

pyC²Ray: A flexible and GPU-accelerated Radiative Transfer Framework for Simulating the Cosmic Epoch of Reionization

Patrick Hirling^a, Michele Bianco^{a,c,*}, Sambit K. Giri^b, Ilian T. Iliev^c, Garrett Mellema^d, Jean-Paul Kneib^a

^a*Institute of Physics, Laboratory of Astrophysics, Ecole Polytechnique Fédérale de Lausanne (EPFL), Observatoire de Sauverny, Versoix, 1290, Switzerland*

^b*Nordita, KTH Royal Institute of Technology and Stockholm University, Hannes Alfvéns väg 12, Stockholm, SE-106 91, Sweden*

^c*Astronomy Centre, Department of Physics & Astronomy, Pevenssey III Building, University of Sussex, Falmer, Brighton, BN1 9QH, United Kingdom*

^d*The Oskar Klein Centre, Department of Astronomy, Stockholm University, AlbaNova, Stockholm, SE-10691, Sweden*

Abstract

Detailed modeling of the evolution of neutral hydrogen in the intergalactic medium during the Epoch of Reionization, $5 \leq z \leq 20$, is critical in interpreting the cosmological signals from current and upcoming 21-cm experiments such as the Low-Frequency Array (LOFAR) and the Square Kilometre Array (SKA). Numerical radiative transfer codes provide the most physically accurate models of the reionization process. However, they are computationally expensive as they must encompass enormous cosmological volumes while accurately capturing astrophysical processes occurring at small scales (\lesssim Mpc). Here, we present pyC²Ray, an updated version of the massively parallel ray-tracing and chemistry code, C²-Ray, which has been extensively employed in reionization simulations. The most time-consuming part of the code is calculating the hydrogen column density along the path of the ionizing photons. Here, we present the Accelerated Short-characteristics Octahedral ray-tracing (ASORA) method, a ray-tracing algorithm specifically designed to run on graphical processing units (GPUs). We include a modern Python interface, allowing easy and customized use of the code without compromising computational efficiency. We test pyC²Ray on a series of standard ray-tracing tests and a complete cosmological simulation with volume size $(349 \text{ Mpc})^3$, mesh size of 250^3 and approximately 10^6 sources. Compared to the original code, pyC²Ray achieves the same results with negligible fractional differences, $\sim 10^{-5}$, and a speedup factor of two orders of magnitude. Benchmark analysis shows that ASORA takes a few nanoseconds per source per voxel and scales linearly for an increasing number of sources and voxels within the ray-tracing radii.

Keywords: Radiative Transfer, Epoch of Reionization, ray-tracing, GPU methods, 21-cm, Cosmology, Intergalactic medium

1. Introduction

The Epoch of Reionization (EoR) is a period of significant interest in the history of the Universe, as it marks the appearance of the very first sources of radiation that drove the transition of the intergalactic medium (IGM) from its primordial cold and neutral state to the present-day hot and highly ionized one (see e.g. Furlanetto et al., 2006; Gorbunov and Rubakov, 2011; Dayal and Ferrara, 2018, for reviews about this era). While indirect observational evidence, such as using high redshift quasar spectra (e.g. Bosman et al., 2022) and the cosmic microwave background (CMB) radiation (e.g. Planck Collaboration et al., 2020), situates the EoR at redshifts between about 5 and 30, its main characteristics are still unknown (Pritchard and Loeb, 2012; Barkana, 2016). Current and upcoming interferometric radio telescopes, such as the Low-Frequency Array (LOFAR; van Haarlem et al., 2013), Hydrogen Epoch of Reionization Array (HERA; DeBoer et al., 2017), Murchison Wide-field Array (MWA; Wayth et al., 2018) and Square Kilometre Array (SKA; Mellema et al., 2013), are expected to uncover the details of this key event in cosmic history by detecting the distribution of the redshifted 21-cm signal in the IGM, produced

by the spin-flip transitions in neutral hydrogen (Pritchard and Loeb, 2012; Zaroubi, 2013). Accurate modeling of the EoR, which is needed to interpret the observational constraints provided by these experiments, will require performing detailed numerical radiative transfer (RT) and radiation hydrodynamics (RHD) studies on large cosmological scales ($\gtrsim 100$ Mpc). These simulations are challenging because the EoR is a non-local process, and the underlying RT equation contains both angular, spatial, and frequency dimensions. Various modeling methods exist, a review of which may be found in, e.g., Gnedin and Madau (2022).

Today, most fully numerical RT codes can be divided into two main classes: moment-based and ray-tracing methods. The former works by considering the hierarchy of angular moments of the RT equation, with some ‘closure relation’ to limit the number of equations to be solved, and treat the radiation as a fluid (e.g. Aubert and Teyssier, 2008). This makes coupling to hydrodynamics natural and, from a computational perspective, has the huge benefit of being independent of the number of ionizing sources in the simulation. On the other hand, moment methods suffer from increased diffusion and unrealistic shadows on optically thick objects. A few examples of codes using moment-based methods are OTVET (Gnedin and Abel, 2001), RAMSES-RT (Rosdahl et al., 2013) and AREPO-RT (Kan-

*Corresponding author

Email address: mb2158@sussex.ac.uk (Michele Bianco)

nan et al., 2019). Although they combine N-body, hydrodynamic and radiative feedback, they tend to be computationally expensive and cannot simulate the required large volumes. Iliev et al. (2014) show that we require simulations with a minimum volume size of ~ 100 cMpc to model the 21-cm signal and cover the large field of view expected by SKA and its precursors (e.g. Mertens et al., 2020; Trott et al., 2020; HERA Collaboration, 2023). Moreover, the small mass sources ($\sim 10^8 M_\odot$), which are expected to drive reionization (e.g. Nebrin et al., 2023; Gelli et al., 2023; Atek et al., 2024), are often not resolved by these simulations.

Ray-tracing methods take a more physical approach by casting *rays* around each source and modeling how the radiation propagates, i.e., is absorbed and scattered, along those rays. The photo-ionization rate occurring at any point in space is then determined by the number of absorptions between the source and said point, normally expressed as the optical depth between the source and that point. This approach can potentially be more accurate and less diffusive than moment methods but is quite expensive, as the cost of ray-tracing generally scales linearly with the number of radiating sources. Thus, in practice, the number of sources that can be considered has been, until recent years, severely limited by the available computational power. C^2 -Ray (Mellema et al., 2006), ZEUS-MP (Whalen and Norman, 2006), CRASH (Ciardi et al., 2001), SPHRAY (Altay et al., 2008), LICORNE (Semelin et al., 2007), ART (Nakamoto et al., 2001), FLASH-HC (Rijkhorst et al., 2006) are a few notable examples of ray-tracing-based codes. Moment and ray-tracing methods have been compared extensively (Iliev et al., 2006, 2009). The main differences are due to numerical diffusion for the different treatments of the energy equation in moment-based methods and how the multi-frequency radiation is implemented. The advantages of using one over the other have been shown to depend greatly on the problem and context. That being said, by requiring huge volumes and large numbers of ionizing sources (Kaur et al., 2020; Giri et al., 2023), developing more efficient RT methods for EoR, especially ray-tracing-based ones, is highly desirable.

In recent years, there has been a significant surge in the use of general-purpose GPUs for numerical scientific research. These devices have enabled remarkable performance improvements when used to develop applications for problems that can be divided into numerous simple and independent tasks suitable for parallel processing. Consequently, GPU acceleration has been integrated in various astrophysics and cosmological-related tools (e.g. Ocvirk et al., 2016; Potter et al., 2016; Rácz et al., 2019; Cavelan et al., 2020; Wang and Meng, 2021). The ATON (Aubert and Teyssier, 2010) and EMMA (Aubert et al., 2015) codes are the first applications of GPU-accelerated algorithms for radiative transfer codes in the context of extragalactic astrophysics. To our knowledge, this technology has not yet been imported to short-characteristic ray-tracing methods, thus making the current work a first.

Given the success of GPUs in accelerating ray-tracing tasks in computer graphics (Owens et al., 2008; Nickolls and Dally, 2010; Navarro et al., 2014), it is reasonable to explore their application to ray-tracing problems in astrophysics. This mo-

tivates our work, where we introduce an Accelerated Short-characteristics Octahedral ray-tracing (ASORA) method designed specifically for C^2 -Ray. By incorporating GPU methods, we anticipate significant performance enhancements and more efficient simulations, thus opening up new possibilities for research and analysis. Our work aims to bridge the gap between the potential of GPU acceleration and the requirements of ray-tracing tasks in astrophysics, providing a promising avenue for further advancements in this domain.

C^2 -Ray is a 3D ray-tracing radiative transfer code designed for simulating the EoR and was initially developed by Mellema et al. (2006) (hereafter: M06). It conserves photons at a voxel-by-voxel level, allowing for large, optically thick grid voxels while maintaining accuracy. Furthermore, the method allows for long time steps, even surpassing the voxel-crossing time of ionization fronts. It has been extensively used in EoR simulations and updated to include photoheating, X-ray radiation, and helium chemistry (Friedrich et al., 2012; Ross et al., 2017, 2019). C^2 -Ray is written in Fortran90 and designed for massively parallel systems, utilizing a hybrid MPI and OpenMP approach for efficient radiation propagation. The ionizing sources are distributed over MPI processes, and each of these processes further employs OpenMP threading to propagate radiation in a domain-decomposed manner.

As a stand-alone code, C^2 -Ray is a *post-processing* code¹. It acts on the output snapshots of a previous (cosmological) hydrodynamical simulation and propagates radiation on the gas fields of these snapshots. As is detailed in the following sections, it is also a grid code, meaning that the gas fields must be projected onto this grid through some smoothing method. Sources are identified in the initial simulation via a variety of models. Currently, the typical approach is to run a halo finder on each snapshot and use a physical model to translate a halo into a radiating source. The update to C^2 -Ray in this work comprises two main aspects:

1. GPU-Accelerated ray-tracing Method: The original ray-tracing method used by C^2 -Ray is not well-suited for GPU parallelization. A new algorithm based on the same short-characteristics scheme has been developed to address this limitation. This new method is specifically designed for running on GPUs, enabling efficient computation of column densities, which is the most computationally intensive task in the radiative transfer (RT) method. The GPU implementation leverages massive multi-threading capabilities, resulting in significantly faster performance than the CPU method. This new algorithm is written as a C++/CUDA (e.g. Garland et al., 2008) library with Python bindings for ease of use and integration.

2. Python Wrapper and Interface: The highly-optimized Fortran90 implementation of C^2 -Ray excels at computationally intensive and time-consuming tasks, such as the solving of chemistry equations and, until now, ray-tracing. However, due to its compiled and statically typed

¹The algorithm can, however, be used in conjunction with a hydrodynamics code, as was, for example, done in Arthur et al. (2011) and Medina et al. (2014)

nature, Fortran is less suited for all the parts of the code that require frequent tweaking, such as the radiation source implementation, interfacing, I/O operations, cosmological model, and more generally the setup of each particular simulation. These tasks contain most of the conceptual baggage of future simulations but only represent a negligible fraction of the computational workload. Thus, to enhance usability and flexibility, we decided to wrap the time-critical core Fortran subroutines of C^2 -Ray and rewrite the non-time-consuming parts of the code in Python, making frequent use of standard libraries.

As a result, users can now write an entire C^2 -Ray simulation as a Python script, making it easier to tweak parameters and add new features without frequently recompiling the core Fortran subroutines. These updates enable more efficient GPU utilization for critical computations and improve the overall accessibility and versatility of the C^2 -Ray code through Python scripting and interface enhancements.

This paper is structured as follows. In § (2), we describe how reionization is modeled and summarize how the C^2 -Ray method works. In § (3), we describe the ray-tracing method used, present our newly developed ASORA algorithm, and briefly discuss the new Python wrapping and interface to the code. Then, in § (4), the updated code is tested on standard idealized situations and benchmarked to determine how much performance improvement is achieved. The source code of py C^2 Ray is publicly available at <https://github.com/cosmic-reionization/pyC2Ray>.

2. Simulating Cosmic Reionization

To study the EoR, we need to model the time evolution of the ionization state of the intergalactic medium (IGM) within a cosmological framework. This involves solving a system of chemistry equations that track the evolution of the ionization state of primordial species, such as hydrogen and helium. These equations take into account various physical processes, including photoionization, collisional excitation, recombination, heating, and cooling (e.g. Furlanetto et al., 2006).

In this paper, we will focus on the simplest case, considering only hydrogen. This choice is justified because hydrogen constitutes the major part of the IGM. The original C^2 -Ray code includes extensions also to consider helium ionization and multi-frequency photo-heating (Friedrich et al., 2012), and we plan to incorporate these extensions into py C^2 Ray gradually. The primary objective of this paper is to present an update to the general ray-tracing method.

The ionization state of the hydrogen gas is described by the following *chemistry equation* (e.g. Choudhury and Ferrara, 2006; Choudhury, 2009),

$$\frac{dx_{\text{HII}}}{dt} = (1 - x_{\text{HII}})(\Gamma + n_e C_{\text{H}}(T)) - x_{\text{HII}} n_e \alpha_{\text{H}}(T), \quad (1)$$

where x_{HII} is the fraction of ionized hydrogen, n_e is the electron number density, Γ is the photo-ionization rate per unit time,

and $C_{\text{H}}(T)$ and $\alpha_{\text{H}}(T)$ are the collisional ionization and recombination coefficients for ionized hydrogen and free electrons, at temperature T . C^2 -Ray uses the on-the-spot (OTS) approximation, which assumes that the diffused photons resulting from recombination to the ground state are reabsorbed locally and, thus, solely accounted for by using a different value for α_{H} (e.g., Ritzerveld, 2005).

The photo-ionization rate Γ quantifies the effect of ionizing UV radiation on the gas and is determined by the distribution of radiation sources. To illustrate this point, consider the simple situation of a single isotropic ionizing source in a homogeneous medium. As photons propagate away from the source in all directions, they form a spherical "shell" of ionizing radiation. The photons are absorbed by gas particles, which subsequently become ionized. These photo-ionizations also attenuate the strength of the radiation further away from the source, in addition to the attenuation occurring due to geometrical effects alone. Photo-ionization is also countered by recombinations. Together, these phenomena result in the formation of a spherical ionized bubble around the source, also known as a Strömgren sphere.

In EoR simulations, more than one source is typically present, and the medium is distinctly *inhomogeneous*, leading to a much more complicated situation. The number of these sources during the EoR depends critically on the size of the volume and minimum mass of source haloes. We typically start with less than a few hundred sources at high redshift ($z \gtrsim 20$) to a few tens of a million at low redshift ($z \approx 6$). Below, we first summarize the method used in C^2 -Ray to solve Equation 1, and then discuss in detail the computation of the photoionization rates.

2.1. Summary of the C^2 Ray Method

To solve the chemistry equation (Equation 1), one could, in principle, use a finite-differencing scheme and assume all rates to be constant over a reasonably short timestep. This approach is used by, e.g., Grackle (Smith et al., 2017) to solve very complex chemistry networks. The problem here lies in the photoionization rate Γ . It is determined by the amount of ionizing photons arriving at the target point where Equation 1 is considered. This amount depends directly on how radiation is absorbed along the path from its source to the target point, which in turn depends on the density n_{H} and ionization state x_{HII} of the medium along this path. This means that Γ is strongly dependent on the solution variables of the problem and that this dependence is also highly non-local. For the finite-differencing scheme to be accurate, this implies very stringent constraints on the timestep size, especially in the presence of fast-moving ionization fronts (I-fronts).

C^2 -Ray overcomes this problem using an alternative approach, illustrated schematically in Figure 1. As is argued in M06, when recombinations and collisional ionizations are neglected, the solution of Equation 1 over any timestep Δt depends only on the time-averaged photoionization rate within that timestep, denoted by $\langle \Gamma \rangle$. Furthermore, only small deviations arise when collisions and recombinations are included, as is tested in M06. The idea behind the C^2 -Ray algorithm

is to converge to the correct $\langle \Gamma \rangle$ within a given Δt by iterating between a *Ray-tracing Step*, which computes $\langle \Gamma \rangle$ based on the currently assumed solution for the time-averaged ionization state $\langle x_{\text{HII}} \rangle$ of the whole medium, and a *Chemistry Step*, which computes an updated $\langle x_{\text{HII}} \rangle$ based on the new $\langle \Gamma \rangle$. This is illustrated in Figure 1 by the long vertical black arrow, which goes through a convergence test to determine whether the iteration needs to be repeated.

The chemistry step itself is not entirely trivial, as it still relies on being able to solve the differential equation. The method used in C^2 -Ray is based on Schmidt-Voigt and Koeppen (1987), who argue that when n_e , Γ , C_H and α_H are assumed to be constant, an analytical solution exists for Equation 1. Using this solution, the time-averaged ionization state can be expressed as

$$\langle x \rangle = x_{eq} + (x_0 - x_{eq})(1 - e^{-\Delta t/t_i}) \frac{t_i}{\Delta t} \quad (2)$$

$$x_{eq} = \frac{\Gamma + n_e C_H}{\Gamma + n_e (C_H + \alpha_H)} \quad (3)$$

$$t_i = [\Gamma + n_e (C_H + \alpha_H)]^{-1}. \quad (4)$$

Here, x_0 is the ionization state at the beginning of the timestep, and x_{eq} is the equilibrium solution, i.e., for $dx_{\text{HII}}/dt = 0$, while t_i is a constant time scale employed for the time-averaged inhomogeneous solution. Note also that Γ has been used instead of $\langle \Gamma \rangle$ to represent the time-averaged photoionization rate to ease up notation. Since the non-time-averaged rate is never used in the algorithm, this new notation shall be used from now on. C^2 -Ray uses this solution, and iterates for the electron density n_e (which depends on x_{HII} through roughly $n_e \sim x_{\text{HII}} n_H$), until $\langle x_{\text{HII}} \rangle$ converges. The thick horizontal arrow within the chemistry step illustrates this second iterative process in Figure 1. Again, a convergence criterion is implicitly used to determine when to end the iteration.

The ray-tracing step requires further consideration, as it is the focus of the present work. It is discussed in detail in §2.2 and §3.1. The main takeaway from this section is that the C^2 -Ray method makes it possible to use very long timesteps while still remaining accurate even in the presence of fast-moving I-fronts.

2.2. Computing Rates

In the "ray-tracing step" introduced above, the properties of ionizing sources are used together with the knowledge of the ionization state of the medium inside a simulation volume to compute the photo-ionization rate Γ occurring at any point in space. An ionizing source of specific luminosity L_ν located at a point \vec{p}_\star produces at a target point \vec{p} a (time-averaged) photoionization rate given by

$$\Gamma = \frac{1}{4\pi r^2} \int_{\nu_{th}}^{\infty} \frac{L_\nu \sigma_\nu e^{-\langle \tau_\nu \rangle}}{h\nu} d\nu, \quad (5)$$

where ν_{th} is the threshold frequency for photoionization ($h\nu_{th} = 13.6$ eV), $\langle \tau_\nu \rangle \equiv \tau_\nu(r)$ is the time-averaged optical depth between source and target and $r = |\vec{p}_\star - \vec{p}|$ is the distance between \vec{p}_\star and \vec{p} . The optical depth is proportional to the column density of neutral hydrogen N_{HI} , and the proportionality

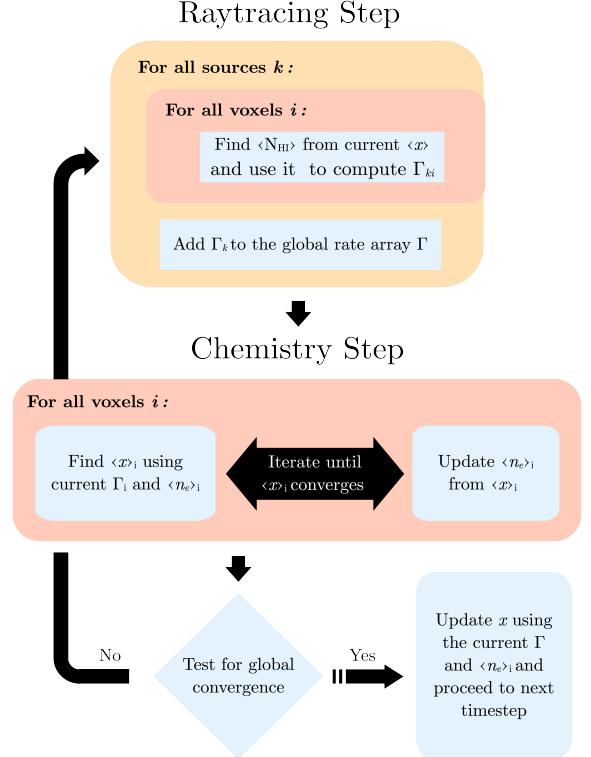


Figure 1: Flowchart representation of the method used by C^2 -Ray. The figure shows the procedure for a single time-step in which the ionized fraction of hydrogen x_{HI} is evolved for the whole 3D grid. The method can be divided into a "ray-tracing" and a "chemistry" step, and multiple iterations of either typically occur in a single time step.

factor is its frequency-dependent photoionization cross-section, $\tau_\nu = \sigma_\nu N_{\text{HI}}$. The frequency-dependence of σ_ν approximately follows a power law whose index depends on the frequency band considered (see, e.g. Friedrich et al., 2012, for further details).

C^2 -Ray in its current form is a Cartesian grid code, which discretizes space using N^3 cubic voxels, where N is the number of voxels in each dimension. Using Equation 5 directly on a grid is problematic when the voxels are optically thick, i.e. the optical depth of a single voxel is non-negligible. The photoionization rate will then vary appreciably from one side of a voxel to the other. Using Equation 5 computed at an arbitrary point in the voxel as a representative rate for the whole voxel will, therefore, lead to an error in photon conservation - the number of ionizations occurring in the voxel will not be equal to the number of absorptions. To avoid this problem without being forced to use impractically small voxels, C^2 -Ray works by *imposing* that the number of ionizations is equal to the number of absorptions used to attenuate the radiation. As is detailed in M06, using this condition leads to an alternative expression for the photoionization rate,

$$\Gamma = \int_{\nu_{th}}^{\infty} \frac{L_\nu e^{-\langle \tau_\nu \rangle} (1 - e^{-\Delta \tau_\nu})}{h\nu n_{\text{HI}} V_{\text{shell}}} d\nu, \quad (6)$$

where $\Delta \tau_\nu$ is the optical depth *through* the voxel, which is proportional to the light travel path length ds through the voxel,

and $\langle \tau_v \rangle$ is the optical depth *up to* the voxel. n_{HI} is the number density of neutral hydrogen inside the voxel, while the factor $V_{\text{shell}} = 4\pi r^2 ds$ accounts for both geometrical diffusion of radiation and the finite size of the cell. Note that, in the optically thin limit ($\Delta\tau_v \rightarrow 0$), the above expression reduces to Equation 5. By defining the function

$$\gamma(N_{\text{HI}}) \equiv \int_{\nu_{th}}^{\infty} \frac{L_{\nu} e^{-\sigma_{\nu} N_{\text{HI}}}}{h\nu} d\nu, \quad (7)$$

Equation 6 can be written in a more suggestive way

$$\Gamma = \frac{1}{n_{\text{HI}} V_{\text{shell}}} [\gamma(N_{\text{HI}}) - \gamma(N_{\text{HI}} + \Delta N_{\text{HI}})]. \quad (8)$$

This means that rather than numerically solving the integral in Equation 6 each time it is required, the function $\gamma(N_{\text{HI}})$ can be pre-calculated and tabulated for a range of column densities and a simple interpolation used to evaluate it for any given value of N_{HI} . Note that the individual properties of the voxel where Γ is computed are not part of the tabulation and are explicitly accounted for in Equation 8.

When more than one source is present, the situation becomes slightly more complicated. The approach described in the original C²-Ray paper (see Figure 4 in M06) involves randomizing the order of sources and performing the chemistry step for each source individually before testing global convergence. However, this approach was modified in subsequent updates to the code, and it is this updated algorithm that we use here. The idea is simply to compute the 3D rate array (one rate per voxel) for each source and sum these arrays to obtain a global rate array. This global rate is then used for the chemistry step, and the process is repeated until convergence. Note that this is the process as illustrated in Figure 1, where we use the notation Γ_{ki} to signify that this quantity applies to a given source indexed by k and a given voxel, indexed by i .

3. Novel ray-tracing Method: ASORA

As shown by Equation 6, the problem of finding ionization rates boils down to computing the column density N_{HI} of neutral hydrogen between a source and grid voxels. This is the process we refer to as *ray-tracing* in this context. In principle, given a cubic grid with N voxels in each dimension, it is possible to compute N_{HI} directly for all voxels of the grid, an approach known as “long characteristics” (LC). For a single source, it scales as $O(N^4)$. This is because, for each of the N^3 voxels to treat, the number of other voxels that lie along the ray coming from the source is on the order $O(N)$. LC has the advantage of being easy to parallelize as all rays are treated independently. However, given that radiation propagates causally outward from the source and that column density is an additive quantity along a given line of sight, this algorithm also contains a lot of redundancy. A variety of methods have been proposed to make ray-tracing more efficient (see, e.g. Rosdahl et al., 2013, for an overview). C²-Ray uses a version of the “short-characteristics” (SC) ray-tracing method (Raga et al., 1999), which reduces the redundancy of the problem by using

interpolation from inner-lying voxels relative to the source to compute the column density to outer-lying ones. This method reduces the complexity to $O(N^3)$ but is harder to parallelize as it introduces voxel dependency.

Since the effect of each source is independent, the total cost of the ray-tracing step is the number of sources N_{src} times whatever the cost for a single source is, e.g., $O(N^4)$ for LC or $O(N^3)$ for SC. On the other hand, the total cost of a chemistry step only scales with the number of voxels in the grid, i.e., N^3 . This clarifies why the ray-tracing step is the primary target for optimization in an EoR code like C²-Ray, where typically $N_{\text{src}} \gg 1$. In fact, at low redshift, it is common to have a source in almost every voxel, so that $N_{\text{src}} \sim N^3$, and that, in turn, the complexity of the ray-tracing step is $\sim O(N^6)$ (using SC) versus $O(N^3)$ for the chemistry step.

We should also mention that we never separately treat more than one source per voxel and instead simply add the luminosity of all sources whenever more than one is present, implying $N_{\text{src}} \leq N^3$. The result is identical to treating them separately and adding up the resulting rates because a source is always assumed to be at the center of a voxel. Note, however, that this approach is possible only because the spectra of all sources are identical, and in the future, when different source types may be considered by the code, this procedure will have to be adapted by only summing up the sources that belong to the same type.

Below, we first discuss in detail the short-characteristics ray-tracing method used in C²-Ray (§ 3.1). Then, we give an overview of the CPU-parallelization strategy the code has used so far (§ 3.2). Next, we introduce the adaptation of the method for GPUs (§ 3.3), and finally, in § 3.4, we discuss the structure of the new Python wrapper built around C²-Ray.

3.1. Ray-tracing in C²Ray

Here, we closely follow the discussion of Appendix A in M06, and in particular, refer the reader to Figure A1, which provides a good visual description of the geometric arguments detailed below. For a voxel located at mesh position $p = (i, j, k)$ and a source at $s = (i_s, j_s, k_s)$, the full column density $N_{\text{HI}} = \tilde{N}_{\text{HI}} + \Delta N_{\text{HI}}$ along the ray from s to p can be decomposed into a part *up to* the voxel \tilde{N}_{HI} and a part *within* the voxel ΔN_{HI} . The latter is proportional to the physical path length $dl = \alpha ds$ through the voxel at p , where ds is the path length in mesh units and α is the physical length of a grid voxel,

$$\Delta N_{\text{HI}} = n_{\text{HI}} \alpha ds. \quad (9)$$

Defining $\Delta i = i - i_s$ (and similarly Δj and Δk), one can determine through which plane the ray coming from s enters the voxel at p . For example, if $\Delta k > \Delta i$ and $\Delta k > \Delta j$, the ray enters through one of the constant- z planes, with the Δk sign indicating which one. In this particular case, the path length through the voxel is

$$ds = \sqrt{1 + \frac{\Delta i^2 + \Delta j^2}{\Delta k^2}}, \quad (10)$$

and the analogous expressions apply if the ray enters through the constant- x or y plane. The main assumption of the short-characteristics method is that \tilde{N}_{HI} can be computed by interpolation with neighboring voxels of p that are closer to s . The

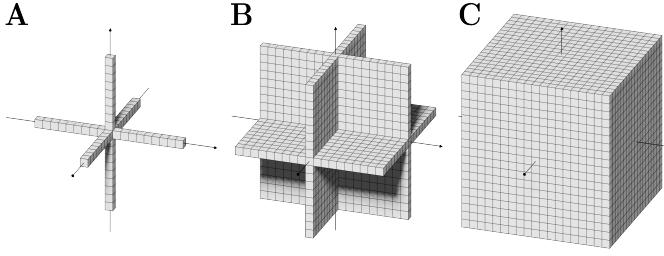


Figure 2: Parallelization strategy used by the original C^2 -Ray code. In the first step (A), 6 grid domains can be treated independently, corresponding to axes around the source voxel. In (B), the 12 planes joining them form independent domains, while in the third one (C), the 8 octants between the planes do.

particular scheme used by C^2 -Ray (Raga et al., 1999) uses 4 neighbours, whose positions are given by

$$\begin{aligned} e_1 &= (i, j, k - \sigma_k), & e_2 &= (i, j - \sigma_j, k), \\ e_3 &= (i - \sigma_i, j, k), & e_4 &= (i - \sigma_i, j - \sigma_j, k - \sigma_k) \end{aligned} \quad (11)$$

where $\sigma_{i,j,k} = \frac{|\Delta i, j, k|}{\Delta i, j, k}$. The interpolated column density up to p then reads

$$\tilde{N}_{\text{HI}} = w_1 N_{e_1} + w_2 N_{e_2} + w_3 N_{e_3} + w_4 N_{e_4}. \quad (12)$$

The interpolation weights w_n are a simple geometric weighting based on the xy -distance from the corner to the point of intersection between the ray and the surface of the cell. They are chosen such that when the ray is parallel to an axis or lies on a grid diagonal, in which case \tilde{N}_{HI} is exactly equal to the column density of only one of the neighbors, all but the weight of that neighbor vanish. The interested reader is referred to Appendix A in M06 for the details of this weighting choice.

The above scheme describes how the column density up to a given voxel can be approximated using the knowledge of the equivalent quantity corresponding to 4 other voxels that lie closer to the source. This inter-voxel dependency naturally implies that for the scheme to be applied correctly, one must treat the voxels in a particular order, starting at the source voxel and moving outward from there. This ensures that the interpolation step does not attempt to use information that doesn't exist yet, so we say that SC is a *causal* algorithm. In fact, the simplest way to traverse the grid is to simply perform a triple loop over the $x \rightarrow y \rightarrow z$ indices of all voxels by starting the loop at the source voxel indices. This is a fully sequential approach. The next two sections deal with the problem of finding parallel alternatives to the latter.

3.2. Existing CPU Parallelization and Optimizations

The current version of C^2 -Ray uses various methods to optimize the cost of ray-tracing and make the procedure scalable to massively parallel CPU systems. A key feature of the code is that the treatment of each source is completely independent. C^2 -Ray harnesses this independence by distributing the full list of sources between MPI ranks. Each rank receives a copy of

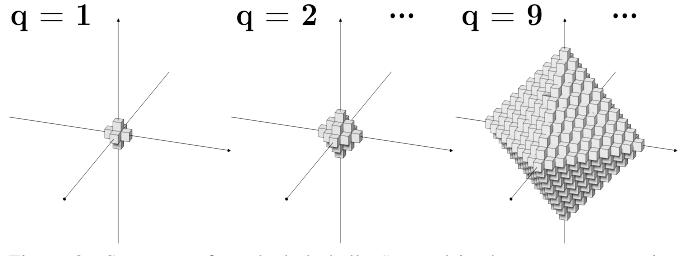


Figure 3: Sequence of octahedral shells S_q used in the ASORA ray-tracing method. All voxels belonging to a shell S_q , with $q > 0$, depend strictly on voxels from previous shells $\{S_r \mid r < q\}$. The shells $q = 1$, $q = 2$ and $q = 9$ are shown, with the source voxel ($q = 0$) at the origin of the axes.

the full grid data and works on a subset of the sources. It performs ray-tracing for each source in this subset and sums together their respective ionization rate arrays (see §2.2). Then, an MPI reduction operation is used to sum the rate arrays of all ranks and obtain a global Γ that includes the contributions of all sources. This allows the full ray-tracing workload to be distributed over many processors in shared and distributed memory setups. The main limitation of this setup is memory since each rank carries a full copy of the 3D grid. As was explained in §3.1, the ray-tracing work for a single source is more challenging to do in parallel due to the inter-voxel dependency of the SC method. However, it is possible to find independent subdomains of the grid and use an approach similar to domain decomposition. This approach performs the following steps in order, which are illustrated in Figure 2:

1. Do the 6 axes outward from the source voxel (**A**) in parallel
2. Do the 12 planes joining these axes (**B**) in parallel
3. Do the 8 octants between the planes (**C**) in $x \rightarrow y \rightarrow z$ order, in parallel,

where the labels **A**, **B** and **C** correspond to the three sketches in Figure 2. C^2 -Ray uses OpenMP tasks to do the independent domains following this approach, which can yield a speedup of $S \lesssim 8$. Finally, the ray-tracing procedure itself is optimized using the following technique: rather than ray-tracing the whole grid, the program first treats only a cubic sub-region around the source, namely a "sub-box", and then calculates the total amount of radiation that leaves this sub-box (i.e. a photon loss). If this loss is above a given threshold, the program increases the size of the sub-box, treats the additional voxels and repeats this procedure until the photon loss is low enough. This allows C^2 -Ray to avoid expensively ray-tracing all voxels when, in fact, almost no radiation reaches the ones far away from the source. The threshold value should be chosen based on convergence studies of the type of problem being simulated. The sub-box technique has been found to work well in EoR settings, where the density field is almost Gaussian. In some more specific situations, where narrow, optically thin tunnels exist in otherwise optically thick regions, the technique might produce inaccurate results. In these cases, using a very small threshold value or, in the worst case, ray-tracing the whole grid may be desirable. Additionally, the user can impose a hard limit on the maximum distance any photon can reach relative to the source.

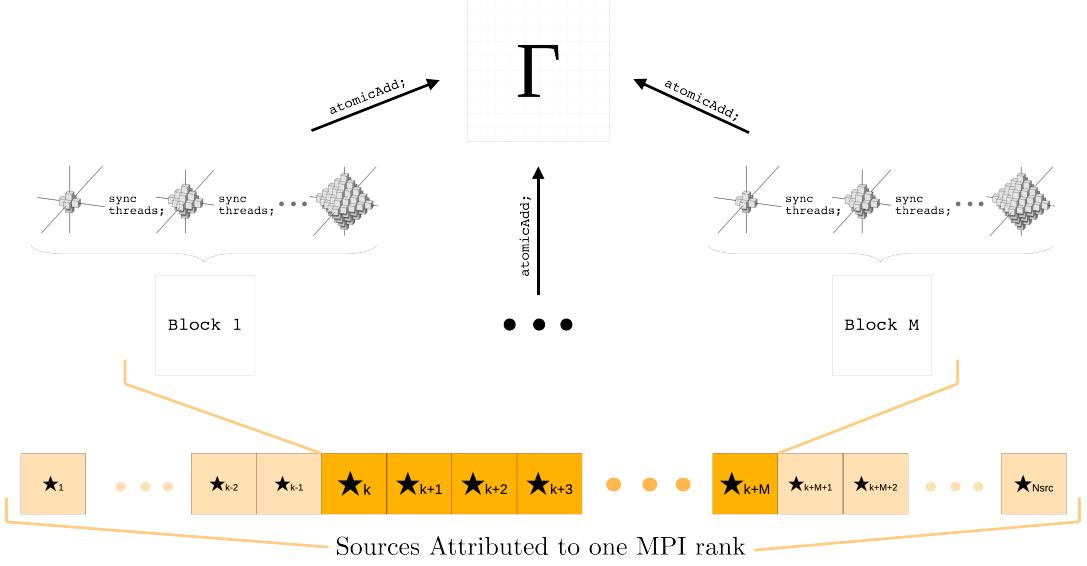


Figure 4: Implementation of the ASORA method. N_{src} sources, labeled by \star_i , are treated in batches of a given size M , and one block is dispatched for each source in the batch. Threads within a block are synchronized between each shell S_q (see Figure 3) but are independent across different sources. Each block atomically contributes to the global ionization rate array. In MPI mode, each rank independently follows the above framework and the Γ arrays of all ranks are sum-reduced to the root.

3.3. GPU Implementation

GPUs are designed to execute numerous concurrent operations, organized into units referred to as *blocks* in CUDA and *workgroups* in AMD terminology. Given that ASORA has been implemented using CUDA, we will continue to use CUDA terminology. We are also planning a future port of the library for AMD platforms. Threads can be synchronized within a block, while blocks run asynchronously (Nickolls et al., 2008). It is possible to perform a synchronization between blocks only globally. To fully harness the resources of a GPU, one aims to ensure that the number of threads active at any given time is as close as possible to the theoretical maximum of the used device. While no universal prescription exists to achieve this, it is generally desirable that blocks have a similar workload and their number is in the same order as the number of streaming multiprocessors (SMs) available on the device. This suggests a natural implementation for the ray-tracing problem: dispatch one block for each source and use intra-block synchronization to respect the causality of the short characteristics algorithm.

For this approach to be efficient, however, the work for a single source cannot be simply parallelized following the domain decomposition approach described in § 3.2 as this would allow at most 8 threads to be active within a block. To parallelize the work for a single source in a way more suited for the capabilities of a GPU, we recall that radiation would propagate as a spherical wavefront around a point source in a continuous medium. This translates to a series of shells around a source voxel in the discretized setting. It turns out that there is a particular sequence of disjoint shells S_q , illustrated in Figure 3, which are causally ordered with respect to the SC scheme used by C²-Ray. q indexes the "distance" of the shell to the source; the $q = 0$ shell is simply the source voxel itself, and $q = 1$ contains the 6 directly adjacent voxels to the source. The causal

ordering can be summarized by the following conditions:

1. The first shell $q = 0$ contains only the source voxel, which can be treated directly without interpolation.
2. For any voxel $p \in S_q$ with $q > 0$, all 4 interpolation neighbors appearing in Equation 11 belong strictly to shells S_r with $r < q$, in other words, only to shells "below" the current one.
3. In particular, all voxels $p \in S_q$ are independent of one another with respect to the interpolation scheme.

This means that the full ray-tracing work for a single source can be divided into the sequence of tasks $\{S_q\}_{q=0}^Q$, where Q is the size of the largest shell. These Q tasks must be done sequentially by definition, but each task comprises subtasks (one subtask for each voxel in the shell) that can be performed in parallel. Note that the number of voxels inside a shell S_q , and hence the number of independent subtasks per task, is $n_q = 4q^2 + 2$. Going back to the discussion above, when, for instance, 100 threads are assigned per source for each task with $q \geq 5$, it is theoretically possible for all threads to be actively engaged in performing work. This effectively resolves the challenge of parallelizing the computation on a per-source basis.

Rather than giving a maximal shell size Q , it is more convenient to set a maximum physical radius R_γ any photon can travel from the source. If the physical size of a grid voxel is, as previously, denoted by α , the (dimensionless) size index of the largest shell required to cover the chosen radius fully is given by $Q = \lceil \frac{R_\gamma}{\alpha \sqrt{3}} \rceil$. Any cell inside S_Q whose distance to the source exceeds R_γ can simply be excluded from the computation to yield a spherical region in which Γ is nonzero.

The full implementation, illustrated in Figure 4, goes as follows. We dispatch one CUDA thread block for each source that works through the sequence of shells. The result, i.e. the

photo-ionization rate produced by that source in each voxel, is *atomically* added to the global rate array Γ . In practice, there is a small additional caveat to consider, namely that by the nature of the algorithm, each source requires a temporary memory space to store the values of the previously interpolated voxels needed for the next interpolation. The required space can typically be a good fraction of the whole grid, so the number of blocks that can be dispatched together is limited by GPU memory. In fact, rather than directly dispatching one block for each of the N_{src} sources in the simulation, we group the sources into batches of size M , and work on these batches one after the other. M is determined by available GPU memory and grid size N . As long as M is large enough to saturate the GPU, this approach should not result in a significant performance loss compared to the ideal scenario of immediately deploying one block per source, without any batching, since the workload for each source is the same.

Finally, ASORA is also MPI-enabled, using `mpi4py` (Dalcin and Fang, 2021) in the same way as it was intended for the original C^2 -Ray. Namely, the sources are evenly distributed to multiple MPI processes. Each MPI rank maps to one GPU, which then uses the model laid out above to process its subset of sources and broadcasts the result Γ to the root rank, using `MPI_REDUCE` with a sum operation. This allows using ASORA on a multi-GPU setup across multiple nodes to further speed up ray-tracing on very heavy workloads.

To conclude this section, we note that the ASORA method, as presented here, only applies to uniform grids, as the octahedral shell approach builds on this assumption. We acknowledge that this is a strong limitation of our method, and we plan to explore its adaptability to non-uniform grids. Further technical details on ASORA can be found in Appendix A.

3.4. Python-wrapping of C^2 -Ray

Here, we provide a brief overview of the `pyC2Ray` interface and architecture, summarized visually in Figure 5. The package amalgamates key components from the original `Fortran90` code, the new ray-tracing library as discussed above, and elements of pure Python. This integration is facilitated through `f2py`, a tool developed as part of the `NumPy` project (Harris et al., 2020). This tool streamlines the creation of extension modules from `Fortran90` source files.

The incorporated `Fortran90` subroutines primarily encompass the chemistry solver and retain the original CPU-based ray-tracing module as a contingency. The novel ASORA method is written in `C++/CUDA` and compiled as a Python extension module natively compatible with `NumPy`. The principal time-evolution function within `pyC2Ray` is implemented in Python, and it invokes the ray-tracing method, choosing between the CPU and GPU versions and the chemistry method sourced from these extension modules. The prior process of precalculating photoionization rate tables, as introduced earlier, has transitioned to direct implementation in Python. This is achieved using numerical integration techniques from the `SciPy` library (Virtanen et al., 2020), which relies on the underlying `QUADPACK` library for lower-level computations. It is worth noting that these integration methods differ from the

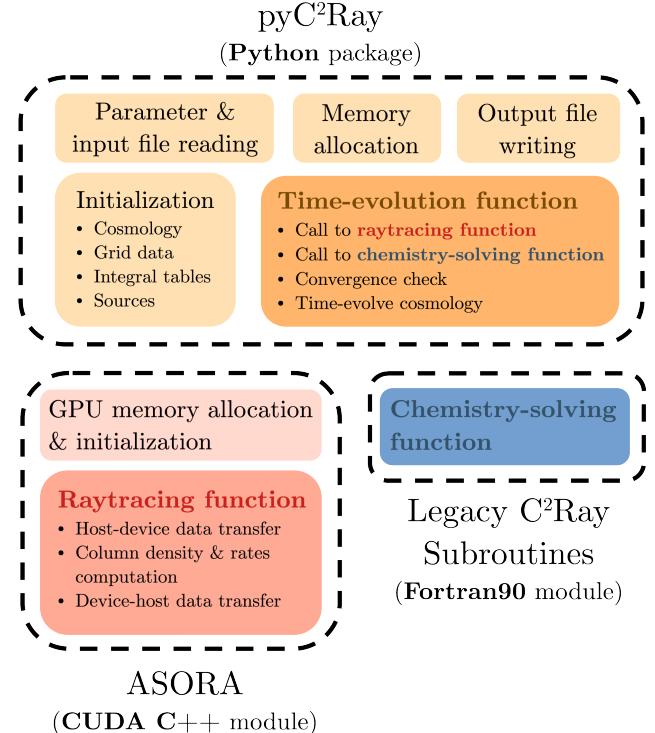


Figure 5: Structure of the `pyC2Ray` code. The main python package, `pyc2ray`, sets up the simulation and acts as the front end to the user. Internally, the time-evolution method of this package executes functions from two compiled extension modules. One is `ASORA`, the new GPU ray-tracing module written in CUDA C++, while the other contains a set of wrapped `Fortran` subroutines taken and adapted from the original C^2 -Ray code.

custom Romberg integration subroutines utilized by the original C^2 -Ray framework. The commonly needed cosmological equations and physical quantities are now provided by `Astropy` (Astropy Collaboration, 2022).

Beyond these technical aspects, the inherent method within `pyC2Ray`—apart from the ray-tracing component—has undergone minimal alteration. Key features of C^2 -Ray, including photoionization and hydrogen chemistry, have been seamlessly migrated to the Python version without compromising computational efficiency. Our strategy involves a gradual integration of additional extensions over time.

4. Validation Testing & Benchmarking

In § 4.1, we validate our new code using a series of well-established tests, comparing our results to analytical solutions and to the results of our original C^2 -Ray code. In § 4.2, we investigate how the updated ray-tracing method scales relative to the main problem parameters. In all tests, the temperature conditions of the gas are assumed to be isothermal, i.e., no heating effects are modeled.

4.1. Accuracy Tests

We begin by conducting Tests 1 and 4 from M06, labeled as *Test 1* and *2* here, to evaluate the precision of our code in

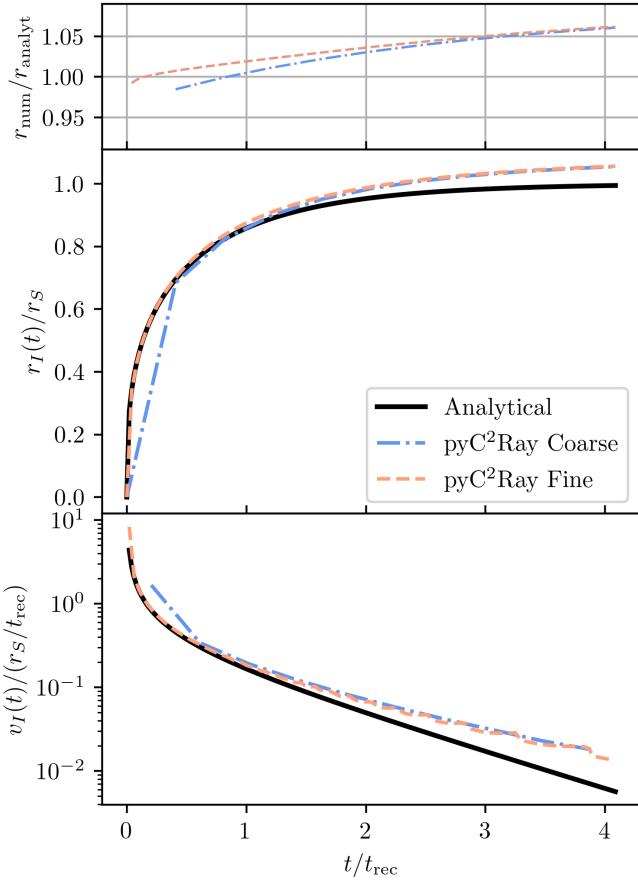


Figure 6: Result for Test 1 (Single-source H II region expansion in uniform gas). The test is conducted with a "coarse" time step $\Delta t_c = t_{\text{evo}}/10$ and a "fine" one, $\Delta t_f = t_{\text{evo}}/100$. The time evolution of the ionization front radius (middle) and velocity (bottom) are shown. The error between the numerical and analytical results can be seen in the top panel.

monitoring I-fronts in single-source mode. This evaluation encompasses scenarios both with and without cosmological background expansion. Following this, we investigate the interplay among multiple sources and the occurrence of shadow formation behind an opaque object, *Test 3* and *Test 4*.

4.1.1. Test 1: Single-Source HII Region Expansion

Consider the classical scenario of a single ionizing source within an initially-neutral, uniformly dense field at a constant temperature. In this case, any cosmological effects are disregarded. Assuming the photoionization cross section remains frequency-independent, $\sigma_\nu = \sigma_0$, known as *grey opacity*, this system has a well-established analytical solution for the velocity and radius of the ensuing ionization front with respect to time. The solution is given by

$$r_I(t) = r_S [1 - \exp(-t/t_{\text{rec}})]^{1/3} \quad (13)$$

$$v_I(t) = \frac{r_S}{3 t_{\text{rec}}} \frac{\exp(-t/t_{\text{rec}})}{[1 - \exp(-t/t_{\text{rec}})]^{2/3}}. \quad (14)$$

The above expressions depend on the Strömgren sphere radius r_S , recombination time t_{rec} and luminosity emitted by the source (or the number of photons per unit time). These quantities are

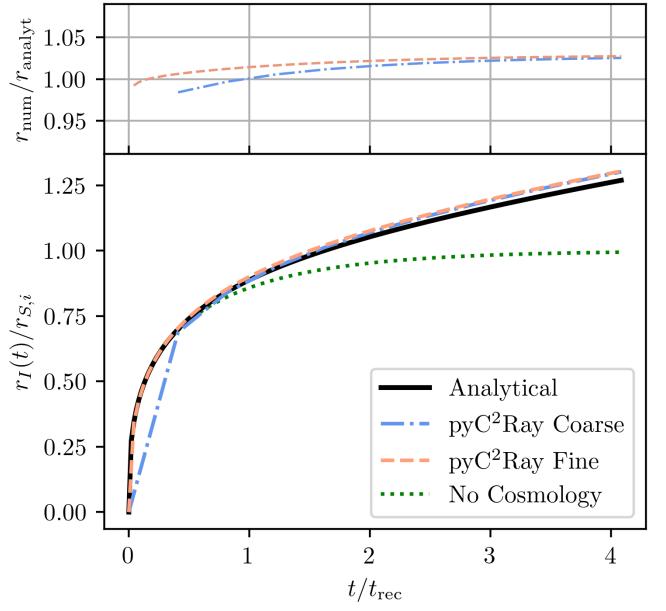


Figure 7: Result for Test 2 (Single-source HII region expansion in cosmological expanding background). Notation is the same as in Figure 6. The source turns on at $z_i = 9$, and the I-front radius is given in comoving kpc, with the scale factor $a(t_i) = 1$, normalized to the instantaneous Strömgren radius at z_i , $r_{S,i}$.

The green dotted line shows the analytical result without cosmological expansion for reference.

defined as,

$$r_S = \left(\frac{3 \dot{N}_\gamma}{4 \pi \alpha_H(T) n_H^2} \right)^{1/3}, \quad (15)$$

$$t_{\text{rec}} = \frac{1}{\alpha_H(T) n_H}, \quad (16)$$

$$\dot{N}_\gamma = \int_{\nu_{\text{th}}}^{\infty} \frac{L_\nu}{h\nu} d\nu. \quad (17)$$

Here, L_ν is again the specific luminosity of the source (power per unit frequency), which is related to the luminosity \dot{N}_γ (number of ionizing photons per unit time) through Equation 17. We conduct our first test using the following numerical parameters: the luminosity of the source is $\dot{N}_\gamma = 10^{48} \text{ s}^{-1}$, the number density of hydrogen $n_H = 10^{-3} \text{ cm}^{-3}$, its temperature $T = 10^4 \text{ K}$ and the simulation box size is 10 kpc. As stated above, we use the case B recombination coefficient for Hydrogen, $\alpha_H(T = 10^4 \text{ K}) = 2.59 \times 10^{-13} \text{ cm}^3 \text{ s}^{-1}$. Using these parameters, the recombination time is $t_{\text{rec}} \approx 122.35 \text{ Myr}$ and the Strömgren radius is $r_S = 3.15 \text{ kpc}$. The simulation is run with mesh size 256^3 for $t_{\text{evo}} = 500 \text{ Myr} \approx 4t_{\text{rec}}$, following the prescription of Test 1 in Iliev et al. (2006). As in M06, the simulation is repeated once with a coarse time step $\Delta_t = 50 \text{ Myr}$ and once with a fine one, $\Delta t = 5 \text{ Myr}$. We track the position of the I-front along the x -axis and define $r_I(t_k)$ as the radius where $x_{\text{HI}} = 0.5$. The precise location within a voxel is found by linear interpolation. The numerical I-front velocity, v_I , is found by finite-differencing r_I , using the same approach as in M06.

The results are shown in Figure 6, where the three panels contain the time evolution of the ratio between numerical to an-

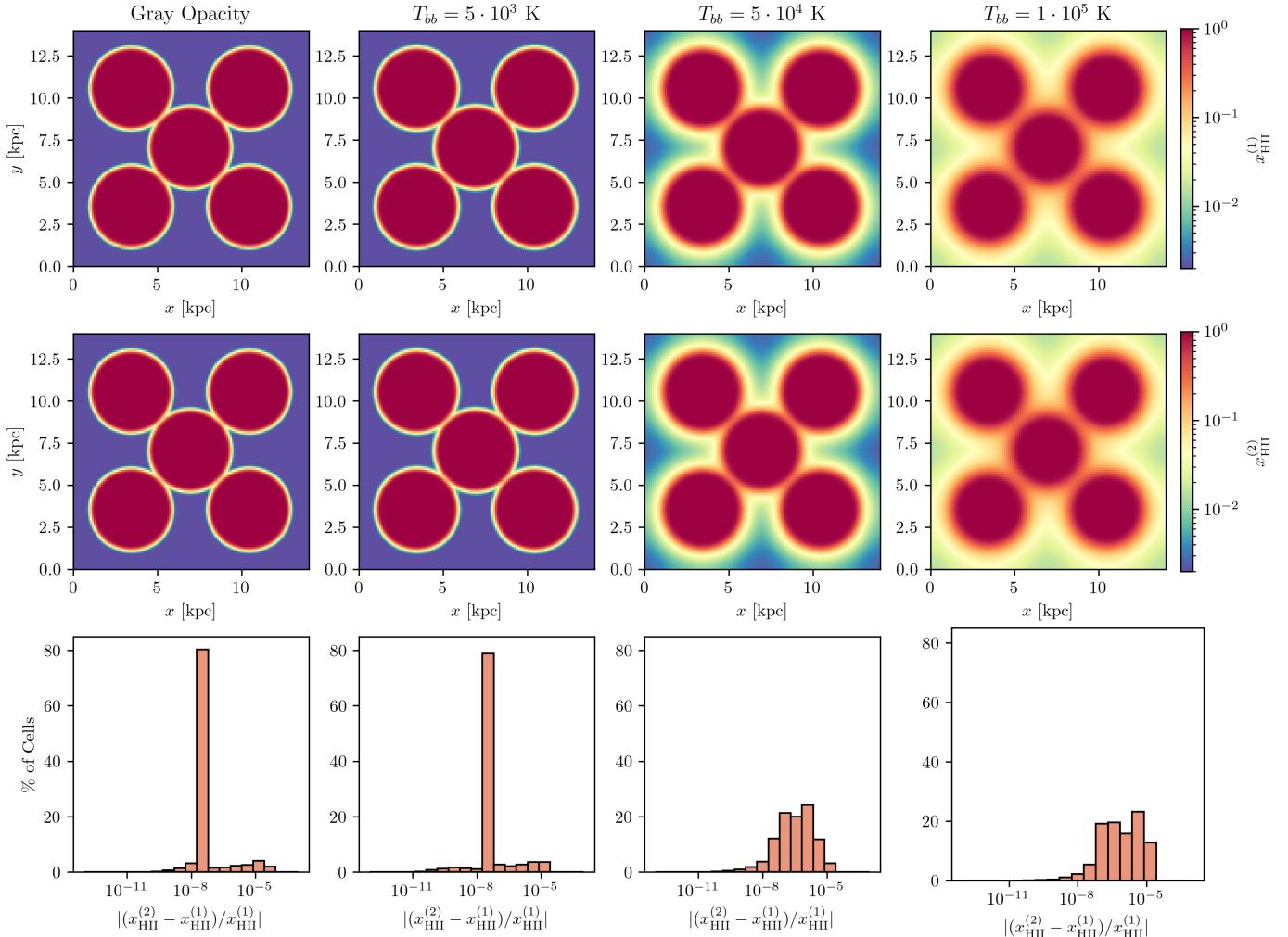


Figure 8: Result for Test 3 (Expansion of Overlapping H II regions around Multiple Black-Body Sources). The top and middle rows show slices through the simulation domain at the z -coordinate of the 5 sources, for \mathcal{C}^2 -Ray and $\text{py}\mathcal{C}^2\text{Ray}$ respectively. The leftmost column corresponds to the case with grey opacity, and the remaining 3 columns to those where black body spectra with different temperatures T_{bb} were used. Colors are normalized across each row. The bottom row shows the distribution of relative per-voxel errors between the 2 codes for the whole 3D grid in all 4 cases.

alytical results (top), the I-front radius (middle) and its velocity (bottom). At times $t \lesssim t_{\text{rec}}$, $\text{py}\mathcal{C}^2\text{Ray}$ is in excellent agreement with the analytical prediction, both with a coarse and a fine time step choice. At $t \gtrsim t_{\text{rec}}$, the numerical I-front overestimates the analytical prediction by as much as 6%. This is consistent with the findings of, e.g., Iliev et al. (2006), where all tested codes predict such an overestimate. Pawlik and Schaye (2008) have demonstrated that this is because, in reality, the ionized fraction varies smoothly within the ionized bubble, whereas the Strömgren argument assumes a sharp transition from fully ionized to fully neutral.

4.1.2. Test 2: Single-Source HII Region in expanding background

We next test if $\text{py}\mathcal{C}^2\text{Ray}$ correctly models the propagation of I-fronts in an expanding universe. Test 2 uses the same source parameters as Test 1, with the source turning on at $z = 9$ and then shining for 500 Myr, while the background density starts with the same value as before and evolves with the expansion of the universe. Shapiro and Giroux (1987) showed

that a generalized analytical solution exists in this case. The comoving I-front radius is given by $r_I(t) = r_{S,i} y(t)^{1/3}$, where $r_{S,i} = (3\dot{N}_\gamma / 4\pi\alpha_H(T)n_{H,i}^2)^{1/3}$ is the instantaneous Strömgren radius at the ignition time, t_i , of the source (with the scale factor set to unity at t_i , $a_i = 1$), and

$$y(t) = \lambda e^{\lambda t_i/t} \left[\frac{t}{t_i} E_2(\lambda t_i/t) - E_2(\lambda) \right], \quad (18)$$

where $E_2(x) = \int_1^\infty t^{-2} e^{-xt} dt$ is the second-order exponential integral. $\lambda = t_i/t_{\text{rec},i}$ is the ratio of the age of the universe at source ignition to the recombination time at that age. We set up the test with $n_{H,i} = 1.87 \times 10^{-4} \text{ cm}^{-3}$ and $L_i = 7 \times 10^{24} \text{ cm}$ and using otherwise the same parameters as before. The result is shown in Figure 7, where $r_I(t)$ represents the comoving I-front radius, keeping in mind that $a(t_i) = 1$. For this test, we used the same cosmology as in M06, namely $h = 0.7$, $\Omega_M = 0.27$ and $\Omega_b = 0.043$.

$\text{py}\mathcal{C}^2\text{Ray}$ again shows excellent agreement with the analytical result. While the effect of cosmic expansion is not evident at

first sight, the analytical prediction without cosmology, Equation 13, is also plotted for reference in the figure (green dotted line), and the difference is clearly visible. Again, results are almost as accurate when using a coarse time step.

4.1.3. Test 3: Expansion of Overlapping HII regions around Multiple Black-Body Sources

Now we turn to the more realistic case of non-grey opacity and parameterize the cross section as $\sigma_\nu = \sigma_0(\nu/\nu_0)^{-\alpha}$, where ν_0 is the ionization threshold frequency. The parameters of the power law are as in M06, $\sigma_0 = 6.3 \times 10^{18} \text{ cm}^{-2}$ and $\alpha = 2.8$. We test how the ionization front is affected by the spectral characteristics of the sources. For harder spectra, where the energy peak is well above the ionization threshold, we expect wider ionization fronts, as the hard photons can penetrate deeper into the medium (Spitzer, 1998). To test this and at the same time visualize how different HII regions overlap, we place 5 black-body sources, each with total ionizing flux $\dot{N}_\gamma = 5 \times 10^{48}$ but with different temperatures T_{bb} , in a dice-like pattern on the same z -plane. The box size is $L = 14 \text{ kpc}$, the mesh 128^3 and the constant hydrogen density is $n_H = 10^{-3} \text{ cm}^{-3}$. We simulate for $t_{evo} = 10 \text{ Myr}$, with time step $\Delta t = 1 \text{ Myr}$. Figure 8 shows cuts through the source plane of the final ionized hydrogen fraction x_{HII} , for pyC²Ray (top) and C²-Ray (middle), along with the distribution of the *absolute relative error*, $\left| (x_{\text{HII}}^{\text{pyC}^2\text{Ray}} - x_{\text{HII}}^{\text{C}^2\text{-Ray}}) / x_{\text{HII}}^{\text{C}^2\text{-Ray}} \right|$, between the two coeval cubes (bottom panels). The leftmost column is the grey-opacity case as in the two previous tests, while the three remaining columns contain the results for $T_{bb} = \{5 \times 10^3, 5 \times 10^4, 1 \times 10^5\} \text{ K}$.

Qualitatively, both C²-Ray and pyC²Ray reproduce the expected softness of ionization fronts for hot spectra, and the overlap of individual H II regions is also correctly modeled. The largest value for the relative error is on the order 10^{-4} in all cases, while the mean increases for harder spectra. Although relatively small, this error requires an explanation, as both codes should, in principle, produce equal results in the absence of unit conversion or floating point errors. In fact, an important technical difference between the two is the choice of numerical integration method used to pre-compute Equation 7 as described in §2.2. pyC²Ray uses the standard quad wrapper of the SciPy package (Virtanen et al., 2020), which uses the adaptive quadrature method from the QUADPACK Fortran library. On the other hand, C²-Ray uses a custom-written Romberg integration scheme. Both methods are valid choices, but they will inevitably yield slightly different results depending on the chosen resolution. We tested this by varying the frequency bins used by the Romberg method in C²-Ray and found that the relative error between the two codes drops significantly as this number increases. We thus conclude that this technical difference is the most likely explanation for this result.

4.1.4. Test 4: I-Front Trapping in a Dense Clump and Formation of a Shadow

Finally, to probe more specifically the ray-tracing method, we test for the formation of a shadow behind an overdense region. Correct modeling of shadows is one of the key advantages

of ray-tracing over other techniques, making this an important check. In this test, the box size is $L = 14 \text{ kpc}$ with mesh 128^3 and a source with total ionizing flux $\dot{N}_\gamma = 10^{49} \text{ s}^{-1}$ is placed at its center. The hydrogen has a mean density $\bar{n}_H = 10^{-3} \text{ cm}^{-3}$, and a spherical overdense region of radius $r = 8.75 \text{ pc}$ is placed on the same z -plane as the source, at a distance $d = 2.01 \text{ kpc}$ diagonally from it. Within this region the density is $n_H^\star = 6 \bar{n}_H$. The source has a black body temperature $T_{bb} = 5 \times 10^4 \text{ K}$, and t_{evo} and Δt are as in *Test 3*. The result is visualized in Figure 9, where a cut through the source plane of the final ionized hydrogen fraction x_{HII} is shown on top and the photoionization rate Γ below, for both codes along with the relative error as before. We want to point out that the fuzziness of the shadow is a feature of the short characteristic ray-tracing. The relative error is small again, and we believe it to be due to the choice of integration method used in the previous test. Interestingly, this error is larger by an order of magnitude at the edge of the overdense region. This is not so surprising, given that the overdensity is very optically thick and thus contains a large density gradient at its boundary. We noticed that the relative error is negative closer to the source, then positive, and then close to 0, reflecting the net photon flux conservation.

4.2. Performance Benchmark

We now examine the performance of the new ray-tracing library more closely. All benchmarks in this section are performed on a size $N = 250$ grid and run on one node of the Piz Daint² computer at CSCS, containing in particular a single NVIDIA® Tesla P100 GPU. First, we determine how the ray-tracing performance scales as more sources are added or the radius of ray-tracing per source increases. We expect the code to scale linearly with the number of sources N_{src} and as $\mathcal{O}(R^3)$ with the ray-tracing radius, $R = N_{\text{mesh}} \cdot R_{\text{max}} / L_B$, where R_{max} is the maximum radius for ray-tracing and L_B the box size, both in cMpc units. The benchmark is set up as follows. For $R = [10, 30, 50, 100]$, the ray-tracing routine is called (on its own, without solving the chemistry afterward) on $N_{\text{src}} = 10^a$, $a = 0, \dots, 6$ sources, and its run time is averaged over 10 executions. The left panel of Figure 10 shows the computation time per source per voxel,

$$\Delta t(N_{\text{src}}, R) = \frac{t(N_{\text{src}}, R)}{\frac{4}{3}\pi R^3 N_{\text{src}}}, \quad (19)$$

where $t(N_{\text{src}}, R)$ is the run time of the function running on N_{src} sources and computing Γ in a spherical volume of radius R (in voxel units) for each of them. With increasing N_{src} , $\Delta t(N_{\text{src}}, R)$ approaches a constant value of about 3.156 ns on our system. Furthermore, this convergence is faster when the radius R is larger. This implies that when few sources are present, overheads represent a non-negligible fraction of the execution time, even more so when the work per source (determined by R) is low. However, we can see that above ~ 1000 sources, the execution time is very close to its minimum, even for a relatively

²<https://www.cscs.ch/computers/piz-daint/>

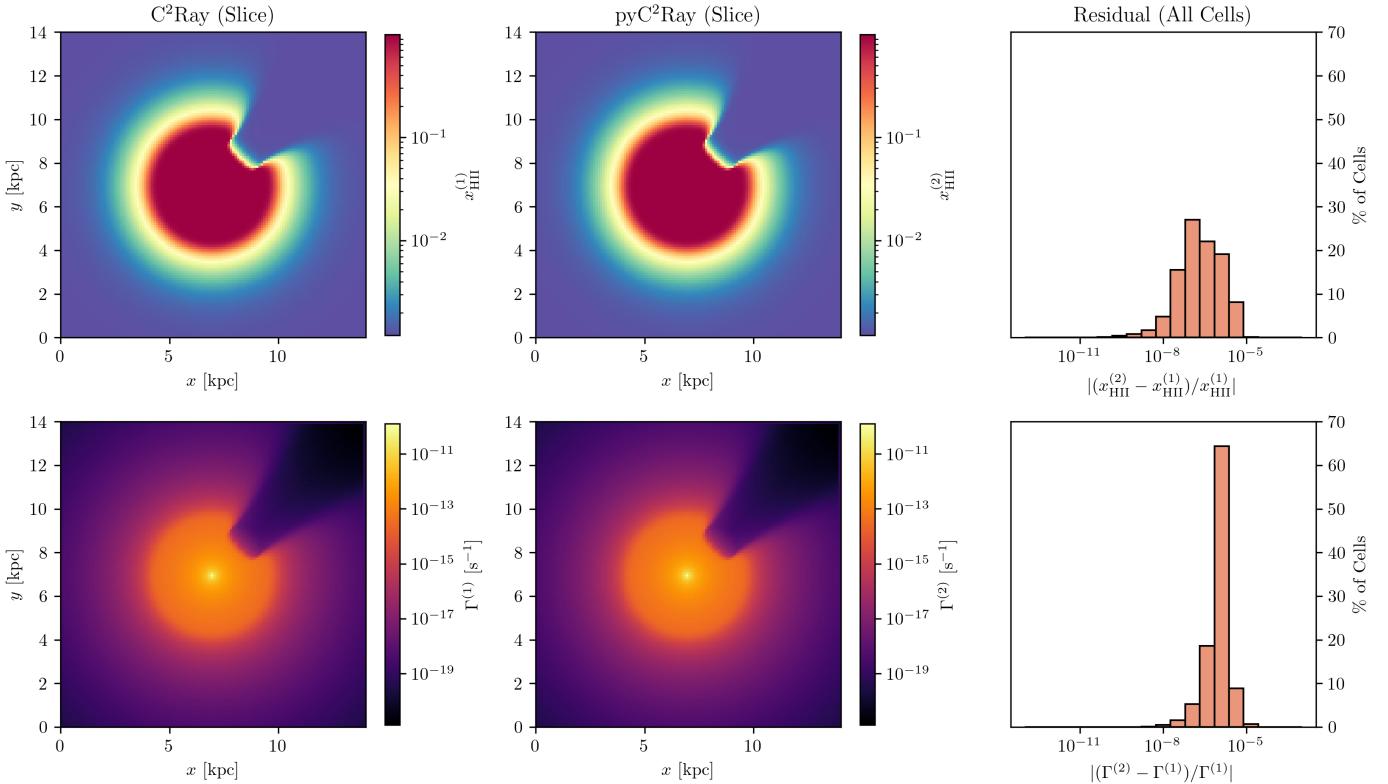


Figure 9: Result for Test 4 (I-Front Trapping in a Dense Clump and Formation of a Shadow). Shown are slices through the z -plane containing one ionizing source at the center and a dense clump of hydrogen diagonally offset from the source. The top row shows the ionized hydrogen fraction for $\text{C}^2\text{-Ray}$ (left) and pyC^2Ray (middle), as well as the relative error between the two (right). The bottom row shows the same comparison for the photoionization rate.

small RT radius. With few sources, the total amount of work is low and is not an expensive calculation. But typically, EoR simulations require $N_{\text{src}} \gg 1000$. Our code runs in a regime where the work and not overheads dominate the performance of the code.

Next, we test how the code scales as the source batch size M increases, corresponding to increasing the number of CUDA blocks dispatched to the device between global synchronizations. The right panel of Figure 10 presents the speedup t_1/t_M (where t_M is the execution time using M blocks) achieved in 3 cases; $(R = 10, N_{\text{src}} = 10^4)$, $(R = 10, N_{\text{src}} = 10^5)$ and $(R = 30, N_{\text{src}} = 10^4)$ to see the impact of both the radius and total number of sources. This test is an analog of the "strong scaling" measurement typically performed on CPU cores. We observe that on our system, in all 3 cases, the code scales well up to $M \sim 32$ and does not gain any performance above $M \sim 50$, which seems to indicate that the sequential portion of the code prevents further scaling (analogously to Amdahl's law in CPU computing). This test, however, only gives a picture of the speedup achieved *relative* to the single-block case for the whole program and hence does not indicate how good the occupancy of the GPU itself is. Detailed profiling using standard NVIDIA software has revealed that the number of registers per thread required by the ray-tracing kernel is likely a limiting factor that prevents the code from ever reaching maximum occupancy in its current state, even on GPUs with higher compute capability than the P100. Overcoming this limitation should be one of the

main targets for future performance updates.

Two conclusions arise from this section: (1) The library is most optimized for use cases where many sources are present in the simulation, as is the case in EoR modeling. However, in cases where few sources are present, it will run optimally if the number of raytraced voxels is large. This may be the case when performing high-resolution radiative transfer simulations of smaller volumes, thus expanding the possible usage scenarios for pyC^2Ray . (2) A good value for the batch size M will depend strongly on the system on which the code is run while simultaneously being limited by the available memory. This is because each block needs a cache space for the ray-tracing, the size of which scales with the grid, i.e., $\mathcal{O}(N^3)$.

5. Running a Cosmological Reionization Simulation

The ultimate test for the updated code is to see whether it can reproduce the results of a simulation performed with the original $\text{C}^2\text{-Ray}$ while at the same time achieving a gain in performance. Here, we post-process a $(349\text{Mpc})^3$ volume N -body simulation run with 4000^3 dark matter particles, which models the formation of high-redshift structures. These N -body simulations used the code CUBEP^3M (Harnois-Déraps et al., 2013)³, which has an on-the-fly halo finder, providing halo catalogs at each redshift snapshot using the spherical overdensity method

³<https://github.com/jharno/cubep3m>

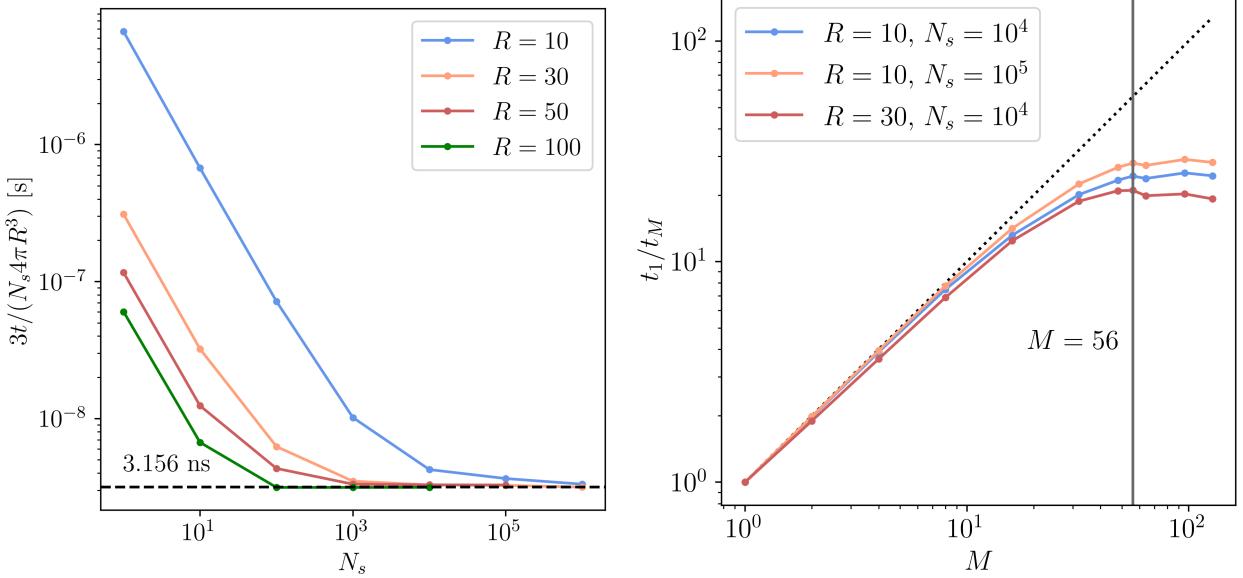


Figure 10: Scaling of the ASORA ray-tracing library. *Left*: Computation time per source per voxel for an increasing number of sources N_{src} and different ray-tracing radii R . This time approaches a constant value as more sources are added and faster for larger R . *Right*: Speedup in terms of the number of blocks M , given by t_1/t_M , where t_1 is the timing when a single block is used. The vertical black line marks $M = 56$, corresponding to the number of SMs on the NVIDIA® P100 GPU used in this benchmark.

(see Watson et al., 2013, for more detail). The N -body dark matter particles and the halo catalog are then gridded, with an SPH-like smoothing technique, onto a regular grid of size $N_{\text{mesh}} = 250^3$ that is later used as inputs for the RT simulation. This simulation resolves dark matter haloes with mass $M_{\text{halo}} \geq 10^9 M_{\odot}$. This simulation contains approximately 10^7 sources toward the end of reionization. See Dixon et al. (2016) and Giri et al. (2018) for more detailed descriptions.

We follow the same source model presented in previous work (e.g. Iliev et al., 2014; Bianco et al., 2021) that assumes a linear relation between the emissivity and the mass of the hosting dark matter halo. In this model, the grand total of ionizing photons, \dot{N}_γ , produced by a source residing in dark matter halo mass M_{halo} is

$$\dot{N}_\gamma = f_\gamma \frac{M_{\text{halo}} \Omega_b}{\Omega_M m_p t_s}, \quad (20)$$

where the efficiency factor $f_\gamma = 30$ and the source lifetime $t_s \approx 10 \text{ Myr}$ is taken to be the time difference between the simulation snapshots. Two time steps are performed for each redshift interval. Here, we choose an extreme value for the efficiency factor to speed up the reionization process so we could run \mathcal{C}^2 -Ray in a reasonable amount of time and computational resources. We should note that reionization ends quite early compared to more realistic models in Dixon et al. (2016) and Giri et al. (2019) produced using \mathcal{C}^2 -Ray—however, the outcomes of the comparison hold for any source model.

In Figure 11, we show slices of the simulated ionized fraction, x_{HII} , comparing \mathcal{C}^2 -Ray (left column) and py \mathcal{C}^2 Ray (middle column) at redshift $z = 11.090, 10.110, 9.457$ and 8.636 , corresponding to a volume-averaged ionized fraction $\langle x_{\text{HII}} \rangle = 0.045, 0.180, 0.420$ and 0.837 . We show the relative error in the right column of the same figure for each redshift. At high redshift, the error distribution is mostly centered at 10^{-6} , simi-

lar to what we show in § 4.1.3 and 4.1.4. While from $z \sim 10$, it shows two peaks with the distribution transitioning from $10^{-4.5}$ to 10^{-10} . The double-peaked feature of the error distribution is visible from the moment the source contribution becomes substantial. This indicates that the error distribution is initially associated with the precision error in the vast neutral field while later with the growing ionized regions. In the left panels of Figure 12, we calculate the volume- and mass-averaged ionized fraction, $\langle x_{\text{HII}} \rangle_v$ and $\langle x_{\text{HII}} \rangle_m$, against redshift. With solid lines, we indicate the results obtained with \mathcal{C}^2 -Ray, while in dashed lines, the one with py \mathcal{C}^2 Ray. Similar to what we show in the previous paragraph, on average, the relative error is at least five orders of magnitude smaller, $\sim 10^{-5}$, compared to the dynamic range of the ionized field, making the difference indiscernible. Notice that we show the result to $z = 8.575$ when the IGM is about 86% ionized. However, at this reionization epoch, the simulation has approximately $\sim 1.5 \times 10^6$ sources, and \mathcal{C}^2 -Ray starts to become computationally demanding.

Radio experiments, such as HERA, LOFAR, and MWA, aim to observe the spatial distribution \mathbf{r} of the differential brightness temperature $\delta T_b(\mathbf{r}, z)$ corresponding to the 21-cm signal. This quantity can be given as (e.g. Pritchard and Loeb, 2012),

$$\delta T_b(\mathbf{r}, z) \approx 27 \text{ mK} \left(\frac{0.15}{\Omega_M h^2} \frac{1+z}{10} \right)^{\frac{1}{2}} \left(\frac{\Omega_b h^2}{0.023} \right) \times [1 - x_{\text{HII}}(\mathbf{r}, z)][1 + \delta_b(\mathbf{r}, z)], \quad (21)$$

where x_{HII} and δ_b are ionization hydrogen fraction and baryon overdensity, respectively. We should note that we have assumed a spin temperature to be saturated and ignore the impact of redshift-space distortion. We refer the interested readers to Ross et al. (2021) for exploration of both these aspects in simulations with \mathcal{C}^2 -Ray. We compute $\delta T_b(\mathbf{r}, z)$ and subse-

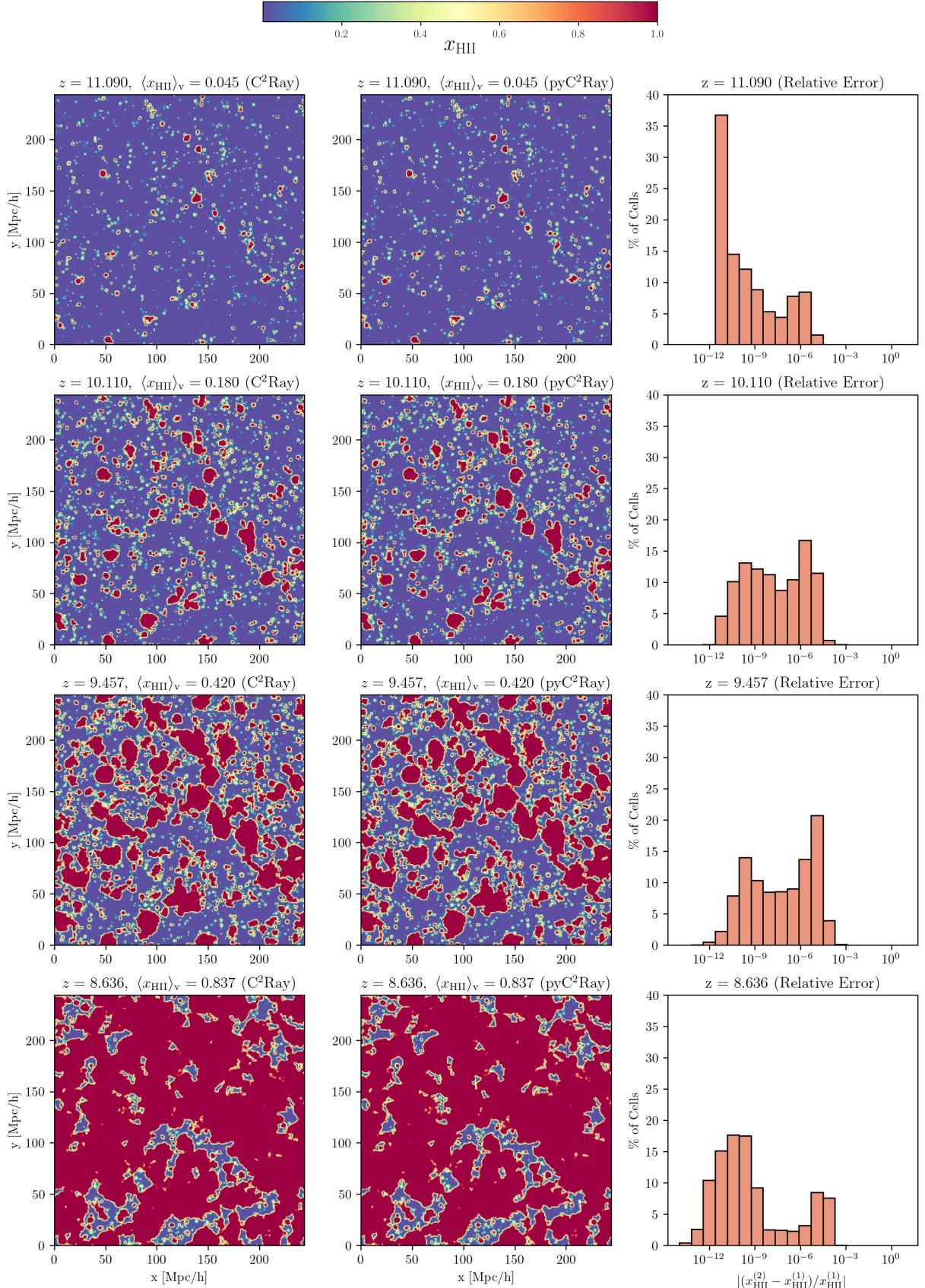


Figure 11: Results from the 349 cMpc EoR test simulation. The left and middle columns show slices through the simulation domain for C^2 -Ray and pyC^2 Ray, respectively. The right column shows the distribution relative per-voxel error for the 250^3 grid. The simulation includes only dark matter halos masses with an efficiency factor $f_\gamma = 30$ and a maximal comoving photon radius $R = 15$ cMpc.

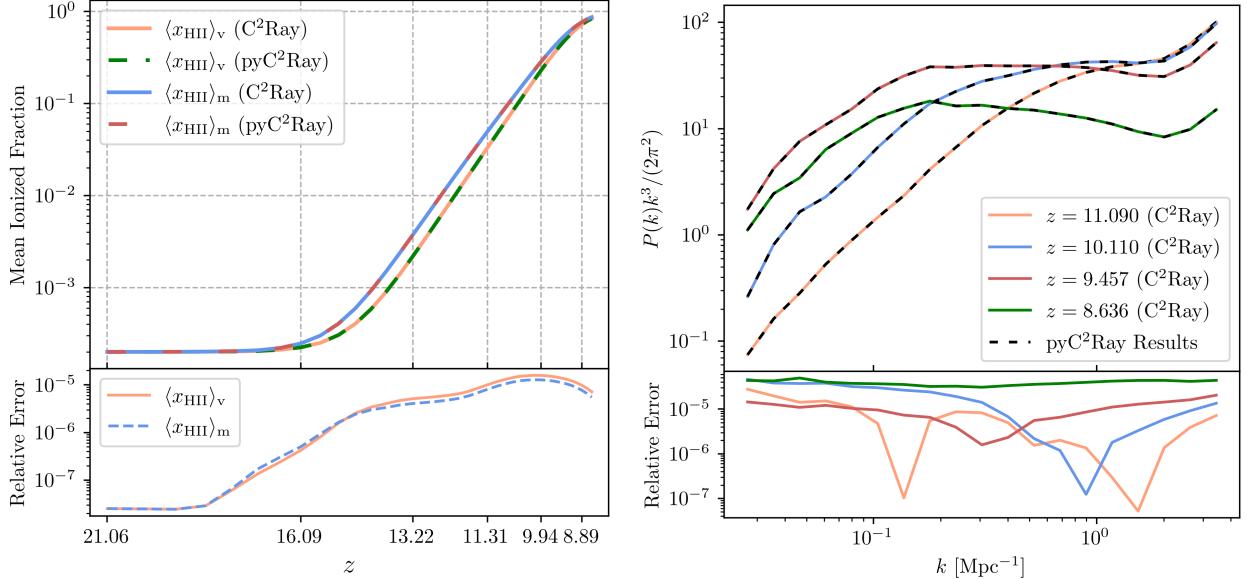


Figure 12: *Left:* Comparison of reionization history from the 349 Mpc EoR test simulation, performed with C^2 -Ray and pyC^2 Ray. The top panel shows the evolution of the volume $\langle x_{HII}^v \rangle$ and mass-averaged $\langle x_{HII}^m \rangle$ fraction of ionized hydrogen over the redshift range $z \in [21.06, 8.636]$ and the bottom panel the relative error between the two codes. *Right:* Comparison of the 21-cm power spectra from the same simulation at different redshifts (indicated in the legend) reveals a consistent match between C^2 -Ray and pyC^2 Ray.

quently the power spectrum using reionization simulation snapshots with our data analysis software, `Tools21cm`⁴ (Giri et al., 2020). In the top-right panel of Figure 12, we present the 21-cm power spectrum at various redshifts. We observe a precise agreement between the results obtained from pyC^2 Ray and C^2 -Ray, also evident from the relative error in the bottom-right panel, demonstrating that these upgrades can accurately replicate the spatial distribution of the 21-cm signal.

The simulation with pyC^2 Ray cost 2.5 GPU-hours on our single-GPU system, while the comparison run, with the Fortran90 CPU version of C^2 -Ray, computed on 128 cores for a total of 13,824 core-hours. While GPU hours are, in general, more expensive than core hours, the observed speedup is so large that pyC^2 Ray is significantly cheaper to run than the original code by a factor of ~ 100 , depending on the computing center, which was part of the motivation behind this update. In Appendix B, we illustrate further the computational advantage of porting algorithms to GPU.

6. Summary and Conclusions

The main challenge in simulating the cosmic Epoch of Reionization is that we must concurrently simulate a large volume of the order of the Gpc scale while resolving compact and dense cosmic structures. These requirements make Radiative Transfer (RT) simulations extremely computationally expensive and demanding. For this reason, most RT codes are implemented with programming languages suited for scientific computing, such as Fortran90 or C/C++. However, this makes any changes or regular updates to the code cumbersome for

new users, as any slight modification requires frequent recompilation and debugging. Moreover, relatively little effort has been made to make ray-tracing algorithms for reionization simulations computationally efficient and functional on general-purpose graphic process units (GPU).

Therefore, this paper introduces pyC^2 Ray, a Python wrapped updated version of the extensively used C^2 -Ray RT code for cosmic reionization simulations. In particular, we present the newly developed Accelerated Short-characteristics Octahedral Ray-tracing algorithm, ASORA, that utilizes GPU architectures to achieve drastic speedup in fully numerical RT simulations.

In § 2, we recap the differential equation solved during a cosmological reionization simulation. In § 2.1, we summarize the well-established time-averaged method that solves the chemistry equation in C^2 -Ray, Equation 1, allowing the solution to be integrated on a larger time-step compared to the reionization time scales, otherwise required by a more direct approach. In § 2.2, we explain in detail the necessity for an efficient ray-tracing method for our code. With Equation 6 and 8, we highlight the core and most computationally expensive operation in RT algorithms, which consists of computing the column density and, thus, the optical depth for each voxel, that ultimately quantifies the number of ionizing photons that are absorbed by a cell along the ray. The combination of the time-averaged and short-characteristics methods are the distinguishing features of the C^2 -Ray code. In Figure 1, we summarize the algorithm for both the C^2 -Ray and pyC^2 Ray methods presented here.

In § 3.1, we remind the reader of the short-characteristic approach of C^2 -Ray inherited by pyC^2 Ray. In § 3.2, we describe the existing CPU parallelization of the current version of C^2 -Ray, which consists of splitting the source input list into equal parts for each MPI processor. For each rank, 8 OpenMP

⁴<https://github.com/sambit-giri/tools21cm>

threads, corresponding to the number of independent domains around each source, compute the HII column density. This parallelization strategy is not optimal for GPU architectures. Therefore, in § 3.3 we propose a new interpolation approach for the C^2 -Ray RT algorithm specifically designed for GPUs. The ASORA interpolation scheme comes from the physical intuition that the radiation propagates as an outward wavefront around a source. This new approach changes the domain decomposition to an interpolation between concentric surfaces of an octahedron centered around the source as illustrated by Figure 3. From a technical perspective, in `pyC2Ray`, we keep the same MPI source distribution, as presented in § 3.2, and instead replace the OpenMP domain decomposition with the ASORA method.

The update also includes the conversion to Python of the non-time-consuming subroutines of C^2 -Ray. In § 3.4, we mention how the use of commonly used libraries, such as `Numpy`, `Scipy` and `Astropy` can be easily included according to the user's need. Moreover, the `pyC2Ray` user interface makes it easier to employ other codes that have also been Python-wrapped. For instance, we can easily incorporate in `pyC2Ray` photo-ionization rates from other spectral energy distributions calculated with a population synthesis code such as `PEGASE-2` (Fioc et al., 2011) or a different chemistry solver such `GRACKLE` (Smith et al., 2017).

In § 4, we show `pyC2Ray` results on a series of standard RT tests. In § 4.1.1 and 4.1.2, we demonstrate that `pyC2Ray` agrees with the analytical solutions of the ionization front size, r_1 , for the single sources in a static and expanding lattice. To test that the conversion to Python of the non-time-critical subroutines was successful and does not introduce substantial differences, in § 4.1.3, we test the results on overlapping HII regions for sources with different black body spectra. In § 4.1.4, we probe the formation of a shadow behind an overdense region, a standard test for ray tracing methods.

In § 4.2, we examine the performance of the new ray-tracing methods accomplished on the Piz Daint cluster at the Swiss National Supercomputing Centre (CSCS) equipped with an NVIDIA® Tesla P100 GPU. Our main finding is that the ASORA RT computing time grows linearly with the increasing number of sources, N_{src} , and in cubic fashion with respect to the maximum radius for ray tracing, so R^3 , i.e., distance is given in a number of voxels. In the case of the Tesla P100 GPU, the computing time per source per voxel within the ray-tracing distance saturates with value 3.156 ns when $N_{\text{src}} > 10^5$. This study allows the user to quantify the computing time and cost of a future simulation run with `pyC2Ray`. If we consider a cosmological simulation with 68 redshift steps, each with 2-time steps, ray-tracing radius $R = 11$ (grid units) and approximately $N_{\text{src}} \approx 4 \times 10^6$ sources. We can run the entire simulation from $z = 21$ to 8.5 with a total of ~ 2.75 GPU-h, corresponding to the cost obtained in the cosmological example presented in §5. Secondly, the method scales strongly with the batch size up to ~ 32 on our system, suggesting that the GPU occupancy is not yet optimal, an issue that may be addressed in future updates. We estimate that running a reionization simulation on the same volume down to $z \sim 6$, where $N_{\text{src}} = 1.5 \times 10^7$, would cost approximately 10.3 GPU-h.

Finally, in § 5, we compare `pyC2Ray` and C^2 -Ray on an actual cosmological simulation. We demonstrate that the differences within the same simulation are negligible with an absolute-relative error between 10^{-4} and 10^{-12} on the HII field, while both mass- and volume-averaged ionized fractions and the power spectra accumulate an error that stays below the order of $< 10^{-5}$. As mentioned in the previous paragraph, the computational cost for this simulation was 2.5 GPU-hours, while the same simulation run on 128 cores with C^2 -Ray took ~ 14 k core-hours. Another way to describe the gain in performance is to consider the monetary cost of running these simulations. The cost of running a code on a GPU or CPU cluster varies based on the electricity consumption and other indirect expenses assessed by the high-performance computer facility. Nowadays, one GPU-hour can cost on average 0.8 Euros, while one core-hour can be 0.01 Euros. Therefore, with these reference fees the simulation presented § 5 would have cost 2 Euros if run with `pyC2Ray` instead of 138.25 Euros with C^2 -Ray.

With this work, we demonstrate that `pyC2Ray` achieves the same result as C^2 -Ray for a cosmological EoR simulation, but with a computing cost and time two orders of magnitude lower than the original code, confirming the motivation behind this modernization of C^2 -Ray. In principle, `pyC2Ray` is not limited by the volume size or the mass resolution but rather by the spatial resolution, N , and the number of sources, N_{src} . The ASORA raytracing algorithm needs to store M copies of the entire double precision grid data directly on the GPU, where M is the source batch size. Therefore, the current limiting factor is the available memory on the GPU, as it is generally desirable to have $M \gtrsim 20$ to achieve optimal GPU occupancy. For instance, the NVIDIA® P100 has 64 GB of memory; we can, in principle, simulate a 1024^3 mesh grid but are then limited to $M < 8$, which is below the optimal regime. We plan to address this issue by reducing the per-source memory requirement in those cases where the ray-tracing radius is significantly smaller than the whole box and the current implementation is needlessly memory-hungry. In this update, we focused on the simplest simulation setup, namely, no photo-heating and only photo-ionization for hydrogen chemistry. As mentioned, C^2 -Ray has been extended to also include helium (Friedrich et al., 2012) and X-ray heating (Ross et al., 2017), and has also been used as a module in a hydrodynamic simulation to follow the evolution of an HII region in the interstellar medium (ISM), see Arthur et al. (2011) and Medina et al. (2014). We aim to gradually include these features and extensions in `pyC2Ray` now that the groundwork has been laid.

Acknowledgements

The authors would like to thank Emma Tolley, Shreyam Krishna and Chris Finlay for their feedback and useful discussions, as well as Hannah Ross, Jean-Guillaume Piccinali, Andreas Fink and Dmitry Alexeev for their help on the technical aspects of the GPU implementation. MB acknowledges the financial support from the Swiss National Science Foundation (SNSF) under the Sinergia Astrosignals grant (CR-SII5_193826). PH acknowledges access to Piz Daint at the

Swiss National Supercomputing Centre, Switzerland, under the SKA’s share with the project ID sk015. This work has been done as part of the SKACH consortium through funding from SERI. GM’s research is supported by the Swedish Research Council project grant 2020-04691_VR. We also acknowledge the allocation of computing resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS) at the PDC Center for High-Performance Computing, KTH Royal Institute of Technology, partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

The image processing tools operated on our data were performed with the help of NumPy and SciPy packages. All plots were created with matplotlib (Hunter, 2007), and the illustration in Figure 3 was made using Blender.

Appendix A. ASORA Implementation Details

Here, we briefly discuss how the ASORA method is implemented in C++/CUDA. As detailed in the paper, each block is assigned to a single source and owns a dedicated memory space to store the values of the column densities of voxels to be used as interpolants in upcoming tasks. Each task S_q comprises the $|S_q| = 4q^2 + 2$ grid voxels belonging to an octahedral shell as illustrated in Figure 3. Threads within a block are labeled by 1D indices $x = 0, \dots, N$, where N is the block size. Labeling the voxels in the shell by $s = 0, \dots, |S_q|$, all voxels can be treated if the threads iterate $\sim |S_q|/N$ times. It then remains to map the 1D indices s to the actual 3D grid positions (i, j, k) of the voxels within the shell. We use the following mapping: separate the octahedron into a "top" part containing all $k \geq k_s$ planes, where k_s is the source plane, and a "bottom" part containing the rest. For the top part, which contains $2q(q + 1) + 1$ voxels in total, the k index of any voxel can be found from its i, j indices through $k = k_s + q - (|i - i_s| + |j - j_s|)$. To find i, j , we follow the procedure illustrated in Figure A.13: map $s = 1, \dots, 2q(q + 1)$ to Cartesian 2D coordinates (a, b) as in (A) and apply a shear matrix $(a, b) \rightarrow (a', b')$ to obtain (B). Apply a translation on the subset of those with $a + 2b > 2q$ (C) and finally map the remaining voxel $s = 0$ to $(i, j) = (i_s + q, j_s)$ to obtain the full squashed top part of the octahedron (D). The same procedure is applied to the lower part, with some slight modifications, as this does not include the source plane and so contains fewer voxels in total ($2q^2 - 1$). For further details, we refer the reader to the source code.

A last key point to address is that since C^2 -Ray uses periodic boundary conditions, it is important to impose a further constraint on the indices (i, j, k) of the voxels that are allowed to avoid race conditions on coordinates that map to the same voxel under periodicity. The simulation domain is cubic, so this constraint is satisfied if we impose that no voxel can be farther away from the source than the edges of the grid, translated under periodicity. On an odd mesh (N odd), this means only considering voxels at most a grid distance $N/2$ away from the source on either side. On an even mesh, a convention must be chosen, and in line with the original C^2 -Ray code, we impose

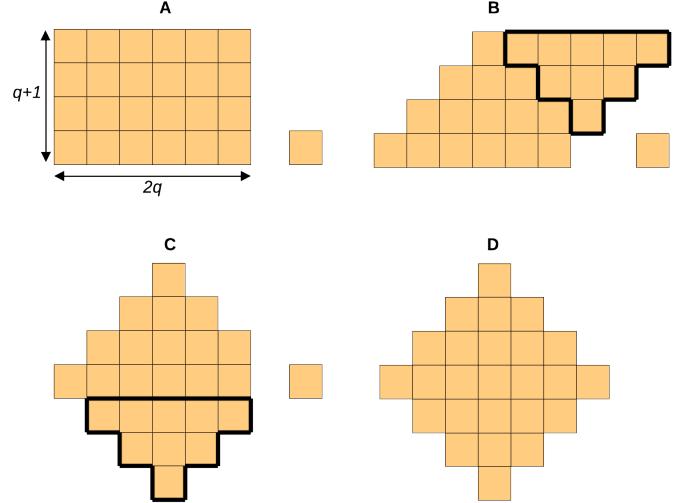


Figure A.13: Schematic representation of the mapping of 1D indices $0, \dots, 2q(q + 1)$ to the 3D grid positions (i, j, k) of voxels in the top part of the $q = 3$ shell. The (i, j) mapping is a combination of a shear (B) and a translation (C), and the k coordinate is determined directly from (i, j) as described in the text.

that the maximum distance in each dimension is $N/2$ on the negative and $N/2 - 1$ on the positive side of the source.

Appendix B. Carbon footprint of cosmological simulations

Numerical simulations for cosmological and astrophysics applications often require immense computational power and extensive data processing, and therefore, their energy demands can be substantial. The environmental impact is often underappreciated and sometimes disregarded. Given the escalating concern over climate change, we want to present the ecological advantage of moving to GPU-based algorithms.

We employed Green Algorithm⁵ to estimate the carbon consumption of the cosmological simulations presented in §5 and compare the run with pyC²Ray and C²-Ray. As we mentioned in §4.2, the cosmological EoR simulation presented in this paper run with pyC²Ray was performed in 2 hours and 30min on 1 GPU NVIDIA® Tesla P100, drawing 1.07 kWh. Based in Switzerland, this has a carbon emission (CO₂e) of 12.31 g. This corresponds to the CO₂ consumption of driving a car for 70 meters or 0.02% of the consumption of the Paris-London flight. Based in Sweden, the same simulation runs with C²-Ray on 128 AMD EPYC Zen 2 CPUs. The cluster draws 105.88 kWh and has a 600.33 g CO₂e, corresponding to the consumption of a car drive for 3.43 Km or the 1% consumption of the Paris-London travel by plane. A mature tree sequesters on average 0.92 g of CO₂ per month (Lannelongue et al., 2021). Based on this estimation, the cosmological run performed, with C²-Ray, would have consumed what one single tree sequesters from the atmosphere in approximately 54 years. Meanwhile, the same simulation run with pyC²Ray would take about one

⁵www.green-algorithms.org

Table A.1: Summary of the carbon footprint consumption of the cosmological simulation presented in this paper if both runs were performed in Switzerland.

Model	CO ₂ emission [kg]	Energy consumption [kWh]	Car drive [km]	CO ₂ absorption [yr]
NVIDIA Tesla P100	0.02	1.07	0.07	1.12
AMD EPYC Zen 2	1.22	105.88	6.97	110.5

year. In Table A.1, we compare the simulations CO₂ consumption if both runs were performed in Switzerland.

While this analysis highlights the environmental footprint of cosmological simulations, its purpose is not to evoke shame or guilt. Rather, it serves as a reminder of the tangible costs of these essential scientific endeavors. Moreover, we did not consider using renewable energy sources and the potential impact reduction of HPC clusters using renewable energy. We aim to highlight the differences in energy consumption between simulation approaches.

References

Altay, G., Croft, R.A.C., Pelupessy, I., 2008. sphray: a smoothed particle hydrodynamics ray tracer for radiative transfer. *MNRAS* 386, 1931–1946. URL: <http://dx.doi.org/10.1111/j.1365-2966.2008.13212.x>. doi:10.1111/j.1365-2966.2008.13212.x.

Arthur, S.J., Henney, W.J., Mellema, G., de Colle, F., Vázquez-Semadeni, E., 2011. Radiation-magnetohydrodynamic simulations of H II regions and their associated PDRs in turbulent molecular clouds. *MNRAS* 414, 1747–1768. doi:10.1111/j.1365-2966.2011.18507.x, arXiv:1101.5510.

Astropy Collaboration, 2022. The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package. *ApJ* 935, 167. doi:10.3847/1538-4357/ac7c74, arXiv:2206.14220.

Atek, H., Labb  , I., Furtak, L.J., Chemerynska, I., Fujimoto, S., Setton, D.J., Miller, T.B., Oesch, P., Bezanson, R., Price, S.H., et al., 2024. Most of the photons that reionized the universe came from dwarf galaxies. *Nature* 626, 975–978.

Aubert, D., Deparis, N., Ocvirk, P., 2015. Emma: an adaptive mesh refinement cosmological simulation code with radiative transfer. *MNRAS* 454, 1012–1037. URL: <http://dx.doi.org/10.1093/mnras/stv1896>. doi:10.1093/mnras/stv1896.

Aubert, D., Teyssier, R., 2008. A radiative transfer scheme for cosmological reionization based on a local eddington tensor. *MNRAS* 387, 295–307.

Aubert, D., Teyssier, R., 2010. Reionization simulations powered by graphics processing units. i. on the structure of the ultraviolet radiation field. *The Astrophysical Journal* 724, 244–266. URL: <http://dx.doi.org/10.1088/0004-637X/724/1/244>. doi:10.1088/0004-637X/724/1/244.

Barkana, R., 2016. The rise of the first stars: Supersonic streaming, radiative feedback, and 21-cm cosmology. *Physics Reports* 645, 1–59.

Bianco, M., Iliev, I.T., Ahn, K., Giri, S.K., Mao, Y., Park, H., Shapiro, P.R., 2021. The impact of inhomogeneous subgrid clumping on cosmic reionization – II. Modelling stochasticity. *MNRAS* 504, 2443–2460. URL: <https://doi.org/10.1093/mnras/stab787>. doi:10.1093/mnras/stab787.

Bosman, S.E., Davies, F.B., Becker, G.D., Keating, L.C., Davies, R.L., Zhu, Y., Eilers, A.C., D’Odorico, V., Bian, F., Bischetti, M., et al., 2022. Hydrogen reionization ends by $z=5.3$: Lyman- α optical depth measured by the xqr-30 sample. *MNRAS* 514, 55–76.

Cavelan, A., Cabe  n, R.M., Grabarczyk, M., Ciorba, F.M., 2020. A Smoothed Particle Hydrodynamics Mini-App for Exascale, in: *PASC ’20: Proceedings of the Platform for Advanced Scientific Computing Conference June 2020*, p. 11. doi:10.1145/3394277.3401855, arXiv:2005.02656.

Choudhury, T.R., 2009. Analytical models of the intergalactic medium and reionization. arXiv:0904.4596.

Choudhury, T.R., Ferrara, A., 2006. Physics of cosmic reionization. arXiv:astro-ph/0603149.

Ciardi, B., Ferrara, A., Marri, S., Raimondo, G., 2001. Cosmological reionization around the first stars: Monte Carlo radiative transfer. *MNRAS* 324, 381–388. URL: <https://doi.org/10.1046/j.1365-8711.2001.04316.x>, doi:10.1046/j.1365-8711.2001.04316.x, arXiv:<https://academic.oup.com/mnras/article-pdf/324/2/381/3361122>.

Dalcin, L., Fang, Y.L., 2021. mpi4py: Status Update After 12 Years of Development. *Computing in Science and Engineering* 23, 47–54. doi:10.1109/ICSE.2021.3083216.

Dayal, P., Ferrara, A., 2018. Early galaxy formation and its large-scale effects. *Physics Reports* 780–782, 1–64. URL: <https://www.sciencedirect.com/science/article/pii/S0370157318302266>, doi:<https://doi.org/10.1016/j.physrep.2018.10.002>. early galaxy formation and its large-scale effects.

DeBoer, D.R., Parsons, A.R., Aguirre, J.E., Alexander, P., Ali, Z.S., Beardsley, A.P., Bernardi, G., Bowman, J.D., Bradley, R.F., Carilli, C.L., et al., 2017. Hydrogen epoch of reionization array (hera). *Publications of the Astronomical Society of the Pacific* 129, 045001.

Dixon, K.L., Iliev, I.T., Mellema, G., Ahn, K., Shapiro, P.R., 2016. The large-scale observational signatures of low-mass galaxies during reionization. *MNRAS* 456, 3011–3029. doi:10.1093/mnras/stv2887, arXiv:1512.03836.

Fioc, M., Le Borgne, D., Rocca-Volmerange, B., 2011. P  GASE: Metallicity-consistent Spectral Evolution Model of Galaxies. *Astrophysics Source Code Library*, record ascl:1108.007. arXiv:1108.007.

Friedrich, M.M., Mellema, G., Iliev, I.T., Shapiro, P.R., 2012. Radiative transfer of energetic photons: X-rays and helium ionization in C²-RAY. *MNRAS* 421, 2232–2250. doi:10.1111/j.1365-2966.2012.20449.x, arXiv:1201.0602.

Friedrich, M.M., Mellema, G., Iliev, I.T., Shapiro, P.R., 2012. Radiative transfer of energetic photons: X-rays and helium ionization in c2-ray. *MNRAS* 421, 2232–2250.

Furlanetto, S.R., Oh, S.P., Briggs, F.H., 2006. Cosmology at low frequencies: The 21cm transition and the high-redshift universe. *Physics Reports* 433, 181–301. URL: <https://doi.org/10.1016/j.physrep.2006.08.002>. doi:10.1016/j.physrep.2006.08.002.

Garland, M., Le Grand, S., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., Volkov, V., 2008. Parallel computing experiences with cuda. *IEEE Micro* 28, 13–27.

Gelli, V., Salvadori, S., Ferrara, A., Pallottini, A., Carniani, S., 2023. Quiescent low-mass galaxies observed by jwst in the epoch of reionization. *The Astrophysical Journal Letters* 954, L11.

Giri, S.K., Mellema, G., Aldheimer, T., Dixon, K.L., Iliev, I.T., 2019. Neutral island statistics during reionization from 21-cm tomography. *MNRAS* 489, 1590–1605.

Giri, S.K., Mellema, G., Dixon, K.L., Iliev, I.T., 2018. Bubble size statistics during reionization from 21-cm tomography. *MNRAS* 473, 2949–2964.

Giri, S.K., Mellema, G., Jensen, H., 2020. Tools21cm: A python package to analyse the large-scale 21-cm signal from the epoch of reionization and cosmic dawn. *Journal of Open Source Software* 5, 2363.

Giri, S.K., Schneider, A., Maion, F., Angulo, R.E., 2023. Suppressing variance in 21 cm signal simulations during reionization. *A&A* 669, A6.

Gnedin, N.Y., Abel, T., 2001. Multi-dimensional cosmological radiative transfer with a Variable Eddington Tensor formalism. *New A* 6, 437–455. doi:10.1016/S1384-1076(01)00068-9, arXiv:astro-ph/0106278.

Gnedin, N.Y., Madau, P., 2022. Modeling cosmic reionization. *Living Reviews in Computational Astrophysics* 8, 3.

Gorbunov, D.S., Rubakov, V.A., 2011. *Introduction to the Theory of the Early Universe: Hot Big Bang Theory*. 2 ed., World Scientific Publishing Company. doi:10.1142/7874.

van Haarlem, M.P., Wise, M.W., Gunst, A., Heald, G., McKean, J.P., Hessels, J.W., de Bruyn, A.G., Nijboer, R., Swinbank, J., Fallows, R., et al., 2013. Lofar: The low-frequency array. *A&A* 556, A2.

Harnois-D  raps, J., Pen, U.L., Iliev, I.T., Merz, H., Emberson, J.D., Desjacques, V., 2013. High-performance P3M N-body code: CUBEP3M. *MNRAS* 436, 540–559. doi:10.1093/mnras/stt1591, arXiv:1208.5098.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P.,

Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerwijk, M.H., Brett, M., Haldane, A., Fernández del Río, J., Wiebe, M., Peterson, P., Gérard-Marchant, P., Shepard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E., 2020. Array programming with NumPy. *Nature* 585, 357–362. doi:10.1038/s41586-020-2649-2.

HERA Collaboration, 2023. Improved Constraints on the 21 cm EoR Power Spectrum and the X-Ray Heating of the IGM with HERA Phase I Observations. *ApJ* 945, 124. doi:10.3847/1538-4357/acaf50.

Hunter, J.D., 2007. Matplotlib: A 2d graphics environment. *CiSE* 9, 90–95. doi:10.1109/MCSE.2007.55.

Iliev, I.T., Ciardi, B., Alvarez, M.A., Maselli, A., Ferrara, A., Gnedin, N.Y., Mellema, G., Nakamoto, T., Norman, M.L., Razoumov, A.O., Rijkhorst, E.J., Ritzerveld, J., Shapiro, P.R., Susa, H., Umemura, M., Whalen, D.J., 2006. Cosmological radiative transfer codes comparison project - I. The static density field tests. *MNRAS* 371, 1057–1086. doi:10.1111/j.1365-2966.2006.10775.x, arXiv:astro-ph/0603199.

Iliev, I.T., Mellema, G., Ahn, K., Shapiro, P.R., Mao, Y., Pen, U.L., 2014. Simulating cosmic reionization: how large a volume is large enough? *MNRAS* 439, 725–743. doi:10.1093/mnras/stt2497, arXiv:1310.7463.

Iliev, I.T., Whalen, D., Mellema, G., Ahn, K., Baek, S., Gnedin, N.Y., Kravtsov, A.V., Norman, M., Raicevic, M., Reynolds, D.R., Sato, D., Shapiro, P.R., Semelin, B., Smidt, J., Susa, H., Theuns, T., Umemura, M., 2009. Cosmological radiative transfer comparison project - II. The radiation-hydrodynamic tests. *MNRAS* 400, 1283–1316. doi:10.1111/j.1365-2966.2009.15558.x, arXiv:0905.2920.

Kannan, R., Vogelsberger, M., Marinacci, F., McKinnon, R., Pakmor, R., Springel, V., 2019. Arepo-rt: radiation hydrodynamics on a moving mesh. *MNRAS* 485, 117–149.

Kaur, H.D., Gillet, N., Mesinger, A., 2020. Minimum size of 21-cm simulations. *MNRAS* 495, 2354–2362.

Lannelongue, L., Grealey, J., Inouye, M., 2021. Green algorithms: Quantifying the carbon footprint of computation. *Advanced Science* 8, 2100707. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/advs.202100707>, doi:<https://doi.org/10.1002/advs.202100707>, arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1002/advs.202100707>.

Medina, S.N.X., Arthur, S.J., Henney, W.J., Mellema, G., Gazol, A., 2014. Turbulence in simulated H II regions. *MNRAS* 445, 1797–1819. doi:10.1093/mnras/stu1862, arXiv:1409.5838.

Mellema, G., Iliev, I.T., Alvarez, M.A., Shapiro, P.R., 2006. C²-ray: A new method for photon-conserving transport of ionizing radiation. *New A* 11, 374–395. doi:10.1016/j.newast.2005.09.004, arXiv:astro-ph/0508416.

Mellema, G., Koopmans, L.V., Abdalla, F.A., Bernardi, G., Ciardi, B., Daiboo, S., de Bruyn, A., Datta, K.K., Falcke, H., Ferrara, A., et al., 2013. Reionization and the cosmic dawn with the square kilometre array. *Experimental Astronomy* 36, 235–318.

Mertens, F.G., Mevius, M., Koopmans, L.V., Offringa, A., Mellema, G., Zaroubi, S., Brentjens, M., Gan, H., Gehlot, B.K., Pandey, V., et al., 2020. Improved upper limits on the 21 cm signal power spectrum of neutral hydrogen at z= 9.1 from lofar. *MNRAS* 493, 1662–1685.

Nakamoto, T., Umemura, M., Susa, H., 2001. 3D Radiative Transfer Effects on the Cosmic Reionization, in: Umemura, M., Susa, H. (Eds.), *The Physics of Galaxy Formation*, p. 143.

Navarro, C.A., Hitschfeld-Kahler, N., Mateu, L., 2014. A survey on parallel computing and its applications in data-parallel problems using gpu architectures. *CiCP* 15, 285–329. doi:10.4208/cicp.110113.010813a.

Nebrin, O., Giri, S.K., Mellema, G., 2023. Starbursts in low-mass haloes at cosmic dawn. i. the critical halo mass for star formation. *MNRAS*, stat1852.

Nickolls, J., Buck, I., Garland, M., Skadron, K., 2008. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue* 6, 40–53. URL: <https://doi.org/10.1145/1365490.1365500>, doi:10.1145/1365490.1365500.

Nickolls, J., Dally, W.J., 2010. The gpu computing era. *IEEE Micro* 30, 56–69. doi:10.1109/MM.2010.41.

Ocvirk, P., Gillet, N., Shapiro, P.R., Aubert, D., Iliev, I.T., Teyssier, R., Yepes, G., Choi, J.H., Sullivan, D., Knebe, A., Gottlöber, S., D’Aloisio, A., Park, H., Hoffman, Y., Stranex, T., 2016. Cosmic Dawn (CoDa): the first radiation-hydrodynamics simulation of reionization and galaxy formation in the Local Universe. *MNRAS* 463, 1462–1485. URL: <https://doi.org/10.1093/mnras/stw2036>, doi:10.1093/mnras/stw2036.

Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., Phillips, J.C., 2008. Gpu computing. *Proceedings of the IEEE* 96, 879–899. doi:10.1109/JPROC.2008.917757.

Pawlak, A.H., Schaye, J., 2008. TRAPHIC - radiative transfer for smoothed particle hydrodynamics simulations. *MNRAS* 389, 651–677. doi:10.1111/j.1365-2966.2008.13601.x, arXiv:0802.1715.

Planck Collaboration, Aghanim, N., Akrami, Y., Ashdown, M., Aumont, J., Baccigalupi, C., Ballardini, M., Banday, A., Barreiro, R., Bartolo, N., Basak, S., et al., 2020. Planck 2018 results-vi. cosmological parameters. *å* 641, A6.

Potter, D., Stadel, J., Teyssier, R., 2016. Pkdgrav3: Beyond trillion particle cosmological simulations for the next era of galaxy surveys. arXiv:1609.08621.

Pritchard, J.R., Loeb, A., 2012. 21 cm cosmology in the 21st century. *Reports on Progress in Physics* 75, 086901.

Raga, A.C., Mellema, G., Arthur, S.J., Binette, L., Ferruit, P., Steffen, W., 1999. 3D Transfer of the Diffuse Ionizing Radiation in ISM Flows and the Preionization of a Herbig-Haro Working Surface. *Rev. Mexicana Astron. Astrofis.* 35, 123.

Rijkhorst, E.J., Plewa, T., Dubey, A., Mellema, G., 2006. Hybrid characteristics: 3D radiative transfer for parallel adaptive mesh refinement hydrodynamics. *A&A* 452, 907–920. doi:10.1051/0004-6361:20053401, arXiv:astro-ph/0505213.

Ritzerveld, J., 2005. The diffuse nature of Strömgren spheres. *A&A* 439, L23–L26. doi:10.1051/0004-6361:200500150, arXiv:astro-ph/0506637.

Rosdahl, J., Blaizot, J., Aubert, D., Stranex, T., Teyssier, R., 2013. RAMSES-RT: radiation hydrodynamics in the cosmological context. *MNRAS* 436, 2188–2231. doi:10.1093/mnras/stt1722, arXiv:1304.7126.

Ross, H.E., Dixon, K.L., Ghara, R., Iliev, I.T., Mellema, G., 2019. Evaluating the qso contribution to the 21-cm signal from the cosmic dawn. *MNRAS* 487, 1101–1119.

Ross, H.E., Dixon, K.L., Iliev, I.T., Mellema, G., 2017. Simulating the impact of x-ray heating during the cosmic dawn. *MNRAS* 468, 3785–3797.

Ross, H.E., Giri, S.K., Mellema, G., Dixon, K.L., Ghara, R., Iliev, I.T., 2021. 2021 RAMSES-RT: shift-space distortions in simulations of the 21-cm signal from the cosmic dawn. *MNRAS* 506, 3717–3733.

Rácz, G., Szapudi, I., Dobos, L., Csabai, I., Szalay, A.S., 2019. Steps: A multi-gpu cosmological n-body code for compactified simulations. arXiv:1811.05903.

Schmidt-Voigt, M., Koeppen, J., 1987. Influence of stellar evolution on the evolution of planetary nebulae. I - Numerical method and hydrodynamical structures. *A&A* 174, 211–222.

Semelin, B., Combes, F., Baek, S., 2007. Lyman-alpha radiative transfer during the epoch of reionization: contribution to 21-cm signal fluctuations. *Astronomy & Astrophysics* 474, 365–374. URL: <http://dx.doi.org/10.1051/0004-6361:20077965>.

Shapiro, P.R., Giroux, M.L., 1987. Cosmological H II Regions and the Photoionization of the Intergalactic Medium. *ApJ* 321, L107. doi:10.1086/185015.

Smith, B.D., Bryan, G.L., Glover, S.C.O., Goldbaum, N.J., Turk, M.J., Regan, J., Wise, J.H., Schive, H.Y., Abel, T., Emerick, A., O’Shea, B.W., Anninos, P., Hummels, C.B., Khochfar, S., 2017. GRACKLE: a chemistry and cooling library for astrophysics. *MNRAS* 466, 2217–2234. doi:10.1093/mnras/stw3291, arXiv:1610.09591.

Spitzer, L., 1998. *Physical Processes in the Interstellar Medium*.

Trott, C.M., Jordan, C., Midgley, S., Barry, N., Greig, B., Pindor, B., Cook, J., Sleap, G., Tingay, S., Ung, D., et al., 2020. Deep multiredshift limits on epoch of reionization 21 cm power spectra from four seasons of murchison widefield array observations. *MNRAS* 493, 4711–4727.

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors, 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17, 261–272. doi:10.1038/s41592-019-0686-2.

Wang, Q., Meng, C., 2021. Photons-gpu: A gpu accelerated cosmological simulation code. *RAA* 21, 281. URL: <https://dx.doi.org/10.1088/1674-453X/21/2/281>.

1674-4527/21/11/281, doi:10.1088/1674-4527/21/11/281.

Watson, W.A., Iliev, I.T., D'Aloisio, A., Knebe, A., Shapiro, P.R., Yepes, G., 2013. The halo mass function through the cosmic ages. *MNRAS* 433, 1230–1245.

Wayth, R.B., Tingay, S.J., Trott, C.M., Emrich, D., Johnston-Hollitt, M., McKinley, B., Gaensler, B.M., Beardsley, A.P., Booler, T., Crosse, B., et al., 2018. The phase ii murchison widefield array: design overview. *Publications of the Astronomical Society of Australia* 35, e033.

Whalen, D., Norman, M.L., 2006. A Multistep Algorithm for the Radiation Hydrodynamical Transport of Cosmological Ionization Fronts and Ionized Flows. *ApJS* 162, 281–303. doi:10.1086/499072, [arXiv:astro-ph/0508214](https://arxiv.org/abs/astro-ph/0508214).

Zaroubi, S., 2013. The Epoch of Reionization, in: Wiklind, T., Mobasher, B., Bromm, V. (Eds.), *The First Galaxies*. Springer Berlin Heidelberg, Berlin, Heidelberg. volume 396, pp. 45–101. doi:10.1007/978-3-642-32362-1_2. series Title: *Astrophysics and Space Science Library*.