User guide for GADGET-2

Volker Springel

volker@mpa-garching.mpg.de

Max-Plank-Institute for Astrophysics, Garching, Germany

May 1, 2005

Contents

1 Introduction

In its current implementation, the simulation code GADGET-2 (**GA**laxies with **D**ark matter and **G**as int**E**rac**T**[1]) supports collisionless simulations and smoothed particle hydrodynamics on massively parallel computers. All communication is done explicitly by means of the message passing interface (MPI). The code is written in standard ANSI C, and should run on all parallel platforms that support MPI. So far, the portability of the code has been confirmed on a large number of systems, including Cray-T3E, IBM-SP/2 and p690 systems, SGI Origins, various systems from Sun, as well as a large number of Linux PC clusters ('beowulfs'), based either on Intel or AMD processors.

The code can be used for plain Newtonian dynamics, or for cosmological integrations in arbitrary cosmologies, both with or without periodic boundary conditions. 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.

The main reference for numerical and algorithmic aspects of the code is the paper '*The cosmological simulation code GADGET-2*' (Springel, 2005, MNRAS, submitted), and references therein. Further information on the previous public version GADGET-1 is also contained in the paper '*GADGET: A code for collisionless and gas-dynamical cosmological simulations*' (Springel, Yoshida & White, 2001, New Astronomy, 6, 51). In the following, these papers will be frequently referenced, and it is recommended to read them before attempting to use the code. This document provides additional technical information about the code, hopefully in a sufficiently self-contained fashion to allow anyone interested an independent use of the code for cosmological N-body/SPH simulations on parallel machines.

GADGET-2 was written by Volker Springel, and 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 author and the Max-Planck-Institute for Astrophysics (MPA). If you find the code useful for your scientific work, we kindly ask you to include a reference to the code paper on GADGET-2 in all studies that use simulations carried out with GADGET-2.

2 Basic usage

2.1 Compilation requirements

GADGET-2 needs the following non-standard libraries for compilation:

- **mpi** - the 'Message Passing Interface' (version 1.0 or higher). Many vendor supplied versions exist, in addition to excellent open source implementations, e.g.
  MPICH (`http://www-unix.mcs.anl.gov/mpi/mpich`), or
  LAM (`http://www.lam-mpi.org`).

- **gsl** - the GNU scientific library. This open-source package can be obtained at `http://www.gnu.org/software/gsl`, for example. GADGET-2 only needs this

[1]This peculiar acronym hints at the code's origin as a tool for studying galaxy collisions.

library for a few very simple cosmological integrations at start-up.

- **fftw** - 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 2.x of FFTW is required, the new version 3 lacks MPI capability at this point. FFTW is only needed for simulations that use the TreePM algorithm.

- **hdf5** - the 'Hierarchical Data Format' (version 5.0 or higher, available at `http://hdf.ncsa.uiuc.edu/HDF5`). This optional library is only needed when one wants to read or write snapshot files in HDF format. It is possible to compile and use the code without this library.

Note that FFTW needs to be compiled with parallel support enabled. This can be achieved by passing the option `--enable-mpi` to its configure script. When at it, you might as well add `--enable-type-prefix` to obtain the libraries in both a single and double precision version. The single-precision version can then be generated after the double precision one by a `make clean` command, followed by a reconfiguration that also includes the `--enable-float` option and a renewed `make install`. Note that without the `--enable-type-prefix` option in the compilation of FFTW, you should set the option `NOTYPEPREFIX_FFTW` in GADGET-2's makefile to generate correct header and library names.

After unpacking the tar-file of GADGET-2, you will find a bunch of `.c`-files, `.h`-files, and a `Makefile`. You should edit the `Makefile` and adjust it to your system, if needed. A number of systems have been predefined in the `Makefile` (look at the `SYSTYPE` definitions), and by following these examples, further customised system-types can be easily added. Note that slight adjustments of the makefile will be needed if any of the above libraries is not installed in a standard location on your system. Also, compiler optimisation options may need adjustment, depending on the C-compiler that is used. The provided makefile is compatible with GNU-make, i.e. typing `make` or `gmake` should then build the executable `Gadget2`. If your site does not have GNU-make, get it, or modify the makefile accordingly.

The makefile also contains a number of compile-time options which determine the types of simulations that can be run with a given GADGET-2 executable. These options are explained in Section 4. Further code options of GADGET-2 are controlled by a parameter file, which is described in Section 5.

2.2 Starting the code

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

```
mpirun -np 32 ./Gadget2 myparameterfile.param
```

This will start the simulation using 32 processors, and with simulation parameters as specified in the parameter file (see below) of name `myparameterfile.param`. Note that on some systems, the `mpirun` command may have a different syntax. 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-2 with a single processor only. In this case, the leading ‘`mpirun -np 1`’ can be omitted, and GADGET-2 behaves like a serial code. It is still necessary to have MPI installed, though, because even then the code will make some (superfluous) calls to the MPI library. For GADGET-1, two separate code versions for serial and parallel mode existed, GADGET-2 uses a single code base to take care of both functionalities. Also, in GADGET-2, there is no restriction anymore for the processor number to be a power of two, even though these partition sizes are in general prefered, because they allow the most efficient communication schemes.

While GADGET-2 is run, it will print out a bunch of 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 at a supercomputing centre, you will usually have to put the above command into a script-file that is submitted to the computing-queue. In this case, the standard output of GADGET-2 should be automatically piped into a file.

2.3 Interrupting a run

Usually, a single submission to a queue system will not provide enough CPU-time to process a big production run. Therefore, a running simulation can be interrupted after any 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 expires, and write a set of ‘restart’-files. Actually, each processor writes its own restart file. These restart-files can be used to continue the run conveniently (see below).

Sometimes you may want to interrupt the code manually with the possibility to resume it later on. 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 is completed, and the file ‘`stop`’ will be erased automatically. Restart files are also generated when the last timestep of a simulation has been completed. They can be used if one later wants to extend the simulation beyond the original final time.

2.4 Restarting a run

2.4.1 Restarting from restart-files

To resume a run from restart-files, start the code with an optional flag in the form

```
mpirun -np 32 ./Gadget2 myparameterfile.param 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 had not been interrupted to begin with. In GADGET-1, this was not the case in general, because a restart always forced the occurrence of a new domain decomposition and tree construction right after the simulation was resumed.

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 Section 5 about the parameterfile.

It is **important** to not forget the `1` if you want to restart – 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 processors used for a simulation cannot be changed. Also note that restart files can in general not be transfered to another system, and that their structure is completely different than that of snapshot files. If you want to continue a simulation on another system or with a different number of processors, restarting from a snapshot file is the method of choice. This will be discussed next.

2.4.2 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 the run from the snapshot-file (without the optional `1`).

An alternative to this somewhat inconvenient 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`. In this case, you only have to change the name of the initial conditions file and make it equal to the name of the snapshot you want to start from, other parameters in the parameterfile need not to be changed. In particular, the code will continue the numbering of all further snapshots starting with 1 plus the number of the snapshot that is used as input file. When you specify a snapshot file that was distributed onto several files (e.g. `dump_007.0 to dump_007.7`) you should only give the base-name (i.e. here `dump_007`) as initial conditions file when using this option.

Note that the restart from snapshot files 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 the preferred method to resume a simulation, because it is much faster, and it minimises possible perturbations in the time integration scheme of a running simulation (restarting from a snapshot forces the individual timestep scheme to be resynchronised). Because in general not all the particles are synchronised at the time of writing a snapshot file (the others are first-order predicted to the output time), a small perturbation in the time integration is introduced when restarting from a snapshot file.

3 Types of simulations

Compared with GADGET-1, there are a number of new types of simulations that can be run with GADGET-2, most importantly variants of the TreePM algorithm. A schematic overview of these simulation types is given in Table 1.

Cosmological integrations with comoving coordinates are selected via the parameter `ComovingIntegrationOn` in the parameterfile. Otherwise the simulations always assume ordinary Newtonian space. Periodic boundary conditions and the various variants of the TreePM algorithm require compile-time switches to be set appropriately in the makefile.

In particular, the TreePM algorithm is switched on by passing the desired mesh-size at compile time via the makefile to the code. The relevant parameter is `PMGRID`, see below. Using an explicit force split, the long-range force is then computed with Fourier techniques, while the short-range force is done 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. Periodic and non-periodic boundary conditions are implemented for the TreePM approach. In the latter case, the code will internally compute FFTs of size `2*PMGRID` in order to carry out the required zero-padding. For zoom-simulations, it is possible to let the code automatically place a refined mesh-layer on the high-resolution region.

Pure SPH simulations in periodic boxes can also be run in periodic boxes whose dimensions in each direction are multiples of the basic boxsize. Such simulations can only be done without self-gravity at the moment.

Finally, it is possible to run SPH runs (also with self-gravity) in two dimensions only. However, the latter feature is provided for test-runs only, and is not optimised; here three coordinates are still stored for all particles and the computations are formally carried out as in the 3D case, except that all particles lie in one coordinate plane, i.e. either have equal x-, y-, or z-coordinates.

4 Makefile options

Many aspects of the new features of GADGET-2 are controlled with compile-time options in the makefile rather than run-time options in the parameterfile. This was done in order to allow the generation of highly optimised binaries by the compiler, even when the underlying source allows for many different ways to run the code. Unfortunately, this technique has the disadvantage that different simulations may require different binary executables of GADGET-2. If several simulations are run concurrently, there is hence the danger that a simulation is started or resumed with the 'wrong' binary. Note that while GADGET-2 checks the plausibility of some of the most important code options, this is not done for all of them. To minimise the risk of using the wrong code for a simulation, my recommendation is therefore to produce a separate executable for each simulation that is run. A good strategy for doing this in practice is to make a copy of the whole simulation source code together with its makefile in the output directory of each simulation run, and then use this copy to compile the code and to run the simulation. The code and its settings become then a logical part of the output generated in the simulation directory.

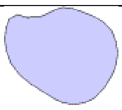
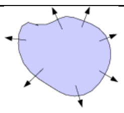
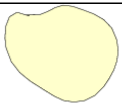
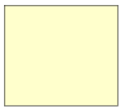
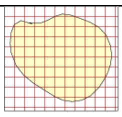
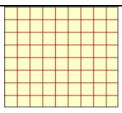
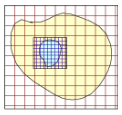
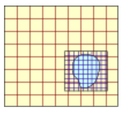
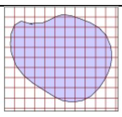
	Type of Simulation		Computational methods	Remarks
1	**Newtonian space**		Gravity: Tree, SPH (optional), vacuum boundary conditions	OmegaLambda should be set to zero
2	**Periodic long box**		No gravity, only SPH, periodic boundary conditions	NOGRAVITY needs to be set, LONG_X/Y/Z may be set to scale the dimensions of the box
3	**Cosmological, physical coordinates**		Gravity: Tree, SPH, vacuum boundaries	ComovingIntegrationOn set to zero
4	**Cosmological, comoving coordinates**		Gravity: Tree, SPH, vacuum boundaries	ComovingIntegrationOn set to one
5	**Cosmological, comoving periodic box**		Gravity: Tree with Ewald-correction, SPH, periodic boundaries	PERIODIC needs to be set
6	**Cosmological, comoving coordinates, TreePM**		Gravity: Tree with long range PM, SPH, vacuum boundaries	PMGRID needs to be set
7	**Cosmological, comoving periodic box, TreePM**		Gravity: Tree with long range PM, SPH, periodic boundaries	PERIODIC and PMGRID need to be set
8	**Cosmological, comoving coordinates, TreePM, Zoom**		Gravity: Tree with long-range and intermediate-range PM, SPH, vacuum boundaries	PMGRID and PLACEHIGHRESREGION need to be set
9	**Cosmological, periodic comoving box, TreePM, Zoom**		Gravity: Tree with long-range and intermediate-range PM, SPH, periodic boundaries	PERIODIC, PMGRID and PLACEHIGHRESREGION need to be set
10	**Newtonian space, TreePM**		Gravity: Tree with long-range PM, SPH, vacuum boundaries	PMGRID needs to be set

Table 1: Schematic overview of the different types of simulations possible with GADGET-2.

The makefile 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 parameter should be commented in, and given the desired value, where appropriate. At the beginning of the makefile, there is also a brief explanation of each of the options.

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

4.1 Basic operation mode of code

`PERIODIC`

Set this if you want to have periodic boundary conditions.

`UNEQUALSOFTENINGS`

Set this if you use particles with different gravitational softening lengths.

4.2 Things that are always recommended

`PEANOHILBERT`

This is a tuning option. When set, the code will bring the particles into Peano-Hilbert order after each domain decomposition. This improves cache utilisation and performance.

`WALLCLOCK`

If set, a wallclock timer is used by the code to measure internal time consumption (see cpu-log file). Otherwise, a timer that measures consumed processor ticks is used.

4.3 TreePM options

`PMGRID=128`

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. 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, using a mesh with dimension twice that specified by PMGRID.

`PLACEHIGHRESREGION=1+8`

If this option is set (will only work together with PMGRID), then the long range force is computed in two stages: One Fourier-grid is used to cover the whole simulation volume, allowing the computation of the large-scale force. A second Fourier mesh is placed on the region occupied by 'high-resolution' particles, allowing the computation of an intermediate-scale force. Finally, the force on very small scales is computed by the tree. 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 option, the parameter needs to be set to an integer that encodes the particle types that make up the high-res particles in the form of a bit mask. For example, if types 0, 1, and 4 are the high-res particles, then the parameter should be set to PLACEHIGHRESREGION=1+2+16, i.e. to the sum $2^0 + 2^1 + 2^4$. The spatial region covered by the high-res grid is determined automatically from the initial conditions. Note: If a periodic box is used, the high-res zone is not allowed to intersect the box boundaries.

`ENLARGEREGION=1.1`

The spatial region covered by the high-res zone normally has a fixed size during the simulation, which initially is set to the smallest region that encompasses all high-res particles. Normally, the simulation will be interrupted if high-res particles leave this region in the course of the run. However, by setting this parameter to a value larger than one, the high-res region can be expanded. For example, setting it to 1.4 will enlarge its side-length by 40% (it remains centred on the high-res particles). Hence, with such a setting, the high-res region may expand or move by a limited amount. If in addition SYNCHRONIZATION is activated, then the code will be able to continue even if high-res particles leave the initial high-res grid. In this case, the code will update the size and position of the grid that is placed onto the high-resolution region automatically. To prevent that this potentially happens every single PM step, one should nevertheless assign a value slightly larger than 1 to ENLARGEREGION.

`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 mesh-cells.

`RCUT=4.5`

This can be used to override the maximum radius in which the short-range tree-force is evaluated (in case the TreePM algorithm is used). The default value is 4.5, given in mesh-cells.

4.4 Single/double precision

`DOUBLEPRECISION`

This makes the code store and compute internal particle data in double precision. Note that output files are nevertheless written by converting the values that are saved to single precision.

`DOUBLEPRECISION_FFTW`

If this is set, the code will use the double-precision version of FTTW, provided the latter has been explicitly installed with a ”d” prefix, and NOTYPEPREFIX_FFTW is not set. Otherwise the single precision version (”s” prefix) is used.

4.5 Time integration options

`SYNCHRONIZATION`

When this is set, particles may only increase their timestep if the new timestep will put them into synchronisation with the higher time level. This typically means that only on half of the timesteps of a particle an increase of its step may occur. Especially for TreePM runs, it is usually advisable to set this option.

`FLEXSTEPS`

This is an alternative to SYNCHRONIZATION. Particle timesteps are here allowed to be integer multiples of the minimum timestep that occurs among the particles, which in turn is rounded down to the nearest power-of-two devision of the total simulated time-span. This option distributes particles more evenly over individual system timesteps, particularly once a simulation has run for a while, and may then result in a reduction of work-load imbalance losses.

`PSEUDOSYMMETRIC`

When this option is set, the code will try to ‘anticipate’ timestep changes by extrapolating the change of the acceleration into the future. This can sometimes improve the long-term integration behaviour of periodic orbits, because then the adaptive integration becomes more akin to a time reversible integrator. Use this option with care. Note: This option has no effect if FLEXSTEPS is set.

`NOSTOP_WHEN_BELOW_MINTIMESTEP`

If this is activated, the code will not terminate when the timestep falls below the value of MinSizeTimestep specified in the parameterfile. This is useful for runs where one wants to enforce a constant timestep for all particles. This can be done by activating this option, and by setting MinSizeTimestep and MaxSizeTimestep to an equal value.

NOPMSTEPADJUSTMENT

When this is set, the long-range timestep for the PM force computation is always determined by MaxSizeTimeStep. Otherwise, it is set to the minimum of MaxSizeTimeStep and the timestep obtained for the maximum long-range force with an effective softening scale equal to the PM smoothing-scale.

4.6 Output options

HAVE_HDF5

If this is set, the code will be compiled with support for input and output in the HDF5 format. You need to have the HDF5 libraries and headers installed on your computer for this option to work. The HDF5 format can then be selected as format ”3” in Gadget’s parameterfile.

OUTPUTPOTENTIAL

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 costs additional time.

OUTPUTACCELERATION

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

OUTPUTCHANGEOFENTROPY

This will include the rate of change of entropy of gas particles in snapshot files.

OUTPUTTIMESTEP

This will include the timesteps actually taken by each particle in the snapshot files.

4.7 Things for special behaviour

NOGRAVITY

This switches off gravity. Makes only sense for pure SPH simulations in non-expanding space.

NOTREERND

If this is not set, the tree construction will succeed even when there are a few particles at identical locations. This is done by ‘rerouting’ particles once the node-size has fallen below 10^{-3} of the softening length. However, when this option is activated, this will be suppressed and the tree construction will always fail if there are particles at identical or extremely close coordinates.

`NOTYPEPREFIX_FFTW`

If this is set, the fftw-header/libraries are accessed without type prefix (adopting whatever was chosen as default at compile-time of fftw). Otherwise, the type prefix 'd' for double-precision is used.

`LONG_X/Y/Z`

These options can only be used together with PERIODIC and NOGRAVITY. When set, the options define numerical factors that can be used to distort the periodic simulation cube into a parallelepiped of arbitrary aspect ratio. This can be useful for idealised SPH tests.

`TWODIMS`

This effectively switches of one dimension in SPH, i.e. the code follows only 2d hydrodynamics in the xy-, yz-, or xz-plane. This only works with NOGRAVITY and if all coordinates of one of the axis are exactly equal. Can be useful for idealised SPH tests.

`SPH_BND_PARTICLES`

If this is set, particles with a particle-ID equal to zero do not receive any SPH acceleration. This can be useful for idealised SPH tests, where these particles represent fixed 'walls'.

`NOVISCOSITYLIMITER`

If this is set, there is no explicit upper limit on the viscosity. In the default version, this limiter will try to protect against possible particle 'reflections', which may in principle occur for poor timestepping in the presence of strong shocks.

`COMPUTE_POTENTIAL_ENERGY`

When this option is set, the code will compute the gravitational potential energy each time a global statistics is computed. This can be useful for testing global energy conservation.

`ISOTHERM_EQS`

This special option makes the gas behave like an isothermal gas with equation of state $P = c_s^2\rho$. The sound-speed squared, c_s^2, is set *instead of* the thermal energy per unit mass in the initial conditions, i.e. you need to specify c_s^2 in the block that is usually reserved for u. The generated snapshot files will also contain the squared sound speed in this field. If the initial value for u is zero, then the initial gas temperature in the parameter file is used to define the sound speed according to $c_s^2 = kT/m_p$, where m_p is the proton mass.

`ADAPTIVE_GRAVSOFT_FORGAS`

When this option is set, the gravitational softening lengths used for gas particles is tied to their SPH smoothing length. This can be useful for dissipative collapse simulations. The option requires the setting of UNEQUALSOFTENINGS.

`SELECTIVE_NO_GRAVITY`

This can be used for special computations where one wants to exclude certain particle types from receiving gravitational forces. The particle types that are excluded in this fashion are specified by a bit mask, in the same way as for the PLACEHIGHRESREGION option.

`LONGIDS`

If this is set, the code assumes that particle-IDs are stored as 64-bit long integers. This is only really needed if you want to go beyond $\sim$ 2 billion particles.

4.8 Testing and debugging options

`FORCETEST=0.01`

This can be set to check the force accuracy of the code, and is only included as a debugging option. The option should be set to a number between 0 and 1 (e.g. 0.01), which specifies the fraction of randomly chosen particles for which at each timestep 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: Particle IDs must be set to numbers $>=$1 for this option to work.

4.9 Glass making

`MAKEGLASS=262144`

This option can be used to generate a glass-like particle configuration. The value assigned gives the particle load, which is initially generated as a Poisson sample and then evolved towards a glass with the sign of gravity reversed.

5 Parameterfile

Many code features 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) from the keyword. 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 sequence of keywords is arbitrary, but each keywords 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 '%' are ignored. In lines with keywords, comments may also be added after the specification of the value for the corresponding keyword.

In the following, each keyword and the meaning of its value are discussed in detail. Keywords marked with * may be changed during a run, i.e. changes of the corresponding values will be taken into account upon starting the code with the restart-flag, but any changes of other parameter values will be ignored. Note that you normally don't need to make any changes in the parameterfile when you restart a run from restart-files.

5.1 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). Note that this directory needs to exist, otherwise an error message will be produced.

`SnapshotFileBase`* `snapshot`

From this string, the name of the snapshot files 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, where a group of processors writes simultaneously to a file. In this case, the filenames are also supplemented with a tailing `.n`, where `n` designates the number of the file in the group of files representing the snapshot.

`SnapFormat`* `1`

A flag that specifies the file-format to be used for writing snapshot files. A value of 1 selects the standard file-format of GADGET-2 (which is compatible with the format used in GADGET-1), while a value of 2 selects a more convenient variant of this simple binary format. A value of 3 selects the use of HDF5 instead. The structure of these files will be discussed in a separate section below. Other values for `SnapFormat` could be used to select customised file-formats if you implement appropriate modifications and/or extensions of the code.

`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 be written in parallel. The number of processors *must* be equal or larger than `NumFilesPerSnapshot`, because each snapshot file will hold the data of a group of processors. Optimum I/O throughput is reached if the number of processors is equal to, or a multiple of `NumFilesPerSnapshot`, and if `NumFilesWrittenInParallel` is equal to `NumFilesPerSnapshot` or a subdivision of it. With `NumFilesPerSnapshot=1` it is possible to write all particle data in just one snapshot file.

`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. The code will recognise the number of files that make up the initial conditions from the header entries, and load all of these files accordingly. There is no constraint for the number of these files in relation to the processor number.

`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. We recommend that you convert your IC files to one of the formats of GADGET-2. Alternatively, you could incorporate a customised reading routine directly into GADGET-2, but this requires intimate understanding of the internal workings of the code.

`RestartFile`* `restart`

This is the base filename of the restart files. For the given example parameter value, each processor will write a restart file `restart.N`, where `N` is the rank number of the processor.

`InfoFile`* `info.txt`

This is the name of a file which will hold 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 then inform you about the current status of the simulation. The meaning of the different entries in the file is discussed in detail later on.

`TimingsFile`*	`timings.txt`

This is the name of a log-file that measures various aspects of the performance of the gravitational force computation for each timestep. The meaning of the individual entries will be described later on.

`CpuFile`*	`cpu.txt`

This is a log-file that keeps track of the cumulative cpu-consumption of various parts of the code. At each timestep, one line is added to this file. The meaning of the individual entries will be described later on. A detailed analysis of them allows an assessment of the code performance, and an identification of potential bottlenecks.

`EnergyFile`*	`energy.txt`

This is the name of a file where information about global energy statistics of the simulation is stored. Checking energy conservation can sometimes be a good proxy of time integration accuracy, but it is not a particularly stringent test. Because computing the gravitational energy requires extra CPU time, it has to be explicitly enabled with the `COMPUTE_POTENTIAL_ENERGY` makefile option. Note also that an accurate computation of the peculiar gravitational binding energy at high redshift is quite subtle, so that checking for energy conservation is not a good test to evaluate the accuracy of cosmological integrations.

5.2 CPU-time limit and restart options

`TimeLimitCPU`*	`40000.0`

This is the CPU-time limit for the current run (one submission to the computing queue) in seconds. This value should be matched to the corresponding limit of the queueing system, if appropriate. The run will automatically interrupt itself and write a restart file, if 85% of this time has been consumed. The extra 15% is introduced to guarantee that there is always enough time left to safely finish the current time step and write the restart file. Note that this CPU time refers to the wall-clock time on one processor only.

`ResubmitCommand`*	`xyz.sh`

If `ResubmitOn=1`, the code will execute the given command (`xyz.sh` in this example) when it interrupts the run because the CPU-time limit is about to be exhausted. Usually, `xyz.sh` should be set to the name of an executable script in which one places the necessary commands for resubmission of the job (with the `1`-flag set for restart) to the computing queue. Note that this feature should be used with care – it is always advisable to supervise a running simulation closely.

`ResubmitOn`* `0`

This flag can be used to enable the automatic re-submission feature of the code (which is switched on for `ResubmitOn=1`).

`CpuTimeBetRestartFile`* `7200`

This is the maximum amount of CPU-time (wall-clock time, in seconds) that may be used by the code before it writes a new restart file. With this parameter the code can hence be asked to write a restart file every once in a while. This is meant to provide a precautionary measure 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. The old set of restart files is renamed into a set of `.bak` files before the new files are written, so that there is some protection against a crash during the writing of the restart files itself (quite typically, this may happen due to disk-full errors, for example).

5.3 Simulation specific parameters

`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`$= a = 1/(1 + z_{\rm start})$.

`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, since it was a special dump out of the normal sequence. For comoving integrations, the time variable is the expansion factor, e.g. `TimeMax`$= 1.0$ will stop the simulation at redshift $z = 0$.

`BoxSize` `10000.0`

The size of the periodic box (in code units) encompassing the simulation volume. This parameter is only relevant if the `PERIODIC` makefile option is used. Particle coordinates need to lie in the interval `[0, BoxSize]` in the initial conditions file.

`PeriodicBoundariesOn` `1`

Enables or disables the use of periodic boundaries. The code needs to be compiled with the `PERIODIC` makefile option for periodic boundaries, otherwise it will complain and stop. (Actually, this parameter may be viewed as redundant given that the same option is specified in the makefile, but it is kept to make sure that you run your simulation always with the 'right' executable.) When you use periodic boundary conditions for the first time, GADGET-2 will compute correction tables for the periodic forces and the periodic peculiar potential by means of Ewald summation (done in parallel). As this computation takes a bit of time, these tables will be stored on disk to speed-up the start of the code in future runs.

`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 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]`.

5.4 Cosmological parameters

`HubbleParam` `0.7`

Value of the Hubble constant in units of $100\,\mathrm{km\,s^{-1}Mpc^{-1}}$. This is only used in cosmological integrations when conversions to cgs units are required (e.g. for radiative cooling physics). Note however that the value of `HubbleParam` is not needed (in fact, it does not enter the computation at all) in purely collisionless simulations. This is because of the definition of GADGET-2's system of units, which eliminates an explicit appearance of the value of the Hubble constant in the dynamical equations.

`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 $z = 0$. Relevant only for comoving integration. For a geometrically flat universe, one has `Omega0 + OmegaLambda = 1`. For simulations in Newtonian space, 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 public version of GADGET-2, but since this paramater may be useful for the implementation of certain physical processes (as it is done in the full-physics version of GADGET-2), it has been kept in the code.

5.5 Memory allocation

`BufferSize`* `15`

This specifies the size (in MByte) of a multi-purpose communication buffer used by the code in various parts of the parallel algorithms, for example during the force computation and the domain decomposition. The buffer should be large enough to accommodate a 'fair' fraction of the particle mix in order to minimise work-load imbalance losses. In practice, sizes between a few to 100 MB offer enough room. Upon start-up, the code informs about how many particles can be fitted into the communication structures during the various parts of the code that make use of them. There is no problem if a small `BufferSize` is used, except that it can lead to somewhat lower overall performance.

`PartAllocFactor`* `1.1`

Each processor allocates space for `PartAllocFactor` times the average number of particles per processor. This number needs to be larger than 1 to allow the simulation to achieve a good work-load balance in the domain decomposition, which requires that some particle-load imbalance is accepted for better work-load balance. The specified value for `PartAllocFactor` acts as a ceiling for the maximum allowed memory imbalance, and the code will observe this limit in the domain decomposition, trying to achieve the best work-load balance within this bound. It is good to make `PartAllocFactor` as large as possible for the available memory per processor, but values in excess of 3 will usually not improve performance any more. The recommended minimum value is ~ 1.05, but practical choices should be more like 1.5-2.0. For a value of 1.0 the code will not be able to succeed in the domain decomposition and crash, unless a single CPU is used.

`TreeAllocFactor`* `0.7`

To construct the BH-tree for N particles, somewhat less than N internal tree-nodes are necessary for 'normal' particle distributions. `TreeAllocFactor` sets the number of internal tree-nodes allocated in units of the particle number. By experience, space for $\simeq 0.65\,N$ internal nodes is usually fully sufficient for typical clustered particle distributions, so a value of 0.7 should put you on the safe side. If the employed particle number per processor is very small (less than a thousand or so), or if there are many particle pairs with identical or nearly identical coordinates, a higher value may be required. Since the number of particles on a given processor may be higher by a factor `PartAllocFactor` than the average particle number, the total amount of memory requested for the BH tree on a single processor scales proportional to `PartAllocFactor*TreeAllocFactor`.

5.6 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 standard Barnes & Hut, while '1' selects a relative criterion that tries to limit the absolute truncation error of the multipole expansion for every particle-cell interaction. This scheme gives usually higher accuracy at a comparable level of computational cost, something that makes it more costly at high redshift because of the small residual forces there.

`ErrTolTheta` `0.7`

This is the accuracy criterion (the opening angle θ) of the tree algorithm if the standard Barnes & Hut (BH) opening criterion is used. If `TypeOfOpeningCriterion=1`, θ and the BH criterion are only used for the first force computation. These first forces are only used as input for a recomputation of the forces with the relative cell opening criterion, which is employed thereafter for the rest of the force computations.

`ErrTolForceAcc`* `0.005`

This controls the accuracy of the relative cell-opening criterion (if enabled), where a cell is opened if $M\,l^2 > \alpha|\mathbf{a}_{\rm old}|r^4$. Here, α is set by `ErrTolForceAcc`, M is the mass inside a node, l is its cell side-length, and $\mathbf{a}_{\rm old}$ is the magnitude of the total acceleration of the particle. The latter is usually estimated based on the gravitational force on the particle in the previous timestep. Note that independent of this relative criterion, the code will always open nodes if the point of reference lies within the geometric boundaries of the cubical cell (enlarged by an additional 10% on each side). This protects against the rare occurrence of very large force errors, which would otherwise be possible.

5.7 Time integration accuracy

`MaxSizeTimestep`* `0.01`

This parameter sets the maximum timestep a particle may take. This needs to be set to a sensible value in order to protect against too large timesteps for particles with very small acceleration. Usually, a few percent up to tens of percent of the dynamical time of the system gives sufficient accuracy. For cosmological simulations, the parameter `MaxSizeTimestep` has undergone an important **change** in its definition. Formerly, the parameter was specifying the maximum allowed step in terms of $\max(\Delta a)$, where a is the scale factor. Now, the parameter specifies $\max(\Delta \ln a)$, where $\ln$ is the natural logarithm. The new definition is equivalent to specifying the maximum allowed timestep for cosmological simulations as a fraction of the *current* Hubble time. A value of $\simeq 0.025$ is usually accurate enough for most cosmological runs.

`MinSizeTimestep`* `0`

If a particle requests a timestep smaller than the specified value, the simulation is terminated with a warning message. This can be used to prevent simulations to continue when the timestep has dropped to a very small value, because such behaviour typically indicates a problem of some sort. However, if `NOSTOP_WHEN_BELOW_MINTIMESTEP` is set in the Makefile, the timestep is forced instead to be at least `MinSizeTimestep`, even if this means that the other timestep criteria are ignored. This option should hence be used with care. It can be useful however to force equal and constant timesteps for all particles.

`TypeOfTimestepCriterion`* `0`

This selects the type of timestep criterion employed by the code for collisionless dynamics. In GADGET-1, there have been 5 possible criteria for this. However, upon systematic testing with high-resolution simulations, none of them showed a clearly superior performance compared with the 'standard' criterion used traditionally in cosmological simulations (see the paper by Powers et al., 2003, MNRAS, 338, 14). Due to this result, the choice of timestep criteria in GADGET-2 has been reduced again, and only the standard criterion is offered at present. So the only possible value for `TypeOfTimestepCriterion` is '0', which selects a timestep $\propto 1/\sqrt{a}$, or more precisely, the timestep is given by $\Delta t = \sqrt{2\eta\epsilon/|\mathbf{a}|}$, where η =`ErrTolIntAccuracy` and ϵ is the gravitational softening length.

`ErrTolIntAccuracy`* `0.025`

This dimensionless parameter controls the accuracy of the timestep criterion selected by `TypeOfTimestepCriterion`.

`TreeDomainUpdateFrequency`* `0.05`

A domain decomposition and a full tree construction are not necessarily carried out every single timestep. Instead, properties of tree nodes can be dynamically updated as needed. This can improve the performance of the code if there are many timesteps that advance only a very small fraction of particles (below a percent or so). This parameter determines how often the domain decomposition and the full tree are reconstructed from scratch. A value of `TreeUpdateFrequency`$= 0.05$, for example, means that the domain decomposition and the tree are reconstructed whenever there have been at least $0.05\,N$ force computations since the last reconstruction, where N is the total particle number. A value of zero for this parameter will reconstruct the tree every timestep.

`MaxRMSDisplacementFac`* `0.25`

This defines an additional timestep limiter for the PM-step when the TreePM method is used. In this case, the code computes the 3D velocity dispersion σ for all particles in the simulation, and restricts the maximum simulation timestep (and hence the PM timestep) such that $\sigma\Delta t$ is less than `MaxRMSDisplacementFac` times the mean particle separation $\overline{d}$, or less than `MaxRMSDisplacementFac` times the scale r_s of the short-range/long-range force split, whichever is smaller.

5.8 Output of snapshot files

`OutputListOn`* `0`

A value of `1` signals that the output times are given in the file specified by `OutputListFilename`. Otherwise output times are generated automatically in the way described below.

`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 produced. 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 350 values. Output times that are in the past relative to the current simulation time will be ignored.

`TimeOfFirstSnapshot` `0.047619048`

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

`TimeBetSnapshot`* `1.0627825`

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 $\log(a)$. The above example steps down to redshift $z = 0$ in 50 logarithmically spaced steps. Note however that if `OutputListOn=1` this parameter is ignored.

`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 given by `EnergyFile`, 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 `TimeBegin`.

`NumFilesWrittenInParallel`* `16`

This parameter determines how many files may be written (or read) simultaneously by the code when generating (reading) snapshot files, or when writing (reading) of restart files. Because each processor writes its own restart file, it is in principle possible to do this fully in parallel, with each processor writing 'his' file at the same time. However, for big runs, say with 512 processors, a very large number of simultaneous I/O requests for large files may cause severe problems for the operating system, since the number of available disk drives on the file server will likely be smaller than the number of processors. As a result, the I/O system may get congested because these requests cannot all be served efficiently at the same time. In order to avoid such a congestion and to reach an optimum overall throughput of the I/O subsystem, it is often better not to write too many files simultaneously. This can be achieved with the parameter `NumFilesWrittenInParallel`, which restricts the number of files GADGET-2 processes concurrently during restart or snapshot file reading/writing. Note that `NumFilesWrittenInParallel` must be equal or smaller than the number of processors used.

5.9 System of units

`UnitVelocity_in_cm_per_s` `1e5`

This sets the internal velocity unit in $\mathrm{cm/sec}$. The above choice is convenient – it sets the velocity unit to $\mathrm{km/sec}$. 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 and hence `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 in internal units, with one internal time unit corresponding to $9.8 \times 10^8\,\mathrm{yr}/h$. However, you are free to choose a different system of units if you like.

Note also that this implies that for purely gravitational dynamics, the code will not need to know the value of h at all. `HubbleParam` is kept in the parameterfile only because additional physics in the hydrodynamical sector (which is not contained in the public distribution of GADGET-2) may require it.

`UnitLength_in_cm` `3.085678e21`

This sets the internal length unit in cm/h, where $H_0 = 100\,h\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$. The above choice is convenient – 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 $H_0 = 100\,h\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$. The above choice is convenient – it sets the mass unit to $10^{10}\,\mathrm{M_\odot}/h$.

`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, $G = 43007.1$ in internal units. For `GravityConstantInternal=0` the code calculates the value corresponding to the physical value of G for you. Sometimes, you might want to set G yourself. For example, by specifying `GravityConstantInternal=1`, `UnitLength_in_cm=1`, `UnitMass_in_g=1`, and `UnitVelocity_in_cm_per_s=1`, one obtains a 'natural' system of units. Note that the code will nevertheless try to use the 'correct' value of the Hubble constant even in this case, so you should always set `GravityConstantInternal=0` in cosmological integrations.

5.10 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`.

`ArtBulkViscCons`* `1.0`

This sets the value of the artificial viscosity parameter α_{visc} used by GADGET-2. See code paper for details.

`CourantFac`* `0.15`

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

`InitGasTemp` `10000`

This sets the initial gas temperature in Kelvin when initial conditions are read. If the temperature is below $10^4\,\mathrm{K}$, a mean molecular weight corresponding to neutral gas of primordial abundance is assumed, otherwise complete ionisation is assumed. 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 is zero, otherwise the initial gas temperature is left at the value stored in the IC file.

`MinGasTemp` `20`

The minimum allowed gas temperature in Kelvin (this is converted by the code to a minimum thermal energy per unit mass assuming the mean molecular weight of neutral gas). This may be set to zero in principle, but often it is desirable to prevent the gas from becoming too cold, e.g. for resolution reasons or because of lower limits in the implemented cooling function.

`MinGasHsmlFractional` `0.1`

This parameter sets the minimum allowed SPH smoothing length in units of the gravitational softening length of the gas particles. The smoothing length will be prevented from falling below this value. When this bound is actually reached, the number of smoothing neighbours will instead be increased above `DesNumNgb`, even if this means that the upper limit `DesNumNgb+MaxNumNgbDeviation` is exceeded. The parameter `MinGasHsmlFractional` can be zero, allowing the smoothing radius to vary without bounds, which is usually adequate.

5.11 Gravitational softening

The code distinguishes between different particle types. Each type may have a different gravitational softening. As far as gravity is concerned, all the types are treated equivalently by the code, the names 'Gas', 'Halo', 'Disk', 'Bulge', 'Stars', and 'Bndry', are just arbitrary tags, still reflecting GADGET-2's origin as a tool to study colliding galaxies. However, the particles of the first type ('Gas') are indeed treated as SPH particles, i.e. they receive an additional hydrodynamic acceleration, and their internal entropy per unit mass is evolved as independent thermodynamic variable.

Gravity is softened with a spline, as outlined in the code paper. The softenings quoted here all refer to ϵ, the roughly equivalent Plummer softening length. Note that for the spline that is actually used, the force will be exactly Newtonian beyond $h = 2.8\epsilon$. The softening lengths are in internal length units. For comoving integration, the softening refers to the one employed in comoving coordinates, which usually stays fixed during the simulation. However, employing the `*MaxPhys` parameters described below the code can also be asked to switch to a fixed softening in physical scale below a certain redshift.

If particle types with different softenings are used, the `UNEQUALSOFTENINGS` option in the makefile needs to be set. Unlike in GADGET-1, the present code stores all particle types in a single tree, and this option tells the code to mark tree nodes that contain particles with different softenings in a special way.

In cosmological simulations, one sometimes wants to start a simulation with a softening $\epsilon_{\rm com}$ that is fixed in comoving coordinates (where the physical softening, $\epsilon_{\rm phys} = a\epsilon_{\rm com}$, then grows proportional to the scale factor a), but at a certain redshift one wants to freeze the resulting growth of the physical softening $\epsilon_{\rm phys}^{\rm max}$ at a maximum value. These maximum softening lengths are specified by the `Softening*MaxPhys` parameters. In the actual implementation, the code uses $\epsilon'_{\rm com} = \min(\epsilon_{\rm com}, \epsilon_{\rm phys}^{\rm max}/a)$ as comoving softening. Note that this feature is only enabled for `ComovingIntegrationOn=1`, otherwise the `*MaxPhys` values are ignored.

`SofteningGas` `0`

`SofteningGas` specifies the (comoving) softening of the first particle group.

`SofteningHalo` `18`

This specifies the (comoving) softening of the second particle group.

`SofteningDisk` `90`

This specifies the (comoving) softening of the third particle group.

`SofteningBulge` `450`

This specifies the (comoving) softening of the fourth particle group.

`SofteningStars` `0`

This specifies the (comoving) softening of the fifth particle group.

`SofteningBndry` `0`

This specifies the (comoving) softening of the sixth particle group.

`SofteningGasMaxPhys` `0`

This specifies the maximum physical softening of the first particle group.

`SofteningHaloMaxPhys` `3`

This specifies the maximum physical softening of the second particle group.

`SofteningDiskMaxPhys` `15`

This specifies the maximum physical softening of the third particle group.

`SofteningBulgeMaxPhys` `75`

This specifies the maximum physical softening of the fourth particle group.

`SofteningStarsMaxPhys` `0`

This specifies the maximum physical softening of the fifth particle group.

`SofteningBndryMaxPhys` `0`

This specifies the maximum physical softening of the sixth particle group.

6 File formats of GADGET-2: Snapshots & Initial conditions

The primary result of a simulation with GADGET-2 are *snapshots*, which are simply dumps of the state of the system at certain times. GADGET-2 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 few GB, say, it is much easier to have several files with a smaller size of a few hundred MB 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`. For $k = 1$, the filenames will just have the form `snapshot_XXX`. The base "`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. However, the files all have the same basic format (this is the case for all three fileformats supported by GADGET-2), and all of them are in binary. A binary representation of the particle data is our preferred choice, because it allows *much faster* I/O than ASCII files. In addition, the resulting files are *much smaller*, and the data is stored *loss-less*.

6.1 Structure of the default snapshot file format

In the default file format of GADGET-2 (selected as fileformat `1` for `SnapFormat`), 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. A list of the blocks defined in the public version of GADGET-2 is given in Table 2. The first block 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.

Within each block, the particles are ordered according to the particle type, i.e. gas particles will come first (type 0), then 'halo' particles (type 1), followed by 'disk' particles (type 2), and so on. The correspondence between type number and symbolic type name is given in Table 3. However, it is important to realize that the detailed sequence of particles within the blocks will still change from snapshot to snapshot. Also, a given particle may not always be stored in the snapshot file with the same 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 coarse of a parallel simulation. In order to trace a particle between different outputs, one therefore has to resort to the particle IDs, which can be used to label particles uniquely.

To allow an easy access of the data also in Fortran[2], 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, say just temperatures and densities, but no coordinates and velocities.

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, massarr, a, redshift, flagsfr, flagfeedb, nall
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`, 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-2. Unlike GADGET-1, GADGET-2 will check explicitly that the block-size fields contain the correct values and refuse to start if not. In the `Analysis` directory, you can also find an example for a read-in routine in IDL.

The sequence of blocks in a file is given in Table 2. Note however that not all of the blocks need to be present, depending on the type of the simulation, and the makefile options of the code. For example, if there are no gas particles in the snapshot file, the blocks for internal energy or density will be absent. 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 absent.

The detailed structure of the fields of the file header is given in Table 4. Note that the header is filled to a total length of 256 bytes with unused entries, leaving space for extensions that can be easily ignored by legacy code. Also note that a number of the flags defined in the header are

[2]This should not be misunderstood as an encouragement to use Fortran.

#	Block ID	HDF5 identifier	Block content
1	`HEAD`		Header
2	`POS`	`Coordinates`	Positions
3	`VEL`	`Velocities`	Velocities
4	`ID`	`ParticleIDs`	Particle ID's
5	`MASS`	`Masses`	Masses (only for particle types with variable masses)
6	`U`	`InternalEnergy`	Internal energy per unit mass (only SPH particles)
7	`RHO`	`Density`	Density (only SPH particles)
8	`HSML`	`SmoothingLength`	SPH smoothing length h (only SPH particles)
9	`POT`	`Potential`	Gravitational potential of particles (only when enabled in makefile)
10	`ACCE`	`Acceleration`	Acceleration of particles (only when enabled in makefile)
11	`ENDT`	`RateOfChangeOfEntropy`	Rate of change of entropic function of SPH particles (only when enabled in makefile)
12	`TSTP`	`TimeStep`	Timestep of particles (only when enabled in makefile)

Table 2: Output blocks defined in GADGET-2's file formats.

Type	Symbolic Type Name
0	Gas
1	Halo
2	Disk
3	Bulge
4	Stars
5	Bndry

Table 3: Symbolic type names and their corresponding type numbers.

not used by the public version of GADGET-2. The data type of each field in the header is either `double` (corresponding to `real*8` in Fortran), or `int` (corresponding to `int*4` in Fortran), with the exception of the particle numbers, which are `unsigned int`.

The format, units, and variable types of the individual blocks in the snapshot files are as follows:

1. **Header:**
   `struct io_header;`
   The individual fields of the file header are defined in Table 4.

2. **Particle positions:**
   `float pos[N][3];` *`real*4 pos(3,N)`*
   Comoving coordinates in internal length units (corresponds to $h^{-1}\mathrm{kpc}$ if the above choice for the system of units is adopted). `N=sum(io_header.Npart)` is the total number of particles in the file. Note: 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.

3. **Particle velocities:**
   `float vel[N][3];` *`real*4 vel(3,N)`*
   Particle velocities u in internal velocity units (corresponds to km/sec if the default choice for the system of units is adopted). Peculiar velocities v are obtained by multiplying u with $\sqrt{a}$, i.e. $v = u\sqrt{a}$. For non-cosmological integrations, $u = v$ are just ordinary velocities.

4. **Particle identifications:**
   `unsigned int id[N];` `int*4 id(N)`
   Particle number for unique identification. 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.

5. **Variable particle masses:**
   `float masses[Nm];` `real*4 masses(Nm)`
   Particle masses for those particle types that have variable particle masses (i.e. zero entries in `massarr`). `Nm` is the sum of those `npart` entries that have vanishing `massarr`. If `Nm=0`, this block is not present at all.

6. **Internal energy:**
   `float u[Ngas];` `real*4 u(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 above system of units, `u` is given in $(\mathrm{km/sec})^2$.

7. **Density:**
   `float rho[Ngas];` `real*4 rho(Ngas)`

Header Field	**Type**	**HDF5 name**	**Comment**
`Npart[6]`	`uint`	NumPart_ThisFile	The number of particles of each type in the present file.
`Massarr[6]`	`double`	MassTable	The mass of each particle type. If set to 0 for a type which is present, individual particle masses from the `mass` block are read instead.
`Time`	`double`	Time	Time of output, or expansion factor for cosmological simulations.
`Redshift`	`double`	Redshift	$z = 1/a - 1$ (only set for cosmological integrations).
`FlagSfr`	`int`	Flag_Sfr	Flag for star formation (unused in the public version of GADGET-2).
`FlagFeedback`	`int`	Flag_Feedback	Flag for feedback (unused).
`Nall[6]`	`int`	NumPart_Total	Total number of particles of each type in the simulation.
`FlagCooling`	`int`	Flag_Cooling	Flag for cooling (unused).
`NumFiles`	`int`	NumFilesPerSnapshot	Number of files in each snapshot.
`BoxSize`	`double`	BoxSize	Gives the box size if periodic boundary conditions are used.
`Omega0`	`double`	Omega0	Matter density at $z = 0$ in units of the critical density (only relevant for cosmological integrations).
`OmegaLambda`	`double`	OmegaLambda	Vacuum energy density at $z = 0$ in units of the critical density.
`HubbleParam`	`double`	HubbleParam	Gives 'h', the Hubble constant in units of $100\,\mathrm{km\,s^{-1}Mpc^{-1}}$ (for cosmological integrations).
`FlagAge`	`int`	Flag_StellarAge	Creation times of stars (unused).
`FlagMetals`	`int`	Flag_Metals	Flag for metallicity values (unused).
`NallHW[6]`	`int`	NumPart_Total_HW	For simulations that use more than 2^{32} particles, these fields hold the most significant word of 64-bit total particle numbers. Otherwise zero.
`flag_entr_ics`	`int`	Flag_Entropy_ICs	Flags that *initial conditions* contain entropy instead of thermal energy in the `u` block.
`unused`			Currently unused space which fills the header to a **total length of 256 bytes**, leaving room for future additions.

Table 4: Fields in the file header of snapshot files. In the native file formats of GADGET-2, these fields are the elements of a structure of total length of 256 bytes. In the HDF5 file format, each field is stored as an attribute to a `Header` data group.

Comoving density of SPH particles. Units are again in internal code units, i.e. for the above system of units, `rho` is given in $10^{10}\,h^{-1}\mathrm{M}_\odot/(h^{-1}\mathrm{kpc})^3$.

8. **Smoothing length:**
   `float hsml[Ngas];` `real*4 hsml(Ngas)`
   Smoothing length of the SPH particles. Given in comoving coordinates in internal length units.

9. **Gravitational potential:**
   `float pot[N];` `real*4 pot(N)`
   Gravitational potential for particles. This block will only be present if it is explicitly enabled in the makefile.

10. **Accelerations:**
    `float acc[N][3];` `real*4 acc(3,N)`
    Accelerations of particles. This block will only be present if it is explicitly enabled in the makefile.

11. **Rate of entropy production:**
    `float dAdt[Ngas];` `real*4 dAdt(Ngas)`
    The rate of change of the entropic function of each gas particles. For adiabatic simulations, the entropy can only increase due to the implemented viscosity. This block will only be present if it is explicitly enabled in the makefile.

12. **Timesteps of particles:**
    `float dt[N];` `real*4 dt(N)`
    Individual particle timesteps of particles. For cosmological simulations, the values stored here are $\Delta \ln(a)$, 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.

6.2 A variant of the default file format

Whether or not a certain block is present in GADGET-2's default fileformat depends on the type of simulation, and the makefile options. This makes it difficult to write reasonably general I/O routines for analysis software, 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 in the full physics version of GADGET-2, where already around 40 I/O blocks are defined.

To remedy this problem, a variant of the default fileformat was implemented in GADGET-2 (based on a suggestion by Klaus Dolag), which can be selected by setting `SnapFormat=2`. Its only difference is that each block is preceeded 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 contained in the second column of Table 2. 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.

6.3 The HDF file format

If the hierarchical data format is selected as format for snapshot files or initial conditions files, GADGET-2 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. A wealth of information about HDF5 can be found on the website of the project.

GADGET-2 stores the snapshot file data in several data groups, which are organised as follows:

```
/Header

/Type0/Positions
/Type0/Velocities
/Type0/IDs
...
/Type1/Positions
/Type1/Velocities
...
```

The `Header` group only contains attributes that hold the fields of the file header in the standard file format of GADGET-2. The names of these attributes are listed in Table 4. For each particle type present in the snapshot file, a separate data group is defined, named `Type0`, `Type1`, and so on. The blocks of the standard file format are then stored as datasets within these groups. The names of these data sets are listed in Table 2.

6.4 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.

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-2 is reading an initial conditions file. Instead, you have to set the time of the start of the simulation with the `TimeBegin` option in the parameterfile.

7 Diagnostic output files

7.1 Information file

In the file with name specified by `InfoFile`, you just find a list of all the timesteps. Typical output in this file looks like this:

```
Begin Step 49, Time: 0.143555, Systemstep: 0.00292969

Begin Step 50, Time: 0.146484, Systemstep: 0.00292969

Begin Step 51, Time: 0.149414, Systemstep: 0.00292969

Begin Step 52, Time: 0.152344, Systemstep: 0.00292969
```

The first number just counts and identifies all the timesteps, the values given after ‘Time’ are their current simulation times, and 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 step. For cosmological integrations, this output is supplemented with the redshift, and the systemstep in linear and logarithmic form:

```
Begin Step 636, Time: 0.487717, Redshift: 1.05037, Systemstep: 0.000724494, Dloga: 0.00148658

Begin Step 637, Time: 0.488443, Redshift: 1.04732, Systemstep: 0.000725571, Dloga: 0.00148658

Begin Step 638, Time: 0.489169, Redshift: 1.04428, Systemstep: 0.000726651, Dloga: 0.00148658

Begin Step 639, Time: 0.489897, Redshift: 1.04125, Systemstep: 0.000727732, Dloga: 0.00148658

Begin Step 640, Time: 0.490626, Redshift: 1.03821, Systemstep: 0.000728814, Dloga: 0.00148658
```

7.2 Performance statistics of the gravitational tree

In the file specified by `TimingsFile`, you get some statistics about the performance of the gravitational force computation by the tree algorithm, which is usually (but not always) the main sink of computational time in a simulation. A typical output may look like this:

```
Step= 482  t= 1.29199  dt= 0.00292969
Nf= 10992  total-Nf= 0016981370
work-load balance: 1.10997  max=0.618782 avg=0.557475 PE0=0.618782
particle-load balance: 1.02133
max. nodes: 15202, filled: 0.422278
part/sec=9858.74 | 8881.96  ia/part=1004.71 (0)

Step= 483  t= 1.29492  dt= 0.00292969
Nf= 60000  total-Nf= 0017041370
work-load balance: 1.01677  max=2.51368 avg=2.47221 PE0=2.43074
particle-load balance: 1.02133
max. nodes: 15202, filled: 0.422278
part/sec=12134.9 | 11934.7  ia/part=805.744 (0)
```

The first line of the block generated for each step informs you 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). `Nf` gives the number of particles that receive a force computation in the current timestep, while `total-Nf` gives the total number of force computations since the simulation was started. `work-load balance` gives the work-load balance in the actual tree walk, i.e. the largest required time of all the processors divided by the average time. The number given by `max` is the maximum time (in seconds) among the processors spent for the tree walk, while `avg` gives the average, and `PE0` the time for processor 0. `particle-load balance` gives the maximum number of particles per processor divided by the average number, i.e. it measures the memory imbalance. An upper bound for this number is `PartAllocFactor`. However, for technical reasons, the domain decomposition always leaves room for a few particles, hence this upper bound is never exactly reached. `max.nodes` informs about the maximum number of internal tree nodes that are used among the group of processors, and the number behind `filled` gives this quantity normalised to the total number of allocated tree nodes, it hence gives the fraction to which the tree storage is filled. Finally, `part/sec` measures the raw force speed in terms of tree-force computations per processor per second. The first number gives the speed that would be achieved for perfect work-load balance. Due to work-load imbalance, the actually achieved average effective force speed will always be lower in practice, and this number is given after the vertical line. `ia/part` gives the average number of particle-node interactions required to compute the force for each of the active particles. The number in parenthesis is only non-zero if the Ewald correction is used to obtain periodic boundaries. It then gives the average number of Ewald correction-forces evaluated per particle.

7.3 CPU-time statistics

In the file specified by `CpuFile`, you get some statistics about the total CPU consumption measured in various parts of the code while it is running. A typical output looks like this:

```
Step 13121, Time: 0.168501, CPUs: 2
  44000.01   32984.59    6496.83      52.40    1318.17    2744.38 ...
Step 13122, Time: 0.168504, CPUs: 2
  44001.11   32985.30    6497.01      52.40    1318.17    2744.59 ...
```

For each timestep, there are two lines. The first line gives the current time step, corresponding simulation time, and number of CPUs used. In the second line, the first number informs you about the total cumulative CPU consumption of the code up to this point (in seconds), while the other numbers measure the time spent in various parts of the code. The numbers give measurements for the following parts (in that sequence):

1. Total consumption.
2. Gravitational force computation (including tree walks, construction, communication).
3. Hydrodynamics (including neighbour search, density determination, and SPH communication).

4. Domain decomposition.
5. Potential energy computation (for the energy statistics).
6. Drifting the particle set.
7. Timestep determination and ‘kicking’ of particles.
8. Writing of snapshot files.
9. Pure time for tree walks (this is a sub-part of 1).
10. Tree construction (this is a sub-part of 1).
11. Communication and summation (this tries to measure the time spent in communication and summation of force components in the gravitational tree part).
12. Imbalance of gravitational tree (this measures work-load imbalance losses directly).
13. SPH computations (finding of neighbours and evaluating SPH-sums with them).
14. Communication and summation in the SPH routines.
15. Losses due to work-load imbalance in the SPH part.
16. Neighbour adjustment (measures the time needed to readjust SPH smoothing lengths if the number of neighbours obtained for the next guess smoothing length falls outside the desired range).
17. Time for PM computation.
18. Time needed to establish Peano-Hilbert order.
19. Time spent in cooling and star formation routines. (Not present in the public version of GADGET-2.)

All these times are wallclock times, provided `WALLCLOCK` is set in the makefile, which is the recommended setting. Otherwise, they correspond to CPU cycles (expressed in seconds) consumed on average per processor. The latter way of measuring things is more meaningful if GADGET-2 has not exclusive access to all of the CPUs (i.e. other processes consume a non-negligible fraction of the available CPU processes), but instead needs to share them with other processes to a significant degree.

7.4 Energy statistics

In the file specified by `EnergyFile`, 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 `EnergyFile`.

Each of these lines contains 28 numbers, 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 `Gas` to `Bndry`. Finally, the last six numbers give the total mass in these components.

While the meaning of kinetic and potential energy in non-cosmological simulations is straightforward (and also the test for conservation of total energy), this is more problematic in cosmological integrations.

For cosmological integrations, the kinetic energy you obtain in this file will be

$$T = \frac{1}{2}\sum_i m_i \mathbf{w}^2, \tag{1}$$

where $\mathbf{w}$ is the peculiar velocity divided by $\sqrt{a}$ (see paper). The potential energy will be computed as

$$U = \frac{1}{2}\sum_i m_i \Phi(\mathbf{x}_i), \tag{2}$$

where $\Phi(\mathbf{x}_i)$ is the peculiar potential when periodic boundaries are used.

If vacuum boundaries are used, and no cosmological integration is done, $\Phi(\mathbf{x}_i)$ is simply the Newtonian potential. However, for comoving integration with vacuum boundaries the potential is computed as

$$\Phi(\mathbf{x}_i) = -G\sum_k m_k\, g\left(|\mathbf{x}_k - \mathbf{x}_i|\right) - \frac{1}{2}\Omega_0 H_0^2 \mathbf{x}_i^2 \tag{3}$$

Frequent outputs of the energy quantities allow a check of energy conservation. While this is straightforward for Newtonian dynamics, it 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, to the point that it is nearly useless for testing code accuracy.

Also note that for vacuum boundary conditions in cosmological integrations, the zero-point of the potential will not be the one of the ordinary peculiar potential. Therefore, the potential energy will be orders of magnitude larger than the kinetic energy, and the kinetic energy will be dominated by the outer shell of the simulation volume in this case.

8 Differences between GADGET-1 and GADGET-2

8.1 Architectural changes

There have been many major changes in the internal workings between GADGET-1 and GADGET-2, affecting nearly all parts of the code. In fact, basically all parts of the code have been completely rewritten, or were revised substantially. 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 types of simulations. Detailed explanations of the reasoning behind each of these modifications is beyond the scope of this short set of notes. This information can be found in the code paper on GADGET-2. However, below, I give a very brief summary of some of the most important changes in code design.

1. The internal organisation of the time integration has been changed substantially. The timestep of a particle is now usually restricted (again) to be a power of 2 subdivision of the total time span that is simulated. (It is however also possible to run the code with a more flexible timestep hierarchy.) The integration proceeds then as an alternation between 'drift' and 'kick' operations, with the leapfrog being a kick-drift-kick scheme now, instead of a drift-kick-drift. There is no explicit prediction step any more, since this part has been absorbed into the drift operation.

2. The quadrupole moments in the gravitational tree have been dropped, i.e. the new tree has only monopole moments. To achieve the same force accuracy, this generally means that more cell-particle interactions have to be evaluated. However, because each interaction is relatively simple, this effect is largely offset by the increased performance of the tree-walk in terms of interactions per second. This has become possible because the monopole interactions are 'simpler' than the quadrupole ones, and in addition, the cache utilisation in the tree walk has been improved. The net result is that the new gravitational code is faster than the old one at the same accuracy. The ultimate reason for dropping the quadrupole moments has however been that this simplifies the dynamic update ('drift') of the tree. This is now done fully consistently, i.e. the tree accuracy does not degrade when the tree has not been updated for a while. (In the old code, the change of the quadrupole moments had been neglected. Also, in the Ewald correction their contribution was neglected.)

3. For cosmological integrations, the integration variables have been changed: The velocity variable is now taken to be $u = a^2\dot{x}$, and not $w = \sqrt{a}\dot{x}$ as before. This was adopted because then the equation of motion for u does not depend on u any more, and only then can the leapfrog be formulated in a symplectic way, which allows the Hamiltonian structure of the system to be preserved. (Note however that the velocities in the snapshot and initial conditions files are still in terms of w in order to have backwards compatability with GADGET-1.)

4. SPH is implemented using the conservative 'entropy formulation' by default.

5. The code may now be run optionally as a Tree-PM hybrid code, where the long-range gravitational force is computed with discrete Fourier transforms. A parallel FFT is used to this end. Both periodic and non-periodic boundary conditions are supported. Furthermore, an extension of the PM algorithm is available that is designed for "zoom" simulations, where a small volume is sampled with very many high-resolution particles. Here, the code can be instructed to place a second PM mesh on the high-res region, such that the full force for the high-res particles is composed of three components, a long-range PM force, an intermediate-range PM force, and a short range tree force. The time integration between short-range and long-range forces is decoupled such that the PM timestep may be larger than the short-range step. Nevertheless the integration scheme in principle still preserves its symplectic character (in an exact fashion only if the short-range steps do not change and are all equal, and if force errors are negligible).

8.2 Notes on migrating from GADGET-1 to GADGET-2

If you have used the earlier version GADGET-1 before, you will be familiar with all the practicalities of running GADGET-2 already. This is because nothing has really changed here. In particular, the default initial conditions file format has not changed (apart from some extensions), and the mechanisms for starting, interrupting, and resuming a simulation are still the same.

However, as described above, there have been a number of changes in the parameterfile that controls each simulation, and there are more compilation options in the makefile of the code. If you want to reuse an old parameterfile with GADGET-2, 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 (see also the information below). Note in particular that the *meaning of the parameter* `MaxSizeTimestep` *has changed*.

As a result of the different time integration scheme used in the new version of the code, considerably smaller steps are done by the code at high redshift than it used to be the case. This is necessary because even in the fully linear regime, the timestep may not exceed a certain small fraction of the Hubble time in the new time integration scheme, otherwise the integration of the linear evolution would become compromised. The old code was able to avoid this problem by a trick – its integration step essentially mimicked the Zeldovich approximation, allowing it to traverse linear evolution with pretty large steps. However, this integration scheme was not symplectic, while the new one is. The advantages brought about by the new integration scheme far outweigh the additional cost of having to do a few more steps at high redshift.

Note that one should also carefully review which makefile options are appropriate for a given simulation. For many types of simulations, there are now different possibilities for running a simulation, e.g. with or without TreePM algorithm, with single or double-precision, etc.

8.3 Dropped makefile options

In case you wonder, GADGET-1 contained a number of makefile options that are not present any more. The following table lists them, and briefly explains the reason why they were dropped.

`-DVELDISP`	The timestep criterion based on an estimate of the local velocity dispersion has been dropped because, disappointingly, it did not do significantly better than simpler timestep criteria based on the velocity alone.
`-DBMAX`	Something equivalent to `BMAX` is always enforced in the new tree walk.
`-DRECTREECONS`	The monopole moments of the tree are always constructed recursively in the new code.
`-DINLINE`	The routines that could have benefitted from inlining have been inlined manually, or have been replaced by macros.
`-DDIAG`	Timing diagnostics are now always collected by the code.

8.4 List of source files of GADGET-2

In Table 5, you can find an updated list of the source files of GADGET-2. The source code is commented and reasonably readable I hope. As an additional help to browse the sources, they can be accessed via a cross-references hypertext documentation that can be found in the `html` subdirectory (open the file `index.html` in your favourite browser). This documentation has been generated directly from the annotated source code with the `doygen` tool. If you change the sources and want to regenerate the sources, you can do so by executing the command `doygen` in the source directory of GADGET-2. This will only work if `doygen` has been installed on your system, of course.

8.5 Notes on memory consumption of GADGET-2

Let $N_{\rm tot}$ be the total number of particles, and $N_{\rm p}$ the number of processors. Then the *average* particle load per processor is $N = N_{\rm tot}/N_{\rm p}$. The code will allocate about

$$M_1 = \texttt{PartAllocFactor} \times N \times (68 + \texttt{TreeAllocFactor} \times 64) \tag{4}$$

bytes for particle storage, for the gravity/neighbour tree, and for the time integration scheme, on each processor. For typical values like
$\texttt{TreeAllocFactor} \simeq 0.65 - 0.7$, this implies a memory cost of $\sim$110 bytes per particle. For SPH particles, additional memory of

$$M_2 = \texttt{PartAllocFactor} \times N_{\rm sph} \times 84 \tag{5}$$

bytes will be required. For an equal mixture of SPH and collisionless particles, the average memory consumption per particle is therefore going to be about 152 byte/particle.

accel.c	Driver routine to carry out force computation
allocate.c	Routines for allocating particle and tree storage
allvars.c	Provides instances of all global variables
begrun.c	Initial set-up of a simulation run
density.c	SPH density computation and smoothing length determination
domain.c	Code for domain decomposition
driftfac.c	Computes look-up tables for prefactors in cosmological integrations
endrun.c	Terminates run upon severe errors
forcetree.c	Gravitational tree code
global.c	Global energy computation
gravtree.c	Main driver routines for gravitational (short-range) force computation
gravtree_forcetest.c	Routines for direct summation forces
hydra.c	Computation of SPH forces and rate of entropy generation
init.c	Code for initialisation of a simulation from initial conditions
io.c	Routines for producing a snapshot file on disk
longrange.c	Driver routines for computation of long-range gravitational PM force
main.c	Start of the program
ngb.c	Neighbour search by means of the tree
peano.c	Routines to compute a Peano-Hilbert order
pm_nonperiodic.c	Code for non-periodic FFT to compute long-range PM force
pm_periodic.c	Routines for periodic PM-force computation
potential.c	Computation of the gravitational potential of particles
predict.c	Drift particles by a small time interval
read_ic.c	Read initial conditions in one of the supported file formats
restart.c	Code for reading and writing restart files
run.c	Iterates over timesteps, main loop
system.c	Miscellaneous routines, e.g. elapsed time measurements
timestep.c	Routines for ‘kicking’ particles and assigning new timesteps
allvars.h	Declares global variables
proto.h	This file contains all function prototypes of the code
tags.h	Declares various tags for labelling MPI messages

Table 5: List of source files of GADGET-2.

Note however that it is not advisable to fill all the available memory of the machine. For the sake of performance one should leave room for memory-imbalance, i.e. `PartAllocFactor` needs to be chosen larger than 1.0. In practice, values below 1.2 can lead to serious work-imbalance, depending on the amount of clustering and on how many processors are involved.

Certain code options will also increase the amount of memory allocated per particle. Obviously, selecting `DOUBLEPRECISION` can increase the storage requirement substantially, almost doubling it. Also, simulations with cooling and star formation require slightly more memory to allow the storage of additional quantities like ionisation fraction, metallicity, etc.

In practice, it is thus best to pay attention to the screen output that is produced in the log-file when GADGET-2 starts. The code will report the sizes of all major chunks of memory that are allocated. Note that these numbers refer to the allocation done by each individual processor. To obtain the total amount of memory needed by the code, one needs to multiply the sum of these numbers by the number of processors that are used.

The communication buffers requires

$$M_3 = \texttt{BufferSize} \times 1024^2 \tag{6}$$

bytes, but this number is independent of the particle load.

Note that the TreePM algorithm can require substantial amounts of additional memory, depending on the choice of `PMGRID`, and the type of TreePM simulation done. In the simplest case, a periodic box, the allocated amount of memory, summed over all processors, is about

$$M_4 = \texttt{PMGRID}^3 \times 12 - 16 \tag{7}$$

bytes, provided single precision is used. For double precision, twice this amount is needed. If the mesh is made a factor of 2 finer than the mean particle spacing in each dimension, then the required storage for the PM algorithm roughly matches that needed for the particle data and the tree, i.e. then the memory requirement of the code essentially doubles.

If the TreePM algorithm with non-periodic boundaries is selected instead, the memory requirements are substantially larger. They are then roughly

$$M_5 = (2 * \texttt{PMGRID})^3 \times 16 \tag{8}$$

bytes. If a two level zoom simulation in a periodic volume is selected, then M_4 and M_5 are both allocated. If the zoom is done in a simulation with vacuum boundaries, then only M_5 is allocated, but the number 16 is replaced by 20.

9 Examples

To get some idea of the usage of the code, you may try a few of the simple examples I provide with the code distribution (these are the same ones as in GADGET-1). The parameterfiles for these examples can be found in the `parameterfiles`-subdirectory of the code-directory. The first thing you have to do before you can run any of the examples is to edit the corresponding parameterfile and adjust the path of the output directory, and the filename of the initial conditions file. Initial conditions (and snapshot files) of GADGET-2 are in binary format, so its important to know the byte order of your system. Intel, AMD, and some Alpha machines (not the T3E, though) use little endian as opposed to big endian, which is the standard on most other RISC machines (IBM, Sun). For each IC-File, I have prepared a corresponding file for little endian machines, designated by an extension '`.intel`'. These files are in the '`ICs`' directory. (Note that you can reverse byte-order easily by writing a little C-routine yourself.) The different examples are explained further below.

9.1 Galaxy collision

This purely collisionless simulation 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.

To allow you to get a first idea whether the examples have worked, I include a few IDL scripts that read snapshot files and diagnostics output. These files are in the directory '`Analysis`'. You can use '`energy_galaxy.pro`' to plot the evolution of kinetic and potential energy, and the variation of the total energy in the galaxy example. You may use the script '`show_galaxy.pro`' to load the snapshot files of the galaxy collision one after the other and display the disk particles of the galaxies.

9.2 Adiabatic collapse of a gas sphere

This simulation considers the gravitational collapse of a self-gravitating sphere of gas which initially has a $1/r$ density profile at a low temperature. The gas falls under its own weight to the centre, where a bounce-back with a strong shock wave 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 ($G = 1$). You can use '`energy_gassphere.pro`' to display the evolution of thermal, kinetic and potential energy for the collapsing gassphere. Note that this is a very tiny simulation of just 1472 particles.

9.3 Cosmological formation of a cluster of galaxies

This problem uses collisionless dynamics in an expanding universe. It is a small cluster simulation that has been set-up 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. The IDL-script '`show_cluster.pro`' is an example for reading one of the snapshots of the cluster simulation and showing the high resolution region.

9.4 Large-scale structure formation including gas

This problem consists of 32^3 dark matter, and 32^3 gas particles, following structure formation in a periodic box of size $(50\,h^{-1}\mathrm{Mpc})^3$ in a ΛCDM 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.

You can use '`show_gas.pro`' to read a snapshot file of this simulation, to extract the gas particles from it, and to plot their spatial distribution. You may use '`plot_hotgas.pro`' to show the gas with temperature higher than 10^6 K at the present time.

10 Disclaimer

It is important to note that the performance and accuracy of the code is a sensitive function of some of the code parameters. Let me also stress that GADGET-2 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 author! In particular, I do not endorse these parameter settings in any way, nor do I claim that they will provide fully converged results for the example problems, or for more general initial conditions. I think that it is extremely difficult to make general statements about how convergence can be obtained, especially when one desires to achieve it with the smallest possible computational effort. For this reason I refrain from making such recommendations. I encourage every simulator to find out for herself/himself what integration settings are needed to achieve sufficient accuracy for the system under study. I strongly recommend to make convergence and resolution studies to establish the range of validity and the uncertainty of any numerical result obtained with GADGET-2.*

Volker Springel
Munich, April 2005