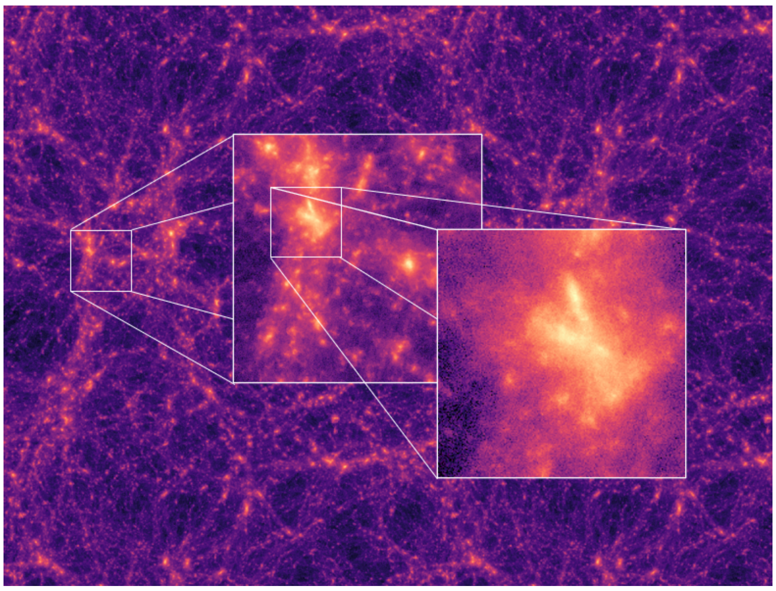
Cosmological Structure Formation with the Performance Portable IPPL Library

Blanca Crazzolara

Supervisor: Dr. Andreas Adelmann
Scientific Advisor: Sonali Mayani



Semester Thesis

Accelerator Modelling and Advanced Simulations Group
Paul Scherrer Institute, ETH Zurich
Switzerland
September 15, 2024

Abstract

This work presents a cosmological simulation code for the ΛCDM model with collisionless dark matter using a high-performance `C++` numerical framework. The code can be used for large-scale cosmological simulations focusing on the formation and evolution of dark matter structures. It is based on existing mini-apps for plasma physics applications using the electrostatic Particle-In-Cell scheme from the performance portable library IPPL. The forces are calculated using the Particle-Mesh algorithm based on Fourier techniques. The time evolution is implemented using the Leapfrog integration scheme.
A scaling study was conducted for CPUs and GPUs. The code has been successfully tested on different computer clusters with 16^3 up to 512^3 particles. A simulation of 512^3 particles with 500 time steps (no output saving) on 8 CPU nodes took $1.33 \times 10^4\,\mathrm{s}$ with the solver contributing 1.24×10^4 s (93%). Using 8 GPUs on one node for the same simulation setup took 238 s, with the solver contributing 18.5 s (7.8%). The largest simulation run was conducted on the Euler cluster using 16 CPUs on 1 node. It ran for 15h simulating 512^3 particles in a 3D Box of size $50\,\mathrm{Mpc}\,h^{-1}$ with 2000 time steps and 30 outputs from $z = 63$ until today ($z = 0$).
The successful parallelization of the code allows us to run large problems which will contribute to our understanding of the large structure formation in our Universe. The code is publicly available on GitHub [1].

[1] https://github.com/bcrazzolara/ippl/tree/cosmology

Contents

List of Figures

List of Tables

1 Introduction

In many physics research areas, numerical simulations have become an important part. Often physical systems are too complex to have analytical predictions and the studies require numerical approaches with the aid of computers. Many algorithms that have been developed for simulations can be used across different research areas. This motivates the implementation of general simulation frameworks. The rapid growth of available computer technology allows us now to conduct large simulations with high precision in a feasible time. Nevertheless, an efficient way of modeling and computing physical simulations is still crucial.

In modern cosmology, research focuses on understanding the Universe's large-scale structure and evolution. Observations of supernovae and the cosmic microwave background (CMB) have confirmed that the Universe is expanding [19]. The rate of this expansion has led to the prediction of a dark energy component (Λ) which forms together with the cold dark matter (CDM), the ΛCDM model. This model has been developed to describe the structure formation process [15]. Testing cosmological models requires large simulations that compare the evolution of initial conditions to observational data from telescopes. Studying these models helps us to understand the nature of dark matter and dark energy that contribute significantly to the present mass-energy density of the Universe.
The complex description of all gravitational interactions requires computational techniques since no simple analytical solutions exist. Simulation results in the past have helped to characterize fundamental results for cosmology. This includes the non-linear clustering of dark matter [7] and the resulting dark matter halo density profile [12]. Dark matter haloes are the first non-linear objects that formed in the Universe with masses of $10^5 - 10^8 M_\odot$ at redshifts $10 - 30$ [14]. Galaxy cluster with masses up to $10^{15} M_\odot$ formed later on. In order to simulate the structure formation on large scales and simultaneously aim for high resolution of the galaxy formation structure, high computational power is required which can only be provided with exascale computer architectures.
Given the still ongoing development in computer technology, it is expected that simulations will become more and more attractive in the future. Since modern computing systems often show diverse architectures, which include varying nodes, CPUs, and GPUs, there is a need for developing versatile software that can be reused without extensive rewriting.
In response to this need, the publicly available `C++` library IPPL was developed [5]. IPPL is a performance-portable code that has been designed to fulfill these flexibility requirements for plasma physics simulations. IPPL uses Message Passing Interface (MPI) and Kokkos for communication during parallelized tasks and can be run on CPUs and GPUs. To demonstrate the performance and versatility, the Accelerator Modelling and Advanced Simulations (AMAS) Group at the Paul Scherrer Institute (PSI) has developed a set of mini-apps (Alpine), for various plasma physics applications using IPPL [11].

The main objective of this thesis was to introduce a new mini-app for cosmological simulations, focusing on dark matter dynamics. This feature is motivated by many similarities in the nature of the physical equations in electrostatic and gravity. In both cases, the interacting forces are derived from a potential field, which is obtained from the charge or mass density field. The cosmology mini-app aims to demonstrate the flexibility and portability of IPPL.
The core part of the framework is based on the Particle-Mesh (PM) algorithm, which models the dynamics of individual particles. The method interpolates the particle positions to an underlying grid to compute the density field. The potential field and the force field are then calculated on the grid points and used to approximate the acceleration of the particles. The successful implementation of the cosmology mini-app will allow for running large-scale cosmological simulations.

This thesis is organized as follows: In section 2 the theoretical models for cosmology and structure formation are summarized. Section 3 contains the description of the numerical implementation of the cosmology simulation. This includes the description of the N-GenIC code for generating initial conditions and the IPPL library. The performance of the code and the results are presented in section 4. A summary of the project results and the outlook for future research can be found in section 6.

2 Theoretical Background

Cosmology focuses on the research of the large-scale properties of the Universe. It includes the study of the origin, evolution, and future of the Universe. The important properties considered in cosmology are space-time, influenced by matter and energy. Ongoing research focuses on testing our models that describe the dynamics of the Universe and how we can describe it. It is believed, that the present distribution of galaxies contains information about the past and the future of the Universe. With theory and large-scale simulations, predictions can be verified with observations.

In this project, a cosmological simulation code will be implemented that will study N-body dynamics where the individual bodies represent stars, galaxies, or even entire galaxy clusters depending on the simulation size and precision. In this section, the relevant equations for the cosmological models are summarized. These equations will build the base of the physics part of the cosmology mini-app.

2.1 Expansion of the Universe

Observations of type Ia supernovae and the cosmic microwave background have shown that the Universe is expanding [19] and the expansion is accelerating at the moment. The accelerated expansion suggests that the Universe's composition contains an additional component called dark energy, or Λ which contributes around 70 % to the total mass-energy density of the Universe today [13].
For this project, the standard model of cosmology was used which assumes that the Universe is spatially homogeneous and isotropic and describes the structure of space-time by Einstein's theory of General Relativity [10]. The four-metric in the standard model is described by the Robertson-Walker metric

$$\mathrm{d}s^2 = a^2(t)[\mathrm{d}\tau^2 - \mathrm{d}\chi^2 - f_k^2(\chi)(\mathrm{d}\vartheta^2 + \sin^2\vartheta\,\mathrm{d}\phi^2]$$

with

$$f_k(\chi) = r = \begin{cases} \sin\chi & (k = +1) \\ \chi & (k = 0) \\ \sinh\chi & (k = -1). \end{cases}$$

χ is the comoving geodesic distance, measuring the length of the shortest path between two connected points, and k is the curvature parameter. The expansion is assumed to be uniform in space (only dependent on time), leading only to radial motions [6]. The relative expansion of space is described by the scale factor

$$a(t) = \mathbf{r}(t)/\mathbf{x}(t) \tag{1}$$

where $\mathbf{r}(t)$ is the physical distance and $\mathbf{x}(t)$ the comvoving distance. The scale factor at present is defined as equal to 1. The expansion rate of the Universe is described by the Hubble parameter

$$H(t) = \frac{1}{a(t)}\frac{\mathrm{d}a(t)}{\mathrm{d}t}.$$

Most astronomical observations are done using light signals and it is therefore crucial to understand light propagation in the Universe. The wavelength of the light will be modified through the expansion of the Universe. A light signal that has been emitted at time t_e with wavelength λ_e that is observed at time t_0 will have the wavelength

$$\lambda_0 = \lambda_e \frac{a(t_0)}{a(t_e)}.$$

For observation today ($a_0 = 1$) we define the redshift z as

$$1 + z = 1/a(t_e) = \frac{\lambda_0}{\lambda_e}.$$

Therefore, the present redshift equals zero. In the following, we will often use the redshift as a measure of time instead of the physical time coordinate t. The current value of the Hubble parameter, known as the Hubble constant $H_0 := H(z = 0) \sim 70\,\mathrm{km\,s^{-1}\,Mpc^{-1}}$ describes the expansion rate today. The value is related to the size, age, and dynamics of the Universe.

2.1.1 Friedmann Equations

The Friedmann equations are derived from Einstein's field equations of General Relativity. They relate the scale factor $a(t)$ to the curvature and matter-energy content of the Universe and its evolution. The two Friedmann equations are given below.

The first Friedmann equation is given by

$$H^2 = \frac{8\pi G}{3}\rho - \frac{k}{a^2} + \frac{\Lambda}{3}, \tag{2}$$

where ρ is the energy density of the Universe, G is the gravitational constant, k is the curvature parameter (with values -1, 0, $+1$ corresponding to open, flat, and closed Universes, respectively), and Λ is the cosmological constant representing dark energy.

The second Friedmann equation is given by

$$\frac{\ddot{a}}{a} = -\frac{4\pi G}{3}(\rho + 3p) + \frac{\Lambda}{3} = -\frac{4\pi G}{3}\rho(1+3w) + \frac{\Lambda}{3}, \tag{3}$$

where p is the pressure of the Universe's contents and we used the equation of state parameter $w = p/\rho$ [10]. Approximating the Universe as an ideal gas gives $w = 0$ for matter, $w = 1/3$ for radiation, and $w = -1$ for dark energy.

Combining the two Friedmann equations for a flat Universe ($k = 0$) we arrive at

$$\frac{d\rho}{dt} = -3H(t)(\rho + p)$$

which has the following analytical solution if w is independent of t:

$$\rho \propto a^{-3(1+w)} \tag{4}$$

The density ρ in eq. 3 can be made up of various components. It is common to include non-relativistic matter, radiation, and dark energy/vacuum energy. Their respective energy-mass densities are defined as ρ_m, ρ_r, and ρ_Λ and they follow eq. 4 with their corresponding values for w. These densities scale differently as the Universe expands and we can rewrite eq. 2 as

$$H^2 = \frac{8\pi G}{3}\left[\rho_{m,0}\left(\frac{a_0}{a}\right)^3 + \rho_{r,0}\left(\frac{a_0}{a}\right)^4 + \rho_{\Lambda,0}\right] - \frac{k}{a^2}.$$

The critical density at time t is defined as

$$\rho_{crit}(t) = \frac{3H^2(t)}{8\pi G}$$

and describes the total density of the Universe if it was flat ($k = 0$) and $\Lambda = 0$. It is common to express the component densities with their dimensionless density parameters Ω_i which specify the average mass density of a competent i that is required for a flat Universe ($k = 0$). Specifically, recent observations suggest the following density parameter values:

$$\Omega_{m,0} = \frac{\rho_{m,0}}{\rho_0} \approx 0.3, \qquad \Omega_{r,0} = \frac{\rho_{r,0}}{\rho_0} \approx 10^{-5}, \qquad \Omega_{\Lambda,0} = \frac{\rho_{\Lambda,0}}{\rho_0} \approx 0.7$$

When only considering matter and dark energy, we obtain

$$H^2(t) = H_0^2\left(\Omega_{\mathrm{m},0}\frac{1}{a^3} + \Omega_{\Lambda,0}\right) \tag{5}$$

as a simplification from the first Friedmann equation. This description of the Hubble parameter as a function of the scaling factor, is the the relation that has been used throughout this project.

2.2 Large Structure Formation in the Universe

The galaxies we see today in the Universe formed from initially small perturbations in the cosmic density field. The perturbations grow under the influence of gravity, collapse, and form systems with high masses over time. The first non-linear objects in the Universe formed as dark matter halos of masses $10^5 - 10^8 M_\odot$ at redshifts $z = 10 - 30$ [14]. Massive galaxy clusters are considered extreme cases of structure formation and have been observed with masses up to 10^{15} $\mathrm{M}_\odot$.
To understand the large structure formation processes, first, the linear evolution of small perturbations in the initial uniform density is studied. Let us consider an initial density perturbation field $\delta(\mathbf{x})$ characterized by its dimensionless density contrast

$$\delta(\mathbf{x}, t) = \frac{\rho(\mathbf{x}, t) - \bar{\rho}(t)}{\bar{\rho}(t)} \tag{6}$$

where $\rho(\mathbf{x}, t)$ describes the matter density at position $\mathbf{x}$, and $\bar{\rho}(t) = \langle \rho \rangle \propto a^{-3}$ is the mean matter density of the Universe at time t. Furthermore, we introduce the autocorrelation function $\xi(r)$ (also known as the two-point correlation function)

$$\xi(r) = \langle \delta(\mathbf{x})\delta(\mathbf{x}') \rangle = \frac{1}{V} \int \delta(\mathbf{x})\delta(\mathbf{x} - \mathbf{r})\, \mathrm{d}^3\mathbf{x} \qquad \text{with } \mathbf{r} = \mathbf{x} - \mathbf{x}, r = |\mathbf{r}|.$$

.

The autocorrelation function is the Fourier transform of the power spectrum $P(k)$ defined as

$$\xi(r) = \int \frac{1}{(2\pi)^3} P(k) e^{i\mathbf{k}\cdot(\mathbf{x}-\mathbf{x}')}\, \mathrm{d}^3 k.$$

The power spectrum can be related to the Fourier transform $\tilde{\delta}(\mathbf{k})$ of the density perturbation $\delta(\mathbf{x})$ as

$$\langle \tilde{\delta}(\mathbf{k})\tilde{\delta}^*(\mathbf{k}') \rangle = (2\pi)^3 P(k) \delta_D^3(\mathbf{k} - \mathbf{k}') \qquad \text{with } k = |\mathbf{k}|. \tag{7}$$

Here δ_D^3 is the 3D Dirac delta function and $\tilde{\delta}^*(\mathbf{k}')$ the complex conjugate of $\tilde{\delta}(\mathbf{k}')$. When considering a non-relativistic pressureless and collisionless fluid that dominates the matter content of the Universe (which is the case for dark matter), the evolution of the density field can be described classically with the continuity equation (eq. 8), the Euler equation (eq. 9) and the Poisson equation (eq. 10) [10]:

$$\frac{\partial \delta}{\partial t} + \frac{1}{a} \nabla \cdot [(1 + \delta)\mathbf{v}] = 0 \tag{8}$$

$$\frac{\partial \mathbf{v}}{\partial t} + \frac{\dot{a}}{a}\mathbf{v} + \frac{1}{a}(\mathbf{v} \cdot \nabla)\mathbf{v} = -\frac{\nabla \Phi}{a} \tag{9}$$

$$\Delta \Phi = 4\pi G a^2 \bar{\rho} \delta \tag{10}$$

with $\Phi = \phi + a\ddot{a}\mathbf{x}^2/2$. Spatial derivatives are taken with respect to the comoving coordinate $\mathbf{x}$. The total velocity of a fluid element is given as $\dot{\mathbf{r}} = \dot{a}\mathbf{x} + \mathbf{v}$ (with $\dot{a}\mathbf{x}$ the Hubble flow and $\mathbf{v} = a\dot{\mathbf{x}}$ the peculiar velocity). $\Phi(\mathbf{x})$ is the gravitational potential and $H(z)$ is defined as in eq. 5.

2.2.1 Dynamics of Individual Particles

From the equations above, the dynamics of individual particles can be described. The dark matter content is described as a self-gravitating and pressure-less fluid. If the fluid is assumed to be non-relativistic, Newtonian theory can be used, namely describing the fluid dynamics with the continuity (eq. 8), the Euler (eq. 9) and the Poisson (eq. 10) equation. The resulting equations of motion for an individual particle with index i are given by

$$\frac{\mathrm{d}\mathbf{x}_i}{\mathrm{d}t} = \frac{1}{a^2}\mathbf{u}_i \quad \text{and} \quad \frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t} = -\nabla\Phi|_{\mathbf{x}_i}.$$

Together with the Poisson equation (eq. 10), the equations form a complete description of a closed system.

2.2.2 Initial Conditions

At early times the density field is modeled as a homogeneous and isotropic Gaussian random field with tiny variations. This means that the perturbation $\delta(\mathbf{x})$ (eq. 6) at different points in space follows a Gaussian distribution. More specifically the amplitudes of the Fourier modes (real and imaginary) of $\tilde{\delta}(\mathbf{k})$ are normally distributed around zero, and the absolute value of the amplitude vector following a Rayleigh distribution [8]. These perturbations can be observed as temperature fluctuations in the CMB [14].
During the inflation period, which happened within the first second after the Big Bang, the Universe experienced a rapid exponential expansion. Tiny perturbations were stretched to a macroscopic level.

2.2.3 Early Linear Growth of Small Density Perturbations

At early times, the evolution of the density perturbation can be approximated using linear perturbation theory driven by gravitational instability [14]. The equations 8 and 9 can be simplified in the linear regime assuming that δ and $\mathbf{v}$ are small which gives

$$\frac{\partial \delta}{\partial t} + \frac{1}{a}\nabla \cdot \mathbf{v} = 0 \quad \text{and} \quad \frac{\partial \mathbf{v}}{\partial t} + \frac{\dot{a}}{a}\mathbf{v} = -\frac{\nabla \Phi}{a}$$

Together with eq. 10, we can derive

$$\frac{\partial^2 \delta}{\partial t^2} + 2\frac{\dot{a}}{a}\frac{\partial \delta}{\partial t} = 4\pi G \bar{\rho} \delta.$$

During the matter dominated era with $a(t) \propto t^{2/3}$ we obtain a decaying mode $\delta_- \propto t^{-1}$ and a growing mode $\delta_+ \propto t^{2/3}$ [10]. The density perturbation at the time of the scaling factor a at a position $\mathbf{x}$ is then given by

$$\delta(\mathbf{x}, a) = D(a)\delta_i(\mathbf{x})$$

where $\delta_i(\mathbf{x})$ is the density perturbation at the initial time t_i and $D(a)$ the linear growth factor normalised to $D(a_i) = 1$. This relation implies that the density perturbation grows self-similar at early times [10]. The linear theory can then be applied to derive the dynamics equations for test particles, which is known as the Zel'dovich approximation [20]. The approximation provides a way to model the evolution of the initial small perturbations. It is a Lagrangian description that specifies the structure growth with the displacement $\mathbf{x} - \mathbf{x}_i$ and the peculiar velocity $\mathbf{v}$ of each mass element as a function of the initial position $\mathbf{x}_i$.
During the matter dominated era ($a(t) \propto t^{2/3}$) the following relations can be derived:

$$\mathbf{v} = -\frac{\dot{D}}{4\pi G \bar{\rho}_m a^2}\nabla \Phi_i = -\frac{1}{4\pi G \bar{\rho}_m a}\frac{\dot{D}}{D}\nabla \Phi \tag{11}$$

Integrating $\mathbf{v} = a\dot{\mathbf{x}}$ leads to

$$\mathbf{x} = \mathbf{x}_i - \frac{D(a)}{4\pi G \bar{\rho}_m a^3}\nabla \Phi_i(\mathbf{x}_i). \tag{12}$$

Linear Perturbation Spectrum
Different Fourier modes of the power spectrum (eq. 7) evolve independently of each other in the linear regime. The initial perturbation spectrum is commonly assumed to be a power law, $P_i(k) \propto k^n$, where n is the spectral index. The Harrison–Zel'dovich spectrum, also called the scale-invariant spectrum uses $n = 1$. There is no refined theory for the cosmological perturbations yet, which is why the amplitude of $P(k)$ needs to be determined by observations. If the shape of the spectrum is known, the amplitude can be obtained by knowing $P(k)$ at any k. Determining the shape and amplitude of $P(k)$ is one of the important ongoing tasks of observational cosmology [10].
One method to normalize the theoretical power spectrum is by using the variance of the galaxy distribution when sampling randomly places spheres of radii R. The predicted variance σ of the density distribution is related to the power spectrum by

$$\sigma^2(R) = \frac{1}{(2\pi)^2} \int P(k) \hat{W}_R^2(k) k^2 dk$$

where

$$\hat{W}_R(k) = \frac{3}{(kR)^2} [\sin(kR) - kR\cos(kR)]$$

is the Fourier transform of the spherical top-hat window function

$$W_R(r) = \begin{cases} \frac{3}{4\pi R^3}, & \text{if } r \leq R \\ 0, & \text{otherwise.} \end{cases}$$

It is common to use $\sigma_8 := \sigma(8\,\mathrm{Mpc}\,h^{-1})$ in order to define the amplitude of the power spectrum [10]. The mode amplitudes of the dark matter power spectrum $P(k)$ grow linearly on large scales, approximately proportional to the cosmological expansion factor a [17]. Once the dimensionless power $\Delta^2(k) = k^3 P(k)/(2\pi)^2$ reaches the value of ~ 1, the non-linear evolution accelerates the growth on small scales. This happens earlier on small scales than on large scales.

2.2.4 Non-Linear Growth

As density perturbations grow due to gravitational collapse, they eventually become non-linear, forming bound structures such as galaxies and clusters of galaxies. Many objects in the observable Universe today, like galaxies and galaxy clusters, have orders of magnitude higher densities than the Universe's average density. These objects are in the highly nonlinear regime, where $\delta \gg 1$.
The description of the large structure formation of those highly non-linear objects requires a gravitational collapse model of overdensities. Generally, non-linear dynamics are hard to calculate analytically and often require computer simulations. In some cases, where symmetries of the system can be used, analytical models can be constructed. These models could for example describe steady end states of the gravitational collapse, applicable to galaxies and dark matter haloes.

2.2.5 Hierarchical Clustering

The first relatively uniform distribution of the Universe starts to form the often-called cosmic web which consists of dark matter clusters and filaments with sizes of $\approx 100\,\mathrm{Mpc\,h^{-1}}$ [17]. Since the matter is attracted to the high-density regions, large voids with low densities between the filaments appear. Generally, the evolved density field is much different from the initial density field and does not follow the Gaussian random field anymore. In particular, the shape of the power spectrum changes, and it no longer completely specifies the density field.
On large scales where the structures are still in the linear regime, $\chi(\mathbf{x}, t)$ grows proportional to $D(t)^2$, where $D(t)$ is the linear growth factor. On small scales, the clustering amplitude increases fast at the early stage of nonlinear evolution and then grows as a^3 due to the formation of clumps whose densities change little while the background density decreases as $1/a^3$. These small structures later merge and form even bigger structures. Small galaxies can lead to the formation of entire galaxy clusters [14].
Even though the physics behind gravitational clustering is relatively simple, in the sense that only gravitational interactions are involved, the resulting dynamics are very complicated. The non-linear evolution can not be predicted anymore, and numerical N-body simulations are required to follow the processes in detail. Numerical models are used to simulate and test the models by comparing the results to observations. From observations, we can extract information about the observable Universe at different time steps. By looking at objects with high redshifts, we can look into the Universe's past and compare different simulation snapshots to the sky observations.

3 Methods

The cosmological simulations shall be done using individual particles representing overdensity regions, making the simulation effectively a N-body problem. The N-body problem is a known classic problem describing the dynamics of N discrete bodies interacting with each other. To describe their dynamics, $N(N-1)/2$ interactions have to be modeled which scales as $\mathcal{O}(N^2)$ for large numbers. This is computationally very expensive and methods like the Particle-Mesh (PM) algorithm have been developed to reduce the computational power to $\mathcal{O}(N \log N)$. The cosmology mini-app will make use of this algorithm to simulate large systems with many particles. The cosmology mini app is set up in the following way:

- Defining the simulation parameters, like initial time, number of particles, size of the simulated volume, mesh size, and multiple other cosmological parameters.
- Generation of the initial conditions. In particular, this means generating the initial positions and velocities of the particles. This step is being done using the public code N-GenIC.
- Setting time evolution parameters like the number of integration steps, end time, and output frequency.
- Running the simulation. This includes a Poisson equation solver to calculate the gravitational potential and its forces, and a Leapfrog integration scheme to update the particle positions and velocities. These functions are obtained from the existing IPPL framework.
- Lastly, a small setup in Python was used to visualize the results and generate pictures and animations.

3.1 Generating Initial Conditions with N-GenIC

The initial dark matter particles with their corresponding positions and velocities are generated with the publicly available N-GenIC code [2]. N-GenIC has been used in the past to generate the initial conditions for the Millenium and the Millenium-XXL projects [17]. The code is MPI-parallel, making it suitable to generate initial conditions containing many particles for very large simulations.

3.1.1 Algorithm

The generation of the cosmological initial conditions with N-GenIC follows the steps below:

- Setup of a Cartesian grid with dimensions corresponding to the volume of the simulated Universe. Periodic boundaries are used.
- Computation of the perturbation field using the inverse Fourier transform of the power spectrum.
- Computation of the perturbation of the velocities using the Zel'dovich approximation.
- Application of these perturbations to the Cartesian grid.

As described in section 2.2.2, the real and imaginary amplitude of the Fourier transform of the overdensity field are normally distributed and the norm of the amplitudes follows a Rayleigh distribution. The algorithm, therefore, starts with an array of Fourier modes, each characterized by its wavevector $\mathbf{k}$, and assigns each a random amplitude $|\delta_k|$ (Rayleigh distributed) and a random phase ϕ_k. The linear overdensity field, $\delta(\mathbf{x})$, can then be obtained using fast Fourier transforms (FFTs).
The code samples the particles homogeneously in a periodic box using Gaussian random fields. The displacement field in Fourier space at the initial time is constructed using the Zel'dovich approximation (eq. 11 and eq. 12). The overdensity field $\delta(\mathbf{x})$ is then represented by discrete particles.
The N-GenIC output format, which is a binary file, is compatible with the GADGET simulation code developed by the Max-Planck Institute for Astrophysics and has been used for the Millenium Simulation [15] [17]. An additional simplified output function was added to the N-GenIC code, which

[2]https://www.h-its.org/2014/11/05/ngenic-code/

writes the particle position and particle velocities to a CSV file. This CSV file contains all required information for the setting up the initial conditions using IPPL and starting the simulation which is described in section 3.2.

3.1.2 Cosmology Parameters

Multiple cosmological parameters have to be set, that describe the ΛCDM model. This also includes defining the initial time of the simulation. The initial time is defined by its redshift z. N-GenIC then creates the initial density distribution of a cube in the Universe for this particular redshift.
The cosmological parameters that have been used, were adapted from the Millennium simulation [4]. However, in comparison to the Millenium simulation, which also included Baryonic matter, only dark matter was simulated in this project. The values that have been used in all simulations in this report are listed in Tab. 1. The files with all input parameters can be found in the GitHub repository [3].

Parameter	Value	Physical Meaning
`Box`	50 000	Physical size of the simulated cube. Length is given in code units (specified below as kpc h^{-1}).
`Omega`	0.3	Total matter density at $z = 0$ (Ω_m).
`OmegaLambda`	0.7	Cosmological constant at $z = 0$ (Ω_Λ).
`OmegaBaryon`	0.0	Baryon density at $z = 0$ (Ω_b).
`HubbleParam`	0.7	Hubble Parameter today.
`Redshift`	63	Redshift at time of initial conditions.
`Sigma8`	0.9	Velocity dispersion coefficient σ_8 of the ΛCDM model.
`TileFac`	$i \in \mathbb{N}$	$16^3 \cdot$ `TileFac`^{3} equals the number of particles.
`Nsample`	$16 \cdot i$	Defines the maximum k and determines the Nyquist frequency. Normally chosen, such that `Nsample`^{3} equals the number of particles. Therefore it was defined as $16 \cdot$ `TileFac`.
`Nmesh`	$16 \cdot i$	Size of the FFT grid for displacement. Must fulfill `Nmesh` $\geq$ `Nsample`. It was chosen to be equal to $16 \cdot$ `TileFac`.
`UnitLength_in_cm`	3.085678e21	Defines the length unit of output (in cm/h). Equals to 1 kpc h^{-1} in this case.
`UnitMass_in_g`	1.989e43	Defines the mass unit of output (in g). Equals to $10^{10} M_\odot$ in this case.
`UnitVelocity_in_cm_per_s`	1e5	Defines velocity unit of output (in cm/sec). Equals to 1 km s^{-1} in this case.

Table 1: N-GenIC Input Parameters.

3.1.3 Number of Particles

The number of particles varied between $16^3 = 4096$ increasing with a factor of 2^3 up to $512^3 \approx 16.8$ million particles. As a comparison, the Millenium simulation contained $2160^3 \approx 10^{10}$ particles, where each particle represented approximately a billion solar masses of dark matter [17]. In this project, all particles obtained the same mass, which corresponded to the total mass in the physical volume divided by the number of particles. To obtain the total mass, the critical density of the matter content of the Universe was used as defined in eq. 13 and further explained in section 2.1.1.

$$\rho_{m,0} = \Omega_m \frac{3H_0^2}{8\pi G} \tag{13}$$

[3]https://github.com/bcrazzolara/SimGadget/tree/master/ngenic

The physical size of the system that was studied corresponds to a cube with sidelength of $50\,\mathrm{Mpc\,h^{-1}}$ which leads to a total mass of $M_{tot} = 1.041 \times 10^{16}\,\mathrm{M}_\odot$ in the simulation. Note: While the total mass is a conserved quantity of the system in the simulation, the average mass density $\bar{\rho}$ (eq. 10) scales as a^{-3} due to the expansion of the physical volume.

3.2 Particle Mesh Algorithm with IPPL

The N-body problem describing the motion of N discrete bodies interacting through gravity requires the calculation of $N(N-1)/2$ interactions. This computational scaling as $\mathcal{O}(N^2)$ is very expensive for large numbers of interacting particles. Methods like the particle-mesh (PM) algorithm have been developed that approximate the dynamics and reduce the computational power to $\mathcal{O}(N \log N)$. The PM method describes the system with both discrete freely moving particles as well as an underlying grid structure [3]. The Poisson equation (eq. 14) is solved on the grid after the density field has been estimated on the grid points using the particle positions and their masses. The gravitational force field is then calculated on the underlying grid using Fourier transformation to reduce computational power. Interpolating the forces back to the particle positions then allows for simulation of the dynamics of the individual particles [3].

The PM algorithm from the IPPL library was used. IPPL uses HeFFTe (highly efficient FFT for exascale) for parallelizable fast Fourier transforms. IPPL can run on CPUs and GPUs [9]. The IPPL framework uses Lagrangian, Eulerian, and hybrid Eulerian-Lagrangian solvers as well as the Particle-In-Cell (PIC) method. The Eulerian methods discretize space and represent variables on a mesh, while Lagrangian methods describe individual particles. In IPPL particles are stored in containers, with each particle attribute saved as a Kokkos View enhanced with expression templates. Fields are like the particle attributes stored as Kokkos Views enhanced with expression templates.
MPI is used to distribute fields and particles across multiple processes and split the work. To share field data across processors, ghost cells are used for domain boundaries. In principle, particle attributes like the charge or the mass are spread onto the grid, and the grid is divided into different sections, each containing a similar amount of particles. Each region is then assigned to one processor. Further information about the structure of IPPL can be found in [11].

IPPL has been used to develop mini-apps (Alpine[4]) for different physics applications [11]. They are based on the PIC method which is easy for parallelization and useful for many physical scenarios. Specifically, the mini-apps have been used for plasma physics and particle accelerator simulations. The mini-apps are performance-portable and can be used as an interface between High-Performance Computing (HPC), applied mathematics, and simulation codes. The mini-apps are used for physical scenarios where pure electrostatic holds and collision of particles can be neglected.
The use of IPPL and its PM algorithm in the cosmology mini-app is strongly based on the existing framework of the Landau damping mini-app from IPPL [11]. The Landau damping mini-app describes the motion of charged particles by solving the Poisson equation of electrostatic and obtaining the particle accelerations from the gradient of the potential field. This overview shows the major changes in the physics framework of the cosmology mini-app compared to the pre-existing mini-apps. The physical equations had to be adapted to gravity, and more specifically gravitational interactions in an expanding Universe. The charge density was substituted by the mass density, the electric field by the gravitational field, and the electric potential by the gravitational potential. The expansion of the Universe leads to additional changes in the equations. The relevant equations that are used for electrostatic respective for cosmology are summarized in Tab. 2. The derivation of the cosmology equations are described in more detail in section 2.

[4]ALPINE: A set of performance portable pLasma physics Particle-in-cell mINi-apps for Exascale

Electrostatic	**Gravity**	
$\Delta\Phi(\mathbf{r}) = -\nabla \cdot \mathbf{E}(\mathbf{r}) = -\frac{\rho(\mathbf{r})}{\epsilon_0}$	$\Delta\Phi(\mathbf{x}) = 4\pi G a^2 \rho(\mathbf{x})$	(14)
$\frac{\mathrm{d}\mathbf{r}_i}{\mathrm{d}t} = \mathbf{v}_i$	$\frac{\mathrm{d}\mathbf{x}_i}{\mathrm{d}t} = \frac{1}{a^2}\mathbf{u}_i$	(15)
$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = -\nabla\Phi(\mathbf{r}_i)$	$\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t} = -\nabla\Phi(\mathbf{x}_i)$	(16)

Table 2: Equations for electrostatic and gravity in an expanding flat Universe [10].

For the Poisson equation (eq. 14), only the prefactor had to be changed compared to the Poisson equation from electrostatic. For the integration scheme, two major changes were implemented. First of all, it was decided to change the traditional velocity attribute $\dot{\mathbf{x}}$ to a re-scaled velocity $\mathbf{u} = a^2\dot{\mathbf{x}}$. Expressing the Leapfrog in terms of $\mathbf{u}$ simplifies the *kick* equation, making the time derivative of $\mathbf{u}$ only explicitly dependent on the gravitational potential in comparison to the physical velocity as shown in eq. 9.
The second major change was the time step. Previously the time step in the Landau damping mini-app was constant (total time range divided by the number of time steps). For the cosmological simulations, it was decided to adjust the time step to the expansion of the Universe. Instead of using a fixed Δt, a fixed $\Delta \log a$ was implemented using the following relations:

$$H = \frac{1}{a}\frac{\mathrm{d}a}{\mathrm{d}t} = \frac{\mathrm{d}\log a}{\mathrm{d}t} \qquad \Rightarrow \qquad \mathrm{d}\log a = H\mathrm{d}t$$

The time step for a fixed $\Delta \log a$ can be estimated as

$$\Delta t = \frac{1}{H}\Delta(\log a).$$

For a fixed $\Delta(\log a)$, this implies that the time step becomes small for high values of H, which is equivalent to a fast expansion of the Universe. In other words, the more the expansion accelerates, the smaller the time step is chosen.

The time derivative of the comoving coordinates $\mathbf{x}$ is given by

$$\frac{\mathrm{d}\mathbf{x}_i}{\mathrm{d}t} = \frac{1}{a^2}\mathbf{u}_i,$$

and the integration of the velocity in terms of $\mathrm{d}\log a$ is given by

$$\int \mathrm{d}\mathbf{x}_i = \int \mathbf{u}_i \frac{1}{a^2}\mathrm{d}t = \int \mathbf{u}_i \frac{1}{Ha^2}\mathrm{d}(\log a). \tag{17}$$

For the integration of the modified velocity $\mathbf{u}$ we obtain

$$\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t} = -\nabla\Phi|_{\mathbf{x}_i}$$

$$\int \mathrm{d}\mathbf{u}_i = -\int \nabla\Phi|_{\mathbf{x}_i}\,\mathrm{d}t = -\int \nabla\Phi|_{\mathbf{x}_i}\frac{1}{H}\,\mathrm{d}(\log a). \tag{18}$$

For the numerical implementation, not only the velocity attribute has been changed, but also the density and the potential. The numerically implemented rescaled density is defined as $\rho'(\mathbf{x}, t) =$

$a^3(t)\rho(\mathbf{x},t)$ such that $\bar{\rho}'(t) = \bar{\rho}'(0)$ and the average normalised density is constant over time and defined as total mass divided by the comoving (constant) volume.
The rescaled potential is defined as

$$\Phi' = \frac{a(t)}{4\pi G}\Phi,$$

which gives the Poisson equation for the rescaled potential Φ':

$$\Delta\Phi' = \frac{a(t)}{4\pi G}\Delta\Phi = a(t)^3(\rho(\mathbf{x},t) - \bar{\rho}(\mathbf{x},t)) = \rho'(\mathbf{x},t) - \bar{\rho}'.$$

Combining the rescaled potential with eq. 17 and eq. 18 gives the Leapfrog scheme:

- **Kick 1:** Update the velocities of the particle positions with their accelerations by half a time step:
$$\mathbf{u}_i^{(k+1/2)} = \mathbf{u}_i^{(k)} + \mathbf{a}_i^{(k)}\frac{\Delta t}{2} = \mathbf{u}_i^{(k)} - \frac{4\pi G}{Ha}\nabla\Phi'|_{\mathbf{x}_i^{(k)}}\frac{\Delta(\log a)}{2} \tag{19}$$
- **Drift:** Update the particle positions with their velocities by one time step:
$$\mathbf{x}_i^{(k+1)} = \mathbf{x}_i^{(k)} + \frac{1}{a^2}\mathbf{u}_i^{(k+1/2)}\Delta t = \mathbf{x}_i^{(k)} + \frac{1}{Ha^2}\mathbf{u}_i^{(k+1/2)}\Delta(\log a) \tag{20}$$
- **Scatter:** Distribute the masses of the particles onto the grid points.
- **Poisson equation:** Use a grid-based solver for the Poisson equation $\Delta\Phi' = \rho'(\mathbf{x},t) - \bar{\rho}'$ to calculate the potential field Φ' and the force field $\mathbf{g} = -\nabla\Phi'$ on the grid.
- **Gather:** Interpolate the force field $\mathbf{g}$ from the grid to the particle positions and calculate their new accelerations $\mathbf{a}^{(k+1)}$.
- **Kick 2:** Update the velocities of the particles with their accelerations by half a time step:
$$\mathbf{u}_i^{(k+1)} = \mathbf{u}_i^{(k+1/2)} + \mathbf{a}_i^{(k+1)}\frac{\Delta t}{2} = \mathbf{u}_i^{(k+1/2)} - \frac{4\pi G}{Ha}\nabla\Phi'|_{\mathbf{x}_i^{(k+1)}}\frac{\Delta(\log a)}{2} \tag{21}$$

3.2.1 Mesh Size

The number of mesh points was chosen to be identical to the number of particles. This means that in average there is one particle per grid cell.

3.2.2 Scatter/Gather

IPPL uses a cell-centered grid and cloud-in-cell shape function for the interpolation from the particles to the grid and vice versa [11]. Periodic boundary conditions are applied in all dimensions. The grid spacings in each dimension is constant to enable a Fast Fourier Transform (FFT) algorithm to solve the Poisson equation (eq. 14) on the grid.

3.2.3 Solving the Poisson equation

After the scattering of the particle masses (or charges) onto the grid, the density ρ can be calculated at the grid points. The Poisson equation (eq. 14) can be solved in Fourier space in order to calculate the gravitational potential at the grid points. Calculating the Poisson equation brute force requires a computational power of $\mathcal{O}(N^2)$ with $N = N_xN_yN_z$ the product of grid points in each direction of a 3D grid [9]. When using uniform meshes (constant spacing along each axis), solving the equation in Fourier space significantly reduces the computation power to $\mathcal{O}(N\log N)$.
This solver method has been implemented in the IPPL library using the `heFFTe` (highly efficient FFT for exascale) library which uses the FFT algorithm. `heFFTe` is designed for exascale supercomputers [2]. It scales linearly with GPUs and CPUs and uses MPI and OpenMP interfaces for communication. The solver has already been used in the Alpine mini-apps, specifically, the Landau Damping mini-app has been used as a template again.

3.2.4 Update Particles

After the potential has been calculated on the grid points, the gravitational force can be extrapolated to the positions of the particles. The force can be used to update the velocity of the particles and evolve also their positions in time. The Leapfrog "kick-drift-kick" (KDK) scheme that has been used is a second-order method specifically designed for second-order differential equations of the form $\dot{\mathbf{r}} = \mathbf{v}$, $\dot{\mathbf{v}} = \mathbf{a}(\mathbf{r})$.
The differential equations for $\mathbf{r}$ and $\mathbf{v}$ are solved at interleaved time steps when using the KDK Leapfrog. The idea behind KDK Leapfrog is to use the acceleration from the previous step (k) to update the velocity to time-step $k + 1/2$ corresponding to the first kick. Then, the position is updated to time $k + 1$ by drifting along the updated velocity. Lastly, a second kick is used to update the velocity based on the acceleration at the current time step ($k + 1$). The 3 steps of the Leapfrog are given in the equations 19, 20, and 21. The method is time-symmetric and provides the same results when run forward in time as when run backward in time [3].

3.3 Code Parallelization

3.3.1 Techniques

Parallel execution of the code is an efficient way to achieve higher performance for large simulations. The code can run on shared memory as well as distributed memory architectures. Both architectures can be applied to the code since a message-passing interface is included. For both methods, the following three points are important to study:

- Minimize processes where all processors have to wait on each other's progress
- Load-Balancing: Ensuring each process is responsible for an equal amount of work, which helps for point 1
- Reducing Communication Overheads: Minimizing the amount of data that has to be shared between the processors.

Since there are many different computer architectures, writing the code flexibly is important.
When using shared memory, several CPUs share the same main memory, such that expensive computation loops can be easily distributed for parallel execution. Modern compilers can create and destroy the threads automatically using OpenMP. This type of parallelization is often easier to implement and requires fewer changes to existing serial code.
A different approach is to use the CPUs as independent computers with each of them having their own physical memory. Ideally, each one only stores the fraction of the data that is relevant for the responsible part of the code (distributed memory). This approach is more complicated as it requires explicit instructions for the communication between the CPUs, memory storing and accessing, and results sharing. However, this implementation provides a bigger potential for scaling up to larger problems and large processor numbers and can be run on computer clusters with many computers.
The cosmology mini-app is compatible with both computer architectures and has a high degree of portability and can be run on an arbitrary number of processors. MPI is used only for the communication between the nodes, while OpenMP-based shared memory parallelism is used within a node [11]. This has been proven by testing the code on different clusters with different settings. The exact setups that have been used are described in section 4.1.

3.3.2 Read-In

The read-in in the cosmology mini-app can be done in two different ways. The first option was to divide the amount of particles into subsets of equal number of particles and then assign each subset to one processor. This option makes the read-in faster since every processor only reads in their fraction of the input file. However, this read-in does not guarantee that the particles are lying in the domain of the processor. The consequence of this read-in method is that during the very first `update` function call, where particles are sent to their corresponding processor, the size of the sent information might exceed the allowed maximum message size. This problem has occurred for problem sizes larger than 256^3.

Due to this issue, it was decided to implement a second method for the read-in that already assigns the particles to the correct processors. This method makes the read-in slower since every processor needs to read the entire input file and check for every particle if it lies in its domain. After the read-in however, the first update will be much faster.
Furthermore, it was observed that the sending process of the particles is unstable in the case where a particle lies precisely at the boundary of the domain of two processors. For those cases, a small perturbation (0.01 %) of the particle positions was included when reading in the position. This issue has been detected since a few particles have been double-counted which throws an error since the total number of particles is not conserved.

3.3.3 Output

The data was chosen to be saved as CSV files as this was the most convenient way to implement it. The output of the cosmological simulations for large numbers of particles leads to a substantial amount of data. An individual file containing the positions and velocity data of all particles ($6 \cdot N$ values) can reach a few GByte. Reading these files (for initial conditions) and writing them (for result storing) becomes very time-consuming, especially on GPUs. It has been decided that the output files from the simulations are split up and every processor generates a file containing the information of its particles. This allows a parallelized output generation as well as storing the files in smaller chunks.

4 Scaling Results

4.1 Setup

The cosmology mini-app was tested with multiple numbers of particles and multiple numbers of CPUs or GPUs. Testing the code for different configurations allows us to understand the time required for simulations and to be able to upscale simulations appropriately.
Studying the problem size by changing the number of particles allows us to estimate the upscaling of the problem for larger or more detailed simulations. Studying the number of processors helps to understand the power of parallelization of the code and the benefit of using computer clusters. The two parameters can be studied using two different scaling studies:

Strong Scaling: In the case of strong scaling, the number of processors is increased while the problem size remains constant. This leads to a reduced workload per processor and helps to reduce the overall running time of large problems. For an ideal speedup, the runtime would scale as $\mathcal{O}(1/N)$ with N the number of processors. Due to non-vanishing communication time, this proportionality will not hold for large numbers of processors.

Weak Scaling: For weak scaling studies, the number of processors and the problem size are increased, with the goal of keeping the workload per processor constant. Good weak scaling means that we are able to solve larger problems on bigger machines in the same time as smaller problems on smaller machines. The limitations are therefore given by the available resources.

The computer clusters that have been used are listed in Tab. 3. Merlin and Gwendolen are both clusters available at PSI. The Euler cluster is available to all ETH members.

Cluster	**Setup**
Merlin6	Up to 16 nodes (exclusive), with one MPI rank per node, and 44 OpenMP threads (equal to the total number of CPU cores on a single node). Multithreading is turned off.
Gwendolen	One node (exclusive) with up to 8 GPUs. One MPI rank per GPU, no OpenMP thread parallelism.
Euler	One node, up to 8 MPI threads using one CPU per task (non-exclusive). Not used for scaling studies in this work.

Table 3: A list of the HPC clusters that have been used to test the code and its performance. The described setup has been used for the scaling studies of this section.

4.1.1 Contributions of Code Sections

This section evaluates the contribution of different parts of the code to the total simulation time for selected problem sizes. The contribution of the simulation time can be divided into computation and communication time. Both sections can be subdivided into individual code sections listed in Tab. 4. The fraction of the total computation time that one particular part of the code contributes, depends on many factors. This includes the particle number, the number of time steps of the simulation, the number of processors, and generally the computer architecture. The read-in time for example contributes less when simulating for longer times. Also, the read-in takes on GPUs proportionally longer than on CPUs (same for the saving process). Then different parts of the code scale differently with the particle number and the communication contribution is relatively high for small problem sizes due to communication overhead. This shows just a few points of the simulation that influenced the individual contribution times.
A specific problem size with 512^3 particles and 500 timesteps has been used to give an example of the above code contributions for CPUs and GPUs. The initial conditions for the simulations have been generated as described in section 3.1, using the specific parameters listed in Tab. 1. The timing for the generation of the initial conditions has not been included in this study, as it is negligible for large simulations and has to be only once in the beginning and can be reused as the data is saved as a CSV file.

Computation kernels	**Communication kernels**
Gather (`gather`)	Particle update (`updateParticle`)
Scatter (`scatter`)	Fill halo cells (`fillHalo`)
Push position (`pushPosition`)	Accumulate halo cells (`accumulateHalo`)
Push velocity (`pushVelocity`)	Particle load balance (`loadBalance`)
Particle BCs (`particleBC`)	
FFT-based field solve (`solve`)	

Table 4: List of computation and communication kernels used for the performance study. The labels in the parentheses correspond to the labels used in the Figures of this section.

The timings as a fraction of the total simulation time were calculated for the computation and communication kernels.
In Fig. 1 it can be observed that the main contribution to the total simulation time originates clearly from the solver ($> 90\%$) almost independent of the number of CPUs. Together with the communication results from Fig. 2, we can conclude that this problem size is well suited for parallelization on CPUs as the solver, which can be executed in parallel (solver scaling will be discussed in section 4.2), is the major component of the simulation and communication between the CPUs is still negligible. All communication kernels are as expected increasing with the number of CPUs and equal to zero when using only one CPU.

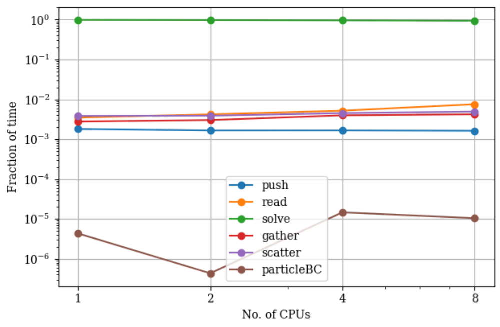


Figure 1: Fractional contribution of computation kernels on CPUs with a problem size of 512^3 particles and 500 steps. Simulation time is dominated by the solver.

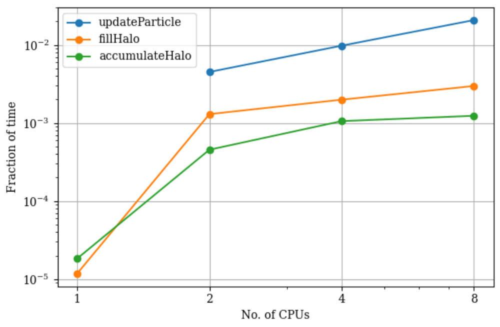


Figure 2: Fractional contribution of communication kernels on CPUs with a problem size of 512^3 particles and 500 steps. Communication time is negligible.

The contributions for the exact same problem size using GPUs differ significantly from the performance on CPUs. In Fig. 3 it can be observed that the main contribution to the total simulation time originates from the read-in ($> 60\%$) while the solver time contributes less than 20%. Even though the total simulation time using GPUs is significantly faster than on CPUs since the solver time decreases substantially, it is a bad sign that the read-in time dominates the overall time. The behavior is not unexpected, however, it proves how important it is to carry out input and output on GPUs carefully in order to reduce their inefficiency. All communication kernels shown in Fig. 4 are as expected increasing with the number of GPUs and equal to zero when using only one GPU.

4.2 Strong Scaling

4.2.1 Read-In

The read-in scales almost ideally with respect to the number of particles, $T_{read}(N_p) = \mathcal{O}(N_p)$. Increasing the number of MPI ranks on 1 node using 1 CPU/rank does not decrease the read-in time. This is expected, as we have used a selective read-in function as described in section 3.3.2. Therefore each MPI rank always needs to read in the entire initial conditions file.
The same scaling study has been conducted using GPUs on Gwendolen with the setup from Tab. 3.

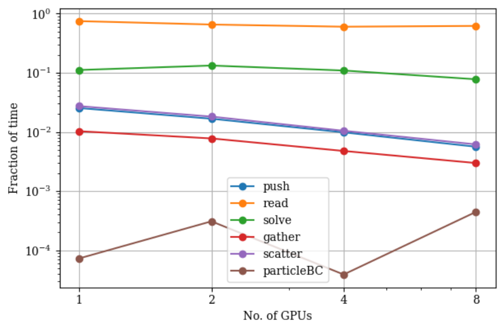


Figure 3: Fractional contribution of computation kernels on GPUs with a problem size of 512^3 particles and 500 steps. Simulation time for this problem set-up is clearly dominated by the read-in.

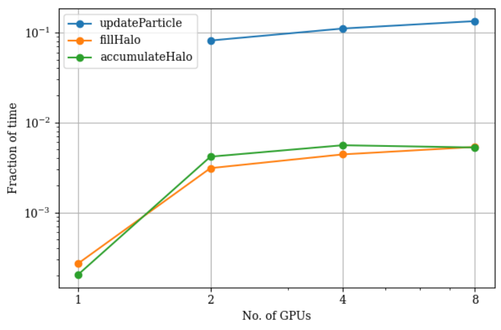


Figure 4: Fractional contribution of communication kernels on GPUs with a problem size of 512^3 particles and 500 steps.

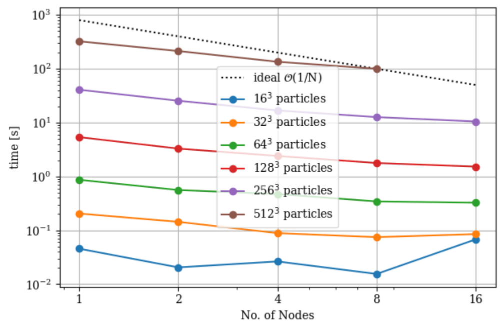


Figure 5: The figure shows the read-in time of different numbers of particles as a function of the number of nodes with each using 1 CPU.

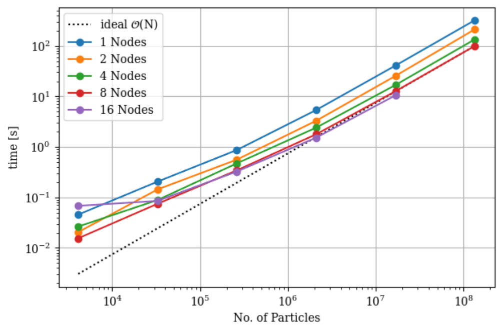


Figure 6: The figure shows the read-in time of different numbers of nodes (1 CPU per node) as a function of the number of particles. It scales as expected.

4.2.2 Solver

The part of the code that is responsible for solving the Poisson equation takes up a significant time of the simulation, especially on CPUs. The solving time has been studied as a function of the number of processors (either CPUs or GPUs) and as a number of particles.

Running the simulation on Merlin CPU nodes (with the setup from Tab. 3), shows a good scaling behavior for problem sizes bigger than 128^3 as shown in Fig. 9. The solver computation time scales well as $\mathcal{O}(1/N)$ with N the number of CPU nodes. For small problem sizes no scaling or even a speed-up can be observed, as the communication time between the CPUs is too long in comparison to the computational speedup. As expected, the solver scales as $\mathcal{O}(N \log N)$ with N the number of particles. As can be seen in Fig. 10, the timings do not increase continuously but show jumps in the timings even though the overall trend is correct. These jumps are believed to be the result of cache memory.

On GPUs the scaling trend only starts for much larger problem sizes. Since the computation is very efficient on GPUs, the communication between the GPUs leads to small speed-ups even for problem sizes of 256^3 particles. The speed up of $\mathcal{O}(1/N)$ for N GPUs only starts to be detectable for problem sizes of $\sim 512^3$ particles as shown in Fig. 11. For very small problem size it can be faster to run the problem on one GPU only. The increase of time from 1 GPU to 2 GPUs is due to the solver time of `heFFTe` which is also the case in their analysis [2]. As a result, the ideal scaling as $\mathcal{O}(N \log N)$ of the FFT solver using `heFFTe` becomes only visible for large number of particles as shown in Fig. 12. For small particle numbers (up to $\sim 256^3$) there is no proper scaling, as the communication time between the GPUs dominates the solving time.

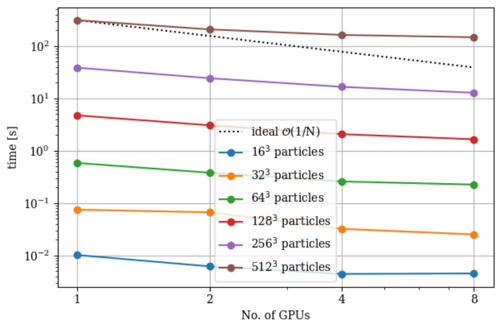


Figure 7: The figure shows the read-in time of different numbers of particles as a function of the number of GPUs used on 1 Node.

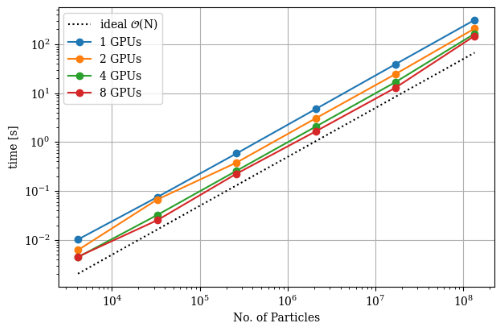


Figure 8: The figure shows the read-in time of different numbers of GPUs on 1 node as a function of the number of particles. It scales as expected.

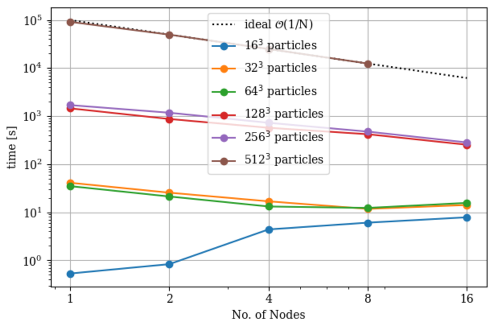


Figure 9: The figure shows the solver time of different numbers of particles as a function of the number of nodes. For each node 1 MPI rank and 44 CPUs were used. For large particle numbers, the time scales as expected. For small particle numbers, the communication overhead makes parallelization inefficient.

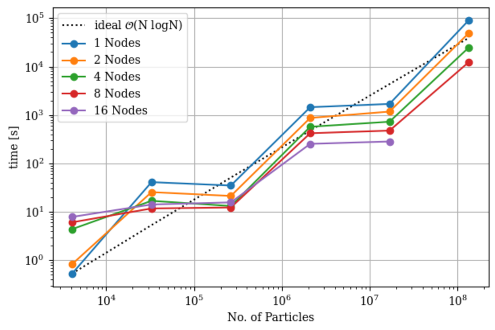


Figure 10: The figure shows the solver time of different numbers of nodes as a function of the number of particles. For each node 1 MPI rank and 44 CPUs were used.

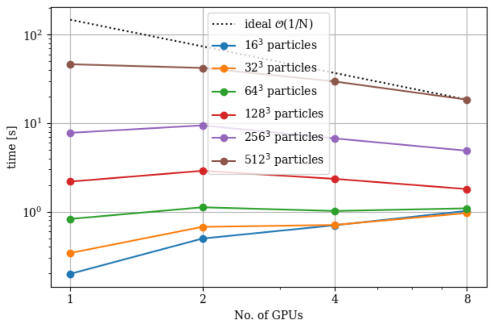


Figure 11: The figure shows the solver time of different numbers of particles as a function of the number of GPUs used on 1 Node. Parallelization only starts to become efficient at large problem sizes.

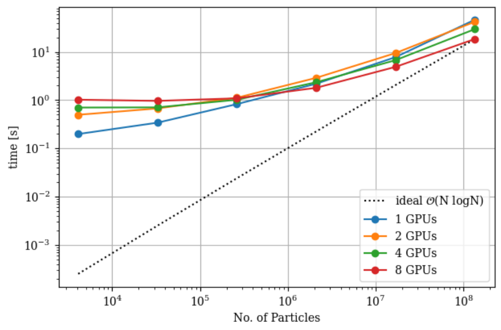


Figure 12: The figure shows the solver time of different numbers of GPUs as a function of the number of particles.

5 Structure Formation Results

5.1 Saving Process

Saving the data of the simulation is expensive not only in terms of computational time but also because it takes a lot of storage. Storing 128^3 particles with their corresponding coordinates and velocities in 3 dimensions takes $\approx 150\,\mathrm{MB}$ and therefore $\approx 10\,\mathrm{GB}$ for 512^3 particles. For this reason, the scaling studies (in section 4.2) have been studied without saving the results.
At the beginning of the project, the results were saved for testing and verification. The visualization of the particles helps to easily spot significant errors, like wrong signs/prefactors, etc. Furthermore, the visualization helps to understand the complex dynamics of large structure formation described in section 2.2.

5.2 Verification of the Results

In order to verify the simulation and the results, the computed particle properties were compared to simulations conducted with GADGET 2.0 [15]. GADGET is a massively parallel TreePM code that can be used for collisionless fluids and ideal gas using smoothed particle hydrodynamics. Short-range forces can be computed with the tree method, while long-range forces are computed with the PM method as in IPPL. Furthermore, it allows adaptive time steps.

To compare the two codes, the initial conditions were identical in both simulations. The initial positions and velocities of 64^3 particles were generated with N-GenIC. The simulations were compared in a statistical sense, by plotting the distribution of the particle positions and velocities of the particles. In Fig. 13, the initial particle distribution along the x-axis (identical for GADGET and IPPL) is plotted for the initial redshift $z = 63$ (left) and the final distribution at $z = 0$ (right). As expected the distribution changes over time and the mass becomes clustered. This is clearly visible in the two peaks at $\sim 3\,h^{-1}\,\mathrm{Mpc}$ and $\sim 11\,h^{-1}\,\mathrm{Mpc}$. It can be observed that the particles get slightly more clustered in our simulation compared to GADGET. Overall, there is a good agreement.

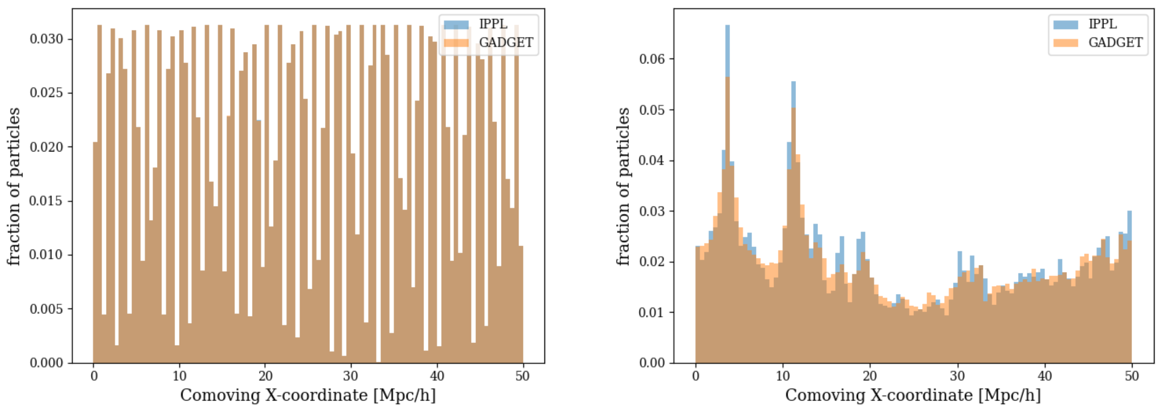


Figure 13: The figure shows the particle distribution along the x-axis at the initial redshift $z = 63$ (left) and the final time at $z = 0$ (right). The particles start with the same initial positions for the two simulation methods and become clustered due to gravitational attraction.

The velocity distribution along one direction and the absolute value velocity distribution were compared to the results of GADGET as well. In Fig. 14, the velocity distribution in the x-direction is shown. It can be seen that the final velocity dispersion with GADGET is smaller than the one from this work but it is still showing acceptable agreement. This behavior also applies to the y and z directions.
The resulting distribution of the absolute value of the velocity also shows a difference between the two simulation methods. Again the velocity distribution at the final time shows a smaller dispersion for GADGET as shown in Fig. 15.

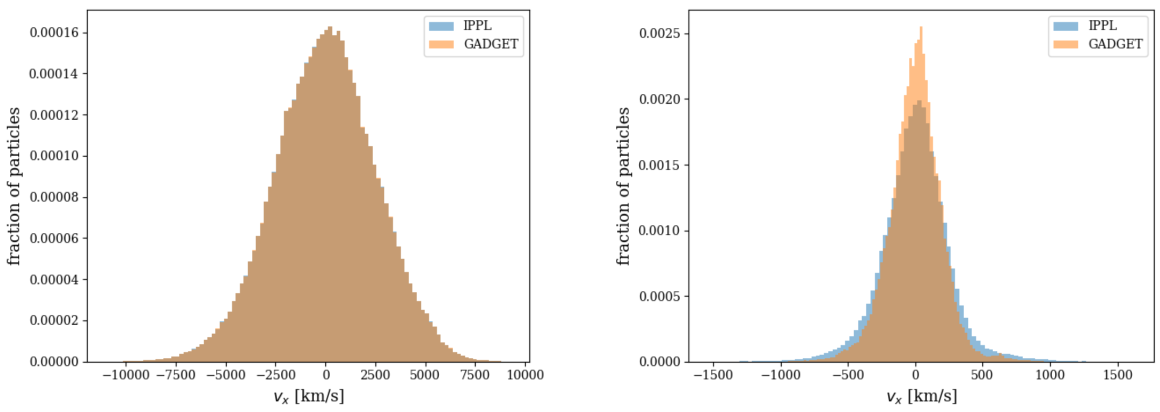


Figure 14: The figure shows the velocity distribution in x direction at the initial time of $z = 63$ (left) and final time $z = 0$ (right). The initial velocity dispersion decreases for both simulation methods similarly.

Overall our simulation results show good agreement with the results of GADGET. The differences between the two simulation methods are believed to be a result mainly due to the difference in the force computation. GADGET uses a TreePM method that computes small-scale forces differently (tree method) than the pure PM method from IPPL. This causes differences, particularly in high-density regions. Since we are considering an N-body system with a high number of particles (64^3), it is expected that small differences in the time integration methods / numerical differences lead to noticeable differences in the simulation results.

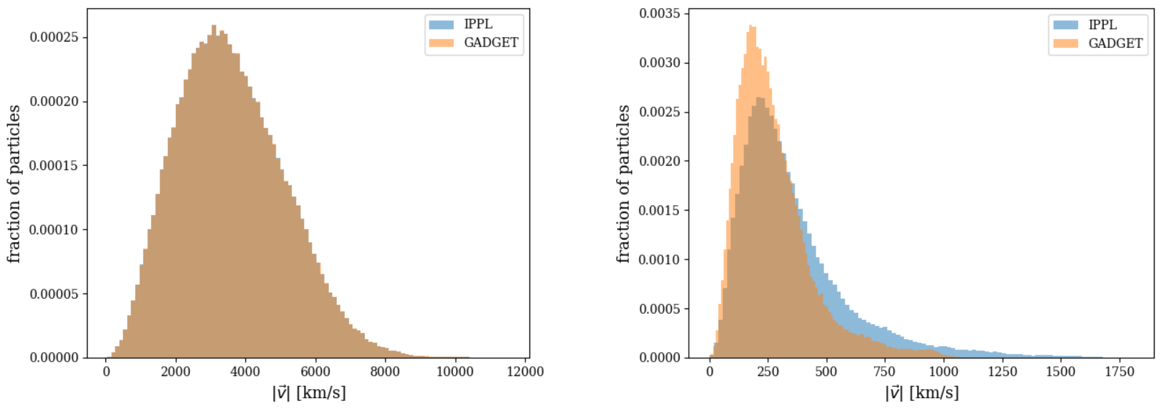


Figure 15: The figure shows the velocity distribution at the initial time of $z = 63$ (left) and final time $z = 0$ (right). The particles start with the same initial velocities for the two simulation methods and become clustered due to gravitational attraction.

5.3 Visualisation

The visualization of the data was done using Python and `matplotlib.pyplot`. It is noted that the plotting process takes quite some time. Not only reading the data but also generating the pictures takes long. The Python script for the visualization of the plots in this section can be found on GitLab [5]. It reads the CSV output files and generates for each timestep one picture which represents a 2D histogram of the particle positions projected onto one axis. The color of each pixel then indicates the accumulated number of particles along the third axis. The pictures can then be combined to form a GIF.
The simulation set-up from Tab. 1 has been used with 512^3 particles and 2000 integration timesteps starting at $z = 63$ until $z = 0$ to generate a few examples of the visualization. Selected timesteps can

[5]https://gitlab.psi.ch/AMAS-students/crazzolara-sem

be found in Fig. 16. Additional plots and GIFs can be found on GitLab. It can be observed from Fig. 16 that the structures grow in a hierarchical fashion as described in section 2.2.5. Small structures grow due to gravitation collapse, grow, and merge later into larger structures. The most prominent large-scale structures are the filaments which are produced by the gravitational collapse of the overdensity regions. At the intersections of filaments, the round dark matter haloes can be found. Smaller haloes can later merge into larger haloes as can be observed in the lower left corner of the simulation. The total mass of the simulation is $M_{tot} = 1.041 \times 10^{16}\,\mathrm{M}_{\odot}$ in a volume of $125\,000\,\mathrm{Mpc}^3\,h^{-3}$. The masses of the individual particles are given by the total mass M_{tot} divided by the number of particles. In this case, this leads to individual masses of $m = 7.756 \times 10^7\,\mathrm{M}_{\odot}$ which means that each particle represents almost 100 million solar masses.

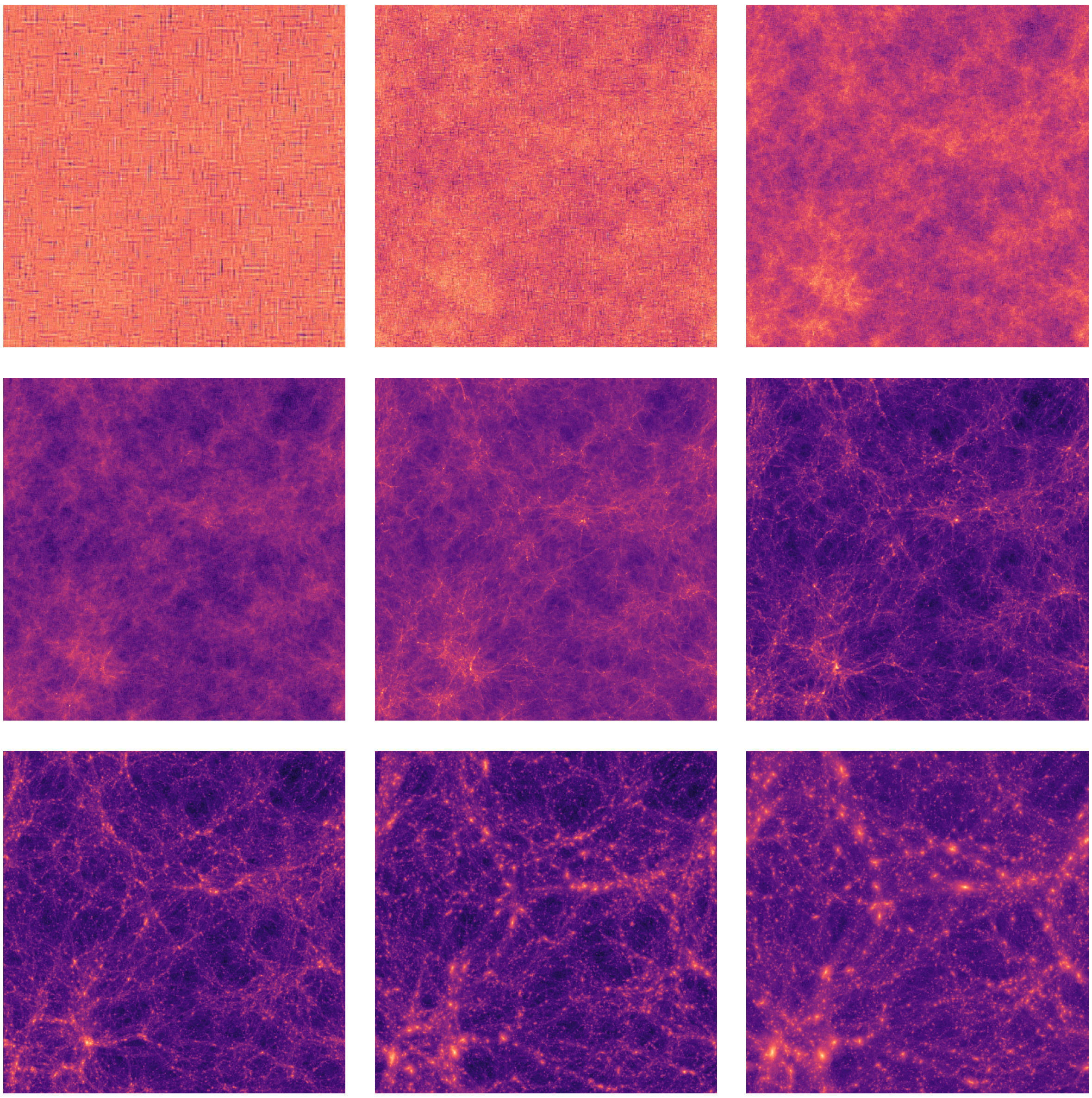


Figure 16: The image shows the structure formation with a comoving sidelength of $50\,\mathrm{Mpc\,h}^{-1}$ simulated with 512^3 particles. The images show the projection of the cube onto one axis, and the colors of the pixels represent the logarithmic accumulated number of particles along the third axis. The pictures (ordered from left to right, top to down) are taken at the following redshifts: 50.98, 26.86, 13.93, 11.13, 7, 4.23, 2.48, 0.86, 0.

6 Conclusion

6.1 Summary and Discussion

During this semester's thesis, an additional mini-app for cosmology has been implemented based on the pre-existing IPPL library initially designed for plasma physics applications [11]. The development of numerical methods for HPC is applicable to many research areas in physics but also in other sciences. This project has proven how valuable it is to develop flexible frameworks that can easily be adapted for several applications. Progresses in the algorithms can therefore be reused without having to rewrite the simulation code.
The cosmology mini-app successfully uses the PM method to solve the Poisson equation for gravity in an expanding Universe. With the Leapfrog integration scheme, the particles evolve in time. The code is performance portable and has been successfully run on CPUs using the computer clusters Merlin and Euler and runs on GPUs which was tested on Gwendolen. It was confirmed that the GPU is much more efficient for the solver part of the code. A simulation of 512^3 particles with 500 time steps (no output saving) on 8 CPU nodes takes 1.33×10^4 s with the solver part contributing 1.24×10^4 s (93%). Using 8 GPUs on one node for the same simulation setup takes 238 s, with the solver contributing 18.5 s (7.8%). Furthermore, it was observed that for this simulation size, the major contribution on a CPU run is the solver, whereas on the GPUs the read-in contributes the most (62%). This shows that GPUs are especially useful for long simulation runs and the read-in (and output-saving) process should be optimized when using GPUs.
The largest simulation that has been run, contained 512^3 particles in a physical volume of 125 000 Mpc h^{-1} and was run on 16 CPUs on one Euler node. The simulation took 15h for 2000 time steps and 30 output timings. Each output contains 10GB of data on all particle positions and velocities.
In comparison, the Millenium simulation conducted by the leading group of the Max-Planck-Institute which was run in 2005 simulated a volume that was 1000 times bigger than the one from this project ($L = 500\,\mathrm{Mpc\,h^{-1}}$). The simulation followed $2160^3 \approx 10^9$ particles, starting at a redshift of $z = 127$ and running for 1 month. The total data they saved was nearly 20 TB for 64 outputs, each with ≈ 300 GB [17].
The Millenium simulation used the GADGET-2 code. This code supports collisionless simulations and smoothed particle hydrodynamics on massively parallel computers [16]. Furthermore, it contains a tree algorithm for short-range forces, while the long-range forces are calculated with a PM algorithm. In comparison to the code of this project, the GADGET code is especially more powerful when working with high-contrast mass distributions. It allows to refine of the computation for high-density regions, by increasing the mesh size, refining the time step, or adding direct force computation [15]. This is in particular useful to refine the simulation details at the location of dark matter haloes and galaxies.

6.2 Further Steps

6.2.1 Comparison to other Cosmology Codes and Analytical Solutions

Due to the complexity of the cosmological large structure formation, it is not possible to verify the correctness of the results easily. Often no analytical solution exists to the problem, which is the reason for conducting the large numerical simulation in the first place. There are still a few properties of the results that can be used to check their correctness.

Halo Mass Function
The halo mass function describes the number density of collapsed objects (dark matter haloes) at a given redshift z within a certain mass range M and $M + dM$ [14]. This function can be analytically approximated and therefore be used as a comparison to the numerical results.

Two-Point Correlation Function
The two-point correlation function $\chi(r)$ that has shortly been introduced in section 2.2 can be used as well to verify the simulated particle distributions. The function essentially quantifies how likely it is to find a particle in distance r to another particle. Determining this function also makes it easier to compare the results to other cosmological simulations and their outputs. Furthermore, it gives an idea at which scales the code works properly and describes the clumping correcltly.

6.2.2 P^{3}M Method or Adaptive Mesh

The PM method is the fasted scheme known to compute the gravitational field, however, it is unsuitable for high spatial resolution, as the force is suppressed for distances smaller than the mesh cell size [15]. It is only suitable for systems where the separations between the particles are larger. The PM method therefore works well at early times of the cosmological simulations where the distribution is close to uniform and even at later times, but only on large scales. The clustering of the particle due to gravity leads to particle densities which differ by orders of magnitude from cell to cell. In high-density regions the PM algorithm therefore fails to describe the detailed particle dynamics. The result is that haloes and individual galaxies are poorly resolved and no internal structure like the bulge or the disk can be observed.
One method would be to use an adaptive mesh refinement where high-density regions are described with a finer grid [3]. This method has already been used for cosmological simulation since it is very useful to describe regions of different densities [1] [18].
The other method would be to use a particle-particle-particle-mesh (P^{3}M) method where long-range forces are modeled with the PM method and short-range forces are directly calculated as particle-particle interactions.

6.2.3 Including Baryonic Matter

In this project, the dynamic behavior of dark matter was implemented. Using only dark matter particles in the simulation greatly simplified the dynamics. Dark matter is modeled as collisionless and only interacting via gravity. To simulate the formation of galaxies, the addition of baryonic matter in the simulations is crucial. Baryonic matter compared to dark matter involves many more complex mechanisms like cooling, viscosity effects, and pressure forces [14].

6.2.4 In-Situ Visualization

It has been suggested at the end of the project to add an in-situ visualization method to the simulation code. This would allow us to visualize the data while the simulation is still running without having to store all the results. Due to time constraints, this has not been implemented yet, but the feature has been tested for plasma physics applications in other Alpine mini-apps.

6.3 Unresolved Issues

6.3.1 Load Balancer

It has been observed that for larger systems, individual particles can "vanish"/"appear" when using load-balancing. Turning the load-balancer off helps to mitigate this problem can however lead to unbalanced workloads for different processors.
In this project turning off the load balancer has not caused any significant unbalance in workload since the particles in cosmological simulations (especially in the beginning) are almost uniformly distributed on large scales.

6.3.2 MPI Message Size

For large simulations (larger than 256^3) it has been observed that the MPI message size might exceed the maximum allowed size depending on the read-in method. This issue has already been addressed in section 3.3.2 and a solution has been provided. The solution (namely doing a selective read-in that makes the first update faster), works but is somewhat inefficient and could be resolved if the particles were generated directly in their domain instead of externally generating them and then reading them in.

A Generating Initial Conditions with N-GenIC

A.1 Installation

The N-GenIC code for this project was initially retrieved from the official GitHub of N-GenIC [6]. For this project an additional function to save the particle positions and velocities in a CSV file has been added to the source code of N-GenIC. The modified code can be retrieved on GitHub [7].
In order to compile to N-GenIC code, the following packages are required: `gcc`, `openmpi`, `cmake`, `gsl`, `hdf5` and `fftw`. For the `fftw` library, an older version, namely version 2 is required. An installation guide for this library is given below:

Installation of `FFTW 2.5`
Get the folder from their website: https://www.fftw.org/download.html

```
$ wget https://www.fftw.org/fftw-2.1.5.tar.gz
$ tar -xvzf fftw-2.1.5.tar.gz
$ cd fftw-2.1.5
$ ./configure --prefix=/data/user/crazzo_b --enable-shared --enable-mpi
$ make
$ make install
```

The library will be installed in the folder `/data/user/crazzo_b/lib`. Without the option `prefix` which specifies the installation location, the library will be installed at the computer's default location. If working on a computer cluster this might create permission errors, which is why the location is explicitly specified in this guide.

A.2 Compiling

In the Makefile in the `ngenic/` folder, the path of the `fftw` library must be specified if the library cannot be found at default installation locations. (The same goes for the `gsl` library). Then the code can be compiled.

```
$ cd ngenic
$ make
```

If the code is successfully compiled, the executable `N-GenIC` will be generated in `ngenic/`.

A.3 Running

The parameters for the initial conditions need to be specified in a parameter file. The default parameter file is `ics.param` and can be found in the `ngenic/parameterfiles/` folder. One parameter in the parameter files is the output directory, where the initial conditions are stored. Make sure, that the folder exists, before running the code.
The parameterfiles that have been used for this projected are included in the `parameterfiles/` folder. An example would be the file `lsf_32.param` which generates 32^3 particles and saves them in the folder `ngenic/ICs/` folder (specified in this parameter file). To generate the particles run the following command:

```
./N-GenIC parameterfiles/lsf_32.param
```

This will create the binary file `lsf_32` compatible with GADGET and the CSV file `lsf_32.csv` file compatible with IPPL. The code can also be run in parallel with the following command:

[6] https://gitlab.mpcdf.mpg.de/rwein/ngenic
[7] https://github.com/bcrazzolara/SimGadget/tree/master

```
$ mpiexec -np 4 ./N-GenIC parameterfiles/lsf_32.param
```

This will run the code on 4 processors in parallel. The initial conditions are then also split up into 4 different files with the corresponding processor number in their file name. In order to combine the CSV files into one file (and make it compatible with IPPL), the following command can be run in the output folder:

```
$ cat lsf_32.*.csv > Data.csv
```

B Simulating with GAGDET

The code for GADGET can be obtained from the GAGDET website by running the following commands:

```
$ wget https://wwwmpa.mpa-garching.mpg.de/gadget/gadget-2.0.7.tar.gz
$ tar -xvzf gadget-2.0.7.tar.gz
$ cd gadget-2.0.7
```

Alternatively, the code can be obtained from the combined directory together with N-GenIC and the parameter files from this project on GitHub [8].

B.1 Compiling

For compiling the code, the libraries as for N-GenIC are required (see section A.2). The Makefile in the folder `Gadget2` must specify the location of the `fftw` library if it cannot be found in the default locations. To compile the code, run the following commands:

```
$ cd Gadget2
$ make
```

After the successful compilation, the executable `Gadget2` will be generated.

B.2 Running a Simulation

Similarly to the N-GenIC, a parameter file with all relevant physical and computational parameters needs to be set. For simplicity, we use the same parameter file names for N-GenIC (saved in `SimGadget/ngenic/parameterfiles/`) and Gadget (saved in `SimGadget/Gadget2/parameterfiles/`). When using initial conditions generated with N-GenIC several values of the 2 parameter files must match. For example, the starting time of the simulation, called `TimeBegin` (otherwise it is physically wrong), all the cosmological constants like the density parameters (`Omega`), Hubble parameters, `BoxSize`, and the system units. The example files are already set up to match their initial condition files. Also here, the output directory for the simulation specified in the Gadget parameter files must exist before running the code.
To run the code the following command can be executed in the folder `SimGadget/`:

```
$ ./Gadget2/Gadget2 Gadget2/parameterfiles/lsf_32.param
```

In this example file, the input file from the folder `SimGadget/ngenic/ICs/` with the name `lsf_32` is acessed and the simulation results are saved in the folder `SimGadget/Results/lsf_32/`.
The code can also be executed in parallel with the command:

```
$ mpiexec -np 4 ./Gadget2/Gadget2 Gadget2/parameterfiles/lsf_32.param
```

which runs the code on 4 processors in parallel.

[8]https://github.com/bcrazzolara/SimGadget

C Cosmology Mini-App

The cosmology mini-app code can be obtained from GitHub [9]. This repository contains the source code of the `IPPL` library and the interface file of the mini-apps. The mini apps can be found in folder `ippl/alpine/`. While the files starting with `Gravity` are for class definitions and solver interfaces, the files `StructureFormation.cpp` and `StructureFormationManager.h` are the main files that have to be adapted if changes to the mini-app shall be implemented.

C.1 Compiling IPPL and the Mini-Apps

For the compilation, the build scripts from GitHub [10] can be used. It is recommended to load the build script repository and the IPPL source code repository into the same parent directory.

C.1.1 In Serial on CPUs

To compile the code in serial mode for CPUs run the following command:

```
$ ./ippl-build-scripts/999-build-everything -t serial --kokkos --heffte --ippl
```

This will build the executables in the folder `./ippl/build_serial/alpine/`.

C.1.2 In Parallel on CPU

Compile the code:

```
$ ./ippl-build-scripts/999-build-everything -t openmp --kokkos --heffte --ippl
```

This will build the executables in the folder `./ippl/build_openmp/alpine/`.

C.1.3 In Parallel on GPU

Compile the code:

```
$ ./ippl-build-scripts/999-build-everything -t cuda --kokkos --heffte --ippl
```

This will build the executables in the folder `./ippl/build_cuda/alpine/`.

C.2 Running the Code

When running the code, the directory of the input data file (which must have the filename `Data.csv` must be specified as an argument for the executable `StructureFormation`. Furthermore, the number of particles must be specified and this number must match the number of particles in the input file `Data.csv`. The starting, end time, and physical volume are specified in the `StructureFormationManager.h` file. The physical volume should match the physical boundaries of the input files.

C.2.1 In Serial on CPUs

For running the code in serial use the following commands:

```
$ cd ./ippl/build_serial/alpine/
$ ./StructureFormation data/lsf_32/ 32 32 32 32768 100 FFT 1.0 LeapFrog --overallocate
1.0 --info 5
```

This runs a simulation with input data in the folder `data/lsf_32/`, a mesh of size 32^3 (same in all directions), with $32^3 = 32\,768$ particles for 100 timesteps.

[9]https://github.com/bcrazzolara/ippl/tree/cosmology
[10]https://github.com/bcrazzolara/ippl-build-scripts

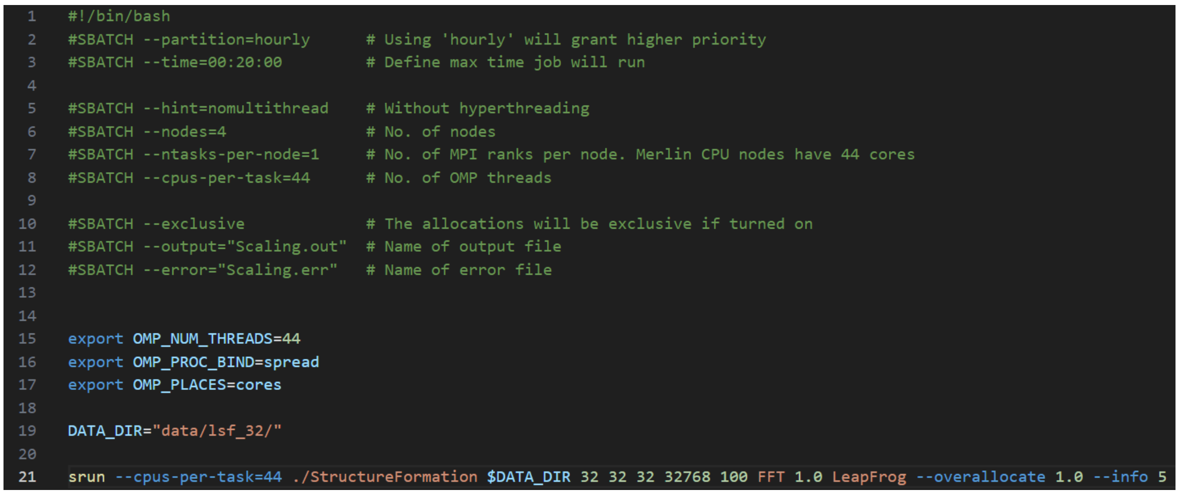
C.2.2 In Parallel on CPU

Use the following command to directly the code in parallel using two processors:

```
$ cd ./ippl/build_openmp/alpine/
$ srun -n 2 ./StructureFormation data/lsf_32/ 32 32 32 32768 100 FFT 1.0 LeapFrog --overallocate
1.0 --info 5
```

To give more specific parameters to the parallelization options, one can use jobscripts. An example is given in Fig. 17. For strong scaling studies in the projects on Merlin or Euler, the scripts have been uploaded to GitLab [11].
With a batch script with file name `jobscript` the following commands would be used for running:

```
$ cd ./ippl/build_cuda/alpine/
$ sbatch jobscript
```



```
#!/bin/bash
#SBATCH --partition=hourly      # Using 'hourly' will grant higher priority
#SBATCH --time=00:20:00         # Define max time job will run

#SBATCH --hint=nomultithread    # Without hyperthreading
#SBATCH --nodes=4               # No. of nodes
#SBATCH --ntasks-per-node=1     # No. of MPI ranks per node. Merlin CPU nodes have 44 cores
#SBATCH --cpus-per-task=44      # No. of OMP threads

#SBATCH --exclusive             # The allocations will be exclusive if turned on
#SBATCH --output="Scaling.out"  # Name of output file
#SBATCH --error="Scaling.err"   # Name of error file


export OMP_NUM_THREADS=44
export OMP_PROC_BIND=spread
export OMP_PLACES=cores

DATA_DIR="data/lsf_32/"

srun --cpus-per-task=44 ./StructureFormation $DATA_DIR 32 32 32 32768 100 FFT 1.0 LeapFrog --overallocate 1.0 --info 5
```

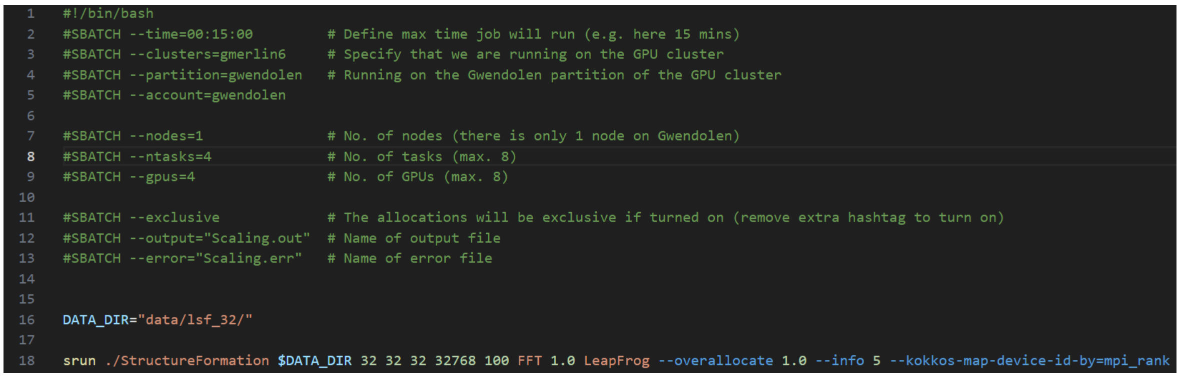
Figure 17: Example jobscript for parallel CPU execution on Merlin.

C.2.3 In Parallel on GPU

To run the code on GPUs (in this project the computer cluster Gwendolen was used), only batch scripts have been used. They are uploaded on GitLab as well. An example is given in Fig. 18. For strong scaling studies in the projects on Gwendolen the scripts have been uploaded to GitLab.
With a batch script with file name `jobscript` the following commands would be used for running:

```
$ cd ./ippl/build_cuda/alpine/
$ sbatch jobscript
```

[11] https://gitlab.psi.ch/AMAS-students/crazzolara-sem



```bash
#!/bin/bash
#SBATCH --time=00:15:00         # Define max time job will run (e.g. here 15 mins)
#SBATCH --clusters=gmerlin6     # Specify that we are running on the GPU cluster
#SBATCH --partition=gwendolen   # Running on the Gwendolen partition of the GPU cluster
#SBATCH --account=gwendolen

#SBATCH --nodes=1               # No. of nodes (there is only 1 node on Gwendolen)
#SBATCH --ntasks=4              # No. of tasks (max. 8)
#SBATCH --gpus=4                # No. of GPUs (max. 8)

#SBATCH --exclusive             # The allocations will be exclusive if turned on (remove extra hashtag to turn on)
#SBATCH --output="Scaling.out"  # Name of output file
#SBATCH --error="Scaling.err"   # Name of error file


DATA_DIR="data/lsf_32/"

srun ./StructureFormation $DATA_DIR 32 32 32 32768 100 FFT 1.0 LeapFrog --overallocate 1.0 --info 5 --kokkos-map-device-id-by=mpi_rank
```

Figure 18: Example jobscript for parallel GPU execution on Gwendolen.

C.3 File Overview

StructureFormationManager.h	Most important file for changes in the simulation parameters. The initial time, end time, and physical volume of the simulation are defined here. The current parameters match the parameter files and initial conditions that have been used in this project. Two read-in functions (`readParticles` and `readParticlesDomain`) are implemented (section 3.3.2) are implemented. The current default is the selective read-in that assigns the particles to their correct processor domain (`readParticlesDomain`). In the `LeapFrogStep` function, the saving frequency of the particle attributes can be adjusted.
StructureFormation.cpp	This file contains the main function of the mini-app. There should be no need to change this file significantly.
GravityFieldContainer.hpp	contains the field container for the gravitational field, density field, and potential field.
GravityFieldSolver.hpp	Contains the function that solves the Poisson equation on the field.
GravityLoadBalancer.hpp	This file contains functions related to load-balancing like repartition functions.
GravityManager.h	This file contains all important functions and parameters that describe the simulation. Functions to calculate the Hubble parameter, redshift, retrieve and set simulation parameters, scattering, and gathering are defined.
GravityParticleContainer.hpp	The particle attributes of the particle container are defined in this file.

D Commands for Linux and Git

- **Cloning a Git repository**
```
$ git clone <remote_repo>
```
- **Cloning a specific branch**
```
$ git clone -b <branch> <remote_repo>
```
- **Searching for a specific file**
```
$ find .  -name "test*.h"
```
- **Searching for a word in all files**
```
$ grep -r "<word>"
```
- **Copying a folder and its content "here"**
```
$ cp -r ~/ngenic/ICs .
```
- **Copying file into a folder (keeping the same file name)**
```
$ cp example.txt <folder>
```
- **Copying a file into a folder (changing the file name)**
```
$ cp example.txt <folder>/file.txt
```
- **Transfering a file from a local machine to the cluster**
```
$ cd /mnt/c/Users/blanc/Downloads
$ scp -r fftw-2.1.5 crazzo_b@merlin-l-002.psi.ch:  ~/data
```
- **Transfering a file from the cluster to the local machine**
```
$ cd /mnt/c/Users/blanc/Downloads
$ scp crazzo_b@merlin-l-002.psi.ch:  <folder>/file.csv .
```
  or
```
$ scp merlin_ext:  <folder>/file.txt .
```
- **Installation of a C++ library at a certain location**
```
$ ./configure --prefix=/usr/local/myapp
$ make
$ make install
```
- **Installing a python library at a certain location**
```
$ pip install -target /usr/libpy <package>
```
- **Showing the biggest files in current directory**
```
$ du -aBM 2>/dev/null | sort -nr | head -n 50 | more
```
- **Connecting to PSI (after 08/2024)**
```
$ ssh crazzo_b@hopx.psi.ch
```
  Password + Authenticator
  In new Terminal:
```
$ ssh -J crazzo_b@hopx.psi.ch crazzo_b@merlin-l-001.psi.ch
```
  Password + Authenticator + Password

References

[1] T. Abel, G. L. Bryan, and M. L. Norman. The formation of the first star in the universe. *Science*, 295(5552), 2002.

[2] A. Ayala, S. Tomov, A. Haidar, and J. Dongarra. heFFTe: Highly efficient fft for exascale. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12137 LNCS, 2020.

[3] A. Brandt. On Distributed Gravitational N-Body Simulations. 2022.

[4] D. J. Croton, V. Springel, S. D. M. White, G. De Lucia, C. S. Frenk, L. Gao, A. Jenkins, G. Kauffmann, J. F. Navarro, and N. Yoshida. Erratum - The many lives of AGN: cooling flows, black holes and the luminosities and colours of galaxies. *Monthly Notices of the Royal Astronomical Society*, 365(1), 2006.

[5] M. Frey, A. Vinciguerra, S. Muralikrishnan, S. Mayani, V. Montanaro, and A. Adelmann. IPPL-framework/ippl: IPPL-3.1.0, 2024.

[6] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles.* 1988.

[7] A. Jenkins, C. S. Frenk, F. R. Pearce, P. A. Thomas, J. M. Colberg, S. D. M. White, H. M. P. Couchman, J. A. Peacock, G. Efstathiou, and A. H. Nelson. Evolution of Structure in Cold Dark Matter Universes. *The Astrophysical Journal*, 499(1), 1998.

[8] A. Klypin, F. Prada, and J. Byun. Suppressing cosmic variance with paired-and-fixed cosmological simulations: Average properties and covariances of dark matter clustering statistics. *Monthly Notices of the Royal Astronomical Society*, 496(3), 2020.

[9] S. Mayani, V. Montanaro, A. Cerfon, M. Frey, S. Muralikrishnan, and A. Adelmann. A Massively Parallel Performance Portable Free-space Spectral Poisson Solver, 2024.

[10] H. Mo, F. van den Bosch, and S. White. *Galaxy Formation and Evolution.* 2010.

[11] S. Muralikrishnan, M. Frey, A. Vinciguerra, M. Ligotino, A. J. Cerfon, M. Stoyanov, R. Gayatri, and A. Adelmann. Scaling and performance portability of the particle-in-cell scheme for plasma physics applications through mini-apps targeting exascale architectures. In *Proceedings of the 2024 SIAM Conference on Parallel Processing for Scientific Computing (PP).* 2024.

[12] J. F. Navarro, C. S. Frenk, and S. D. M. White. The Structure of Cold Dark Matter Halos. *The Astrophysical Journal*, 462, 1996.

[13] S. Perlmutter, G. Aldering, G. Goldhaber, R. A. Knop, P. Nugent, P. G. Castro, S. Deustua, S. Fabbro, A. Goobar, D. E. Groom, I. M. Hook, A. G. Kim, M. Y. Kim, J. C. Lee, N. J. Nunes, R. Pain, C. R. Pennypacker, R. Quimby, C. Lidman, R. S. Ellis, M. Irwin, R. G. McMahon, P. Ruiz-Lapuente, N. Walton, B. Schaefer, B. J. Boyle, A. V. Filippenko, T. Matheson, A. S. Fruchter, N. Panagia, H. J. M. Newberg, W. J. Couch, and T. S. C. Project. Measurements of Ω and Λ from 42 High-Redshift Supernovae. *The Astrophysical Journal*, 517(2), 1999.

[14] S. Planelles, D. R. Schleicher, and A. M. Bykov. Large-Scale Structure Formation: From the First Non-linear Objects to Massive Galaxy Clusters, 2015.

[15] V. Springel. The cosmological simulation code GADGET-2, 2005.

[16] V. Springel. User guide for GADGET-2. *https://wwwmpa.mpa-garching.mpg.de/gadget/users-guide.pdf*, 2005.

[17] V. Springel, S. D. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce. Simulations of the formation, evolution and clustering of galaxies and quasars. *Nature*, 435(7042), 2005.

[18] R. Teyssier. Numerical and analytical predictions for the large-scale Sunyaev-Zel'dovich effect. *Physical Review D - Particles, Fields, Gravitation and Cosmology*, 66(4), 2002.

[19] F. Y. Wang and Z. G. Dai. Weak gravitational lensing effects on cosmological parameters and dark energy from gamma-ray bursts. *Astronomy and Astrophysics*, 536, 2011.

[20] Y. B. Zeldovich. Gravitational Instability: An Approximate Theory for Large Density Perturbations. *Astronomy & Astrophysics*, 5, 1970.

Acknowledgement

I want to express my gratitude to all those who made this amazing project possible. Firstly, I want to thank Dr. Andreas Adelmann, the group leader of the AMAS group at PSI. He designed this thesis for me and matched it to my interests extremely well. His ongoing support through this project motivated me a lot and the frequent exchanges about work and life made this project successful. Secondly, I would like to thank my supervisor Sonali Mayani. Like Andreas, she has been an incredible supervisor to me and helped me a lot to solve problems. I am grateful for the weekly brain-storm sessions and discussions the three of us had. Overall, my experience in the AMAS group was great. Many members have supported me during the project, helped me to learn more, and made me feel very welcome.
Lastly, I want to thank everyone else who contributed to the project, my family, and my friends who always support me in my life and show interest in my work.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. In consultation with the supervisor, one of the following three options must be selected:

○ I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].

○ I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used and cited generative artificial intelligence technologies[2].

[✓ I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used generative artificial intelligence technologies[3]. In consultation with the supervisor, I did not cite them.]

Title of paper or thesis:

Cosmological Structure Formation with the Performance Portable IPPL Library

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	**First name(s):**
Crazzolara	Blanca

With my signature I confirm the following:
- I have adhered to the rules set out in the Citation Guide.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date	**Signature(s)**
04.07.2024	[illegible]

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] E.g. ChatGPT, DALL E 2, Google Bard
[2] E.g. ChatGPT, DALL E 2, Google Bard
[3] E.g. ChatGPT, DALL E 2, Google Bard