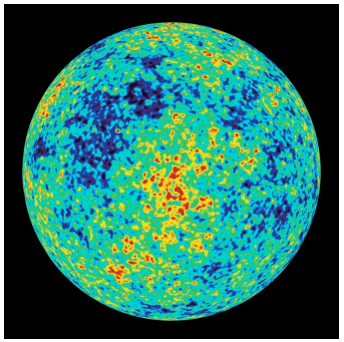
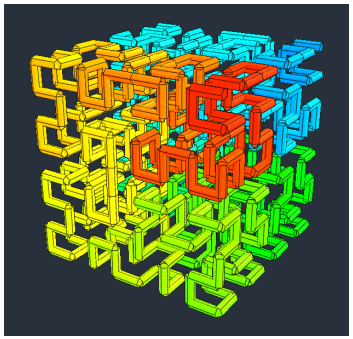
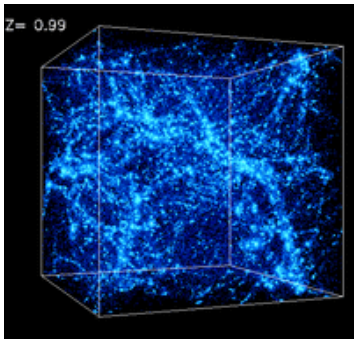
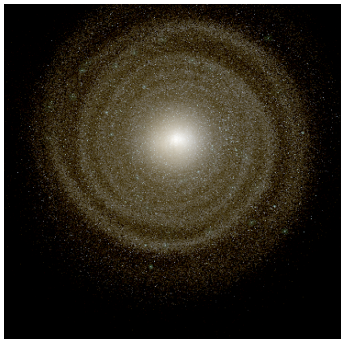
A new data structure for the RAMSES code: `mini_ramses`

with Andreas Bleuler, Claudio Gheller, Doug Potter, Joachim Stadel

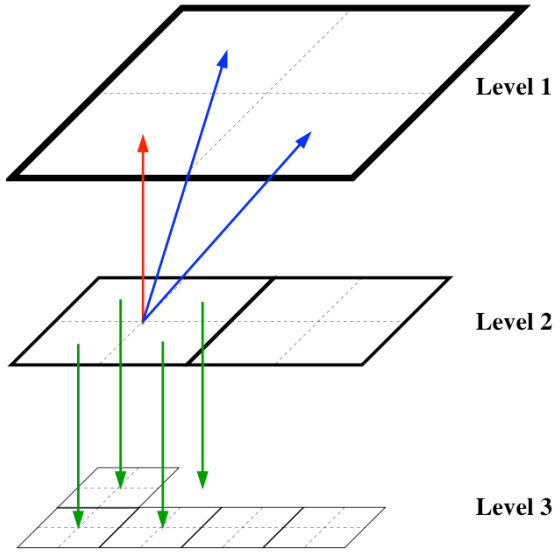
University of Zurich[UZH]



RAMSES: parallel Adaptive Mesh Refinement

- Graded octree structure: the cartesian mesh is refined on a cell by cell basis
- Full connectivity: each oct have direct access to neighbouring parent cells and to children octs (memory overhead 2 integers per cell).
- Optimise the mesh adaptivity to complex geometry but CPU overhead can be as large as 50%.



N body module: Particle-Mesh method on AMR grids. Poisson equation solved using a multigrid solver.

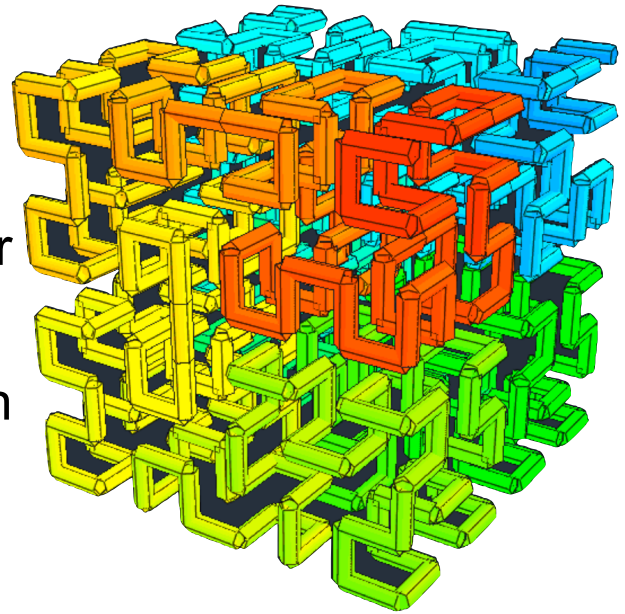
Hydro module: unsplit second order Godunov method (MUSCL) with various Riemann solvers and slope limiters. MHD solver with Constrained Transport.

Time integration: single time step or sub-cycling.

Other: radiative transfer, star formation, sink particles, stellar and AGN feedback

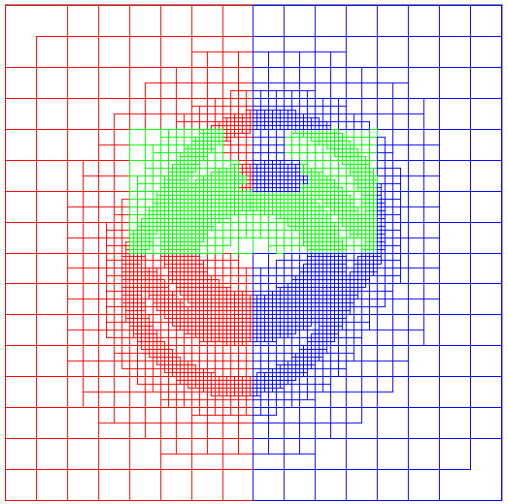
MPI-based parallel computing using time-dependent domain decomposition based on Peano-Hilbert cell ordering.

Download at **https://bitbucket.org/rteyssie/ramses**



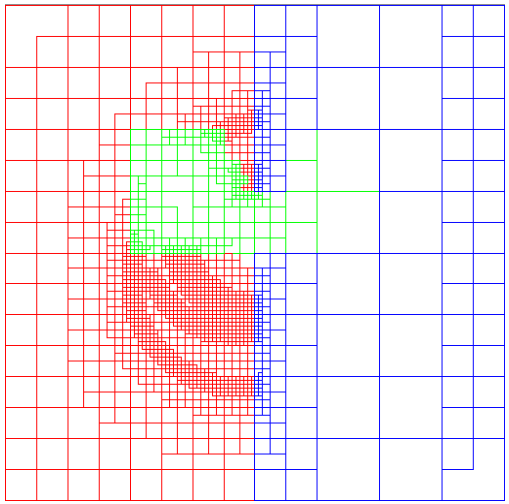
Locally essential trees

Salmon 90
Warren 92
Dubinski 96

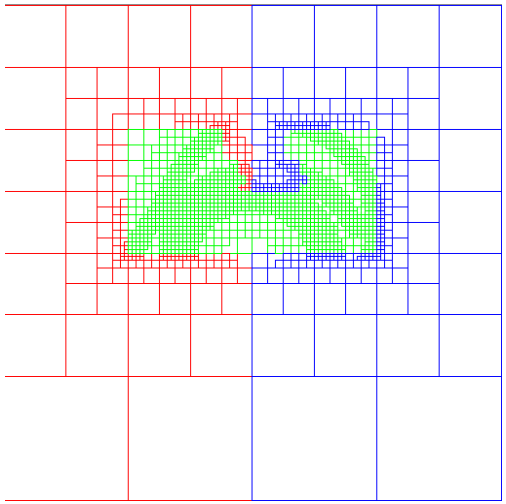


Domain decomposition over 3 processors

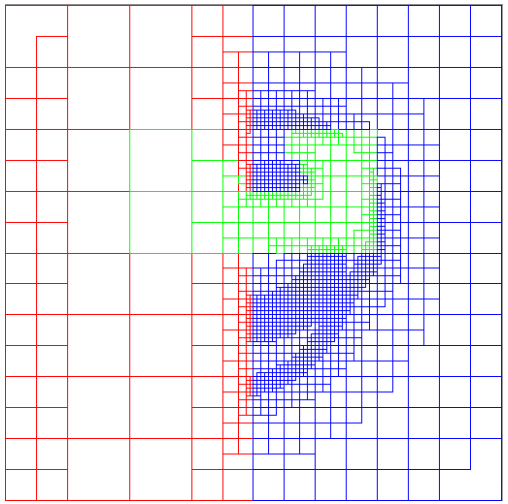
Each processor octree is surrounded by ghost cells (local copy of distant processor octrees) so that the resulting local octree contains all the necessary information.



Locally essential tree in processor #1



Locally essential tree in processor #2



Locally essential tree in processor #3

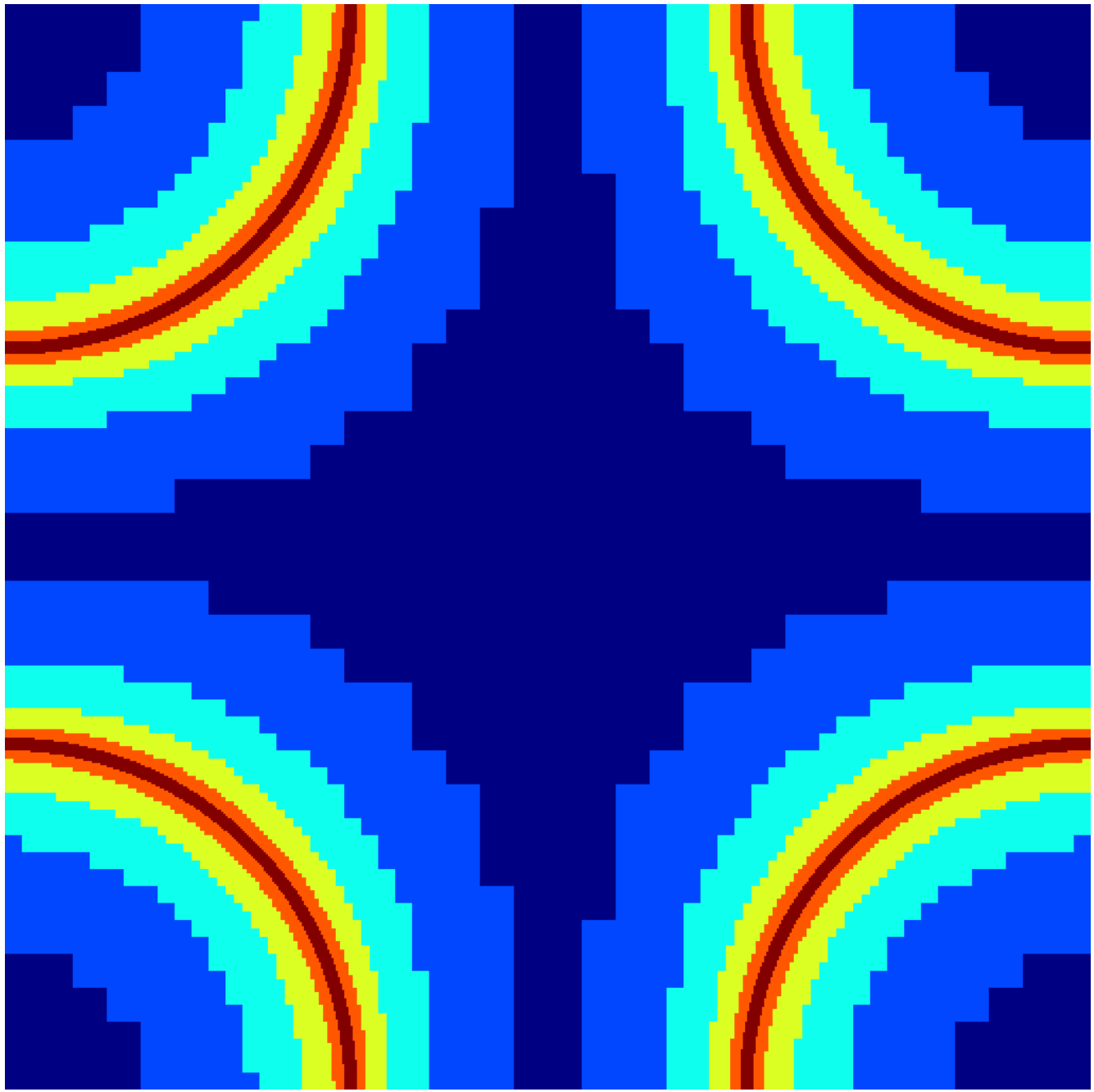
Load balancing issues in RAMSES

```
Mesh structure
Level  1 has        1 grids (      0,       1,       0,)
Level  2 has        8 grids (      0,       1,       0,)
Level  3 has       64 grids (      0,       4,       1,)
Level  4 has      512 grids (      0,      26,       8,)
Level  5 has     4096 grids (      0,     208,      64,)
Level  6 has    32768 grids (      0,    1655,     512,)
Level  7 has   262144 grids (      0,   13239,    4096,)
Level  8 has    26900 grids (      0,    2622,     420,)
Level  9 has    29518 grids (      0,    2604,     461,)
Level 10 has    36760 grids (      0,    3893,     574,)
Level 11 has    48542 grids (      0,    4933,     758,)
Level 12 has    69920 grids (      0,    4964,    1092,)
Level 13 has    98892 grids (      0,    5460,    1545,)
Level 14 has   126544 grids (      0,    6360,    1977,)
Level 15 has   130262 grids (      0,    6741,    2035,)
Level 16 has   101331 grids (      0,    9497,    1583,)
```

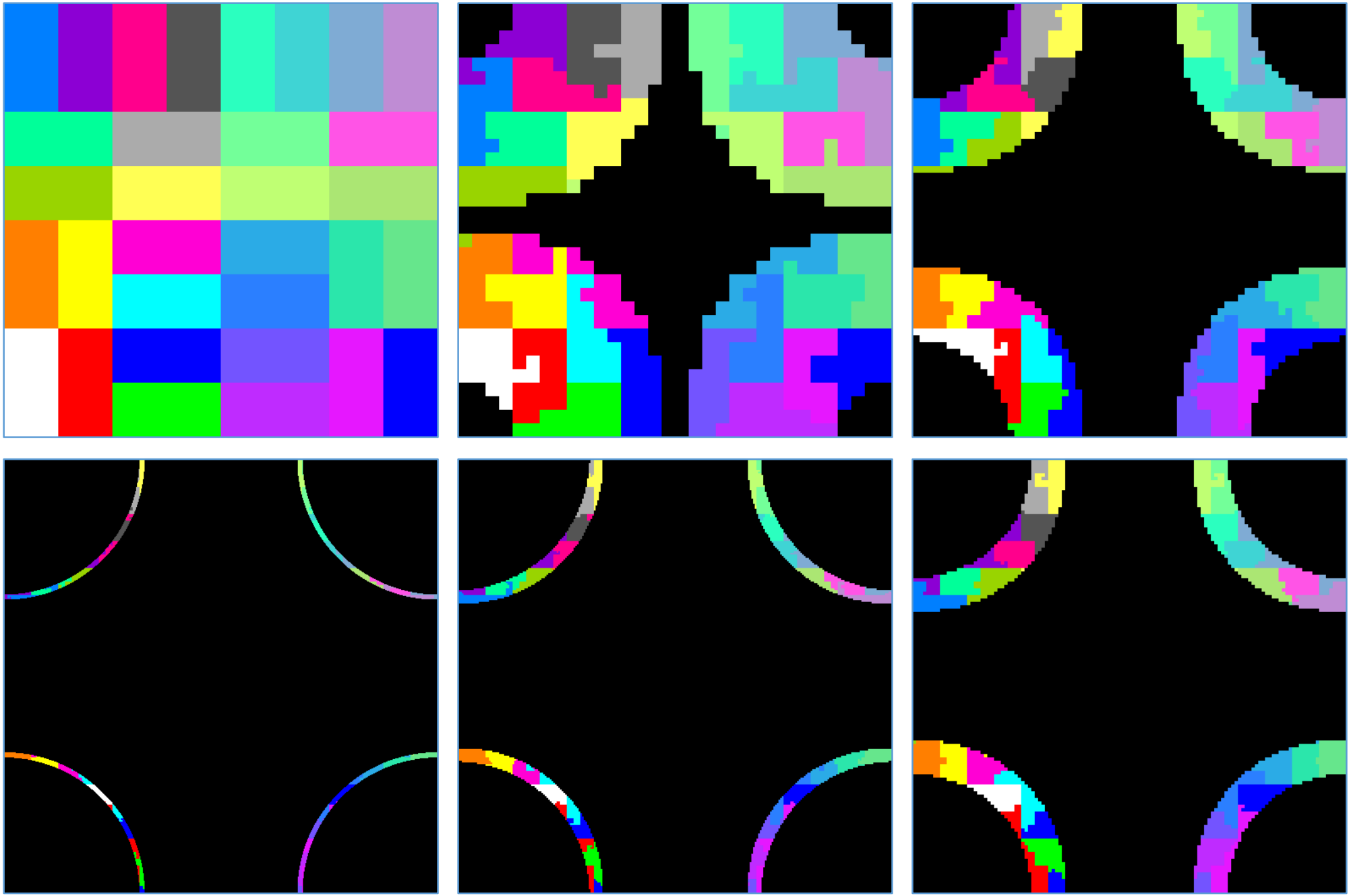
mini_ramses: a new strategy for an octree AMR

```
Mesh structure
Level  7 has     262144 grids (    4096,    4096,    4096,)
Level  8 has      26999 grids (     421,     422,     421,)
Level  9 has      27895 grids (     435,     436,     435,)
Level 10 has      30651 grids (     478,     479,     478,)
Level 11 has      37373 grids (     583,     584,     583,)
Level 12 has      51546 grids (     805,     806,     805,)
Level 13 has      70825 grids (    1106,    1107,    1106,)
Level 14 has      99427 grids (    1553,    1554,    1553,)
Level 15 has     118720 grids (    1855,    1855,    1855,)
Level 16 has     120961 grids (    1890,    1891,    1890,)
```

`mini_ramses`: a new strategy for an octree AMR



`mini_ramses`: a new strategy for an octree AMR



`mini_ramses`: a new strategy for an octree AMR

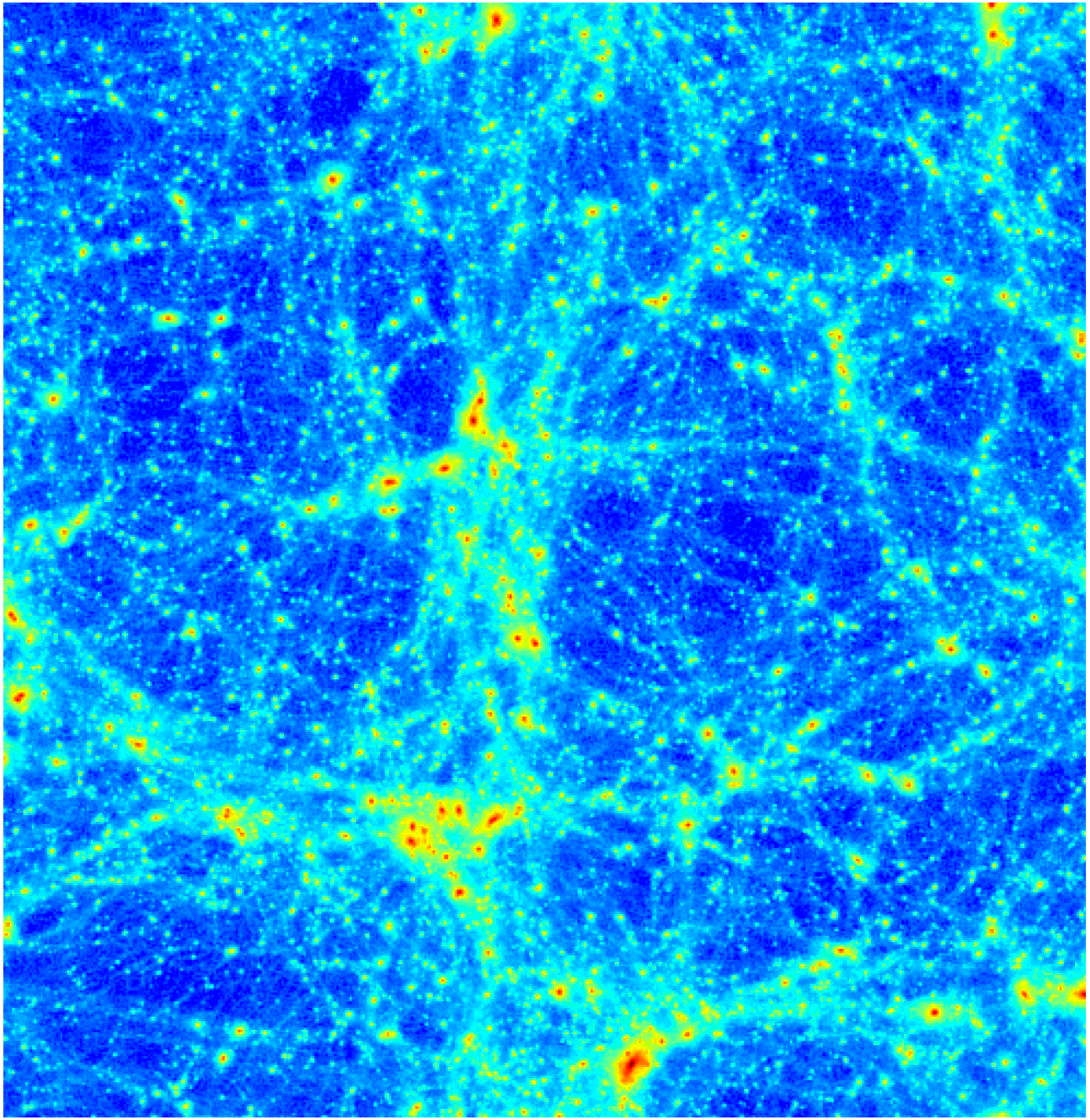
- New data structure based on sorted arrays of structures
- Oct is still the basic building block, but no more linked lists
- Octs are sorted per refinement level, and then per Hilbert key
- Perfect load balancing enforced at every time step and for each level
- Basic algorithm used: radix sort
- Octs are grouped in superocts if possible (larger Cartesian arrays)
- Particles are perfectly load balanced every step
- Particles are sorted per level and then per Hilbert key

- No more octree.
- Neighbouring cells are accessed in memory using a hash table
- Remote information is managed by a software cache (using the MDL library)
- Global cell-indexing uses a quadruplet `(l,i,j,k)`
- If quadruplet not in hash table, then convert quadruplet into a Hilbert key.
- If local, then cell does not exist.
- If remote, acquire the information using MPI short messages (cache line)

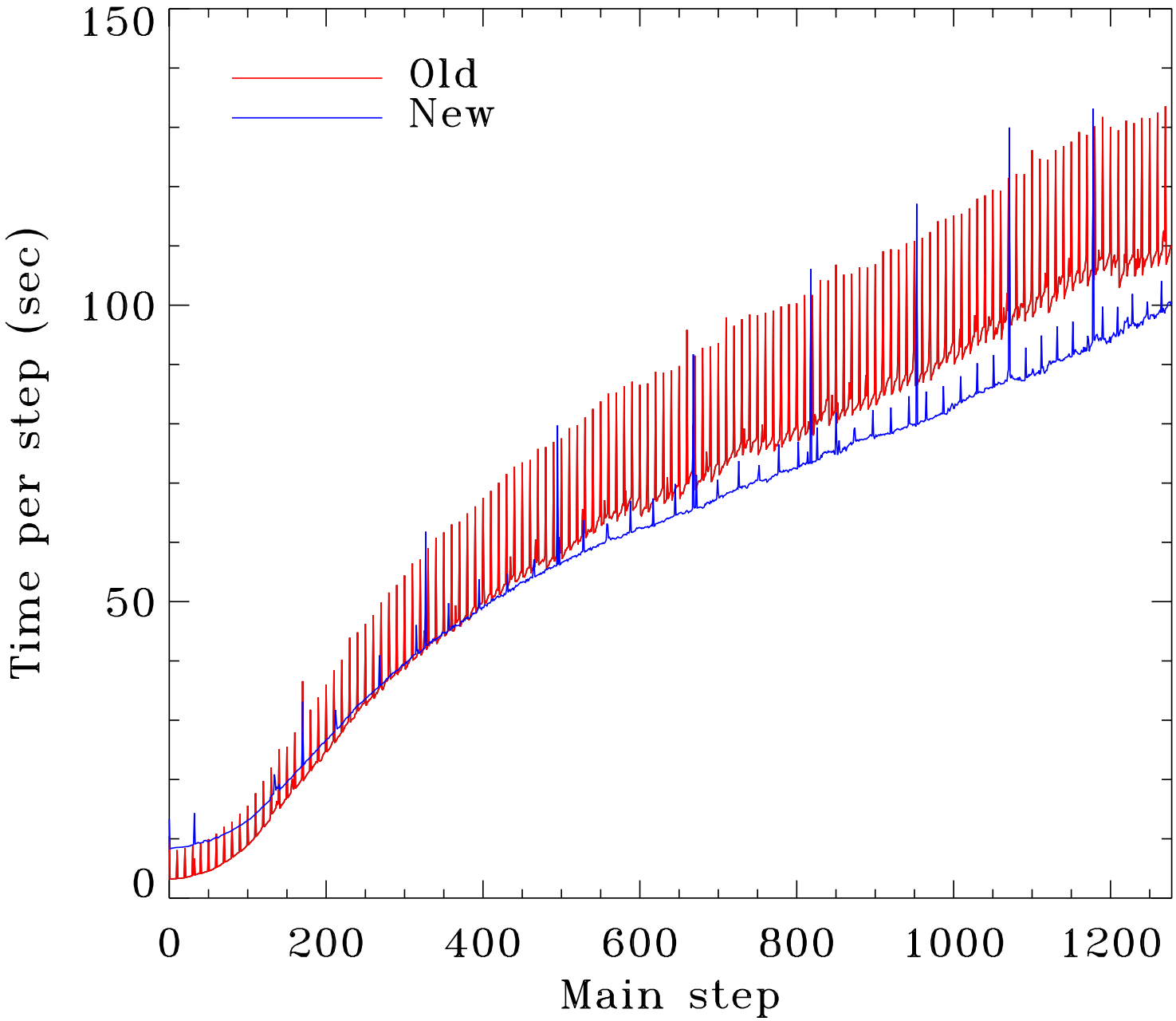
Sample code for the PIC solver

```
! Open write-only cache for array rho
hash_nbor(0)=ilevel+1
call open_cache(operation_rho,domain_decompos_amr)

! Loop over particles in Hilbert order
do i=headp(ilevel),tailp(nlevelmax)
   ipart=sortp(i)

   ! Rescale particle position at level ilevel
   do idim=1,ndim
      x(idim)=xp(ipart,idim)/dx_loc
   end do

   ! CIC at level ilevel (dd: right cloud boundary;
   do idim=1,ndim
      dd(idim)=x(idim)+0.5D0
      id(idim)=dd(idim)
      dd(idim)=dd(idim)-id(idim)
      dg(idim)=1.0D0-dd(idim)
      ig(idim)=id(idim)-1
   end do

   ! Periodic boundary conditions
   do idim=1,ndim
      if(ig(idim)<0)ig(idim)=ckey_max(ilevel+1)-1
      if(id(idim)==ckey_max(ilevel+1))id(idim)=0
   enddo

   ! Compute cloud volumes
   vol(1)=dg(1)*dg(2)*dg(3)
   vol(2)=dd(1)*dg(2)*dg(3)
   vol(3)=dg(1)*dd(2)*dg(3)
   vol(4)=dd(1)*dd(2)*dg(3)
   vol(5)=dg(1)*dg(2)*dd(3)
   vol(6)=dd(1)*dg(2)*dd(3)
   vol(7)=dg(1)*dd(2)*dd(3)
   vol(8)=dd(1)*dd(2)*dd(3)

   ! Compute cells Cartesian key
   ckey(1:3,1)=(/ig(1),ig(2),ig(3)/)
   ckey(1:3,2)=(/id(1),ig(2),ig(3)/)
   ckey(1:3,3)=(/ig(1),id(2),ig(3)/)
   ckey(1:3,4)=(/id(1),id(2),ig(3)/)
   ckey(1:3,5)=(/ig(1),ig(2),id(3)/)
   ckey(1:3,6)=(/id(1),ig(2),id(3)/)
   ckey(1:3,7)=(/ig(1),id(2),id(3)/)
   ckey(1:3,8)=(/id(1),id(2),id(3)/)

   ! Update mass density
   do ind=1,twotondim
      hash_nbor(1:ndim)=ckey(1:ndim,ind)
      ! Get parent cell using write-only cache
      parent_cell=get_parent_cell(hash_nbor,grid_dict,.true.,.false.)
      if(parent_cell>0)then
         igrid=(parent_cell-1)/twotondim+1
         icell=parent_cell-(igrid-1)*twotondim
         vol2=mp(ipart)*vol(ind)/vol_loc
         grid(igrid)%rho(icell)=grid(igrid)%rho(icell)+vol2
      endif
   end do

end do
! End loop over particles

call close_cache(grid_dict)
```

mini_ramses: early results for cosmology



mini_ramses: timing results for cosmology



mini_ramses: timing results for cosmology

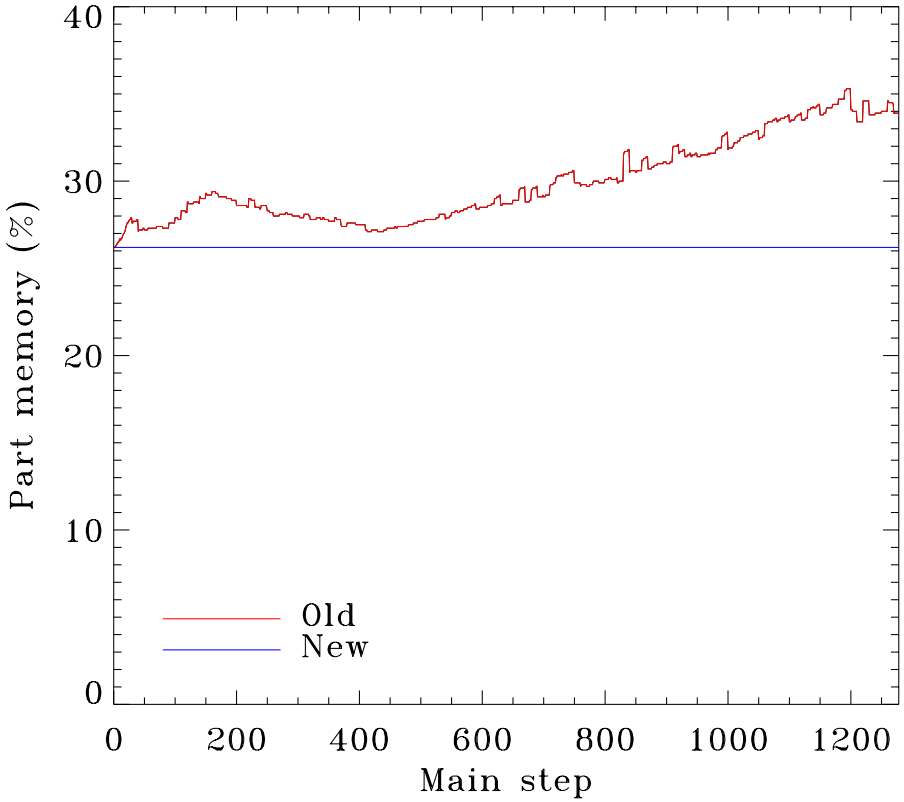
mini_ramses

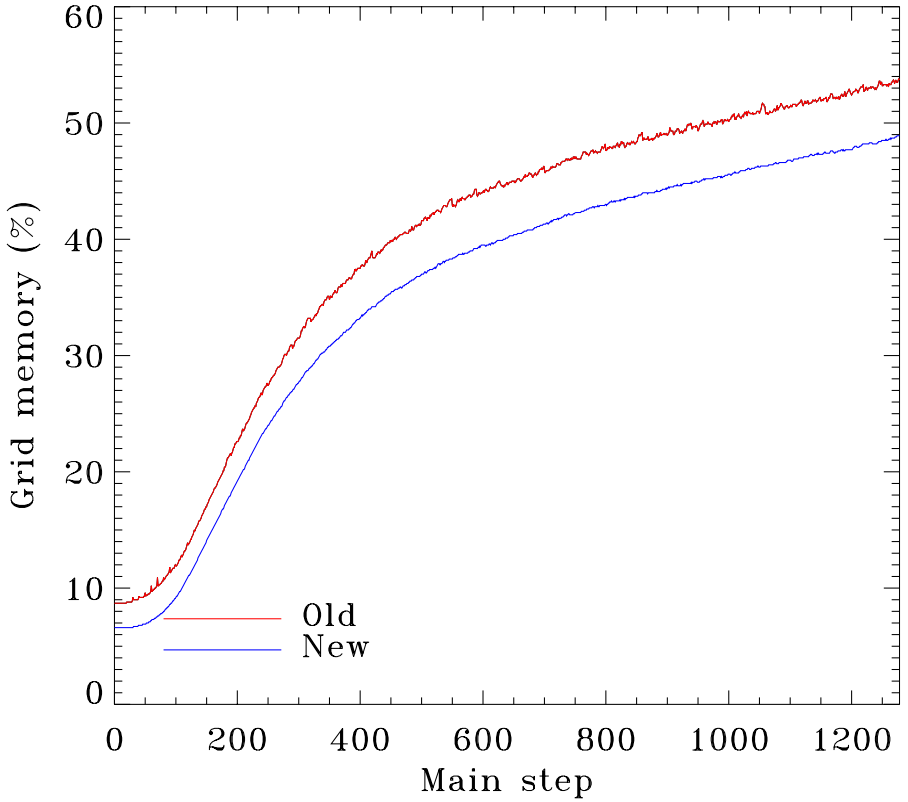
minimum	average	maximum	standard dev	std/av	%	TIMER
51854.983	51855.163	51855.295	0.096	0.000	66.3	poisson
13296.873	13324.774	13372.833	24.369	0.002	17.0	particles
3757.890	3806.317	3834.963	24.518	0.006	4.9	rho
6146.099	6146.457	6146.834	0.194	0.000	7.9	flag
1682.109	1682.551	1682.584	0.114	0.000	2.2	refine
1338.918	1339.660	1340.096	0.483	0.000	1.7	load balance
78181.700					100.0	TOTAL

old_ramses

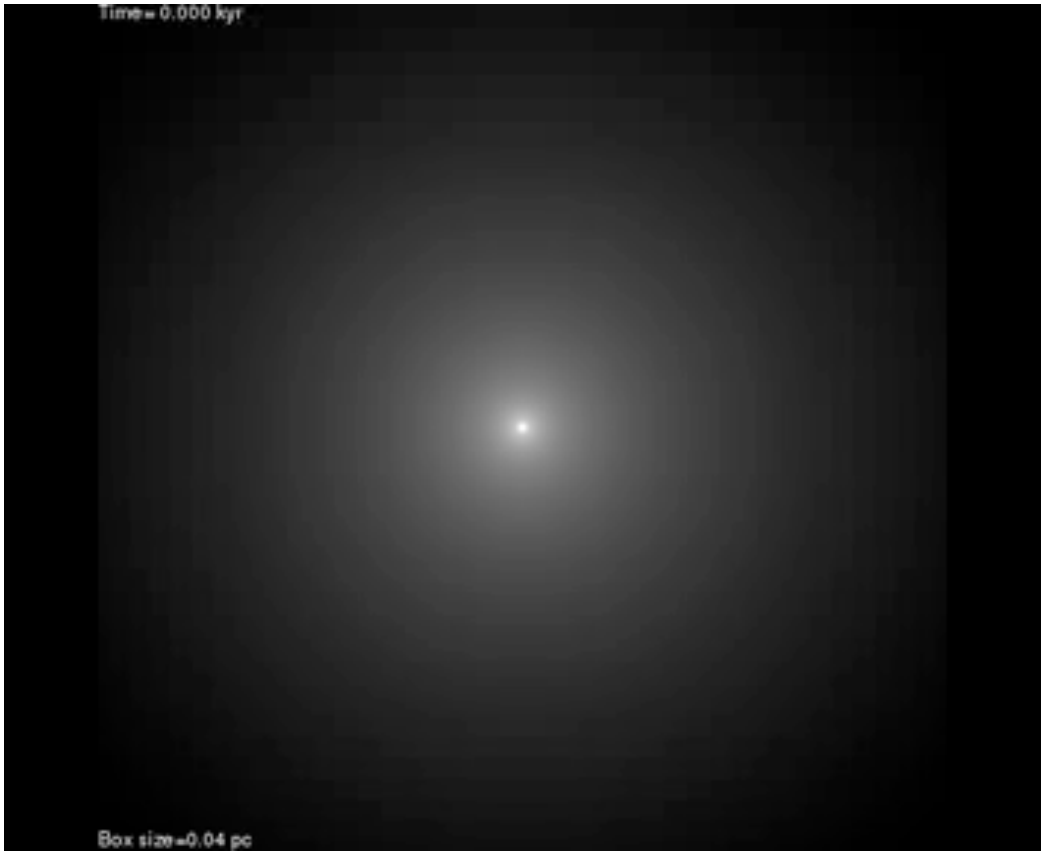
minimum	average	maximum	standard dev	std/av	%	TIMER
69344.377	69346.724	69348.789	1.291	0.000	82.4	poisson
3883.513	4550.793	5619.058	449.243	0.099	5.4	particles
2494.461	2870.349	3106.273	151.909	0.053	3.4	rho
2548.103	2548.106	2548.107	0.001	0.000	3.0	load balance
2610.543	2909.316	3076.730	139.194	0.048	3.5	flag
997.104	1060.769	1110.828	32.557	0.031	1.3	refine
84168.286	100.0	TOTAL				

mini_ramses: optimal memory balance

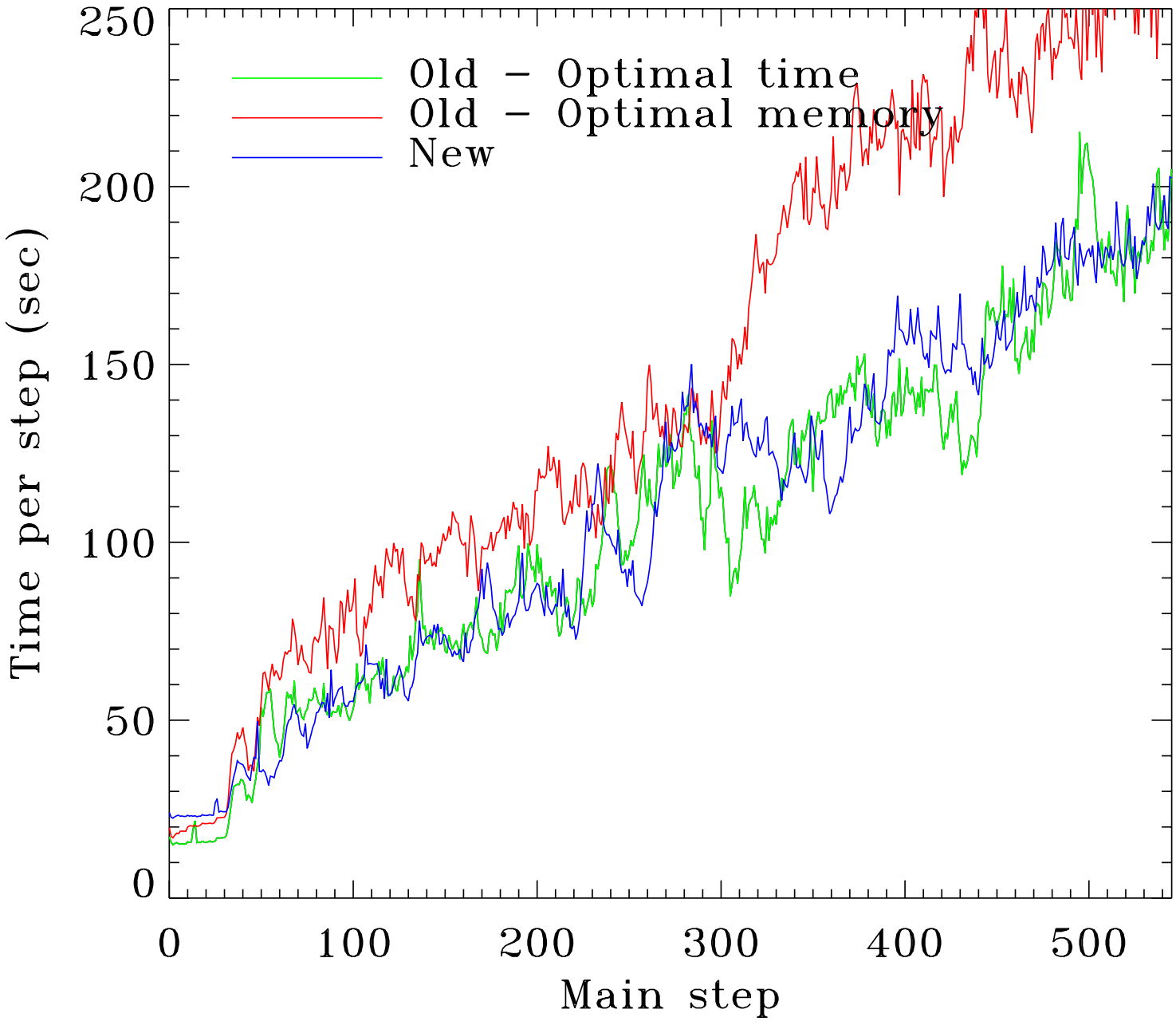




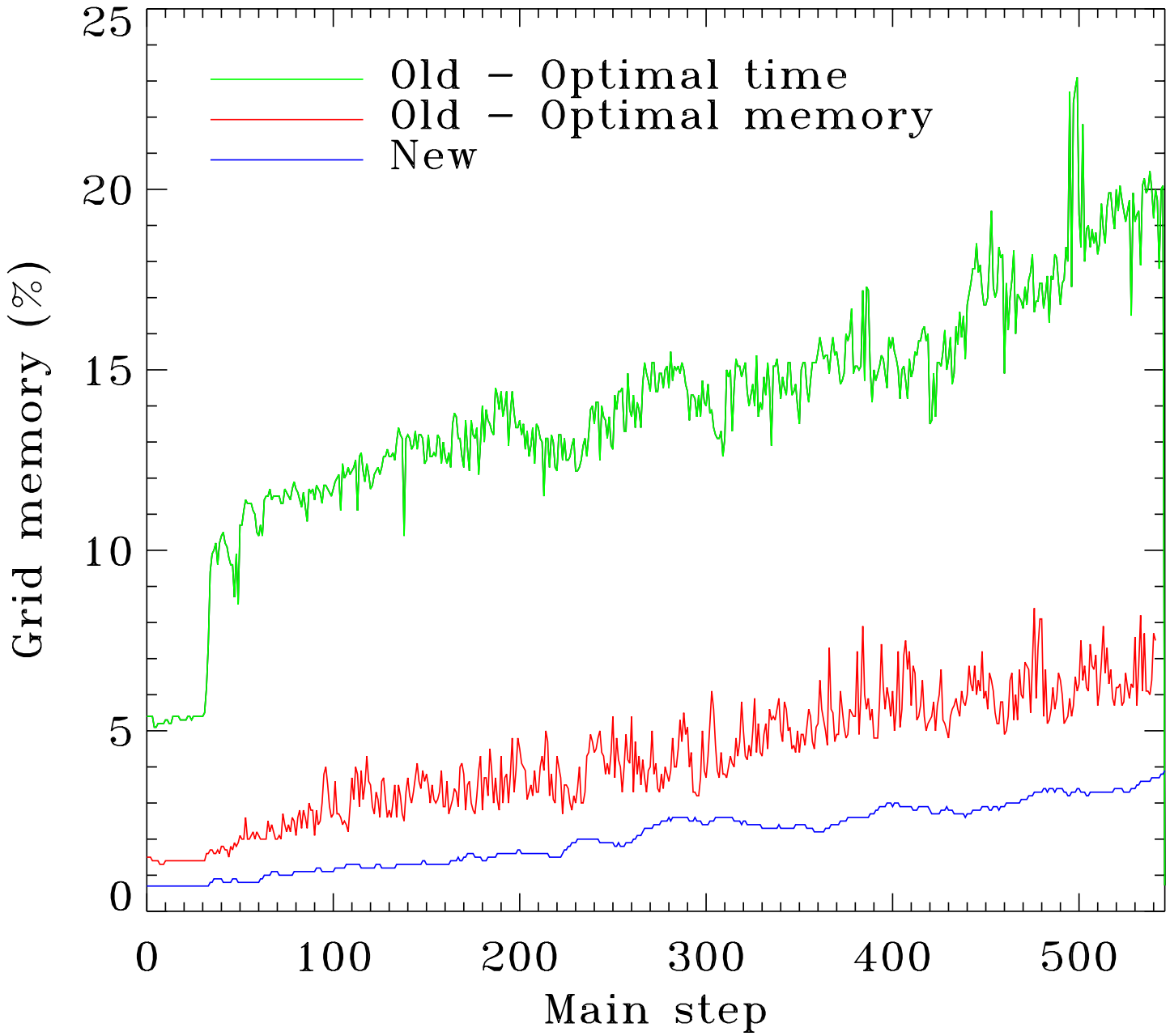
mini_ramses: early results for star formation



mini_ramses: early results for star formation



mini_ramses: early results for star formation



Next steps

- Optimisation (cache, data structure, GPU data loading…)

  Early results with superocts.
  Test performed: Sedov 3D point explosion with 256^3 grid points and 8 proc

  `old_ramses`:

  ```
  2x2x2:              2.8 sec/step
  ```

  `mini_ramses`:

  ```
  2x2x2:              6.8 sec/step
  4x4x4:              4.2 sec/step
  8x8x8:              2.1 sec/step
  ```

- Integration with the MDL library

- Import other physics modules into `mini_ramses`