The Rockstar Halo Finder

Science/Documentation Paper: http://arxiv.org/abs/1110.4372

Contents

1 Compiling

If you use the GNU C compiler version 4.0 or above on a 64-bit machine, compiling should be as simple as typing "`make`" at the command prompt. If you need HDF5 support to read in snapshots, use "`make with_hdf5`".

Rockstar does not support compiling on 32-bit machines and has not been tested with other compilers. Additionally, Rockstar does not support non-Unix environments. (Mac OS X is fine; Windows is not).

2 Running

2.1 Quick start (single snapshot, single CPU)

Several example configuration files have been provided. If you have a small simulation file and you'd like to run Rockstar on a single processor to test its output, edit the file "`quickstart.cfg`" and change the file format to one of `ASCII`, `GADGET`, `AREPO`, `ART`, or `TIPSY` to match your simulation file. If you use the `ART` option, only PMss files are currently supported. (PMcrs files, which do not include particle IDs explicitly, are not supported).

If you use `ASCII`, `ART`, or `TIPSY`, you will additionally have to set the particle mass; if you use `ASCII` or `TIPSY`, you will also have to set parameters for the cosmology and box size.

If you use `GADGET`, you should check to make sure that the length and mass conversion multipliers are correct (`GADGET_LENGTH_CONVERSION` and `GADGET_MASS_CONVERSION`) to convert Gadget internal units to comoving Mpc/h and $M_\odot$/h.

Note that `GADGET` only works with the binary GADGET formats. For GADGET as well as AREPO HDF5 snapshots, use `AREPO` for the file format. You will then have to set `AREPO_LENGTH_CONVERSION` and `AREPO_MASS_CONVERSION` to convert internal snapshot units to comoving Mpc/h and $M_\odot$/h, respectively. Make sure to compile Rockstar with "`make with_hdf5`"; otherwise, the code will not accept these config options.

If you use `TIPSY`, you should set the length and velocity conversion multipliers (`TIPSY_LENGTH_CONVERSION` and `TIPSY_MASS_CONVERSION`) to convert Tipsy internal units to comoving Mpc/h and physical km/s. Note that Tipsy format is in a *beta stage of support*—please contact me if you have issues!

One final important variable to set is the force resolution of the simulation:

```
FORCE_RES = <force res. of sim., in Mpc/h; default 0.003>
```

Halos whose centers are closer than `FORCE_RES` are usually noise, and are subject to stricter removal tests than other halos.

Then, you can run Rockstar:

```
./rockstar -c quickstart.cfg <particle file>
```

Note that periodic boundary conditions are **not** assumed for a single-cpu run. (For that, see the next section, §2.2).

2.2 More Complete Setup (multiple snapshots/input files/CPUs)

Note: Rockstar does *not* use MPI for its parallelization. As such, there should be no headaches with finding a working MPI installation to use. :)

For running on multiple snapshots or simulations with multiple input files per snapshot, Rockstar uses a master process to direct the order of processing steps and several client processes to do the actual work (see Fig. 1).

For historical reasons, enabling Rockstar to run on multiple input files, multiple snapshots, or multiple CPUs is controlled by setting the following "Parallel IO" option in the config file:

```
PARALLEL_IO = 1
```

Besides setting the file type and conversion options in the previous section, several more options are necessary in the configuration file to specify filenames, paths, and information about the number of processes you want to use.

By default, the number of reading tasks is set to the number of files being read for each simulation snapshot:[1]

```
NUM_BLOCKS = <number of files per snapshot>
```

You will also have to provide information on where the data files are located. To do so, you will have to set:

```
INBASE = "/directory/where/files/are/located"
FILENAME = "my_sim.<snap>.<block>"
```

In this example, "`my_sim`" should be the base of your simulation filename. For multiple snapshots, the text "`<snap>`" will be automatically replaced by the snapshot number, and the text "`<block>`" will be automatically replaced by the block number (0 to `NUM_BLOCKS`-1). To specify the range of snapshot numbers, you may set

[1] If the number of blocks is more than the number of CPUs you wish to use, then you should also specify: `NUM_READERS = <number of CPUs>`

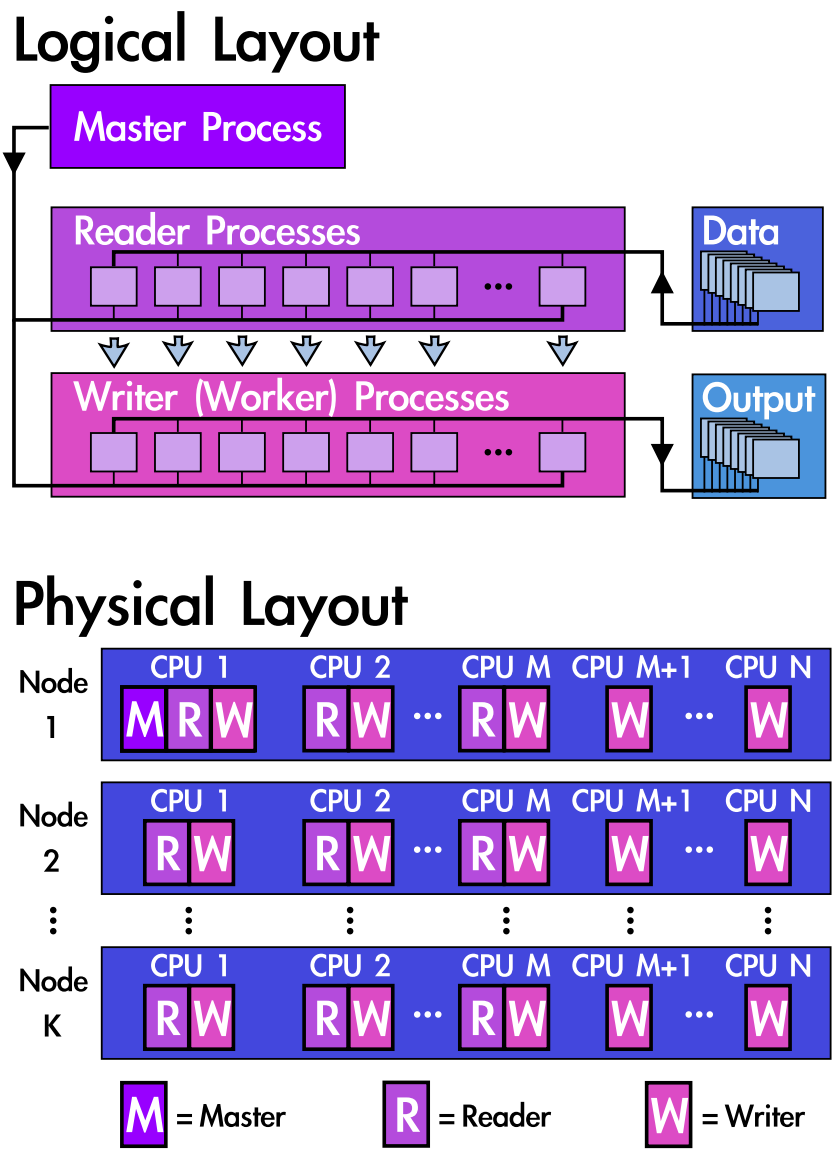

Figure 1: **Logical Layout of Parallel Processing**: One master process controls multiple reader and writer (worker) processes. The reader processes are responsible for loading data and rearranging it into rectangular volumes according to the number of writers. The writer processes receive data from the readers, analyze it for halos, and write output catalogs to one or more files. **Physical Layout**: Every CPU should contain one writer process, since this is where the bulk of the analysis time is spent. The master process consumes very little CPU power, and so can easily be accommodated by the first CPU. The reader processes also consume very little CPU power; however, they require enough memory to load in all the particle data. For this reason, they should be distributed evenly across all the nodes. Rockstar has several configuration parameters to help with launching the right number and the right type of tasks per node; see §2.2.

```
NUM_SNAPS = <total number of snapshots>
STARTING_SNAP = <first snap> #defaults to 0
```

If you have nonstandard names for your snapshots (e.g., "001" instead of "1"),[2] you may create a text file with one snapshot name per line and set:

```
SNAPSHOT_NAMES = </path/to/snapshot names>
```

This will automatically override the number of snapshots *and* the starting snapshot, if specified. You may do the same thing for the block names, too:

```
BLOCK_NAMES = </path/to/block names>
```

(Note that NUM_BLOCKS should still be specified in this case, so that the server knows how many reader connections to accept).

The number of writer (worker) tasks can be set to either 1 or a multiple of 8. If set to 1, periodic boundary conditions are *not* assumed; otherwise periodic boundary conditions *are* assumed (see also the PERIODIC option in the full list of config options). Usually, for best performance, you should set the number of writers to the total number of CPUs that you want to use for analysis:

```
NUM_WRITERS = <number of CPUs>
```

You can also optionally specify an output directory where Rockstar will write all of its data products:

```
OUTBASE = "/desired/output/path" # default is current directory
```

Starting Rockstar (under the default options) is then accomplished by first starting the server process:

```
/path/to/rockstar -c server.cfg
```

The server process will generate a file called "auto-rockstar.cfg" in the data output directory (OUTBASE) with contact information.

You will then have to start NUM_READERS (usually, NUM_BLOCKS) reader tasks and NUM_WRITERS writer tasks (refer to Fig. 1 for task CPU assignment). To simplify this process, there are two additional options you can set:

[2] Since GADGET always uses "001" instead of "1" for its snapshot numbers, Rockstar will automatically use the correct formatting without needing a snapshot filename list if the particle file type is GADGET or AREPO.

```
FORK_READERS_FROM_WRITERS = 1
FORK_PROCESSORS_PER_MACHINE = <number of processors per node>
```

The first option will automatically split off the reader tasks for you (which are idle most of the time) so that you only have to start the writer tasks. The second option will automatically split each writer task you start into many copies (according to the number of processors you specify to use per node). Thus, if these options are set properly, you would only have to start Rockstar running once on each compute node, instead of once per CPU. The command to start the Rockstar reading/writing processes on each compute node would be:

```
/path/to/rockstar -c OUTBASE/auto-rockstar.cfg
```

As noted above, `OUTBASE` is the current working directory by default; however, if you change it in the config file, the `auto-rockstar.cfg` file will be written to that location as well. Rockstar does not come with a command to start itself on many machines at the same time; however, the utilities which come with various MPI installations (e.g., `mpirun`/`mpiexec`/`ibrun`) are often suitable for doing so. See §2.12 for info about example startup scripts.

In terms of memory usage, Rockstar will use about 60 bytes / particle *maximum total* for a cosmological simulation. Thus, if you have a 1024^3 particle simulation and 2GB of memory available per processor, you should plan on using at least 32 CPUs in parallel.

If Rockstar is terminated for any reason, it is easy to restart it where it left off. Simply start the server process by running:

```
/path/to/rockstar -c OUTBASE/restart.cfg
```

Then, start client processes as you would normally. Rockstar will resume analysis from the last incomplete snapshot.

2.3 Inline Analysis for Simulations

For very large simulations, it is possible to have Rockstar run on particle snapshots as they become available. If writing the snapshots to disk is not an option, Rockstar can also accept particle input from pipes (see §3.2). It is strongly recommended that you contact the authors for assistance before running an important simulation this way.

2.4 Output

Rockstar can output in many different formats (see §2.8 below). If run in single-processor mode, halos are saved to a file called `halos_0.0.ascii` with one line per halo. If run in multi-processor mode, halos are saved to files called `out_0.list`, `out_1.list`, etc., with one file per particle snapshot. Many halo properties are currently calculated by default:

- Halo masses at several radii: $M_{\rm vir}$, M_{200b}, M_{200c}, M_{500c}, M_{2500c}. These masses always include any contributions from substructure. Also, masses with higher density thresholds (e.g., 2500c) can sometimes be zero if the density of the halo never rises above the threshold.[3] By default, only bound particles are included; see §2.11.1 if this is not what you want.
- Halo maximum circular velocity and velocity dispersion.
- Halo radii: $R_{\rm vir}$ and the scale radius r_s, calculated both using profile fitting and using the Klypin $v_{\rm max}$ method.
- Halo center positions and velocities.
- Halo spin (both Bullock and Peebles) and angular momentum.
- Halo shapes and principal axes, using the Allgood method (iterative, weighted by $1/r^2$), at both $R_{\rm vir}$ and $R_{\rm 500c}$. See §2.11.1 for how to change the shape calculation method.
- The ratio of halo kinetic to potential energy, and the center position and velocity offsets from the halo's bulk average position and velocity.

Information about each of the columns in both file types (including units) is available in the ASCII headers. If you would like to calculate more halo properties, please see §3.4.

[3] Particles are assumed to have an effective radius of `FORCE_RES` for the purposes of density calculations near the halo center.

2.5 Merger Trees

By default, the output files for Rockstar include some basic descendant information for each halo. However, it is strongly recommended that you use a more sophisticated package, such as Consistent Trees

http://code.google.com/p/consistent-trees

to improve consistency between timesteps before using the descendant information for science purposes (e.g., calculating merger rates). Rockstar includes a script to autogenerate configuration files for Consistent Trees (see the Consistent Trees README file for use). It is *very* important to use the same version of this script as the Rockstar version that you used for analysis.

2.6 Host / Subhalo Relationships

By default, the output files for Rockstar do not include information about which halos are hosts (as opposed to subhalos). This is because the consistent tree code (see §2.5, above) automatically calculates such relationships. However, if you do not intend to use the merger tree code, then there is a utility included with Rockstar to postprocess the output halo catalogs (i.e., the `out_*.list` files). To compile, run "`make parents`" from the Rockstar source directory. Then, run the following command

```
/path/to/rockstar/util/find_parents <box_size> out_XYZ.list
```

on each halo catalog for which you want to find host halos. Information about which halos are hosts and which are subs will be output as an extra column, the parent ID. Host halos will have a parent ID of -1; subhalos will have a parent ID which corresponds to their host halo ID. Note that the files produced will *not* be suitable for input into the merger tree code above.

2.7 Lightcones

Lightcones are currently only supported with parallel IO (see options above). To turn on lightcone support, set

```
LIGHTCONE = 1
```

The scale factor will be automatically computed from the distance to the origin, which is set by

```
LIGHTCONE_ORIGIN = (x,y,z) #default: (0,0,0)
```

If you want a lightcone created by joining two different fields of view from the same box, you will have to set the following options:

```
LIGHTCONE_ALT_SNAPS = /path/to/second_lightcone_snapnames
LIGHTCONE_ALT_ORIGIN = (x,y,z) #default: (0,0,0)
```

(The LIGHTCONE_ALT_ORIGIN option translates all particles in the second field of view from (x,y,z) to the first LIGHTCONE_ORIGIN.) If using two fields of view, it is assumed that they have equal numbers of input files; NUM_BLOCKS should be set to the combined total number of input files.

If you have more than two fields of view from the same box, you will have to make sure that their positions are already translated appropriately so that they can be joined/unioned together without any overlap issues. In this case, you should not set any of the ALT lightcone options. Additionally, you should set "IGNORE_PARTICLE_IDS = 1" so as to prevent confusion with the same particle ID appearing multiple times.

2.8 Controlling Output Formats

Rockstar can currently output in either ASCII or binary formats. You can control the output directory for halo catalogs by setting:

```
OUTBASE = "/desired/output/path" # default is current directory
```

By default, both binary and ASCII catalogs are printed. This is set by

```
OUTPUT_FORMAT = BOTH # or "ASCII" or "BINARY"
```

However, as the binary outputs are required for generating merger trees, merger trees will *not* be generated if you select the ASCII option. If you want merger trees but not the binary outputs (which take up lots of space), you can set the following option:

```
DELETE_BINARY_OUTPUT_AFTER_FINISHED = 1
```

For the binary outputs, there is a 256-byte header (detailed in io/meta_io.h) followed by a binary dump of the halo structures (see halo.h), followed by a binary dump of the particle ids in each halo (type int64_t). See the load_binary_halos() routine in io/meta_io.c for an example of how to read in the binary files.

To change the minimum particle size of output halos, set

```
MIN_HALO_OUTPUT_SIZE = <minimum number of particles> #default: 20
```

Note that this option does *not* specify the minimum number of particles bound within the virial or other halo radius. Instead, it specifies the minimum number of particles which Rockstar identifies as uniquely assigned to that halo. The virial mass can (and usually will) be much less.

To avoid printing spurious halos, Rockstar excludes significantly unbound objects by default. To change the threshold of what is considered "significantly unbound"—e.g., for studying tidal remnants, you may consider lowering the value of the following parameter:

```
UNBOUND_THRESHOLD = <minimum fraction of bound mass> #default: 0.5
```

Generally, this affects the halo mass function at the few percent level for halos of 100 bound particles or less (see the Rockstar science paper for details).

Rockstar does not by default output full particle information to save space. In order to turn on this capability, set

```
FULL_PARTICLE_CHUNKS = <number of writers which will output particles>
```

to the desired number of writer processes which you want to output particles (i.e., something between 1 and `NUM_WRITERS`) at every timestep.

Alternately, you can choose to use the BGC2 binary output format, which records full particle information as well as some halo information for an entire snapshot. In order to turn on this capability, set

```
BGC2_SNAPNAMES = </path/to/snapshot_names>
```

Where the file contains a list of snapshots (formatted the same way as in `SNAPSHOT_NAMES`, if you used that option) for which you want BGC2 output. Note that the BGC2 format is currently incompatible with processing lightcones.

Once the BGC2 files have been generated, a short postprocessing step is necessary to bring them into full compliance with the BGC2 format specification (calculating subs/centrals, etc.). You will need to compile the postprocessing code with

```
make bgc2
```

Then, for every BGC2 snapshot you have generated, you will need to run

```
/path/to/rockstar/util/finish_bgc2 -c rockstar.cfg -s <snap>
```

Note that particle coordinates in BGC2 files are automatically wrapped around box edges if periodic boundary conditions are enabled; this way, they always form contiguous regions.

For additional useful source code to load in both the BGC2 and full-particle halo/particle formats, check the `examples/` subdirectory.

2.9 Comparing to Published Mass Functions

By default Rockstar calculates halo and subhalo masses using particles from the surrounding friends-of-friends group with unbound particles removed. This leads to halo masses which are very consistent for merger trees, but which are a few percent below published spherical overdensity (SO) calibrations. For comparing to calibrated mass functions, the simplest approach is to generate BGC2 files (see §2.8, above). If you don't have software to read BGC2 files, you can use the BGC2 to ASCII converter supplied with Rockstar:

```
/path/to/rockstar/util/bgc2_to_ascii -c rockstar.cfg -s <snap>
```

You should then calculate mass functions from the resulting output; subhalos can be excluded by skipping halos which do not have `PID=-1`.

2.10 Infiniband / Network Connectivity Notes

As particle transfer is usually a small fraction of the total analysis time, Rockstar performs well on pretty much any gigabit or faster network.

If you have IP over Infiniband, Rockstar can use this natively. If the primary server IP address is not for an Infiniband network, you may force Rockstar to use the Infiniband interface (e.g., "`ib0`") with the following config option:

```
PARALLEL_IO_SERVER_INTERFACE = "ib0"
```

Rockstar supports IPv4 and IPv6 natively; addresses may be specified in either format.

2.11 Full Configuration Options

2.11.1 Commonly-Used Options

For those options not mentioned directly above:

```
MASS_DEFINITION = <"vir" or "XXXc" or "XXXb" etc.> #default: vir
```

This lets you specify how you want masses calculated. "vir" uses the formula from Bryan & Norman (1998); a number plus "c" or "b" calculates masses relative to the critical or background density, respectively. (E.g. "200b" or "200c", or even fractional values like "100.5c"). Changing the main mass definition will also change the halo radius at which most other properties are calculated (e.g., $v_{\rm max}$, spin, shape, etc.).

Besides the main mass definition, you can specify four other halo mass definitions to calculate, in MASS_DEFINITION2 through MASS_DEFINITION5. The syntax for these is identical to that for MASS_DEFINITION; the defaults are listed in §2.4. Note that for very low density thresholds (e.g., < 150b), you may have to increase the value of FOF_LINKING_LENGTH below to obtain accurate masses.

By default, all masses are calculated from FOF groups after unbound particles are removed. This is appropriate for merger trees, where halos/subhalos need to be treated identically for consistency across timesteps. However, if you are calculating *mass functions*, you will want to use spherical overdensities including unbound particles and particles which may exist outside of the FOF group for the halo. To enable these strict SO masses, use

```
STRICT_SO_MASSES = 1 #default: 0
```

The resulting masses will be present in the auxilliary mass columns (columns 20-24) of the out_*.list files. This option is only available in parallel processing mode (§2.2). If computing mass functions for snapshots widely separated in redshift, you may also wish to disable temporal halo finding (i.e., using information from previous snapshots to determine host/sub relationships):

```
TEMPORAL_HALO_FINDING = 0 #default: 1
```

Shape calculations are by default performed with the Allgood method. If you wish to change this, you can specify

```
WEIGHTED_SHAPES = 0 #default: 1
```

which will calculate the mass tensor without weighting by $1/r^2$. In addition, if you do not want the shapes to be calculated iteratively, specify

```
SHAPE_ITERATIONS = 1 #default: 10
```

This setting specifies the maximum number of shape calculation iterations to perform.

Options to set the base cosmology are available:

```
SCALE_NOW = <current cosmological scale factor>
h0 = <hubble constant today> # in units of 100 km/s/Mpc
Ol = <Omega_Lambda> # in units of the critical density
Om = <Omega_Matter> # in units of the critical density
```

These cosmology options are only relevant if one is reading from an ASCII or TIPSY particle file. (For TIPSY, the `SCALE_NOW` parameter may be omitted). For other data formats, these values are `automatically overwritten` with values in the particle data files.

Non-LCDM equations of state are also available. Since none of the particle input formats include a standard way of specifying these, the equation of state must be specified in the config file:

```
W0 = <W_0> # dark energy equation of state at z=0
WA = <W_A> # scaling of DE equation of state with scale factor.
```

By default, these are set to `W0=-1` and `WA=0`.

It is not necessary to specify the particle mass for GADGET2 files, but it is necessary for all other file types:

```
PARTICLE_MASS = <mass of each particle> #in Msun/h
```

However, for GADGET2 files, you will need to specify the conversion to Rockstar's internal length (comoving Mpc/h) and mass ($M_\odot$/h) units:

```
GADGET_MASS_CONVERSION = <conversion from GADGET units to Msun/h>
GADGET_LENGTH_CONVERSION = <conversion from GADGET units to Mpc/h>
```

Usually, these will be `1e10` (mass conversion) and either `1` or `1e-3` (length conversion for Mpc/h and kpc/h, respectively).

Similarly, for TIPSY files, you will need to convert lengths and velocities to comoving Mpc/h and physical km/s:

```
TIPSY_LENGTH_CONVERSION = <conversion from TIPSY units to Mpc/h>
TIPSY_VELOCITY_CONVERSION = <conversion from TIPSY units to km/s>
```

To disable periodic boundary conditions (only applicable for `PARALLEL_IO`), you can set:

```
PERIODIC = 0
```

Note that setting this to 1 does *not* enable periodic boundary conditions for single-processor halo finding.

For ASCII particle data, it is necessary to specify the box size:

```
BOX_SIZE = <side length of cosmological box in comoving Mpc/h>
```

To run postprocessing scripts after each snapshot finishes, you can use the RUN_ON_SUCCESS config option:

```
RUN_ON_SUCCESS = "/path/to/myscript"
```

The script will be executed with two arguments: the snapshot number followed by the snapshot name (only different if specified via the SNAPSHOT_NAMES file). If parallel processing is enabled, it will run in the background on the first compute node only.

To run postprocessing scripts in parallel (i.e., one per writer task), you can use

```
RUN_PARALLEL_ON_SUCCESS = "/path/to/myscript"
```

The script will be executed by each writer task, and further analysis will pause until all the scripts finish. The script will be given five arguments, which are the current snapshot number, the total number of snapshots, the writer task number (0 to NUM_WRITERS-1), the total number of writers, and the path to the Rockstar configuration file. Spaces in the pathname must be escaped, because the script is run via the shell instead of executed directly.

To restart Rockstar from a specific snapshot, use

```
RESTART_SNAP = <snapshot number> #default: 0
```

This will resume analysis with the assumption that all snapshots previous to RESTART_SNAPSHOT have been analyzed.

2.11.2 Rarely-used Options

```
GADGET_SKIP_NON_HALO_PARTICLES = <0 or 1> #default = 1
```

By default, Rockstar only considers dark matter particles; the preceding option can be set to 0 to force consideration of other particles as well in GADGET2 files. The default halo particle type in GADGET is 1; however, if you need to change this, you can use the following option:

```
GADGET_HALO_PARTICLE_TYPE = <0 to 5> #default = 1
```

Note that Rockstar has no current support for multiple particle masses. A beta version of Rockstar with support for multi-mass particles is available on request from the authors.

```
RESCALE_PARTICLE_MASSES = <0 or 1> #default 1
```

If only dark matter particles are used from GADGET2 files in a simulation which also includes gas particles, it is necessary to rescale the particle masses so as to preserve the correct matter density; setting this option tells Rockstar to do so.

If for some reason your simulation data has inconsistent or duplicate particle IDs, you can set the following option to prevent problems with halo finding:

```
IGNORE_PARTICLE_IDS = 1
```

Note that in this case, merger trees are disabled.

As mentioned above, Rockstar requires one master server accessible by all reader and writer tasks. The server does not require access to any data files, but it must be of the same architecture as the reader/writer tasks. In order to contact the master server, the reader/writer tasks need to know its address, specified by the following two options in the config file:

```
PARALLEL_IO_SERVER_ADDRESS = <server address> #default: auto
PARALLEL_IO_SERVER_PORT = <server port> #default: auto
```

The server port should be in the range 1025-65000, and it should be unblocked by the server firewall. Alternately, you may specify "`auto`" for both the server address and the server port. In this case, once you launch the server process, it will write to a file called "`auto-rockstar.cfg`" in the launch directory (or the directory that `OUTBASE` is set to). This file will contain updated settings with the machine name and port for the server, which you can then use to launch the reader/writer tasks. To force Rockstar to use a specified interface (useful for machines on multiple networks), you can use:

```
PARALLEL_IO_SERVER_INTERFACE = <interface name> #ib0, for example
```

Since analysis tasks have to acquire particle data from reading tasks, the analysis tasks need to be contactable by the reading tasks. You should choose a port number for the analysis tasks which is distinct from the server port above:

```
PARALLEL_IO_WRITER_PORT = <port from 1025-65000>
```

If you have a firewall, you will have to open this port for access *as well as* the N ports above this port, where N is the number of writer tasks you are running per compute node.

The following options determine details of the halo finding. To tell Rockstar to calculate halo radii as well as Vmax, Rvmax, and other halo properties from the *unbound* particles, set the following option to 0:

```
BOUND_PROPS = <0 or 1> #default 1
```

This is likely not what you want if you are calculating mass functions; you should use `STRICT_SO_MASSES` instead (§2.11.1).

To change the default FOF refinement fraction (see the Rockstar paper), change:

```
FOF_FRACTION = <FOF refinement fraction> #default 0.7
```

To change the default 3D FOF linking length, set:

```
FOF_LINKING_LENGTH = <FOF linking length> #default 0.28
```

This is generally only necessary to increase if trying to find halos with very low spherical overdensity thresholds.

To change the minimum (internal) number of particles considered to be a halo seed, change:

```
MIN_HALO_PARTICLES = <minimum number of halo particles> #default: 10
```

See also `MIN_HALO_OUTPUT_SIZE` in §2.8.

To specify your own load-balancing distribution, you can use the `LOAD_BALANCE_SCRIPT` option. There is an example of such a script in the scripts directory; to use, set the `LOAD_BALANCE_SCRIPT` option to

```
"perl /path/to/rockstar/scripts/sample_loadbalance.pl"
```

The data given to the load-balancing script are the number of writer tasks, the recommended dimensional divisions, the box size, the scale factor, as well as the IP addresses and ports of all the writer tasks. The script is expected to reoutput the information for each writer task along with an additional set of six numbers specifying the boundary region (min_x, min_y, min_z) to (max_x, max_y, max_z). The boundary region must be contained within the box size in each dimension; the boundary regions are also assumed not to be overlapping (it is up to your script to check this).

2.12 Full Example Scripts

For full example configuration files, used for running on the Bolshoi and Consuelo simulations, see `scripts/pleiades.cfg` or `scripts/ranger.cfg`.

For the example PBS submission script used to run the halo finding on Pleiades, see `scripts/pleiades.pbs`; for Ranger, see `scripts/ranger.pbs`.

3 Extending Rockstar

Rockstar is written in C, and requires no external libraries by default. Please see the `SOURCE_LAYOUT` file for a brief description of all source code files.

3.1 Adding More Configuration Parameters

Config parameters are defined in `config.template.h`. The syntax is

```
<type>(VARIABLE_NAME, DEFAULT_VALUE)
```

where <type> is one of "`string`" (`char *`), "`integer`" (64-bit signed integer, i.e., `int64_t`), "`real`" (`double`), or "`real3`" (`double[3]`).

Config parameters added here will be globally visible in the code and automatically interpreted from the config file. The default value will be assigned if the config file does not include the variable.

3.2 Adding More Input Formats

If you run Rockstar through YT (http://yt-project.org/), you may find that your desired input format is already supported. Alternatively, if you have existing code to read in particles and output in binary GADGET or

ART format (not including HDF5), you can use named pipes to directly communicate the data to Rockstar. This technique is also useful for analyzing simulation data as the simulation is running. You can use the script in `scripts/mkpipes.pl` to make named pipes; you should ask it to make as many named pipes as you have files per snapshot. Then, you would set `FILENAME = /path/to/pipe.<block>`. To get data into the pipes, you would simply open and print to them like normal files (or use output redirection from `stdout`). For example, you could have one script per pipe, which would be responsible for writing one block of particles to the pipe from each snapshot. Since write operations to named pipes will pause execution of the writing program if the pipe has reached its buffer capacity (typically 4KB), the scripts will automatically pause until Rockstar has finished processing an entire snapshot and is ready to read in new data.

For highest efficiency, it is appropriate to copy one of the existing `io/io_*.c` routines and modify it to accept your input format directly. You will have to define a routine that looks like:

```
void load_particles_mytype(char *filename, struct particle **p,
      int64_t *num_p)
```

This routine should add the number of particles in the file to `num_p` and grow the particle structure appropriately, e.g.,

```
check_realloc_s(p[0], sizeof(struct particle), num_p[0] + num_new_p);
```

Note that `check_realloc_s()` is defined in `check_syscalls.h`. It is generally a good idea to use the IO routines defined in that header, as they will automatically halt the program if they encounter any errors reading, writing, or opening files, making debugging much easier. Routines helpful for reading Fortran-unformatted files and files encoded in a different endianness are available from `io/io_util.h`.

The particle structure is very simple; it is described in `particle.h`. Your routine should convert everything to Rockstar's internal units: i.e., comoving Mpc/h for positions and non-comoving km/s for peculiar velocities. Your routine should also set as many config variables as possible (e.g., `BOX_SIZE`, `PARTICLE_MASS`, cosmology, current scale factor, etc.) from the data header, as that will reduce the chance for errors in the analysis from incorrectly-supplied config variables.

Once you have created a new `io/io_mytype.c` file, you should create a corresponding header file and include it in `io/meta_io.c`. You can then add

a new condition in the `read_particles()` routine in `io/meta_io.c` to call your code when the `FILE_FORMAT` variable matches your filetype. Finally, you should add the new `io/io_mytype.c` file to the `Makefile` so that it gets compiled along with the other Rockstar source code.

If you think your code may be useful for others, please consider submitting it as a patch to the authors. Others will no doubt appreciate it!

3.3 Adding More Output Formats

Halos are output in the `output_halos()` routine in `io/meta_io.c`. Halos are stored in the `halos` array, for which the structure is defined in `halo.h`. Particles are stored in the `p` array, for which the structure is defined in `particle.h`. It is best to use the `output_binary()` routine from `io/io_internal.c` as a starting template for outputting halo and particle data.

3.4 Adding More Halo Properties

All halo properties are calculated in `properties.c`, in the routine `_calc_additional_halo_props()`. All particles associated with the current halo being analyzed are in the `po` array, which is of type `struct potential` (defined in `potential.h`); particles in the array are sorted by distance from the halo center. You may either add calculations directly in the main particle loop in `_calc_additional_halo_props()`, or you may decide to create your own particle loop as a separate routine. If the latter case, don't forget to honor the option to skip over unbound particles unless your calculation always needs to include them; otherwise, you may get strange results for subhalos.

You may store your calculated values by extending the halo structure, defined in `halo.h`. To print out the new properties, you should modify `gen_merger_catalog()` in `io/meta_io.c`. As with new input formats, if you think your code may be useful for others, please consider submitting it as a patch to the authors.