WE-Heraeus and NARIT Cosmology School 2025
How to make a Universe in a computer

Rainer Weinberger

last modified: October 10, 2025

There is a git repository with example scripts available on github. To download visit the website or clone it via

```
git clone https://github.com/rainerweinberger/CosmoComputingSchool.git
```

Note that the repository contains more examples than the ones covered here.

1 Cosmological simulations in theory

Cosmological simulations are performed to solve the *initial value problem* of the matter distribution in a representative part of the Universe. There are a number of assumptions from our standard picture about the Universe on the largest scales, in particular

- We have a good understanding of the distribution of matter in the early Universe from the *cosmic microwave background* radiation, and can describe these matter inhomogeneities using perturbation theory.
- The Universe is homogenous and isotropic on large scales. Thus, we can set up a finite simulation volume with *periodic boundary conditions*.
- The expansion history of the Universe can be solved independently from the formation of structure. This allows us to solve the mutual interaction of particles in *comoving coordinates*.
- Apart from the expansion itself, we can treat gravity Newtonian. In particular, gravity acts instantaneously.

Cosmological simulations are by now a well-established technique, with a solid physical, algorithmic and numerical basis. Mastering all the details of this modelling is a substantial undertaking that typically takes years. Thus, the instructions found here are solely to help with the practical problem of getting started to run a full simulation, and to get a first sense of the data-analysis of the simulation results. For the interested reader, further (substantially more comprehensive) reading is provided below.

1.1 Literature

- A comprehensive description about the equations and a high-level description of the numerical methods used in cosmological simulations of galaxy formation can be found in (Vogelsberger *et al.*, 2020)
- The detailed algorithms can be found in dedicated code papers.
  - The Gadget-4 code is described in (Springel *et al.*, 2021).
  - The Arepo code is described in (Springel, 2010; Weinberger *et al.*, 2020)
- To learn more about the additional physics in galaxy formation simulations (Feldmann and Bieri, 2025) provide an up-to-date overview.
- One of the reference graduate-level textbooks on galaxy formation, covering topics from cosmological structure formation to galaxy evolution is Mo *et al.* (2010).

2 Compiling Gadget-4 – common obstacles

Simulation codes are complex software that make use of libraries and aspects you might not be familiar with. This includes numerics libraries you haven't installed on your computer or libraries to enable parallel computation on multiple, distributed compute cores. This generally increases the complexity of using such codes and complicates the process of getting started. This section aims to address some of the most common obstacles.

2.1 Get code and compilation

To run a simulation, you first need to download the simulation code. Similar to the scripts, modern simulation codes are usually managed in version-control systems, mostly git.

```
> git clone https://github.com/rainerweinberger/gadget4.git gadget4
```

This will download the code in your present directory. Change into the `gadget4` directory

```
> cd gadget4
```

To run a simulation, we need a number of files. First, a configuration file, which specifies which of the built-in algorithms are actually used. In gadget-4, this file is typically called `Config.sh`. There are several examples provided with the code. Copy the provided configuration file to the code directory

```
> cp examples/DM-L50-N128/Config.sh ./
```

Before we can compile the code, we need to ensure that the paths to the required libraries are known to the build system. While these paths are generally standard in normal installations, say, e.g. of a Ubuntu Linux system, these paths can vary on high-performance supercomputers and need to be set accordingly (this is information that should be accessible in the respective machine documentation or from the support staff). For computers on which the code has already been used, these generally do not change. These settings are already stored and can be used by setting the `SYSTYPE` variable. To do so, make sure there exists a file called `Makefile.systype`. If not create one by copying the template:

```
> cp Template-Makefile.systype Makefile.systype
```

Open `Makefile.systype` and add **one** of the following line

```
SYSTYPE="macOShomebrew" # mac os
SYSTYPE="Ubuntu" # Ubuntu Linux
```

to set your operating system. If none of these work, the paths to the libraries are incorrect.

2.2 Compilation error

Errors like, e.g.,

```
fatal error: 'hdf5.h' file not found
```

indicate that the include path (in this case of the hdf5 library) is incorrect. These can be set in `./buildsystem/Makefile.path.*`, where "*" denotes the `SYSTYPE` you are using. The different `*_INCL` paths, starting with `-I`, should point to a directory in which the relevant header `*.h` files are located. In our case, the error indicates that the path listed after `HDF5_INCL = -I` is incorrect, i.e., does not contain the file hdf5.h.

2.3 Linking error

For error messages like

```
error: linker command failed with exit code 1 (use -v to see invocation)
```

the library path, i.e. the path to the compiled $*.o$ or $*.so$ files is incorrect (to find out which library, read through the messages prior to this error message). These are given in the same buildsystem file for the `*_LIB` paths.

2.4 Finding libraries

Finding the library paths can be done differently, depending on system. For compute clusters and supercomputers, the cluster documentation of the module manager is the first place to consult. For your own laptop/workstation, you can search for the libraries in the common system library paths (here MacOS with homebrew package manager), e.g., to search for the hdf5 library:

```
find /usr/lib /usr/local/lib /opt/homebrew/lib -name "libhdf5*"
```

In case you are searching for the header `hdf5.h`, this would look the following:

```
find /usr/local/include /opt/homebrew/include -name "hdf5.h"
```

3 Cosmological N-body simulation.

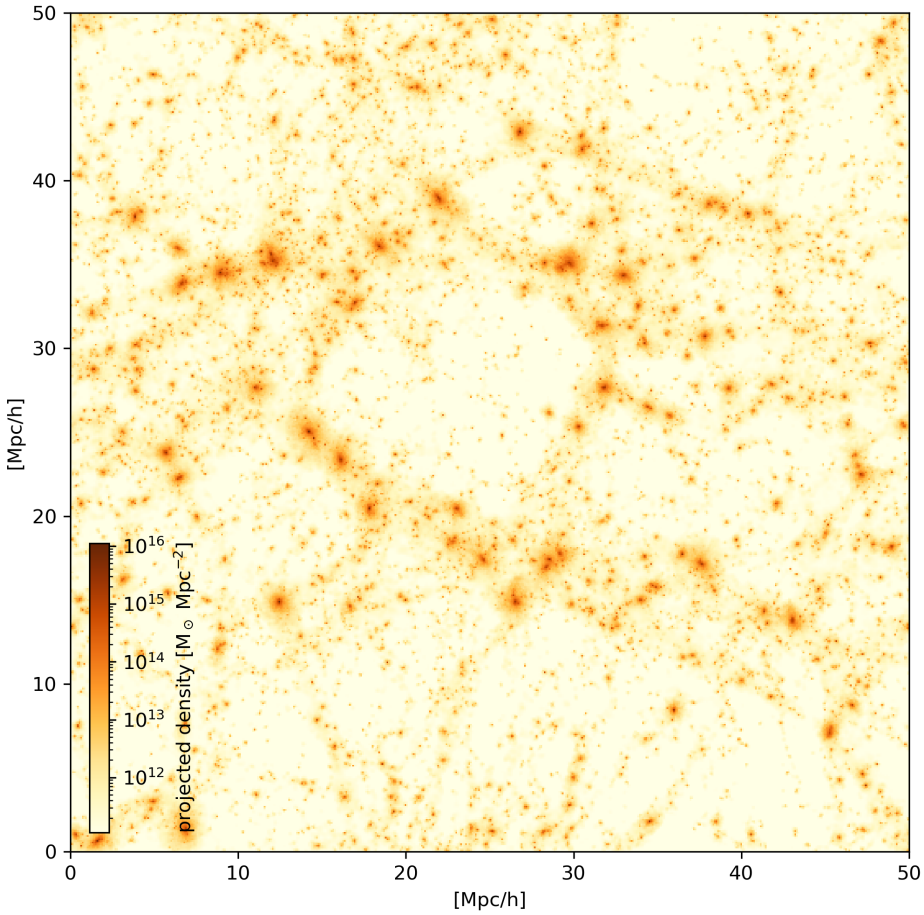

Figure 1: Density projection of matter structure of a cosmological N-body simulation.

This example will cover a full cosmological N-body simulation performed with the Gadget-4 code. A visualization of the output (at slightly higher resolution) is shown in Figure 1. The initial condition generator is part of the code and executed at startup. The settings are chosen to create a computationally relatively inexpensive simulation that can be performed on a laptop within a few minutes. **The repository also contains the simulation output needed to do the analysis (in case you are unable to run the simulation).**

Learning goals

1. Learn how a cosmological N-body simulation is set up
2. Understand the main simulation parameters
3. Visualise the output of an N-body simulation
4. Analyse the output of an N-body simulation

3.1 Simulation setup

Make sure you are in the directory with a downloaded repository

```
> ls
CosmoComputingSchool
```

Create a simulation directory and change directory into it.

```
> mkdir sim_cosmobox_nbody
> cd sim_cosmobox_nbody
```

Get code

Get the Gadget-4 code

```
> git clone https://github.com/rainerweinberger/gadget4.git gadget4
```

Copy configuration and parameter files.

Copy the respective `Config.sh` and `param.txt` files into code directory

```
> cp ../CosmoComputingSchool/cosmobox_nbody/Config.sh ./gadget4/
> cp ../CosmoComputingSchool/cosmobox_nbody/param.txt ./
> cp ../CosmoComputingSchool/cosmobox_nbody/outputs.txt ./
```

Compile Gadget-4

Move to the gadget-4 directory

```
> cd gadget4
```

Open the file `Makefile.systype` and select your system type by removing the leading `#`. Once this is done, you can compile the code by typing

```
> make
```

If there are error messages, follow the problem solution presented in Section 2. If the compilation was successful, the directory should contain the executable called `Gadget4` (can be checked by typing `ls`). If this is the case, leave the directory.

```
> cd ..
```

3.2 Main parameters and running

Prior to performing the simulation, let's have a look into the configuration and parameter files. You can open these files with any text editor or display the content on the terminal using

```
> cat gadget4/Config.sh
```

We will only discuss a subset of the parameters and options here. For a full description, have a look at the code's documentation that is accessible via the website

`https://wwwmpa.mpa-garching.mpg.de/gadget4/`

Config.sh options

The options in Config.sh need to be set prior to compilation. Each time these are changed, the code needs to be re-compiled since those flags literally hide or reveal different parts of the source code to the compiler.

```
# Basic code operation
    PERIODIC  < Boundary conditions for gravity
    SELFGRAVITY  < Gravitational forces between particles

# Gravity options
    PMGRID=256  < Particle-mesh algorithm for long-range forces, number is size of grid

# Floating point accuracy
    POSITIONS_IN_32BIT  < 32 bit precision positions. Important for memory use
    DOUBLEPRECISION=2  < mode of use of single/double precision internally
```

```
# Group finding
    FOF  < friend-of-friends group identification algorithm before each output

# Miscellaneous code options
    POWERSPEC_ON_OUTPUT  < matter power spectrum calculation before each output

# IC generation via N-GenIC
    NGENIC=256  < Initial condition generation, number is the grid for IC generation
    IDS_32BIT  < Precision of IDs. Needs to be larger for VERY LARGE simulations
```

param.txt options

The parameter file contains the parameter that are read-in at startup. This means they can be modified after compilation. Here again a selection of the parameters:

```
%----  Relevant files
OutputDir            ./output < output directory relative to execution directory
SnapshotFileBase     snapshot < filename of snapshot files
OutputListFilename   outputs.txt < file containing the output times


%---- File formats
SnapFormat           3  < format 3 is hdf5 files, which should be the standard

%---- CPU-time limits
TimeLimitCPU              86400 < maximum runtime of the code in s (can be restarted!)
CpuTimeBetRestartFile     7200 < frequency of a 'restart' file in s

%----- Memory alloction
MaxMemSize                1800 < max. memory allocation per MPI task in MB

%---- Caracteristics of run
TimeBegin                 0.015625 < scale factor at start of the simulation, z = 63
TimeMax                   1.0 < scale factor at end of the simulation,   z = 0

%---- Basic code options that set the type of simulation
ComovingIntegrationOn     1 < cosmological integration; time is scale factor

%---- Cosmological parameters
Omega0                   0.308  < Matter density today relative to critical
OmegaLambda               0.692  < Cosmological constant relative to critical
OmegaBaryon               0.0482 < Baryon density today relative to critical
HubbleParam               0.678 < Hubble parameter 'h'
BoxSize                   50.0 < size of simulation volume in code units / h

%---- Output frequency and output paramaters
OutputListOn              1 < we are using the output list
NumFilesPerSnapshot       1 < can split snapshots into multiple files for large sims
MaxFilesWithConcurrentIO  1 < limit the number of files written at a time for large sims

%---- System of units < code units defined by Length, Mass and Velocity
UnitLength_in_cm         3.085678e24        ;  Mpc / h
UnitMass_in_g            1.989e43           ;  1.0e10 Msun / h
UnitVelocity_in_cm_per_s 1e5                ;  1 km/sec

%---- Gravitational softening length < particle softening lengths of n-body algorithm
SofteningComovingClass0      0.04           ; 40 kpc/h
SofteningMaxPhysClass0       0.04
```

```
%----- N-GenIC
NSample          32 < number of particles per dimension in initial conditions
GridSize         32 < should be GridSize=NSample
```

Running the simulation

Once you have Gadget-4 compiled, i.e. the file `Gadget4` exists in your code directory, you can run the simulation using

```
> mpiexec -np 4 ./gadget4/Gadget4 param.txt
```

The simulation will take a few minutes. If the simulation executes correctly, it will end with

```
...
endrun called, calling MPI_Finalize()
bye!
```

The simulation creates an output in the directory `output` (or whatever was specified in the `param.txt` file). Listing the files in the output direcetory might look the following:

```
> ls output
balance.txt      fof_tab_002.hdf5       restartfiles
cpu.csv          growthfac.txt          snapshot_000.hdf5
cpu.txt          hydro.txt              snapshot_001.hdf5
density.txt      info.txt               snapshot_002.hdf5
domain.txt       inputspec_snapshot.txt timebins.txt
energy.txt       memory.txt             timings.txt
fof_tab_000.hdf5 parameters-usedvalues  variance.txt
fof_tab_001.hdf5 powerspecs
```

possibly with a lot more fof tables and snapshot files (depending on the specification in outputs.txt). The `snapshot...` files contain the simulation (particle) data. On top of this, Gadget-4 performs a number of common analysis algorithms prior to output automatically. Among them is a halo-finding algorithm. Its output is in the `fof_tab_...` files. On top of this, a power spectrum analysis is also performed and stored in `powerspecs`.

3.3 Visualization

If the simulation did not work, copy the output data from the repository

```
> cp -r ../CosmoComputingSchool/cosmobox_nbody/output ./
```

There is also the last snapshot of a higher resolution (128^3 particles) version in

```
../CosmoComputingSchool/cosmobox_nbody/output128
```

The first thing to do after running a simulation is to inspect the results visually. While this seems obvious, this step can already pose significant challenges, in particular for very large simulations. For cosmological N-body simulations, the obvious thing to plot is the position of the simulation particles. For state of the art simulations, however, the number of simulation particles vastly exceeds the number of dots that can reasonably be plotted on a canvas, thus plotting a random subset is reasonable. For our small test simulation, however, it is not a problem to create a scatter plot with the positions of all simulation particles. To create plots of the positions of the simulation particles, execute

```
> python ../CosmoComputingSchool/cosmobox_nbody/analysis_scatter.py ./output/
```

which will create position-position scatter plots of all snapshots in

```
output/plots
```

and merge them together to a movie (requires `ffmpeg`) in

```
./output/movie_cosmobox.mp4
```

Have a look at them. The redshift 0 snapshot (number 62) is shown in Figure 2.

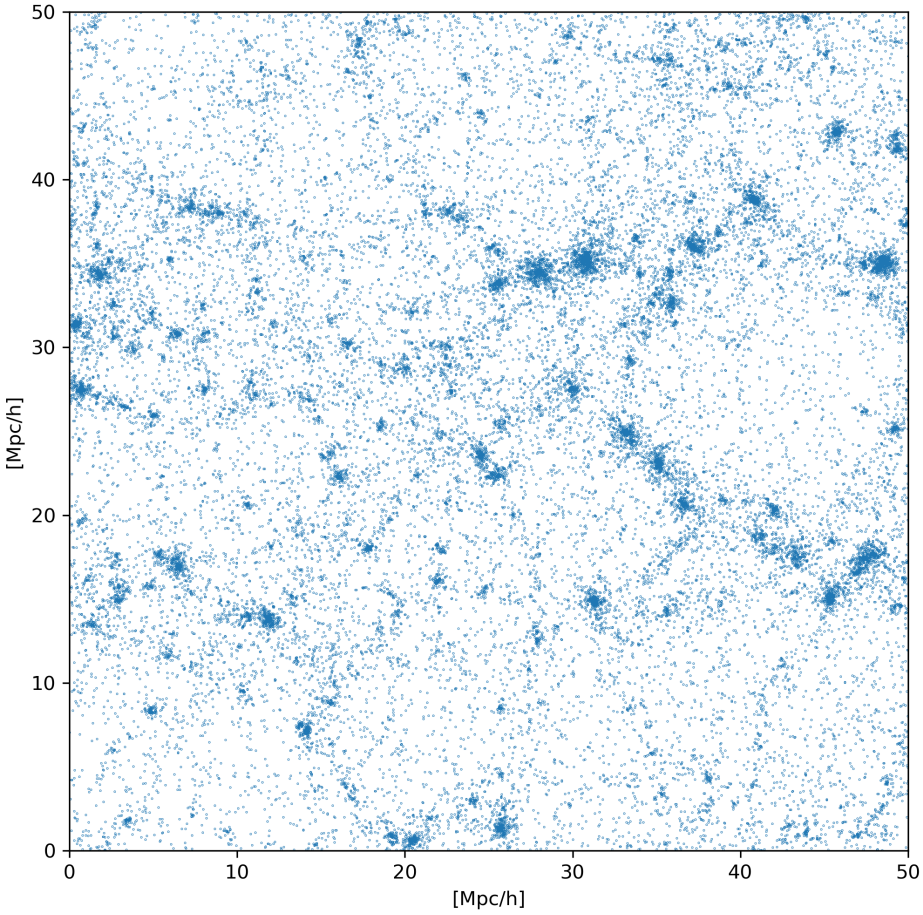

Figure 2: Scatter plot of x-y positions of all simulation particles at $z = 0$.

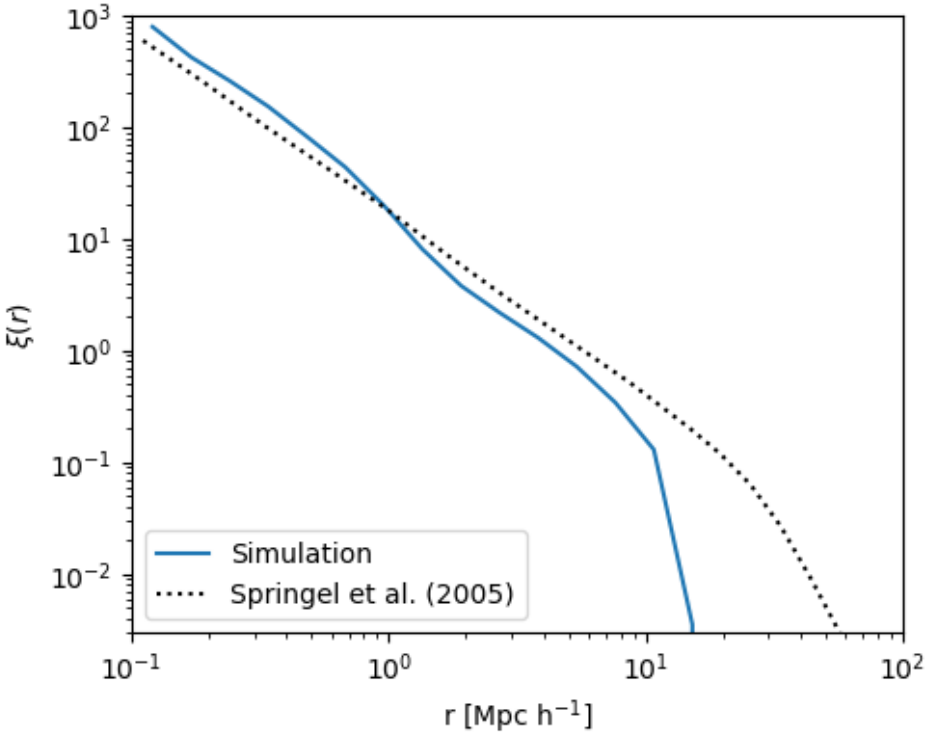

Figure 3: Two-point correlation function of the simulation and of the Millennium simulation as a reference at $z = 0$.

Projections from particle distributions

While simply creating a position-position scatter plot is useful to get an idea what the simulation is doing, it is not particularly scientifically useful. In order to obtain physically meaningful quantities, it is important to understand how a field can be estimated from a particle distribution. In the following we project first the column density and then for the dark matter velocity dispersion.

```
> python ../CosmoComputingSchool/cosmobox_nbody/projection_density.py ./output 62
```

```
> python ../CosmoComputingSchool/cosmobox_nbody/projection_veldisp.py ./output 62
```

Optional tasks:

- Copy the analysis script for velocity dispersion and calculate other fields: ρ^2, kinetic energy density $\rho v^2/2$ etc.
- Run density projection script on higher resolution output; look at result; figure out what went wrong by looking at the script and correct it.

3.4 Quantitative analysis of the simulation results

A common way to quantify the matter clustering is the two-point correlation function $\xi(r)$.It measures how likely particles are to be separated by a distance r compared to a random distribution.

- $\xi(r) = 0$: random distribution
- $\xi(r) > 0$: excess pairs; clustering
- $\xi(r) < 0$: fewer pairs

A simple estimator is:

$$\xi(r) = \frac{DD(r)}{RR(r)} - 1$$

with $DD(r)$ and $RR(r)$ the normalized pair counts from data and random samples. A simple script to perform this analysis is given:

```
> python ../CosmoComputingSchool/cosmobox_nbody/correlation_function.py ./output 62
```

The resulting plot is shown in Figure 3.

Optional task:

- Run a simulation with box size to 250 Mpc h^{-1} and plot the correlation function. What is the difference and why?

4 Structure formation with gas and dark matter

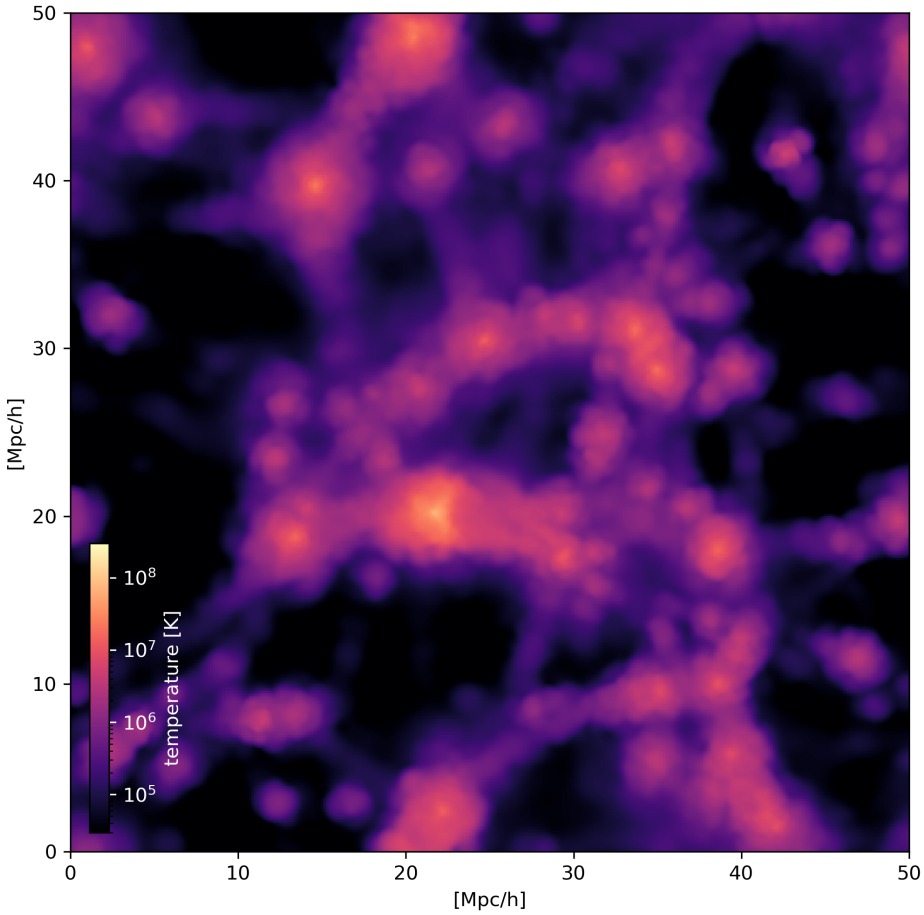

Figure 4: Temperature projection of gas of a cosmological hydrodynamics + N-body simulation.

Definition of the Problem

In this example we are simulating structure formation in the Universe for dark matter and gas. The initial conditions are created with using the matter power spectrum at redshift $z = 127$ and the Zel'dovich approximation to calculate the initial displacement and velocities of otherwise uniformly distributed simulation particles. Different from the pure N-body example, the particles are split into dark matter particles and gas cells, with the relative mass according to the cosmic baryon to matter fraction. The subsequent evolution is also governed by different equations: While dark matter only interacts gravitationally, i.e. only accelerations are calculated, gas dynamics is described by the Euler equations, which are solved separately.

In this case, using the Arepo code, the Euler equations are solved using a finite-volume approach on a moving, unstructured mesh. This technique has proven to be very advantageous in cosmological galaxy formation simulations.

Learning goals

1. Learn how to run cosmological N-body simulations with gas.
2. Learn how to use the Arepo code.
3. Understand the different behaviour of gas and dark matter.

4.1 Simulation setup

Make sure you are in the directory with a downloaded repository

```
> ls
CosmoComputingSchool
```

Create a simulation directory and change directory into it.

```
> mkdir sim_cosmobox_hydro
> cd sim_cosmobox_hydro
```

Get code

```
> git clone https://github.com/rainerweinberger/arepo.git arepo
```

Copy configuration, parameter and initial condition files

```
> cp ../CosmoComputingSchool/cosmobox_hydro/ics.hdf5 ./
> cp ../CosmoComputingSchool/cosmobox_hydro/param.txt ./
> cp ../CosmoComputingSchool/cosmobox_hydro/output_list.txt ./
> cp ../CosmoComputingSchool/cosmobox_hydro/Config.sh ./arepo/
```

Compile Arepo

Move to the arepo directory

```
> cd arepo
```

This time, instead of using a file called `Makefile.systype`, we employ an alternative by setting an environment variable

```
> export SYSTYPE="macOShomebrew"
```

or alternatively "Ubuntu", depending on your system. To compile Arepo, type

```
> make
```

and change to one directory up

```
> cd ..
```

4.2 Main parameters and running

Running the simulation

```
> mpiexec -np 4 ./arepo/Arepo param.txt
```

4.3 Analysis

Similarly to the N-body example, we first plot the positions of the simulation particles as a scatter plot.

```
> python ../CosmoComputingSchool/cosmobox_hydro/analyze.py ./output/
```

In addition to the scatter plot, the script also draws circles around groups of particles that were identified as halos, with the radius matching the size of the halo ($R_{200,c}$, to be precise). Having information about the halos, it is also possible to bin them as function of their mass, and plot a so-called halo mass function, as shown on the top right plot.

For a gas temperature projection (Figure 4), run

```
> python ../CosmoComputingSchool/cosmobox_hydro/projection_temperature.py ./output/
```

Comparing the scatter plot and the temperature projection, it seems like the temperature is correlated with halos. To quantify this, we plot the halo mass with the maximum temperature reached in the respective halo

```
> python ../CosmoComputingSchool/cosmobox_hydro/temperature_halomass.py ./output/
```

The resulting figure can be seen in Figure 6.

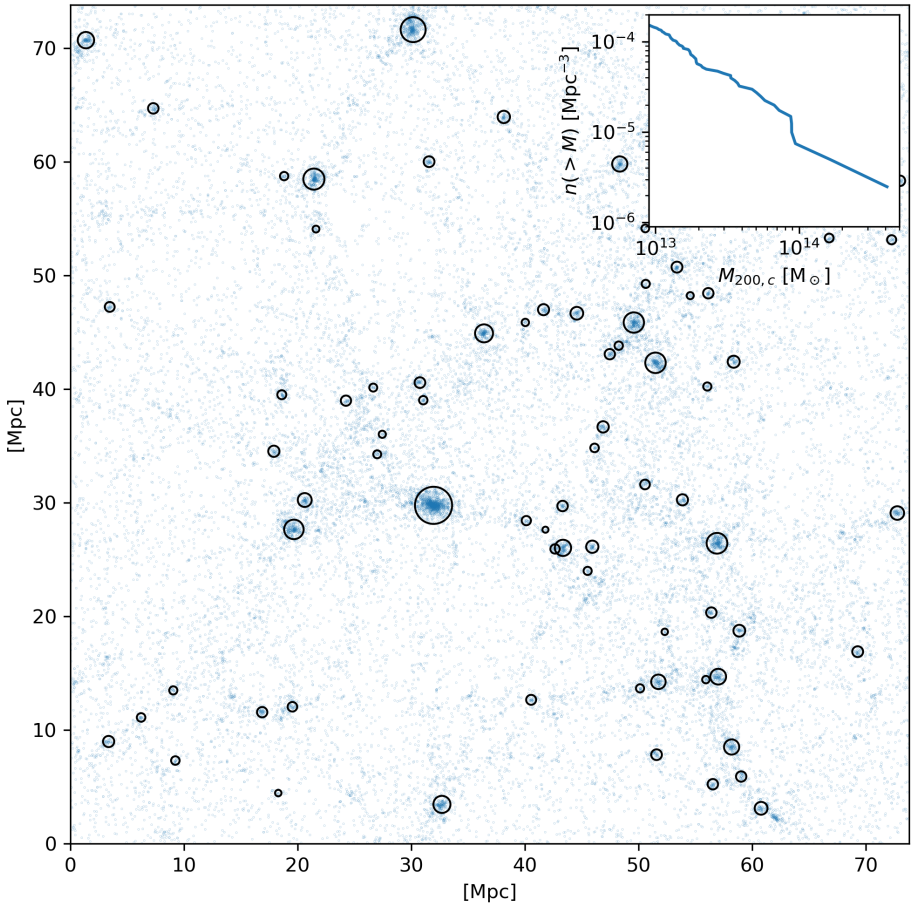

Figure 5: Scatter plot of x-y positions of dark matter simulation particles at $z = 0$. The circles denote halos identified in the snapshot. The inlay shows the corresponding halo mass function.

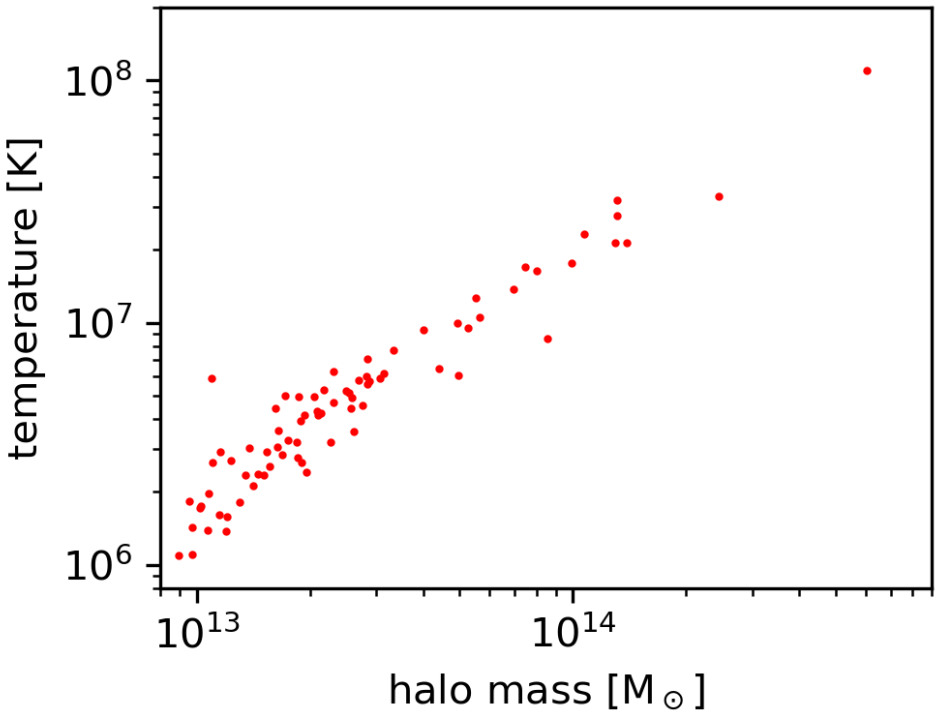

Figure 6: Scatter plot of maximum gas temperature vs. halo mass of the simulation.

References

M. Vogelsberger, F. Marinacci, P. Torrey, and E. Puchwein, Nature Reviews Physics **2**, 42 (2020), arXiv:1909.07976 [astro-ph.GA] .

V. Springel, R. Pakmor, O. Zier, and M. Reinecke, **506**, 2871 (2021), arXiv:2010.03567 [astro-ph.IM] .

V. Springel, **401**, 791 (2010), arXiv:0901.4107 [astro-ph.CO] .

R. Weinberger, V. Springel, and R. Pakmor, **248**, 32 (2020), arXiv:1909.04667 [astro-ph.IM] .

R. Feldmann and R. Bieri, arXiv e-prints , arXiv:2507.08925 (2025), arXiv:2507.08925 [astro-ph.GA] .

H. Mo, F. C. van den Bosch, and S. White, *Galaxy Formation and Evolution* (2010).