METROC++

A parallel code for the computation of merger trees in cosmological simulations

Edoardo Carlesi
Leibniz Institut für Astrophysik
Potsdam, Germany
ecarlesi@aip.de

January 9, 2019

Contents

1 Introduction **2**

2 Basic setup **2**
2.1 Compiling the code 2
2.2 Code operation modes 3
2.3 Configuration file 3
2.4 Temporary files 3

3 Advanced features **3**
3.1 Code structure 3
3.2 Compatibility with other halo finders 4

4 Examples **4**
4.1 Full box simulation 4
4.2 Zoom simulation 4

1 Introduction

METROC++ is an acronym that stands for **ME**rger**TR**ees **O**n **C++**, and refers to the infamous **Metro C** subway line in Rome, where the author of the code grew up and lived for more than 20 years.

This legendary piece of infrastructure (which is still - in 2019 - largely under construction, though a shorter section of it is already operating) took something of the order of a few Giga-Years to be built, a time span which can be compared to the formation time of most dark matter halos. The name of the code is a tribute to this mythological pillar of the Roman public transportation system, which connects through space different points of the city just like a merger tree connects a Halo through different points in time.

The code is distributed under the terms of the GNU Public License, and can be freely downloaded from this GitHub repository.

METROC++ has been written in C++ and relies on MPI2.0 C-bindings for the parallelization.

Since C++14-style syntax is used at some points, in particular for the manipulation of (ordered) maps, the source code needs to be compiled with a non-prehistoric version of the compiler. A series of python and bash scripts for the post processing, analysis and visualization of the merger trees are also provided together with the sources, inside the scripts/ and python/ subfolders.

The structure of the algorithms of the code as well as its performance, cosmological applications and physical modelling are described in the: code paper (Carlesi, MNRAS, 2018).

In this user's guide, we explain the basics for the setup and running of METROC++

2 Basic setup

2.1 Compiling the code

The main folder contains a Makefile.config file

-DCMP_MAP: This flag controls the core algorithm for the comparison of the particle content of the different halos. **It is recommended to always use this flag** unless you have very strong reasons for limiting the memory usage at the expense of speed and performance. When this flag is switched on, the particles on each task are stored in a C++ map, speeding up the comparison process by (at least) a factor of 5, reaching up to a factor of 20 for very large datasets. Hash functions can do miracles sometimes. However, maps come with a substantial amount of overhead, and the total memory required on each task is

around three times the memory required when running the standard, slow and boring algorithm.

`-DZOOM`: When this flag is switched on, the comparison algorithm is optimized for the comparison of the particle content of halos in zoom-in simulations. This means that only one particle type will be

`-DVERBOSE`: This flag controls the output of the program - when enabled, the code will dump a lot more of (boring) information at runtime, on inter-task communication, buffer sizes, number of particles and halos exchanged and so on.

`-DNPTYPES=N`

2.2 Code operation modes

The code has two basic modes of operation: *tree-building* and *post-processing*.

2.3 Configuration file

2.4 Temporary files

The code produces a number of temporary files (`.tmp` format extension, located in the `tmp/` folder). These files are needed at runtime to They can be produced manually or using the scripts (`find_z.sh`, `find_n.sh` located in the `scripts/` folder) and contain the list of files on which the halo finder will run on. This can be the full list of snapshots in one simulation, or can be edited to be only a subset of it.

3 Advanced features

3.1 Code structure

- `main.cpp` A wrapper for the functions determined elsewhere
- `utils.cpp` General functions and utilities are implemented here
- `spline.cpp` The spline class is used for interpolation
- `global_vars.cpp` A list of global variables accessible throughout the whole program
- `MergerTree.cpp` This file contains the merger tree class, that tracks the merging of the halos, and series of functions (FindProgenitors) that compute the trees themselves.

- `IOSetting.cpp` Input/Output settings
- `Cosmology.cpp` Everything related to cosmological calculations, gravity solver to reconstruct the orphan halo positions and velocities
- `Communication.cpp` Handles most of the communication among tasks - sending / receiving buffers and so on
- `Grid.cpp` This handles the grid on which halos are placed and the buffers
- `Halo.cpp` The halo class contains the main halo properties

3.2 Compatibility with other halo finders

Although `METROC++` was conceived and mainly tested using the `AHF` halo finder, it can be easily extended to support other software as well, as long as:

- Halo catalogues include informations about the number and types of particles, positions and velocities for each object
- Particle catalogs contain the (unique) IDs for each halo particles' content

To add

4 Examples

4.1 Full box simulation

To properly run the code for full box simulations, the `-DZOOM` flag needs to be commented in the `Makefile.config` file.

4.2 Zoom simulation

To properly run the code on zoom simulation, the `-DZOOM` flag needs to be switched on in the `Makefile.config` file.