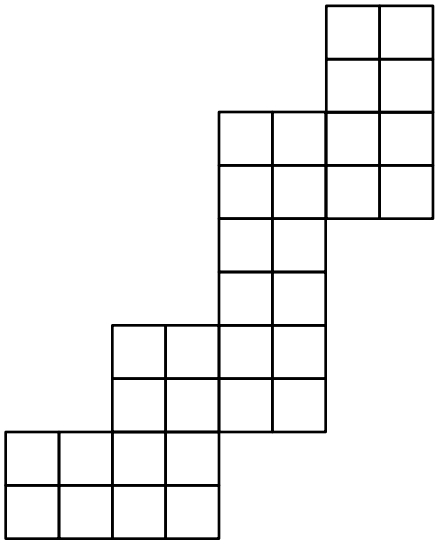
mini-ramses: data structure

- Global oct indexing uses a quadruplet hash_key=(l,i,j,k).
- Hash table: igrid=get_grid(hash_key)
  - Octs are pulled from memory using a simple hash function, collisions are handled using a linked list (same a python dictionary)
- Octs are sorted in memory first per refinement level, then per Hilbert key (using a fast radix sort)
- Octs can be grouped in blocks called superocts
- Perfect load balancing at every time step and for each level
- Remote information is managed (using MPI) via a software cache
  - Use only 10'000 ghost octs to store (temporarily) remote information.
  - Use many small MPI messages while computing
  - Use pre-fetch for optimized MPI communications (require large cache size)

mini-ramses: data structure

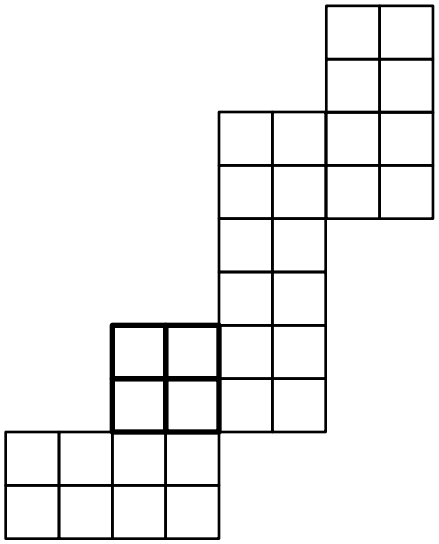
- Optimized execution using clean octs, dirty octs and ghost octs
- Clean octs have all neighbors in local memory
- Dirty octs have at least one neighbor in:
  - Physical boundaries
  - Other MPI domain
  - Coarser refinement level
- Ghost octs are created temporarily in the cache memory from:
  - Physical BC rules (reflexive, zero gradient, imposed…)
  - MPI communications
  - Coarse-to-fine interpolation
- Use GPU or OpenMP asynchronicity to:
  - Build the cache while updating the clean octs
  - Then update the dirty octs

mini-ramses: hydro kernel



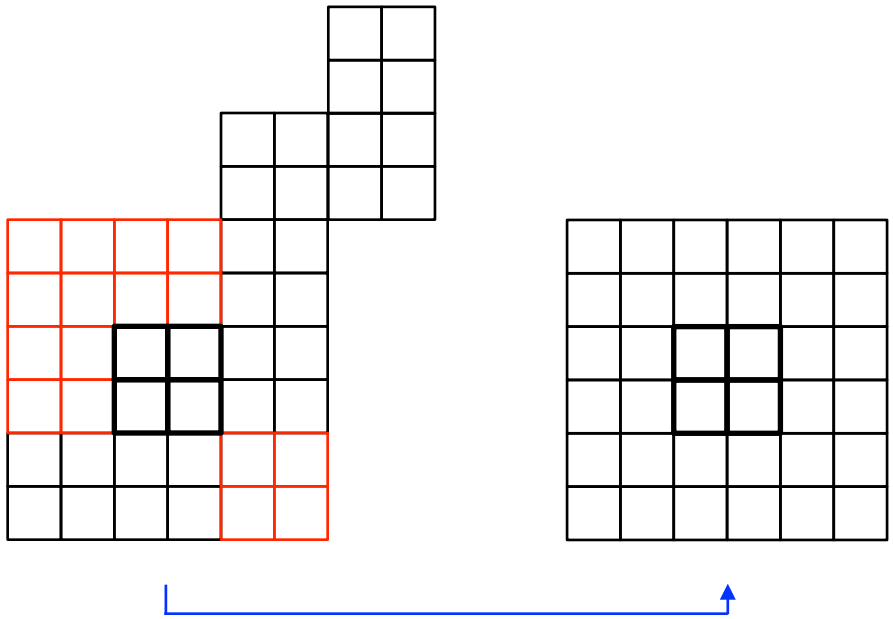
Octs are sorted in memory following a Hilbert space filling curve

mini-ramses: hydro kernel



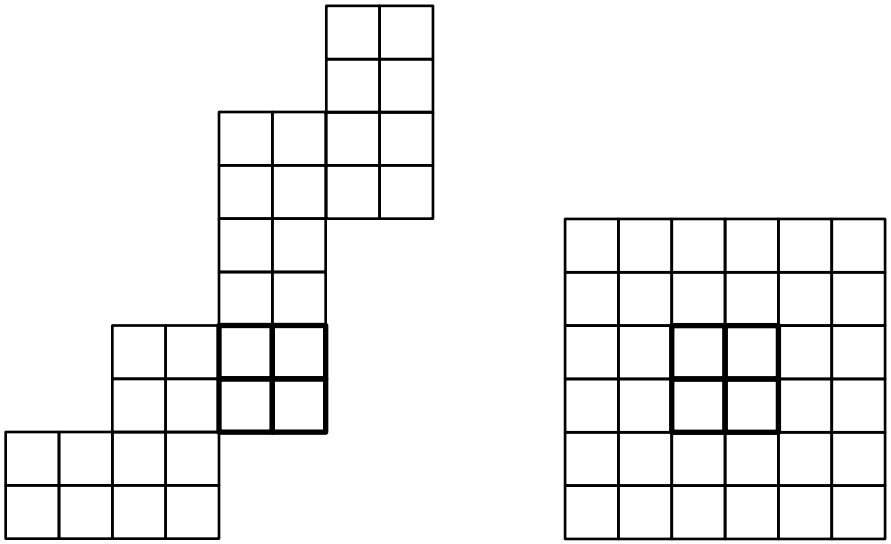
We loop over octs (here oct number 3)

mini-ramses: hydro kernel



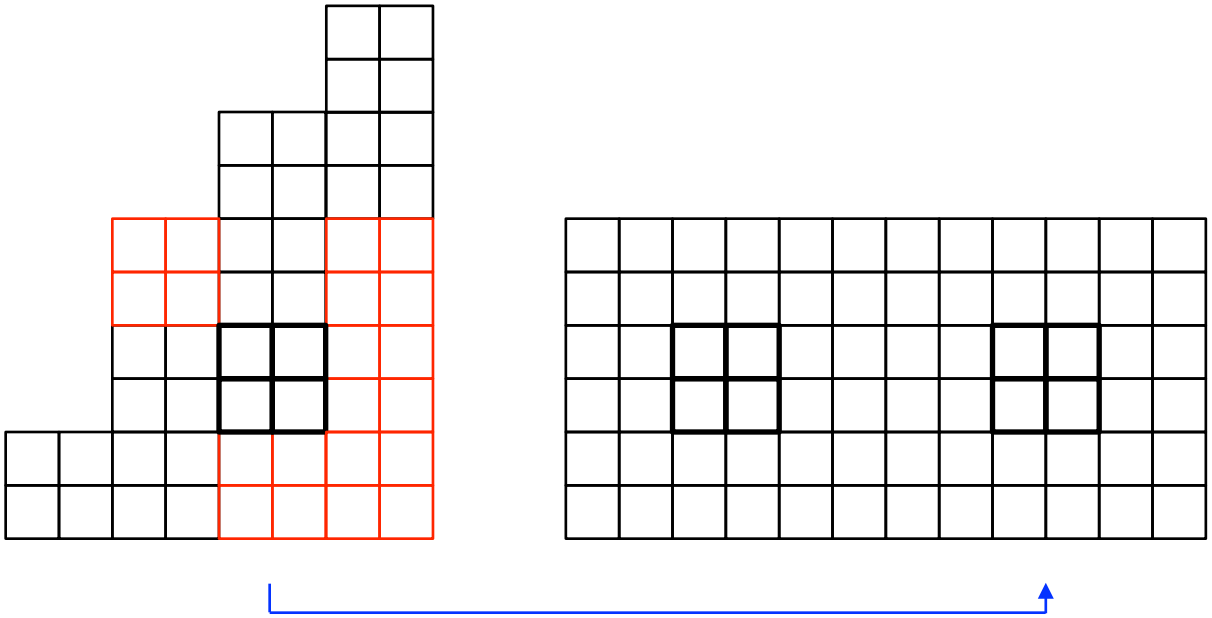
We create ghost octs in cache memory (here in red)
and copy 3x3x3 octs in the kernel at position 3.

mini-ramses: hydro kernel

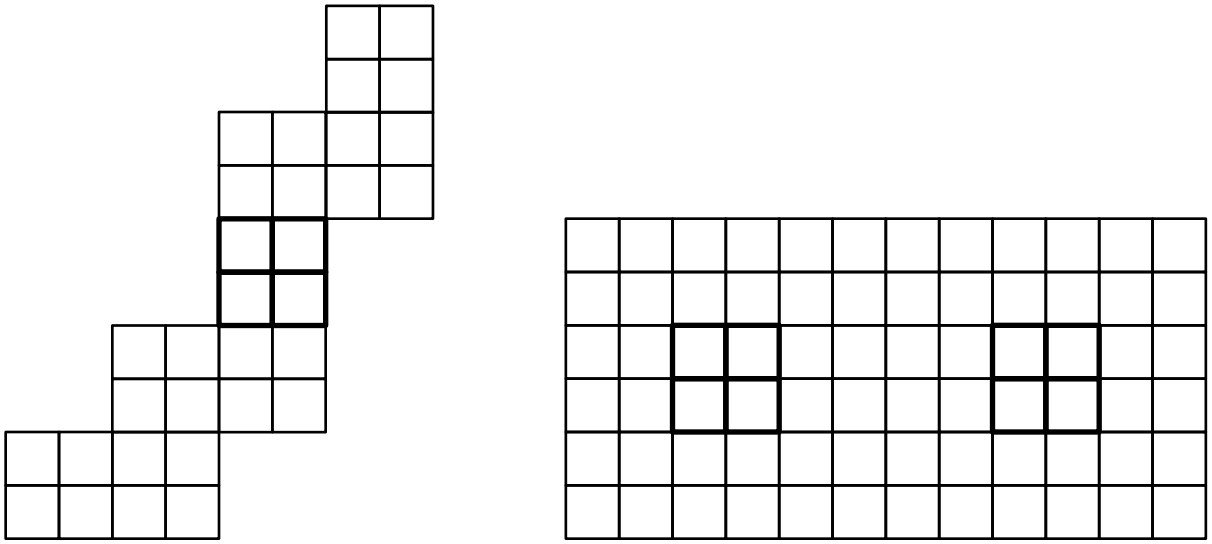


We loop over octs (here oct number 4)

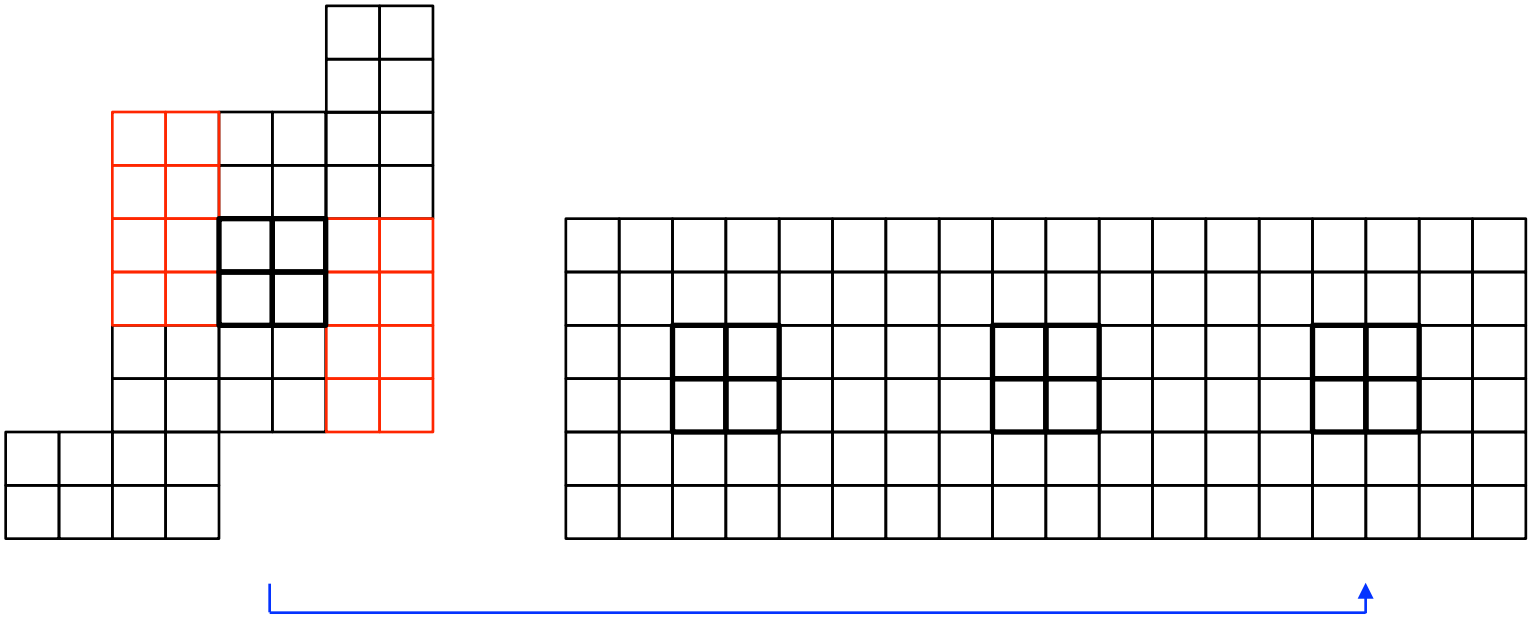
mini-ramses: hydro kernel



We create ghost octs in cache memory (here in red)
and copy 3x3x3 octs in the kernel at position 4.

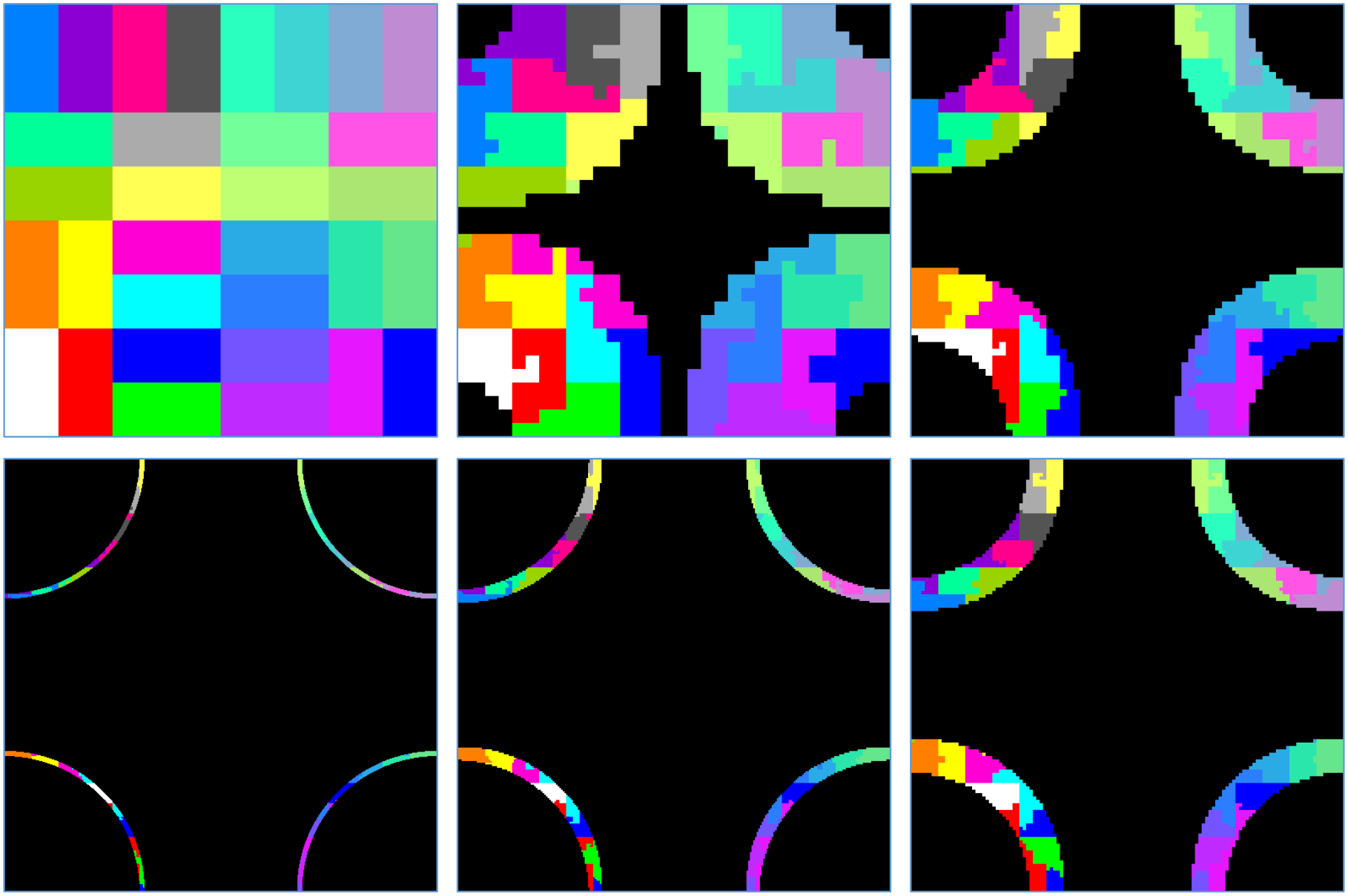


We loop over octs (here oct number 5)



We create ghost octs in cache memory (here in red)
and copy 3x3x3 octs in the kernel at position 5.

mini-ramses: level-by-level load balancing



mini-ramses: initial conditions

- Various predefined initial conditions
  - Use condinit.f90 initial condition setup file.
  - Default using regions and namelist block INIT_PARAMS
  - Predefined ICs: INIT=COEUR, INSTA, DOUBLEMACH, OT, PONO, ABC, LOOP…
  - PATCH=../patch/init/halo and PATCH=../patch/init/merger

- MUSIC initial conditions using filetype='grafic' or filetype='grafic_zoom'

- GADGET initial conditions using filetype='gadget'
  - Generated using e.g. the DICE code

- RAMSES initial conditions using filetype='ramses'
  - Reads RAMSES output files (not restart files)
  - Still need to specify the initial magnetic field separately

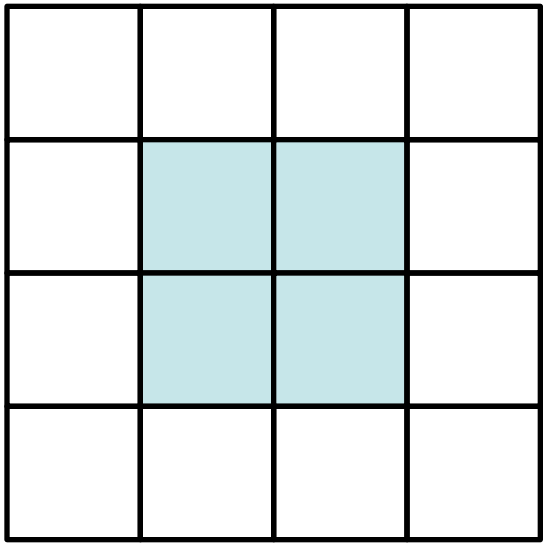
mini-ramses: new features

- Various boundary conditions (default is periodic)
  - Arbitrary box geometry using oct Cartesian indexing at levelmin:
  - Reflexive, zero gradient, imposed…

Example from sedov2d.nml

```
levelmin=3
box_size=1.0
box_xmin=1
box_ymin=1
box_xmax=-1
box_ymax=-1
```

```
periodic=.false.,.false.
nbound=4
bound_dir=1,1,2,2
bound_type=1,1,1,1
bound_shift=1,-1,1,-1
bound_xmin= 0,-1, 0, 0
bound_xmax= 1, 0, 0, 0
bound_ymin= 1, 1, 0,-1
bound_ymax=-1,-1, 1, 0
```

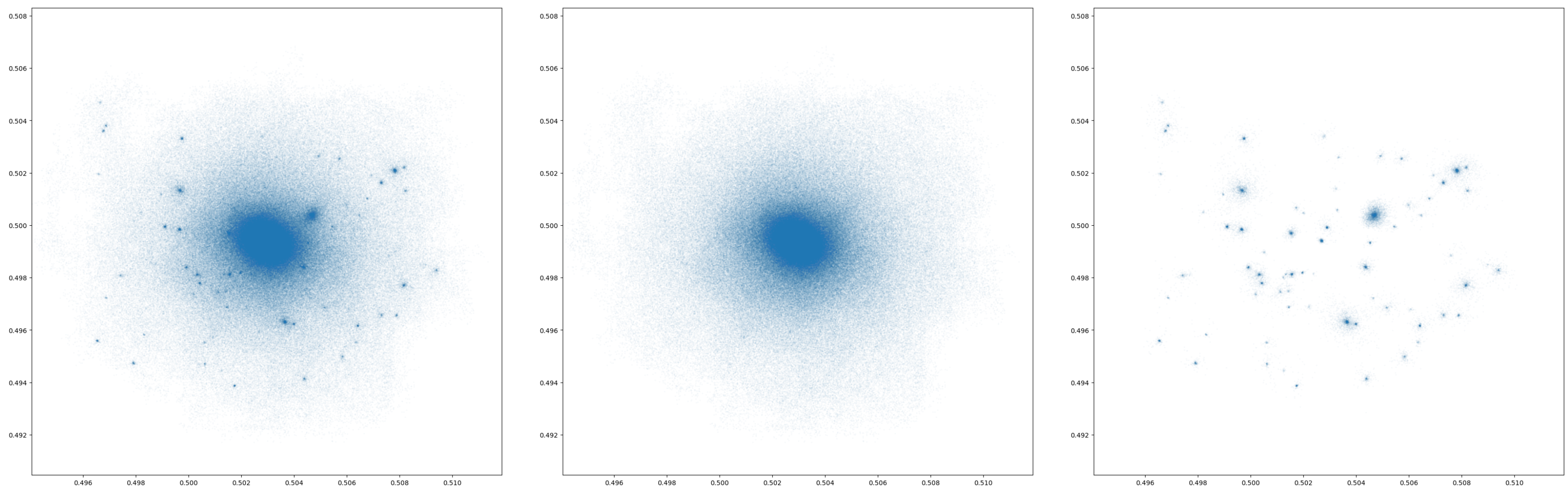


mini-ramses: new features

- Physics routines
  - Star formation recipe with 3 different models
  - Dual energy update using entropy
  - Thermal supernova feedback
  - Mechanical supernova feedback
  - Subgrid turbulence LES model
  - Equilibrium cooling, constant UV background with self-shielding
  - Non-equilibrium cooling with H and He chemistry
  - MHD solver now merged with hydro solver
  - switch_llf_dmin=-1 and switch_llf_pmin=-1

mini-ramses: new features

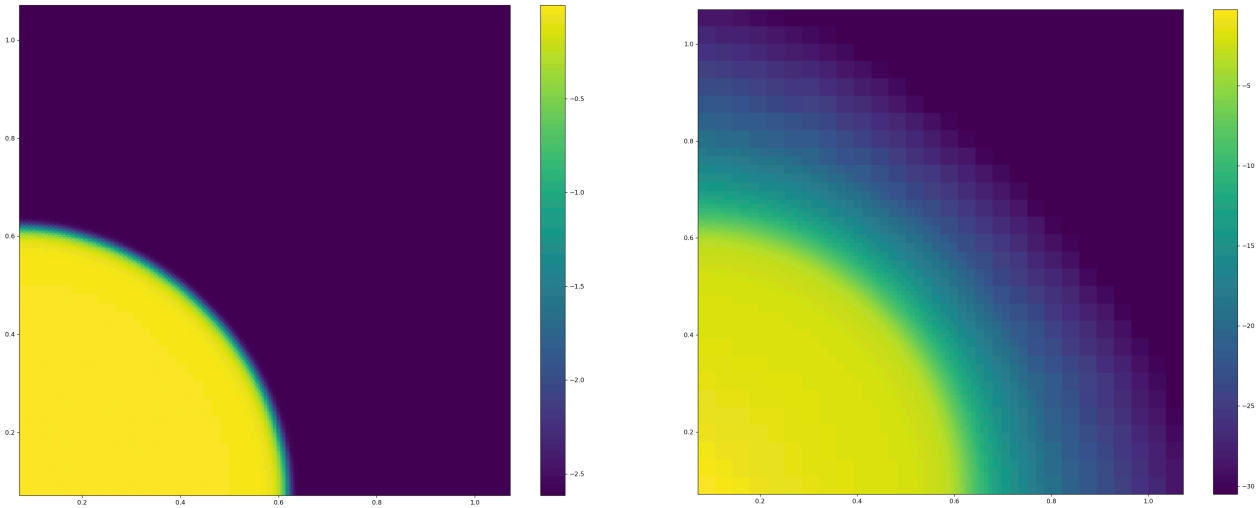
- Clump finder
  - PHEW implementation with software cache
  - Introducing central halos and satellite halos
  - Particle unbinding
  - Merger tree particles



mini-ramses: new features

- Sink formation and evolution
  - Seeding using clump finder
  - Reset position to clump center and/or artificial friction
  - Accretion and feedback (see Nick's talk)

- Radiative transfer with M1 solver (see Joki's talk)



mini-ramses: initial conditions

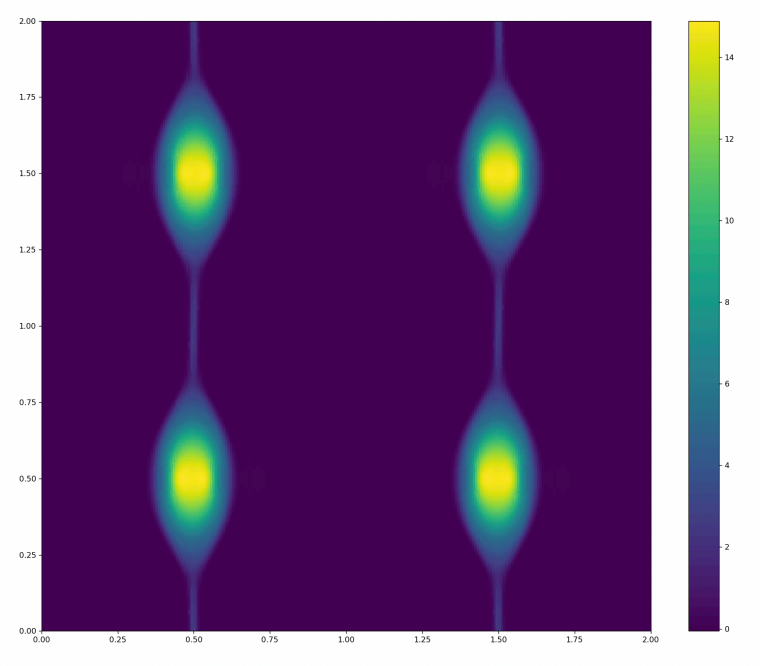
- 1D tests:
  - HYDRO: advect1d.nml, blast1d.nml, sedov1d.nml, tube1d.nml
  - MHD: brio_wu.nml, dai_woodward.nml

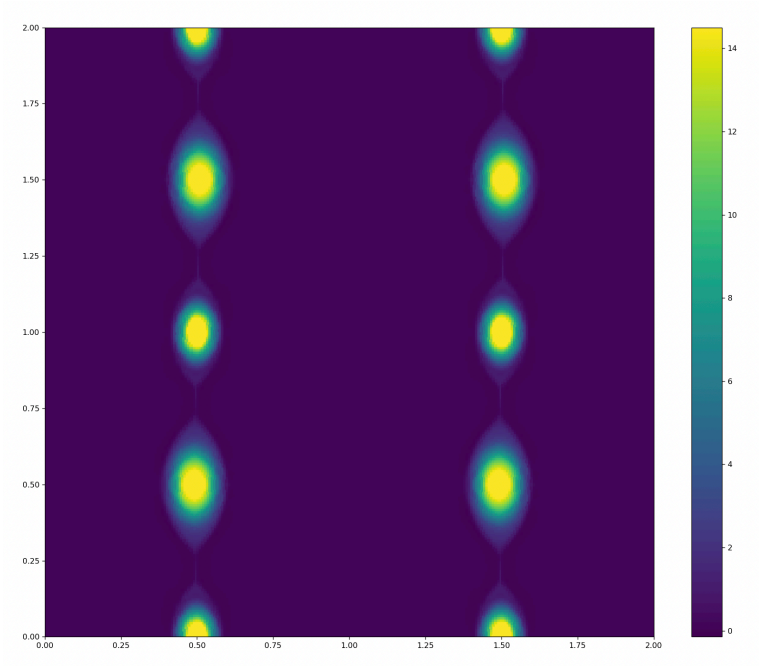
- 2D tests:
  - HYDRO: sedov2d.nml, double_mach.nml, rt.nml
  - MHD: loop.nml, ot.nml, current_sheet.nml
  - RT: stromgren2d.nml, beam2d.nml, shadow2d.nml

- 3D tests:
  - N-BODY: dmo.nml, dmo_zoom.nml
  - HYDRO: sedov3d.nml, coeur.nml, cosmo.nml, cosmo_zoom.nml, halo.nml, merger.nml, dice.nml
  - MHD: pono.nml, abc.nml, halo_mhd.nml
  - SINK: bondi.nml, agn_jet.nml

Double Mach reflection

Current sheet





mini-ramses: new features

- New output strategy
  - access='stream' python read using numpy.fromfile
  - Backup frequency bkp_timrs_hrs=2 and bkp_modulo=3
  - Restart folder: double precision, face-centered magnetic field (6 vars)
  - You can change the number of cpu at restart
  - Output folder: single precision, cell-centered magnetic field (3 vars)
  - You can specify the number of files per output folder
    - Default nfile=1 with $1 \leq \text{nfile} \leq \text{ncpu}$

- Maximum level 64 using HILBERT=3 (no time penalty).

mini-ramses: new features

- Makefile:
  - make NDIM=3 MPI=1 DEBUG=1 UNITS=MERGER GRAV=1 PATCH=../patch/init/merger NPSCAL=2 MHD=1

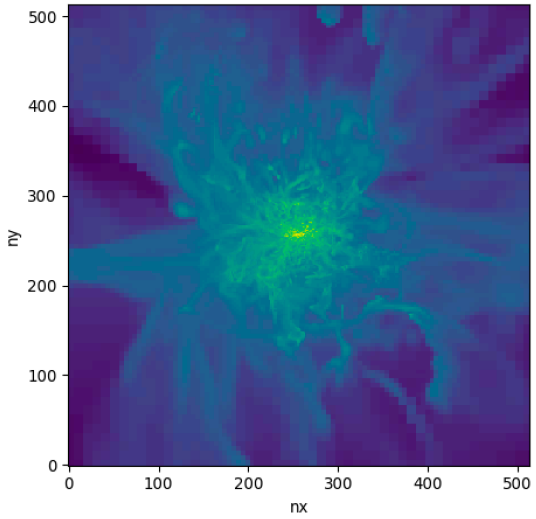
- cmake also available:
  - cmake .. -DNDIM=3 -DINIT=PONO -DMHD=1

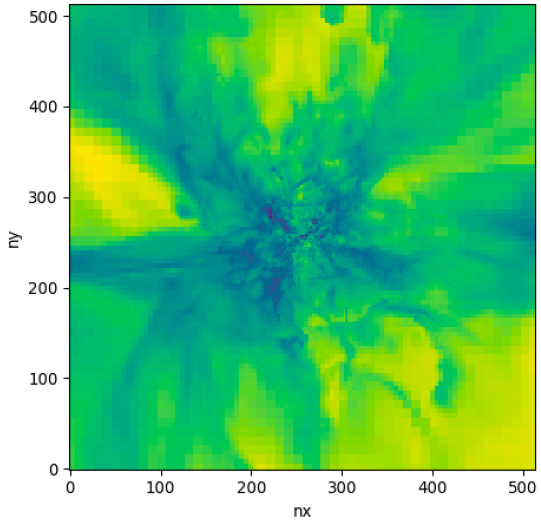
- NENER=0 implemented but not tested
- 5 hydro variables for NDIM=1, 2 and 3 and also for MHD
- NPSCAL=0 for additional passive scalars

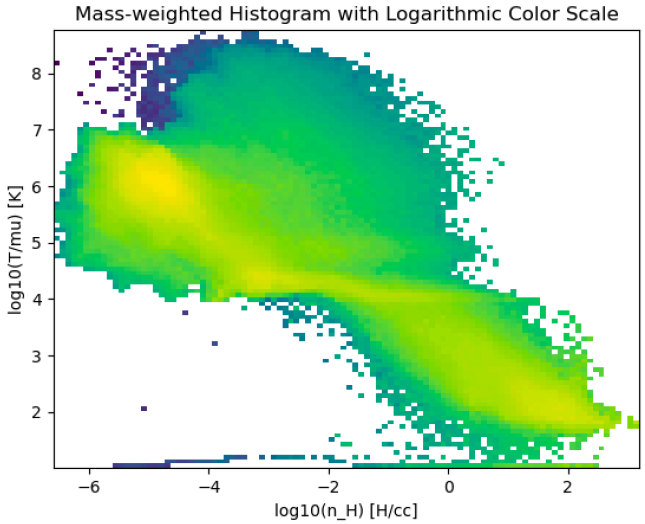
- Python package miniramses.py (reading and plotting)
- Python utils amr2img.py, map2img.py, run2img.py, debug.py, history.py, log.py, sfr.py…

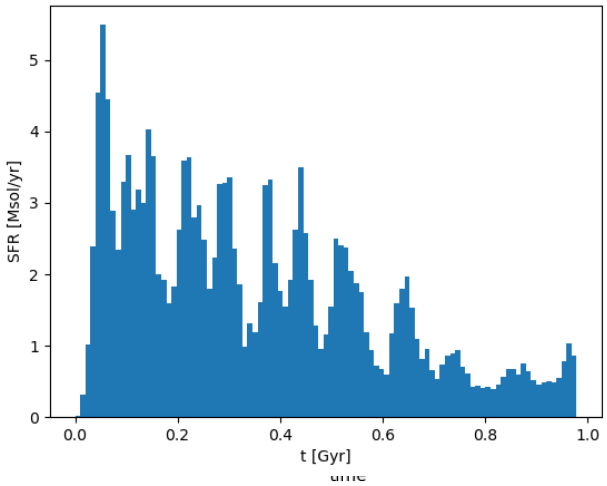
- F90 utils amr2map.f90 and part2map.f90

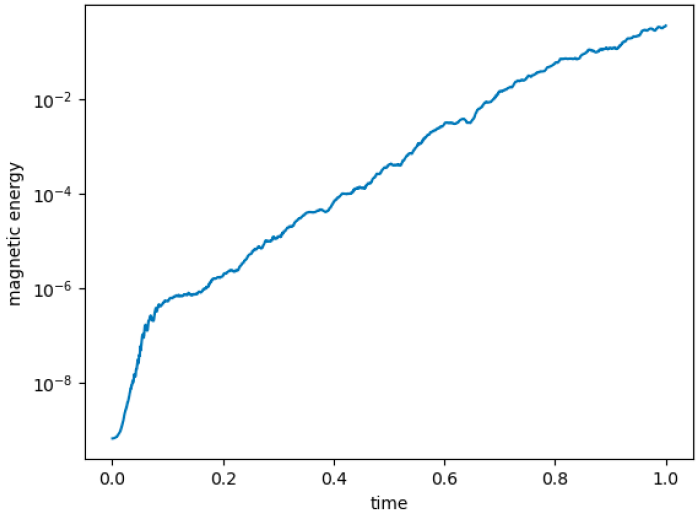
mini-ramses: galactic dynamo in a cooling halo



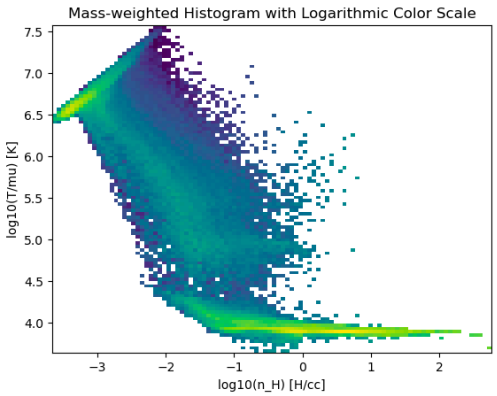


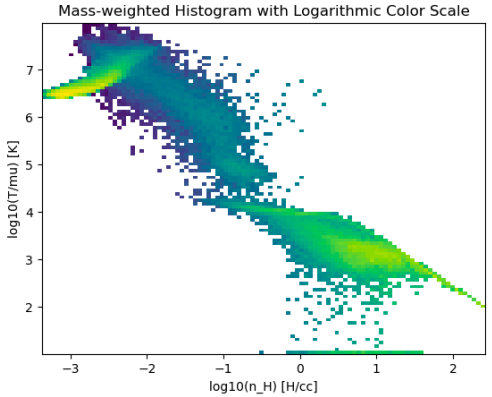


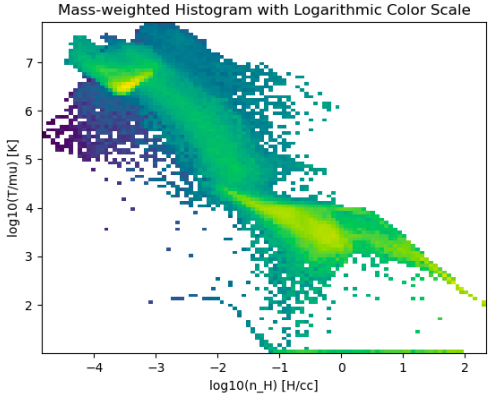


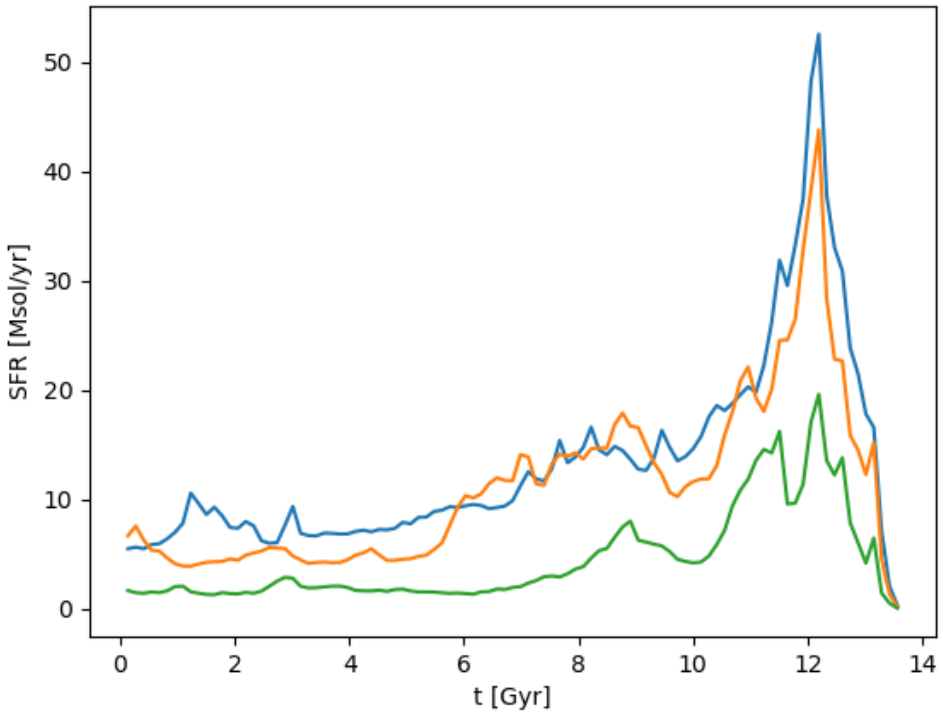


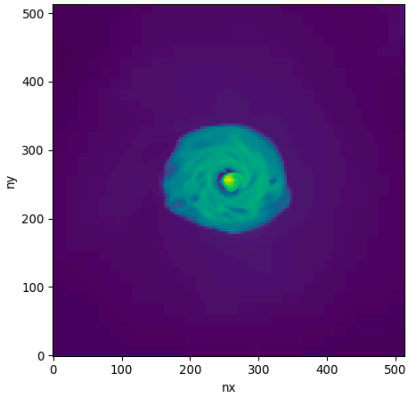
mini-ramses: cosmological galaxy formation

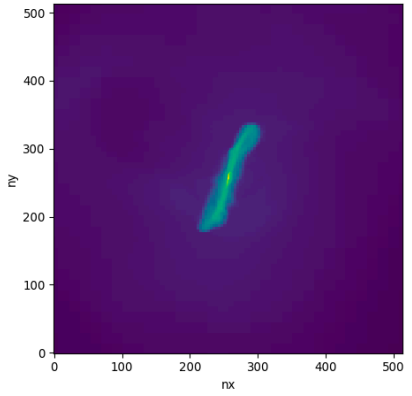












mini-ramses: software engineering

- https://bitbucket.org/rteyssie/mini-ramses

- GPU hydro kernel prototype developed in Nvidia Fortran by Bob Caddy (Princeton Research Software Engineer) in folder mini-ramses/gpu
- https://robertcaddy.com/posts/GPU-Mini-Ramses-Port/

- Princeton Open Hackathon with Nvidia in June 2025 to complete the hydro kernel (MPI and AMR) and explore options for the Poisson kernel.

- GPU kernel for RT should follow quickly (non-equilibrium chemistry)

- OpenMP/MPI implementation in progress

- Folder mini-ramses/doc