GADGET-4-CPPC documentation

Authors: J. Z. Chen, M. R. Mosbech,
G. Pierobon, A. Upadhye, Y. Y. Y. Wong

October 10, 2024

GitHub repository: github.com/cppccosmo/gadget-4-cppc
Papers describing the physics: [2011.12504], [2011.12503], [2210.16012], [2410.05816]

This is a brief technical documentation regarding the usage of the `gadget-4-cppc` code, including the compile-time and run-time additions to the base `gadget-4` code. All the relevant information regarding the physics can be found in the papers listed above. In the following, config options are defined in capital characters, while run-time parameters are specified in lower-case letters.

Compile-time additions

- `N_TAU`: This option sets the number of flows, historically defined as N_τ, used for the MFLR and hybrid runs.
- `N_MU`: This option sets the number of Legendre multipole moments N_μ included in the MF equations, used for the MFLR and hybrid runs.
- `MFLR_RST`: This is an option for both linear response and hybrid runs, therefore does not require `ADDITIONAL_GRID` to also be chosen. If chosen, the multi-fluid linear response perturbation module will not compute a new set of perturbations at the start of the simulation, and instead look for a `.dat` file containing the information. This is necessary if a simulation containing multi-fluid neutrinos is restarted for any reason. The usual initialisation procedure will be completely linear and thus lose any response to non-linear potential the neutrinos would have accumulated prior to the restart. The `allvars.cc` parameter input reading will therefore expect a new parameter, `ynuFile`.

- `ADDITIONAL_GRID`: This is the switch for turning on the hybrid mode. It tells the code that it should expect a snapshot to be read in as IC, but also still run the `NGENIC` routine as more particles are to be inserted. In the stock version of gadget-4, `NGENIC` and reading a snapshot IC are conflicting options. However, in the hybrid version, `NGENIC` needs to be switched on for the creation of neutrinos/HDM alongside `ADDITIONAL_GRID`. On the other hand, this option will conflict with the `CREATE_GRID` option, so *do not turn both*. `CREATE_GRID` is used at the start of a linear response or CDM only simulation, while `ADDITIONAL_GRID` is only used for restarted hybrid runs.

  The `allvars.cc` parameter input reading will expect all the usual `NGENIC` parameters and in addition two new parameters: `N_tau_part` and `Nu_part_deg`.

- `THERMAL_VEL_IC`: This is a necessary option when `ADDITIONAL_GRID` is chosen. The `NGENIC` routine will add a randomly drawn zeroth-order thermal velocity to each 'new' particle converted. A new run-time parameter, `stream_vel` is therefore expected.

- `CB_PHASE`: This is an additional option when `ADDITIONAL_GRID` is chosen. The `NGENIC` routine will not generate new random numbers for the 3D phase of the particles if this is activated. Instead, it will look for a `.dat` file containing the snapshot's phase information. The phase of all matter N-body particles in the snapshot (including existing HDM particles) will be used to initialise the 'new' HDM particles. The `allvars.cc` parameter input reading will expect two new parameters: `deltagridFILE`, and `CBPowerSpectrumFile`.

- `HDM_POWER`: This option allows the user to compute the full power spectrum for non-CDM species such as neutrinos/HDM. The code removes the contribution of CDM (`type1`) particles and creates and additional file on the output `powerspecs` folder, called `powerspec_hdm_XXX.txt`.

Run-time additions

- `OmegaNuPart`: A floating point number. The density parameter of neutrinos that are in the representation in the simulation. `OmegaNuPart` + `OmegaNuLin` (see below) should add up to the physical total $\Omega_{\nu/\mathrm{hdm}}$ in the cosmology specified. When converting neutrinos/HDM from multi-fluid perturbations to N-body particles, a fraction of `OmegaNuLin` will need to be transferred to `OmegaNuPart`.

  The code will not check this for you, if you enter it wrong, the neutrino N-body particle masses will be off. This might be changed in the future.

- `OmegaNuLin`: A floating point number. The density parameter of neutrinos that are in multi-fluid perturbative representation in the simulation. See above for precautions to take regarding this parameter when running in hybrid mode. In linear response mode this is equal to $\Omega_{\nu/\mathrm{hdm}}$.

- `MassHDM`: A floating point number specifying the neutrino/HDM mass in eV. For effective HDM distributions, this corresponds to the effective mass.

- `NLR`: An integer that indicates the linear response type modification in Poisson equation

  - `NLR`=1: SuperEasy linear response, introduced in [2011.12504].
  - `NLR`=2: MultiFluid linear response, introduced in[2011.12503].
  - `NLR`=3: Generalised SuperEasy linear response, introduced in [2410.05816].

- `CBPowerSpectrumFile`: A `.txt` file formatted as `%g %g`. The first column is the wavenumber k in h/Mpc. The second column is the power spectrum $P(k)$ in $(\mathrm{Mpc}/h)^3$. The information corresponds to the estimated power spectrum of *all* existing N-body particles in the simulation prior to the hybrid restart. It is used to compute the 3D phase for the 'new' batch of neutrino/HDM particles to be converted. The output from the previous paused simulation `powerspec_XXX.txt` contains the exact information needed. Note, if neutrinos are already present, there will be several outputs of `powerspec_typeN_XXX.txt`, corresponding to power spectra of that specific batch of particles.

- `GrowthRateFile`: A `.txt` file formatted as `%g %g`. The first column is the wave number k in h/Mpc. The second column is the growth rate. The information is necessary for linear response and hybrid mode of running. The inclusion of massive neutrinos implies the growth factor for CDM+baryons and the neutrinos themselves are k-dependent. The initialisation of the velocity field can no longer be derived from the density field using a multiplicative factor. The `NGENIC` routine would initialise a new θ field for extracting the initial velocity. This for linear response mode is calculated from the `MuFLR` code (or by using the htools automation scripts). In hybrid mode of running, this can be computed for a particular flow using the outputs in the `neutrino_stream_data` output folder. In the folder, the monopole velocity divergence and density contrast of each flow is recorded at the time of a snapshot output.

- `StreamVelListFilename`: A `.txt` file containing a column with floating point numbers, in units of km/s. It is used with the `THERMAL_VEL_IC` option, and the velocities are randomly selected to set the magnitude of the velocity assigned to each new HDM particle. The `NGENIC` routine will only draw a unit vector for direction, which is then rescaled one the values in magnitude. The velocity is unique for each neutrino fluid flow in the multi-fluid decomposition, so when converting a single flow only one velocity value needs to be provided.

- `N_tau_part`: An integer number specifying the total number of neutrino/HDM N-body particle types there are in the simulation, including the current batch being converted. In the stock `gadget-4` implementation, the number of particles is set by user by providing a value for the `NTYPES` compile-time option. `type0` and `type1` are reserved for hydrodynamic particles and CDM+baryons, respectively, while `type2` and higher are free to be assigned HDM particles.

- `Nu_part_deg`: An integer number. It specifies how many multi-fluid flows are in each of the batch specified by `N_tau_part`. There are occasions where in one conversion several flows are averaged into one representative flow to be tracked by a single batch of N-body neutrino particles, however with an efficient momentum binning scheme, such Gauss-Laguerre, it is recommended to convert one stream per particle type, therefore setting `Nu_part_deg` to 1. The user needs carefully assign this value, as it is used to exclude the correct number of flows from the linear response perturbation module to avoid double counting.

- `ynuFile`: A `.dat` file formatted in binary. The content contains all the perturbation information of the multi-fluid linear response corresponding to a snapshot. One such file is written every time a snapshot is outputted. The information is used by the code to restart the multi-fluid module without re-initialising the neutrino perturbations. It is important for all restarted runs in MFLR and hybrid mode.

- `deltagridFILE`: A `.dat` file formatted in binary. The content contains the 3D density contrast array of *all* the N-body particles present in the simulation corresponding to a snapshot. One such file is written every time a snapshot is outputted (more specifically, when a power spectrum output is written, as it is part of the power spectrum estimation routine). The information is used in the `NGENIC` routine to extract the 3D phase of the snapshot. The neutrino particles are then initialised with that exact phase, rather than redrawing the random numbers. This is technically redundant information, since the snapshot is read in as part of the hybrid restart IC anyway. The choice to double up on the input is purely for coding convenience, as it is much easier than trying to extract the phase from the snapshot during the `NGENIC` routine. This might be changed in the future.

Output folder

- `neutrino_stream_data`: A pair of `.csv` files written whenever a snapshot is outputted. The two files are the delta and theta of each individual neutrino flow respectively. The file comes with $1 + N_\tau$ number of columns, and N_k number of rows. The first column is the wavenumber k, and then the remaining columns (comma separated) are the delta or theta information of that particular neutrino flow. The file writing routine is in the `pm_periodic.cc` file where the multi-fluid linear response calculation is done. These files are outputted in MFLR and hybrid mode.

- `y_nu`: A `.dat` binary file written whenever a snapshot is outputted. The file comes as one single line, where the entire multi-fluid linear response perturbation array is encoded in binary and written in. The file is only to be used when restarting the simulation and the `MFLR_RST` config option is chosen. Multi-fluid initialisation routine will not compute new linear perturbations to populate the `y_nu` array, but instead read this file and transfer the data over.

- `delta_grid`: A `.dat` binary file written whenever a snapshot is outputted. The file comes as one single line, where the first 8-bytes are reserved for a single `long int` which encodes how long the rest of the file is. The remaining information are 8-byte floats that contain the 3D density contrast information of the snapshot in row-major format. The file is to be read-in by the NGENIC routine if the `CB_PHASE` config option is chosen. The neutrino/HDM particles will decode the 3D phase in this file and follow it instead of generating new random numbers.

Automation tools

The folder `htools` and the script `hsetup.sh` provide an automated pre-processing of a typical simulation and handle the MF to hybrid conversions consistently. This is particularly helpful in the case of hybrid simulations, where several executable files and input data files are needed to restart the simulation. A typical simulation involves the use of three different codes: `class` for the linear initialisation, `MuFLR` for the neutrinos/HDM fluid perturbations and a modified version of `gadget-4` for the N-body run, hence it is vital that in all the stages a consistency in the cosmological and simulation parameters is respected. The `hsetup.sh` script is the starting point, where you can setup a given simulation by running the command

```
bash hsetup.sh </path/to/sim/folder> <system>
```

This will create a directory in `/path/to/sim/folder` and interactively ask the user if a MF restart or a hybrid simulation needs to be performed, building the right executable files. The `<system>` command selects the modules that need to be loaded depending on the HPC system. So far, two systems are included (HPC facilities at UNSW and NCI Australia), however a new entry can be added in the `load_modules.sh` file.

Once in the simulation folder, the user can find the `htools` script that can be used for pre-processing of all linear response simulations. In the file the user can select one of the cosmological models defined in the file `scripts/cosmologies.sh` and set other parameters such as N-body initialisation time, etc. By calling

```
./htools start
```

the script runs `class`, will uses the outputted linear transfer function to initialise and run the `MuFLR` code, which eventually sets the HDM perturbations at the simulation initial time (including the scale-back procedure). The final output is a creation of the IC files needed to start the simulation, and the automatic generation of the `param.txt` file, with the correct values for all the parameters, according to the cosmology selected.

The command

```
./htools restart
```

will instead perform a MF to hybrid conversion pre-processing step and automatically generate a `param_restart.txt` file according to the options specified in the master `htools` script. The hybrid simulation can be then be run by calling the right executable and param files, such as

```
mpirun -np <n_procs> ./Gadget4-hybrid-restart param_restart.txt
```

Example scripts are provided in the `htools/jobs` folder of the source code.

How the neutrinos/HDM are added

The goal is to perform the neutrino/HDM calculation entirely outside of the `gadget-4` architecture. The connection between the non-linear N-body potential and the linear perturbative calculation is an interface that is independent of the neutrino/HDM method. The primary interaction point between neutrinos/HDM and `gadget-4` is through the particle-mesh (PM) force routine. This interface is in fact completely modular and is the same across all linear response methods. In our implementation we include two additional arrays of the same size, one indexing the wavenumber magnitude $k = |\vec{k}|$, the other containing the corresponding *modification factor* $\tilde{g}(k, s)$, included in Poisson equation

$$k^2\Phi(\vec{k}, s) = -(3/2)\mathcal{H}^2(s)\Omega_{\rm cb}(s)\tilde{g}(k, s)\delta_{\rm cb}(\vec{k}, s),$$

at each k. The size of the arrays is therefore determined by the simulation k-range, set by the `PMGRID` config option. The modification factor becomes one for zero neutrino mass.

Another important neutrino related modification is in the Hubble parameter. This technically is also agnostic to the neutrino/HDM method, since the background evolution can be assumed to either be fully non-relativistic matter, or a transition between radiation and matter of varying degrees of sophistication. Overall, if the same implementation of the Hubble parameter is included in all stages (IC generation, linear response, N-body evolution), the exact method is not important. In the case of our code, the Hubble parameter is calculated by the multi-fluid module. The N-body code centralises the Hubble parameter in `driftfac.h`, and a simple redirection from there to source the parameter from `neutrino.cc` is sufficient in unifying the Hubble calculation.

The multi-fluid linear response method requires a sizeable storage in memory for the perturbation arrays. There are in total

$$(2N_\tau N_\mu + 2)N_k$$

number of floating-point numbers to store for the duration of the simulation. This array is internally called `y_nu` and needs to be placed somewhere, and it can't be where the PM routine is located. This is due to the way `gadget-4` dynamically allocates memory in an ordered stack. Any long-term storage needs to be at the bottom of the stack to not get in the way of the upper layers being freed if they need to be. Our approach is to then declare the array as a *public member* of the `Nulinear` class, since classes are instantiated alongside all other global objects at the beginning of `main.cc`. This ensures the memory allocation is at the very beginning of the stack.

The calculation of the multi-fluid evolution during the PM step is MPI-parallelised. The calculation takes place right after the N-body density fluctuations are placed onto the PM grid via CIC interpolation and transformed into k-space, but before the Poisson equation coefficients are multiplied. The routine declares the arrays for the modification factor and k-index and a loop over the k-range is performed. However, each process only loops through their *local* range of k-modes. The partitioning of the N_k modes is efficient for any value of N_k and CPU cores N_{cores}, as long as $N_{\mathrm{cores}} < N_k$. Within the loop, an array called `y_tmp`, sized $(2N_\tau N_\mu + 2)N_k^{\mathrm{local}}$, will temporary store the corresponding set of modes that is held by the global array `y_nu`. The reason for this is due to the multi-fluid module's equation of motion function takes only one k-mode worth of info at a time. The `y_tmp` array is then fed into the neutrino/HDM class function `evolve_step` defined in `neutrino.cc`. The perturbation data in `y_tmp` is then updated to the current timestep in-place. After the multi-fluid evolution is done over the timestep, the N-body particle power spectrum is computed. The density δ_{parts} is then updated into the last component of the `y_tmp` array. Note that in pure MFLR simulations δ_{parts} corresponds to $\delta_c b$, however in hybrid mode, if some HDM flows have been converted, the δ_{parts} will include the HDM particles. Either way, δ_{parts} is the quantity that is used for the 'response' part, where the non-linear information info is fed to the neutrinos/HDM.

The order of operation here is worth emphasising. The exact sequence must be done as follows:

- Fetch neutrino/HDM perturbation data from the global `y_nu` array.
- Feed the neutrino perturbations into multi-fluid to evolve by one timestep.
- Update the CDM+baryon(+HDM particles) density in the temporary local neutrino array.
- Compute the modification factor.
- Replace the array's updated information back into the global `y_nu` array.

The reason for this order is that the neutrino/HDM information is always one timestep behind the actual PM timestep being currently computed. Therefore, we need to update the neutrino info up to the current timestep before updating the CDM+baryon info, and finally compute the modification factor once everything is in sync. During timesteps where the snapshot is written, the `pmforce_periodic` function in `pm_periodic.cc` is called with a different argument (`mode` > 0). When this happens, the pm force is not calculated, but the density grid is still constructed for the purpose of calculating the power spectrum. The code exploits this functionality and calculates things like neutrino/HDM δ and θ arrays and output them all into files as described above. For methods such as SuperEasy and Generalised SuperEasy, the entire process simplifies to just calculating the modification factor array every timestep. Ultimately, the difference between the methods is just how much other auxiliary functions are required for getting the modification factors.