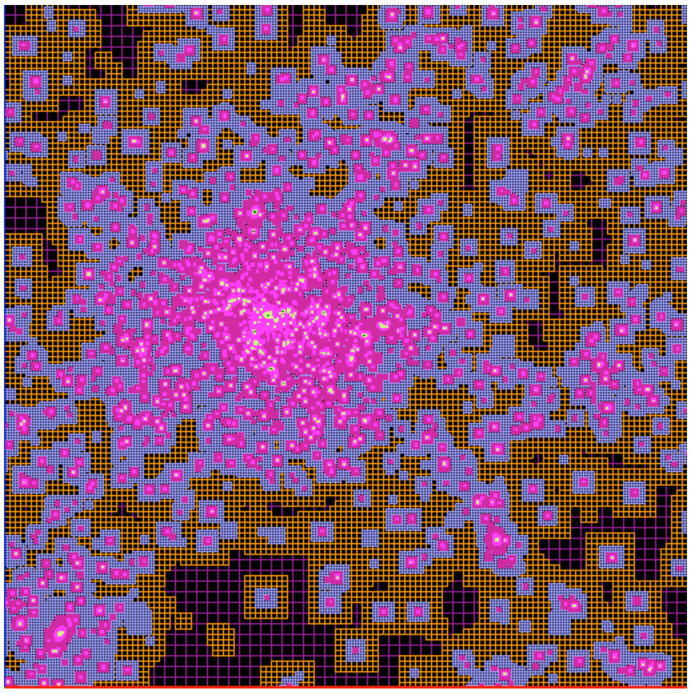

RAMSES User's Guide

Self-gravitating fluid dynamics

with Adaptive Mesh Refinement

using massively parallel computers

dapnia

cea

saclay

Romain Teyssier

Version Issue: Version 1.0 Last Update: 15 May 2006

Table of Contents

Introduction

The RAMSES package is intended to be a versatile platform to develop applications using Adaptive Mesh Refinement for computational astrophysics. The current implementation allows solving the Euler equations in presence of self-gravity and cooling, treated as additional source terms in the momentum and energy equations. The RAMSES code can be used on massively parallel architecture when properly linked to the MPI library. It can also be used on single processor machine without MPI. Output files are generated using native RAMSES 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.

About This Guide

The goal of this User's Guide is to provide a step-by-step tutorial in using the RAMSES code. This guide will first describe simple example of its use. More complex set-up will be addressed at the end of the document. Typical RAMSES users can be regrouped in 3 categories:

- Beginners: It is possible to execute RAMSES using only run parameter files. The code is compiled once, and the user only modifies the input file to perform various simulations. Cosmological simulations can be performed quite efficiently in this way, using the initial conditions generated by external packages such as **`mpgrafic`**.

- Intermediate users: For more complex applications, the user can easily modify a small set of routines in order to specify specific initial or boundary conditions. These routines are called "patches" and the code should be re-compiled each time these routines are modified.

- Advanced users: It is finally possible to modify the base scheme, add new equations, or add new routines in order to modify the default RAMSES application. This guide will not describe these advanced features. In this case, a new documentation would be given separately.

Getting RAMSES

RAMSES software can be downloaded in the *Codes* section from http://www.projet-horizon.fr. It is freely distributed under the CeCILL software license (http://www.cecill.info) according the French legal system *for non-commercial use only*. For commercial use of RAMSES, please contact the author: be prepared for a massive financial compensation.

Main features

RAMSES contains various algorithms designed for:

- Cartesian AMR grids in 1D, 2D or 3D

- Solving the Poisson equation with a Multi-grid and a Conjugate Gradient solver

- Using various Riemann solvers (Lax-Friedrich, HLLC, exact) for adiabatic gas dynamics

- Computing collision-less particles (dark matter and stars) dynamics using a PM code
- Computing the cooling and heating of a metal-rich plasma due to atomic physics processes and an homogeneous UV background (Haardt and Madau model).
- Implementing a model of star-formation based on a standard Schmidt law with the traditional set of parameters.
- Implementing a model of supernovae-driven winds based on a local Sedov blast wave solution.

All these features can be used and parameterized using the RAMSES parameter file, based on the Fortran "namelist" technique.

Acknowledgements

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

- Matthias Gonzalez (initial conditions)
- Stéphane Colombi (cooling and atomic physics)
- Yann Rasera (star formation, post-processing)
- Dominique Aubert (initial conditions)
- Philippe Wautelet (code optimization)
- Philippe Sériès (code optimization)

I would like to thanks my collaborators for helping me developing more advanced versions of RAMSES, not yet available as complete releases, since it is mostly work in progress.

- Benoit Commerçon (thermal conduction)
- Sébastien Fromang, Patrick Hennebelle and Emanuel Domry (MHD).
- Edouard Audit and Dominique Aubert (radiative transfer)
- Remi Abgrall and Richard Saurel (multifluid)

The CeCILL License

This software is under Copyright of CEA and its author, Romain Teyssier.

This software is governed by the CeCILL license under French law and abiding by the rules of distribution of free software. You can use, modify and/ or redistribute the software under the terms of the CeCILL license as circulated by CEA, CNRS and INRIA at the following URL http://www.cecill.info.

As a counterpart to the access to the source code and rights to copy, modify and redistribute granted by the license, users are provided only with a limited warranty and the software's author, the holder of the economic rights, and the successive licensors have only limited liability.

In this respect, the user's attention is drawn to the risks associated with loading, using, modifying and/or developing or reproducing the software by the user in light of its specific status of free software, that may mean that it is complicated to manipulate, and that also therefore means that it is reserved for developers and experienced professionals having in-depth IT knowledge. Users are therefore encouraged to load and test the software's suitability as regards their requirements in conditions enabling the security of their systems and/or data to be ensured and, more generally, to use and operate it in the same conditions as regards security.

The fact that you are presently reading this means that you have had knowledge of the CeCILL license and that you accept its terms.

Getting started

In this section, 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 web site http://www.projet-horizon.fr in the *Codes* section. You will get a tar ball named **`ramses.tar.gz`**. The first thing to do is to un-tar the archive on your preferred computer home directory.

```
$ gunzip ramses.tar.gz | tar xvf
```

This will create a new directory named **`ramses`**. In this directory, you will find the followings directory list

```
amr/  bin/ doc/ hydro/ namelist/ patch/ pm/ poisson/ utils/
```

Each directory contains a set of files with a given common purpose. For example, **`amr/`** contains all F90 routines dealing with the AMR grid management and MPI communications, while **`hydro/`** obviously contains all F90 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

In this **`bin/`** directory, you will find a Makefile. 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=pgf90
FFLAGS=-Mpreprocess -DWITHOUTMPI –DNDIM=$(NDIM) –DSOLVER=$(SOLVER)
```

The first variable is obviously the command used to invoke the Fortran compiler. In this case, this is the Portland Group compiler. The second variable contains Fortran flags and preprocessor directives. The first directive, **`-DWITHOUTMPI`**, when used, switch 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. Theses directives are called *Compilation Time Parameters*. They should be defined within the Makefile. Default values are

```
NDIM=1
SOLVER=hydro
```

The first variable, **`NDIM`**, sets the dimensionality of the problem. The default value is for 1D, plan-parallel flows. Allowed values are 1, 2 and 3. The second directive defines the solver, which in the current release of RAMSES is only **`hydro`**. Other solvers are currently under development, such as **`mhd`**, **`rad`** and so on.

There are 3 other preprocessor directives that can be use in RAMSES: **`-DNVAR=NDIM+2`**, useful to set more variables in the hydro solver, **`-DNPRE=8`**, to set the number of bytes used for real numbers. **NPRE=8** corresponds to double precision arithmetic and **NPRE=4** to single precision. This option is useful to save memory during memory intensive runs. Finally, you can use **-DNVECTOR=500** to set the size of the vector sweeps for computationally intensive operations.

To compile RAMSES, execute

```
$ make
```

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

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

```
$ bin/ramses1d namelist/tube1d.nml
```

The first part of the command is the executable, and the second part, the only command line argument, is an input file containing all *Run Time Parameters*. Several examples of such parameter files are given in directory **namelist/**. The run we have just performed, **tube1d.nml**, is the Sod's test, a simple shock tube simulation in 1D. For sake of 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         15 grids (      15,      15,      15,)
 Level 10 has         12 grids (      12,      12,      12,)
 Main step=    43 mcons= 0.00E+00 econs= 0.00E+00 epot= 0.00E+00 ekin= 0.00E+00
 Fine step=   689 t= 2.45202E-01 dt= 3.560E-04 a= 1.000E+00 mem= 7.5%
 Run completed
```

To save the standard output in a file, the user is encouraged to redirect the standard output in a *Log File*, in which all control variables are outputted and stored, as well as simulation data for 1D cases only

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

Reading the Log File

We will now briefly describe the structure and the nature of the information available in the Log Files. We will use as example the file **`tube1d.log`**, which should contain, starting from the top

```
_/_/_/       _/_/     _/    _/    _/_/_/   _/_/_/_/    _/_/_/
 _/    _/    _/  _/    _/_/_/_/   _/    _/  _/         _/    _/
 _/    _/   _/    _/   _/ _/ _/   _/        _/         _/
 _/_/_/     _/_/_/_/   _/    _/     _/_/    _/_/_/       _/_/
 _/    _/   _/    _/   _/    _/         _/  _/               _/
 _/    _/   _/    _/   _/    _/   _/    _/  _/         _/    _/
 _/    _/   _/    _/   _/    _/    _/_/_/   _/_/_/_/    _/_/_/
                         Version 2.0
        written by Romain Teyssier (CEA/DSM/DAPNIA/SAP)
                      (c) CEA 1999-2005

 Working with nproc =    1 for ndim = 1

 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
 Fine step=     0 t= 0.00000E+00 dt= 6.603E-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 code then reports that it is building the initial AMR grid, and give the initial mesh structure. The following lines give the current mesh structure. The first level of refinement in RAMSES covers the whole computational domain with 2, 4 or 8 cells in 1, 2 or 3 space dimensions. The grid is then further refined up to **`levelmin`**, which is this case is defined in the parameter file to be **`levelmin=3`**. The grid is then further refined up to **`levelmax`**, which is 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 is equal to 2^{l-1} in 1D, to 4^{l-1} in 2D and to 8^{l-1} in 3D. The numbers inside the parenthesis give the minimum, maximum and average number of grids per processor. This is obviously relevant for parallel runs only.

The code then indicates the time integration starts, and 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 AMR grid and to store each collision-less particle, if any.

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, for 689 Fine Steps. 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, the data are also outputted using the standard output, and thus the Log File. For 2D and 3D, data are outputted using unformatted Fortran binary files. 1D data are shown using 5 columns (level of refinement, position of the cell, density, velocity and pressure) as in the following Sod's test example

```
Output   140 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  1.467E-09  1.000E+00
 6  1.32813E-01  1.000E+00  1.972E-08  1.000E+00
 6  1.48438E-01  1.000E+00  2.619E-07  1.000E+00
 6  1.64063E-01  1.000E+00  3.423E-06  1.000E+00
 6  1.79688E-01  1.000E+00  4.081E-05  1.000E+00
 6  1.95313E-01  9.996E-01  4.921E-04  9.994E-01
 7  2.07031E-01  9.955E-01  5.299E-03  9.938E-01
 7  2.14844E-01  9.838E-01  1.927E-02  9.774E-01
 7  2.22656E-01  9.660E-01  4.082E-02  9.527E-01
 7  2.30469E-01  9.451E-01  6.650E-02  9.240E-01
 7  2.38281E-01  9.232E-01  9.388E-02  8.942E-01
 7  2.46094E-01  9.016E-01  1.214E-01  8.650E-01
```

You can cut and paste the 140 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 in Figure 1. 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.log 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.

Do not forget to recompile entirely the code with **`NDIM=2`** for 2D cases like **`sedov2d.nml`** or **`NDIM=3`** for 3D cases such as **`sedov3d.nml`**.

In the next section, we will describe in more details the various Runtime Parameters available within RAMSES.

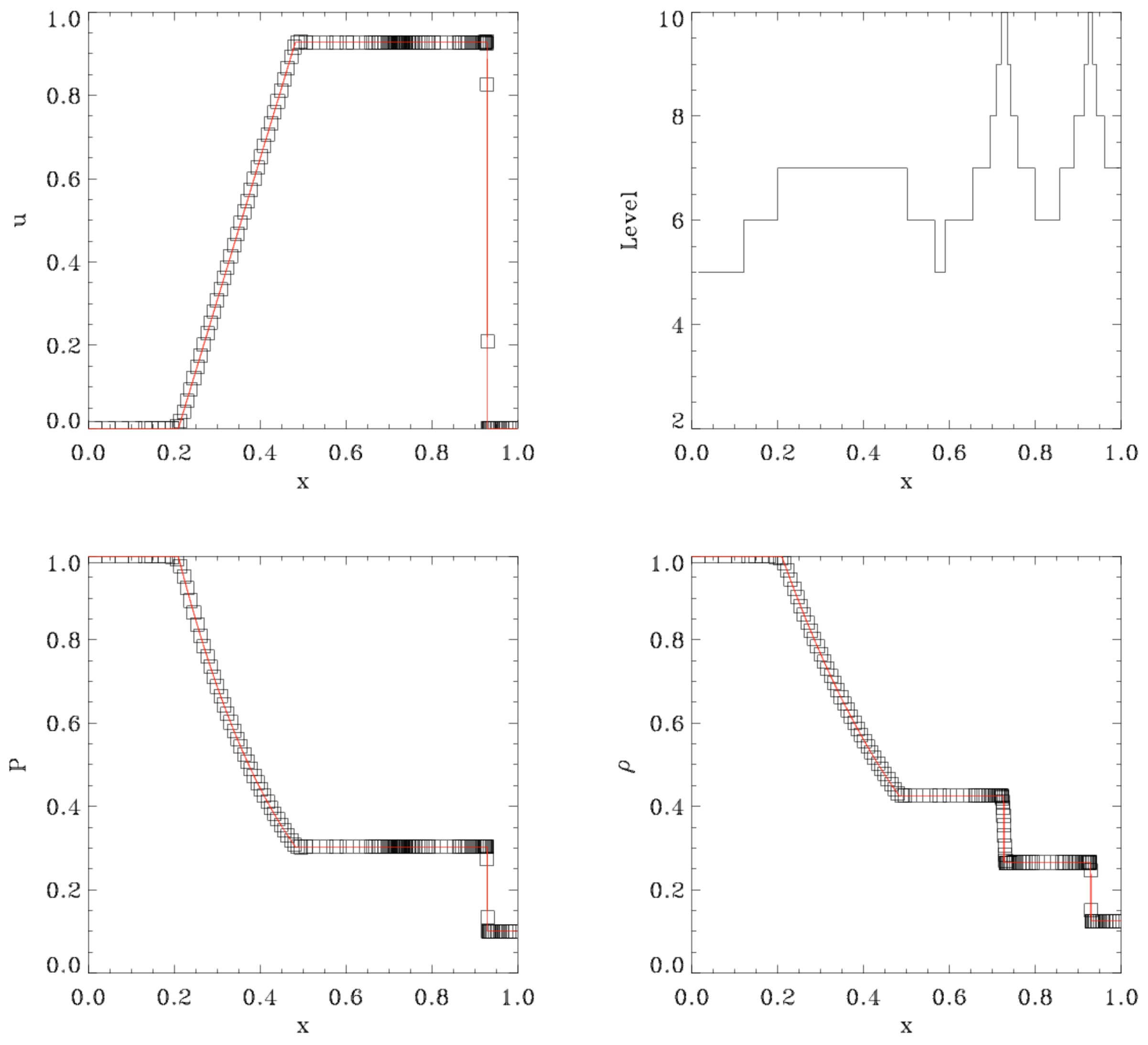

Figure 1: Numerical results obtained with RAMSES for the Sod shock tube test (symbols) compared to the analytical solution (red line).

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.

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

&AMR_PARAMS
levelmin=3
levelmax=10
ngridmax=10000
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
output_mode=1
noutput=1
tout=0.245
/

&HYDRO_PARAMS
gamma=1.4
courant_factor=0.8
slope_type=2
scheme='muscl'
/

&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 character **`/`**. Within each block, you can specify parameter values using standard Fortran namelist syntax. There are currently 9 different parameter blocks implemented in RAMSES.

⚠ 4 parameter blocks are mandatory and must always be present in the parameter file. These are **`&RUN_PARAMS`**, **`&AMR_PARAMS`**, **`&OUTPUT_PARAMS`** and **`&INIT_PARAMS`**.

The 5 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`** and finally **`&REFINE_PARAMS`**. We now describe each parameter block in more details.

Global parameters

This block, called **`&RUN_PARAMS`**, contains the run global control parameters. These parameters are now briefly described. More complex explanation will be given in dedicated sections.

Variable name, syntax and default value	*Fortran type*	*Description*
`cosmo=.false.`	Boolean	Activate cosmological "super-comoving coordinates" system and expansion factor computing.
`pic=.false.`	Boolean	Activate Particle-In-Cell solver
`poisson=.false.`	Boolean	Activate Poisson solver.
`hydro=.false.`	Boolean	Activate hydrodynamics solver.
`verbose=.false`	Boolean	Activate verbose mode
`nrestart=0`	Integer	Backup 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. You should use the same number of processors than the one used during the backup run.
`nstepmax=1000000`	Integer	Maximum number of coarse time steps.
`ncontrol=1`	Integer	Frequency of screen output for Control Lines (for standard output of into the Log File).
`Nremap=0`	Integer	Frequency of calls, in units of coarse time steps, for the load balancing routine, for MPI runs only, the default value, zero, meaning "never".
`ordering='hilbert'`	Character LEN=128	Cell ordering method used in the domain decomposition of the grid among the processors, for MPI runs only. Possible values are **`hilbert`**, **`planar`** and **`angular`**.
`nsubcycle=2,2,2,2,2,`	Integer array	Number of fine level sub-cycling within one coarse level time step. Each value corresponds to a given level of refinement, starting from the coarse grid defined by **`levelmin`**. , up to the finest level defined by **`levelmax`**. For example, **`nsubcycle(1)=1`** means that **`levelmin`** and **`levelmin+1`** are synchronized. To enforce single time stepping for the whole AMR hierarchy, you need to set **`nsubcycle=1,1,1,1,1,`**

AMR grid

This set of parameters, called **`&AMR_PARAMS`**, controls the AMR grid global properties. Parameters specifying the refinement strategy are described in block **`&REFINE_PARAMS`**, which is used only if **`levelmax>levelmin`**.

Variable name, syntax and default value	***Fortran type***	***Description***
`levelmin=1`	Integer	Minimum level of refinement. This parameter sets the size of the coarse (or base) grid by **`nx=2`**$^{\texttt{levelmin}}$.
`levelmax=1`	Integer	Maximum level of refinement. If **`levelmax=levelmin`**, this corresponds to a standard Cartesian grid.
`ngridmax=0`	Integer	Maximum number of grids (or octs) that can be allocated during the run *within each MPI process*.
`ngridtot=0`	Integer	Maximum number of grids (or octs) that can be allocated during the run *for all MPI processes*. One has in this case **`ngridmax=ngridtot/ncpu`**.
`npartmax=0`	Integer	Maximum number of particles that can be allocated during the run *within each MPI process*.
`nparttot=0`	Integer	Maximum number of particles that can be allocated during the run *for all MPI processes*. Obviously, one has in this case **`npartmax=nparttot/ncpu`**.
`nexpand=1`	Integer	Number of mesh expansion (mesh smoothing).
`boxlen=1.0`	Real	Box size in user units

Initial conditions

This namelist block, called **`&INIT_PARAMS`**, is used to set up the initial conditions.

Variable name, syntax and 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'`	Character LEN=10 array	Geometry defining each region. **`square`** defines a generalized ellipsoidal shape, while **`point`** defines a delta function in the flow.

`x_center=0.0`
`y_center=0.0`
`z_center=0.0`
	Real arrays	Coordinates of the center of each region.

`length_x=0.0`
`length_y=0.0`
`length_z=0.0`
	Real arrays	Size in all directions of each region.

`exp_region=2.0`	Real array	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`
`u_region=0.0`
`v_region=0.0`
`w_region=0.0`
`p_region=0.0`
	Real arrays	Flow variables in each region (density, velocities and pressure). For **`point`** regions, these variables are used to defined extensive quantities (mass, velocity and specific pressure).

`filetype='ascii'`	Character LEN=20	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 (**`grafic`** or **`ascii`**).
`multiple=.false.`	Boolean	If **`.true.`**, each processor reads its own IC file (**`grafic`** or **`ascii`**). For parallel runs only.
`initfile=' '`	Character LEN=80 array	Directory where IC files are stored. See initial conditions section for details.

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 and default value	***Fortran type***	***Description***
`noutput=1`	Integer	Number of specified output time. At least one output time should be given, corresponding to the end of the simulation.
`tout=0.0,0.0,0.0,`	Real array	Value of specified output time.
`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".
`fbackup=0`	Integer	Frequency of output for backup files, in units of coarse time steps. These files are put into incrementing directories called **`backup_00001/`**, **`backup_00002/`** and so on. The default value, zero, corresponds to "never".
`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 superseeded by this parameter.

Boundary conditions

This namelist block, called **`&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 reader is invited to read the corresponding section. 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. Not that the order that boundary regions are following in the namelist is very important, especially when one defined reflexive or zero gradient boundaries. Specific examples can be found in this document or in the **`namelist/`** directory of the package.

Variable name, syntax and 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	Same thing for the upper, right and upper corner of each boundary region.

Hydrodynamics solver

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 and default value	***Fortran type***	***Description***
`gamma=1.4`	Real	Adiabatic exponent for the perfect gas EOS.
`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='acoustic'`	Character LEN=20	Name of the desired Riemann solver. Possible choices are **`'exact'`**, **`'acoustic'`** or **`'llf'`**.
`niter_riemann=10`	Integer	Maximum number of iterations used in the exact Riemann solver.
`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 MC limiter. In 1D runs only, it is possible to choose also **`slope_type=4`**: Superbee limiter **`slope_type=5`**: Ultrabee limiter
`pressure_fix=.false.`	Logical	Activate hybrid scheme (conservative or primitive) for high-Mach flows. Useful to prevent negative temperatures.

Physical parameters

This namelist, called **`&PHYSICS_PARAMS`**, is used to specify physical quantities used in cosmological applications (cooling, star formation and supernovae feedback). We briefly describe them now.

Variable name, syntax and default value	***Fortran type***	***Description***
`cooling=.false.`	Logical	Activate the cooling and/or heating source term in the energy equation.
`metal=.false.`	Logical	Activate metal enrichment, advection and cooling. In this case, preprocessor directive **`–DNVAR=6`** should be added in the Makefile before the compilation.
`haardt_madau=.false.`	Logical	Use the UV background model of Haardt and Madau. Default value **`.false.`** corresponds to a simple analytical model with parameters **`J21`** and **`a_spec`**.
`J21=0.0`	Real	Normalization for the UV flux of the simple background model. Default means "no UV".
`a_spec=1.0`	Real	Slope for the spectrum of the simple background model. Default value corresponds to a standard "quasars + OB stars" spectrum.
`z_ave=0.0`	Real	Average metallicity used in the cooling function, in case **`metal=.false.`**
`t_star=0.0`	Real	Star formation time scale in Gyr. Default value of zero means no star formation.
`n_star=0.1,` **`del_star=200.0`**	Real	Typical interstellar medium physical density or commoving overdensity, used as star formation density threshold and as EOS density scale.
`T2_star=0.0,` **`g_star=1.6`**	Real	Typical interstellar medium polytropic EOS parameters.
`eta_sn=0.0,` **`yield=0.1`**	Real	Mass fraction of newly formed stars that explode into supernovae. Default value of zero means no supernovae feedback.

Poisson solver

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

Variable name, syntax and default value	***Fortran type***	***Description***
`gravity_type=0`	Integer	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=1d-4`	Real	Stopping criterion for the iterative Poisson solver: residual 2-norm should be lower than **`epsilon`** times the right hand side 2-norm.
`gravity_params=0.0, 0.0, 0.0, 0.0,`	Real array	Parameters used to define the analytical gravity field (routine **`gravana.f90`**) or the analytical mass density field (routine **`rho_ana.f90`**).

Refinement strategy

This namelist, **`&REFINE_PARAMS`**, is used to specify refinement parameters controlling the AMR grid generation and evolution during the course of the run. It is used only if **`levelmax> levelmin`**.

Variable name, syntax and default value	***Fortran type***	***Description***
`mass_sph=0.0`	Real	Quasi-Lagrangian strategy: **`mass_sph`** is used to set a typical mass scale. For cosmo runs, its value is set automatically.
`m_refine=-1.,-1.,-1.,`	Real array	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.
`jeans_refine=-1.,-1.,`	Real array	Jeans refinement strategy: each level is refined if the cell size exceeds the local Jeans length divided by **`jeans_refine(ilevel)`**.

`floor_d=1d-10,`
`floor_u=1d-10,`
`floor_p=1d-10`
	Real	Discontinuity-based strategy: density, velocity and pressure floor below which gradients are ignored.

`err_grad_d=-1.0,`
`err_grad_u=-1.0,`
`err_grad_p=-1.0`
	Real	Discontinuity-based strategy: density, velocity and pressure relative variations above which a cell is refined.

`x_refine=0.0,0.0,0.0,`
`y_refine=0.0,0.0,0.0,`
`z_refine=0.0,0.0,0.0,`
	Real arrays	Geometry-based strategy: center of the refined region at each level of the AMR grid.

`r_refine=1d10,1d10,`
`a_refine=1.0,1.0,`
`b_refine=1.0,1.0,`
`exp_refine=2.0,2.0,`
	Real arrays	Geometry-based strategy: size and shape of the refined region at each level.

`interpol_var=0`	Integer	Variables used to perform interpolation (prolongation) and averaging (restriction). **`interpol_type=0`**: conservatives (ρ, ρu, ρE) **`interpol_type=1`**: primitives (ρ, ρu, ρε)
`interpol_type=1`	Integer	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.

Cosmological simulations

In this section, we describe in more details how one can use RAMSES 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.

Parameter file and initial conditions

The first thing to do when performing cosmological simulations is to generate initial conditions as Gaussian random fields. The easiest way is to use the freely available **`grafic2`** code, developed by Edmund Bertschinger at MIT (see http://web.mit.edu/edbert) or its parallel version **`mpgrafic`** developed by Christophe Pichon and Simon prunet at IAP (see http://www.projet-horizon.fr/). These codes will generate initial conditions according to a given cosmological model and for a given box size. As outputs, 7 files will be generated, called **`ic_deltab`**, **`ic_velcx`**, **`ic_velcy`**, **`ic_velcz`**, **`ic_velbx`**, **`ic_velby`** and **`ic_velbz`**. The directory in which these files are stored should be enter in the Parameter File as parameter **`initfile(1)`** in namelist **`&INIT_PARAMS.`** Index 1 stands here for **`levelmin`**, and corresponds to the coarse grid. RAMSES will automatically read the cosmological parameters and the physical box length contained in these initial conditions files. The so-called "super-comoving" coordinate system is used in RAMSES for cosmological runs (see Martell & Shapiro 2003). If necessary, the translation from this scale-free coordinate system (**`boxlen=1.0`**) to the *cgs* system is performed using scaling factors stored in the output files. The units are set in routine **`units.f90`** in directory **`amr/`**.

The following namelist can be found in directory **`namelist/`** in the RAMSES package as file **`cosmo.nml`**. It is the Parameter File for a pure N body simulation, using 128^3 particles and a 128^3 coarse grid with 7 additional levels of refinement. To specify that initial conditions are to be read in grafic files, **`filetype='grafic'`** should be set in namelist **`&INIT_PARAMS`**.

```
&RUN_PARAMS
cosmo=.true.
pic=.true.
poisson=.true.
nrestart=0
nremap=10
nsubcycle=1,2
ncontrol=1
/

&OUTPUT_PARAMS
fbackup=10
noutput=10
aout=0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0
/

&INIT_PARAMS
filetype='grafic'
initfile(1)='/scratchdir/grafic_files'
/
```

```
&AMR_PARAMS
levelmin=7
levelmax=14
ngridtot=2000000
nparttot=3000000
nexpand=1
/

&REFINE_PARAMS
m_refine=7*8.
/
```

Parameters **`ngridtot`** and **`nparttot`** specify the maximum memory allocated for AMR grids and collisionless particles respectively. These numbers should be greater than or equal to the actual number of AMR grids and particles used during the course of the simulation.

`ngridtot` stands for the total number of AMR grids allocated over all MPI processes. Parameter **`ngridmax`** can be used equivalently, but stands for the local number of AMR grids within each MPI process. Obviously, one has **`ngridtot=ngridmax*ncpu`**.

Recall that, in RAMSES, we call "AMR grid" or "oct" a group of 2^{ndim} cells. If for some reason, during the course of the execution, the maximum allowed number of grids or particles have been reached, the simulation stops with the message:

```
 No more free memory
 Increase ngridmax
```

In this case, don't panic: just increase **`ngridmax`** in the Parameter File and resume the execution, starting from the last backup file.

Memory management

These two parameters control the memory allocated by RAMSES. It is important to know how much memory in Gb is actually allocated by RAMSES for a given choice of parameters. This can be approximated by:

- $0.7(\mathrm{ngridmax}/10^6)+0.7(\mathrm{npartmax}/10^7)$ Gbytes for pure N body runs,
- $1.4(\mathrm{ngridmax}/10^6)+0.7(\mathrm{npartmax}/10^7)$ Gb for N body and hydro runs,
- $1.0(\mathrm{ngridmax}/10^6)$ Gb for pure hydro runs.

Because of MPI communications overheads, the actual memory used can be slightly higher. Note that these numbers are valid for double precision arithmetic. For single precision runs, using the preprocessor directive **`–DNPRE=4`**, you can decrease these figures by 40%.

Restarting a run

As we just discussed, it is possible to resume a RAMSES run if the execution has stopped abnormally. For that, RAMSES uses "backup files", stored in directories called

```
backup_00001/
backup_00002/
```

```
backup_00003/
backup_00004/
```

Each directory contains all the necessary information for RAMSES to resume the execution. The frequency at which these backup files are created is specified by parameter **`fbackup`**, in units of coarse time steps. If you want to resume the execution starting from backup directory number 4, you need to specify the corresponding number in parameter **`nrestart=4`**. If you set **`nrestart=0`**, the run will start from the beginning as a completely new run.

When restarting a job, you can change almost all run parameters. There are however some notable exceptions. The number of output times can only be increased, and only new output times can be added after the old ones. The number of processors used with MPI cannot change. If you want to change the number of processes, you should start from the very beginning.

Parallel computing

We are now reading to address the complex issue of parallel computing with RAMSES. It is based on the MPI library through regular calls of MPI communication routines. In order to compile and link RAMSES with the MPI library, you need first to remove the preprocessor directive **`-DWITHOUTMPI`** from the compilation options in the Makefile. Don't forget to type **`make clean`** and then **`make`** again.

In order to launch a parallel RAMSES run, type for example

```
$ mpirun –np 4 bin/ramses3d namelist/sedov3d.nml
```

The two key parameters for parallel runs are **`nremap`** and **`ordering`**, both contained in the **`&RUN_PARAMS`** namelist block. The first one specifies the frequency at which the code will optimize load balancing by adjusting the domain decomposition, in units of coarse time step. Since it is a rather costly operation, this should not be performed to frequently. On the other hand, if the AMR grid evolves significantly with time, the computational and memory load might be very inhomogeneous across the processors. The optimal choice for parameter **`nremap`** is therefore application dependent. Bear in mind that if **`nremap>10`**, the associated overhead should be negligible.

The second important parameter to set up an efficient parallel computing strategy is **`ordering`**. This character chain specifies the type of domain decomposition you want to use. There are 3 possible choice in RAMSES currently implemented: **`'hilbert'`** (default value), **`'planar'`** and **`'angular'`**. Each cell in RAMSES is ordered with an integer index according to a given space ordering. One of the most famous ordering used in computer science is the Peano-Hilbert space-filling curve. This is a one-dimensional object filling up the three-dimensional space. An example of such domain decomposition is shown in Figure 2. This strategy is known to be the optimal choice if one considers the rather large ensemble of all possible AMR grids. In some cases, however, it is now longer an efficient strategy. The planar decomposition, for example, set up computational domains according to only one coordinate (the altitude z for example). Each processor receives a layer of cells whose thickness is automatically adjusted in order to optimize load balance. The angular decomposition follows the same strategy, except that now the coordinate is the polar angle around a given axis in the simulation box. These various ordering can be adapted easily to account for specific constraints. The user is encouraged to edit and modify the routine **`load_balance.f90`** in directory **`amr/`**.

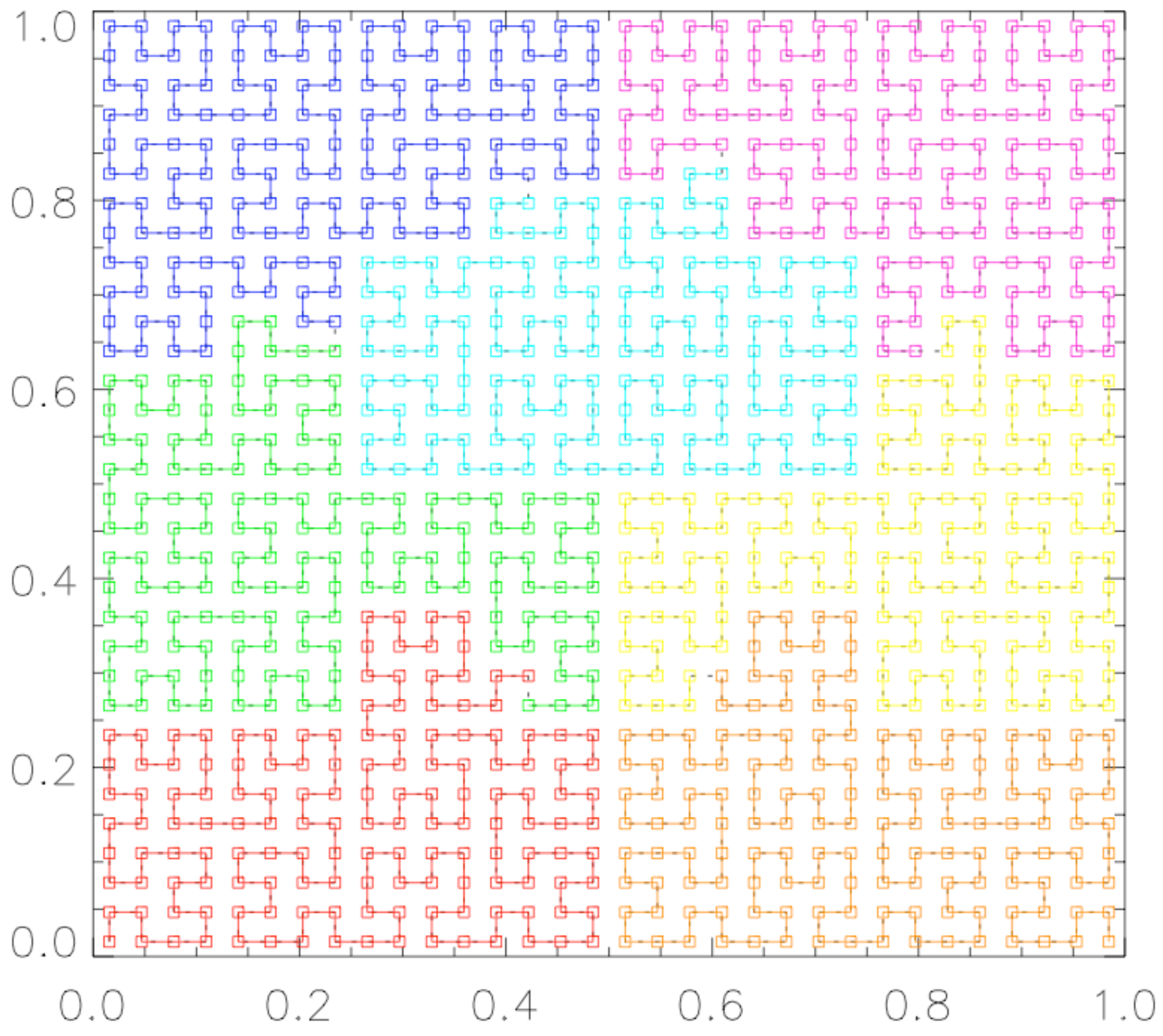

Figure 2: Domain decomposition of the unit square for a 32^2 grid over 7 processors using the Peano-Hilbert space-filling curve shown as the continuous line.

In case of parallel execution, RAMSES performs disk outputs in a very simple way: each processor creates its own file. Therefore, in directory **`output_00001/`**, one can find several files with a numbering corresponding to the processor number. One should bear this in mind when using the snapshots generated by RAMSES.

Post-processing utilities

Several post-processing codes are available in the current package in directory **`utils/`**. These are very simple pieces of software performing rather basic operation on RAMSES generated outputs. Users are encouraged to edit and modify these routines in order to design specific post-processing applications. We briefly describe them now.

- **`amr2map`**: this application reads RAMSES outputs and generates a projected map along one principal axis. The output is a binary Fortran image that can be read using any image-processing tool.

- **`amr2cube`**: this application reads RAMSES outputs and generates a 3D Cartesian cube with only one flow variable represented. The output is a binary Fortran file or a VTK file that can be read using any data visualization tool.

- **`amr2gmsh`**: this application reads RAMSES outputs and generates a gmsh format file with only one flow variable. **`gmsh`** is a freely available 3D visualization software that can be downloaded from http://www.geuz.org/gmsh/.

- **`part2map`**: this application reads RAMSES outputs for collisionless particles only and projected the particle distribution along one principal axis. The output is a binary Fortran image.

- **`log2col`**: this application reads a Log File that contains data from **`stdout`** and outputs an ASCII file with several columns corresponding to key control parameters (time step number, size, energy conservation and so on).

- **`sod`**: this application reads a RAMSES output for dark matter particles only and detect automatically over-dense dark matter halos in the density field.

- **`histo`**: this application reads a RAMSES output for gas only and performs a 2D histogram of all gas cells in the density-temperature plane.

Each one of these codes is a stand-alone application that can be compiled straightforwardly by typing directly for example:

```
$ f90 amr2map.f90 –o amr2map
```

Zoom simulations

Another interesting cosmological application for RAMSES is the so-called "zoom" technology or "re-simulation" technology. Let us consider the large-scale periodic box simulation we have presented in the previous section, performed with 128^3 particles by RAMSES with the **`grafic`** files stored in directory **`/scratchdir/grafic_files`**. After using the **`sod`** application, all dark matter halos in the final output have been identified. One halo is believed to be a good candidate for re-simulation. A quasi-spherical region must be determined, whose size and position are optimized in order to contain all the particles ending up in the final halo. A new set of initial conditions must then be generated using **`mpgrafic`**, providing the same large-scale modes than the previous run, but allowing now to simulate up to 1024^3 particles. A suite of applications is available in directory **`utils/`** to perform the extraction of a high-resolution region containing the halo particles only. These codes are called **`center_grafic`**,

extract_grafic and **degrade_grafic**. The idea is to center the simulation box on the high-resolution region, to extract a nested collection of rectangular grids around this position with various particle masses. The initial conditions data for each box are stored in a different directory. The original data centered on the region center can be called **boxlen100_n256**, where **100** stands for the box size in h^{-1} Mpc and **128** for the number of particles per box length. The nested box hierarchy we obtained using our various utilities is now:

```
boxlen100_n128/
boxlen50_n128/
boxlen25_n128/
boxlen12p5_n128/
```

Each of these directories should contain 7 grafic files. These names should be now inserted in the Parameter File, in the **&INIT_PARAMS** block, as

```
&INIT_PARAMS
filetype='grafic'
initfile(1)='/scratchdir/grafic_directories/boxlen100_n128'
initfile(2)='/scratchdir/grafic_directories/boxlen50_n128'
initfile(3)='/scratchdir/grafic_directories/boxlen25_n128'
initfile(4)='/scratchdir/grafic_directories/boxlen12p5_n128'
/
```

The re-simulation is now ready to go. This is our last words on cosmological simulations and how to run them using only Parameter Files as Runtime Parameters. We now describe how to use RAMSES with more advanced settings.

Advanced simulations

For truly innovative scientific applications, the user is usually forced to define complex initial conditions, to impose time varying boundary conditions or to use more than the 5 standard Euler variables (chemical species for example). We briefly describe the easiest way to do it in RAMSES.

Patching the code

The general philosophy to design advanced RAMSES applications is to "patch the code". What do we mean by that? A few key routines have been designed in RAMSES in a user's friendly fashion, in order for the user to edit the file, modify it according to it needs and recompile the code. For that, it is recommended to create your own directory, for example **`mypatch/`**, in which you will copy the various files you plan to modify. In the **`Makefile`**, you need to specify the complete path of this directory in the **`PATCH`** variable, as:

```
PATCH=/home/toto/mypatch
```

The **`make`** command will seek for source file in this directory first, compile and linked them if present. If not, it will use the default source files already available in the RAMSES package. Virtually any RAMSES source file can be modified to obtain a "patched" version of RAMSES that fulfill your needs. Usually, however, only a few routines need to be modified in order to perform advanced simulations. These routines will be described now in more details. The corresponding files are stored in various RAMSES subdirectories. These are: **`amr/units.f90`**, **`hydro/boundana.f90`**, **`hydro/condinit.f90`**, **`poisson/gravana.f90`**, **`poisson/rho_ana.f90`**, **`hydro/cooling_fine.f90`**. Of course, other routines of RAMSES can be modified at will, although potential changes might be more complicated to implement. A simple example of patch can be found in directory **`patch/`** of the package.

Physical units

This very simple routine can be found in directory **`amr/`** and is called **`units.f90`**. It is used to set the conversion factors from the user units into the cgs unit system. In this routine, the user must provide 3 scaling factors, namely **`scale_d`** for the density units in g/cc, **`scale_l`** for the length scale in cm and **`scale_t`** for the time scale in seconds. For self-gravity runs, since RAMSES assumes G=1 in the Poisson equation, it is mandatory to define the time scale as **`scale_t=1.0/sqrt(G*scale_d)`** with **`G=6.67d-8`**. These scaling factors are stored in the output files and can be used later on during post-processing.

Initial conditions

This routine can be found in directory **`hydro/`** and is called **`condinit.f90`**. It is self-documented. The calling sequence is just **`call condinit(x,u,dx,ncell)`**, where **`x`** is an input array of cell center positions, **`u`** is an output array containing the volume average of the fluid conservative variables, namely (ρ, ρu, ρv, ρw and E), in this exact order. If more variables are defined, using the **`–DNVAR`** directive, then the user should exploit this routine to define them too. **`dx`** is a single real value containing the cell size for all the cells and **`ncell`** is the number of cells. This routine can be used to set the initial conditions directly with Fortran instructions. Examples of such instructions can be found in directory **`patch/`**.

Another way to define initial conditions in RAMSES is by using input files. For the hydro solver, these files are always in the **`grafic`** format. We have already explained how to use the **`grafic`** format for cosmological runs. For non-cosmological runs, initial conditions can be defined using the exact same format, except than instead of 4 files (**`ic_deltab`**, **`ic_velbx`**, **`ic_velby`** and **`ic_velbz`**), one needs now 5 files called (**`ic_d`**, **`ic_u`**, **`ic_v`**, **`ic_w`** and **`ic_p`**) and containing the fluid primitive variables.

For collisionless particles, the **`grafic`** format is used only for cosmological runs, for which particles are initially placed at the cell centers of a Cartesian grid. For non-cosmological runs, the particles initial positions, velocities and masses are read in an ASCII file, in which each line corresponds to one particle, and should contain the following particle attributes: **`x,y,z,vx,vy,vz,m`**.

Boundary conditions

As explained in the previous sections, RAMSES can provide boundary conditions of different types: periodic (default mode), reflexive, outflow and imposed. This is performed in RAMSES using ghost regions, in which the fluid variables are set in order to obtain the required boundary. Ghost regions are defined in the namelist block &**`BOUNDARY_PARAMS`**. Each region is identified by its position, its type and eventually by the value of the fluid variables.

The exact order with which boundary regions are entered in the namelist block is very important. Let us consider the 4 boundary regions shown in Figure 3. They are defined by the following namelist block:

```
&BOUNDARY_PARAMS
nboundary=4
ibound_min=-1,+1,-1,-1
ibound_max=-1,+1,+1,+1
jbound_min= 0, 0,+1,-1
jbound_max= 0, 0,+1,-1
bound_type= 1, 1, 1, 1
/
```

The first region is located in the rectangle defined by coordinate (i=-1,j=0), while the third region is defined by coordinates (-1≤i≤+1, j=+1). The boundary type for all 4 regions is set to "reflexive" (**`bound_type=1`**). The fluid variables within the ghost region are therefore taken equal to the values of their symmetric cells, with respect to the boundary. This is why the order of the ghost regions is so important: regions 1 and 2 are updated first, using only the fluid variables within the computational domain. Regions 3 and 4 are updated afterwards, using the fluid variables within the computational domain, but also within regions 1 and 2. In this way, all cells within boundary regions are properly defined, especially in the 4 corners of the computational domain.

It is possible to define only 2 regions (say region 1 and 2 in Figure 3), the orthogonal direction will be considered as periodic. For gravity runs, the gravitational force is also updated in the ghost regions, following the same rules than for the velocity vector.

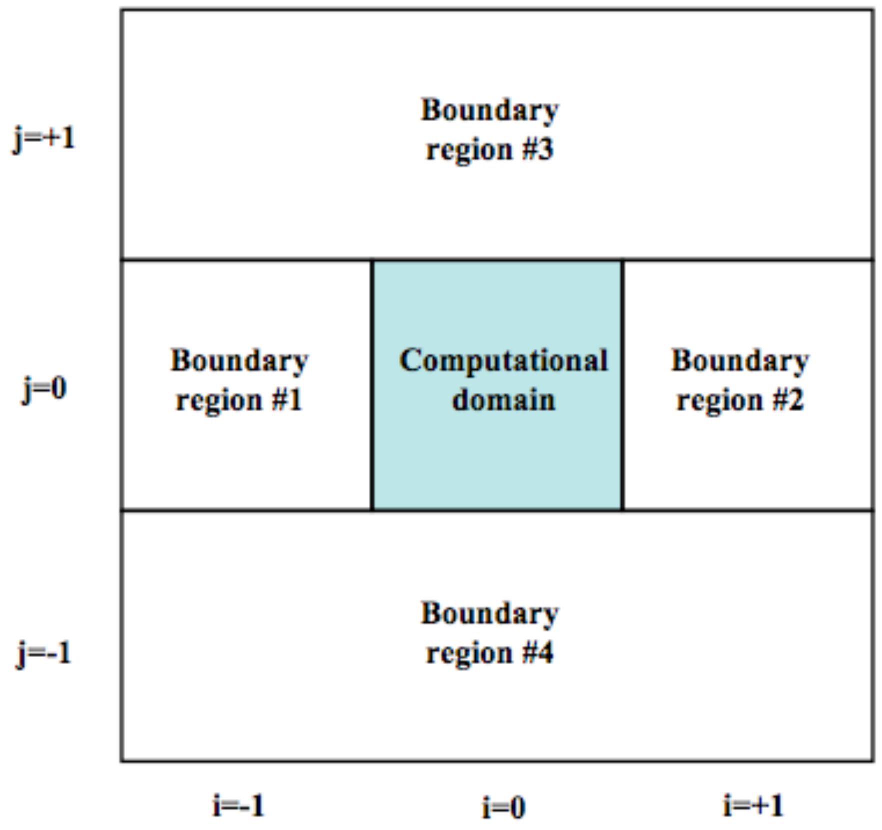

Figure 3: example of ghost regions used in RAMSES to impose specific boundary conditions.

For the Poisson equation, however, boundary conditions are either periodic, if no ghost regions are defined in the corresponding direction, or "Φ=0" Dirichlet boundary conditions within ghost regions. No other types of boundary conditions for the Poisson equation have been implemented (such as isolated, reflexive and so on).

If **`bound_type=3`**, boundary conditions must be imposed by the user. The first possibility is to use parameters **`d_bound`**, **`u_bound`**… to set a constant fluid state within the desired ghost region. For more advanced applications, the user is encouraged to patch the routine **`boundana.f90`** within directory **`hydro/`**. This routine is very similar to routine **`condinit.f90`**. The calling sequence is **`call boundana(x,u,dx,ibound,ncell)`**. The ghost region number is therefore provided, so the user can specify some region-dependent fluid conditions.

External gravity sources

If **`bound_type=3`**, boundary conditions must be imposed also for the gravitational force. This is performed by modifying routine **`gravana.f90`** in directory **`poisson/`**. If **`gravity_type>0`**, this routine is also used to specify the gravitational force within the computational domain. Note that in this case, the fluid is not self-gravitating anymore. there is another way to impose an external gravity source and in the same time, to solve for the Poisson equation. This is done using routine **`rho_ana.f90`** in directory **`poisson/`**. In this routine, again very similar to the previously presented Fortran routines, the user can specify the density profile for the external gravity source. This density profile will be added to the fluid density as the source term in the Poisson equation.

External thermal sources

The final routine that can be easily modified by the user is **`cooling_fine.f90`** in directory **`hydro/`**. This routine is used if **`cooling=.true.`** or if the polytropic temperature **`T2_star>0.0`**. In the first case, cooling and heating source terms are added to the energy equation and solved by a fully implicit integration scheme. In the second case, the thermal energy of the fluid is not allowed to be lower than a polytropic Equation-Of-State, for which one has $T/\mu = (T/\mu)_* (\rho/\rho_*)^{\gamma_* -1}$. All parameters denoted by index "*" can be set within the namelist block **`&PHYSICS_PARAMS`**. On the other hand, the user might want to modify routine **`cooling_fine.f90`** in order to implement more complex thermal modeling of the fluid.

Publication policy.

The RAMSES code is freely available for non-commercial use under the CeCILL License agreement. If a paper is published with simulations performed with RAMSES, the authors should cite the following paper, in which the RAMSES code was presented for the first time.

Teyssier, Romain, "Cosmological hydrodynamics with Adaptive Mesh Refinement: a new high-resolution code called RAMSES", Astronomy and Astrophysics, 2002, volume 385, page 337

If the same users need some basic help from the author on how to use RAMSES, or if the simulations performed have needed from the code's author some small adaptation of the code, a small sentence like "We thank Romain Teyssier for…" in the Acknowledgment section will do it.

If, on the other hand, the simulations performed requires the code's author to be more deeply involved in the project (new developments, new simulations from the author's side), then co-authorships of the paper is asked.