Ramses

Version

Table of Contents

User Documentation

Ramses Documentation

The *ramses* code is intended to be a versatile platform to develop applications using Adaptive Mesh Refinement (AMR) for computational astrophysics. The current implementation allows solving the classical and relativistic Euler equations in presence of self-gravity, magnetic field and radiation field. The *ramses* code can be used on massively parallel architectures, if properly linked to the MPI library. It can also be used on single processor machines without MPI. Output data are generated using Fortran unformatted files. A suite of post-processing routines is delivered within the present release, allowing the user to perform a simple analysis of the generated output files. A pdf version of this documentation can be found here .

Table of Contents

Getting started

In this chapter, we will explain step by step how to get the RAMSES package and install it, then how to perform a simple test to check the installation.

Obtaining the package

The package can be downloaded from the GitHub repository using git:

```
$ git clone https://github.com/ramses-organisation/ramses
```

This will create a new repository called `ramses/` In this directory, you will see:

```
$ ls -F
README          bin/            mhd/            patch/          rt/
amr/            doc/            namelist/       pm/             utils/
aton/           hydro/          pario/          poisson/
```

Each directory contains a set of files with a given common purpose. For example, `amr/` contains all Fortran 90 routines dealing with the AMR grid management and MPI communications, while `hydro/` obviously contains all Fortran 90 routines dealing with hydrodynamics. The first directory you are interested in is the `bin/` directory, in which the code will be compiled.

Compiling the code

You need to go first to the `bin/` directory:

```
$ cd trunk/ramses/bin
$ ls -F
Makefile        Makefile.rt
```

We will use the first `Makefile` to compile the code. The first thing to do is to edit the `Makefile` and modify the two variables `F90` and `FFLAGS` . Several examples corresponding to different Fortran compilers are given. The default values are:

```
F90 = gfortran -O3 -frecord-marker=4 -fbacktrace -ffree-line-length-
none
FFLAGS = -x f95-cpp-input -DWITHOUTMPI $(DEFINES)
```

The first variable is obviously the command used to invoke the Fortran compiler. In this case, this is the GNU Fortran compiler . The second variable contains Fortran compilation flags and preprocessor directives. The first directive, `-DWITHOUTMPI` , switches off all MPI routines. On the other hand, if you don't use this directive, the code must be linked to the MPI library. We will discuss this point later.

Other preprocessor directives are defined in variable `DEFINES` in the `Makefile` :

```
# Compilation time parameters
NVECTOR = 64
NDIM = 3
NPRE = 8
NVAR = 8
SOLVER = hydro
PATCH =
EXEC = ramses
DEFINES = -DNVECTOR=$(NVECTOR) -DNDIM=$(NDIM) -DNPRE=$(NPRE) -DNVAR=$
(NVAR) -DSOLVER$(SOLVER)
```

These additional directives are called *Compilation Time Parameters* . They should be defined in the `Makefile` and the code must be recompiled entirely using:

```
$ make clean
$ make
```

We list now the definitions of these parameters.

`NVECTOR=64` : This parameter is used to set the vector size for computation-intensive operations. It must be determined experimentally on each new hardware.

`NPRE=4` : This parameter sets the precision of the floating point operations. `NPRE=4` stands for single precision arithmetics, while `NPRE=8` is for double precision.

`NENER=0` : This parameter sets the number of energy variables used in the hydro or mhd solver.

`NDIM=3` : This parameter sets the dimensionality of the problem. The value `NDIM=1` is for 1D, plan-parallel flows. `NDIM=2` and `NDIM=3` are resp. for 2D and 3D flows.

`SOLVER=hydro` : This parameter selects the type of hyperbolic solver used. Possible values are: `hydro` for the adiabatic Euler equations, `mhd` , the Constrained Transport scheme for the ideal MHD equations. and `rhd` for relativistic hydro.

`NVAR=8` : This parameters defines the number of variables in the hyperbolic solver. For `SOLVER=hydro` , `NVAR>=NDIM+2` . For `SOLVER=mhd` , `NVAR>=8` and for `SOLVER=rhd` , one has `NVAR>=5` .

Our goal is now to compile the code for a simple one-dimensional problem. You need to modify the `Makefile` so that:

```
NDIM=1
SOLVER=hydro
NVAR=3
```

Then type:

```
$ make
```

If everything goes well, all source files will be compiled and linked into an executable called `ramses1d` .

Additional compilation preprocessor flags

`-DTSC` : This parameter sets the triangular shape cloud approximation; only works with `NDIM=3`

`-DOUTPUT_PARTICLE_POTENTIAL` : This parameter forces the code to output particle potentials at snapshots

`-DQUADHILBERT` : This parameter sets longer Hilbert curve necessary if `levelmax>19`

`-DLONGINT` : This parameter switches to long ints (necessary when one has lots of particles)

`-DNOSYSTEM` : This parameter handles operating system commands

Executing the test case

To test the compilation, you need to execute a simple test case. Go up one level and type the following command:

```
$ cd trunk/ramses
$ bin/ramses1d namelist/tube1d.nml
```

The first part of the command is the executable we have just compiled. The second part, the only command line argument, is an input file containing the *run time parameters* . Several examples of such parameter files are stored in the `namelist/` directory. The namelist file we have just used `tube1d.nml` is the Sod test, a simple shock tube simulation in 1D. For comparison, we now show the last 14 lines of the standard output:

```
 Mesh structure
 Level  1 has           1 grids (       1,       1,       1,)
 Level  2 has           2 grids (       2,       2,       2,)
 Level  3 has           4 grids (       4,       4,       4,)
 Level  4 has           8 grids (       8,       8,       8,)
 Level  5 has          16 grids (      16,      16,      16,)
 Level  6 has          27 grids (      27,      27,      27,)
 Level  7 has          37 grids (      37,      37,      37,)
 Level  8 has          17 grids (      17,      17,      17,)
 Level  9 has          16 grids (      16,      16,      16,)
 Level 10 has          13 grids (      13,      13,      13,)
 Main step=    43 mcons=-1.97E-16 econs= 1.61E-16 epot= 0.00E+00
ekin= 1.38E+00
 Fine step=   688 t= 2.45047E-01 dt= 3.561E-04 a= 1.000E+00 mem= 7.6%
 Run completed
```

If your execution looks similar, it means your installation was successfull. Users are encouraged to redirect the standard output into a *log file* . This log file contains all simulation control variables, as well as output variables, but for 1D simulations only.

```
$ bin/ramses1d namelist/tube1d.nml > tube.log
```

Reading the log file

We will now briefly describe the structure and the nature of the information available in the log file. We will use as example the file `tube.log` we have just created. It should contain, starting from the top:

```
 _/_/_/       _/_/     _/    _/    _/_/_/   _/_/_/_/    _/_/_/
 _/    _/    _/  _/    _/_/_/_/   _/    _/  _/         _/    _/
 _/    _/   _/    _/   _/ _/ _/   _/        _/         _/
 _/_/_/     _/_/_/_/   _/    _/     _/_/    _/_/_/       _/_/
 _/    _/   _/    _/   _/    _/         _/  _/              _/
 _/    _/   _/    _/   _/    _/   _/    _/  _/         _/    _/
 _/    _/   _/    _/   _/    _/    _/_/_/   _/_/_/_/    _/_/_/
                        Version 3.0
        written by Romain Teyssier (CEA/DSM/IRFU/SAP)
                        (c) CEA 1999-2007
```

```
 Working with nproc =     1 for ndim = 1
 Using the hydro solver with nvar =  3

 Building initial AMR grid
 Initial mesh structure
 Level  1 has          1 grids (       1,       1,       1,)
 Level  2 has          2 grids (       2,       2,       2,)
 Level  3 has          4 grids (       4,       4,       4,)
 Level  4 has          8 grids (       8,       8,       8,)
 Level  5 has          8 grids (       8,       8,       8,)
 Level  6 has          8 grids (       8,       8,       8,)
 Level  7 has          8 grids (       8,       8,       8,)
 Level  8 has          8 grids (       8,       8,       8,)
 Level  9 has          6 grids (       6,       6,       6,)
 Level 10 has          4 grids (       4,       4,       4,)
 Starting time integration
 Output    58 cells
 ================================================
 lev      x           d           u           P
  4  3.12500E-02  1.000E+00  0.000E+00  1.000E+00
  4  9.37500E-02  1.000E+00  0.000E+00  1.000E+00
...
  4  9.06250E-01  1.250E-01  0.000E+00  1.000E-01
  4  9.68750E-01  1.250E-01  0.000E+00  1.000E-01
 ================================================
 Fine step=     0 t= 0.00000E+00 dt= 6.603E-04 a= 1.000E+00 mem= 3.2%
 Fine step=     1 t= 6.60250E-04 dt= 4.420E-04 a= 1.000E+00 mem= 3.2%
```

After the code banner and copyrights, the first line indicates that you are currently using 1 processor and 1 space dimension for this run. The second line confirms the solver used and the number of variables defined for this run. The code then reports that it is building the initial AMR grid. The next lines give the resulting mesh structure.

The first level of refinement in *ramses* covers the whole computational domain with 2 (resp. 4 and 8) cells in 1 (resp. 2 and 3) space dimension. The grid is then entirely refined up to `levelmin` , which in this case is defined in the parameter file to be `levelmin=3` . This defines the *coarse grid* . The grid is then adaptively refined up to `levelmax` , which in this case `levelmax=10` . Each line in the log file indicates the number of octs (or grids) at each level of refinement. The maximum number of grids in each level `level` is equal to `2**(level-1)` for `NDIM=1` , to `4**(level-1)` for `NDIM=2` and to `8**(level-1)` for `NDIM=3` .

The numbers inside parentheses give the minimum, maximum and average number of grids per processor. This is obviously only relevant to parallel runs.

The code then indicates that the time integration starts. After outputting the initial conditions to screen, the first *control line* appears, starting with the words `Fine step=` . The control line gives information on each *fine step* , its current number, its current time coordinate, its current

time step. Variable `a` is for cosmology runs only and gives the current expansion factor. The last variable is the percentage of allocated memory currently used by ramses to store each flow variable on the grid.

In ramses, adaptive time stepping is implemented, which results in defining *coarse steps* and *fine steps* . Coarse steps correspond to the coarse grid, which is defined by variable `levelmin` . Fine steps correspond to finer levels, for which the time step has been recursively subdivided by a factor of 2. Fine levels are sub-cycled, twice as more as their parent coarse level. This explains why, at the end of the log file, only 43 coarse steps are reported (1 through 43), for 689 fine steps (numbered from 0 to 688).

When a coarse step is reached, the code writes in the log file the current mesh structure. A new control line then appears, starting with the words `Main step=` . This control line gives information on each coarse step, namely its current number, the current error in mass conservation within the computational box `mcons=` , the current error in total energy conservation `econs=` , the gravitational potential energy and the fluid total energy (kinetic plus thermal).

This constitutes the basic information contained in the log file. In 1D simulations, output data are also written to standard output, and thus to the log file. For 2D and 3D, output data are stored into unformatted Fortran binary files (named `output_00001` , `output_00002` ...). In our example, the fluid variables are listed using 5 columns: level of refinement, position of the cell, density, velocity and pressure:

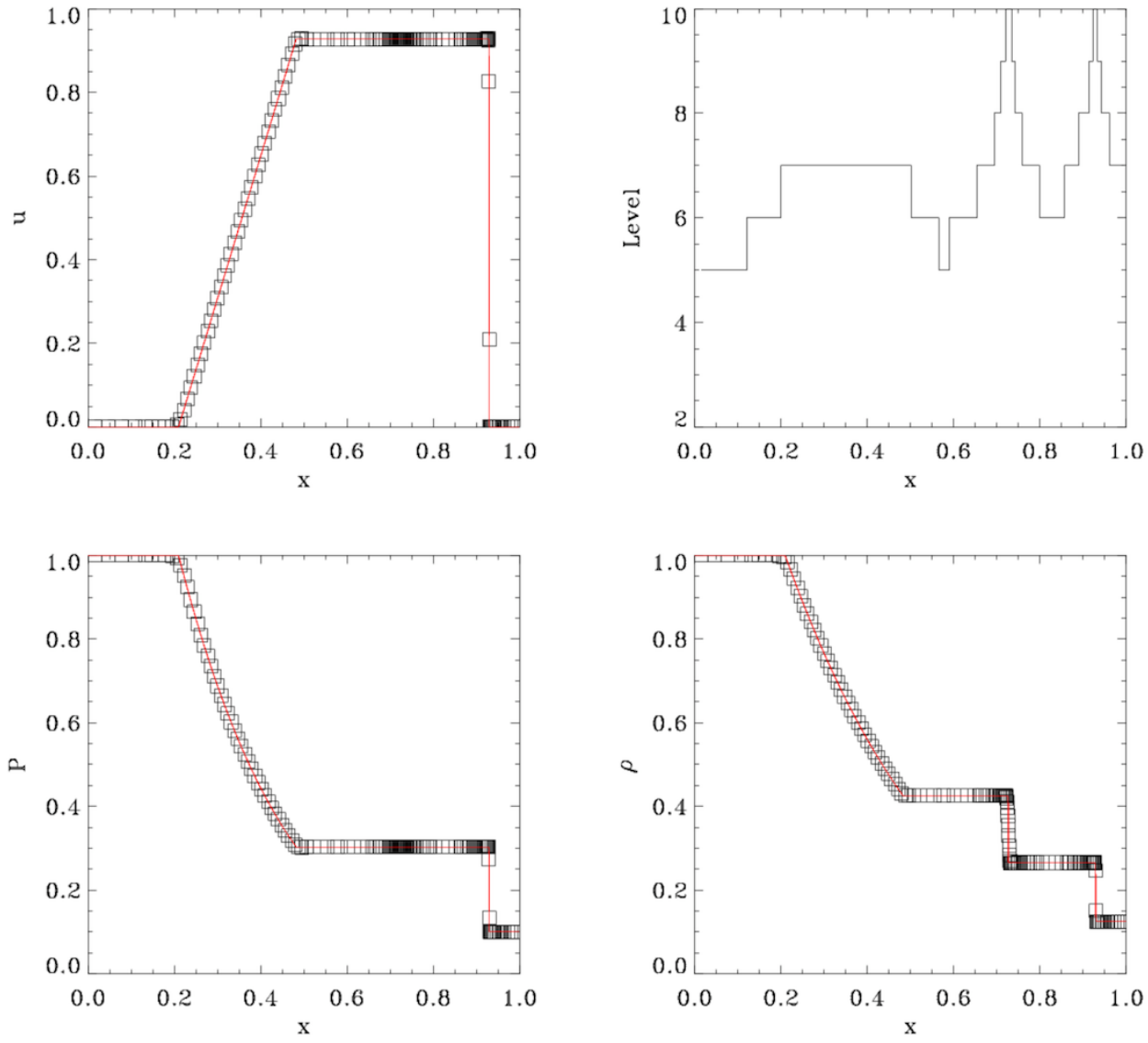
```
Output   142 cells
================================================
lev      x           d          u          P
 5  1.56250E-02  1.000E+00  0.000E+00  1.000E+00
 5  4.68750E-02  1.000E+00  0.000E+00  1.000E+00
 5  7.81250E-02  1.000E+00  0.000E+00  1.000E+00
 5  1.09375E-01  1.000E+00  1.564E-09  1.000E+00
 6  1.32812E-01  1.000E+00  2.112E-08  1.000E+00
```

You can cut and paste the 142 lines into another file and use your favorite data viewer like `xmgrace` or `gnuplot` to visualize the results. These should be compared to the plots shown on the figure below. If you have obtained comparable numerical values and levels of refinements, your installation is likely to be valid. You are encouraged to edit the parameter file `tube1d.nml` and play around with other parameter values, in order to test the code performances. You can also use other parameter files in the `namelist/` directory

If you would like to run a 2D simulation (using file `sedov2d.nml` for example), do not forget to recompile entirely the code using:

```
$ cd trunk/ramses/bin
$ make clean
$ make NDIM=2
```

This last image shows the numerical results obtained with ramses for the Sod shock tube test (symbols) compared to the analytical solution (red line).



Runtime Parameters

The Ramses parameter file is based on the Fortran namelist syntax. The Sod test parameter file is shown below, as it should appear if you edit it.

```
$ more tube1d.nml
This is the RAMSES parameter file for Sod''s shock tube test.

&RUN_PARAMS
hydro=.true.
nsubcycle=3*1,2
/

&AMR_PARAMS
levelmin=3
levelmax=10
ngridmax=2000
nexpand=1
boxlen=1.0
/

&BOUNDARY_PARAMS
nboundary=2
ibound_min=-1,+1
ibound_max=-1,+1
bound_type= 1, 1
/

&INIT_PARAMS
nregion=2
region_type(1)='square'
region_type(2)='square'
x_center=0.25,0.75
length_x=0.5,0.5
d_region=1.0,0.125
u_region=0.0,0.0
p_region=1.0,0.1
/

&OUTPUT_PARAMS
noutput=1
tout=0.245
/

&HYDRO_PARAMS
gamma=1.4
courant_factor=0.8
slope_type=2
riemann='hllc'
/

&REFINE_PARAMS
err_grad_d=0.05
err_grad_u=0.05
err_grad_p=0.05
interpol_var=0
```

```
interpol_type=2
/
```

This parameter file is organized in namelist blocks. Each block starts with &BLOCK_NAME and ends with the character "/". Within each block, you can specify parameter values using standard Fortran namelist syntax. There are currently 11 different parameter blocks implemented in RAMSES.

4 parameters blocks are mandatory and must always be present in the parameter file. These are `&RUN_PARAMS` , `&AMR_PARAMS` , `&OUTPUT_PARAMS` and `&INIT_PARAMS` . The 8 other blocks are optional. They must be present in the file only if they are relevant to the current run. These are `&BOUNDARY_PARAMS` , `&HYDRO_PARAMS` , `&PHYSICS_PARAMS` , `&POISSON_PARAMS` , `&REFINE_PARAMS` , `&CLUMPFIND_PARAMS` , `&SINK_PARAMS` and `&MOVIE_PARAMS` .

The variables you can set or adjust in each namelist block are described in more details in the following sections.

Global parameters

This block, called `&RUN_PARAMS` , contains the run global control parameters. These parameters are now briefly described. More thorough explanations will be given in dedicated sections of the wiki.

Variable name, syntax, default value	Fortran type	Description
`cosmo=.false.`	`logical`	Activate cosmological supercomoving cooordinates and computes the expansion factor
`pic=.false.`	`logical`	Activate Particle-In-Cell solver
`sink=.false.`	`logical`	Activate sink particles
`clumpfind=.false.`	`logical`	Activate the clump finder

Variable name, syntax, default value	Fortran type	Description
`tracer=.false.`	`logical`	Use Monte Carlo tracer particles
`unbind=.false.`	`logical`	Activate the particle unbinding for clumps
`make_mergertree=.false.`	`logical`	Make merger trees
`poisson=.false.`	`logical`	Activate Poisson solver for self-gravity
`hydro=.false.`	`logical`	Activate hydro or MHD solver.
`rt=.false.`	`logical`	Activate radiative transfer using CPU-based M1 solver. This solver works on the AMR grid.
`aton=.false.`	`logical`	Activate radiative transfer using GPU-based M1 solver. This solver works only on unigrid at `levelmin` .
`verbose=.false.`	`logical`	Activate verbose mode.
`cost_weighting=.true.`	`logical`	Load balancing based on computational cost, not memory. This is rather expensive in term of memory usage. For memory limited runs, using

Variable name, syntax, default value	Fortran type	Description
		`cost_weighting=.false.` is better.
`nrestart=0`	`integer`	Output file number from which the code loads backup data and resumes the simulation, The default value, zero, is for a fresh start from the beginning (time=0).
`nrestart_quad=0`	`integer`	Restart with double precision Hilbert keys. Must be equal to `nrestart` . Default value is 0.
`nstepmax=1000000`	`integer`	Maximum number of coarse time step. This can be used to terminate the simulation after a fixed amount of main steps.
`ncontrol=1`	`integer`	Frequency of screen output for control lines (to stdout), usually redirected into a log file.
`nremap=0`	`integer`	Frequency of call, in units of coarse time steps, for the load balancing routine, for MPI runs only. The default value, zero, means never. Load balancing is a slow operation, so use as high a value as possible.

Variable name, syntax, default value	Fortran type	Description
`ordering="hilbert"`	`character(len=128)`	Cell ordering method used in the domain decomposition of the grid, for MPI runs only. Possible values are `hilbert`, `planar` and `angular`.
`nsubcycle=2,2,2,2,2,2,`	`integer array`	Number of fine level sub-cycling steps within one coarser level time step. Each value in the array corresponds to a given level of refinement, starting from the coarse grid at `levelmin` up to the finest level at `levelmax`. `nsubcycle(1)=1` means that `levelmin` and `levelmin+1` are synchronized. To enforce single time stepping across the whole AMR hierarchy, you need to set `nsubcycle=1,1,1,1,1,1,1,`
`static=.false.`	`logical`	Activate full static mode (RT post processing)
`static_dm=.false.`	`logical`	Activate static mode for DM particles only (isolated initial conditions relaxation)

Variable name, syntax, default value	Fortran type	Description
`static_stars=.false.`	`logical`	Activate static mode for star particles only (isolated initial conditions relaxation)
`remap_pscalar=ndim+3,ndim+4,...,nvar`	`integer array`	Mapping for the passive scalars and non-thermal pressures. Value indicates in which variable the scalar from the restart should be loaded. [0 = ignore this scalar in the restart output, -N = do not read but initialise ivar=N to 0, N = read and initialise ivar=N from the restart output]. For example `remap_pscalar=-6,0,7,8,9` translates to: do not read first restart var but initialise ivar=6 to 0, skip second restart var, read ivar=[8,9,10] from the restart snapshot and store it in the current ivar=[7,8,9].

AMR Parameters

This sets of parameters, contained in the namelist block `&AMR_PARAMS` , controls the AMR grid global properties. Parameters specifying the refinement strategy are described elsewhere , and the corresponding namelist block `&REFINE_PARAMS` is used only if `levelmax>levelmin` .

Variable name, syntax, default value	Fortran type	Description
`levelmin=1`	`integer`	Minimum level of refinement. This parameter sets the size of the coarse (or base) grid to `nx=2**levelmin` .
`levelmax=1`	`integer`	Maximum level of refinement. If `levelmax=levelmin` , the simulation will be executed on a standad Cartesian grid of linear size `nx=2**levelmin`
`ngridmax=1`	`integer`	Maximum number of grids (or octs) that can be allocated during the run within each MPI process.
`ngridtot=1`	`integer`	Maximum number of grids (or octs) that can be allocated during the run for the entire set of MPI processes. One has `ngridmax=ngridtot/ncpu` .
`npartmax=1`	`integer`	Maximum number of particles of all types that can be allocated during the run within each MPI process.
`nparttot=1`	`integer`	Maximum number of particles of all types that can be allocated during the run for the entire set of MPI processes. One has `npartmax=nparttot/ncpu` .
`nsinkmax=1`	`integer`	Maximum number of sink particles during the run. Sink particles are duplicated over all MPI processes.
`nexpand=1`	`integer`	Number of times the mesh expansion is applied to the refinement map (see mesh smoothing).
`boxlen=1.0`	`real`	Box size in user's units

Variable name, syntax, default value	Fortran type	Description
`nlevel_collapse=3`	`integer`	Number of levels above `levelmin` to follow initial halos collapse with a constant comoving resolution (`cosmo=.true.` only)

Initial Conditions Parameters

This sets of parameters, contained in the namelist block `&INIT_PARAMS` . This is used to set up the initial conditions.

Variable name, syntax, default value	Fortran type	Description
`nregion=1`	`integer`	Number of independent regions in the computational box used to set up initial flow variables.
`region_type=square`	`10*char`	Geometry defining each region. `square` defines a generalized ellipsoidal shape, while `point` defines a delta function in the flow.
`x_center=0.0`	`real arrays`	X coordinate of the center of each region.
`y_center=0.0`	`real arrays`	Y coordinate of the center of each region.
`z_center=0.0`	`real arrays`	Z coordinate of the center of each region.
`length_x=0.0`	`real arrays`	Size in X direction of each region.
`length_y=0.0`	`real arrays`	Size in Y direction of each region.

Variable name, syntax, default value	Fortran type	Description
`length_z=0.0`	`real arrays`	Size in Z direction of each region.
`exp_region=2.0`	`real arrays`	Exponent defining the norm used to compute distances for the generalized ellipsoid. `exp_region=2` corresponds to a spheroid, `exp_region=1` to a diamond shape, `exp_region>=10` to a perfect square.
`d_region=0.0`	`real arrays`	Density. For `point` regions this is used to define mass.
`u_region=0.0`	`real arrays`	X velocity. For `point` regions this is used to define velocity.
`v_region=0.0`	`real arrays`	Y velocity. For `point` regions this is used to define velocity.
`w_region=0.0`	`real arrays`	Z velocity. For `point` regions this is used to define velocity.
`p_region=0.0`	`real arrays`	Pressure. For `point` regions this is used to define specific pressure.
`filetype=ascii`	`20*char`	Type of initial conditions file for particles. Possible choices are `ascii` or `grafic` .
`aexp_ini=10.0`	`real`	This parameter sets the starting expansion factor for cosmology runs only. Default value is read in the IC file.

Variable name, syntax, default value	Fortran type	Description
`multiple=.false.`	`logical`	If `.true.` , each processors reads its own IC file. For parallel runs only.
`initfile=`	`80*char`	Directory where IC files are stored.

WORK IN PROGRESS

Output Parameters

This namelist block, called `&OUTPUT_PARAMS` , is used to set up the frequency and properties of data output to disk.

Variable name, syntax, default value	Fortran type	Description
`noutput=1`	`integer`	Number of specified output times. At least one output time should be given, corresponding to the end of the simulation.
`foutput=1000000`	`integer`	Frequency of additional outputs in units of coarse time steps. `foutput=1` means one output at each time step. Specified outputs (see above) will not be superseded by this parameter.
`tout=0.0,0.0,0.0,`	`real array`	Value of specified output times.
`aout=1.1,1.1,1.1,`	`real array`	Value of specified output expansion factor (for cosmology runs only). `aout=1.0` means “present epoch” or “zero redshift”.
`delta_tout=0`	`real`	Frequency of outputs in user time units.

Variable name, syntax, default value	Fortran type	Description
`delta_aout=0`	`real`	Frequency of outputs in expansion factor (for cosmology runs only).
`tend=10`	`real`	End time of the simulation, in user time units.
`aend=0`	`real`	End time of the simulation, in expansion factor (for cosmology runs only).
`walltime_hrs=-1`	`real`	Wallclock time given in ramses job submission, used for dumping an output at the end. Default value of -1 means this is not used.
`minutes_dump=1`	`real`	Dump an output this many minutes before walltime_hrs

Boundary Parameters

This set of parameters, contained in the namelist block `&BOUNDARY_PARAMS` , is used to set up boundary conditions on the current simulation. If this namelist block is absent, periodic boundary conditions are assumed. Setting up other types of boundary conditions in RAMSES is quite complex. The default setting, corresponding to a periodic box, should be sufficient in most cases. The strategy to set up boundary conditions is based on using “ghost regions” outside the computational domain, where flow variables are carefully specified in order to mimic the effect of the chosen type of boundary. Note that the order in which boundary regions are specified in the namelist is very important, especially for reflexive or zero gradient boundaries. Specific examples can be found in the namelist/ directory of the package.

Variable name, syntax, default value	Fortran type	Description
`nboundary=1`	`integer`	Number of ghost regions used to specify the boundary conditions.
`bound_type=0,0,0,`	`integer array`	Type of boundary conditions to apply in the corresponding ghost region. Possible values are: `bound_type=0` (periodic), `bound_type=1` (reflexive), `bound_type=2` (outflow, zero gradients), `bound_type=3` (inflow, user specified).

`d_bound=0.0`
`u_bound=0.0`
`v_bound=0.0`
`w_bound=0.0`
`p_bound=0.0`
	`real arrays`	Flow variables in each ghost region (density, velocities and pressure). They are used only for inflow boundary conditions.

`ibound_min=0`
`jbound_min=0`
`kbound_min=0`
	`integer arrays`	Coordinates of the lower, left, bottom corner of each boundary region. Each coordinate lies between -1 and +1 in each direction.

`ibound_max=0`
`jbound_max=0`
`kbound_max=0`
	`integer arrays`	Likewise, for the upper, right and upper corner of each boundary region.

Hydro parameters

This namelist is called &HYDRO_PARAMS, and is used to specify runtime parameters for the Godunov solver. These parameters are quite standard in computational fluid dynamics. We briefly describe them now

Variable name, syntax, default value	Fortran type	Description
`gamma=1.4`	`Real`	Adiabatic exponent for the perfect gas EOS
`gamma_rad=1.333`	`Real`	Adiabatic exponent for the non-thermal pressure EOS (if NENER>1)
`courant_factor=0.5`	`Real`	CFL number for time step control (less than 1)
`smallr=1d-10`	`Real`	Minimum density to prevent floating exceptions
`smallc=1d-10`	`Real`	Minimum sound speed to prevent floating exceptions
`riemann='llf'`	`Character LEN=20`	Name of the desired Riemann solver. Possible choices are 'exact', 'acoustic', 'llf', 'hll' or 'hllc' for the hydro solver and 'llf', 'hll', 'roe', 'hlld', 'upwind' and 'hydro' for the MHD solver.
`riemann2d='llf'`	`Characher LEN=20`	Name of the desired 2D Riemann solver for the induction equation (MHD only). Possible choices are 'upwind', 'llf', 'roe', 'hll', and 'hlld'.
`scheme='muscl'`	`Character LEN=20`	Name of the desired Godunov integrator. The hydro solver accepts 'muscl' (MUSCL-HANCOCK scheme) or 'pldme' (Collela's PLMDE scheme). The MHD solver accepts 'muscl' or 'induction'.
`niter_riemann=10`	`Integer`	Maximum number of iterations used in the exact Riemann solver

Variable name, syntax, default value	Fortran type	Description
`slope_type=1`	`Integer`	Type of slope limiter used in the Godunov scheme for the piecewise linear reconstruction: slope_type=0: First order scheme, slope_type=1: MinMod limiter, slope_type=2: MonCen limiter, slope_type=3: Multi-dimensional MonCen limiter. In 1D runs only, it is also possible to choose:slope_type=4: Superbee limiter, slope_type=5: Ultrabee limiter
`slope_mag_type=1`	`Integer`	Type of slope limiter used in the Godunov scheme in the MHD solver
`pressure_fix=.false.`	`logical`	Activate hybrid scheme (conservative or primitive) for high-Mach flows. Useful to prevent negative temperatures.
`beta_fix=0d0`	`Real`	With pressure_fix=.true., changes the threshold at which the energy is truncated
`difmag=0d0`	`Real`	Modifies diffusive flux in the PLDME integrator (hydro only).
`eta_mag=0d0`	`Real`	Modifies dtdiff in PLDME integrator (MHD only, warning, divide by zero error if eta_mag=0d0)

In the version of RAMSES after RUM 2017 (PR #284 and after), new blocks were introduced instead of one large `&PHYSICS_PARAMS` :

Parameters

Cooling parameters

The block named `&COOLING_PARAMS` contains the parameters related to cooling / basic chemistry

Variable name	Fortran type	Default value	Description
`cooling`	`boolean`	`.false.`	Enable gas atomic & metal cooling
`metal`	`logical`	`.false.`	Enable metals advection (requires 1 hydro variable)
`isothermal`	`logical`	`.false.`	(deprecated) Enable equation of state for gas (heating and cooling disabled if `.true.`)
`barotropic_eos`	`logical`	`.false.`	Enable barotropic equation of state for gas (heating and cooling disabled if `.true.`). Replaced 'isothermal'
`barotropic_eos_form`	`string`	`legacy`	Type of barotropic EOS. Options: isothermal, polytrope, double_polytrope, custom, legacy
`polytrope_rho`	`real`	1.0d50	sets rho0 in EOS (density normalisation or knee-density), in g/cm3
`polytrope_index`	`real`	1.0d0	sets gamma in EOS (polytropic index)
`T_eos`	`real`	10	sets T0 in EOS (isothermal temperature or temperature normalisation), in K
`mu_gas`	`real`	1d0	molecular weight

Variable name	Fortran type	Default value	Description
`haardt_madau`	`logical`	`.false.`	Enable UV background
`J21`	`real`	0.0	UV flux at threshold in 10^21 units
`a_spec`	`real`	1.0	Slope of the UV spectrum
`self_shielding`	`logical`	`.false.`	Enable self-shielding for densities above 0.01 g.cm^-3
`z_ave`	`real`	0.0	Initial average metal abundance
`z_reion`	`real`	8.5	Reionization redshift (UV background disabled for higher redshifts)
`ind_rsink`	`real`	4.0	Number of cells defining the radius of the sphere where AGN feedback is active
`T2max`	`real`	HUGE	Temperature ceiling for gas heating (heating ceiling if `isothermal=.false.`)
`neq_chem`	`logical`	`.false.`	Enable non-equilibrium chemistry

Star formation parameters

The block named `&SF_PARAMS` contains the parameters related to star formation

Variable name	Fortran type	Default value	Description
`m_star`	`real`	-1.0	Star particle mass in units of mass_sph

Variable name	Fortran type	Default value	Description
`n_star`	`real`	0.1	Star formation density threshold in H/cc
`T2_star`	`real`	0.0	Typical ISM polytropic temperature (cooling floor if `isothermal=.false.`)
`g_star`	`real`	1.6	Typical ISM polytropic index (cooling floor if `isothermal=.false.`)
`del_star`	`real`	2D2	Star formation density threshold in critical density (`cosmo=.true.` only)
`eps_star`	`real`	0.0	Star formation efficiency
`jeans_ncells`	`real`	-1.0	Jeans polytropic equation of state (cooling floor if `isothermal=.false.`)
`sf_virial`	`logical`	`.false.`	Enable turbulent star formation prescriptions (requires 1 hydro variable)
`sf_trelax`	`real`	0.0	Relaxation time for star formation (`cosmo=.false.` only)
`sf_tdiss`	`real`	0.0	Dissipation timescale for subgrid turbulence in units of turbulent crossing time (`sf_virial=.true.` only)
`sf_model`	`integer`	3	Turbulent star formation model (`sf_virial=.true.` only)
`sf_log_properties`	`logical`	`.false.`	Output gas properties of cells undergoing a star particle birth or supernova event

Variable name	Fortran type	Default value	Description
`sf_compressive`	`logical`	`.false.`	Store and advect the compressive and solenoidal turbulent field in two different hydro variables (`sf_virial=.true.` only)

Feedback parameters

The block named `&FEEDBACK_PARAMS` contains the parameters related to cooling / basic chemistry

Variable name	Fortran type	Default value	Description
`eta_sn`	`real`	0.0	Supernova mass fraction
`eta_ssn`	`real`	0.95	Single supernova ejected mass fraction (`sf_imf=.true.` only)
`t_sne`	`real`	10.0	Supernova blast time in Myr
`delayed_cooling`	`logical`	`.false.`	Enable delayed cooling through passive energy scalar advection (requires 1 hydro variable)
`t_diss`	`real`	20.0	Dissipation timescale for supernova feedback in Myr (`delayed_cooling=.true.` only)
`yield`	`real`	0.0	Supernova metal yield
`mass_gmc`	`real`	0.0	Stochastic exploding GMC mass in solar mass
`kappa_IR`	`real`	0.0	IR dust opacity for supernova feedback

Variable name	Fortran type	Default value	Description
`f_ek`	`real`	0.0	Supernova kinetic energy fraction (only between 0 and 1)
`f_w`	`real`	0.0	Supernova mass loading factor (`f_ek>0` only)
`rbubble`	`real`	0.0	Supernova superbubble radius in pc (`f_ek>0` only)
`ndebris`	`integer`	1	Supernova debris particle number (`f_ek>0` only)
`mass_star_max`	`real`	120.0	Maximum mass of a star to undergo a supernova with `eta_ssn` efficiency in solar mass (`sf_imf=.true.` only)
`mass_sne_min`	`real`	10.0	Minimum mass of a star to undergo a supernova blast in solar mass (`sf_imf=.true.` only)
`ir_feedback`	`logical`	`.false.`	Enable IR feedback from accreting sink particles
`ir_eff`	`real`	0.75	Efficiency of the IR feedback on sink particles (`ir_feedback=.true.` only)

Units parameters

The block named `&UNITS_PARAMS` contains the parameters related to units that are set “by hand” (`cosmo=.false.` only!).

Variable name	Fortran type	Default value	Description
`units_density`	`real`	1.0	Density unit in cgs (only for `cosmo=.false.` , requires G=1)
`units_time`	`real`	1.0	Time unit in cgs (only for `cosmo=.false.` , requires G=1)
`units_length`	`real`	1.0	Length unit in cgs (only for `cosmo=.false.` , requires G=1)

GRACKLE parameters

Additionally, if the code has been compiled with GRACKLE=1 in the Makefile and properly linked against the hdf5 and grackle libraries, it is possible to define a `&GRACKLE_PARAMS` block to control the grackle parameters. Please visit https://grackle.readthedocs.io/en/grackle-3.0/Parameters.html for a more extensive description of the grackle parameters.

Variable name	Fortran type	Default value	Description
`use_grackle`	`integer`	1	Activate Grackle
`grackle_with_radiative_cooling`	`integer`	1	Include radiative cooling
`grackle_primordial_chemistry`	`integer`	0	Primordial chemistry flag
`grackle_metal_cooling`	`integer`	0	Enable metal cooling using Cloudy tables

Variable name	Fortran type	Default value	Description
`grackle_UVbackground`	`integer`	0	Enable UV background
`grackle_cmb_temperature_floor`	`integer`	1	Enable effective CMB temperature floor
`grackle_h2_on_dust`	`integer`	0	Enable H2 formation on dust grains
`grackle_photoelectric_heating`	`integer`	0	Enable a spatially uniform heating term approximating photo-electric heating
`grackle_use_volumetric_heating_rate`	`integer`	0	Signal that an array of volumetric heating rates is being provided
`grackle_use_specific_heating_rate`	`integer`	0	Signal that an array of specific heating rates is being provided

Variable name	Fortran type	Default value	Description
grackle_three_body_rate	integer	0	Flag to control which three-body H2 formation rate is used
grackle_cie_cooling	integer	0	Enable H2 collision-induced emission cooling
grackle_h2_optical_depth_approximation	integer	0	Enable H2 cooling attenuation
grackle_ih2co	integer	1	
grackle_ipiht	integer	1	
grackle_NumberOfTemperatureBins	integer	600	
grackle_CaseBRecombination	integer	0	
grackle_Compton_xray_heating	integer	0	Flag to enable Compton heating from an X-ray background
grackle_LWbackground_sawtooth_suppression	integer	0	Flag to enable suppression of Lyman-

Variable name	Fortran type	Default value	Description
			Werner flux due to Lyman-series absorption
`grackle_NumberOfDustTemperatureBins`	`integer`	250	
`grackle_use_radiative_transfer`	`integer`	0	Signal that arrays of ionization and heating rates from radiative transfer solutions are being provided
`grackle_radiative_transfer_coupled_rate_solver`	`integer`	0	Flag that must be enabled to couple the passed radiative transfer fields to the chemistry solver
`grackle_radiative_transfer_intermediate_step`	`integer`	0	Flag to enable intermediate stepping in applying radiative transfer fields to chemistry solver

Variable name	Fortran type	Default value	Description
grackle_radiative_transfer_hydrogen_only	integer	0	Flag to only use hydrogen ionization and heating rates from the radiative transfer solutions
grackle_self_shielding_method	integer	0	Switch to enable approximate self-shielding from the UV background
grackle_Gamma	real	5./3.	The ratio of specific heats for an ideal gas
grackle_photoelectric_heating_rate	real	8.5D-26	The heating rate in units of erg cm-3 s-1
grackle_HydrogenFractionByMass	real	0.76	
grackle_DeuteriumToHydrogenRatio	real	2.0*3.4e-5	
grackle_SolarMetalFractionByMass	real	0.01295	
grackle_TemperatureStart	real	1.0	

Variable name	Fortran type	Default value	Description
grackle_TemperatureEnd	real	1.0D9	
grackle_DustTemperatureStart	real	1.0	
grackle_DustTemperatureEnd	real	1500.	
grackle_LWbackground_intensity	real	0.0	Intensity of a constant Lyman-Werner H2 photo-dissociating radiation field in units of 10-21 erg s-1 cm-2 Hz-1 sr-1
grackle_UVbackground_redshift_on	real	7.0	
grackle_UVbackground_redshift_off	real	0.0	
grackle_UVbackground_redshift_fullon	real	6.0	
grackle_UVbackground_redshift_drop	real	0.0	
grackle_cloudy_electron_fraction_factor	real	9.153959D-3	
grackle_data_file	character		Path to the data file containing the metal cooling and UV

Variable name	Fortran type	Default value	Description
			background HDF5 tables

Physics parameters (LEGACY ONLY)

The block named `&PHYSICS_PARAMS` contains the parameters related to physical models.

Variable name	Fortran type	Default value	Description
`units_density`	`real`	1.0	Density unit in cgs (only for `cosmo=.false.` , requires G=1)
`units_time`	`real`	1.0	Time unit in cgs (only for `cosmo=.false.` , requires G=1)
`units_length`	`real`	1.0	Length unit in cgs (only for `cosmo=.false.` , requires G=1)
`cooling`	`boolean`	`.false.`	Enable gas atomic & metal cooling
`isothermal`	`logical`	`.false.`	Enable equation of state for gas (heating and cooling disabled if `.true.`)
`metal`	`logical`	`.false.`	Enable metals advection (requires 1 hydro variable)
`haardt_madau`	`logical`	`.false.`	Enable UV background

Variable name	Fortran type	Default value	Description
`self_shielding`	`logical`	`.false.`	Enable self-shielding for densities above 0.01 g.cm^-3
`smbh`	`logical`	`.false.`	Enable super massive black holes for sink particles
`agn`	`logical`	`.false.`	Enable AGN feedback from super massive black holes
`neq_chem`	`logical`	`.false.`	Enable non-equilibrium chemistry
`ir_feedback`	`logical`	`.false.`	Enable IR feedback from accreting sink particles
`sf_virial`	`logical`	`.false.`	Enable turbulent star formation prescriptions (requires 1 hydro variable)
`sf_compressive`	`logical`	`.false.`	Store and advect the compressive and solenoidal turbulent field in two different hydro variables (`sf_virial=.true.` only)
`sf_log_properties`	`logical`	`.false.`	Output gas properties of cells undergoing a star particle birth or supernova event
`delayed_cooling`	`logical`	`.false.`	Enable delayed cooling through passive energy scalar advection (requires 1 hydro variable)
`sf_imf`	`logical`	`.false.`	Model Initial Mass Function during thermal feedback events

Variable name	Fortran type	Default value	Description
`T2_star`	`real`	0.0	Typical ISM polytropic temperature (cooling floor if `isothermal=.false.`)
`g_star`	`real`	1.6	Typical ISM polytropic index (cooling floor if `isothermal=.false.`)
`jeans_ncells`	`real`	-1.0	Jeans polytropic equation of state (cooling floor if `isothermal=.false.`)
`T2max`	`real`	1D50	Temperature ceiling for gas heating (heating ceiling if `isothermal=.false.`)
`n_star`	`real`	0.1	Star formation density threshold in H/cc
`m_star`	`real`	-1.0	Star particle mass in units of mass_sph
`del_star`	`real`	2D2	Star formation density threshold in critical density (`cosmo=.true.` only)
`eps_star`	`real`	0.0	Star formation efficiency
`t_star`	`real`	0.0	Star formation time scale in Gyr (only used if `eps_star>0`)
`sf_trelax`	`real`	0.0	Relaxation time for star formation (`cosmo=.false.` only)
`sf_tdiss`	`real`	0.0	Dissipation timescale for subgrid turbulence in units of turbulent crossing time (`sf_virial=.true.` only)

Variable name	Fortran type	Default value	Description
`sf_model`	`integer`	3	Turbulent star formation model (`sf_virial=.true.` only)
`eta_sn`	`real`	0.0	Supernova mass fraction
`eta_ssn`	`real`	0.95	Single supernova ejected mass fraction (`sf_imf=.true.` only)
`t_sne`	`real`	10.0	Supernova blast time in Myr
`t_diss`	`real`	20.0	Dissipation timescale for supernova feedback in Myr (`delayed_cooling=.true.` only)
`yield`	`real`	0.0	Supernova metal yield
`mass_gmc`	`real`	0.0	Stochastic exploding GMC mass in solar mass
`kappa_IR`	`real`	0.0	IR dust opacity for supernova feedback
`f_ek`	`real`	0.0	Supernova kinetic energy fraction (only between 0 and 1)
`f_w`	`real`	0.0	Supernova mass loading factor (`f_ek>0` only)
`rbubble`	`real`	0.0	Supernova superbubble radius in pc (`f_ek>0` only)

Variable name	Fortran type	Default value	Description
`ndebris`	`integer`	1	Supernova debris particle number (`f_ek>0` only)
`mass_star_max`	`real`	120.0	Maximum mass of a star to undergo a supernova with `eta_ssn` efficiency in solar mass (`sf_imf=.true.` only)
`mass_sne_min`	`real`	10.0	Minimum mass of a star to undergo a supernova blast in solar mass (`sf_imf=.true.` only)
`J21`	`real`	0.0	UV flux at threshold in 10^21 units
`a_spec`	`real`	1.0	Slope of the UV spectrum
`z_ave`	`real`	0.0	Initial average metal abundance
`z_reion`	`real`	8.5	Reionization redshift (UV background disabled for higher redshifts)
`ind_rsink`	`real`	4.0	Number of cells defining the radius of the sphere where AGN feedback is active
`ir_eff`	`real`	0.75	Efficiency of the IR feedback on sink particles (`ir_feedback=.true.` only)

Poisson Parameters

This namelist, `&POISSON_PARAMS` , is used to specify runtime parameters for the Poisson solver. It is used only if `poisson=.true.` or `pic=.true.`

Two different Poisson solvers are available in RAMSES: conjugate gradient (CG) and multigrid (MG). Unlike the CG solver, MG has an initialization overhead cost (at every call of the solver), but is much more efficient on very big levels with few “holes”. The multigrid solver is therefore used for all coarse levels.

In addition, MG can be used on refined levels in conjuction with CG. The parameter `cg_levelmin` selects the Poisson solver as follows:

- Coarse levels are solved with MG
- Refined levels with *l* < `cg_levelmin` are solved with MG
- Refined levels with *l* >= `cg_levelmin` are solved with CG

Variable name	Fortran type	Default value	Description
`gravity_type`	`int`	0	Type of gravity force. Possible choices are: `gravity_type=0` self-gravity (Poisson solver); `gravity_type>0` analytical gravity vector; `gravity_type<0` self-gravity plus additional analytical density profile
`epsilon`	`real`	1e-4	Stopping criterion for the iterative Poisson solver: residual 2-norm should be lower than `epsilon` times the right hand side 2-norm.
`gravity_params`	`real array`	0.0,	Parameters used to define the analytical gravity field (routine `gravana.f90`) or the analytical mass density field (routine `rho_ana.f90`).
`cg_levelmin`	`integer`	999	Minimum level from which the Conjugate Gradient solver is used in place of the Multigrid solver.
`cic_levelmax`	`integer`	0	Maximum level for CIC dark matter interpolation (default `cic_levelmax=nlevelmax`)

Refinement parameters

The block named `&REFINE_PARAMS` contains the parameters related to grid refinement.

Variable name	Fortran type	Default value	Description
`x_refine`	`real array`	0.0	Geometry-based strategy: center of the refined region at each level of the AMR grid.
`y_refine`	`real array`	0.0	Geometry-based strategy: center of the refined region at each level of the AMR grid.
`z_refine`	`real array`	0.0	Geometry-based strategy: center of the refined region at each level of the AMR grid.
`r_refine`	`real array`	1e10	Geometry-based strategy: radius of the refined region at each level.
`a_refine`	`real array`	1.0	Geometry-based strategy: ratio Y/X of the refined region at each level.
`b_refine`	`real array`	1.0	Geometry-based strategy: ratio Z/X of the refined region at each level.
`exp_refine`	`real array`	2.0	Geometry-based strategy: exponent of the norm.
`jeans_refine`	`real array`	-1.0	Jeans refinement strategy: each level is refined if the cell size exceeds the local Jeans length divided by jeans_refine(ilevel).
`mass_cut_refine`	`real`	-1.0	Mass threshold for particle-based refinement

Variable name	Fortran type	Default value	Description
`m_refine`	`real array`	-1.0	Quasi-Lagrangian strategy: each level is refined if the baryons mass in a cell exceeds `m_refine(ilevel)*mass_sph` , or if the number of dark matter particles exceeds `m_refine(ilevel)` , whatever the mass is.
`mass_sph`	`real`	0.0	Quasi-Lagrangian strategy: `mass_sph` is used to set a typical mass scale. For cosmo runs, its value is set automatically.
`err_grad_d`	`real`	-1.0	Discontinuity-based strategy: density gradient relative variations above which a cell is refined
`err_grad_u`	`real`	-1.0	Discontinuity-based strategy: velocity gradient relative variations above which a cell is refined
`err_grad_p`	`real`	-1.0	Discontinuity-based strategy: pressure gradient relative variations above which a cell is refined
`floor_d`	`real`	1e-10	Density floor below which gradients are ignored
`floor_u`	`real`	1e-10	Velocity floor below which gradients are ignored
`floor_p`	`real`	1e-10	Pressure floor below which gradients are ignored
`ivar_refine`	`int`	-1	Variable index for refinement

Variable name	Fortran type	Default value	Description
`var_cut_refine`	`real`	-1.0	Threshold for variable-based refinement
`interpol_var`	`int`	0	Variables used to perform interpolation (prolongation) and averaging (restriction). `interpol_type=0` : conservatives; `interpol_type=1` : primitives
`interpol_type`	`int`	1	Type of slope limiter used in the interpolation scheme for newly refined cells. `interpol_type=0` : Straight injection (1st order), `interpol_type=1` : MinMod limiter, `interpol_type=2` : MonCen limiter, `interpol_type=3` : unlimited central slope, `interpol_type=4` : type 3 for velocity and type 2 for density and internal energy (if `interpol_var=2`)
`sink_refine`	`bool`	`.false.`	Refines grid around sinks to levelmax

Clumpfinder (PHEW)

The block named `&CLUMPFIND_PARAMS` contains the parameters related to the built-in RAMSES clump finder (PHEW). The parameters are described only briefly, for more backround information read the PHEW paper .

Variable name, syntax, default value	Fortran type	Description
`ivar_clump=1`	`integer`	Control which density field is used for clump finding (1: gas density, 0: particle density)

Variable name, syntax, default value	Fortran type	Description
`density_threshold=-1.d0`	`float`	Density threshold for the clump finder (code units)
`rho_clfind=-1.d0`	`float`	Density threshold for the clump finder (g/cc). Not recommended - use `density_threshold` instead.
`n_clfind=-1.d0`	`float`	Density threshold for the clump finder (H/cc). Not recommended -use `density_threshold` instead.
`relevance_threshold=2.d0`	`float`	Relevance (peak-to-saddle ratio) threshold for a clump to be considered real (instead of noise).
`saddle_threshold=-1.d0`	`float`	Saddle density threshold for sub-structure merging (code units). A negative value turns sub-structure merging off (PHEW will only merge noise).
`mass_threshold=0.d0`	`float`	When set to a value > 0, the properties of those clumps/haloes with `mass > mass_threshold * particle_mass` are written to disk. `particle_mass` is the smallest strictly positive particle mass in the simulation. Setting this parameter does NOT affect the merging of noise/clumps.
`age_cut_clfind=0.d0`	`float`	When set to a value > 0, only the stars with an age lower than `age_cut_clfind` are used by the clump finder. Stellar and DM particles are used otherwise.

Particle Unbinding

Quickstart Guide

To activate particle unbinding, you need to set two runtime parameters in the `&RUN_PARAMS` namelist (in addition to whatever you have in there):

```
&RUN_PARAMS
clumpfind=.true.
unbind=.true.
/
```

The code will create `output_XXXXX/unbinding_XXXXX.outYYYYY` files following the same logic as the other particle output files `output_XXXXX/part_XXXXX.outYYYYY` files. They will contain the associated clump ID of each particle. In only works when particles, i.e. dark matter, is present in your simulation.

Important notes

- You can't do particle unbinding without doing clump/halo finding.
- Some parts of the code (e.g. binning particles in mass profiles of halos) rely on consistent floating-point operations. The (intel) fortran compiler however doesn't necessarily use value-safe optimisations , which may lead to **errors resulting in warnings** , but the code doesn't crash. The error should be small (~1e-16), and you may choose to ignore it. Otherwise, you might want to compile the code with the `-fp-model precise` flag for intel, or the appropriate flag for the compiler you'd like to use.
- The previously available `unbinding_formatted_output` namelist parameter has been removed. Make sure you remove it from your namelists if you used it!
- For anything else regarding particle unbinding, feel free to contact Mladen Ivkovic (mladen.ivkovic [at] hotmail DOT com)

Documentation

What it does

The purpose of particle unbinding is to identify unbound particles in clumps as identified by the clumpfinder and pass them on to the parent clumps, until the halo-namegiver clumps are reached (where there are no more parent structures to pass the particles on to.)

It will write unformatted output in `output_XXXXX/unbinding_XXXXX.outYYYYY` files the same way it is done for any other backup files in `ramses` , containing the assigned clump IDs of every particle after unbinding. The clump IDs correspond to the clump IDs as used in the `halo_XXXXX.txtYYYYY` and `clump_XXXXX.txtYYYYY` files. If a particle has clump ID 0, it wasn't found to be in any clump.

How it works

First all particles that are in a clump are gathered and assigned the corresponding clump ID. Simultaneously, linked lists of these particles are created for each clump that is not a halo-namegiver, i.e. that is not a clump whose ID will be the halo ID. Such clumps are never merged into another clump, but may have arbitrarily many clumps merged into them. Then clump properties such as centre of mass, the mass profile and bulk velocity are acquired using the linked lists. Starting with the lowest clump level, particles are checked if they are bound to the clump they are assigned to. By default, this unbinding is done iteratively: The clump properties are recomputed using only the remaining particles, and then all particles checked again. Furthermore, by default particles are not allowed to leave the boundaries of the clump they're assigned to in order to be considered bound; For this, the potential at the closest saddle from the clump's centre of mass to a neighbouring clump is subtracted from the particle's energy. (This default behaviour can however be changed with the namelist parameters `saddle_pot` and `iter_properties`). Also note that the bulk velocity and centre of mass of a clump are recovered using only the bound particles per iteration, the mass profile however always uses all included particles, including the substructure particles. The iteration per clump level stops when the bulk density of each clump of that level has converged, i.e. `v_clump_old/v_clump_new < conv_limit` (or when a maximal number of iterations is reached). Particles that are found to be not bound are passed to the parent structure for examination, provided such a structure exists, and the iterations repeated for the next clump level, provided there are clumps of a higher level. There are two possibilities implemented to define the clump's centre. By default, the centre will be the position of the associated density peak. However using the `-DUNBINDINGCOM` preprocessing flag, you can make the centre be the centre of mass, which will also be determined iteratively like the bulk velocity.

More details can be found here .

Namelist Parameters for unbinding

Can be set in the `UNBINDING_PARAMS` block

Name	default	type	function
`particlebased_clump_output=`	`.false.`	logical	write resulting clump properties based on particles after unbinding, not default cell-based properties

Name	default	type	function
`nmassbins=`	`50`	integer	Number of bins for the mass binning of the cumulative mass profile. Any integer > 1.
`logbins=`	`.true.`	logical	use logarithmic binning distances for cumulative mass profiles (and gravitational potential of clumps). If false, the code will use linear binning distances.
`saddle_pot=`	`.true.`	logical	Take neighbouring structures into account; Cut potential off at closest saddle.
`iter_properties=`	`.true.`	logical	whether to unbind multiple times with updated clump properties determined by earlier unbindings
`conv_limit =`	`0.01`	real	convergence limit. If `v_clump_old/v_clump_new < conv_limit`, stop iterating for this clump. (only used when `iter_properties=.true.`)
`repeat_max =`	`100`	integer	maximal number of loops per level for iterative unbinding (in case a clump doesn't converge) (shouldn't happen) (only used when `iter_properties=.true.`)

Making Mergertrees (And Mock Galaxies)

Quickstart Guide

To get your merger trees, you need to set three runtime parameters in the `&RUN_PARAMS` namelist (in addition to whatever you have in there):

```
&RUN_PARAMS
clumpfind=.true.
unbind=.true.
make_mergertree=.true.
/
```

For dark matter only (DMO) simulations, the clump finder recovers good halos and subhalos with these `CLUMPFIND_PARAMS` :

```
&CLUMPFIND_PARAMS
relevance_threshold=3
density_threshold=80
saddle_threshold=200
/
```

By default, mock galaxy catalogues will be created. You can turn this behaviour off by setting

```
&MERGERTREE_PARAMS
make_mock_galaxies=.false.
/
```

in the `&MERGERTREE_PARAMS` block in your namelist.

What output files are created and how to read them is described further below.

Important notes

- To make merger trees, you need to use the clump finder and the particle unbinding routines first. More on clump finding parameters can be found [here][4]. More on particle unbinding can be found [here][6]
- Some parts of the code (e.g. binning particles in mass profiles of halos) rely on consistent floating-point operations. The (intel) fortran compiler however doesn't necessarily use value-safe optimisations , which may lead to **errors resulting in warnings** , but the code doesn't crash. The error should be small (~1e-16), and you may choose to ignore it. Otherwise, you might want

to compile the code with the `-fp-model precise` flag for intel, or the appropriate flag for the compiler you'd like to use.

- For accurate merger trees, consider running your simulation for a handful (I used 3) snapshots more than you actually need to make sure past merging events are actually mergers, not just two clumps too close to each other to be recognized as distinct clumps. You'll also need to check in these "extra snapshots" later whether any clump re-emerged later.
- I'm not really able to predict how much memory the patch will need, because it will accumulate orphan galaxies over the simulation. It shouldn't be much, but obviously will depend on how big of a simulation you are trying to run. It never was a significant amount of memory (`< 10 Mb`) when I tried it with `512^3` particles, but I'd keep that in mind if you have memory issues.
- For anything else regarding the merger trees, feel free to contact Mladen Ivkovic (mladen.ivkovic [at] hotmail DOT com)

Documentation

What it does

This functionality creates dark matter halo merger trees. Essentially, clumps as identified by the clumpfinder PHEW between two snapshots are linked as progenitors and descendants, if possible. Preferrably clumps between two adjacent snapshots are linked, but if a descendant has no direct progenitor in the adjacent snapshot, the program will try to find progenitors in older snapshots.

Optionally, it can create mock galaxy catalogues. Using a parametrised SHAM stellar-mass-halo-mass relation, the most bound particle of each clump is assigned a stellar mass. Once a subhalo is merged, the galaxy, now an orphan, is still being tracked until the end of the simulation.

Mergertree Output

The merger trees are stored in `output_XXXXX/mergertree_XXXXX.txtYYYYY` files. Each file contains 11 columns:

- `clump` : clump ID of a clump at this output number
- `progenitor` : the progenitor clump ID in output number "prog_outputnr"
- `prog_outputnr` : the output number of when the progenitor was an alive clump
- `desc_mass` : mass of the current clump.
- `desc_npart` : number of particles of the current clump.

- `desc_x,_y,_z` : x, y, z position of current clump.
- `desc_vx,_vy,_vz` : x, y, z velocities of current clump.

desc_mass and desc_npart will be either inclusive or exclusive, depending on how you set the `use_exclusive_mass` parameter. (See below for details)

How to read the output:

- A clump > 0 has progenitor > 0: Standard case. A direct progenitor from the adjacent previous snapshot was identified for this clump.
- A clump > 0 has progenitor = 0: no progenitor could be established and the clump is treated as newly formed.
- A clump > 0 has progenitor < 0: it means that no direct progenitor could be found in the adjacent previous snapshot, but a progenitor was identified from an earlier, non-adjacent snapshot.
- A clump < 0 has progenitor > 0: this progenitor merged into this clump, but is not this clump's main progenitor.
- A clump < 0 has progenitor < 0: this shouldn't happen.

Mock Galaxy Output

The mock galaxy output is stored in `output_XXXXX/galaxies_XXXXX.txtYYYYY` files. Every file contains 5 columns:

- `Associated_clump` : Provided this is not an orphan galaxy, the clump ID in which this galaxy is. Orphan galaxies have associated clump = 0
- `Stellar_Mass` : The galaxy's stellar mass in units of solar mass.
- `x, y, z` : position of the galaxy.
- `Galaxy_Particle_ID` : The ID of the particle this galaxy is attributed to.

How it works

After every clumpfinding and unbinding step in the simulation, the merger tree code is called. For every clump, the `nmost_bound` number of most bound (= with lowest total energy) particles are found and written to file, as they will be treated as tracers for this clump. In the next output step, those files will be read in and sorted out: The clumps of the previous output will be progenitors of this output. Based on in which descendant clump each progenitor's particles ended up in, progenitors and descendants are linked, i.e. possible candidates are indentified this way. Next, the main progenitor of each descendant and the main descendant of each progenitor need to be

found. A descendant may have multiple progenitors, but only one main progenitor. Progenitors however are only allowed to have one descendant, their main descendant.

The tree-making is performed iteratively. A main progenitor-descendant pair is established when the main progenitor of a descendant is the main descendant of said progenitor. At every iteration, all descendant candidates of all progenitors that haven't found their match yet are checked; The descendants however only move along one progenitor candidate. The iteration is repeated until every descendant has checked all of its candidates or found its match. Progenitors that haven't found a main descendant that isn't taken yet will be considered to have merged into their best fitting descendant candidate.

After the iteration, any progenitor that is considered as merged into its descendant will be recorded as a "past merged progenitor". Then descendants that haven't got a progenitor will try to find a progenitor in non-adjacent snapshots, which are stored as "past merged progenitors". (Obviously not in one of the newly added past merged progenitors.) As the past merged progenitors are traced via 1 particle ("galaxy particle"), the past merged progenitor of the most bound "galaxy particle" that is also assigned as a particle of a descendant will be considered the main progenitor of the descendant under consideration.

Mock galaxy catalogues are created using a parametrised SHAM relation between (sub)halo mass and stellar mass adapted from Behroozi, Wechsler and Conroy 2013 . Once a subhalo merges into another clump, its (orphan) galaxy is still being tracked for a user-specified number of snapshots (`max_past_snapshots` paarameter below) by tracking what was the last identifiable most bound particle of that subhalo.

For more details on how it works, some tests and results, you can have a look [here][5]

New namelist parameters for this pach

Can be set in the `MERGERTREE_PARAMS` block.

Name	default	type	function
`nmost_bound =`	`200`	integer	Up to how many particles per clump to track between two snapshots.
`max_past_snapshots =`	`0`	integer	maximal number of past snapshots to store. If = 0, all past merged progenitors will be stored. If `make_mock_galaxies=.true.` , it will also limit the number of snapshots for which orphan galaxies are tracked.

Name	default	type	function
`use_exclusive_mass =`	`.true.`	logical	how to define clump mass: If false, all substructure of a clump is considered part of its mass. Otherwise, use only particles that are bound to the clump itself (excluding main haloes: main haloes always consist of all the particles within them). Note that this mass definition is only used for creating the merger trees, not for the clump/halo output!
`make_mock_galaxies =`	`.true.`	logical	whether to also create mock galaxy catalogues on the fly.

Visualisation and Postprocessing

`ramses/utils/py/mergertreeplot.py` is a python 2 script to plot the merger trees as found by this patch. `ramses/utils/py/mergertree-extract.py` is a python3 script to extract the mass evolution of a single clump, a halo with all its subhaloes, or all haloes in the simulation. Details on options and usage are at the start of the scripts as a comment, or can be called using the `--help` flags.

Crashing on MPI writing routines?

Apparently some MPI implementations have issues with collective writing routines, which are used by default in the merger tree patch. To circumvent this problem, the `-DMTREE_INDIVIDUAL_FILES` preprocessing directive can be set in the Makefile. Just add it to the `DEFINES=` line at the top of the file. With this flag in use, instead of collective files every MPI task will write an individual unformatted Fortran file, and then read it back in at the later snapshot and communicate the data appropriately.

However, be advised: Using this flag creates a lot of small files (`4* #MPI tasks` number of extra files in addition to `2-3 * #MPI tasks` to result files for particle unbinding, merger trees, and, if chosen, galaxy files per snapshot). This might become an issue if the machine you're working on applies file number quotas.

[4]: PHEW Section [5]: https://drive.google.com/open?id=1q0RSMeTIF7gQ7s2DXzYZUSG6cQ1LstkF [6]: Unbinding Section

Sink particles

The block named `&SINK_PARAMS` contains the parameters related to the sink particle implementation, which can be used for

- star formation (most recently Bleuler & Teyssier 2015)
- supermassive black hole evolution and AGN feedback (most recently Biernacki, Teyssier and Bleuler 2017)

Overview of parameters

Variable name	Fortran type	Default value	Description
`smbh`	`boolean`	`.false.`	Controls if sink behaves as a star or SMBH
`agn`	`boolean`	`.false.`	Controls if SMBH sink produces feedback
`create_sinks`	`boolean`	`.false.`	Specifies if sinks are formed with clump finder
`nsinkmax`	`integer`	`2000`	Maximum number of sinks allowed to form
`mass_sink_direct_force`	`float`	`-1`	Mass in Msun above which sinks are treated with direct N-body solver
`ir_cloud`	`integer`	`4`	Radius of cloud region in unit of grid spacing
`ir_cloud_massive`	`integer`	`3`	Radius of massive cloud region in unit of grid spacing

Variable name	Fortran type	Default value	Description
`sink_soft`	`integer`	`2`	Gravitational softening length in dx at levelmax for “direct force” sinks
`n_sink`	`float`	`.false.`	Sink (as a star) formation density threshold in H/cc
`rho_sink`	`float`	`.false.`	Sink (as a star) formation density threshold in g/cc
`d_sink`	`float`	`.false.`	Sink (as a star) formation density threshold in code units
`clump_core`	`boolean`	`.false.`	Trims the clump (for sinks as stars only)
`mass_sink_seed`	`float`	`0.0`	Dynamical mass of sink seed in Msun
`mass_smbh_seed`	`float`	`0.0`	Accretion mass of sink seed in Msun, if 0, then dynamical and accretion masses are equivalent
`mass_halo_AGN`	`float`	`1e10`	Mass of a halo in which AGN sinks are seeded
`mass_clump_AGN`	`float`	`1e10`	Mass of a clump in which AGN sinks are seeded
`accretion_scheme`	`string`	`none`	Accretion scheme: none, bondi

Variable name	Fortran type	Default value	Description
`eddington_limit`	`boolean`	`.false.`	Controls if accretion rate should be limited by Eddington rate
`acc_sink_boost`	`float`	`1`	Value of boost factor in Bondi velocity (-1 to depend on density)
`boost_threshold_density`	`float`	`0.1`	Threshold density for boosting, typically the same as n_star; in H/cc
`bondi_use_vrel`	`boolean`	`.false.`	Excludes relative velocity between gas and sink from the accretion calculations
`mass_merger_vel_check_AGN`	`float`	`-1`	Mass above which the check for two sinks' binding energy is applied, in Msun
`merging_timescale`	`float`	`-1`	Time during which sinks are considered for merging (non-SMBH only)
`verbose_AGN`	`boolean`	`.false.`	Controls verbosity of AGN in the log file
`AGN_fbk_mode_switch_threshold`	`float`	`0.1`	Controls the AGN feedback switching

Variable name	Fortran type	Default value	Description
`AGN_fbk_frac_ener`	`float`	`1.0`	Controls what fraction of energy goes into thermal feedback
`T2_min`	`float`	`1e7`	Minimum temperature to which AGN blast should heat the gas in K
`T2_AGN`	`float`	`1e12`	Temperature of AGN blasts in K - feedback efficiency
`AGN_fbk_frac_mom`	`float`	`1.0`	Controls what fraction of energy goes into kinetic feedback
`epsilon_kin`	`float`	`1.0`	Kinetic feedback coupling efficiency
`kin_mass_loading`	`float`	`100.`	Kinetic feedback mass loading
`cone_opening`	`float`	`180.`	Opening angle of the cone through which momentum feedback proceeds

Example set of parameters for cosmological simulations with AGN feedback

The example listed below by no means can fit everyone's needs, but can serve as a minimum starting example.

```
#!fortran
&SINK_PARAMS
! General switches
smbh=.true.                  ! turns sinks into SMBH
```

```
agn=.true.                    ! enables AGN feedback
create_sinks=.true.           ! enables formation of new sink particles
mass_sink_direct_force=1.0    ! minimum mass of sink to treat it with
direct solver, in M_sun

! Seeding masses
mass_sink_seed=1.0d6          ! dynamical mass of sink particle
mass_smbh_seed=1.0d6          ! accretion mass of sink particle
mass_halo_AGN=5.d10           ! minimum mass of PHEW halo in which a
sink is seeded
mass_clump_AGN=1.d9           ! minimum mass of PHEW clump in which a
sink is seeded

! Accretion
accretion_scheme='bondi'      ! selects Bondi accretion as accretion
mode
eddington_limit=.true.        ! enables Eddington limit on accretion
acc_sink_boost=-1             ! boosts accretion according to
Booth&Schaye 2009
boost_threshold_density=0.1 ! threshold density for boosting,
typically the same as n_star; in H/cc
bondi_use_vrel=.false.        ! excludes relative velocity between gas
and sink from the accretion calculations

! Merging
mass_merger_vel_check=1e8     ! sum of sinks' masses for which
velocities are checked upon merging to determine if the system is
bound

! Feedback
T2_min=0                      ! if feedback can heat the gas to this
temperature then deposit it; here deposits at every fine step
T2_AGN=0.15d12                ! thermal feedback efficiency
AGN_fbk_frac_ener=1.0         ! controls what fraction of energy goes
into thermal feedback, here 100%
AGN_fbk_frac_mom=0.0          ! controls what fraction of energy goes
into momentum feedback, here 0%
AGN_fbk_mode_switch_threshold=1d-2 ! switches between thermal (above)
and momentum modes for this ratio of Bondi-to-Eddington
epsilon_kin=1.                ! momentum feedback efficiency
kin_mass_loading=100.         ! mass loading factor of momentum
feedback
cone_opening=90.              ! opening angle of the cone in which
momentum feedback is deposited (180. means full sphere)
/
```

Notes:

- if `agn=.false.` all feedback settings are disregarded

- in order to seed the sink *both* `mass_halo_AGN` and `mass_clump_AGN` have to be satisfied (as well as condition of only one sink per halo and minimum density of the halo - at least star forming)
- currently the only other accretion mode besides `bondi` is `none` ; please feel free to add more
- to not boost accretion, set `acc_sink_boost=0.0`
- it is *highly advised* to use `T2_min=0.0` in order to have all feedback modes on the finest timestep
- if `chi_switch=0.0` , then the initial values of `AGN_fbk_frac_*` will be used throughout the simulation

Movies

The block named `&MOVIE_PARAMS` contains the parameters related to the movies. One can produce up to 5 movies, which center at different parts of the simulation box and have various camera behavior (see example blocks in the bottom).

The movie routine is very useful for creating on the fly visualizations of a simulation with a small time step between frames, without having to write and process many huge snapshots for that. For turning the binary images created by this routine into a movie that can be watched on any media player, check out RAM .

Variable name, syntax, default value	Fortran type	Description
`movie`	`boolean`	Turns movies on and off
`imov`	`integer`	Number of starting frame
`imovout`	`integer`	Total number of frames
`tstartmov`	`float`	Start time of the movie
`astartmov`	`float`	Start aexp of the movie

Variable name, syntax, default value	Fortran type	Description
`tendmov`	`float`	End time of the movie
`aendmov`	`float`	End aexp of the movie
`levelmax_frame`	`integer`	Maximum level of the frame
`nw_frame`	`integer`	Width of frame in px
`nh_frame`	`integer`	Height of frame in px
`xcentre_frame`	5* `float, float, float, float`	Four (**for each projection**) coordinates for z position of frame centre [code units] - spline coefficients
`ycentre_frame`	5* `float, float, float, float`	Four (**for each projection**) coordinates for y position of frame centre [code units] - spline coefficients
`zcentre_frame`	5* `float, float, float, float`	Four (**for each projection**) coordinates for z position of frame centre [code units] - spline coefficients
`deltax_frame`	`float, float`	Two coordinates for x delta of frame [code units]
`deltay_frame`	`float, float`	Two coordinates for y delta of frame [code units]
`deltaz_frame`	`float, float`	Two coordinates for z delta of frame [code units]

Variable name, syntax, default value	Fortran type	Description
`zoom_only_frame`	5* `boolean`	Consider only particles in the zoom region of cosmological runs
`proj_axis`	5* `character`	Letter code of projection axis; x, y, z - maximum 5 line of sights
`movie_vars_txt`	`strings`	Determines which variables to save - `dens` , `temp` , `pres` , `vx` , `vy` , `vz` , `varX` , `FpX` (RT-only), `stars` (star particles), `dm` (dark matter particles), `lum` (stellar luminosity)
`theta_camera`	5* `float`	Azimuthal angle of the camera with respect to the line of sight [degrees]
`phi_camera`	5* `float`	Polar angle of the camera with respect to the line of sight [degrees]
`dtheta_camera`	5* `float`	Azimuthal angle rotation completed between `tstartmov` and `tendmov` [degrees]
`dphi_camera`	5* `float`	Polar angle rotation completed between `tstartmov` and `tendmov` [degrees]
`focal_camera`	5* `float`	Distance of the focal plane of the camera [code units]. Camera distance is set to `boxlen` and `focal_camera` is set to `boxlen` by default.
`dist_camera`	5* `float`	Distance of the camera [code units]. Camera distance is set to `boxlen` and `focal_camera` is set to `boxlen` by default.

Variable name, syntax, default value	Fortran type	Description
`ddist_camera`	5* `float`	Motion of the camera between `tstartmov` and `tendmov` [code units]
`perspective_camera`	5* `boolean`	Perspective corrections for the projected cells
`shader_frame`	5* `strings`	Shader applied for the leaf cells projection [`square` , `sphere` , `cube`]
`ivar_frame`	5* `integer`	Index of hydro variable to use for selecting cells to project (default=-1)
`varmin_frame`	5* `integer`	Only project cells within `varmin_frame<uold(*,ivar_frame)<varmax_frame` . Density is expressed in cm^-3, velocities in km/s, temperature in K/mu, all other hydro variables in code units.
`varmax_frame`	5* `integer`	Only project cells within `varmin_frame<uold(*,ivar_frame)<varmax_frame` . Density is expressed in cm^-3, velocities in km/s, temperature in K/mu, all other hydro variables in code units.
`method_frame`	5* `strings`	Available projection methods: `mean_mass` (default), `mean_dens` , `mean_vol` , `sum` , `min` , `max`

Examples:

- cosmological simulation

```
#!fortran
&MOVIE_PARAMS
```

```
movie=.true.
imov=0
aendmov=1.
imovout=1000
nw_frame=1920
nh_frame=1080
levelmax_frame=18
xcentre_frame=0.49944825,-0.00391524,0.02661462,-0.01714339
ycentre_frame=0.49512072,0.00498905,-0.01826436,0.01097619
zcentre_frame=0.483284613,0.0246328776,-0.0149075526,-1.06750114e-04
deltax_frame=0.0,0.001
deltay_frame=0.0,0.0005625
deltaz_frame=0.0,0.001
proj_axis='z'
movie_vars_txt='dm','stars','temp','dens','var6'
/
```

This will result in 1000 frames 1920x1080 (Full HD) between starting redshift and z=0. Values for `xcentre_frame` , `ycentre_frame` and `zcentre_frame` come from fitting the spline coefficients with `utils/py/get_spline_coeffs.py` (it requires previous run of the same setup with clump finder; e.g. DM-only). Size of the frame will be reflecting physical units (e.g. delta_x=deltax_frame[0]+deltax_frame[1]/a; where a is the explansion factor). Frames are going to be projected along the z axis and projections of dark matter density, stellar density, temperature, gas density and var6 (extra variable, here metallicity) will be saved.

- non-cosmological

```
#!fortran
&MOVIE_PARAMS
movie=.true.
imov=0
tendmov=10.
imovout=1000
nw_frame=1920
nh_frame=1080
levelmax_frame=14
xcentre_frame=5.0,0.,0.,0.,5.0,0.,0.,0.
ycentre_frame=5.0,0.,0.,0.,5.0,0.,0.,0.
zcentre_frame=5.0,0.,0.,0.,5.0,0.,0.,0.
deltax_frame=1.0,0.,1.0,0.
deltay_frame=1.0,0.,1.0,0.
deltaz_frame=1.0,0.,1.0,0.
proj_axis='zy'
movie_vars_txt='dm','stars','temp','dens','var6'
/
```

Same variables as for the cosmological run are going to be saved with the same resolution. Here, the camera is always centred at the centre of the box (assuming boxlen=10) and encompasses volume of 10% x 10% x 10% of the box. Tendmovie is set in the code units of time.

Turbulence driving

The block named `&TURB_PARAMS` contains the parameters related to the turbulence driving. Originally implemented by Andrew Mcleod

Overview of parameters

Variable name	Fortran type	Default value	Description
`turb`	`boolean`	`.false.`	Turn on or off driving
`turb_seed`	`integer`	`-1`	Random number generator seed. -1 = random
`turb_type`	`integer`	`1`	How the driving changes over time. 1=driven evolving, 3=decaying
`instant_turb`	`boolean`	`.true.`	Generate initial turbulence before start
`comp_frac`	`float`	`0.3333`	The weight of compressive over solenoidal modes
`turb_T`	`float`	`1`	Turbulent velocity auto-correlation time in code units.
`turb_Ndt`	`integer`	`100`	Number of timesteps per auto-correlation time

Variable name	Fortran type	Default value	Description
`turb_rms`	`float`	`1`	Root-mean-square turbulent forcing in code units.
`turb_min_rho`	`float`	`1d-50`	Minimum density for turbulence. Not forcing is added onto cellswith a density less than this value.
`forcing_power_spectrum`	`string`	`parabolic`	Power spectrum type of the forcing, which describes the relative strength of individual modes. Options are: power_law, parabolic, konstandin

Monte Carlo tracer particles

The block named `&TRACER_PARAMS` contains the parameters related to the Monte Carlo tracer particles, see Cadiou et al 2018

Overview of parameters

Variable name	Fortran type	Default value	Description
`MC_tracer`	`boolean`	`.false.`	Activate MC tracers
`tracer_feed`	`string`	`none`	Filename to read the tracer from
`tracer_feed_fmt`	`string`	`ascii`	Format of the input (ascii,binary or inplace)

Variable name	Fortran type	Default value	Description
`tracer_mass`	`float`	`-1.0`	Mass of the tracers, used for outputs and seed
`tracer_first_balance_levelmin`	`integer`	`-1`	Set to >0 to add more weight on level finer than this
`tracer_first_balance_part_per_cell`	`integer`	`0`	Typical initial number of parts per cell

Cosmological simulations

In this section, we describe in more detail how ramses can be used to perform cosmological simulations. Useful concepts related to parallel computing or post-processing will be introduced, and can also be used for non-cosmological runs. Cosmological simulations are performed by specifying `cosmo=.true.` in the `&RUN_PARAMS` namelist block.

Restart from previous output

A simulation which has been terminated during run time can be restarted from the last (or any) snapshot output, by setting

```
nrestart=64
```

in the namelist file to the output number. If you don't want to change the namelist file, simply append the restart output number to the command execution, e.g.

```
./ramses3d parameters.nml 64
```

Saving progress before job limit termination

SLURM jobs on computer clusters often have a time limit, after which the process will be terminated. If you don't want to lose the computation progress since your last regular output, you can instruct the SLURM scheduler to send a "warning" signal to the process seconds before killing it with **–signal=10@** . An example sbatch script with set to 120 seconds would look like this:

```
#!/bin/bash
#SBATCH -J simulation
#SBATCH -p normal
#SBATCH -n 128
#SBATCH --time=24:00:00
#SBATCH --output=logfile-%j.txt
#SBATCH --error=error-%j.txt
#SBATCH --signal=10@120

aprun -B ./ramses3d parameters.nml 64
```

RAMSES will catch this signal and dump the current simulation state to a new output, which can be used to restart the simulation from.

The signalling does not work on all machines. Sometimes the signal 10 is accompanied by a kill signal and the job is dead before it can perform an output. In this case, there are a couple of useful parameters in the output_params namelist: walltime_hrs can be used to specify the walltime given to a job in hours, and minutes_dump can then be used to tell RAMSES to dump an output a few minutes before the walltime runs out.

Dump immediate output

The above mechanism can be used to force an output be written to the disk immediately at any time during the simulation by sending signal 10 to the process:

```
scancel --signal=10 <jobid>
```

or, if you run without SLURM:

```
kill --signal=10 <processid>
```

Advanced simulations

In this section, we describe in more detail how to configure RAMSES to perform more advanced simulations.

Changing the source code

In order to perform more customised simulations, the source code of RAMSES can be changed/extended.

There are two different ways in which this task can be managed:

1. Create an appropriately called git branch and all the development changes will be reflected in the code commits. Great variety of resources can be found on this, but see e.g. this brief introduction . This is the **recommended** way of patching RAMSES.
2. Create a patch directory and store there your changed files. This is presently **disfavoured** , but will be discussed here due to historical reasons.

Patch directory

After storing all the changed source files in the directory of choice you have to pass this path to Makefile - adjust the `PATCH` variable (make sure there is no trailing space!).

During the compilation, `ramses/utils/scripts/cr_write_patch.sh` will be invoked. This script prepares a Fortran source code file which contains all the patches as a long string. This is written in each RAMSES output directory for later reference (`patches.txt`).

WARNING : Changes in RAMSES source code that are uncommitted or not in patch directory are not being tracked!

Radiation Hydrodynamics in RAMSES

Radiation hydrodynamics are implemented in RAMSES, as described in those papers:

- RAMSES-RT: radiation hydrodynamics in the cosmological context
- A scheme for radiation pressure and photon diffusion with the M1 closure in RAMSES-RT
- A simple model for molecular hydrogen chemistry coupled to radiation hydrodynamics

Compiling for RHD runs

An example makefile for an RHD compilation, `Makefile.rt` , is included under `ramses/bin` . To activate radiative transfer, you must compile with the flag `-DRT` , and some RHD specific .f90 files must be included in the compilation (all of which is included in the `Makefile` example).

For a run with only atomic hydrogen (i.e. ionisation species HI and HII), you should use `NIONS=1` . For including molecular hydrogen, add one to `NIONS` , and for including helium ionization, add two to `NIONS` . The number of hydro variables needs to increase accordingly, but this is done automatically in `Makefile.rt` . You also set the number of photon groups in the Makefile with the `NGROUPS` parameter. When running with IR radiation trapping (`rt_isIRtrap=.true.`), you must also compile with a dedicated non-thermal energy variable, by setting `NENER=1` in `Makefile.rt` (`NVAR` is incremented automatically in the `Makefile`).

RHD outputs

The radiation field variables are written separately in each output to files named `rt_XXXXX.outYYYYY` , where XXXXX is the output number and YYYYY the cpu number. The naming convention and format of the files is exactly the same as for the hydro variables. For each photon group, there are four cell variables, `c_r*N` , `Fx` , `Fy` , `Fz` , where `c_r` is the reduced light speed, `N` the photon number density, and the rest are the photon number flux in the x, y, and z directions. The factors for converting those to cgs units are stored in a file named `info_rt_XXXXX.txt` in each output directory (`unit_np` and `unit_pf`), along with the reduced light speed and the photon group properties. The non-thermal energy (trapped IR radiation pressure) is stored in runtime next after the thermal pressure, but in the output it comes just before.

RHD Runtime parameters

Radiative transfer is activated by setting `rt=.true.` in `&RUN_PARAMS` . For RT post-processing, set also `static_gas=.true.` in the same namelist. Note that this is not sufficient to turn on the advection of radiation – this is done in the code (with `rt_advect=.true.`) only if sources of radiation (stars, gas, or idealised sources) are detected in the run.

If `rt=.true.` , non-equilibrium thermochemistry of hydrogen (and optionally helium) is used instead of the default equilibrium chemistry in RAMSES. The equilibrium chemistry can also be turned on without RHD, by setting `neq_chem=.true.` in `&PHYSICS_PARAMS` (but you must still compile with `Makefile.rt` , with `NGROUPS=0`).

For RHD runs, there are two additional dedicated namelists:

- RT_PARAMS
- RT_GROUPS

Generation of spectral energy distribution (SED) tables for RHD simulations

The (metallicity and age dependent) radiative luminosity of stellar population particles and photon group properties are calculated on-the-fly in RAMSES runs using SED tables. RAMSES-readable tables can be generated from Starburst99 and BC03 formats using a python utility found in `utils/py/sed_utils.py` . The generated files are linked to the RAMSES run with the `sed_dir` parameter in the `&RT_PARAMS` namelist.

Testing

1. Running the automatic test suite

To run the automatic tests, navigate to the tests directory, and run the `run_test_suite.sh` script:

```
>$ cd tests
>$ ./run_test_suite.sh
```

The tests will begin and the output should look like:

```
################################################
#   Running RAMSES automatic test suite    #
################################################
Will perform the following tests:
 [ 1] hydro/implosion
 [ 2] hydro/sod-tube
 [ 3] mhd/imhd-tube
 [ 4] mhd/orszag-tang
 [ 5] rt/stromgren2d
 [ 6] sink/smbh-bondi
------------------------------------------------
Test 1/6: hydro/implosion
Cleanup
Compiling source
```

and so on. Once the tests have completed, a report is generated in a `.pdf` file named `test_results.pdf` , alongside a log file `test_suite.log` .

Options

- Run the suite in parallel (on 4 cpus):

```
./run_test_suite.sh -p 4
```

- Do not delete results data:

```
./run_test_suite.sh -d
```

- Run in verbose mode:

```
./run_test_suite.sh -v
```

- Select individual tests (for tests 3 to 5, and 10):

```
./run_test_suite.sh -t 3-5,10
```

- Run all tests in `mhd` directory:

```
./run_test_suite.sh -t mhd
```

2. Creating a new test

The following steps describe how to add a new test to the test suite. In this example, the test will be named `sedov-3d` .

The first step is to create a new directory `sedov-3d` in one of the `hydro` , `mhd` , `rt` , or `sinks` directories. No need to modify the `run_test_suite.sh` script, the new test will automatically be picked up and added to the list. We will choose to place it inside the `hydro` directory. Please use hyphens (`-`) in your test names instead of underscores (`_`) as `latex` does not like underscores.

```
>$ cd hydro
>$ mkdir sedov-3d
```

Note : use one directory per test. If you want to run a 2D and a 3D sedov test, create separate `sedov-2d` and `sedov-3d` directories.

In that directory, you will need:

- A `config.txt` file: usually just contains the Makefile flags, e.g. `FLAGS: NDIM=3 PATCH= SOLVER=hydro`

- A namelist: `sedov-3d.nml` (the name needs to be the same as the test directory)
- A file for plotting and checking the solution against a reference: `plot-sedov-3d.py` . It is advised to copy a file from the other directories to see how to write this. **Note that this file needs to contain at least one call to** `visu_ramses.check_solution(data["data"], 'sedov-3d')` .
- A reference solution: `sedov-3d-ref.txt` . To create it, run your test and once the final output (number 2 in this case) has been created, do the following:

```
import visu_ramses
data = visu_ramses.load_snapshot(2)
visu_ramses.check_solution(data["data"], 'sedov-3d', overwrite=True)
```

- A `Readme.md` containing a short description of the test

Optional files:

- `condinit.f90` : you can have your own initial setup if it's not entirely definable in a namelist. REMEMBER to set the correct `PATCH` in the `config.txt` file! (e.g. `PATCH=../tests/hydro/sedov-3d`)
- `before-test.sh` : if this file is present in the test directory, it will be run before the test begins (useful for e.g. creating symbolic links to libraries...)
- `after-test.sh` : if this file is present in the test directory, it will be run after the test begins (useful for e.g. cleaning up symbolic links to libraries...)

Tuning tolerances for solution verification

By default, relative differences between the sums of all the variables inside all leaf cells in the domain and the reference solution cannot exceed `3.0e-13` . Sometimes, some variables are more volatile than others when running simulations on different numbers of CPUs, and this limit is too low, leading to false failed tests. The `check_solution` method in the `visu/visu_ramses.py` module can be tuned to work for your test using the following options:

- `tolerance` : a dictionary listing the allowed relative difference between the sum over all leaf cells and reference value. The default is `{"all":3.0e-13}` . To make the check on `density` less restrictive, use for instance `tolerance={"density":1.0e-10}` .
- `threshold` : relative value below which a vector component is set to zero. Default is `2.0e-14` .
- `norm_min` : minimum value for the norm of a vector, to protect against null vectors. Default is `1.0e-30` .

- `min_variance` : if the data differs by less than this value from the average value, it is set to the average. Default is `1.0e-14` .

3. Creating a new group of tests

If your test does not fall under the categories already present in the `tests` directory (`hydro` , `mhd` , `rt` , `sinks`), you can create a new directory and put your tests in there. You will then have to edit the `run_test_suite.sh` file to ensure your new tests will be picked up.

Say you want to create 3 new tests, `sedov-1d` , `sedov-2d` , and `sedov-3d` inside a new `sedov` directory, you have to find the line describing the list of directories to be scanned at the top of the `run_test_suite.sh` file:

```
# List of directories to scan
testlist="hydro,mhd,rt,sink";
```

and add your new directory separated from the previous one by a comma, i.e.

```
# List of directories to scan
testlist="hydro,mhd,rt,sink,sedov";
```

Documentation

Some notes about the Sphinx and Read-The-Docs (RTD) documentation builders:

- The documentation has two sections, the user documentation section with files located in the 'wiki' directory and developer documentation in the `dev_docs` directory. When a PDF is generated, only the wiki version is included.
- Requirements for the docs can be installed via `pip install -r doc/rtd_requirements.txt` .
  - For MacOS pdf builds you may need to install `pango` as follows `brew install pango` . See `sphinx_simplepdf` docs at the link below.
- The build can be configured in `doc/conf.py` .
  - To build pdf locally run `sphinx-build -M html doc` or in the `doc` folder `make html` . The html will be built in `_build/html/index.html`
- To use markdown files, i.e `.md` files, the `myst_parser` extension is used. Documentation for `myst_parser` .

- To convert the docs to PDF, `sphinx_simplepdf` extension is used.
  - The RTD build is configured in `.readthedocs.yaml` under the `commands` list.
  - If `sphinx_simplepdf` ever breaks, you can remove the custom build instructions there.
  - To build pdf locally run `sphinx-build -M simplepdf . _build` .
  - Documentation for `sphinx_simplepdf` .
- As of this version, the color scheme for the docs are the following (can be changed in `doc/conf.py`):
  - `primary_color = '#000000'`
  - `secondary_color = '#FFD587'`
  - `text_color = '#000000'`
- If creating a page without listing it in the toctree, add the page to `doc/wiki/orphan_pages.rst` .
  - If you do not do this, you will get the following error: `WARNING: document isn't included in any toctree` .

Acknowledgements

The development of the RAMSES code has been initiated and coordinated by the main author. The main author would like to thank all co-authors who took an active role in the development of this version. They are cited in alphabetical order.

- Dominique Aubert (radiative transfer, initial conditions)
- Edouard Audit (radiative transfer)
- Andreas Bleuler (sink particle, clump finder)
- Stephane Colombi (cooling and atomic physics)
- Benoit Commercon (radiative transfer, MHD)
- Stephanie Courty (cooling and atomic physics)
- Julien Devriendt (Hilbert curve)
- Emmanuel Dormy (MHD)
- Yohan Dubois (supernovae feedback, AGN feedback, MHD)

- Sebastien Fromang (MHD, relativistic HD)
- Claudio Gheller (GPU optimisation)
- Matthias Gonzalez (radiative transfer, initial conditions)
- Thomas Guillet (Multigrid Poisson solver)
- Patrick Hennebelle (MHD)
- Troels Haugboelle (MHD, gravity solver)
- Astrid Lamberts (relativistic HD)
- Davide Martizzi (AGN feedback, clump finder)
- Aake Nordlund (MHD, gravity solver)
- Joakim Rosdahl (radiative transfer)
- Yann Rasera (star formation)
- Philippe Series (code optimization)
- Neil Vaytet (automatic testing)
- Philippe Wautelet (code optimization)