GADGET-4

Volker Springel

Max-Planck-Institute for Astrophysics

Karl-Schwarzschild-Str. 1, 85748 Garching, Germany

Table of Contents

Parameterfile

Snapshot Format

Diagnostics Output

Group Finders

- Groups and subhalos
- FOF and SUBFIND
- Format of group catalogues
    - FOF catalogue
    - SUBFIND catalogue

Special Features

- Special code features
- Initial conditions
- Merger trees
- Lightcone output
- Power spectra
- I/O bandwidth test

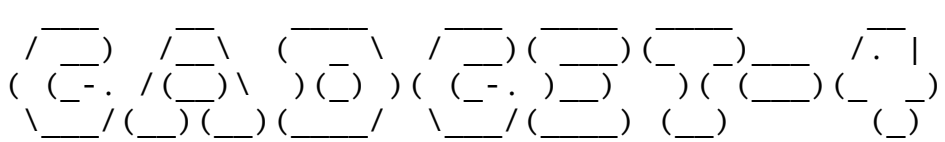
Examples

- Example Setups
- Cosmological DM-only simulation with IC creation
- Aquarius Milky-Way Zoom
- Colliding galaxies with star formation
- Santa Barbara cluster
- Old examples from GADGET-2
    - Galaxy collision
    - Adiabatic collapse of a gas sphere
    - Cosmological formation of a cluster of galaxies
    - Large-scale structure formation including gas

Changing the Code

- Guide to code changes
- Coding style guide
    - Code extensions
    - General code-style principles
- Adding a config-option
- Adding parameters
- Adding a source file
- Migration to GADGET4
    - Main changes in the code
    - Migrating old set-ups to GADGET-4
- Notes on memory consumption of GADGET-4
- Notes on various limits in GADGET-4

Introduction to GADGET-4



GADGET-4 is a massively parallel code for N-body/hydrodynamical cosmological simulations. It is a flexible code that can be applied to a variety of different types of simulations, offering a number of sophisticated simulation algorithms. An account of the numerical algorithms employed by the code is given in the original code paper, subsequent publications, and this documentation.

GADGET-4 was written mainly by Volker Springel, with important contributions and suggestions being made by numerous people, including Ruediger Pakmor, Oliver Zier, and Martin Reinecke.

Except in very simple test problems, one or more specialized code options will typically be used. These are broadly termed modules and are activated through compile-time flags. Each module may have several optional compile-time options and/or parameter file values.

Overview and history

In its current implementation, the simulation code GADGET-4 (**GA** laxies with **D** ark matter and **G** as int **E** rac **T** - this peculiar acronym hints at the code's origin as a tool for studying galaxy collisions) supports collisionless simulations and smoothed particle hydrodynamics on massively parallel computers. All communication between concurrent execution processes is done either explicitly by means of the message passing interface (MPI), or implicitly through shared-memory accesses on processes on multi-core nodes. The code is mostly written in ISO C++ (assuming the C++11 standard), and should run on all parallel platforms that support at least MPI-3. So far, the compatibility of the code with current Linux/UNIX-based platforms has been confirmed on a large number of systems.

The code can be used for plain Newtonian dynamics, or for cosmological integrations in arbitrary cosmologies, both with or without periodic boundary conditions. Stretched periodic boxes, and special cases such as simulations with two periodic dimensions and one non-periodic dimension are supported as well. The modeling of hydrodynamics is optional. The code is adaptive both in space and in time, and its Lagrangian character makes it particularly suitable for simulations of cosmic structure formation. Several post-processing options such as group- and substructure finding, or power spectrum estimation are built in and can be carried out on the fly or applied to existing snapshots. Through a built-in cosmological initial conditions generator, it is also particularly easy to carry out cosmological simulations. In addition, merger trees can be determined directly by the code.

The main reference for numerical and algorithmic aspects of the code is the paper "Simulating cosmic structure formation with the GADGET-4 code" (Springel et al., 2020, MNRAS, submitted), and references therein. Further information on the previous public versions of GADGET can be found in "The cosmological simulation code GADGET-2" (Springel, 2005, MNRAS, 364, 1105), and in "GADGET: A code for collisionless and gas-dynamical cosmological simulations" (Springel, Yoshida & White, 2001, New Astronomy, 6, 51). It is recommended to read these papers before attempting to use the code. This documentation provides additional technical information about the code, hopefully in a sufficiently self-contained fashion to allow anyone interested to learn using the code for cosmological N-body/SPH simulations on parallel machines.

Most core algorithms in GADGET-4 have been written by Volker Springel and constitute evolved and improved versions of earlier implementations in GADGET-2 and GADGET-3. Substantial contributions to the code have also been made by all the authors of the GADGET-4 code paper. Note that the code is made publicly available under the GNU general public license. This implies that you may freely copy, distribute, or modify the sources, but the copyright for the original code remains with the authors. If you find the code useful for your scientific work, we kindly ask you

to include a reference to the code paper on GADGET-4 in all studies that use simulations carried out with the code.

Disclaimer

It is important to note that the performance and accuracy of the code is a sensitive function of some of the code parameters. We also stress that GADGET-4 comes without any warranty, and without any guarantee that it produces correct results. If in doubt about something, reading (and potentially improving) the source code is always the best strategy to understand what is going on!

Please also note the following:

The numerical parameter values used in the examples contained in the code distribution do not represent a specific recommendation by the authors! In particular, we do not endorse these parameter settings in any way as standard values, nor do we claim that they will provide fully converged results for the example problems, or for other initial conditions. We think that it is extremely difficult to make general statements about what accuracy is sufficient for certain scientific goals, especially when one desires to achieve it with the smallest possible computational effort. For this reason we refrain from making such recommendations. We encourage every simulator to find out for herself/himself what integration settings are needed to achieve sufficient accuracy for the system under study. We strongly recommend to make convergence and resolution studies to establish the range of validity and the uncertainty of any numerical result obtained with GADGET-4.

Compilation and basic usage

If you are already familiar with older versions of GADGET or with AREPO, you will feel quickly at home in compiling and working with GADGET-4, since the code follows very similar usage concepts. However, GADGET-4 is now a C++ code, and has a more elaborate build concept that in particular attempts to prevent basic forms of code rot. This makes the build process slightly more advanced, as described below.

Compilation requirements

GADGET-4 needs the following non-standard libraries for compilation:

1. **mpi**: The `Message Passing Interface' (version 3.0 or higher). Many vendor supplied versions exist, in addition to excellent open source implementations, e.g. MPICH https://www.mpich.org, or OpenMPI http://www.open-mpi.org.

2. **gsl**: The GNU scientific library. This open-source package can be obtained at http://www.gnu.org/software/gsl. GADGET-4 only needs this library for a few very simple cosmological integrations at start-up.

3. **fftw3**: The `Fastest Fourier Transform in the West'. This open-source package can be obtained at http://www.fftw.org. Note that the MPI-capable version of FFTW-3 is not explicitly required (it makes no difference whether it is available or not as GADGET-4 implements its own communication routines when MPI is used). FFTW is only needed for simulations that use the TreePM algorithm, or if power spectra are estimated, or cosmological ICs are created.

4. **hdf5**: The `Hierarchical Data Format' (available at http://hdf.ncsa.uiuc.edu/HDF5. This library is needed when one wants to read or write snapshot files in HDF5 format. It is highly recommended to use HDF5 instead of the plain binary output format that can be used as an alternative and still available for historical reasons.

5. **hwloc** : The `hardware locality library' is useful for allowing the code to probe the processor topology it is running on and enable a pinning to individual cores. This library is optional and only required if the IMPOSE_PINNING flag is set. Note that many MPI libraries nowadays in any cases enable pinning by default.

6. **vectorclass** : This is a C++ library that the code utilizes when explicit vectorization via the AVX instruction set is enabled in the SPH compute kernels. This is then implemented with the vectorclass library by Agner Fog. For simplicity, the source code of this library (which takes the form of C++ header files) is included in the GADGET-4 distribution in directory src/vectorclass. Note that potential updates and bug fixed in this library require a corresponding update of this subdirectory.

Compilation of GADGET-4 needs a working C++ compiler, supporting at least the C++11 standard. For GNU gcc, this means version 4.x or later. The code also makes use of GNU-Make and Python as part of its build process. Those should hence be available as well.

Building the code

After obtaining GADGET-4 from the code repository, you will find a bunch of subdirectories as well as few further files for the build system in the top-level directory. The most important subdirectory is `src/` , which contains the actual source code files, distributed into subdirectories according to the functionality of each file.

The GADGET-4 code is controlled and configured by two different files, one containing compile-time options, and one listing runtime parameters. The default name for the file with compile-time options is `Config.sh` , and the meaning of the different options specified there is explained in a special section of this manual. The run-time options of GADGET-4 are controlled by a parameter file, and are described also in a separate section of this documentation.

One possible way to create the file `Config.sh` needed for compilation is to make a copy of `Template-Config.sh` and then modify it as needed by commenting in or commenting out the desired options. Given the length of this template file, better overview is provided by assembling only the enabled options in a file (which can also be gleaned from a previous GADGET-4 run). Another option is to use one of the Config.sh files that come with the example problems, and modify it if needed. The code uses the GNU make utility for controlling the build process, which is specified by the file `Makefile`. Typing `make` will attempt to build the executable `Gadget4` (hint: using `make -j X`, where X is some number of threads, the build process can be carried much faster in parallel on multi-core machines). It is also possible to override the default names for the configuration file and the executable by specifying them as parameters for this command, for example as `make CONFIG=MyNewConf.sh EXEC=Gadget4new`. Also, one may pass a directory to make, for example in the form `make DIR=mysim/newmodels/runA`, and then the configuration file contained in this directory will be used, the build process will be carried out there, and the executable appears there as well. This can be very useful to build and run different code configurations from a common code base without risking to accidentally mix up the executables. Each simulation should be organized into a different subdirectory in this case, with configuration file and executable being placed into the same subdirectory.

Often, one needs to specify (possibly non-standard) locations of the libraries required by GADGET-4, or the name of the compiler and the settings of compiler flags to be used in order to build the code. To make the selection of the target system and changes between different compiler setups relatively simple, these settings have been largely decoupled from the `Makefile` , i.e. the `Makefile` itself should usually not be changed in any significant way. Only if you want to add additional source files to the code or introduce a new target computer system a small change is needed there.

This is facilitated through the concept of a system type (which is the target computer combined with compiler settings), which is selected through the environment variable `SYSTYPE`, or by the file `Makefile.systype`. The simplest way to create the file `Makefile.systype` is to copy it from `Template-Makefile.systype`, and then comment in/out the desired type. For the selected symbolic name of the target system type, there should then be a short section in the `Makefile` that includes usually two makefile stubs from the folder `buildsystem/` which specify the paths to the different libraries if they are not in standard locations, and determine the compiler name(s) and compiler options that should be used. Once this is all set-up properly, you can then switch to a different compiler setting by toggling one line in `Makefile.systype`. Also, if you set the environment variable `SYSTYPE` in your login-script (.profile, .bashrc, etc.) on your target computer (recommended for convenience), you will normally not have to deal with the file `Makefile.systype` at all and can compile right away.

To summarize, if you want to set-up GADGET-4 for compilation on a new, not yet defined computer system, you have to go through the following steps:

- Make sure that the needed libraries are available, either by compiling them yourself (they can all be installed in user space if needed) or by loading the right modules on systems that support the module-environment.

- Add a new symbolic name for your computer system to `Template-Makefile.systype` (this file is meant to keep a record of all defined machine types), and then select this name through the `SYSTYPE` environment variable or through the file `Makefile.systype`.

- In the file `Makefile`, create an if-clause in the "define available Systems" section, in which you include two short files from the `buildsystem` directory that define path names to the libraries that you need (i.e. if not in default locations), and that specify the compiler and its flags that you want to select. Follow the provided examples to create your own versions of these files, as needed. The reason why this information is normally not directly included in an if-clause in the `Makefile` (which would be possible too) is to avoid repetition of identical information in the Makefile (for example, because the same settings for the `gcc` compiler may be used on many different target systems). Instead, the same small sub-file from `buildsystem/` can be included for many different target systems.

We note that many compile-time options are introduced with `#ifdef` / `#endif` statements in the code in order to encapsulate the corresponding code extensions and allow them to be fully disabled (i.e.~to be **completely** eliminated from the compiled code). This is done for performance and memory reasons, which for us take precedence over the detrimental impact on readability/ clarity of the code brought about by such compiler pragmas. Another, more problematic side effect of using such compile-time symbols is however that typos in specifying any of them may go easily undetected. To provide protection against this, the code automatically runs a python-based check script over the source code as part of the build process that will complain if any undefined or misspelled compile time symbols are listed in `Config.sh`, or, conversely, if (new) symbols in the source code are present that are neither defined nor briefly explained in `Template-Config.sh`. This checking mechanism can be disabled if desired by using `make`

`build` instead of `make`, but we strongly advise against this. Also, it is possible to deliberately exempt a symbol from the checking procedure. This is needed, for example, for header-file guards, and those symbols should be added to the file `defines_extra`.

As a further extension of these checks, we have also added functionality that demands that all compile-time and run-time options are actually documented. To this end, if you add a new compile time option yourself, this needs to be documented in `documentation/04_config-options.md`, and if you add a new run-time parameter, it needs to be documented in `documentation/05_parameterfile.md`, otherwise the code will refuse to compile and complain accordingly. Similarly, documented options that do no longer exist in the code will lead to error messages when compiling. This checking of the documentation can be disabled by using `make build` instead of `make`, but as pointed out above, this constitutes bad practice and should not be done.

Finally, note that the GADGET-4 code is now written in the C++ language, and the source files can only be compiled with a C++ compiler. While many useful and advanced features of the C++ language are used (like templating and operator overloading), because GADGET-4 evolved from an older code base in C, it effectively represents in many places a mixture of C++ and C-styles of coding.

Starting the code

To start a simulation, invoke the executable with a command like

```
mpirun -np 32  ./Gadget4  param.txt
```

This will start the simulation using 32 MPI ranks, and with simulation parameters as specified in the parameter file (see below) of name param.txt. Note that on some systems, the command to launch parallel execution may have a different name or syntax (e.g mpiexec, srun, poe, etc). Consult the man-pages or the local support if in doubt.

The code does not need to be recompiled for a different number of processors, or for a different problem size. It is also possible to run GADGET-4 with a single processor only. In this case, the leading `mpirun -np 1` can normally be omitted, and GADGET-4 will behave like a serial code. It is still necessary to have MPI installed, though, because even then the code will make some calls to the MPI library (but none that actually do non-trivial communications). There is no restriction for the processor number to be a power of two, even though these partition sizes are slightly preferred, because they allow the most efficient communication schemes. However, in regimes where the code works well in terms of scalability, the communication times should is subdominant anyhow, so this is not an important consideration in practice.

When more than one MPI-rank is used, the code will use a hybrid communication scheme in which data stored by different MPI processes on the same compute node are accessed directly via shared-memory. The code automatically detects groups of MPI ranks running on the same node. If more than one node is in use, at least one MPI process on each node is set aside for asynchronously serving incoming communication requests from other nodes (if only a single shared-memory is used, this is not done). This means that multi-node jobs must have a minimum of two MPI ranks on each node. Today's machines offer typically many more cores per node than 2, and their full power is made available to GADGET-4 if one MPI rank is placed on every core.

At least one MPI communication rank needs to be set aside for every 64 ranks on a shared memory node in multi-node jobs. If the number of MPI ranks per shared memory node is larger than 64, one therefore needs to specify the `NUMBER_OF_MPI_LISTENERS_PER_NODE=X` option, with X larger than 1 (which is the default).

While GADGET-4 is running, it will print out many log-messages that inform about the present steps taken by the code. When you start a simulation interactively, these log-messages will appear on the screen, but you can also redirect them to a file. For production runs on a cluster controlled by a queuing system, you will usually have to put the above start-up command of GADGET-4 into a batch script-file that is submitted to the queuing system. In this case, the standard output of GADGET-4 is usually automatically piped into a file.

For example, assuming that the batch system SLURM is in use, a batch-script similar to

```
#!/bin/bash
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --mail-user=vspringel@mpa-garching.mpg.de
#SBATCH --time=24:00:00
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=40
#SBATCH --job-name SB128

echo
echo "Running on hosts: $SLURM_NODELIST"
echo "Running on $SLURM_NNODES nodes."
echo "Running on $SLURM_NPROCS processors."
echo "Current working directory is `pwd`"
echo

mpiexec -np  $SLURM_NPROCS  ./Gadget4 param.txt
```

could be used to start a job with 2 nodes with 40 MPI ranks each (i.e. using 80 cores in total). Here a semi-automatic setting of the appropriate value of the total number of cores through SLURM environment variables has been used. Note that in this example, GADGET-4 will only use 78 MPI processes for the computational work, because on each node one process will be set aside by the code purely for communication purposes.

Interrupting a run

Often, a single submission to a queue system will not provide enough CPU-time to finish the computation of a large production simulation. Therefore, a running simulation can be interrupted after every timestep, and resumed at the very same place later on. If the CPU-time limit is specified correctly in the parameter file, the code will interrupt itself automatically before the CPU-time limit is reached, and write a set of restart-files. Actually, each processor writes its own restart file. These restart-files can be used to resume the simulation conveniently (see below) at the point where it was interrupted.

Sometimes you might want to interrupt the code manually with the possibility to continue it later on without losing any of the calculations. This can be achieved by generating a file named `stop` in the output-directory of a simulation, e.g.

```
echo > /u/vrs/myoutput/stop
```

The code will then write a restart-file and terminate itself after the current timestep has been completed, and the file `stop` will be erased automatically. If a simulation reaches the specified CPU-time limit, it creates a file `cont` in the output directory (besides the restart files). This can be used in a job batch script to detect that a run has ended gracefully and is hence ready for continuation, at which point the script may submit another job for continuation. Modern queuing

systems also allow job dependencies, which can be used to submit a chain of jobs that are executed sequentially, but only of the previous job finishes with an OK status. In case a run crashes for some reason, the file `cont` is absent and hence a restart should not be initiated until the cause for the problem has been investigated and removed. Note that restart files are also generated when the last timestep of a simulation has been completed and the final time has been reached. They can be used if one later wants to extend the simulated timespan beyond the original final time.

One can also instruct the code to write restart files in regular intervals (see parameterfile description) without stopping the simulation. This is meant to protect against system failures or code problems in long-running simulations. If such a thing happens, one can then resume the run from the most recent set of restart files, which limits the amount of lost time. Furthermore, whenever a new set of restart files is written, the old set is first renamed into a set of backup restart files. Hence, if a system failure occurs while the new restart files are written, the old set of files still exits and is valid, and can hence be used for resuming the run. To this end, one needs to rename the backup versions of the restart files and give them the standard name again, followed by resuming the run with the restart option (see below).

Restarting a run

Restarting from restart-files

To resume a run from restart-files, start the code with an optional flag 1 after the name of the parameterfile, in the form

```
mpirun -np 32  ./Gadget4  param.txt  1
```

This will continue the simulation with the set of restart files in the output-directory, with the latter being specified in the parameterfile. Restarting in this fashion is transparent to the code, i.e. the simulation behaves after the restart exactly as if it had not been interrupted to begin with. Strictly speaking this is only true if the `PRESERVE_SHMEM_BINARY_INVARIANCE` option is activated. This prevents that sums of partial forces may be computed for certain particles in different order when a run is repeated due to varying machine weather. While mathematically equivalent, this will introduce differences in floating point round-off that can be quickly amplified in regimes of non-linear evolution.

When the code is started with the restart-flag, the parameterfile is parsed, but only some of the parameters are allowed to be changed, while any changes in the others will be ignored. Which parameters these are is explained in the section about the parameterfile, and if such a new value for a parameter is accepted this is also reflected in the log-files of the run.

It is **important** to not forget the 1 if you want to resume a run -- otherwise the simulation will restart from scratch, i.e. by reading in the initial conditions again! Also note that upon restarting from restart-files, the number of MPI ranks used for a simulation cannot be changed; if this is attempted an error message is issued. Note that restart files can not necessarily be transferred from one computer system to another one, or reused when the compiler is changed, because the layout and padding of structures stored in the binary restart files may then be different. Hence, if you want to continue a simulation with a different number of MPI ranks, on another computer architecture, or with a different compiler, restarting from a snapshot file is the method of choice. This will be discussed next.

Restarting from snapshot-files

There are two possibilities to restart a simulation from a previous snapshot file. In the first possibility, one simply adopts the snapshot file as new initial conditions. Note that this option requires changes in the parameterfile: You need to specify the snapshot-file as initial-conditions-file, you also need to set `TimeBegin` (see below) to the correct time corresponding to the snapshot-file. In addition, you should change the base filename for the snapshot files, since the counter for the outputs will start at 0 again, thereby possibly overwriting outputs you might already have obtained. Once this is done, you can continue/restart the run from the snapshot-file (without the optional 1).

An alternative to this somewhat contrived procedure is to restart the code with a flag equal to `2` after the name of the parameterfile, i.e. just like above for the restart from restart-files, but with the `1` replaced by `2` , and with an additional further parameter that specifies the number of the snapshot in the output file that you want to start from. The parameterfile normally needs not to be changed in this case, in particular, the code takes the new starting time from the snapshot file that is read. In addition, this also will cause any further snapshots that are generated to have higher sequence numbers than this starting number, i.e. the code will number all further snapshots starting with one plus the number of the snapshot that is used as input file. For example, the command for restarting from snapshot 7 would look as follows:

```
mpirun -np 32  ./Gadget4  param.txt  2 7
```

Note that the restart-from-snapshot-files option allows a change of the number of processors used for the simulation. This is not possible if you restart from restart-files. However, restarting from restart-files is always the preferred method to resume a simulation, because it is faster (various start-up calculations do not have to be done), and it minimises perturbations in the time integration scheme of a running simulation.

Starting postprocessing

It is also possible to apply certain postprocessing algorithms built into GADGET-4 to a snapshot file residing in the output directory. This works similarly to the restart from a snapshot file option in that one specifies both a start-up option and a snapshot number (as well as potentially further parameters). For example, to calculate a FOF/Subfind group catalogue for a snapshot file with the number 7, one could start the code as

```
mpirun -np 32  ./Gadget4  param.txt  3  7
```

because the `3` selects the group finding algorithm. If such a postprocessing option is selected, the code will not carry out any time evolution. Instead, the given snapshot is just read in, some postprocessing is done, the output is written to the output directory, and then the code ends. One can obtain a short list of the available postprocessing options when the code is started without any paramater. Other postprocessing options for example include the calculation of a matter power spectrum, the conversion of a snapshot file from one of the supported file formats to another, or the creation of cosmological initial conditions (the latter is then actually more of a preprocessing and not a postprocessing option).

The available postprocessing options are as follows:

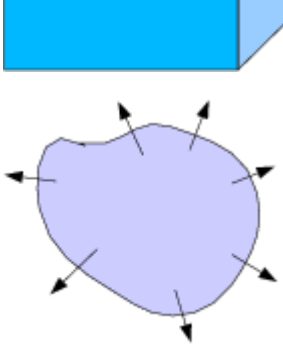
RestartFlag	**Action**
0	Read initial conditions and start simulation

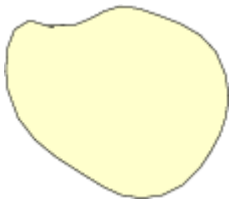
RestartFlag	Action
1	Read restart files and resume simulation
2	Restart from specified snapshot dump and resume simulation
3	Run FOF and optionally SUBFIND
4	Calculate a matter power spectrum for specified snapshot number
5	Convert snapshot file to different format
6	Create cosmological initial conditions
7	Create descendant/progenitor information when merger tree is done in postprocessing
8	Arrange halos in merger trees, using all group catalogues up to given snapshot number
9	Carry out an I/O bandwidth test to determine best setting for the number of concurrent reads/writes
10	Rearrange particle-lightcone data in merger tree order
11	Rearrange most-bound snapshot data in merger tree order

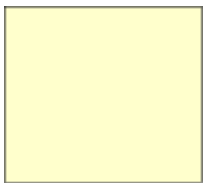
Types of simulations

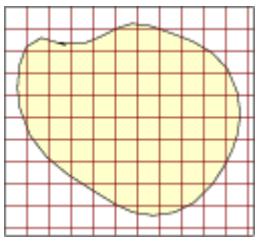
There are a number of qualitatively different simulation set-ups that can be run with GADGET-4, which differ mostly in the type of boundary conditions employed, in the physics that is included, whether or not cosmological integrations with comoving coordinates are used, and in the selection of numerical algorithms. A schematic overview of a subset of these simulation types, concentrating on how gravity calculations are done, is given in the following table.

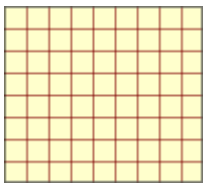
Overview of simulation types

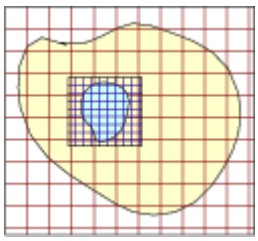
Type of Simulation	
	Newtonian space
	Periodic long box
	Cosmological, physical coordinates

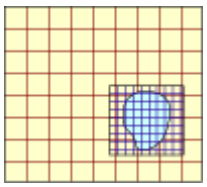

Cosmological, comoving coordinates

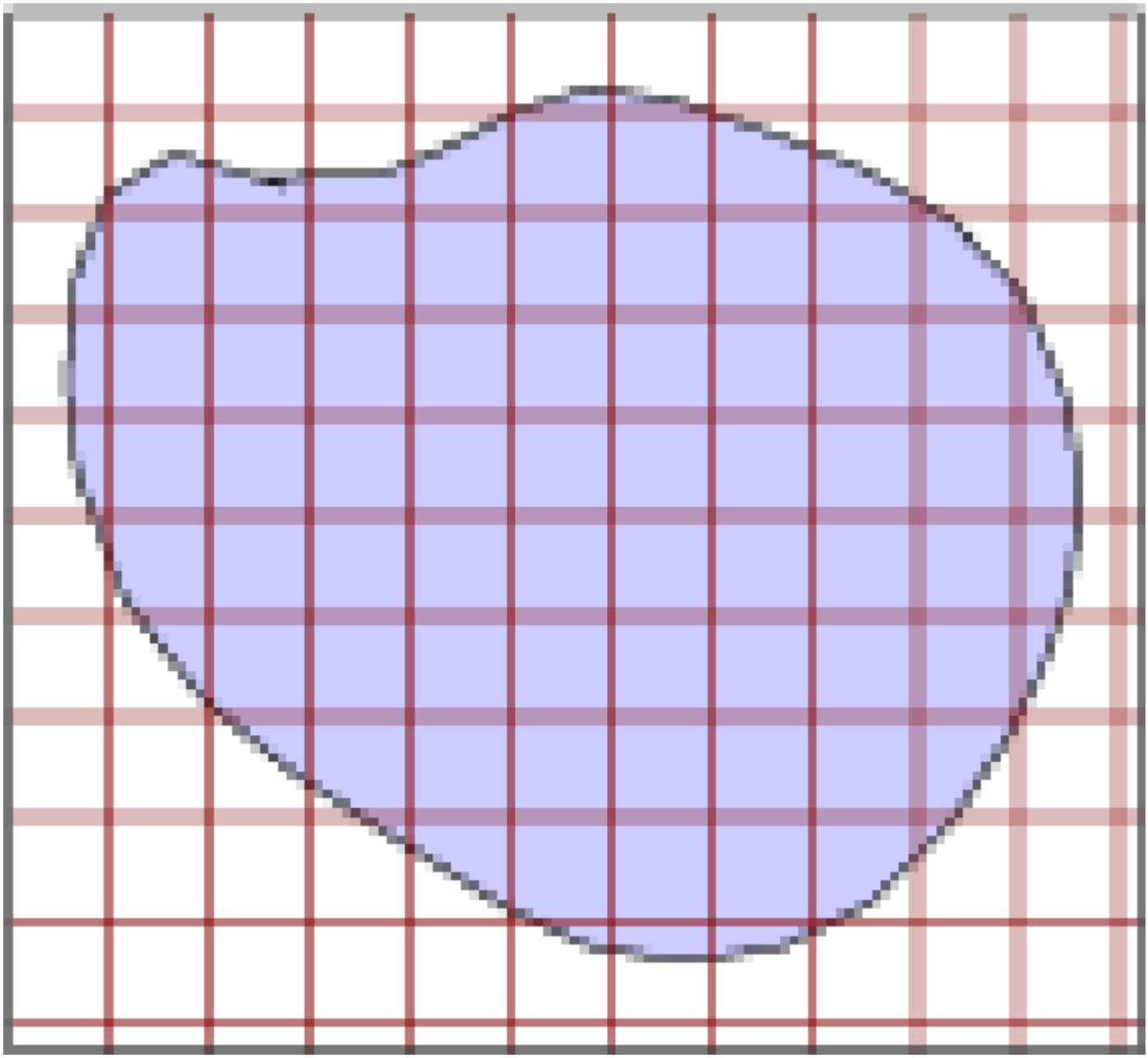

Cosmological, comoving periodic box


Cosmological, comoving coordinates, TreePM


Cosmological, comoving periodic box, TreePM


Cosmological, comoving coordinates, TreePM, Zoom


Cosmological, periodic comoving box, TreePM, Zoom



Newtonian space, TreePM

Cosmological Simulations

Cosmological integrations with comoving coordinates are selected via the parameter `ComovingIntegrationOn` in the parameterfile. Otherwise the simulations always assume ordinary Newtonian space. The use of self-gravity needs to be explicitly enabled through the `SELFGRAVITY` compile-time switch, otherwise only external gravitational fields and/or hydrodynamical processes are computed. Periodic boundary conditions and the various variants of the Tree, FMM, TreePM, and FMM-PM algorithms require compile-time switches to be set appropriately in the configuration file.

In particular, the TreePM algorithm is switched on by passing the desired mesh-size at compile time via the Config.sh file to the code. The relevant parameter is `PMGRID`, see below. Using an explicit force split, the long-range force is then (normally) computed with Fourier techniques, while the short-range force is calculated with the tree. Because the tree needs only be walked

locally, a speed-up can arise, particularly for near to homogeneous particle distributions, but not only restricted to them. Both periodic and non-periodic boundary conditions are implemented for the TreePM and FMM-PM approaches. In the non-periodic case, the code will internally compute FFTs of size `HRPMGRID`. In order to accommodate the required zero-padding, only half that size is actually used to cover the high-res region. If `HRPMGRID` is not specified, a default value equal to `PMGRID` is used. For zoom-simulations, the code automatically places the refined mesh-layer on the high-resolution region. This is controlled with the `PLACEHIGHRESREGION` option.

Newtonian space

Periodic boxes need not necessarily be cubical in GADGET-4. They can be, if desired, stretched independently in any of the coordinate directions through `LONG_X`, `LONG_Y` and `LONG_Z` options. This works both for SPH simulations and also for simulations including self-gravity. In the latter case, one may also choose one direction to be non-periodic in self-gravity, realizing mixed boundary conditions that allow one to simulate infinitely extended two-dimensional systems.

Stretched boxes

This can now also be done with gravity and periodic boundaries in just two dimensions.

Two-dimensional simulations

It is also possible to run SPH simulations in two dimensions only, which is primarily provided for test-purposes. Note that the code is not really optimised for doing this; three coordinates are still stored for all particles in this case, and the computations are formally carried out as in the 3D case, except that all particles lie in one coordinate plane, i.e. either with equal x, y, or z-coordinates.

Code Configuration

Many aspects of GADGET-4 are controlled with compile-time options rather than run-time options in the parameterfile. This is done in order to allow the generation of highly optimised binaries by the compiler, even when the underlying source code allows for many different ways to run the code. Unfortunately, this technique has the disadvantage that different simulations typically require different binary executables of GADGET-4, so "installing" GADGET-4 on a computer system is not possible, and the act of compiling the code is an integral part of working with the code. Keeping the executable at a single place is not recommended either, because if several simulations are run concurrently, this invokes the danger that a simulation is started or resumed with the wrong binary. Note that while GADGET-4 checks the plausibility of some of the most important code options, this is not done for all of them. Hence, to minimise the risk of using the wrong code for a simulation, it is highly recommended to produce a separate executable for each simulation that is run.

As a piece of advice, a good strategy for doing this in practice is to create a separate directory for each simulation that is made, place a copy of the whole simulation source code together with its makefile into this directory, compile the code there and run it in this directory as well, with the output directory specified as a subdirectory of the simulation directory. In this way, the code and

its settings become a logical and integral part of the output generated by the code. Everything belonging to a given simulation is packaged up in one directory, and it becomes easy to reproduce what was done even if considerable time should have passed, because the precise version of the original code that was used and all produced log files are readily available. An alternative is to have only a single copy of the code, but then to use a separate directory for each simulation that is done, including placing its configuration file there and carrying out the compilation in this directory as well (by passing the `make DIR=` option to the build as well). Then at least all object files and the executable are unambiguously associated with a particular simulation run.

The easiest way to create the file `Config.sh` needed for compilation is to produce it as a copy of `Template-Config.sh` , and then modify it as needed. When created from `Template-Config.sh` , the `Config.sh` file contains a dummy list of all available compile-time code options, with most of them commented out by default. To activate a certain feature, the corresponding symbol should be commented in, and given the desired value, if appropriate.

Important Note:

Whenever you change one of the options described below, a full recompilation of the code may in general be necessary. For this reason, the `Config.sh` itself has been added to the list of dependences of each source file in the makefile, such that a complete recompilation should happen automatically when the configuration file is changed and the command `make` is given. Note that a manual recompilation of the whole code can be enforced with the command `make clean` , which will erase all object files, followed by `make` .

Most code options in `Config.sh` are switches that toggle a certain feature on/off. Some of the symbols also take on a value if set. The following list shows these switches with fiducial example values where appropriate.

Parallelization options

IMPOSE_PINNING

Ask the code to pin MPI processes to cores in an optimum fashion. This requires the hwloc library for detecting the processor topology. It will generally only work on Linux. Note that most modern MPI libraries can also be asked to arrange for the pinning via options to the MPI start-up command, or they do this per default anyhow.

IMPOSE_PINNING_OVERRIDE_MODE

In case the MPI start-up has already established a pinning, this is normally detected and then `IMPOSE_PINNING` does not do anything. Overriding this pre-established pinning can be enforced with this option.

EXPLICIT_VECTORIZATION

This enables a few compute kernel (currently in SPH only) to explicitly use AVX instructions through the use of the vectorclass C++ library.

PRESERVE_SHMEM_BINARY_INVARIANCE

This can be used to preserve the order in which partial results are added in case the parallel tree walks use shared memory to access tree branch data that has been imported by different processes on the same shared memory node. In this case, exact binary invariance of output can be retained upon reruns of the code.

SIMPLE_DOMAIN_AGGREGATION

This tries to very roughly restrict the domain decomposition to place adjacent domain pieces on the same shared memory node. This will then (in some cases significantly) reduce the number of imports of particles and nodes that need to be made, but at the price of a higher imbalance overall. Whether this is worth it depends strongly on the problem type. A better (forthcoming) solution will be to do the domain decomposition hierarchically right away, taking the outline of the shared-memory nodes into account from the outset.

Basic operation mode of code

PERIODIC

Set this option if you want to have periodic boundary conditions. In this case, the `BoxSize` parameter in the parameterfile becomes relevant.

TWODIMS

This effectively switches off one spatial dimension in SPH, i.e. the code follows only 2D hydrodynamics either in the xy-, yz-, or xz-plane. One doesn't need to tell the code explicitly which of these planes are used, but all coordinates of the third dimension need to be exactly equal (usually set to zero).

ONEDIMS

Similarly to `TWODIMS`, this effectively only allows one spatial dimension in SPH, i.e. the code follows only 1D hydrodynamics either in the x-, y-, or z-directions. One doesn't need to tell the code explicitly which of these directions are used, but all coordinates of the other dimensions should be set to zero.

LONG_X_BITS = 2

In case periodic boundary conditions are used (i.e. `PERIODIC` is on), one can stretch the x-dimension of the box relative to `BoxSize` by the factor `1 / 2^LONG_X_BITS`. A setting equal to 2 like in this example, would hence mean that the boxsize in the x-direction will be `BoxSize/4`.

LONG_Y_BITS = 2

Similarly to the above, this switch can stretch the periodic box in the y-direction by the factor `1/2^LONG_Y_BITS` relative to BoxSize.

LONG_Z_BITS = 1

Finally, this option implements a possible stretch in the z-direction, similar to `LONG_X` and `LONG_Y`. Only a subset of the stretch factors or several/all of them may be used. The `LONG_X/Y/Z_BITS` values must be positive integers.

NTYPES = 6

Number of particle types that should be used by the code. If not set, a default value of 6 is adopted (which is the fixed value realised in GADGET-2/3). The different particle types are useful as a means to organize the simulation particles into different logical sets, which can be helpful for analysis (e.g., one may have a type for dark matter, one for stars, one for disk stars, etc.). The type 0 is reserved for gas particles, and to them SPH is applied. All other particle types are treated as collisionless particles which are treated on the same footing as far as gravity is concerned. For each of the types, one needs to specify a gravitational softening class in the parameterfile.

GADGET2_HEADER

This should be set if backwards compatibility of the snapshot file header format with GADGET-2/3 is required. Applies only to file formats 1 and 2. This can be useful, for example, to read in old initial conditions. Note that in this case NTYPES may not be larger than 6, and it should be at least as large as the number of types actually contained in the legacy initial conditions. Also, files cannot have more than 2^31 particles of any type in total in this case.

SECOND_ORDER_LPT_ICS

Processes the initial conditions before actual simulation start-up to add in second order Lagrangian perturbation theory corrections. This only works for specially constructed initial conditions created with Adrian Jenkin's IC code.

LEAN

This option is meant for special DM-only simulations that aim at high savings in memory use. Only a uniform particle mass, a single particle type, no hydrodynamics, and a single softening class are allowed. In addition, one should use `TREEPM_NOTIMESPLIT`, and refrain from using double precision to obtain a very small memory footprint.

Gravity calculation

SELFGRAVITY

This needs to be switched on if self-gravity should be computed by the code. If it is off, one can still have hydrodynamics, and optionally a prescribed external gravitational field.

HIERARCHICAL_GRAVITY

If this is enabled, the time integration is carried out in a hierarchically fashion in which the gravitational Hamiltonian is hierarchically split into slow and fast dynamics. The advantage of this approach is that small subsystems on short timesteps can be cleanly decoupled from the more slowly evolving rest of the system. This can be advantageous especially if there is a very deep and increasingly thinly populated tail in the timestep distribution. It also allows a time integration that is formally momentum conserving despite the use of individual timesteps. However, as additional partial forces need to be computed, this approach typically entails a somewhat higher number of force calculations. This can still be more efficient overall, but only if the number of particles per timebin quickly declines when going to shorter timebins.

FMM

If this is enabled, the Fast Multipole Method for gravity calculations instead of the one-sided classic tree algorithm is used.

MULTIPOLE_ORDER = 2

If this is enabled, one can control the order of the multipole expansion that is used. For a value of 3, quadrupole moments are included in the normal tree calculation and/or in the FMM calculations. A value of 4 includes octupole moments as well, and 5 goes further to hexadecupole moments. Note that these higher orders increase the memory and CPU time needed for the force calculations, but deliver more accurate forces in turn. It depends on the specific application whether this is worthwhile. The default is a value of 2.

EXTRA_HIGH_EWALD_ACCURACY

If this is extrapolated, the Ewald corrections are extrapolated from the look-up table with a third order Taylor expansion (quite a bit more expansive), otherwise a second order Taylor expansion is used.

EXTRAPOTTERM

If this is activated, the extra multipole term in principle present for the potential computation is evaluated, even though it does not enter the force. For example, for monopole/dipole order (p=2), the code will then compute quadrupole moments and use them in the potential, but not in the force computation.

ALLOW_DIRECT_SUMMATION

When this is set (it requires `HIERARCHICAL_GRAVITY`), the code will calculate the gravitational forces for very small groups of particles (the threshold for the group size is given by the constant `DIRECT_SUMMATION_THRESHOLD`, which is set per default to 500, but one can override this value in the config file if desired) with direct summation. This can be useful if the timestep distribution has a tail to very short, poorly occupied timebins. Doing the corresponding timesteps frequently for very small sets of particles can be faster with direct summation than doing it via the tree or FMM because then all overhead associated with tree construction and tree walks can be avoided.

RANDOMIZE_DOMAINCENTER

When this is activated the whole particle set is randomly shifted in the simulation box whenever a domain decomposition is done. This can be useful to average out in time subtle spatial correlations in the force errors that arise from the relative positioning of the particle to the oct-tree geometry. Since integer coordinates are used for particle positions, this shifting does not entail any round-off error, and is reversible. In particular, the shifts will not be visible in any of the outputs created. This option is basically always recommended, and should have only positive effects.

RANDOMIZE_DOMAINCENTER_TYPES = 2

Can be set to select one or several types (this is a bitmask) which will then be used to locate the extension of a certain region. When the particle set is randomly translated throughout the box, the code will then try to avoid intersecting large oct-tree node boundaries with this region. When this option is not set explicitely but PLACEHIGHRESREGION is active, then this is automatically done with a default setting RANDOMIZE_DOMAINCENTER_TYPES=PLACEHIGHRESREGION.

EVALPOTENTIAL

When this is activated, the code also computes the gravitational potential (which is not needed for the dynamics). This costs a bit of extra memory and CPU-time.

TREE_NUM_BEFORE_NODESPLIT = 4

The number of particles that may be in a tree node before it is split into daughter nodes. If the number is reduced to 1, a fully threaded tree is obtained in which leaf nodes contain one particle each. (Note that empty nodes are not stored, except potentially for part of the top-level tree if it is very finely refined.)

EXTERNALGRAVITY

If this is switched on, an external gravitational field can be added to the dynamics. One still has to define with different switches and/or parameters what this external field is.

EXTERNALGRAVITY_STATICHQ

Activates a simple external potential due to a Hernquist dark matter halo, with parameters specified in the parameterfile.

TreePM Options

PMGRID = 512

This enables the TreePM method, i.e. the long-range force is computed with a PM-algorithm, and the short range force with the tree or with FMM. The parameter has to be set to the size of the mesh that should be used, e.g. 64, 96, 128, etc. The mesh dimensions need not necessarily be a power of two, but the FFT is fastest for such a choice. Note: If the simulation is not in a periodic box, then a FFT method for vacuum boundaries is employed, where due to the required zero-

padding, only half the mesh is covering the region with particles. (see also the HRPMGRID option).

ASMTH = 1.25

This can be used to override the value assumed for the scale that defines the long-range/short-range force-split in the TreePM algorithm. The default value is 1.25, in units of mesh-cells. A larger value will make the transition region better resolved by the mesh, yielding higher accuracy and less residual scatter in the force matching region, but at the same time the region that needs to be covered by the tree/FMM grows, which makes the computation more expensive.

RCUT = 6.0

This can be used to override the maximum radius out to which the short-range tree-force is evaluated in case the TreePM/FMM-PM algorithm is used. The default value is 4.5, given in mesh-cells. Going much beyond 6 should not yield further improvements in the way the force matching region is treated.

NTAB = 128

This can be used to define the size of the short-range lookup table. The default should normally be sufficient to have negligible influence on the obtained force accuracy.

PLACEHIGHRESREGION = 2

If this option is set (will only work together with PMGRID), then the short range force computation in the TreePM/FMM-PM algorithm is accelerated by an additional intermediate mesh with isolated boundary conditions that is placed onto a certain region of high-res particles. This procedure can be useful for zoom-simulations, where the majority of particles (the "high-res" particles) are occupying only a small fraction of the volume. To activate this, the option needs to be set to an integer value in the form of a bit mask that encodes the particle type(s) that should be used to define the spatial location of the high-resolution patch. For example, if types 1 and 4 are supposed to define this region, then the parameter should be set to PLACEHIGHRESREGION=2+16 , i.e. to the sum 2^1+2^4. The actual spatial region is then determined automatically from the current locations of these particle types. This high-res zone may also intersect with the box boundaries if periodic boundaries are used. Once the spatial region is defined by the code, it however applies to all particle types. In fact, the short range interactions between particle pairs that fall fully inside the high-res region are computed in terms of two contributions, one from the intermediate PM mesh that covers the high-res region, and the other from a tree/FMM force that however now only extends to a region that is reduced in size (which is the origin of a possible speed-up with this option). Particle pairs for which at least one of the partners is outside the high-res region get the normal Tree/FMM contribution with the standard cut-off region, corresponding to the plain TreePM approach with the course grid covering the full simulation volume. Also note that when the particle set is randomized throughout the box (with the RANDOMIZE_DOMAINCENTER option), the code additionally tries to avoid intersecting larger oct-tree node boundaries by imposing certain restrictions in the randomization.

HRPMGRID = 512

When PMGRID is set and non-periodic simulations are used, or when PLACEHIGHRESREGION is active, the FFT algorithm is used to compute non-periodic gravitational fields, which requires zero-padding, i.e. only one octant of the used grid can actually be covered by the mass distribution. The grid size that the code uses for these FFTs is equal to HRPMGRID if this is set, otherwise the value of PMGRID is used for this grid dimension as well. This option is hence optional, and allows if desired the use of different FFT sizes for the periodic calculation covering the whole region, and for the non-periodic calculations covering the zoom region.

FFT_COLUMN_BASED

Normally, the code employs a slab-based approach to parallelize the FFT grids across the MPI ranks. However, for a Ngrid^3 FFT, this means that at most Ngrid different MPI ranks can participate in the computation. This represents a serious limit to scalability as the minimum computational effort per MPI rank then scales as Ngrid^2, and not more than Ngrid MPI ranks can anyhow be used. If one is in the regime where the number of MPI ranks exceeds Ngrid, it is therefore a good idea to activate FFT_COLUMN_BASED, which will use a slab-decomposition instead. Now the minimum effort per MPI rank scales only as Ngrid, and the maximum number of ranks that can participate is Ngrid^2, meaning that in practice this limit will not be encountered. The results should not be affected by this option. Because the column-based approach requires twice the number of transpose operations, it is normally somewhat slower than the slab-based approach in the regime where the latter can still scale (i.e. for Ngrid >= Ncpu), so only for very large small processor numbers and large grid sizes, the column based approach can be expected to yield a speed advantage, aside from the better memory balance it provides.

PM_ZOOM_OPTIMIZED

Set this option if the particle distribution is spatially extremely inhomogeneous, like in a zoom simulation. In this case, the FFT algorithm will use a different communication strategy that can better deal with this inhomogeneity and maintain work balance despite of it. If the particle distribution is reasonably homogenous (like in a uniformly sampled cosmological box), it is normally better to leave this option off. In this case, a simpler communication strategy well adapted to this situation is used which avoids some of the overhead required by the PM_ZOOM_OPTIMIZED option.

TREEPM_NOTIMESPLIT

When activated, the long- and short-range gravity forces are simply summed up to a total force, and then integrated in time with a common timestep. Otherwise, the short-range forces are subcycled, and the PM force size is stored separately and integrated on a global, longer timestep.

GRAVITY_TALLBOX = 2

This can be used to set-up gravity with mixed-mode boundary conditions. The spatial dimension selected by the option is treated with non-periodic boundaries, while the other two are treated with periodic boundary conditions. This can facilitate, for example, stratified box simulations in which there is periodicity in the transverse directions (which define, e.g., the plane of a sheet) and open boundary conditions in the perpendicular direction. Set-ups of this kind are often used in simulations of star formation.

Treatment of gravitational softening

NSOFTCLASSES = 4

Number of different softening values. Traditionally, this is set equal to the number of particle types, but it can also be chosen differently. The mapping of a particle type to a particular softening class is normally done through the parameter values `SofteningClassOfPartTypeX` as specified in the parameterfile. Several different particle types could be mapped to the same softening class in this case, and not all softening classes actually must be used by particles. With the help of the `INDIVIDUAL_GRAVITY_SOFTENING` option, the mapping can also be based on the particle mass, so that particles of the same type may be mapped to different softening lengths if their masses are different. This is an attractive option especially for zoom simulations in order to allow heavier boundary particles to be automatically associated with the closest natural softening length from the tableau of available softening lengths.

INDIVIDUAL_GRAVITY_SOFTENING = 4+8+16

If this option is enabled, the selected particle types (`INDIVIDUAL_GRAVITY_SOFTENING` is interpreted as a binary mask specifying these types) calculate for each of their particles a target softening based on a cube-root scaling with the particle mass. As a reference point, the code takes the softening class assigned for particle type 1, and the average particle mass of the type 1 particles. The idea is to use this option for zoom simulations, where one assigns the high-resolution collisionless particles to type 1 (all with equal mass) and maps the type 1 to a certain softening class, which then fixes the softening assigned for type 1. For all other used particles types (which typically may involve variable masses within the type), one then activates this option. For the corresponding particles, a desired softening is computed based on a cube-root scaling with their masses relative to the reference mass and softening of the type 1 particles. From all the available softening classes, the code then assigns the softening class with the smallest logarithmic difference in softening length to the computed target softening length. Since only the available softening classes can be used for this, one should aim to supply a fine enough set of available softening classes when this option is used.

ADAPTIVE_HYDRO_SOFTENING

Sometimes, one may want to scale the gravitational softening length of gas particles with their SPH smoothing length. This can be achieved with this option. Enabling it, requires additional parameters in the parameterfile, namely `GasSoftFactor`, `MinimumComovingHydroSoftening` and `AdaptiveHydroSofteningSpacing`. When this option is used, the softening type 0 is not available for any other particle type (and also won't be selected by types included in `INDIVIDUAL_GRAVITY_SOFTENING`). `SofteningClassOfPartType0` needs to be 0, and all other particle types need to have a non-zero value for their SofteningClassOfPartType. The softening values specified in the parameterfile for softening type 0 are ignored, instead the softening is selected for all gas particles based on their smoothing length from a finely spaced discrete table (see explanations for the above softening parameters).

SPH formulation

PRESSURE_ENTROPY_SPH

Enables the pressure-entropy formulation of SPH (similar to Hopkins 2013), otherwise the default density-entropy formulation (Springel & Hernquist, 2002) is used.

OUTPUT_PRESSURE_SPH_DENSITY

Outputs also density computed as in the standard SPH pressure-entropy formulation. This is only useful if `PRESSURE_ENTROPY_SPH` is used.

INITIAL_CONDITIONS_CONTAIN_ENTROPY

The intial conditions file contains entropy instead of the thermal energy.

GAMMA = 1.4

Sets the equation of state index in the ideal gas law that is normally used in GADGET-4's SPH implementation. If not set, the default of 5/3 for a mono-atomic gas is used.

ISOTHERM_EQS

This defines an isothermal equation of state, P = rho c^2, where c^2 is kept constant for every particle at the value u = c^2 read from the initial conditions in the internal energy per unit mass field. `GAMMA=1` is set automatically in this case.

REUSE_HYDRO_ACCELERATIONS_FROM_PREVIOUS_STEP

If this option is enabled, the code does not recompute the SPH hydrodynamical acceleration at the beginning of a timestep, but rather reuses the one computed at the end of the previous timestep, which is typically good enough. The two accelerations can in principle differ slightly due to non-reversible viscous effects, or external source functions (e.g. radiative heating or cooling).

IMPROVED_VELOCITY_GRADIENTS

Use more accurate estimates for the velocity gradients following Hu et. al (2014), which enter the calculation of a time-dependent artificial viscosity.

VISCOSITY_LIMITER_FOR_LARGE_TIMESTEPS

Limits the maximum hydrodynamical acceleration due to the artificial viscosity such that the viscosity term cannot change the sign of the relative velocity projected on the particle distance vector. This should not be necessary if small enough timestepping is chosen.

SPH kernel options

CUBIC_SPLINE_KERNEL

Enables the cubic spline kernel as defined in Springel et al. (2001).

WENDLAND_C2_KERNEL

Enables the Wendland C2 kernel as discussed in Dehnen & Aly (2012).

WENDLAND_C4_KERNEL

Enables the Wendland C4 kernel as discussed in Dehnen & Aly (2012).

WENDLAND_C6_KERNEL

Enables the Wendland C6 kernel as discussed in Dehnen & Aly (2012).

WENDLAND_BIAS_CORRECTION

Reduces the self contribution for Wendland kernels following Dehnen & Aly (2012), their equations (18) and (19). Only works in 3D.

SPH viscosity options

TIMEDEP_ART_VISC

Enables-time dependent viscosity.

HIGH_ART_VISC_START

Start with high rather than low viscosity.

NO_SHEAR_VISCOSITY_LIMITER

Turns of the shear viscosity suppression.

Extra physics

COOLING

Enables radiative cooling based on the collisional ionization equilibrium in the presence of a spatially constant but time-variable UV background. The network computed is similar to that

described in the paper by Katz et al. (1996). Note that metal-line cooling is not included in this module.

STARFORMATION

If this is enabled, the code can create new star particles out of SPH particles. The default star formation model that is implemented corresponds to a basic varient of the sub-resolution multi-phase model described in Springel & Hernquist (2003, http://adsabs.harvard.edu/abs/2003MNRAS.339..289S). By default, the new star particles are created as particle type 4, but if desired, one can also specify another type for them by setting the compile time flag `STAR_TYPE` to a different value. Make sure that a gravitational softening length is defined for the chosen type.

Time integration options

FORCE_EQUAL_TIMESTEPS

This adopts a global timestep for all particles, determined by pushing all particles down to the smallest timestep desired by any of the particles. The step size may still be variable in this case, but it is the same for all particles.

Single/double precision and data types

POSITIONS_IN_32BIT

When this is set, the internal storage of positions will be based on 32-bit unsigned integers. If single precision is used as default in the code (i.e. `DOUBLEPRECISION` is not set), then this is the default if none of the `POSITIONS_IN_XXBIT` options is selected.

POSITIONS_IN_64BIT

When this is set, the internal storage of positions will be based on 64-bit unsigned integers, otherwise on 32-bit unsigned integers. If double precision is used as default in the code (i.e. `DOUBLEPRECISION` is set to 1), then this is the default if none of the `POSITIONS_IN_XXBIT` options is selected.

POSITIONS_IN_128BIT

With this option, the internal storage of positions will be based on 128-bit unsigned integers, offering an extreme spatial dynamic range. Use of this should be only of interest in truly extreme scenarios.

DOUBLEPRECISION = 1

This makes the code store all internal particle data in double precision. Note that output files may nevertheless be written by converting the values in files to single precision, unless `OUTPUT_IN_DOUBLEPRECISION` is activated.

DOUBLEPRECISION_FFTW

If this is set, the code will use the double-precision version of FTTW and store the corresponding field values in real and complex space in double. Otherwise single precision is used throughout for this.

OUTPUT_IN_DOUBLEPRECISION

The output snapshot files will be written in double precision when this is enabled. This is helpful to avoid round-off errors when using snapshot files for a restart, but will rarely be required for scientific analysis, except perhaps for spatial coordinates, where many zoom simulations are insufficiently represented by single precision. Outputting in mixed precision, double precision for coordinates and single precision for everything else, can therefore be a useful option for these simulations to save storage space without sacrificing anything in the analysis.

ENLARGE_DYNAMIC_RANGE_IN_TIME

This extends the dynamic range of the integer timeline from 32 to 64 bit, i.e. the smallest possible timestep is approximately 2^64 times smaller than the simulated timespan instead of 2^32 times. Correspondingly, the number of timebins grows from 32 to 64 in this case.

IDS_32BIT

If this is set, the code uses 32-bit particle IDs, hence at most 2^32 particles may be used. This is the default setting.

IDS_48BIT

If this is set, the code uses 48-bit particle IDs, allowing some smaller fields to be packed into this variable before a long-word boundary is reached. At most 2^48 particles can be used.

IDS_64BIT

If this is set, the code uses 64-bit particle IDs.

USE_SINGLEPRECISION_INTERNALLY

If this is activated, internal computations are carried out in single precision instead of double precision. On some architectures (but not on all -- often current CPUs use the same number of cycles for a single and a double precision operation), this can make some of the computations faster, but comes with a loss of precision, of course. Use with extreme care.

NUMBER_OF_MPI_LISTENERS_PER_NODE = 1

For multi-node runs, normally one MPI rank is set aside per shared-memory node to asynchronously process incoming communication requests. If the shared-memory node has a very large number of cores, it might be helpful to have more than such communication processes, which can be set with this parameter. If the number of cores per shared memory node is larger

than 64, then this in fact has to be done, see also the `MAX_NUMBER_OF_RANKS_WITH_SHARED_MEMORY` option.

MAX_NUMBER_OF_RANKS_WITH_SHARED_MEMORY = 64

This sets the maximum number of MPI ranks on a node that have one MPI listener assigned to them, and can access each other via shared memory. Possible values are 32 and 64, the default being 64. If the total number of cores per node divided by `NUMBER_OF_MPI_LISTENERS_PER_NODE` is at most 32, a setting of 32 can be used to save a small amount of memory. If the total number of cores per node divided by `NUMBER_OF_MPI_LISTENERS_PER_NODE` is above 64, you need to increase `NUMBER_OF_MPI_LISTENERS_PER_NODE`.

Output/Input options

POWERSPEC_ON_OUTPUT

Creates a power spectrum measurement for every output time, i.e. for every snapshot that is created. This is meant to be used in cosmological simulations.

REDUCE_FLUSH

The code produces relatively verbose log-file messages. To make sure that they immediately appear in the log-file, a flush statement on the output stream is executed when outputting a new log message. On slow or overloaded filesystems, this can become a significant source of overhead. To avoid this, this option can be used. A log-file flush is then only done on intervals determined via the `FlushCpuTimeDiff` parameter.

OUTPUT_POTENTIAL

This will force the code to compute gravitational potentials for all particles each time a snapshot file is generated. These values are then included in the snapshot files. Note that the computation of the values of the potential incurs some computational overhead.

OUTPUT_ACCELERATION

This will include the physical gravitational acceleration of each particle in the snapshot files.

OUTPUT_TIMESTEP

This outputs the timestep used by the particles. Useful only to test whether the timestep criteria behave in the intended way.

OUTPUT_PRESSURE

This outputs values for the pressure of each SPH particle (i.e. only for particles of type 0) to the snapshot files.

OUTPUT_VELOCITY_GRADIENT

This outputs the SPH estimates of the velocity gradients to the snapshot files, separately for the vx, vy, and vz components, i.e. one gets three 3-vectors.

OUTPUT_ENTROPY

When this option is activated, the code writes the values of the entropic variable associated with each SPH particle to the snapshot files.

OUTPUT_CHANGEOFENTROPY

This outputs the rate of change of entropy for each particle. Only the dissipative change due to shock heating is normally included here, meaning that radiative changes of the entropy are not included in this field, only the viscous heating from the artificial viscosity is.

OUTPUT_DIVVEL

With this option, one can request an output of the velocity divergence for each SPH particle in the snapshot files.

OUTPUT_CURLVEL

Likewise for the curl of the velocity field of all SPH particles, which is output in snapshots when this option is activated.

OUTPUT_COOLHEAT

With this option, the actual rate of energy loss due to radiative cooling (or heating) can be output to the snapshot files. This option requires `COOLING` to be active as well.

OUTPUT_VISCOSITY_PARAMETER

With this option, one can request an output of the viscosity parameter for each SPH particle in the snapshot files. This option requires `TIMEDEP_ART_VISC` to be active as well.

OUTPUT_NON_SYNCHRONIZED_ALLOWED

If this option is activated, outputs occur precisely at the prescribed desired output times, with particles being (linearly) drifted to this time, while velocities stay at the values they had after the last kick. Otherwise (which is the default), snapshots are only written at times when all particles are synchronized, and all timesteps have been completed with closing half-step velocity kicks. The desired output times are mapped to the closest full synchronization points, to the extent possible (note that if the spacing of desired output times is finer than the largest timestep size, some desired output times may have to be skipped).

OUTPUT_VELOCITIES_IN_HALF_PRECISION

Stores particle velocities in half-precision format.

OUTPUT_ACCELERATIONS_IN_HALF_PRECISION

Stores accelerations in half-precision. To prevent a potential overflow of the values, the actually stored values are normalized by the factor 10*H*V200, with V200 = 1000 km/sec.

OUTPUT_COORDINATES_AS_INTEGERS

Does not convert the internal integer coordinations to floating point before output, but rather outputs them in integer representation. This retains more bits of information, and may give rise to better compression possibilities if the particle data is spatially ordered.

ALLOW_HDF5_COMPRESSION

When this is enabled, certain output fields in the HDF5 output are compressed when written. The compression/decompression is done on the fly, and is transparent to the user (but implies slightly slower I/O speed).

On the fly FOF groupfinder

FOF

This switch enables the friends-of-friends group finder of the code. In this case, the code will always run the FOF group finder whenever a snapshot is produced. This is done directly before the snapshot is written to disk, allowing the particle data in the snapshot files to be arranged in group order, i.e. it is easy to read the particles of any desired group. Also, when FOF is set, the code can be applied in postprocessing mode to compute group catalogues for existing snapshots. The snapshot in question is then written a second time with a name suffix "reordered", with the particle data arrange in group order.

FOF_PRIMARY_LINK_TYPES = 2

This identifies via a bitmask the particle type or types to which the friends-of-friends linking algorithm should be applied. Conventionally these are (high-resolution) dark matter particles stored in type 1, so the default for this parameter is 2.

FOF_SECONDARY_LINK_TYPES = 1+16+32

All particles of types selected by this bitmask are looking for the nearest particle from the set covered by `FOF_PRIMARY_LINK_TYPES` and are then made member of the FOF group this particle belongs to. The idea is that the groups found with the plain FOF algorithm incorporate all co-spatial particles from the set described by `FOF_SECONDARY_LINK_TYPES`. This is useful especially for hydrodynamical cosmological simulations with radiative processes and star formation. In this case, the spatial distribution of baryons becomes very different from the dark matter, so that applying the friends-of-friends method to both dark matter and baryonic particles at the same time would yield highly distorted results. It is then much cleaner to do this only for

the non-dissipative dark matter, and let these halos collect the baryons in the same halo with this option.

FOF_GROUP_MIN_LEN = 32

The minimum length a group needs to have before being stored in the group catalogue is set with this parameter. The default number is 32.

FOF_LINKLENGTH = 0.2

Dimensionless linking length for the friends of friends algorithm. The code will cast this into a comoving linking length by estimating the mean particle spacing from the dark matter density (as given by `OmegaDM - OmegaBaryon`) and the mean particle mass of all the particles selected with the `FOF_PRIMARY_LINK_TYPES` mask. It makes sense that the mass of all these particles is equal, only then the FOF algorithm is known to produce well understood results. If in doubt, check the log file for the linking length that is computed.

FOF_ALLOW_HUGE_GROUPLENGTH

Normally, the length of individual FOF groups and subhalos is restricted to reach at most 2^31 ~ 2 billion particles. If this is activated, it can be more.

Subfind

SUBFIND

When this is switched on, all identified FOF groups are subjected to a search for gravitationally bound substructures with the SUBFIND algorithm, as described in Springel et al. (2001), http://adsabs.harvard.edu/abs/2001MNRAS.328..726S. The snapshot outputs that are produced are automatically ordered in group plus subhalo order, i.e. all particles of the same group are stored subsequently, and subhalos are nested within each group, i.e. the particles are arranged in the order of the groups and subhalos.

SUBFIND_HBT

This enables an implementation of the hierarchical bound tracing algorithm, where subhalo candidates are identified with the help of a substructure catalogue from a precious time instead of doing this with density excursion sets. This option requires both `SUBFIND` and `MERGERTREE`.

SUBFIND_STORE_LOCAL_DENSITY

This will calculate local densities and velocity dispersions for *all* particles, not only for particles in FOFs, and store them in snapshot files.

SUBFIND_ORPHAN_TREATMENT

This produces special snapshot files after group catalogues are produced that contain only particles that have formerly been most bound particles of a subhalo. This can later be used by semi-analytic models of galaxy formation coupled to the merger tree output to treat temporarily lost or disrupted subhalos.

Merger tree algorithm

MERGERTREE

This options enables an on-the-fly calculation of descendant subhalos, which are then stored along with the group catalogues. This requires `FOF` and `SUBFIND` to be set, and is meant to be used in cosmological simulations with a large number of outputs. The merger tree information can then be used in a final postprocessing step to construct merger trees from all the group catalogues and descendant/progenitor links produced on the fly (or in postprocessing). The merger tree construction done in this way can in principle be carried out without having to store the actual particle data of the snapshot files.

On-the-fly lightcone creation

LIGHTCONE

When this option is enabled, particle data is output continuously as particles cross the backwards lightcone. The geometry of the lightcone (or of several lightcones) is described by a separate input file as specified in the parameterfile. If needed, also periodic replications of the box are used to fill the lightcone volume. Note that either `LIGHTCONE_PARTICLES` or `LIGHTCONE_MASSMAPS`, or both, need to be selected as well if this option is activated.

LIGHTCONE_PARTICLES

When this option is enabled, particle data is output continuously as particles cross the backwards lightcone. The geometry of the lightcone (or of several lightcones) is described by a separate input file as specified in the parameterfile. If needed, also periodic replications of the box are used to fill the lightcone volume.

LIGHTCONE_OUTPUT_ACCELERATIONS

This outputs the gravitational accelerations of particles on the lightcone, which can be used for gravitational lensing applications.

LIGHTCONE_MASSMAPS

This option creates projected mass shells along the backwards lightcone, for weak lensing applications. Requires the `LIGHTCONE` option.

LIGHTCONE_PARTICLES_GROUPS

This option runs the FOF (and SUBFIND if enabled) group finders on the lightcone particle data before they are written to disk. Requires the `LIGHTCONE` and `LIGHTCONE_PARTICLES` options.

LIGHTCONE_IMAGE_COMP_HSML_VELDISP

This special option is only relevant for lightcone image creation, and (re)computes adaptive smoothing lengths as well as local velocity dispersions.

REARRANGE_OPTION

This option needs to be enabled to allow the rearrange lightcone particle feature to work. It only needs to be on for the special postprocessing option, otherwise it can be disabled to save memory.

IC generation

NGENIC = 256

This master switch enables the creation of cosmological initial conditions for simulations with periodic boundary conditions. The value of `NGENIC` should be set to the FFT-grid size used for IC generation, which should be at least as fine as the particle resolution per dimension. If the code is started without restartflag (i.e. when normally initial conditions are read), the code instead creates the ICs first, followed by evolving them with the code, i.e. in this case the initial conditions do not need to exist on disk. One can however also start the code with restartflag 6, in which case the ICs are produced and written to disk, followed by a stop of the code. One can then also use the produced files as ICs in a regular start of GADGET-4 without having the `NGENIC` option set.

CREATE_GRID

If this is activated, the IC creation is carried out with a regular Cartesian particle grid that is produced on the fly. Otherwise, the unperturbed particle load is read in from the specified IC file, which allows the use of a so-called glass files (which arise from evolving random Poisson samples with the sign of gravity reversed until they settle in a quasi-equilibrium without preferred directions) or of spatially variable resolution.

GENERATE_GAS_IN_ICS

This option can be used to modify dark matter-only initial conditions for cosmological simulations upon start-up of the simulation code. The modification is to add gas to the simulation by splitting up dark matter particles into a dark matter and gas particle, with masses set by the specified cosmological parameters. The particle pair is displaced in opposite directions from the original coordinate keeping the center-of-mass fixed. The separation is such that the new dark matter and gas particles form two interleaved grids that maximize the relative distance between the two particle types and minimizes pairing correlations. The velocities of the particles inherit the velocity of the original dark matter particle. Note that here the transfer functions of dark matter and gas are not distinguished, so this procedure is only a rough approximation. It is however typically sufficient on large scales and for galaxy formation.

SPLIT_PARTICLE_TYPE = 4+8

This bitmask determines which of the dark matter particles contained in the dark matter only initial conditions that are processed with `GENERATE_GAS_IN_ICS` should be split up into gas and dark matter particles. Normally, this should be done for all of the dark matter particles to produce a volume filling gas phase. However, sometimes this is restricted to the high-resolution region in zoom simulations, which can then be picked out with this option.

NGENIC_FIX_MODE_AMPLITUDES

When activated, this leaves the mode amplitudes at sqrt[P(k)], instead of sampling them from a Rayleigh distribution. This can be useful in the context of the variance suppression technique of Angulo & Pontzen (2016).

NGENIC_MIRROR_PHASES

If this is activated, all phases in the created realization are turned by 180 degrees. This is useful to realize a pair of simulations that differ only by the sign of the initial density perturbations but which are otherwise identical.

NGENIC_2LPT

This option creates the initial conditions based on second-order Lagrangian perturbation theory, instead of just using the Zeldovich approximation. Especially when the starting redshift is low, this option is recommended.

NGENIC_TEST

This option is purely for testing purposes. When the code creates ICs on the fly, it just measures the power spectrum of the produced ICs and terminates.

MPI related settings

MPI_MESSAGE_SIZELIMIT_IN_MB = 200

Some (especially older) MPI libraries are not overly stable when very large transfers are done. Such transfers can however happen in GADGET-4 for large simulations, for example in the domain decomposition. With this option one can ask the code to automatically split up such large transfers in a sequence of smaller transfers. The maximum allowed size of one of the transfers in MB is set by the value given to `MPI_MESSAGE_SIZELIMIT_IN_MB`.

NUMPART_PER_TASK_LARGE

Set this if the number of particles per task is quite large, in particular so large than 8 times this number can overflow a 32-bit integer. This means that once you expect ~500 million or more particles on a single MPI rank, this option needs to be set to guarantee that the PM algorithms still work correctly. Of course, once you reach more than 2 billion particles per MPI rank, the

code will stop working anyhow due to integer overflows. The easy solution to this, of course, is to increase the number of MPI ranks.

ISEND_IRECV_IN_DOMAIN

This option can be used to replace the default communication pattern used in the domain decomposition (and also in FOF and SUBFIND) which is based on a hypercube with synchronous MPI_Sendrecv() calls, with a bunch of asynchronous communications. This should be faster in principle, but it also tends to result in a huge number of simultaneously open communication requests which can also choke the MPI communication subsystem. Whether this works robustly and is indeed faster will depend on the system and the simulation size. If in doubt, rather stick with the default algorithm.

USE_MPIALLTOALLV_IN_DOMAINDECOMP

Another approach to carry out the all-to-all communication occurring in the domain decomposition is to simply use MPI's Alltoallv function. This is done when this option is set, and one then effectively hopes that the internal algorithm used by Alltoallv is the most robust and fastest for the communication task at hand. This may be the case, but there is no guarantee for it. The default algorithm of GADGET-4 (hypercube with synchronous MPI_Sendrecv), which is used when this option is not used, should always be a reliable alternative, however.

MPI_HYPERCUBE_ALLGATHERV

Another issue with some MPI-libraries is that they may use quite a bit of internal storage for carrying out MPI_Allgatherv. If this turns out to be a problem, one can set this option. The code will then replace all uses of MPI_Allgatherv() with a simpler communication pattern that uses hypercubes with MPI_Sendrecv as a work-around.

Testing and Debugging options

DEBUG

This option is only meant to enable core-dumps (which are typically disabled by calling MPI_Init() on program start-up). This can be useful to allow post-mortem analysis of a crash by loading the core file with a debugger. Of course, the code should be compiled with symbols included (-g option) to facilitate this, and it may also help to set the optimization level to something low or disable optimizations entirely to avoid confusing the debugger in some situations.

DEBUG_ENABLE_FPU_EXCEPTIONS

This option is useful in combination with DEBUG and tries to enable FPU exceptions. In this case, an illegal mathematical floating point instruction that creates a dreaded "Not a Number" (NaN) will trigger a core file. With the debugger one can then quickly find the line in the code that is the culprit.

DEBUG_SYMTENSORS

This option executes a few selected unit tests on the symmetric-tensor subroutines on start-up of the code.

HOST_MEMORY_REPORTING

This option reports, when the code starts, available system memory information by analyzing / proc/meminfo on Linux systems. It is enabled by default on Linux. Output of this option is found at the beginning of the stdout log-file, and for example looks like this:

```
------------------------------------------------------------------------------
AvailMem:        Largest =   29230.83 Mb (on task=  24), Smallest =   2920
Total Mem:       Largest =   32213.01 Mb (on task=   0), Smallest =   3221
Committed_AS:    Largest =    3011.91 Mb (on task=  12), Smallest =    298
SwapTotal:       Largest =   23436.99 Mb (on task=   0), Smallest =   2343
SwapFree:        Largest =   22588.00 Mb (on task=   0), Smallest =   2160
AllocMem:        Largest =    3011.91 Mb (on task=  12), Smallest =    298
------------------------------------------------------------------------------
Task=0 has the maximum committed memory and is host: sandy-022
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

ENABLE_HEALTHTEST

When this is enabled, the code tries to assess upon start-up whether all CPU cores are freely available and show the same execution speed. Also, the MPI bandwidth both inside nodes and between nodes is tested.

FORCETEST = 0.001

Calculates for the specified fraction of particles direct summation forces, which can then be compared to the forces computed by the Tree/PM/FMM algorithms of GADGET-4 in order to check or monitor the force accuracy of the code. This is only included as a testing and debugging option. The value of the option should be set to a number between 0 and 1 (e.g. 0.001), and this number gives the fraction of randomly chosen particles at each timestep for which forces by direct summation are computed. The normal tree-forces and the exact direct summation forces are then collected in a file `forcetest.txt` for later inspection. Note that the simulation itself is unaffected by this option, but it will of course run much(!) slower, particularly if `FORCETEST * NumPart * NumPart >> NumPart` Note: The particle IDs must be set to numbers >= 1 for this option to work.

FORCETEST_TESTFORCELAW = 1

Special option for measuring the effective force law. Can be set to 1 or 2 for checking with TreePM/FMM-PM, or TreePM/FMM-PM + PLACEHIGHRESREGION. The option `FORCETEST` must be activated as well. The simulation needs to be fed with a special initial conditions file for which only one particle has mass, the others are massless test particles. The code will then go through cycles in which the particle with the mass is randomly placed, and the other particles are randomly placed around it, with distance spacing uniform in log(r). After 40 cycles are carried out, the code terminates, and the force-law accuracy can be examined by analysing the file `forcetest.txt` .

FORCETEST_FIXEDPARTICLESET

This always checks the same particle IDs if force accuracy is checked during a run.

VTUNE_INSTRUMENT

This option creates additional instrumentation instructions for the Intel VTune code performance tool, based on the internal timing routines. This can be used for a performance analysis based on this tool.

DEBUG_MD5

This option can be used to compute MD5 checksums of the P[] and SphP[] arrays regularly in the code, with the results being written to the log-file memory.txt. Using this, one can check for binary invariance of the code when the code is interrupted and resumed from restart files.

TILING = 2

Replicates the read-in ICs the specified number of times in each dimension. This can be used for scaling tests.

SQUASH_TEST

Squeezes the ICs on read-in on order to create a distortion from spherical symmetry in certain force calculation tests.

DOMAIN_SPECIAL_CHECK

Outputs test data to check the balancing algorithms.

EWALD_TEST

A development test for testing the accuracy of the Ewald table lookup.

RECREATE_UNIQUE_IDS

This option can be used to reinitialize the particle IDs upon start-up. Useful if one has to deal with a broken IC file.

NO_STOP_BELOW_MINTIMESTEP

Do not stop when the code wants to adopt a timestep below the specified minimum timestep, but rather enforce this step size.

DO_NOT_PRODUCE_BIG_OUTPUT

This special option allows one to refrain from writing large output files (restart files, snapshots, and group catalogues), which can be useful for scaling tests.

STOP_AFTER_STEP = 8

After the corresponding step has been completed, the simulation ends. This is meant to simplify certain performance and scalability tests.

MEASURE_TOTAL_MOMENTUM

This option computes the total conjugate momentum after every step. Can be used to check for manifest momentum conservation of different force computation schemes.

TREE_NO_SAFETY_BOX

When enabled, this disables the geometric 'near node' protection, i.e. for the one-sided tree, one may then be closer to a node's center than 1.5 times the node size, and for FMM, adjacent nodes may interact.

Parameterfile

Many features of GADGET-4 are controlled by a parameterfile that has to be specified whenever the code is started. Each parameter value is set by specifying a keyword, followed by a numerical value or a character string, separated by whitespace (spaces, tabs). For each keyword, a separate line needs to be used, with the value and keyword appearing on the same line. Between keywords, arbitrary amounts of whitespace (including empty lines) may be used. The order in which the keywords are specified is arbitrary, but each keyword needs to appear exactly once, even if its value is not relevant for the particular simulation (otherwise you'll get an error message). Note that the keywords are type-sensitive.

Lines with a leading '%' or '#' are ignored. In lines with keywords, comments may also be added after the specification of the value for the corresponding keyword, and in this case do not need to begin with any special character.

In the following, each keyword and the meaning of its value are discussed in detail, and typical example values are provided as an illustration. Some keywords may be changed during a run, i.e. changes of the corresponding values will be taken into account upon resuming the code from restartfiles whenever this is reasonable, but changes of certain other parameter values will be ignored. For example, while changing the memory- or cpu-time limit specified for the code is always possible, changing the cosmological parameters in the middle of a run will be prevented. If a change in the middle of a run is accepted by the code, this will also be reflected in the log messages when starting the code. Note, however, that you normally do not need to make any changes in the parameterfile when you restart a run from restart-files.

Filenames and file formats

OutputDir /home/volker/galaxy_collision

This is the pathname of the directory that holds all the output generated by the simulation (snapshot files, restart files, diagnostic files). The code will try to create this directory if it does not yet exist, but the directory's parent directory needs to exist otherwise an error message will be produced.

SnapshotFileBase snapshot

From this string, the name of the snapshot file is derived by adding an underscore, and the number of the snapshot in a 3-digits format. If `NumFilesPerSnapshot > 1`, each snapshot is distributed into several files, with a group of processors writing their data to one of the files (these files can be written concurrently). In this case, the filenames are supplemented with a tailing `.n`, where `n` designates the rank of the file in the group of files representing the snapshot. If the HDF5 file format is used, an identifier `.hdf5` is also appended automatically.

SnapFormat 3

A flag that specifies the file-format to be used for writing snapshot files. A value of 1 selects the simple legacy binary file-format of GADGET-1/2/3, while a value of 2 selects a more convenient variant of this simple binary format (which has been available from GADGET-2 onwards). A value of 3 selects the use of HDF5 instead, which is the strongly recommended format. This is because this data format allows a simple browsing of its contents, access to individual data items is easily possible through the name of the data set, conversions between endianness and single/double precision are automatically done if needed, and pesky I/O errors (like reading too many items from a given data set) are detected reliably. HDF5 output has been introduced for snapshots already in GADGET-2/3, but in GADGET-4, it is now also available for group catalogues, merger trees, light-cone data, etc., and this parameter also regulates the file format of these outputs. If HDF5 is selected, the filenames will be appended automatically with a `.hdf5` suffix. The structure of the snapshot files and of other outputs is discussed in a separate part of the documentation.

ICFormat 1

This flag selects the file format of the initial conditions read in by the code upon start-up. The possible format choices are the same as for `SnapFormat`. It is therefore possible to use different formats for the initial conditions and the produced snapshot files. In case your initial conditions is in a different file format, we recommend that you convert your IC files to one of the three formats supported by GADGET-4 (preferably HDF5). Alternatively, you could incorporate a customised reading routine directly into the GADGET-4 code, but this requires an intimate understanding of the internal workings of the code.

InitCondFile /home/volker/ICs/galaxy.dat

This sets the filename of the initial conditions to be read in at start-up. Note that the initial conditions file (or files) does not have to reside in the output directory. The initial conditions can be distributed into several files, in the same way as snapshot files. In this case, only the basename without the tailing `.n` number should be specified as initial conditions filename. Likewise the `.hdf5` file-name suffix should be omitted if HDF5 is used. The code will recognise the number of files that make up the initial conditions from the file header entries, and load all of these files accordingly. There is no constraint on the number of these files in relation to the processor number used.

NumFilesPerSnapshot 2

The code can distribute each snapshot onto several files. This leads to files that are easier to handle in case the simulation is very large, and also speeds up I/O, because these files can then be written or read in parallel. The number of processors should be equal or larger than `NumFilesPerSnapshot` , because each snapshot file will hold the data of a group of processors (otherwise the code reduces the value to the number of MPI ranks used). Optimum I/O throughput is reached if the number of processors is equal to, or a multiple of `NumFilesPerSnapshot` , and if `MaxFilesWithConcurrentIO` is reasonably large. With the setting `NumFilesPerSnapshot=1` it is possible to write all particle data in just one snapshot file but then no parallel I/O is used.

MaxFilesWithConcurrentIO 8

This sets the number of concurrent I/O operations the code is allowed to carry out. If 0 is specified, the code adopts a value equal to the number of MPI ranks, and the same is done if the specified value is larger than the number of MPI rank. However, it can sometimes be sensible to limit this to a smaller number, especially on very large MPI partitions in order to prevent that the I/O subsystem is overloaded. Whether or not this is an issue can be found out by starting the code with restartflag 9, which carries out a special I/O bandwidth test where MaxFilesWithConcurrentIO is systematically varied from the number of MPI-ranks down to 1 by factors of 2, and in each case a write-test is performed using parallel I/O from all ranks. This can be used to determine a reasonable setting for MaxFilesWithConcurrentIO that is not causing choking of the filesystem.

CPU-time limit and restarts

TimeLimitCPU 40000.0

This is the wallclock time limit for the current execution of the code, in seconds. Often the code will be run through a submission to a computing queue, and hence this value should be matched to the corresponding time limit of the computing queue or job submission script, as appropriate. The run will automatically interrupt itself and write restart files if 85% of this time has elapsed. The extra 15% is introduced to make sure that there is always enough time left to safely finish the current time step (or FOF/SUBFIND group finding) and for writing the restart files before the time limit is reached. Note that this time refers to the wall-clock time on one processor only. The total CPU time consumed by the code is obtained by multiplying with the total number of cores that are used/occupied by the run.

CpuTimeBetRestartFile 7200

This is the maximum amount of wall-clock time, in seconds, that may elapse before the code writes a new set of restart file for regular checkpointing. With this parameter the code can hence be asked to write a restart file every once in a while. This is meant to provide some protection against hardware or software failures, in which case one can resume a simulation from the last set of restart files. In the above example, a restart file would be written automatically every 2 hours, so that the lost time in case of such an issue would be at most 2 hours of computing. The old set of restart files is renamed into a set of `bak-restart.X` files before the new files are written, so that there is some protection against a crash during the writing of the restart files themselves. The latter could happen, for example, because of a disk-full error. It is not possible to resume a

simulation from a corrupted set of restart files, or with restart files that correspond to a mix of different output times. In case the code should really crash while writing restart files, it is best to discard all `restart.X` files, and then to rename the `bak-restart.X` files into `restart.X` files (where `X` runs from 0 to the number of MPI ranks minus 1), for example with the command:

```
rename bak-restart restart bak-restart.*
```

Note that depending on your system, the convenient rename command may have a slightly different syntax or may not be available at all (consult the man pages).

FlushCpuTimeDiff 120

The GADGET-4 code provides quite verbose output in its log-files, with each timestep producing some entries. If you carry out a simulation with a very large number of very short timesteps and you have a slow or busy filesystem, this I/O can slow down the code if the filesystem buffers are flushed to disk after every output. With this parameter, the flush operations are only carried out with a reduced frequency, in the above example every 120 seconds. This should avoid any significant cost of this I/O, but it also means that you may not see right away what the code has been doing when inspecting the log-files, because the information in these files will typically only be updated when a new flush operation is triggered by the code.

Memory allocation

MaxMemSize 2000

This value gives the maximum amount of memory (in MByte) the code is allowed to use per MPI process. The code will strictly enforce this limit, and terminate if a higher use is attempted by the code. If this should occur, a table with the memory allocated for different code parts is output to the log file, together with information in which line of the code each allocation has happened. Note that the code will automatically try to make good use (in communication phases) of any extra memory you specify here. It is therefore a good idea to set `MaxMemSize` to something close to the amount of physical memory that can be used per MPI rank on the target compute nodes. The code will check whether the memory on the target machines is actually sufficient for the setting chosen, and otherwise terminate (this check only works on Linux). This check should safely prevent that you accidentally run the code with a too high memory use that could make a compute-node start swapping. (Driving a node into swapping is generally a really bad thing and would lead to dismal performance and/or node crashes. Most HPC systems prevent this nowadays at the system level for their compute nodes.) If you want to find out the smallest amount of memory a given simulation would have needed to complete, you can grep the log-file `memory.txt` for the values reported behind the phrase "Largest Allocation Without Generic". This gives the maximum that was needed by any of the MPI ranks over the course of the simulation and corresponds to the lower possible limit for `MaxMemSize` for the given run minus a small amount of communication buffer space that you have to allow the code to use. So in practice, the setting for MaxMemSize needs to be at least slightly larger than the "Largest Allocation Without Generic" value reported in `memory.txt`.

Simulated time span and spatial extent

TimeBegin 0

This initialises the time variable of the simulation when a run is started from initial conditions (in internal units). If comoving integration is selected (`ComovingIntegrationOn=1`), the time variable is the dimensionless expansion factor `a` itself, i.e. `TimeBegin = a = 1 / (1 + z_start)` , otherwise it is simply physical time in the internal system of units of the code.

TimeMax 3.0

This marks the end of the simulation. The simulation will run up to this point, then write a restart-file and a snapshot file corresponding to this time (even if the time `TimeMax` is not in the normal sequence of snapshot files). If `TimeMax` is increased later on, the simulation can be simply continued from the last restart-file. Note that this last snapshot file will then be overwritten in case this was a special dump out of the normally expected output sequence. For comoving integrations, the time variable is the expansion factor, e.g. `TimeMax=1.0` will stop the simulation at redshift `z=0` . Otherwise the value of `TimeMax` refers to physical time.

ComovingIntegrationOn 0

This flag enables or disables comoving integration in an expanding universe. For `ComovingIntegrationOn=0` , the code uses plain Newtonian physics with vacuum or periodic boundary conditions. Time, positions, velocities, and masses are measured in the internal system of units, as specified by the selected system of units. For `ComovingIntegrationOn=1` , the integration is carried out in an expanding universe, using a cosmological model as specified by `Omega0` , `OmegaLambda` , etc. In this cosmological mode, coordinates are comoving, and the time variable is the natural logarithm of the expansion factor itself. If the code has not been compiled with the `PERIODIC` makefile option, the underlying model makes use of vacuum boundary conditions, i.e. density fluctuations outside the particle distribution are assumed to be zero. This requires that your particle distribution represents a spherical region of space around the origin. If `PERIODIC` is enabled, the code expects the particle coordinates to lie in the interval `[0, BoxSize]`.

BoxSize 10000.0

The size of the periodic box (in code units) encompassing the simulation volume. This parameter is only relevant if the `PERIODIC` option is activated.

Cosmological parameters

Omega0 0.3

Cosmological matter density parameter in units of the critical density at `z=0`. Relevant only for comoving integration.

OmegaLambda 0.7

Cosmological vacuum energy density (cosmological constant) in units of the critical density at `z=0`. For a geometrically flat universe, one has `Omega0 + OmegaLambda = 1`. Important: For simulations in Newtonian space that do not account for cosmological expansion, this parameter has to be set to zero.

OmegaBaryon 0.04

Baryon density in units of the critical density at `z=0`. This is not explicitly used in the time integration of GADGET-4, but the parameter is relevant when initial conditions are created, or when dark matter-only initial conditions are outfitted with gas particles upon code start-up with the `GENERATE_GAS_IN_ICS` option.

HubbleParam 0.7

This dimensionless parameter enters the definition of GADGET's system of units, and can be used to eliminate an explicit dependence on the value of the Hubble constant, like it has been traditionally done in cosmology. Most often, the value of `HubbleParam` is chosen to express the value of the Hubble constant in units of 100 km/s/Mpc. While the value of `HubbleParam` is not needed (in fact, it does not enter the computations at all) in purely collisionless simulations, the value of `HubbleParam` is still relevant when conversions to physical cgs units are required, for example to compute rate equations in radiative cooling physics.

Hubble 100.0

Value of the Hubble constant in internal units. Since the internal units contain a factor `HubbleParam`, one can basically choose whether one wants to set the Hubble constant via `HubbleParam` (then `Hubble` has the same value in all simulations, even if the cosmological factors and the Hubble constant change), or one sets `HubbleParam` to unity and uses `Hubble` to directly set the Hubble constant. Both is possible, and intermediate forms in principle as well.

System of units

UnitLength_in_cm 3.085678e21

This sets the internal length unit in cm/h, where H_0 = 100 h km/s/Mpc. The above choice is convenient for cosmology, as it sets the length unit to 1.0 kpc/h.

UnitMass_in_g 1.989e43

This sets the internal mass unit in g/h, where H_0 = 100 h km/s/Mpc. The above choice is convenient for cosmology, as it sets the mass unit to 10^10 M_sun/h.

UnitVelocity_in_cm_per_s 1.0e5

This sets the internal velocity unit in cm/sec. The above choice corresponds to a velocity unit of km/sec, which is the commonly used and most convenient unit in cosmology. Note that the specification of `UnitLength_in_cm` , `UnitMass_in_g` and `UnitVelocity_in_cm_per_s` also determines the internal unit of time. The definitions made above imply that in internal units the Hubble constant has a numerical value independent of h (where h is given by `HubbleParam`). For the numerical examples above, the Hubble constant has always the value 0.1 in internal units, independent of h, and the Hubble time is always 10.0 in internal units, with one internal time unit corresponding to 9.8 x 10^8 yr/h. However, of course, you are free to choose a different system of units if you like. Note again that this implies that for

purely gravitational dynamics, the code will not need to know the value of h at all. `HubbleParam` is nevertheless kept in the parameterfile because additional physics in the hydrodynamical sector may require it.

GravityConstantInternal 0

The numerical value of the gravitational constant G in internal units depends on the system of units you choose. For example, for the numerical choices made above, the physical value of G corresponds to G=43007.1 in internal units. For `GravityConstantInternal=0` (the normal choice for cosmological simulations), the code calculates the internal value corresponding to the physical value of G automatically. But sometimes, you might want to set G yourself. For example, in scale-free test simulations, specifying `GravityConstantInternal=1`, `UnitLength_in_cm=1`, `UnitMass_in_g=1`, and `UnitVelocity_in_cm_per_s=1`, yields a natural system of units in which one may also want to adopt `G=1` as well, which can then be achieved by specifying a non-zero value for `GravityConstantInternal`, in this example `GravityConstantInternal=1`.

Gravitational force accuracy

TypeOfOpeningCriterion 0

This selects the type of cell-opening criterion used in the tree walks for computing gravitational forces. A value of 0 results in a geometric opening criterion which is primarily governed by the opening angle theta, while 1 selects a relative criterion that tries to limit the absolute truncation error of the multipole expansion for every particle-cell or cell-cell interaction. The latter scheme usually gives slightly higher accuracy at a comparable level of computational cost compared with the geometric criterion. When it becomes more demanding to calculate accurate forces due to strong cancellation effects (such as at high redshift with nearly uniform mass distribution), the relative criterion automatically invests more effort because of the small residual forces there. This adaptivity makes it advantageous especially for cosmological simulations, where a single value of theta for the geometric criterion is not ideal at all redshifts.

ErrTolTheta 0.7

This is the accuracy criterion parameter (the opening angle theta) of the tree algorithm if the geometric opening criterion (i.e. `TypeOfOpeningCriterion=0` is used). If `TypeOfOpeningCriterion=1` is adopted, then theta and the geometric opening criterion are only used for a first force computation whose purpose is only to provide an estimate for the current acceleration of each particle, which in turn is needed to compute forces with the relative cell opening criterion. Hence, theta needs to be set to a sensible value even if the relative criterion is used and only forces computed with the relative criterion enter the dynamics.

ErrTolForceAcc 0.005

This controls the accuracy of the relative cell-opening criterion (if enabled). Here, alpha is given by `ErrTolForceAcc`. Note that independent of this relative criterion, the code will always open nodes if the point of reference lies within a geometric boundary box around the cubical cell (unless `TREE_NO_SAFETY_BOX` is enabled). This protects against the possibility of an occurrence of unusually large force errors for very particular particle configurations.

ErrTolThetaMax 1.0

When the relative opening criterion is used, the effective opening angle allowed for a node of little mass may grow very large, possibly approaching the convergence radius of the multipole expansion. To protect against the possibility to get unexpectedly large errors from this, the maximum allowed geometric opening angle can be limited with this parameter.

ActivePartFracForPMinsteadOfEwald 0.1

This parameter is only needed when the TreePM scheme is used in combination with the `TREEPM_NOTIMESPLIT` option. Then the time integration does not distinguish between a long range and a short range force, instead the total force is integrated with a single (variable) timestep. The TreePM method here only functions as a method for accelerating the computation of the total force, with the alternative being to do it with a pure tree calculation (if needed with Ewald correction for periodic boundaries). If only a small number of particles is active, doing the force calculation as a pure tree can be faster than doing it with the TreePM approach, because for the PM part always full FFTs have to be computed independent of the number of active particles. This parameter is used to specify a threshold above which the active fraction needs to lie before TreePM is applied for a given timestep, otherwise the force calculation is done with a pure tree. The same applies to FMM-PM and pure FMM. In principle, this should only affect the performance of the calculation, not its accuracy in any significant way (this is true in the limit when the TreePM and pure Tree force errors are comparable in magnitude, and are both negligible).

Time integration accuracy

MaxSizeTimestep 0.01

This parameter sets the maximum timestep a particle may take. This should be set to a sensible value in order to protect against too large timesteps for particles with very small acceleration. Usually, a few percent of the dynamical time of the system gives sufficient accuracy. For cosmological simulations, the parameter specifies the maximum allowed step in ln(a), because the natural logarithm of the scale factor is discretized for the time integration in this case. Hence, specifying the maximum allowed timestep for cosmological simulations is equivalent to specifying it as a fraction of the current Hubble time. A value of ~0.01 is usually accurate enough for most cosmological runs.

MinSizeTimestep 0

If a particle requests a timestep smaller than the specified value of this parameter, the simulation terminates with an error message. This is meant to prevent simulations from continuing when the timestep has dropped to an unreasonably small value, because such behaviour typically indicates a problem of some sort. Setting the parameter to zero disables this safety check.

ErrTolIntAccuracy 0.025

This dimensionless parameter controls the accuracy of the simple kinematical timestep criterion commonly employed in cosmological simulations, and which is also used in GADGET-4. The timestep constraint is given by dt = sqrt(2 eta epsilon/|a|), where eta=`ErrTolIntAccuracy`,

epsilon is the gravitational softening length, and a the acceleration experienced by the particle. The actual timestep taken by the particle will always be shorter than dt, as the particle will be forced onto the power-of-two hierarchy of allowed timestep sizes by reducing the step to the next available shorter step.

CourantFac 0.15

This sets the value of the Courant coefficient used in the determination of the hydrodynamical timestep of SPH particles. Note that GADGET-4's definition of the SPH smoothing length differs by a factor of 2 from that found in some part of the SPH literature. As a consequence, comparable settings of `CourantFac` may be a factor of 2 smaller in GADGET4 when compared with codes using a different convention.

Domain decomposition

ActivePartFracForNewDomainDecomp 0.01

A new domain decomposition is not necessarily determined for every single timestep. A value of `ActivePartFracForNewDomainDecomp=0.01`, for example, means that the domain decomposition is reconstructed whenever there are at least `0.01 N` particles active at the current synchronization time, where `N` is the total particle number. Note that the gravitational tree is always reconstructed in every step, whereas the neighbor search tree is only reconstructed in case a domain decomposition is done for the current step. Otherwise it expands its nodes as needed to accommodate all the SPH particles that were grouped into each node.

TopNodeFactor 2.5

The domain decomposition involves the construction of a coarse oct-tree whose leaf nodes tessellate the simulation volume. The `TopNodeFactor` regulates how fine this top-level tree gets (it is designated as f_top in the code paper). The code will roughly produce `TopNodeFactor * NTask * f_mult` leaf-nodes in the top-level tree, focusing always on refining the most loaded one first until the desired fineness is reached. Here f_mult refers to the number of different cost categories that are balanced simultaneously. Since a tree node can only be assigned in full to individual domains, this parameter influences the level of discreteness fluctuations present in the load among the set of multiple domains that are mapped to individual MPI ranks. A value of a few for this parameter is usually good enough. If a very large value is adopted, the top-level tree (which is identically stored on all nodes) may get very large, making its memory use and construction time costly.

Output frequency

OutputListOn 0

A value of 1 signals that the output times are given in the file specified by `OutputListFilename` Otherwise, the output times are generated automatically in the way described below. We note that the code will only generate snapshot files if full timesteps have been finished (this is different from GADGET-2) and thus the full system is synchronized in time. This means that in GADGET-4, each particle has finished an integer number of full KDK

timesteps when stored in snapshots. Desired output times are given either in the file with output times or are created in a regularly spaced way (as described below). The corresponding desired output times will always be mapped to the closest available output time. The set of these available output times is basically given by the simulation timespan divided by maximum used time step size. The mapping then means that the actual output time of a snapshot can deviate from the desired output at most by 0.5 times the maximum timestep actually used in the simulation when the output occurs. If you want to have many outputs with very fine spacing, it makes sense to set MaxSizeTiStep sufficiently small, in particular smaller than the desired output spacing otherwise the number of created snapshots could be lower than desired in case the simulation takes timesteps for some particles that are larger than the desired output spacing.

OutputListFilename output_times.txt

This specifies the name of a file that contains a list of desired output times. If `OutputListOn` is set to 1, this list will determine the times when snapshot-files are desired. The file given by `OutputListFilename` should just contain the floating point values of the desired output times in plain ASCII format. The times do not have to be ordered in time, but there may be at most 1100 values (this is the default, but it can be enlarged if desired by setting the `MAXLEN_OUTPUTLIST` contant in Config.sh). Output times that are in the past relative to the current simulation time will always be ignored.

TimeOfFirstSnapshot 0.047619048

This variable selects the time for the first snapshot (relevant only if `OutputListOn=0`). For comoving integration, the above choice would therefore produce the first dump at redshift `z=20`.

TimeBetSnapshot 1.0627825

If `OutputListOn=1` this parameter is ignored. Otherwise, after a snapshot has been written, the time for the next snapshot is determined by either adding `TimeBetSnapshot` to `TimeOfFirstSnapshot`, or by multiplying `TimeOfFirstSnapshot` with `TimeBetSnapshot`. The latter is done for comoving integration, and will hence lead to a series of outputs that are equally spaced in ln(a). The above example steps down to redshift `z=0` in 50 logarithmically spaced steps.

TimeBetStatistics 0.1

This determines the interval of time between two subsequent computations of the total potential energy of the system. This information is then written to the file `energy.txt`, together with information about the kinetic energies of the different particle types. A first energy statistics is always produced at the start of the simulation at time `TimeBegin`.

SPH parameters

DesNumNgb 64

This is the desired number of SPH smoothing neighbours. Normally, the effective number of neighbours (defined as the mass inside the kernel divided by the particle mass) is kept constant very close to this value. Should it ever try to get outside a range +/- `MaxNumNgbDeviation`

from DesNumNgb, the code will readjust the smoothing length such that the number of neighbours is again in this range.

MaxNumNgbDeviation 2

This sets the allowed variation of the number of neighbours around the target value DesNumNgb.

InitGasTemp 10000

This sets the initial gas temperature in Kelvin when initial conditions are read. However, the gas temperature is only set to a certain temperature if InitGasTemp > 0 and if at the same time the temperature of the gas particles in the initial conditions file was found to be zero, otherwise the initial gas temperature is left at the value stored in the IC file. If the temperature is set through this parameter, and if it is below 10^4 K, a mean molecular weight corresponding to neutral gas of primordial abundance is assumed, otherwise complete ionisation is assumed.

ArtBulkViscConst 1.0

This sets the value of the artificial viscosity parameter alpha_visc used by GADGET-4. See code paper for details.

ViscosityAlphaMin

This sets the minimum value of the artificial viscosity parameter when a time-dependent viscosity is enabled.

Gravitational softening

The code distinguishes between different particle types. As far as gravity is concerned, all the types are treated equally by the code. The particles of the first type (type=0) are treated as SPH particles and receive an additional hydrodynamic acceleration from pressure gradients. The concept of particle types is primarily introduced to simplify analysis and to give certain particles an easily identifiable role.

The default number of types is NTYPES=6, which was also used as a fixed setting in GADGET-1/2/3. There, the six particle types were referred to with the symbolic tags "Gas", "Halo", "Disk", "Bulge", "Stars", and "Bndry", in this order, but these names are now dropped in favour of just numerical type specifiers. The number of available types can be enlarged or reduced if needed, but a value equal to 6 needs to be used if backwards compatibility to the format of older versions of GADGET is desired.

Normally, each particle type is mapped to a certain gravitational softening length. The number of available different softening lengths is given by NSOFTCLASSES, and does not necessarily have to be equal to NTYPES (this is however the default).

SofteningClassOfPartType0 0

Specifies the softening class that should be assigned to particle type=0. Depending on the setting of NTYPES, additional such parameters are needed, one for each particle type. One hence needs to specify SofteningClassOfPartType0, SofteningClassOfPartType1, ..., SofteningClassOfPartTypeX, where X = NTYPES - 1. The values that are assigned to these parameters need to be in the range [0, NSOFTCLASSES - 1]. It is allowed to map several particle types to the same softening class.

SofteningComovingClass0 0.5

This specifies the (comoving) softening length of the first softening class. Depending on the setting of NSOFTCLASSES, additional such parameters are needed, one for each softening class. One hence needs to specify SofteningComovingType0, SofteningComovingType1, ..., SofteningComovingTypeX, where X = NSOFTCLASSES - 1.

Gravity is softened with a spline kernel in GADGET-4, as outlined in the code paper. The softenings quoted here all refer to epsilon, the equivalent Plummer softening length. Note that for the spline that is used, the force will be exactly Newtonian beyond r = 2.8 epsilon, and the potential of a point mass m at zero lag is phi(0) = -G*m/epsilon. The softening lengths are given in internal length units. For comoving integration, the softening refers to the one employed in comoving coordinates, which usually stays fixed during the simulation.

SofteningMaxPhysClass0 0.5

In cosmological simulations, one sometimes wants to start a simulation with a softening epsilon_com that is fixed in comoving coordinates (where the physical softening, epsilon_phys = a * epsilon_com , then grows proportional to the scale factor a), but at a certain redshift one may want to freeze the resulting growth of the physical softening epsilon_phys at a certain maximum value. These maximum softening lengths are specified by the SofteningMaxPhysClassX parameters. In the actual implementation, the code uses epsilon_com = min(epsilon_com, epsilon_phys^max / a) as comoving softening. Note that this feature is only enabled for ComovingIntegrationOn=1, otherwise the SofteningMaxPhysClassX values are ignored. The specific parameter SofteningMaxPhysClass0 specifies the maximum physical softening of the first softening class. Depending on the setting of NSOFTCLASSES, additional such parameters are needed, one for each softening class. One hence needs to specify SofteningMaxPhysClass0, SofteningMaxPhysClass1, ..., SofteningMaxPhysClassX, where X = NSOFTCLASSES - 1.

GasSoftFactor 1.5

This parameter is only needed if ADAPTIVE_HYDRO_SOFTENING is activated. In this case, the gravitational softening for the gas particles is individually selected based on their smoothing length from a logarithmic table of available softening classes. The organization of this table is described by the parameters below. In practice, the smoothing length of the gas particle is multiplied by GasSoftFactor and then the closest softening (in terms of smallest difference in the log of the softenings) from the table is assigned as the softening class of the gas particle.

MinimumComovingHydroSoftening 0.001

Specifies the smallest allowed gaseous softening value. Together with the multiplicative factor AdaptiveHydroSofteningSpacing that describes the increase from step to step, this

defines the discrete table of available softenings for SPH particles when ADAPTIVE_HYDRO_SOFTENING is used.

AdaptiveHydroSofteningSpacing 1.05

This parameter defines the spacing between two adjacent available SPH softening classes when ADAPTIVE_HYDRO_SOFTENING is enabled. The total number of the set of available softenings classes is given by the constant NSOFTCLASSES_HYDRO, which is normally set to 64 as default. (This can be changed, if desired, by overriding the value of this constant in Config.sh). The largest available gaseous softening length is then given by MinimumComovingHydroSoftening * AdaptiveHydroSofteningSpacing ^ (NSOFTCLASSES_HYDRO - 1).

Subfind parameters

DesLinkNgb 20

This sets the number of neighboring particles that are examined in the SUBFIND algorithm to identify locally isolated peaks in the density field, and to link overdensity candidates across saddle points. The typical value for this quantity is 20, and results of SUBFIND should be quite insensitive to the exact choice. The parameter needs only be specified if SUBFIND is actually enabled in Config.sh.

Initial conditions generation

The following initial conditions parameters are only used if NGENIC is activated in the code configuration file. The value of NGENIC is interpreted as the size of the FFT grid used to compute the displacement field. One should have NGENIC >= Nsample. The redshift of the initial conditions is the same as the defined starting redshift of the simulation, hence is given by TimeBegin.

NSample 128

This sets the maximum wave number k that the code uses, i.e. this effectively determines the Nyquist frequency that the code assumes, k_Nyquist = 2*PI / BoxSize * Nsample/ 2 Normally, one chooses Nsample such that Ntot = Nsample^3, where Ntot is the total number of particles.

GridSize 128

This parameter is only needed if CREATE_GRID is activated. In this case, the code will create the initial particle load itself in terms of a uniform Cartesian grid with particles of constant mass. The total number of particles that is used is then GridSize^3 . In cold dark matter, perturbations should then be imprinted all the way to the Nyquist frequency of this particle grid, i.e. one should pick GridSize=Nsample. If CREATE_GRID is not active, then this parameter is not present. In this case, the initial unperturbed particle load is read in as the specified IC file.

PowerSpectrumType 2

This can be used to select the parameterization of the linear theory input spectrum. For a value of 1, an analytic fitting function by Eisenstein & Hu is selected, while 2 uses a tabulated power spectrum in the file specified with `PowerSpectrumFile` A value of 3 (or any other value) will use an analytic parameterization from Efstathiou.

PowerSpectrumFile cmb_code_wmap7_spectrum.txt

This file gives a tabulated linear theory input power spectrum, which can be computed by a Boltzmann code, like CAMB, for example. The file format is ASCII, and should contain two columns, with a pair of values on every line:

log(k) log(Delta^2)

Here log is the base10 logarithm, and k is given in units of `h / cm / InputSpectrum_UnitLength_in_cm` (see below). Delta^2 refers to the dimensionless power spectrum at wavenumber k, which is related to the ordinary power spectrum P(k) through Delta^2 = 4 PI k^3 P(k).

InputSpectrum_UnitLength_in_cm 3.085678e24

Defines the length unit used in the tabulated input spectrum in cm/h. If desired, this can be chosen differently from `UnitLength_in_cm`, so that one can for example have an input spectrum table based on Mpc/h while the simulation one carries out uses kpc/h.

ShapeGamma 0.21

This parameter is only relevant when `PowerSpectrumType=3` is used (Efstathiou parameterization). In this case, `ShapeGamma` should usually be set to something close to `Omega0 * HubbelParam`.

PrimordialIndex 1.0

This may be used to specify a tilt in the primordial index, provided this has not already been taken care of in a tabulated input spectrum. Effectively, this option multiplies the spectrum (whatever the source) with an additional factor k^(PrimordialIndex-1.0), i.e. if one does not want to do this, one has to set this parameter to a value of 1. In particular, if one uses a tabulated input spectrum and this already accounts for a primordial tilt, this parameter nevertheless still has to be set to 1.0.

Sigma8 0.86

Normalization of the linear theory input power spectrum when extrapolated to z=0. As is commonly done, `sigma8` gives the rms density contrast fluctuations in top-hat spheres of radius 8 Mpc/h.

ReNormalizeInputSpectrum 1

If set to zero, the tabulated input spectrum is assumed to be correctly normalized already in its amplitude to the adopted starting redshift, otherwise the normalization is recomputed based on the specified `sigma8` value, and the linear theory growth factor for the specified cosmology. Normally, this renormalization is always recommended.

SphereMode 0

If this is activated by setting it to 1, only modes with |k| < k_Nyquist are used (i.e. a sphere in k-space), otherwise modes with |k_x|,|k_y|,|k_z| < k_Nyquist are used (i.e. a cube in k-space).

Seed 123456

An integer number that serves as seed for the random number generator used by the IC code. Because the k-modes are filled systematically "inside-out" in k-space, changing the resolution but keeping the seed the same will yield the same large-scale modes. This means that for a fixed seed, one can easily carry out resolution studies on an object-by-object basis, if desired.

Lightcone output

The lightcone output will first collect particles that cross the backwards lightcone in an intermediate storage buffer. The particle positions and velocities are extrapolated to the point of the light-cone crossing. Once the intermediate storage buffer has accumulated a certain particle number, the particles are written to disk in a light-cone file similarly to a snapshot file. After that the intermediate buffer is emptied, and gradually filled again by the continuing lightcone until the next output occurs. The individual lightcone files are thus spherical shells (or segments thereof) around the observer position. When concatenated they yield the full lightcone.

LightConeDefinitionFile lightcones.txt

When continuous light-cone outputting is activated via the switch `LIGHTCONE`, the geometry of the lightcone(s) is specified via this file. In particular, the start and end redshifts of the lightcone(s) are specified in this file. Note that the lightcone feature only makes sense for cosmological simulations with `ComovingIntegrationOn=1`. The constant `LIGHTCONE_ALLOC_FAC` (default value 0.1) determines how much storage space the code sets aside for the intermediate buffer before lightcone files are flushed to disk. This is done by prescribing the particle number that can fit into the intermediate light-cone buffer as a fraction of the total particle number. This indirectly determines on how many output files the lightcone is distributed. By overriding the constant `LIGHTCONE_ALLOC_FAC` in the `Config.sh`-file, this can be influenced if desired.

The file defining the lightcones has the following format. Each line defines a separate lightcone, and is defined at least by four numbers:

Here can be either 0, 1, 2, 3, or 4, and defines the geometric selection of the specific lightcone, according to: 0 = full sky, 1 = one octant, 2 = a pencil beam cone with circular aperture, 3 = a disk like region, 4 = a pencil beam with a square-shaped aperture.

sets the far edge of the lightcone to redshift zmax = 1/astart - 1 sets the near edge of the lightcone to redshift zend = 1/aend - 1

is normally 0. A value of 1 only makes sense combined with the SUBFIND_ORPHAN_TREATMENT option and then restricts the output to formerly most bound particles.

For type=1 (octant), an additional number with value 0, 1, ..., or 7 is needed to select the specific octant.

For type=2 (cone), three additional numbers are expected to define the principal direction vector of the cone (this does not need to be normalized), followed by its half opening angle in degrees.

For type=3 (disk), three additional number are expected that define the normal of the disk region, followed by a further number defining its comoving thickness.

For type=4 (pyramid with square base), first three additional number are expected to define the direction vector of the pencil beam, then a further vector is expected that is used to set the orientation of the x-direction of the patch of sky that is mapped by the lightcone, with the y-direction being orthogonal to that. Finally, a last number gives the half opening angle of the pyramid-shaped pencil beam in degrees.

Multiple lightcones, also of the same type, can be specified if desired. Note that the output will get extremely large if you select even a moderate redshift depth, because the code will automatically periodically replicate the simulation box as needed to cover the specified lightcone geometry.

An example for a lightcone definition file could look like this:

```
0 0    0.5     1.0
1 0    0.4     1.0     0
```

This would define a full-sky light cone from z=1 to z=0, and an octant covering positive x>0,y>0,z>0 from redshift z=1.5 to z=0.

LightConeMassMapsNside 12

The healpix Nside parameter defining the angular resolution used for the mass maps projections of lightcone particles. To enable this, the `LIGHTCONE_MASSMAPS` option needs to be set.

LightConeMassMapThickness 25

Comoving thickness of one lightcone massmap shell.

LightConeMassMapMaxRedshift 5.0

Redshift out to which the massmaps should extend.

Cooling and star formation

The following parameters refer to the simple cooling and star formation model described in Springel & Hernquist (2003, http://adsabs.harvard.edu/abs/2003MNRAS.339..289S).

MaxSfrTimescale 1.5

Gas consumption time scale in internal time units at the threshold density for star formation. This sets the parameter t_0^star in the above paper.

TempClouds 1000.0

This is given in Kelvin and corresponds to the T_c parameter (or correspondingly to u_c when expressed as thermal energy per unit mass) in the above paper.

TempSupernova 1.0e8

This is given in Kelvin and corresponds to the T_SN parameter in the above paper, or correspondingly to u_SN when expressed as thermal energy per unit mass. The relation between these two quantities is T_SN = (2/3) m u_SN / k_B, where m is the mean molecular mass and k_B the Boltzmann constant.

FactorEVP 1000.0

This is the cloud evaporation parameter A_0 in the above paper.

FactorSN 0.1

This parameter (which corresponds to the symbol beta in the above paper) is the mass fraction of short-lived massive stars (>8 Msun) formed for each initial population of stars. This depends on the adopted stellar initial mass function.

CritPhysDensity 0

The critical physical hydrogen number density in cm^(-3) above which star formation is allowed. In the model of Springel & Hernquist (2003), this is computed via eqn (23) of the this paper if the parameter is set to zero (which is the recommended setting for this model).

CritOverDensity 57.7

If a cosmological simulation would be started at very high redshift, then the physical baryon density could exceed the prescribed physical star formation threshold computed above. To prevent this, a second criterion is imposed, namely to require a minimum comoving overdensity as well, which is given in dimensionless form by this parameter. `CritOverDensity=57.7` is the extrapolated overdensity for an NFW halo at the R200 radius, i.e. this choice corresponds to requiring that star-forming gas should be contained inside the virial radius of halos. This cures the high-z problem without restricting low-redshift star formation in any way.

TreecoolFile data/TREECOOL_fg_dec11

This file is used in cosmological simulations to tabulate the time evolution of an externally imposed UV background that is responsible for cosmic reionization. Typically something like a Haard & Madau or Faucher-Giguere model is used. The file tabulates the base10 logarithm of (1+z), followed by the photoionization rates of HI, HeI and HeII, and the associated heating rates.

The simulation code will then linearly extrapolate from this table to set the UV background parameters at any given redshift. Only in cosmological simulations with comoving integration the UV background will be used, although the file is always expected if `COOLING` is enabled.

MinEgySpec 0

This parameter can be used to effectively impose a minimum allowed temperature onto the gas. This is however done in terms of a minimum energy per unit mass `u` , i.e. if `MinEgySpec` is set to some finite value, `u` will not be allowed to drop below this value.

Special features

A_StaticHQHalo 5.0

In case the `EXTERNALGRAVITY_STATICHQ` option is activated, this specifies the scale length of a static Hernquist halo whose gravitational potential is added to the force calculation. The halo is centered at the origin, and the scale length is given in internal length units.

Mass_StaticHQHalo 100.0

This parameter is only active when `EXTERNALGRAVITY_STATICHQ` is enabled, and then gives the total mass (in internal units) of the halos that is added as a static potential to the force computation.

Snapshot file format

The primary result of a simulation with GADGET-4 are snapshots, which are simply dumps of the state of the system at certain times. GADGET-4 supports parallel output by distributing a snapshot into several files, each written by a group of processors. This procedure allows an easier handling of very large simulations; instead of having to deal with one file of size of a dozens of GB, say, it is much easier to have several files with a smaller size of a few hundred MB to a couple of GB instead. Also, the time spent for I/O in the simulation code can be reduced if several files are written in parallel.

Each particle dump consists of `k` files, where `k` is given by `NumFilesPerSnapshot`. For `k > 1`, the filenames are of the form `snapshot_XXX.Y`, where `XXX` stands for the number of the dump, `Y` for the number of the file within the dump. Say we work on dump 7 with `k=16` files, then the filenames are `snapshot_007.0` to `snapshot_007.15`, and if HDF5 is in use (recommended!), they will be `snapshot_007.0.hdf5` to `snapshot_007.15.hdf5`. They will actually be stored in a directory called `snapdir_007`, created in the output directory. For `k=1`, the filenames will just have the form `snapshot_XXX` and no separate subdirectory is created for the snapshot. The base name "snapshot" can be changed by setting `SnapshotFileBase` to another value.

Each of the individual files of a given set of snapshot files contains a variable number of particles, but the files all have the same basic format (this is the case for all three fileformats supported by GADGET), and all of them are in binary. A binary representation of the particle data

is the preferred choice, because it allows much faster I/O than ASCII files, and in addition, the resulting files are much smaller while still providing loss-less storage of the data.

Legacy Format 1

In the original default file format of GADGET (selected with `SnapFormat=1`), the data is organised in blocks, each containing a certain information about the particles. For example, there is a block for the coordinates, and one for the temperatures, etc. The sequence of blocks in snapshots files of GADGET-4 (which for the most part is compatible with older versions of GADGET) is given in the table below. Not all blocks are necessarily present in all simulations, for example, the blocks describing gas properties such as internal energy or density will only be included in hydrodynamic simulations, and some output blocks are an optional `Config.sh` option. The presence of the mass block depends on whether or not particle masses are defined to be constant for certain particle types by means of the `Massarr` table in the file header. If a non-zero mass is defined there for a certain particle type, particles of this type will not be listed with individual masses in the mass block. If such fixed particle masses are defined for all types that are present in the snapshot file, the mass block will be completely absent.

Nr	Format2-ID	HDF5 Identifier	Block contents
1	HEAD	Header	File header
2	POS	Coordinates	Particle positions
3	VEL	Velocities	Particle velocities
4	ID	ParticleIDs	Particle IDs
5	MASS	Masses	Masses (only for particle types with variable masses)
6	U	InternalEnergy	Thermal energy per unit mass (only SPH particles)
7	RHO	Density	Density of SPH particles
8	HSML	SmoothingLength	SPH smoothing length h
9	POT	Potential	Gravitational potential of particles
10	ACCE	Acceleration	Acceleration of particles
11	ENDT	RateOfChangeOfEn	Rate of change of entropic function of SPH particles
12	TSTP	TimeStep	Timestep of particles

Within each block, the particles are ordered according to their particle type, i.e. gas particles will come first (type 0), then type-1 particles, followed by type-2 particles, and so on. However, it is important to realize that the detailed sequence of particles within the blocks may change from snapshot to snapshot. Also, a given particle may not always be stored in the snapshot file with the same sub-number among the files belonging to one set of snapshot files. This is because particles may move around from one processor to another during the course of a parallel simulation. In order to trace a particle between different outputs, one therefore has to resort to the particle IDs, which are intended to be used to label particles uniquely. (In fact, the uniqueness of the IDs in the initial conditions is checked upon start up of the code.)

The first block (number 1) has a special role, it is a header which contains global information about the particle set, for example the number of particles of each type, the number of files used for this snapshot set, etc. The fields of the file header for formats 1 and 2 in GADGET-4 is given in the table below:

Header Field	Type	HDF5 name	Comment
Npart[6]	uint	NumPart_ThisFile	The number of particles of each type in the present file.
Nall[6]	uint64	NumPart_Total	Total number of particles of each type in the simulation.
Massarr[6]	double	MassTable	The mass of each particle type. If set to 0 for a type which is present, individual particle masses are stored for this type.
Time	double	Time	Time of output, or expansion factor for cosmological simulations.
Redshift	double	Redshift	z=1/a-1 (only set for cosmological integrations)
BoxSize	double	BoxSize	Gives the box size if periodic boundary conditions are used.
NumFiles	int	NumFilesPerSnapshot	Number of files in each snapshot.

In GADGET-1/2/3, the outline of the fields in the header has been slightly different, in particular the number of particle types was always fixed to 6 (now this is given by `NTYPES`), and the total particle number was stored as a 32-bit integer, such that simulations exceeding particle numbers of a couple of billion needed as special (and ugly) extension of the header, where the high-order bits in the particle numbers where stored in a separate entry. In addition, cosmological parameters like `Omega0` and various flags informing about enabled/disabled code features were redundantly stored there as well. Finally, the total header length was filled to a total length of exactly 256 bytes, with a view to reserve the extra space for future extensions. However, as the block-size guards (see below) anyhow store the length of the header in the file format, this has been a superfluous restriction. In GADGET-4, this is hence lifted, and also other clean-ups of the header structure are implemented (such as going to 64-bit integers for the particle numbers). While this makes the new file format incompatible with older versions of GADGET, backwards compatibility can be enforced by setting the `GADGET2_HEADER` switch.

To allow an easy access of the data also in Fortran (this should not be misunderstood as an encouragement to use Fortran), the blocks are stored using the "unformatted binary" convention of most Fortran implementations. In it, the data of each read or write statement is bracketed by block-size fields, which give the length of the data block in bytes. These block fields (which are two 4-byte integers, one stored before the data block and one after) can be useful also in C-code to check the consistency of the file structure, to make sure that one reads at the right place, and to skip individual blocks quickly by using the length information of the block size fields to fast forward in the file without actually having to read the data. The latter makes it possible to efficiently read only certain blocks, for example, just temperatures and densities but no coordinates and velocities. Note however that this block size structure imposes the restriction that individual blocks may not be larger than 4 GB in file formats 1 and 2.

Assuming that variables have been allocated/declared appropriately, a possible read-statement of some of these blocks in Fortran could then for example take the form:

```
read (1) npart, nall, massarr, a, redshift
read (1) pos
read (1)
read (1) id
read (1) masses
```

In this example, the block containing the velocities, and several fields in the header, would be skipped. Further read-statements may follow to read additional blocks if present. In the file

`read_snapshot.f` included in the code distribution, you can find a simple example of a more complete read-in routine. There you will also find a few lines that automatically generate the filenames, and further hints how to read in only certain parts of the data.

When you use C, the block-size fields need to be read or skipped explicitly, which is quite simple to do. This is demonstrated in the file `read_snapshot.c`. Note that you need to generate the appropriate block field when you write snapshots in C, for example when generating initial conditions for GADGET.

GADGET-4 will check explicitly that the block-size fields contain the correct values and refuse to continue if not. It will also use the block-size information to convert between single and double precision (or vice versa), and between 32-bit and 64-bit, where appropriate, i.e. when reading in files the code will autodetect if single or double precision is used and do conversions if needed.

Legacy Format 2

Whether or not a certain block is present in GADGET's snapshot file format depends on the type of simulation, and the makefile options. This makes it difficult to write reasonably general I/O routines for analysis software when file formats 1 and 2 are used, capable of dealing with output produced for a variety of makefile settings. For example, if one wants to analyse the (optional) rate of entropy production of SPH particles, then the location of the block in the file changes if, for example, the output of the gravitational potential is enabled or not. This problem becomes particularly acute if more complicated baryonic physics modules or optional output fields are added to GADGET runs.

To remedy this problem, a variant of the default fileformat was implemented in GADGET (based on a suggestion by Klaus Dolag), which can be selected by setting `SnapFormat=2`. Its only difference is that each block is preceded by a small additional block which contains an identifier in the form of a 4-character string (filled with spaces where appropriate). This identifier is listed in the "Format2-ID" in the above block table. Using it, one can write more flexible I/O routines and analysis software that quickly fast-forwards to blocks of interest in a simple way, without having to know where it is expected in the snapshot file.

HDF5 file format

A yet more general solution is provided by HDF5, which is nowadays the *strongly* recommended format for the code. If the hierarchical data format is selected as format for snapshot files or initial conditions files, GADGET-4 accesses files with low level HDF5 routines. Advantages of HDF5 lie in its portability (e.g. automatic endianness conversion) and generality (which however results in somewhat more complicated read and write statements), and in the availability of a number of tools to manipulate or display HDF5 files. Also, attributes such as units or conversion factors can be stored along-side data-sets. Together with HDF5 utility commands like h5ls, h5dump, etc., this makes the file format self-documenting to a significant degree. A wealth of further information about HDF5 can be found on the website of the HDF5 library.

In the HDF5 file format of GADGET, the blocks are stored as datasets with names as given in the column "HDF5 Identifiers" in the table above. To make accessing different particle types easier, for each particle type presented in the snapshot file, a separate data group is defined, named `ParticleType0`, `ParticleType1`, and so on, and the blocks of the standard file format are appearing as datasets within these groups. These data groups can be thought of as subdirectories in the HDF5 file, with each being devoted to one particular particle type. In addition, in HDF5 each of these datasets is also equipped with attribute values that store conversion factors to cgs

units, as well as factors that inform whether one has to multiply with a certain power of the scale factor and/or of the dimensionless Hubble parameter `h` to get to truly physical quantities.

Finally, the file header is also moved to a special `Header` group in the HDF5 output. This group does not contain datasets, but only attributes that hold the values for all the fields defined for file format 1 and 2 of GADGET-4. The names of these attributes are listed in the column "HDF5 name" in the table above.

To graphically explore the contents of HDF5 files, the program HDFView, available for free from the HDF-Group, is one good possibility. It allows an easy exploration of the structure and the contents of the various HDF5 outputs produced by GADGET4. It also readily reveals the type of data set fields, the values of attributes, etc., and hence greatly facilitates the writing of corresponding analysis scripts.

Alternatively, one can also explore the contents of HDF5 files at the command line, using simple tools that are part of the HDF5 library. For example, to list contents of the header of a snapshot file on the command line, use the following command:

```
h5dump -g Header  snapshot_007.3.hdf5
```

To list, for example, which blocks are available for type-0 particles, you can use the following command:

```
h5ls  snapshot_007.3.hdf5/ParticleType0
```

Just using `h5ls snapshot_007.3.hdf5` would tell you which particle types are available as separate data groups.

Finally, GADGET-4 also stores all the parameters and config file options with which it was run to produce the given snapshot in all its HDF5 files. This is realized through Parameter attributes stored in two special data groups called `Parameters` and `Config`.

To examine the parameter file settings that were used, you can hence issue the command:

```
h5dump -g Parameters  snapshot_007.3.hdf5
```

And for retrieving the `Config.sh` options, you can use:

```
h5dump -g Config  snapshot_007.3.hdf5
```

Here are a couple of additions noted on the format, units, and variable types of the individual blocks in the snapshot files:

- Particle positions: A floating point 3-vector for each particle, giving the comoving coordinates in internal length units (corresponds to kpc/h if the above choice for the system of units is adopted). N=sum(header.Npart) is the total number of particles in the file. Note: For file formats 1 and 2, the particles are ordered in the file according to their type, so it is guaranteed that particles of type 0 come before those of type 1, and so on. However, within a type, the sequence of particles can change from dump to dump. For file format 3, different particle types appear in separate particle groups.

- Particle velocities: Particle velocities `u` are in internal velocity units (corresponds to km/sec if the default choice for the system of units is adopted). For cosmological simulations, peculiar velocities `v` are obtained from `u` by multiplying `u` with sqrt(a), i.e. v = u * sqrt(a).

- Particle identifiers: These are unsigned 32-bit or 64-bit integers (if `IDS_64BIT` is set) and intended to provide a unique identification of particles. The order of particles may change between different output dumps, but the IDs can be easily used to bring the particles back into the original order for every dump.

- Variable particle masses: Single or double precision floats, with a total length Nm. Only stored for those particle types that have variable particle masses (indicated by zero entries in the corresponding `massarr` entry in the header). Nm is thus the sum of those `npart` entries that have vanishing `massarr` . If `Nm=0`, this block is not present at all.

- Internal energy: Single/double precision float values for Ngas. Internal energy per unit mass for the `Ngas=npart(0)` gas particles in the file. The block is only present for `Ngas>0`. Units are again in internal code units, i.e. for the standard system of units, `u` is given in `(km/sec)^2` .

- Density: The comoving density of SPH particles. Units are again in internal code units, i.e. for the above system of units, `rho` is given in `10^10 Msun/h / (kpc/h)^3`.

- Smoothing length: Smoothing length of the SPH particles. Given in comoving coordinates in internal length units.

- Gravitational potential: This block will only be present if it is explicitly enabled in the makefile.

- Accelerations: Likewise, only present if it is explicitly enabled in the makefile.

- Rate of entropy production: The rate of change of the entropic function of each gas particles. For adiabatic simulations, the entropy can only increase due to the implemented artificial viscosity. This block will only be present if it is explicitly enabled in the makefile.

- Timesteps of particles: Individual particle timesteps of particles. For cosmological simulations, the values stored here are d ln(a), i.e. intervals of the natural logarithm of the expansion factor. This block will only be present if it is explicitly enabled in the makefile.

Format of initial conditions

The possible file formats for initial conditions are the same as those for snapshot files, and are selected with the `ICFormat` parameter. However, only the blocks up to and including the gas temperature (if gas particles are included) need to be present; gas densities, SPH smoothing lengths and all further blocks need not be provided and are ignored.

In preparing initial conditions for simulations with gas particles, the temperature block can be filled with zero values, in which case the initial gas temperature is set in the parameterfile with the `InitGasTemp` parameter. However, even when this is done, the temperature block must still be present. Note that the field `Time` in the header will be ignored when GADGET-4 is reading an initial conditions file. Instead, you have to set the time of the start of the simulation with the `TimeBegin` parameter in the parameterfile.

Diagnostic outputs

Aside from the verbose stdout log-file that is produced when the code is run (take a detailed look!), various files in the output directory provide useful metrics about the progress of a simulation, its memory usage, work-load balance, and cpu consumption. Here, these files are described in turn. (Make sure that your browser window is set wide enough to avoid line-breaks in some of the wide tables.)

stdout

After starting the code, the stdout will report a welcome message that includes the git version hash-key and the date this corresponds to. This uniquely identifies the last update/pull of the code from the version control system. The output then lists all the compile time options that were set. Note that these can be easily copied into an empty `Config.sh` file to configure the source with the same configuration. The code then reports the number of MPI ranks used, as well as the detected node configuration and the pinning settings that are applied, if any. This is followed by a detailed examination of the memory available on the execution hosts (unless this is disabled). After that, the parameterfile settings that are used for the run are listed. Again, note that these can be easily pasted into an empty parameter file to reproduce the settings used here, if needed. The stdout-file thus always contains all the information needed to reproduce a given run. Before the actual simulation begins, some other information that is sometimes useful is reported as well, such as the unit system that is used and the sizes of the most important data structures used by the code.

During a run, the code outputs frequent log-message what is currently done by GADGET-4. Usually, the corresponding lines begin with a capitalized key-word that identifies the corresponding code part. For example, lines beginning with "DOMAIN:" refer to information issued by the domain decomposition, lines starting with "PM-PERIODIC:" will be identified with the periodic FFT-based computation of the long-range gravitational force. It can be useful to filter the file with grep for one of these phrases to get a clearer picture of what is happening in a particular code part.

In case a problem should occur that forces the code to stop, you will typically see a line starting with "Code termination on task" in the output file, followed with information on which task, in which function, and in which line number of a particular source file the crash was triggered. This is followed by some information about the nature of the problem. For example, such a message could look like this:

```
Code termination on task=48, function restart(), file src/io/restart.c, li
```

All of this together hopefully provides a good clue about what may have happened, and how the issue can be fixed.

info.txt

The file with name `info.txt` in the output directory just contains a list of all the timesteps. The last entry always holds the timestep that is currently processed. For a running simulation, the command

```
tail info.txt
```

will thus inform you about the current progress of the simulation. Typical output in this file looks like this:

```
Sync-Point 26566, Time: 0.789584, Redshift: 0.26649, Systemstep: 5.84556e-

Sync-Point 26567, Time: 0.789642, Redshift: 0.266396, Systemstep: 5.84599e

Sync-Point 26568, Time: 0.789701, Redshift: 0.266302, Systemstep: 5.84642e
```

The first number just counts and identifies all the timesteps in terms of the synchronization points of the timestep hierarchy, which are the places where force computations occur. The values given after Time/Redshift are the current simulation times (time is the scale factor in cosmological simulations). The Systemstep-values give the time difference to the preceeding step, which is equal to by how much the simulation as a whole has been advanced since the last synchronization point. For cosmological integrations, this is supplemented with the systemstep in logarithmic form in terms of the ln of the scale factor. Finally, the number of collisionless particles that are in sync with the current system time (i.e. these are the ones that have finished their timestep there and start a new one) is reported. This is also done separately for the SPH particles.

timebins.txt

The file `timebins.txt` in the output directory provides a much more detailed account of the distribution of particles onto the timestep hierarchy. A typical entry, created for every synchronization point, of this file looks as follows:

```
Sync-Point 16521, Time: 0.934985, Redshift: 0.0695359, Systemstep: 6.92201
Occupied timebins: gravity         sph          dt              cumul-grav
    bin=17        12287592           0     0.001184577701         27292287
    bin=16         8012442           0     0.000592288851         15004695
 X  bin=15         4946259           0     0.000296144425          6992253
 X  bin=14         1809726           0     0.000148072213          2045994
 X  bin=13          236268           0     0.000074036106           236268
               ------------------------
Total active:      6992253           0
```

The first line corresponds to information also included in `info.txt`. This is followed with a table that shows the distribution of all particles (listed unter "gravity") onto different timebins. The different possible timestep sizes are identified via the timebin number and the corresponding timestep size, which is listed in column "dt". Also given is the distribution of gaseous SPH particles onto the timebins. The columns "cumul-grav" and "cumul-sph" list the cumulative numbers of particles in this timebin and below in the categories of gravity and SPH calculations. The "X" symbols in the beginning mark the timebins that are synchronized with the current system time. All particles that are included in these timebins need a force calculation at the current system time, hence the cumulative numbers reported for the top timebin of this set, which is marked with a "<" sign in the "A" column give an indication of the amount of computational work required for this step. The value reported under "avg-time" represents an estimated execution time of this current step, obtained by averaging around five past executions of this timebin when it was equally marked with a "<" sign. Note that sometimes these values can be distorted a bit when the code had to do a special operation during one of these last averaging steps, like computing a group catalogue, or writing restart files. Also note that lower timebins need to be executed more frequently than higher timebins. For example, there will be twice as many executions of timebin 15 than of timebin 16 for every execution of timebin 17. The last column represents the resulting distribution of consumption of total CPU-time when the different

execution frequency of the steps is taken into account, based on the values reported under `avg-time` . While this estimate is approximate in nature, it gives a good idea about what cost is incurred in the total simulation run-time due to the presence of certain timebins. In particular, a situation where the lowest occupied timebins consume the dominant fraction of the CPU time despite the fact that these are only thinly populated normally indicates that the simulation does not run very efficiently and only slowly makes progress.

cpu.txt

In the file cpu.txt, you get detailed statistics about the total CPU consumption measured in various parts of the code while it is running. At each timestep, a table is added to this file, which roughly looks like this:

```
Step 328, Time: 0.0485236, CPUs: 2496, HighestActiveTimeBin: 19
                          diff                  cumulative
total                   260.48  100.0%   63526.74  100.0%
  treegrav              213.41   81.9%   52556.59   82.7%
    treebuild             3.01    1.2%     931.09    1.5%
      insert              1.56    0.6%     524.64    0.8%
      branches            0.34    0.1%     113.39    0.2%
      toplevel            0.19    0.1%      72.99    0.1%
    treeforce           210.39   80.8%   51622.08   81.3%
      treewalk          174.82   67.1%   42617.69   67.1%
      treeimbalance      27.99   10.7%    6866.13   10.8%
      treefetch           0.53    0.2%     105.94    0.2%
      treestack           7.05    2.7%    2032.32    3.2%
  pm_grav                31.68   12.2%    6395.72   10.1%
  ngbtreebuild            0.00    0.0%       0.00    0.0%
  ngbtreevelupdate        0.11    0.0%      25.42    0.0%
  ngbtreehsmlupdate       0.00    0.0%       0.21    0.0%
  sph                     0.00    0.0%       0.00    0.0%
    density               0.00    0.0%       0.00    0.0%
      densitywalk         0.00    0.0%       0.00    0.0%
      densityfetch        0.00    0.0%       0.00    0.0%
      densimbalance       0.00    0.0%       0.00    0.0%
    hydro                 0.00    0.0%       0.00    0.0%
      hydrowalk           0.00    0.0%       0.00    0.0%
      hydrofetch          0.00    0.0%       0.00    0.0%
      hydroimbalance      0.00    0.0%       0.00    0.0%
  domain                 11.95    4.6%    2177.10    3.4%
  peano                   1.24    0.5%     291.91    0.5%
  drift/kicks             1.44    0.6%     312.75    0.5%
  timeline                0.07    0.0%      18.62    0.0%
  treetimesteps           0.00    0.0%       0.00    0.0%
  i/o                     0.00    0.0%      71.90    0.1%
  logs                    0.18    0.1%      43.06    0.1%
  fof                     0.00    0.0%       0.00    0.0%
    fofwalk               0.00    0.0%       0.00    0.0%
    fofimbal              0.00    0.0%       0.00    0.0%
  restart                 0.00    0.0%    1506.94    2.4%
  misc                    0.40    0.2%     126.52    0.2%
```

Two columns of measurements are provided. The entries under "diff" measure the time difference in seconds relative to the last time step. This total elapsed time reported in the row "total" is then subdivided onto different code parts, as labelled. If the indention level is increased, a further subdivision of the corresponding code part is provided in terms of the subsequent indented items. The relative fractions of these various code parts are also reported as a percentage of the total time of the step.

In addition, the values reported under "cumulative" give the same analysis for the cumulative elapsed time since the start of the simulation. Again, the first number gives absolute elapsed times, so that the number reported under "total" is the total consumed time since the start of the simulation. The count continues across restarts, i.e. when the simulation is completed, this number gives a faithful account of the total time (in seconds) needed to bring the simulation to completion. Note that to get the full CPU-time consumption in core-hours, this number has still to be multiplied with the number of cores occupied (and to be divided by 3600 to convert from seconds to hours, of course).

The data contained in the `cpu.txt` file is additionally output as a column-separated file `cpu.csv` , where all the numbers for one timestep appear in a one line entry. This can be used to more easily make plots of the CPU time consumption in different code parts, if this is desired.

domain.txt

The log file `domain.txt` receives a new entry whenever a new domain decomposition is carried out by the code. A typical output for one step may look like this:

```
DOMAIN BALANCE, Sync-Point 31488, Time: 0.994978
Timebins:       Gravity       Hydro  cumulative      grav-balance       hy
 |bin=17     5882690219           0 10204440916  m  1.106 | 1.000       0.
 |bin=16     2353095591           0  4321750697  m  1.525 | 1.000       0.
>|bin=15     1425860069           0  1968655106  m  1.031 | 1.016  *    0.
 |bin=14      503495925           0   542795037  m  1.070 | 1.031  *    0.
 |bin=13       39299112           0    39299112  m  2.247 | 1.040  *    0.
-------------------------------------------------------------------------
BALANCE,  LOAD:   1.003       0.000       1.003  WORK:      1.087
```

Here the first line indicates the current system time of the code, and the following table reports all timebins and their occupancy with gravity-only and hydrodynamic particles, as well as the total cumulative occupancy up to the given timebin. The currently highest synchronized timebin is marked with a "<" sign. The current timestep distribution and the settings of the code (in particular the parameter `ActivePartFracForNewDomainDecomp`), allow the code to tell when the next domain decomposition will be done. In particular, in the above example, this will not happen when timebins 14 or 13 are the highest active timebin. Therefore, the current domain decomposition has attempted to balance timebins 15, 14, and 13 simultaneously, which is indicated by the `*` marker sign. Only the timebins marked by `*` need to be balanced by the current domain decomposition. In this balancing, the code attempts to reach a good balance for the particle load, the gravity work-load, and the hydrodynamic work-load. For the latter two, the algorithm takes into account that several steps need to be executed until the next domain decomposition will occur, and that the goal should be to minimize the overall execution time until then. This for example means that a larger relative imbalance for bin 13 may be acceptable if the absolute time for executing this step is reasonably short.

The result of the domain decomposition is reported under "grav-balance" and "hydro-balance", respectively. The first number gives the work-imbalance if this step would be executed alone.

This imbalance factor is defined as the maximum (estimated) execution time among the MPI-ranks divided by the average of the execution times. Because the total work required per step is invariant under the domain decomposition, this should be approximately independent of the domain decomposition, so the goal is to push the maximum execution time as close to the average as possible. The imbalance factor gives the relative slowdown due to an imperfect work-load balance.

The values reported under grav-balance and hydro-balance inform about the relative success of the domain decomposition to reach the desired balance among multiple different quantities. The first number gives the imbalance factor if only the current step would have to be executed. This is however only the full story if another domain decomposition will be carried out in the next step immediately. If this is not the case, several steps need to be averaged appropriately, and the relative slow-down of the residual imbalance in the present step is reported as the second number, while the overall imbalance over all simultaneously balanced steps is reported behind "WORK" (first number in the case of gravity.) The second numbers in the table above should add up to this number, as they give for every timebin involved in the balancing the relative contribution they are responsible for in this overall imbalance. Effectively, the "WORK" factor tells how much faster the code may run if the domain decomposition could be carried out perfectly for every involved step. Note that timebins that are occupied with particles but do not need to be balanced by the current domain decomposition may have a large intrinsic imbalance, but this doesn't affect the run-time behaviour at all, hence the second number is reported as a 1.0 in this case.

Similarly, the hydrodynamic imbalance is reported in a second pair of columns, yielding a further overall imbalance factor that is reported as second number after "WORK".

Finally, there is an overall memory-load imbalance factor reported behind "LOAD", which is also subdivided into gravity and SPH-particles. This is another important metric as GADGET-4 attempts to balance the work-load without allowing for significant memory imbalance (because often simulations can be memory-bound). This is in practice achieved by trying to push down all values reported under LOAD and WORK simultaneously. The categories memory-load and work-load in gravity and SPH is given equal weight in this context.

balance.txt

The file `balance.txt` provides another quick look at the performance and execution pattern of the code. It is important to view this in a terminal window that is set wide enough to avoid extra line wrapping. In this file, each step of the code is reported with a single line, giving step number, total execution time, and the number of active gravity and hydro particles. This is followed with a block of characters of a total length representing the full execution time of the step. Different code parts are represented by different characters, and are filling this block with their corresponding character in proportion to the time spent in this code part. A piece of the the resulting output may then for example look something like this:

```
Step=    761  sec=     9.406 Nsync-grv=  27292287 Nsync-hyd=         0  :r
Step=    762  sec=     0.373 Nsync-grv=     15778 Nsync-hyd=         0  ::
Step=    763  sec=     8.025 Nsync-grv=  26662053 Nsync-hyd=         0  ::
Step=    764  sec=     0.371 Nsync-grv=     16226 Nsync-hyd=         0  ::
Step=    765  sec=     9.385 Nsync-grv=  27292287 Nsync-hyd=         0  :r
Step=    766  sec=     0.403 Nsync-grv=     16748 Nsync-hyd=         0  ::
Step=    767  sec=     8.049 Nsync-grv=  26662117 Nsync-hyd=         0  ::
Step=    768  sec=     0.419 Nsync-grv=     17202 Nsync-hyd=         0  ::
Step=    769  sec=     9.399 Nsync-grv=  27292287 Nsync-hyd=         0  :r
Step=    770  sec=     0.407 Nsync-grv=     17695 Nsync-hyd=         0  ::
```

```
Step=    771  sec=     8.016 Nsync-grv=  26662178 Nsync-hyd=         0  ::
Step=    772  sec=     0.430 Nsync-grv=     18152 Nsync-hyd=         0  ::
Step=    773  sec=     9.517 Nsync-grv=  27292287 Nsync-hyd=         0  :r
Step=    774  sec=     0.435 Nsync-grv=     18726 Nsync-hyd=         0  ::
Step=    775  sec=     8.062 Nsync-grv=  26662000 Nsync-hyd=         0  ::
Step=    776  sec=     0.506 Nsync-grv=     19217 Nsync-hyd=         0  ::
Step=    777  sec=    19.647 Nsync-grv=  27292287 Nsync-hyd=         0  ::
Step=    778  sec=     0.445 Nsync-grv=     19759 Nsync-hyd=         0  ::
Step=    779  sec=     8.101 Nsync-grv=  26662013 Nsync-hyd=         0  ::
Step=    780  sec=     0.373 Nsync-grv=     20232 Nsync-hyd=         0  ::
Step=    781  sec=     9.538 Nsync-grv=  27292287 Nsync-hyd=         0  :r
Step=    782  sec=     0.366 Nsync-grv=     20811 Nsync-hyd=         0  ::
Step=    783  sec=     8.053 Nsync-grv=  26662006 Nsync-hyd=         0  ::
```

A key to the different symbols used for different code parts is included in the beginning of the file `balance.txt`. The idea of this output is to allow a quick graphical analysis of the execution patterns of the code, in particular also to allow visual identification of sudden changes of it. For example, normally the appearance of additional timestep bins, or the dominance of certain code parts can be readily inferred from this graphical text output. Likewise, the occurrence of things like group finding or light-cone file output tends to show up. Also imbalances in certain code parts are reported separately by different symbols, so this can also be a way to tell whether imbalances in certain places are particularly strong.

One should be cautious, however, to avoid over-interpreting this graphical text output. Because short and long steps are all stretched to the same width in their corresponding output lines, short timesteps (which may be comparatively unimportant for the total CPU budget) tend to be overrepresented in this graphical representation. Note that by filtering out certain timebins, this effect can be avoided and the variation in the execution metrics of this particular step as the simulation progresses can be monitored.

memory.txt

Another interesting diagnostic information about the simulation code is contained in the file `memory.txt`. There, each time a new high-watermark is reached in the total memory consumption on any of the MPI-ranks, a new entry in the form of an extended table is produced. An example for this table is reproduced below.

One of the most important numbers is the value reported behind "Largest Allocation Without Generic". This is the minimum amount of memory the code needed to run in the present configuration during this timestep, excluding communication buffers. The latter are flexible in size and will automatically adjust to the amount of free memory left according to the `MaxMemSize` parameter.

For a more detailed view, the table tells the name (normally identical to the variable name in the source code used to refer to the buffer) of each allocated memory block. This is followed by its size in MBytes, and the cumulative size of all allocated blocks up to this point. In addition, the function, file name, and line number where this particular block was allocated is given as well.

GADGET-4 organizes its internal memory handling in the form of a stack in order to avoid memory fragmentation. The flag reported under "F" shows whether the block has been explicitly allocated as movable (so that previous blocks may be freed or resized), or whether this hasn't been done by the source code. The number reported for "Task" is just the MPI-rank on which this maximum allocation had occurred.

```
MEMORY:  Largest Allocation = 1542.21 Mbyte  |  Largest Allocation Without

------------------------- Allocated Memory Blocks---- ( Step     5086 )--
Task    Nr F                  Variable      MBytes   Cumulative  Function|
-------------------------------------------------------------------------
  12     0 0                                Exportflag      0.0002       0.
  12     1 0                               Exportindex      0.0002       0.
  12     2 0                          Exportnodecount      0.0002       0.
  12     3 0                                    Send      0.0005       0.
  12     4 0                                    Recv      0.0005       0.
  12     5 0                              Send_count      0.0002       0.
  12     6 0                             Send_offset      0.0002       0.
  12     7 0                              Recv_count      0.0002       0.
  12     8 0                             Recv_offset      0.0002       0.
  12     9 0                        Send_count_nodes      0.0002       0.
  12    10 0                       Send_offset_nodes      0.0002       0.
  12    11 0                        Recv_count_nodes      0.0002       0.
  12    12 0                       Recv_offset_nodes      0.0002       0.
  12    13 0                         Tree.Send_count      0.0002       0.
  12    14 0                        Tree.Send_offset      0.0002       0.
  12    15 0                         Tree.Recv_count      0.0002       0.
  12    16 0                        Tree.Recv_offset      0.0002       0.
  12    17 1                               IO_Fields      0.0035       0.
  12    18 1                               IO_Fields      0.0123       0.
  12    19 0                           slab_to_task      0.0020       0.
  12    20 0                       slabs_x_per_task      0.0002       0.
  12    21 0                  first_slab_x_of_task      0.0002       0.
  12    22 0                       slabs_y_per_task      0.0002       0.
  12    23 0                  first_slab_y_of_task      0.0002       0.
  12    24 0                           slab_to_task      0.0020       0.
  12    25 0                       slabs_x_per_task      0.0002       0.
  12    26 0                  first_slab_x_of_task      0.0002       0.
  12    27 0                       slabs_y_per_task      0.0002       0.
  12    28 0                  first_slab_y_of_task      0.0002       0.
  12    29 0                             kernel[1]      8.0312       8.
  12    30 1                   def->ntype_in_files      0.0015       8.
  12    31 1                                     P     98.3903     106.
  12    32 1                                  SphP      0.0001     106.
  12    33 1               NextActiveParticleHydro      0.0001     106.
  12    34 1                    NextInTimeBinHydro      0.0001     106.
  12    35 1                    PrevInTimeBinHydro      0.0001     106.
  12    36 1             NextActiveParticleGravity      3.6441     110.
  12    37 1                  NextInTimeBinGravity      3.6441     113.
  12    38 1                  PrevInTimeBinGravity      3.6441     117.
  12    39 1                          D->StartList      0.0020     117.
  12    40 1                            D->EndList      0.0020     117.
  12    41 1               D->FirstTopleafOfTask      0.0002     117.
  12    42 1                 D->NumTopleafOfTask      0.0002     117.
  12    43 1                          D->TopNodes      0.0234     117.
  12    44 1                        D->TaskOfLeaf      0.0103     117.
  12    45 1                 D->ListOfTopleaves      0.0103     117.
  12    46 1                                 PS     65.5936     183.
  12    47 1                              Group      0.0610     183.
  12    48 1                           SubGroup      2.0081     185.
```

```
  12    49 1                          D->StartList      0.0001     185.
  12    50 1                            D->EndList      0.0001     185.
  12    51 1                  D->FirstTopleafOfTask     0.0001     185.
  12    52 1                    D->NumTopleafOfTask     0.0001     185.
  12    53 1                            D->TopNodes     0.0011     185.
  12    54 1                          D->TaskOfLeaf     0.0005     185.
  12    55 1                     D->ListOfTopleaves     0.0005     185.
  12    56 1                              IndexList     2.5806     187.
  12    57 1                          D->StartList      0.0001     187.
  12    58 1                            D->EndList      0.0001     187.
  12    59 1                  D->FirstTopleafOfTask     0.0001     187.
  12    60 1                    D->NumTopleafOfTask     0.0001     187.
  12    61 1                            D->TopNodes     0.0004     187.
  12    62 1                          D->TaskOfLeaf     0.0002     187.
  12    63 1                     D->ListOfTopleaves     0.0002     187.
  12    64 1                                     sd     8.5197     196.
  12    65 1                                   Head      2.1299     198.
  12    66 1                                   Next      2.1299     200.
  12    67 1                                   Tail      2.1299     202.
  12    68 1                                    Len      1.0650     203.
  12    69 1                        coll_candidates      0.2130     203.
  12    70 1                          D->StartList      0.0001     203.
  12    71 1                            D->EndList      0.0001     203.
  12    72 1                  D->FirstTopleafOfTask     0.0001     203.
  12    73 1                    D->NumTopleafOfTask     0.0001     203.
  12    74 1                            D->TopNodes     0.0011     203.
  12    75 1                          D->TaskOfLeaf     0.0005     203.
  12    76 1                     D->ListOfTopleaves     0.0005     203.
  12    77 1                            unbind_list     1.1685     205.
  12    78 0                                   dold      1.1685     206.
  12    79 0                                 potold      2.3369     208.
  12    80 1                       Tree.NodeLevel      0.0001     208.
  12    81 1                     Tree.NodeSibling      0.0005     208.
  12    82 1                       Tree.NodeIndex      0.0005     208.
  12    83 1                       Tree.Task_list      2.1603     210.
  12    84 1                       Tree.Node_list      2.1603     212.
  12    85 1                           Tree.Nodes     35.7231     248.
  12    86 1                          Tree.Points      0.0001     248.
  12    87 1                        Tree.Nextnode      3.6446     252.
  12    88 1                          Tree.Father      3.6441     255.
  12    89 1                             PartList   1019.9025    1275.
  12    90 1                              Ngblist      2.1603    1277.
  12    91 1                               DataIn     13.8409    1291.
  12    92 1                           NodeDataIn     29.4413    1321.
  12    93 1                              DataOut      2.3068    1323.
  12    94 1                              DataGet      6.2735    1329.
  12    95 1                           NodeDataGet     19.1392    1348.
  12    96 1                            DataResult      1.0456    1349.
-------------------------------------------------------------------------
```

timings.txt

The file timings.txt contains detailed performance statistics of the gravitational tree algorithm for each timestep, which is usually (but not always) the main sink of computational time in a simulation. A typical output for a certain step may look like this:

```
Step(*): 364, t: 0.050221, dt: 4.79423e-05, highest active timebin: 19  (l
Nf=8589934592  timebin=19  total-Nf=2087354121396
   work-load balance: 1.15054   part/sec: raw=19746.3, effective=17162.6
   maximum number of nodes: 636059, filled: 0.943865
   NumForeignNodes: max=395371 avg=258155 fill=0.201039
   NumForeignPoints: max=1.51499e+06 avg=973285 fill=0.0961538  cycles=42
   avg times: <all>=208.996  <tree>=174.285  <wait>=26.7201  <fetch>=0.593
   total interaction cost: 5.46276e+12  (imbalance=1.13168)
```

The first line of the block generated for each step informs about the number of the current timestep, the current simulation time, and the system timestep itself (i.e. the time difference to the last force computation). Also the highest active time at this step and the range of occupied timebins is reported. Then, the number behind `Nf` is the number of gravitational forces that are computed in this invokation of the tree code, whereas the number behind `total-Nf` gives the total number of such force calculations since the beginning of the simulation.

The line starting with `work-load balance` gives the actual work-load balance measured for the summed execution times of the tree walks. The next number, for `part/sec`, measures the raw force speed in terms of tree-force computations per processor per second. The first number basically gives the speed that would be achieved for perfect work-load balance, while the actually achieved average effective force speed will always be lower in practice due to work-load imbalance, and this number is given after the label `effective`. The number reported for `ia/part` gives the average number of particle-node interactions required to compute the force for each of the active particles, and this should be roughly anti-proportional to the raw calculational speed. The following two numbers in parathenses give the average number of particle-particle and particle-node interactions that were computed per particle.

The next line reports the maximum number of tree nodes that were used among the MPI-ranks, while the number behind `filled` gives this quantity normalised to the number of allocated tree nodes, and hence indicates the degree to which the tree storage was filled. The next two lines report how many nodes and particles were imported from other nodes by each MPI rank, both in terms of the maximum that occurred, and the average values. Again, the values behind 'fill' give the maximum degree to which the buffer storage for these two components were filled. The value behind cycles indicates the maximum number of times an MPI process needed to call the routine that fetches foreign nodes or particles before it could continue.

The line beginning with `avg times` reports the average execution times of different parts in the tree calculation. For `all` the total execution time is reported, for `tree` the time the code carries out actual tree walks and gravity calculations, and for `wait` the time lost because some processes finish before others and then need to wait until everybody is done (this reflects the work-load imbalance). The time reported for `fetch` is the time MPI ranks needed to wait for the arrival of data requested from foreign nodes, while `stack` measures further bookkeeping time to organize the importing of data in the first place. If the PM-algorithm is used, a number in parenthesis gives the execution time of the most recent PM-force calculation.

Finally, the last line reports the total cost measure the code computes for the work done in this step, and the imbalance therein. It is this cost measure that the code tries to balance in the domain decomposition. This is thus the imbalance the code expects to be there based on the domain decomposition it has done, whereas the one reported for `work-load balance` is the one measured based on the actual execution time.

density.txt

The file `density.txt` contains detailed performance statistics of the SPH density calculation for each timestep. This is very simular in structure and information content to the `timings.txt` output for the gravitational tree walks. A typical output for a certain step may look like this:

```
Step: 404, t: 0.0940046, dt: 6.98469e-05, highest active timebin: 17  (low
Nf= 16777216  highest active timebin=19  total-Nf=3875640800
   work-load balance: 1.18233   part/sec: raw=224112, effective=189550
   maximum number of nodes: 11024, filled: 0.835658
   NumForeignNodes: max=8058 avg=3646.1 fill=0.00688725
   NumForeignPoints: max=99570 avg=52765 fill=0.010639  cycles=8
   avg times: <all>=1.24922  <tree>=0.959757  <wait>=0.168166  <fetch>=0.0
```

Just like for the tree-gravity, the `work-load balance` gives the ratio between the maximum execution time for SPH density loops relative to the average among all MPI ranks. The numbers behind `part/sec` and `effective` give the number of SPH density computations per second completed per particle, ignoring work-load imbalance or including it, respectively. The numbers reported for imported nodes and particles have the same meaning as for the gravity tree, except that they here refer to the neighbor tree, and imported particles are exclusively SPH particles (type 0).

Finally, the last line reports the average times spent in different parts of the calculation. Note that `tree` refers here to walking of the neighbor tree to find neighbours and doing all SPH calculations on them.

hydro.txt

There is also a log file informing in detail about the performance of the SPH hydrodynamical force calculations. Its structure and information content follows very closely the `density.txt` file for the SPH density computation, we therefore refrain from inlining an example output here.

energy.txt

In the file `energy.txt`, the code gives some statistics about the total energy of the system. In regular intervals (specified by `TimeBetStatistics`), the code computes the total kinetic, thermal and potential energy of the system, and it then adds one line to this file. Each of these lines contains 28 numbers (if `NTYPES=6` is used), which you may process by some analysis script. The first number in the line is the output time, followed by the total internal energy of the system (will be 0 if no gas physics is included), the total potential energy, and the total kinetic energy.

The next 18 numbers are the internal energy, potential energy, and kinetic energy of the six particle types. Finally, the last six numbers give the total mass in these components.

Note that while frequent outputs of the energy quantities allow a check of energy conservation in Newtonian dynamics, this is more difficult for cosmological integrations, where the Layzer-Irvine equation is needed. (Note that the softening needs to be fixed in comoving coordinates for it.) We remark that it is in practice not easy to obtain a precise value of the peculiar potential energy at high redshift (should be exactly zero for a homogeneous particle distribution). Also, the cosmic energy integration is a differential equation, so a test of conservation of energy in an expanding cosmos is less straightforward that one may think.

sfr.txt

This is only present in simulations with cooling and star formation. It can then be used to obtain a simple global overview of the total star formation rate in the simulation.

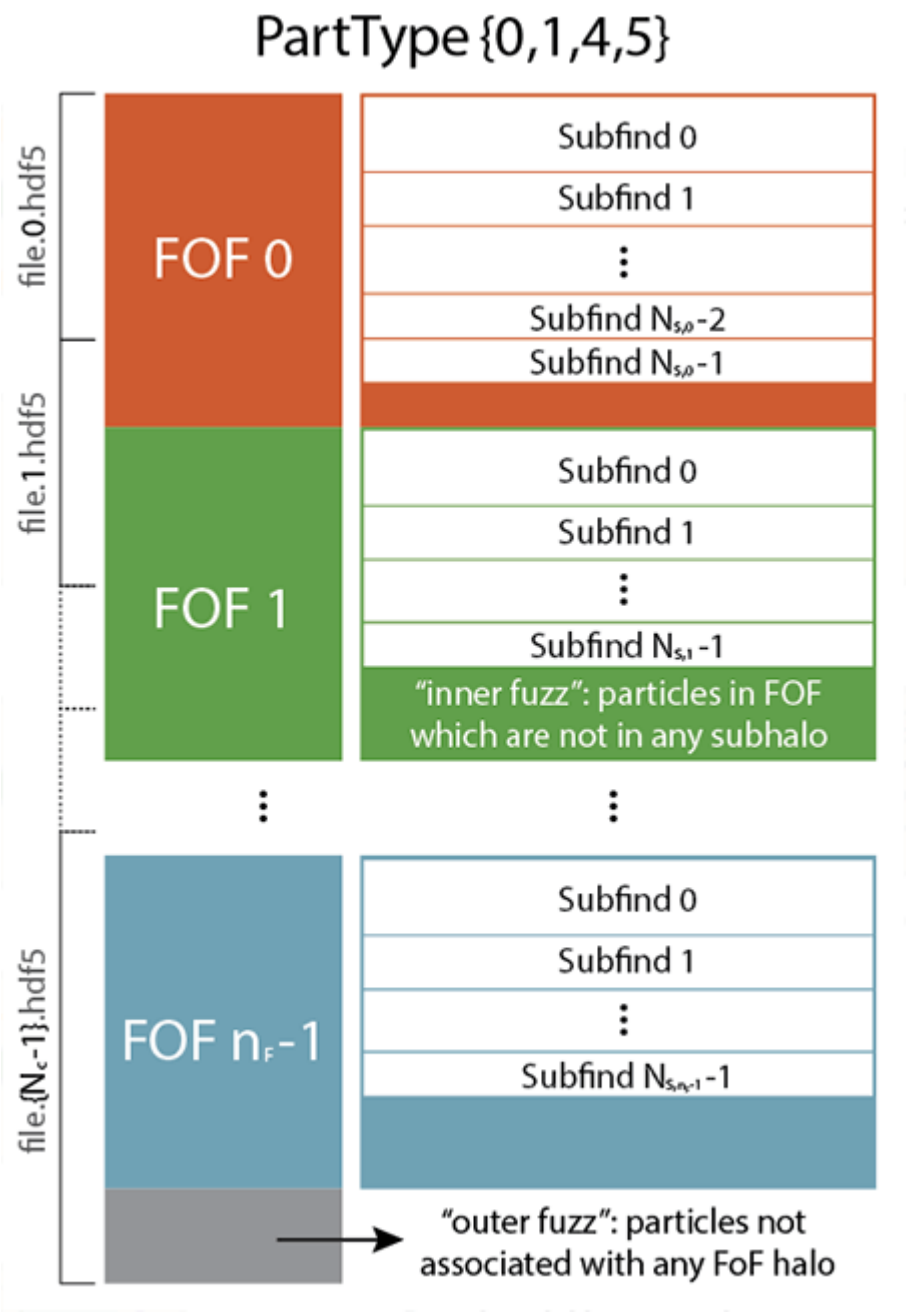
Groups and subhalos

Group finding is a basic analysis task of cosmological simulations of structure formation. GADGET-4 contains parallel algorithms for finding virialized dark matter halos and their embedded gravitationally bound subhalos which can be run both on the fly and in postprocessing. The group finders may also be applied to particle data accumulated directly on past backwards lightcone. In the following, a basic description of the storage format is given, which is largely identical with the one introduced in AREPO in the context of the Illustris and IllustrisTNG projects (note that read-scripts from the data-releases of these two projects may thus be easily adapted for GADGET-4 usage, too).

FOF and SUBFIND

Group finding in GADGET-4 is supported through two main algorithms, the classic friends-of-friends (FOF) approach to find groups of particles of approximately virial overdensity, and the SUBFIND algorithm to identify gravitationally bound substructures in these groups in configuration space. SUBFIND hence relies on FOF, and in can only be used if also FOF is enabled. Furthermore, there is the SUBFIND_HBT variant of SUBFIND which identifies the substructure candidates based on past membership in gravitationally bound subhalos.

In case group finding is enabled, the snapshot output of GADGET-4 will occur in group order. Specifically, the particles in the output files will appear in the order of the group catalogue itself, giving them the following logical structure:



In case the HDF5 output is used, the order is imposed individually on every particle type in its corresponding data set group. In the classic file format, it applies to each particle type individually as well (here each block in the snapshot contains the particle types jointly in type-order). If a snapshot is split over multiple files, the contents of these files in each dataset are treated logically as if they were concatenated in the order of the partial files. Note that individual groups are allowed to spill over across file boundaries.

The snapshot files first contain the particles found in FOF groups, and these groups are ordered descending in size. This means that the particle data begins with the particles contained in the largest FOF group. Normally, not all particles are contained in FOF groups above the imposed minimum threshold for the group particle number. The particles outside the resolved groups then come at the end of snapshot file.

Each FOF halo is decomposed by SUBFIND or SUBFIND_HBT into a set of disjoint gravitationally bound subhalos. They are nested inside the FOF group, again in an order of decreasing length. This means that the particles with the largest subhalo in a FOF group will

come first, followed by the second largest subhalo, and so on. Within each subhalo, the particles are additionally sorted according to their binding energy, i.e. the most bound particle in a subhalo will come first. Since not all particles within a FOF group need to be part of a subhalo, the sequence of subhalo particles is in general followed by a set of particles that are members of the FOF group but are not gravitationally bound to any of the subhalos. There may also be no subhalo for a given FOF group at all, meaning that there is no gravitationally bound subset of particles in the FOF group above detection threshold.

Format of group catalogues

The structure and organization of the group catalogues is quite similar to the snapshot files. They consist of different blocks that are stored subsequently in the binary files corresponding to formats 1 and 2, and in different data groups called Header, Groups, and Subhalos when HDF5 is selected. Like the snapshot files, the group catalogues can be split over multiple files (in which they are stored in separate `groupdir_XXX` directories), or they can be stored in a single file. If they contain only FOF information, they start with the basename `fof_tab` otherwise they start with the basename `fof_subhalo_tab`.

The most important fields of the Header in the group catalogues are:

Header Field	Type	HDF5 name	Comment
Ngroups	int64	Ngroups_ThisFile	number of groups in this file
Nsubhalos	int64	Nsubhalos_ThisFile	number of subhalos in the present file
Nids	int64	Nids_ThisFile	number of particles in groups in this file
NgroupsTot	int64	Ngroups_Total	total number of groups
NsubhalosTot	int64	Nsubhalos_Total	total number of subhalos
NidsTot	int64	Nids_Total	total number of particles in groups
NumFiles	int	NumFiles	number of subfiles of this catalogue
Time	double	Time	output time/scale factor
Redshift	double	Redshift	output redshift

FOF catalogue

The information about the FOF groups consists of the following blocks. They are effectively a table with properties for each FOF group. Some of the fields are only present if the FOF halos have also been processed with SUBFIND, such as the number of subhalos contained in a FOF halo, for example.

Nr	HDF5 Identifier	Fmt2-ID	Block contents
1	GroupPos		File header
2	GroupVel		Particle positions
3	GroupMass		Particle velocities

To locate a certain FOF halo in the corresponding snapshot file, one has to get the offset from the beginning of the file for each particle type, and then skip fast forward in the snapshot file to the corresponding starting position. One can then start reading there, taking the number of particles for the reported group length. This is then the FOF halo.

SUBFIND catalogue

The SUBFIND catalogue extends the group catalogue with additional blocks (i.e. datasets) that give further information for each subhalo. The total length of these entries is equal to the total number of subhalos given in the Header, i.e. TotNsubhalos. The entries in the catalogue are as follows:

Nr	HDF5 Identifier	Fmt2-ID	Block contents
1	SubhaloPos		File header
2	SubhaloVel		Particle positions
3	SubhaloMass		Particle velocities

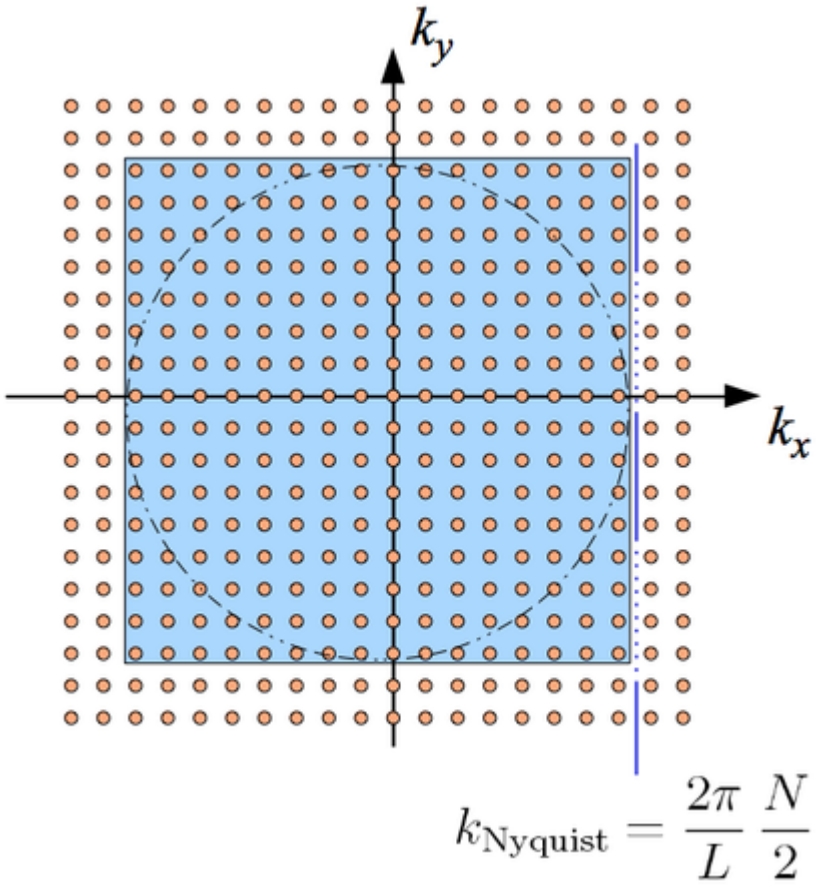
To read the particle data of an individual subhalo, one again needs to compute the correct file offset into the corresponding snapshot file. This is obtained here by first identifying the offset of the corresponding parent FOF group, and then computing an additional offset by summing up the lengths of all previous subhalos in the same FOF group. This needs to be done for each particle type separately. One can then skip towards the beginning of the corresponding subhalo and read the right number of particles there.

Special code features

The GADGET-4 code contains a number of modules that take the form of extensions of the code for specific science applications or common postprocessing tasks. Examples include merger-tree creation, lightcone outputs, or power spectrum measurements. Here we briefly describe the usage of the most important of these modules in GADGET-4.

Initial conditions

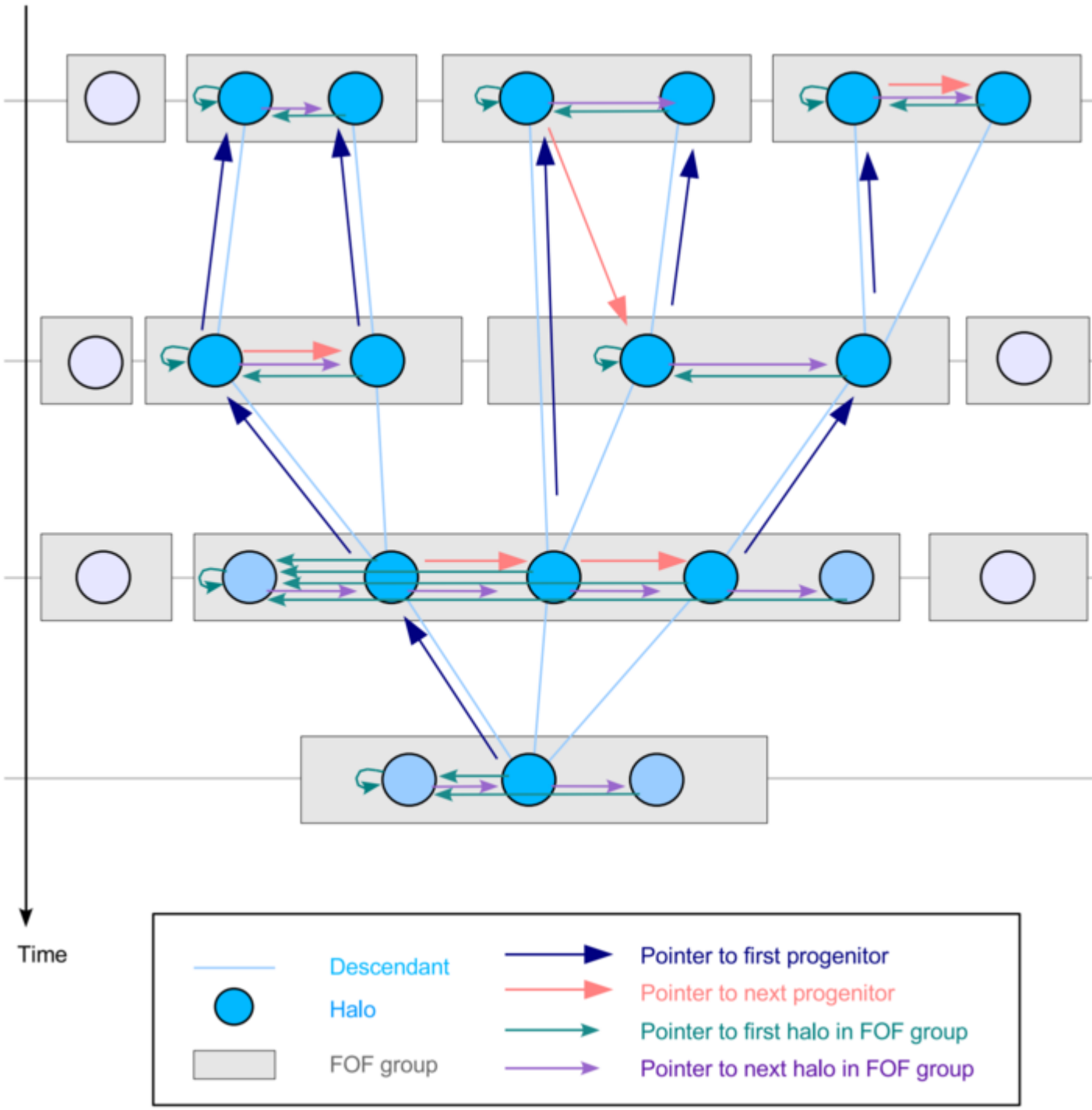
GADGET-4 contains a built-in initial conditions generator for cosmological simulations (based on the N-GenIC code), which supports both DM-only and DM plus gas simulations. Only cubical periodic boxes are supported at this point. Once the IC-module is compiled in (by setting `NGENIC` in the configuration), the code will create initial conditions upon regular start-up and then immediately start a simulation based on them. It is also possible to instruct the code to only create the ICs, store them in a file and then end, which is accomplished by launching the code with restartflag 6.



The NGENIC option needs to be set to the size of the FFTs used in the initial conditions creation, and the meaning of the other code parameters that are required for describing the initial conditions is described in detail in the relevant section of this guide.

Merger trees

The merger tree construction follows the concepts introduced in the paper Springel et al. (2005), http://adsabs.harvard.edu/abs/2005Natur.435..629S. It is a tree for subhalos identified within FOF groups, i.e. it requires group finding carried out with `FOF`, and `SUBFIND` or `SUBFINF_HBT`, and hence these options need to be enabled when `MERGERTREE` is set. The schematic organisation of the merger tree that is constructed is depicted in the following sketch:



At each output time, FOF groups are identified which contain one or several (sub)halos, and the merger tree connects these halos. The FOF groups play no direct role for the tree, except that the largest halo in a given FOF group is singled out as main subhalo in the group. To organize the tree(s), a number of pointers for each subhalo need to be defined.

Each halo must know its **descendant** in the subsequent group catalogue at later time, and the most important step in the merger tree construction is determining this link. This can be accomplished in two ways with GADGET-4. Either one enables `MERGERTREE` while a simulation is run. Then for each new snapshot that is produced, the descendant pointers for the previous group catalogue are computed as well and accumulated in the output directory. The results will be written in special files called `sub_desc_XXX`. In essence, these provide the glue

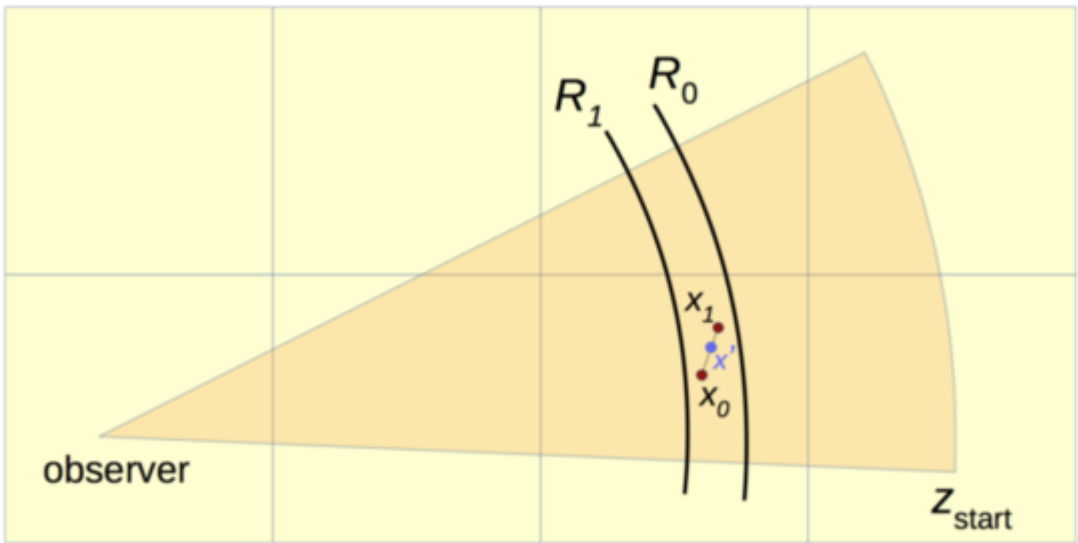
between two subsequent group catalogues. One advantage of doing this on the fly is that this allows merger tree constructions without ever having to output the particle data itself.

Alternatively, one also create these files in postprocessing for a simulation that was run without the `MERGERTREE` option. This however requires that snapshot files are available, or at the very least, that particle IDs have been included in the group catalogue output. The process of creating these link files can be accomplished with restartflag 7, which does this for the given snapshot number and the previous output. This has to be repeated for all snapshots except the first one (i.e. one starts at output number 1 until the last one) that should be part of the merger tree.

Finally, one can ask GADGET-4 to isolate individual trees and to arrange the corresponding subhalos in a format that allows easy processing of the trees, for example, in a semi-analytic code for galaxy formation. This process also computes the other links shown in the above sketch. To this end, one starts GADGET-4 with restartflag 8, and provides the last snapshot number as additional argument. GADGET-4 will then process all the group catalogue data and the descendant link files, and determine a new set of tree-files. The algorithms are written such that they are fully parallel and should be able to process extremely large simulations, with very large group catalogues and tree sets. The tree files will normally be split up over many files in this case, and the placing of a tree into any of these files is randomized in order to balance them roughly in size, which simplifies later processing. In order to quickly look-up based, based on a given subhalo number from one of the timeslices, in which tree this subhalo is found, corresponding pointers are added to the group catalogues as well.

Lightcone output

One new feature in GADGET-4 is the ability to output continuous light cones, i.e. particles are stored at the position and velocity at the moment the backwards lightcone passes over them. This is illustrated in the following sketch, which shows how the code determines an interpolated particle coordinate x' in between two endpoints of the timestepping procedure.



This option is activated with the `LIGHTCONE` switch, and needs to be active while the simulation is run. In this case, additional particle outputs are created, which have a structure similar to snapshot files.

While it is possible also here to use the file format 1 or 2, it is highly recommended to not bother with this but rather use HDF5 throughout for such more complicated output. This is the only

sensible way to not get caught up in struggles to parse the (possibly frequently varying) binary file format.

Power spectra

The GADGET-4 code can also be used to measure matter power spectra with a high dynamic range through the "folding technique", described in more detail in Springel et al. (2018) http://adsabs.harvard.edu/abs/2017arXiv170703397S. In essence, three power spectra are measured in each case, one for the unmodified periodic box, yielding a conventional measurement that extends up to close to the Nyquist frequency of the employed Fourier mesh (which is set by `PMGRID`). The other two are extensions to smaller scales by imposing periodicity on some inter division of the box, with the box folded on top of itself. The default value for this folding factor is `POWERSPEC_FOLDFAC=16` but this value can be modified if desired by overriding it with a configuration option.

The measured power spectra are outputted in a finely binned fashion in k-space as ASCII files. This data can be easily rebinned by band-averaging to any desired coarser binning (which then also reduces the statistical error for each bin), which is a task relegated to a plotting script. This can then also be used to combine the coarse and fine measurements into a single plot, and to do a shot-noise subtraction if desired. The shot-noise, allowing for variable particle masses if present, is also measured and output to the file. Example plotting scripts to parse the powerspectrum are provided in the code distribution.

There are two ways to measure the power spectra. This can either be done on the fly whenever a snapshot file is produced, by means of the `POWERSPEC_ON_OUTPUT` option. Or one can compute a power spectrum in postprocessing by applying the code with restartflag 4 to any of the snapshot numbers. In both cases, power spectra are measured both for the full particle distribution, and for every particle type that is present.

I/O bandwidth test

Another small feature of GADGET-4 is a stress test for the I/O subsystem of the target compute cluster. This is meant to get some information about the available I/O bandwidth for parallel write operations, and in particular, to find out whether `MaxFilesWithConcurrentIO` should be made smaller than the number of MPI-ranks for a specific setup to avoid that too many files being written at the same time, because this can be counter-productive in terms of throughput or cause a too high load on the I/O subsystem that inconveniences other users or jobs.

To this end, GADGET-4 can be started with the restartflag 9 option, using the same number of MPI ranks that is intended for a relevant production run. The code will then not actually carry out a simulation but instead carry out a number of systematic write tests. The tests are repeated for different settings of `MaxFilesWithConcurrentIO` , starting at the number of MPI ranks, and then halving this number until it drops below unity. For each of the tests, each MPI-rank tries to write 10 MB of data to files stored in the output directory (these are again deleted after the test automatically). The code then reports the effective I/O bandwidth reached for the different settings of `MaxFilesWithConcurrentIO`, and the results should inform about which setting is reasonable. In particular, in a regime where the I/O bandwidth only very weakly increases (i.e. strongly sub-linearly) with `MaxFilesWithConcurrentIO`, it will usually be better to go with a lower value where such linearity is still approximately seen to retain some responsiveness of the filesystem when GADGET-4 does parallel I/O.

Example Setups

To get an idea of some of the different possible usages of GADGET-4, and as a starting point for your own simulations and numerical experiments, we include a small set of examples with the code distribution. These can be found in the folder `examples`. Each of the examples has its own subdirectory containing the example's configuration and parameterfile, and sometimes auxiliary files for the specific runs.

To compile the code for one of the examples, you can either copy its configuration file `Config.sh` to the main code directory and run `make` there, or you execute `make` by passing it the `DIR` variable with a value that points to the subdirectory of the example. The executable for the example is then created right in the subdirectory of the corresponding problem, allowing you to run it right there. This is the recommended approach.

For convenience, the corresponding make-commands are contained in the file `make-examples.sh`, with one line with a suitable call of `make` for each of the examples. Copying the corresponding line and pasting it as a command into the main directory of the code will then build the executable for the example. Executing the full shell script will compile all the examples at once (provided there are no compilation problems, of course).

Below, we describe each of the examples and also give a few suggestions for setup-variants that could be of interest. Note that for some of the examples, you need to obtain initial conditions that are available as part of a separate IC-package (183 MB) on the GADGET-4 web-site. This also contains the ICs for the examples adopted from the GADGET-2 code distribution (these ICs are identical to those distributed with GADGET-2).

Cosmological DM-only simulation with IC creation

The setup in `DM-L50-N128` simulates a small box of comoving side-length 50 Mpc/h using 128^3 dark matter particles. The initial conditions are created on the fly upon start-up of the code, using second order Lagrangian perturbation theory with a starting redshift of z=63. The `LEAN` option and 32-bit arithmetic are enabled to minimize memory consumption of the code.

Gravity is computed with the TreePM algorithm at expansion order p=3. Three output times are defined, for which FOF group finding is enabled, and power spectra are computed as well for the snapshots that are produced. Also, the code is asked to compute a power spectrum for each output.

Aquarius Milky-Way Zoom

The directory `DM-Zoom-Aq-C-5` contains a setup for a cosmological zoom-simulation that follows the formation of a Milky Way-sized dark matter halo. To carry out this example, you need access to the `Aq-C-5-dm` initial conditions files, which stem from the Aquarius Project and were also used as part of the Aquila Project. These IC files are available as part of the IC-package on the GADGET-4 web-site.

The example uses particle type 1 for the high-resolution dark matter particles, while types 2 to 4 are employed for more massive boundary particles of increasingly higher mass. For computing the gravitational forces, the TreePM algorithm is used.

The code is instructed to create 16 dumps, with output times that are specified through an input file. Before each snapshot is written, the FOF group finding algorithm is run on the high-resolution particles, followed by SUBFIND, and the snapshot dumps will be stored according to group order.

Interesting variants of this setup:

- The default setup of this example does not enable a high resolution mesh, which could however be used in principle.
- If the resolution is as high and spartially concentrated as here, `HIERARCHICAL_GRAVITY` can also be of interest for these types of zoom simulations.
- If a merger tree is desired, simulations of this kind could be run with the `MERGERTREE` option.

Colliding galaxies with star formation

This simulation with setup in the folder `CollidingGalaxiesSFR` considers the collision of two compound galaxies made up of a dark matter halo, a stellar disk and bulge, and cold gas in the disk that undergoes star formation.

Radiative cooling due to helium and hydrogen is included. Star formation and feedback is modelled with a simple subgrid treatment. The simulation corresponds closely to the model of a galaxy collisions considered in the code paper.

Santa Barbara cluster

The Santa Barbara cluster is a hydrodynamical simulation of the formation of a galaxy cluster, which was introduced originally in the code comparison paper by Frenk et al. (1999).

In this example, we consider the problem at 2 x 64^3 resolution, using the pressure-based formulation of SPH (for the sake of change, and not because we think this is necessarily to be recommended for this problem).

Old examples from GADGET-2

Galaxy collision

This purely collisionless simulation in `G2-galaxy` runs two disk galaxies into each other, leading to a merger between the galaxies. Each galaxy consists of a stellar disk, and a massive and extended dark matter halo. This example uses plain Newtonian physics, with 20000 disk and 40000 halo particles in total.

The setup provided corresponds to a traditional tree algorithm with ordinary timestepping. Alternatives to this now possible with GADGET-4 include the use the FMM algorithm for gravity, and higher order multipole expansion. Note that this example from GADGET-2 is almost trivially small, and some of these alternatives will only show their real advantages in much larger simulations of higher resolution.

To get a first idea whether the example has worked you may check for energy conservation by analysing the log-file `energy.txt`. A simple example for doing this is provided in the form of the IDL script file `plot_energy.pro`.

Adiabatic collapse of a gas sphere

This simulation in `G2-gassphere` considers the gravitational collapse of a self-gravitating sphere of gas which initially has a 1/r density profile and a very low temperature. The gas falls under its own weight to the centre, where it bounces back and a strong shock wave that moves outwards develops. This common test problem of SPH codes has first been described by Gus Evrard.

The simulation uses Newtonian physics in a natural system of units (G=1). The setup corresponds to vanilla density-based SPH with the entropy formulation introduced by Springel & Hernquist (2002). You can use the IDL-script `plot_energy_gassphere.pro` to display the evolution of thermal, kinetic and potential energy for the collapsing gassphere. Note that this is a really tiny simulation of just 1472 particles.

Cosmological formation of a cluster of galaxies

This problem in `G2-cluster` uses collisionless dynamics in an expanding universe. It is a small cluster simulation that has been set-up (a long time ago) with Bepi Tormen's initial conditions generator ZIC using vacuum boundaries and a multi-mass technique. The simulation has a total of 276498 particles. In a central high-resolution zone there are 140005 particles, surrounded by a boundary region with two layers of different softening, the inner one containing 39616 particles, and the outer one 96877 particles.

Note that while this simulation is a cosmological simulation in comoving coordinates, it is unusual in that it doesn't use periodic boundary conditions but rather follows a sphere of matter around the origin with average density equal to the mean density. This technique is not very commonly used any more.

Large-scale structure formation including gas

This problem in `G2-lcdm-gas` consists of 32^3 dark matter, and 32^3 gas particles, following structure formation in a periodic box of 50 Mpc/h on a side in a LCDM universe. Only adiabatic gas physics is included, and the minimum temperature of the gas is set to 1000 K. This simple example uses grid initial conditions, where gas particles are put at the centres of the grid outlined by the dark matter particles. The simulation starts at z=10, and the code will produce snapshot files at redshifts 5, 3, 2, 1, and 0. As in the other old GADGET-2 examples, the SPH setup is density-based SPH based on the entropy formation.

Guide to code changes

In the following, we give a set of assorted hints and recommendations about modifying and/or extending the code. We also comment about some differences with respect to GADGET-2/3 in terms of code usage and its architecture.

Coding style guide

We strongly recommend to follow the general coding practices (even if you don't like them) and architecture of GADGET-4 in modifying or extending the code. Only then different extensions by different people have a chance to happily coexist with each other. This concerns mostly the following points:

Code extensions

- Non-standard code extensions should always be written such that they can be switched off if not needed, and have no side effects on existing code when this is done. Normally this means that they have to be enclosed in conditional compilation precompiler statements (#ifdef), especially if variables in the global structures of the code need to be allocated for the extension. However, if the extension's execution can also be controlled at run-time by simple variables, then consider introducing a parameterfile variable to control the extension. In general, the number of symbols (and this additional `Config.sh` options) to control conditional compilation should be kept to a minimum.

- Do not place any substantial piece of code belonging to your extension into existing functions of the code. Write your own classes or functions for a code extension, and only place a function call (if needed bracketed by an #ifdef) into the appropriate place of the primary code. Also, place your extension functions into separate source files.

General code-style principles

- Code formatting: Be consistent with the code formatting of the main code, which is more or less GNU-style, and is obtained by running the code formatting tool "clang-format" of the clang compiler suite. The specific options used by GADGET-4 are defined in the hidden file ".clang-format" in the code's main directory. Running there something like

```
clang-format-mp-6.0 -style=file -i src/*/*
```

should then make sure that all your source file(s) have a consistent indention and code formatting.

- Name functions all in lower case as a "command" that is descriptive of what the function does. Different words should normally be separated by underscores, e.g. calculate_normal_vector_for_triangle(...) For all functions, input arguments should come first, output arguments last.

- Global variables (whose use should be kept to an absolute minimum, and the sins of GADGET-4 in this regard should not be repeated) start with an upper case character. Global variable names are nouns, with words separated by mixed lower/upper case characters, and contain no underscores. Use abbreviations that are clear, e.g. `NumForceCalculations`. If you have to use a global variable in your own code class or module, try to restrict its scope to the classes or files of your module through appropriate declarations.

- Local variables start with lowercase, and should have descriptive names, too, except for simple loop iterators and the like. Try to narrow the scope of a variable as much as possible, for example by declaring them inside a block where they are needed. Generally define them as close as possible to where they are needed for the first time. Declaration and initialization should be combined in one command where possible, e.g.

```
int n = get_particle_count();
```

instead of

```
int n;
  ...
n = get_particle_count();
```

- Avoid repetition of code, i.e. do not use cut & paste to implement the same or similar functionality multiple times. This is always an indication that a new function is in order here. Break up long functions into smaller more manageable pieces.

- Preprocessor macros that have arguments should be avoided whenever possible, and really should only be used for a compelling reason. Note that in C++ you can use inlined and type-safe functions instead. In any case, preprocessor macros should be fully capitalized.

- Magic numbers and non-trivial numerical constants should be avoided in the actual code and instead be replaced by a symbolic constant declared within a header file, with a name all in uppercase. Consider enums instead of the use of numerical constants to distinguish between different cases.

- All warnings emitted by the compiler upon compilation with "-Wall" should be addressed. Unless there are extremely good reasons, the code should compile without any warnings left.

- Include consistent commenting in your code. The meaning of all global variables should be commented where they are introduced, ideally with doxygen syntax, e.g:

```
int MyGlobalCount;   /*!< counts the number of timesteps */
```

- Functions should be preceded by a brief explanation of what the function does, including instruction, if any, about how the function may be used, e.g.:

```
/*!
 * Insert the point P[i] into the partice list. Start
 * the search at the current particle t.
 */
 int insert_point(int i, int t)
 {
   ...
 }
```

- You do not need to state here *how* the function achieves what it does; this can be stated if appropriate in comments in the function body. There, avoid superfluous comments that just reflect what's obvious from the code anyway, like

```
do_domain_decomposition();   /* call domain decomposition */
```

- Instead, focus on comments that help one to quickly understand/check what the code tries to do at an algorithmic level. If complicated formulae are implemented, try to include in a comment a reference to the equation that is implemented.

Adding a config-option

To add a new symbolic configuration option to the code, you need to add it to the file `Template-Config.sh` besides just using it in the code. In addition, you must add a short explanation of the option to the file `documentation/04_config-options.md` following the syntax used for the other configuration option. Only then the checking scripts of the source code will be happy and accept the new option. Note that for this the option really needs to be used somewhere in the source code that is described as belonging to GADGET-4 by the `Makefile`.

Adding parameters

To add a new parameter to the parameterfile of GADGET-4, the following steps are necessary:

- Add the new variable to the structure `global_data_all_processes` in the file `data/allvars.h`. The variable has to be of type double, int, or a character string.
- In the file `io/parameters.c` , add a function call to add_param(...) for the new parameter in the function `register_parameters()` . Simply follow the examples for the other parameters. If your parameter is optional and should only be active when a specific option is activated, bracket the add_param() call with a corresponding #ifdef/ #endif
- Add a short explanation of the new parameter in the file `documentation/05_parameterfile.md` This is needed otherwise the code checking scripts will not accept the new parameter.

Once these steps are followed, the new parameter will be included in the parsing of the parameterfile, and will also be stored in restart files as well as in HDF5 output files.

Adding a source file

To add a new source file, create it in a subdirectory of `src/` , not in `src/` itself. Preferably you create your own subdirectory for this, say `src/mymodule` Then in the `Makefile` of the code, add appropriate statements that list the file(s) and header file(s) belonging to your new source file. For example, something like this:

```
SUBDIRS += mymodule
OBJS    += mymodule/mysource1.o  mymodule/mysource2.o
INCL    += mymodule/mymodule.h
```

Your new source file should also come with a header file, at the very least containing a function prototype for one of your new functions that is intended to be called from the main code. This header file can then be included in the main code at an appropriate place to facilitate a call to your new code. The header file should be protected by a header file guard, i.e. you should add something like

```
#ifndef MYMODULE_H
#define MYMODULE_H
```

at the beginning of the header file, and

```
#endif
```

at the end. To tell the code checks scripts that the constant `MYMODULE_H` is not an undocumented code configuration parameter, you also need to add it to the file `defines_extra` in the main directory of the code. Importantly, you should also add your new source file to version control system. Ideally, you make a git-branch of GADGET-4 and then implement your extension/ modification in this branch. Try to keep the branch current with the main line of the code as long as it is still kept alive as a separate branch. If possible, try to integrate your feature into a main line of development for GADGET-4, for example through a pull request, and if successful close your branch again. To update the html-documentation of the code with your new configuration and parameterfile options, and to also include your source in the cross-referenced source code documentation, run the command `doxygen` in the main directory of the code. (If doxygen should be unavailable on your local machine, you can always easily install it.)

Migration to GADGET4

Main changes in the code

There have been many major changes in the internal workings between previous versions of GADGET and GADGET-4, affecting nearly all parts of the code. In fact, basically all parts of the code have been completely rewritten since GADGET-3 (sometimes several times), or were in any case substantially revised. These modifications were done to improve the accuracy and the performance of the code, to reduce its memory consumption, and to make it applicable to a wider range of simulation types. Detailed explanations of the reasoning behind each of these modifications is beyond the scope of this short set of notes. Some of this information can be however found in the code paper on GADGET-4.

Some of the most important changes in code design include:

- A new hierarchical timestepping option for gravity
- An optional FMM solver for gravity
- Availability of hybrid parallelization with shared memory use
- Support for very large simulation sizes
- Stretched boxes also for gravity, as well as mixed boundary conditions for gravity
- Inclusion of FOF and SUBFIND in the public code version, a new parallelization scheme for SUBFIND, and the addition of SUBFIND_HBT
- Inclusion of a power spectrum estimator
- An optional column-based FFT can be used
- Inclusion of an updated version of the N-GenIC initial conditions generator that also supports 2LPT
- Explicit vectorization support in some parts of the code
- New domain decomposition algorithm
- Support of two formulations of SPH

On-the-fly merger tree construction

- Continuous light-cone output
- Integration of a simple cooling and star formation formulation in the public version of the code

Migrating old set-ups to GADGET-4

If you have used earlier version of GADGET before, you will be familiar with all the practicalities of running GADGET already. This is because nothing much has really changed here in GADGET-4. In particular, the mechanisms for starting, interrupting, and resuming a simulation are still the same.

However, as described in this guide, there have been a number of changes in the parameterfile that controls each simulation, and there are more compilation options that are now more conveniently controlled through the `Config.sh` file. If you want to reuse an old parameterfile with GADGET-4, you therefore have to make a few modifications of it. If you simply start the new code with the old parameterfile, you will get error messages that complain about obsolete or missing parameters. Delete or add these parameters as needed. Also, carefully review which makefile options are still appropriate for a given simulation. While many of the basic compile time options available in previous versions of GADGET are still around, others have become obsolete. The code will complain about options that are set in `Config.sh` but don't exist any more, so you should be automatically informed about this when you try to use an old setup. Importantly, however, the structure and format of the snapshot files produced by the code has hardly changed (and the system of units has not changed at all), meaning that you should still be able to use old analysis software from previous versions of GADGET essentially unchanged.

Notes on memory consumption of GADGET-4

GADGET-4 uses an internal memory manager where a large block of memory is allocated once at the beginning of the simulation, and the code then satisfies all its internal allocations out of this block. The size of this block corresponds to the maximum allowed memory usage per MPI process, and it is set by the parameter file `MaxMemSize`.

The code will periodically output information to the file `memory.txt` about all allocated memory blocks and their sizes. This happens whenever a new high watermark in the memory allocation is reached. It is best to check this table to get an accurate assessment of the amount of memory the code at least needs to run successfully for a certain particle number. For communication and particle exchanges, the code will use buffers that automatically adjust their size to the amount of still available memory. It is hence advantageous to set `MaxMemSize` as large as possible for the amount of physical memory available on the compute nodes. An overcommitment of the physical memory of a node is prevented by the code (it then exists with an error message), but this check only works if `HOST_MEMORY_REPORTING` is enabled. At the beginning of the stdout-file, you can also find the sizes of the most important code structures for the chosen configuration. In particular this tells about how many bytes per collisionless particle are needed, and how many bytes are in addition needed to store one SPH particle.

Notes on various limits in GADGET-4

- Maximum number of particles per MPI-rank is restricted to 2^31 ~ 2 Billion
- Maximum number of groups/subhalos per MPI-rank is restricted to 2^31 ~ 2 Billion
- The total number of particles, and the total number of groups/subhalos, can however be much larger than this if a sufficiently large number of MPI ranks is used.
- Maximum particle number in a single group or subhalo is for the default setting 2^31 ~ 2 billion, but this can be enlarged if needed by setting the option `FOF_ALLOW_HUGE_GROUPLENGTH`
- The maximum number of subhalos per group is restricted nevertheless to 2^31 ~ 2 billion.
- If the legacy output formats 1/2 are used, each block in the output is restricted to 2 Gbyte in size or less. This also means that the maximum number of particles per single output file, and the maximum number of groups/subhalos per single group catalogue file can be at most of order 2 Billion, normally substantially less than that. While this restriction can be circumvented by splitting output over enough files, it provides one further motivation to use the HDF5 format!
- Maximum number of lightcones: 256