# HI Lightcones for LADUMA Using GADGET-3

## Performance Profiling and Application of an HPC Code




**Presented by:**

**Matthew Roy Cawood**

Dept. of Electrical Engineering
University of Cape Town

**Prepared for:**

**Prof. Michael Inggs**
Dept. of Electrical Engineering, University of Cape Town

**David Macleod**
Centre for High Performance Computing

**Prof. Catherine Cress**
Centre for High Performance Computing

Submitted to the Department of Electrical Engineering at the University of Cape Town in fulfilment of the requirements for the degree of

Master of Science in Engineering

**November 2014**

**Key Words:** HPC, GADGET-3, MeerKAT, performance profiling, HI lightcone

# Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature:.....

M. R. Cawood

Date:..... 13 November 2014 .....

# Terms of Reference

This project concerns the investigation, performance profiling and optimisation of the high performance cosmological code, GADGET-3. This code was used to develop a synthetic field-of-view, or lightcone, for the MeerKAT telescope to replicate what it will observe when it conducts the LADUMA ultra-deep HI survey. This lightcone will assist in the planning process of the survey.

The deliverables for this project are summarised as follows:

- Provide an up-to-date performance evaluation and optimisation report for the cosmological simulation code GADGET-3 .
- Use GADGET-3 to produce an sufficiently high resolution simulation of a region of the Universe.
- Develop a Python code to produce a lightcone which represents the MeerKAT telescope's field-of-view, by post-processing simulation output snapshots.
- Extract relevant metadata from the simulation snapshots to provide additional insight into the simulated observation.
- Produce an efficiently written and well documented software package to enable other researchers to produce synthetic lightcones.

# Acknowledgements

I would like to convey my gratitude to the following individuals and institutions who assisted me in completing this research dissertation.

Prof. Michael Inggs for his supervision and assistance in producing this dissertation.

David Macleod for providing me the opportunity to study through a CHPC studentship, as well as assistance in familiarising me with the CSIR. Furthermore, for assisting me with developing high performance code for HPC systems, and finally for proofreading the dissertation.

Prof. Catherine Cress for her guidance and tremendous assistance in the field of cosmology. Moreover, for providing immeasurable experience and collaboration opportunities in support of this research.

Dr. Sean February for his assistance with GADGET-3 and knowledge of computation astrophysics, as well as general encouragement regarding the field of astronomy.

Miss Ka Wai Ho for her support and assistance with proof reading this dissertation.

Members of the CHPC research group for assistance in data visualisation and high performance code development, as well as general support, specifically Nick Thorne, Kevin Colville and Dr. Charles Crosby.

Finally, members of the UWC computational astrophysics research group, who provided software tools and expertise in support of this dissertation. Specifically, GADGET-3 developer Prof. Romeel Davé and SPHGR developer Dr. Robert Thompson.

This dissertation was funded through a studentship granted by the Human Capital Development Programme of the CHPC, an initiative of the Department of Science and Technology (DST) and managed by the Meraka Institute of the Council for Scientific and Industrial Research (CSIR).

# Abstract

*The MeerKAT radio telescope being built in South Africa will provide unprecedented sensitivity and resolution for performing radio frequency astronomical observations. The 'Looking at the Distant Universe with the MeerKAT Array' (LADUMA) survey is one such project scheduled for MeerKAT upon its completion. LADUMA is a ultra-deep neutral hydrogen survey that aims to investigate cosmic structure formation and galaxy evolution at distances never before achieved.*

*The objective of this research was to produce an artificial patch of sky, or 'lightcone', which accurately represents MeerKAT's view of the sky. Predictions made from measurements of this synthetic field-of-view will assist in planning the LADUMA survey.*

*An investigation was conducted into the performance characteristics of the cosmological simulation code, GADGET-3. Following this process, the code was used to produce a set of simulation snapshots. A Python program was developed to process this cosmological data into a lightcone. This procedure involved stacking simulation snapshots using a method derived from the Blaizot et al. technique. The lightcone volume was extracted and a pixelisation process was applied to partition the simulation particles into a discrete three dimensional spatial grid. The code was written using efficient subroutines, with multi-threading and vectorisation implemented to improve performance.*

*Benchmarks showed that Intel's ICS software provided a significant performance improvement for GADGET-3 compared to GCC. In addition, a poor performance scaling was observed when running the code across multiple compute nodes within an HPC cluster. The Python code successfully produced a synthetic lightcone containing a pixelised volume of particle properties, with the results stored in a NumPy dense array.*

*Various analyses were conducted on the lightcone to infer information about the nature of its contents. The Python code proved very efficient; the LADUMA scale lightcone test case used for code development completed in a five minutes.*

**Key Words:** HPC, GADGET-3, MeerKAT, performance profiling, HI lightcone

# Table of Contents

# List of Acronyms

**ACE** Advanced Computer Engineering, the research and development lab within the CHPC.

**API** Application Programming Interface, a set of functions which enable the creation of applications which access the features of another service.

**ASKAP** Australian SKA Pathfinder, a new radio telescope currently being commissioned in Western Australia.

**ATLAS** Automatically Tuned Linear Algebra Software, an open source implementation of BLAS which performs architecture specific tuning.

**AU** Astronomical Unit, a unit of length which approximately equals the distance between the Earth and the Sun.

**BAO** Baryon Acoustic Oscillations, regular and periodic fluctuations in the density of the ordinary matter within the Universe.

**BLAS** Basic Linear Algebra Subprograms, a set of routines that provide standard building blocks for performing basic vector and matrix operations.

**CDM** Cold Dark Matter, a hypothetical form of matter which moves slowly compared to the speed of light and interacts weakly with EM radiation.

**CentOS** Community Enterprise Operating System, an open source enterprise Linux distribution.

**CHPC** Centre for High Performance Computing, a South African research facility which hosts high performance computing infrastructure.

**CMB** Cosmic Microwave Background, the thermal radiation assumed to be remnants of the "Big Bang" event.

**CMP** Cloud Management Platform, a set of technologies for managing and monitoring resources residing in cloud-like environments.

**COTS** Commercial-off-the-Shelf, commercial products which can be purchased from a marketplace, often configured for specific applications.

**CPU** Central Processing Unit, the main processing unit in a computer, typically used in reference to a von Neumann type processor.

**CSIR** Council for Scientific and Industrial Research, South Africa's central and premier scientific research and development organisation.

**DFT** Discrete Fourier Transform, a numerical variant of the Fourier Transform.

**DRAM** Dynamic Random Access Memory, the most common kind of random access memory (RAM) for personal computers and workstations.

**E-CDF-S** Extended Chandra Deep Field-South, a survey conducted in the X-Ray spectrum, in the southern hemisphere sky.

**FDR** Fourteen Data Rate, a standard for the Infiniband network protocol designating the 56 Gbps data rate implementation.

**FFT** Fast Fourier Transform, an efficient algorithm to compute the discrete Fourier transform of an input vector.

**FIFO** First In First Out, a data structure resembling an ordered queue, in which the first element is addressed first.

**FITS** Flexible Image Transport System, a data container standard for storing 3D astronomical data.

**FoV** Field-of-View, the extent of the observable sky which can be observed at any given moment.

**GADGET** GAlaxies with Dark matter and Gas intEracT, an SPH simulation code designed for cosmological simulations and developed to run on distributed memory systems.

**GCC** GNU Compiler Collection, a compiler system produced by the GNU Project supporting the compilation of various programming languages.

**GPGPU** General Purpose Graphics Processing Unit, a GPU used for general purpose computing.

**HCA** Host Channel Adaptor, a network fabric interconnect based on InfiniBand technology.

**HI** Hydrogen I, the neutral atomic form of hydrogen which emits 21cm wavelength EM radiation.

**HPC** High Performance Computing, use of parallel processing for running advanced application programs efficiently, reliably and quickly.

**ICS** Intel Composer Suite, a set of compilers developed by Intel Corp. for use with their microprocessors.

**ICs** Initial Conditions, a large dataset which defines the initial the state of a particle simulation such as GADGET-3.

**IPoIB** IP over Infiniband, standardised IP encapsulation for InfiniBand interconnect fabrics.

**KAT** Karoo Array Telescope, a 7 dish radio telescope built in South Africa, completed in 2012.

**LADUMA** Looking at the Distant Universe with the MeerKAT Array, an ultra deep neutral hydrogen survey planned for the MeerKAT telescope.

**LFAA** Low-Frequency Aperture Array, an SKA project which will perform low frequency radio observations from 50MHz up to 350MHz.

**LHC** Large Hadron Collider, a particle accelerator developed by CERN beneath the Franco-Swiss border.

**MCA** Modular Component Architecture, a series frameworks, components, and modules which forms the backbone of OpenMPI's functionality.

**MIC** Many Integrated Core architecture, a multiprocessor computer accelerator architecture developed by Intel.

**MKL** Math Kernel Library, a set of numerical libraries developed by Intel Corp. for use with their microprocessors.

**MPI** Message Passing Interface, standardised and portable message-passing system designed by distributed computing systems.

**MWA** Murchison Widefield Array, a low-frequency radio array operating in the frequency range 80300 MHz, situated in Australia.

**NUMA** Non-Uniform Memory Access, a memory design used in distributed multiprocessing, where memory access time depends on relative memory location.

**OFED** OpenFabrics Enterprise Distribution, a set of software standards for the Infiniband network protocol.

**PAPER** Precision Array for Probing the Epoch of Re-ionisation, a radio interferometer built to detect 21 cm Hydrogen occurring when the first galaxies ionized intergalactic gas.

**RDMA** Remote Direct Memory Access, a memory access from a computer into a network attached destination without involving the operating system.

**SAM** Semi-Analytical Model, a simulation approach which uses numerical as well as theoretical methods.

**SKID** Spline Kernel Interpolative Denmax, a numerical method for determining the spatial locality of a set of particles in 3D space.

**SMP** Symmetric Multiprocessor, a computer architecture where two or more identical processing units connect to a single, shared main memory.

**SPH** Smoothed Particle Hydrodynamics, a computational method used for simulating fluid flows, often used in astrophysical simulations.

**WMAP** Wilkinson Microwave Anisotropy Probe, a NASA Explorer mission that launched June 2001 to take measurements of the CMB.

# List of Figures

# List of Equations

# 1 Introduction

This dissertation investigates the construction of a synthetic observable sky, or 'lightcone', to assist in planning the LADUMA survey which will be conducted by the MeerKAT telescope. This lightcone was developed by post-processing GADGET-3 simulation data into a form which resembles a telescope's view of the cosmos. This introduction begins with a brief background of the SKA project and the field of HPC, followed by the project's objectives and concluding with an overview of the remaining chapters.

## 1.1 Research Background

The Square Kilometre Array (SKA) is an international scientific collaboration, with the goal of looking deeper into the origins of the Universe to answer some of humanity's biggest questions. This project involves constructing large radio interferometers in Australia and South Africa. The development process for this ambitious scientific and engineering project necessitates simulation and modelling, as well as hardware prototyping. These prototypes, or precursor telescopes, are currently being developed to assess the effectiveness of designs and technologies for the greater SKA project. One such precursor, the MeerKAT telescope, is an initially independent 64 dish radio interferometer which will serve as a technology showcase and later form part of the larger SKA Phase 1 facility. Although MeerKAT will be smaller than the SKA, it will still be one of the most powerful radio telescope ever built and promises to expand the forefront of radio astronomy when it begins observations.

'Looking at the Distant Universe with the MeerKAT Array' (LADUMA) is one of the two Priority Group 1 surveys planned for the MeerKAT telescope once completed [17]. This survey aims to probe the neutral hydrogen content of the Universe with previously impossible sensitivity. Currently, 5000 hours of MeerKAT's observation time have been committed to performing the LADUMA survey [18]. The MeerKAT project is estimated

to cost more than two billion rand. Due to this sizeable financial investment, it is imperative that the survey produces meaningful results. Planning a survey and interpreting the results from observations requires detailed simulations, including the production of synthetic lightcones. The MeerKAT telescope and LADUMA survey are detailed further in Section 2.2 of the literature review.

Due to the large amounts of data produced by scientific instruments such as the MeerKAT telescope, High Performance Computing (HPC) is often required to process, analyse and store the results. Inter-disciplinary collaboration is a necessity when developing effective engineering solutions to meet the scientific goals of projects like the SKA. The Centre for High Performance Computing (CHPC) is Africa's largest facility for HPC and provided the computing infrastructure to conduct this research.

## 1.2 Objectives

This dissertation presents research conducted for producing artificial telescope data for the LADUMA survey; a high resolution deep sky radio survey planned for the MeerKAT radio telescope. The data resembles a portion of the sky as seen by the telescope. It can be used to assist in planning the survey as well as to test data processing algorithms, such as source finding. The process of producing a virtual 'radio sky' (referred to as a 'lightcone' given its geometry) requires an accurately simulated region of the observable Universe generated by the GADGET-3 scientific code. This simulation data must be post-processed to resemble a portion of the sky (lightcone) visible to the MeerKAT telescope. This radio sky analogue must contain the data necessary to perform validation and other source finding processes. Such information includes; neutral hydrogen mass, received flux from radio sources, radio continuum caused by star formation and line-of-sight peculiar velocities. The data should be contained in an appropriate data structure so that it can be conveniently stored and accessed.

In addition to the main scientific deliverable for this research, a number of supplementary investigations are presented. An analysis of the GADGET-3 scientific simulation was conducted to assess procedures for optimising the runtime performance of the code in order to reduce runtime. A report of this evaluation is presented to assist members of the computational astrophysics community with maintaining well optimised code within the ever-changing HPC field. Finally, a brief overview of NumPy optimisation for Python will be presented, which was explored as part of the lightcone post-processing code development process.

### 1.2.1 Purpose of the Study

The process of planning a radio telescope survey is a complicated endeavour. A vast amount of money and time is dedicated to producing results. A synthetic (or virtual) 'image' from the telescope is useful for predicting the results of a hypothetical observation, before physical resources are expended. Producing an accurate input dataset which accurately represents the radio sky is a complex task due to the scale of MeerKAT observations. Intricate physical and cosmological processes determine the contents of a radio telescope's Field-of-View (FoV).

This study aims to provide a realistic virtual FoV of the MeerKAT telescope. The data will be in the form of a lightcone which contains information which would ordinarily be received by the MeerKAT's front-end radio interferometer. This data must be stored in an efficient manner to allow astronomers to analyse it based on their requirements. Focus will be placed on documenting the configuration and optimisation of the GADGET-3 scientific code to support the active community developing and maintaining the code.

### 1.2.2 Research Questions

A number of research questions were developed based on the objectives and requirements described above. These questions identify the main focus areas for this research and will be used when evaluating the success of the project in the Conclusion. In completion of this dissertation, the following problems will be addressed:

- What effect does compiler selection and software environment optimisation have on GADGET-3 performance?
- Which post-processing technique is best suited for converting GADGET-3 simulation data into a lightcone?
- How effective is the Python programming language for scientific processing tasks such as constructing lightcones?
- How can simulation metadata, including masses and flux, be stored inside a virtual lightcone?
- How can a large multidimensional data structure, such as the LADUMA lightcone, be stored to hard disk?

- What is the most effective approach for visualising complex multidimensional data to enable intuitive observations.

The questions answered in this research will aid in developing a program for constructing virtual lightcones. These lightcones will potentially be used to aid in the development of future neutral hydrogen surveys, with specific focus currently placed on the LADUMA survey. Researchers planning such surveys would greatly benefit from a high performance and fast survey simulation which can provide predictions of detectable sources and mass distributions.

## 1.3 Scope and Limitations

The scope of this study is to produce a simulated radio sky to assist in planning the LADUMA survey. In addition, a comprehensive study of the cosmological simulation code GADGET-3 will be presented. The scope is limited by a number of factors:

- GADGET-3 will be the only simulation code examined for generating cosmological data. Other codes, such as Enzo, will not be considered due to the familiarity that the local academic community has with GADGET.
- Due to hardware availability, the GADGET-3 code will be benchmarked on an HPC cluster using a maximum of six compute nodes, comprising a total of 120 CPU cores. Further detail of the hardware utilised in this research is provided in Section 2.3.2.
- The smallest galaxies in the Universe will not be represented in the simulation volume as they will be below the resolution limit.
- The first phase of the LADUMA survey is planned to operate to a minimum frequency of 900 MHz. The post-processing tool will be developed to be capable of a minimum upper redshift limit of z=1.0 to improve it's flexibility and potential for use in other applications. While higher redshifts may be possible, it will not be officially supported due to the added complexity of development.

Furthermore, the contents of the lightcone will be limited to properties which can be extracted from the GADGET-3 simulation outputs, or can be extrapolated from this data

in a straight forward manner. Focus will be placed on producing efficient code which is not wasteful of hardware resources. However, due to the nature of the computational and memory requirements imposed by the size of input data, the code will require specialised computer hardware to operate on typical problem sizes. The GADGET-3 code will be evaluated to determine its efficiency as well as to report on the progress of the code which is still in development. Focus will not be placed on analysing the fundamental code mechanics of GADGET-3, as this lies outside the scope of this research.

## 1.4 Dissertation Outline

This dissertation continues as follows:

Chapter 2 will present a comprehensive literature review on relevant topics in cosmology, the Square Kilometre Array and High Performance Computing. In order to provide context for this research, a concise introduction to cosmology and galaxy evolution is presented. This covers topics including the standard model of cosmology, referred to as the Lambda-CDM model, as well as cosmological expansion and redshift. A brief explanation of the hyperfine transition which occurs in a neutral hydrogen atom (referred to as HI) is provided to illustrate the importance HI plays in astronomy, specifically in radio frequency observations. Expanding on this topic, a history to the field of radio astronomy is presented. Current trends, projects and obstacles within the field of observational radio astronomy are discussed and the SKA as well as the MeerKAT precursor telescopes are introduced. Details for the LADUMA deep sky HI survey are presented which forms the framework in which this research project falls. Figure 1.1 and 1.2 provide a comparison between an optical survey conducted with the Hubble Space Telescope and a virtual lightcone containing simulated galaxies. Detailed simulations such as this one are necessary for planning surveys such as LADUMA and interpreting the results.

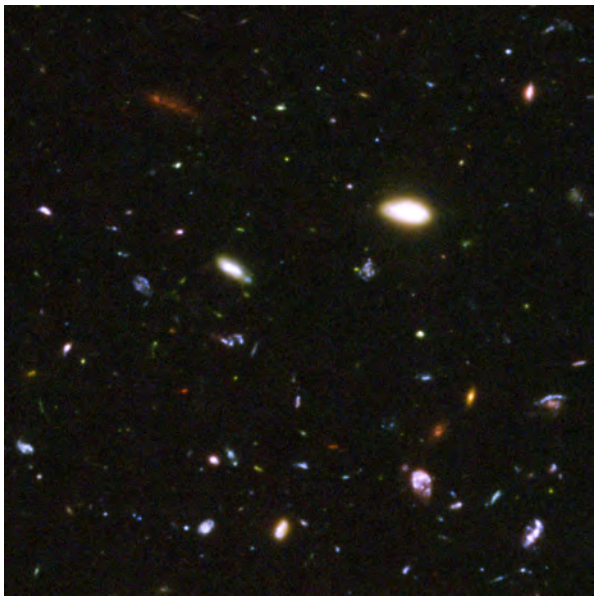

Figure 1.1: An image captured by the Hubble Space Telescope as part of the Ultra-Deep-Field survey, where each light source is a galaxy [1].

Figure 1.2: A visualisation produced from a synthetic lightcone, or mock map, illustrating the apparent magnitudes of simulated galaxies [2].

A history of High Performance Computing (HPC) is given, followed by a summary of current trends and areas of interest. This continues on to an introduction of the Centre for High Performance Computing (CHPC) and its Advanced Computer Engineering (ACE) lab. Details are provided on the hardware and software environment made available for conducting this research.

An overview of the HPC cosmological simulation code GADGET-3 is presented. This code is the latest release of the N-body particle simulation used for modelling gas and dark matter interaction and formation on cosmological distance and time scales. A history of its development and a breakdown of the code mechanics provide some background into the code used for producing the deliverable of this research.

Finally a number of published papers with similar research themes are presented. The papers include the 2013 Davé *et al.* which describes a GADGET HI post processing technique and the 2010 Carlson & White paper describing a 3D volume remapping of simulation data. The 2009 Obreschkow *et al.* paper on a cone construction technique using volume stacking is also reviewed. Finally a method for converting HI mass to flux spectral density is presented in the 2004 Abdalla and Rawlings paper.

Chapter 3 presents the research methodology overview of this research. This includes detailing the process followed to install and run the GADGET-3 simulation code, as

well as the visualisation and validation of simulation results. The process of selecting the optimal method for post-processing the simulation data is also documented, which includes implementing and evaluating the methods introduced in the literature review. The post-processing method selected for this research is presented followed by a motivation of how this method improves upon existing techniques.

A walk-through is provided for compiling and configuring the GADGET-3 simulation code. This code was used to generate the raw cosmological data from which the synthetic lightcones are produced. Optimisation of HPC codes, such as GADGET-3, is crucial for efficient operation within the expensive hardware infrastructure of HPC facilities. Figure 1.3 presents an explanation of the scale simulated by codes like GADGET-3, illustrating the need for HPC resources.

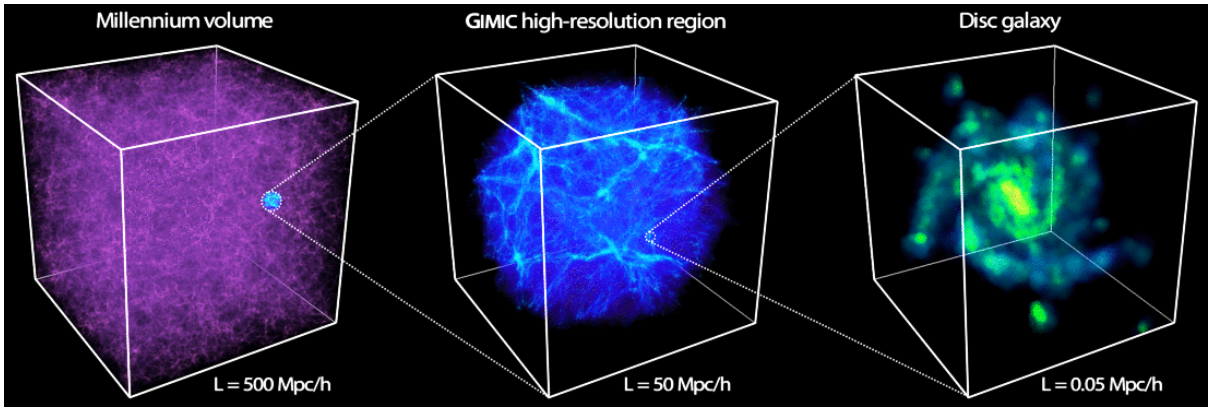


Figure 1.3: A series of images illustrating the scale of cosmological simulations. The first image shows the GADGET-2 Millennium Simulation, with a volume of $500^3(Mpc/h)^3$. The middle image presents a GIMIC high resolution simulation of a subregion one thousandth the scale of Millennium. Finally, a simulation of a single disc galaxy is shown which occupies a region $1x10^9$ times smaller than the GIMIC simulation [3].

This chapter also includes a more comprehensive break-down of the methodology followed during the performance evaluation of the GADGET-3 code. This includes identifying the components which contribute to GADGET-3 performance within the software environment of a computer cluster and developing means of measuring performance. Finally a detailed description of the development process for the lightcone post-processing code is presented. This focuses on describing the software tools which were developed and the way in which these programmes interface. Python code optimisations, which were extensively researched, are also introduced in this section.

Chapter 4 provides specific details of the software tool developed for this research used to process GADGET-3 simulation data into a synthetic radio telescope lightcone. The

Research Methodology chapter provided an overview of the development process as well as design choices based on an evaluation of existing research. This chapter builds on this with a more in-depth look at the code.

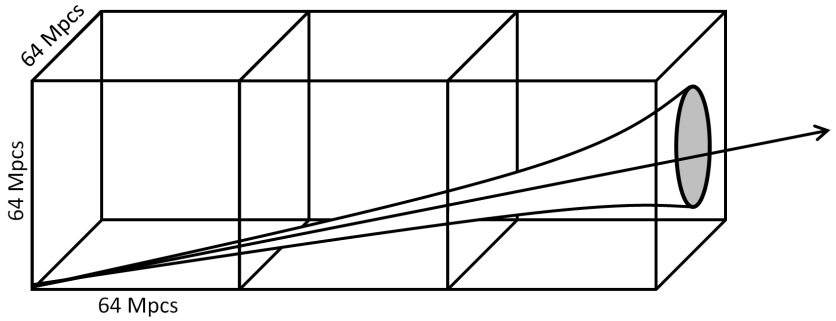


Figure 1.4: An illustration depicting the lightcone construction process. The lightcone volume is extracted from a series of stacked GADGET-3 simulation cubes, according to a relation between beam-width and frequency.

Details are provided for the lightcone construction algorithm used to extract particles from GADGET-3 simulation snapshots. Figure 1.4 provides a conceptual overview of the simulation post-processing technique. This includes an explanation of the lightcone's geometry, which uses a Cartesian co-ordinate system and a distance-to-radius relation to calculate the boundaries of the cone's volume. The pixelisation scheme used to collapse the data into a grid structure is also described, this method was implemented to reduce the large amount of simulation data contained within the lightcone as well as to recreate the sky as seen by a radio telescope. Several of the particle post-processing methods for calculating properties of simulation gas particles are also explained. Finally the NumPy data structure used for storing the multi-dimensional lightcone contents to hard disk is described. Figure 1.5 presents this data structure graphically, showing the volumetric partitioning of the lightcone in three dimensions.

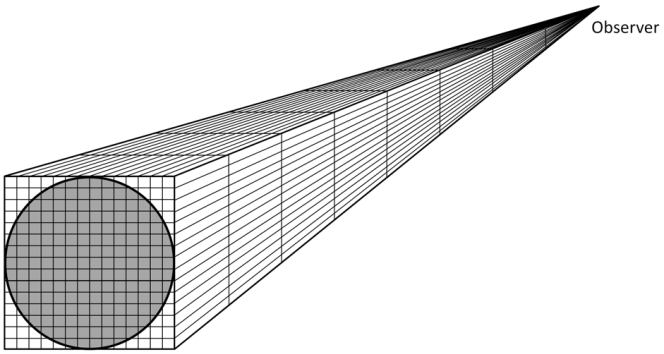


Figure 1.5: An image portraying the pixelisation grid which partitions the lightcone volume in three dimensions. Each pixel holds a set of properties for all the particles contained within that region of the lightcone. The pixels are sized proportional to their contents, with the dark matter, ionized gas, HI and stellar mass contents overlayed - in that order.

Chapter 5 presents a collection of results produced during this research. A selection of performance results are presented from the GADGET-3 performance optimisation process. These results focus on the compilers, libraries and the software environment upon which GADGET-3 relies. It was found that the Intel Compiler Suite provided a reasonable performance increase over the open source GCC compiler. Unfortunately, in its current state, the GADGET-3 code demonstrated poor performance scaling within an HPC cluster.

This chapter also presents the results of several Python optimisations implemented in order to reduce the post-processing time taken to generate a lightcone from simulation data. Results from the NumPy and SciPy library optimisation are presented which show real performance improvements by wrapping a high performance math library such as Intel MKL or ATLAS. The use of parallelism via multiple execution threads also showed significant performance improvements. In order to further reduce CPU workload and increase performance, the code itself was optimised to reduce the number of function calls and avoid costly operations when possible. The last optimisation result presented is the vectorisation of the Python code, which showed a tremendous increase in code efficiency and runtime reduction. The lightcone code was parallelised in order to take full advantage of the multi-core computers available.

Finally a number of visualisations are presented of the lightcone produced in this research. These results increase various statistical measures such as mass functions and peculiar velocity plots, as well as 3D visualisations and other data validations. Several test cases are presented which include failure conditions, illustrating the limits of the code in its

current form. A feature of the code allows the user to define the resolution at which the lightcone is pixelised, this allows for a multi-resolution analysis of the lightcone's contents. Figure 1.6 illustrates the contents of three stacked frequency slices within a relatively high resolution lightcone.

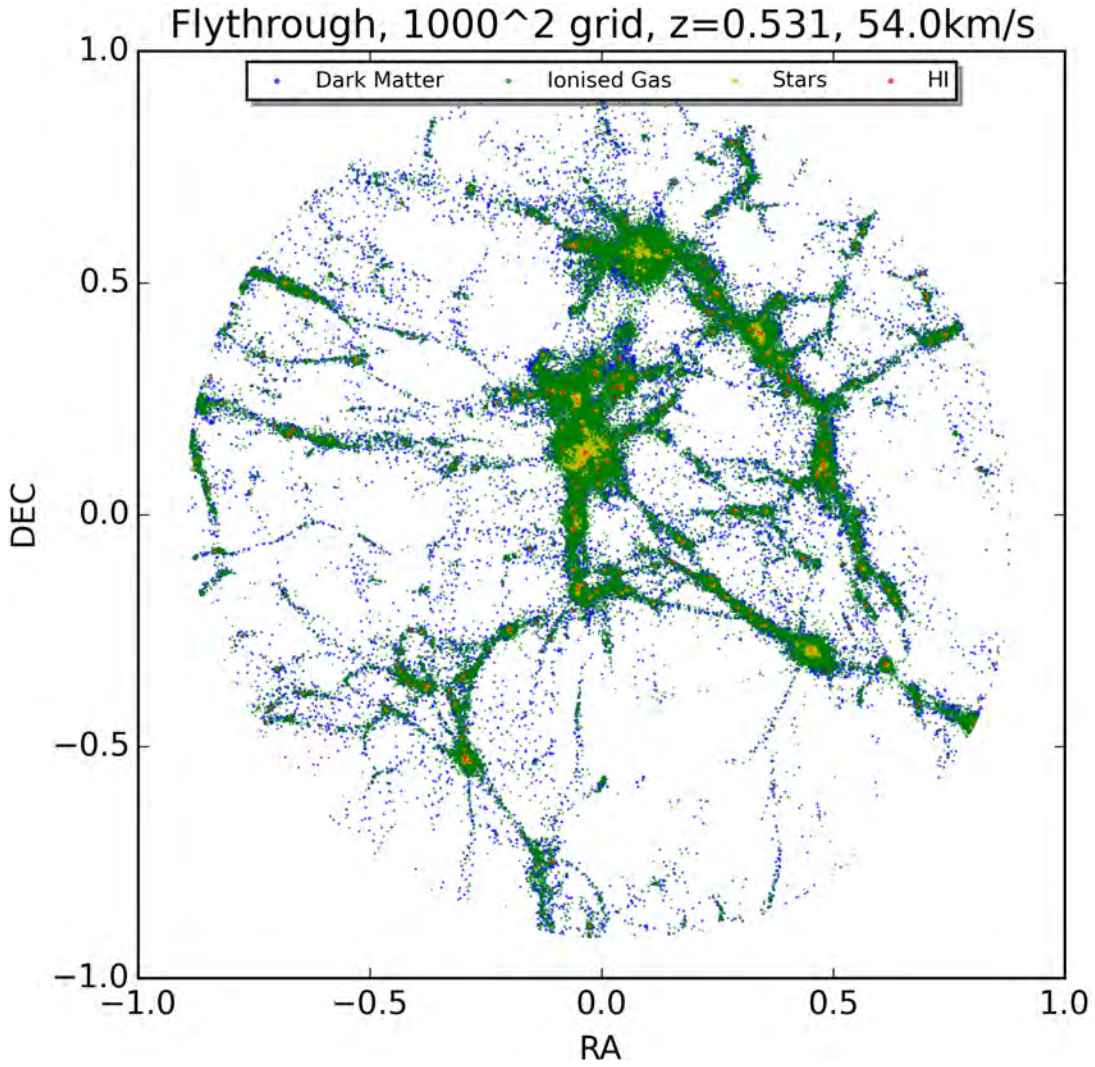


Figure 1.6: A set of three stacked lightcone frequency slices, showing the grid of angular pixels. Each dot represents the contents of a pixel, with blue denoting dark matter, green ionised gas, red HI gas and yellow stellar mass.

Chapter 6 provides a discussion of the results presented in the previous section. The performance achieved through optimising the GADGET-3 code is examined and explanations provided for some of the results observed. Focus is placed on discussing the performance improvements observed from the Intel ICS compiler, as well as the lack lustre performance scaling observed within the HPC cluster.

The results of the Python performance optimisations incorporated into the lightcone

code are also reviewed. Specifically the performance improvements achieved through parallelisation and Python vectorisation are considered. The advantages and disadvantages of several design decisions and a motivation supporting the selected approach are provided. A justification is also provided for several of the limitations placed on lightcone construction.

Finally, the selection of lightcone results, previously presented in Chapter 5, are discussed in order to determine their validity. Statistical measurements such as mass functions are discussed and compared with findings from existing research. In order to justify the design decisions made during code development several of the outputs will be considered.

Chapter 7 draws conclusions from the results and observations presented. The GADGET-3 code provided simulation data for the lightcone construction. The Obreschkow method of simulation cube stacking was implemented in order to avoid the one-to-one volume relation that exists with the Carlson & White remapping approach. The cone was constructed from a volume generated with this technique, with special focus placed on maintaining consistency when converting simulation data to a synthetic observation. The merits and drawbacks of this lightcone construction methodology are further discussed.

The success of the research is determined by considering the design requirements and research objectives stated above. Results such as the calculated flux spectral density received from HI gas within the lightcone are reviewed. Several recommendations are also given for expanding upon this research in future work. This includes modifications to the lightcone construction algorithm, allowing it to operate in a wider range of conditions. Finally, several options are proposed for expanding upon the lightcone's data structure to include additional information.

# 2 Literature Review

This chapter will present a background to topics relevant to this research. An introduction of several cosmological concepts is provided in Section 2.1 to provide a scientific context for this research. This includes the standard model of cosmology, referred to as the $\Lambda CDM$ model, as well as cosmological expansion and redshift. In addition, the importance of the neutral hydrogen atom, referred to as HI in the astronomical community, in radio frequency observations is clarified.

Thereafter a brief overview of the conception and evolution of the field of radio astronomy is presented Section 2.2. A background into the South African centred SKA project is presented to provide a more detailed engineering context. The MeerKAT 64 dish precursor telescope is also introduced, as it is the scientific apparatus for which this research is conducted. Details regarding the 'Looking at the Distant Universe with the MeerKAT Array' (LADUMA) survey, planned for MeerKAT, are presented to support the parameters and constraints established for this research.

An overview of High Performance Computing (HPC) topics relevant to this field of study, including the Centre for High Performance Computing (CHPC), is presented in Section 2.3. The hardware and software specifications are presented to provide a framework for the HPC applications used.

Section 2.4 will provide an overview of the history and code mechanics of GADGET-3, an HPC N-body cosmological simulation code. This code was used to generate the raw cosmological data from which synthetic lightcones were produced. Finally a review and comparison of several existing studies with similar themes is highlighted in Section 2.5.

## 2.1 Background to Cosmology and Galaxy Evolution

Physical cosmology refers to the study of the origin and evolution of our Universe, containing both theoretical and observational disciplines. Physical cosmology emerged in the early 20th century with Albert Einstein's Theory of General Relativity, followed by Edwin Hubble's observations of extra-galactic objects outside of our own Milky Way [19]. Some of the current areas of research within the field of physical cosmology include: measuring the Cosmic Microwave Background (CMB) in order to more accurately parametrise the $\Lambda CDM$ model; the study of large scale structures within the Universe in support of the Standard Model; and investigating the nature of dark matter and dark energy [20].

### 2.1.1 The Standard Model, Cosmological Expansion and Redshift

The Big Bang theory is a cosmological model which describes the evolution of our Universe, it was first proposed in 1927 by Georges Lematre [21]. Two years later, it was directly observed by Edwin Hubble when he discovered that the apparent velocity of distant galaxies was proportional to their respective distances, regardless of direction. The models developed to describe the early Universe and the sequence of events thereafter has been extrapolated from what we can directly observe today [22].

The Big Bang is believed to be the single event circa 13.7 Gyr (1 Gyr = $10^9$ years) ago [23]. It began as the sudden existence of space-time, which is the four dimensional continuum that relates time and space. At the beginning, the Universe was a singularity - an infinitely dense region of space-time which causes Albert Einstein's laws of general relativity to breakdown. The singularity immediately expanded in all dimensions. Due to the high temperature and pressure of the early Universe, matter existed in a plasma form of all elementary particles. It is believed that within the first 20 minutes of the Universe's existence, Big Bang nucleosynthesis occurred. During this period, fusion of primordial hydrogen nuclei (single protons) produced heavier isotopes such as helium-4, helium-3, lithium-7 and the hydrogen isotope deuterium [24].

After approximately 378,000 years, the early Universe had expanded and cooled enough to allow the plasma of protons, electrons and neutrons to bind together into neutral hydrogen in a epoch known as Recombination [25]. Shortly after this period, photon decoupling occurred and the Universe became opaque to radiation, allowing it to travel freely without interacting with matter. We can witness this event in the CMB, observed

as the visible afterglow of this transition [26]. The CMB is the oldest light in the Universe - simply because light did not travel before the photon decoupling event. The CMB is almost entirely uniform in all directions. Thus its discovery in 1964 provided strong evidence of an expanding Universe and resolved a divided opinion within the scientific community regarding the origins of the Universe. The CMB has a thermal black body spectrum, and originally had a temperature of approximately 3000 K. However due to redshift, the signal we receive only has a temperature of 2.7260 +- 0.0013 K. The incredible uniformity of CMB measurements provides support for the Big Bang cosmological model [27]. Figure 2.1 illustrates the CMB as measured in all directions by the Wilkinson Microwave Anisotropy Probe (WMAP) CMB survey. After removing the mean temperature and other interferences, the CMB contains very small fluctuations. This very slight unevenness is believed to be a result of quantum perturbations which occurred in the very early Universe. These perturbations produced very small over-densities on a quantum scale. As the Universe expanded, these variations in density allowed gravity to draw matter together, from low density into high density regions. Over cosmic time scales, this resulted in the formation of large scale structures (or 'cosmic web') of matter we observe today. Figure 2.2 presents a visualisation of a GADGET cosmological simulation conducted by the Max Planck Institute, showing this cosmic web. Measurements of the nearly uniform CMB have also provided supporting evidence that the curvature and topology of the Universe is flat, or nearly flat, within a 0.4% margin of error [28].

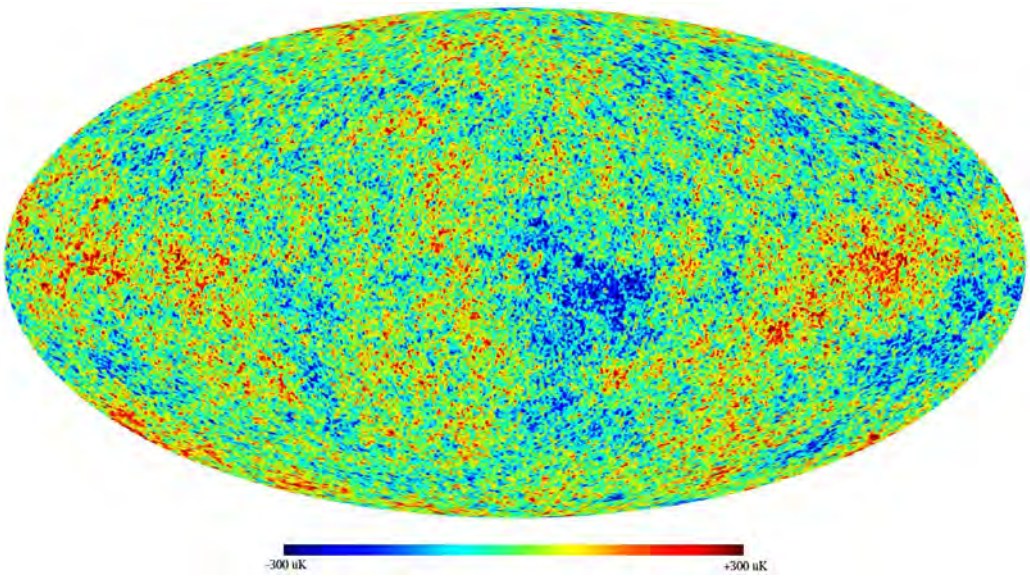


Figure 2.1: The 'all-sky map' from the 9 year WMAP Cosmic Microwave Background survey, showing the small fluctuations after subtracting the mean, dipole and the galactic disk. The positive and negative variations in measured temperature are displayed as red and blue respectively while the mean is represented as cyan [4].

During observations of other galaxies, astronomers discovered that all the galaxies appeared to be 'missing' matter. Simulations conducted could not replicate their characteristics, such as rotational velocity, without adding in additional mass [29]. The galaxies simply appeared to be spinning too fast given their mass, calculated from their apparent luminosity [30]. This lead to the proposal of a hypothetical form of matter referred to as dark matter. It was called dark matter because it interacted very weakly with electromagnetic radiation. The most common form of dark matter is cold dark matter (CDM). It is considered 'cold' because it moves relatively slowly compared to the speed of light. CDM is considered to be a collisionless substance as it does not directly react with the ordinary observable matter, however it operates according to regular laws of gravity. By adding this mysterious substance to simulation models, scientists were able to not only replicate the formation of rotating galaxies but also other large structures. These models showed that for galaxies to exist as we observe them, they must contain approximately five times as much cold dark matter compared to ordinary matter by mass.

Through observations of extra-galactic objects conducted in the '90s, it was concluded that our Universe was expanding at an increasing rate [31]. This observation contradicted much of the theory at the time, which assumed the initial expansion force caused by the Big Bang should have slowed. Thus at some point the expansion would eventually stop and a collapse would begin, due to the gravitational forces between all matter within the Universe. The increase in expansion rate suggested that there were previously unknown forces at work. This phenomenon has been defined as a form of energy; namely dark energy [32]. Like dark matter, dark energy does not directly interact with ordinary matter. However unlike dark matter, dark energy is not affected by gravity. Instead it acts to fuel expansion and inflate the Universe. Using the massenergy equivalence theory, the mass density of dark energy was calculated to be extremely low, only $1.67\times10^{27}$ kg/m$^3$. However because it is believed to be uniformly distributed throughout the Universe, it constitutes the large majority of all matter. The mass-energy distribution of the known Universe is calculated to be 4.9% baryonic matter (ordinary matter we can observe), 26.7% dark matter and 68.3% dark energy [33]. Figure 2.4 provides a graphical representation of this mass-energy distribution, illustrating just how diluted the baryonic matter is.

The theory of an expanding Universe, as described by the Big Bang theory, has been parametrised into a mathematical model referred to as the $\Lambda CDM$ model, or standard model of cosmology [34]. Lambda ($\Lambda$) is the term given to the cosmological constant associated with dark energy and cold dark matter (abbreviated to CDM). This model operates on the cosmological principle, which states that, if viewed on a sufficiently large scale, the Universe is uniform in all directions (isotropy) and from any location

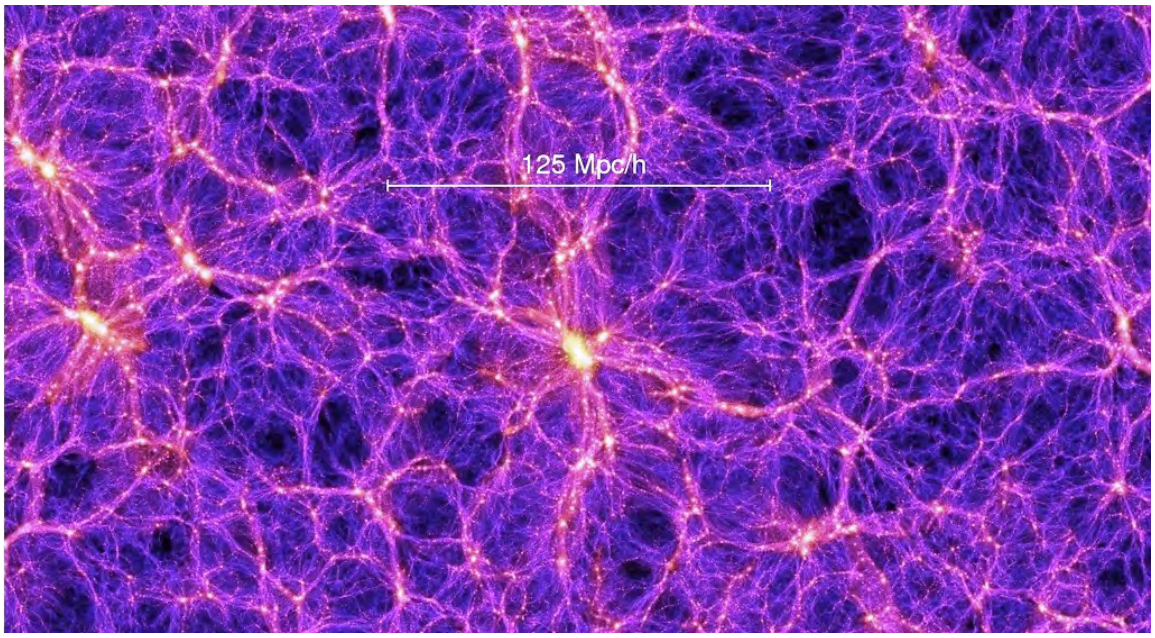


Figure 2.2: A visualisation of the Millennium dark matter simulation, showing the present day large scale structure. This is known as the 'cosmic web'. Colour denotes density - with yellow being high density and blue low density [5]

(homogeneity) thus our location in the Universe is not significant. Although referred to as the standard model, much scientific uncertainty still surrounds the $\Lambda CDM$ model with particular concern regarding the unproven hypothetical dark energy. Opposing theories which describe the state and evolution of the Universe are continually proposed and debated [35]. Figure 2.3 provides a condensed graphical description of the Universe's time line, from the Big Bang to the present day. It shows the hot and dense beginning, followed by a period of rapid expansion. Over time, matter cooled and condensed enough to form stars and galaxies. Thus the Universe was filled with the objects we observe today.

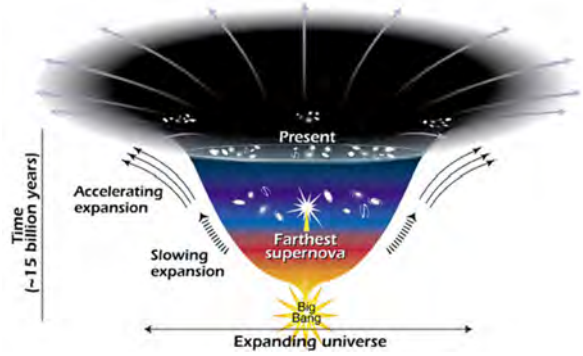


Figure 2.3: Diagram showing the expansion of the Universe from its early hot, high density origins to the present [6].

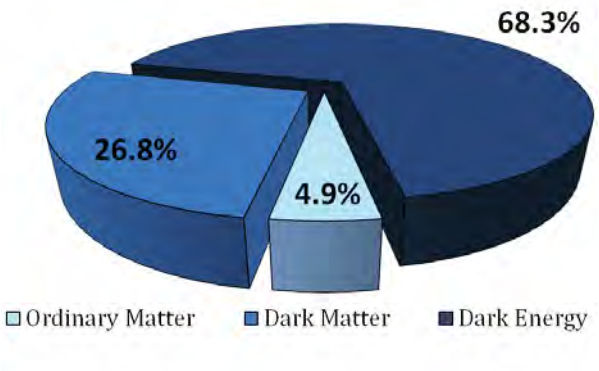


Figure 2.4: Diagram of the matter-energy distribution of the present day Universe [7].

The constant speed of light of $3\times10^8$ m/s defines that emissions detected from distant sources have been travelling for some period of time before they are detected by our eyes (or telescopes). The further away the object is, the more time the light must travel. The light-year unit for measuring large distances was defined for this reason. We perceive an object one light-year ($9.4\times10^{12}$ km) away as it was one year ago because its light has taken one year to reach us. Another unit used to measure scales larger than our solar system is the parsec (pc), defined as the distance at which a disc with a radius of 1 Astronomical Unit (AU) appears to subtend an angle of 1 arc-second, or 3.26 light-years [36]. The average distance between galaxies is approximately 1 Mpc ($3.26\times10^6$ light-years).

Because the Universe is expanding, electromagnetic radiation travelling through expanding space-time is effectively stretched. In the visible spectrum, this effect is observed as a 'reddening' of the original light source, thus this astronomical phenomenon is named cosmological redshift. This phenomenon appears similar to the Doppler Effect. A signal emitted by a source moving at some velocity relative to an observer is perceived at a different frequency, related to the relative velocity. Redshift cannot be considered equivalent to the Doppler Effect however. Two objects with zero velocity will experience no Doppler frequency shift in a signal sent between them. However on a sufficiently large scale, the same two objects could perceive a frequency shift caused by cosmological expansion, due to expansion of the space between them [37]. The redshift of the object can be calculated by measuring the frequency shift of spectral features (such as absorption and emission lines) between the emitted and the detected signal. The redshift of an object is defined as by the equation:

$$z = \frac{\lambda_0 - \lambda}{\lambda} \tag{2.1}$$

where z is the symbol for redshift, $\lambda$ is the emission wavelength and $\lambda_0$ is the observed wavelength. This relation yields a redshift range of zero to infinity, with zero representing the present day (i.e. no change in apparently wavelength), and redshift of infinity representing the Big Bang. Due to the ratio of wavelengths, a redshift of z=1, represents a wavelength shift corresponding to approximately half the age of the Universe. Figure 2.4 shows the apparent change in frequency of a received signal, relative to a non-redshifted source. This effect is fundamentally important to observational astronomy, as it creates a relationship between a source's apparent velocity and its distance from the observer.

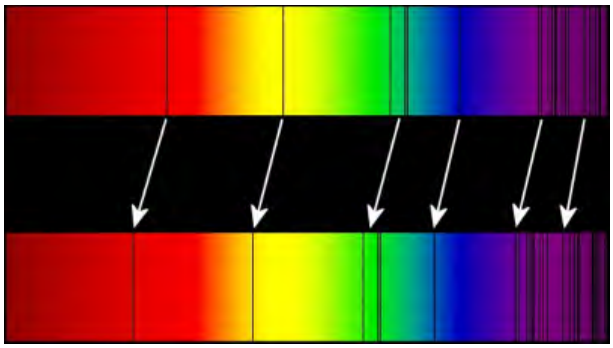

Figure 2.5: Diagram illustrating the frequency shift (redshift) resulting from a signal's passage through expanding space-time [8]. This shift results in a emitted signal being detected at a lower frequency. If the original emission frequency is known, the distance to the source of the signal can be calculated using Equation 2.1.

### 2.1.2 The Importance of Atomic Hydrogen

The hydrogen element exists in three forms; atomic, molecular and ionic. On Earth, hydrogen forms diatomic molecules where it exists as bonded hydrogen atom pairs. Alternatively, it forms part of other molecular compounds. However atomic hydrogen (a single proton orbited by a single electron), referred to as HI in the astronomical community, is of great significance in radio astronomy [38]. HI atoms are abundant and ubiquitous in regions of low density, often found in the Inter Stellar Medium (ISM) of galaxies. HI also has a very precise emission signature which can be used to determine its redshift accurately.

An atomic transition exists between two hyperfine levels of the atom's ground state. It occurs statistically once every 10 million years (fortunately as mentioned, HI is abundant). When this transition occurs, the change in energy state produces electromagnetic radiation at the very precise frequency of 1420.40575177 MHz with a corresponding wavelength of 21.10611405413 cm in a vacuum [39]. Figure 2.6 illustrates this 'spin flip' effect and the resulting EM emission. This emission falls into the microwave-radio band of the EM spectrum and therefore can penetrate interstellar debris, such as clouds of cosmic dust, encountering less interference than shorter wavelengths such as visible light. This makes the HI emission (often referred to as 21 centimetre line or HI line) a visible and highly accurate astronomical indicator for observing the matter distribution in a large scale cosmological structure formation throughout the age of the Universe.

It is interesting to note, the HI line frequency was considered of such scientific significance that it was etched onto the gold-anodised aluminium plaques of the Pioneer 10 and 11 spacecraft [40]. The plaques were attached to the deep space probes as part of a SETI

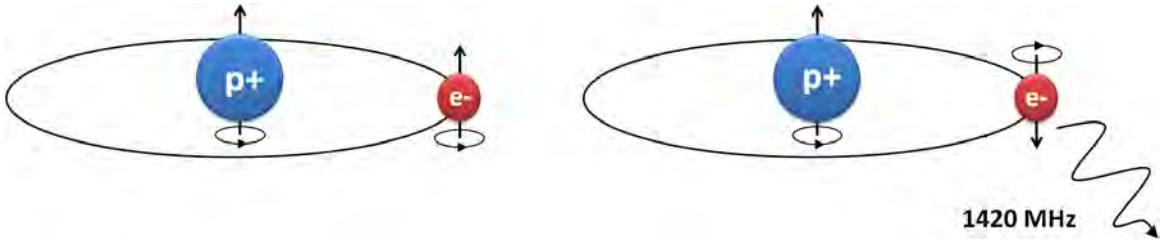


Figure 2.6: Diagram demonstrating the hyperfine ground state transition which occurs in a neutral hydrogen atom and the EM emission produces at approximately 1420 MHz [9].

(Search for Extraterrestrial Intelligence) project. The wavelength of the HI hyperfine transition was used as a unit of length for describing the height of a human male and female figure. This was included as a description of our species to potential extraterrestrial recipients.

## 2.2 Radio Astronomy

Radio astronomy is a category of observational astronomy which observes celestial objects in the radio band of the electromagnetic spectrum. In 1931, the first astronomical radio source was discovered by Karl Jansky, a Bell Telephone Laboratories engineer. While investigating static interference on short wave voice communications [41], he discovered that the source of the radio interference was not of terrestrial origin. It was in fact emitted from the central region of our Milky Way. Later it was discovered that Jansky's radio detection was produced by a complex source labelled Sagittarius A, which comprises of a supernova remnant and spiral structure as well as Sagittarius A*, the super massive black hole at the centre of our galaxy. This source is the brightest radio source in the sky.

Observational astronomy performed at radio frequencies has several key benefits over observations performed in other bands of the electromagnetic spectrum [42]. One of the primary advantages of observing in the radio frequency range is that the Earth's atmosphere is transparent to radio signals. With the exception of the radio and visible bands, the Earth's atmosphere is opaque to electromagnetic radiation to various degrees. Refer to Figure 2.7 for a graphical illustration of the opacity of Earth's atmosphere to various wavelengths. Due to this phenomenon, expensive space based telescopes are needed to observe other electromagnetic frequencies, such as infra-red, UV, X-Ray and Gamma rays.

While visible light can also penetrate the atmosphere, it undergoes atmospheric diffraction caused by changing air density. Furthermore electromagnetic radiation at visible light frequencies does not penetrate cosmic dust and therefore cannot be used for observations through dust clouds. This allows radio astronomy to play an important role in probing distant objects in the Universe.

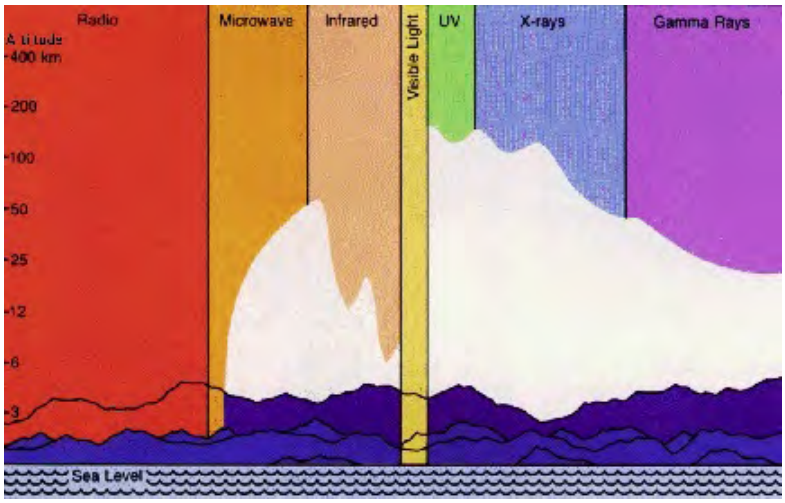


Figure 2.7: Diagram illustrating the opacity of Earth's atmosphere to various frequencies of electromagnetic radiation [10]. Only radio and visible bands are unobstructed.

The first radio antennae (referred to as radio telescopes) were rudimentary detectors which did not provide high angular resolution resulting in poor resolving power and the inability to isolate and characterise radio sources. Figure 2.8 shows Karl Jansky standing in a field with his early radio antennae array, which he used to detect the radio source and the centre of our galaxy. Modern radio telescopes are far more technologically sophisticated. A technique called astronomical interferometry is used, whereby multiple receivers are configured in an array. The signals received by each of the detectors in the array are superimposed by means of interferometry to produce a higher sensitivity or better resolution than possible using only one detector [43]. The effect is an aperture comparable to that of a much larger single receiver which encompasses all of the interferometer elements. This approach can be used to perform high resolution imaging as well as Very Long Baseline Interferometry (VLBI) which uses radio telescopes located thousands of kilometres apart.

Today radio telescopes are being used to study HI in nearby galaxies by detecting the HI 21cm emission line. In addition, experimental radio interferometers are being developed to perform new kinds of observations. The Precision Array for Probing the Epoch of Reionization (PAPER) is a one such project located in the Karoo [44]. Its goal is to observe the re-ionisation of HI in the early Universe, at low frequencies of between 100

and 200 MHz. The Murchison Widefield Array (MWA) is another such project based in Australia which also aims to explore HI in the early Universe [45]. These projects function as path finders, or precursors, for the SKA Low-Frequency Aperture Array (LFAA) [46].

Another important task for radio telescopes is the search for and timing of pulsars; highly energetic objects typically rotating at a fixed frequency and emitting charged particles as jets from their poles [47]. Pulsars can be used as accurate clocks to measure the as-of-yet undetected presence of gravity waves.

The SKA will expand the forefront of observational cosmology by surveying enormous volumes of the Universe. New investigations into Baryon Acoustic Oscillations (BAO) and cluster formation will be possible. In addition, new observational techniques will be possible, such as detections using weak gravitational lensing of radio signals [48].

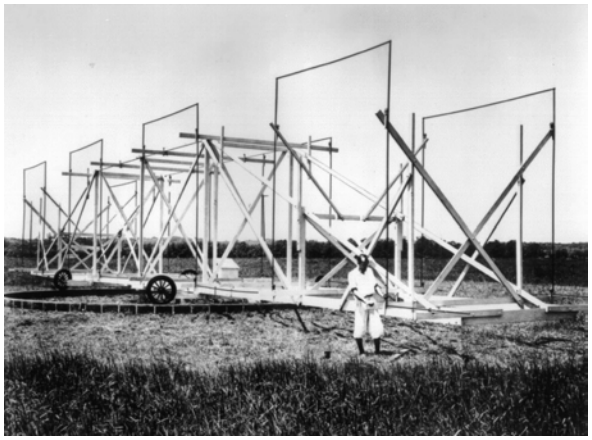

Figure 2.8: Karl Jansky's and his radio antennae array used to detect radio emissions from Sagttarius A in the 1930's [11].

Figure 2.9: The first of 64 13.5 meter diameter dishes of the MeerKAT SKA precursor radio interferometer, completed in early 2014 [12]

### 2.2.1 SKA and the MeerKAT Radio Telescope

The Square Kilometre Array (SKA) is an international scientific project with collaboration from ten participating countries. The project aims to answer fundamental questions and challenge existing theories about the evolution and current state of the Universe. The SKA will be a large radio interferometer comprising thousands of radio telescopes, with a dense central core of dishes and remote stations extending to a distance for 3000 kilometres providing a massive baseline. As the name suggests, it will have a total collecting area of 1,000,000 $m^2$ [49]. This collection of large dishes will allow the SKA telescope to survey the sky at 50 times greater sensitivity than any existing observatory .

In May 2012, South Africa won the bid to host the primary SKA facility situated in the Karoo region. While design specifications have yet to be finalised, South Africa will host a large portion of the dish array as well as the dense aperture array [50]. Australia was also selected to host several arrays, including the Australian SKA Pathfinder (ASKAP). The project has a budget of R 21 billion and is split into three Phases. Phase 1 includes the SKA's central dense core of 256 dishes, is planned for completion by 2023, with Phase 2 extending to at least 2030. Currently SKA Phase 3 remains largely undefined and uncertain.

The MeerKAT Precursor telescope is a technology demonstrator currently being built at the SKA site in the Northern Cape of South Africa. MeerKAT is a purely South Africa funded project, which had secured its funding before South Africa was awarded the SKA bid. Nevertheless, MeerKAT and SKA Phase 1 will share the same site location. The 64 dish array of 13.5 meter dishes currently being constructed is intended to come online by 2017 [51]. Figure 2.9 shows a photograph of the first of 64 dishes which will make up MeerKAT, after being installed in early 2014. MeerKAT will be used to conduct experiments after its commissioning and calibration process. While this 64 dish interferometer will be relatively tiny compared to the completed 2500 dish SKA telescope, it will still be one of the most powerful radio telescope ever built once complete in 2017 [52]. MeerKAT was developed to focus on L-Band (1.0 - 1.75 GHz) radio frequencies and until SKA Phase 1 is complete, it will be the most sensitive L-Band detector in the world. This provides some perspective of the scale of the SKA project as a whole. As they share the same site, once SKA Phase 1 is complete, MeerKAT will be fully integrated into the central dense core region of the larger interferometer.

The MeerKAT telescope itself has a 7 dish precursor array. The Karoo Array Telescope (KAT-7), which was developed as part of an engineering test bed; has been collecting data since 2012 [53]. The first papers on KAT-7 results were published in 2013 [54].

### 2.2.2 The LADUMA Deep Sky Survey

Looking at the Distant Universe with the MeerKAT Array (LADUMA) is an ultra deep HI survey which has been awarded 5000 observation hours on the MeerKAT telescope. This survey, along with the Radio Pulsar Timing survey, are two science projects which form the Priority Group 1 science workload for MeerKAT when it becomes operational [55]. The LADUMA survey will use all of its available observation time looking at a single narrow patch of sky, in order to investigate HI mass distribution at a redshift of upto

z=0.58 (approximately 2100 Mpc or $7\times10^9$ light-years away). At the furthest extent of the LADUMA survey, MeerKAT would see the Universe as it was 5.7 billion years ago, in a different state to how it exists today.

The observation field of the survey will be centred at the Extended Chandra Deep Field-South (E-CDF-S) region of the southern hemisphere sky. Significant data already exists for this region from surveys conducted at other wavelengths. This is essential, as LADUMA aims to perform composite imaging using a stacking method with detections obtained by these other surveys [13]. Using this approach, HI will be detectable at a redshift of up to z=1.2.

LADUMA will provide new insights into galaxy formation in an evolving Universe by producing HI density functions over a wide redshift range. These findings will also be used to broadly characterise galaxies as well as to probe empirical models such as the Tully-Fisher mass-luminosity relation [17].

The LUDUMA survey was proposed to investigate several key questions in cosmology. These questions were summarised in the original proposal for the survey [18]. The first research aim is to investigate the relationship between the cold gas mass of galaxies and the size of the dark matter over densities (halos) in which they are found. Secondly, the survey will help investigate the relationship between the content of HI within galaxies and their corresponding stellar mass. The survey will also probe the relationship between HI emission line profiles and neutral gas absorption lines. Finally, the survey will provide insights into the effect's of the local environment on the cold gas content of galaxies.

In addition to these large scale investigations, LADUMA will also provide the opportunity to conduct studies on individually detected and well resolved galaxies.

The HI gas being observed by the MeerKAT telescope will have a wavelength of 21cm (frequency of 1.42 GHz) at minimal distances. However, at the maximum range of the initial survey stage (z=0.58), the HI emission lines will be frequency shifted down to 900 MHz due to cosmological redshift [56]. This frequency range over which MeerKAT operates has implications for the survey. The opening angle selected for the LADUMA survey is 0.948 deg, giving it a field-of-view (FoV) of 0.9 $deg^2$, at the upper 1.4 GHz detection frequency [57]. However the effective beam-width of a radio telescope increases as an inverse function of frequency. Therefore when observing HI at the maximum redshift of z=1.2, the frequency of HI spectra is shifted to 636 MHz, resulting in a telescope FoV of 4.5 $deg^2$. This expansion, or widening, of the telescope's FoV creates a geometric volume similar to that of a Vuvuzela; the plastic horn blown by soccer fans. This lead to the

South African acronym LADUMA, meaning 'goal' in isiZulu. Figure 2.10 describes the geometry of sky surveyed by MeerKAT, showing the aperture of the LADUMA survey.

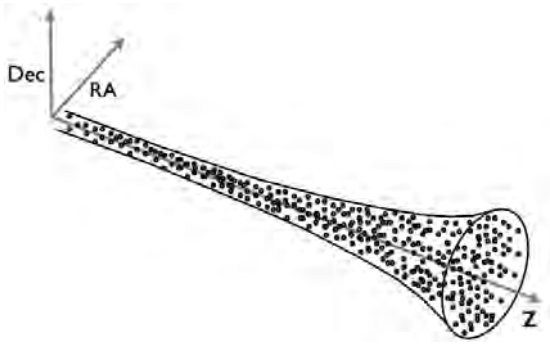


Figure 2.10: Diagram describing how the LADUMA survey's effective aperture increases with redshift (z) [13].

## 2.3 High Performance Computing

High Performance Computing (HPC) is a rapidly evolving field within computer science, focused on developing specialised computing systems in order to solve large computational problems. HPC facilities provide the infrastructure to perform the immense data processing required for ambitious scientific projects, such as the Large Hadron Collider (LHC) and SKA.

### 2.3.1 A Brief History of HPC

Originally HPC system designers focused on developing powerful monolithic architecture systems, or 'supercomputers'. However, it was soon realised that the physical limitations inherent with the microprocessor manufacturing process restricted the development rate of these individual complex machines, thus another approach was required. Multi-processor systems were developed in order to distribute the workload over multiple execution threads. These machines, called Symmetric Multiprocessors (SMP), solved the operating frequency performance limitation. They were nonetheless still bottlenecked by the supporting memory subsystem which operated at considerably slower speeds. This limited the scalability of such a design [58]. Figure 2.11 illustrates the shared memory structure of an SMP architecture computer and demonstrates the inherent memory bottleneck of such a system.

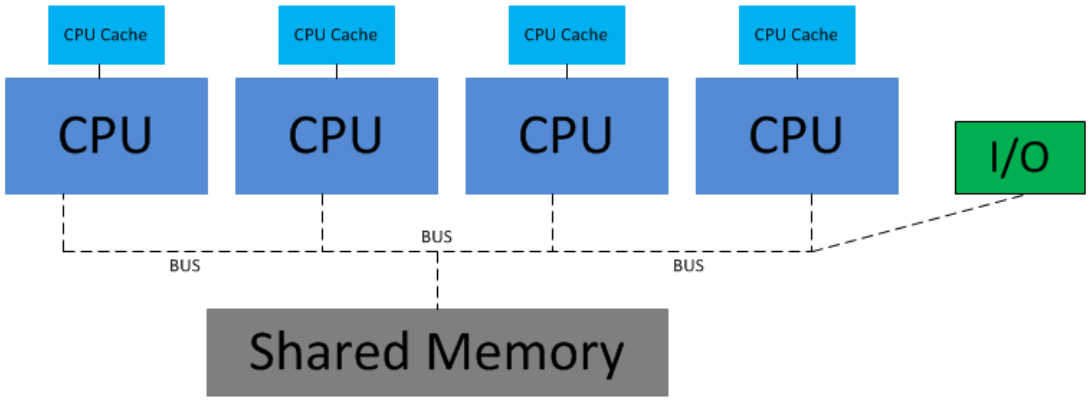


Figure 2.11: Diagram describing an SMP architecture computer. Each processing unit (CPU) has access to a private local cache as well as global shared memory. The available bandwidth of the shared memory imposes a computational bottleneck in such an architecture.

Distributed computing was developed to tackle larger computational tasks. While still remaining within the performance limits imposed by silicon microprocessors and a tiered memory hierarchy. In recent years, the term high performance computing has become synonymous with computer clusters. A computer cluster (alternatively called cluster computer) consists of a group of standard SMP computers (referred to as hosts or nodes) which communicate over a network in order to distribute a workload to solve large computational problems. Since the invention of the classical Beowulf cluster architecture in 1994, the distributed computing approach has rapidly gained popularity to replace the existing centralised supercomputing systems. Computer clustering has become the preferred approach for designing high performance computing systems for many applications. The primary reason for this is the flexibility, expandability and cost effectiveness inherent in a cluster's modular and standardised design [59]. Given a sufficiently fast network, this design has yet to reach the limits of its scalability. As of November 2012, clusters constitute 82% of the top 500 most powerful computers in the world [60].

A hybrid computing approach has become standard within HPC in recent years [61]. This hybrid model typical consists of multiple SMP architecture computers connected together via a network to form a greater distributed memory computer architecture. This results in a NonUniform Memory Access (NUMA) architecture, where the access time to a memory address is dependent on its location. The computing architecture can be expanded further to include non-x86 based task specific accelerators such as GPGPUs (General Purpose Graphics Processing Units) or the Intel Xeon Phi coprocessor [62].

Other approaches do exist for solving HPC problems, however these solutions are typically

suited for very specific problems. One such example is Field Programmable Gate Arrays (FPGAs) which can provide significant performance improvements over other approaches but are often limited in the scope of their application and require long development processes using non-commodity hardware.

### 2.3.2 The Centre for High Performance Computing

The Centre for High Performance Computing (CHPC) is an initiative of the Meraka Institute, which forms part of the ICT operating unit of the Council for Scientific and Industrial Research (CSIR) [63]. The CHPC aims to accelerate Africa's socio-economic upliftment through the use of world class Cyber-infrastructure. The Advanced Computer Engineering (ACE) Lab within the CHPC is responsible for the Research and Development of new HPC technologies, tools and methodologies. The ACE Lab's research cluster was utilised for much of the computationally intensive processing performed in this research.

The ACE Lab HPC cluster consists of a head node, storage node and six computation nodes connected via a high performance network interconnect. The nodes contain Intel IvyBridge architecture CPUs. Within each computation node, 2 Intel E5-2690v2 processors operate at up to 3.6 GHz, totalling 20 physical cores (hyper-threading is disabled as is standard practise in HPC). Each node also contains 128GB of high speed DDR3-1866 MHz memory and an Infiniband Fourteen Data Rate (FDR) 56 Gbps Host Channel Adaptor (HCA). This interface is primarily used for high performance runtime communication between nodes, typically via Message Passing Interface (MPI), as well as for accessing the shared file system hosted on the storage node. In addition, each host on the network uses a Gigabit Ethernet network interface for out-of-band management and monitoring. The cluster by default hosts a virtual environment of KVM virtual machines, managed by the OpenNebula Cloud Management Platform (CMP). This virtualised environment makes scheduling jobs and managing resources for multiple users more efficient. However there is a performance overhead associated with virtualisation [64], therefore both virtual machines and the native hardware environments were considered during the evaluation of the GADGET-3 code in Chapter 4.

## 2.4 The GADGET Cosmological Simulation

### 2.4.1 A Background to GADGET

GADGET (GAlaxies with Dark matter and Gas intEracT) is an open source cosmological scientific code distributed under the GNU general public license [65]. It was developed to perform collisionless particle simulations for accurately modelling the large scale cosmic structure formation of dark matter. The code relies on well defined mathematical models for gravitational interactions in large systems to replicate the evolution of the Universe. Using the dark matter solver, the GADGET code can model the cold dark matter distribution within the given parameters thus accounting for 80% of the total matter. Furthermore the code supports a Smoothed Particle Hydrodynamics (SPH) solver, used for modelling the complex interaction and collision of ordinary matter (baryons). Typically these interactions occur when baryons fall into gravity wells created by collapsing dark matter. This interacting matter, in the form of ionised gas, coalesces into structures and eventually form galaxies. By combining these two solvers, GADGET can effectively model cosmological evolution at a computationally limited resolution [5].

The code was written in ANSI C and intended for use of distributed memory computers, using the MPI standard for communication between threads. It was developed by Dr. Volker Springel from the Max-Planck-Institute for Astrophysics over a period of several years. The first version, GADGET 1.0, was released in 2000 followed by a complete rewrite of most algorithms as GADGET-2 in 2005. To date no official release exists of GADGET-3; however several individuals and research groups are working on numerous versions of the code. This research relied on a beta edition of GADGET-3, version 3.25.

### 2.4.2 The N-GenIC tool

The GADGET simulation works on a set of Initial Conditions (ICs) which contain a large number of particles classified into gas (baryons) and cold dark matter. The ICs are vitally important to the accuracy of the simulation, as they represent the Universe at a young age when it had an almost uniform matter density distribution. The simulation begin by evaluating each of the particles and applying forces to them, calculated from the surrounding particles [16]. The particles are contained within a three dimensional cube which represents the simulation space.

The series of renders in Figure 2.12 illustrate the passage of time within the simulation, from an early uniform Universe to the present day. A isometric perspective was captured using the GADGET visualisation tool - Gadget File Viewer [66]. Table 2.1 provides additional details for images presented.

Table 2.1 describes the redshift, age of the Universe and light travel distance of each of the snapshots presented in Figure 2.12. As the speed of light is constant, more distant objects are seen as they were at earlier times, since the energy they emitted can only be received after it has travelled through space.

Table 2.1: GADGET-3 simulation snapshot details for Figure 2.12.

| **Snapshot** | **Redshift** | **Time since Big Bang (Billion years)** | **Light travel distance (Mpc)** |
|---|---|---|---|
| A | 30.0 | 0.09 | 11 337 |
| B | 6.75 | 0.78 | 8 536 |
| C | 2.25 | 2.86 | 5 520 |
| D | 1.25 | 4.87 | 3 870 |
| E | 0.51 | 8.35 | 1 921 |
| F | 0.0 | 13.4 | 0 |

The initial conditions required by GADGET-3 are typically generated with the software package N-GenIC (alternatively using PgenIC or MUSIC) [65]. This tool considers perturbation theory to replicate the densities fluctuations of the early Universe, discussed in Section 2.1.1. The GADGET software then applies its solver models to this set of particles, manipulating them over time. Typically an even number of gas and dark matter particles are included in this set of Initial Conditions. The ICs used for this research were generated using $256^3$ dark matter and $256^3$ gas particles (16.7 million of each particle type), at redshift z=30 (100 million years after the Big Bang). The number of particles defined in the ICs, in relation to the simulation volume, determines the resolution and the computational workload of the subsequent simulation.

### 2.4.3 GADGET Code Mechanics

As the GADGET-3 code is typically used to simulate large numbers of particles (millions or even billions), the computational requirements are immense. In order to meet the computational throughput and memory footprint requirements for the code, GADGET-3 runs on a distributed memory HPC cluster computer. The code implements a domain

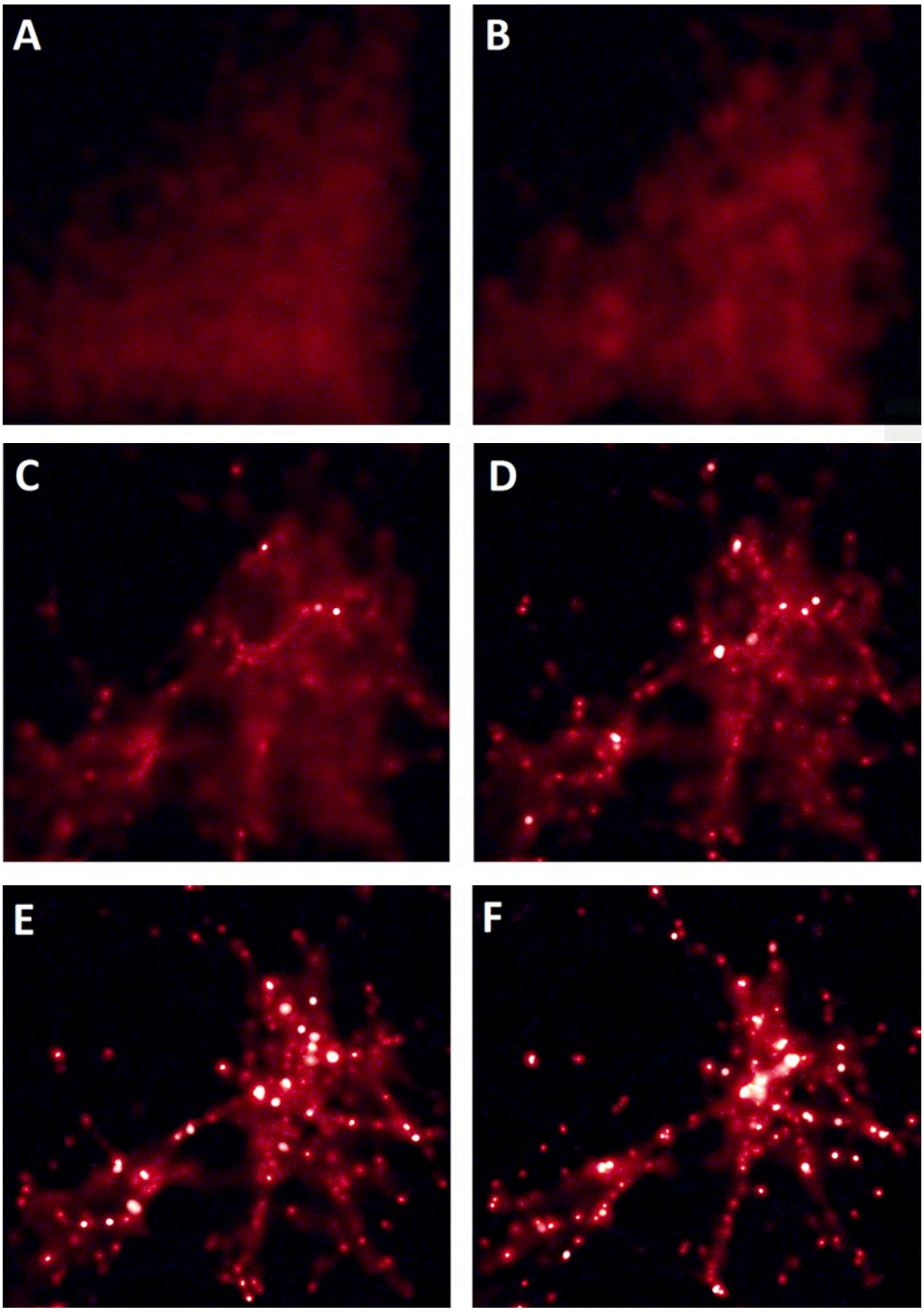


Figure 2.12: A set of visualisations produced from GADGET-3 simulation snapshots. The images show the distribution of gas (red) and star (white) particles, with colour denoting concentration. The series of images show the formation of structure over time beginning at early time (snapshot A) and evolving to the present day (snapshot F). Snapshot A depicts the Universe as it was 97 million years after the Big Bang (redshift z=30) while snapshot represents the present day; 13.5 billion years after the Big Bang (redshift z=0). Refere to Table 2.1 for further snapshot details.

decomposition technique to partition the simulation volume and distribute it to individual threads. As stated previously, GADGET-3 uses the MPI protocol for communication between threads on both local and remote hosts. Even when using a large HPC cluster with hundreds or thousands of CPU cores, GADGET-3 simulations can run for weeks or months depending on the number of particles being simulated.

In order to decrease the solver's inter-thread communication overhead and improve parallelisation, an approximation method is implemented. Gravitational forces between particles are solved using two methods. Near forces are computed using the TreePM method which groups forces by proximity using a Friends-of-Friends algorithm. The far field forces are calculated using a Fourier technique to approximate forces. In general, the code aims to calculate forces for each particle with sufficient accuracy based on their impact. This is determined by the inverse power law which defines the strength of gravitational interaction. Following this approach, near and far forces are integrated over different time intervals to reduce the total number of numeral operations required [16]. This multi-resolution spatial and time based force approximation technique also serves to lessen the computational requirements of the code and reduce the runtime from $O(n^2)$ to a runtime that resembles O(n.log(n))).

All the particles in the simulation are contained within a three dimensional cube, the size of this cube is specified in the global parameter file in units of Kpcs. Gas particles and dark matter particles are calculated simultaneously with forces interacting between the two particle types. Given the right conditions (namely temperature and pressure), a gas particle can split to form a star particle. Star particles typically form in high density regions and develop into galaxies over time. While the name suggests that these particles represent individual stars, they are in fact of significantly greater scale. The resolution of the simulation defines the mass of these particles. In simulations, star particles with a mass of $65\times10^6$ Solar Masses ( $M_\odot$), or 65 million times the mass of our Sun were used. Therefore a typical galaxy would have hundreds or thousands of these particles tightly bound gravitationally. Figure 2.13 presents a visualisation of a small region within the GADGET-3 simulation used in this research. The three particle types; dark matter, gas and stars, have been rendered separately to demonstrate how they would interact.

When modelling large structure formation, on the order of Mpcs in scale, the region inside the simulation volume cannot be assumed to be a closed system, as this does not realistically represent the Universe. In order to preserve realistic gravitational force approximation within the cube, in a Universe assumed to be homogeneous, it is necessary to enforce Periodic Boundary Conditions. Periodic boundaries creates an infinitely large simulation space, by using the Ewald summation technique for computing long-range

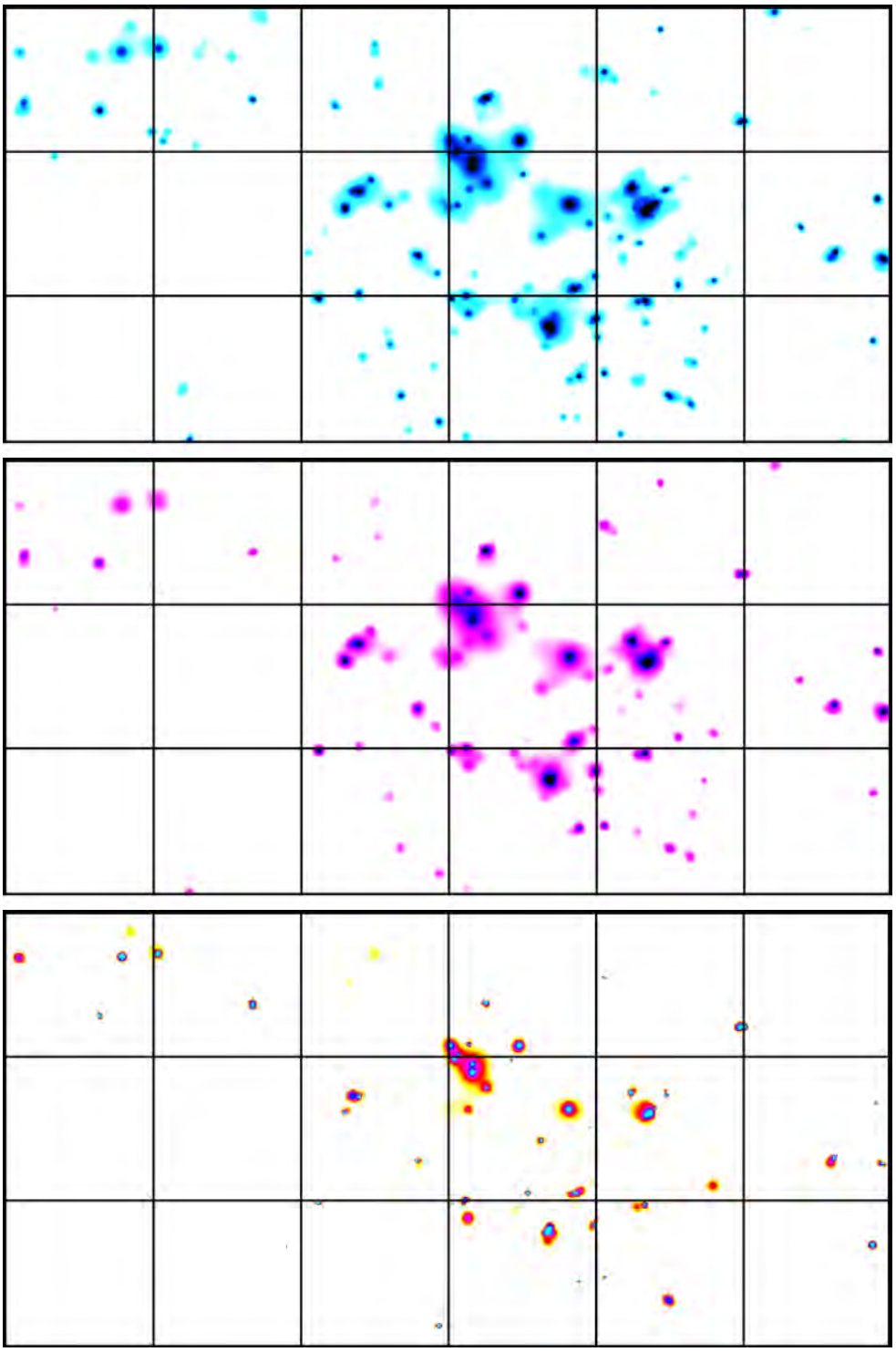

Figure 2.13: A set of perspective 3D visualisations of GADGET-3 simulation snapshots. A subsection of the z=0 snapshot containing a large cluster of galaxies has been decomposed into its cold dark matter (top), gas (middle) and star (bottom) particle components. Colour denotes concentration; in the star particle case blue denotes a dense galactic core while red denotes a galaxy's halo of stars. The large plumes of ejected hot gas, produced from galaxy outflows, can be seen as defuse halos. The square overlay is approximately 500 kpc on a side.

force interactions [5]. Particles are 'wrapped around' the boundary of the simulation volume. Thus from a perspective within the simulation space one see repeating copies of the cube's contents in all directions. However this approach does have implications for the validity of results however, therefore a volume as large as possible should be chosen to minimise this effect.

In addition to the gravitational forces, GADGET-3 also models the interactions of gas particles in close proximity. As these particles are not collisionless like cold dark matter, the interaction becomes considerably more complex. Enabling the SPH solver to model gas particles results in simulation runtime that are considerably longer than a CDM only approach. Due to this, several other methods have been developed for simulating large structure formation. A Semi-Analytical Model (SAM) ignores the gas content entirely during the simulation and attempts to add in galaxies using a post processing method [67]. AMIGA (Analytical Model for IGM and GAlaxy evolution) [68] and GECO (Galaxy Evolution COde) [69] are two examples of current SAM implementations. This approach to simulation is often used to understand the processes which contribute to galaxy formation, and are often preferred as they are computationally less demanding. However a SAM model cannot replicate the fidelity and resolution of a true analytical method.

The research team lead by Prof. Davé has improved several of GADGET-3' gas models to better represent physical processes [70]. A feedback system which exists between star formation and the ionisation of neutral gas has been developed. Using available data, the creation of star mass can result in outflows of hot gas. These additions to GADGET are important for more accurately modelling the neutral gas content of galaxies and relating it to the rate of star formation and metallicity.

## 2.5 Existing Research

Several existing research papers have been presented on the relevant topics relating to this research. A selection of these papers have be reviewed, thus an analysis of results and comparison of research methodologies is presented in this section. A knowledge base of existing research is required in order to assess the reliability and validity of the methods and results presented in this dissertation.

### 2.5.1 GADGET-3 HI Post-Processing

The main deliverable of this research project define in the Terms of Reference is producing software to create an artificial radio telescope observation referred to as a 'lightcone'. This lightcone was constructed from cosmological simulation data, generated using the GADGET-3 code. However GADGET-3 does not provide a sufficient level of detail for all criteria concerned in this research. As previously stated GADGET-3 operates on gas and dark matter particles, yet these gas particles are not classified or designated into any particular form. In the radio spectrum, the 21cm line is by far the most common emission from gas in galaxies and it is thus necessary the determine which gas is neutral and thus capable of producing 21cm emission. While GADGET-3 does not provide this distinction by default, research has been done on post-processing GADGET-3 snapshots to extract HI information using a semi-analytical approach.

The paper "The neutral hydrogen content of galaxies in cosmological hydrodynamic simulations" by Davé *et al.* provided a method for extracting HI fraction from gas particles [70]. The method they developed involves using Spline Kernel Interpolative Denmax (SKID) to identify and categorise groups of close proximity star and gas particles which are tightly bound by gravity. This method produced a catalogue of galaxies that meet the detection threshold. In order to extract the HI fraction from these gas particles, several factors needed to be calculated, such as states in which the gas particle exists. Firstly, the self-shielded fraction of gas was calculated, in order to separate the primordial neutral gas at the centre of gas structures from the gas in the outer regions. The latter is exposed to metagalatic ionising flux and thus is in an ionised state. Secondly, the ratio of atomic hydrogen to molecular hydrogen was calculated. The mathematical method followed can be summarised as follows.

A hydrogen ionisation balancing formula was used to calculate the neutral hydrogen fraction of each gas particle:

$$f_{HI} = \frac{2C + 1 - \sqrt{(2C+1)^2 - 4C^2}}{2C} \tag{2.2}$$

with

$$C = \frac{n\beta(T)}{\Gamma_{HI}}, \tag{2.3}$$

where $n$ is the number density of hydrogen, $T$ is the gas particle temperature (inferred from its internal energy), $\Gamma_{\rm HI}$ is the HI photoionisation rate and $\beta$ is the recombination rate giving by function:

$$\beta(T) = a\Big[\sqrt{(T/T_0)}(1+\sqrt{(T/T_0)})^{1-b}(1+\sqrt{(T/T_0)})^{1+b}\Big]^{-1}, \tag{2.4}$$

using fitting parameters of $a$=7.982×10$^{-11}$ cm$^3$s$^{-1}$, $b$ = 0.7480, T$_0$ = 3.148 K and T$_1$ = 7.036×10$^5$ K specific to HI.

At this point, the shelf-shielding ratio is known. An SPH spline kernel is then used to calculate the fraction of this hydrogen which is neutral. The radial column density profile is calculated as follows:

$$N_{HI}(r) = \frac{0.76 f_{HI}\rho_g}{m_p}\int_r^h W(r')dr', \tag{2.5}$$

where $\rho$ is the SPH density of the gas particle, $m_p$ is the proton and $h$ is the particle's SPH smoothing length.

Finally the ratio of molecular to atomic hydrogen is calculated using the ISM pressure relation:

$$R_{mol} = (P/P_0)^{\alpha}, \tag{2.6}$$

where $P_0 = 1.7\times10^{-3}K$ and $\alpha = 0.8$.

This method gives the neutral hydrogen fraction of gas particles. For non star-forming particles, this required splitting the ionised and neutral component, assuming the fraction of molecular gas is 0. For a star-forming particle, an additional step was required to calculate the molecular fraction. Findings from this paper will be considered when evaluating the HI mass functions produced in the results chapter.

### 2.5.2 Existing Lightcone Construction Techniques

As stated in Section 2.4.3, the GADGET-3 code performs its simulations within a cube geometric volume in order to compute the periodic boundary conditions necessary for large structure formation. While this method is effective for modelling structure in a regular volume, it is not well suited to the irregular volume required for the current research. The lightcone being constructed is required to extend at least to a redshift of z=0.58. Via the $\Lambda CDM$ model to describe the expansion of the Universe, the physical distance to this redshift can be calculated. Using the standard cosmological parameters of $\Omega_m = 0.3$, $\Omega_\Lambda = 0.7$, $\Omega_r = 0$, $w_0 = -1.0$, $w_a = 0$ and $H_0 = 70km/s/Mpc$, the relation between redshift and physical distance can be produced. Figure 3.1 presents this relation, showing that a redshift of z=0.58 corresponds to a physical (referred to as co-moving) distance of $2144h^{-1}Mpc$. The $h^{-1}$ factor is a parameter used to reflect the uncertainty of the Hubble constant which describes the expansion of the Universe. $h = H/(100km/s/Mpc) = 0.7$. Removing the $h^{-1}$ factor gives a distance of $1501Mpc$. It is infeasible to generate a GADGET-3 simulation volume with a side length of $1501Mpc$, particularly when the computationally complex SPH modelling of gas is being performed. There are two proposed solutions to this problem.

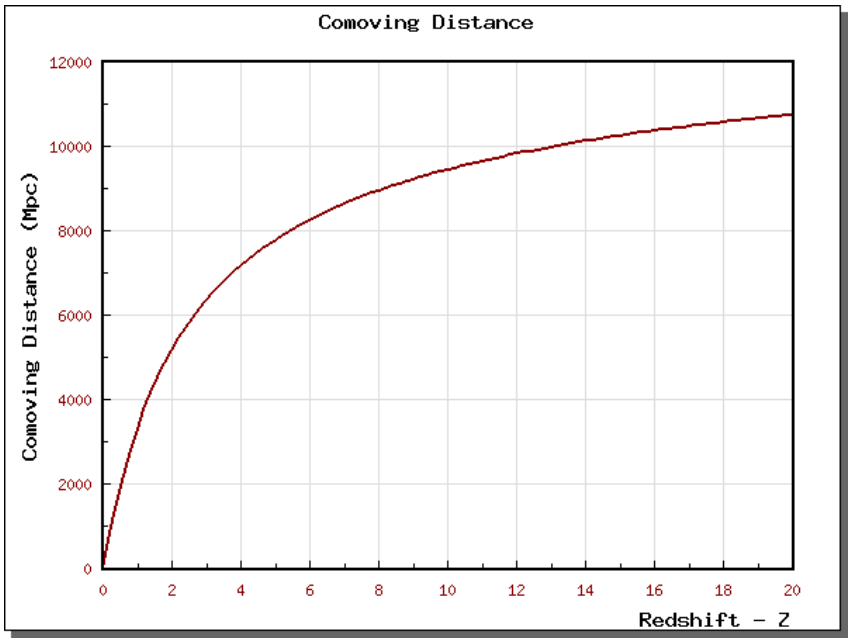


Figure 2.14: A graph depicting the relationship between redshift and co-moving distance in a $\Lambda CDM$ Universe.

In their 2010 paper, "Embedding Realistic Surveys in Simulations though Volume Remapping" Carlson & White presented a method for producing the irregular volume required for a lightcone [14]. The method was developed specifically for better aligning the geometry of

cosmological simulations and real-world observational programs. It applied a mathematical volumetric remapping onto the simulation volume, in order to remap it from a cube into an irregular cuboid volume. Periodic boundary conditions were required, to allow the simulation cube to model infinite space. This technique is one-to-one (no duplication of data), volume preserving and keeps intact smaller scale structure as well as requiring relatively low computational overhead. The original simulation cube is normalised to a unit cube and tiled in all directions, with each point conditioning to a shifted copy of the original. The generalised 3D remapping is performed by applying integer shears to the cube. Thus a parallelepiped is produced with integer vectors satisfying:

$$det \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{pmatrix} = 1 \tag{2.7}$$

The parallelepiped is squared up into a cuboid after applying two final shears, by choosing coefficients $\alpha$, $\beta$ and $\gamma$ such that

$$\begin{aligned} e_1 &= u_1, \\ e_2 &= u_2 + \alpha u_1, \\ e_3 &= u_3 + \beta u_1 + \gamma u_2, \end{aligned} \tag{2.8}$$

are mutually orthogonal. This corresponds to a remapping of the original unit cube into a cuboid with side lengths $L_i = |e_i|$.

There are however limitations to this approach. Firstly, due to the remapping processing, large scale structure within the simulation space may not be preserved. This limits the usefulness of this method for performing statistical analysis. Secondly, the volume of the remapping cuboid is bound to the volume of the original simulation volume. Thus in order to produce a larger lightcone, a larger simulation will need to be run. In addition, if the target geometry is too thinly distributed, the resulting remapping may contain irregular correlations, due to far apart points being mapped closed together. Finally, because only a single simulation output is used (in this case a GADGET-3 snapshot), the apparent age of the objects being observed does not decrease as it would with sufficiently large surveys.

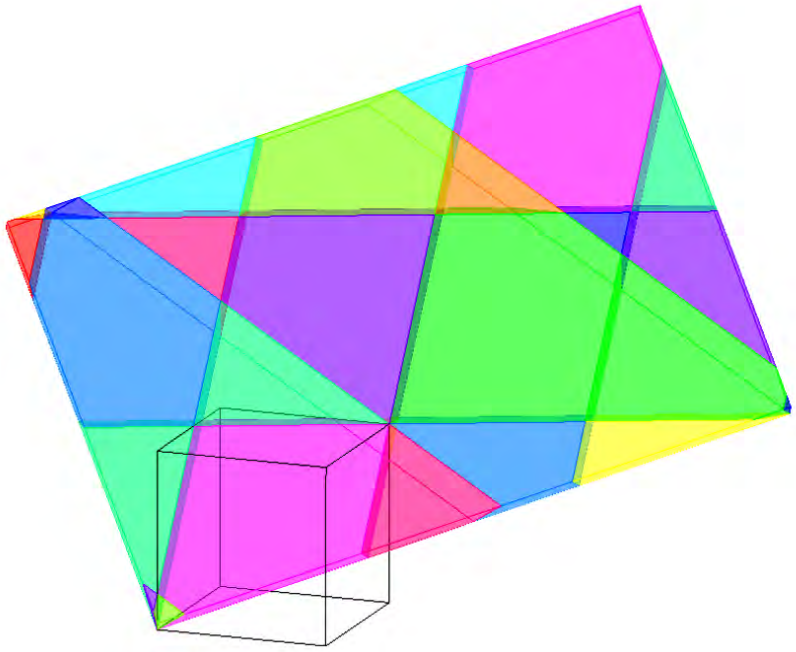

Figure 2.15: An illustrative example of the remapping tool converting a unit cube into an irregular three dimensional cuboid [14].

Obreschkow *et al.* presented an alternate method in a 2009 paper "A Virtual Sky with Extragalactic HI and CO Lines for the SKA and ALMA" [15]. The paper presented a technique for estimating the detection of HI and CO from large radio telescopes, such as the SKA, LMT and ALMA interferometers, at large redshifts (z ¡ 10). The lightcone construction method presented was adopted from an earlier paper published by Blaizot *et al.* [2]. As with the Carlson & White approach, cosmological simulation data was used as the source for these predictions. However another method was used to produce the irregular volume required for modelling ultra deep HI surveys. The method implemented an additional level of processing (post processing) following the initial GADGET-3 simulation to produce the cosmological data. The simulation boxes are stacked end on end, with the most recent boxes placed near the observer and boxes with a higher simulation redshift placed further away. To avoid the inherent periodicity of repeating copies of effectively the same information, a random geometry symmetry operation is applied to each box before being stacked. Figure 3.3 describes this box stacking process graphically.

Once an appropriate number of boxes have been stacked to encompass a sufficiently deep volume, a conic volume can be defined by its opening angle via:

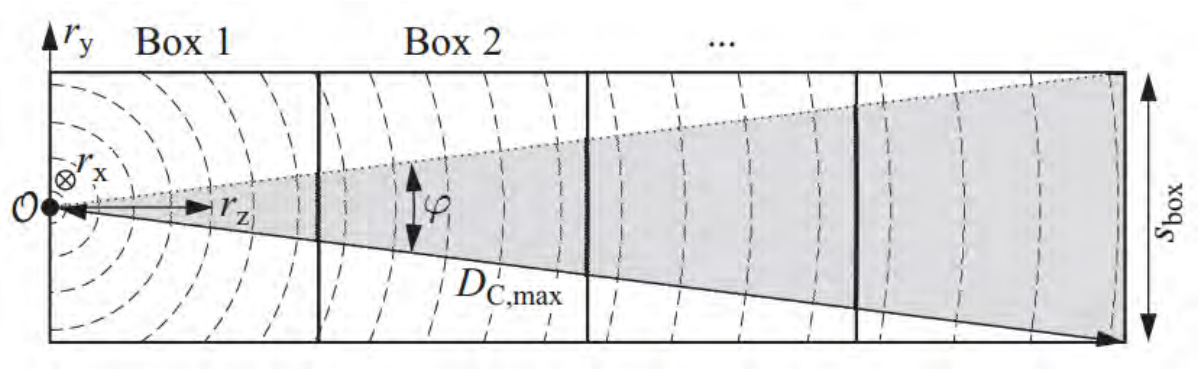


Figure 2.16: A diagram of how the box stacking approach is implemented in the Obreschkow *et al.* method of cone construction [15].

$$\varphi = 2arcsin\left(\frac{s_{box}}{2DC_{max}}\right), \tag{2.9}$$

where $s_{box}$ is the co-moving side length of the given simulation box and $DC_{max}$ maximal co-moving distance. A useful translation performed in this paper is the Euclidian projection formulae for mapping a Cartesian co-ordinate system onto a projected celestial sphere centred around the vernal point (right ascension = 0, declination = 0). The mapping is defined as:

$$RA = arctan(\frac{r_x}{r_z}) \tag{2.10}$$

$$DEC = arctan\left(\frac{r_y}{\sqrt{r_x^2 + r_z^2}}\right) \tag{2.11}$$

Finally, Duffy *et al.* present predictions for radio surveys using simulations in the paper "Predictions for ASKAP Neutral Hydrogen Surveys" published in 2012 [71]. Similar to the Obreschkow *et al.* paper, a SAM approach was used to produce a cosmological simulation through post processing the dark matter only Millennium simulation. They mapped stellar mass to dark matter halos through the conversion:

$$R = \left(\frac{M_{HI}}{M_{cold}}\right)_0 \left[\left(\frac{M_{cold}}{M_{stellar}}\right)^{-\alpha} + \left(\frac{M_{cold}}{M_{stellar}}\right)^{\beta}\right]^{-1}, \tag{2.12}$$

where $M_{HI}/M_{cold} = 0.41$, the faint end slope ($\alpha$) = 0.52 and the bright end slope ($\beta$) = 0.56.

Another useful equation defined in this paper is the estimation of thermal noise for a single dish, which acts as a lower sensitivity limit or source detection threshold. For a single beam, the thermal noise is given as:

$$\sigma_{noise} = \sqrt{2}\frac{kT_{sys}}{A_{eff}}\frac{1}{\sqrt{\Delta T \Delta\nu}}, \tag{2.13}$$

where $A_{eff} = \alpha_{eff} a\sqrt{N(N-1)}$ (N being number of dishes), $T_{sys}$ is the system temperature, $\Delta T$ observation time, and $\Delta\nu$ observation bandwidth.

Each of the methods reviewed in this section had strengths and weaknesses. The Carlson & White method did not replicate the source simulation data, resulting in a fixed 1:1 ratio of simulation to cone volume. Thus a limitation on the size of the lightcone was produced, caused by the computational complexity of the original simulation. The Obreschkow *et al.* method removed this fixed volume relationship by stacking, or tiling, the original simulation boxes. However to avoid a spurious periodicity caused by the replication of data, symmetry operations are applied to each box, this resulted in discontinuities at the boundaries of each box. In addition, this approach still requires that the box side length of the original simulation must to be larger than the opening diameter of the lightcone.

### 2.5.3 The Mass-Flux Relation

The last paper presented in this review focuses on another important process in producing an accurate lightcone; producing flux data. The paper "Probing dark energy with baryonic oscillations and future radio surveys of neutral hydrogen" by Abdalla and Rawlings, published in 2005 presented a method for estimating the received energy flux from a source. In addition a number of equations were presented for predicting the sensitivity of radio telescopes in order to predict the number of detections within a survey

[72].

$$M_{HI}(z) = \frac{16\pi}{3}\frac{m_H}{A_{12}hc}\frac{D_L^2(z)}{1+z}\int S_\nu dV, \tag{2.14}$$

where the integral is over V, line-of-sight width corresponding to projected circular velocity of the galaxy. This expression can be presented in a more useful form as:

$$\frac{M_{HI}(z)}{M_\odot} = \frac{0.235}{1+z}\frac{D_L^2(z)}{Mpc^2}\frac{S_\nu}{\nu Jy}\frac{V}{kms^{-1}}, \tag{2.15}$$

This relation of HI mass to received flux does not consider the HI self-absorption effect, where the 21 cm emission of one HI particle is reabsorbed by another HI particle in the low energy state. Therefore this relation may lead to an under estimate of HI flux, however the bias is expected to be small [73].

## 2.6 Conclusion

This literature review presented background information on a number of topics related to the theme of the current research.

A scientific context was provided with an overview of cosmology and galaxy evolution in Section 2.1. The Universe is continuously expanding while matter collapses into structures. Cosmologists can place tighter bounds on models which define how dark matter and ordinary matter interact in the presence of dark energy. This is investigated by better understanding how large cosmic structures form. An explanation of the hyperfine transition and the 21cm emission line was given to demonstrate how HI can be detected by radio telescopes. This allows HI to be used as a marker for the presence of galaxies on cosmological scales.

Information was also provided on the MeerKAT radio telescope and LADUMA ultra deep HI survey. This survey will likely be the deepest direct measurement of HI ever attempted, observing for 5000 hours to detect objects at a redshift of z=0.58. This survey is one of two Priority Group 1 surveys planned for MeerKAT. Therefore it is important that

LADUMA produce the expected results. Effective planning and preparation are key for conducting a successful survey. Part of this planning process requires a virtual field-of-view (or lightcone) of what the telescope would observe. This lightcone can be used for statistical measurements and estimating the number of detections for a given set of parameters.

In Section 2.3, an introduction to High Performance Computing (HPC) was presented to provide background to the current state of the industry. HPC systems are implemented as large distributed memory clusters, comprising many Commercial-off-the-shelf (COTS) SMP computers connected via a high speed network. This modular design allows for modular construction of HPC facilities and provides redundancy in the event of a hardware failure. However this distributing topology creates complications for software developers writing HPC codes. The inherent NonUniform Memory Access (NUMA) within a network of SMP machines necessitates the use of advanced inter-node communication techniques, typically via the standard called Message Passing Interface (MPI).

Following this broad background into HPC systems, an HPC code is introduced. GADGET-3 is a cosmological simulation code used for modelling structure formation and evolution on cosmological scales in the expanding Universe. This code uses MPI to communicate between a set of processor threads. Using the HPC cluster computer facilities available at the Centre for High Performance Computing (CHPC), a GADGET-3 simulation can be performed to produce accurate data for lightcone construction.

Finally a selection of journal articles provided insights into the methods used by researchers who conduct similar research. A post processing method was presented for estimating the HI component of simulation gas particles based on their temperature and pressure from the Davé *et al.* paper. Three approaches for producing virtual observation data from simulations were also summarised in order to highlight the strengths and weaknesses of each approach. In review, using a fixed relation of simulation volume to observation volume results is a computationally limited solution, while replicating simulation data results in spurious periodicity. A mass to flux relation was also presented from the Abdalla and Rawlings paper, which provides a semi-analytical method for estimating the 21cm flux received from the hyperfine transition which occurs in HI.

# 3 Research Methodology

This chapter will provide an overview of the methodology undertaken to produce accurate simulation data using GADGET-3 as well as the Python code developed to post-process this data into a lightcone. The plan of development will be discussed in Section 3.1 which broadly describes the course of action followed during this research. This will include the process of setting up the GADGET-3 simulation code used for generating cosmological data. Once installed, the performance of the code was evaluated to determine if any optimisation was possible. A brief introduction will be also given on various visualisation tools used for validating simulation results. Thereafter, the process followed to utilise the 3D remapping technique as well as the alternate stacking approach will be introduced in Section 2.5.2.

Section 3.2 builds upon the introduction to GADGET-3 provided in the literature review by presenting specific information on the software environment used for optimising the simulation code. This HPC code requires high performance hardware and correctly configured software in order to operate optimally. The evaluation process of an HPC code like GADGET-3 is complex because of the inter-dependent nature of various software components upon which it relies. A systematic approach was therefore selected to best isolate the effects of each optimisation step. Section 3.3 provides an overview of this evaluation, with the results presented in Chapter 5.

Finally, an in-depth description of the code written to post-process the GADGET-3 data will be provided in Section 3.4. This includes a motivation to support the post-processing method selected as well as the description differences between this code and existing techniques. Significant effort was placed on developing highly efficient Python code in order to reduce post-processing execution time. An in-depth report on this Python optimisation process will also be presented.

## 3.1 Plan of Development

The process followed for producing a lightcone in this research can be summarised as follows:

- Setup, investigate and configure the GADGET-3 simulation code.
- Investigate the lightcone construction methods introduced in Section 2.5.2.
- Develop initial Python code to produce a lightcone from GADGET-3 simulations.
- Optimise and improve the functionality of the code to improve its usefulness.

The first task in producing data for this research was installing the GADGET-3 scientific code. The first attempt of installing and running the code was conducted within a single Symmetric Multi-Processor (SMP) computer, to serve as a proof of concept. GADGET-3 was specifically developed for use on Linux computer clusters. However in an attempt to produce the simplest working example, a basic laptop computer operating Linux was used. The installation process required configuring relevant software dependencies and libraries, as well as configuring the various parameter and configuration files required by the code. A design feature of the GADGET-3 program includes the simulation parameters within the compiled binary (such as enabling periodic boundary conditions) while software environment configuration is handled via a parameter file. A set of input files (including the Initial Conditions, described in Section 2.4.2) are linked to the binary executable via the parameter file, located in the same directory. After the code was successfully compiled, a selection of initial test simulations were run to ensure the code configuration and software environment were operating correctly. The visualisation tool, GADGET File Viewer, was installed to render the snapshot output files generated by the simulation.

Figure 3.1 displays an example of a GADGET-3 snapshot, demonstrating the collision of two galaxies. This simulation contains a meagre 60,000 dark matter particles and no gas particles. As such the computationally expensive SPH model and the periodic boundary conditions were disabled. As a result, this simulation is completed within an hour despite using a workstation laptop computer. In addition to running and testing the code, several log files and execution monitoring outputs produced at runtime were also analysed.

In order to implement the volume remapping and box stacking post-processing techniques described in Section 2.5.2, a significantly larger simulation volume was required. The

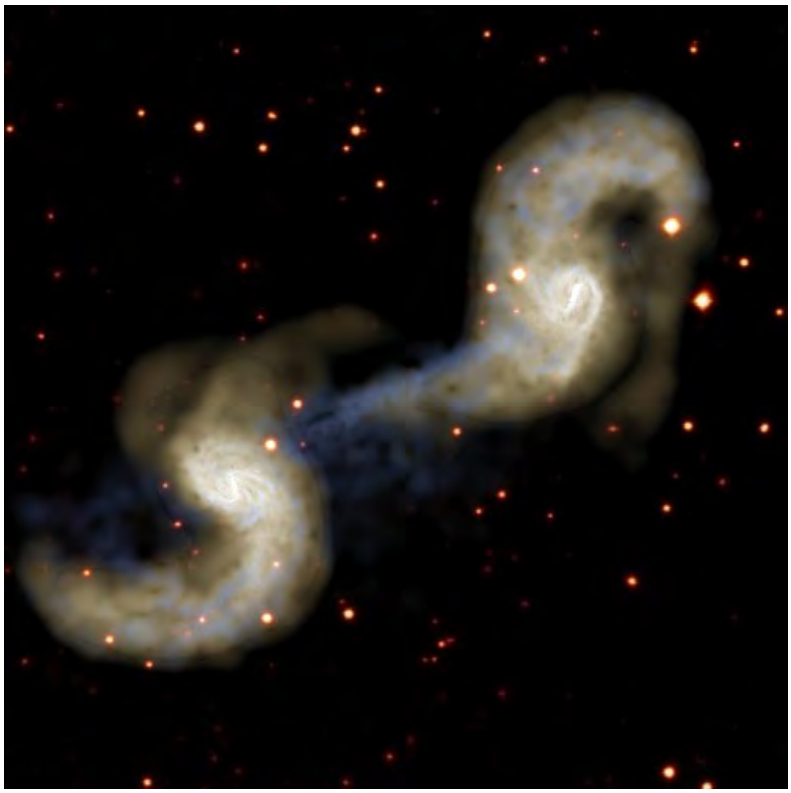

Figure 3.1: A render of two galaxies in mid-collision, simulated using GADGET-3. Only the baryonic matter component (stars and gas) is visualised [16].

GADGET-3 code was recompiled on the ACE Lab HPC cluster (detailed in Section 2.3.2) in order to utilise more CPU cores and memory capacity. By using a set of existing ICs, a 16 $Mpc^3$ volume was simulated, containing $128^3$ dark matter particles and $128^3$ gas particles (approximately 2 million of each particle type). By enabling periodic boundary conditions, gravitational effects were calculated in a 'wrap-around' manner, so as to realistically replicate gravitational forces interacting on a cosmological scale. Figure 3.2 illustrates the results of this simulation, by rendering the final snapshot (produced at present day, or redshift z=0). A composite image is created by overlaying each of the particle types: dark matter, gas and stars. Gadget File Viewer's particle smoothing feature was enabled in order to reduce noise and emphasise the larger structures.

Visualising a simulation's output can often be a useful 'by eye' verification to ensure expected results have been produced.

Following this initial investigation of GADGET-3, focus was placed on producing a small scale HI lightcone proof of concept. The Obreschkow box stacking and Carlson & White volume remapping techniques were presented in Section 2.5.2. An investigation was conducted in order to determine which of these methods would best suit the requirements for producing a LADUMA lightcone. The cuboid remapping tool, developed by Carlson

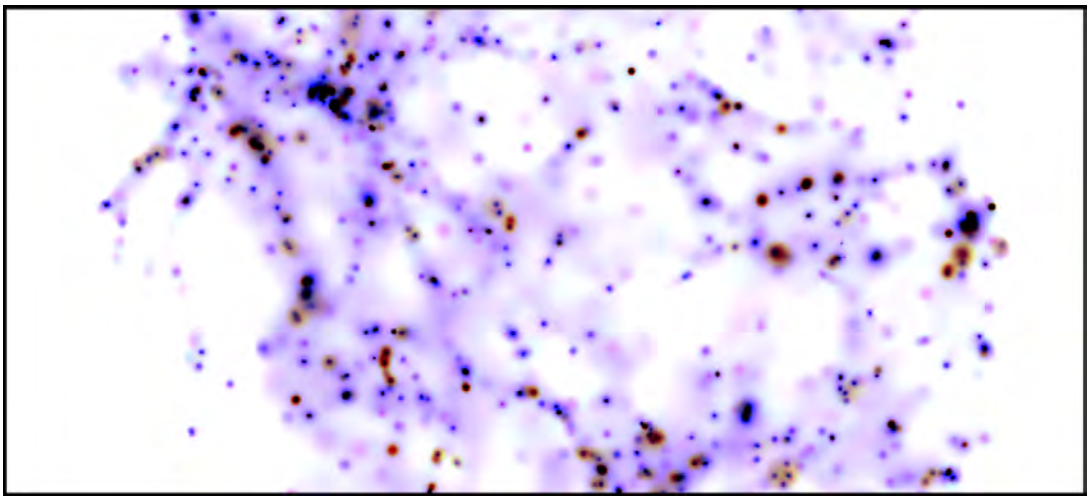

Figure 3.2: Composite image of a GADGET-3 16 $Mpc^3$ simulation snapshot at z=0, using the visualisation tool Gadget File Viewer. The perspective view covers a region approximately 20 Mpc by 8 Mpc.

& White was installed on the ACE Lab cluster and a set of possible geometric remappings generated using the included calculator. The LADUMA survey will have a 1 $deg^2$ opening angle (or field-of-view) at low redshifts, so the ideal remapping should produce a long and thin cuboid shape. Therefore the following remapping matrix was selected:

$$(u_1, u_2, u_3) = \begin{vmatrix} 17 & 3 & 15 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{vmatrix}$$

which corresponds to a remapping scaling function of:

$$(L_1, L_2, L_3) = (22.8692, 0.2051, 0.2132)$$

These parameters were supplied to the remapping tool, and the 16 $Mpc^3$ box provided as input. The resulting cuboid which was produced resembled a 3D rectangle with dimensions of 365 Mpc x 3.2 Mpc x 3.4 Mpc. Thus effectively increasing the maximum possible depth of a cone within this volume from 16 Mpc to 365 Mpc.

In order to extract the correct conic volume from this new cuboid, a Python program was written. It operated on a line-of-sight vector and used a simple linear relationship to calculate the radius of the cone as a function of distance from the observer. This approach

does not accurately represent the mechanics of radio telescope observations. This is because beam-width increases as a function of redshift which is not linearly related to co-moving distance. However for sufficiently small scales, and as a proof-of-concept, this approximation proved adequate. Figure 3.3 presents the rudimentary lightcone produced using this method. This lightcone spans 55 Mpc in physical space and extends to a redshift of z=0.019.


Figure 3.3: A ParaView visualisation of an initial lightcone containing gas particles, using a linear beam width function and contained within a single GADGET-3 snapshot.

Several drawbacks were identified upon investigating and implementing this Carlson & White volume remapping approach for producing simulation volumes analogous to real-world observations [14]. As previously stated, the arithmetic operation is a one-to-one relation; therefore the remapped volume used for cone construction is directly limited by the volume (effectively resolution) of the simulation. In order to produce a synthetic lightone that is twice the depth, a GADGET-3 simulation would need to be run at twice the resolution. This is not ideal as GADGET-3 is a computationally intensive application which typically requires HPC infrastructure. Another consequence of this one-to-one mapping is that a compromise must be made in order to fit the cone's length and circular diameter within the remapped cuboid's volume. As stated, the opening area of the lightcone increases as a function of redshift, therefore at larger redshifts, the width of the cone increases significantly. This, combined with the increasing length, compounds the volume limitation of this technique. This method is best suited for operating on Semi-Analytical Model (SAM) simulations, as the modelling of dark matter interaction alone is computationally less demanding than gas particle interactions. Thus SAM simulations are typically of a larger scale than gas simulations.

Due to the reasons described above, an alternate method was selected for the production of synthetic lightcones. As stated in the literature review, the Obreschkow method used stacking of simulation boxes to construct a volume of sufficient depth for producing a

lightcone. The main drawback of this method was the random symmetry operations which were applied to each box in order to avoid the spurious periodicity inherent with replicating data. These operations, namely rotation, inversion or continuous translations, produced discontinuities at the boundaries of each simulation box. While this may be acceptable for conducting statistical measurements, such as mass functions, it does negatively influence the validity of more precise measurements, such as galaxy detections. Thus, a modification of this method was developed and is detailed in Section 3.4 below.

Following the initial investigation regarding cosmological simulations and lightcone construction, an extensive investigation into GADGET-3 performance was conducted. This was done in preparation for a larger and more computationally intensive simulation. In addition to improving the runtime of GADGET-3 simulations for this research, the investigation serves as an up-to-date performance optimisation guide for other GADGET-3 users in the scientific community.

Upon completion of the performance evaluation and obtaining larger simulation results from GADGET-3, efforts were focused on developing a more functional HI lightcone that met the LADUMA survey requirements. This code was developed in Python, so conformed to an existing set of GADGET-3 post-processing tools; SPHGR. The details of this process are presented in Chapter 4.

In order to store and perform calculations on the vast number of simulation particles contained within the lightcone, a pixelisation scheme was designed, which decreases the resolution of the data within the cone. Particles which fall inside the geometry of the cone are binned into a pixel corresponding to their physical position. The pixel ranges are defined as fractions of the opening angle in two planes, and as frequency channels in the line-of-sight dimension. After a particle is binned, various metadata properties are calculated for that particle depending on what type of particle it is. For dark matter and star particles the number and mass of particles is stored. For gas particles additional properties are also stored such as line-of-sight peculiar velocity, HI mass, radio continuum, ionised mass and flux received.

## 3.2 GADGET-3 Optimisation

This section will detail the configuration of the software libraries, dependencies and compilers used in evaluating GADGET-3. An in depth overview will also provide specifics on the approach implemented in order to identify and analysis performance factors of the

code.

### 3.2.1 GADGET-3 Evaluation Environment

The GADGET-3 code was designed to run a Linux HPC cluster thus it is highly configurable. This section details the software and hardware used for profiling GADGET-3 in order to evaluate its performance and determine if any optimisation is possible. Due to the time constraints of the project and the complexity of the code, the GADGET-3 source code itself was not edited. Rather, focus was placed on improving the computational efficiency through the use of optimised compilers, software libraries and the software environment. In a Linux based HPC cluster, such factors are often the main determining factor of an application's performance.

The performance of a code running on a cluster is dependent on many factors; the hardware configuration, the software environment as well as individual configuration parameters within the code. This situation results in a multi-variable optimisation problem with a vast number of potential configuration combinations. Therefore a systematic evaluation approach was implemented, whereby an individual performance aspect was isolated and considered, irrespective of the other variables. This approach may overlook more subtle dependencies by assuming the variables are independent. However due to a lack of alternatives, it was selected as the most efficient and comprehensive evaluation approach.

Through investigating the GADGET-3 code, its dependencies and the software environment which exists in a Linux HPC cluster; the following criteria were selected for evaluation. GADGET-3 has undergone many updates throughout its development, with features being added and changes in performance with every build. Therefore the first aspect investigated was a comparison between two recent versions of GADGET-3 to illustrate performance differences. Within a Linux software environment and particularly in HPC, software is compiled specifically for the hardware architecture on which it is installed. This places a large importance on compilers needed to produce effective code in order to execute efficiently on the hardware. Thus the second aspect investigated of GADGET-3 was compiler performance. The ACE Lab cluster implements a Cloud computing infrastructure of KVM virtual machines which are managed by OpenNebula, a Cloud Management Platform (CMP). Initial testing was conducted within this virtual environment. In order to investigate the poor scaling observed for GADGET-3, a comparison was conducted between the virtual and hardware cluster environment with regards to computational

throughput scaling. Finally, a simple test was conducted to determine what effects the Intel Compiler Suite flags had on code runtime.

#### 3.2.1.1 The HPC Cluster Configuration

The ACE Lab HPC cluster used for this research comprises of six compute nodes. In addition, a storage node is used to host a shared file system and a head node used for management, compilation and an interface to users. Each of the compute nodes contain two Intel Xeon E5-2690 v2 IvyBridge CPUs (totalling physical 20 cores), 128 GB of high speed DDR3 1866 MHz RAM and a Infiniband Fourteen Data Rate (FDR) 56 Gbps Host Channel Adaptor (HCA) for high speed and low latency network communication. Figure 3.4 provides the topology of the ACE Lab's HPC computer cluster, showing the networked hosts and interconnect infrastructure.

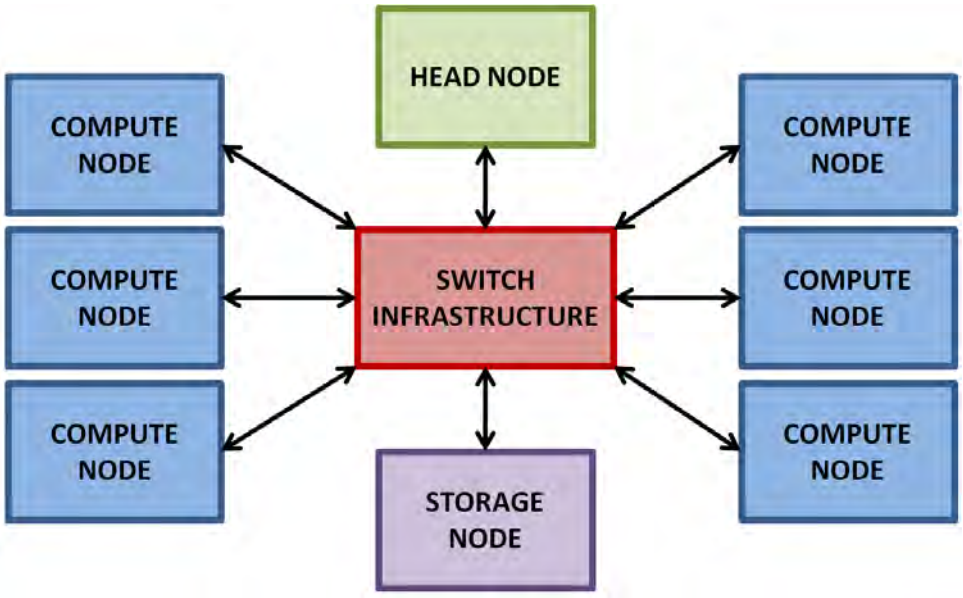


Figure 3.4: A network topology diagram showing the layout of ACE Lab's HPC cluster.

The cluster uses Community Enterprise Operating System (CentOS) 6.5, an unbranded enterprise class Linux distribution based on Red Hat Enterprise Linux (RHEL). The operating system is widely used in industry and has a large user base. Furthermore it is actively developed for enterprise and HPC environments. OpenNebula 4.8.1 was deployed as part of the software stack to provide the Cloud computing infrastructure that is available on the ACE Lab cluster. This Cloud Management Platform (CMP) provides a user interface for managing and provisioning virtual machines. Initial testing of GADGET-3 was conducted within virtual machines, after which the code was rerun within the host operating systems to test virtualisation overheads.

To provide improved performance and advanced functionality for the Infiniband FDR interconnect, the Mellanox distribution of the Open Fabrics Enterprise Distribution (OFED) software stack was installed. This software also adds support for features such as Single Root Input Output Virtualisation (SR-IOV) for Infiniband. This allows PCIe hardware pass-through for virtual machines, version 2.2-1.0.1 was installed at the time of testing.

#### 3.2.1.2 Software Compilers

During the initial phase of testing, open source compilers were used to compile GADGET-3. Specifically, the popular GNU Compiler Collection (GCC) compiler was used to generate the compiled binary from source code. In addition, an implementation of Message Passage Interface (MPI) is required for the runtime network communication component of GADGET-3. OpenMPI was chosen as the open source option of MPI as its use with GADGET-2 is well documented [65] [74]. GCC version 4.8.1 and OpenMPI version 1.6.5 were used during this evaluation.

As part of the optimisation process, Intel's proprietary Intel Compiler Suite (ICS) was used to provide a comparison between the popular open source GCC compiler and a proprietary alternative. ICS is based on the GCC compiler but adds architecture specific optimisations for Intel processor micro-architectures. This potentially allows this GCC compatible compiler to provide greater performance through improved hardware utilisation. Included in the 2013.1.117 release of ICS used in this evaluation, is Intel's MPI implementation (IMPI) version 4.1.0.024, which provides more optimised MPI code compilation and runtime communication.

#### 3.2.1.3 GADGET-3 Library Dependencies

The first approach used to improve an application's performance on a Linux based HPC cluster is often to update to the newest compilers, libraries and dependencies. This is because Linux relies on specific architecture compiled binaries and libraries to fully utilise the available resources. Depending on the nature of the computational workload and the hardware architecture of the system, certain compilers would have a performance edge. Hence extensive testing is required to identify the best option.

GADGET-3 relies on three libraries, namely Fastest Fourier Transform in the West

(FFTW), GNU Scientific Library (GSL) and Hierarchical Data Format (HDF5). FFTW is a library used for computing discrete fast Fourier transforms, it is generally considered the fastest open source implementation for computing multidimensional FFTs [75]. GSL is an open source library which provides subroutines for common computational tasks in applied mathematics and other sciences. GSL was written in C and provides support for Basic Linear Algebra Subprograms (BLAS) as well as many other mathematical operations. HDF5 is a data container standard for storing and maintaining both structured and unstructured large format datasets [76]. This functionality was not required for this research so the HDF5 module was deactivated during compilation thus the package not installed.

GADGET-3 requires the older version 2 of FFTW, due to the lack of multi-threading support in newer versions [77]. FFTW 2.1.5 and GSL 1.16 were installed onto the shared file system within the ACE Lab cluster using GCC 4.8.1. In order to keep the compiler comparison consistent, libraries were compiled with ICS during that testing phase.

Figure 3.5 presents a diagram of GADGET-3's dependency structure as well as its output data.

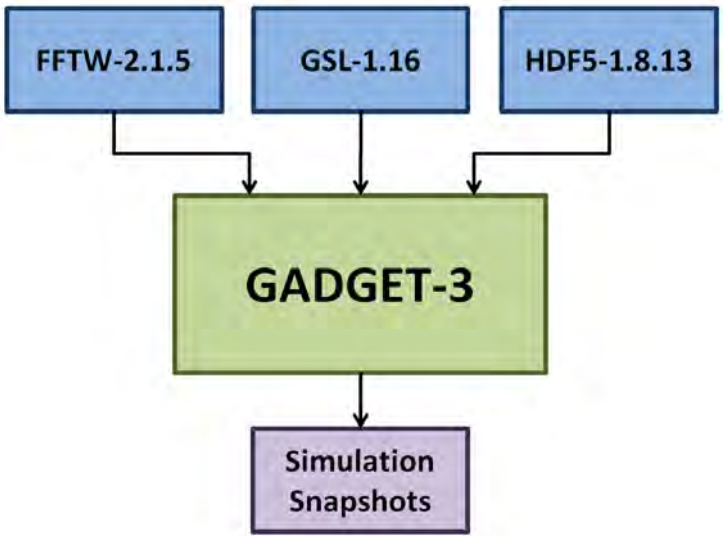


Figure 3.5: A software dependency diagram showing GADGET-3 library requirement needed to produce simulation snapshots.

### 3.2.2 The GADGET-3 Optimisation Plan

A relatively small GADGET-3 test case simulation was created in order to conduct a thorough investigation in a timely fashion. This test case was based on a previous

simulation using a 16 $Mpc^3$ box containing $128^3$ dark matter and gas particles. In this simulation, gas particles were included thus the SPH solver will be implemented. Even though this simulation was eight times smaller than the one used to generate raw data for the HI skycones, a complete run would take several hours to complete. Therefore a subset of the full simulation time period was selected. Rather than running from an initial redshift of z=125 to present day (z=0), a small subset of this period was selected (z=2.5 to z=2.0). This period was specifically chosen as significant star formation and other complex, computationally intensive processes occur at that time. The GADGET-3 solver calculates the necessary time resolution for each region based on domain decomposition, which in turn is dependent on the number of cores and hosts on which the simulation is run. Therefore the number of time steps produced between z=2.5 and z=2.0 differs when ran using different cluster and software configurations. These variations are however small, with the all test cases producing approximately 4000 time steps.

GADGET-3 has a built in module-level monitoring system which logs data for each time step during runtime. This information includes a hierarchical structure which breaks down execution time into a gravity algorithm (for collisionless gravitational interactions), SPH (for gas particle interactions) and other functions such as I/O and the Friends-of-Friends (FoF) algorithm used for grouping particles. For optimal performance, the application should spend as much time as possible executing the computationally intensive tasks like gravity and SPH calculations, while spending less time on I/O and communication components.

A number of output were used to analyse the code and compare performance between different software configurations. Superficially the GADGET-3 data logs, hardware utilisation logs and runtime measurements were considered.

#### 3.2.2.1 Version Improvements

GADGET-3 was in pre-release status at the time of this investigation. Initially code from the April 2014 release of GADGET-3 version 3.25 was used. During the evaluation process, the code received several important updates which modified and optimised the solver algorithm. An important modification that dramatically sped up code execution was altering how the FoF algorithm was implemented. This function is used to identify particle groups, such as galaxies, and is computationally costly. The code was updated in the June 2014 release to implement the algorithm less frequently; resulting in a shorter run time, with effectively identical results.

#### 3.2.2.2 Software Compiler Comparison

Intel Compiler Suite (ICS) is a software suite maintained by the microprocessor manufacturer Intel Corp. This proprietary software development toolkit includes a GCC compatible compiler for C, C++, Fortran as well as several other languages. Additionally, it includes an implementation of MPI which is compatible with both the Intel and GCC compiler. The suite is intended to optimise code at compile time to take advantage of Intel hardware. This provides more efficiently code execution and communication between nodes over an Infiniband network. The June release of GADGET-3 and its dependencies, FFTW and GSL, were recompiled using Intel's ICS 2013 and a comparison was done to investigate any performance improvements.

#### 3.2.2.3 Virtual Versus Native Software Environments

Further testing with the ICS 2013 demonstrated that the code was suffering poor scaling when run on multiple nodes within the cluster of virtual machines (VMs). Performance overheads are a well documented drawback to virtualisation, specifically with regards to HPC applications. To confirm virtualisation was the cause of this poor scaling, the code was moved out of the virtual environment to a cluster of native hardware machines. This infrastructure is reserved for hosting the virtual cluster. The test case was reran to compare performance scaling from one to five compute nodes.

#### 3.2.2.4 ICS Optimisation Flags

The final subject of this investigation was to determine if any performance improvements were available through compiler optimisation flags. For this study, the Intel ICS 2013 compiler's standard optimisation flags were compared. By default, the GADGET-3 makefile invokes the -o2 optimisation flag which is generally the recommended optimisation level for performance [78]. However there are additional options which provide further optimisation for specific configurations and architectures. Level -o1 optimises for size, whereas level -o3 implements more aggressive, potentially unsafe, optimisation techniques to further boost performance.

### 3.2.3 Lightcone Code Development

The decision was made to use the Obreschkow method [15] of box stacking to produce a lightcone from GADGET-3 simulation data. A number of design decisions were necessary in order to determine the specific requirements of the code. The software tool-kit developed in this research may be incorporated into a larger GADGET-3 post-processing suite, called SPHGR, developed by Prof. Romeel Davé and Dr. Robert Thompson. This provided some context for the requirements of the code. Firstly, the existing software package has been developed in Python, therefore for the purpose of uniformity; this code was also developed in Python. However, the code should also be sufficiently fast such that a researcher could run it on modest hardware and obtain results in a reasonable amount of time. Finally, as this code may be used by others in applications for which it was not specifically developed, the lightcone generation algorithm should be as generalised as possible.

The development followed an iterative process, in which initial code was implemented, tested and improved upon. Documentation was continually updated in order to keep the code legible for potential further development in the future. Focus was originally placed on producing a volume extraction algorithm to partition gas particles from GADGET-3 output snapshots. A Cartesian co-ordinate system was established which orientated each box being processed to a fixed observing point, such that the cone's geometry remained consistent between boxes. After this co-ordinate system became functional, a snapshot tiling process was developed to allow for the workable volume to be extended in both depth and breadth, for creating larger lightcones. A generalisation of this process was then coded to create lightcones of arbitrary depth and geometry in any desired direction.

Upon testing the code on larger datasets, the post-processing runtime was found to be undesirable (in the region of 3 hours). This lengthy runtime was due to poor hardware utilisation. This is often the case with code implemented as loops in a high-level language such as Python. Therefore effort was placed on optimising the Python code, which included linking high performance math libraries for the NumPy and SciPy Python packages. In addition, code was restructured and function calls optimised to reduce the computational workload. Finally, code vectorisation was implemented which dramatically improved code efficiency and decreased runtime.

The typical lightcone created using the z=0.58 depth specified for the LADUMA survey contained approximately 75 million particles. A pixelisation procedure was developed in order to reduce the amount of data contained within the lightcone. In addition the

pixelisation produces more realistic representation of the particles, as viewed by a radio telescope. The lightcone was partitioned into a discrete grid aligned volume, implemented as a four dimensional array. The first three dimensions defined the spatial location of a 'pixel', while the fourth dimension was used to store data of particles found within each pixel.

Once a reasonably efficient lightcone construction and pixelisation procedure was developed, focus was placed on generating useful metadata for the processed particles. The HI mass content of gas particles was approximated using the semi-analytical method described in the Davé *et al.* paper, reviewed in Section 2.6.2. This process proved challenging due to the mathematical and scientific nuances associated with operating on cosmological simulation data. Other properties were also calculated for each of the simulation particles falling within the lightcone's geometry, including the line-of-sight velocity. This peculiar velocity is the rate at which the particle is travelling through physical space as opposed to its apparent recession velocity caused by cosmological expansion. The component of this peculiar velocity along the line-of-sight creates a bias on the recession velocity for radio frequency observations, thus it is a useful property for the lightcone. The received flux of each particle was also computed using the Abdalla and Rawlings method published in their 2005 paper [72].

In order to improve the usability of the code, a parameter file was generated for users to specify key configuration variables. This included the directory of GADGET-3 snapshots, a vector defining the line-of-sight, and the desired depth of the lightcone in redshift. In addition, hardware information such as the number of CPU threads available to the code could be provided. A front-end Python file was executed by the user which initialised a pool of worker threads to perform lightcone processing on multiple GADGET-3 snapshots simultaneously. Upon completion of all worker threads, a final data post-processing phase was initialised. This phase stitched the outputs of the individual worker threads together and performed analytical operations on the entire lightcone volume, including a source finding algorithm developed to find flux sources which broke pixels' bounds.

## 3.3 Conclusion

The GADGET-3 code was optimised to reduce the lengthy runtime for simulating galaxy evolution in an expanding Universe. A methodical approach was used to optimise the software environment, compiler selection and software libraries upon which GADGET-3 relied in order to reduce the duration of simulations.

This chapter presented the development methodology for generating GADGET-3 simulation data and producing a lightcone from it. The Carlson & White volume remapping approach for extracting data from GADGET-3 simulation snapshots was attempted but ultimately determined not to have met the requirements of a LADUMA lightcone. A modified Obreschkow box stacking method was implemented instead. This method did not have a one-to-one volume relation between the simulation and lightcone and therefore was not computationally bound by the GADGET-3 simulation.

An overview of the post-processing Python code design was presented. This included the design choices and order of events during the development process. An expanded description is provided in Chapter 4.

# 4 Design of Lightcone Code

This chapter provides details of the code developed for processing the GADGET-3 simulation data into a dataset resembling a radio telescope's observational field-of-view. Section 4.1 discusses the code responsible for the lightcone's construction. A summary of the particle selection algorithm is described; this code tested particles within the GADGET-3 simulation to determine if they fall within the bounds of the lightcone's geometry. After each particle has been tested a pixelisation process was performed. This partitions the lightcone's volume into a fixed grid structure and adds the particles into this grid. As part of the scientific deliverable of this research, metadata was derived for the simulation particles. Attributes of each particle are either accumulated to, or averaged with, the other particles in that pixel. The following section described equations used to calculate some of these properties, such as HI mass and measured flux.

Section 4.2 presents several steps taken to optimise the Python post-processing code in order to improve efficiency and reduce runtime. Special emphasis was placed on reducing the code's hardware requirements, thus making it more accessible. This code is intended for researchers and scientists who may not have access to HPC resources. Included in this overview is an introduction to the NumPy and SciPy math libraries for Python. Linking these libraries to high performance BLAS implementations such as Intel MKL or ATLAS provided significant performance improvements for the mathematically intensive lightcone program. Parallelisation was implemented to improve the code's performance on multi-core computers. The processes followed to re-factor and vectorise the code will be detailed to provide a basis for the performance improvements presented in the Results.

## 4.1 Lightcone Construction Approach

A large GADGET-3 simulation was required in order to produce the synthetic lightcone for the LADUMA survey. Following the GADGET-3 optimisation process (detailed in the

previous chapter), this simulation was computed. The parameters for the simulation were set to $256^3$ (16.77 million) gas and $256^3$ dark matter particles in a 64 $Mpc^3$ simulation volume. The initial conditions for the simulation were generated with the P-GenIC tool. A number of additional parameters were set which defined the cosmology of the simulation to best model the evolution of the observable Universe. These parameters were chosen as they were consistent with the current parametrisation of the $\Lambda CDM$ model. A Hubble parameter of 0.7 was used, along with $\Omega_M$ (fractional matter density - including dark matter and baryons) of 0.3 while the fractional density of baryons was defined to be 0.045. Finally, the $\Omega_\lambda$ (fractional dark energy density) was selected to be 0.7.

The first task necessary to generate a lightcone from this simulation was to define the geometry of the cone. This geometry emulates the Field-of-View (FoV) observed by the hypothetical radio telescope (referred to as the 'Observer').

### 4.1.1 Defining Lightcone Geometry

As documented, the main drawback to the Obreschkow cube stacking technique is the symmetry operations implemented to avoid spurious periodicity introduced by duplicating simulation data in the Observer's FoV. While these operations effectively reduce the perceived repetition of data, they also disrupt the patterns in the large scale structures in the boundary regions. A modification of this method was therefore developed. Rather than rotating the simulation cubes, the axis along which the cone is constructed is 'skewed' relative to the stacked cubes. This results in the lightcone bisecting a different region of each consecutive simulation cube.

At certain observational wavelengths, such as in the optical band, the aperture (or opening area) of the telescope's FoV expands at a fixed rate with distance. This produces a cone volume in three-dimensional space expanding away from the Observer, placed at a fixed location in space. In radio frequency observations however, the beam-width of the telescope's FoV is inversely proportional to its operating frequency. The beam-width increases at lower frequencies (corresponding to higher redshifts). The LADUMA survey will have an observational redshift range of z=0 to z=0.58. This corresponds to a minimum operating frequency of 900 MHz in order to observe the 21cm (or 1420 MHz) emission line of HI at a redshift of z=0.58. MeerKAT will begin observing with an opening area of 1 $deg^2$, however because the HI emission frequency decreases with redshift, the opening area of its FoV increases with redshift by:

$$A(z) = A_o(1+z)^2 \tag{4.1}$$

where z is redshift, $A(z)$ is opening area as a function of redshift, and $A_o$ is the opening area at zero redshift. Using Equation 4.1, it follows that at a redshift of z=0.58 the opening area of the MeerKAT telescope will increase to 2.5 deg$^2$. Figure 4.1 provides a simple illustration of how the lightcone's opening angle increases with redshift.

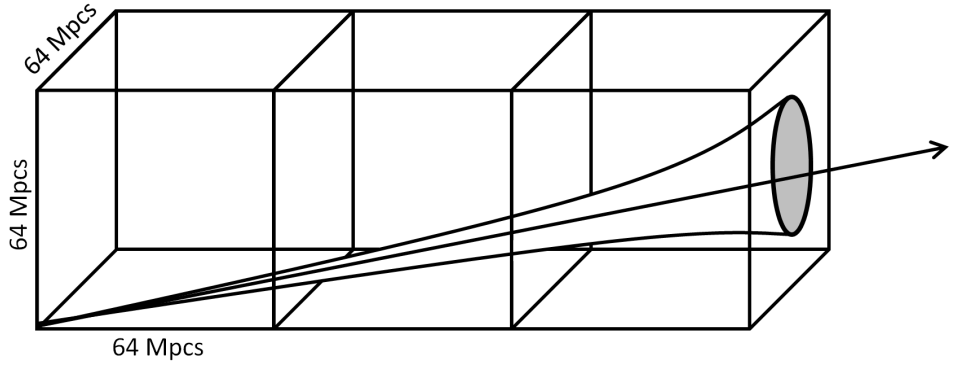


Figure 4.1: An illustration of a virtual telescope's Field-of-View expanding with redshift. A set of stacked simulation cubes are also shown.

Using this relationship between redshift and the opening area, a particle selection algorithm was developed to operate on GADGET-3 snapshots. A set of 'pyGadgetReader' functions from the TipsyTools software suite were used to interface with the snapshot data files. A test was performed using the position of each particle in the simulation, to determine if the particle falls within the bounds of the lightcone. It was necessary to test every particle within the simulation space, as the data was unstructured and unordered. Initial testing was conducted on the gas participles only, but later star and dark matter particles were also tested to include these types within the lightcone. The lightcone's instantaneous opening radius was used to determine its bounds for any given distance from the Observer. The cone's radius as a function of redshift is defined by:

$$R = tan(\frac{z(L)}{\pi}) * L \tag{4.2}$$

where L is the co-moving length of the lightcone and $z(L)$ is the equivalent redshift to this co-moving distance.

This particle testing was performed within each simulation cube using the particle's position, as well as the lightcone's axis (or pointing) vector and aperture radius. Two angles were calculated for every particle, one between the lightcone's pointing vector and aperture radius, and another between the pointing vector and the position of the particle. The lightcone's aperture radius increases with distance, thus the radius is recalculated for every particle. By comparing these two angles, each of the tested particles were categorised as inside or outside the lightcone. Figure 4.2 provides a graphical illustration of the testing procedure.

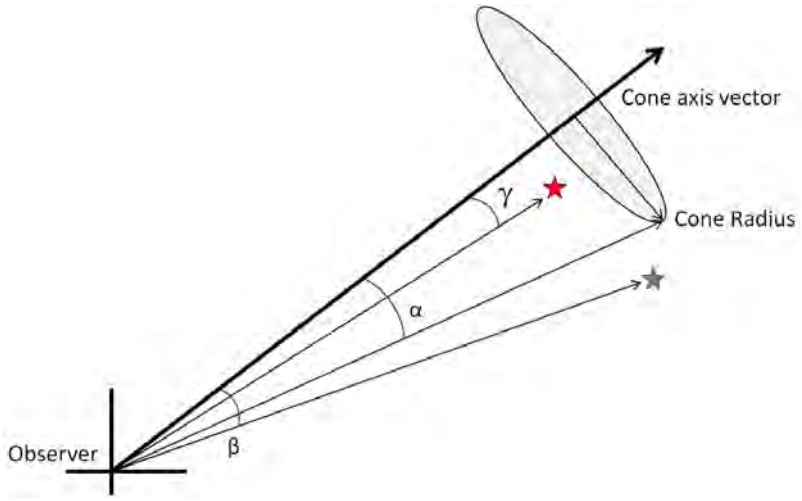


Figure 4.2: An illustration demonstrating the particle testing procedure. The angles between the pointing vector (cone axis vector), cone radius and particle position are compared. The angle between the red particle and the pointing vector ($\gamma$) is smaller than the angle between the pointing vector and the lightcone's radius ($\alpha$), in this case the particle is considered inside the lightcone. The opposite is true for the blue particle ($\beta$).

The GADGET-3 code was configured to produce 140 snapshot outputs over its simulation duration. Each snapshot was selected using its simulation redshift value, in order to keep the cosmology consistent with observations. The redshift of the lightcone at each cube was calculated using it's co-moving distance from the Observer. A corresponding snapshot was then selected to minimise the difference between the physical and simulated redshifts.

A lightcone could now be created within a single simulation volume of 64 $\text{Mpc}^3$. This was considerably smaller than the 1500 Mpc required to reach a redshift of 0.58. Thus, the modified Obreschkow cube stacking method was implemented. A number of GADGET-3 snapshots were combined into a larger volume from which the lightcone could be constructed. Approximately 24 of the 64 $\text{Mpc}^3$ snapshots (cubes) stacked end-on-end were required to produce a sufficiently large sampling volume. Therefore, the code was written in a generalised fashion such that the same particle testing algorithm could be applied to every cube regardless of its location. Each of the cubes along the length of the

lightcone was provided with a set of parameters, including the lightcone's pointing vector as well as an ID used orientate the cube relative to the Observer. Using this information, the code determines the bounds of the lightcone as it passes through that specific cube.

Figure 4.3 provides a 3D depiction the lightcone's volume within a simulation cube. This cube is located some distance from the Observer; this is evident as the lightcone's radius is not zero at the entry point. Refer to Appendix A: "Python Function for particle testing" for the code implementation of the testing algorithm.

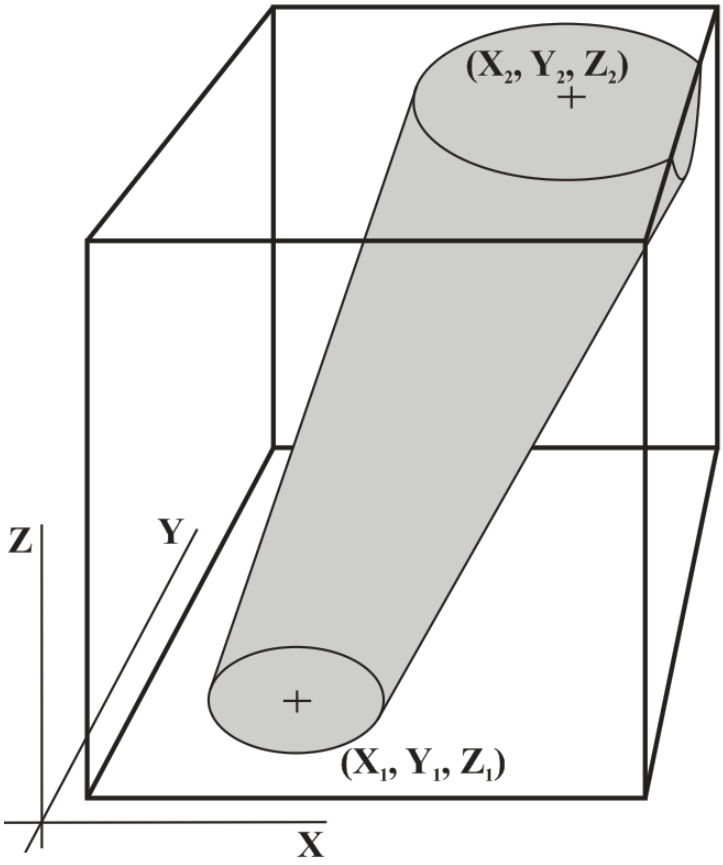


Figure 4.3: A 3D diagram showing a single simulation cube, with the lightcone structure displayed in grey. Particles found within this grey volume are considered to be inside the lightcone. In this example, it is clear that the simulation cube should be tiled to the right to prevent the lightcone from breaking out of the cube's boundary.

From Figure 4.3, it is important to note the spatial co-ordinates of the lightcone's entry point into this volume, denoted by $(X_1, Y_1)$, differ from the exit co-ordinates $(X_2, Y_2)$. It follows that the lightcone would enter the next simulation cube at position $(X_2, Y_2, Z_2)$. At some distance from the Observer, the lightcone's entry co-ordinates would extend beyond the limits of the cube, in order to avoid the offset growing too large, a modulus operation was applied to the entry co-ordinates. As the lightcone's entry and exit points from the cube were at different locations, a new region of each snapshot volume was

sampled.

```
while tiling:
    newTile = oldTile +offSet*cubeSize
    testTile(newTile)
```

The skewed lightcone traverses sideways through the simulation cubes. At greater distances the lightcone breaks the lateral bounds of the cube's volume. A test is therefore performed to check if the cube's bounds have been broken in order to avoid sampling empty space. In such a situation, the cube was duplicated (or tiled) in the appropriate direction. In its current form, the simulation cubes are only tiled a maximum of one time in any direction. Motivation for this decision is provided in the Discussion chapter. Figure 4.4 illustrates the cumulative volume generated by the first twelve of the twenty four simulation cubes required for a z=0.58 redshift lightcone. Note that in some cases the cubes were tiled sideways to accommodate the skewed cone axis vector. All testing and tiling is performed automatically by the code, such that the pointing vector can be simply changed in the parameter file and a new lightcone created. In addition, each stacked simulation cube is independent of the others above and below it. The cube's position relative to the Observer is determined by its ID number. This provided the possibility to implement a simple parallelisation technique of 'embarrassingly' parallel cube testing and binning threads.

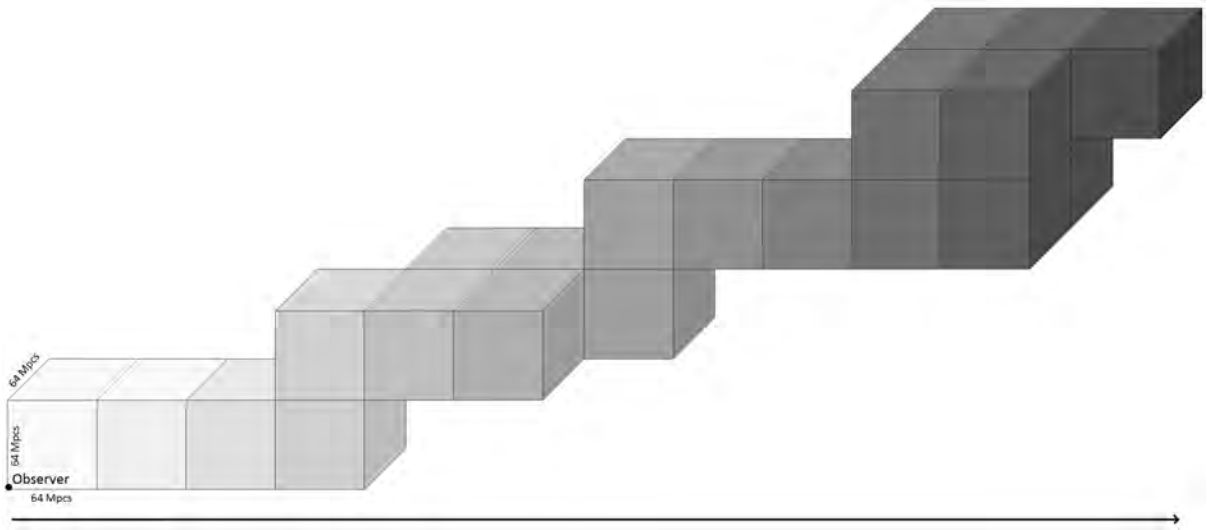


Figure 4.4: A graphic illustrating the GADGET-3 simulation cube stacking method. The first 12 stacked cubes required to produce a lightcone with pointing vector [10,3,-2] are shown. The shades of grey illustrate the cube's distance from the Observer, with darker being further away. Sideways tiling of cubes is necessary to contain the expanding lightcone.

### 4.1.2 Lightcone Volume Pixelisation

After the lightcone geometry was defined, the volume of the lightcone was partitioned into a grid structure. This 'datacube' structure was implemented to reduce the amount of data within the lightcone. This process also allowed for easier analysis of the cube's contents.

The grid represents a three-dimensional discrete 'binning' (or histogram) array, with each cell of the array being a 3D equivalent of a 2D 'pixel'. The lightcone's radius increases with distance from the Observer, thus a fixed Cartesian grid structure proved ineffective and wasteful for describing the lightcone's contents. Rather the two spatial dimensions, which represent the lightcone's angular resolution, are defined as a fraction of the cone's radius. Therefore, at a lower redshift (closer to the Observer) the volume of each cell is significantly smaller than its volume at a higher redshift. The third dimension exists along the Observer's line-of-sight. This dimension represents the frequency resolution of the MeerKAT telescope. As the lightcone is considerably longer than it is wide, the frequency resolution of a radio telescope is lower than the angular resolution. Within the lightcone, the frequency divisions are calculated as equal fractions (or channels) of the total frequency range. The sizes of the spatial and frequency dimensions within the array are defined by the user in the parameter file. Figure 4.5 shows how the frequency and spatial dimensions are partitioned. N represents an arbitrary frequency channel, while R is the cone's radius and M is the number of spatial pixels per dimension.

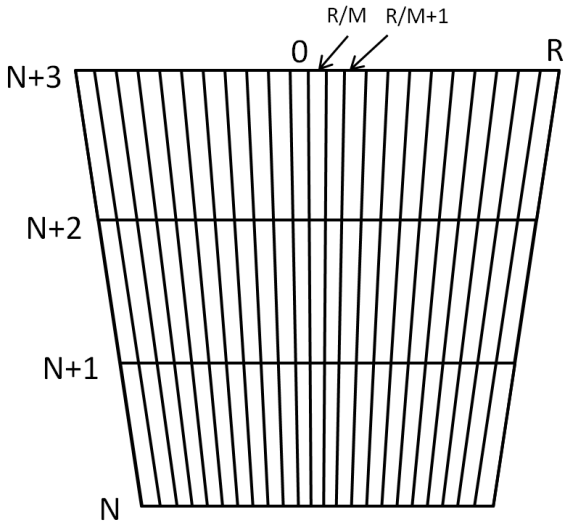


Figure 4.5: An illustration of the lightcone pixelisation technique, the image describes a section of the cone along one of the lateral dimensions. N denotes an arbitrary position along the frequency axis, while R denotes the cone's radius and M the number of spatial pixels.

Figure 4.6 presents a three dimensional perspective of the pixelisation grid which contains the lightcone. In this example, both of the spatial dimensions is partitioned by 15 pixels while the frequency dimension is partitioned into 8 discrete bins.

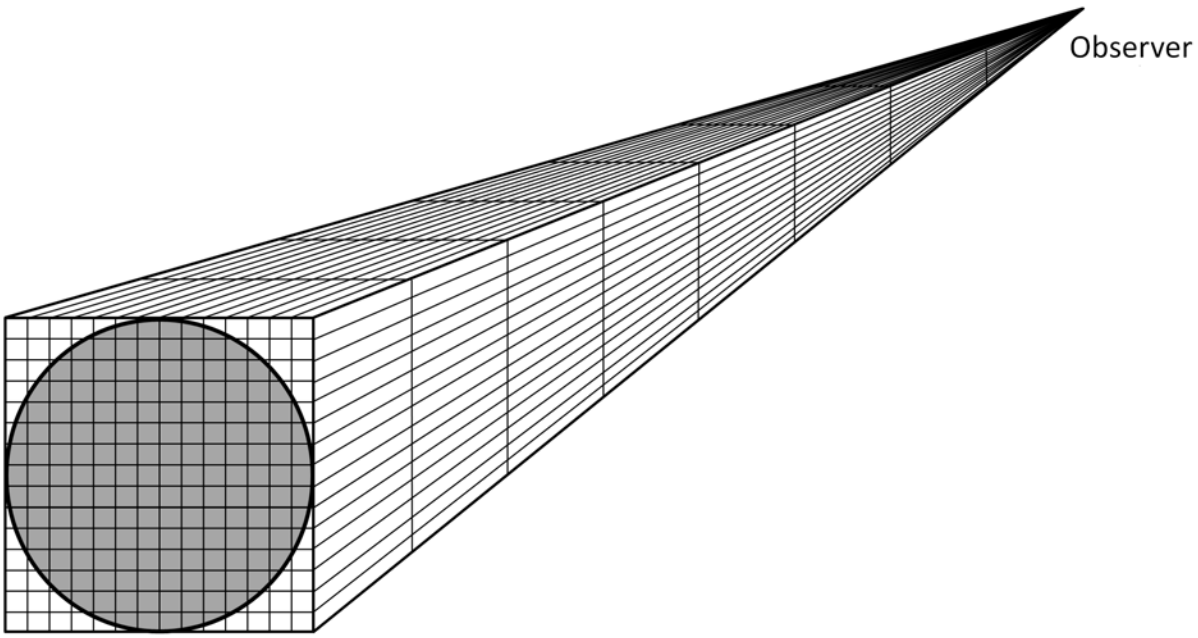


Figure 4.6: A graphical depiction of the pixelised lightcone, the opening area of the lightcone is partitioned into a 15x15 grid, and the depth partitioned into 8 discrete bins.

The geometry construction process described above was developed to operate independently on each simulation cube so that the process could be parallelised simply. This however led to complications when defining the grid structure between independent cubes. Each cube contained a subset of the total pixelisation array. The number of line-of-sight frequency bins per cube is calculated given its offset from the Observer. The particles within each cube are pixelised using this information, after which all the lightcone sections are stitched together in the final post-processing phase.

Multiple properties were stored for each pixel in order to provide a comprehensive description of the lightcone's contents. Therefore a four dimensional NumPy array was implemented. Three dimensions were used to spatially identify a pixel, while the fourth dimension was used to store particle metadata. Section 4.1.4 provides additional information on the datacube structure.

Refer to Appendix B: "Python Function for particle binning", for the code implementation used to pixelise the particles within the lightcone.

### 4.1.3 Particle Metadata Extraction Algorithms

Scientists with different research objectives are involved with the LADUMA survey, each of whom have a unique research focus. Thus, it was imperative that the HI lightcone conserve as much of the information extracted from the original simulation data as possible. A number of properties were determined to be relevant to the LADUMA survey and included within the lightcone.

As LADUMA is a deep field radio survey, its focus is placed on detecting the 21cm emission line from neutral hydrogen. Therefore, it was crucial to add this data to the lightcone. However, GADGET-3 currently does not quantify the gas within the simulation as neutral or ionised, nor does it differentiate between hydrogen, helium or any other elements. It was therefore necessary to post-process the simulation to extract this information. This was achieved using the Davé *et al.* method; introduced in the literature review. This technique was implemented within the Python code which conducts the particle testing and binning process. Each particle in the simulation is tested, binned, and post-processed - in that order.

Each particle was classified into one of two categories in order to calculate its HI mass fraction. The particle could either be star-forming or non star-forming. This distinction was made using the SFR (star formation rate) property which GADGET-3 generates for each gas particle. Appendix C: "Python Function for Evaluating HI Content of GADGET-3 gas particles" provides the code extract used for calculating this mass ratio.

Another useful property provided in the lightcone was the received flux from each particle. Flux spectral density (or spectral irradiance) is a unit of power received per square meter per hertz and is measured in janksy (Jy). One jansky is defined as:

$$1Jy = 10^{-26}\frac{W}{m^2.Hz} \tag{4.3}$$

This forms an important part of the information provided by the lightcone, as the flux received from an object determines if it is detectable above the ambient and system noise. Typically in radio astronomy, the flux received from an object must be $5\sigma$ above the noise level to be considered a detection. The flux of an object was extrapolated from its HI mass using the Abdalla and Rawlings method. The flux spectral density of a particle is calculated using equation 2.15.

In addition to these properties, a number of others were stored for gas particles. Namely; the number of particles, the mass of ionised gas, star formation rate, and line-of-sight peculiar velocities were also calculated and stored. After these properties have been calculated for each GADGET-3 gas particle, they are accumulated into pixels. The GADGET-3 star particles were also processed into the lightcone, the number and mass of star particles were binned into pixels and stored for further analysis. The user has the option to enable dark matter testing in the parameter file, including dark matter within the lightcone. The number and mass of dark matter particles per pixel are stored in the datacube along with the gas and star particle data. The properties stored in each pixel are a summation of the simulation particles found within that pixel. The exception was the redshift data which was not accumulated, as the distance to a given pixel remains fixed regardless of the number of particles within that pixel. Results of this process are presented in the Results chapter.

The lightcone construction process can be broadly classified into three subsections; particle remapping (cube tiling), particle testing and particle binning. Figure 4.7 presents these three main subsections, as well as their individual subroutines.

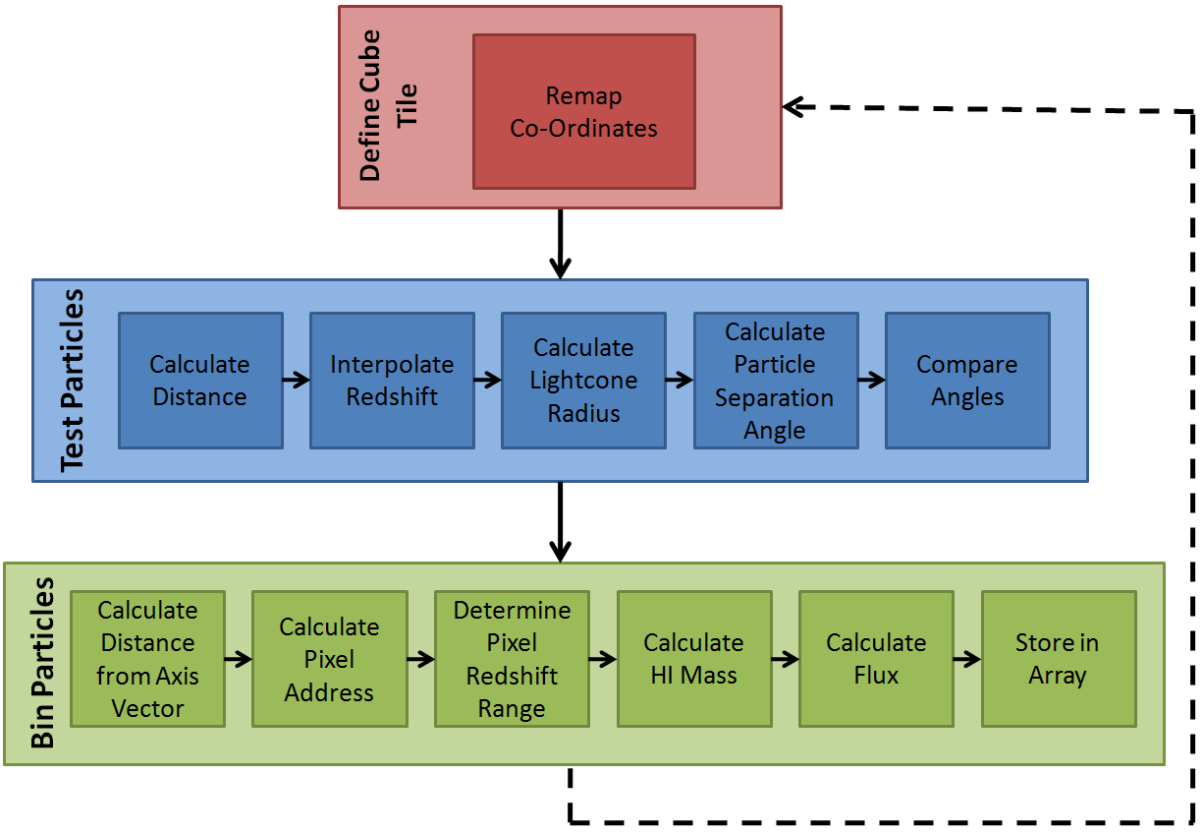


Figure 4.7: A graphical depiction of the particle processing pipeline developed in order to produce a lightcone from simulation particles.

The first process necessary is to remap the set of particle positions to tile the simulation cube to a suitable location in 3D space. These remapped particles are then tested to determine if they fall within the geometry of the expanding lightcone volume. This requires calculating the distance to the particles, then finding the opening angle of the lightcone at the corresponding redshift. The angle between the particle and the lightcone axis vector is then calculated, and the particle tested by comparing this to the cone's opening angle.

The resulting set of positively tested particles are then binned into the pixelised lightcone volume. Each particles distance from the axis vector is used to calculate its spatial pixel address. The minimum and delta redshift information is calculated for that pixel, as well as the particles metadata properties. This particle is then merged into the existing pixel.

### 4.1.4 Lightcone Data Structure

An N-dimensional NumPy data structure was constructed to store the lightcone structure. In order to store multiple properties for each pixel, found within a three dimensional spatial domain, a four dimensional array was selected. The size of this array is dependent on three user defined variables. Firstly, the number of spatial divisions (pixels) is determined by the pixel and frequency channel values in the parameter file. Additionally, the size of the fourth dimension is dependent on whether the user has enabled dark matter testing. If dark matter testing is disabled, the fourth dimension contains 10 single precision elements; alternatively it will contain 12 elements (to include the dark matter particle count and mass properties). Table 4.1 describes the contents of each 3D pixel, including the particle aggregation method.

There are several important implications resulting from the use of a dense NumPy N-Dimensional array for this dataset. Due to the nature of the cosmological data being sampled, the array is typically very sparse. On average, only 5% of all spatial positions contains any data. The result is a dense data structure that makes inefficient use of storage capacity and memory consumption. Several alternatives were explored in order to reduce the size of the data structure. An investigation into NumPy Structured arrays was conducted. These arrays utilise objects to store several different data types per element. This approach did not however address the wasted space of empty array elements.

A sparse array was instead considered to address the amount of wasted space. While SciPy has sparse array APIs, they are poorly suited to the high dimensionality of the

Table 4.1: Description of the data structure implemented for storing lightcone pixel information.

| **Array position** | **Property** | **Particle Aggregation Method** |
|---|---|---|
| 0 | Gas particle count | Accumulated |
| 1 | Redshift Minimum | Fixed |
| 2 | Delta Redshift | Fixed |
| 3 | Ionised gas mass | Accumulated |
| 4 | HI gas mass | Accumulated |
| 5 | Flux spectral density | Accumulated |
| 6 | Peculiar Velocity | Averaged |
| 7 | Star formation rate | Averaged |
| 8 | Star particle count | Accumulated |
| 9 | Stellar Mass | Accumulated |
| 10 (Optional) | Dark matter particle count | Accumulated |
| 11 (Optional) | Dark matter mass | Accumulated |

datacube. In addition, a sparse data structure increases the complexity of indexed operations. Such operations are useful for performing spatial operations (such as source finding) on the pixels. As memory was the limiting factor in array size and not disk space, this approach was also dismissed. Instead, a down casting of the array contents from double precision to single precision floating points proved sufficient to meet the requirements of a LADUMA scale lightcone.

Using a simple calculation, the amount of memory occupied by the datacube array could be calculated:

$$\frac{Size}{MB} = \frac{(FrequencyChannels)(SpatialPixels)^2(PropertiesPerPixel)(4Bytes)}{2^{10}} \quad (4.4)$$

Thus a typical lightcone with a pixelisation scheme of 4096x200x200, containing 12 elements per pixel, required 7 680 MB (7.5 GB) of RAM. This was found to be acceptable for the majority of test cases explored.

In addition to storing the contents of the pixelised lightcone to a NumPy array, another data format was also used. The Flexible Image Transport System (FITS) is a widely used open source standard for storing observational imaging data [79]. A FITS file contains a header (or data block) which describes the contents of the file. The data within a FITS

file can take any dimensionality. Multiple APIs exist for interfacing with a FITS file for programming languages such as FORTRAN, C++ and Python.

The flux data from the lightcone was stored to a FITS file, as the standard is well established in the observational community. Collaborators working with the lightcone data have access to this file for source finding and other common observational processes.

## 4.2 Performance Optimisations for Python

This section presents details on the optimisation process of the Python code developed for this research. This process begins with a discussion on software compilers and math libraries for Python. An investigation into the parallelisation of the lightcone construction process is also explored. Finally, a number of the optimisations implemented to further reduce the runtime of the post-processing code are discussed.

### 4.2.1 High Performance BLAS Libraries

Much of the computation done by the post-processing tool developed for this research operated on large datasets. Therefore the popular numerical and scientific packages, NumPY and SciPY, were installed onto the ACE Lab cluster. These packages provided easy-to-use APIs for linking Python code to high performance C and FORTRAN mathematical subroutines. The performance available from these packages is largely dependent on the specific backend math library implementations on which they rely. By default, the NumPy package uses its own implementation of the Basic Linear Algebra Subprograms (BLAS) standardised math library. However, the package also supports linking to non-standard libraries during compilation. Due to the potential performance improvements offered by linking to a more efficient BLAS implementation, an investigation was conducted using several alternatives.

A comparison was conducted between three BLAS implementations: Intel's Math Kernel Library (MKL); the open source BLAS implementation Automatically Tuned Linear Algebra Software (ATLAS), and the default NumPy version of BLAS. The MKL package is included in Intel's ICS software suite and promised improved performance through specific Intel hardware optimisation. ATLAS is considered one of the best open source implementations of BLAS; as the name suggests, it performs an automatic performance

tuning process during installation.

A benchmark was constructed comprising of five basic linear algebra operations; matrix eigenvalue, matrix inversion, single value decompositions, matrix determinant and matrix dot product. 1000 iterations of each test case were performed to produce measurable runtimes.

### 4.2.2 Code Parallelisation

The lightcone construction technique, described above, was intentionally developed so that each simulation cube could be considered an independent subsection within the larger lightcone structure. This design decision was chosen so that a simple data-parallel parallelisation technique could be implemented.

The program was initialised from a single Python program named "launcher.py". This code extracted relevant information from the user's parameter file. Thereafter, it calculated the number of stacked boxes required to produce a lightcone of the specified redshift using the simulation snapshots provided. The launcher then spawned a thread pool of processing threads using the "cube.py" program. To improve performance on SMP machines, the user can specify the maximum number of concurrent threads in the parameter file. After each cube construction thread completed, the launcher spawned another thread for the next cube and added it to the pool. The launcher program used a thread barrier, such that the code would only continue once all the simulation cubes were processed. The launcher spawned threads for the cubes in a reverse order along the lightcone's path in order to reduce the amount of waiting time at the barrier. An assumption was made that those cubes will be tiled to accommodate the lightcone's larger diameter at high redshift. Thus these threads would process more particles, resulting in a longer runtime. Figure 4.6 illustrates the execution of the lightcone construction code. This software architecture resembles a simple First In First Out (FIFO) queue.

In this example, six simulation cubes were required to create the lightcone. The user specified a maximum of four concurrent threads. Therefore the furthest four cubes (green) started immediately, while cube00 and cube01 (orange) started when a thread became available. After all the threads were completed, the postProcessing.py program was called by the launcher. This code 'stitched' together all the lightcone sections produced by each of the worker threads in the thread pool.

On a simulation cube level, the lightcone construction process was completely self contained.

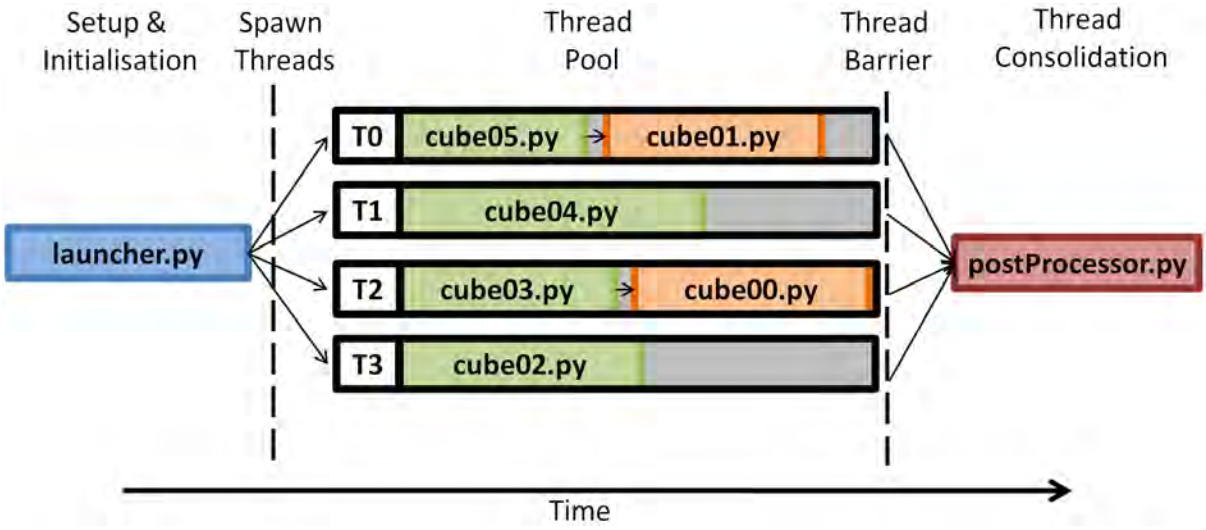


Figure 4.8: A sequence diagram describing the multi-threaded approach implemented for the lightcone construction code. The "launcher.py" program performs initialisation and spawns a pool of four threads. Threads operate on lightcone subsections, the maximum number of concurrent threads is define by the user. After the thread pool completes, the "postProcessor.py" program stitches together the resulting lightcone subsections.

Each cube required its own inputs, and produced its own output. Furthermore, because the resulting lightcone is stitched together after the thread pool completes, no mutual exclusion of shared memory was necessary. Figure 4.9 presents the Input/Output digram of a single cube instance. The cubeID is provided by the launcher, after which the thread is entirely self contained.

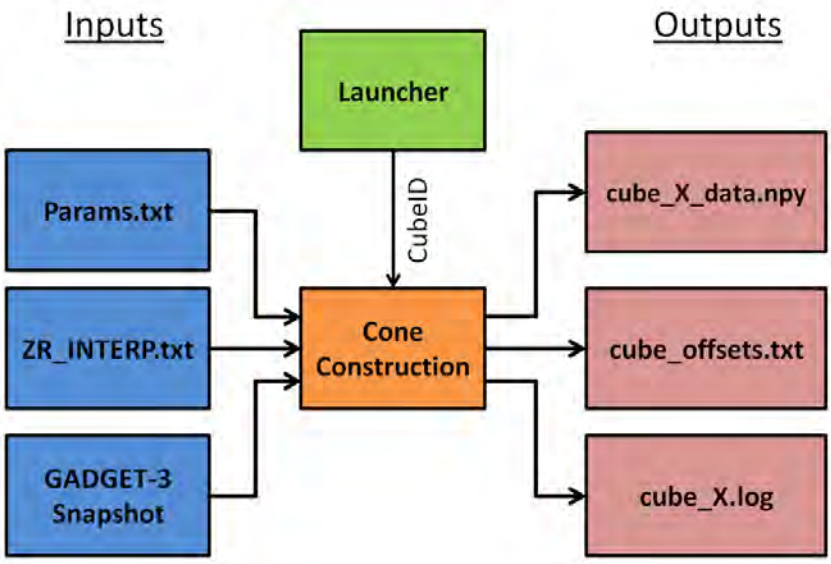


Figure 4.9: A flow diagram showing the inputs and outputs of a single lightcone construction instance.

The threads takes in inputs from the parameter file ('params.txt'), the interpolation data point set ('ZR_INTERP.txt') and simulation data from a GADGET-3 snapshot. The code then produces a log file ('cube_X.log') while it produces a subsection of the greatly lightcone data structure ('cube_X_data.npy'). The array offset of this particular subsection is stored to file ('cube_offsets.txt').

### 4.2.3 Algorithm Optimisations

Upon producing the first working solution of the lightcone construction code in Python, the runtime was found to be unsatisfactory for larger datasets. This was a concern, as the code was developed to be scalable for producing larger lightcones for other potential observations, such as LADUMA, in the future. A process of improving the performance of the code was conducted to increase its efficiency and reduce post-processing runtime. A number of specific subroutines and code implementations were explored to provide the best software solution. The focus areas included:

- Reducing the number of costly function calls required to process a lightcone section.
- Managing NumPy arrays to reduce the memory footprint of the lightcone code.
- Selecting the most efficient SciPy interpolation function for converting co-moving distance to redshift.
- Vectorising the code to improve hardware utilisation and dramatically reduce runtime.

#### 4.2.3.1 Costly Function Calls

The first and simplest technique explored was to streamline the Python code so fewer instructions were required to process the simulation data. Primarily, this involved reducing the number of calls to the 'angle_between()' function within the code. This function took in two 3D vectors, normalised their length, and calculated the angle between them using a dot product. This function was used extensively during the particle testing and particle binning phase of lightcone construction.

The 'angle_between()' function proved to be computationally expensive during the particle testing process. The most costly subroutine within this function was found to be the normalisation of the 3D vector. In addition to reducing the number of calls to this

function, an investigation was conducted into improving the efficiency of the function. Several Python and NumPy subroutines were compared to determine which provided the most efficient implementation of the costly normalisation operation used in this function.

A more intelligent particle testing mask was also implemented. The volume of the lightcone within the simulation cube was calculated before particle testing began. This was done by determining the radius of the cone at the entry and exit points of the cube. If the volume of the lightcone did not break the bounds of the cube, tiling was not required. However, if a bound was broken, as depicted in Figure 4.3, a duplicate particle array was created with a set of shifted co-ordinates. All of the duplicated particles required testing, as the particles were randomly distributed within the volume. This approach of selectively tiling a cube, rather than blindly tiling in all direction greatly reduced the number of particles tested. The results of this approach are visible in the structure of stacked cubes, presented in Figure 4.4.

#### 4.2.3.2 Memory Management

A concerning code memory footprint was observed when testing the code on larger problem sizes. Memory usage increased when cubes were tiled to accommodate the lightcone's increasing radius. Thus, at high redshifts (towards to end of the lightcone) the cubes are often tiled multiple times (up to 8 times in extreme cases). The memory consumption of the code increased relative to this tiling, with threads using up to 6 GB of RAM. This posed less of a concern on the ACE Lab cluster nodes which contain 128 GB of memory. However for desktop or workstation computers, this severely limited the possible number of concurrent threads. Focus was therefore placed on managing memory during the lightcone construction process. Arrays containing particle information of cubes were created and held in memory only for as long as required. After which, the memory was released for use by the next tiled simulation cube. This reduced the memory footprint of the test case cube from 3.8 GB to 2.4 GB. Several test were conducted which probed the memory usage of the code for a variety of lightcone pixelisation resolutions, these results are presented in Figure 5.21.

#### 4.2.3.3 Interpolation Functions Comparison

The redshift of the particle from the Observer was required during the particle processing procedure. Calculating the redshift of a particle from its co-moving distance is not

a straightforward process, as these two properties are not directly related. Thus an interpolation function was used. An interpolation function provides the ability to create a mapping between two datasets using a series of discrete data points. A set of five million distance-redshift pairs was used to generate a forward and reverse linear interpolation function. SciPy provides two such implementations; the Univariate Spline API and the 1D interpolation function 'interp1d'. The performance of these two alternatives was investigated.

#### 4.2.3.4 Code Vectorisation

The final step in optimising the lightcone construction code provided the largest gain in performance. After an efficient algorithm had been developed and tested, vectorisation of the Python code was explored. NumPy provides powerful multi-dimensional array functionality. This allowed for array-wise operations to be performed on large datasets, rather than operating on individual elements within an iterative loop. The high performance back-end to NumPy could be fully utilised by using these array-wise operations. This resulted in code executing efficiently on the complex micro-architectures of modern processors. The vectorisation process allows the processor to fill the data pipeline with multiple operations and execute several instructions per cycle. The Intel IvyBridge architecture CPUs used in the ACE Lab cluster support the 256 bit AVX instruction set, allowing for the operation of 8 single precision floats simultaneously.

While vectorisation promised significant performance benefits, it also had several drawbacks. Firstly, the syntax required to perform multi-dimensional array operations often proved complex and problematic to troubleshoot. A greater obstacle observed with code vectorisation was dealing with conditional operations. As the vectorisation process generates a set of unconditional instructions for the hardware, the standard iterative approach to conditional testing within an array was not possible. A masking array was required to select a subset of the larger data structure. The process of adapting the existing algorithm to fit within the limits imposed by vectorised operations was not trivial and proved to be time consuming. However, as seen in the results section, the benefits observed from vectorisation were well worth the effort.

```
partArray # array full of particles in the lightcone

#get datacube indices of particles
xyz = coords(partArray)

#multiple particle can map to single index

sortedCoords = lexicalSort(xyz)

#remove and accumulate repeated co-ords

assign accumulated values to datacube
```

## 4.3 Conclusion

This chapter provided specific details for the lightcone construction code. A description was given of how the GADGET-3 simulation volume stacking method was implemented. Furthermore, an explanation was provided on how this implementation differs from existing techniques such as the Obreschkow approach. The axis along which the lightcone was constructed relative to the simulation volume (referred to as a cube) could be specified by the user. A procedure of cube stacking and tiling was implemented in order to ensure that the volume of the lightcone was kept within the bounds of the simulation cubes.

Details were also provided for the lightcone pixelisation process which was used to condense the millions of particles found within the lightcone into a fixed grid structure. This process was performed in order to more accurately represent the contents of the lightcone, as seen by a radio telescope. This discrete binning of simulation particles also allowed data to be stored in a more traditional data structure. This data structure was discussed in depth, with a motivation provided to support the use a dense NumPy array.

The second half of this chapter discussed the several important optimisations performed on the Python code and NumPy libraries. Significant effort was placed on producing the most efficient code possible for rapid construction of the lightcone. This was achieved through the use of optimised math libraries and multi-threaded programming. In addition, the code itself was improved through optimisations of costly subroutines and vectorisation

of the particle testing and binning procedures.

A selection of results obtained from the lightcone construction code and Python optimisation documented in this chapter are presented next in the Results.

# 5 Results

This chapter presents a selection of results obtained from the GADGET-3 evaluation, Python code optimization and lightcone construction process. Section 5.1 provides a collection of performance benchmark results from the GADGET-3 evaluation. A number of software optimisations were performed in order to improve the efficiency of the simulation code. This evaluation was conducted to support other GADGET-3 users running the code on HPC systems. Focus was thus placed on optimising the software environment, compilers and libraries upon which GADGET-3 relies. A comparison of software compilers, code implementations and runtime scaling provides insight into the ideal software configuration for GADGET-3.

Section 5.2 continues with results from a thorough investigation into the Python and NumPy performance improvements detailed in the previous chapter. These optimisations were implemented to reduce the hardware requirements of the post-processing code as well as to improve the execution time. The efficiency of the Python code was optimised over a number of stages. The code was first streamlined to reduce the number of function calls and thus CPU workload. The NumPy package was linked to a high performance math library to boost the performance of linear algebra computation. In addition, several subroutines were analysed to determine which performed best in the given context. A multi-threaded implementation of the code was developed to utilise a greater portion of a modern SMP computer's memory bandwidth. Finally, the code was vectorised to produce a set of unconditional matrix operations. Operations on large datasets are more efficiently mapped to the complex micro-architecture of modern processors. Thus, a significant performance increase was observed for the lightcone construction code.

Upon developing a program which efficiently produces a lightcone from simulation data, a number visualisations were generated in order to validate the results. Section 5.3 presents these results to support the lightcone code developed for this research. A mass function is presented which shows the composition of the lightcone's volume. Several other statistical measurements are included, such as a plot of line-of-sight peculiar velocities. In the next

chapter, these results will be compared with existing research which was introduced in the literature review.

## 5.1 GADGET-3 Performance Evaluation Results

The first set of results presented for the GADGET-3 performance evaluation concern algorithm modifications within the code. During the development process of the code a new Friends-of-Friends (FoF) algorithm was introduced which greatly improved the performance of the code. A FoF algorithm is used to identify clumps of particles into gravitationally bound groups. The velocity and proximity of the particles is used to identify these groups. These simulation benchmarks were constructed to operate on a fixed number of timesteps. A timestep represents the progression of time within the simulation and describes the temporal resolution of the simulation. Additional information on the GADGET simulation can be found in Section 2.4. The time taken to process a timestep is determined to the complexity of particle interactions which occur in that timestep. Therefore, using a fixed selection of timesteps is necessary to perform a fair comparison.

A test case comprising approximately 3500 timesteps was constructed to compare the performance of these two FoF algorithms. Figures 5.1 and 5.2 provide a comparison between the runtime performance achieved by the April and June releases of GADGET-3.

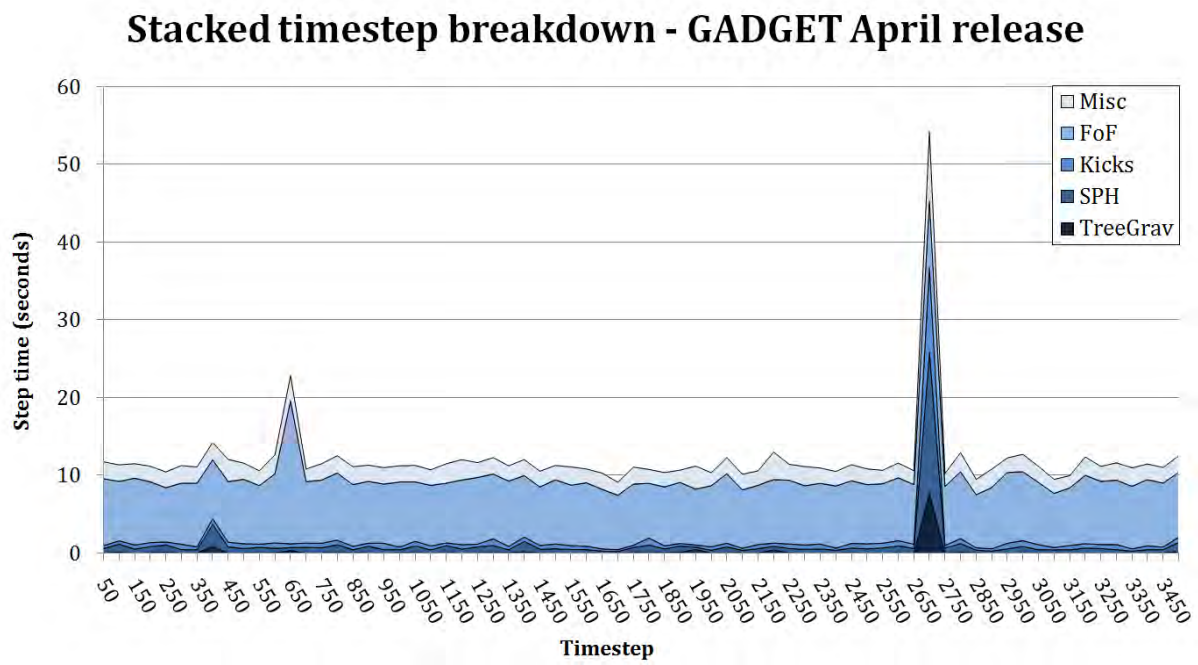


Figure 5.1: A stacked timestep graph for the April release of GADGET-3. The graph shows the breakdown of CPU time allocation between the code's subroutines.

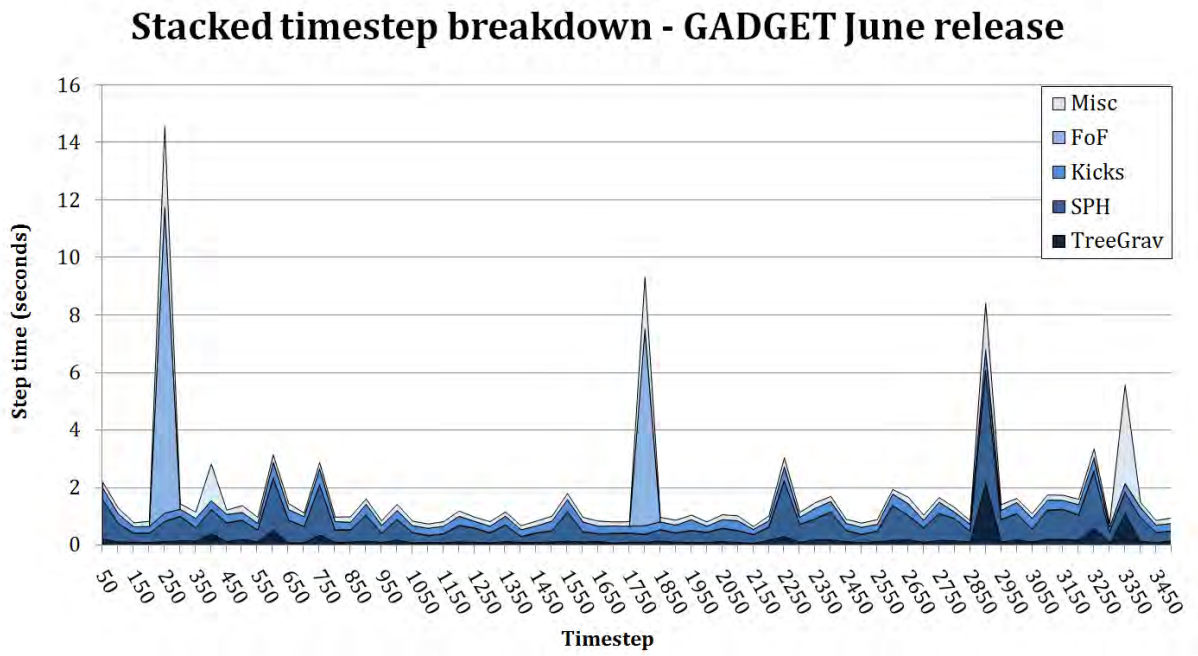


Figure 5.2: A stacked timestep graph for the June release of GADGET-3. The graph shows the breakdown of CPU time allocation between the code's subroutines.

Figures 5.1 and 5.2 present a set of stacked graphs of CPU time consumption between the FoF, Smoothed Particle Hydrodynamics (SPH), TreeGrav (far field N-body), particle

momenta kicks (Kicks) and miscellaneous processing tasks. It is import to note the different time scales on the vertical axis between the two graphs. It can be seen in Figure 5.1 that the FoF algorithm is dominating the time spent for each timestep of the April release, resulting in an average step processing time of 12.07 seconds. At approximately step number 2700 in the test, the PMGrid function is called for far field particle interaction - hence a spike is observed in total execution time. The June release of GADGET-3, shown in Figure 5.2, performs significantly better, with an average processing time per step of 1.76 seconds. This is primarily due to the costly FoF function being executed in only two of the 3500 timesteps. This change to an improved implementation of the FoF algorithm resulted in an average overall speedup of 686%.

In order to further evaluate the result of the newer FoF algorithm, a fractional time allocation chart was generated. Figures 5.3 and 5.4 provide a comparison between the fractional runtimes achieved by the April and June versions of GADGET-3 using the above test case.

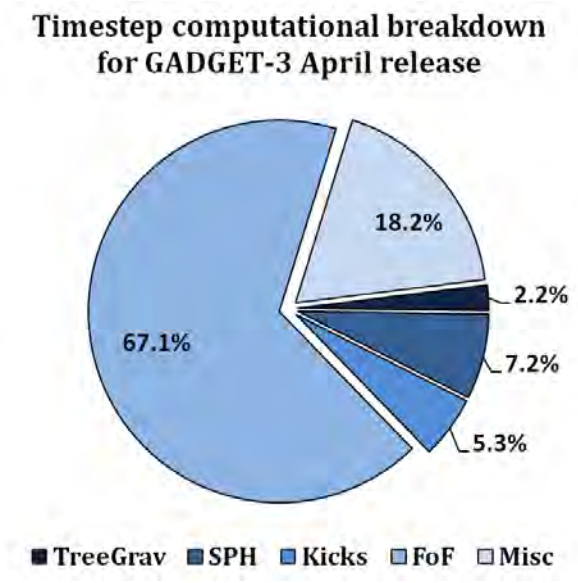


Figure 5.3: A fractional representation of CPU time spent per timestep for the April release of GADGET-3, averaged over the 3500 timestep test case simulation run.

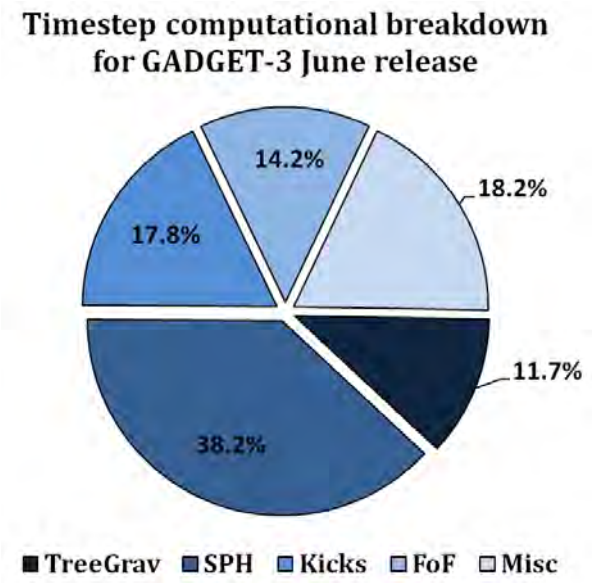


Figure 5.4: A fractional representation of CPU time spent per timestep for the June release of GADGET-3, averaged over the 3500 timestep test case simulation run.

Figure 5.3 shows the average time breakdown of the 3500 simulation steps for the April release. On average, 67.1% of the CPU's time is spent executing the FoF algorithm. The more computationally intensive tasks; SPH, particle momenta kicks and gravity tree components consume only 14.7% of the total CPU time on average. Figure 5.4 illustrates the improvements made in June release; the FoF method only consumes 14.2% of the CPU time. This provides more CPU time for the other tasks. This is an example which

illustrates that the most effective optimisation is often achieved at a application level and not system level.

The next component under investigation was the choice of software compiler within the HPC cluster environment. The open source GCC and proprietary Intel ICS compilers were compared in order to determine what performance benefits existed from selecting one over the other. A new test case was constructed which operated on a 16 $Mpc^3$ simulation over a limited redshift range. The result for each benchmark was generated by averaging three separate simulation runs, the standard deviation of these runs is shown by error bars. Figure 5.5 presents the results of this compiler comparison.

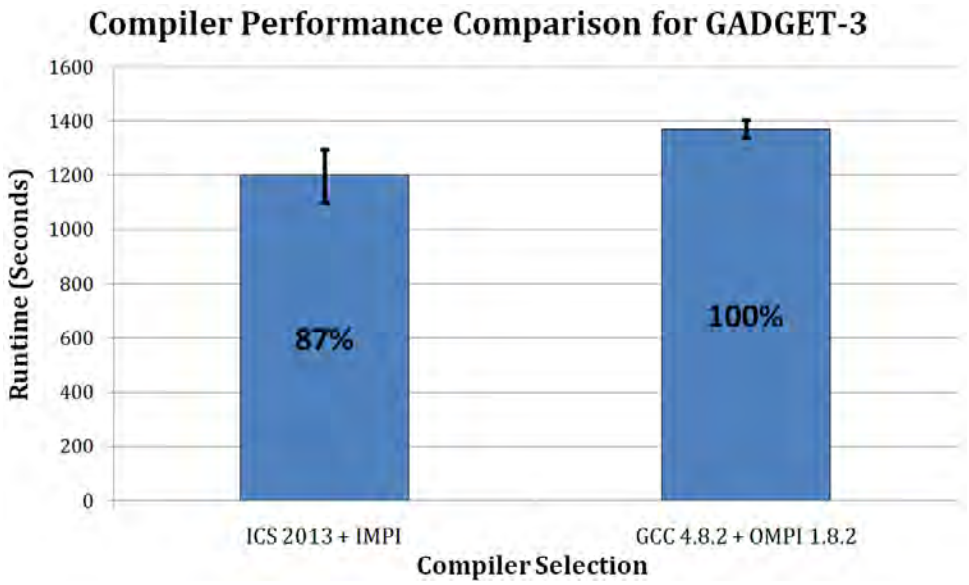


Figure 5.5: A compiler comparison for GADGET-3, showing the relative benchmark runtime for GCC and Intel Compiler Suite.

It can be seen from Figure 5.5 that the Intel Compiler Suite alternative provided a reasonable performance increase over the GCC compiler. ICS compiled code completed the benchmark 13% faster on average compared to the GCC code. The ICS code does however show a larger variance in its runtime over the three benchmark runs. In order to further explore the performance advantage provided by Intel's ICS software, an analysis was conducted using Ganglia; a hardware monitoring software. Figure 5.6 and Figure 5.7 provide some insight into a possible cause of the performance difference observed.

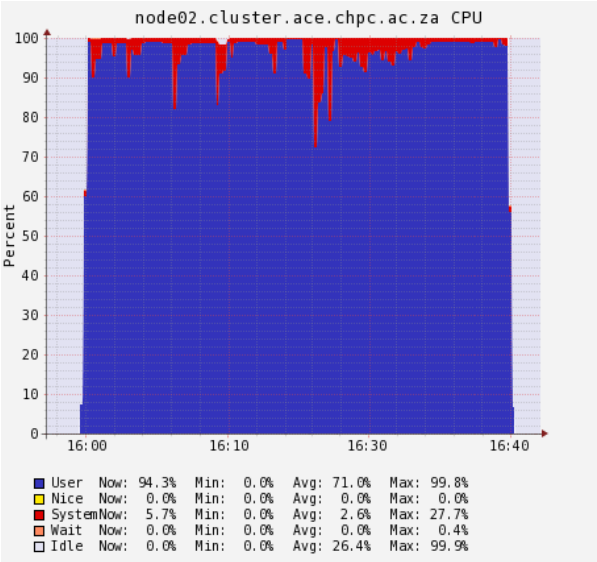


Figure 5.6: A CPU utilisation report generated by Ganglia. This data was extracted from the compute node during a benchmark of GADGET-3 compiled with GCC 4.8.2.

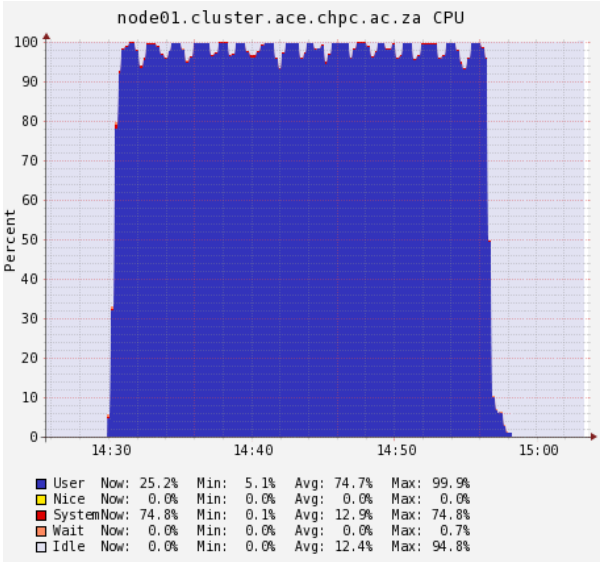


Figure 5.7: A CPU utilisation report generated by GangliaThis data was extracted from the compute node during a benchmark of GADGET-3 compiled with ICS 2013.

Figure 5.6 shows the CPU utilisation graph of a single node during the benchmark of GADGET-3 which was compiled with GCC. The red component of the CPU load graph represents system time, where CPU cycles are effectively wasted on non-application tasks. As can be seen, a portion of CPU time was being wasted in the GCC case. Figure 5.7 shows less system time for code compiled with ICS, which indicates more efficient code compilation.

Focus was next placed on evaluating the scaling efficiency of the GADGET-3 code. The code was expected to scale well, as the simulation was developed specifically for distributed computing systems. However, this was not the case. Initial testing of GADGET-3 was conducted with a cluster of virtual machines. The code was tested on both virtual and physical clusters to determine if the the virtual machines performed poorly. Figure 5.8 presents the results of a runtime scaling test conducted on both a cluster of virtual and physical machines. The same test case was executed on an increasing number of compute nodes to investigate the resulting execution time speedup.

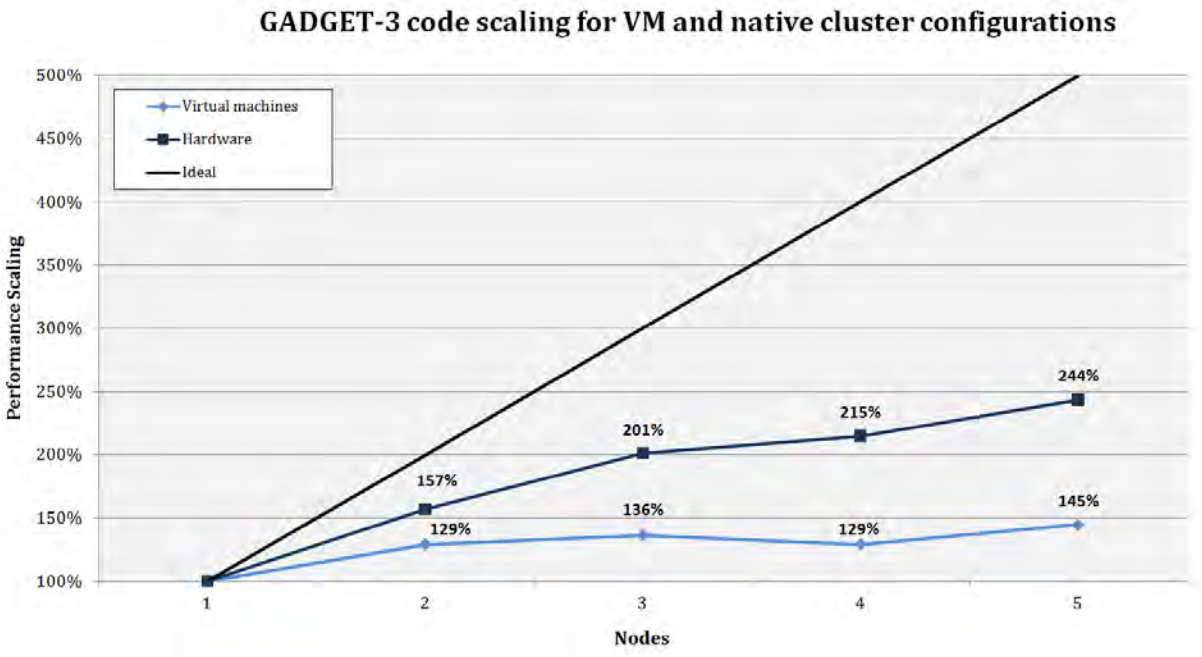


Figure 5.8: A graph showing the observed scaling of a GADGET-3 test problem on multiple compute nodes for both virtual machines and native hardware clusters. The black line represents ideal speedup.

Figure 5.8 shows a poor runtime scaling for both the hardware and virtual cluster configurations. The cluster of physical machines did however complete the benchmark 58% faster than the virtual cluster using five compute nodes.

An example of the MPI command used to execute the simulation within the HPC cluster is provided below.

```
mpirun -np 100 -hosts n01,n02,n03,n04,n05 ./Gadget3 sr16n128.p
```

In this example, the code is run of five hosts, each of which contain 20 CPU cores, resulting in a total core count of 100. The linking of the GADGET-3 executable 'Gadget3' and parameter file 'sr16n128.p' is also displayed.

The hardware still only provided a 244% performance improvement, considerably worse than ideal performance scaling. Figure 5.9 presents the results as a fraction of the ideal speedup.

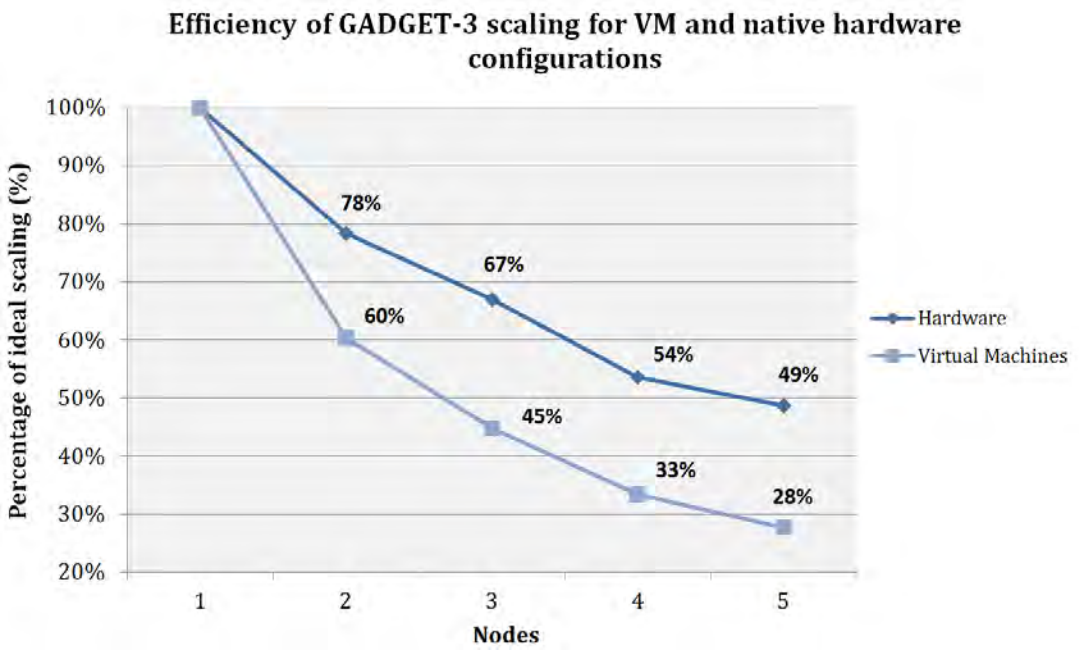


Figure 5.9: The relative efficiencies of virtual and native cluster configurations observed when scaling the code across multiple compute nodes.

Figure 5.9 shows the relative performance efficiencies of the physical and virtual clusters when running GADGET-3 across multiple compute nodes. The efficiencies were calculated using the measured runtime as a fraction of ideal execution runtime. The performance delta observed between the two configurations increases with the number of compute nodes assigned to the benchmark. This suggests a communication overhead associated with virtualisation may be responsible for the performance delta. Further investigation is required to uncover the cause of this virtualisation performance overhead. When executing on five compute nodes (100 CPU cores), the hardware cluster operated nearly twice as efficiently as the virtual cluster, achieved a scaling efficiency of 49%, compared to 28%. The cause of this generally poor scaling is discussed further in Chapter 6.

The final test conducted during the GADGET-3 performance investigation was to evaluate the performance benefits which may exist from implementing Intel's advanced compiler optimisations. These features are enabled via optimisation flags during compilation of the GADGET-3 code. Table 5.1 presents a number of compiler optimisation flag options which were tested using the same GADGET-3 problem as before. For each test case, the code was recompiled and the resulting binary file size recorded. The code's execution time on five compute nodes was then documented.

Table 5.1: ICS Optimisation Flag Comparison for GADGET-3

| **Optimisation** | **Compiler Flags** | **Binary Size (KB)** | **Execution Time (s)** |
|---|---|---|---|
| Optimisation off | N/A | 1336 | 750 |
| Optimisation level 1 | -O1 | 1336 | 720 |
| Size optimisation | -Os | 1292 | 515 |
| Optimisation level 2 | -O2 | 1440 | 494 |
| Optimisation level 3 | -O3 -msse4 -opt-prefetch -unroll-4 -M64 | 1560 | 485 |

Notably from Table 5.1, disabling all compiler optimisations resulted in a 50% performance loss. By default, GADGET-3 implements the -02 level of optimisation, which provides various optimisations while not being overly aggressive with code re-factoring. The level 2 optimisation provided a near optimal runtime for the sample problem, however further improvement was possible by enabled additional flags. Level 3 optimisation provides more aggressive loop transformations as well as specific optimisations for AVX and SSE4 instruction set compatible micro-architectures [80].

## 5.2 Lightcone Code Optimisation Benchmarks

This section presents the results obtained from various optimisations conducted on the Python lightcone construction code as well as the math libraries on which it relied. While the first implementation of the code was functional, it was too slow to perform all but the smallest problem sizes. The first task in improving the code's performance was to reduce the computational workload by removing unnecessary function calls and redundant operations.

The 'angle_between()' function was initially called multiple times for each particle being tested to determine its position relative to the lightcone. Figure 5.10 presents the effects of reducing the number of calls to this function.

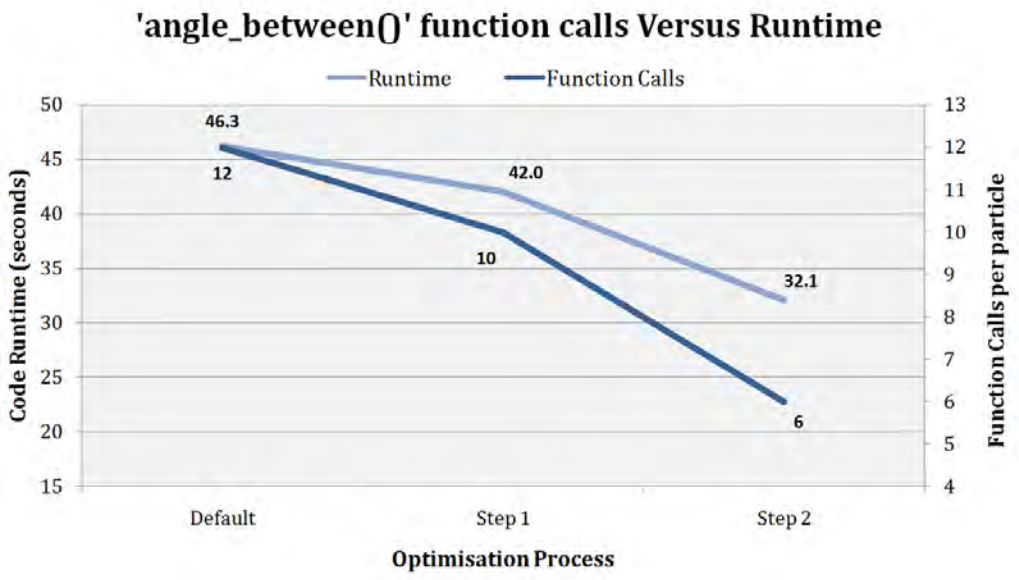


Figure 5.10: A graph of relative runtime compared to number of 'angle_between()' function calls.

Figure 5.10 shows that initially the function was called 12 times per particle. This number was decreased by using temporary variables and by modifying particle binning procedure. The first attempt at optimising the algorithm reduced this number to 10 calls, with the final version reducing the number further to 6 function calls. By halving the number of 'angle_between()' function calls, the overall code runtime was reduced by 31%. Analysing this result, the percentage of total execution time spent performing this single function dropped from 61% to 44% after these optimisations.

The notable performance improvement observed by reducing the number of function calls motivated an investigation into the performance of the 'angle_between()' function itself. A comparison was conducted in order to determine to most efficient implementation of the costly vector normalisation performed within the function. Four methods were tested; the standard NumPy norm API, the NumPy square root API with standard Python square function, the math.sqrt with math.pow and finally the math.sqrt with standard python square. Figure 5.11 presents the results of this normalisation test using 2000 random 3D vectors. Error bars represent the standard deviation recorded between four repeated tests.

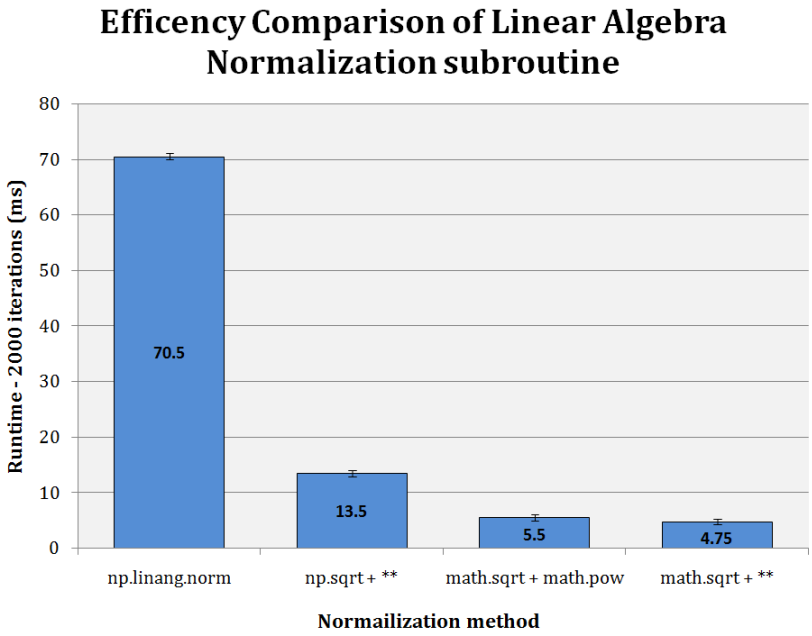


Figure 5.11: A relative runtime comparison of four Python 3D vector normalisation implementations.

Surprisingly, Figure 5.11 shows the NumPy API for linear algebra normalisation is the most inefficient implementation. The 'math.sqrt + **' implementation proved to be the fastest approach; completing the 2000 vectors almost 15 times faster than the NumPy API.

An investigation into memory allocation was conducted. A vast number of particle position need to be remapped due to the tiling of simulation cubes which occurs during the lightcone construction process. There were two alternate approaches of implementing this remapping; either via recalculating the position for each particle as its processed, or by calculating all of the remapped co-ordinates before testing began. The second method had the disadvantage of requiring more memory as all remapped positions resided in

RAM simultaneously, as opposed to generating new co-ordinates one at a time. However, by using a streamlined array operation, the pre-allocation method promised to be faster. A test was constructed to test the two methods using multiple problem sizes. Table 5.2 presents the results of this comparison.

Table 5.2: Runtime comparison of Pre-Allocation vs. Iterative particle testing methods.

| | **Runtime (s)** | | | |
|---|---|---|---|---|
| Problem Size | 10k | 100k | 1M | 10M |
| | **Pre-Allocation Method** | | | |
| Memory Allocation | 10.8 | 10.8 | 10.8 | 10.8 |
| Particle Testing | 2.7 | 22.1 | 220.5 | 2176.5 |
| Total Time | 14.8 | 37.0 | 256.8 | 2403.1 |
| | **Iterative Method** | | | |
| Particle Test time | 13.1 | 128.3 | 1271.3 | 12813.1 |
| Total Time | 18.2 | 163.2 | 1605.8 | 16086.8 |

From Table 5.2 it can be seen that the pre-allocation method, labelled 'Pre-Allocation Method', provided a significant performance advantage over the 'Iterative Method'. These results are displayed graphically in Figure 5.12 for better interpretation (note the logarithmic runtime axis).

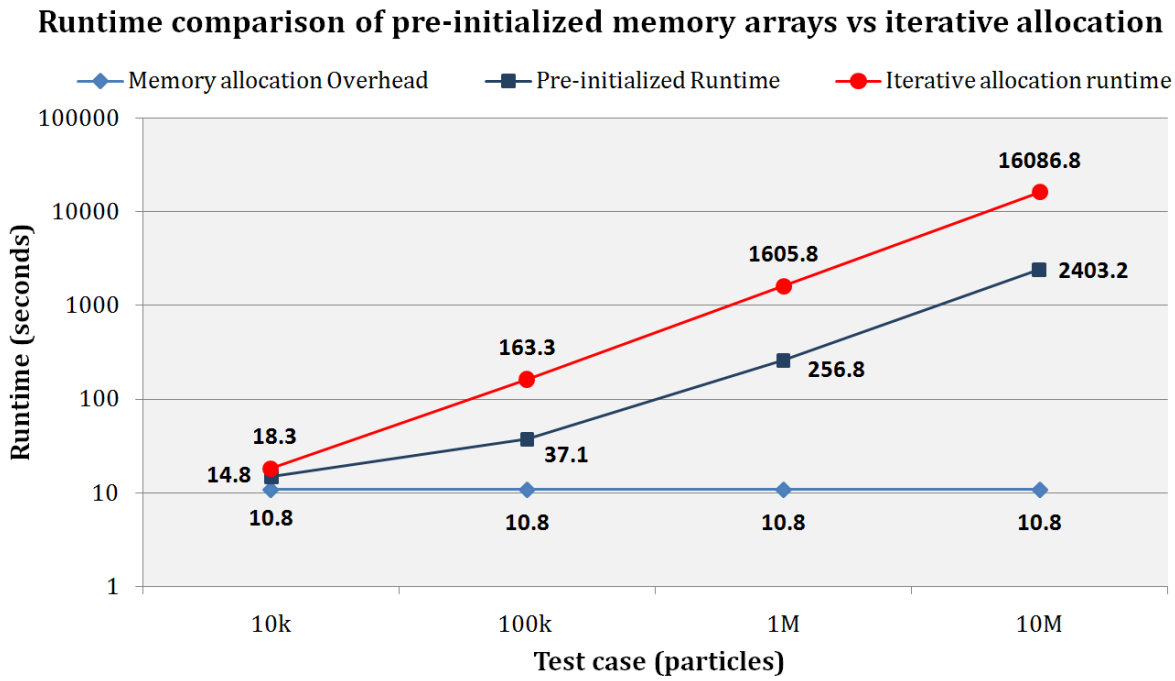


Figure 5.12: A graph comparing the pre-initialised memory allocation and iterative co-ordinate remapping approaches. The red line represents the total time taken for the iterative method. The dark blue line shows the total remapping time for the pre-allocation approach, with the light blue line representing the memory allocation overhead of remapping co-ordinates.

It can be seen that the memory pre-allocation method had a fixed overhead for generating the remapped co-ordinates in memory prior to execution. This operation was conducted using a NumPy vectorised array operation and as such was particularly efficient. For the test cases of one million and ten million particles, the total runtime of the pre-initialisation solution was almost six times faster than the iterative version. Although the faster method consumed slightly more memory, the resulting decrease in runtime made it the favourable option.

An investigation was conducted of the two interpolation functions available in the SciPy Python package. The Univariate Spline approach performs a smoothing function on the dataset in order to reduce interpolation time. The interp1d method uses a more basic approach with a faster initialisation time. The two functions are implemented in different ways and thus are more suited to specific tasks. In addition, a comparison was performed between various file I/O operations. This investigation was conducted to improve the time required to read in the large file of 5 million discrete sample points for the interpolation function. Figure 5.13 presents the I/O function comparison while Figure 5.14 illustrates the results of the interpolation test.

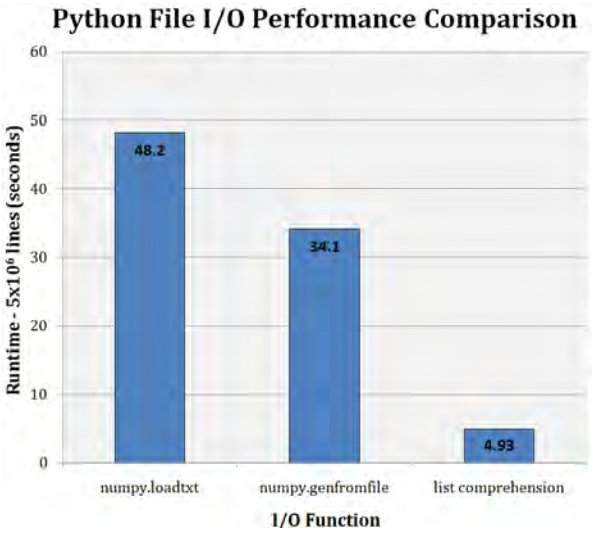


Figure 5.13: A graph showing the relative execution times of three Python text based I/O functions. A file containing 5 million pairs of floats was read from disk in this test.

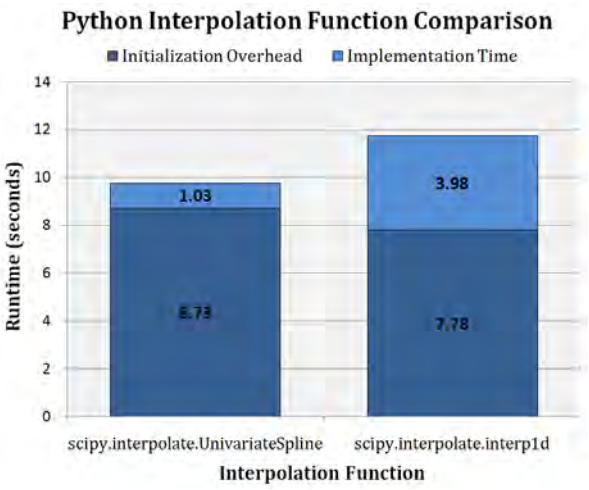


Figure 5.14: A graph comparing the performance of two SciPy interpolation functions. The dark blue bars represent the initialisation time, while the light blue bars show the interpolation time for 100,000 random samples.

The performance comparison shown in Figure 5.13 presents interesting results. The highly efficient NumPy functions 'loadtxt' and 'genfromfile' both performed worse than a simple list comprehension implemented in standard Python code. This looped implementation performed 9.8 and 6.9 times faster compared to numpy.loadtxt and numpy.genfromfile respectively. The UnivariateSpline function demonstrated a slightly slower initialization time than the simpler interp1d method, however it provided a more efficient interpolation once constructed. Therefore, this method was preferred due to the large number of redshift interpolations required for lightcone construction.

At this point of the investigation, the performance of NumPy had come under question. There is the option to link to a non-standard Basic Linear Algebra Subprograms (BLAS) library during the installation of this Python package. A benchmark was constructed in order to test the efficiency of basic NumPy math operations. Three instances of NumPy were installed using different BLAS libraries; the standard (or 'vanilla') NumPy, an Intel Math Kernel Library (MKL) version, and an ATLAS implementation. A selection of generic linear algebra operations were selected to provide a fair comparison between the three BLAS implementations. The tests included calculated: matrix eigenvalue, matrix inversion, single value decompositions, matrix determinant and matrix dot product. For all test cases, a 2D NumPy array containing 1 million randomly generated floats was processed. Figure 5.15 presents the benchmark results.

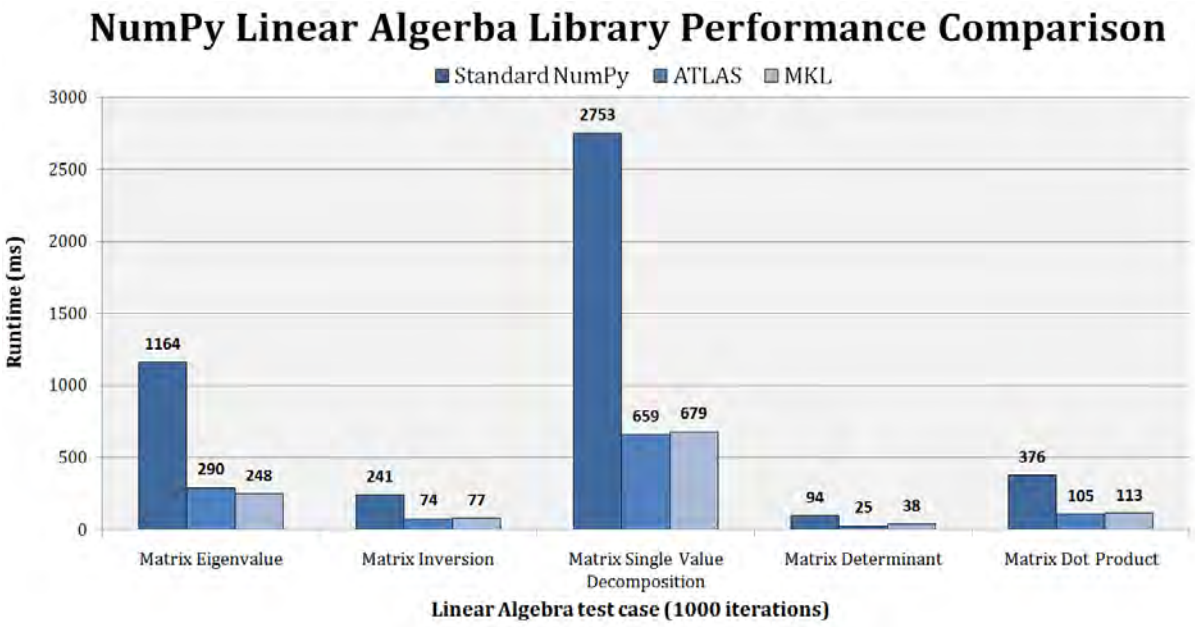


Figure 5.15: A linear algebra performance evaluation of Intel MKL (light blue) and ATLAS (blue) BLAS implementations, compared to standard NumPy (dark blue) subroutines. A set of five linear algebra test cases were selected to provide a comparison for a variety of conditions.

Figure 5.15 illustrates the performance benefits of utilising a high performance linear algebra subroutine for NumPy. Both Intel MKL and ATLAS perform significantly better than the standard NumPy by taking advantage of architecture specific optimisations. ATLAS performed best in four of of the five tests, with Intel MKL besting it in the eigenvalue benchmark. However, in the majority of cases both third party libraries produced similar results. Linking to another BLAS library provided a performance increase of between 246% and 468%, with the improvement being dependent on the nature of the mathematical operation.

Several verifications of the code were performed to ensure that CPU time was being used efficiently. A test was conducted which compared the number of particles tested to the overall program runtime to identify any overheads. The test was conducted by varying the pointing vector along which the lightcone was constructed. This test also provided information on what effect altering this vector had on overall post-processing runtime. Figure 5.16 presents the comparison of particles tested to runtime.

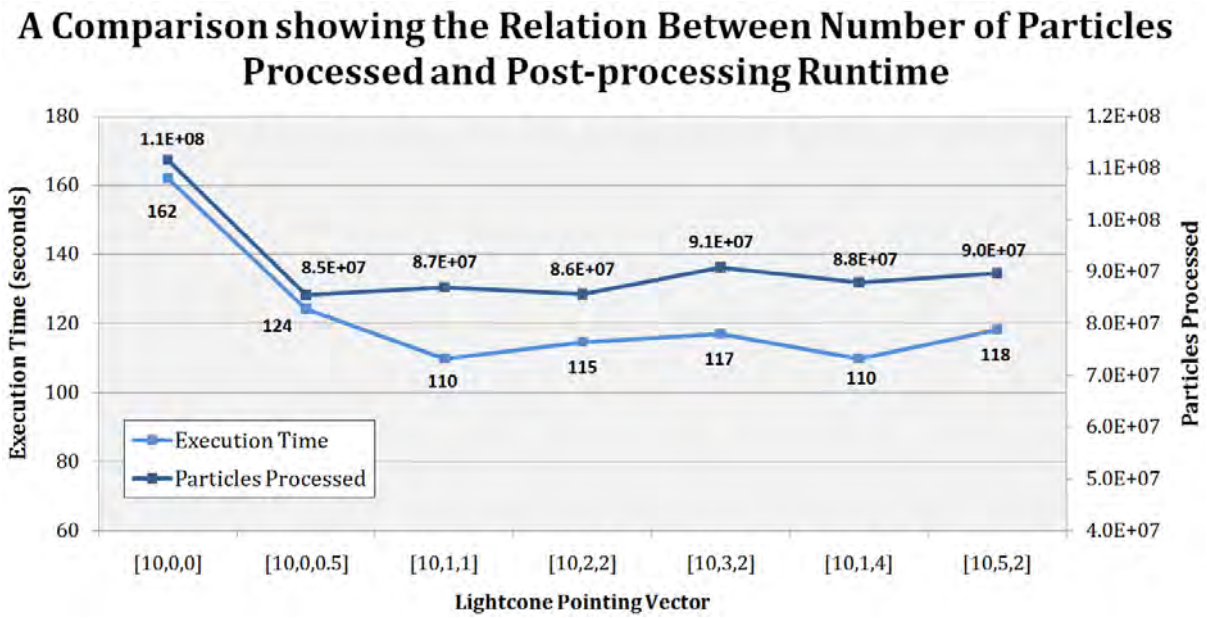


Figure 5.16: A graph comparing the number of particles tested by the lightcone construction code, with the overall runtime of the application. The pointing vector was skewed to produce seven different lightcone realisations, sampling a different number of particles.

It is evident from Figure 5.16 that there is a positive correlation between number of particles tested and the execution time of the code. In the test case using pointing vector [10,0,0], 110 million particles were tested in a time of 162 seconds. In contrast, the test case using a vector [10,2,2], 22% fewer particles were tested, with a corresponding runtime decrease of 29%. This result suggests that the code has a reasonably linear relationship with number of particles tested.

A multi-threaded code implementation was developed to utilise a greater portion of the available hardware, and thus further improve code performance. Each of the lightcone sections were independent of each other, thus the parallelisation process was simplified. Figure 5.17 presents to results of a benchmark conducted within an single SMP machine, comparing the number of allocated CPU cores with runtime.

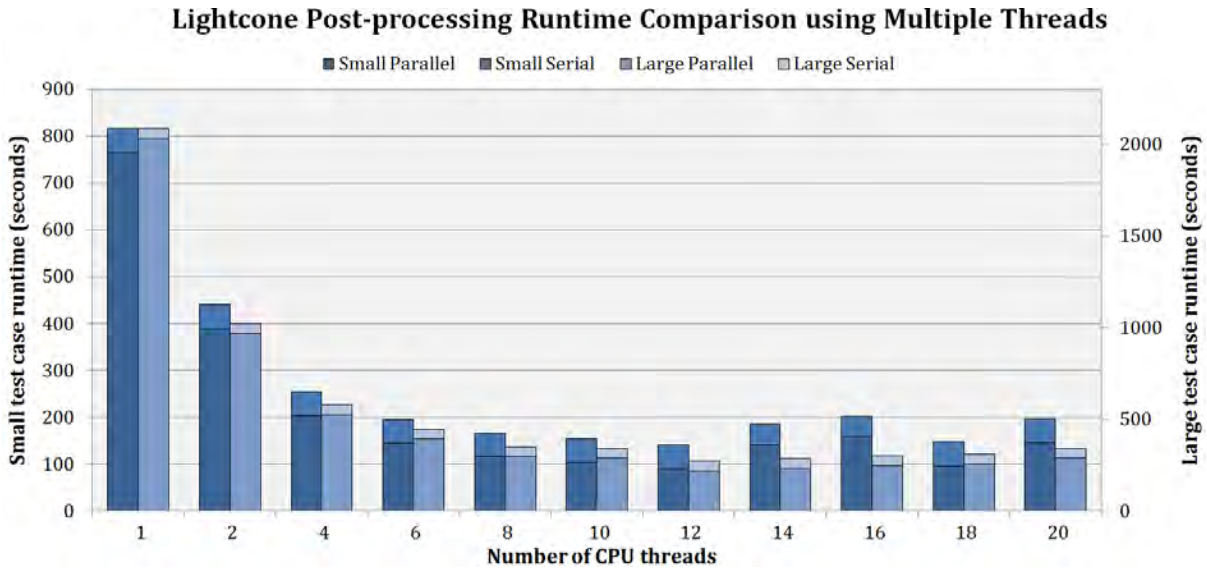


Figure 5.17: A graph showing the effect parallelism had on post-processing runtime. Two lightcone test cases were considered; one large (light blue) and one small (dark blue). In addition, the runtime of each test case is separated into its parallel and serial components.

Figure 5.17 shows the considerable runtime improvement achieved from code parallelisation. It can been seen however that there were diminishing returns for a higher number of allocated CPU threads. A 3.6 times speedup was achieved when increasing the allocated threads from one to four. However, increasing the threads further from four to eight only yielded an additional 1.7 times speedup. It should also be noted that the for larger problem size, the serial runtime component consumes a smaller fraction of the overall runtime. This suggests that the serial overhead is not dependent on the size of the lightcone being constructed.

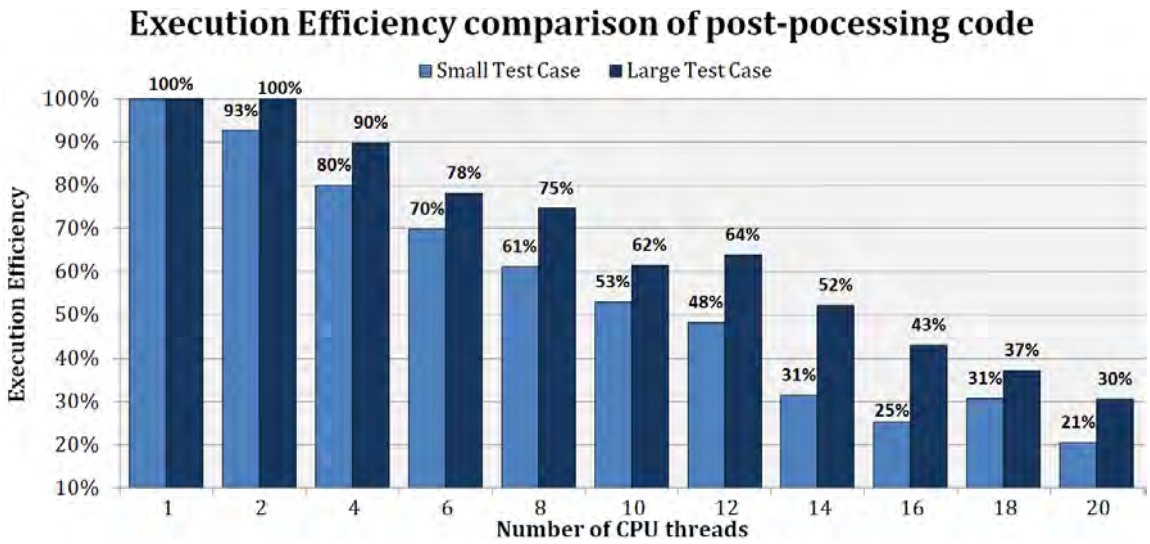


Figure 5.18: The relative runtime efficiency of a large and small lightcone test case, as a function of allocated CPU threads.

It can be seen in Figure 5.18 that the runtime efficiency consistently decreases as more are threads allocated to the post-processing code. These results represent the fraction of ideal speedup achieved by increasing CPU threads. It is evident that some factor was limiting performance of the multi-core computer. The effects of memory bandwidth are discussed in the next chapter.

A final set of tests were conducted to evaluate the significant performance increase attained by vectorising the post-processing code. A significant rewrite of the code was required to produce a full NumPy array implementation of the code. One drawback that exists with vectorisation is the need for masking arrays to handle conditional matrix operations. These additional arrays increase the total memory required by the code. Figure 5.19 below illustrates the memory requires of both the original iterative implementation of the code and the vectorised solution, while Figure 5.20 presents the corresponding decrease in runtime.

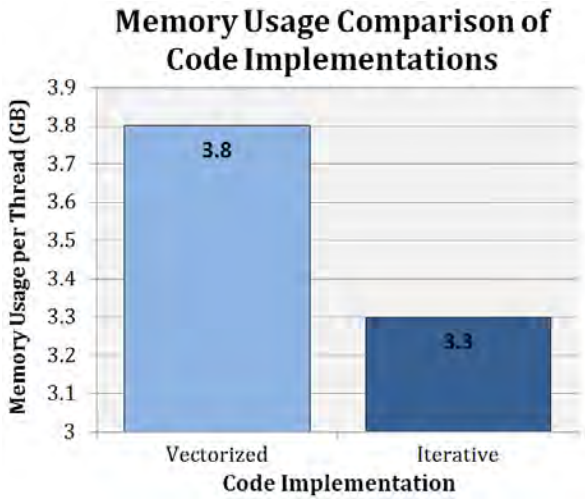


Figure 5.19: A plot showing the relative memory consumption of the iterative and vectorised lightcone post-processing implementations.

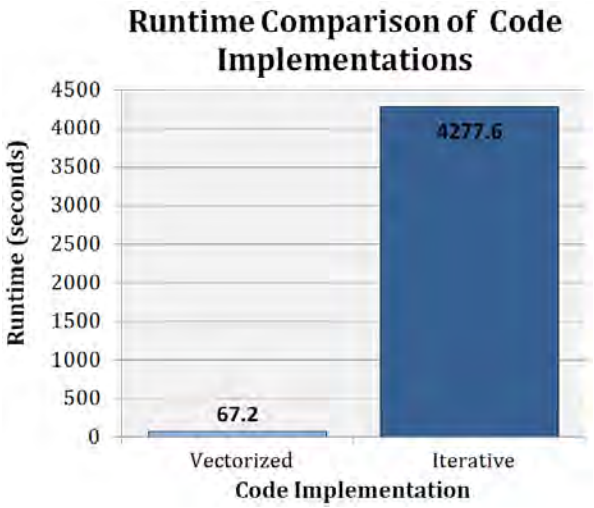


Figure 5.20: A runtime comparison of the iterative and vectorised codes, showing the immense speedup provided by vectorisation.

Figure 5.19 shows that the amount of memory consumed by the code increased by 14% as a result of vectorisation. Conversely, Figure 5.20 shows the tremendous performance gains achieved. An overall application speedup of 64.4 times was observed. This significant deduction in runtime was due to the use of highly optimised NumPy array operations for locating, classifying and binning GADGET-3 particles. This matrix operations can be interpreted and executed by the hardware far more efficiently compared to the more standard iterative approach first implemented. This massive improvement in execution time significantly altered the way in which the code could be used. Larger problem sizes and finer grained testing were made possible due to the dramatic decrease in post-processing time.

A feature of the code allows the user to alter the resolution of the pixelisation grid. This in term allows for a more fine grained analysis of the lightcone's contents. There were however implications of increasing this grid resolution. Figure 5.21 illustrates the impact altering the pixelisation grid had on memory consumption and code runtime.

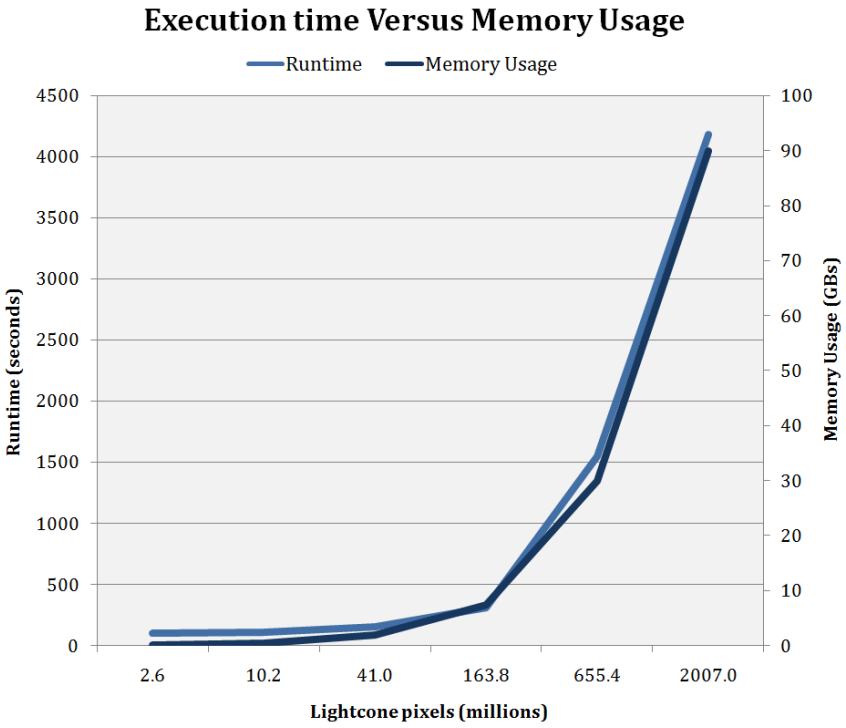


Figure 5.21: A chart illustrating the relationship between memory usage and post-processing runtime, as a function of lightcone grid size.

As the size of the NumPy pixel array grows larger, the 'stitching' process described in Chapter 4 because more time consuming. The simple task of merging large arrays from disk becomes the dominant portion of the code's overall execution time. A clear correlation between memory size and runtime is visible in Figure 5.21. In addition, it can be seen that the code required over 90 GB of memory for the highest resolution test case. Figure 5.22 shows the breakdown of parallel and serial execution times as a function of array size to further investigate the relationship between resolution and runtime.

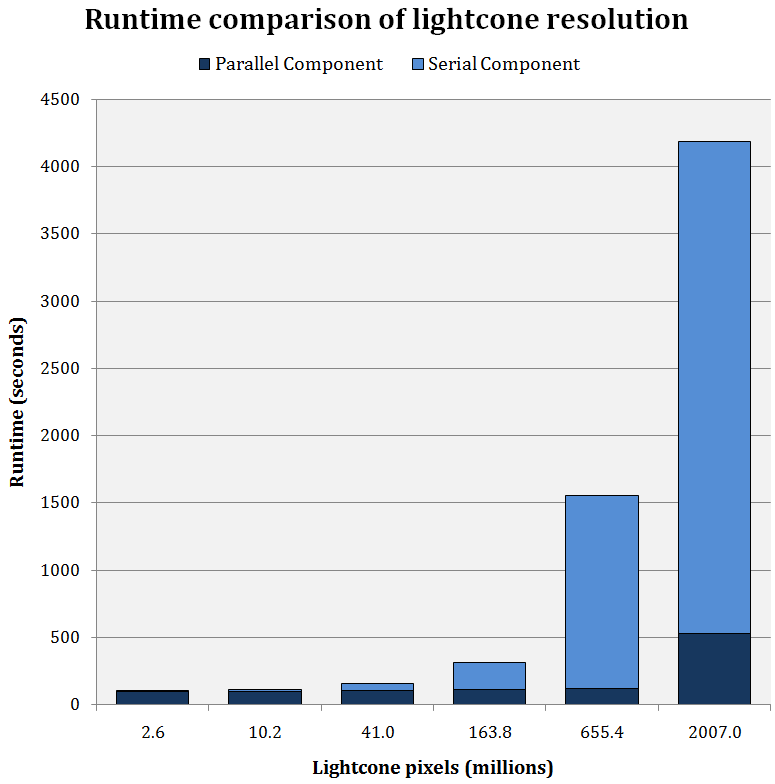


Figure 5.22: A bar graph representing the distributing of post-processing runtime spent on parallel and serial components, as a function of lightcone grid size.

Figure 5.22 shows the undesirable effect pixelisation grid size has on the serial section of the code. As the number of pixels within the lightcone increases, so too does the serial runtime. For a increase in pixel count of 12.1 times, the serial runtime component increased by 17.7 times. For this test, the parallel runtime component also increased but only by 4.9 times. The implications of this effect are explored further in the Discussion.

## 5.3 Lightcone Results

This section presents a number of visualisations generated from the results of the lightcone post-processing code. A selection of statistical measurements are provided of the GADGET-3 input data as well as the lightcone's contents to validate the findings. In addition, a number of visualisations were generated using Python's matplotlib and matplot3d packages. These images are included to provide additional insight into the mechanics of the lightcone construction.

A 64 $Mpc^3$ GADGET-3 simulation volume containing $256^3$ dark matter and $256^3$ gas particles was executed to generate the input data for the lightcone construction process. A plot was generated showing the relative densities of ionised gas mass, HI mass, $H_2$ mass and stellar mass as fraction of the Universe's critical density. This served as a validation that the results produced by GADGET-3 are in line with current models. Figure 5.23 presents the results of this validation.

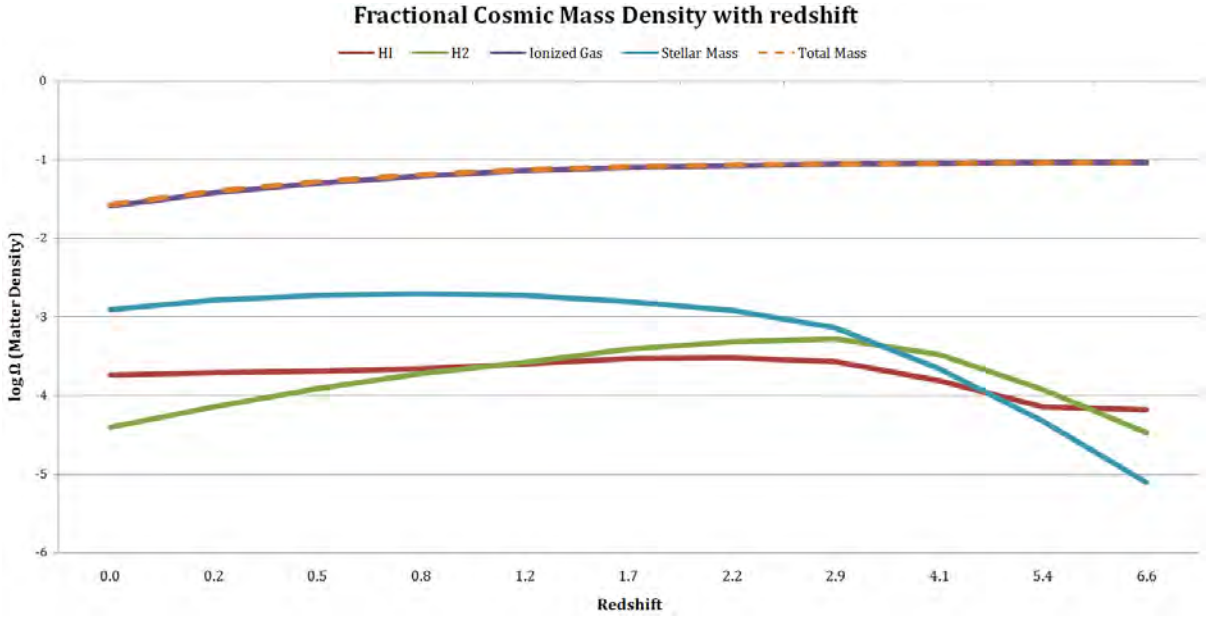


Figure 5.23: A plot showing the fractional densities of HI, $H_2$ and ionised gas as well as stellar mass as a fraction of the critical density. Such plots are useful for evaluating cosmic evolution within the simulation and for comparing the results with existing research.

The graph in Figure 5.23 shows the fractional densities of a selection of gas types. The data is normalised to the decreasing critical density of the expanding Universe in order to remove the effects of expansion from the data. As can be seen, the HI density plateaus at a density of -3.6 in log units, corresponding to $2.4{\times}10^{-4}$ $M_{\odot}/h^{-1}kpc^3$. The density of

HI plateaus because of feedback processes. Hot gases collapse and cool to form neutral gas, which in turn create stars. These stars ignite which ionises the remaining gas and creates outflows. As expected, the ionised gas dominates the mass distribution, with 95% of all baryons existing in an ionised state. In addition, the $H_2$ density decreases as gases are consumed and ionised by star formation over time. The Davé *et al.* density result is provided as reference in Figure 5.24.

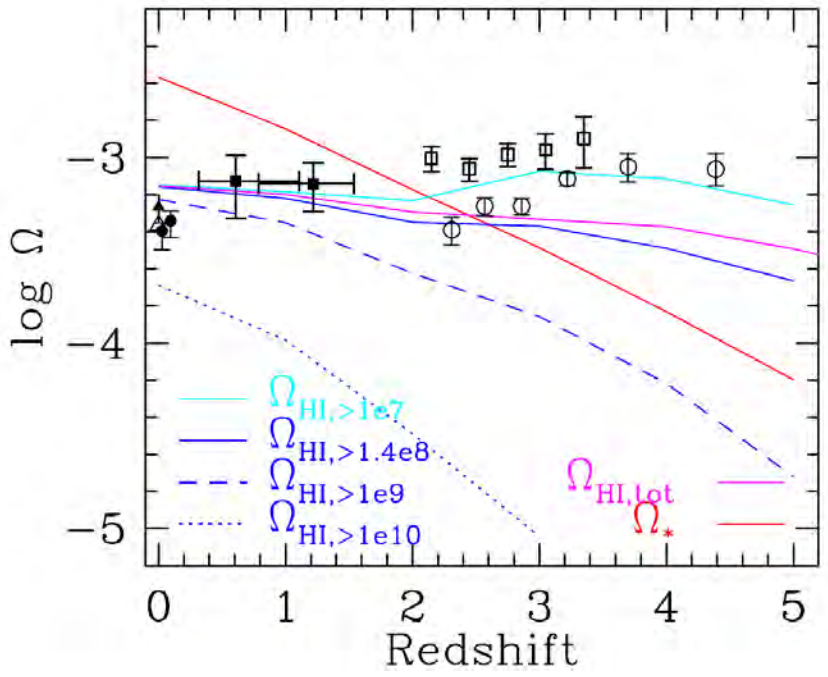


Figure 5.24: A density plot of a GADGET simulation presented in the Davé *et al.* 2013 paper. The magenta line represents the fractional density of all HI, while the red line shows stellar mass density.

One important fact to note is the disparity observed between the stellar mass density of this version of GADGET-3 and the version used in the Davé *et al.* paper. As can be seen, the density of stellar mass for the current version of GADGET-3 decreased at lower redshift. One potential cause of this disparity is the significant difference in simulation resolution. The Davé *et al.* simulation was run using a smaller volume with more particles, resulting in a 64 times higher simulation resolution. This allowed for smaller galaxies to be resolved, which would have increased the amount of stellar mass produced at lower redshift. Further investigation of GADGET-3 code mechanics is outside the scope of this research.

A set of visualisations of the lightcone were generated to ensure that it functioned as expected when post-processing GADGET-3 data. One predictable result of the lightcone, was that its opening diameter should increase as a function of redshift, or distance, from

the Observer. This process was described in Section 4.1.1. Due to the high dimensionality of the datacube, producing meaningful visualisations from its contents proved challenging. A 'fly through' of the lightcone (from the Observer to the furthest distance) was performed to analyse its contents. Figures 5.25 and 5.26 below show two examples of the lightcone's field-of-view at a low redshift (close to the Observer).

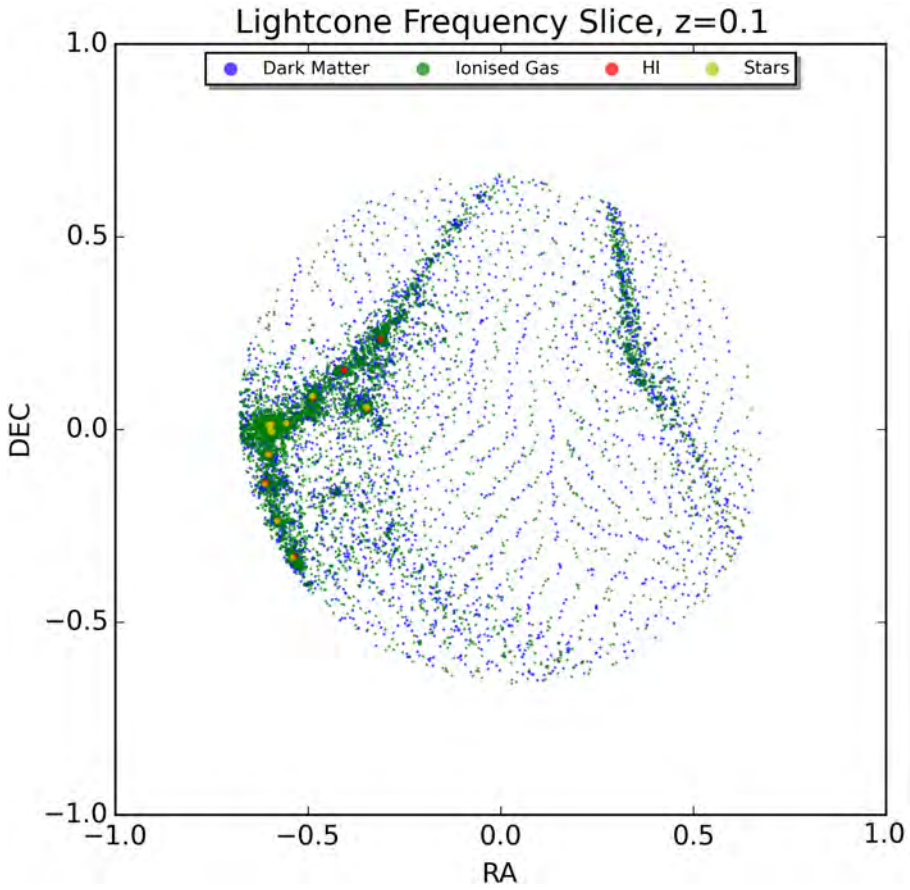


Figure 5.25: A composite image of the lightcone produced by stacking three consecutive frequency slices. This image was produced at a low redshift, with relatively little structure present. Each dot illustrates a pixel, with blue representing dark matter and grey displaying ionised gas.

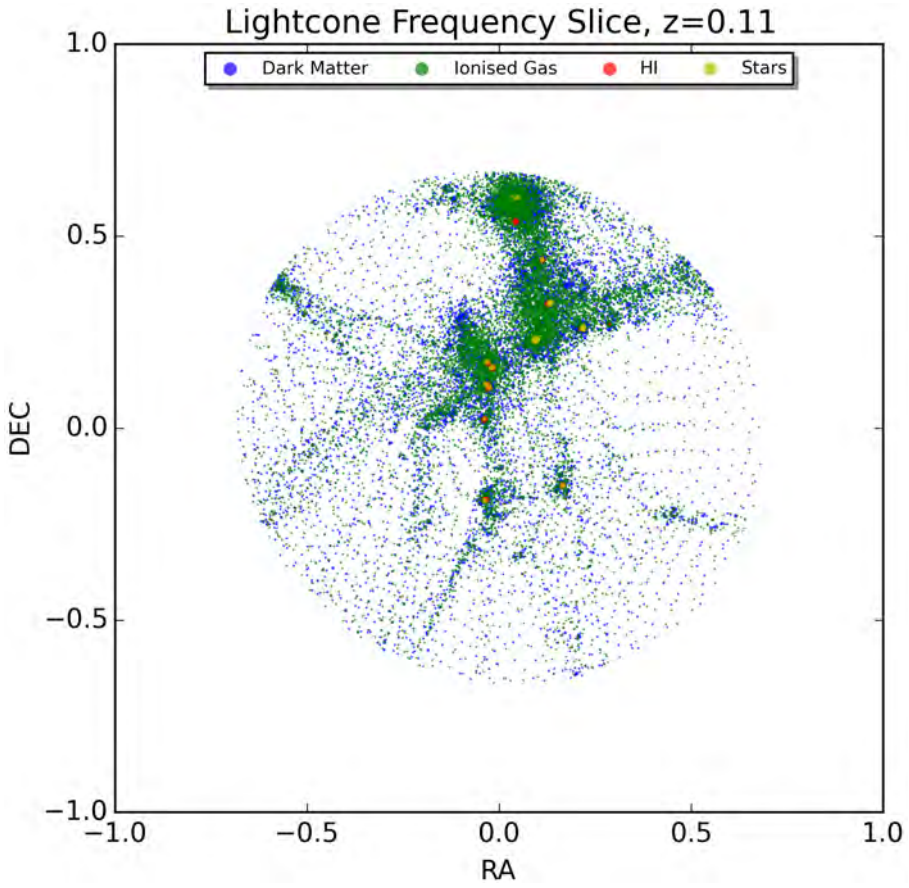


Figure 5.26: A composite image of the lightcone produced by stacking three consecutive frequency slices. This image was produced at a low redshift, with a significant amount of structure present. Each dot illustrates a pixel, with blue representing dark matter, grey ionised gas, red HI gas and yellow stellar mass.

The pair of figures show a subsection of the lightcone at a low redshift. A group of three frequency channels were stacked together to capture a larger number of particles. Figure 5.25 shows a slice containing relatively little matter, while Figure 5.26 shows considerably more activity, including a galaxy (in yellow). Each dot on the plot represents a single pixel in the angular plane of the lightcone. In this example a pixel grid size of 4096x200x200 was selected, thus the lightcone contains 40,000 spatial pixels per frequency channel. The blue pixels represent the distribution of dark matter particles within the lightcone, while green represents the abundant ionised gas. Red pixels represent HI gas and yellow represents stellar mass. It is important to note that each particle type is portrayed on a different scale. This was done to address the large range of masses between different particle types. This effectively exaggerated the presence of HI and stellar mass while the abundance of dark matter was suppressed. From the plot, it can be seen that the lightcone's opening diameter is approximately 10 Mpc at a redshift of z=0.15. In order to produce comparable lightcone plots, the spatial dimensions were converted to angular

units of Right Ascension (RA) and Declination (DEC).

A second pair of frequency cuts were generated at a higher redshift. By looking at the contents of the lightcone at a greater distance, a comparison can be made with the previous figures. Figure 5.27 and 5.28 show three stacked frequency slices near the lightcone's maximum redshift of z=0.58.

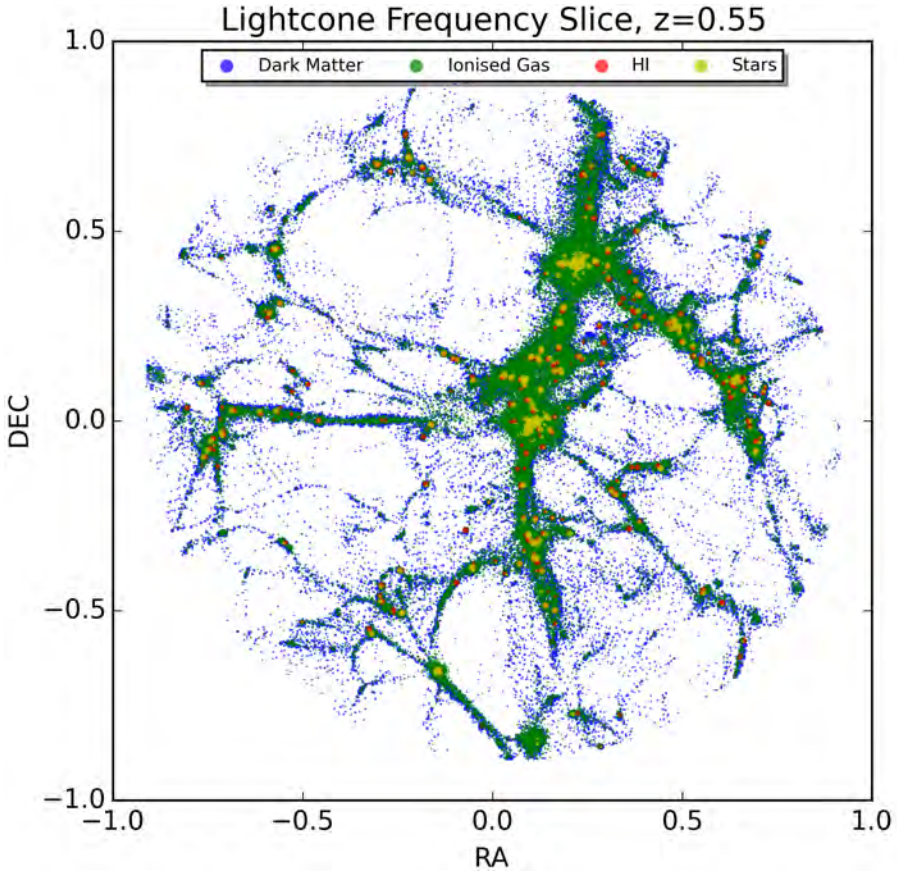


Figure 5.27: A composite image of the lightcone produced by stacking three consecutive frequency slices. This image was produced at a high redshift, with relatively little structure. Each dot illustrates a pixel, with blue representing dark matter, green ionised gas, red HI gas and yellow stellar mass.

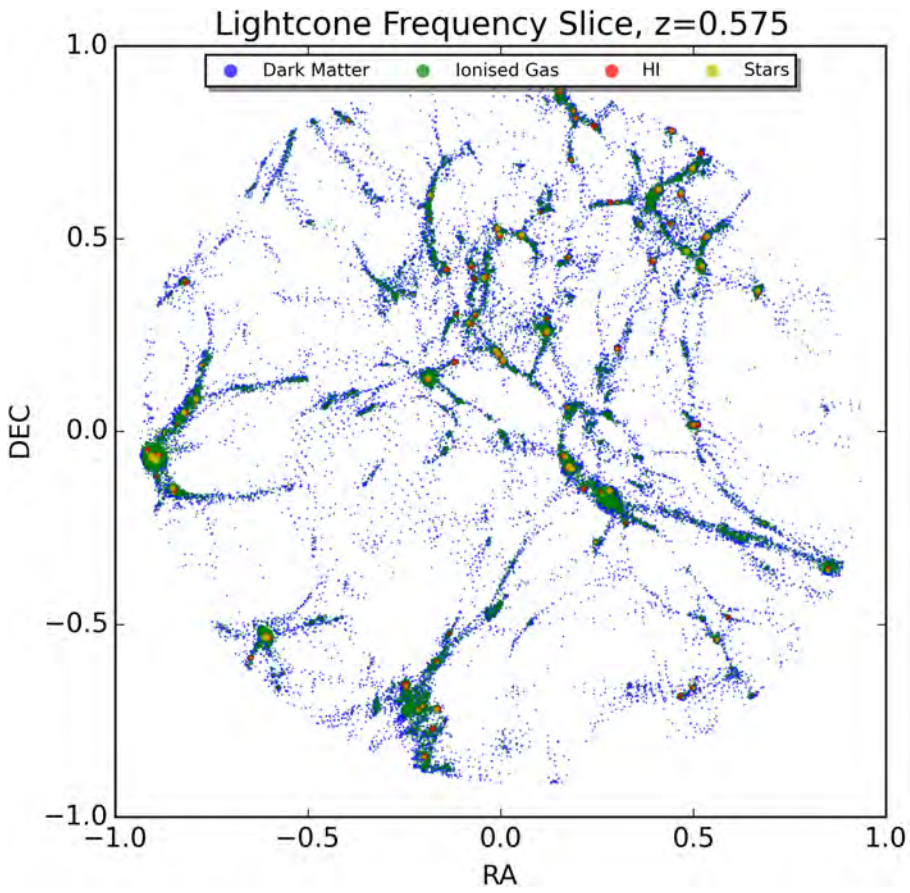


Figure 5.28: A composite image of the lightcone produced by stacking three consecutive frequency slices. This image was produced at a high redshift, with a large amount of structure visible. Each dot illustrates a pixel, with blue representing dark matter, green ionised gas, red HI gas and yellow stellar mass.

Figures 5.27 and 5.28 show a notably different image compared to the lightcone cross sections at a lower redshift. Structure formation is clearly visible, with dark matter and ionised gas particles forming the cosmic web. At intersection points of this web, galaxies are visible as clumps of HI gas and stellar mass pixels. Figure 5.27 shows a region within the lightcone which has relatively little activity, while Figure 5.28 shows a considerable amount of ionised and HI gas, as well as stellar mass pixels. It should be noted that the field-of-view of the lightcone increased considerably, with the opening diameter at a redshift of z=0.58 expanding to approximately 45 Mpc.

The lightcone construction code was developed such that the user could specify the resolution of the pixelisation grid. A comparison was conducted in order to understand and evaluate the results of changing this binning grid. Figures 5.29 and 5.30 present a graphical comparison of pixelisation resolution.

All four images were produced from the same region within the lightcone, with spatial pixelisation resolution being the only variable. Using a binning grid of 100x100 (10,000) pixels, structure is visible though high activity areas are poorly resolved. Increasing the grid by a factor of four up to 200x200 (40,000) pixels yields a significant improvement in resolution. At this resolution, the clumps of star and gas particles which represent galaxies are clearly visible. By using a 400x400 (160,000) pixel grid, addition detail can be seen in the high density clusters of gas and stars. In addition the cosmic web of dark matter is more defined. Finally, increasing the resolution further to 600x600 (360,000) pixels provided little addition information about the lightcone's contents. This was the largest test case considered, consuming over 90 GB of memory - representing the limit of hardware's capacity.

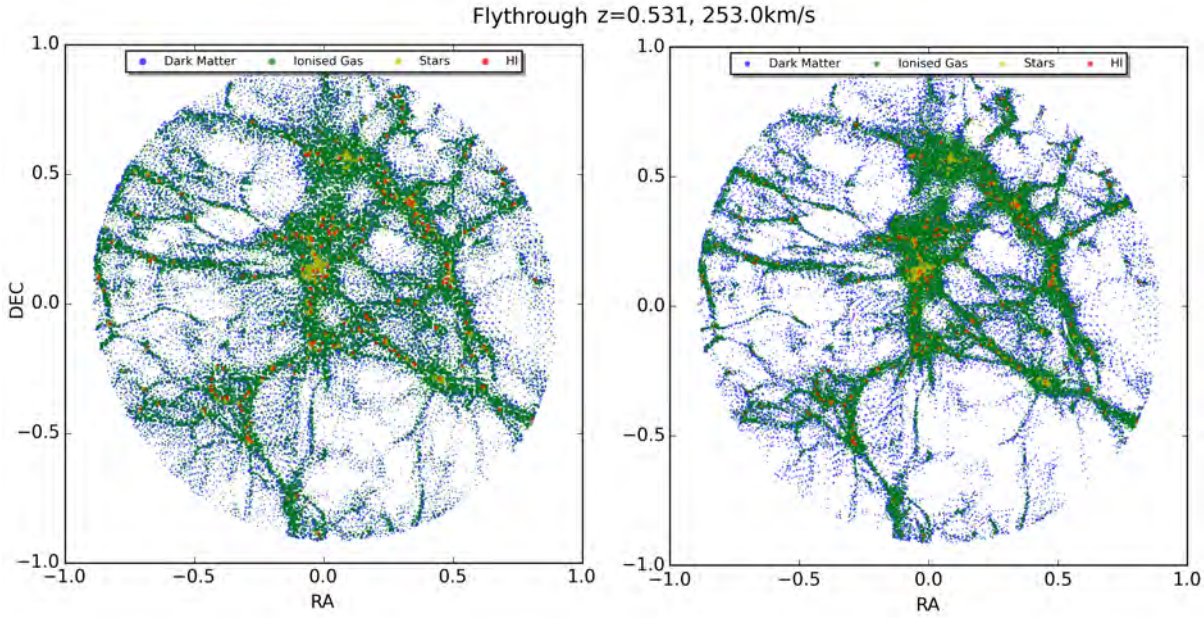


Figure 5.29: A set of three stacked lightcone frequency slices, showing the grid of angular pixels. The image on the left shows a 100x100 grid of spatial pixels, while the image on the right shows a 200x200 grid - four times the resolution.

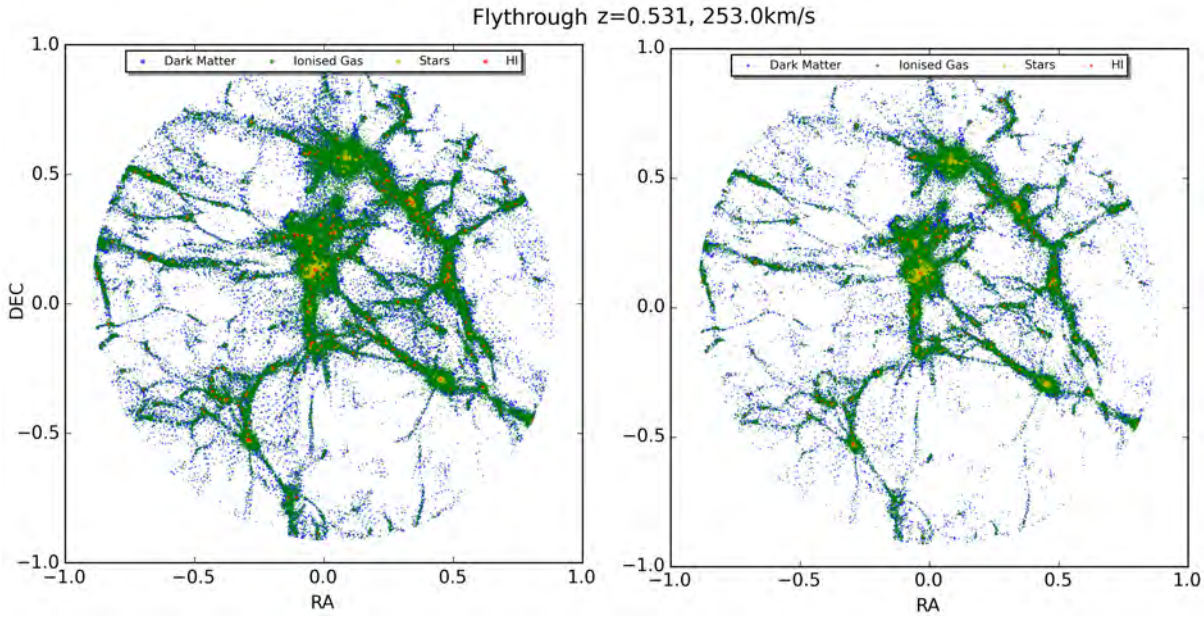


Figure 5.30: A set of three stacked lightcone frequency slices, showing the grid of angular pixels. The image on the left shows a 400x400 grid of spatial pixels, while the image on the right shows a 600x600 grid.

The set of figures above show the noticeable increase in lightcone fidelity when increasing the pixelisation grid. While this increase in detail was advantageous for resolving structures, it did have a drawback. The number of pixels increased while the total effective lightcone volume remained constant. Thus, the volume per pixel decreased as the pixelisation resolution was increased. Figure 5.31 illustrates this effect with a histogram of pixel mass as a function of pixel resolution.

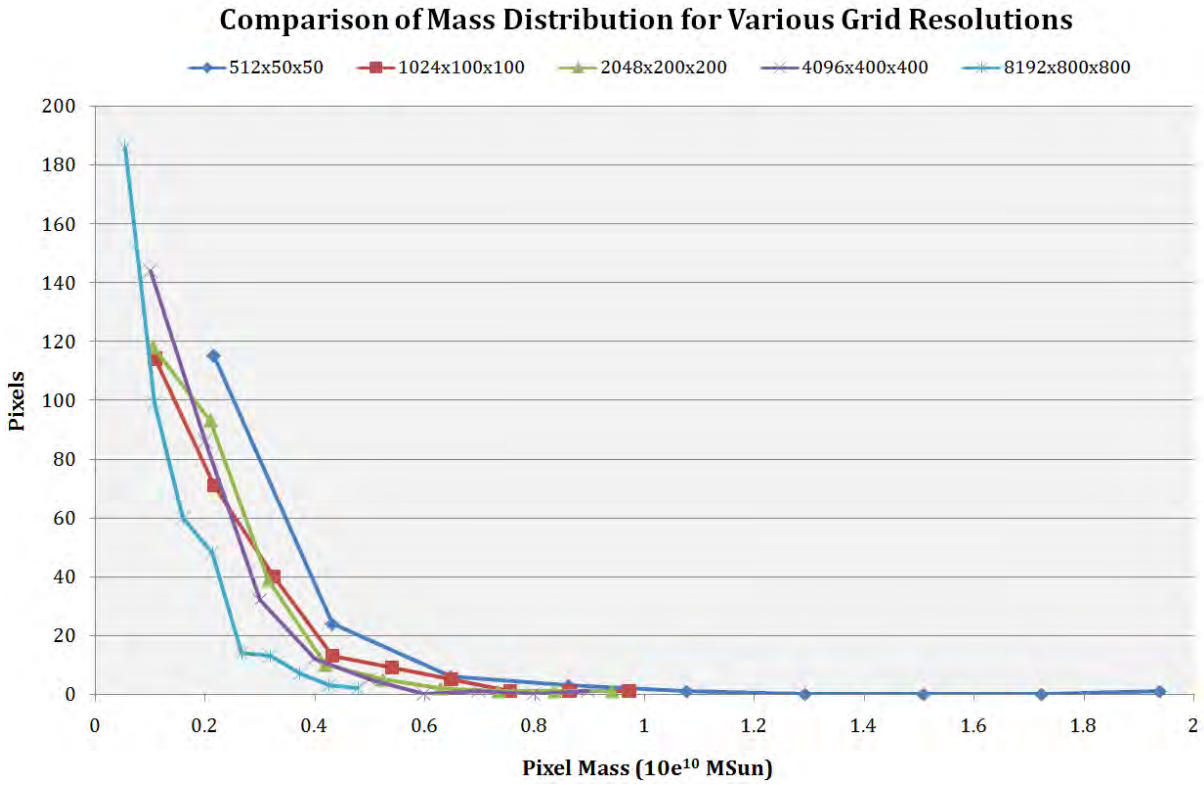


Figure 5.31: A histogram of pixel gas mass as a function of pixel resolution, this illustrates that the volume sampled by each pixel decreased as resolution increases. The dark blue line represents a course grained pixelisation while light blue represents fine grained pixelisation.

The histogram in Figure 5.31 shows the results of increasing pixelisation resolution. For a course grid structure of 512x50x50, the lightcone's fixed volume is sampled by 1.3 million pixels. As such, there a more pixels with a higher mass, as described by the dark blue line. As the resolution increases, a general trend illustrates that the mass within each pixel decreased. Using a pixelisation grid of 8192x800x800 (5.2 billion pixels), produces a large number of pixels with low mass, but far fewer pixels with higher mass. Increasing the resolution by 4000 times results in 1.7 times fewer pixels containing a mass of more than $3\times10^9$ solar masses. This result is significant as the flux spectral density generated of each pixel is determined by its HI mass.

A 3D plot was generated which depicts the dark matter content of the lightcone. An image of such structure is observationally not possible because dark matter is not directly observable. This rendering serves as a verification of the construction process. There are no visible discontinuities or obvious errors in the results. Figure 5.32 presents this visualisation of the lightcone.

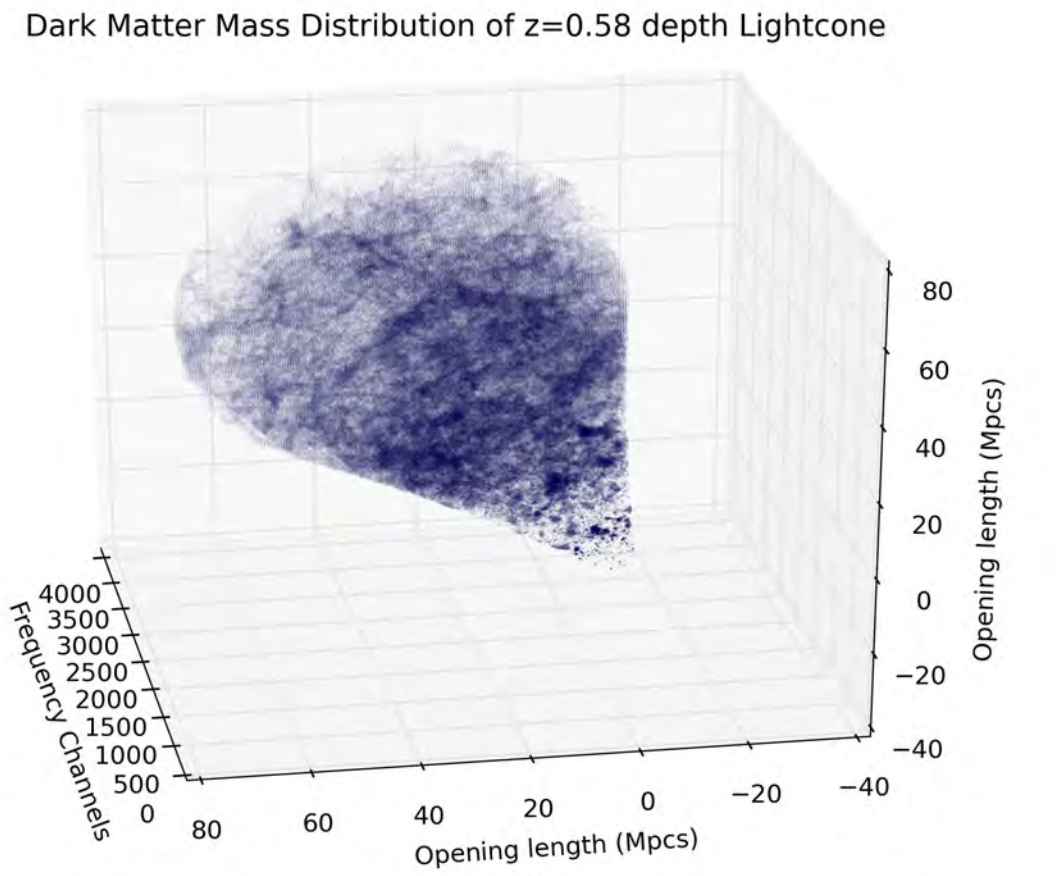


Figure 5.32: A 3D render of the dark matter mass distribution within the lightcone. The cosmic web is clearly visible.

From the 3D plot in Figure 5.32 the cosmic web is visible. The variations in dark matter density are a clear indicator of the structure formation which has occurred through the evolution of the Universe. The plot also demonstrates the expanding beam width inherent in a radio telescope's field-of-view.

Several validations were produced to better quantify the contents of the lightcone. As opposed to considering a series of frequency slices as presented previously, the lightcone was instead collapsed along its two spatial axes. This produced a 1D dataset which represented the total contents of the lightcone in each frequency channel. Figure 5.33 presents the first of these analyses.

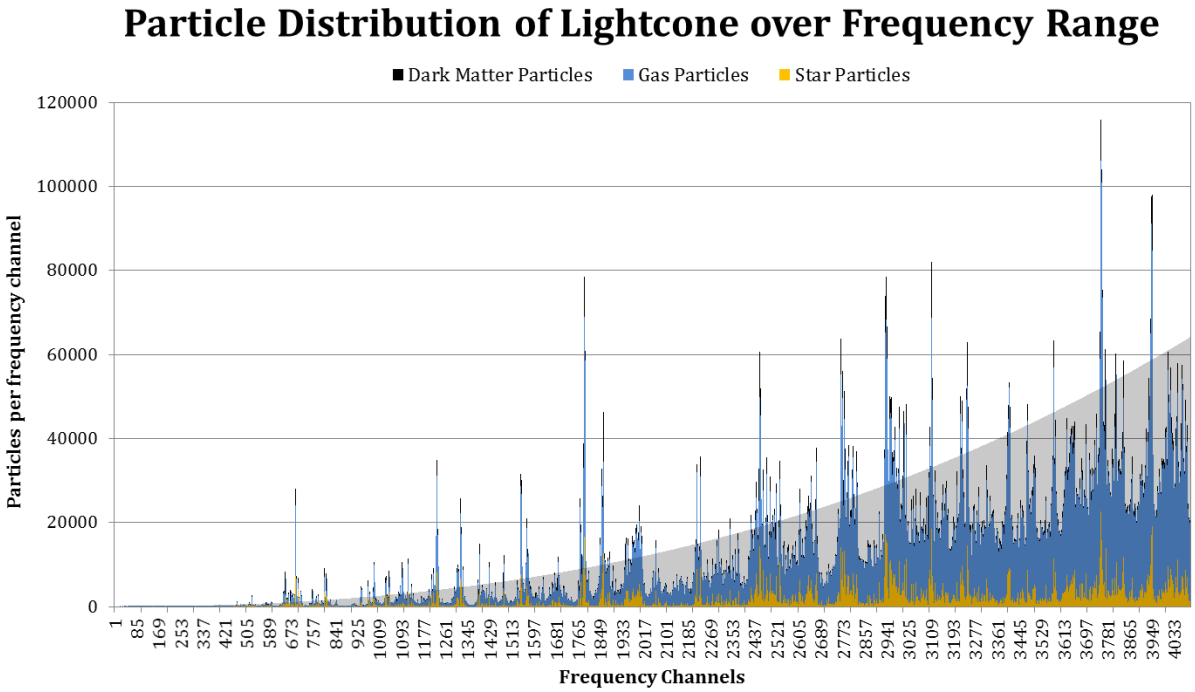


Figure 5.33: A plot of the total number of particles found per frequency channel of a z=0.58 lightcone. The yellow series represents the star particles, while blue shows gas and black described the dark matter content. The shaded background illustrates the cone's opening area at each frequency channel.

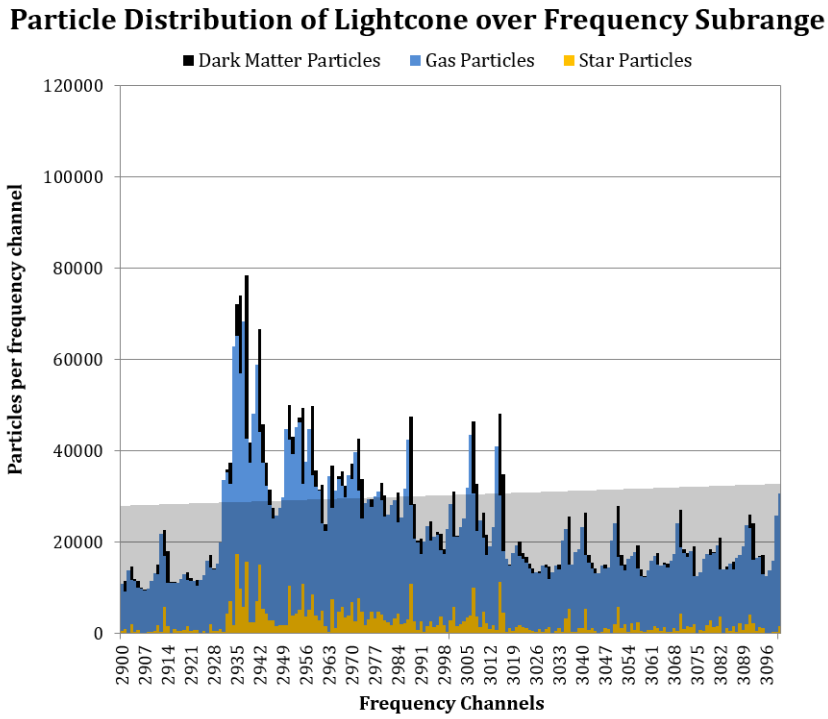


Figure 5.34: A zoomed in view of Figure 5.33 showing 200 of the 4096 frequency channels within a z=0.58 lightcone. A strong correlation between the quantity of star (yellow), gas (blue) and dark matter (black) particles is apparent.

From Figure 5.33, a number of observations can be made. A clear relationship is visible between the number of star, gas and dark matter particles. As expected, they all appear to follow the same pattern. The shaded area shows the lightcone's opening angle at each successive frequency channel. A good correlation is observed between the area of the lightcone and the number of particles contained within it. Figure 5.34 provides a small subsection (approximately 5%) of the data series presented in Figure 5.33. As expected, the quantities of star, gas and dark mater particles are closely related. This represents the variations in density within the cosmic web. In order to further explore the obvious relationship between the gas and dark matter particles, another plot was generated. Figure 5.34 presents a direct relationship between HI mass and dark matter mass within the lightcone.

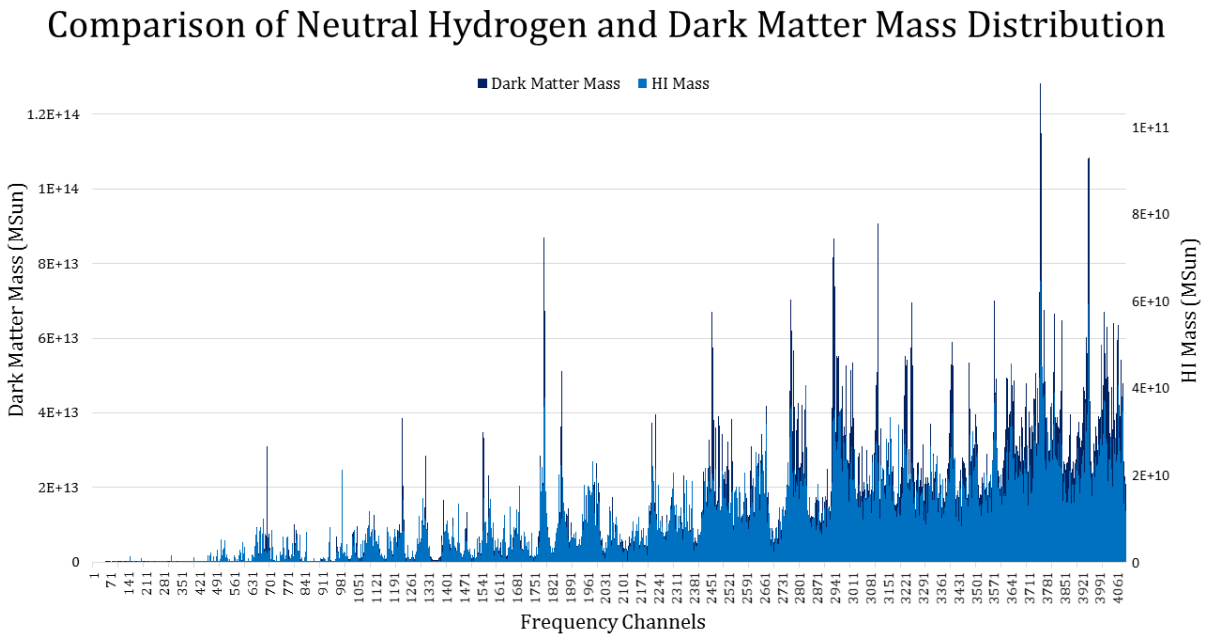


Figure 5.35: A plot showing the relationship between HI mass and dark matter mass within the lightcone. The two spatial axes were collapsed to produce a 1D representation of the total mass per frequency channel within the lightcone.

This plot shows the clear relationship between HI gas and dark matter mass. It is important to note that there exists approximately $10^3$ times more dark matter than HI in the lightcone. This result is in-line with the fractional density result displayed in Figure 5.23.

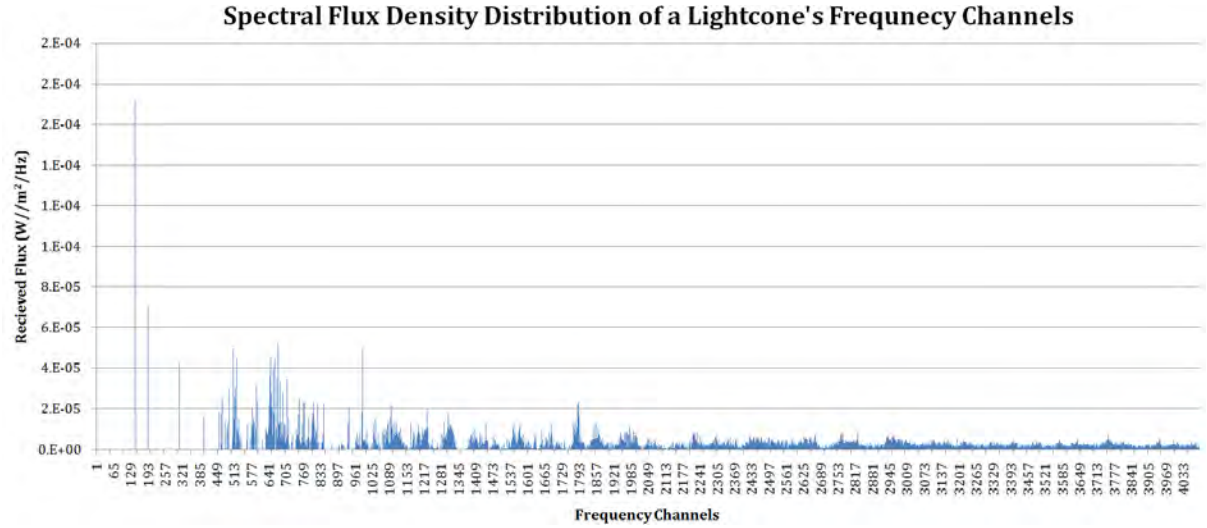


Figure 5.36: A plot illustrating the total flux spectral density of each frequency channel within the lightcone received by the Observer.

Expanding on this results, a plot was produced of the flux spectral density received by the lightcone's Observer. Figure 5.36 presents the summed flux for each frequency channel within the lightcone. As described by Equation 2.14, the flux received by an object follows an inverse power law. Therefore several large peaks are seen in the lower frequency bins - corresponding to a higher frequency or closer to the Observer. As the distance to the observed object increases, the peaks in flux decrease rapidly. Such a plot can be used to estimate the number of detections for a survey; given the telescope's sensitivity and a detection confidence margin.

The flux information obtained from the lightcone post-processing is presented in another way. A set of visualisations were generated which show the distribution of flux within the lightcone. These images are presented in Figures 5.38 and 5.39.

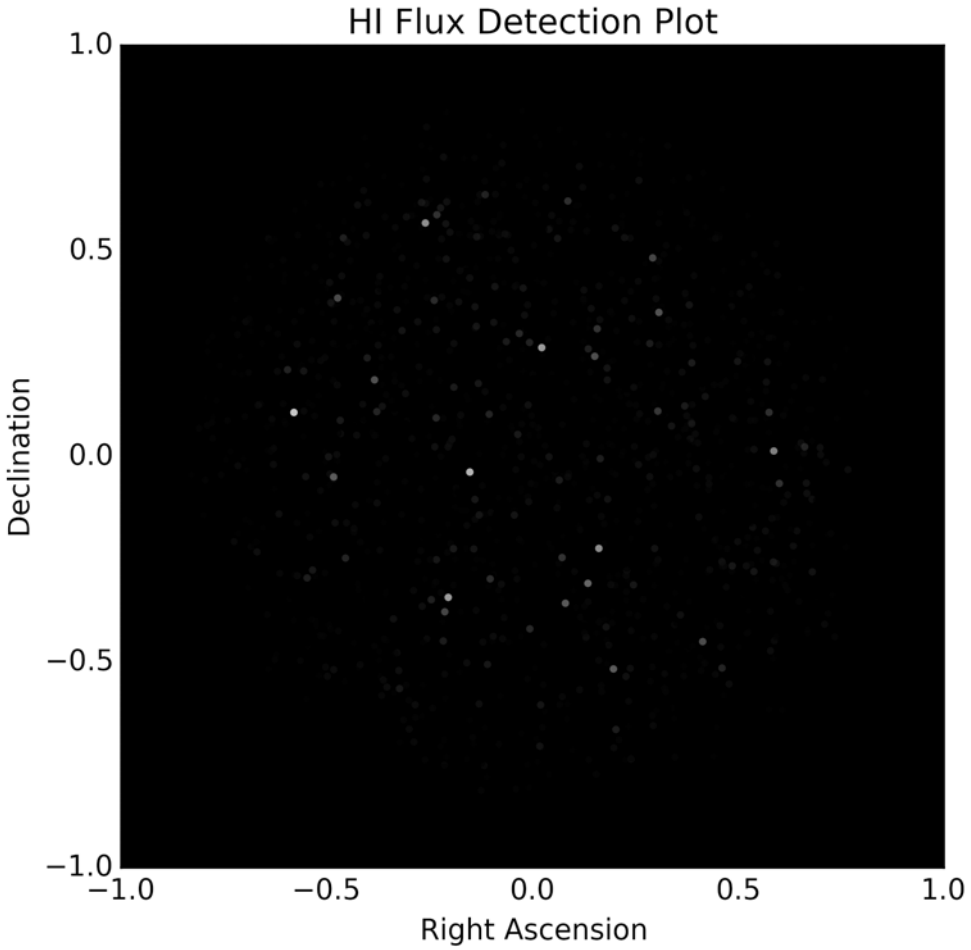


Figure 5.37: A 2D plot showing the flux received by the Observer from HI sources within the lightcone.

Figure 5.37 presents a pixel plot representing individual flux emissions, as would be detected by the Observer. The brightness of the pixels indicate the intensity of the HI

detection. The lightcone's volume expands with distance from the Observer. Thus, it follows that the brightest detections were found in the center of the image, with fainter sources scattered in the background. It is important to note that these flux values are limited to single pixels. In cases where an emission source spans more than one pixel, its total flux is effectively truncated by the pixels. Limited research was conducted on performing source finding to better characterise multi-pixel sources. Figure 5.37 shows the same data from a side-on perspective.

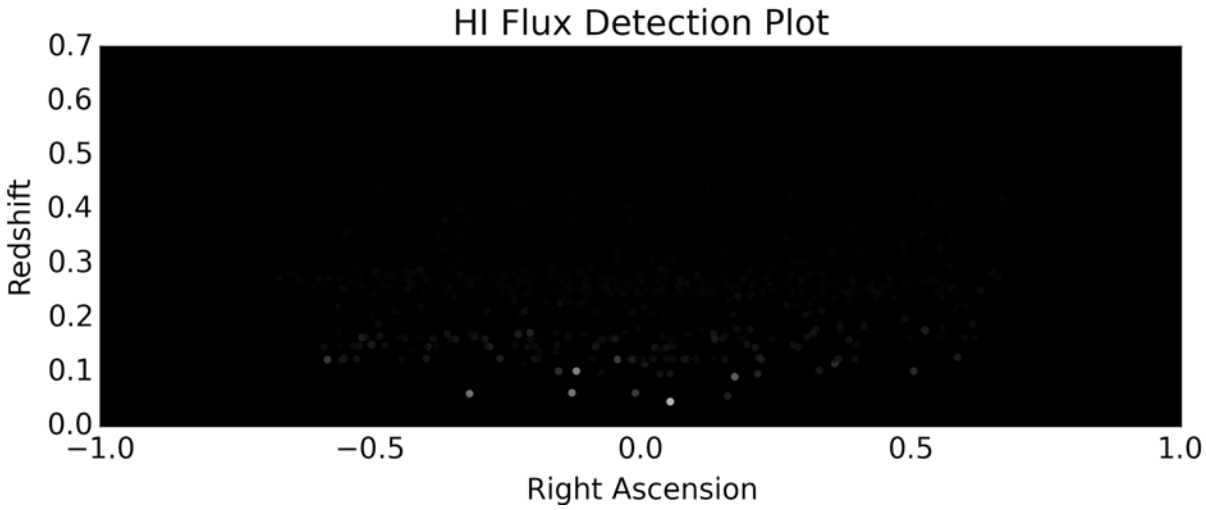


Figure 5.38: A 2D view of the lightcone's flux content from a side-on perspective, the Observer is situated at the bottom of the image with the lightcone expanding away from it.

The relationship between distance and measured flux is more apparent in Figure 5.38. This illustration however ignores the more practical aspects of radio interferometry such as environmental and system noise. Therefore this image portrays a best case scenario with an effective zero-flux detection threshold. In practise many of these sources would not be observable.

Additional research was conducted to explore the effect of a flux threshold on the detections achieved by a virtual survey. To conduct this investigation a new dataset was considered. A galaxy catalogue was used to avoid the truncation of flux emissions which existed in the pixelisation approach. This catalogue was provided from the same GADGET-3 snapshots as the pixelisation results as thus can be considered consistent. The catalogue consists of a list of galaxies extracted from the simulation using a Friends-of-Friends algorithm to identify high density structures (galaxies). Each galaxy contains a set of useful data including; positions and masses, as well as rotational and peculiar velocities. The lightcone code was used to generate four z=1.0 lightcone realizations using this

catalogue. A flux threshold was then applied to the set of galaxies within the lightcone to determine the number of detections achieved. A typical z=1.0 lightcone contained approximately 40,000 galaxies. The data is presented as a detection density to remove the effects of an expanding lightcone volume. Figure 5.39 presents the results, with four selected flux thresholds representing different telescope sensitivities.

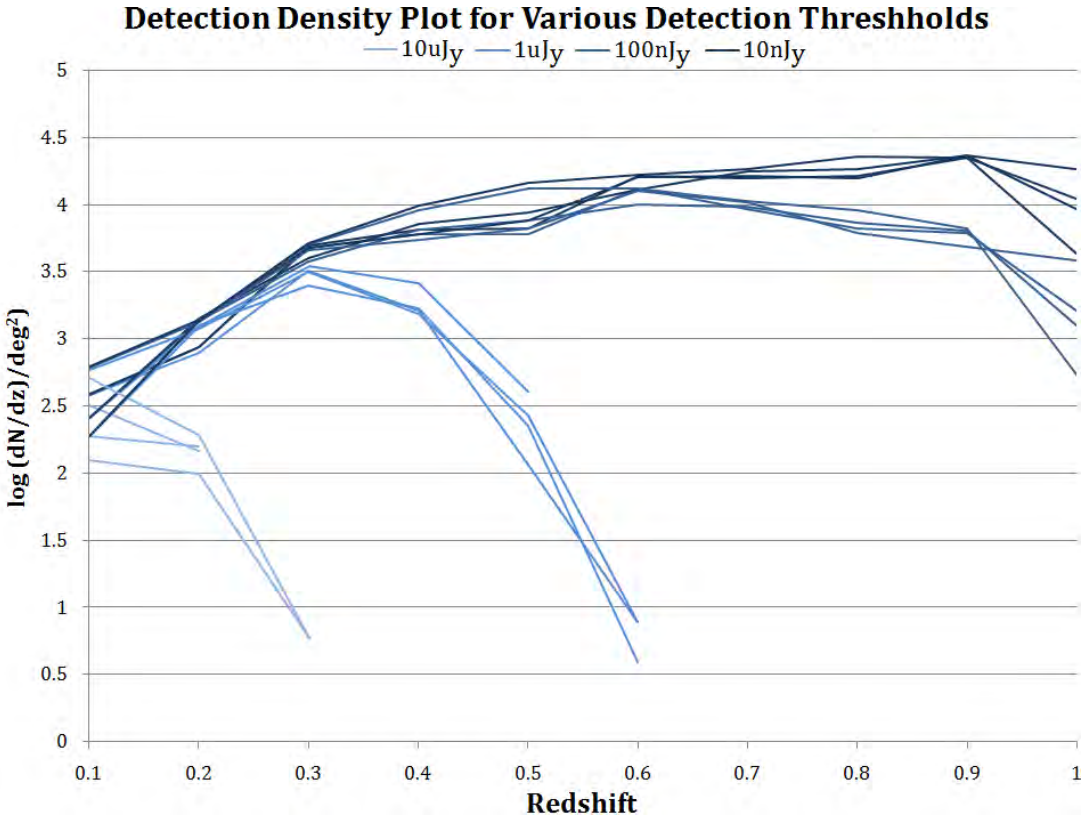


Figure 5.39: A log plot of detections per redshift per square degree. Four flux thresholds are provided; 10uJ, 1uJ, 100nJ and 10nJ. Four lightcone realisations were generated to illustrate the cosmic variance provided by sampling different simulation volumes. Predictably for higher flux thresholds, the number of detections drops off sharply with redshift.

To determine the total number of detections for the virtual survey, the averaged flux threshold series is integrated over the redshift range. For higher sensitivities (lower detections thresholds) the number of detections remained relatively high for higher redshifts, as the lower emission sources were included. For higher (more realistic) flux limits, the number of detections decreased sharply because the received flux decreased below the threshold, following an inverse power law.

A set of tests were conducted to ensure that the user defined pointing vector effectively randomised the lightcone's sampling of simulation space. A set of three lightcone pointing

vectors were chosen. The code was rerun using each of these vectors, creating three different lightcone realisations which followed a different path through the stacked simulation cubes. To verify the randomisation, the line-of-sight peculiar velocities were averaged over each frequency channel for the three test cases. The cumulative velocity of each pixel is calculated by averaging the velocity of the particles within that pixel. A dot product is then used to find the component of that velocity along the lightcone's axis vector. Figure 5.40 illustrates these averaged peculiar velocities for the three pointing vector test cases.

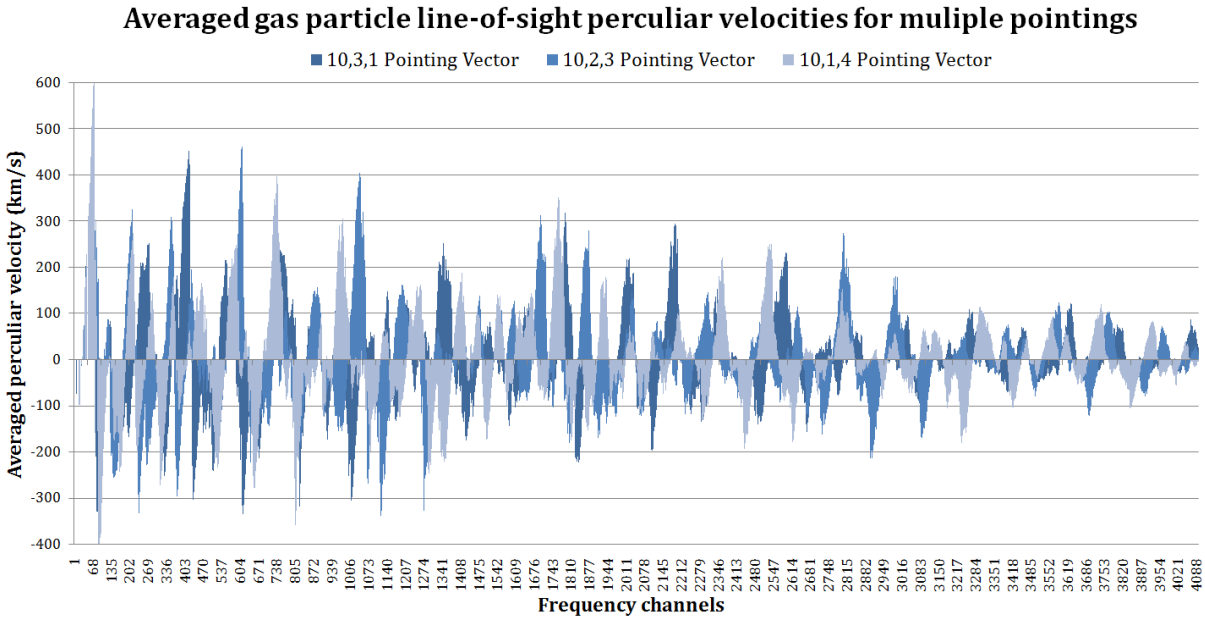


Figure 5.40: A plot illustrating the averaged line-of-sight peculiar velocity of each frequency channel within the lightcone. A set of three pointing vector test cases are included to illustrate the resulting lightcone randomisation.

It is evident that the averaged line-of-sight peculiar velocities has a greater magnitude nearer to the Observer. This is expected, as the beam-width of the lightcone was narrower at lower redshifts and thus fewer particles were being averaged in each frequency channel. At higher redshifts, the beam-width increases significantly, allowing for more particles to be averaged. This resulted in an overall lower averaged velocity. It is important to note that the three test cases produced noticeably different results, although they were all generated from the same simulation input data.

A plot showing the maximum line-of-sight peculiar velocity for each frequency channel is provided in Figure 5.41. Unlike the averaged velocity case shown above, the maximum velocity increased with distance. This occurred because more volume is sampled within the lightcone's increasing beam-width, increasing the probability of detecting particles with more extreme velocities. These extreme velocities describe the gravitational interaction

of particles within energetic galaxy clusters. The apparent periodicity of repeating positive and negative velocities was produced by particles falling towards over dense regions.

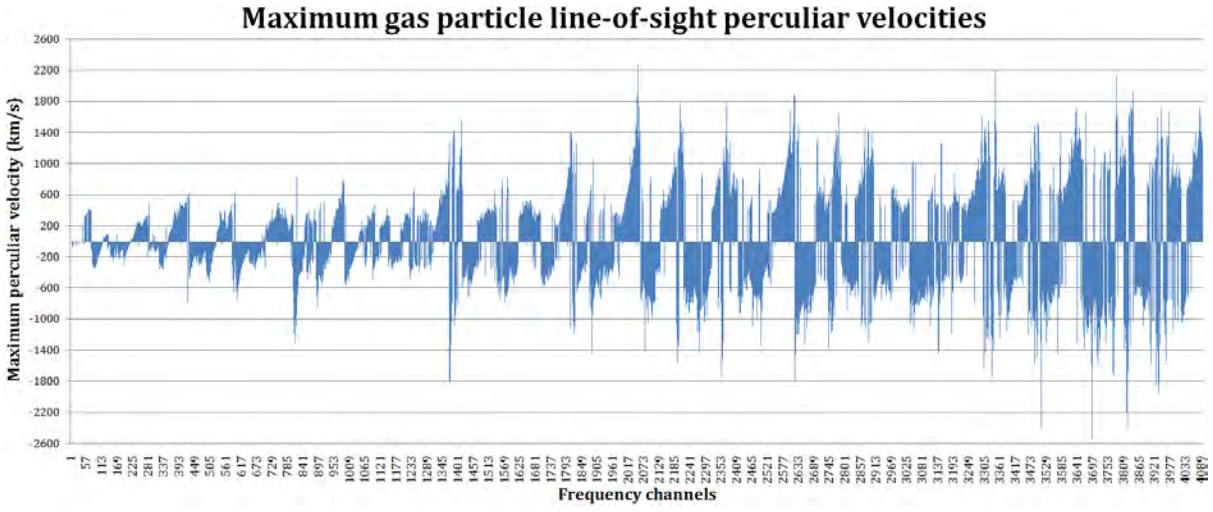


Figure 5.41: A plot showing the maximum line-of-sight peculiar velocity of each frequency channel within the lightcone.

A set of three dimensional visualisations were produced from the data above to provide additional insight into the peculiar velocities within the lightcone. Figures 5.42 through 5.45 illustrate the line-of-sight velocities for a set of four lightcone pointing vectors. From Figure 5.42 a repeating pattern is immediately obvious, this was produced because the pointing vector was aligned with the axis of stacked simulation cubes. Thus, the same region of each successive cube was sampled - producing spurious periodicity. By skewing the pointing vector along which the lightcone was constructed, a different region of each simulation cube was sampled. This resulted in dramatically different results obtained from the same original input data, presented in Figures 5.43, 5.44 and 5.45.

An effect which is not immediately apparent was the distortion of the lightcone's contents due to axes of differing scales. The opening length (diameter) of the cone was presented in co-moving Mpc units, while the length axis was presented as a range of frequency channels. In reality, a lightcone extending to a redshift of z=0.58 is approximately 1500 Mpc deep. However, to better illustrate the lightcone's contents two independent axes units were selected. The rings observed in the images are the result of a very narrow cone structure being rescaled for better visualisation.

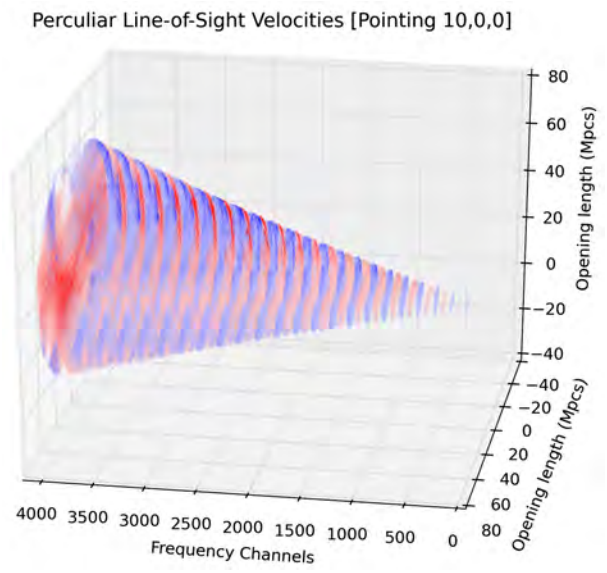


Figure 5.42: A 3D render of the line-of-sight peculiar velocities within a lightcone centred around a pointing vector of [10,0,0]. Blue pixels represent a negative light-of-sight velocity (towards the Observed) while red pixels describe a positive velocity.

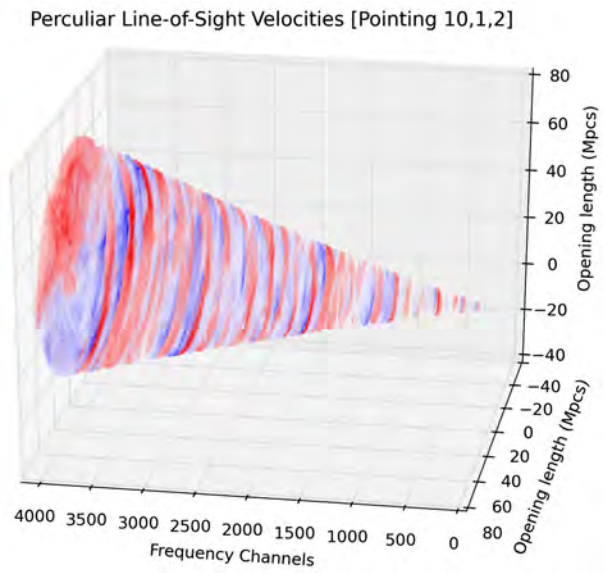


Figure 5.43: A 3D render of the line-of-sight peculiar velocities within a lightcone centred around a pointing vector of [10,1,2]. Blue pixels represent a negative light-of-sight velocity (towards the Observed) while red pixels describe a positive velocity.

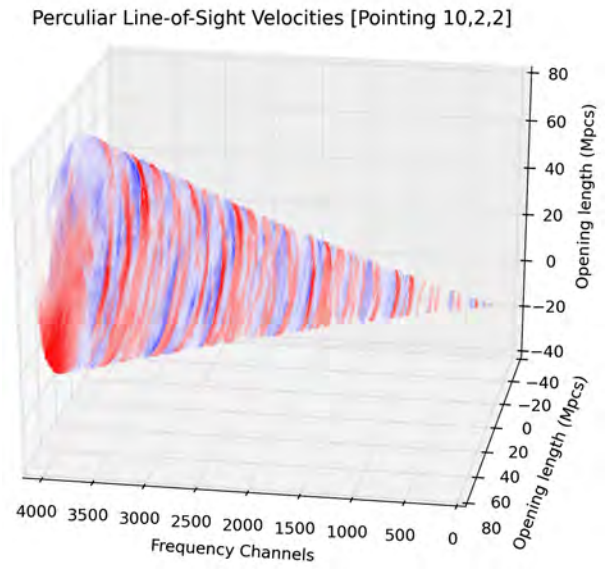


Figure 5.44: A 3D render of the line-of-sight peculiar velocities within a lightcone centred around a pointing vector of [10,2,2]. Blue pixels represent a negative light-of-sight velocity (towards the Observed) while red pixels describe a positive velocity.

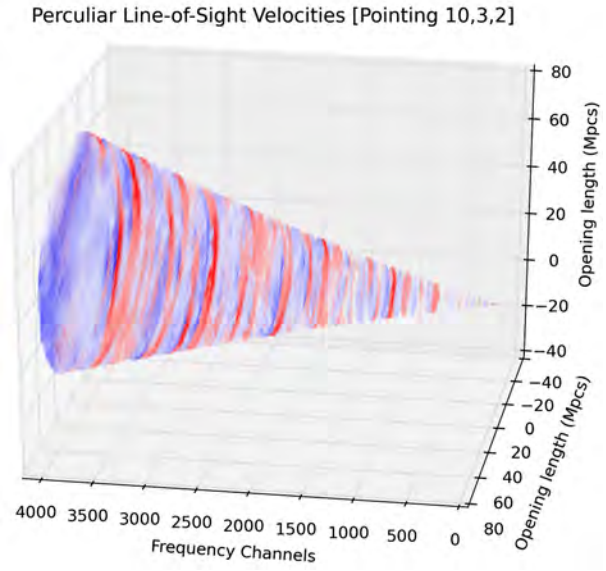


Figure 5.45: A 3D render of the line-of-sight peculiar velocities within a lightcone centred around a pointing vector of [10,3,2]. Blue pixels represent a negative light-of-sight velocity (towards the Observed) while red pixels describe a positive velocity.

A set of tests were conducted in order to determine the failure condition of the lightcone construction code. The code was required to produce a lightcone which met the specifications of the LADUMA survey. This required a maximum redshift of z=0.58 for phase 1 of MeerKAT. However, the code was developed to be a general as possible to used for other potential applications. Therefore, the maximum generated lightcone achievable from the post-processing code was explored. Figure 5.46 presents the resulting lightcone produced after a failure case within the code.

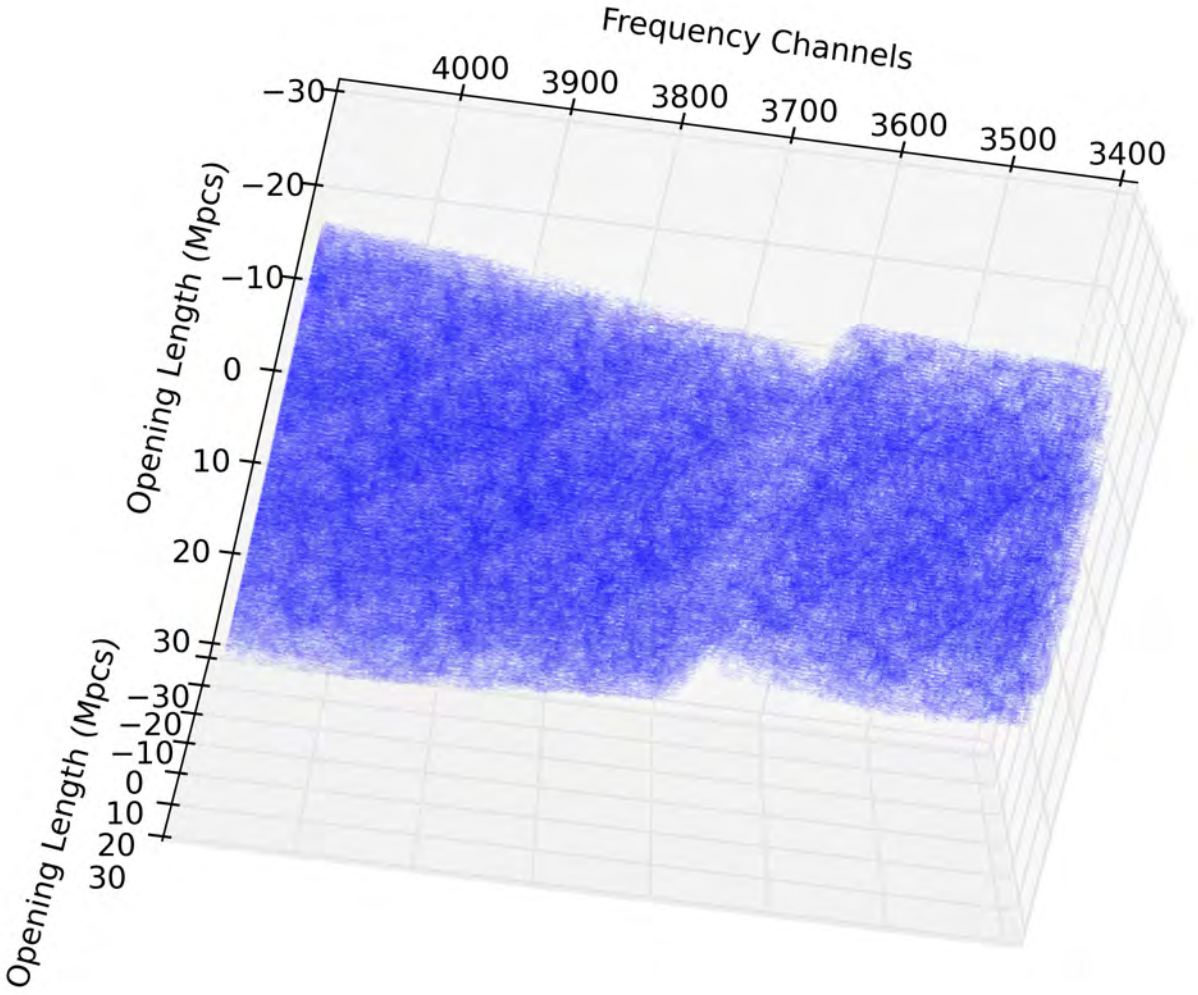


Figure 5.46: A 3D rendering showing a failure condition of the lightcone code. This failure occurred when the opening diameter of the lightcone extended beyond volume of the tiled GADGET-3 snapshots. This situation however occurred at a high redshift, well beyond the requirements of this code.

The discontinuity observed in Figure 5.46 was a result of a lightcone being generated to a redshift of z=2.0. It should be noted that the code correctly generated a lightcone, without discontinuities to a redshift of z=1.5. The cause of this error and a justification for why it is acceptable is provided in the next chapter.

## 5.4 Conclusion

In this chapter, results on a number of topics were presented. A selection of performance figures were provided from the GADGET-3 code evaluation process. It was observed that Intel's ICS compiler provided a meaningful performance increase over the GCC open source compiler. The 13% improvement in performance is approximately in-line with Intel's advertised gains. This topic is expanded upon in the next chapter. In addition, it was found that the previous release of GADGET-3 implemented an inefficient version of the Friends-of-Friends (FoF) algorithm, resulting in woeful performance. The later version of the code remedied this problem by implementing the costly FoF algorithm less frequently.

A poor scaling of also observed from the GADGET-3 code. Increasing the number of processing cores fivefold resulted in a mere 2.5 times reduction in runtime. Finally, the compiler flags provided by the Intel ICS compiler proved to offer little benefit over GADGET-3's standard configuration.

A selection of Python optimisation steps were also presented. Most notable of these was the performance gained through the use of fast math libraries, multi-threaded parallelisation and code vectorisation. Using either the open source ATLAS or Intel MKL libraries resulted in a performance increase of up to 400% in linear algebra operations. Using a multi-threaded approach to generate lightcone sections in parallel provided a near linear performance increase with thread allocation. The increase however trailed off after approximately eight threads, with memory bandwidth likely being the performance limiting factor. The most significant increase in performance was witnessed through vectorisation of the code. This was achieved by converting the main lightcone construction pipeline from a standard Python iterative solution to a set of NumPy array operations. These operations on large datasets were able to take full advantage of the high performance C subroutines within NumPy. As a result, a runtime improvement of 63 times was observed for the given test case.

Finally, this chapter presented several visualisations of the lightcone and its contents. This section was included to provide verification of the lightcone construction process, as well as to provide further insight into the scientific deliverables of this research.

# 6 Discussion

In this chapter, a number of the results presented in Chapter 5 will be discussed. The results obtained from the GADGET-3 optimisation section will be assessed to further evaluate the findings. Focus will be placed on significant outcomes, such as the scaling of the code within an HPC cluster and the performance improvement observed using Intel's ICS compiler.

A discussion will also be provided for the Python post-processing code which was developed to construct lightcones from GADGET-3 simulation outputs. A number of performance optimisations were implemented in this code to reduce the hardware requirements and decrease processing time. Several of the noteworthy benchmark results will be discussed; this includes several subroutine optimisations within the code, Numpy libraries, as well as the parallelisation and vectorisation of the code. In addition, several concerns will also be explored, such as the effect pixelisation resolution had on runtime. A justification is also provided for the functional limitations imposed on the current code implementation.

An analysis of the lightcone will be conducted. Numerous visualisations and statistical plots were presented in the Results chapter. This section will discuss these results to verify the lightcone's construction process and the scientific accuracy of the results. Finally, an explanation is given and an argument made, for the code's conditions of failure illustrated in the Results.

## 6.1 GADGET-3 Performance Evaluation

A comparison was conducted between two recent versions of GADGET-3. It showed a significant increase in the simulation's performance. The cause of this improvement was identified as being the implementation of a better Friends-of-Friends algorithm. The new function was invoked more sparingly, providing more CPU time for the other

computationally intensive simulation subroutines. This modification resulted in a simulation execution rate speedup of over six times. Improvements to code efficiency allow for simulations to be conducted at higher resolutions with no additional hardware resources.

A performance advantage was observed when compiling the GADGET-3 code with the Intel Compiler Suite software, compared to the open source GCC compiler. On average, a 13% decrease in simulation runtime was reported for the multiple tests conducted between the two compilers. This increase in code efficiency was achieved through Intel's proprietary architecture specific software optimizations. The performance increase observed was approximately in-line with Intel's advertised gains of upto 24% for floating point applications [81].

GADGET-3 relies not only on computational throughput, but also memory bandwidth and network communication performance factors. Therefore, while this improvement may appear modest, a 13% boost in performance is a noteworthy result. In HPC industry, small incremental improvements in performance and efficiency provide a means of evolutionary advancement, much like other leading edge fields. GADGET-3 simulations often take days or weeks to complete. The Millennium XXL Simulation, completed in 2010, is one of the well known GADGET simulations. It required over 300 core-years to complete, and executed on 120,000 CPU cores. In such a case, a 13% improvement in runtime would have provided a significant cost savings in terms of power and infrastructure usage.

Benchmarks conducted within the cluster of virtual machines revealed an notably poor runtime speedup for GADGET-3 when the hardware resources were scaled up. The GADGET-3 code was thus moved to the cluster of physical nodes to further investigate this scaling issue. As observed in Figure 5.8, the execution time of the GADGET-3 test simulation improved from a speedup of 145% to 244% when increasing the computational resources by 500%. However, the scaling observed was still far from ideal. This result contradicted the original results documented for GADGET-2. Official documentation for the code reported near linear scaling of GADGET-2 for very large clusters of 512 processors [16]. An important factor which differentiated that evaluation from the one conducted in this research is the nature of simulation being conducted. The large simulations, for which GADGET-2 is well known, such as the Millennium Simulation, were all exclusively dark matter simulations. The additional computational workload introduced by the SPH algorithm is likely the cause of the poor performance scaling observed in this evaluation. New developments such as the "SPH With Inter-dependent Fine-grained Tasking" (SWIFT) project, aim to improve the scalability of the SPH solver for larger simulations [82].

## 6.2 Lightcone Code Optimisation

During the development of the Python code, a number of comparisons were made between alternate software solutions to determine which was most the efficient. One such test was conducted on the simulation cube tiling mechanism for construction of the lightcone. This feature was implemented as a remapping of old co-ordinates to new ones by a fixed scalar. A test was conducted whereby all 16.7 million particles in the simulation were calculated prior to the particle testing, rather than calculating the new co-ordinates of each particle individually. As shown in Figure 5.12, this approach provided a noticeable performance improvement; this was primarily due to the co-ordinate remapping being implemented as a vectorised NumPy arithmetic operation. A small memory allocation overhead exists for this approach (10.8 seconds for the given test case), however the performance gained during the particle testing phase quickly outweighs this overhead for larger problem sizes. Code improvements such as these allowed for larger problem sets to be considered without a significant increase in runtime. In addition, these efficient subroutines had a compound effect on the code's performance when combined with other optimisation techniques.

An evaluation was conducted of several approaches for performing a simple normalisation of a 3D vector. This was presented in Figure 5.11. Surprisingly, the NumPy normalisation API proved to be the slowest implementation in the test. It was concluded that the poor performance observed was due to the manner in which the function was being called. NumPY can provide significant performance benefits, however this performance primarily comes from the vectorised array operations it supports. In this benchmark, the function was implemented in a loop, where the vectors were normalised one at a time. Operating on individual scalars using the NumPy function did not make efficient use of the package's multi-dimensional capabilities. After the lightcone construction code was converted to a vectorised set of array operations, the NumPy APIs such as the numpy.norm() function proved immensely efficient.

A similar situation occurred in the file I/O test. Figure 5.13 showed that the numpy.loadtxt() function was considerably slower than a standard Python list comprehension. There are two primarily reasons for the NumPy API being almost ten times slower. Firstly, Python list comprehensions are incredibly efficient in the right applications, as they can be translated by the interpreter. Secondly, once again NumPy works best on large datasets such as arrays. The data file used for I/O in this test was a cleanly formatted set of delimited floating points. In this case NumPy's advanced exception handling and multi-dimensional data type handling only served to slow it down. If the data from the input file was of a higher dimensionality or contained errata, the NumPy implementation

would prove to be more efficient.

The thread level parallelisation implemented for the lightcone code provided a significant performance improvement over the serial implementation. However, the speed-up observed in Figure 5.17 levelled off at around 6-8 allocated CPU threads. At this point, it is believed that the memory bandwidth of the SMP computers became saturated and bottlenecked CPU performance. Increasing the thread allocation beyond this point did not provide any additional performance. In fact, allocating over 10 threads resulted in a longer runtime; this was caused by the active threads creating contention for memory bandwidth. The runtime of the parallel implementation depended greatly on the order in which the lightcone cubes were allocated to threads. A larger number of simulation cubes were required to produce the lightcone, compared to the number of available CPU threads. Thus, the cubes were queued until a thread became available. This resulted in the thread barrier waiting for a few threads to complete before the program could continue. The cubes were scheduled to threads in descending order; from furthest away from the Observer to nearest. This was done on the assumption that the far end of the lightcone contained more particles for processing and thus required a larger runtime per thread. An undesirable relationship was observed between the resolution of the lightcone's pixelisation grid and the runtime of the code's serial component, as shown in Figure 5.19. This relationship existed because the individual lightcone sections were stitched together after the particle testing and binning had been performed. As the array increased in size, the stitching process relied more heavily on memory bandwidth. At the highest resolutions this involved operating on arrays containing almost eight billion elements occupying 60 GB of memory. While this did impose an upper limit on the pixelisation grid's maximum effective resolution, it should be noted that these tests were conducted at resolutions greater than required for a LADUMA analogue lightcone. On average, the serial and parallel runtime components were equal for a typical resolution test case of 4096x200x200 pixels. Nevertheless, a potential solution to this concern will be presented in the next chapter. Vectorisation of the lightcone construction code resulted in a tremendous 64.4 times decrease in code runtime. This was possible through the use of NumPy matrix APIs, which operate on large arrays of data. These arrays are re-factored into a stream of unconditional CPU instructions. Such an approach avoids the costly branch detections and pipeline flushes which occur in an iterative programming approach. There were however drawbacks to such an approach, the code required to produce a vectorised solution was syntactically more complicated and difficult to troubleshoot. In addition, special care was necessary when handling conditional cases and data dependencies between vectorised instructions. One such example of the added complexity of vectorised code was the particle binning procedure. In the iterative implementation, each particle was binned into a pixel. This was implemented as a simple

increment of the pixel's array elements with particle properties. Such a straightforward approach was not possible with a vectorised algorithm, as all the particles are being assigned to pixels in an atomic fashion. This resulted in a race condition, where pixels had multiple concurrent assignments with data being overwritten. Thus a substantial redesign of the algorithm was necessary. The particles were sorted and aggregated into groups; these sets of combined properties could then be assigned to pixels in a vectorised manner.

## 6.3 Observations of Lightcone Results

A plot was provided in Figure 5.23 which showed the baryonic mass densities of the 64 $Mpc^3$ GADGET-3 simulation. The densities generated from the simulation were approximately in-line with findings in previous research. The Davé *et al.* paper, introduced in the literature review, presented a fractional density of -3.2 (log units) for HI at a redshift of z=0. This research showed a density of -3.6. Various factors contribute to the discrepancy, including the precise configuration of the solver used by GADGET-3, as well as the resolution at which the simulation was conducted.

A set of frequency slices were presented in Figures 5.28 and 5.29, these were produced by varying the resolution of the lightcone pixelisation grid. The fidelity of the data increased with the number of pixelisation bins as expected. However, increasing the number of pixels also resulted in the quantity of HI in each pixel to decrease, as shown in Figure 5.30. As a result, the flux spectral density calculated for each pixel also decreased. This translated to an increased number of pixels falling below a given flux detection threshold. It is therefore advised to select a moderate resolution, in the range of 200x200 to 400x400 spatial pixels. This allows a high definition lightcone to be constructed, while also producing measurable flux emissions. A method of performing multi-resolution sampling will be described in the following chapter.

The cosmic web produced by the dark matter content of the lightcone provides useful validation of its construction process. In addition, the particle, mass and flux distributions presented in Figures 5.32 - 5.35 showed the distribution of matter within the lightcone. In the case of the mass and particle plots, the quantities predictably increased as the aperture of the lightcone expanded. The 2D visualisations of flux received by the Observer clearly showed the inverse power law that characterises these detections. It also illustrated the need for larger and more sensitive telescopes in order to explore deeper into the cosmos.

As illustrated in Figure 5.44, the current lightcone code has a depth limitation of approximately z=1.8. At greater distances from the Observer, the lightcone's opening diameter increased dramatically. Extending the lightcone's depth from the LADUMA specification of z=0.58 to z=2.0 resulted in an increased opening diameter from 46 Mpc to 220 Mpc. As the diameter increased, it became more difficult to encapsulate the lightcone's volume within the static bounds of the simulation cubes. Currently, the simulation cubes are tiled in any direction a maximum of one time to fill the cone's volume. The simulation cubes could be tiled multiple times to fill the lightcone's volume, however this was purposefully avoided. As the number of cubes stacked abreast increased, so did the apparent periodicity of the lightcone's contents. Each simulation cube was merely a copy of the original, thus a repetitive pattern was quickly noticeable. A limit of one tiled cube in any direction was specifically chosen to reduce this effect. While the number of tiled cubes was limited, a larger lightcone could still be created by generating a larger GADGET-3 simulation.

A 2D representation of the tiling limits is displayed in Figure 6.1. which illustrates a failure condition in the code.

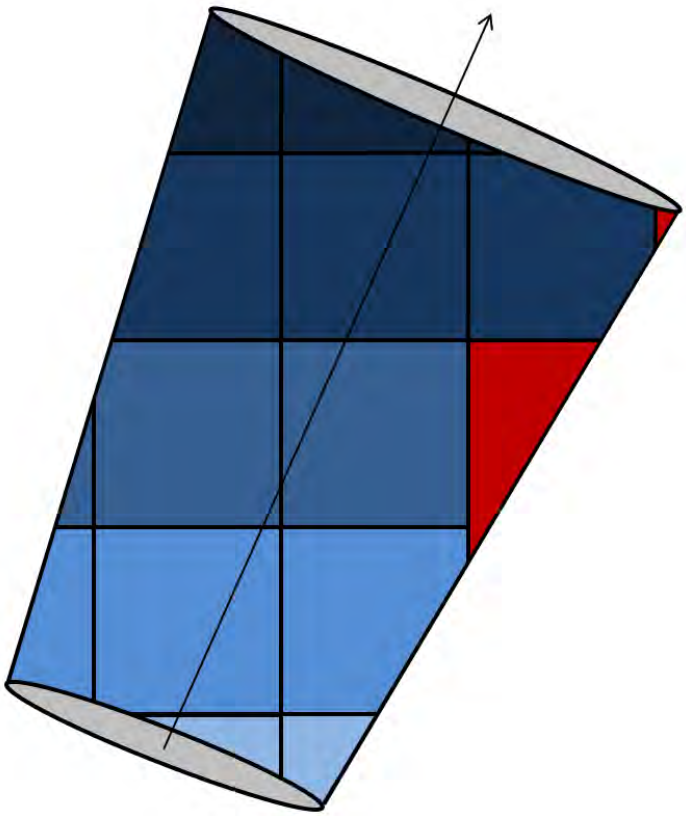

Figure 6.1: A 2D illustration of the failure case for the lightcone code. At high redshifts (z=2.0) the diameter of the lightcone extends beyond the bounds of the tiled cubes. The layers of stacked cubes are represented in shades of blue, while the un-sampled lightcone volume is shown in red.

The image in Figure 6.1 shows a small portion of the lightcone at a high redshift. The lightcone's beam width increases considerably thus the simulation cubes appear relatively small. In the second, third and fourth rows of cubes, the lightcone's volume cannot be fully encompassed by the tiled cubes. In these cases there exists un-sampled volume within the cone, displayed in red. This would be observed as voids within the lightcone.

The lightcone, in its current state, does not take into account the effects of peculiar velocity on the apparent redshift of simulation particles. The line-of-sight peculiar velocity is however calculated and stored within the lightcone as a pixel property. Using the physical redshift of the object, the velocity corrected redshift (and hence frequency) can be calculated.

# 7 Conclusions

This chapter begins with an assessment of the completion of the original research questions posed in Section 1.2.2. Following this, a review of the project's objectives and requirements specified in the Terms of Reference is provided. A summary of the deliverables produced from this research will then be reviewed to determine the success of the project, using the the requirements as reference. This includes the outcomes observed from the GADGET-3 simulation evaluation, as well as the Python code developed for producing lightcones. Finally several recommendations will be given for expanding upon this research in future work. Specifically, design aspects of the lightcone construction code and the accuracy of the resulting lightcone are considered.

## 7.1 Response to Research Questions

This section addresses the research questions which were posed in the introduction. The questions will be answered, with references to the relevant outcomes in the results and discussion chapters.

### 7.1.1 Question 1

*Question: "What effect does compiler selection and software environment optimisation have on GADGET-3 performance?"*

Results obtained from the performance analysis of GADGET-3 showed that the use of the Intel ICS compiler provided a 13% decrease in code runtime, compared to the current GCC compiler. In additional, it was observed that executing GADGET-3 within a cluster of physical machines, provided a considerable performance improvement over

virtual machines. Finally, the effect of an advanced compiler flag setting was determined to be insignificant.

### 7.1.2 Question 2

*Question: "Which post-processing technique is best suited for converting GADGET-3 simulation data into a lightcone?"*

A method derived from the Blaizot *et al.* method of skewing the path of a pencil beam lightcone through a simulation volume was determined to be the most effective approach. It eliminated the spurious periodicity of sampling similar volume repeatedly, but did not introduce discrete discontinuities at the boundaries of simulation volumes.

### 7.1.3 Question 3

*Question: "How effective is the Python programming language for scientific processing tasks such as constructing lightcones?"*

After considerable effort in optimising the lightcone construction algorithm, the Python code was found to be highly efficient. A total speedup of several orders of magnitude was observed over the optimisation process. A LADUMA scale lightcone could be constructed from simulation data in approximately 5 minutes, demonstrating that the Python language provided a sufficient level of performance for this application.

### 7.1.4 Question 4

*Question: "How can simulation metadata, including masses and flux, be stored inside a virtual lightcone?"*

The twelve particle attributes were stored as a pixelised 3D grid within a four dimensional datacube. This provided a logical ordering to the data, where the first three elements of an array index denote the two spatial and one frequency dimensions within the pixelised lightcone. The fourth dimension is used to store the 12 properties extracted from the simulation data. This structure allowed for a standardised and efficient method of queries the contents of the lightcone.

### 7.1.5 Question 5

*Question: "How can a large multidimensional data structure, such as the LADUMA lightcone, be stored to hard disk?"*

The four dimensional datacube was implemented as a NumPy dense array. While this approach required considerably more storage capacity than other solutions, it does provide high performance access to the data. This proves crucial when performing post processing operations on the lightcone, such as source finding.

### 7.1.6 Question 6

*Question: "What is the most effective approach for visualising complex multidimensional data to enable intuitive observations?"*

Due to the high dimensionality of the datasets being visualised, the best approach to intuitively present them was to remove a dimension. This entailed either extracting a single property from the set of particle metadata, such as dark matter mass, and plotting this independently in three dimensions. Alternatively, a spatial dimension could be removed and several particle properties could be presented in 2D.

## 7.2 Review of Research Objectives

The first requirement of this dissertation was to present an up-to-date analysis on the performance of GADGET-3. In addition, any potential performance optimisations should have also been explored and documented. This objective was addressed through a comprehensive evaluation of GADGET-3 conducted within the ACE Lab's HPC cluster. Emphasis was placed on benchmarking the software environment and dependencies upon which GADGET-3 relies.

Another objective required the post-processing of GADGET-3 cosmological data to produce a synthetic telescope's field-of-view. A highly efficient Python code was written specially for this research, which operated on GADGET-3 output snapshots. A technique of stacking and tiling simulation snapshots was developed such that a elongated cone geometry could be produced from the simulation volume. This method was developed to avoid

sampling the snapshots in a repetitive manner.

Lastly, a functional software package was required, to allow other users to produce their own lightcones. The code was thus developed as efficiently as possible and made use of an intuitive parameter file to simplify the end-user's experience. The program relies on standard Python packages such as NumPy and SciPy. Multi-threading and code vectorisation were also implemented to reduce the runtime of the code, thus improving its usefulness.

## 7.3 Research Outcomes

### 7.3.1 GADGET-3 Performance Evaluation

Two main conclusions can be drawn from the performance evaluation of the GADGET-3 cosmological simulation code. The first conclusion is that Intel's proprietary ICS software package provided a noteworthy performance improvement for GADGET-3 over the open source equivalent; GCC. The increase in performance and corresponding decrease in runtime enabled more efficient use of HPC resources when conducting this research.

The second conclusion is that GADGET-3 exhibited a generally poor performance scaling when executing on multiple computer hosts within the ACE Lab's HPC cluster. Runtime speedup tapered off once the code was assigned more than 60 CPU cores. This is a well known issue with GADGET-3. The scaling problem stems from the complex interactions which occur between gas particles modelled by an SPH computational method. Projects such as SWIFT are exploring more efficient simulation algorithms designed for distributed computing architectures.

### 7.3.2 Lightcone Construction

Python code was developed for extracting data from GADGET-3 snapshots and post-processing it into a synthetic lightcone. This code operated efficiently, producing a LADUMA scale lightcone in under 10 minutes. By using optimised math libraries, parallelism and vectorisation, the Python language provided an adequate level of performance for this application. Given the correct inputs, the lightcone program was capable of consistently processing 15 million particles per second.

The program excels when using a lower resolution lightcone pixelisation scheme. An overhead was observed at higher resolution binning due to the 'stitching' of lightcone subsections which is constrained by memory bandwidth. A potential remedy for this issue is presented in the Future Work section.

Vectorisation of the code resulted in a massive 64 times speedup over the most efficient iterative implementation. This was achieved at a cost of only a 15% increase in memory usage, due to the additional arrays required for masking vectorised operations. This compromise was found to be practical, as the number of concurrent threads could be limited to reduce total memory usage of the code, while still achieving a faster overall runtime.

The lightcone produced by the Python code contained various properties which characterise its contents. Analyses were conducted which generally agreed with finding in existing research. By skewing the axis vector along which the lightcone was created, multiple lightcone instances could be created using the same input simulation data. This proved useful for exploring cosmic variance as well as for further investigating the results. By producing multiple lightcone instances, a total of between 8825 and 9445 galaxies were found within a LADUMA scale lightcone. The number of galaxies captured by the lightcone depended greatly on the resolution of the original GADGET-3 simulation, this is explored further in the Future Work section.

## 7.4 Future Work

Based on the research outcomes described above, it is clear that a functional lightcone construction software tool was developed. This tool is capable of producing a synthetic telescope's Field-of-View (FoV) of the sky. Various properties which define the contents of this lightcone were calculated and stored to a dense array data structure for further analysis. The code was developed to be sufficiently fast such that a lightcone could be generated in a reasonable amount of time; with typical problem sizes taking under ten minutes to complete. There are however modifications possible which would further improve the functionality of the Python code and improve the fidelity of the resulting lightcone.

### 7.4.1 Further Lightcone Code Development

The lightcone construction code was developed using the Python programming language specifically so it could be incorporated into the SPHGR GADGET-3 post-processing suite. This tool-kit abandoned a C implementation due to the relative ease of use and performance available through Python. Developing a C or Fortran version of this code would however provide valuable insights into the true performance achieved by the vectorised Python implementation.

A dense data structure was chosen to store the pixelised lightcone for simplicity, as well as to conform with existing file containers such as the FITS standard. In addition, an alternative approach such as implementing a sparse data structure was not selected due to the lack of support for N-dimensional sparse arrays in the SciPy package. However the data stored within the structure is indeed relatively sparse, with less than 5% of the pixels containing anything at all. An efficient sparse array approach to storing the four dimensional lightcone would likely result in considerably lower memory consumption, further improving the code's functionality.

In its current form, the code stitches together the lightcone sections into the final data structure using a single thread. Equation 4.4 shows that using a typical grid size of 4096x200x200 results in a 7.5 GB array. This places a limitation on the hardware capable for executing this code. Focus should therefore by placed on reducing the memory requirements of the code. One possible solution is to use the HDF5 standard which can transparently manage large data structures between memory and the hard drive. This approach would allow for significantly larger problems to be considered, as the memory capacity of the hardware would no longer be a concern. This method would however have drawbacks. The contents of array is unstructured, this may result in disk access times limiting the performance of I/O intensive operations such as source finding.

The lightcone construction process was parallelised in order to take better advantage of modern multi-core computer architectures. These approach provided significant improves and fully saturated the memory bandwidth of the computer. The time taken to process each sub-section of the code varies depending on the number of particles being processed. This results in CPU threads waiting idle for the longer tasks to complete, before triggering the synchronisation barrier. A finer grainer parallelisation implementation would avoid wasting resource and produce a faster time-to-solution.

### 7.4.2 Improvements of Lightcone Precision

Research is currently being done to calculate the relationship between the star formation rate of gas simulation particles and radio continuum emission. Radio continuum occupies a much wider frequency spectrum compared to the HI 21cm line and will prove useful in future SKA observations. This information will be added into the lightcone to supplement the existing suite of metadata.

The lightcone construction method is being modified to operate under a wider range of input parameters which will allow for larger scale lightcones to be created. These will prove useful for larger surveys conducted on the MeerKAT and SKA telescopes in future.

A web based front-end for users would greatly improve the accessibility of the lightcone data. By using a markup language like PHP, an intuitive interface could be created to allow scientists to make specific queries to a database of lightcone results.

The completeness of the lightcone developed for this research depended greatly on the resolution of the cosmological simulation used to generate the input data. The resolution of the GADGET-3 simulation determines the accuracy to which it can model gravitational interaction and structure formation. A 64 $\mathrm{Mpc}^3$ volume containing $256^3$ particles was simulated with GADGET-3 to produce the input data for this research. Using this simulation, galaxies could be reliably detected down to $5x10^9$ solar masses in stellar mass. Measuring the HI mass function is a key science question for LADUMA and thus all the galaxies detectable by MeerKAT need to be represented in the simulation.

The simulation snapshots were repeated several times along the lightcone's line-of-sight to produce a sufficiently deep sample space - typically containing 24 stacked cubes for the LADUMA survey. This method cannot however fully remove the repetition inherent with replicated data. This is particularly true for larger lightcones, as the cone's opening area increases to a more significant fraction of the simulation box length. This problem is avoided by generating larger GADGET-3 simulations which provide larger building blocks for the lightcone stacking process. The number of particles included within a GADGET-3 simulation determines its resolution and therefore the lower mass limit of resolved galaxies. Increasing the number of particles within an N-body simulation is computationally costly, however doing so would provide increased detail of low mass objects which are not currently being accurately modelled.

The memory constraint on the pixelisation resolution could be avoided by implementing

a multi-resolution sampling approach. This algorithm may include an initial course grained sampling of the lightcone to identify areas of significant activity. Once these regions are located, a higher resolution sampling could be performed of these subregions. The lightcone's volume is predominantly empty space, therefore a considerable saving in memory and disk space may be achieved.

# Bibliography


[1] S. V. Beckwith, M. Stiavelli, A. M. Koekemoer, J. A. Caldwell, H. C. Ferguson, R. Hook, R. A. Lucas, L. E. Bergeron, M. Corbin, S. Jogee, *et al.*, "The hubble ultra deep field," *The Astronomical Journal*, vol. 132, no. 5, p. 1729, 2006.

[2] J. Blaizot, Y. Wadadekar, B. Guiderdoni, S. T. Colombi, E. Bertin, F. R. Bouchet, J. E. Devriendt, and S. Hatton, "MoMaF: the mock map facility," *Monthly Notices of the Royal Astronomical Society*, vol. 360, no. 1, pp. 159–175, 2005.

[3] R. Crain, "A first look at GIMIC,"

[4] G. Hinshaw, M. Nolta, *et al.*, "Three-year Wilkinson microwave anisotropy probe (WMAP) observations: temperature analysis," *The Astrophysical Journal Supplement Series*, vol. 170, no. 2, p. 288, 2007.

[5] V. Springel, S. White, *et al.*, "Simulations of the formation, evolution and clustering of galaxies and quasars," *nature*, vol. 435, no. 7042, pp. 629–636, 2005.

[6] L. Johnson, "Rediscovering the Universe," 2012. [Online; accessed 18-June-2014].

[7] P. Ade, N. Aghanim, *et al.*, "Planck 2013 results. I. Overview of products and scientific results," *arXiv preprint arXiv:1303.5062*, 2013.

[8] Las Cumbres Observatory, "What is Redshift?," 2012. [Online; accessed 15-July-2014].

[9] S. R. Furlanetto, P. Oh, *et al.*, "Cosmology at low frequencies: The 21cm transition and the high-redshift Universe," *Physics Reports*, vol. 433, no. 4, pp. 181–301, 2006.

[10] B. Mattson, "The Transparency of the Earth's Atmosphere," 2004. [Online; accessed 22-July-2014].

[11] F. Ghigo, "Karl Jansky and the Discovery of Cosmic Radio Waves," 2008. [Online; accessed 21-July-2014].

[12] C. Greenwood, "Precursors And Pathfinders," 2014. [Online; accessed 22-July-2014].

[13] N. Maddox, "Simulations and Observations for the LADUMA Survey," Presented at 2011 SA SKA Bursary Conference, 2011.

[14] J. Carlson and M. White, "Embedding realistic surveys in simulations through volume remapping," *The Astrophysical Journal Supplement Series*, vol. 190, no. 2, p. 311, 2010.

[15] D. Obreschkow, H.-R. Kloeckner, *et al.*, "A Virtual Sky with Extragalactic HI and CO Lines for the SKA and ALMA," *arXiv preprint arXiv:0908.0983*, 2009.

[16] V. Springel, "The cosmological simulation code GADGET-2," *Monthly Notices of the Royal Astronomical Society*, vol. 364, no. 4, pp. 1105–1134, 2005.

[17] B. Holwerda, S.-L. Blyth, and A. Baker, "Looking at the distant universe with the meerkat array (laduma)," *Proceedings of the International Astronomical Union*, vol. 7, no. S284, pp. 496–499, 2011.

[18] Andrew Baker, "LADUMA! - Looking At the Distant Universe with the MeerKAT Array."

[19] G. F. Ellis, "Issues in the Philosophy of Cosmology," *arXiv preprint astro-ph/0602280*, 2006.

[20] J. L. Cervantes-Cota and G. Smoot, "Cosmology today-A brief review," *arXiv preprint arXiv:1107.1789*, 2011.

[21] A. Berger, "The big bang and Georges Lemaitre," 1984.

[22] W. Fischler and J. Meyers, "Dark radiation emerging after big bang nucleosynthesis?," *Physical Review D*, vol. 83, no. 6, p. 063520, 2011.

[23] G. Lemaitre and A. Berger, *The Big bang and Georges Lemaitre: proceedings of a symposium in honour of G. Lemaitre fifty years after his initiation of big-bang cosmology*. D. Reidel Pub. Co., 1984.

[24] B. D. Fields and K. A. Olive, "Big bang nucleosynthesis," *Nuclear Physics A*, vol. 777, pp. 208–225, 2006.

[25] R. Banerjee and K. Jedamzik, "Evolution of cosmic magnetic fields: From the very early Universe, to recombination, to the present," *Physical Review D*, vol. 70, no. 12, p. 123003, 2004.

[26] J. R. Bond, "Cosmic microwave background overview," *Classical and Quantum Gravity*, vol. 15, no. 9, p. 2573, 1998.

[27] D. Fixsen, "The temperature of the cosmic microwave background," *The Astrophysical Journal*, vol. 707, no. 2, p. 916, 2009.

[28] D. T. Chuss, "Will the Universe expand forever?," 2014. [Online; accessed 22-September-2014].

[29] B. Moore, S. Ghigna, *et al.*, "Dark matter substructure within galactic halos," *The Astrophysical Journal Letters*, vol. 524, no. 1, p. L19, 1999.

[30] W. De Blok, F. Walter, *et al.*, "High-resolution rotation curves and galaxy mass models from things," *The Astronomical Journal*, vol. 136, no. 6, p. 2648, 2008.

[31] A. G. Riess, A. V. Filippenko, P. Challis, A. Clocchiatti, A. Diercks, P. M. Garnavich, R. L. Gilliland, C. J. Hogan, S. Jha, R. P. Kirshner, *et al.*, "Observational evidence from supernovae for an accelerating universe and a cosmological constant," *The Astronomical Journal*, vol. 116, no. 3, p. 1009, 1998.

[32] E. J. Copeland, M. Sami, *et al.*, "Dynamics of dark energy," *International Journal of Modern Physics D*, vol. 15, no. 11, pp. 1753–1935, 2006.

[33] P. J. E. Peebles and B. Ratra, "The cosmological constant and dark energy," *Reviews of Modern Physics*, vol. 75, no. 2, p. 559, 2003.

[34] J. D. Bjorken, "Cosmology and the standard model," *Physical Review D*, vol. 67, no. 4, p. 043508, 2003.

[35] R.-G. Cai, Z.-L. Tuo, *et al.*, "Notes on ghost dark energy," *Physical Review D*, vol. 84, no. 12, p. 123501, 2011.

[36] S. Bender, "What is a Parsec?," 2013. [Online; accessed 30-September-2014].

[37] D. Maulik and I. Zalud, *Doppler Ultrasound in Obstetrics and Gynecology*. Springer, 2005.

[38] N. R. A. Observatory, "The HI 21 cm Line," 2010. [Online; accessed 1-October-2014].

[39] D. J. Griffiths, *Hyperfine splitting in the ground state of hydrogen*. 1982.

[40] R. Freitas, "Interstellar probes- A new approach to SETI," *British Interplanetary Society, Journal(Interstellar Studies)*, vol. 33, pp. 95–100, 1980.

[41] S. Singh, *O livro dos códigos*. 2004.

[42] P. A. R. Institure, "Lessons Overview : Advantages of Radio Astronomy," 2007. [Online; accessed 1-October-2014].

[43] P. Hariharan, *Basics of Interferometry*. Academic Press, 1991.

[44] A. R. Parsons, D. C. Backer, G. S. Foster, M. C. Wright, R. F. Bradley, N. E. Gugliucci, C. R. Parashare, E. E. Benoit, J. E. Aguirre, D. C. Jacobs, *et al.*, "The precision array for probing the epoch of re-ionization: eight station results," *The Astronomical Journal*, vol. 139, no. 4, p. 1468, 2010.

[45] C. J. Lonsdale, R. J. Cappallo, M. F. Morales, F. H. Briggs, L. Benkevitch, J. D. Bowman, J. D. Bunton, S. Burns, B. E. Corey, L. deSouza, *et al.*, "The Murchison widefield array: Design overview," *Proceedings of the IEEE*, vol. 97, no. 8, pp. 1497–1506, 2009.

[46] J. G. B. de Vaate, E. de Lera Acedo, G. Virone, A. Jiwani, N. Razavi, F. Perini, K. Zarb-Adami, J. Monari, S. Padhi, G. Addamo, *et al.*, "Low frequency aperture array developments for phase 1 SKA," in *XXXth URSI General Assembly and Scientific Symp*, pp. 1–4, 2011.

[47] V. Kaspi, R. Manchester, *et al.*, "A search for radio pulsars in southern supernova remnants," *The Astronomical Journal*, vol. 111, p. 2028, 1996.

[48] M. Brown, F. Abdalla, A. Amara, D. Bacon, R. Battye, M. Bell, R. Beswick, M. Birkinshaw, V. Böhm, S. Bridle, *et al.*, "Probing the accelerating Universe with radio weak lensing in the JVLA Sky Survey," *arXiv preprint arXiv:1312.5618*, 2013.

[49] G. Hampson, B. Smolders, and A. Joseph, "One square metre of a million," in *Microwave Conference, 1999. 29th European*, vol. 1, pp. 111–114, IEEE, 1999.

[50] N. R. F. SKA, "Everything you wanted to know about the SKA," 2014. [Online; accessed 30-September-2014].

[51] Keith Campbell, "New design, configuration, timeline for MeerKAT radio telescope programme," 2011. [Online; accessed 19-October-2014].

[52] N. R. F. SKA, "MeerKAT," 2014. [Online; accessed 2-October-2014].

[53] A. Young, M. A. Terada, D. I. de Villiers, and D. B. Davidson, "Assessment of the sensitivity of the south african kat-7 and meerkat/ska radio telescope reflector antennas," in *Electromagnetics in Advanced Applications (ICEAA), 2012 International Conference on*, pp. 486–489, IEEE, 2012.

[54] C. Carignan, B. Frank, K. Hess, D. Lucero, T. Randriamampandry, S. Goedhart, and S. Passmoor, "Kat-7 science verification: Using hi observations of ngc 3109 to understand its kinematics and mass distribution," *The Astronomical Journal*, vol. 146, no. 3, p. 48, 2013.

[55] N. R. F. SKA, "MeerKAT Science," 2014. [Online; accessed 2-October-2014].

[56] M. Santos, M. Silva, *et al.*, "HI Specs - Sensitivity Calculations," 2014. [Online; accessed 3-October-2014].

[57] W. de Blok, "The meerkat karoo array telescope and its hi emission line surveys," *Proceedings of the International Astronomical Union*, vol. 6, no. S277, pp. 96–99, 2010.

[58] H. P. Hofstee, "Power efficient processor architecture and the Cell processor," in *High-Performance Computer Architecture, 2005. HPCA-11. 11th International Symposium on*, pp. 258–262, IEEE, 2005.

[59] M. Baker and R. Buyya, "Cluster computing: the commodity supercomputer," *Software-Practice and Experience*, vol. 29, no. 6, pp. 551–76, 1999.

[60] TOP500, "TOP500 List - June 2014," June 2014. [Online; accessed 24-July-2014].

[61] G. Mateescu, W. Gentzsch, and C. J. Ribbens, "Hybrid computingwhere hpc meets grid and cloud computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 440–453, 2011.

[62] J. Jeffers and J. Reinders, *Intel Xeon Phi coprocessor high-performance programming*. Newnes, 2013.

[63] C. for High Performance Computing, "ACE Lab," 2013. [Online; accessed 24-July-2014].

[64] M. Cawood, "Performance Analysis of Virtualization for High Performance Computing," UCT, 2013.

[65] V. Springel, *User guide for GADGET-2*. Max-Plank-Institute for Astrophysics, Garching, Germany.

[66] J. Helly, "Gadget File Viewer," 2005. [Online; accessed 3-October-2014].

[67] T. S. Selles, "Galaxy formation and evolution through simulation and SAMs," Department d'Astronomia i Meteorologia, Universitat de Barcelona, 2011.

[68] R. S. Somerville, P. F. Hopkins, T. J. Cox, B. E. Robertson, and L. Hernquist, "A semi-analytic model for the co-evolution of galaxies, black holes and active galactic nuclei," *Monthly Notices of the Royal Astronomical Society*, vol. 391, no. 2, pp. 481–506, 2008.

[69] E. Ricciardelli and A. Franceschini, "GECO: Galaxy Evolution COde-A new semi-analytical model of galaxy formation," *arXiv preprint arXiv:1004.3289*, 2010.

[70] R. Davé, N. Katz, *et al.*, "The neutral hydrogen content of galaxies in cosmological hydrodynamic simulations," *Monthly Notices of the Royal Astronomical Society*, vol. 434, no. 3, pp. 2645–2663, 2013.

[71] A. R. Duffy, M. J. Meyer, *et al.*, "Predictions for ASKAP neutral hydrogen surveys," *Monthly Notices of the Royal Astronomical Society*, vol. 426, no. 4, pp. 3385–3402, 2012.

[72] F. Abdalla and S. Rawlings, "Probing dark energy with baryonic oscillations and future radio surveys of neutral hydrogen," *Monthly Notices of the Royal Astronomical Society*, vol. 360, no. 1, pp. 27–40, 2005.

[73] W. Lane, A. Smette, *et al.*, "HI 21 Centimeter Absorption in Two Low-Redshift Damped Ly$\alpha$ Systems," *The Astronomical Journal*, vol. 116, no. 1, p. 26, 1998.

[74] H. A. Council, "GADGET-2 Best Practices for Intel Cluster Ready," 2010. [Online; accessed 7-August-2014].

[75] M. Frigo, "A fast Fourier transform compile," in *Acm sigplan notices*, vol. 34, pp. 169–180, ACM, 1999.

[76] M. Folk, G. Heber, *et al.*, "An overview of the HDF5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, pp. 36–47, ACM, 2011.

[77] V. Springel, "Reference documentation for GADGET-2," 2005. [Online; accessed 7-August-2014].

[78] Y. Wang, "Step by Step Performance Optimization with Intel C++ Compiler," 2013. [Online; accessed 11-August-2014].

[79] W. Pence, "FITS Documentation," 2014. [Online; accessed 20-October-2014].

[80] M. J. Corden and D. Kreitzer, "Consistency of floating-point results using the intel compiler or why doesnt my application always give the same answer," tech. rep., Technical report, Intel Corporation, Software Solutions Group, 2009.

[81] Intel Developer Zone, "Intel C and C++ Compilers," 2013. [Online; accessed 4-November-2014].

[82] M. Schaller, R. Bower, and T. Theuns, "On the use of particle based methods for cosmological hydrodynamical simulations," in *8th International SPHERIC Workshop*, 2013.

# A Python Code for Particle Testing

```
def test_particle(particle, starFlag, dmFlag):

    #calculate co-moving distance to each particle in snapshot
    length = np.linalg.norm(particle, axis=1)

    #calculate redshift of particle using Spline
    z = zrFunc(length)

    #calculate opening angle of cone
    coneAng = np.radians((z+1)*OneOverPi)

    #calculate angle between cone pointing vector and particle
    partAng = angle_between(particle, coneVec)

    #find particle angles within cone
    binned = abs(partAng) < coneAng

    #check that there are any particles (boundary case)
    numParts = np.sum(binned)

    if (numParts > 0):
        print numParts, "gas particles found in this tile \n"

        #begin binning procedure
        bin_Gas(particle[binned], length[binned], z[binned],
           binned)

        else:
            print "No particles found in this tile"
```

# B Python Code for Particle Binning

```
def bin_Gas(particle, length, z, binned):

    #get Y and Z components of particle angle
    cone = length*coneVec[:, None]

    #calculate radius of cone opening at this redshift
    coneRadius =  np.tan(np.radians((z+1)/np.sqrt(np.pi)))*
       length

    #calculate flattened distance in plane perpendicular to axis
    particle = np.transpose(particle)

    #find of particle distance from cone pointing vector
    yDist = particle[1]-cone[1]
    zDist = particle[2]-cone[2]

    #mapping of co-ordinates to array indices
    yBin = ((yDist/coneRadius)*(pixels/2)).astype(int)+(pixels
       /2)
    zBin = ((zDist/coneRadius)*(pixels/2)).astype(int)+(pixels
       /2)
    xBin = ((z / zUpperLimit)*(zBins-1) - offSet).astype(int)

    #------Non cumulative data (redshifts)------

    #calculate redshift of pixels which contain particles
    floor = zMin + (zMax-zMin)*((xBin.astype(float))/cubeZBins)
    ceiling = zMin + (zMax-zMin)*((xBin+1.)/cubeZBins)
    delta = ceiling - floor
```

```
#1 is for z floor
#2 is for z delta
binArray[xBin,yBin,zBin,(1,2)] = [floor, delta]


#------Cumulative data (everything else)------

#calculate HI masses
h1Gas = calcHI()

#calculate peculiar light-of-sight velocities
v = np.dot(vel[binned], coneVec)/np.sqrt(scaleFactor)

#calculate flux for HI particles
flux = calcFlux(h1Gas, z, length, delta)

#list of particle co-ordinates
XYZ = np.vstack((xBin,yBin,zBin)).T

#sort particles by address
order = np.lexsort(XYZ.T)
diff = np.diff(XYZ[order], axis=0)

#produce list of particles with same address
uniq_mask = np.append(True, (diff != 0).any(axis=1))
uniq_inds = order[uniq_mask]
inv_idx = np.zeros_like(order)
inv_idx[order] = np.cumsum(uniq_mask) - 1

#assume zero molecular gas
ionGas = gasMass[binned]-h1Gas

#sum particles into pixels
incr = np.bincount(inv_idx)
ionGas = np.bincount(inv_idx, weights=ionGas)
h1Gas = np.bincount(inv_idx, weights=h1Gas)
flux = np.bincount(inv_idx, weights=flux)
v = np.bincount(inv_idx, weights=v)
```

```
starF = np.bincount(inv_idx, weights=sfr[binned])

#find unique array co-ordinates for summed data
xBin,yBin,zBin = XYZ[uniq_inds].T

#unit conversion, 10e10 Msun -> Msun
h1Gas = h1Gas*1.0e10
ionGas = ion*1.0e10

#0 is for particle count
#3 is for total gas mass
#4 is for H1Mass
#5 is for flux
#6 is for perculiar V
#7 is for SFR / radio continuum

#data cube assignments
binArray[xBin,yBin,zBin,(0,3,4,5,6,7)] += [incr, ionGas,
    h1Gas, flux, v, starF]
```

# C Python Code for Calculating HI Content of Gas Particles

```
def calcHI():


    #get subset of snapshot data for particles inside cone
    T_p          = T[binned]
    rho_p        = rho[binned]
    mass_p       = gasMass[binned]
    sfr_p        = sfr[binned]
    fneut_p      = fneut[binned]
    hsml_p       = hsml[binned]

    H2_frac = 0.

    #---if it is not a star forming particle---

    #select particles with zero SFR
    nonStarForming = np.where(sfr_p == 0.)[0]

    #initialize arrays
    ilo = np.zeros(nonStarForming.size)
    ihi = np.zeros(nonStarForming.size)
    ihi[:] = (NINTERP-1)

    #calculate fraction of gas which is hydrogen
    frh = fneut_p[nonStarForming] * XH * rho_p[nonStarForming] /
        (MHYDR*hsml_p[nonStarForming]*1.3737)
```

```
#vectorized optimization loop
loop = ((ihi-ilo) > 1.)
while(np.count_nonzero(loop) > 0):
    #condition mask 2 = if statement
    mask = ((np.array(KernIntTable[[(ilo[loop]+ihi[loop])
        /2],1])*frh[loop] < NHILIM)).flatten()
    ihi[mask] = (ilo[mask]+ihi[mask])/2
    ilo[~mask] = (ilo[~mask]+ihi[~mask])/2
    loop = ((ihi-ilo) > 1.)

#select particles with temperature below 3E4 and that are
    shielded
mask = np.asarray(np.where((T_p[nonStarForming] < 3.e4) & (
    ilo > 0.))).flatten()
index = ((ilo[mask]+ihi[mask])/2).astype(int)

#calculate fraction of hydrogen that is neutral
fneut_p[mask] = ((fneut_p[mask] * KernIntTable[index,0]) +
    FSHIELD * (1.0 - KernIntTable[index,0]))

#---if particle is star forming---
starForming = np.where(sfr_p > 0.)[0]
coldphasemassfrac           = (1.0e8-T_p[starForming])/1.0e8
    ;
Rmol                        = (rho_p[starForming]*T_p[
    starForming] / (P0BLITZ*MHYDR))**ALPHA0BLITZ
fneut_p[starForming]        = FSHIELD * coldphasemassfrac /
    (1.0+Rmol)

return (fneut_p   *  mass_p * XH )
```