

GRACOS Users Reference

For version 1.0.1a8, 8 November 2007

Alexander Shirokov

Copyright © 2002,2003,2004,2005,2006,2007 by Alexander Shirokov (CITA/MIT) and Edmund Bertschinger (MIT).

This material is distributed under the terms of the current GNU general public license, version 2.

This package distribution includes other software whose copyrights are listed below

- Hilbert Curve implementation by Doug Moore (packaged as ‘`hilbert`’) are Copyright © 1998, Rice University.
- GIF image maker, packaged as ‘`wgifs`’.

This package was originally in the form of `cmap.h`, `wgif.c` and `compress.c` files produced by Spencer W. Thomas, Jim McKie, Steve Davies, Ken Turkowski James A. Woods, and Joe Oros.

- M4 Macros:

Macro ‘`gse_get_version.m4`’ and script executable ‘`get_gse_version`’ for generating package version number are the direct derivatives of ‘`lam_get_version.m4`’ and ‘`get_lam_version`’ of the *lam-7.1.2* package.

Macro ‘`acx_mpi.m4`’ for detection of MPI installation is written by Steven G. Johnson, copied from the *fftw-2.1.5* package.

- The fitting formulae for CDM + Baryon + Massive Neutrino (MDM) cosmologies are by Daniel J. Eisenstein & Wayne Hu, of the Institute for Advanced Study.

Table of Contents

1	Introduction	1
2	gracos Executable Reference	3
2.1	Running <code>gracos</code>	3
2.2	<code>gracos-examples</code>	5
2.3	Interface Reference	6
2.3.1	Command Line Options	6
2.3.2	Commands	6
2.3.3	Variables	7
2.3.3.1	Example: Manipulating <code>gracos</code> Variables	8
2.3.4	Bitwise Variables	9
2.3.4.1	Example: Manipulating Bitwise Variables	10
2.3.5	Runtime Administration	11
2.3.5.1	One Time Administration	11
2.3.5.2	Repeatable Administration	12
2.3.5.3	Scheduled Administration	13
2.3.5.4	List of Runtime Administration Checkpoints	14
2.3.6	Primary Output Files	14
2.3.7	Error Behavior	15
2.4	File Input and Output with <code>dataman</code>	16
2.4.1	Physical Units	18
2.4.2	<code>binpart</code> : The Simplest Format	18
2.4.3	<code>udf</code> : Unified Datafile Format	19
2.4.4	<code>tpm</code> : Tree-Particle-Mesh format	20
2.4.5	<code>p3mdat</code> : Fortran p3m Datafile Format	20
2.4.6	<code>mwhite</code> : Martin White format	21
2.4.7	Example: Datafile Manipulations	21
2.4.8	Example: Santa Barbara Cluster Project	24
2.5	<code>image</code> : Particle Data Imaging	28
2.5.1	Example: Particle Data Imaging	29
2.6	Initial Condition Generators	33
2.6.1	<code>grafic</code> : the Initial Conditions Generator	33
2.6.1.1	Example: Tabulated Transfer Function	37
2.6.1.2	Example: <code>lingers</code> Initial conditions	44
2.6.1.3	Example: BBKS Initial Conditions	46
2.6.1.4	Example: Scale-Free Initial Conditions	47
2.6.1.5	Example: Glass Extension	48
2.6.2	<code>poisson</code> : Poisson Distribution	48
2.6.2.1	Example: Poisson Distribution	49
2.6.2.2	Example: generating the glass Distribution	50
2.6.2.3	Example: Making the Glass	51
2.7	<code>mpower</code> : Power Spectrum Estimation	54
2.8	<code>integ</code> : Running an N-body simulation	54

2.8.1	Checkpoint Control Variables	56
2.8.2	checkpoint Command	56
2.8.3	Automatic Backup Option	57
2.9	halo_finder: Halo Finder	57
2.10	compare_particles_serial: Particle Data Comparison	57
2.10.1	Comparing gracos with Fortran p3m and grafic1	58
2.11	Tables of Reference	60
2.11.1	gracos Commands	60
2.11.2	gracos Variables	65
3	Additional Programs and Libraries	73
3.1	gracos-pkgs: Required Package Installer	73
3.2	gracos-clean: Directory Cleanup Utility	74
3.3	gracos-config: Utility for Linking Programs with GRACOS ..	75
3.4	GRACOS Library	76
3.5	gracos-rw: Cosmographic Quantities for FLRW Cosmology ...	77
3.6	gracos-ehu: Eisenstein-Hu Transfer Function	78
3.7	gracos-hist: Utility for Plotting Histograms	79
3.8	Serial Fortran N-body Codes	79
3.9	bits	80
3.10	System Tests	81
3.10.1	ibctest: System Call Portability Test	81
3.10.2	mentest: Hardware Memory Test	82
4	Installation Procedure	83
4.1	System Requirements	83
4.2	Required and Recommended Packages	83
4.3	GRACOS Package Installation	83
4.3.1	Configuration	83
4.3.2	Compilation and Installation	84
4.3.3	Environment Setup	85
4.3.4	Uninstallation	85
4.4	Source code Directory Cleanup	85
5	Authors and Copyrights	87
6	Acknowledgements	89

1 Introduction

This reference documents version 1.0.1a8 of *GRACOS*, the *GRA*vitational *COS*mology suite, or a collection of software for cosmological N-body type simulations and data analysis. The primary component of *GRACOS* is the *gracos code*, – a parallel cosmological N-body code.

The *gracos* executable uses a set of optimization techniques developed over many years and first then completely documented on March 2004 in the *MIT Ph.D. Thesis* (<http://library.mit.edu>) by *Alexander Shirokov*, and astro-ph/0505087 (<http://arxiv.org/abs/astro-ph/0505087>) (in the reduced form). The most up-to-date version of *gracos* can be found at <http://www.gracos.org>.

Cosmological N-body type work generally employs highly computationally challenging and versatile applications. However it is not just the solving the computational algorithm that is challenging, but also bringing it in a usable form to *you*, the end user. It is generally considered very complicated to run an N-body simulation, especially for large problems, because many, if not all, of the current N-body code distributions come without many commonly used options, such as ability to generate the initial conditions given a set of cosmological parameters, especially when heavy amounts of data are involved; file input-output format is still an outstanding issue. This situation gets more complicated after one includes common data analysis tasks, such as the group finder, imaging or power spectrum estimation.

gracos comes with a built-in command-line interface that makes it easy to manipulate with the parallel application, raising the level of the communication between the researcher and the implementation, and thus substantially simplifying all the engineering tasks associated with the scientific problems and helping document the simulations. New commands can be introduced easily (once you know how to do it). *GRACOS* distribution provides a rather extensive the *gracos-examples* directory containing the scripts, that can be readily used upon the completion of the installation procedure. These scripts are to be used for testing, as templates for large N-body simulations and for doing some common parallel cosmological simulation tasks. Command line interface extends *gracos* development horizons to a new level.

This reference is mostly error-free, because a substantial fraction of its content is automatically generated in a one time procedure that runs *GRACOS* to generate most plots, images and tables presented throughout this reference (*doc/src/README* shows how to run the procedure). In particular, the procedure executes many scripts later described, following the guidelines in this reference.

In addition to the main executable, *GRACOS* installation provides the C-libraries and the associated C-header files that are used for *gracos* but are also suitable for more generic applications.

2 `gracos` Executable Reference

This chapter documents `gracos` executable, - our main product.

N-body simulations tend to be massive; however before any computationally large problem is submitted to a cluster it is more than useful to test the same problem for a reduced problem size. The script examples presented throughout this reference are just those small test cases. In order to extend them to larger problems just increase the number of processes, the gridsize and make sure no part of the script is intrinsically unscalable. This chapter assumes the successful completion of the installation procedure, See Chapter 4 [Installation Procedure], page 83.

2.1 Running `gracos`

`gracos` can be run both in parallel and serially; typing `gracos --help` or `gracos -?` yields a useful help message.

Usage: `gracos [OPTION...]`

A parallel program for N-body simulations and data analysis.

<code>-c, -e, --command=string</code>	Read and execute commands from the string
<code>-f</code>	Forward the standard output of the head process to an <code>so-0</code> file
<code>-i, --input=FILE</code>	Read and execute commands from FILE
<code>-t, --term</code>	Read and execute commands from standard input
<code>-v, --verb=integer</code>	Set the verbosity level
<code>-?, --help</code>	Give this help list
<code>--usage</code>	Give a short usage message
<code>-V, --version</code>	Print program version

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

Report bugs to Alexander Shirokov.

The user interacts with `gracos` by passing the commands within the `gracos` scripting session. The commands can be entered from standard input (option `-t` or the default), a file (option `-i`), or passed in the command line (option `-e`). The commands are executed using the *Tcl* embedded interpreter, See Section 2.3.2 [Commands], page 6 for more detail.

To enter the commands through the standard input, simply type `gracos` or, for a parallel run on *Nproc* logical processes type

```
SHELL$ mpirun -np Nproc gracos
```

You may use any integer number *Nproc* of processes greater than zero on any machine, however it is best to accommodate the agreement between the number *Nproc* of logical processes and the number of physical processes and cores available on the system.

The `gracos%` token that will appear prompts the user to enter a new command from the standard input in the newly opened `gracos scripting session`. As can be seen from the following example by a user familiar with *Tcl*, indeed `gracos` uses the facilities provided by the *Tcl* computing language:

```

SHELL$ mpirun -np 4 gracos
/-----\
|          GRACOS, VERSION: 1.0.1a8          |
\-----/
gracos% puts "Hello"
Hello
gracos% foreach i { 1 2 3 4 } {
.....   puts $i
..... }
1
2
3
4
gracos%

```

The input is distinguished from the output by the presence or absence of the prompts (“gracos%” and “.....” - for incomplete commands). In the above example, `$i` refers to a *Tcl* variable, and `foreach` is a *Tcl* built-in command. As soon as the complete command is entered within the scripting session, its copies are sent to the remaining MPI processes and the command is processed with the embedded *Tcl* interpreter individually on each process, See Section 2.3.2 [Commands], page 6 for more information.

In order to master `gracos`, the user should learn some basics of *Tcl*. Fortunately, *Tcl* is usually found easy to learn. A number of resources are available online; we refer the users to the *Tcl* reference (http://aspn.activestate.com/ASPN/docs/ActiveTcl/8.4/tcl/tcl_contents.htm), the *Tcl* manual (<http://www.tcl.tk/man/tcl8.4>) and the *Tcl* tutorial (<http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html>) for version 8.5. In `gracos` we avoid the use of any advanced *Tcl* feature, when possible. For those readers whose access to the above online resources is restricted, let us just say that the *Tcl* built-in commands `set` and `puts` are used to manage the *Tcl* variables (setting a variable and displaying its value); the square brackets '[' and ']' are used for an in-line expansion of a *Tcl* expression; the curly brackets '{' and '}' as well and the double quotation marks are used for quoting in slightly different contexts (mainly: the expansion of subexpressions is allowed in double quoted expressions but is not allowed in the curly bracket quotations); the `expr` *Tcl* built-in command is used for arithmetic expression evaluation, the `exec` is for external shell expression evaluation, and the `eval` is for the evaluation of the arguments passed to the command in the current scripting session. Finally `cd` can be used to change the current working directory path.

In addition to the more generic commands intrinsic to the *Tcl*, `gracos` has a number of commands that are only specific to the `gracos` executable; their implementation forms the trunk of the `gracos` C-code.

As soon as `gracos` executable is ran, a number of useful files appear in the working directory. File `gracos_history` contains the current listing of all the executed commands; files `so-1` ... `so-(Nproc-1)` show the standard output on non-server processes. The standard output on process zero appears on the screen, it is not redirected unless the `--f` command line option is used.

To execute the same commands as in the example above by reading them from a file, instead of standard input, we create a temporary file with the `mktemp` linux command, fill it with the (identical) commands to be executed and pass the filename with the `-i` option to `gracos`

```

SHELL$ TmpFile='mktemp tmp.XXXXXX'
SHELL$ cat > $TmpFile
puts "Hello"
foreach i { 1 2 3 4 } {
    puts $i
}
SHELL$ mpirun -np 4 gracos -i $TmpFile
gracos% puts "Hello"
Hello
gracos% foreach i { 1 2 3 4 } {
    puts $i
}

1
2
3
4
SHELL$

```

Now, instead of having to type the commands in through the shell session, we can place them into a new file, make the file executable and run. Shown below is the copy of the working script, also provided at `gracos-examples/hello.sh`

```

#!/bin/bash
# File: gracos-examples/hello.sh
# Author: Alexander V. Shirokov

# GRACOS Session
TmpFile='mktemp tmp.XXXXXX'
cat <<EOF > $TmpFile
puts "Hello"
foreach i { 1 2 3 4 } {
    puts \$i
}
EOF

mpirun -np 4 gracos -i $TmpFile
rm $TmpFile

```

Typing

```

chmod +x ./hello.sh
./hello.sh

```

yields the expected result. The important detail to notice in the last example above is using the backslash character (`\`) necessary to avoid the conflict between the interpretations of the `'$'` character, which is used for variable dereferencing by both *Tcl* and *bash*. If you do not use the backslash character in the example above the variable `$i` is interpreted as a **bash** shell variable and, according to the *bash* syntax, be expanded to empty value because no such **bash** variable is defined in the script. Because of the use of backslash character, this variable is interpreted as a *Tcl* variable which is set in the preceding `foreach` *Tcl* statement. One just has to be careful using backslashes to avoid the similar misinterpretations.

2.2 `gracos-examples`

Each case considered in this reference is presented as a `gracos` script ready to be executed in your Unix-flavored shell, any time following the *GRACOS* installation procedure. These scripts are organized within the so called `gracos-examples` directory whose archive is one of the products of *GRACOS* installation. In order to make a local copy of the directory, simply type:

```
SHELL$ cp 'gracos-config info pkgdatadir'/gracos-examples.tar.gz .
SHELL$ tar xzf gracos-examples.tar.gz
```

in your Unix flavored shell. The procedure creates the `gracos-examples` directory which contains all the example scripts presented in this reference. For example, the `hello.sh` script presented in Section 2.1 [Running `gracos`], page 3 is included as `gracos-examples/hello.sh`; you may execute it any time following the installation procedure, See Chapter 4 [Installation Procedure], page 83.

The user may become confused by the large number of files automatically generated as the examples are ran; type `gracos-examples -r` to clean the `gracos-examples` directory and return to the initial pristine state.

2.3 Interface Reference

The use of sophisticated interface extends `gracos` capabilities and makes it easier to document the simulations. On the other hand it poses a problem because, the practical users will be reluctant to learn the details of interface. We suggest that such readers should skim this section and go back as they feel needed, or guided by the later references.

2.3.1 Command Line Options

The full list of command line options is provided in Section 2.1 [Running `gracos`], page 3.

If you use a batch system for job submission we recommend using option `-f`. Job submission systems spool extra information into their standard output file, making it difficult to extract the standard output of process zero from the standard output of the job, the latter produced only at the end of the run also. Do not use this option for interactive sessions as you will not be able to see the prompt.

Regarding the option `-t`, note that one of the previously released versions of LAM MPI are known to show an erroneous behavior on standard input: the interactive session would terminate as soon as it is started without prompting the user. Even though the bug was corrected in the subsequent versions of LAM, it may be a good idea to always use a non-interactive session where possible (copy your input to a file and pass the using the `-i` option to `gracos`).

Option `-v` controls the verbosity associated with the `tverb` internal `gracos` bitwise variable. Use this option to set “on” the memory allocation verbosity when debugging memory leaks. Using `-v-1` sets verbosity to the maximum (all bits are “on” for `-1` in its bitwise representation).

2.3.2 Commands

`gracos` interacts with the user via its own extendable command line-based language

The complete list of `gracos` commands is given in Chapter 2 [`gracos` Executable Reference], page 3.

`gracos` uses the *Tcl* (*Tool Command Language*) as the *embedded interpreter*; which means that any command processed within the `gracos` scripting session is parsed and executed by the *Tcl* (version 8.4) interpreter. This interpreter is extended within `gracos` to include new `gracos`-specific commands and procedures. *Tcl* is currently chosen as the interpreter due to its ease of use, portability, light weight, and the open source license. Typing `tclsh` in your Unix-type prompt brings you to the default *Tcl* scripting session (in the same way as typing `python` for example brings you to the *Python* scripting session). We refer the users who are new to *Tcl* to the online *Tcl* reference (http://aspn.activestate.com/ASPN/docs/ActiveTcl/8.4/tcl/tcl_contents.htm), *Tcl* manual (<http://www.tcl.tk/man/tcl8.4>) and *Tcl* (version 8.5) tutorial (<http://www.tcl.tk/man/tcl8.5/tutorial/tcltutorial.html>). The book *Tcl and Tk Toolkit 1994*, by John K. Ousterhout is also recommended for its coverage of embedding; however it covers a much earlier version of *Tcl*. The `gracos` interpreter differs from the default *Tcl* interpreter exactly by the presence of the specific *Tcl* variables, procedures defined in the `init.tcl` startup file, and the new commands linked to the internal `gracos` C-functions using the `Tcl_CreateCommand` function. File `init.tcl` is sourced each time `gracos` is started. This file contains definitions for custom `gracos` *Tcl* procedures and variables. *Tcl* commands always originate on process zero for parallel runs with MPI. As soon as a complete *Tcl* command is entered on process zero its copies are immediately sent to the remaining MPI processes. The identical commands are then synchronously interpreted on all the remaining MPI processes.

2.3.3 Variables

In addition to the usual *Tcl* variables, `gracos` operates with the so called `gracos` variables (those variables are bound to the pointers inside the C-code).

The complete list of the `gracos` variables is given in Chapter 2 [`gracos` Executable Reference], page 3.

`gracos` variables are operated by `varset`, `varunset`, `varputs`, `vars`, and `varchmod` `gracos` commands, See Section 2.11.1 [`gracos` Commands], page 60. In addition, they are automatically set or updated within the source code, at any point in the source code where such setting is rational.

`gracos` variables are linked to the internal source code variables so that whenever a value of a `gracos` variable is changed in the script the corresponding variable within the source code is also immediately changed. *Tcl* uses the similar kind of variables by introducing the *Tcl linked variables*. We found however that the management facilities provided for the *Tcl* linked variables not sufficient for our purposes at this time and are therefore not currently using *Tcl* linked variables.

`gracos` variables are not associated with *Tcl* variables in any way and are also referred to in this reference as *gracos variables* or simply *variables*. The list of all `gracos` variables for this version of `gracos` is given in Section 2.11.2 [`gracos` Variables], page 65.

Commands, `varset`, `varunset`, `varputs`, `varchmod` and `vars` are used to manage the `gracos` variables from within the `gracos` scripting session. C-macros `varset`, `varget` are used to manage `gracos` variables within the C-code. A `gracos` variable should be carefully distinguished from a regular *Tcl* variable; the easiest way to do it is by keeping in mind that the '\$' character is always used for referring to a *Tcl* variable, but is never used for a `gracos` variable.

Intentionally, no new **gracos** variables can be introduced in the runtime. Each variable is assigned a unique key and a type which do not change in the runtime, See Section 2.11.2 [gracos Variables], page 65 for the complete list of **gracos** variables with their keys and types; however, as we already mentioned, a given variable can be either set or unset, that is - its status may change in the runtime.

gracos variables provide us with an important method of error control intrinsic to the *gracos* scripts. In particular, any attempt to access a variable whose current status is unset results in termination with an error message, regardless of the value of the memory space associated with the variable. The termination behavior can be avoided if necessary, by setting the **unvar** mode of **tetol** bitwise variable, See Section 2.3.4 [Bitwise Variables], page 9.

A given **gracos** variable can in principle have different values on different processes of a parallel run. For example an **nploc** variable, designed to hold the current number of particles on the local process will naturally have different value from process to process because each process generally contains different number of particles.

The list of all the currently set **gracos** variables is displayed by typing **vars** within the *gracos* scripting session (use the **vars -a** for the complete list for the current version of **gracos**). The first three columns in the output of this command are each one character wide. The first column says if the variable is set (' ') or unset ('? '); the second column says if the variable is currently writable (' ') or not ('r'); the third column says if the variable is an array ('A') or a scalar variable (' '). The last two columns of **vars** output show respectively the variable name and its value or values (for the array variables).

Command **varchmod** can be used to switch write protection status of a variable and is useful for various mostly purposes, such as keeping the verbosity constant while loading a datafile which may or may not contain verbosity assignments. The syntax is

```
varchmod {+w|-w} KEY
```

where *KEY* is the name of the **gracos** variable

2.3.3.1 Example: Manipulating **gracos** Variables

In the following example, we illustrate the use of the **gracos** variables versus usual *Tcl* variables. You may execute this *bash* script in your shell to see the result; the script is located under **gracos-examples/interface/vars.sh**.

In this example, we first display the list of the **gracos** variables using **vars** command twice: with and without the **-a** flag. Then we set a **gracos** variable **nstep** to a specific legal value using the **varset** command; then we set a *Tcl* variable to a different value choosing the same variable name. Then we display their values by using **varputs** for **gracos** variable and **puts** for *Tcl* variable. We observe that the variables have kept their originally assigned values and are therefore indeed distinct.

```
#!/bin/bash
# File: gracos-examples/interface/vars.sh
# Author: Alexander V. Shirokov

# GRACOS Session
TmpFile='mktemp ./tmp.XXXXXXXXXX'
cat > $TmpFile <<EOF
```

```

# Display the list of the currently set application variables
vars

# Display the list of all application variables
# (notice that variable nstep is currently unset)
vars -a

# Set an application variable nstep to value 2
varset nstep 2

# Check to see that the status of the variable is indeed changed
vars

# Display the value of the application variable now
varputs nstep

# Define a new Tcl variable with the same name (nstep)
# and set its value to 100
set nstep 100

# Show the value of Tcl variable nstep (the Tcl variable
# dereferencing character is used)
puts \$nstep

# The application variable with this name has kept its original value
varputs nstep

# Application variables are entirely separate from the usual Tcl variables
# even if they are assigned identical names

EOF
mpirun -np 4 gracos -i $TmpFile
rm $TmpFile

```

2.3.4 Bitwise Variables

Bitwise variables are technically just regular `gracos` 4 byte integer variables; they are used to control miscellaneous bitwise (“yes” or “no”s) modes. The `varset` and `varputs` commands applicable to the `gracos` variables in general can be used to manage values of bitwise variables as well. Each bitwise variables can currently be assigned no more than 32 modes.

Type `bitvar -vars` to see the list of all bitwise variables; `bitvars` is a *Tcl* procedure defined in `init.tcl`, See Section 2.3.2 [Commands], page 6.

In *bitwise string* representation bitwise variable is shown as a string of zeros and ones.

The *integer representation* of bitwise variables is more compact than bitwise string representation and is convenient in certain situations. Setting a bitwise variable to `-1` enables all the modes because all bits of the `-1` integer are “yes”. Setting the integer to zero disables all the modes at once. It is not convenient however to operate with integer values if one wants to set some specific bit of a bitwise variable.

Token representation uses tokens to specify values of each mode. The user conveniently operates with tokens, combining them with the `OR` (`'|'`), `AND` (`'&'`) and `NOT` (`'~'`) bitwise operators. Section Section 2.11.2 [gracos Variables], page 65 lists all tokens with brief description for each bitwise variable.

2.3.4.1 Example: Manipulating Bitwise Variables

The `bitvar` procedure defined in `init.tcl` can be used to manipulate bitwise variables in their token representations. The highly commented example below illustrates different ways of manipulating bitwise variables on the example of `tverb` bitwise variable.

```
#!/bin/bash
# File: gracos-examples/interface/bitvars.sh
# Author: Alexander V. Shirokov

# GRACOS Session
TmpFile='mktmp ./tmp.XXXXXXXXXX'
cat > $TmpFile <<EOF

# DISPLAY all bitwise variables
bitvar -vars

# tverb is a bitwise variable controlling the
# gracos technical verbosity modes.

# DISPLAY variable tverb
varputs tverb

# Store its current value
set storedvalue [varputs tverb]

# DISPLAY its value as a string of '0's and '1's
eval [bitvar -bits tverb]

# DISPLAY the its complete list of modes
eval [bitvar -toks tverb]

# We allow unconventional order of the first two flags
eval [bitvar tverb -toks]

# DISPLAY its token representation
eval [bitvar -see tverb]
```

```

# TURN ON the allocation (alc) and MPI use (mpi) modes
eval [bitvar -add tverb {alc | mpi}]

# TURN OFF both allocation and MPI use verbosity
eval [bitvar -del tverb {alc | mpi}]

# Set verbosity to allocation (alc) and MPI use (mpi) only
eval [bitvar -set tverb {alc}]

# Revert to the previously stored value
varset tverb \${storedvalue}

puts "bye!"

EOF
mpirun -np 4 gracos -i $TmpFile
rm $TmpFile

```

2.3.5 Runtime Administration

Runtime administration technique is the perfect tool for doing things on the fly (as the code runs).

The user requests runtime administration by creating specifically named *runtime administration file(s)* within the working directory either before or during the run. The runtime administration files are read by `gracos` as soon as the control reaches any of the *runtime administration checkpoints* within the source code (functions `admin_checkpoint`) with a unique label `admin_label` passed as an argument, See Section 2.3.5.4 [List of Runtime Administration Checkpoints], page 14 for their complete list.

Runtime administration can be invoked explicitly, using the `admin` command. There are a few kinds of runtime administration described in the following text.

There are two important points to keep in mind

- *Avoid partial updates of the runtime administration files* If you edit runtime administration file during the run, casual saving an incomplete version of file may lead to errors if the checkpoint is reached at the time.
- *Avoid possibility of infinite recursion loop.* If your runtime administration method invokes one of the commands that themselves contain a runtime administration checkpoint the infinite recursion loop is avoided by restricting the scope of administration to particular checkpoints; See Section 2.6.1.1 [Example: Tabulated Transfer Function], page 37 for example.

The runtime administration files do not need to have the executable file permissions.

2.3.5.1 One Time Administration

File `inc.tcl`, if found at the runtime administration checkpoint, is interpreted as an input `gracos` script and is then moved to `inc.tcl~`, unless the new `admin_keep` variable is set to 1 while processing that script.

Tcl variable `admin_label`, temporarily defined as the label of the current administration checkpoint, may be used to restrict the scope of one time administration to a particular administration checkpoint in order to avoid infinite recursion. A good example of one time administration with such restricted scope of administration is presented in Section 2.6.1.1 [Example: Tabulated Transfer Function], page 37.

As the most simple example, try executing the following *bash* shell command within the working directory during any of the *GRACOS* runs to see the run stop:

```
SHELL$ echo end > inc.tcl
```

The newly created file `inc.tcl` contains just a single `gracos` command `end` to request an end of the `gracos` session as soon as possible. One should not worry about infinite recursion in this example because, as we know from Section 2.3.5.4 [List of Runtime Administration Checkpoints], page 14 the command `end` does not itself contain another runtime administration checkpoints.

2.3.5.2 Repeatable Administration

The `gracos` variable `admin_path`, if set, defines the name of the *repeatable administration file*, which gets processed at each administration checkpoint without getting deleted. If the repeatable administration file is defined but does not exist the code shall terminate with the error message. In contrast to the one time administration file, the path of the repeatable administration file is not fixed and the file is not removed after getting executed, thus getting processed each administration checkpoint for as long as it is defined and the run is continued.

The content of the repeatable administration file is processed according to its filename extension.

A file whose extension is `.tcl` is interpreted in the same way as described in Section Section 2.3.5.1 [One Time Administration], page 11, except that the file is not replaced after the execution.

A file whose extension is `.py` is processed under *Python* interpreter. The details of the intrinsic procedure are as follows. First an automatically generated header *Python* script is processed in which a number of variables are defined. Type

```
gracos -e env
```

to see what this *Python* header roughly looks like (the real header will differ only by the runtime generated values in the assignments). First, all the *GRACOS* variables are set under their original *GRACOS* names (floating point numbers may lose significant bits in this assignment); those *GRACOS* variables that are unset are initialized to the *Python* `None`. Next, an unrestricted choice of variables are defined by convenience, according to their setting within the source code. The `admin_label` is also defined and can be used to restrict the scope of the runtime administration to a particular checkpoint. The content of the repeatable administration file is then processed in the same *Python* session. Should the user define the `admin_out` *Python* variable, the content of this variable is sources as an input *Tcl* script within the *GRACOS* scripting session immediately following this checkpoint instance.

Note that *Python* interpreter is invoked by using the `system` C-function call, which is known to have portability issues on some rare systems (the *Infiniband* networking systems is the only such system, as far as we currently know). You must have configured *GRACOS*

without `--disable-system-calls` configure option for the *Python* runtime administration to work; See Sections Chapter 4 [Installation Procedure], page 83, and Section 3.10.1 [ibtest: System Call Portability Test], page 81 for more details.

2.3.5.3 Scheduled Administration

Scheduled administration method is useful for scheduling certain tasks to be executed at a pre-specified time.

File `'sched.txt'`, if present in the working directory, must be the text file containing the integer (of may be floating point number) representing the number of seconds since 1970-01-01 00:00:00 UTC at which to schedule executing a command. the standard of output of C-command `time(NULL)` of shell command `date +%s`.

The command, which is the content of variable `cmd_sched`, will be executed as soon as control reaches one of the scheduled administration checkpoints and time exceeds the number written in file `'sched.txt'`. The content of the variable is subject to the same substitution rules as those applicable to the output control variables described in Section 2.8.1 [Checkpoint Control Variables], page 56.

Below is the example of scheduled administration

```
#!/bin/bash
# File: gracos-examples/problems/sched.sh
# Author: Alexander V. Shirokov

input_file='gracos-config info pkgdatadir'/ParticleData/p3m-0.8_'uname -m'.dat

# Setup scheduled administration in 5 seconds from now
sched_sec=5
echo 'date +%s'+$sched_sec | bc > sched.txt

# GRACOS Session
TmpFile='mktemp ./tmp.XXXXXXXXXX'
cat > $TmpFile <<EOF

# Load particle data from a file
dataman -mode=load -fmtid=p3mdat -masseq -path=$input_file

# Define scheduled administration command:
# "checkpoint using a restart outout and exit"
varset cmd_sched "checkpoint --labels=r; exit"

# Specify restart output
varset out_restart "dataman @MODE@ -fmtid=udf -spec=nfio:1 -path=out-@LABEL@.udf-"

# Run integration with the target expansion factor '1' for restart output.
# Note: it will be interrupted in 5 seconds rather than actually
# finishing at this expansion factor.
integ -aouts=1.:r
EOF
```

```
mpirun -np 4 gracos -i $TmpFile
rm $TmpFile
```

2.3.5.4 List of Runtime Administration Checkpoints

The following is the list of all runtime administration checkpoint ids.

<code>integ</code>	Invoked each timestep of the particle integration run, See Section 2.8 [integ: Running an N-body simulation], page 54.
<code>integ_pre</code>	Invoked at the beginning in the <code>integ</code> command, just before the main timesteping loop of the integration of particle trajectories starts.
<code>image</code>	Invoked just after the image data is projected to form a numerical matrix-plate but before it is converted to produce a 'gif' file, during the <code>image</code> command, See Section 2.5 [image: Particle Data Imaging], page 28. This administration checkpoint can be used to tune the dynamic range covered by the image to the brightness of the dimmest and the brightest spots on the image plate.
<code>admin</code>	Invoked when command <code>admin</code> is executed in the interpreter.
<code>env</code>	Invoked when command <code>env</code> is executed in the interpreter.

2.3.6 Primary Output Files

A run with `gracos` executable produces a number of *primary output files* within the working directory. These files contain useful information about the run and the used executable.

- File `gracos_history`: The list of commands submitted to `gracos` interpreter so far. This file is created on process zero and is updated during the run as soon as a new command starts being interpreted. If you look at this file during the runtime, the line at the bottom indicates the currently processed command or procedure.
- Files `so-Rank`: The standard output of process whose MPI rank is *Rank* (an integer). The files are created, one for each process, and are updated as the run progresses. The amount of data written into these files critically depends on the settings of the verbosity, controlled by the bitwise variables `tverb` and `sverb`, See Section 2.3.3 [Variables], page 7, See Section 2.3.4 [Bitwise Variables], page 9.
- File `gracos_info`: Structured document containing the version- and run-specific information. This file is updated only a few times during the run and is created on process zero only.

The file is structured as an XML document, and is thus easily parseable by any of the standard XML parsers, such as `xml.dom` in *Python*.

The first XML node *configure* contains the configuration and version information on the executable. The second node *run* contains information specific to each particular run: the time at which run was started in two formats (*time* and *date*), the number of MPI processes associated with the run (*nproc*), and the process specific information for each process (*processes*): the *rank*, the *hostname* of the hosting node as the current process, the process ID on hosting node, and the *multiplicity* - the total number of processes assigned to the same hosting node as the current process. The greater than one value of the multiplicity indicates that the resources of the node are shared with the other processes associated with the same run.

Shown below is the example of this file, when created by the run on four processes in the `hello.sh` example in Section 2.1 [Running `gracos`], page 3

```
<configure>
  <property key="PACKAGE" value="gracos"/>
  <property key="PACKAGE_VERSION" value="1.0.1a8"/>
  <property key="CONFIG_CFLAGS" value="-Wall -O3 -fno-strict-aliasing"/>
  <property key="CONFIG_CPPFLAGS" value="-I/home/shirokov/local/arch/ [TRUNCATED]
  <property key="CONFIG_LDFLAGS" value="-L/home/shirokov/local/arch/i [TRUNCATED]
  <property key="CONFIG_DATECFG" value="Thu Nov  8 16:30:08 EST 2007"/>
  <property key="CONFIG_USERCFG" value="shirokov"/>
  <property key="CONFIG_ABS_TOP_SRCDIR" value="/home/shirokov/gracos- [TRUNCATED]
</configure>
<run status="initial">
  <property nproc="4"/>
  <property time="1194557544"/>
  <property date="Thu Nov  8 16:32:24 2007"/>
  <processes>
    <process rank="0" hostname="procion" pid="1690" multiplicity="4">
    <process rank="1" hostname="procion" pid="1691" multiplicity="4">
    <process rank="2" hostname="procion" pid="1692" multiplicity="4">
    <process rank="3" hostname="procion" pid="1693" multiplicity="4">
  </processes>
</run>
<run status="final">
  <property runtime="0">
</run>
```

2.3.7 Error Behavior

If `gracos` halts before having reached the `return` statement of `main` C-function it does so abnormally. These failures may be due to a variety of reasons. Excepting the batch job resource limit and

1. *Internal test failures* always trigger a call to one of the C-macros `stop` or `stopm("ERROR MESSAGE")` defined in `zero.h`. The macros produce message(s) similar to these

```
Exit (gracos.c:278)
```

or

```
ERROR MESSAGE
Exit (gracos.c:278)
```

in the standard output of the process(es) where the internal test failure(s) occurred. These messages appear at the bottom of one of the standard output files `so-Rank`, See Section 2.3.6 [Primary Output Files], page 14; and show the location of the failed test within the source code. The *ERROR MESSAGE*, if present, provides useful information. For example it may say that a particular variable is left uninitialized or that a command with an unknown name is used.

In some cases, a termination on test failure is unnecessary. Various error tolerance modes are provided to avoid abnormal termination for these cases (See the modes `tetol` `gracos` variable in Section 2.11.2 [gracos Variables], page 65). For example, if a user runs `gracos` interactively using standard input and does not wish run to be terminated in case of a typo, setting the `unfun` and `unvar` modes “on” may be useful.

A memory leak always indicates the presence of a bug, and resolving it is a very high priority. It is extremely easy to mismanage memory in a C-application: the most common situation is when the memory is allocated somewhere in the code but is not released when it is no longer needed. If such an error has occurred in a loop the amount of the allocated memory space accumulates reducing and in some cases exhausting the available memory resources of the machine.

`gracos` uses the macros declared in `galloc.h` of `basal` sub package for memory management, See Section 3.4 [GRACOS Library], page 76. If a memory leak occurs on any MPI process and does not lead to a crash it can be reliably detected at the end of the run by comparing (for each process) the amount of total memory space allocated during the run with the same released during the run. If the two numbers do not agree it becomes established that there is a memory leak. In this case, independently of the verbosity and other settings a conspicuous message similar to the following appears at the bottom of the standard output file `so-Rank` of each process:

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WARNING: memory leak on this process:
cntpc=1  mempc=7
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
WARNING: memory leak on at least one process:
cntsum=4  memsum=20
```

assuming this message appeared at the bottom of file `so-2` the first warning indicates that cumulatively, seven bytes of memory were allocated and were not released by the end of the run (follows from the statement `mempc=7`); one memory block of an unknown size was allocated and was not released on this process (follows from `cntpc=1`). The second warning message indicates that the memory leak has occurred on *any* process of the world group (the numbers shown are sums of absolute values for all processes) and appears at the bottom of all files `so-Rank` independently on whether the memory leak is local to the process associated with the file.

The location of a memory leak in the source code can be found using the utilities `pairs`, `cmlchk.py` and `ptrchk.py`. Repeat the same run with the allocation verbosity turned on (e.g. by using the `gracos` command line option `--verb=-1`) and do not modify the verbosity setting in the runtime.

In most cases the location of the memory leak in the source code (filename and line number) can be found by processing the standard output file(s) `so-Rank` whose bottom indicating the presence of the local memory leak with the ‘`pairs`’ executable as follows

```
SHELL$ pairs aptr= fptr= so-Rank
```

The other two utilities are applicable in more complicated and fortunately much rarer situations when the above method does not work: `cmlchk.py` checks the cumulative values of memory displayed in the standard output file against the values computed using the memory increments displayed in the same file; `ptrchk.py` checks memory balance for each pointer individually.

2.4 File Input and Output with `dataman`

`GRACOS` supports a number of formats for primary (e.g. suitable for a whole restart of the simulation) file input-output; they all are implemented with the `dataman` command managing high level file input-output. The data is converted when needed from the the

units particular to the datafile format into the units used by `gracos`. Sometimes, as in case considered in Section Section 2.4.8 [Example: Santa Barbara Cluster Project], page 24 such unit conversion requires same `gracos` variables to be set; if so, not having them set will result in an error message indicating which of the required parameters are missing.

A data file format can be *static* or *dynamic*. The static datafile format allows only a predefined number and type of the simulation parameters be stored; while the dynamic format allows more freedom. In addition, a data file format can be *distributed* and *serial*. In the distributed File I/O format, more than one process are allowed to write data on disks. For the serial datafile format the whole file I/O procedure is always performed on one process only. In the parallel runs the processes exchange messages with the process using MPI.

Below is the full description of the options; Section Section 2.4 [File Input and Output with `dataman`], page 16 lists all the available datafile formats for `dataman`.

`dataman OPTIONS...`

Load or save the data from or into a datafile(s).

The following table lists the available *OPTIONS*.

`-mode=string`

Mandatory flag used to specify the required action. The possible values of the string argument are `save`, `load`, or `delete`, indicating file output, input or deletion.

`-fmtid=string`

This is a mandatory flag. The data formats used for file I/O are identified by their *Format ID*, the argument of `-fmtid` specifies the *Format ID* (See below for the list of all the supported formats). In the absence of this option, the code tries to determine the format using the extension of the path given by the `-path` flag argument. If both methods fail the run stops with an error message, since the machine does not know which format to use for file I/O.

`-path=string`

This is a mandatory flag; its argument provides the longest path prefix that is a prefix of all the paths used for I/O in the current instance of file input-output. The final file pathname will be appended when necessary: for example, in parallel I/O the path is appended with MPI ID of the process performing the I/O; in case when many file paths are used on one process a proper extension used to distinguish the path is added; See also the `-suf` and `-prefixes` options.

`-suf=string`

Optional flag, used to append a string to the end of the file path.

`-prefixes=string`

When local disks are cross-mounted, this option can be used to modify the leading parts of `-path`. The argument of `-prefixes` is a text file, readable on process zero, and is the list of the prefixes in order of their process ids. The leading portion of the `-path` argument is appended individually for each process whose process id is *rank* with the content of the row number *rank* in this file. If the file has less than *rank* rows the path is reset to *NULL* for this process. The file should end with a newline.

-spec=string

The argument provides the format specification string and is used in conjunction with the format ids for which such format specification is applicable; see the description for each format below for documentation.

-masseq If masses of all particles are the same it is not efficient to store them in a datafile. If masses of particles are not stored in a datafile used for particle data input, this option should be used to initialize them to the same value.

-sort Sort particles with respect to their particle ids before writing them to disk. This option is applicable only for data output (*-fmtid=save*) using serial data file formats and is silently ignored when used in all other cases.

2.4.1 Physical Units

The particle data are normally expressed within **gracos** source code in so called *gracos code units*: the particle positions are comoving, in the units of \mathbf{dx} ; the velocities are proper, in the units of $(h_0 \cdot \mathbf{dx}/a)$, See Section 2.11.2 [gracos Variables], page 65 for the definitions of h_0 , \mathbf{dx} , and a . The units of mass are chosen so that the total mass of all the particles in the simulation box is normalized to equal the number of PM density/force mesh grid points $n_x \cdot n_y \cdot n_z$. For the case of equivalent particles, one particle per each PM-mesh cell, each particle has a unit weight in code units.

2.4.2 binpart: The Simplest Format

In this dynamic serial format (**binpart** signifies the *Format ID*) the particle data are stored in a single file in the most simple possible way, controlled by the user. No header is used, and the data is read or written in a simple sequence of records, one for each particle in the simulation volume.

The user must provide the definition for the layout of data in the record using the *format specification* option **-spec** to the **dataman** command. The syntax for the **-spec** argument string is a comma-separated list of tokens given below

x, y, or z

The x,y, or z- components of the comoving particles coordinates, in Megaparsec.

vx, vy, or vz

The x,y, or z- components of the proper peculiar velocity, in km/s.

vxc, vyc, or vzc

The x,y, or z- components of the comoving peculiar velocity, in km/s.

m

The mass of the particle in the solar masses.

id

The particle id, currently a 4 byte integer.

xs, ys, zs

vxs, vys, or vzs

ms

`gxs`, `gys`, or `gzs`

The coordinates, velocities, masses and gravitational accelerations, in the *GRACOS* code units, See Section 2.4.1 [Physical Units], page 18.

If the `id` specifier is not supplied on input, the particle ids are assigned sequentially in order of their appearance in the file, starting from 1.

The `binpart` format is already used widely due to its simplicity. For example The Santa Barbara Cluster project (<http://t8web.lanl.gov/people/heitmann/axiv/sb.html>) has adopted a variation of this format to publish their particle data. Section Section 2.4.8 [Example: Santa Barbara Cluster Project], page 24 shows how, using the `dataman` to load the particle data in their format directly into `gracos` and start N-body simulation in the same session.

This format does not have any built-in header for holding any variables. Some additional parameters are generally required if one uses this format for input, e.g. for the domain decomposition. One has to manually supply the missing information with `varset` before the `dataman` command can be called successfully. If a required variable is unset the run terminates with an error message indicating the name of the required variable. The `npartall` variable is set automatically based on the input file size and the format specification.

2.4.3 `udf`: Unified Datafile Format

This dynamic distributed format (*Format ID: `udf`*) is the perfect choice for easy simulation restart and runtime backups automatically stores all the necessary `gracos` variables and data, being also very fast due to the usage of local disk space (some networking systems however do not have local disk spaces). The `udf` format is however somewhat frequently changed between different `gracos` versions and is hard to be interpreted by other software. However, if the data is written in this format one can always restart the simulation and convert the data to other easily interpretable formats such as `binpart`, See Section 2.4.2 [binpart: The Simplest Format], page 18.

The `udf` format is customizable and self-describing. It allows the option for the arbitrary number of the output files not exceeding the number of running MPI processes. Each process writes at most one output file. This format is portable between the machines of the same CPU-endian types. The datafiles are currently designed to store all the code variables, particle data, and the refinement numbers of the chaining mesh cell.

The `-spec` argument is used for file output only. The argument of `-spec` option is a string: a comma-separated list of tokens with their subarguments, following the below syntax

`eq` Request equal sized output files. The cost is slightly slower completion due to the necessary interprocessor communications. This option is useful for use in clusters with local disc spaces on each node; it ensures that the disc space gets used up uniformly.

`nfio:int32`

Integer (the default value is `Nproc`); option results in the specified number of the output files. If option `-eq` is given, these files will be equal sized. The actual data is not moved while their copies are

moved between the processes in the failure tolerant way so as to accommodate the specified number of the output files. This option should be used if one wants to reduce the number of output files without compressing the data to the bottom processes. if `nfio` is less than `nshr` (below) then the option `-eq` is automatically set. If `nshr` is less than `nfio` then `nfio=nshr` is set automatically.

`nshr: int32`

Integer, less than or equal to N_{proc} ; the option allows user the ability to later restart the simulation on fewer number of processes than the running number of processes (N_{proc}). Repartitioning of domain decomposition is performed so that the whole domain gets divided evenly (weighed by the workload per cell) in the result between the `nshr` (as opposed to N_{proc} as it normally is) bottom processes before data is saved into the datafiles. The simulation can be restarted in a separate `gracos` session using a number of processes (that is equal or higher than `nshr`). Do not use this option if you do not plan to restart the same simulation on fewer number of processes.

2.4.4 tpm: Tree-Particle-Mesh format

The static serial data format (*Format ID*: `tpm`) adopted by the TPM code version 1.2, See <http://www.astro.princeton.edu/~bode/TPM> is composed of 6 binary files with the same basename and extensions `.px`, `.py`, `.pz` and `.vx`, `.vy`, `.vz`. They contain identical information in the header and the corresponding component of position or velocities of the particles in their main bodies.

2.4.5 p3mdat: Fortran p3m Datafile Format

The static serial format (*Format ID*: `p3mdat`) used by the p3m serial *Fortran 77* codes. The data is saved onto one file consisting of the header and the main body. The number of particles that can be saved is limited in the format to the maximum number held by an integer (`INT_MAX` as defined in the standard `limits.h` C-header). The error message will appear and the run will stop if this limit is exceeded.

Particle IDs are not stored. However one may use particle sorting procedure to store particle IDs implicitly. Pass `-sort` option to `dataman` and particles will be written into the disk in the order of increasing particle IDs. While loading from a `p3mdat` datafile, particle ids are automatically set to the sequential number (starting from one) in which the particle appears in the datafile. In this way particle IDs are easily recovered.

The masses of particles are also not directly stored. The `-masseq` option must be used to initialize the masses to the equal value. The `p3mdat` format is supported by the Serial *Fortran 77* codes that we are using for tests and verification against the older `p3m2_93` codes.

The following table lists the data items stored in a `p3m.dat` header in the order of their appearance in case.

long unsigned

Fortran 77 conventional mark for the start of a binary File I/O record.

`npartall` Same as `npartall` from Section 2.11.2 [`gracos` Variables], page 65, but here constrained to 32 bits. The number of particles can not exceed `INT_MAX` in this format.

`nx, ny, nz`
See Section 2.11.2 [`gracos` Variables], page 65.

`dx, epsilon`
Expressed in the comoving Megaparsecs.

`a, omegam, omegav, h0, dtsin, etat, and nstep`
See Section 2.11.2 [`gracos` Variables], page 65.

`ekin, egrav, and egint`
Same as in Section Section 2.11.2 [`gracos` Variables], page 65, but expressed in the physical units; the conversion factor is given in Section Section 2.4.1 [Physical Units], page 18.

long unsigned
End of the *Fortran 77* record.

2.4.6 `mwhite`: Martin White format

This format has at some point been adopted by *Martin White* for his codes. Its original description follows

The phase space data files have to be in a specific format. First there is a 4-byte integer, which should be 1. Then an integer 20. Then a header which contains: number of DM particles, number of SPH particles (0 in our case), number of star particles (0), scale-factor of the output, spline softening length in units of the box size. The first 3 entries are 4 byte ints, the last 2 are 4 byte floats. If you know a Plummer softening, the spline softening is a factor 2.8 larger. Then all of the particle positions, 3x 4-byte floats per particle, in units of the box length. So all coordinates should run [0,1) and the order is `x0,y0,z0,x1,y1,z1`, etc. After the positions are the velocities `vx0,vy0,vz0,vx1,vy1,vz1,...` These are stored in units of the Hubble velocity across the box. If you have them in km/s then divide by `a.H.Lbox`. Finally the particle ID number for each particle, as a 4-byte integer. And that's it.

This format allows the following specification with `-spec` option, whose argument should be a comma-separated list of tokens given in table below

`nfh:int32`

Allows one to use `nfh` files instead of just one (the default); the files are still written/read on process zero. Files written with some value of the argument of `nfh` should be read with the same argument to `nfh`.

2.4.7 Example: Datafile Manipulations

N-body problems generally require storing many kinds of data on a disk for various purposes such as to be able to restart the simulation from a midpoint, or perform data analysis on particles in a separate run. New datafile formats are routinely devised in the N-body community in order to fit particular problems.

`gracos` offers a file interface that supports a few data formats, See Section 2.4 [File Input and Output with `dataman`], page 16. The interface is implemented through the `dataman`

command, allowing the user to load, save, or delete the data stored in dataliles, or make a conversion between the supported formats. The list of the supported file formats will be updated based on the need.

Most of the examples in this reference use `dataman` in one way or another. Any of the supported datafile formats can be used, for example, to start a full N-body simulation with `gracos` in the same session. The example in Section 2.4.8 [Example: Santa Barbara Cluster Project], page 24 shows how to load data files using the format adopted by the Santa Barbara Cluster Project.

In the densely commented example below we perform a few conversion between various supported formats listed in Section 2.4 [File Input and Output with `dataman`], page 16.

```

#!/bin/bash
# File: gracos-examples/interface/fio.sh
# Author: Alexander V. Shirokov

# Terminate shell session on any errors
set -e

# Specify the input datafile
InputFile='gracos-config info pkgdatadir'
InputFile=${InputFile}/ParticleData/p3m-0.8_'uname -m'.dat

# Running gracos on 8 processes
TmpFile='mktmp ./tmp.XXXXXXXXXX'
cat > $TmpFile <<EOF

# Load data from a p3m.dat file, assign equal masses to all
# the loaded particles.
dataman -mode=load -fmtid=p3mdat \
  -masseq \
  -path=$InputFile

# Compute acceleration to find cell workloads
accel

# Save data in three files with possibility to restart
# on four or more processes.
dataman -mode=save -fmtid=udf \
  -spec=nshr:4,nfio:3 \
  -path=a.udf-

EOF
mpirun -np 8 gracos -i $TmpFile

# Restarting gracos on the smaller number (4) of processes

```

```

cat > $TmpFile <<EOF

# Read the output of the previous run
dataman -mode=load \
  -fmtid=udf \
  -masseq \
  -path=a.udf-

# Save into four files of equal size
dataman -mode=save -fmtid=udf \
  -spec=eq \
  -path=b.udf-

# Save particle data into one file in the most simple binary
# format possible: the sequence of binary numbers representing the
# particle positions (x,y,z), velocities (vx,vy,vz), the mass (m)
# and the particle id (id) for each particle in the simulation volume.
dataman -mode=save -fmtid=binpart \
  -spec=x,y,z,vx,vy,vz,m,id \
  -path=t1.bin

#
# Save particle data similarly to the above case, now using
# other choice for units of coordinates and velocities;
# and the produced sequence of particles in the file is now
# sorted with particle ids.
#
dataman -mode=save -fmtid=binpart \
  -spec=xs,ys,zs,vxs,vys,vzs,id \
  -sort \
  -path=t2.bin

# Again, a similar procedure
dataman -mode=save -fmtid=binpart \
  -spec=x,y,z,vx,vy,vz \
  -path=t3.bin

EOF

mpirun -np 4 gracos -i $TmpFile

# Restart gracos on a larger number (10) of processes
# and read the output produced by the previous run.
cat > $TmpFile <<EOF

# Set the necessary missing parameters

```

```

varset epsilon 0.1
varset nx 32
varset ny 32
varset nz 32
varset dx 1.0
varset h0 50
varset a 0.8
varset omegam 1.0

# Load particle data from the file written in the above
# gracos session.
dataman -mode=load -fmtid=binpart \
  -spec=x,y,z,vx,vy,vz -masseq \
  -path=t3.bin

# Save particles
dataman -fmtid=binpart -suf=.bin \
  -spec=xs,ys,zs,vxs,vys,vzs,ms \
  -mode=save -path=a-0

vars

EOF
mpirun -np 10 gracos -i $TmpFile
rm $TmpFile

```

2.4.8 Example: Santa Barbara Cluster Project

In our next example, we start the simulation with the Santa Barbara Cluster Project (SBCP) initial conditions. These particle data initial conditions must first be downloaded and unarchived using the following Unix-flavored shell commands

```

SHELL$ wget http://t8web.lanl.gov/cosmic/ic_sb128.tar
SHELL$ tar xvf ic_sb128.tar
particles_ic_sb128
input_sb

```

The data in the unpacked data files is written using the following format, as quoted from the *SBCP* webpage:

```

... There are two files in the archives: particles_name, which has the following content:
x[Mpc], v_x[km/s], y[Mpc], v_y[km/s], z[Mpc], v_z[km/s], particle mass[M_sol], particle
tag. This file is written in single precision and binary format. ... All data are given
in comoving units ...

```

Note that the velocities are expressed in comoving km/s, which is unusual.

In order to load the datafiles into `gracos`, we use the above description to set the `-spec` flag appropriately in the example below, See Section 2.4.2 [binpart: The Simplest Format], page 18.

The example below is a script that should be executed in the same directory as the above download procedure. The important thing to notice here is that the particle datafile alone does not completely specify the simulation parameters in this case. Additional simulation parameters must be set in order for *GRACOS* to read the particle datafile; this is because part of the necessary information is supplied in the text file `input_sb`, some more information (the initial redshift or the starting expansion factor `a`, etc) are provided in the *SBCP* web page; additionally, *GRACOS* uses its own units for data and domain decomposition which depend on a few more simulation parameters such as `epsilon` and `etat`. It is easy to find the list of the necessary parameters since skipping any of the settings inevitably results in a crash with the appropriate error message indicating which of the parameters is missing. Another important thing is the use of the `-spec` option to `dataman`. The format specification is made consistent with the description quoted above.

Next to loading the particle data from the SBCP initial conditions data, the script executable below generates the power spectrum, and the images of the density distribution simulation box as commented in the script. According to the setting for `img_slabs` code variable, four images are produced in slabs along the line of sight, See Section 2.5 [image: Particle Data Imaging], page 28 for more examples and explanations on the image parameters.

We encourage the users to execute the below script (after finishing the above procedure for downloading the SBPC data) to reproduce the results.

```
#!/bin/bash
# File: gracos-examples/problems/sbarb.sh
# Author: Alexander V. Shirokov

# Make bash session halt on any errors
set -e
TmpFile='mktemp tmp.XXXXXXXXXX'

# Files downloaded from the Santa Barbara Cluster Project webpage
SB_header=input_sb
SB_particles=particles_ic_sb128
if [ ! -f $SB_header ] || [ ! -f $SB_particles ]; then
    echo "Missing one or both files: \"$SB_header\", \"$SB_particles\""
    exit 2
fi
# Extract more simulation parameters from the header datafile.
# This defines: Lb, Om, Ov, h0
sed 's/ //g' input_sb > $TmpFile
. $TmpFile

z=63.;
Ngrid=128;
astart=$(echo "1./(1.+$z)" | bc -l)
```

```

dx=$(echo "$Lb/$Ngrid" | bc -l)

# GRACOS Session
cat > $TmpFile <<EOF

# Set the necessary parameters
varset nx $Ngrid
varset ny $Ngrid
varset nz $Ngrid
varset dx $dx
varset h0 $H0
varset a $astart
varset omegam $Omega_m

# Specify the Plummer softening distance and
# Load the Santa Barbara Cluster Project datafiles into GRACOS.
varset epsilon 0.1
dataman -mode=load \
  -fmtid=binpart \
  -spec=x,vxc,y,vyc,z,vzc,m,id \
  -path=$SB_particles

# Measure the power spectrum of density perturbations;
# display power spectrum with the "pmpower" MATLAB command.
pmpower

# Produce the images of the simulation box:

# First, Set the image parameters "reasonably", without producing
#       the actual image yet
image --set-defaults -no-image

# Second, Set a few selected image parameters to more preferred values
varset img_rbl 0.95
varset img_rbr 1.4
# Make four image snapshots in slabs, along the line of sight
varset img_slabs 4
varset img_pixw 500
# Increase the default img_rsmoo by a factor of 1.5
varset img_rsmoo [expr 1.5*[varputs img_rsmoo]]

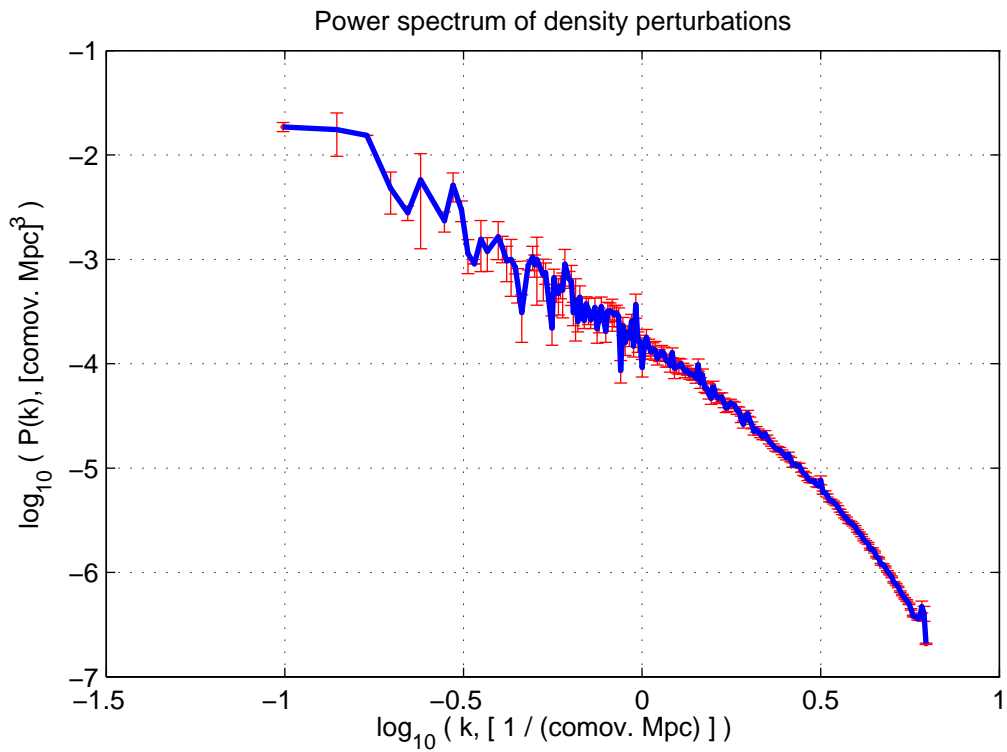
# Finally, produce the actual images
image -path=sbarb.gif

EOF

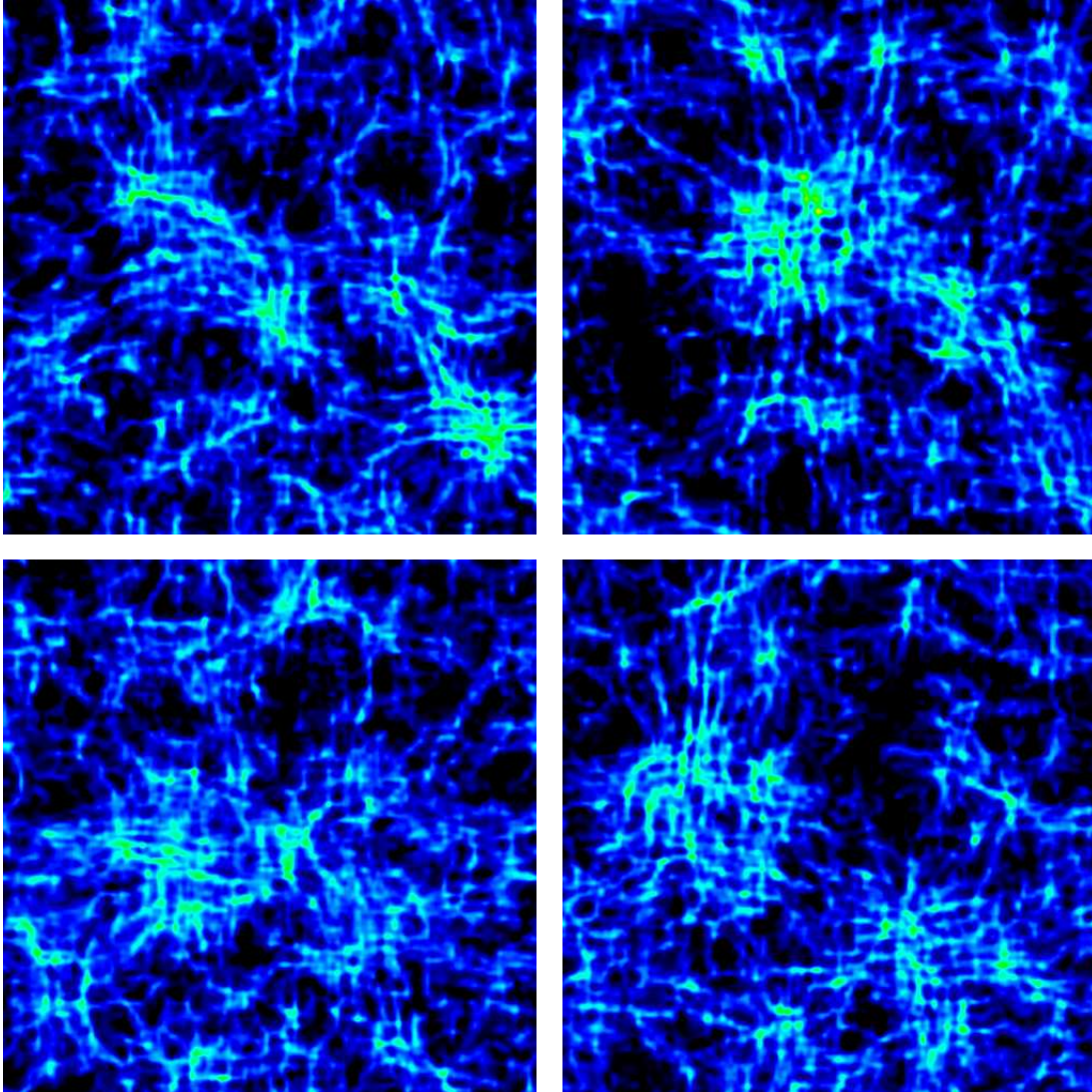
```

```
mpirun -np 8 gracos -i $TmpFile
rm $TmpFile
```

The image below shows the power spectrum measurement for matter distribution in the particle data volume. The image data are generated as the result of the `pmpower` call in the above example; the data are written into file `pmpower.dat` and the image itself is generated by typing the same `pmpower` (in *MATLAB*) in the same directory, See Section 2.7 [pmpower: Power Spectrum Estimation], page 54. The lower half of error bars is longer than the upper one due to the non-linearity of logarithm while we use linear values power spectra for variance estimators.



The following four images appear as the result of the `image` call in the above example; the number of images is defined by the `img_slabs` variable setting the number of image slab along the line of sight.



2.5 `image`: Particle Data Imaging

`image` is a `gracos` command implementing the imaging of particles with various choices for the projection volume, direction and other imaging parameters. The maximum dynamic range of 256 colors is currently allowed for imaging. The color 0 in the palette corresponds to the lowest threshold value below which no contrast can be discerned from the image. Similarly, the color 255 corresponds to the highest threshold in the plotted value.

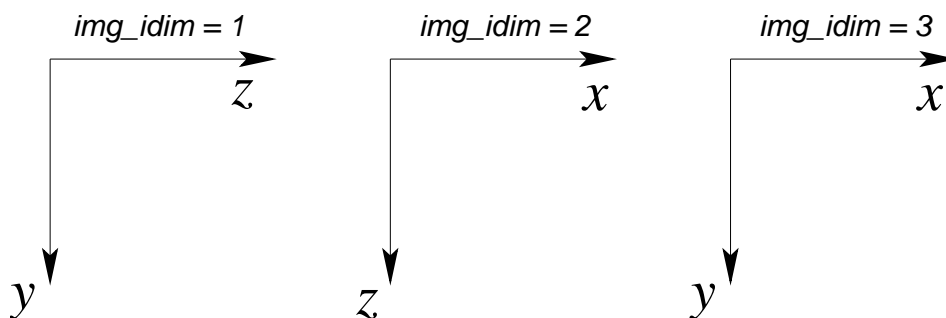
A number of code variables must be set before this command is called. One can avoid having to set them all manually by using the `-set-defaults` flag:

```
gracos% image -set-defaults
```

to generate a reasonable guess to all the required variables from the current data, See Section 2.5.1 [Example: Particle Data Imaging], page 29 or Section 2.6.1.1 [Example: Tabulated Transfer Function], page 37 for the examples of the use of this command.

The following is the complete list of the required variables

- `img_cmap` An integer within the range 0-4, specifying the colormap to be used for images. We found the colormaps 1 and 4 to be most useful. The colormap 4 produces black and white images.
- `img_dim` An integer 1,2 or 3, stands for the direction (x,y or z) of the axis used for the projection of the simulation box. The plane of the generated image will span the remaining two dimensions as shown in the following figure.



- `img_cx, img_cy, img_cz`
Specify the center of the main image box within the simulation volume.

- `img_bx, img_by, img_bz`
Specify the size of the main image box.

- `img_slabs`
An integer greater than zero, setting the number of images in the output of the `image` command. If this number is 1 (the default), then the content of the whole main image box, whose dimensions are given by (`img_cx, img_cy, img_cz`), and (`img_bx, img_by, img_bz`) is projected into the image plane. If this number is greater than one, then the main image box is split into the same number of slabs of equal thickness and the output images are the projections for each individual slab in the sequence.

- `img_pixw` Specify the output image width, in pixels. The height is found automatically so as to observe isotropic scaling in all directions within the image plane.

- `img_rsmoo`
Image smoothing length.

- `img_rbl, img_rbr`
The two floating point variables set the contrast.

2.5.1 Example: Particle Data Imaging

In the example below, we first load the particle data from a sample datafile (supplied along with the source code distribution) using `dataman`, then we produce the images of

the distribution of particles in the simulation box using different colormaps, projection dimensions, image sharpness and image dimensions.

```
#!/bin/bash
# File: gracos-examples/problems/imaging.sh
# Author: Alexander V. Shirokov

# Specify input datafile
pkgdatadir='gracos-config info pkgdatadir'
InputFile=$pkgdatadir/ParticleData/p3m-0.8_'uname -m'.dat

# GRACOS Session
TmpFile='mktemp ./tmp.XXXXXXXXXX'
cat > $TmpFile <<EOF

# Load particles from a sample datafile
dataman -mode=load -fmtid=p3mdat -masseq -path=$InputFile

# Set image parameters "reasonably", do not actually write the image
image --set-defaults -no-image

# Produce four images with various choices on image parameters:
# Image 1: the default parameters, set by "image --set-defaults"
image -path=im1.gif

# Image 2: change the projection dimension (project along the Y-axis)
#           change to another colormap
varset img_cmap 4
varset img_dim 2
image -path=im2.gif

# Image 3: Recenter the frame to the halo at the upper right edge
#           zoom in, sharpen the resolution scale by the factor of two,
#           increase the upper brightness limit range by a factor of two.
varset img_cmap 0
varset img_cx [expr [varputs img_cx]+0.5*[varputs img_bx]]
varset img_cz [expr [varputs img_cz]-0.2*[varputs img_bz]]
varset img_bx [expr 0.3*[varputs img_bx]]
varset img_bz [expr 0.3*[varputs img_bz]]
varset img_rsmoo [expr 0.5*[varputs img_rsmoo]]
image -path=im3.gif

# Image 4: Invoke Runtime Administration
varset admin_path imaging-admin.py
image -path=im4.gif
varunset admin_path
```

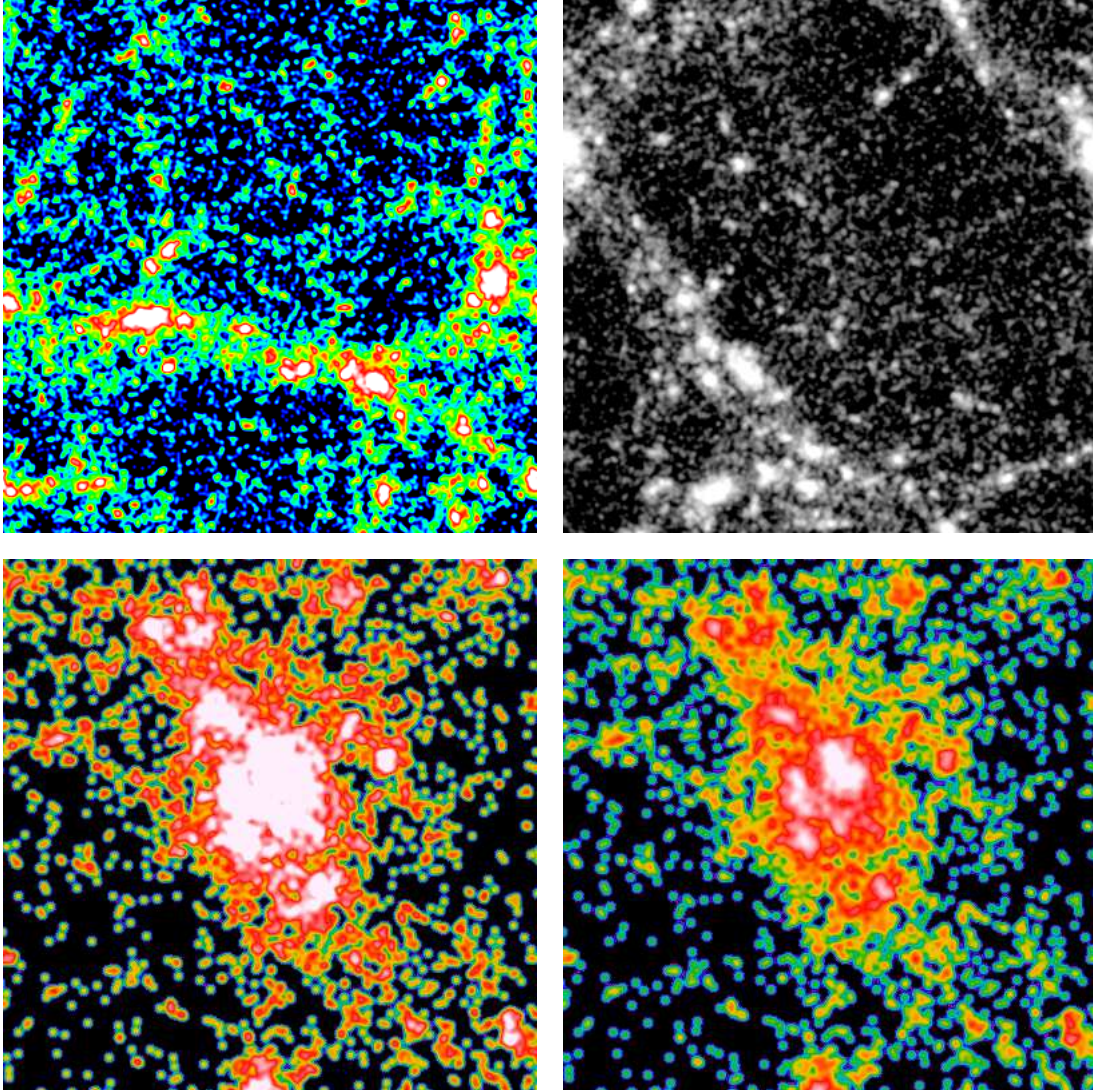
EOF

```
mpirun -np 4 gracos -i $TmpFile  
rm $TmpFile
```

The following runtime administration file must be placed into the same directory under the name `imaging-admin.py`, See Section 2.3.5 [Runtime Administration], page 11.

```
# Runtime administration file for imaging.sh  
# File: gracos-examples/problems/imaging-admin.py  
# Author: Alexander V. Shirokov  
  
if admin_label == "image":  
    admin_out = "varset img_rbr %E" % readonly_cmlup
```

The four images below have been produced by running the above script. They are originated in the left-to-right, top to bottom order as files `im1.gif`, `im2.gif`, `im3.gif`, and `im4.gif`.



Note that the third image is particularly overexposed. This is because, the default value (10.) for the `img_rbr` code variable, giving the upper threshold for the brightness range sampled by the available colormap, is too low for this overdense halo; so that most pixels on the image fall above the threshold (this default value is set by the `image --set-defaults` in the above script). In order to tune the threshold to the brightness distribution of the pixels we have to use runtime administration technique described in Section 2.3.5.2 [Repeatable Administration], page 12 because it is generally not possible to determine the correct value for the upper brightness threshold before the actual data is projected onto the image plate. The instance of runtime administration is invoked for the last image, at the `image` checkpoint in which the upper brightness threshold is set exactly to the value of

the brightness threshold `Out_cmlup` exceeded by a low fraction (0.003) of the total number of pixels in the image. This is a good example where runtime administration technique is useful.

2.6 Initial Condition Generators

In general, any data simulation or data analysis task is preceded by the specification of the current state of the system. In our case, we specify the data such as the particle positions, velocities etc; followed by the parallel domain decomposition.

This section shows the examples of running a large simulation with *GRACOS* for various choices of the initial conditions; these examples are presented in the form of readily executable scripts. Cosmological initial conditions are currently generated using the Zeldovich approximation with four methods of specifying the initial power spectrum of density perturbations. This section lists all the currently available methods of automatically generating the initial conditions; both rectangular and glass initial particle grid can be used.

Once the initial conditions are generated: the data can be analyzed or evolved, and saved into the datafiles using one of the methods described in Section Section 2.4 [File Input and Output with `dataman`], page 16. Once saved into datafiles, the data can be loaded evolved and analyzed again in a separate *GRACOS* scripting session. In general, to start a cosmological simulation, the user is only required to provide the cosmological and other simulation parameters. *GRACOS* starts with those parameters in the input and finishes with the complete realization of the required data, usually the particle data at the given expansion factors.

2.6.1 `grafic`: the Initial Conditions Generator

This command implements an extension of the `grafic1` initial conditions generator originally presented as an *Fortran77* code by *E. Bertschinger*, See Section 3.8 [Serial Fortran N-body Codes], page 79. Section Section 2.10.1 [Comparing `gracos` with Fortran `p3m` and `grafic1`], page 58 provides us with the test scripts that should be used to verify the compatibility of the result of the `gracos` command with the output of original *Fortran* codes. Any extensions do not currently have the serial Fortran analogue.

The following `gracos` variables must be set before this command is called

`nx,ny,nz,dx`

Mandatory

`epsilon` Unless the option `-nodom` is used

`omegam, omegav` and `h0`

Unless the option `-lingerdat` is used

`a` If this variable is set, it defines the starting expansion factor of the universe. Either set this variable or use the `-sigstart` option described below; error message terminates the run when attempting to use both ways.

The `grafic` command accepts the following options

-icase=int32

Select the type of initial power spectrum (matter transfer function $T(k)$). The Integer takes the following values:

- 0 Use tabulated values for the power spectrum. See the **-table** option.
- 1 Use the transfer command tabulated in a 'linger.dat' file. See the **-lingerdat** option. Set the **omegam**, **omegav** and **h0** code variables from the 'linger.dat' file header.
- 2 Use the BBKS analytical approximation for the $T(k)$.
- 3 Use $T(k) = 1$ (the scale-free case).

-pnorm=float32

Mandatory option. A floating point number, sets the desired normalization at $a = 1$:

for icase = 0

Mandatory option. **pnorm** equals **-sigma_8** at the present epoch (only negative **pnorm** is currently allowed).

for icase = 1,2

if **pnorm** > 0

pnorm equals $Q_{rms-ps}/micro-K$

if **pnorm** < 0

pnorm equals **-sigma_8** at the present epoch.

for icase = 3

if **pnorm** > 0

pnorm equals $k^3 * P(k)$, where $P(k)$ is an unfiltered value, e.g. with the Gaussian or Hanning filters, of the initial power spectrum, and $k = \pi/dx$ (or $k = \pi/rsmoo$, if the **-rsmoo** option (down the list) is set and $rsmoo > dx$).

if **pnorm** < 0

pnorm equals **-sigma_8** at the present epoch.

-lingerdat

Provide the name of the lingerdat file (for **-icase** value 1 only).

-enable-baryons

GRACOS currently does not have the capability for gas N-body simulations. By using this option, however, one generates the velocities for baryonic matter, just as they would be generated with the original *grafic1* code, See Section 3.8 [Serial Fortran N-body Codes], page 79; this data, however is not used for CDM or any other particle initializations. Using this option in conjunction with **-ic-vel-files** allows one to save baryonic data into files and analyze it.

`-offvelc=int32`

`-offvelb=int32`

Integer parameters for offsetting baryon and CDM velocity fields by 0.5 grid spacing. The argument should be +/-1 or 0; the default value is +1, just as the default in the *grafic1* code. The origin of the baryon velocity field will be $\text{offvelb}*(0.5,0.5,0.5)*dx$ and similarly for `offvelc`.

`-sigstart=float32`

This option is mandatory unless the `a` code variable is set before calling `grafic`. `sigstart` sets the initial linear density fluctuation amplitude on the ultimate sub grid scale (or the approximately starting expansion factor). You must either set this option or preset the `a` code variable before calling `grafic`.

If the power spectrum is sufficiently steep so that $k^2 P(k)$ converges slowly at low wavenumbers (consider for example the scale-free free power spectrum with $ns=-3$) the internal romberg integrator fails; use the `-trlowk` option to avoid this problem.

`-ns=float32`

Long-wave spectral index (scale-invariant is $ns = 1.$), a floating point number; defines the shape of the primordial power spectrum. This option is mandatory when `-fn1` is used or `-table` is used with `choice:transfer_function`.

`-seed=uns32`

Mandatory option. The random number `seed`, - a positive integer smaller than $2^{32-5}=4294967291$.

`-power-plot=kmin:kmax`

Use this option to make a plot of the power spectrum used to generate the initial conditions. Floating point numbers `kmin` and `kmax` define the range of the wave vectors (in the units of inverse comoving Megaparsecs) covered by the plot. The text file `power.dat` is produced in the working directory as the initial conditions are generated. The first row in this file shows `ns` and `pnorm`; the remaining rows are

$$k/h, \quad P(k)*h^3, \quad \text{and} \quad 4*\pi*P(k)*k^3$$

where `h` equals `h0/100`; `k` is the wave vector in the units of inverse comoving Megaparsecs; and `P(k)` is the power spectrum in comoving Megaparsecs cubed.

In addition to the data we described each rows may contain additional data appended at the end of each line. If this is the case, the specification for the additional data is outside the scope of this reference.

`-unscalable`

Using this option gives the advantage that the particle realization does not depend on the number of processes used while running `gracos`. Thus one is able to completely replicate the initial particle data realization of the original *grafic-2_101* code, See Section 3.8 [Serial Fortran N-body Codes], page 79, See Section 2.10.1 [Comparing `gracos` with Fortran `p3m` and `grafic1`], page 58. The expense is somewhat slower performance because the last process has to replay the complete sequence of random numbers almost for the entire box in order to arrive to its starting position in the sequence of random numbers.

Random number generators are very fast, and we would generally recommend using this option for any runs smaller than 800^3 particles.

-ic-vel-files

Generate files `ic_deltab`, `ic_velbx`, `ic_velby`, `ic_velbz`, `ic_deltac`, `ic_velcx`, `ic_velcy`, `ic_velcz`; replicating the output of the original `grafic1` code. If `gracos` is ran in parallel, then for each of the `file` of the above list, a set of files `file-rank` are generated instead, where `rank` goes from 0 to `Nproc-1`. This option may be useful if you are familiar with the traditional `grafic1` output and, perhaps, have some data analysis routines that take these file as the input. It may as well be useful as a way to extract the generated baryonic initial conditions in case you have chose option `-enable-baryons`. Note if you are using the `-fnl` or `-glass` extensions (See the following table): their effect does not extend to the generated `ic_*` files.

-disable-domain-decomposition

After having completely computed the overdensities, and velocities while calling the `grafic` command, `gracos` by default uses this information to completely initialize the particle data array and perform the parallel volume domain decomposition, sending particles between the processes; the latter procedure results in the complete setup, such that the commands `pmpower` or `integ` may be used, for example without much additional work. The domain decomposition may take a considerable amount of time for very large problem sizes, but is necessary for the followup with most of the other `gracos` commands. Using the `-disable-domain-decomposition` option will disable particle domain decomposition.

This option combined with the `-ic-vel-files` and `-unscalable` for `icase>0` will just result in the very close replication of the result as computed by the serial `grafic1` code, See Section 3.8 [Serial Fortran N-body Codes], page 79.

If only the above options are used (excluding the case `icase = 0`) the produced result will be identical to the roundoff error to the equivalent serial run with the `grafic-2_101` code, See Section 3.8 [Serial Fortran N-body Codes], page 79. The following list adds important optimization options and a few new options compared to those provided by the original `grafic-2_101` code.

-table=path:string,choice:string

This option is applicable for the case `icase=0` only. Both `path` and `choice` suboptions are mandatory. `path` gives the name of the two column ASCII file, whose first column is the wave vector expressed in inverse comoving Megaparsecs. The interpretation of the second column depends on the other suboption. If `choice` equals “`power_spectrum`” then the second column gives the primary power spectrum to be used to setup the initial conditions; if `choice` instead equals “`transfer_function`” then the second column yields the transfer function. The normalization of the data in the second column is irrelevant as the power spectrum is normalized intrinsically.

The rows in file `path` can be given in an arbitrary order. However, the wavevector values given in the table should cover the complete wavevector range used in the code, since no automatic extrapolation in the wavevector space is performed. A table with a limited wavevector range should be manually appended with one or two entries to extend the wavevector range as needed at both high and low wavevector values.

-disable-hanning

Disable the use of Hanning filter, particular to the `grafic1` code.

-fnl=float32

Apply non-Gaussianity correction to the generated initial conditions, `fnl` is the second order correction to the primordial proper potential ϕ . The motivation for the `fnl` model of initial conditions, pioneered, among others, by *E. Komatsu* is to explore the possibility that the initial conditions in our universe are slightly non-gaussian, in which the true initial conditions potential ϕ can be expanded in terms of the gaussian potential ϕ_0 as $\phi = \phi_0 + f_{nl}(\phi^2 - \langle \phi_0^2 \rangle)$ when expressed in the units of the speed of light squared. The `-ns` option to `grafic` must also be set.

-rsmoo=float32

Apply the gaussian filter; the power spectrum used for the generation of the initial conditions is multiplied by $\exp(-k^2 \text{rsmoo}^2)$; `rsmoo` is expressed in comoving Mpc.

-trlowk Limit the integration to the frequencies above $\pi/(\text{dx}*\text{nx})$, while computing power spectrum normalization to find starting expansion factor using `-sigstart` option; see the description above for the `-sigstart` option.

-glass=seed:uns32,nexp:float

Generate glass initial conditions, instead of those on the regular grid. The `seed` suboption is mandatory and is used to initialize the initial poisson distribution, necessary for making the glass particle distribution. The `nexp` suboption is the number of expansion factors to evolve the poisson distribution with negative gravity requested to arrive to the target glass distribution. The quality of the latter improves with the increase `nexp`; if no option is given, this number is automatically initialized to the theoretical estimate $nmax^1/2$, where `nmax` equals the maximum of `nx`, `ny` and `nz`; even though the theoretically estimated numerical value of `nexp` is rather large, the nbody simulation will proceed rapidly under the repulsive gravity. One may run the power spectrum tests like those shown in Section 2.6.2.3 [Example: Making the Glass], page 51 in order to show that the generated glass is reliable.

2.6.1.1 Example: Tabulated Transfer Function

In this section, we show how to start an arbitrarily large simulation starting from the values of cosmological parameters and tabulation for the power spectrum $P(k)$ of matter density perturbations; we present the readily executable `gracos` script below.

Let us assume that we are having a table of the power spectrum $P(k)$: the values of P_i , tabulated at the wave numbers k_i ; where P is expressed in arbitrary units (consistent for

different i) and k_i is expressed in the units of $h/(\text{Mpc comov})$, where h is the present time Hubble constant, in the units of $100 \cdot \text{km/s/Mpc}$. The power spectrum table should cover the entire range of the wave vectors used during the `gracos` run because `gracos` does not do any extrapolation of power spectrum in this case. The *GRACOS* installation provides a sample of such power spectrum table for use in this example of the `gracos-examples`, but you should use your own power spectrum table.

At the end of this subsection we provide the complete details on how we generated this power spectrum table using *CMBFAST*).

First, we define the *WMAP3* cosmological parameters (we remind that the “.” at the beginning of the line in *bash* means sourcing the path as if it were included into the script). The `gracos-config` executable is a part of *GRACOS* installation and is used to locate files provided with *GRACOS* installation; type

```
cat 'gracos-config info pkgdatadir'/CosmoParams/WMAP3
```

in your shell prompt to see the *WMAP3* parameters used.

The following line in the script defines the path to the power spectrum file that we generated using *CMBFAST*, also the part of the installation for the purpose of this reference; again, the user should generally use their own power spectrum table as the power spectrum used in our example is in not suitable for a general situation (as we see at the end of this section).

Next, we specify the random number seed `seed` used to generate the sequence of random numbers used by the initial conditions generator and the size of the particle mesh `Ngrid` the user is free to change this number. The `Ngrid` parameter is low for the purpose of this example; the user is free to change this number as needed to change the resolution of the N-body simulation (if you significantly increase this number you should also increase the number of processes for the `mpirun` command line prefix). Next, we arbitrarily set the simulation boxsize `Lbox`, expressed in comoving Megaparsec (not the Megaparsec/h!) and compute the cell spacing `dx` using `bc` (the `bc` is a simple calculator, usually provided with any standard Unix distribution).

Next, we set the Plummer force softening length `epsilon` (expressed in the units of `dx`) and the integration timestepping parameter `etate`; the `0.05` is a good value; using lower value will result in more timesteps needed to complete the simulation. Finally, we provide the comma separated list of output specifiers with `aouts` (only one in our example); each formatted as `aout[:modifier]`, where `aouts` is the output expansion factor and the modifier is a character specifying what to do at the given expansion factor, See Section 2.8 [integ: Running an N-body simulation], page 54 for more explanations.

Now that we have defined all the simulation parameters, we begin writing the body of the `gracos` script into a temporary file whose name is generated with a standard `mktemp` Unix command.

First we set the necessary simulation parameters for `gracos` with a set of `varset` commands, See Section 2.3.3 [Variables], page 7, Section 2.3.3.1 [Example: Manipulating `gracos` Variables], page 8 and Section 2.11.2 [`gracos` Variables], page 65. The list of all the necessary parameters for this run can easily be found by the trial and error method (this is indeed how we knew which parameters are needed to be set): `gracos` terminates with a useful error message any time a required variable is uninitialized.

Next, we generate the particle initial conditions by calling the `grafic` command using the `icase=0` selection, See Section 2.6.1 [`grafic`: the Initial Conditions Generator], page 33. The arguments of this command completely specify all the additional information needed to generate the particle realization of the given power spectrum. The `-power-plot` flag provides the option to output the power spectrum plot in the given range of wave vectors.

Next, we call `image` to generate an image of the simulation box, See Section 2.5 [`image`: Particle Data Imaging], page 28. The `-set-defaults` flag automatically makes a good selection for all the image parameters. The result is a new *GIF* file `'i.gif'` in the current working directory.

Next, using `dataman` (See Section 2.4 [File Input and Output with `dataman`], page 16) we save the initial conditions data into file using the distributed file input-output, See Section 2.4.3 [`udf`: Unified Datafile Format], page 19; call `integ` to start an N-body simulation, make an image using `image`, and call `dataman` again to save the data in the evolved state.

Finally, we make another image of the simulation box, producing file `'f.gif'`.

The entire `gracos` session is written into a temporary file `$TmpFile`, whose name is passed as the argument to the `-i` option to `gracos`; `gracos` is ran in parallel with `mpirun` command on four processes.

The user is free to increase the problem size by increasing `Ngrid` and the number of processes, up to the system resource limitation.

```
#!/bin/bash
# File: gracos-examples/problems/pktable.sh
# Author: Alexander V. Shirokov

# Define cosmological parameters: H0,Om,Ob,Ov,Oc,ns,sig8,taulss
. 'gracos-config info pkgdatadir'/CosmoParams/WMAP3

# Produce the transfer function table using
# the Eisenstein and Hu fitting formula
table=table.dat
gracos-ehu \
  --hubble='echo "$H0/100." | bc -l' \
  --krange=1.E-5:300:100. \
  --omega_baryon=$Ob \
  --omega_lambda=$Ov \
  --omega_matter=$Om \
  --redshift=0 \
  > $table
# Extend the wavevector range covered by the table manually
# with one k_low, T(k_low) pair, borrowing the value of the transfer
# function from the first entry of the already existing table
# (since T=const at low k).
table_ext=table-ext.dat
awk 'NR == 1 {print 1.0E-50, $2}' $table > $table_ext
cat $table >> $table_ext
```

```

# Set simulation Parameters for GRACOS
# Random number seed
seed=123456789
# Number of particles and PM-grid points per one side
Ngrid=32
# Simulation boxsize, comoving Megaparsec
Lbox=200.
# Grid spacing
dx=$(echo "$Lbox/$Ngrid" | bc -l)

# Integration timestep parameter
etat=0.05
# Plummer force softening length (in units of dx)
epsilon=0.1

# GRACOS Session
TmpFile='mktemp ./tmp.XXXXXXXXXX'
cat > $TmpFile <<EOF

varset epsilon $epsilon

varset nx $Ngrid
varset ny $Ngrid
varset nz $Ngrid
varset omegam $Om
varset omegav $Ov
varset h0 $H0
varset dx $dx

grafic -icase=0 \
  -table=path:$table_ext,choice:transfer_function \
  -offvelc=1 -ns=$ns \
  -sigstart=0.2 \
  -pnorm=-$sig8 -seed=$seed \
  -power-plot=1E-5:10

# Measure the power spectrum from the box.
pmpower -nbins=100

# Make an image
image --set-defaults -path=ini.gif

# Integration parameter
varset etat $etat

# Measure the initial conditions power spectrum

```

```
pmpower -path=pmpower-ini.dat

# Optionally pass extra commands from users SHELL environment
$gracos_extra_commands

# Saving data using the Unified Data Format
dataman -mode=save -fmtid=udf -path=ini.udf-

# Start the N-body simulation
# The -aouts argument is a comma-separated list of output specifiers
integ -aouts=1.:n

# Make an image
image --set-defaults -path=fin.gif

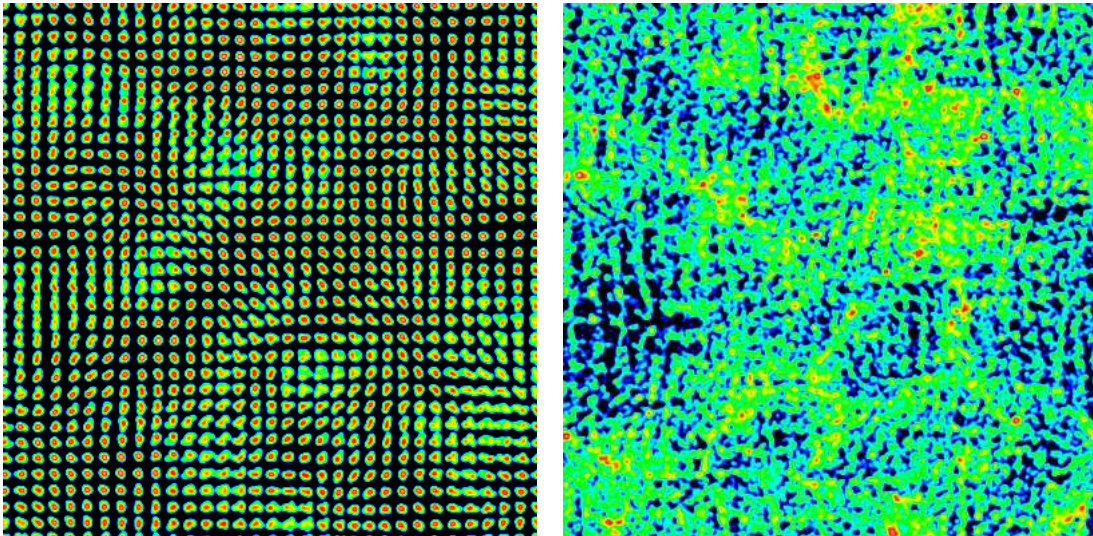
# Measure the evolved state power spectrum
pmpower -path=pmpower-fin.dat

# Again, saving data using the Unified Data Format
dataman -mode=save -fmtid=udf -path=fin.udf-
EOF
mpirun -np 4 gracos -i $TmpFile
rm $TmpFile
```

The Images

Shown below are the images in files `ini.gif` and `fin.gif` (the latter is at redshift zero) produced by the `image` command just before and after the particles are evolved with the `integ GRACOS` command as shown in the script above. If we makes a sequence of these images as we evolve the box we can actually observe that the large scale structure is smoothly transformed from that of shown on the left panel to that on the right (please See Section 2.3.5

[Runtime Administration], page 11 for the available options on executing arbitrary commands, such as `image` for making images, in the runtime).



An Example of Run Time Administration

It is generally possible to make the images on the fly, using the technique described in See Section 2.3.5 [Runtime Administration], page 11. Here we demonstrate an example of this. By copying the example below into file `inc.tcl` within the working directory *during the run*, we execute the imaging command (see the example) as soon as the runtime control reaches the `integ` checkpoint; the current images are then created and this file is automatically erased. We do not show the image generated; because they apparently depend on the exact time at which the `inc.tcl` file is created. You should try this copy command while running the above script.

```
# File: gracos-examples/problems/admin-image.tcl

if { $admin_label eq "integ" } {
    # Make the images as soon as the 'integ' checkpoint
    # is reached
    # This file is deleted because 'admin_keep' is unset.
    image -set-defaults
} else {
    # Keep this file in case of other checkpoints
    set admin_keep 1
}
```

The Power Spectrum

Now, typing

```
pmpower('pmpower-fin.dat')
```

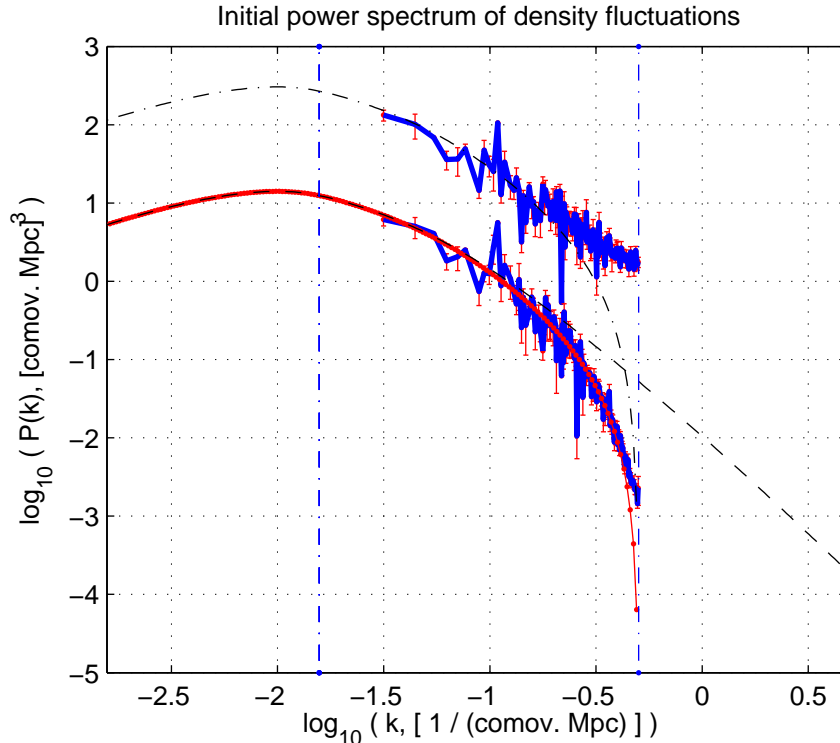
```
pmpower('pmpower-ini.dat')
icpower
```

in *MATLAB* in the same directory yields the plot below of the power spectrum of density perturbations as a function of the wave number, spanning the range of $[2\pi/(\mathbf{n}\mathbf{x} \cdot \mathbf{d}\mathbf{x}), \pi/\mathbf{d}\mathbf{x}]$ as shown by the two vertical dash-dotted green lines.

The red line at the bottom is the power spectrum used for the generation of particle initial conditions. The oscillations of this curve above the Nyquist frequency are caused by the use of the Hanning filter and should not worry the reader, as they are outside the wavevector range used. Passing the `-disable-hanning` option to `grafic` disables the Hanning filter; the black dashed line shows the unfiltered power spectrum.

The lower broken blue line is the measurement of the power spectrum based on the unevolved particle positions (See the first `pmpower` call in the *GRACOS* script above). The deviations from the red curve are due to the following effects: 1) Cosmic variance at low wavenumbers; 2) the effect of the randomness of the current particle realization given the power spectrum; 3) coarse graining near the Nyquist frequency.

The upper blue broken line is the power spectrum of matter perturbations in the evolved state (the second `pmpower` call) and the black dash-dotted line shows the initial power spectrum given by the red line corrected multiplied by the linear growth rate characteristic for the expansion from the starting expansion factor to redshift zero, the final point of our simulation. The systematic deviation of the black dash dotted line with the upper blue broken line is due to the nonlinear evolution.



2.6.1.2 Example: lingers Initial conditions

In this section we show how to start an arbitrarily large simulation of CDM with `gracos` directly, starting from the values of the cosmological parameters using the included `lingers` package, a subpackage by *C. Ma, E. Bertschinger, 1995* included into *GRACOS*, See Section 3.8 [Serial Fortran N-body Codes], page 79. `lingers` is automatically installed during the Section 4.3 [GRACOS Package Installation], page 83. The transfer function tabulated in both redshift and k-space is included into the `linger.dat` file automatically generated using `lingers`.

As we know, the baryonic, CDM and matter transfer functions agree to within the percent level close to $z = 0$. For the power spectrum normalization we use σ_8 of the power spectrum of all matter linearly evolved to $z = 0$. σ_8 is measured as an integral with respect to k over the power spectrum at a given a . Usually it is assumed that the power spectrum grows as D_+ . With `lingers` however this dependence is intrinsic and is not factored out.

If we plot the power spectrum at $a=0.02$ (normalized at $z=0$), then we go back by the intrinsic growth factor and by the D_+ in the two cases. In the result we get very good agreement between the curves. The `ehu` curve goes right through the middle of the matter power spectrum oscillations. But the CDM particle displacements are generated with the CDM power spectrum, not matter. The CDM and MATTER power spectra, while almost matching exactly each other at $z=0$, are very far apart at $a=0.02$ (about 12% or $d\log_{10} = 0.05$).

Now, if we assume that $\Omega_m = \Omega_c + \Omega_b$, we imply that the power spectra of Ω_c and Ω_b exactly match each other at $a=0.02$. The correct thing to do would be to generate the displacements using the matter power spectrum at a_{start} .

The evolution factor of CDM is different than that for MATTER.

```
#!/bin/bash
# File: gracos-examples/problems/lingers.sh
# Author: Alexander V. Shirokov

# Define cosmological parameters: H0,Om,Ob,0v,0c,ns,sig8,taulss
. 'gracos-config info pkgdatadir'/CosmoParams/WMAP3

# Run lingers from the included grafic1 package to generate transfer
# function and store it in 'linger.dat' file.
tcmb=2.726; yhe=0.24
kmin=1.E-5; kmax=10.; nk=121
zend=0.; case=1
TmpFile='mktemp ./tmp.XXXXXXXXXX'
cat > $TmpFile <<EOF
$Ob $Oc $Ov 0
$H0 $tcmb $yhe 0
$kmin $kmax $nk $zend
$case
EOF
cat $TmpFile | lingers
```

```

# Set the simulation Parameters for GRACOS
# Random number seed
seed=123456789
# Number of particles and PM-grid points per one side
Ngrid=32
# Simulation boxsize, comoving Megaparsec
Lbox=200.
# Grid spacing
dx=$(echo "$Lbox/$Ngrid" | bc -l)

# Integration timestepping parameter
etat=0.05
# Plummer force softening length (in units of dx)
epsilon=0.1

# GRACOS Session
cat > $TmpFile <<EOF

varset epsilon $epsilon
varset dx $dx
varset nx $Ngrid
varset ny $Ngrid
varset nz $Ngrid

# Generate Initial Conditions with grafic
grafic -icase=1 \
-offvelc=1 \
-sigstart=0.2 -ns=$ns -lingerdat=linger.dat \
-pnorm=-$sig8 -seed=$seed \
-power-plot=$kmin:$kmax \
$grafic_flags_extras

# Measure the power spectrum; display with pmpower MATLAB command.
pmpower

# Make an image
image --set-defaults -path=ini.gif

# Optionally pass extra commands from users SHELL environment
$gracos_extra_commands

# Integration parameter
varset etat $etat

# Saving data using the Unified Data Format
dataman -mode=save -fmtid=udf -path=ini.udf-

```

```

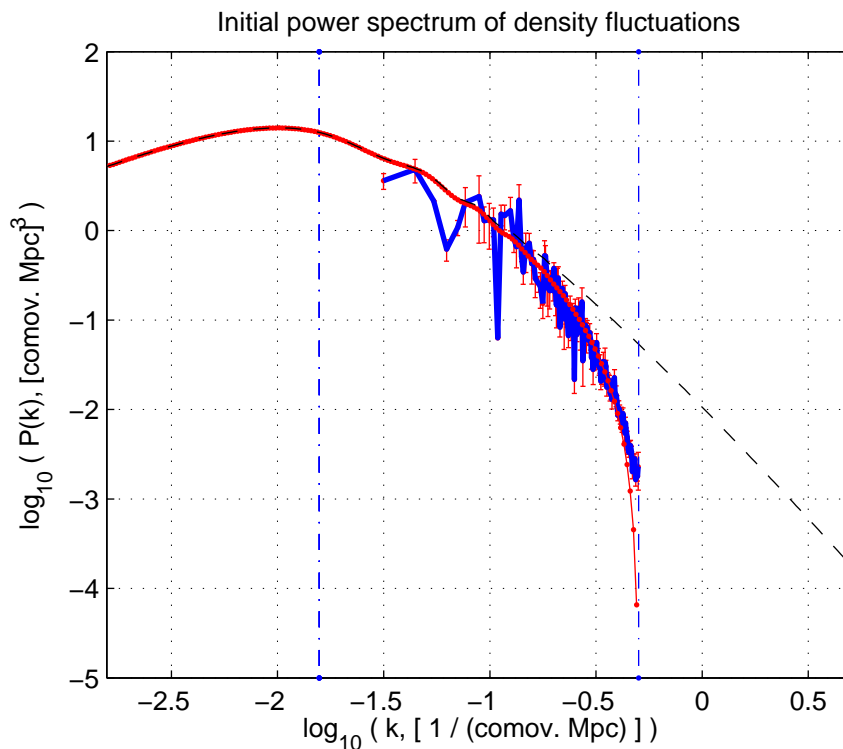
# Start the N-body simulation
# The -aouts argument is a comma-separated list of output specifiers
integ -aouts=1.:n

# Make an image
image --set-defaults -path=fin.gif

# Again, saving data using the Unified Data Format
dataman -mode=save -fmtid=udf -path=fin.udf-
EOF
mpirun -np 8 gracos -i $TmpFile
rm $TmpFile

```

Typing `icpower` and `pmpower` now, in *MATLAB*, within the working directory yields the power spectrum plot. The resulting plot is distinguishable from the plot in the previous section, by the presence of the baryonic oscillations.



2.6.1.3 Example: BBKS Initial Conditions

An analytical approximation to the initial power spectrum by Bardeen, Bond, Kaiser and Szalay named thus the *BBKS approximation* is directly applicable in `gracos`.

File `gracos-examples/problems/bbks.sh` In this section we show how to start an arbitrarily large cosmological N-body simulation starting with values for a few cosmological

and simulation parameters, and assuming that the initial power spectrum is described by the BBKS approximation.

The user should be familiar with the content of the script shown below from the discussion in the two previous sections. The script below does not make calls to executables rather than `gracos`: using the BBKS approximation is reserved for `icase=2`:

Typing `icpower` in *MATLAB* within the working directory yields the power spectrum plot below. As we see by comparison with the power spectrum plot generated with *CMB-FAST* and *lingers*, the *BBKS* approximation is slightly underestimates power for lower wave vectors, while slightly exaggerating for larger k 's.

2.6.1.4 Example: Scale-Free Initial Conditions

Sometimes it is useful to start a simulation using scale-free initial conditions with $P(k)$ proportional to k^{-n_s} .

This section shows how, given a few required parameters, to start an arbitrarily large simulation with scale-free power spectrum of matter perturbations. The `icase=3` option to `grafic` is dedicated to scale free initial conditions. The script below is very similar to the one in the previous section, which, similarly to the present case, uses an analytical expression for the initial power spectrum of CDM density perturbations.

```
#!/bin/bash
# File: gracos-examples/problems/scalefree.sh
# Author: Alexander V. Shirokov

# Define cosmological parameters: H0,Om,Ob,Ov,Oc,ns,sig8,taulss
. 'gracos-config info pkgdatadir'/CosmoParams/WMAP3

# Set simulation Parameters for GRACOS
# Random number seed
seed=123456789
# Number of particles and PM-grid points per one side
Ngrid=32
# Simulation boxsize, comoving Megaparsec
Lbox=200.
# Grid spacing
dx=$(echo "$Lbox/$Ngrid" | bc -l)

# Integration timestep parameter
etat=0.05
# Plummer force softening length (in units of dx)
epsilon=0.1

# GRACOS Session
TmpFile='mktmp ./tmp.XXXXXXXXXX'
cat > $TmpFile <<EOF

varset epsilon $epsilon
```

```

varset nx $Ngrid
varset ny $Ngrid
varset nz $Ngrid
varset omegam $Om
varset omegav $Ov
varset h0 $H0
varset dx $dx

grafic -icase=3 \
  -seed=$seed \
  -offvelc=1 \
  -sigstart=0.2 -ns=$ns -pnorm=-$sig8 \
  -power-plot=1.E-5:10 \
  $grafic_flags_extras

# Make an image
image --set-defaults -path=ini.gif

# Integration parameter
varset etat $etat

# Saving data using the Unified Data Format
dataman -mode=save -fmtid=udf -path=ini.udf-

# Start the N-body simulation
# The -aouts argument is a comma-separated list of output specifiers
integ -aouts=1.:n

# Make an image
image --set-defaults -path=fin.gif

# Again, saving data using the Unified Data Format
dataman -mode=save -fmtid=udf -path=fin.udf-
EOF
mpirun -np 4 gracos -i $TmpFile
rm $TmpFile

```

2.6.1.5 Example: Glass Extension

2.6.2 poisson: Poisson Distribution

This routine may be a good starting point if you want to write your own initial conditions generator that can not be expressed as an extension of `grafic`.

This command generates N Poisson-distributed particles across the box. The syntax is

```
poisson -seed=seed
```

where *seed* is a random number seed.

Variables `nx`, `ny` and `nz` must be set. If `npartall` code variable is unset, then the total number of generated particles along with the `npartall` code variable are set to `nx*ny*nz`; otherwise N equals `npartall`.

The internal procedure is as follows. The volume domain decomposition is performed so that approximately equal volumes are assigned to each process. The Poisson random number generator is called on each process with exactly the same random number seed so that each process scans the identical sequence of randomly distributed particles. If a particle belongs to a given process, that process assigns that particle to itself; the entire sequence of particles is thus assigned to all N_{proc} processes.

The Poisson random number generation is relatively fast. However, the procedure described above, is not scalable. In principle, by generating particles with varying random number seed on each process, one can make this procedure scalable: each process would try to assign each particle in the sequence by a certain mapping from the uniform distribution within simulation volume into the irregular volume particular to each process. The Jacobian of the mapping must equal to one; we are not certain at this point however on how to setup such a mapping, suggestions are welcome however. Hence, we are currently employing the above non-scalable procedure which is safe against any tiling effects.

2.6.2.1 Example: Poisson Distribution

In the example below we generate the power spectrum of the Poisson particle distribution (See Section 2.6.2 [poisson: Poisson Distribution], page 48) and compare it with the analytic slope (zero).

```
#!/bin/bash
# File: gracos-examples/tests/powersp/poisson.sh
# Author: Alexander V. Shirokov

Ngrid=128

# GRACOS Session
TmpFile='mktmp ./tmp.XXXXXXXXXX'
cat > $TmpFile <<EOF

# Gridsize and particle number
varset nz $Ngrid
varset ny $Ngrid
varset nx $Ngrid

# Needed for domain decomposition
varset epsilon 0.1

varset dx 1.0

# Generate Poisson particle distribution
poisson -seed=123456789

# Measure the power spectrum. Use pmpower MATLAB command to visualize.
```

```

pmpower

dataman -mode=save -fmtid=binpart -spec=xs,ys,zs -path=poisson.bin

EOF

mpirun -np 4 gracos -i $TmpFile
rm $TmpFile

```

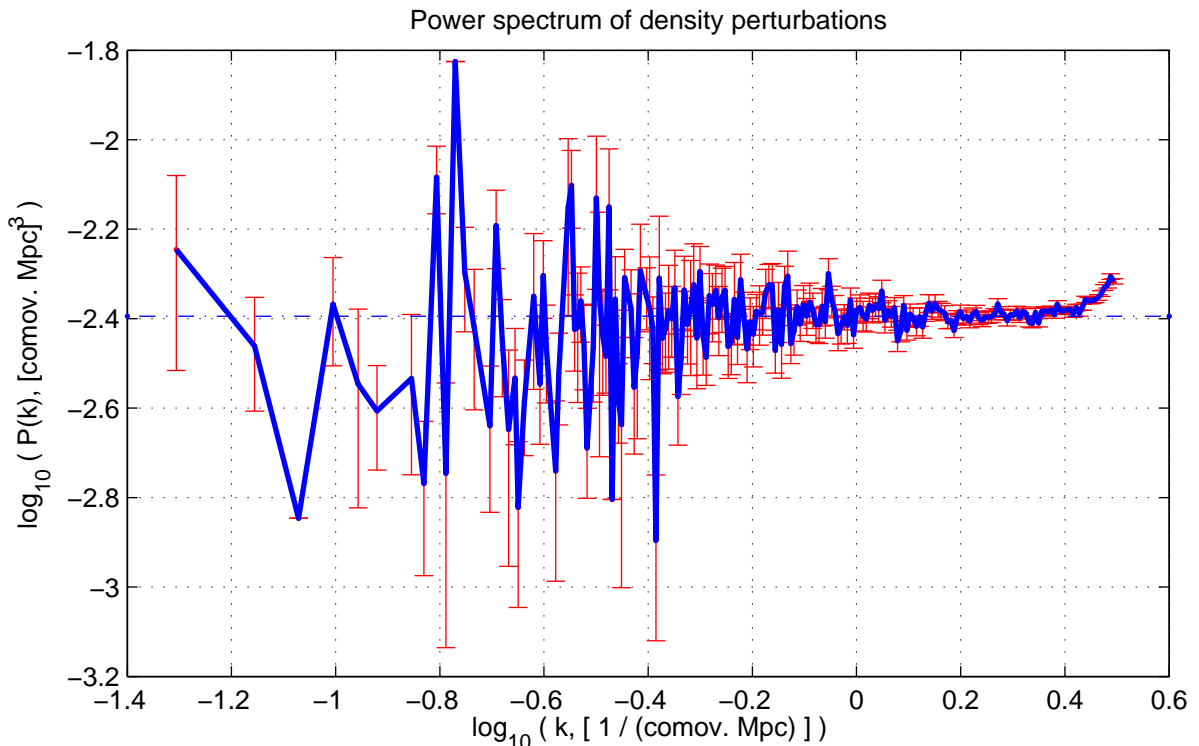
Typing

```

pmpower
yline( log10(1/(2*pi)^3) )

```

in *MATLAB* yields the image below. The theoretical power spectrum of the Poisson distribution is the constant $1/(2\pi)^3$, shown with the horizontal slash-dotted line; this prediction is mostly observed in the plot generated. The large error bars on large scales (low wavevectors) are due to cosmic variance or low statistical sample of these modes; while at the very high wavenumbers there is a characteristic upturn, whose nature is not entirely obvious. The plot shows the bounds of the validity of the power spectrum generated with `pmpower`.



2.6.2.2 Example: generating the glass Distribution

This command generates N Glass-distributed particles across the box. The syntax is

```
glass -seed=seed -nexp=nexp
```

where *seed* is a random number seed and *nexp* is the number of expansion factors requested to evolve glass. In Section Section 2.6.2.3 [Example: Making the Glass], page 51 we provide a good example of using this command.

In *GRACOS*, the procedure for making glass distribution is defined as follows: first we distribute particles uniformly across the box, using the Poisson distribution; then we evolve the particles with negative gravity. By measuring the power spectrum, we find that the power spectrum, initially dominated by the shot noise, settles down to form the single power law power spectrum characteristic for glass $P(k)$.

2.6.2.3 Example: Making the Glass

Glass distribution for particles, introduced by *S. White*, is similar to the distribution of molecules in a liquid, (we know from the statistical mechanics background that the glass material is liquid). *GRACOS* provides the utility for generating glass distribution and glass initial conditions, See Section 2.6.2.2 [Example: generating the glass Distribution], page 50 for more details.

As a good test of *GRACOS* one can run a simulation to produce the glass structure and measure the power spectrum during the inversed gravity run. Shown below, as usual is the *GRACOS* script that does the job. This is also, the first script in this reference that makes use of the runtime administration technique to measure the power spectrum on the fly (See further below for more details).

```
#!/bin/bash
# File: gracos-examples/tests/powersp/glass/generate.sh
# Author: Alexander V. Shirokov

TmpFile='mktmp ./tmp.XXXXXXXXXX'
cat > $TmpFile <<EOF

# Set simulation parameters
varset nz 32
varset ny 32
varset nx 32
varset omegam 1.0
varset omegav 0.0
varset dx 1.
varset a 0.00001
varset epsilon 0.1
varset etat 0.05

# Request a repeated runtime administration
# to measure power spectrum on the fly.
varset admin_path generate-admin.py

# Generate glass particle distribution by expanding the
# uniform poisson initial particle distribution with inverse gravity
# for the 1.E15 expansion factors.
glass -seed=123456789 -nexp=1.E15

# Invoke an additional instance of runtime administration explicitly
admin
```

```

# Save the data into one file in the xs,ys,zs binary format
dataman -mode=save -fmtid=binpart -spec=xs,ys,zs -path=glass.bin
EOF
mpirun -np 4 gracos -i $TmpFile
rm $TmpFile

```

The above script as is will just generate the assumed glass particle distribution. But in order to test whether a reliable glass distribution is indeed generated we must measure the power spectrum and compare it with the power spectrum of glass distribution which is shown theoretically to be a fourth power law of the wave number k . We can just place the `pmpower` command at the end of the above script. But it would be useful to see how the power spectrum really evolves as we evolve particles with negative gravity for this purpose we make use of the repeated runtime administration method described in Section Section 2.3.5.2 [Repeatable Administration], page 12. Shown below is the *Python* runtime administration file that should be placed into the same directory as the script above. The runtime administration script conditionally invokes a set of *GRACOS* commands to execute in a *GRACOS* subsession. The script mostly consists of the useful comments but is essentially just a few lines long.

```

#!/usr/bin/python
# File: gracos-examples/tests/powersp/glass/generate-admin.py
# Author: Alexander V. Shirokov

# Runtime administration file to measure the power spectrum
# at the specified timesteps and on any explicit administration call.

# First, restrict the administration to the specified checkpoints

if admin_label == 'integ_pre' or admin_label == 'integ' or admin_label == 'admin':

    # The condition to measure the power spectrum:
    # Specify the timesteps at which to measure the power spectrum
    # 1. Measure the power spectrum at the specified timesteps
    # 2. Measure the power spectrum each 'admin' administration checkpoint
    # independently of the timestep

    specific_nsteps = [0, 22, 62, 116, 149, 194]
    tag = None
    # Check to see if the current administration checkpoint is labeled 'admin'
    if admin_label == 'admin':
        tag = "fin"
    # Check to see if the current timestep is one of the specified in the list
    elif nstep in specific_nsteps:
        tag = "%d" % nstep

    # Conditionally measure the power spectrum. The admin_out variable,
    # when set in this script, yields the GRACOS script to execute.

```

```

if tag:
    admin_out = [
        '# :::::: Instance of Runtime Administration at timestep %d"' % nstep
        , 'pmpower --path=glass-%s.dat' % ( tag )
        , '# :::::: Runtime administration finished'
    ]

```

Using the Python's flexibility it is quite easy to make any other selection in the above script, say, for example, to make power spectrum measurements each specified number of timesteps, make images of the particle distribution, save the data into files, or any other operation allowed by the set of the *GRACOS* commands. One can modify the content of this file *during the run* to change the administration mode as needed. Using runtime administration thus yields a very powerful method of control.

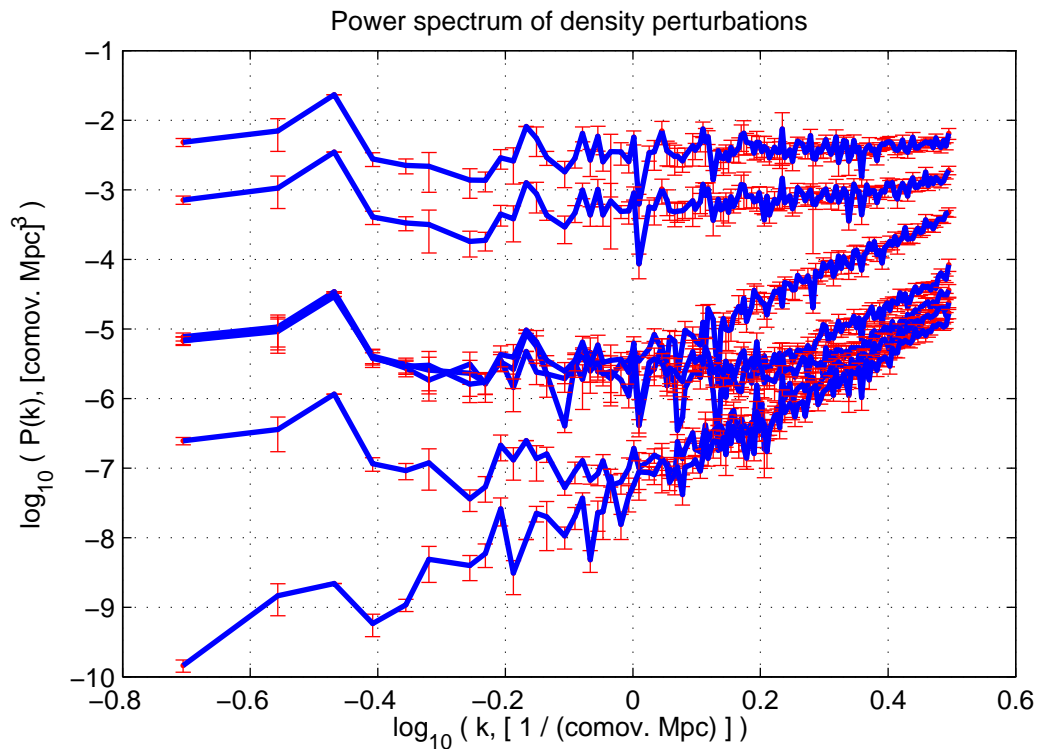
Now, if you run the `bash` shell script at the beginning of this section, and issue the following commands in *MATLAB* (in the same directory)

```

>> pmpower('glass-0.dat')
>> pmpower('glass-22.dat')
>> pmpower('glass-62.dat')
>> pmpower('glass-116.dat')
>> pmpower('glass-149.dat')
>> pmpower('glass-194.dat')
>> pmpower('glass-fin.dat')

```

you will see the following plot of the power spectra at the timesteps specified in the runtime administration script.



The upper curve on the plot just shows the white noise of the initial Poisson distribution (glass generation starts with uniformly distributed random poisson particle distribution, See Section 2.6.2.2 [Example: generating the glass Distribution], page 50 for more details). The white noise curve is in agreement with the theoretically predicted zero white noise power slope. As we evolve matter with the negative gravity to form glass distribution the matter becomes less and less chaotic as particles move under the repulsive force from each other; the power spectrum thus settles down on all scales. Since the theoretical slope of the glass distribution is approximately the fourth power of the wave vector, the modes with the highest wave number hit their theoretical glass power spectrum curve, in agreement with the behavior shown, where for the timestep 62, for example, the small scale perturbations are smoothly distributed along the theoretical curve and the largest scale perturbations are still settling down. Finally, as we evolve matter for 15 powers of the expansion factor, the matter settles down on all scales sampled by the simulation resolution. We have thus reached the accurate distribution on all scales.

2.7 pmpower: Power Spectrum Estimation

Power spectrum is a very frequently used statistics of density field with many analytical fits and expressions given in the literature and is directly related to two point correlation function.

Typing in *gracos* prompt

```
pmpower [--path=FILE ] [--nbins=NBINS]
```

yields the binned power spectrum of the currently loaded particle data written in the text file *FILE* (the default is 'pmpower.dat'); *NBINS* is the number of bins used within the range of the wavevectors of the main Brillouin zone.

The data can be visualized by typing the same command in *MATLAB*:

```
>> pmpower [(FILE)]
```

using the same default name for *FILE*.

In Section 2.4.8 [Example: Santa Barbara Cluster Project], page 24 and Section 2.6.1.1 [Example: Tabulated Transfer Function], page 37 we show an example of thus produced image of the power spectrum.

The internal procedure for computing the power spectrum is as follows. First, particle mass is interpolated on the grid using the TSC density assignment scheme; then the Fourier transform is performed. The value of the power spectrum is estimated for each grid point in the fourier space as proportional to the square of the Fourier transform of the density at the same gridpoint. The binned values of the power spectrum are found by averaging of these values for each wave vector bin, up to the Nyquist frequency. The rms deviation (the error bars) of the binned values are found by dividing the rms deviation of the measurements by the square root of the number of measurements for each bin (the Central Limit Theorem).

2.8 integ: Running an N-body simulation

The *integ* command starts the N-body simulation on the currently loaded particle data. The acceleration mode is controlled by the *acc* bitwise variable whose default value is "full acceleration": (PM | PP | FPM | FPP), See Section 2.3.4.1 [Example: Manipulating Bitwise Variables], page 10.

Timesteps are normally computed using a particular time stepping criterion during an N-body simulation; the positions and velocities of the particles are made synchronized at the certain points in time. The output expansion factors specified using the `-aouts` option do not generally coincide with the expansion factors corresponding to the particle data synchronization points; when, during the integration procedure, the next output expansion factor falls within the next timestep, the value of this timestep is normally truncated to match the output expansion factor to the roundoff error; See the `disable_timestep_truncation` sub-option.

In addition to variables needed for domain decomposition, variable `etat` must be set before calling `integ`. A reasonable value for this variable is 0.05; setting this variable to lower values can be used to increase time resolution of the simulation.

The following two options are available

`-aouts`

Mandatory option, a string containing the comma separated list of output specifiers each formatted as $A[:B]$, where A is either the floating point number setting the output expansion factor at which to produce data output or the word `INF` to indicate unlimited integration of particle trajectories. Optional column separated modifier B (the default value is `r`) may have the following mnemonic values

- `r` *Restart output*, using the expansion of variable `out_restart`, See Section 2.8.1 [Checkpoint Control Variables], page 56.
- `a` *Analysis output*, using the expansion of variable `out_analysis`, See Section 2.8.1 [Checkpoint Control Variables], page 56.
- `b` *Backup output*, using the expansion of variable `out_backup`, See Section 2.8.1 [Checkpoint Control Variables], page 56.
- `n` No output

For example, command `integ -aout=0.1:a,1:ra` indicates that the analysis output will be executed at expansion factor 0.1; both the restart and analysis outputs will be executed at expansion factor 1.

The user may define actions performed at the restart, backup and analysis outputs by setting the variables `out_restart`, `out_backup` and `out_analysis`, See Section 2.8.1 [Checkpoint Control Variables], page 56.

`-mode` Optional argument. The default value is “`-mode=scheme:DKD`”

`scheme:string`

The `scheme` specifies the time integration scheme used for particles. The allowed values are `KDK` for Kick-Drift-Kick integration scheme and `DKD` for Drift-Kick-Drift integration scheme; the default is `DKD`.

`reset_econ`

Reset the energy conservation measure (`egrav+ekin-egint`) to zero at the beginning of the run, by modifying `egint`. The status of energy conservation is normally something that accumulates from the

initial conditions produced by the particle initial conditions generator. The use of this option leads to resetting the energy conservation history to zero, so that any accumulated value during the integration run will have resulted from energy computation occurred during the current run as opposed to the cumulative value.

disable_timestep_truncation

By default, this option is disabled. The use of this option disables timestep truncation needed to match the output expansion factors set by the `-aouts` flag. For each output expansion factor the data is saved at the end of the current timestep without timestep truncation. This leads to preserving the natural, non-truncated values for all time steps in the simulation at the inconvenience of having slightly *higher* output expansion factors than those specified by the user.

break:nsteps

This option should not be normally used. The mandatory positive integer argument `nsteps` of this option gives number of time steps for the simulation, after which the integration loop is exited even if the aimed expansion factors specified by the `aout` are not reached.

mult

Use multiple timestepping scheme (this option is not currently available).

2.8.1 Checkpoint Control Variables

The expanded values of *output control variables*

`out_restart`

`out_analyze`

`out_backup`

yield the command to be executed at a checkpoint, See Section 2.8 [integ: Running an N-body simulation], page 54.

In addition to usual Tcl expansion, the following pattern expansion takes place

@MODE@ Expands to `-mode=save`, this pattern should be used in the `dataman` commands.

@OUTID@ In case of `integ`, expands to the id of the current output specifier given in `-aouts` flag, See Section 2.8 [integ: Running an N-body simulation], page 54. In all other cases expands to “outid”.

@LABEL@ expands to “r”, “a”, “b”, or “n” for restart, analysis, backup or no outputs, respectively.

2.8.2 checkpoint Command

If you wish to checkpoint at an arbitrary time, independently of the outputs specified by the `-aouts` argument to `integ`, use the `checkpoint` command; Its usage is as follows

`checkpoint -labels=LABELS`

where `LABELS` is a string containing any combination of “r”, “a”, and “b”, See Section 2.8.1 [Checkpoint Control Variables], page 56.

2.8.3 Automatic Backup Option

Variable `nchkpt` (See Section 2.11.2 [`gracos` Variables], page 65), if set, indicates the backup period, in seconds. For large integration times it may be necessary to leverage the system failure risk with this option. The timer is started at the beginning of a run with `gracos`. Once `nchkpt` seconds are passed, variable `out_backup` is expanded and its content is executed. No backups are performed if `nchkpt` variable is unset.

Given extremely large integration times, many instances of the backup are executed. To avoid unnecessary file accumulation, old backups are removed at each new backup output, at this point the `@MODE@` pattern is expanded into `-mode=delete` and `@OUTID@` expands to the value `OUTID-1`.

2.9 `halo_finder`: Halo Finder

This command is currently under development; email *GRACOS* authors to indicate your interest in this command or any suggestions you may have.

2.10 `compare_particles_serial`: Particle Data Comparison

Command `compare_particles_serial` is a serially working tool for statistical comparison of particle data sets, written using any of the formats listed in Section 2.4 [File Input and Output with `dataman`], page 16 compatible with runs on a single process.

This particle data comparison tool is not designed to handle large data volumes but is just perfect for small ones by intrinsically avoiding any complications due to the particle domain decomposition. More sophisticated tools should be used for particle data volumes exceeding 256^3 particles.

The computed statistics is based solely on the information stored within the files: no additional computations are performed. The utility accepts the following flags

`-load1=load1`

The `gracos` command (including the flags) used to load the first set of particle data.

`-load2=load2`

The `gracos` command (including the flags) used to load the second set of particle data.

`-include=include`

A string: comma separated list of tokens each corresponding to an action performed for computing the comparison statistics. The following tokens are currently accepted

`idsort` Sort particles in both sets with their particle ids.

`positions`

Output unary and comparison statistics on particle positions, periodic boundary conditions taken into account.

`velocities`

Output unary and comparison statistics on velocities.

accelerations

Output unary and comparison statistics on particle accelerations.

energies

Output the kinetic and potential energies computed for each set.

The example of using this tool is presented in Section 2.10.1 [Comparing `gracos` with Fortran `p3m` and `grafic1`], page 58.

2.10.1 Comparing `gracos` with Fortran `p3m` and `grafic1`

In this section we compare `gracos` with the original serial Fortran codes by measuring the particle and energy comparison statistics of the outputs of the initial conditions generator and N-body simulation runs for the identical small-size problems, reproduced with the two codes.

Fortran 77 p3m2 and *grafic* codes written in 1993 by Gelb and Bertschinger have served the base for parallel *gracos* C-code. A number of changes have been introduced however into the algorithm for various reasons (optimization, performance improvement, etc). Setting `origmode GRACOS` bitwise code variable to `-1` disables most of the changes of `gracos` compared to the original Fortran codes. In order to compare how well the `gracos` results reproduce the results for the same problem produced with the original Fortran codes the `origmode` variable must be set to `-1`, in which case all the deviations are due to the roundoff errors. On the other hand, it is also interesting to examine the discrepancies introduced by the `gracos` optimizations and accuracy improvement work - they should not be large in any case; the proper setting for `origmode` in this case is `0`.

Further in this section, we show how to reproduce the same problem with two different codes and compare the results. Any serious discrepancy in force computation between the two codes will be easily noticeable in the comparison statistics, even for relatively small number of particles.

We only use 32^3 particle simulations in this section. The users may change the number of particles as they wish. This requires the changes in the Fortran source code however (Fortran 77 codes can do not have dynamic memory allocation), at `src/f77/grafic-2_101/grafic1.inc` and `src/f77/p3m2-93/p3m2.inc` and the recompilations by typing `make install`; no recompilation is needed for the `gracos` code.

The `grafic-2_101` package currently offers only three choices for generating the initial conditions: *lingers*, *BBKS* and *scale-free* with a given power spectrum index; the comparison statistics is described below.

lingers Initial Conditions

Type

```
./compare-lingers.sh
```

under the `gracos-examples/tests` directory to run the full comparison tests, ignoring the prompts seen in the standard output (the full procedure should take less than 3 minutes to execute on a modern PC). By typing the above command, the following procedures are automatically executed in the order listed below:

- Executable '`./lingers.f.sh`': Running `lingers` to generate the `linger.dat` file; running `grafic-2_101` to read the `linger.dat` file and generate the CDM particle initial conditions saving them into `ic_velcx`, `ic_velcy` and `ic_velcz` files; running `ic2dat`

to transfer the `ic_velc*` files into the `ic_vel.dat` file whose format is suitable for input to `p3m2-93` code; finally running the `p3m2-93` code to generate the particle data evolved to the present time ($a = 1$, or $z = 0$) and writing it into file `'p3m.1.dat'`.

- Executable `'./lingers.sh'`: Reproducing the same procedure as the above, running `gracos` on 4 processes. The initial conditions file is written into file `'ic_vel.bin'`, and the output of particle data evolved to the present time is written into the `'p3m.1.dat.gracos'` file. The user is encouraged to
- Executable `'./compare.sh'`: Output unary and comparison statistics on the two pairs of particle data files (each file having 32^3 particles) produced in the two above procedures, using `compare_particles_serial`, See Section 2.10 [`compare_particles_serial`: Particle Data Comparison], page 57.

The output of the `./compare-lingers.sh` command indicates an excellent agreement between the outputs of the two codes, as quoted below for the initial state (all the numbers are expressed in code units, See Section 2.4.1 [Physical Units], page 18)

```
# Particle coordinates
Report: x1 N=98304 avg=16.134115 rms=18.591766 min=0.00019446253 max=31.99962
Report: x2 N=98304 avg=16.134115 rms=18.591766 min=0.00019448809 max=31.99962
Report: dx N=98304 avg=-7.6160039e-10 rms=2.3746358e-07 min=-1.9073486e-06 max=1.9073486e-06
max|x1-x2| / rms x1 = 1.025910E-07

# Particle Velocities
Report: v1 N=98304 avg=-9.9363324e-13 rms=0.024306851 min=-0.08448007 max=0.084013671
Report: v2 N=98304 avg=-1.0620486e-10 rms=0.024306845 min=-0.08448007 max=0.084013663
Report: dv N=98304 avg=-1.0521124e-10 rms=7.7444335e-09 min=-3.7252903e-08 max=3.3527613e-08
max|x1-x2| / rms x1 = 1.532609E-06
```

As we see, the root mean square particle velocity components included for all the three Cartesian coordinate system is `0.02430687` in both sets, while the maximum difference in velocities is `3.725e-08`, yielding the remarkably low ratio of `1.5326e-6` indicating the excellent agreement in the initial conditions for particle data between the two runs.

The agreement is slightly worse for the evolved state

```
# Particle coordinates
Report: x1 N=98304 avg=16.715494 rms=19.059017 min=0.00033187866 max=31.999931
Report: x2 N=98304 avg=16.715494 rms=19.059017 min=0.00032806396 max=31.999929
Report: dx N=98304 avg=-9.684312e-09 rms=5.0071503e-06 min=-7.9154968e-05 max=0.0001077652
max|x1-x2| / rms x1 = 5.654290E-06

# Particle Velocities
Report: v1 N=98304 avg=-8.3879342e-10 rms=0.90070075 min=-5.489048 max=6.3266931
Report: v2 N=98304 avg=1.0821464e-08 rms=0.9006995 min=-5.4890208 max=6.326447
Report: dv N=98304 avg=1.1660258e-08 rms=2.9376577e-05 min=-0.00091254711 max=0.00084149837
max|x1-x2| / rms x1 = 1.013152E-03
```

The evolved state comparison is still excellent however: the comparison is worse than that for the initial state, as a consequence of the Osedelec theorem, stating that any small initial state perturbation in a chaotic dynamic system grows exponentially in the subsequent evolution with the rate of divergence given by the Lyapunov exponent.

BBKS and *scale-free* Initial Conditions

Type

```
./compare-bbks.sh
and
./compare-scale-free.sh
```

under the `gracos-examples/tests` directory to run the full comparison tests. The executed procedures and the analysis are fully analogous to those presented in previous discussion on *lingers* except for the choice of the initial conditions. Each of the runs should complete in less than 2 minutes on a modern desktop.

2.11 Tables of Reference

This section provides the tables of all the available `gracos` commands, variables and modes. A short description (a *docstring*) is supplied for each entry; the detailed descriptions may be found in more specialized sections of this reference.

2.11.1 `gracos` Commands

The following table lists all the available `gracos` commands and options. The internal *Tcl* procedures and the procedures sourced in the `gracos` startup file `init.tcl` (See Section 2.3.2 [Commands], page 6) are not listed here.

`help`

```
Display Reference Tables in XML format
Location: gracos.c:295
ARGUMENTS
-o string
```

`end`

```
Request an end to the scripting session as soon as possible.
Location: cdcom.c:407
```

`inc`

```
Process commands given in the file.
Location: cdcom.c:389
ARGUMENTS
-path string
```

`admin`

```
Invoke an instance of runtime administration now
Location: gracos.c:82
ARGUMENTS
-i string
```

`env`

```
cvars , Print values of environment variables
Location: cdcom.c:505
ARGUMENTS
-xml int32
-o string
```

`varset`

Set a given variable to a given value.
Location: cdvar.c:423

varunset

Unset a variable
Location: cdvar.c:354

varputs

Print the value of a specified variable
Location: cdvar.c:570

vars

Show current values for a code variable
Location: cdvar.c:610

ARGUMENTS

-a *int32*

-v *string*

varchmod

Set write permission for a given variable.
Location: cdvar.c:395

dataman

Manage data and datafiles

Location: dataman.c:204

ARGUMENTS

-prefixes *string*

-suf *string*

-mode *string*

-fmtid *string*

-masseq *int32*

-path *string*

-spec *string*

-sort *int32*

grafic

Generate particle cosmological initial conditions

Location: grafic.c:224

ARGUMENTS

-lingerdat *string*
 -disable-hanning *int32*
 -ic-vel-files *int32*
 -unscalable *int32*
 -rsmoo *float32*
 -power-plot *string*
 -disable-domain-decomposition *int32*
 -sigstart *float32*
 -pnorm *float32*
 -trlowk *int32*
 -glass *string*
 -seed *long unsigned*
 -enable-baryons *int32*
 -fnl *float32*
 -icase *int32*
 -table *string*
 -ns *float32*
 -offvelc *int32*
 -offvelb *int32*

glass

Generate glass particle distribution

Location: glass.cc:107

ARGUMENTS

-nexp *float32*
 -seed *long unsigned*

poisson

Generate Poisson particle distribution

Location: poisson.cc:154

ARGUMENTS

-seed *long unsigned*

integ

Start an N-body simulation

Location: integ.c:320

ARGUMENTS

-aouts *string*
 -mode *string*

checkpoint

Checkpoint your simulation

Location: misc.cc:273

ARGUMENTS

-labels *string*

accel

Compute particle accelerations
Location: singular.cc:122

image

Visualization
Location: wimage.c:276
ARGUMENTS
`-path string`
`-no-image int32`
`-set-defaults int32`

pmpower

Density power spectrum estimation using, PM interpolation.
Location: pmpower.c:244
ARGUMENTS
`-path string`
`-nbins int32`

lbal

Repartitioning: Calculate current loads and adjust partitions.
Location: lbal.c:712

hc_partit_set

Set partitions to the given state.
Location: repartit.cc:1369

hc_shrink

Shrink particle data to the bottom processes
Location: repartit.cc:1508
ARGUMENTS
`-np int32`

hc_adj

Move the top partition of the specified process by a specified number of cells

Location: repartit.cc:1194

hc_adj_shift

Shift all the partitions by the given number calls along the SFC.

Location: repartit.cc:1403

hc_loadplot

Output ASCII data files with information on current workload distribution.

Location: lbal_loads.cc:727

compare_particles_serial

Analyze particle data in data files

Location: gtool.c:49

ARGUMENTS

-load1 *string*

-load2 *string*

-include *string*

sam_selectbox

Select all particles within the given box. By marking them with negative ids.

Location: sampling.c:99

ARGUMENTS

-xc *float32*

-yc *float32*

-b *float32*

-zc *float32*

sam_unselectall

Unselect particles by setting their ids to be positive.

Location: sampling.c:123

sam_traj

Update an ASCII datafile the trajectories of the selected particles

Location: sampling.c:229

an

Given the particle id, show the process it resides on and its

x,m,v,g.

Location: gtoolp.cc:101

ARGUMENTS

-id *int32*

an_idsort

Sort particles by their ids, placing them contiguously and destroying the HC structure.

Location: gtoolp.cc:148

an_compare

Compare in parallel, id sorted sets of particles in two datafiles at given paths

Location: gtoolp.cc:527

ARGUMENTS

`-path2 string`

`-path1 string`

euler_planes

Output some eulerian space analysis.

Location: sampling.c:502

ARGUMENTS

`-what string`

`-id int32`

2.11.2 `gracos` Variables

The following table lists of all the `gracos` variables for version 1.0.1a8. The description of bitwise variables also lists the available modes.

version

Version number, when last configured

Format: `string`

Initialization value: **1.0.1a8**

nproc

The number of processes

Format: `int32`

Initialized to the current number of MPI processes

tverb

Global verbosity

Format: `int32`

Initialization value: **8451**

(which is, in token representation: **prm | eval | sci | dat**)

Variable "tverb" is bitwise, the following table lists all the allowed modes.

prm Show Prompt

eval Show all Tcl commands being processed

progr Show progress updates in the position and potentially lengthy routines

timers Timer operations

mpi	Show where MPI Commands start and finish
fftw	Show where FFTW starts and finishes
alc	Memory allocation and release
paths	Show file I/O paths
sci	Show some science data
pars	Parsing results for options and suboptions
var	Variable management
mk	Selected data on all processes
mk1	Extension of mk (large, use with caution)
dat	Technical data in a human readable form
dat1	Extension of dat: arbitrary form of output
dat2	Extension of dat1 for more verbose outputs
dump	Debug data dumps (large, use with caution)

tmode

Global mode

Format: *int32*

Initialization value: **1**

(which is, in token representation: **warn**)

Variable "tmode" is bitwise, the following table lists all the allowed modes.

warn	Show warning messages
deb2	Run heavy error checking routines along the way
wraps	Enable misc. (e.g. MPI and FFTW) wrappers

tetol

Global error tolerance mode

Format: *int32*

Initialization value: **0**

Variable "tetol" is bitwise, the following table lists all the allowed modes.

var	Ignore any uses of a variable with unknown name
fun	Ignore calls unknown function calls
sys	Ignore system calls that return error

smode

`gracos-specific mode`
 Format: `int32`
 Initialization value: `0`

Variable "smode" is bitwise, the following table lists all the allowed modes.

fast Speedup, by disabling accompanying time consuming science computations
speedup Speedup by disabling some inexpensive error checks
fftw_work Enable the FFTW work buffer allocation while computing FFT's..

`origmode`

Mode to immitate the Fortran codes
 Format: `int32`
 Initialization value: `0`

Variable "origmode" is bitwise, the following table lists all the allowed modes.

old_tschm Do not use new scheme for timesteps
old_pmtab Do not use new pmtables
old_spl Do not use splines for short range forces
old_crmax Do not use new method for crmax
old_rfnt Do not select refinement number based on timing
old_lintab Do not use linear table at inner radii
only_dkd Halt, if trying to use not the original (DKD, single timestepping) integration scheme.
orig_filter Do not use A.V.S. Hanning filter use correction.
lowk_bug Do not apply low k interpolation bugfix in linger.dat input.
lingers_norm
 Do not fix lingers expansion factor normalization bug
grafic1 Do not apply various `grafic1` fixes

`admin_path`

The default name of the script to process to invoke subinterpreter.
 Format: `string`
 Not initialized

`h0`

Present value of the Hubble constant, expressed in (km/s)/Mpc
 Format: `float32`
 Not initialized

`a`

Universe expansion factor
Format: *float32*
Not initialized

omegam
Universe matter density parameter (CDM + baryonic + cold neutrinos)
Format: *float32*
Not initialized

omegav
Universe vacuum energy density parameter
Format: *float32*
Not initialized

dx
The density and force mesh cell spacing, expressed in comoving
Megaparsecs.
Format: *float32*
Not initialized

nx
Simulation volume size along the X-dimension, in PM-cells
Format: *int32*
Not initialized

ny
Simulation volume size along the Y-dimension, in PM-cells
Format: *int32*
Not initialized

nz
Simulation volume size along the Z-dimension, in PM-cells
Format: *int32*
Not initialized

fm_cf
FPM Cf-parameter
Format: *float32*
Initialization value: **15**

fm_lmax
FPM maximum alias number
Format: *int32*
Initialization value: **0**

pm_eta
PM kernel size, in code units
Format: *float32*
Initialization value: **3.29999995**

pm_lmax

Parameter lmax in PM summation
Format: *int32*
Initialization value: **2**

epsilon
Plummer force softening length.
Format: *float32*
Not initialized

ntab
The number of elements in the force tables
Format: *int32*
Initialization value: **20001**

pm_work
Set to non-zero value to enable *work array allocation for FFTW
Format: *int32*
Initialization value: **1**

acc
Acceleration mode
Format: *int32*
Initialization value: **15**
(which is, in token representation: **PM | PP | FPM | FPP**)

Variable "acc" is bitwise, the following table lists all the allowed modes.

PM	Enable PM force
PP	Enable PP force
FPM	Enable FPM force
FPP	Enable FPP force

nstep
The current timestep number
Format: *int32*
Not initialized

etat
Integration timestep parameter
Format: *float32*
Not initialized

nchkpt
Checkpoint each nchkpt seconds
Format: *int32*
Not initialized

dt_sing

The value of timestep suitable for singular timestepping scheme
Format: *float32*
Not initialized

egrav
Potential Energy, at the last synchronization point
Format: *float32*
Not initialized

egint
The integral term in the Lazer-Irvine equation
Format: *float32*
Not initialized

ekin
Kinetic energy, at the last synchronization point
Format: *float32*
Not initialized

out_restart
output flags for restart output
Format: *string*
Not initialized

out_backup
output flags for backup output
Format: *string*
Not initialized

out_analyze
output flags for analysis output
Format: *string*
Not initialized

cmd_sched
Scheduled administration command
Format: *string*
Not initialized

ufp
Particle type
Format: *string*
Initialization value: *xvi*

ufc
Saved (udf) members of chaining mesh structured.
Format: *string*
Initialization value: *r*

pa_incr

Extra margin allocation parameter
Format: *float32*
Initialization value: **1.005**

`npartall`
The total number of particles in the simulation.
Format: *long long unsigned*
Not initialized

`nploc`
Local Number of particles
Format: *unsigned*
Not initialized

`hc_rawb`
The index of the first cell in this process domain
Format: *long long unsigned*
Not initialized

`hc_rawn`
The number of space filling curve cells within this process
Format: *long long unsigned*
Not initialized

`lbal_tol`
Load imbalance tolerance to initiate repartitioning
Format: *float32*
Initialization value: **0**

`lbal_fudge`
Workload estimator parameter
Format: *float32*
Initialization value: **0.699999988**

`img_cmap`
Image colormap, integer within the range 0-4
Format: *int32*
Not initialized

`img_dim`
Projection dimension, integer ranged 1-3
Format: *int32*
Not initialized

`img_slabs`
Image the specified number of slices along the line of sight
Format: *int32*
Not initialized

`img_pixw`

Image width, in pixels
Format: *int32*
Not initialized

img_cx
X-coordinate of the center of image
Format: *float32*
Not initialized

img_cy
Y-coordinate of the center of image
Format: *float32*
Not initialized

img_cz
Z-coordinate of the center of image
Format: *float32*
Not initialized

img_bx
Size of the whole image box in X-dimension
Format: *float32*
Not initialized

img_by
Size of the whole image box in Y-dimension
Format: *float32*
Not initialized

img_bz
Size of the whole image box in Z-dimension
Format: *float32*
Not initialized

img_rsmoo
Image smoothing length
Format: *float32*
Not initialized

img_rbl
Black color threshold
Format: *float32*
Not initialized

img_rbr
White color threshold
Format: *float32*
Not initialized

3 Additional Programs and Libraries

3.1 `gracos-pkgs`: Required Package Installer

The fastest way to assure the proper installation of the required packages listed in Section 4.2 [Required and Recommended Packages], page 83 is probably using the `gracos-pkgs` script-executable (only 70+ lines) included with *GRACOS* source code distribution, located under the `bin` subdirectory of the *GRACOS* top source code directory tree. This script-executable is just a sequence of simple shell commands sufficient for completing all the necessary downloads and installations for the required packages, and is entirely self-sufficient, - there are no prerequisites except for the bare Unix-flavored computer system, an internet connection and the reliability of our server storing the packages.

The `gracos-pkgs` executable can be copied to any machine and should simply be executed (no root password is needed). The automatic procedure that follows, involving all the downloads and installations takes about ten minutes. In the result, all the packages required for *GRACOS* installation are automatically downloaded under `$HOME/downloads.tmp` and installed under `$HOME/local` with no other system paths involved. After the installation procedure has completed for all the packages, the `$HOME/downloads.tmp` directory is automatically removed. There are a few important points to be aware of before you actually run the executable:

- Your home directory should have at least 24Mb permanently available space for installations and about 200Mb temporarily available space for package compilation products
- Beware if you are already using either the `$HOME/local` or `$HOME/downloads.tmp` directory paths for any other purpose; - please modify the script if necessary.
- The `configure` commands used to build the makefiles for the compiled packages assume your current environment settings. Beware if you are using your special user specific settings for such environment variables as `LDFLAGS` or `CPPFLAGS`. The user environment is imported by the package specific `configure` scripts invoked within `gracos-pkgs`. It is safest to run `gracos-pkgs` on the default unmodified user system environment.
- Even if some or all of them are already installed on your system, it is all right to install them again. Not every systemwide *FFTW* installation, follows our selection for the *FFTW* `./configure` flags, for example.
- If you invoke the executable on the same system (including the CPU type) more than once, all the previous effects of this executable are automatically erased. If you invoke the executable on the same file on the machines of different CPU types (for example, when your home directory is NFS mounted across many machines) the effects of running this executable will add up since the produced installation path is specific to the CPU type (`i686`, `x86_64`, etc) of the target machine.

The installation locations of the packages are `$HOME/local/arch/'uname -m'/PACKAGE_NAME`. Architecture specific directory paths are absolutely required when installation paths are accessible via NFS (Network File System) by the machines of different CPU type. Generally, the installation procedure should be repeated for each architecture type.

3.2 `gracos-clean`: Directory Cleanup Utility

As with any other package, the users who look into the source code of will have to deal with many files automatically generated as various utilities or applications (for example `configure` or `make`) are ran. When the number of these files is large it generally becomes difficult to distinguish the automatically generated files from those manually updated. A user who modifies a source code will generally want to package the directory into another `.tar.gz` file and give it to a collaborator; in order to do that he or she will have to first clean the directory of all the unnecessary files, such as the executable, the object files or even the `Makefiles`, which are automatically generated by the `configure`.

The *Python* `gracos-clean` utility we are introducing does not have any prerequisites, except for the `.rules` files (see below) properly maintained by the user, which is usually not hard to do.

Usage: `gracos-clean [OPTION...]`

Clean the GraCos package directory tree of the unnecessary files.

The selection procedures for the "unnecessary" files are path specific and defined according to the `.rules` files distributed throughout the GraCos package directory tree, please see more information in the GraCos reference.

<code>-?, -h</code>	Produce this help message and exit
<code>--initial</code>	Equivalent to " <code>-u b,g,d -r</code> ".
<code>-r</code>	Apply recursively on the subdirectories. By default this option is not used.
<code>-m SETS</code>	Select the files matching any of the group patterns specified by SETS (the comma separated list of the group patterns defined in the <code>.rules</code> files). By default this option is not used.
<code>-u SETS</code>	Select files that match any defined pattern different from those specified by SETS. Passing an empty string will select files matching any defined pattern. This option is mutually exclusive with " <code>-m</code> ".
<code>-q</code>	Only show what will be done and exit.

Report bugs to Alexander Shirokov.

In the list below we classify all the possible procedures that locally automatically generate new files; the characteristic patterns of the these files can be grouped by the types of the generating procedure. The `.rules` text files present in most *GRACOS* source code directories are used to specify the group selection. Because the first three of the procedures in the list are not as portable as the the more standard `configure` and `make` procedures that follow, the portable *GRACOS* package distribution includes the files generated by the former.

- The `bootstrap` procedure (label 'b').
In *GRACOS*, we run `./bootstrap` in the top *GRACOS* source code directory; the package distribution includes all the files generated by this procedure.
- The generated source code files (label 'g').
In *GRACOS*, we run `./srcgen.sh` in the top source code directory; the package distribution includes all the files generated by this procedure.
- Generated document files (label 'd').
In *GRACOS*, we use the directions in `doc/src/README` under the top source code directory; the package distribution includes all the files generated by this procedure.
- The `configure` procedure (label 'c').
The procedure is described for *GRACOS* in Section 4.3.1 [Configuration], page 83.
- The `make` procedure (label 'm').
The `make` procedure is described for *GRACOS* in Section 4.3.2 [Compilation and Installation], page 84.
- Running `gracos` or other executables, label 'r'.
In *GRACOS*, the procedure is to run any of the examples in Chapter 2 [gracos Executable Reference], page 3.
- Any other procedure introduced by a user (label 'u').

3.3 `gracos-config`: Utility for Linking Programs with *GRACOS*

`gracos-config` is a utility for linking programs with *GRACOS*. By installation procedure (See Chapter 4 [Installation Procedure], page 83), the `PATH` shell environment variable directly points to the location of the `gracos-config` executable. Now, using this executable allows one to immediately retrieve the locations for any other components of *GRACOS* installation (the location of include files and libraries, package data, etc). `gracos-config` is used in most of the cases presented in Chapter 2 [gracos Executable Reference], page 3. Typing `gracos-config --help` yields the following message:

```
Usage: gracos-config [OPTION...]
  or: gracos-config [OPTION...] link_mpi
  or: gracos-config [OPTION...] link
  or: gracos-config [OPTION...] compile
  or: gracos-config [OPTION...] info [VAR]
gracos-config - GRACOS 1.0.1a8
```

```
info [VAR]:  Display the value of the Makefile variable VAR used when GRACOS
              was built.  If VAR is omitted, display all Makefile variables.
              Use this command to find out where GRACOS was installed,
              where it will look for libraries at run-time, and so on.

link_mpi:    Print the linker command-line flags necessary to link against
              the parallel GraCos library, and any other libraries it
              requires.
```

```

link:          Print the linker command-line flags necessary to link against
               the serial GraCos library, and any other libraries it requires.

compile:       Print C compiler flags for compiling code that uses GRACOS.
               This includes any '-I' flags needed to find GRACOS's header
               files.

-?, --help    Give this help list
  --usage     Give a short usage message
-V, --version Print program version

```

Report bugs to Alexander Shirokov.

The `info` argument to `gracos-config` is used to retrieve the standard locations; the values of the following `VARIABLES` are currently displayed by typing `gracos-config info`:

`datadir`, `pkgdatadir`, `prefix`, `bindir`, `top_srcdir`, `libexecdir`, `libdir`, `matlabdir`, and `includedir`

See Section *Variables for Installation Directories* in GNU coding standards (<http://www.gnu.org/prep/standards/standards.html>) for explanations.

3.4 GRACOS Library

A number of special purpose auxiliary sub packages were written to for applications with `gracos` and packaged as the so called *GRACOS library*

- `libgracos.a`
- `libgracos_mpi.a`

(the paths are lower-cased for simplicity). The libraries and the corresponding header files get installed (unless the `--disable-libs` option is used for `configure`) during `make install` into one of the installation directories.

The following table lists the C-headers for the sub packages whose compiled object files are merged to create the `libgracos.a` and `libgracos_mpi.a` libraries.

Header	Serial/MPI	Description
<code>basal.h</code>	Serial	Base Initializations
<code>random9.h</code>	Serial	Random number generators
<code>circ.h</code>	Serial	Operations on Circular topology
<code>sparse.h</code>	Serial	Sparse Volume Management
<code>lft.h</code>	Serial	Sorting
<code>pack.h</code>	Serial	Run-level packing and unpacking of data

<code>shift.h</code>	Serial	Non-overlapping partition shift implementation
<code>typemap.h</code>	Serial	Type-map data manipulation
<code>find.h</code>	Serial	Linear logarithmic finders
<code>rw.h</code>	Serial	Robertson-Walker Universe
<code>mpim.h</code>	MPI	Customized MPI_Xxx style functions
<code>heavy.h</code>	MPI	Error tolerant MPI heavy routines
<code>crown.h</code>	Joint MPI	Session initialization and code variable management
<code>mathmeth.h</code>	Joint MPI	Mathematical and statistical methods
<code>hilbert.h</code>	Serial	Hilbert curve implementation written by by Doug Moore
<code>fftmesh.h</code>	MPI	Fixing FFTW package
<code>wgifs.h</code>	Joint MPI	Raw and Smoothed GIFimage generator

The second column shows the use of MPI in the sub packages. The sub packages marked 'Serial' are compiled with a serial *C* compiler and are included into both `libgracos.a` and `libgracos_mpi.a`.

The sub packages marked 'Joint MPI' are compiled twice: with serial and MPI compiler, using different *C* preprocessing directive to separate group in and out the calls to *MPI* functions in the source code; these packages are also included into both libraries providing both serial and parallel implementations of their declarations.

The sub packages marked 'MPI' contain unavoidable *MPI* library function calls. They are compiled with `mpirun` and are included only into the `libgracos_mpi.a` library.

3.5 `gracos-rw`: Cosmographic Quantities for FLRW Cosmology

Usage: `gracos-rw [OPTION...] { dplus | ETA }`

Evaluate miscellaneous, mostly cosmographic quantities for Friedmann Lemaitre Robertson Walker (FLRW) cosmology. Most of the quantities require the `--omegam`, `--omegav` and `--aexp` options to be set; use the `-v` option to check the parameter settings. The source code is located in file `dplus.c`.

```

-a, --aexp=float      Universe expansion factor
  --omegam=float      Omega matter
  --omegav=float      Omega vacuum
-v, --verbose        Produce verbose output
-?, --help           Give this help list
  --usage            Give a short usage message
-V, --version        Print program version

```

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

Report bugs to Alexander Shirokov.

3.6 `gracos-ehu`: Eisenstein-Hu Transfer Function

A well-established fit for the matter transfer function (TF) by Daniel J. Eisenstein & Wayne Hu yields the precision of matter transfer function within a few percent level across wide range of the cosmological parameters. For convenience, *GRACOS* installation includes the executable, `-` the generator of the transfer function.

Usage: `gracos-ehu [OPTION...]`

Produce a transfer function table using the transfer function fitting formulae for CDM + Baryon + Massive Neutrino (MDM) cosmologies, by Daniel J. Eisenstein & Wayne Hu, based on astro-ph/9710252. This fitting function is for the CDM Variants accurate for ($W_b / W_0 < 0.3$, $W_n / W_0 < 0.3$). The default values for all option arguments are zeros. The source code used is originally downloaded from the authors resource at

<http://background.uchicago.edu/~whu/transfer/transferpage.html>

on Mar 25, 2007 and is packaged within GRACOS with file 'ehu.c'.

```

--degen_hdm=int      Number of degenerate massive neutrino species.
--hubble=float       Hubble constant, in units of 100 km/s/Mpc.
--krange=str         The range of the wavevectors, formatted as
                    "kmin:nk:kmax", where kmin and kmax are expressed
                    in comoving Mpc-1.
--omega_baryon=float Density of baryons.
--omega_hdm=float    Density of massive neutrinos.
--omega_lambda=float Cosmological constant.
--omega_matter=float Density of CDM, baryons, and massive neutrinos.
--redshift=float     The redshift at which to evaluate the transfer
                    function.
-v, --verbose        Produce verbose output
-?, --help           Give this help list
  --usage            Give a short usage message
-V, --version        Print program version

```

Report bugs to Alexander Shirokov.

3.7 `gracos-hist`: Utility for Plotting Histograms

Usage: `gracos-hist` [OPTION...]

Take numbers from the standard input and produce a histogram of the specified range to the specified output file. This utility will work for any size of the input dataset.

<code>-h, --high=float</code>	The highest value to include into the histogram (default: 0).
<code>-l, --low=float</code>	The lowest value to include into the histogram (default: 0).
<code>-n, --nbins=int</code>	Number of bins in the histogram (default: 100).
<code>-o, --o=string</code>	The name of file for output (default: "hist.dat")
<code>-?, --help</code>	Give this help list
<code>--usage</code>	Give a short usage message
<code>-V, --version</code>	Print program version

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

Report bugs to Alexander Shirokov.

3.8 Serial Fortran N-body Codes

The `src/f77` subdirectory in *GRACOS* contains the serial *Fortran 77* N-body codes written in 1993-1994 that have been well known to the cosmological community (especially the `grafic-2_101` and `p3m2-93` packages) and have served as the development base for *GRACOS*. The complete list is shown below, these codes are installed as a part of *GRACOS* package; they are used for testing and verification purpose in Section 2.10.1 [Comparing `gracos` with Fortran `p3m` and `grafic1`], page 58 and for generating the matter transfer function, See Section 2.6.1 [`grafic`: the Initial Conditions Generator], page 33.

The installed versions of these codes limited to a very narrow range of the input parameters. First of all, all of these codes are serial and the `p3m2-93` and `grafic1` executables are not suitable for large particle numbers (`lingers` is free of this limitation as it does not use particles); second, since the Fortran 77 compiler does not allow dynamic memory allocation, both the particle number and the grid size are fixed in `grafic-2_101` and `p3m2-93` to 32^3 particles and mesh size. In order to change those parameters one needs to modify `grafic1.inc` under `grafic-2_101` and `p3m2.inc` under `p3m2-93` and type `make` to recompile. The products of these non-default compilations should only be installed (by typing `make install`) after having introduced the corresponding changes in the parameters in the *gracos* scripts described in Section Section 2.10.1 [Comparing `gracos` with Fortran `p3m` and `grafic1`], page 58.

The following is the complete list of the Fortran packages

grafic-2_101

Installed executables: **grafic1** and **lingers**.

lingers generates the transfer function and/or the initial particle velocities from a given power spectrum of matter density perturbations. Only minor differences from the original version, available at <http://web.mit.edu/edbert/grafic-2.101.tar.gz>; except for the new **random9** random number generator that fixes the portability bug found in the original version.

p3m2-93

Installed executables: **iniftab** and **p3m2-93**.

iniftab is used to generate force tables, **p3m2-93** is used to run an N-body simulation using the generated force tables. The **p3m2-93** code accepts files written in the Section 2.4.5 [p3mdat: Fortran p3m Datafile Format], page 20 as the initial conditions and outputs the particle data in the same format at the universe expansion factors passed to the standard input by the user.

ic2dat

A C-code converting the velocities of the particles stored in the **ic_vel** files produced by the **grafic-2_101** code into the particle data file suitable for input for **p3m2-93** executable, See Section 2.4.5 [p3mdat: Fortran p3m Datafile Format], page 20.

swapb

Convert the **float32** and **int32** data from big endian to little and vice versa.

The **grafic-2_101** and **p3m2-93** packages distributed as the part of *GRACOS* slightly differ from their original versions. A very limited number of changes were introduced for good reasons, See the ‘**ChangeLog**’ files. These changes do not introduce any errors: the output of a run with the original version of the codes is identical to the equivalent run accomplished with the modified versions; the latter however introduce some additional options and minor bug fixes.

3.9 bits

bits is a utility used to convert between a 32 bit integer and its bitwise representation in the form of a string of zeros and ones. The least significant bit corresponds to the rightmost character in the string, unless the **-t** option is given (see below). The following flags to **bits** are accepted

- s string** Input the **string** and convert it to an integer
- n int32** Input the integer and convert it to a string
- t** Reverse the order of the significance of the bits in the bit string, making the leftmost character represent the least significant bit.
- L** If the **-s** option is present, check that the input string is 32 character long.
If the **-n** option is present, ensure that the output string is exactly 32 characters long by padding it with '0's if necessary.

As an example, the string ‘1100’ represents the number 12 in its bitwise representation, when the least significant bit corresponds to the rightmost character of the string. On

the other hand the same string shows the bitwise representation of 3 when the rightmost character represents the least significant bit. We can use the `bits` as follows to perform the same and the inverse conversions

```
SHELL$ bits -n12
1100

SHELL$ bits -t -n3
11

SHELL$ bits -s1100
12

SHELL$ bits -t -s1100
3
```

For a demonstration of the use of the `-L` flag execute the same commands with the flag added.

3.10 System Tests

This chapter describes a few non-standard system tests that have been successfully used for identification of some particular system-related problems running `gracos`. The executables are installed during a regular `gracos` installation, See Section 4.3.2 [Compilation and Installation], page 84. If your system fails on any of these tests, then you should either reconfigure `gracos` with some specific `./configure` options to work around limitations, or replace the faulty hardware. We hope that the need for this chapter will diminish as the Linux systems improve over time.

3.10.1 `ibtest`: System Call Portability Test

Some networking systems are known to have `fork/system` call portability issues for parallel applications. *Infiniband*, is the example: a parallel application using system calls terminates randomly without an error message, leaving the impression that the errors are due to erroneous handling of dynamic memory allocation. This problem has initially been submitted as a bug report to the TACC computer center and according to *Karl W. Schulz* of the support staff, as of January 1, 2007 there is no available fix.

`ibtest` is the utility for testing your system for this issue. The following usage applies

```
Usage: ibtest [OPTION...]
```

```
Diagnositics tool for testing the portability of system calls in parallel
applications to your networking system. Use the appropriate mpirun command
prefix to run it in parallel, use four processes for optimal timing.
```

```
-n, --no-sys-calls      Do not make system calls
-?, --help              Give this help list
--usage                 Give a short usage message
-V, --version           Print program version
```

Report bugs to Alexander Shirokov.

To test your networking system, run `ibtest` on your system in parallel on four processes. The job should complete within a few minutes. The standard output of process zero in-

icates the test result. If it is positive you should not use `--disable-system-calls` for *GRACOS* configuration.

3.10.2 memtest: Hardware Memory Test

`memtest` is a testing utility for memory hardware on your cluster and can be used to test memory on all the nodes of your cluster simultaneously. The following usage applies

Usage: `memtest [OPTION...]`

Parallel utility for testing memory hardware simultaneously on the entire cluster of computers. The test is performed for each node participating in the run.

```

-f, --mem-frac=NUMBER      The fraction of total system memory per node
                           [0.2]
-n, --number-iterations=NUMBER  The number of test iterations [5]
-?, --help                 Give this help list
    --usage                 Give a short usage message
-V, --version               Print program version

```

Mandatory or optional arguments to long options are also mandatory or optional for any corresponding short options.

Report bugs to Alexander Shirokov.

Run `memtest` in parallel on all the nodes available in your cluster. The message shown in the standard output of process zero indicates the hostnames of the nodes (if any) that have faulty memory chips.

The memory is tested by allocation of the total memory fraction given by the `-f` option on all nodes, filling it with a prescribed sequence of data and later verifying that the sequence of data as written in the machine is authentic.

The default value used for `--mem-frac` option is sufficiently low to avoid swapping, increase this number cautiously to improve the quality of the test result. Also, use larger value for `--number-iterations` than the default at your discretion using larger numbers increases the intrinsic number of iterations and therefore also increases the quality of the diagnostics.

Generally, only the manufacturer supplied diagnostic tools can reliably test the RAM chips on your system. Running manufacturers diagnostics tools is often hardware specific and running it in parallel may be tricky. `memtest` on the other hand is portable and will diagnose only serious problems if they exist. The probability of a failure of any one memory RAM is quite low but is considerable for large clusters of computers. We suggest running `memtest` from time to time to ensure no serious hardware problems.

The need to write this testing utility originated when one of our early `gracos` runs crashed repeatedly with an error message indicating a segmentation fault. The error message always appeared on one of the nodes, which lead us to believe that the problem was caused by a faulty memory chip as it was wearing down. `memtest` has been used to prove this and was used to identify many more faulty memory chips on our cluster.

4 Installation Procedure

4.1 System Requirements

GRACOS is ported to most *UNIX* flavored systems, except for the *Mac OS*, whose native C-compiler does not support `ntptimeval`. It may be possible to compile *GRACOS* on this system using a different C-compiler (either *gcc* or *icc*), however this has not been tested.

4.2 Required and Recommended Packages

GRACOS installation *requires* the following freely downloadable open source packages to be installed on your system

- A Message Passing Interface (MPI) implementation must be installed; the list of freely available implementations includes LAM (<http://www.lam-mpi.org>), MPICH (<http://www-unix.mcs.anl.gov/mpi/mpich/>), and Open MPI (<http://www.open-mpi.org>).
- The FFTW (<http://www.fftw.org>) package `fftw-2.1.5` (<http://www.fftw.org/fftw-2.1.5.tar.gz>) must be installed, configured with an `--enable-mpi --enable-float --enable-type-prefix` options to enable support for MPI, single precision mode and type prefix specification for the FFTW header files.
- *Tcl* library version 8.4 (<http://www.tcl.tk/man/tcl8.4>) must be installed.

The following packages are *recommended*

- *MATLAB*; *GRACOS* provides a few *MATLAB* scripts to analyze various data from the datafiles produced in the output. Those scripts, located upon finishing the installation procedure at `$PREFIX/matlab` (See Section 4.3.3 [Environment Setup], page 85 for the definition of `$PREFIX`), can easily be rewritten for other plotting software.

4.3 *GRACOS* Package Installation

GRACOS installation procedure below follows the standard path widely adopted in the open source software world. Once the package is installed it is not necessary to keep the source code directory; keeping it may be useful however for reference or revision purposes.

If you have any problems with the *GRACOS* installation procedure below please consult the *GRACOS* FAQ (<http://www.gracos.org/faq/index.html>) webpage. Then, if that does not help please let us know.

4.3.1 Configuration

The *GRACOS* installation procedure starts with the conventional

```
./configure [OPTION]... [VAR=VALUE]...
```

assuming that the required packages are already installed. The setting of `configure` `VARIABLES` is probably simpler via the use of shell environment variables rather than the `configure` command line (type `./configure -h` for more information). One should make careful considerations for the settings for `OPTIONS` and `VARIABLES` before proceeding with the `configure` command. We provide useful guidelines below.

First, one has to make a choice of the installation path for *GRACOS* and specify it with the `--prefix` option to `configure`. It is recommended to install *GRACOS* into a unique directory by using `--prefix=...SPECIFY.../gracos-1.0.1a8`. If you skip the `--prefix` `configure` option the default systemwide setting (`/usr/local`) will be automatically used and you will be required to have a root password for install.

Second, one has to make sure that the required packages are found by `configure`. The `PATH` shell environment variable must be set appropriately so that the location of the compiled MPI executables (usually `mpicc` and `mpirun`) is automatically detected. If the location of any of the required *Tcl* and *FFTW* packages is not standard, the `LDFLAGS` and `CPPFLAGS` `configure` variables must be set accordingly.

Third, one has to make a wise decision for the choice of the *C*-compiler flags, particularly the optimization options. Setting `CFLAGS="-Wall -O3 -fno-strict-aliasing"` is a good initial guess. If you are using the `icc` compiler, you might be able to use `'-fast'` instead of `'-O3'`. If you plan running a massive N-body simulation, it may very well be worth researching the compiler documentation in order to choose the most efficient optimization options. For example, if you are using the `icc` compiler, read the "Optimization Levels" and "Automatic Processor-specific Optimization" sections in the manual page and find options for optimizing by *vectorization*. Vectorization yields the runtime speedup by a significant fraction (roughly 20-30% or more) at the cost of slightly increased compilation and installation time.

You may switch between different C-compilers by using the `CC` `configure` variable.

Finally, some important non-standard `configure` options are listed below (type `./configure -h` for the complete list of `configure` options).

Using `--enable-tests` option results in production of a number of light weight executables used for testing in the subpackages at the cost of slightly increased installation time. Those executables may serve as the hello world examples for the users who link their codes against *GRACOS* libraries.

The `--disable-slibs` option disables compilation of the serial libraries. The `--disable-libs` options disables the installation of all libraries and header files at `make install`.

The `--disable-system-calls` `configure` option must be set on the systems that have `fork/system` support issues. If you do not use *Infiniband* networking system in your cluster (ask your system administrator) you are not likely to have this problem as this is the only system of the currently known to us having this problem. Use `ibtest` utility at any time to test your system for this issue, See Section 3.10.1 [ibtest: System Call Portability Test], page 81.

4.3.2 Compilation and Installation

After the `configure` command has finished successfully, type

```
make
```

to compile and, if succeeded,

```
make install
```

to install all the *GRACOS* components on your system.

4.3.3 Environment Setup

Once `make install` is finished, the user should append the `PATH` and `MATLABPATH` environment variable in their shell profile appropriately. For example, the `sh` or `bash` shell users should put the following into their `~/.bashrc` file

```
export PATH=$PREFIX/bin:$PATH
export MATLABPATH=$PREFIX/matlab:$MATLABPATH
```

where `$PREFIX` is normally the argument of the `--prefix` option used for the `./configure`; the `tcsh` or `csh` users should instead put

```
setenv PATH $PREFIX/bin:$PATH
setenv MATLABPATH $PREFIX/matlab:$MATLABPATH
```

into their `~/.tcshrc` or `~/.cshrc` files (whichever is applicable, or ask your system administrator).

4.3.4 Uninstallation

GRACOS can be uninstalled either by typing `make uninstall` within the source code directory or, if you did specify a unique installation path for `$PREFIX`, by invoking `rm -rf` for the installation path.

4.4 Source code Directory Cleanup

As a side effect of the package installation procedure, many files are automatically generated throughout the source code directory tree, making it heavy sized and sometimes confusing for the users who may wish to work with the source code. Type `gracos-clean --initial` anywhere within the source code directory to bring all the subdirectories to the initial pristine state of the package; See Section 3.2 [`gracos-clean`: Directory Cleanup Utility], page 74 for more options and details.

5 Authors and Copyrights

The `gracos` package is written by

- Alexander Shirokov, (CITA/MIT)
- Edmund Bertschinger (MIT)

Copyright (C) 2002-2007 Alexander Shirokov and Edmund Bertschinger

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Software Included in GRACOS:

Hilbert Curve implementation: files `hilbert.h` and `hilbert.c`
Copyright 1998, Rice University

GIF maker: based on the original files `wgif.c`, `compress.c` and `cmmap.h`
Copyright 1989, 1990, 1991, 1992, 1993 by John Bradley

M4 Macros: files `gse_get_version.m4` and `get_gse_version`
based on the original files `lam_get_version.m4` and `get_lam_version`
Copyright (c) 2001-2006 The Trustees of Indiana University.
Copyright (c) 1998-2001 University of Notre Dame. All rights reserved.
Copyright (c) 1994-1998 The Ohio State University. All rights reserved.

file `acx_mpi.m4` by Steven G. Johnson
Copyright 1997-1999, 2003 Massachusetts Institute of Technology

Transfer Function Fitting Formula: file `ehu-power.c`
based on the original files `power.c` by
Daniel J. Eisenstein & Wayne Hu, Institute for Advanced Study

6 Acknowledgements

Alexander Shirokov would like to thank Neal Dalal for suggestions that helped to develop this work.

This material is based upon work supported by the National Science Foundation under Grant No. 0407050. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

This work was financially supported by the Canadian Institute for Theoretical Astrophysics (CITA) and the Natural Sciences and Engineering Research Council of Canada (NSERC).

