Quickstart

Table of contents

Ginnungagap consists of a set of separate tools to prepare cosmological initial conditions:

- `ginnungagap` to prepare velocity fields starting from a white noise field (or just a random seed), and a set of cosmological parameters
- `generateICs` to convert velocity fields into GADGET-files

For a simple (non-zoom) simulations these two programs are sufficient, but for a multi-level zoom-in simulation other tools are needed:

- `realSpaceConstraints` to rescale the white noise fields,
- `refineGrid` to increase or decrease the resolution of the velocity fields,
- `prepare_ini.sh` script to make preparation of complex multi-scale initial conditions easy.

Ginnungagap basics

Initial conditions are started from a white noise field that is a 3D array of random numbers. This array is convolved with the Transfer function to make velocity fields, which are then transformed to particle offsets using Zel'dovich approximation.

Ginnungagap can be used for the following purposes:

- Making full box initial conditions. Note that Ginnungagap is fully MPI-parallel.
- Rescaling a full box initial conditions (increasing or decreasing resolution)
- Rescaling a white noise field.
- Gas support (by simply duplicating DM particle parameters).
- Making multi-scale (zoom) initial conditions. The zoom can be done on the basis of any full box simulation by adding more scales with increased/decreased resolution. The zoom region may be placed anywhere, including the boundaries of the full box. Afterwards it can be shifted to the box center.
- Output can be written in GADGET-2 (full support) or GRAFIC (only full grid currently) formats. In GADGET-2 format, particles of different mass can be assigned to different GADGET-2 particle types to optimize memory usage.
- Updating the initial conditions already made with Ginnungagap, without re-making everything from scratch. Only the parts that change will be

updated. E.g. making gas/no-gas versions requires only running the conversion tool (`generateICs`).

For multi-scale initial conditions, a set of velocity fields with different resolutions are needed. If the zoom region is small, the high resolution velocity fields do not need to cover the whole volume, so these velocity fields differ not only by resolution, but also by their size.

The velocity fields of different resolution can be prepared in different ways:

1. by rescaling a white noise with `realSpaceConstraints` and making new velocity fields with `ginnungagap` just from it,
2. by using an additional information from the lower resolution velocity fields to improve the accuracy of the previous method with `refineGrid`,
3. by downgrading a high-resolution velocity field using nearest-grid-point interpolation with `refineGrid`.

Method 2 is recommended over method 1 when possible. Method 2 also allows to make high resolution white noise and velocity fields only in a small cubic box surrounding the zoom region.

Installing Ginnungagap

The main requirements of Ginnungagap are:

- FFTW-3
- Gnu Scientific Library
- SPRNG-2
- MPI and HDF5 are highly recommended.

The simplest way to download and build them is to use the following installer script:

scripts/g9p_installer.sh

Otherwise, you can use it as an example of what build flags are required.

It is usually *NOT* recommended to use a version of HDF5 provided by your system. After the installation you need to update your LD_LIBRARY_PATH with the path to the newly installed libraries.

Example

For a three-level zoom-in simulation with scales of 256ˆ3, 512ˆ3 and 1024ˆ3, the example workflow looks like this:

1. Make a 256ˆ3 white noise (`ginnungagap`)
2. Make 256ˆ3 full grid velocity fields (`ginnungagap`)
3. Make GADGET-2 files (`generateICs`)

4. Run the low resolution full grid simulation
5. Identify the zoom region and make the mask using e.g. `tools/zoomTools/LareWriter`
6. Rescale the white noise from step 1 to 512^3 (`realSpaceConstraints`)
7. Make 512^3 velocity fields with only the small scale information (`ginnungagap`)
8. Interpolate 256^3 velocity fields from step 2 and add to them small scale information from step 7 to produce the 'final' 512^3 velocity fields (`refineGrid`)
9. Rescale the white noise from step 6 to 1024^3 (`realSpaceConstraints`)
10. Make 1024^3 velocity fields with only the small scale information (`ginnungagap`)
11. Interpolate 1024^3 velocity fields from step 8 and add to them small scale information from step 11 (`refineGrid`)
12. Make GADGET-2 files for the zoom simulation (`generateICs`)

This workflow becomes more complicated if you need more levels of zoom. Each tool is controlled by a separate `.ini` file with parameters, such as the input/output filenames, grid sizes, etc.

In order to avoid writing the `.ini` files for every operation manually it is suggested to ue a script called `prepare_ini.sh`. It takes only one 'master' `.ini` file as an input and produces all the required 'child' ini files from it, as well as prepares tasks and a Makefile that allows you to submit all the required tasks to your computer within one single command `make gadget`.

In the `doc/examples/zoom` you can find two example ini files: `example_64.ini` for a single level simulation and `example_zoom.ini` for a multi scale one. To start a project for a zoom simulation, do the following:

1. Create a directory for non-zoom simulation, e.g. 'my__project__nozoom'.

2. Copy the following files to that directory: `prepare_ini.sh`, `batTemplate_local.sh`, `example_64.ini`, `ginnungagap` and `generateICs`

3. Create a directory for the zoom simulation, 'my__project__zoom', and copy there the same files, but `example_zoom.ini` instead of `example_64.ini`, and also `realSpaceConstraints`, `refineGrid` and `lare.dat`.

4. In the folder 'my__project__nozoom' type the following commands:

   ```
   ./prepare_ini.sh example_64.ini
   make gadget
   ```

   after a while the file named 'GADGET' will appear in that folder and this is your ICs. Some other files are also created.

5. Now cd to 'my__project__zoom'.

6. Make a soft link to the white noise field from the no-zoom simulation:

   ```
   ln -s ../my_project_nozoom/wn_64.h5 .
   ```

7. Again, run:

   ```
   ./prepare_ini.sh example_zoom.ini
   make gadget
   ```

   The files GADGET.0 ... GADGET.3 will be created which contain the zoom ICs.

Now let's look what the `example_zoom.ini` contains:

```
[ICs]
Box = 64.0
; box size in Mpc/h

meshes = 32 64 128 256
; resolution for each mesh

startMesh = 64
; for meshes with lower resolution than startMesh the velocity fields will be
; produced using nearest grid point interpolation from the startMesh velocity
; filed. For meshes with higher resolution the complex rescaling algorithm
; described in the ginnungagap paper will be used.

seeds = 1001 1002 1003 1004
; the seeds will be used corresponding to the meshes above when they are needed

zInit = 49
; initial redshift

[options]
doGas = false
doLongIDs = false
autoCenter = false
; place zoom region at the box center
useKpc = false
; which gadget output units to use

[WN]
wnStartFile = wn_64.h5
; initial white noise field (set to 'none' to generate new):
; don't include path in the filename!
; use none if you are not using an input file.

wnStartType = hdf5
; type of the input white noise field
; can be hdf5 or grfic
; note that output is hdf5
```

```
wnPrefix = wn_
; prefix for all the rest WN files that are generated from the initial one
; don't include path in prefix!

[files]
velPrefix = g9p
; don't include path in prefix!
gadgetPrefix = GADGET
; don't include path in prefix!
doPatch = true
patchStartMesh = 256
; patch will be cut starting with this mesh, for lower resolution the whole
; fields will be written. The patch position and dimentions calculated from
; maskFile given below.

[HDF5]
chunk = 128
; start do chunking for mesh > chunk and use chunk as a chunkSize

[gadget]
gadgetTypes = 4 2 2 1
; gadget particle types in the same order like meshes.
gadgetNFiles = 1 1 1 1
; number of files for each zoom level in the same order like meshes.

[mask]
; mask will be applied if you spiecify more than one mesh in the [ICs] section
maskFile = lare.dat
; file with the lagrangian region mask
maskMesh = 64
; the mesh at which the mask was constructed

[Cosmology]
modelOmegaRad0 = 0.0
modelOmegaLambda0 = 0.692885
modelOmegaMatter0 = 0.307115
modelOmegaBaryon0 = 0.048206
modelHubble = 0.6777
modelSigma8 = 0.8288
modelNs = 0.9611
# You can use built-in transfer function:
powerSpectrumKmin = 1e-6
powerSpectrumKmax = 1e3
powerSpectrumNumPoints = 501
transferFunctionType = EisensteinHu1998
# or a file with the power spectrum:
```

```
#powerSpectrumFileName = mySpectrum.txt
; here if you want to use the input file with power spectrum,
; just uncomment that line

[submit]
batTemplate = batTemplate_local.sh
; template file with submitting commands for your system
```

The file `example_64.ini` has almost the same contents, the difference is only in `meshes`, `seeds`, `gadgetTypes`, `gadgetNFiles`.

The comments in the example explain all the options.

The script `prepare_ini.sh` creates a bunch of .ini files for all the tools of Ginnungagap, also it creates bat_* files which contain scripts to run all these tools. A Makefile is generated which is used to run all these files in a right sequence and track dependencies: if you change some parameters, `prepare_ini.sh` will update only the *.ini and bat_* files which are affected by the change, and `make` will take care to run only those of them which are needed.

Changing resolution for a non-zoom run

If you have a white noise field for some non-zoom simulation and would like to make ICs for another non-zoom simulation with different resolution, you should specify the resolution of your existing ICs as `startMesh` and the target resolution as `meshes`. In this case the script will decide what is need to be done: downgrading or upgrading the resolution, and it will produce the relevant files for this purpose. The non-zoom ICs can be created with `make gadget` command.

IMPORTANT! You need to pay attention to have `doPatch = false` in case of a non-zoom simulation.

Known bugs

You should always monitor the output of `make gadget` or tasks submitted by it to look for errors.

Zeros in statistics

The programs `ginnungagap`, `realSpaceConstraints` and `refineGrid` calculate statistics on grids they are working with. If you see somewhere output like this:

```
 Calculating statistics on second input grid... took 0.13692s
                  mean  :     0.0000000000
    standard deviation  :     0.0000000000
```

it means that something is wrong. If this happens during `refineGrid`, most probably it is due to too large `chunkSize` - when `doPatch = true` the patch size could get smaller than the chunkSize if the latter is too big.

Segmentation fault during generateICs

Sometimes generateICs finishes with Segmentation Fault during execution on the local machine. The bug is hard to reproduce and usually if you run `make gadget` again, it does not appears at the same place.

Running on different clusters

The scripts are provided for a number of clusters. Each script contains some routines to compute the number of nodes and cores needed to execute each task based on the memory requirements.

After the execution of `prepare_ini.sh` some useful information is written on the screen, like this:

```
Recommendations:
Maximal memory needed: 68719 MB
Maximal number of nodes: 1
```

from which you know how many nodes do you need. This information is also saved to `recommendations.txt`. Sometimes you will need to select the queue and allocate resources according to this recommendations yourself.

SuperMUC

Use the template file `batTemplate_supermuc.sh`. In the beginning of the script set `variant = hw` for haswell nodes or `variant = thin` for thin nodes. After running `prepare_ini.sh`, run `make gadget` in your shell and a number of files with the names ending by `.seq` will be generated. They are copies of `bat*` files with one additional command added to each of them which will submit the next script. In order to submit the whole sequence, type

```
    llsubmit 1.seq
```

If after making some changes you will need to remake the ICs, type again `make gadget` and the contents of `*.seq` files will be updated automatically. During this procedure all previously created files ending with `.seq` will be deleted.

Jureca

Use the template file `batTemplate_jureca.sh`.

Each `bat_\*` file contains `srun` command. In order to submit all the scripts there are two options:

1. Do it interactively by invoking:

   ```
   salloc --partition=devel --nodes=1 --time=01:00:00
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/libs
   make gadget | tee LOG
   exit
   ```

2. Or in a script like this:

   ```
   #!/bin/bash -x
   #SBATCH --nodes=1
   #SBATCH --ntasks-per-node=24
   #SBATCH --time=01:00:00
   #SBATCH --partition=devel
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/libs
   make gadget
   ```