epcc

Galaxy Exploration: Accessible galaxy simulations in education and outreach with Wee Archie

Feasibility Study

Xinyi Ding

March 29, 2024

1 Introduction

N-body galaxy simulations are currently successfully implemented in a large number of projects, and a large amount of research has been done to investigate how GPU acceleration can be used to reduce the computational time required for these simulations. Parallel computing has also advanced significantly in the field of N-body galaxy simulations, particularly in splitting up complex calculations over several processors. This method significantly improves the efficiency of the simulation process and makes it easier to construct more complex and comprehensive galaxy models. This feasibility study provides an in-depth look at histories and potential advances in computational astrophysics, focusing on the optimization of N-body galaxy simulations for mini supercomputers such as Wee Archie[1].

By downscaling traditionally large and complex simulations and developing a dedicated parallel version of the galaxy simulation designed for the mini supercomputer Wee Archie, the project aims to show how parallel computing can make complex simulations more accessible and manageable, especially for educational and outreach purposes. This approach highlights the potential of advanced computing technologies to increase understanding and encourage greater engagement in computational astrophysics.

2 Background and Literature Review

2.1 Historical Context and Evolution

2.1.1 The N-body Method

To understand the dynamics of celestial mechanics, Holmberg[2] used light bulbs and galvanometers to calculate forces and tracked the evolution of 37 particle systems in 1941laying the early theoretical and practical foundations for N-body simulation. With the advent of computer simulations in the 1960s, Aarseth [3] was one of the first to develop direct N-body codes for star cluster simulations. His pioneering work demonstrated the power of numerical simulations to reveal the dynamic evolution of stellar systems and marked an important milestone in computational astrophysics.

The evolution of N-body simulations over subsequent decades was marked by substantial developments, primarily attributed to the integration of advanced algorithms that significantly enhanced both computational efficiency and precision. A pivotal development in this era was the Barnes-Hut hierarchical tree method [4], introduced by Barnes and Hut, which effectively reduced the computational load from $O(N^2)$ to $O(N \log N)$, thereby enabling the simulation of considerably larger systems. At the same time, Greengard and Rokhlin [5] introduced the Fast Multipole method (FMM) representing a revolutionary improvement, applying the compression of the multipole expansion of the potential field to speed up the force calculation. Complementary to these advances, the Particle Grid (PM) code [6] developed by Hockney and Eastwood combines particle techniques with grid strategies to efficiently handle long-range forces. The multigrid method and its algebraic extension by Brandt [7] and the algebraic multilevel Iteration (AMLI) method by Axelsson and Vassilevski [8] contributed further to the significant progress of this period. It provides an efficient method to derive solutions of linear systems from discrete differential equations, and improves the convergence speed of solutions.

Significant progress was made with the advent of cosmological N-body simulations, as detailed by Efstathiou et al. [9] and Davis et al. [10] in 1985, which were crucial for exploring the development and structure of the universe on a grand scale. By 2008, according to Trenti and Hu

[11], the scope of N-body simulations had expanded significantly, with simulations incorporating up to 10^5 particles for analyses using direct integration methods over the duration of two-body relaxation timescales, as illustrated by Baumgardt & Makino [12]. Moreover, simulations involving up to 10^{10} particles for studying collisionless dynamics and cosmological phenomena had been realized, exemplified by the Millennium Run conducted by Springel et al. [13].

2.1.2 Smoothed Particle Hydrodynamics (SPH) Method

With the in-depth research and development of N-body galaxy simulation, the introduction of the smoothed particle hydrodynamics (SPH) method marks a revolutionary progress in galaxy simulation, especially showing unique advantages in simulating complex gas dynamic processes and feedback mechanisms within galaxies.

SPH originated in the late 1970s as a meshless Lagrangian method for solving fluid dynamics problems. It was independently developed by Gingold and Monaghan [14] and Lucy [15] as a method for solving astrophysics involving fluid dynamics problem method. The core principle is to represent a fluid as a collection of discrete particles, each with properties such as mass, velocity, and internal energy, smoothed over a finite volume of space.
In the context of galaxy simulation, SPH is essential for simulating complex processes such as gas dynamics, star formation, and feedback mechanisms within galaxies. The 1990s saw significant progress in the use of SPH in galaxy simulations. It is worth noting that Navarro and White [16] used the SPH method in conjunction with N-body techniques to simulate galaxy formation within a cosmometric framework, representing a huge leap forward in the realism of simulated galactic phenomena. The 2000s have seen significant advances in galaxy simulation techniques, particularly by Springel and Hernquist [17], who combined sophisticated star formation models with comprehensive feedback mechanisms. This integration significantly improves the accuracy and realism of smoothed particle hydrodynamics (SPH) -based galaxy simulations. GADGET code introduced by Springel et al. [18] is efficient and parallelizable code for cosmogical N-body and SPH simulations and represents a milestone in the field, enabling simulations of astonishing scale and complexity.

2.2 Current Galaxy Simulation: Methodologies and Parallelization

Many sophisticated simulation models have been created in the expanding field of computational astrophysics to investigate the intricacy of galaxy formation. These models mainly adopt a dual approach, using an N-body algorithm to model the gravitational interactions of dark matter, combined with smoothed particle hydrodynamics (SPH) to capture the hydrodynamics of baryonic matter.
This section studied the simulation and parallelization techniques detailed in the galaxy simulation papers spanning from 1997 to 2020, as summarized by Vogelsberger et al. [19]., additionally collects important research from 2020 to the present, summarizes their simulation and parallelization methods, recorded in Table 1, arranged in chronological order.

By comparing and observing the contents of Table 1, it can be seen that the simulation methods of these models are mostly a combination of N-body and SPH simulations, revealing the core position of N-body simulation in simulating gravitational dynamics and the key role of SPH in simulating gas phenomena in the galactic environment.

The development of parallelization techniques in computational astrophysics reflects the maturation of computing infrastructure. In early practice, models such as ART [20] and RAMSES [21] applied shared memory systems for parallel computing, mainly through the OpenMP API.

Table 1: Major Galaxy Simulation Model and their Parallel Techniques

Code name	Simulation Method	Parallelization Technique	Year
ART [20]	Adaptive Refinement Tree N-body code	shared memory mode on an HP-Convex SPP-1200 Exemplar	1997
RAMSES [21]	N-body and hydrodynamical code	OpenMp, Domain Decomposition (under construction)	2002
GADGET-2/3 [22]	Cosmological N-body/SPH simulations	Domain decomposition, MPI, OpenMP, parallel FFT, Parallel I/O	2005
Arepo [23]	Moving mesh for hydrodynamics	MPI parallelization and dynamic load balancing	2010
Enzo [24]	N-body and hydrodynamics code	Hybrid MPI + OpenMP parallelism	2014
CHANGA [25]	N-body/SPH simulations	Charm++ for adaptive mesh refinement, over-decomposition, parallel FFT	2014
GIZMO [26]	N-body with SPH or meshless finite mass approach	Hybrid MPI+OpenMP, task-based parallelism	2015
HACC [27]	Hardware/Hybrid Accelerated Cosmology Code	MPI, parallel FFT, and parallel tessellation	2016
PKDGRAV3 [28]	N-body tree code	Multigrid solvers, parallel tree construction, MPI	2016
Gasoline2 [29]	SPH and N-body dynamics	Parallel tree with custom interaction-list based parallelism for gravity	2017
SWIFT [30]	N-body/SPH simulations	Task-based parallelism with asynchronous communication and graph partition-based domain decomposition	2023

As Message Passing Interface (MPI) technology advances, there has been a shift toward a hybrid MPI and OpenMP paradigm. Subsequent models, including GADGET-2/3 [22], Enzo [24] and GIZMO [26], adopted this hybrid approach, taking full advantage of the potential of multi-core processors.

Notably, domain decomposition has been a mainstream technique since the early 21st century and continues to demonstrate its enduring efficacy in managing complex simulations. In addition, methods such as parallel tree construction and parallel fast Fourier transform (FFT) are also widely adopted, as evidenced by their integration in GADGET-2/3 [22], CHANGA [25], HACC [27], and PKDGRAV3 [28]. These methods embody innovative advances in computational techniques aimed at optimizing large-scale astrophysics simulations and reflect the field's ongoing pursuit of improved the precision and breadth of the galaxy simulations.

2.3 High Performance Computing

High performance computing (HPC) uses supercomputers and parallel processing to solve complex computing problems. The system contains multiple processors for concurrent task execution. There are significant differences in parallel processing principles between distributed and shared-memory system architectures, which directly affect the efficiency and scalability of their complex tasks. The following two subsections dig deeper into these architectures, exploring the unique characteristics and operational dynamics of distributed and shared memory systems in the context of high performance computing.

2.3.1 Distributed Memory Systems

Distributed memory systems are characterized by a computing architecture where each processor has its own private memory. Communication among processors is achieved through a network and requires explicit message passing for data exchange. This model, as elucidated by Gropp et al. [31], inherently supports scalability to a large number of processors, making it well-suited for expansive HPC applications. The MPI (Message Passing Interface) has become the standard for programming within distributed memory environments, offering a comprehensive set of library routines to facilitate inter-process communication [32].

2.3.2 Shared Memory Systems

In contrast to distributed memory systems, shared memory systems employ a single address space accessible to all processors, allowing direct data sharing without explicit message passing. OpenMP is a widely adopted API that supports multi-threaded programming in shared memory systems, simplifying the development process by abstracting the complexities of thread management and synchronization [33]. Despite their ease of use, shared memory systems encounter limitations in scalability with an increasing number of processors, chiefly because of conflicts over memory access [34].

2.3.3 ARCHER2 and WeeArchie

ARCHER2 [35] is an advanced HPE Cray EX supercomputing system designed for high performance computing tasks, boasting an estimated peak performance of 28 Pflop/s. It features 5,860 compute nodes, each equipped with dual AMD EPYC 7742 64-core processors, culminating in a total of 750,080 cores.

As an outreach of the ARCHER2, WeeArchie [1] is a mini supercomputer designed for educational purpose. Designed for portability, it is small scale compared to ARCHER2, featuring 16 compute nodes with four cores each, thus offering a practical demonstration of supercomputing. This setup positions WeeArchie as an ideal platform for introducing the fundamentals of supercomputing to a wide audience.

3 Preliminary Investigations and Findings

3.1 Analysis of Reference Code - Galaxy Simulation

An analysis was conducted on an open-source project created by Uriot [36] that provides an N-body type simulation with GPU acceleration. This project demonstrates the capability to simulate galaxies, galaxy collisions, and expanding universes, featuring an interactive menu and camera for real-time adjustments and observations.

3.1.1 Overview of Uriot's Project

Core Components:

1. **N-body Algorithm and GPU Acceleratio**: In Uriot's framework[36], as performed in his public code, the computational core of N-body simulations, specifically the force interactions among particles, is offloaded to the GPU, using OpenCL kernels for execution. This approach is further elucidated through the developer's video exposition[37], revealing that these computational tasks are encapsulated within compact programs referred to as

"shaders." Traditionally, shaders are designed for graphical operations, primarily rendering visuals on screen, rather than conducting simulations. Hence, his application in simulation contexts, such as this, necessitates additional considerations. Uriot[37] showed that integration of more complex algorithms like Barnes-Hut for efficiency improvements in force calculation remains an area necessitating further exploration.

2. **Rendering and Visualization:** The rendering and visualization aspects of the Uriot project were designed by utilizing OpenGL for graphics rendering and custom shader programs to create visual effects. Although these components might not be the central focus of the project, they hold significant reference value for future efforts in visualizing galaxy simulations.

3.1.2 Setup Environment Challenges

The initial endeavor to explore the Galaxy Simulation project necessitated the configuration of the source code for local execution. However, this setup phase was fraught with multiple challenges. A detailed information of each encountered issue, along with its corresponding cause, implemented solution, and the rationale behind the chosen solution, is provided in the sections below. For comprehensive details on the errors encountered, refer to the project's GitLab page.

First Attempt with macOS:

- **Problem 1 - `OpenGL` Linking Error**
  - **Error Description:** Could not find the target `OpenGL::OpenGL` during the linking phase, resulting in a CMake configuration error.
  - **Root Cause:** macOS uses a slightly different naming convention and path for the `OpenGL` library compared to other operating systems, resulting in a link error.
  - **Solution:** Modified the `CMakeLists.txt` files to replace `OpenGL::OpenGL` with `OpenGL::GL`, in keeping with the macOS library naming convention.
  - **Rationale:** TThis change instructs CMake to properly link to the `OpenGL` library available on macOS, ensuring that the build process goes smoothly.
- **Problem 2 - `ImGui` Library Compatibility**
  - **Error Description:** Compilation errors related to the use of `constexpr` in the `ImGui` library, indicating a compatibility issue with the expected `C++` standard version.
  - **Root Cause:** The use of `constexpr` in the `ImGui` library suggests a C++11 or later feature that the compiler might not be recognizing or using by default.
  - **Solution:** Explicitly set the C++ standard to C++17 in the project's `CMakeLists.txt` file to ensure compatibility with the `ImGui` library.
  - **Rationale:** Specifying C++17 ensures that the compiler supports all language features used by the ImGui library, resolving the compilation errors.
- **Problem 3 - Missing OpenCL Header**
  - **Error Description:** Fatal error due to the absence of the `CL/cl.hpp` header file, which is essential for OpenCL integration.
  - **Root Cause:** The standard OpenCL headers were not present in the expected directory, likely due to an incomplete or missing `OpenCL SDK` installation.

  - **Solution:** Manually downloaded the missing `OpenCL` headers from the official OpenCL-CLHPP GitHub repository and placed them in the project's `includes/CL` directory.
  - **Rationale:** Adding the missing headers manually resolves the immediate compilation issue, allowing the project to recognize `OpenCL` functions and types.

- **Problem 4 - `OpenCL` Version Mismatch**
  - **Error Description:** Compilation errors due to undeclared identifiers within the `OpenCL` headers, suggesting a mismatch with the expected `OpenCL` version.
  - **Root Cause:** The project was attempting to use features from a newer version of `OpenCL` not supported by the installed `OpenCL` headers or the `OpenCL` version provided by the macOS environment.
  - **Solution:** Defined `OpenCL` version macros in the `libraries.hpp` file (`CL_HPP_MINIMUM_OPENCL_VERSION` and `CL_HPP_TARGET_OPENCL_VERSION`) to 120, ensuring compatibility with available `OpenCL` features.
  - **Rationale:** The `OpenCL` version macros within the header files are set using an integer format where the version number 120 corresponds to `OpenCL` version 1.2. This approach prevents the use of undeclared or unsupported identifiers, thereby ensuring that the code remains compatible with the targeted `OpenCL` version.

- **Problem 5 - Shader Compilation Issue**
  - **Error Description:** There was a problem initializing `cl::Program::Sources`, causing the shader to compile incorrectly.
  - **Root Cause:** The syntax or method used to initialize `cl::Program::Sources` is incorrect, resulting the shader program cannot be compiled correctly.
  - **Solution:** Revised the shader initialization code by creating an instance of `cl::Program::Sources` and then adding the shader source directly to this instance before creating the program.
  - **Rationale:** This method ensures that the shader source is correctly passed to the `OpenCL` program, allowing for successful compilation and linking of the shader programs.

- **Problem 6 - `SFML` Architecture Mismatch**
  - **Error Description:** Linker warnings and errors indicating an architecture mismatch for `SFML` libraries, which were compiled for `x86_64` architecture instead of the required `arm64`.
  - **Root Cause:** The `SFML` libraries used were not compatible with the `arm64` architecture of the macOS, leading to linkage failures.
  - **Solution:** Updated the project to use `SFML` version 2.6.1, sourced through `Homebrew`, which provides the correct architecture support. This involved modifying the `find_package` command in the `CMakeLists.txt` file to locate and link against the appropriate `SFML` libraries.
  - **Rationale:** Ensuring that all libraries are compiled for the same architecture as the target system is crucial for successful linkage and execution. Using Homebrew allows for automatic handling of these architecture-specific dependencies.

- **Execution Attempt and `OpenGL` Compatibility Issue** Upon resolving the compilation and linkage issues, the application was successfully built. However, executing the built application revealed runtime issues related to `OpenGL` compatibility.

- **Error Description:** Runtime warnings indicated that the created `OpenGL` context did not fully meet the requested settings, specifically highlighting a version mismatch where version 4.1 was created instead of the requested 4.3.
- **Root Cause:** The macOS environment, along with the hardware capabilities and available drivers, could not provide an `OpenGL` 4.3 context, instead defaulting to the highest supported version, 4.1.
- **Solution:** During the search for solutions, it was discovered that macOS would not update the supported `OpenGL` version, meaning the highest version that could be used was 4.1. If a higher version was needed, the introduction of third-party support might be necessary. Ultimately, based on the supervisors' recommendation, the use of a virtual machine (VM) to run the project was attempted as an alternative to executing it on the local computer.

Second Attempt with EIDF (Edinburgh International Data Facility) VM: In transitioning to a Linux-based EIDF VM, the required packages were installed, addressing the issues previously encountered on macOS. Like macOS, the `OpenGL` version on the VM was 4.5, which does not match as it is higher than needed. Considering the difficulty in degrading the `OpenGL` version, the feasibility of upgrading the project's `OpenGL` requirement was explored. It was determined that the application required `OpenGL` 4.3, potentially due to the `GLEW` library. Unfortunately, efforts to reinstall an older version of `GLEW` were unsuccessful and during the reinstallation process of `GLEW`, critical components were accidentally deleted, leading to the death of the VM.

Third Attempt with New EIDF VM: Learning from the previous attempt, a conda virtual environment was established to minimize the use of `sudo` and reduce the risk of system-wide alterations. This time, however, `OpenGL` could not even be found on Ubuntu, and attempts with the likely required packages failed to resolve the issue. After discussions with two supervisors, it was decided to pivot away from `OpenGL` dependency at this stage due to persistent compatibility issues. The research focus was adjusted accordingly, which will be detailed in the subsequent section.

Revised Research Plan The initial research objective was to run Uriot's galaxy simulation project locally [36], to delve into how such a simulation is realized and visualized. However, a series of `OpenGL`-related issues during the setup phase, necessitating a subsquent alteration of the project plan. Given that the impact of `OpenGL` was limited to visualization and that galaxy simulations could be developed independently, it was decided to refocus the research. The first priority now is to understand Uriot's approach [36] to simulating galaxies. The aim is to develop a basic version of an N-body galaxy simulation and then try to implement parallel computing using MPI. This will enable the project to run on Wee Archie, which runs on distributed systems. Subsequently, efforts will be made to see if combining OpenMP can further optimize computational speed. For the detailed plan, refer to section 5.

4 Project Proposal

This project aims to address the challenge of implementing N-body galaxy simulations for distributed memory system, specifically focusing on the mini-supercomputer WeeArchie. A core goal of this research is to take advantage of parallel computing to significantly improve the efficiency and scalability of galaxy simulations. The application of parallel computing is expected to play a key role in refining computational astrophysics models, increasing the speed

and complexity of simulations. This study will demonstrate the noteworthy improvements in computational performance that can be achieved by parallelizing the system development and subsequent analysis of N-body galaxy simulations. This will involve detailed benchmarking against current non-parallel models to quantify improvements in processing speed and simulation accuracy, thereby solidifying the key role of parallel computing in this research.

4.1 Success Criteria

Success in this project will be defined by the ability to:

- Construct a reliable basic N-body galaxy simulation model.
- Develop a parallelized version of an N-body galaxy simulation that runs efficiently on WeeArchie.
- Demonstrate a significant improvement in computational performance compared to non-parallel versions, as measured by processing speed and resource utilization.

The "minimum viable product" will consist of a basic parallelized simulation that demonstrates improved performance over serial implementations, even if optimizations are not fully realized.

If the project goals are achieved early, an additional objective will be to realize a simple visualization of the simulation results, which can be demonstrated on Wee Archie.

5 Workplan

This research will start with an in-depth study of Uriot's galaxy simulation methods [36], with the goal of developing a basic version of N-body galaxy simulations. The subsequent phase will explore the application of MPI (Message Passing Interface) to implement parallel computing capabilities, enabling the simulation to run on the Wee Archie, which utilizes a distributed system. Further down the line, attempts will be made to integrate OpenMP to potentially optimize computational performance and efficiency.

This pivot reflects a targeted approach, allowing the research to progress in areas unconstrained by the previous graphical issues. By separating the visualization from the computational aspects, the project maintains its agility and advances in the exploration and enhancement of the core simulation processes. The research will proceed as follows:

Phase 1: Simulation Study Analysis

- **Literature Review**: Start with a thorough literature review to understand the current state of N-body simulations in astrophysics.
- **Codebase Examination**: Dive into Uriot's simulation codebase [36] and examine the structure, algorithms, and methods used.
- **Algorithm Understanding**: Identify and understand the key algorithms driving the simulation, noting any areas that could benefit from parallelization or optimization.

Phase 2: Basic N-body Simulation Development

- **Simplified Model Design**: Design a simplified model of the N-body simulation, focusing on core dynamics and interactions.
- **Coding and Testing**: Develop the simplified simulation code, ensuring that basic functionalities are correctly implemented.

Phase 3: Parallel Computing Implementation

- **MPI Basics**: Gain a comprehensive understanding of MPI and its application in parallel computing.
- **MPI Integration**: Integrates MPI into a simplified N-body simulation to distribute tasks across multiple processors, ensuring that this can be performed on Wee Archie.
- **Efficiency Testing**: Conduct tests to evaluate the efficiency and scalability of the MPI implementation, making adjustments as needed.

Phase 4: Performance Optimization

- **OpenMP Familiarization**: Learn about OpenMP and its potential for optimizing parallel computations.
- **OpenMP Integration**: Apply OpenMP to the MPI-enabled simulation code to enhance performance on multi-core processors.
- **Optimization Testing**: Test the simulation for improved performance metrics, identifying and implementing further optimizations where possible.

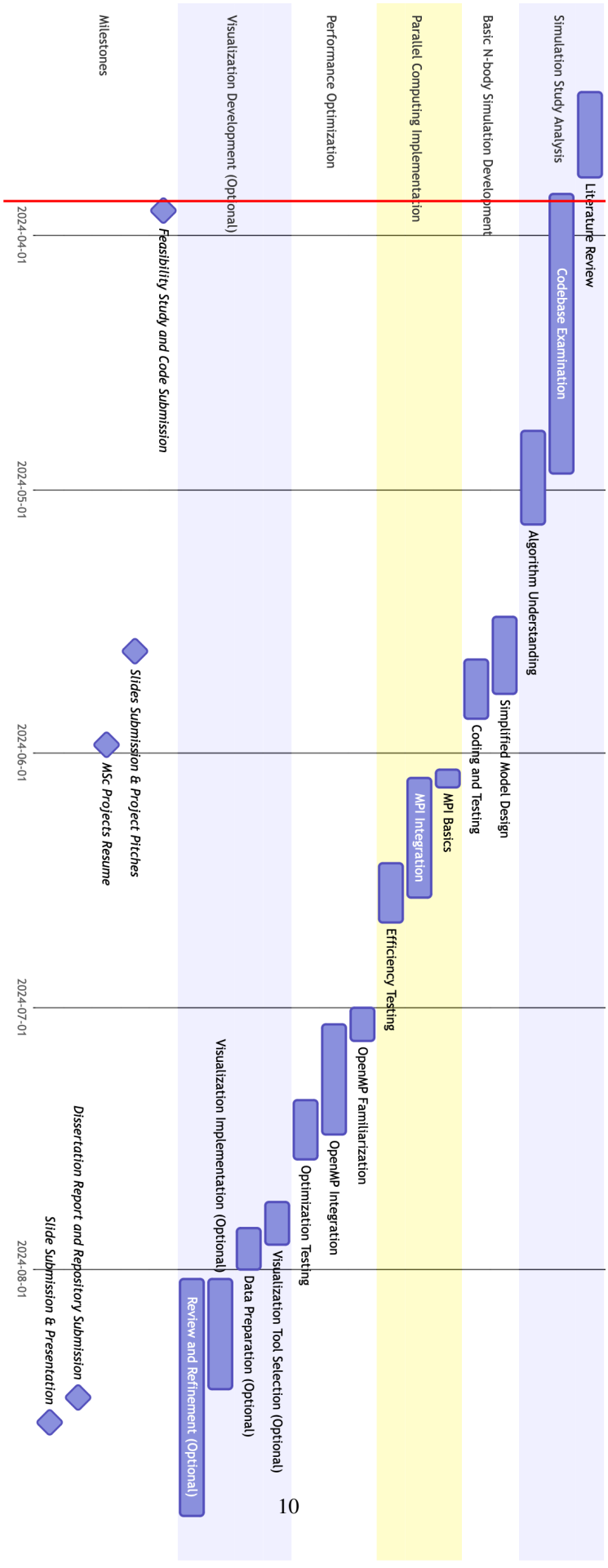
Phase 5: Visualization Development (Optional)

- **Visualization Tool Selection**: If time permits, select appropriate visualization tools compatible with Wee Archie.
- **Data Preparation**: Prepare simulation output data for visualization, ensuring it is in a format suitable for the chosen tools.
- **Visualization Implementation**: Develop visualization scripts or utilize existing tools to generate graphical representations of the simulation results.
- **Review and Refinement**: Review the visualizations for accuracy and clarity, making refinements to better convey the simulation outcomes.

5.1 Timeline and Milestones

The Gantt chart in Figure 1 presented on the following page outlines the planned phases of work as previously detailed. It incorporates corresponding milestones, which are represented by diamond shapes on the timeline. A red horizontal line indicates the current stage within the overall project timeline, providing a visual marker of progress to date. *This Gantt chart is created with mermaid.*

Figure 1: Gantt chart of N-body simulation development and optimization project



6 Resources Estimation

6.0.1 Computing Resources

For this project, the primary computational resource will be the Wee Archie mini-supercomputer. Given the specialized nature of the simulations and the efficiency of parallel computing techniques employed, Wee Archie is expected to fulfill all computational needs without the requirement for additional CPU/GPU resources.

6.0.2 Backup and Redundancy

Given the best practices of computing projects, backup systems are essential to ensure continuity in the event that Wee Archie is affected during system failures or maintenance. For this purpose, the EIDF (Edinburgh International Data Facility) VM (Virtual machine) will act as a backup. EIDF VM provides a robust and reliable backup solution that ensures project work can continue without interruption, thereby minimizing the risk associated with potential downtime.

7 Risk Analysis

Risk analysis is an important part of project management. This section outlines the potential risks that could hinder the progress of the project, assesses their likelihood and potential impact, and proposes mitigation strategies and corrective plans to effectively manage these risks.
Each risk is detailed in the following table (table 2) along with the corresponding mitigation strategies and corrective plans. This proactive approach ensures that the project is capable of addressing challenges and adapting to unforeseen events, thus enhancing the resilience and robustness of the project management process.

Table 2: Risk assessment and mitigation strategies for the galaxy simulation project.

Risk	Probability	Impact	Mitigation Strategy	Correction Plan
MPI Implementation Complexity	Probable	Moderate	Schedule more time for implementation	Refer to course material of AMPP; regular consultations
Inadequate OpenMP Optimization	Probable	Moderate	Early optimization and testing	Check course material of TP; Seeking expert advice
Wee Archie Unavailability	Possible	Severe	Have backup resources	EIDF VM
Scope Creep	Possible	Moderate	Strict change control	Regularly review project scope and objectives
Milestone Delays	Probable	Moderate	Implement progress tracking	Adjust project timeline; Increase working hours
Missed Academic Deadlines	Possible	High	Create detailed work schedule	Allocate tasks; Set interim deadlines

References

[1] EPCC, "Wee archie," https://www.archer2.ac.uk/community/outreach/materials/wee_archie, (Accessed on 03/23/2024).

[2] E. Holmberg, "On the Clustering Tendencies among the Nebulae. II. a Study of Encounters Between Laboratory Models of Stellar Systems by a New Integration Procedure." , vol. 94, p. 385, Nov. 1941.

[3] S. J. Aarseth, "Dynamical evolution of clusters of galaxies, I," , vol. 126, p. 223, Jan. 1963.

[4] J. Barnes and P. Hut, "A hierarchical O(N log N) force-calculation algorithm," , vol. 324, no. 6096, pp. 446–449, Dec. 1986.

[5] L. Greengard and V. Rokhlin, "A fast algorithm for particle simulations," *Journal of Computational Physics*, vol. 73, no. 2, pp. 325–348, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0021999187901409

[6] R. W. Hockney and J. W. Eastwood, *Computer simulation using particles*. USA: Taylor & Francis, Inc., 1988.

[7] A. Brandt, "Multi-level adaptive solutions to boundary-value problems," *Mathematics of Computation*, vol. 31, no. 138, pp. 333–390, 1977. [Online]. Available: http://www.jstor.org/stable/2006422

[8] O. Axclsson and P. S. Vassilevski, "Algebraic multilevel preconditioning methods. i," *Numer. Math.*, vol. 56, no. 2–3, p. 157–177, feb 1989. [Online]. Available: https://doi.org/10.1007/BF01409783

[9] M. Davis, G. Efstathiou, C. S. Frenk, and S. D. M. White, "The evolution of large-scale structure in a universe dominated by cold dark matter," , vol. 292, pp. 371–394, May 1985.

[10] G. Efstathiou, M. Davis, S. D. M. White, and C. S. Frenk, "Numerical techniques for large cosmological N-body simulations," , vol. 57, pp. 241–260, Feb. 1985.

[11] M. Trenti and P. Hut, "N-body simulations (gravitational)," *Scholarpedia*, vol. 3, no. 5, p. 3930, 2008, revision #91544.

[12] H. Baumgardt and J. Makino, "Dynamical evolution of star clusters in tidal fields," *Monthly Notices of the Royal Astronomical Society*, vol. 340, no. 1, pp. 227–246, 03 2003. [Online]. Available: https://doi.org/10.1046/j.1365-8711.2003.06286.x

[13] V. Springel, S. D. M. White, A. Jenkins, C. S. Frenk, N. Yoshida, L. Gao, J. Navarro, R. Thacker, D. Croton, J. Helly, J. A. Peacock, S. Cole, P. Thomas, H. Couchman, A. Evrard, J. Colberg, and F. Pearce, "Simulations of the formation, evolution and clustering of galaxies and quasars," , vol. 435, no. 7042, pp. 629–636, Jun. 2005.

[14] R. A. Gingold and J. J. Monaghan, "Smoothed particle hydrodynamics: theory and application to non-spherical stars." , vol. 181, pp. 375–389, Nov. 1977.

[15] L. B. Lucy, "A numerical approach to the testing of the fission hypothesis." , vol. 82, pp. 1013–1024, Dec. 1977.

[16] J. F. Navarro and S. D. White, "Simulations of dissipative galaxy formation in hierarchically clustering universes–ii. dynamics of the baryonic component in galactic haloes," *Monthly Notices of the Royal Astronomical Society*, vol. 267, no. 2, pp. 401–412, 1994.

[17] V. Springel and L. Hernquist, "Cosmological smoothed particle hydrodynamics simulations: the entropy equation," *Monthly Notices of the Royal Astronomical Society*, vol. 333, no. 3, pp. 649–664, 2002.

[18] V. Springel, N. Yoshida, and S. D. White, "Gadget: a code for collisionless and gasdynamical cosmological simulations," *New Astronomy*, vol. 6, no. 2, pp. 79–117, 2001.

[19] M. Vogelsberger, F. Marinacci, P. Torrey, and E. Puchwein, "Cosmological simulations of galaxy formation," *Nature Reviews Physics*, vol. 2, no. 1, pp. 42–66, 2020.

[20] A. V. Kravtsov, A. A. Klypin, and A. M. Khokhlov, "Adaptive refinement tree: A new high-resolutionn-body code for cosmological simulations," *The Astrophysical Journal Supplement Series*, vol. 111, no. 1, p. 73–94, Jul. 1997. [Online]. Available: http://dx.doi.org/10.1086/313015

[21] R. Teyssier, "Cosmological hydrodynamics with adaptive mesh refinement: A new high resolution code called ramses," *Astronomy amp; Astrophysics*, vol. 385, no. 1, p. 337–364, Apr. 2002. [Online]. Available: http://dx.doi.org/10.1051/0004-6361:20011817

[22] V. Springel, "The cosmological simulation code gadget-2," *Monthly Notices of the Royal Astronomical Society*, vol. 364, no. 4, p. 1105–1134, Dec. 2005. [Online]. Available: http://dx.doi.org/10.1111/j.1365-2966.2005.09655.x

[23] ——, "E pur si muove:galilean-invariant cosmological hydrodynamical simulations on a moving mesh," *Monthly Notices of the Royal Astronomical Society*, vol. 401, no. 2, p. 791–851, Jan. 2010. [Online]. Available: http://dx.doi.org/10.1111/j.1365-2966.2009.15715.x

[24] G. L. Bryan, M. L. Norman, B. W. O'Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman, B. Smith, R. P. Harkness, J. Bordner, J.-h. Kim, M. Kuhlen, H. Xu, N. Goldbaum, C. Hummels, A. G. Kritsuk, E. Tasker, S. Skory, C. M. Simpson, O. Hahn, J. S. Oishi, G. C. So, F. Zhao, R. Cen, and Y. Li, "Enzo: An adaptive mesh refinement code for astrophysics," *The Astrophysical Journal Supplement Series*, vol. 211, no. 2, p. 19, Mar. 2014. [Online]. Available: http://dx.doi.org/10.1088/0067-0049/211/2/19

[25] H. Menon, L. Wesolowski, G. Zheng, P. Jetley, L. Kale, T. Quinn, and F. Governato, "Adaptive techniques for clustered n-body cosmological simulations," 2014.

[26] P. F. Hopkins, "A new class of accurate, mesh-free hydrodynamic simulation methods," *Monthly Notices of the Royal Astronomical Society*, vol. 450, no. 1, p. 53–110, Apr. 2015. [Online]. Available: http://dx.doi.org/10.1093/mnras/stv195

[27] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Lukić, S. Sehrish, and W. keng Liao, "Hacc: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49–65, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138410761500069X

[28] D. Potter, J. Stadel, and R. Teyssier, "Pkdgrav3: Beyond trillion particle cosmological simulations for the next era of galaxy surveys," 2016.

[29] J. W. Wadsley, B. W. Keller, and T. R. Quinn, "Gasoline2: a modern smoothed particle hydrodynamics code," *Monthly Notices of the Royal Astronomical Society*, vol. 471, no. 2, p. 2357–2369, Jun. 2017. [Online]. Available: http://dx.doi.org/10.1093/mnras/stx1643

[30] M. Schaller, J. Borrow, P. W. Draper, M. Ivkovic, S. McAlpine, B. Vandenbroucke, Y. Bahé, E. Chaikin, A. B. G. Chalk, T. K. Chan, C. Correa, M. van Daalen, W. Elbers, P. Gonnet, L. Hausammann, J. Helly, F. Huško, J. A. Kegerreis, F. S. J. Nobels, S. Ploeckinger, Y. Revaz, W. J. Roper, S. Ruiz-Bonilla, T. D. Sandnes, Y. Uyttenhove, J. S. Willis, and Z. Xiang, "Swift: A modern highly-parallel gravity and smoothed particle hydrodynamics solver for astrophysical and cosmological applications," 2023.

[31] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[32] D. W. Walker and J. J. Dongarra, "Mpi: a standard message passing interface," *Supercomputer*, vol. 12, pp. 56–68, 1996.

[33] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: portable shared memory parallel programming*. MIT press, 2007.

[34] R. Rabenseifner, G. Hager, and G. Jost, "Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes," in *2009 17th Euromicro international conference on parallel, distributed and network-based processing*. IEEE, 2009, pp. 427–436.

[35] EPCC, "Archer2 hardware & software," https://www.archer2.ac.uk/about/hardware.html, (Accessed on 03/23/2024).

[36] A. Uriot, "Github - angeluriot/galaxy_simulation: An n-body type simulation using gpu acceleration to simulate galaxies, galaxy collisions and expanding universes." https://github.com/angeluriot/Galaxy_simulation, 2023, (Accessed on 03/10/2024).

[37] D. CODE, "Simuler 1 000 000 de galaxies ! - youtube," https://www.youtube.com/watch?v=dFqjqRUWCus, (Accessed on 03/29/2024).