Master in High Performance Computing

GInX: A Geodesics Integrator tool for particles in GRMHD using Adaptive Mesh Refinement

Supervisors:
Manuela Campanelli,
Yosef Zlochower,
Irina Davydenkova

Candidate:
Jhon Sebastián Moreno Triana

11th edition
2024–2025

Acknowledgments

I would like to thank SISSA and ICTP, the MHPC's staff and professors, for making this possible. Being here at Trieste has been an important step in my professional and personal development. Thanks to Ivan, Axel, Moreno, Fernando, Lucca, and the others for putting all their effort into transmitting their knowledge to us.

Thanks to professors Yosef and Manuela, and Liwei, for trusting me and being present at each step of the project. I have leaned a lot from this project and I hope to keep learning and growing with you. Thanks to the entire RIT group for accepting me and for offering a space to discuss and present the project. Thanks to professor Roland Haas for taking his time to review the present work.

Special thanks to Irina, for following us at each step in the program and the project. I hope I'm not a kindergarten kid anymore. Also thanks to her husband for taking the time to review parts of the present work.

Thanks to the people I have met during my time here in Trieste, who have helped me feel at home and get through the program. Especially, Ronald, Weiner and Isaí, I hope we can meet anytime soon back home. To my classmates Nicola, Ludwig, André, Abou Bakr (Bellou) and Nisha, I wish you all the best.

Special thanks to Giacomo Zuccarino, Christian Tica, Gustavo Paredes Torres and Laura Zamora Cabrera. I'm sure they will do well at anything they set out to do. Take your time, one problem at a time, and be patient.

Thanks to all my closest friends, who are scattered around the world: Carlos Julio Ramos Salas (CJ), Juan Bueno Ramirez, Luisa Velasquez Malagón (amigui) and Daniela Piraquive Abaunza. You really helped me to get through this experience. It is always valuable for me to know I can count on you at any time.

Last and most important, my family: my sister Yuri Milena Moreno Triana, my parents Sandra Patricia Triana Triana and Eduardo Moreno Rincón, and my brother-in-law Wilson Alzate Calderón. Everything is for you, and thanks to you.

Gracias totales. Jhon Sebastián Moreno Triana.

Abstract

The integration of null geodesic equations within the 3+1 formalism of general relativity plays a key role in the process of computing images via ray tracing and analyzing spectra in GRMHD simulations. The objective of this project is to develop a geodesic integrator for particles in a generic GRMHD background using the Fast-Light approximation with Adaptive Mesh Refinement (AMR), and to accelerate the integration with GPU computing. The implementation will utilize the C++ programming language in conjunction with the AMReX framework, which has been built with GPU support enabled. The project will focus both on the accurate numerical integration of null geodesic and on evaluating GPU-accelerated performance. The purpose of this tool is to serve as a post-processing module for the visualization and analysis of GRMHD simulation data. Additionally, it represents a critical first step toward the development of a full Monte Carlo radiation transport code, where fast and efficient geodesic integration is essential.

Keywords: Particle geodesics, GRMHD simulations, Ray tracing in curved spacetime, Monte Carlo radiation transport, GPU acceleration, Adaptive Mesh Refinement (AMR), AMReX framework, Parallel numerical integration

Contents

Chapter 1

Introduction

General Relativity is the theory that describes the effects of gravity through the curvature produced by matter and energy in a four-dimensional spacetime manifold. This description of gravity has introduced new physical ideas about the nature of the universe. Some of the predictions made by General Relativity are the Mercury's anomalous precession, the existence of black holes, gravitational lensing, and gravitational waves. The theory can be expressed by using the tools of differential geometry, where the basic object to represent a manifold is the metric g_{ab}. The Einstein equations describe the dynamics of the four-dimensional spacetime metric and how it is deformed by a given mass-energy distribution. On the other hand, energy and linear momentum conservation can be expressed in the language of differential geometry, which describes how matter moves in a curved spacetime.

Numerical relativity is a field aimed at numerically solving physical problems of general relativity on supercomputers. This includes the magnetohydrodinamical and astrophysical problems in strong-gravity regimes where no other approximations are available. Most of the numerical relativity codes use adaptive mesh refinement algorithms to ensure the simulations are accurate and practical, and they leverage high-performance computing to solve problems efficiently. A strong motivation for the expansion of numerical relativity has been the development of gravitational wave detectors, which offer direct information about the sources powering these phenomena.

General relativistic magnetohydrodynamics (GRMHD) simulations remain an indispensable tool for probing the intricate dynamics of relativistic plasmas in the vicinity of compact astrophysical sources, such as binary neutron stars (BNS), binary black hole-neutron star systems (BHNS), and binary black holes (BBH). However, GRMHD simulations alone do not produce observable quantities; they must be coupled with ray-tracing to generate synthetic images and spectra that can be compared directly with observations [1]

For instance, the X-ray profiles of spinning neutron stars, which originate from their surface, are affected by the gravitational field of the star [2]. Moreover, the bending of light by massive objects like galaxy clusters can create brightness amplification [3], deformed images, or even multiple images of background objects such as quasars [4]. These effects also influence the emission spectra produced in black hole accretion disks, in addition to relativistic redshift and Doppler effects. To theoretically reproduce the spectrum and light curves of these sources with high precision requires accurate general-relativistic ray-tracing and radiative transfer codes.

On the other hand, the detection of the electromagnetic (EM) emission from astrophysical systems aids in the detection of gravitational waves through direct evidence. In particular, the identification of photons coming from supermassive binary black holes systems (SMBBHs) by current EM telescopes will help refine our estimates of their population and evolution. However, to accurately predict distinct EM signatures from SMBBHs, we must understand how they behave, and to accomplish this we rely on numerical simulations that can handle all of the relevant physical ingredients. Some work has been done in this field, such as the studies presented in [5] and [6]. Some of the limitations of these works are related to the models and approximations used, as well as the computational constraints that do not allow the simulations to evolve for sufficient time.

Given the computational demands of GRMHD simulations, modern codes are being developed on infrastructures that support graphics processing units (GPUs). One of the frameworks available for the parallel management on both GPU and CPU architectures is the C++ framework AMReX [7, 8]. For this reason, developing a performant and accurate geodesic integration code able to manage a large particle counts, complex spacetimes, and adaptive mesh refinement is a critical first step towards the development of a Monte Carlo radiation transport code and ray-tracing algorithms for post-processing visualization and analysis.

GInX is a null geodesic integrator using the fast light approximation, which allows to integrate timelike geodesic on static spacetimes or snapshots of dynamic ones. This code is written in C++17, is open source, and can be accessed from [9], with the documentation available at [10]. It uses barycentric Lagrange interpolation and a fourth-order Runge-Kutta method to accurately integrate the geodesics. The code's accuracy has been verified using conserved quantities in Schwarzschild and Kerr spacetimes. The code uses AMReX to enable parallel execution on both GPU and CPU architectures and to support Adaptive Mesh Refinement (AMR). Its performance was evaluated by measuring strong and weak scaling on CINECA's Leonardo, TACC Frontera and Vista clusters. It is designed to be easily integrated with others Einstein Toolkit thorns.

This thesis is organized as follows:

- **Chapter 2** introduces the necessary theoretical background for solving the physical problem, including the 3+1 formalism and geodesic equations.
- **Chapter 3** introduces the necessary theoretical background for the numerical methods used in the simulation, including finite element methods, adaptive mesh refinement, Runge-Kutta methods, and barycentric Lagrange interpolation.
- **Chapter 4** describes the software ecosystem, including Cactus, the Einstein Toolkit, and AMReX.
- **Chapter 5** presents the code structure, implementation, and capabilities.
- **Chapter 6** validates the code through convergence tests and conserved quantity tracking for both the Schwarzschild and Kerr spacetimes.
- **Chapter 7** evaluates the performance on HPC clusters, including scaling studies and AMR overhead.
- **Chapter 8** concludes the thesis and discusses future work.

Chapter 2

Geodesic Equations in 3+1 Decomposition

2.1 3+1 Formalism

The 3+1 formalism is an approach to general relativity that relies on the slicing of the four-dimensional spacetime by three-dimensional spacelike hypersurfaces, serving as foundation of Hamiltonian formulations of general relativity by Paul A.M. Dirac, and Richard Arnowitt, Stanley Deser and Charles W. Misner (ADM). Some open-source GRMHD codes uses the 3+1 formalism to evolve their simulations, for instance, AsterX for dynamical spacetimes [11]. Moreover, this approach allows us to rewrite the geodesic equations so they can be solved using spacetimes computed with the same formalism. See [12] for a more detailed discussion.

2.1.1 Hypersurfaces

Let $(\mathcal{M}, g)$ be a 4-dimensional spacetime endowed with a metric g of signature $(-,+,+,+)$. A hypersurface Σ is a three-dimensional submanifold that can be either timelike, spacelike, or null [13]. This submanifold has an induced metric γ associated with it, also called the first fundamental form of Σ. The hypersurface Σ is said to be

- **spacelike** if and only if γ is Riemannian, i.e., has signature (+,+,+);
- **timelike** if and only if γ is Lorentzian, i.e., has signature (-,+,+);
- **null** if and only if γ is degenerate, i.e., has signature (0,+,+).

In this case, for the 3+1 formalism, we are interested in spacelike hypersurfaces.

2.1.1.1 Normal Vector

Given a scalar field t on $\mathcal{M}$ such that the spatial hypersurface Σ is defined as a level surface of t and ∇ is the Levi-Civita connection (covariant derivative) associated with g, ∇t is normal to Σ, i.e., for every vector v tangent to Σ, $\langle \nabla t, v \rangle = g(\nabla t, v) = 0$. In our case, we can re-normalize $\nabla^\mu t$ to make it a unit vector by setting

$$n^\mu = (-\nabla^\nu t \nabla_\nu t)^{-\frac{1}{2}} \nabla^\mu t. \tag{2.1}$$

which satisfies

$$n^\mu n_\mu = -1. \tag{2.2}$$

2.1.1.2 The Orthogonal Projector

The orthogonal projector onto Σ is the operator γ that satisfies

$$\gamma^\mu{}_\nu n^\nu = \gamma^\mu{}_\nu n_\mu = 0. \tag{2.3}$$

The orthogonal projector is

$$\gamma^\mu{}_\nu = \delta^\mu{}_\nu + n^\mu n_\nu. \tag{2.4}$$

If we project the metric g using the projector $\gamma^\mu{}_\nu$, we can obtain the induced metric $\gamma_{\mu\nu}$,

$$\gamma_{\mu\nu} = g_{\mu\nu} + n_\mu n_\nu. \tag{2.5}$$

In addition, we can project a tensor T of type $\binom{p}{q}$ from $\mathcal{M}$ to the hypersurface Σ by using $\gamma^\mu{}_\nu$ on all indices:

$${}^3T^{\alpha_1 \ldots \alpha_p}{}_{\beta_1 \ldots \beta_q} = \gamma^{\alpha_1}{}_{\mu_1} \ldots \gamma^{\alpha_p}{}_{\mu_p} \gamma^{\nu_1}{}_{\beta_1} \ldots \gamma^{\nu_q}{}_{\beta_q} {}^4T^{\mu_1 \ldots \mu_p}{}_{\nu_1 \ldots \nu_q}. \tag{2.6}$$

2.1.1.3 Intrinsic Curvature

If the hypersurface Σ is spacelike or timelike, then the induced metric γ is not degenerate. This implies that there is a unique connection (or covariant derivative) D on the manifold Σ that is torsion-free [12] and satisfies

$$D_\mu \gamma_{\nu\sigma} = 0. \tag{2.7}$$

D is the Levi-Civita connection associated with the induced metric γ. This is connected with the Levi-Civita connection associated with the metric g, ∇, via the orthogonal projector, i.e.

$$D_\rho T^{\alpha_1 \ldots \alpha_p}{}_{\beta_1 \ldots \beta_q} = \gamma^{\alpha_1}{}_{\mu_1} \ldots \gamma^{\alpha_p}{}_{\mu_p} \gamma^{\nu_1}{}_{\beta_1} \ldots \gamma^{\nu_q}{}_{\beta_q} \gamma^\sigma{}_\rho \nabla_\sigma T^{\mu_1 \ldots \mu_p}{}_{\nu_1 \ldots \nu_q}. \tag{2.8}$$

Now, we can define the curvature tensor in the same way we can define the Riemann tensor. Then

$$[D_i, D_j]v^k = {}^3R^k{}_{lij}v^l, \tag{2.9}$$

where $[a, b] = ab - ba$ is the commutator and v^k is a tangent vector of Σ. The Ricci tensor is then ${}^3R_{ij} = {}^3R^k{}_{ikj}$ and the Ricci scalar is ${}^3R = \gamma^{ij}\,{}^3R_{ij}$ [12].

2.1.1.4 Extrinsic Curvature

We can also define the extrinsic curvatures related to the "bending" of Σ in $\mathcal{M}$. This corresponds to the change of direction of the normal vector n as one moves on Σ. The extrinsic curvature, or second fundamental form of Σ, is defined as follows:

$$K_{\mu\nu} = -\gamma^\sigma{}_\mu \gamma^\lambda{}_\nu \nabla_\sigma n_\lambda, \tag{2.10}$$

and its trace is

$$K = -\nabla^\mu n_\mu. \tag{2.11}$$

We can also define the 4-acceleration as

$$a^\mu = n^\nu \nabla_\nu n^\mu. \tag{2.12}$$

One can verify that a^μ is orthogonal to n^μ, i.e., $a^\nu n_\nu = 0$.

Using the relation between γ and g and substituting into equation (2.10), we obtain

$$\begin{aligned} K_{\mu\nu} &= -(\delta^\lambda_\mu + n^\lambda n_\mu)(\delta^\sigma_\nu + n^\sigma n_\nu)\nabla_\lambda n_\sigma \\ &= -\nabla_\mu n_\nu - n^\sigma n_\nu \nabla_\mu n_\sigma - n^\lambda n_\mu \nabla_\lambda n_\nu - n^\lambda n_\mu n^\sigma n_\nu \nabla_\lambda n_\sigma \quad (n^\sigma \nabla_\mu n_\sigma = 0, n^\sigma n^\lambda \nabla_\lambda n_\sigma = 0) \\ &= -\nabla_\mu n_\nu - n_\mu n^\lambda \nabla_\lambda n_\nu \\ &= -\nabla_\mu n_\nu - n_\mu a_\nu. \end{aligned} \tag{2.13}$$

Using the fact that $K_{\mu\nu} = K_{\nu\mu}$ and reordering the equation, we obtain the following relation:

$$\nabla_\nu n_\mu = -K_{\mu\nu} - a_\mu n_\nu. \tag{2.14}$$

One can also show that $a^\mu = D^\mu \ln \alpha$ (see [12]), where $\alpha = (-\nabla^\nu t \nabla_\nu t)^{-\frac{1}{2}}$. Therefore, we can rewrite (2.14) as:

$$\nabla_\nu n_\mu = -K_{\mu\nu} - D_\mu(\ln \alpha) n_\nu. \tag{2.15}$$

2.1.2 Foliations

A foliation (or slicing) means that there exists a smooth and regular scalar field t on $\mathcal{M}$ such that each hypersurface is a level surface of the scalar field:

$$\forall t_i \in \mathbb{R}, \quad \Sigma_t = \{p \in \mathcal{M}, t(p) = t_i\}. \tag{2.16}$$

Since t is regular, the hypersurfaces Σ_t are non-intersecting [12].

This is true for any globally hyperbolic spacetime $(\mathcal{M}, g)$, which can be foliated by a family of spacelike hypersurfaces $(\Sigma_t)_{t\in\mathbb{R}}$. Also, we will assume that

$$\mathcal{M} = \bigcup_{t\in\mathbb{R}} \Sigma_t. \tag{2.17}$$

2.1.2.1 Lapse Function

As defined in Section 2.1.1.1, the unit vector pointing in the time direction is

$$\vec{n} = -\alpha\vec{\nabla}t, \tag{2.18}$$

with

$$\alpha = \left(-\vec{\nabla}t \cdot \vec{\nabla}t\right)^{-\frac{1}{2}}. \tag{2.19}$$

The scalar field α is called the lapse function.

2.1.3 Normal Evolution Vector

The normal evolution vector is defined as

$$\vec{m} = \alpha\vec{n}. \tag{2.20}$$

We then have

$$\nabla_m t = m^\mu \nabla_\mu t = \alpha n^\mu \nabla_\mu t = \alpha^2 \nabla^\mu t \nabla_\mu t = \alpha^2\alpha^{-2} = 1. \tag{2.21}$$

This relation means that the vector $\vec{m}$ is "adapted" to the scalar field t. It also implies that we can obtain $\Sigma_{t+\delta t}$ by displacing Σ_t by $\delta t\vec{m}$. In other words, the hypersurfaces (Σ_t) are Lie-dragged by the vector $\vec{m}$ (see [12]).

2.1.4 Shift Vector

We can define a coordinate system adapted to the foliation $(\Sigma_t)_{t\in\mathbb{R}}$. Let's define on each hypersurface Σ_t some coordinate system $(x^i) = (x^1, x^2, x^3)$. If this system varies smoothly between hypersurfaces, then $(x^\mu) = (t, x^i)$ is a well-behaved coordinate system on $\mathcal{M}$.

Then, the natural basis of the tangent space of $\mathcal{M}$ associated with the coordinates (x^μ) is $(\partial_\mu) = (\partial_t, \partial_i)$. The difference between ∂_t and $\vec{m}$ is called the shift vector and is denoted β:

$$(\partial_t)^\mu = m^\mu + \beta^\mu. \tag{2.22}$$

The shift vector satisfies

$$n_\mu \beta^\mu = 0. \tag{2.23}$$

Using the definition (2.20), we obtain

$$(\partial_t)^\mu = \alpha n^\mu + \beta^\mu. \tag{2.24}$$

Since n is normal to Σ_t and β is tangent to Σ_t, this can be seen as a 3+1 decomposition of ∂_t onto Σ_t. Using the previous decomposition, we can get the components of n^α; we can easily check that

$$n^\mu = \frac{(\partial_t)^\mu - \beta^\mu}{\alpha} = \frac{1}{\alpha}\left(1, -\beta^i\right). \tag{2.25}$$

Now, using the fact that $n^\mu n_\mu = -1$ and $\beta^\mu n_\mu = 0$, we have

$$\begin{aligned} n_\mu n^\mu &= n_\mu \frac{(\partial_t)^\mu - \beta^\mu}{\alpha} \\ &= \frac{n_t}{\alpha} + \frac{n_\mu \beta^\mu}{\alpha} \\ &= \frac{n_t}{\alpha} = -1, \end{aligned} \tag{2.26}$$

which yields

$$n_\mu = (-\alpha, 0, 0, 0). \tag{2.27}$$

2.1.5 3+1 Decomposition of the Metric g

Now, with all the previous elements, we can perform a 3+1 decomposition of the metric g. First, the induced metric γ satisfies

$$g_{ij} = \gamma_{ij}, \tag{2.28}$$

where $i, j = 1, 2, 3$ are the spatial indices. By definition, $g_{\alpha\beta} = g(\partial_\alpha, \partial_\beta) = \langle \partial_\alpha, \partial_\beta \rangle$. We obtain

$$g_{00} = \alpha^2 n^\mu n_\mu + \beta^\mu \beta_\mu = -\alpha^2 + \beta^2, \tag{2.29}$$

$$g_{0i} = g_{i0} = (m_j + \beta_j) dx^j \partial_i = \beta_j \delta^j{}_i = \beta_i. \tag{2.30}$$

Thus

$$g_{\mu\nu} = \begin{pmatrix} -\alpha^2 + \beta_k \beta^k & \beta_j \\ \beta_i & \gamma_{ij} \end{pmatrix}, \tag{2.31}$$

which can be rewritten as

$$g_{\mu\nu} dx^\mu dx^\mu = -\alpha^2 dt^2 + \gamma_{ij}(dx^i + \beta^i dt)(dx^j + \beta^j dt). \tag{2.32}$$

The elements of $g^{\mu\nu}$ are the elements of the inverse matrix of $g_{\mu\nu}$. One can verify that

$$g^{\mu\nu} = \begin{pmatrix} -\dfrac{1}{\alpha^2} & \dfrac{\beta^j}{\alpha^2} \\ \dfrac{\beta^i}{\alpha^2} & \gamma^{ij} - \dfrac{\beta^i \beta^j}{\alpha^2} \end{pmatrix}. \tag{2.33}$$

Finally, we have the needed ingredients for the 3+1 geodesic equations.

2.2 Geodesic Equations

Let us consider a particle of 4-momentum p_μ. This particle satisfies

$$p_\mu p^\mu = -m_p^2, \tag{2.34}$$

where m is the particle mass. In the case of photons, we have $m_p = 0$, i.e.

$$p_\mu p^\mu = 0. \tag{2.35}$$

If the particle is subject only to the gravitational field, its worldline is either a null (photon) or a timelike (massive particle) geodesic of $(\mathcal{M}, g)$. Then the 4-momentum obeys:

$$p^\mu \nabla_\mu p^\nu = 0. \tag{2.36}$$

This is the geodesic equation in covariant 4-dimensional form [1].

2.2.1 3+1 decomposition of geodesic equations

To write the equation (2.36) in 3+1 form we can do an orthogonal decomposition of p^μ

$$p^\mu = E(n^\mu + V^\mu), \tag{2.37}$$

with $n_\mu V^\mu = 0$.

In this decomposition the scalar E is the energy of the particle as measured by the Eulerian observer $\mathcal{O}_E$; n^μ is the 4-velocity of the $\mathcal{O}_E$, and then $E = -p_\mu n^\mu$. The vector V^μ is by construction tangent to Σ_t and coincides with the 3-velocity of the particle as measured by $\mathcal{O}_E$ [1]. In this section we are going to consider a congruence of geodesics. This means that p^μ, E and V^μ are fields defined on the spacetime.

We can plug the momentum 3+1 decomposition of the momentum into the equation (2.36):

$$\begin{aligned} p^\nu \nabla_\nu p^\mu &= E(n^\nu + V^\nu)\nabla_\nu(E(n^\mu + V^\mu)) \\ &= E(n^\nu + V^\nu)\left(\nabla_\nu E(n^\mu + V^\mu) + E\nabla_\nu n^\mu + E\nabla_\nu V^\mu\right) \\ &= E(n^\nu + V^\nu)\nabla_\nu E(n^\mu + V^\mu) + E^2(n^\nu + V^\nu)\left(\nabla_\nu n^\mu + \nabla_\nu V^\mu\right) = 0. \end{aligned} \tag{2.38}$$

We could use the equations (2.10) and (2.14) to expand $\nabla_\mu n^\alpha$ and then

$$\begin{aligned} (n^\nu + V^\nu)\left(\nabla_\nu n^\mu + \nabla_\nu V^\mu\right) &= -n^\nu K^\mu{}_\nu - (D^\mu \ln\alpha) n^\nu n_\nu + n^\nu \nabla_\nu V^\mu \\ &\quad - V^\nu K^\mu{}_\nu - (D^\mu \ln\alpha) V^\nu n_\nu + V^\nu \nabla_\nu V^\mu \\ &= n^\nu \gamma^{\sigma\mu}\gamma^\lambda{}_\nu \nabla_\sigma n_\lambda + (D^\mu \ln\alpha) + n^\nu \nabla_\nu V^\mu \\ &\quad - V^\nu K^\mu{}_\nu + V^\nu \nabla_\nu V^\mu \\ &= \gamma^{\sigma\mu} n^\nu \nabla_\sigma n_\nu + D^\mu \ln\alpha + n^\nu \nabla_\nu V^\mu \\ &\quad - V^\nu K^\alpha{}_\nu + V^\nu \nabla_\nu V^\mu \\ &= D^\mu \ln\mu + n^\nu \nabla_\nu V^\mu - V^\nu K^\alpha{}_\nu + V^\nu \nabla_\nu V^\mu. \end{aligned} \tag{2.39}$$

Then the equation becomes:

$$(n^\mu + V^\mu)\nabla_\mu E(n^\nu + V^\nu) + E\left(D^\nu \ln\alpha + n^\mu \nabla_\mu V^\nu - V^\mu K^\nu{}_\mu + V^\mu \nabla_\mu V^\nu\right) = 0. \tag{2.40}$$

Now, in the 3+1 formalism, the natural evolution operator is the Lie derivative $\mathcal{L}_m$ along the normal evolution vector (2.20); therefore, we can use that $n^\mu = \alpha^{-1} m^\mu$ and with the definition of the Lie derivative over a tensor T of order $\binom{p}{q}$ [12], i.e.

$$\begin{aligned}\mathcal{L}_u T^{\alpha_1\ldots\alpha_p}{}_{\beta_1\ldots\beta_q} &= u^\mu \nabla_\mu T^{\alpha_1\ldots\alpha_p}{}_{\beta_1\ldots\beta_q} - \sum_{i=1}^{p} T^{\alpha_1\ldots\sigma\ldots\alpha_p}{}_{\beta_1\ldots\beta_q} \nabla_\sigma u^{\alpha_i} \\ &\quad + \sum_{i=1}^{q} T^{\alpha_1\ldots\alpha_p}{}_{\beta_1\ldots\sigma\ldots\beta_q} \nabla_{\beta_i} u^\sigma,\end{aligned} \tag{2.41}$$

we can write:

$$\begin{aligned}n^\nu \nabla_\nu V^\mu &= \alpha^{-1} m^\nu \nabla_\nu V^\mu \\ &= \alpha^{-1} \left(\mathcal{L}_m V^\mu + V^\nu \nabla_\nu (m^\mu)\right) \\ &= \alpha^{-1} \left(\mathcal{L}_m V^\mu + V^\nu \nabla_\nu (\alpha n^\mu)\right) \\ &= \alpha^{-1} \mathcal{L}_m V^\mu + V^\nu \left(-K^\mu{}_\nu + D_\nu \ln \alpha n^\mu\right) \\ &= \alpha^{-1} \mathcal{L}_m V^\mu - K^\mu{}_\nu V^\nu + V^\nu D_\nu \ln \alpha n^\mu.\end{aligned} \tag{2.42}$$

Similarly, since E is a scalar field,

$$n^\mu \nabla_\mu E = \alpha^{-1} \mathcal{L}_m E, \tag{2.43}$$

also, we can check that

$$\begin{aligned}V^\mu D_\mu E &= V^\mu \gamma^\nu{}_\mu \nabla_\nu E \\ &= V^\mu \left(\delta^\nu{}_\mu + n^\nu n_\mu\right) \nabla_\nu E \\ &= V^\mu \left(\nabla_\mu E + n^\nu n_\mu \nabla_\nu E\right) \\ &= V^\mu \nabla_\mu E.\end{aligned} \tag{2.44}$$

Using the equation (2.8) and the projector definition (2.4) we deduce:

$$\begin{aligned}D_\nu V^\mu &= \gamma^\mu{}_\lambda \gamma^\sigma{}_\nu \nabla_\sigma V^\lambda \\ &= (\delta^\mu{}_\lambda + n^\mu n_\lambda)(\delta^\sigma_\nu \ + n^\sigma n_\nu) \nabla_\sigma V^\lambda \\ &= (\delta^\mu{}_\lambda \delta^\sigma{}_\nu + \delta^\mu{}_\lambda n^\sigma n_\nu + \delta^\sigma{}_\nu n^\mu n_\lambda + n^\mu n_\lambda n^\sigma n_\nu) \nabla_\sigma V^\lambda \\ &= \nabla_\nu V^\mu + n^\sigma n_\nu \nabla_\sigma V^\mu + n^\mu n_\lambda \nabla_\nu V^\lambda + n^\mu n_\lambda n^\sigma n_\nu \nabla_\sigma V^\lambda \\ &= \nabla_\nu V^\mu + n^\sigma n_\nu \nabla_\sigma V^\mu - n^\mu V^\lambda \nabla_\nu n_\lambda + n^\mu n_\lambda n^\sigma n_\nu \nabla_\sigma V^\lambda \\ &= \nabla_\nu V^\mu + n^\sigma n_\nu \nabla_\sigma V^\mu + n^\mu V^\lambda K_{\nu\lambda} - n^\mu V^\lambda D_\lambda \ln \alpha n_\nu + n^\mu n_\lambda n^\sigma n_\nu \nabla_\sigma V^\lambda,\end{aligned} \tag{2.45}$$

where we have used that $n_\nu \nabla_\mu V^\nu = -V^\nu \nabla_\mu n_\nu$. We can contract with V^ν, use that $V^\nu n_\nu = 0$, and rewrite some indexes to finally obtain

$$V^\nu \nabla_\nu V^\mu = V^\nu D_\nu V^\mu - K_{\nu\sigma} V^\nu V^\sigma n^\mu. \tag{2.46}$$

Inserting the equations (2.42), (2.44), (2.43) and (2.46) into (2.40), we get for the first term:

$$\begin{aligned}(n^\mu + V^\mu)\,\nabla_\mu E\,(n^\nu + V^\nu) &= (n^\mu \nabla_\mu E + V^\mu \nabla_\mu E)\,(n^\nu + V^\nu) \\ &= \left(\alpha^{-1}\mathcal{L}_m E + V^\mu D_\mu E\right)(n^\nu + V^\nu) \\ &= \alpha^{-1}\mathcal{L}_m E n^\nu + \alpha^{-1}\mathcal{L}_m E V^\nu + V^\mu D_\mu E n^\nu + V^\mu D_\mu E V^\nu,\end{aligned} \tag{2.47}$$

meanwhile, for the second term, ignoring $D^\alpha \ln\alpha$, we have:

$$\begin{aligned}n^\mu \nabla_\mu V^\nu - V^\mu K^\nu{}_\mu + V^\mu \nabla_\mu V^\nu = \alpha^{-1}\mathcal{L}_m V^\nu - 2K^\nu{}_\mu V^\mu + V^\mu D_\mu \ln\alpha n^\nu \\ + V^\mu D_\mu V^\nu - K_{\mu\nu}V^\mu V^\nu n^\nu.\end{aligned} \tag{2.48}$$

We can split this into two terms, the one multiplied by n^α and the ones that are not:

$$\begin{aligned}E\left(\alpha^{-1}\mathcal{L}_m V^\nu - 2K^\nu{}_\mu V^\mu + V^\mu D_\mu V^\nu + D^\nu \ln\alpha\right) + \alpha^{-1}\mathcal{L}_m E V^\nu + V^\mu D_\mu E V^\nu \\ + \left(E\left(V^\mu D_\mu \ln\alpha - K_{\mu\nu}V^\mu V^\nu\right) + \alpha^{-1}\mathcal{L}_m E + V^\mu D_\mu E\right) n^\nu = 0.\end{aligned} \tag{2.49}$$

The projection of the previous equation along n^α and onto Σ_t, and rewriting it a bit, gives us the following system:

$$\alpha^{-1}\mathcal{L}_m E + V^\mu D_\mu E + E\left(V^\mu D_\mu \ln\alpha - K_{\mu\nu}V^\mu V^\nu\right) = 0, \tag{2.50}$$

$$\alpha^{-1}\mathcal{L}_m V^\nu - 2K^\nu{}_\mu V^\mu + V^\mu D_\mu V^\nu + D^\nu \ln\alpha + E^{-1}\left(\alpha^{-1}\mathcal{L}_m E V^\nu + V^\mu D_\mu E V^\nu\right) = 0. \tag{2.51}$$

This equations just involves objects related with Σ_t, then we can write it using a 3-dimensional form with latin indexes. Using the shift vector we know that

$$\mathcal{L}_m = \frac{\partial}{\partial t} - \mathcal{L}_\beta. \tag{2.52}$$

In addition, we an substitute the $\mathcal{L}_m E$ of equation (2.51) using the equation (2.50) we will get:

$$\frac{1}{\alpha}\left(\frac{\partial}{\partial t} - \mathcal{L}_\beta\right) E + V^j D_j E + E\left(V^j D_j \ln\alpha - K_{jk}V^j V^k\right) = 0 \tag{2.53}$$

$$\frac{1}{\alpha}\left(\frac{\partial}{\partial t} - \mathcal{L}_\beta\right) V^i + V^j D_j V^i - 2K^i{}_j V^j + V^i\left(K_{jk}V^j V^k - V^j D_j \ln\alpha\right) + D^i \ln\alpha = 0. \tag{2.54}$$

This system constitutes the 3+1 geodesic equation in covariant form for a congruence of geodesics.

2.2.2 3+1 Geodesic Equation for a Single Geodesic

Let's consider a specific geodesic on the geodesic congruence. In a coordinate system $(x^\alpha) = (t, x^i)$ adapted to the 3+1 foliation $(\Sigma_t)_{t\in\mathbb{R}}$, we can rewrite

$$x^i = X^i(t), \tag{2.55}$$

where X^i is the parametrization of the geodesic by the coordinate t. By definition, the velocity with respect to $\mathcal{O}_E$ is

$$V^i = \frac{dl^i}{d\tau_E}, \tag{2.56}$$

where dl^i is the displacement vector of the particle's worldline with respect to $\mathcal{O}_E$ between t and $t + dt$, and τ_E is the increment in $\mathcal{O}_E$'s proper time between t and $t + dt$.

By construction [12], we have that

$$d\tau_E = \alpha dt \text{ and } dl^i = \beta^i dt + dX^i. \tag{2.57}$$

Then,

$$V^i = \frac{1}{\alpha}\left(\dot{X}^i + \beta^i\right), \text{ with } \dot{X}^i = \frac{dX^i}{dt} \tag{2.58}$$

and

$$\frac{dE}{dt} = \frac{\partial E}{\partial t} + \dot{X}^j \partial_j E. \tag{2.59}$$

Using (2.58) and (2.59), together with the fact that $\mathcal{L}_\beta E = \beta^i \partial_i E$ and $D_j f = \partial_j f$ for a scalar field f, we rewrite the first two terms of (2.53) as:

$$\begin{aligned}\frac{1}{\alpha}\left(\frac{\partial}{\partial t} - \mathcal{L}_\beta\right) E + V^j D_j E &= \frac{1}{\alpha}\left(\frac{\partial E}{\partial t} - \beta^j \partial_j E + \left(\dot{X}^j + \beta^j\right)\partial_j E\right) \\ &= \frac{1}{\alpha}\left(\frac{dE}{dt} - \dot{X}^j \partial_j E - \beta^j \partial_j E + \dot{X}^j \partial_j E + \beta^j \partial_j E\right) \\ &= \frac{1}{\alpha}\frac{dE}{dt}.\end{aligned} \tag{2.60}$$

Now, we can rewrite (2.53) as follows:

$$\frac{dE}{dt} = E\left(\alpha K_{jk} V^j V^k - V^j \partial_j \alpha\right), \tag{2.61}$$

where we have used that $D_j \ln \alpha = \dfrac{1}{\alpha}\partial_j \alpha$. The previous equation is the evolution equation for the particle energy relative to the Eulerian observer.

Recall that the covariant derivative can be written as:

$$\begin{aligned} D_j V^i &= \gamma^l{}_j \gamma^i{}_m \nabla_l V^m \\ &= \gamma^l{}_j \gamma^i{}_m \left(\partial_l V^m + \Gamma^m_{ln} V^n\right) \\ &= \partial_j V^i + {}^3\Gamma^i_{jl} V^l, \end{aligned} \tag{2.62}$$

where ${}^3\Gamma^i_{jl} = \gamma^n{}_j \gamma^i{}_m \Gamma^m_{nl}$. The Lie derivative of V^i along the shift vector is

$$\mathcal{L}_\beta V^i = \beta^j \partial_j V^i - V^j \partial_j \beta^i. \tag{2.63}$$

The total time derivative can be expressed as

$$\frac{dV^i}{dt} = \frac{\partial V^i}{\partial t} + \dot{X}^j \partial_j V^i. \tag{2.64}$$

Now we can rewrite the first two terms of the equation (2.54) as

$$\begin{aligned} \frac{1}{\alpha}\left(\frac{\partial}{\partial t} - \mathcal{L}_\beta\right) V^i + V^j D_j V^i &= \frac{1}{\alpha}\left(\frac{\partial V^i}{\partial t} - \beta^j \partial_j V^i + V^j \partial_j \beta^i\right) + V^j \partial_j V^i + {}^3\Gamma^i_{jk} V^j V^k \\ &= \frac{1}{\alpha}\left(\frac{dV^i}{dt} - \dot{X}^j \partial_j V^i - \beta^j \partial_j V^i + V^j \partial_j \beta^i\right) \\ &\quad + V^j \partial_j V^i + {}^3\Gamma^i_{jk} V^j V^k \\ &= \frac{1}{\alpha}\left(\frac{dV^i}{dt} - \left(\alpha V^j - \beta^j\right) \partial_j V^i - \beta^j \partial_j V^i + V^j \partial_j \beta^i\right) \\ &\quad + V^j \partial_j V^i + {}^3\Gamma^i_{jk} V^j V^k \\ &= \frac{1}{\alpha}\left(\frac{dV^i}{dt} + V^j \partial_j \beta^i\right) + {}^3\Gamma^i_{jk} V^j V^k. \end{aligned} \tag{2.65}$$

where we use $\dot{X}^j = \alpha V^j - \beta^j$ from (2.58). Using that $D^i \ln\alpha = \gamma^{ij}\partial_j \ln\alpha = \frac{1}{\alpha}\gamma^{ij}\partial_j\alpha$ and reordering the equation, we finally can rewrite the equation (2.54) in the following form:

$$\frac{dV^i}{dt} = \alpha V^j \left(V^i \left(\partial_j \ln\alpha - K_{jk} V^k\right) + 2K^i{}_j - {}^3\Gamma^i_{jk} V^k\right) - \gamma^{ij}\partial_j\alpha - V^j \partial_j \beta^i. \tag{2.66}$$

Finally, we get the following system of equations:

$$\frac{dX^i}{dt} = \alpha V^i - \beta^i \tag{2.67}$$

$$\frac{dV^i}{dt} = \alpha V^j \left(V^i \left(\frac{1}{\alpha}\partial_j\alpha - K_{jk} V^k\right) + 2K^i{}_j - {}^3\Gamma^i_{jk} V^k\right) - \gamma^{ij}\partial_j\alpha - V^j \partial_j \beta^i \tag{2.68}$$

Given the spacetime metric in 3+1 form, the terms $\alpha, \beta^i, \gamma^{ij}, {}^3\Gamma^i_{jk}$ and K_{ij} constitute a system of six ordinary differential equations that are sufficient to integrate with respect to t from initial data $(X^i(0), V^i(0))$ to get the geodesic worldline of the particle.

The equation (2.68) is a valid differential equation for the particle velocity. However, we can also derive the differential equation of $V_i = \gamma_{ij}V^j$ to see if a simpler form emerges. First, we have

$$\frac{dV_i}{dt} = \frac{d\left(\gamma_{ij}V^j\right)}{dt} = \gamma_{ij}\frac{dV^j}{dt} + V^j\frac{d\gamma_{ij}}{dt}. \tag{2.69}$$

The total derivative of the induced metric can be expressed as follows:

$$\begin{aligned}\frac{d\gamma_{ij}}{dt} &= \frac{\partial\gamma_{ij}}{\partial t} + \dot{X}^k\frac{\partial\gamma_{ij}}{\partial X^k} \\ &= \frac{\partial\gamma_{ij}}{\partial t} + \left(\alpha V^k - \beta^k\right)\frac{\partial\gamma_{ij}}{\partial X^k}.\end{aligned} \tag{2.70}$$

According to [12], the time partial derivative of the induced metric can be written in terms of the extrinsic curvature K_{ij} as

$$\begin{aligned}\frac{\partial\gamma_{ij}}{\partial t} &= -2\alpha K_{ij} + \mathcal{L}_\beta\gamma_{ij} \\ &= -2\alpha K_{ij} + D_i\beta_j + D_j\beta_i \\ &= -2\alpha K_{ij} + \gamma^{ik}\gamma^{jl}\nabla_k\beta_l + \gamma^{ik}\gamma^{jl}\nabla_l\beta_k \\ &= -2\alpha K_{ij} + \gamma^{ik}\gamma^{jl}\left(\partial_k\beta_l - \Gamma^n_{kl}\beta_n + \partial_l\beta_k - \Gamma^n_{kl}\beta_n\right) \\ &= -2\alpha K_{ij} + \partial_i\beta_j + \partial_j\beta_i - 2\,{}^3\Gamma^k_{ij}\beta_k.\end{aligned} \tag{2.71}$$

Then, we are going to have:

$$\begin{aligned}\frac{dV_i}{dt} &= \gamma_{ij}\frac{dV^j}{dt} + V^j\frac{d\gamma_{ij}}{dt} \\ &= \gamma_{ij}\frac{dV^j}{dt} - V^j 2\alpha K_{ij} + V^j\partial_i\beta_j + V^j\partial_j\beta_i - 2V^j\,{}^3\Gamma^k_{ij}\beta_k + V^j\left(\alpha V^k - \beta^k\right)\partial_k\gamma_{ij} \\ &= V^jV_i\partial_j\alpha - \alpha V^jK_{jk}V^kV_i + \alpha V^j 2K_{ij} - \alpha V^j\gamma_{il}\,{}^3\Gamma^l_{jk}V^k - \partial_i\alpha - \gamma_{il}V^j\partial_j\beta^l \\ &\quad - V^j 2\alpha K_{ij} + V^j\partial_i\beta_j + V^j\partial_j\beta_i - 2V^j\,{}^3\Gamma^k_{ij}\beta_k + V^j\left(\alpha V^k - \beta^k\right)\partial_k\gamma_{ij} \\ &= -\partial_i\alpha - \alpha K_{jk}V^jV^kV_i + \partial_j\alpha V^jV_i + \alpha V^jV^k\left(\partial_k\gamma_{ij} - \gamma_{il}\,{}^3\Gamma^l_{jk}\right) \\ &\quad + V^j\left(\partial_i\beta_j + \partial_j\beta_i - 2\,{}^3\Gamma^k_{ij}\beta_k - \gamma_{il}\partial_j\beta^l - \beta^k\partial_k\gamma_{ij}\right).\end{aligned} \tag{2.72}$$

For the first term inside of the parentheses we would have

$$\begin{aligned}
\alpha V^j V^k \left(\partial_k \gamma_{ij} - \gamma_{il}\, {}^3\Gamma^l_{jk}\right) &= \alpha V^j V^k \left(\partial_k \gamma_{ij} - \frac{\gamma_{il}\gamma^{ln}}{2}\left(\partial_j \gamma_{nk} + \partial_k \gamma_{nj} - \partial_n \gamma_{ij}\right)\right) \\
&= \alpha V^j V^k \left(\partial_k \gamma_{ij} - \frac{\delta^n_i}{2}\left(\partial_j \gamma_{nk} + \partial_k \gamma_{nj} - \partial_n \gamma_{jk}\right)\right) \\
&= \alpha V^j V^k \left(\partial_k \gamma_{ij} - \frac{1}{2}\left(\partial_j \gamma_{ik} + \partial_k \gamma_{ij} - \partial_i \gamma_{jk}\right)\right) \\
&= \alpha V^j V^k \partial_k \gamma_{ij} - \frac{\alpha V^j V^k}{2}\partial_j \gamma_{ik} - \frac{\alpha V^j V^k}{2}\partial_k \gamma_{ij} + \frac{\alpha V^j V^k}{2}\partial_i \gamma_{jk} \\
&= \alpha V^j V^k \partial_k \gamma_{ij} - \frac{\alpha V^k V^j}{2}\partial_k \gamma_{ij} - \frac{\alpha V^j V^k}{2}\partial_k \gamma_{ij} + \frac{\alpha V^j V^k}{2}\partial_i \gamma_{jk} \\
&= \frac{\alpha V^j V^k}{2}\partial_i \gamma_{jk} \\
&= \frac{\alpha V_l V_m}{2}\gamma^{lj}\gamma^{mk}\partial_i \gamma_{jk} \\
&= \frac{\alpha V_l V_m}{2}\left(\partial_i\left(\gamma^{lj}\gamma^{mk}\gamma_{jk}\right) - \gamma_{jk}\gamma^{km}\partial_i \gamma^{lj} - \gamma_{jk}\gamma^{lj}\partial_i \gamma^{mk}\right) \\
&= \frac{\alpha V_l V_m}{2}\partial_i \gamma^{lm} - \frac{\alpha V_l V_m}{2}\delta^m_j \partial_i \gamma^{lj} - \frac{\alpha V_l V_m}{2}\delta^l_k \partial_i \gamma^{mk} \\
&= \frac{\alpha V_l V_m}{2}\partial_i \gamma^{lm} - \frac{\alpha V_l V_m}{2}\partial_i \gamma^{lm} - \frac{\alpha V_l V_m}{2}\partial_i \gamma^{ml} \\
&= -\frac{\alpha V_j V_k}{2}\partial_i \gamma^{jk}.
\end{aligned} \tag{2.73}$$

Where we used the chain rule for partial derivatives. Now, for the second term inside the parentheses we would have

$$\begin{aligned}
\partial_i \beta_j + \partial_j \beta_i - 2\,{}^3\Gamma^k_{ij}\beta_k - \gamma_{il}\partial_j \beta^l - \beta^k \partial_k \gamma_{ij} &= \partial_i \beta_j + \partial_j \beta_i - 2\,{}^3\Gamma^k_{ij}\beta_k - \partial_j(\beta^l \gamma_{il}) \\
&\quad + \beta^l \partial_j \gamma_{il} - \beta^k \partial_k \gamma_{ij} \\
&= \partial_i \beta_j + \partial_j \beta_i - 2\,{}^3\Gamma^k_{ij}\beta_k - \partial_j \beta_i \\
&\quad + \beta^l \partial_j \gamma_{il} - \beta^k \partial_k \gamma_{ij} \\
&= \partial_i \beta_j - 2\,{}^3\Gamma^k_{ij}\beta_k + \beta^l \partial_j \gamma_{il} - \beta^k \partial_k \gamma_{ij} \\
&= \partial_i(\beta^k \gamma_{kj}) - \gamma^{kn}\beta_k\left(\partial_j \gamma_{ni} + \partial_i \gamma_{nj} - \partial_n \gamma_{ij}\right) \\
&\quad + \beta^l \partial_j \gamma_{il} - \beta^k \partial_k \gamma_{ij} \\
&= \gamma_{kj}\partial_i \beta^k + \beta^k \partial_i \gamma_{kj} - \beta^n \partial_j \gamma_{ni} - \beta^n \partial_i \gamma_{nj} \\
&\quad + \beta^n \partial_n \gamma_{ij} + \beta^l \partial_j \gamma_{il} - \beta^k \partial_k \gamma_{ij} \\
&= \gamma_{kj}\partial_i \beta^k,
\end{aligned} \tag{2.74}$$

after renaming dummy indices and canceling terms. Finally, the differential equation for V_i is:

$$\begin{aligned}\frac{dV_i}{dt} &= -\partial_i\alpha - \alpha K_{jk}V^jV^kV_i + \partial_j\alpha V^jV_i - \frac{\alpha V_jV_k}{2}\partial_i\gamma^{jk} + V^j\gamma_{kj}\partial_i\beta^k \\ &= -\partial_i\alpha + \left(V^j\partial_j\alpha - \alpha K_{jk}V^jV^k\right)V_i - \frac{1}{2}\alpha V_jV_k\partial_i\gamma^{jk} + V_j\partial_i\beta^j.\end{aligned} \tag{2.75}$$

where we recover equation (4) of [14], with $V_i = \Pi_i$, the same variable used in [1]. This is a more convenient differential equation because it has fewer spatial and time derivatives of the induced metric. The full system in terms of V_i is

$$\begin{aligned}\frac{dX^i}{dt} &= \alpha\gamma^{ij}V_j - \beta^i \\ \frac{dV_i}{dt} &= -\partial_i\alpha + \left(\gamma^{kj}V_k\partial_j\alpha - \alpha K_{jk}\gamma^{jl}\gamma^{km}V_mV_l\right)V_i + \frac{1}{2}\alpha\gamma^{jl}\gamma^{km}V_lV_m\partial_i\gamma_{jk} + V_j\partial_i\beta^j.\end{aligned} \tag{2.76}$$

Now, if we want to include the energy E, is more stable to compute $\ln E$, especially close to the event horizon. The equation for the energy is then:

$$\frac{d\ln E}{dt} = \left(\alpha K_{jk}\gamma^{lj}\gamma^{mk}V_lV_m - V_l\gamma^{lj}\partial_j\alpha\right). \tag{2.77}$$

2.2.3 Invariant Mass Conservation

Using the 3+1 decomposition of the 4-momentum shown in equation (2.37), we can rewrite the right-hand side of equations (2.34) and (2.35) as:

$$\begin{aligned}p_\mu p^\mu &= E^2(n_\mu + V_\mu)(n^\mu + V^\mu) \\ &= E^2(n_\mu n^\mu + V_\mu n^\mu + n_\mu V^\mu + V_\mu V^\mu) \\ &= E^2(n_\mu n^\mu + V_\mu V^\mu) \\ &= E^2(-1 + V^2).\end{aligned} \tag{2.78}$$

This gives us a normalization rule for the initial conditions and a quantity to track throughout the simulation. This results in the following constrains:

$$\begin{aligned}V_iV^i &= 1 && \text{for null geodesics,} \\ V_iV^i &= 1 - \frac{m_p^2}{E^2} && \text{for timelike geodesics.}\end{aligned} \tag{2.79}$$

The previous constraint can be used to verify that the solver evolves geodesics as expected throughout the simulation and to perform convergence tests for code validation. One

important remark about the system of equations (2.76) and (2.77) is that we have not assumed whether the particles are null or timelike. Their behavior is determined solely by the velocity normalization at the beginning of the simulation.

Chapter 3

Numerical Methods

In order to solve the system of equations (2.76) and (2.77) we need to use numerical methods due to the lack of a general analytical solution. In this project we have used the finite element method to store the fields' values, the barycentric Lagrange interpolation to interpolate the values of the fields at the particles' position and Runge-Kutta method for solving the differential equation in time.

3.1 Finite Differences

Given a spatial domain $\Omega \subset \mathbb{R}^3$ there exists a triangulation T where we could use the finite element method [15]. In this case we are interested in triangulations made of polyhedra in three dimensions. We can divide each axis of the domain Ω into $I \times J \times K$ cell units of size $\Delta x, \Delta y, \Delta z$ such that $I\Delta x = x_{\max} - x_{\min}$, $J\Delta y = y_{\max} - y_{\min}$ and $K\Delta z = z_{\max} - z_{\min}$ where $(x_{\min}, y_{\min}, z_{\min})$ and $(x_{\max}, y_{\max}, z_{\max})$ are the lowest and the highest coordinate point on the domain Ω, respectively. The set of points $\{\vec{x}_{i,j,k}\}$ where $\vec{x}_{i,j,k} = (x_{\min} + i\Delta x)\hat{x} + (y_{\min} + j\Delta y)\hat{y} + (z_{\min} + k\Delta z)\hat{z}$ and $i = 0, \ldots, I$, $j = 0, \ldots, J$ and $k = 0, \ldots, K$ is named gridpoints [16].

At the same time, we could have a function $f : \Omega \to \mathbb{R}$ that we could approximate using our gridpoints; that is, $\tilde{f}_{i,j,k} : T \to \mathbb{R}$. We can see that $\tilde{f}$ is compatible with what we expect to work with when using a computer. For instance, Figure **3.1** shows the discretization of the function $f(x, y) = \sqrt{x^2 + y^2}$ over a domain with the points $(-3, -2)$ and $(3, 2)$ as edges. The points that have no values inside each cell unit can be interpolated to get better precision on the value of $f(x, y)$.

In the case of this project, the K, γ, β and α fields are defined over a triangulation made of rectangular polyhedrons over an arbitrary domain contained in $\mathbb{R}^3$. This domain is the

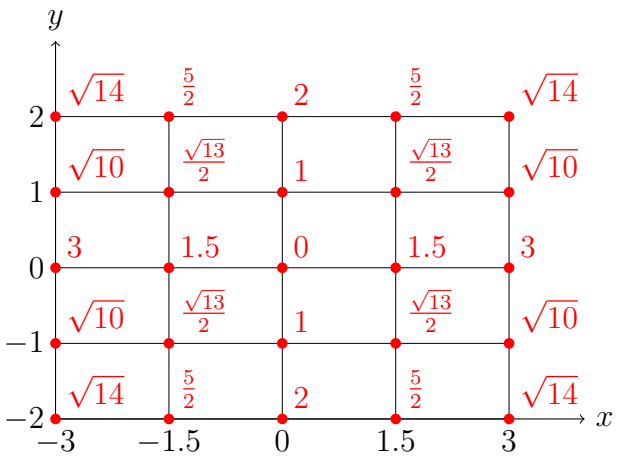

Figure 3.1: Example of discretization of the function $f(x,y) = \sqrt{x^2+y^2}$ over a 2-dimensional domain with irregular boxes.

spacetime where the particles are going to be evolved and its discretization is what we will call a mesh.

3.1.1 Adaptive Mesh Refinement (AMR)

The mesh can be refined by a complete re-meshing of the domain Ω by overlaying patches of finer mesh sizes $\Delta x_l, \Delta y_l, \Delta z_l$ [17]. In cases of non-smooth functions, adaptive mesh refinement can be used to concentrate the computational effort to regions where it is needed most.

AMReX, the framework we will use, employs block-structured AMR. In [7], the authors describe block-structured AMR as having the following features:

- The mesh covering the computational domain is decomposed spatially into structured patches, each of which covers a logically rectangular region of the domain.
- Patches with the same mesh spacing are disjoint; the union of such patches is referred to as a level. Only the coarsest level has to cover the computational domain; the other levels can cover it as well.
- The complete mesh hierarchy is the union of all the levels. At the same time, proper nesting is enforced, that is, the union of grids at level $l > 0$ is strictly contained within the union of grids at level $l - 1$.

For instance, a mesh refinement of the grid in Figure **3.1** can be seen in Figure **3.2**, where all the levels follow the AMReX considerations previously stated. On each of the grid points, we can still define the values of some function f, improving the resolution on some zones

that could need a higher precision. For instance, near the black hole's event horizon in some spacetimes.

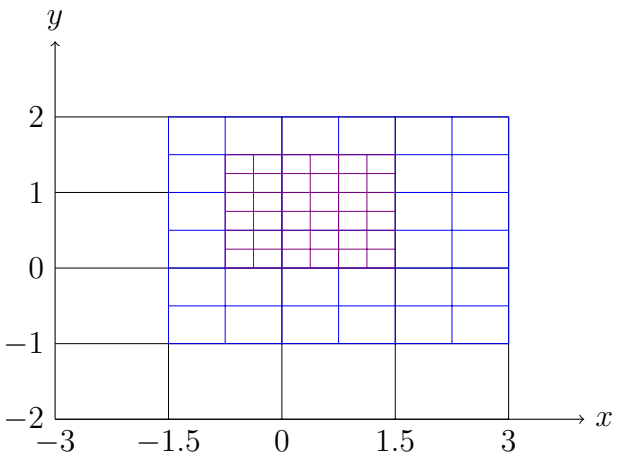

Figure 3.2: Re-meshing of the mesh shown in Figure **3.1** using three refinement levels. Each refinement level halves the mesh size of the previous one

3.2 Numerical Integration of Geodesics

In order to successfully solve the geodesic equations. it is necessary to employ both interpolation and time integration techniques that provide sufficient precision for the entire simulation. The method used for the numerical time integration is the "classical or standard" Runge-Kutta fourth order method. For the spatial interpolation we implemented a barycentric Lagrange interpolation for the function values and their derivatives.

3.2.1 Runge-Kutta fourth order method

The Runge-Kutta methods are a family of implicit and explicit iterative methods for solving the initial-value problems of differential equations. The Runge-Kutta method of order p has a global truncation error of $O(\Delta t^p)$.

Given a differential equation of the type

$$\frac{dy}{dt} = f(t, y), \tag{3.1}$$

with initial condition $y_0 = y(t_0)$, we discretize the interval $[t_0, t_f]$ using $\Delta t = \dfrac{t_f - t_0}{N}$, where N is a positive integer. The Runge-Kutta methods compute an approximation to the function at $y(t_{n+1}) = y_{n+1}$ by constructing a set of intermediate points Y_i and using the known value

$y(t_n) = y_n$ for $n = 0, \ldots, N$. A p-th order Runge-Kutta method can be written in the following way:

$$y_{n+1} = y_n + \Delta t \sum_{i}^{i=p} b_i Y_i, \tag{3.2}$$

with

$$Y_i = f\left(t + c_i \Delta t, y_n + \Delta t \sum_{j<i}^{j=p} A_{ij} Y_j\right), \tag{3.3}$$

where c_i, b_i and A_{ij} are coefficients that have to be determined. It is convenient to represent a Runge-Kutta by a partitioned tableau of the form [18]:

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array}$$

In the case of the explicit methods, which we are using to integrate the geodesics, the matrix A is a strictly lower triangular matrix. Some analysis of the coefficients can be performed; for instance in [18] the authors obtain the conditions for Runge-Kutta methods of different orders.

In particular, we focus on the fourth-order Runge-Kutta method. The solution for the coefficient values defines a family of solutions. The method we are using is the so-called "classical or standard" fourth-order Runge-Kutta method, defined by the tableau

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

The previous definitions can be extended to vector functions. Given the vector differential equation

$$\frac{d\vec{U}}{dt} = \vec{f}(t, \vec{U}), \tag{3.4}$$

with the initial conditions $\vec{U}_0 = \vec{U}(t_0)$. The fourth order standard Runge-Kutta computes the solution as

$$\vec{U}_{n+1} = \vec{U}_n + \frac{\Delta t}{6}\left(\vec{Y}_1 + 2\vec{Y}_2 + 2\vec{Y}_3 + \vec{Y}_4\right), \tag{3.5}$$

with

$$\begin{aligned}
\vec{Y_1} &= \vec{f}\left(t, \vec{U}_n\right), \\
\vec{Y_2} &= \vec{f}\left(t + \frac{\Delta t}{2}, \vec{U}_n + \frac{\Delta t}{2}\vec{Y_1}\right), \\
\vec{Y_3} &= \vec{f}\left(t + \frac{\Delta t}{2}, \vec{U}_n + \frac{\Delta t}{2}\vec{Y_2}\right), \\
\vec{Y_4} &= \vec{f}\left(t + \Delta t, \vec{U}_n + \Delta t\vec{Y_3}\right).
\end{aligned} \tag{3.6}$$

3.2.2 Numeric Interpolation on the grid

Given some discretization performed over the domain, we can define and get the value of the needed 3+1 fields on each of the gridpoints. However, we may need to know the fields' values at a point located inside the box defined by a set of gridpoints. We have used the Barycentric-Lagrange Interpolation to obtain an approximation of the field's values at the particles' positions during the simulation.

3.2.2.1 Polynomial Lagrange Interpolation

Let us consider a one-dimensional discretization and $n+1$ gridpoints x_i with $i = 0, \ldots, n$, together with a function $f(x)$ defined over it satisfying $f_i = f(x_i)$. The goal is to find the polynomial p of degree at most n that interpolates f. The solution can be written in Lagrange form [19]:

$$p(x) = \sum_{i=0}^{n} f_i l_i(x), \qquad l_i(x) = \frac{\prod\limits_{j=0, j\neq i}^{n} (x - x_j)}{\prod\limits_{j=0, j\neq i}^{n} (x_i - x_j)}. \tag{3.7}$$

Where $l_i(x)$ at $x = x_j$ satisfies

$$l_i(x_j) = \begin{cases} 1, & i = j \\ 0, & \text{otherwise.} \end{cases} \tag{3.8}$$

The main disadvantages of using the Lagrange Interpolator are the following ones [19]:

1. Each evaluation of $p(x)$ requires $O(n^2)$ operations.
2. Adding a new data pair (gridpoint and f evaluated on it) requires a new computation from scratch.
3. The computation is numerically unstable at the edges of the interval (Runge's phenomenon).

3.2.2.2 Barycentric-Lagrange Interpolation

The Lagrange interpolation can be re-written in such a way that it can be computed with a complexity of $O(n)$ by redefining

$$l_i(x) = l(x)\frac{\omega_i}{x - x_i}, \tag{3.9}$$

with

$$\omega_i = \frac{1}{\prod_{j \neq i}(x_i - x_j)} \text{ and } l(x) = (x - x_0)(x - x_1)\dots(x - x_n). \tag{3.10}$$

Then,

$$p(x) = l(x)\sum_{i=0}^{n}\frac{\omega_i}{x - x_i}f_i. \tag{3.11}$$

Now, we can pre-compute the values of ω_i, store them and use use them each time we want to compute $p(x)$ by evaluating the sum over n, reducing the complexity to $O(n)$.

We can go one more step forward. Suppose we interpolate $f(x) = 1$. Then

$$1 = l(x)\sum_{j=0}^{n}\frac{\omega_j}{x - x_j}. \tag{3.12}$$

Dividing 3.11 by the constant function 1 we just interpolated we finally obtain the Barycentric formula of the Lagrange interpolation [19]:

$$p(x) = \frac{\sum_{i=0}^{n}\frac{\omega_i}{x - x_i}f_i}{\sum_{j=0}^{n}\frac{\omega_j}{x - x_j}}. \tag{3.13}$$

We can see that we have a divergency when $x = x_i$; in this case, we can directly return the value of f_i because it is known.

Derivative extension: In order to interpolate the derivative of $f(x)$, we can use the polynomial p and differentiate it, where we now obtain

$$
\begin{aligned}
p'(x) &= \sum_{i=0}^{n} f_i \frac{d}{dx}\left(\frac{\dfrac{\omega_i}{x-x_i}}{\sum\limits_{j=0}^{n}\dfrac{\omega_j}{x-x_j}}\right) \\
&= \frac{\sum\limits_{i=0}^{n} f_i \dfrac{\omega_i}{x-x_i}\sum\limits_{k=0}^{n}\dfrac{\omega_k}{(x-x_k)^2} - \sum\limits_{i=0}^{n} f_i \dfrac{\omega_i}{(x-x_i)^2}\sum\limits_{k=0}^{n}\dfrac{\omega_k}{x-x_k}}{\left(\sum\limits_{j=0}^{n}\dfrac{\omega_j}{x-x_j}\right)^2}.
\end{aligned} \tag{3.14}
$$

If $x = x_i$, one of the gridpoints with $i \neq j$, we can rewrite the expression in order to get the value of the derivative on that point. The barycentric representation of $l_j(x)$ is

$$
l_j(x) = \frac{\dfrac{\omega_j}{x-x_j}}{\sum\limits_{k=0}^{n}\dfrac{\omega_k}{x-x_k}}. \tag{3.15}
$$

Rewriting the expression and multiplying it by $x - x_i$, we have

$$
l_j(x)\sum_{k=0}^{n}\omega_k\frac{x-x_i}{x-x_k} = l_j(x)s(x) = \omega_j\frac{x_i-x_j}{x-x_j}, \tag{3.16}
$$

then we have

$$
l_j'(x)s(x) + l_j(x)s'(x) = w_j\left(\frac{x-x_i}{x-x_j}\right)' = \omega_j\frac{x_i-x_j}{(x-x_j)^2}, \tag{3.17}
$$

when $x = x_i$ then $s(x_i) = \omega_i$, $l_j(x_i) = 0$, and $s'(x_i)$ exists. Now we can obtain an expression for $l_j'(x_i)$:

$$
l_j'(x_i) = \frac{\omega_j}{\omega_i}\frac{1}{x_i-x_j}. \tag{3.18}
$$

Plugging it back into the derivative of the polynomial p, we obtain

$$
p'(x_i) = \sum_{j=0}^{n}\frac{\omega_j}{\omega_i}\frac{f_j}{x_i-x_j}. \tag{3.19}
$$

It is important to note that Lagrange interpolation polynomials are not differentiable at the interval endpoints. As we always interpolate in the central part of the interval, this is not an issue.

Three-dimensional extension: For the purposes of the project, we need to interpolate functions that are defined along a three-dimensional grid. In order to extend the interpolation, we use five points to build a polynomial of fourth degree in each direction x, y, z as is done in [20]. Now, given a function $f(x, y, z)$ and the gridpoints defined by x_i, y_j, z_k, with $i, j, k = 0, \ldots, n$, the interpolation is written as follows:

$$\begin{aligned} f(x,y,z) &\approx \frac{\sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=0}^{n} f(x_i,y_j,z_k)\dfrac{w_i}{x-x_i}\dfrac{v_j}{y-y_j}\dfrac{u_k}{z-z_k}}{\sum_{l=0}^{n}\sum_{m=0}^{n}\sum_{p=0}^{n}\dfrac{w_l}{x-x_l}\dfrac{v_m}{y-y_m}\dfrac{u_p}{z-z_p}} \\ &= \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=0}^{n} f(x_i,y_j,z_k)W(x,x_i)V(y,y_j)U(z,z_k), \end{aligned} \tag{3.20}$$

where,

$$w_i = \frac{1}{\prod_{j=0,j\neq i}^{n}(x_i - x_j)}, \quad v_i = \frac{1}{\prod_{j=0,j\neq i}^{n}(y_i - y_j)} \quad \text{and} \quad u_i = \frac{1}{\prod_{j=0,j\neq i}^{n}(z_i - z_j)}. \tag{3.21}$$

We have also defined

$$W(x,x_i) = \frac{\dfrac{w_i}{x-x_i}}{\sum_{l=0}^{n}\dfrac{w_l}{x-x_l}}, \quad V(y,y_j) = \frac{\dfrac{v_j}{y-y_j}}{\sum_{m=0}^{n}\dfrac{v_m}{y-y_m}} \quad \text{and} \quad U(z,z_k) = \frac{\dfrac{u_k}{z-z_k}}{\sum_{p=0}^{n}\dfrac{u_p}{z-z_p}}. \tag{3.22}$$

Now, we can deduce the partial derivatives in a similar way. The three partial derivatives needed are going to be written as:

$$\begin{aligned} \partial_x f(x,y,z) &\approx \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{k=0}^{n}\left(f(x_i,y_j,z_k)\frac{d}{dx}W(x,x_i)\right)V(y,y_j)U(z,z_k), \\ \partial_y f(x,y,z) &\approx \sum_{i=0}^{n}\sum_{k=0}^{n} W(x,x_i)\frac{d}{dy}\left(\sum_{j=0}^{n} f(x_i,y_j,z_k)V(y,y_j)\right)U(z,z_k), \\ \partial_z f(x,y,z) &\approx \sum_{i=0}^{n}\sum_{j=0}^{n} W(x,x_i)V(y,y_j)\frac{d}{dz}\left(\sum_{k=0}^{n} f(x_i,y_j,z_k)U(z,z_k)\right). \end{aligned} \tag{3.23}$$

Where each of the derivatives has the same structure as the results for the barycentric Lagrange interpolation in one dimension. Then, we can use the expression 3.14 or 3.19 in

order to compute the value of the partial derivatives using the neighboring gridpoints.

An important remark is that we are using the interpolation for a box centered on the particle's location. This means that the functions are being interpolated at a location close to the center of the interval. This allows us avoid significant errors coming from the Runge's phenomenon.

3.2.3 Error sources

The two main error sources of the numerical methods used in this project are the Runge-Kutta truncation error and the barycentric Lagrange interpolation error arising from the function and its derivatives. The final error measured in the solution is a combination of these two errors.

The global truncation error of the Runge-Kutta method is the cumulative effect of error in many steps leading to an error in a final output point. Given a p-th order Runge-Kutta method with a fixed timestep Δt, the global error is proportional to Δt^p [18]. In this case, we are using a fourth-order Runge-Kutta method. This means that the global truncation error should converge with order Δt^4; that is, halving the timestep makes the error sixteen times smaller. This also depends on the smoothness of the solution and its sufficient differentiability..

In addition, we have the error arising from polynomial interpolation. Given some function $f(x)$, a set of interpolation points $\{x_i\}_n$ over some interval $I \subset \mathbb{R}$, and the polynomial interpolant $p_n(x)$, the error at any point in the domain is defined by

$$E_n(x) = p_n(x) - f(x) = \frac{f^{(n+1)}(c)}{(n+1)!}\prod_{i=0}^{n}(x - x_i) \qquad \text{(for some } c \in I\text{)}. \tag{3.24}$$

For the project we will use points with constant spacing Δx as interpolant points. Any point x in the interval can be written as $x = x_0 + k\Delta x$, where x_0 the lowest point in the interval for some $k \in [0, n]$. The error can be rewritten as

$$\begin{aligned} E_n(x) &= \frac{f^{(n+1)}(c)}{(n+1)!}\prod_{i=0}^{n}(x_0 - k\Delta x - x_0 + i\Delta x) \\ &= \frac{f^{(n+1)}(c)}{(n+1)!}\prod_{i=0}^{n}(i-k)\Delta x \\ &= \frac{\Delta x^{n+1}}{(n+1)!}f^{(n+1)}(c)\prod_{i=0}^{n}(i-k). \end{aligned} \tag{3.25}$$

This means that the barycentric Lagrange interpolator converges with order $n+1$ in the grid spacing Δx. For the case of the first derivative, the order is reduced by one; therefore, Δx^n is the dominant error contributed by interpolation. In order to be consistent with the fourth-order Runge-Kutta method, the error from interpolation must be at least the same order of the Runge-Kutta global error. For this reason, we have chosen a barycentric Lagrange interpolation using five gridpoints.

The total error in the numerical integration will be dominated by the combination of both the time integration error and the interpolation errors in each direction:

$$\Delta E \approx A\Delta t^4 + B_x\Delta x^4 + B_y\Delta y^4 + B_z\Delta z^4 + \mathcal{O}\left(\Delta t^5, \Delta x^5, \Delta y^5, \Delta z^5\right), \tag{3.26}$$

where A, B_x, B_y and B_z are constants.

Chapter 4

Software Ecosystem

The Geodesic Integrator tool is intended to be integrated with other tools that are part of the Einstein Toolkit architecture; that is why the design is strongly defined in a Cactus-like way. The project consists of the development of a thorn that can be called with the other existing thorns in the Einstein Toolkit framework. Moreover, the code uses particles and parallel management capabilities that the framework AMReX offers, such as the *ParticleContainers*.

In the following sections provide a brief description of Cactus, Einstein Toolkit and AMReX.

4.1 Cactus

Cactus is an open-source modular and portable environment designed for scientists and engineers for high-performance computing simulations. Cactus has several application domains such as computational fluid dynamics, quantum gravity and numerical relativity [21].

Cactus allows the user to develop code in one of the four supported programming languages: F77, F90, C, C++. The developers write their own components called "thorns" which are connected together by the Cactus' central core "flesh". Thorns can implement custom-developed scientific or engineering applications, in this case a numerical relativity tool. Cactus also includes a set of existing thorns such as HDF5, PETSc and PAPI to manage the needed utilities [21].

4.1.1 Cactus Thorn

The geodesic integrator tool is a set of thorns working together in order to compute the particles' trajectories in a general GRMHD background. A thorn is the basic working

module within Cactus. All the project code that defines the simulation behavior goes into thorns, and each thorn should be independent of each other [22].

A thorn consists of a subdirectory of an arrangement containing four configuration files:

- **interface.ccl:** The Cactus interface, which defines the thorn's relationship with other thorns; the include files used from other thorns and those provided by this thorn; the aliased functions provided or used by the thorn; and the thorn's global variables.
- **param.ccl:** This file defines the parameters that are used to control the thorn, along with their visibility to other implementations. It also includes the parameters needed from other thorns.
- **schedule.ccl:** The scheduling information for routines implemented by the thorn that are called by the flesh.
- **configuration.ccl:** This file defines the capabilities which a thorn provides or requires, or may use if available.

Our thorns will contain:

- A subdirectory called **src/**, which contains the source files and compilation instructions for the thorn.
- A subdirectory called **par/** for example parameter files.

Other directories, such as the **test/** directory, will be created in future development stages.

4.2 Einstein Toolkit

The Einstein Toolkit is a community-driven software platform of core computational tools to advance and support research in relativistic astrophysics and gravitational physics [23]. This is based on Cactus and contains over 270 Cactus thorns for computational relativity. The Einstein Toolkit is supported by a distributed model, combining core support of software, tools, and documentation with partnerships with other developers who contribute open software and coordinate on development [23].

Einstein Toolkit contains two of the main pillars of the project: CarpetX and AMReX thorns.

4.2.1 CarpetX

CarpetX is a driver thorn of the Cactus framework built on top of AMReX, a software framework designed for block-structured adaptive mesh refinement (AMR). CarpetX is developed as part of the Einstein Toolkit and provides the core infrastructure required for AMR-based simulations [24].

Within the Cactus architecture, driver thorns are responsible for managing distributed data structures, mesh refinement, memory allocation, and interfaces to parallel computing hardware and software. As such, they supply the fundamental low-level functionality upon which scientific simulation components rely [24].

For this project we will use CarpetX to obtain the ADM variables defined by the other thorns, or even define them if needed.

4.3 AMReX

AMReX is a separate software product that was incorporated into the Einstein toolkit. AMReX serves as a software framework that offers a cohesive infrastructure equipped with the necessary features for AMR applications to efficiently and effectively leverage machine resources [7]. AMReX includes the support for particle simulations and manages the parallelization via flat MPI, OpenMP, hybrid MPI/OpenMP, hybrid MPI/(CUDA or HIP or SYCL) and Parallel I/O.

4.3.1 Particles and ParticleContainer

The main pillars of the project are the data structures and iterators provided by AMReX for managing and performing the data-parallel particle simulations and their interactions with the meshes where the particles reside. AMReX provides the *Particle* and *ParticleContainer* classes that contain and manage all the particle data.

The *Particle*$<n, m>$ templated class is a data type that stores n double values and m integer values along with three double values for the particle's position and two integer type values for the particle's id and the CPU where it is located.

The *ParticleContainer*$<n, m, o, p>$ templated class is a larger data type that stores the particle data for a set of multiple particles. The first two templated parameters, n and m, have the same meaning as the ones in the *Particle* class, but here the data is stored in Array-of-Struct style, while the other two, o and p, are stored in the Struct-of-Arrays

form. The *ParticleContainer* class also has the *AmrParticleContainer* version of it that manages all the AMR distribution of the particles. We are using the *AmrParticleContainer* to construct the thorn. All the AMReX documentation related to it and more features is available at [25].

4.4 Parallelization, GPU and AMR Integration

All parallelization related to the numerical integration on both CPU and GPU is handled internally by AMReX, and the same applies to the integration of adaptive mesh refinement (AMR). Nevertheless, several aspects must be carefully considered, including the strategy employed by AMReX to distribute grid and particle data, the correct procedures for accessing these data structures, and the methods required to manage computations efficiently on the GPU.

4.4.1 AMReX Grid Creation and load balancing

In order to run a simulation, a set of variables must be defined for the simulation space and the grid creation over it. The variables used by AMReX are the following [25]:

- *n_cell:* The number of cells in each direction.
- *max_grid_size:* The default load balancing algorithm divides the domain in each direction so that each grid is no larger than *max_grid_size*.
- *blocking_factor:* The value of *blocking_factor* imposes a constraint on grid creation, requiring that each grid size be divisible by *blocking_factor*.
- *refine_grid_layout:* If it is true and the number of grids created is less than the number of processors, then grids will be further subdivided until the number of grids is no longer less than the number of processors.

These grid parameters can be seen in some of the *param.ccl* files included in the developed thorns. The grid creation process at level 0 proceeds as follows [25]:

1. The domain is created as a single grid of size *n_cell*.
2. If *n_cell* is greater than *max_grid_size*, then the grids are subdivided until each grid is not larger than *max_grid_size*, while ensuring the *blocking_factor* criterion is satisfied.
3. If *refine_grid_layout* is true and there are more processors than grids at this level, then the grids at this level are divided until the opposite is true (unless doing it would violate the *blocking_factor* criterion).

Once the grids are created, the load balancing process starts. The inputs to the load balancing process are a set of grids and a weight assigned to each grid. Single-level load balancing algorithms are applied in sequence to each AMR level independently, and the resulting grid distributions are then assigned to MPI ranks while accounting for the workload already present on each rank (the heaviest grid sets are placed on the least loaded ranks). Note that a process's load is defined by the amount of memory it has already allocated, rather than the memory it is expected to allocate [25].

AMReX has a set of distribution options available, like Knapsack algorithm, Round-robin with the SFC algorithm being the default one. The SFC algorithm lists grid cells using a space-filling Z-Morton curve, then divides this sequence among ranks in a manner that balances the workload. AMReX repeats the algorithm level by level to redistribute the new grids into the processes and balance the load.

4.4.2 AMReX Iterator and ParallelFor

In order to manage the access to each grid, AMReX has defined an Iterator that goes through the data along each of the grids. Moreover, in order to use GPU streams while spawning GPU kernels, AMReX has defined the *ParallelFor* function. These two elements enable access to and modification of the data in an effective and efficient manner.

4.4.2.1 MFIter

AMReX provides an iterator, *MFIter*, that allows the user to loop over the data stored on the grids owned by each process. This iterator can be used both with or without tiling. The *MFIter* is used for accessing multiple data stored in each process at the same time.

For instance, the following code shows the way to interact with the mesh data and the particle data whithin each grid using *MFIter*:

```
for (MFIter mfi = MakeMFIter(); mfi.isValid(); ++mfi){
    // Get the particles data present on the current grid.
    auto &particle_tile = DefineAndReturnParticleTile(mfi);
    const long number_of_particles = particle_tile.GetArrayOfStructs().size();

    // Get the metric data present on the current grid.
    const auto metric_array = metric.array(mfi);

    for(long i = 0; i < number_of_particles; i++){
    // Do changes to the particles
    }
}
```

Listing 1: AMReX *MFIter* loop example to access the particle and the field data on each of the grids.

AMReX provides *ParIter* to iterate over the particle data more easily. It has the same purpose as the *MFIter*, but applied to the particle data.

4.4.2.2 Parallel For

In order to manage the parallelization, AMReX also provides a function template for writing a for loop in a concise and performance-portable way. *ParallelFor* is a function that takes as parameters the iteration index space a lambda function that captures the values and iterates over them.

```
// 1D for loop
ParallelFor(N, [=] AMREX_GPU_DEVICE (int i) {
                        // lambda function
                    });

// 3D for loop
ParallelFor(box,
            [=] AMREX_GPU_DEVICE (int i, int j, int k) {
                    // lambda function
                });
```

Listing 2: AMReX 1D and 3D *ParallelFor* loop usage.

For instance, Listing 2 shows the use of the *ParallelFor* function iterating along one and three dimensions. This function works with and without GPU support. When AMReX

is compiled with GPU support enabled, AMREX_GPU_DEVICE marks the lambda as a device function, and *ParallelFor* launches a GPU kernel to carry out the computation. If AMReX is compiled without GPU support, AMREX_GPU_DEVICE has no effect at all.

The *ParallelFor* function for 3D has the following internal syntax:

```
void ParallelFor (Box const& box, /* LAMBDA FUNCTION */){
    const Dim3 lo = amrex::lbound(box);
    const Dim3 hi = amrex::ubound(box);

    for (int z = lo.z; z <= hi.z; ++z) {
        for (int y = lo.y; y <= hi.y; ++y) {
            AMREX_PRAGMA_SIMD
            for (int x = lo.x; x <= hi.x; ++x) {
                /* LAUNCH LAMBDA FUNCTION (x,y,z,n) */
            }
        }
    }
}
```

Listing 3: Internal view of *ParallelFor* syntax.

4.4.3 AMReX GPU support

AMReX employs a hierarchical parallelism model based on MPI+X. On CPU systems, X corresponds to OpenMP, where multiple threads simultaneously process tiles that belong to the same MPI rank. On GPUs, X refers to one of CUDA, HIP, or SYCL. Kernels are typically launched at the Box level, and each GPU thread is assigned one or more cells from a given Box.

To achieve performance portability, GPU kernels are commonly invoked via *ParallelFor* loop constructs, which rely on GPU-extended lambdas to define the operations carried out for each loop element.

AMReX implements the GPU parallelization using streams. AMReX places each iteration of *MFIter* loops on separate streams, allowing each independent iteration torun concurrently while maintaining sequential execution within each stream, thereby maximizing GPU usage.

Figure **4.1** illustrates how this is achieved. First, the CPU runs a *MFIter* loop iteration (blue), which contains three GPU kernels. The kernels are sent to one of the streams and

run in the same order. The next iterations (red and green) do the same process but using the idle streams and execute their kernels. The fourth and fifth iterations (orange and purple) must wait until a stream becomes free, and then repeat the same process as the other iterations. Finally, the GPU synchronizes all the streams and completes with the iteration.

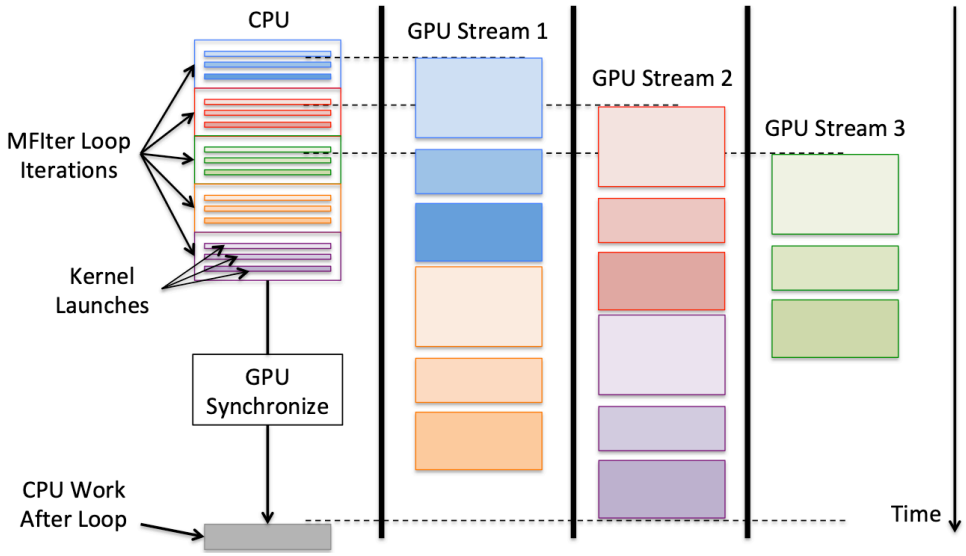

Figure 4.1: Timeline illustration of AMReX GPU streams during five MFIter loop iterations with three kernels and a GPU with 3 streams. Image taken from [25], the Overview of AMReX GPU Support section.

4.4.3.1 Particle support

All the particle data can be stored in GPU-accessible memory. The *dataPtr* associated with particle data can be passed into GPU kernels, allowing the update of the data inside of the GPU.

Finally, the parallel communication of particle data is also handled by AMReX. It includes the *Redistribute()* function, which moves the particles back to the proper grids after their positions have changed. The *Redistribute()* function has been designed to minimize host/device traffic as much as possible.

4.4.4 AMR extension

Once the multilevel grids have been created, AMReX allows the use of the iterators and mesh data to be accessed on each level by specifying the level when a getter method is used. In this case, we can evolve the particles or fields by iterating through the levels using a for loop. For instance, the Listing 1 can be rewritten as follows to allow the AMR integration and to use the *ParallelFor* function:

```
...
for(int level = 0; level < max_num_levels; level++){
    // Get the iterator on the current refinement level
    for (MFIter mfi = MakeMFIter(level); mfi.isValid(); ++mfi){
        // Get the particles data present on the current grid.
        auto &particle_tile = DefineAndReturnParticleTile(level, mfi);
        const long number_of_particles = particle_tile.GetArrayOfStructs().size();

        // Get the metric data present on the current grid.
        const auto metric_array = metric.array(mfi);

        ParallelFor(number_of_particles, [=] AMREX_GPU_DEVICE (int i){
            // Do changes to the particles on each level
        });
    }
}
...
```

Listing 4: Accessing the particle and mesh data using the *MFIter* and *ParallelFor* iterating over each of the AMR levels.

Listing 4 shows how most of the functions m managing the particles have been developed: accessing each grid using the *MFIter* iterator while iterating over the particles using a *ParallelFor* function to ensure a portable parallelization. Once each timestep in the simulation is done, the call of the *Redistribute()* function sends the particles' data to each correspondent refinement level and process.

Chapter 5

Code Design and Implementation

GInX is an open-source numerical relativity tool designed to integrate with other thorns in the Einstein Toolkit platform. It enables parallel execution on both CPU and GPU platforms and supports AMR through the facilities provided by AMReX. The code is designed using the fast-light approximation, where we assume the spacetime does not change during the particle evolution. This approximation is useful for static spacetimes or dynamic ones that evolve slowly compared to the light-crossing time.

5.1 Code Structure Overview

The code is designed to be implemented as an Einstein Toolkit thorn. Consequently, the code has been structured analogous to Cactus, adhering to the folders and file structure described in the preceding chapter. The code is fully available in the GInX GitHub repository [9], which contains the implementation of three thorns.

- The *BaseParticlesContainer* thorn, which defines the *BaseParticleContainer* abstract class. The class encapsulates all the functions that need to define in the derived classes, as well as the the AMReX output used for all particles.
- The *ParticlesSolverUtilities* thorn, which defines the interpolation functions and algebraic utilities.
- The *ParticlesContainer* thorn, which defines the functions needed to numerically integrate the geodesic equations.

The code was developed using C++17 with an object-oriented programming approach, which facilitates the efficient and comprehensive utilization of AMReX classes. This facilitates efficient and comprehensive utilization of AMReX classes. The documentation was generated using Doxygen, a documentation generator tool, and is hosted on GitHub Pages [10].

5.1.1 Classes' Attributes and Methods

Figure **5.1** shows the UML class diagram, illustrating the inheritance relationship between the abstract class *BaseParticlesContainer*, the AMReX class *AmrParticleContainer*, and the derived class *ParticlesContainer*. The *ParticlesContainer* class is responsible for particle evolution using equations (2.76) and (2.77) exclusively; however, because this system can be described in terms of other variables or components, we created the *BaseParticlesContainer* to extend the range of solvable particle-related equations for other upcoming tools.

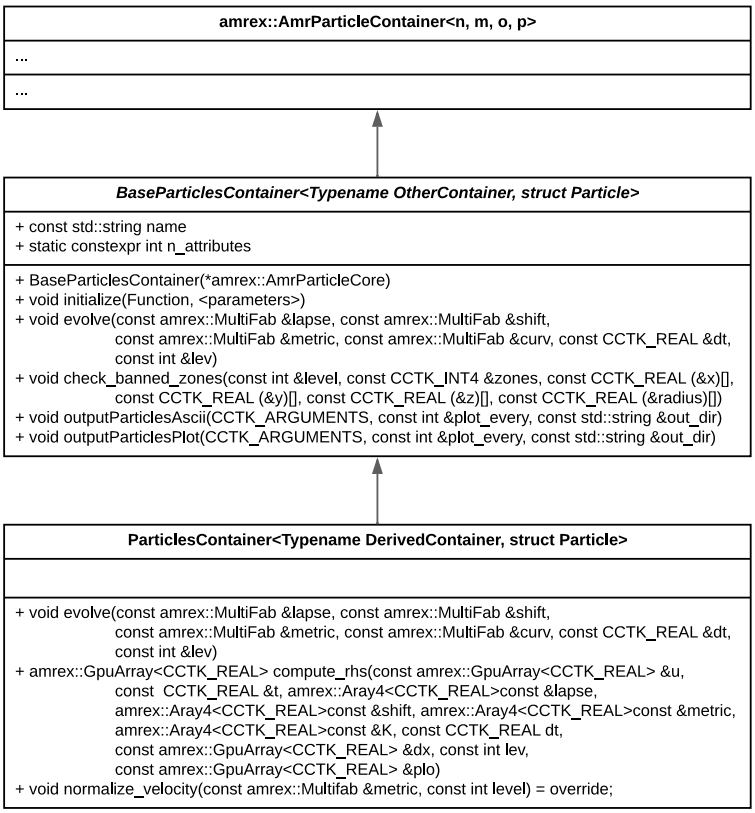

Figure 5.1: Project's UML class diagram.

The *BaseParticlesContainer* abstract class contains methods related to the evolution of a generic AMReX particle container in a numerical relativity background. These include the *evolve* method, the *initialize* method, the *checked_banned_zones* method to exclude numerically unstable regions such as Black hole singularities, and two methods for particle data output. Inside the class, only the output and the *checked_banned_zones* methods are defined; the others must be defined in the derived classes.

The *ParticlesContainer* derived class is the class that the end user will utillize. This class includes the definition of the virtual methods declared in the *BaseParticlesContainer*

abstract class, along with the necessary components to solve the system of differential equations (2.76) and (2.77). Additionally, it implements the *normalize_velocity* method, which sets the particles' velocity as shown in equation (2.79).

5.2 Algorithm

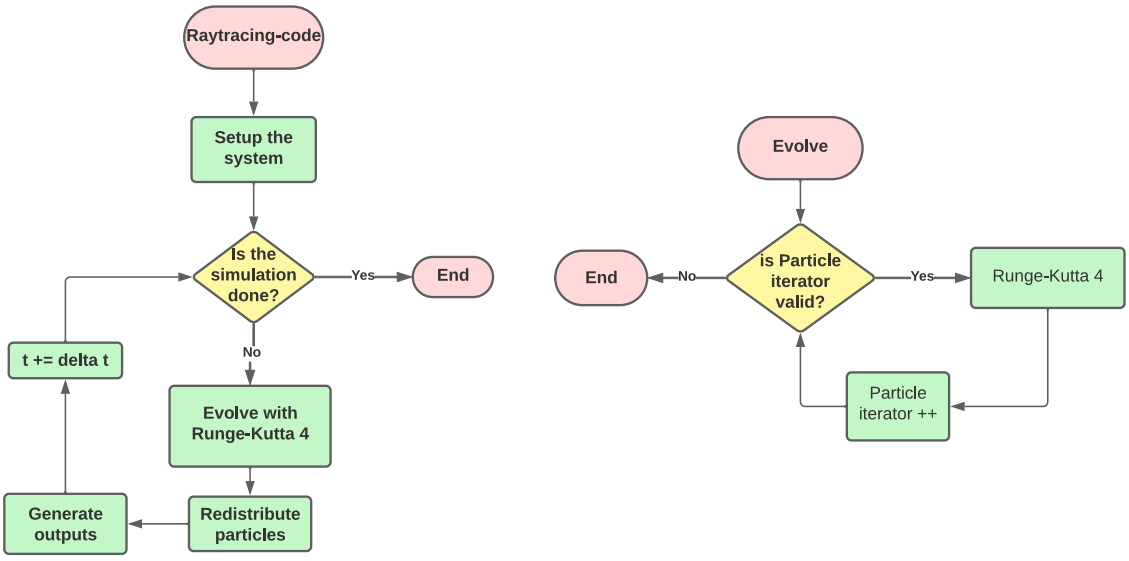

Figure 5.2: The code's flow for solving particle geodesics given the ADM 3+1 parameters (left) and the evolve function flow at each timestep (right).

The geodesic solver for particles can follow the flow of a typical ray-tracing like code. The code follows the flow shown in Figure **5.2** to solve the geodesic equations whitin the 3+1 formulation of general relativity. The code goes through the following steps:

- **Set up the system:**
  - → Set up the metric and the grid based variables, this can be done by any other thorn or to use the one defined by GInX.
  - → Initialize the particle positions and velocities, filling the ParticleContainer with them.
  - → Redistribute the particles to their correspondent AMR level and processor.
  - → Normalize the velocity depending on the particle type.
- **Evolve:** Inside the Evolve process, each particle container solves the system of differential equations. This is also split into sub-processes:

→ Compute the differential equation's right-hand side. Given a vector of variables u we approximate the ADM variables needed for the differential equation using spatial interpolation and the their values at each neighboring gridpoint, depending on the interpolation order.

→ Use the fourth-order Runge-Kutta method to solve the differential equation, computing the right-hand side at each Runge-Kutta step.

→ Check if the particles remain in a valid domain; otherwise, remove them from the container.

- **Redistribute particles:** During the simulation, particles may move outside each process's local domain. Therefore, AMReX manages the redistribution of particles to their appropriate local domains. This is crucial because each process only contains information relevant to its local domain.

- **Generate outputs:** Some simulations may require to outputting particle's data; AMReX handles this by writing to files.

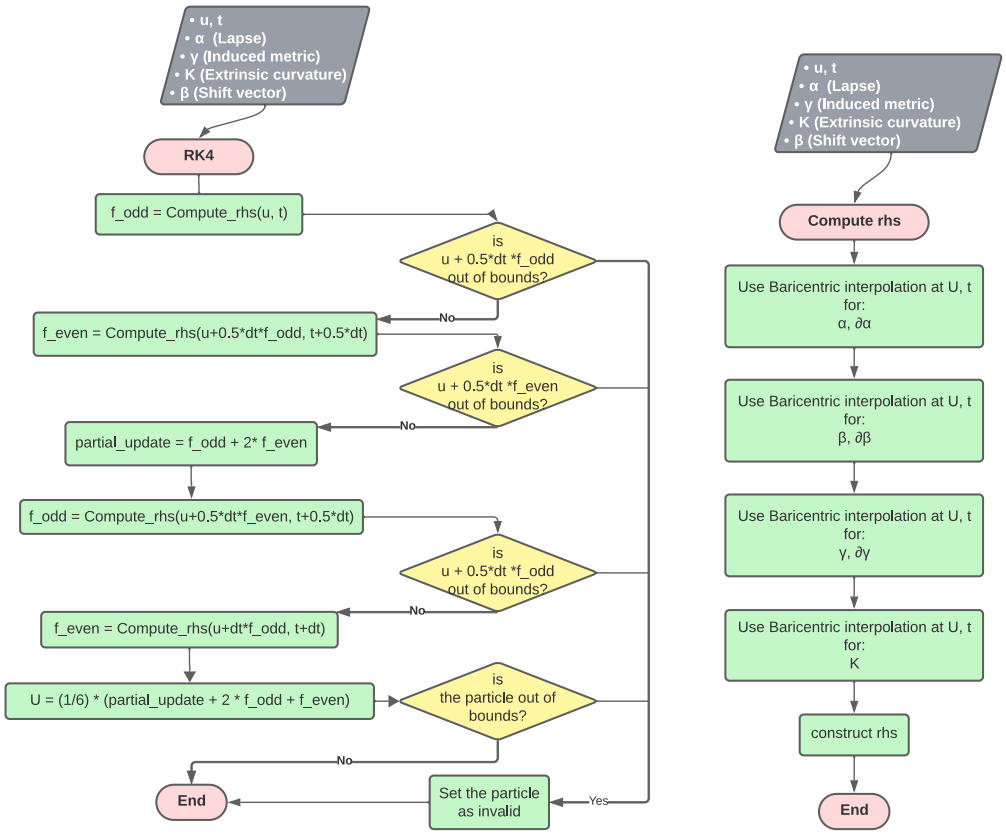

Figure 5.3: Code's Runge-Kutta 4 flow (left) and the right-hand side computation flow (right).

Inside the Runge-Kutta 4 algorithm, we follow the flow shown in Figure **5.3**. This flow consists of:

- **Compute the $f_{\mathbf{odd}}$ right-hand side:** Given a vector u and time t, compute the first Runge-Kutta step.

  → **Prepare the vector $u_{\mathbf{tmp}}$ and check the boundaries:** Prepare the vector u_{tmp} for the next Runge-Kutta step and verify that the particle is valid (inside the global bounds). If it is not, mark it as invalid and stop the iteration.

- **Compute the $f_{\mathbf{even}}$ right-hand side:** Given a vector u_{tmp} and time t, compute the second Runge-Kutta step.

  → **Prepare the vector $u_{\mathbf{tmp}}$ and check the boundaries**

- **Store partial updates:** To reuse f_{odd} and f_{even}, we store partial results at the midstep.

- **Compute the $f_{\mathbf{odd}}$ right-hand side:** Given a vector u_{tmp} and time t, compute the third Runge-Kutta step.

  → **Prepare the vector $u_{\mathbf{tmp}}$ and check the boundaries**

- **Compute the $f_{\mathbf{even}}$ right-hand side:** Given a vector u_{tmp} and time t, compute the fourth Runge-Kutta step.

- **Update the final particle:** Perform the second half of the Runge-Kutta 4 steps.

  → **Check the boundaries:** Finally, we check the boundary conditions for the final particle position.

The previous flow is repeated for each particle in the particles container. Continuing with the flow chart, the *compute_rhs* process takes as input parameter a parameter vector u, the current time t, the timestep dt and the 3+1 parameters γ, α, β and K. The code then performs the following steps:

- **Barycentric interpolation:** Obtain approximation at the position x, y, z of the lapse function, the shift vector, the induced metric and its partial derivatives, and the extrinsic curvature.

- **Construct the right-hand side:** Construct the right-hand side of equations (2.76) and (2.77) using the interpolated data.

5.3 Code Capabilities

Understanding the algorithm behind the code, we can now go into more detail about the capabilities and implementation of the code itself. In the following subsections, we present the configuration of the tool as multiple thorns and their relationships, the implementations of core functionalities shown in the previous subsection, and a set of complementary features. These features have been useful for code testing and can facilitate integration with other thorns.

5.3.1 Thorns configuration

The project has been split into three thorns: two that define utilities common to future particles implementations, and one that defines the class used for geodesic integration.

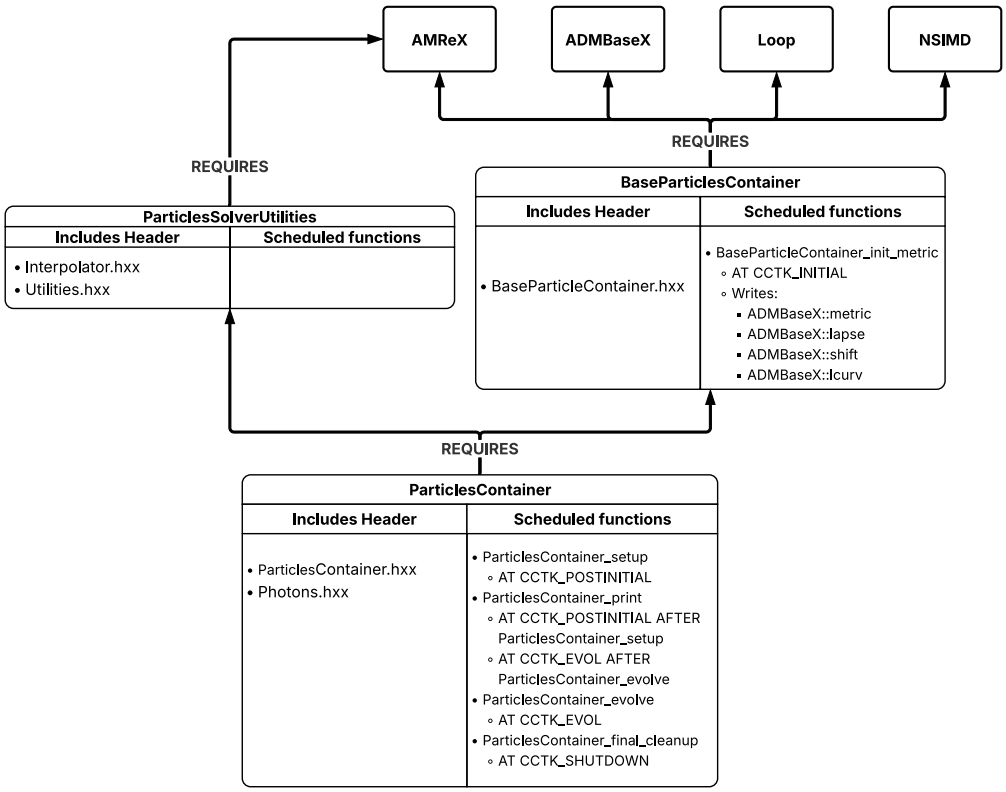

Figure 5.4: GInX's implemented thorns, showing their relationships, exposed header files and scheduled functions.

Figure **5.4** shows the thorns created, their scheduled functions, and the files that are exposed to other thorns. The *BaseParticlesContainer* thorn requires the headers and implementations from several sources: *ADMBaseX* for spacetime variables, *AMReX*, *Loop* a

CarpetX thorn, and the *NSIMD* thorn. The *ParticlesSolverUtilities* requires the capabilities defined in *AMReX*. The *ParticlesContainer* thorn requires the features defined by the other two GInX thorns.

Using the *schedule.ccl* file, each thorn can define where its functions will be called by the Cactus flesh. The location where a function can be scheduled is called a timebin. All the standard timebins can be found in [22]. Those used in this project are *CCTK_INITIAL*, *CCTK_POSTINITIAL*, *CCTK_EVOL* and *CCTK_SHUTDOWN*. The scheduled functions are called in the following order:

```
...
Initialisation
    ...
    [CCTK_INITIAL]
      BaseParticlesContainer::BaseParticlesContainer_init_metric: \
          [local] Fields setup
    [CCTK_POSTINITIAL]
      ParticlesContainer::ParticlesContainer_setup: [global] setup
      ParticlesContainer::ParticlesContainer_print: \
          [global] print the particles
    ...
...
do loop over timesteps
  ...
  [CCTK_EVOL]
    ParticlesContainer::ParticlesContainer_evolve: \
      [global] Photons evolution
    ParticlesContainer::ParticlesContainer_print: \
      [global] print the particles
  ...
enddo
...
Shutdown routines
  [CCTK_SHUTDOWN]
    ParticlesContainer::ParticlesContainer_final_cleanup: [global] \
              cleaning everything
```

Listing 5: GInX scheduled functions within the corresponding timebins. The other timebins are not used and have been omitted.

5.3.2 Code Implementation

In order to numerically solve the particle geodesics, the simulation employs three important functions, which will be discussed next. These are: the interpolation functions, the method to compute the right-hand side, and the method that calls the Runge-Kutta 4 integrator for the time evolution.

5.3.2.1 Interpolation Implementation

For the interpolation of the functions and their derivatives, we interpolate using polynomials on each direction sequentially, starting with the x direction and ending with the z direction. This implies that we ultimately compute

$$f(x,y,z) \approx \frac{\sum_{k=0}^{n} \frac{u_k}{z-z_k} H(x,y,z_k)}{\sum_{l=0}^{n} \frac{u_l}{z-z_l}}, \tag{5.1}$$

using

$$H(x,y,z_k) \approx \frac{\sum_{j=0}^{n} \frac{v_j}{y-y_j} G(x,y_j,z_k)}{\sum_{l=0}^{n} \frac{v_l}{y-y_l}}, \tag{5.2}$$

which in turn uses

$$G(x,y_j,z_k) \approx \frac{\sum_{i=0}^{n} \frac{w_i}{x-x_i} f(x_i,y_j,z_k)}{\sum_{l=0}^{n} \frac{w_l}{x-x_l}}. \tag{5.3}$$

These expressions are equivalent to equation (3.20).

For the derivatives, we use a similar approach, adapting the interpolation formula accordingly. For instance, the partial derivative with respect to x is given by:

$$\partial_x f(x,y,z) \approx \frac{\sum_{k=0}^{n} \frac{u_k}{z-z_k} \frac{\sum_{j=0}^{n} \frac{v_j}{y-y_j} \frac{d}{dx} \left(\frac{\sum_{i=0}^{n} f(x_i,y_j,z_k) \frac{w_i}{x-x_i}}{\sum_{l=0}^{n} \frac{w_l}{x-x_l}} \right)}{\sum_{l=0}^{n} \frac{v_l}{y-y_l}}}{\sum_{l=0}^{n} \frac{u_l}{z-z_l}}. \tag{5.4}$$

This means that we first interpolate the derivative along x, and then apply the usual interpolation along y and z. For the partial derivative along y, we first compute the interpolation along x, then the the derivative interpolation along y, and finally an interpolation along the z direction. This approach is equivalent to equations (3.23).

To compute the three-dimensional interpolation, we can make use of a function that interpolates along one dimension and call it sequentially for each direction. This is illustrated by the following pseudo-code:

```
barycentric_cubic_1d(interpolated_value, f, x, weight, gridpoints){
    for (int i = 0; i < INTERPOLATION_ORDER; i++){
        if ( x is a gridpoint ){
            interpolated_value = f(gridpoints[i]);  return;
        }
        common_term = weight[i] / (x - gridpoints[i]);
        numerator += common_term * f(gridpoints[i]);
        denominator += common_term;
    }
    interpolated_value = numerator / denominator;
}
barycentric_cubic_3d(f, x, y, z) {
    gridpoints = compute gridpoints given the interpolation order;
    weights = compute weights given the interpolation order;
    // Interpolate along x
    for (int j = 0; j < INTERPOLATION_ORDER; j++){
        for (int k = 0; k < INTERPOLATION_ORDER; k++){
            for (int i = 0; i < INTERPOLATION_ORDER; i++){
                values[i] = f(x_i, y_j, z_k);
            }
            barycentric_cubic_1d(G[j][k], values, x, weights, gridpoints);
        }
    }
    // Interpolate along y using G
    for (int k = 0; k < INTERPOLATION_ORDER; k++){
        for (int j = 0; j < INTERPOLATION_ORDER; j++){
            values[j] = G(x, y_j, z_k);
        }
        barycentric_cubic_1d(H[k], values, y, weights, gridpoints);
    }
```

```
    // Interpolate along z using H
    for (int k = 0; k < INTERPOLATION_ORDER; k++){
        values[k] = H(x, y, z_k);
    }
    barycentric_cubic_1d(interpolated_value, values, z, weights, gridpoints);
    return interpolated_value;
}
```

Listing 6: Pseudo-code of the *barycentric_cubic_1d* and *barycentric_cubic_3d* functions defined within the *ParticlesSolverUtilities* thorn.

In a similar way, we can compute the interpolated value and the interpolation of the derivative in just one call. The function *der_barycentric_cubic_1d* is shown in Listing 7 computes both values. In this way, we can alternate it between calls within a function similar to *barycentric_cubic_3d* that computes both the value and the gradient of a given function f. The pseudo-code for the derivative along one direction is as follows:

```
der_barycentric_cubic_1d(f_x, df_dx, f, x, weight, gridpoints){
    for (int i = 0; i<INTERPOLATION_ORDER; i++){
        if(x is a gridpoint){
            for(int j=0; j < INTERPOLATION_ORDER; j++){
                if(i!=j) continue;
                df_dx += weights[j] * (f(gridpoints[i]))
                            / (gridpoints[j] - gridpoints[i]);
            }
            df_dx /= weights[i];
            f_x = f(gridpoints[i]);
            return;
        }
        common_term = weight[i] / (x - gridpoints[i]);
        numerator += common_term * f(gridpoints[i]);
        denominator += common_term;
        den_sqr += common_term / (x - gridpoints[i]);
    }
```

```
    for (int i = 0; i<INTERPOLATION_ORDER; i++){
        term += (weights[i] / (x - gridpoint[i];
        der_num += (-term * denominator
                    / (x -gridpoints[i]) +
                    term * den_sqr) * f(x[i]);
    }

    f_x = numerator / denominator;
    df_dx = der_num / (denominator * denominator);
}
```

Listing 7: Pseudo-code of the *der_barycentric_cubic_1d* function defined within the *ParticlesSolverUtilities* thorn.

Where 7 computes the interpolation of $f(x)$ and $f'(x)$. In order to compute the gradient of a multivariate function, we alternate between the *der_barycentric_cubic_1d* and the *barycentric_cubic_1d* functions as needed. This approach is shown in the following pseudo-code:

```
barycentric_derivate_and_interpolate(f_xyz, df_xyz_0, df_xyz_1, df_xyz_2,
                                     f, x, y, z){
    gridpoints = compute gridpoints given the interpolation order;
    weigths = compute weights given the interpolation order;
    // Interpolate along x and the partial derivative of x
    for (int j = 0; j < INTERPOLATION_ORDER; j++){
        for (int k = 0; k < INTERPOLATION_ORDER; k++){
            for (int i = 0; i < INTERPOLATION_ORDER; i++){
                values[i] = f(x_i, y_j, z_k);
            }
            der_barycentric_cubic_1d(G[j][k], G_dxyz, values, x,
                                     weights, gridpoints);
        }
    }
    // Interpolate along y and the partial derivative of y
    for (int k = 0; k < INTERPOLATION_ORDER; k++){
        for (int j = 0; j < INTERPOLATION_ORDER; j++){
                values[j] = G(x, y_j, z_k);
                values_dx[j] = G_dxyz(x, y_j, z_k);
        }
```

```
        der_barycentric_cubic_1d(H[j][k], H_xdyz, values, y, weights,
            gridpoints);
        barycentric_cubic_1d(H_dxyz[j][k], values_dx, y, weights,
            gridpoints);
    }
    // Interpolate along z and the partial derivative of z
    for (int k = 0; k < INTERPOLATION_ORDER; k++){
        values[k] = H(x, y, z_k);
        values_dy[k] = H_xdyz(x, y, z_k);
        values_dx[k] = H_dxyz(x, y, z_k);
    }
    // f(x,y,z) and \partial_z f(x,y,z)
    der_barycentric_cubic_1d(f_xyz, df_xyz_2, values, z, weights,
        gridpoints);
    // \partial_x f(x,y,z)
    barycentric_cubic_1d(df_xyz_0, values_dx, z, weights, gridpoints);
    // \partial_y f(x,y,z)
    barycentric_cubic_1d(df_xyz_1, values_dy, z, weights, gridpoints);
}
```

Listing 8: Pseudo-code of the *barycentric_derivate_and_interpolate* function defined within the *ParticlesSolverUtilities* Thorn.

To extend the previous functions to tensors or vectors, we cat iterate over each component and obtain their interpolated value.

5.3.2.2 Compute the Right-Hand Side Implementation

The system of equations (2.76) and (2.77) can be expressed as a vector $\vec{U}$ of size seven, where the first three elements are the particle coordinates x, y, z, the next three elements are the components of the V_i tensor, and the last element is $\ln E$ for each particle. Using the equation (3.5), we can implement the numerical integration. The system, expressed in terms of U, is

$$
\begin{aligned}
\frac{dU[i]}{dt} &= \alpha\gamma^{ij}U[3+j] + \beta^i, \\
\frac{dU[3+i]}{dt} &= -\partial_i\alpha + \left(\gamma^{kj}U[3+k]\partial_j\alpha - \alpha K_{jk}\gamma^{jl}\gamma^{km}U[3+l]U[3+m]\right)U[3+i] \\
&\quad + \frac{1}{2}\alpha\gamma^{jl}\gamma^{km}U[3+l]U[3+m]\partial_i\gamma_{jk} + U[3+j]\partial_i\beta^j, \\
\frac{dU[6]}{dt} &= \alpha K_{jk}\gamma^{jl}\gamma^{km}U[3+l]U[3+m] - U[3+l]\gamma^{lj}\partial_j\alpha
\end{aligned}
\tag{5.5}
$$

with $i, j, k, l, m = 0, 1, 2$ and using Einstein summation convention.

The pseudo-code for the function that computes the right-hand side is as follows:

```
compute_rhs(u, lapse, shift, metric, curv) {
    lapse = interpolate the lapse function;
    d_lapse = interpolate the partial derivatives of the lapse;
    shift = interpolate the shift function;
    d_shift = interpolate the partial derivatives of the shift;
    gamma = interpolate the induced metric;
    d_gamma = interpolate the partial derivatives of the induced metric;
    curv = interpolate the extrinsic curvature;
    gamma_inv = compute the inverse of the gamma_{ij} matrix;
    V_up = gamma_inv @ V_down;

    rhs[0] = lapse_x * V_up[0] - shift_x[0];
    rhs[1] = lapse_x * V_up[1] - shift_x[1];
    rhs[2] = lapse_x * V_up[2] - shift_x[2];

    for(int i = 0; i < 3; i++){
        rhs[3+i] = -d_lapse[i]
                   + (d_lapse @ V_up
                     - lapse * curv @ V_up) @ V_up)) * V_down[i]
                   + 0.5 * lapse * (d_gamma[i] @ V_up) @ V_up)
                   + V_down @ d_shift[i];
    }
```

```
    rhs[6] = lapse * (curv_x @ V_up) @ V_up - V_up @ d_lapse;
    return rhs;
}
```

Listing 9: Pseudo-code of the *compute_rhs* function defined within the *ParticlesContainer* thorn. The symbol @ represents a matrix-vector multiplication.

With this, we can use the interpolated values into the implementation of the Runge-Kutta method.

5.3.2.3 Particles Evolution Implementation

The *evolve* method of the *ParticlesContainer* class goes through each grid associated with it and retrieves the data of the needed fields. It then iterates over each particle using a *ParallelFor* loop, integrating each trajectory with the Runge-Kutta method. The pseudo-code is shown bellow:

```
ParticlesContainer<...>::evolve(lapse, shift, metric, curv, dt, level){
    for(ParticleIterator pti; pti.isValid(); pti++) {
        np = number of particles inside the current pti;
        lapse_array = lapse.array(pti); shift_array = shift.array(pti);
        metric_array = metric.array(pti); curv_array = curv.array(pti);
        ParallelFor(np, [=](int i) {
            U = { particle[i].x, particle[i].y, particle[i].z,
                  particle[i].vx, particle[i].vy, particle[i].vz,
                  particle[i].ln_E};
            k_odd = compute_rhs(U, lapse_array, shift_array,
                        metric_array, curv_array);
            U_tmp = U + 0.5 * dt * k_odd;
            if (U_tmp is out of bounds){
                mark particle as invalid;  return;
            }
            k_even = compute_rhs(U_tmp, lapse_array,
                        shift_array, metric_array, curv_array);
            U_tmp = U + 0.5 * dt * k_even;
```

```
            if (U_tmp is out of bounds){
                mark particle as invalid;  return;
            }
            particle[i] += (1/6) * dt * (k_odd + 2 * k_even);
            k_odd = compute_rhs(U, lapse_array, shift_array,
                        metric_array, curv_array);
            U_tmp = U + dt * k_odd;
            if (U_tmp is out of bounds){
                mark particle as invalid; return;
            }
            k_even = compute_rhs(U_tmp, lapse_array,
                        shift_array, metric_array, curv_array);
            particle[i] += (1/6) * dt * (2 * k_odd + k_even);
            if (particle[i] is out of bounds){
                mark particle as invalid; return;
            }
        });
    }
}
```

Listing 10: Pseudo-code of the *evolve* method from the *ParticlesContainer* class.

5.3.3 Thorns Utilities

The *ParticlesContainer* thorn includes other utilities that are not directly related to numerical geodesic integration but can potentially be used along the simulation. These utilities are:

- **Initial conditions:** The thorn provides two functions that initialize an arbitrary number of particles with random positions and velocities but fixed energy. The main difference is that one function initializes particles distributed within a sphere of radius R, while the other distributes them across the entire rectangular grid. This can be managed by the *num_photons*, *initializer*, *init_params_d[10]*, and *init_params_i[10]* parameters.

- **Velocity normalization:** The *ParticlesContainer* includes the method *normalize_velocity*, which uses the induced metric to normalize the particle velocities to the values given in equation (2.79). One advantage of the system we are solving is that it is valid for both null and timelike particles. The only difference is the initial normalization, making this a potentially important step in the simulation.

The *BaseParticlesContainer* thorn includes a set of more general functions and methods that can be used by any derived class. The available capabilities are:

- **Metric Initialization:** The thorn can initialize the *ADMBaseX* variables using several spacetimes in Cartesian coordinates. The available spacetimes are: Minkowski, Schwarzschild in Isotropic, Gullstrand–Painlevé, and standard coordinates; and Kerr spacetime in Kerr-Schild coordinates. This can be managed by the *metric*, *metric_params_d[10]*, and *metric_params_i[10]* parameters.

- **Banned zones:** In the simulation, some regions bust be excluded. For instance, spacetime singularities or the black hole event horizon. The thorn can mark particles as invalid if they are inside of a ball centered at x, y, z with radius r, and then remove them from the simulation. The thorn also allows the user to define up to ten banned zones. This is managed by the *banned_region*, *region_N_position[3]* and *region_N_radius* parameters, where N is a number from 1 to 10.

- **Output method:** AMReX provides parallel I/O functions. The thorn extends these capabilities, which can be accessed by the user through parameters defined within the *.par* file. The type of output can be managed by the *particle_plot_every* and *particle_tsv_every* parameters.

Chapter 6

Code Validation and Verification

In order to verify that the code is performing as intended and achieves the expected order of accuracy, we can perform several spacetime tests. First, the code is intended to solve a real physical system; therefore, we track physical quantities that should remain constant throughout the simulation. To measure the deviations with respect to the expected values we are using the root squared mean error (RSME).

On the other hand, we must also test the spatial interpolation in order to confirm that the error indeed decreases as expected.

6.1 Conserved Quantities

The spacetimes that we used for testing the code are the Kerr and Schwarzschild solutions. These, along with the Reissner–Nordström and Kerr-Newman spacetimes, form the main family of analytically known black hole solutions. Since both the Kerr and Schwarzschild spacetimes are stationary and axisymmetric, two quantities must be conserved along each geodesic:

- The 4-momentum component p_t.
- The 4-momentum component p_ϕ, which in Cartesian coordinates corresponds to $L_z = xp_y - yp_x$.

The momentum components must satisfy the invariant mass conservation given by equation 2.79. Moreover, given the spherical symmetry in the Schwarzschild spacetime, the angular momentum components along each direction are also conserved. Specifically, the quantities $L_x = yp_z - zp_y$ and $L_y = xp_z - zp_x$ must remain constant throughout the simulation. These conserved quantities, together with the velocity normalization condition for particles (equation (2.79)), allow us to track the error introduced by the integrator throughout the simulation and to verify the correctness of the geodesic solution.

6.1.1 3+1 Decomposition of 4-momentum p_μ

Before we can verify the conservation of p_t along the trajectories during the integration, we need express the time component of the particle's 4-momentum in terms of the 3+1 variables. Using the definition in equation (2.37) with (2.25), we can rewrite it as

$$\begin{aligned} p_t &= g_{t\mu}p^\mu \\ &= g_{tt}p^t + g_{ti}p^i \\ &= E(-\alpha^2 + \beta_k\beta^k)n^t + E\beta_i(n^i + V^i) \\ &= E\left(-\alpha + \frac{\beta_k\beta^k}{\alpha} + \beta_i\left(-\frac{\beta^i}{\alpha} + V^i\right)\right) \\ &= E\left(-\alpha + \beta^i V_i\right). \end{aligned} \tag{6.1}$$

Repeating the previous procedure for the p_i components, we obtain

$$\begin{aligned} p_i &= g_{i\mu}p^\mu \\ &= g_{it}p^t + g_{ij}p^i \\ &= E\beta_i n^t + E\gamma_{ij}(n^i + V^i) \\ &= E\left(\frac{\beta_i}{\alpha} + \gamma_{ij}\left(-\frac{\beta^j}{\alpha} + V^j\right)\right) \\ &= E\left(\frac{\beta_i}{\alpha} - \frac{\beta_i}{\alpha} + V_i\right) = EV_i. \end{aligned} \tag{6.2}$$

Then, the conserved quantities for both spacetimes are

$$\begin{aligned} p_t &= E\left(-\alpha + \beta^i V_i\right), \\ p_\phi &= E\left(xV_y - yV_x\right). \end{aligned} \tag{6.3}$$

Additionally, for the Schwarzschild spacetime we have conservation of the following quantities:

$$\begin{aligned} L_x &= E\left(yV_z - zV_y\right), \\ L_y &= E\left(xV_z - zV_x\right). \end{aligned} \tag{6.4}$$

6.2 Schwarzschild Spacetime

The Schwarzschild spacetime provides a good test for the code. It is more complex than Minkowski space and has a set of conserved quantities that can be tracked throughout the

simulation, which help us verify the convergence of the solver with increasing resolution, as expected from the Runge-Kutta methods.

In order to test the code, we can use the spacetime described in different coordinate systems of increasing complexity. We have tested with isotropic and Painlevé-Gullstrand coordinates, which are described in the following subsections.

6.2.1 Isotropic Schwarzschild Coordinates

One of the known analytic solutions to the Einstein equations is the Schwarzschild metric and its equivalent solutions in other coordinate systems. The Isotropic coordinate system is convenient when working with Cartesian coordinates. The line element $ds^2 = g_{\alpha\beta}dx^\alpha dx^\beta$ in Isotropic Schwarzschild coordinates is defined as follows:

$$ds^2 = -\frac{\left(1-\frac{M}{2R}\right)^2}{\left(1+\frac{M}{2R}\right)^2}dt^2 + \left(1+\frac{M}{2R}\right)^4\left(dx^2+dy^2+dz^2\right), \tag{6.5}$$

where $R = \sqrt{x^2+y^2+z^2}$. From this, we can identify:

$$\alpha = \frac{1-\frac{M}{2R}}{1+\frac{M}{2R}}, \tag{6.6}$$

$$\beta^i = 0, \tag{6.7}$$

$$\gamma_{xx} = \gamma_{yy} = \gamma_{zz} = \left(1+\frac{M}{2R}\right)^4, \quad \gamma_{xy} = \gamma_{xz} = \gamma_{yz} = \gamma_{yx} = \gamma_{zx} = \gamma_{zy} = 0, \tag{6.8}$$

$$K_{ij} = 0. \tag{6.9}$$

6.2.1.1 Conserved Quantities in Isotropic Coordinates

Based on the results shown in Figure **6.5**, error saturation occurs at a grid resolution of approximately 10^{-2}. Balancing this with computational feasibility, we ran the test cases with a grid spacing of $10 \times 2^{-6} = 0.078125$ in each direction for a box of size $40M$. The timestep was set equal to the grid spacing. The black hole's mass is $M = 0.5$. The particles' position and velocity are random, and the initial energy is $E = 1$ for both timelike and null geodesics. The timelike particles have a mass of $m = 0.01$. The errors have been measured only for particles that remain valid at the end of the simulation. In this case, we are not including errors introduced by the event horizon and singularities.

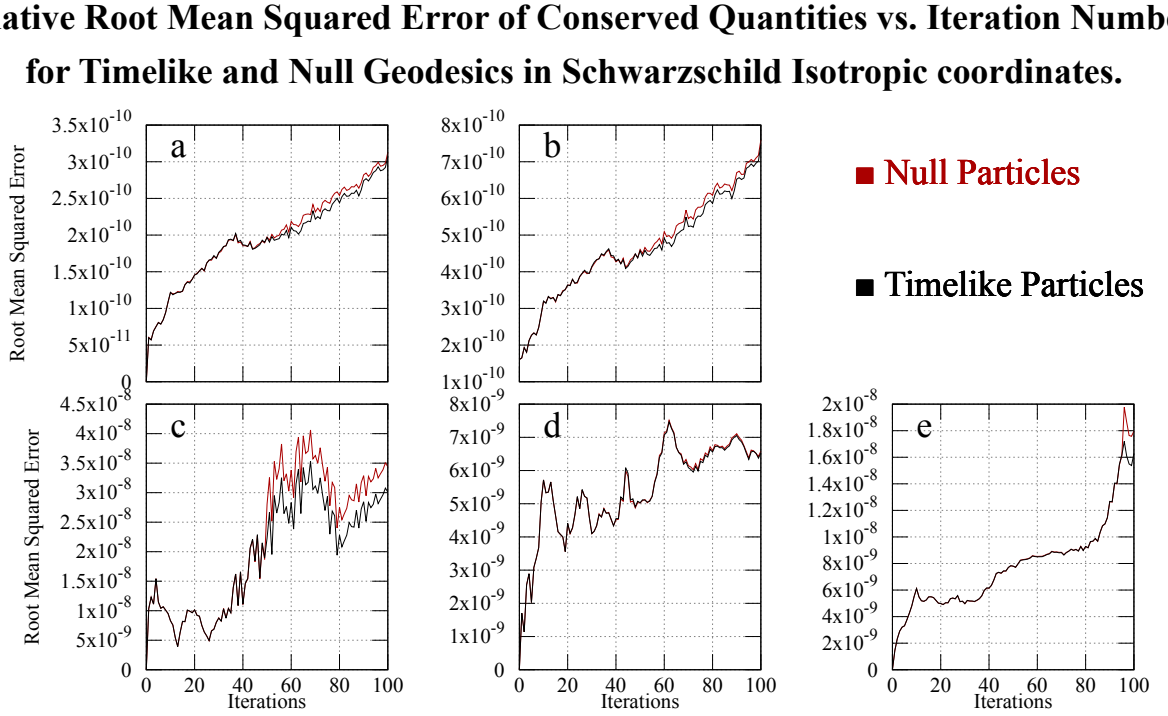

Figure 6.1: Relative Root Mean Squared Error (RMSE) of the conserved quantities as a function of iteration number for timelike and null geodesics integrated over 100 steps using the Schwarzschild isotropic coordinates.

Figure **6.1** shows the relative deviation $(C(t = 0) - C(t))/C(t = 0)$ for the conserved quantities in Schwarzschild spacetime. The panels (a)–(e) present p_t, the velocity normalization, L_x, L_y, and L_z, respectively. It also shows the absolute RMSE for the velocity normalization over a 100-step simulation integrating null and timelike geodesics.

The graphs for p_t, L_x, L_y, L_z show how the relative error increases as the system evolves for both particle types. The plots have a maximum RMSE values of approximately 3×10^{-10}, 4.5×10^{-8}, 8×10^{-9}, and 2×10^{-8} for the conserved quantities p_t, L_x, L_y, L_z, respectively. This indicates that the quantities are conserved, as expected from the theory, within the limits permitted by the grid resolution.

As illustrated in the upper-center plot, the RMSE of the velocity normalization constraint, which both null and timelike particles must satisfy, is monitored. The measured relative error remains below 10^{-9} throughout the simulation for both timelike and null geodesics.

The conserved quantities and the velocity normalization for Schwarzschild spacetime in isotropic coordinates, exhibit the expected behavior shown in theory, with maximum RMSE

deviations bellow 10^{-7}. The measured values are consistent with the anticipated behavior, given the grid spacings (Δx, Δy and Δz) and the timestep (Δt). The Schwarzschild spacetime in isotropic coordinates exhibits lower accumulated error at the chosen resolution compared to the Painlevé-Gullstrand coordinates and the Kerr spacetime.

6.2.2 Painlevé-Gullstrand Coordinates

The Painlevé-Gullstrand coordinates introduce a coordinate system with a non-zero shift vector and extrinsic curvature. The Schwarzschild line element in these coordinates, using Cartesian coordinates, is described as follows:

$$ds^2 = -\left(1 - \frac{2m}{r}\right) dt^2 + \frac{x_i}{r}\sqrt{\frac{2m}{r}} dx^i dt + dx^2 + dy^2 + dz^2. \tag{6.10}$$

Where $r^2 = x^2 + y^2 + z^2$ and $i = 1, 2, 3$. From this, we can deduce the 3+1 variables needed for the simulation:

$$\gamma_{ij} = \delta_{ij}, \tag{6.11}$$

$$\beta^i = \frac{x^i}{r}\sqrt{\frac{2m}{r}}, \quad \text{and since } \gamma_{ij} = \delta_{ij}, \text{ we have } \beta_i = \beta^i. \tag{6.12}$$

$$\begin{aligned} \alpha^2 &= 1 - \frac{2m}{r} + \beta_i \beta^i \\ &= 1 - \frac{2m}{r} + \delta_{ij}\beta^i\beta^j \\ &= 1 - \frac{2m}{r} + \frac{x^2 + y^2 + z^2}{r^2}\frac{2m}{r} \\ &= 1. \end{aligned} \tag{6.13}$$

For the extrinsic curvature, we can use the equation (2.71) to extract the following relation:

$$K_{ij} = \frac{1}{2\alpha}\left(\partial_i \beta_j + \partial_j \beta_i - 2\,{}^3\Gamma^k_{ij}\beta_k\right). \tag{6.14}$$

Since $\alpha = 1$ and $\gamma_{ij} = \delta_{ij}$ implies ${}^3\Gamma^k_{ij} = 0$, the expression simplifies to:

$$K_{ij} = \frac{1}{2}\left(\partial_i \beta_j + \partial_j \beta_i\right). \tag{6.15}$$

Computing the first derivative, we obtain:

$$\begin{aligned}\partial_j \beta_i &= \partial_j \left(x_i \sqrt{\frac{2m}{r^3}} \right) \\ &= \delta_{ij} \sqrt{\frac{2m}{r^3}} - \frac{3m x_i x_j}{r^4 \sqrt{\frac{2m}{r}}} \\ &= \sqrt{\frac{2m}{r^3}} \left(\delta_{ij} - \frac{3 x_i x_j}{2r^2} \right),\end{aligned} \tag{6.16}$$

Then, we can write the extrinsic curvature as follows:

$$\begin{aligned}K_{ij} &= \frac{1}{2} \sqrt{\frac{2m}{r^3}} \left(\delta_{ij} - \frac{3 x_i x_j}{2r^2} + \delta_{ji} - \frac{3 x_j x_i}{2r^2} \right) \\ &= \sqrt{\frac{2m}{r^3}} \left(\delta_{ij} - \frac{3 x_i x_j}{2r^2} \right).\end{aligned} \tag{6.17}$$

6.2.2.1 Conserved Quantities in Painlevé-Gullstrand Coordinates

We performed the same test for the Schwarzschild spacetime in isotropic coordinates and repeated for the Painlevé-Gullstrand coordinates. For consistency, we ran a simulation with the same grid spacing, which is equal to 10×2^{-6} for all three dimensions, and with a timestep of 0.078125. The black hole's mass is $M = 0.5$. The particles' position and velocity are random, and the initial energy is $E = 1$ for both timelike and null geodesics. The timelike particles have a mass $m = 0.01$. The errors were measured only for particles that remain valid at the end of the simulation. In this case, we are not including errors introduced by the event horizon and singularities.

Figure **6.2** is similar to what we showed for the isotropic coordinates. It displays the relative deviation $(C(t=0) - C(t))/C(t=0)$ for the conserved quantities. The panels (a)–(e) present p_t, the velocity normalization, L_x, L_y, and L_z, respectively. It also shows the absolute RMSE for the velocity normalization over a 100-step simulation for both null and timelike geodesics.

The graphs of p_t, L_x, L_y, L_z show how the relative error increases as the system evolves for both particles types. The plots show maximum RMSE values of approximately 3.5×10^{-9}, 2.5×10^{-6}, 1.6×10^{-7}, and 4×10^{-8} for the conserved quantities p_t, L_x, L_y, L_z, respectively. The results are still in line with the quantities conservation and the grid resolution should produce.

As illustrated in the upper-center plot, the RMSE of the velocity normalization constraint, which both null and timelike particles must satisfy, is monitored. The measured relative

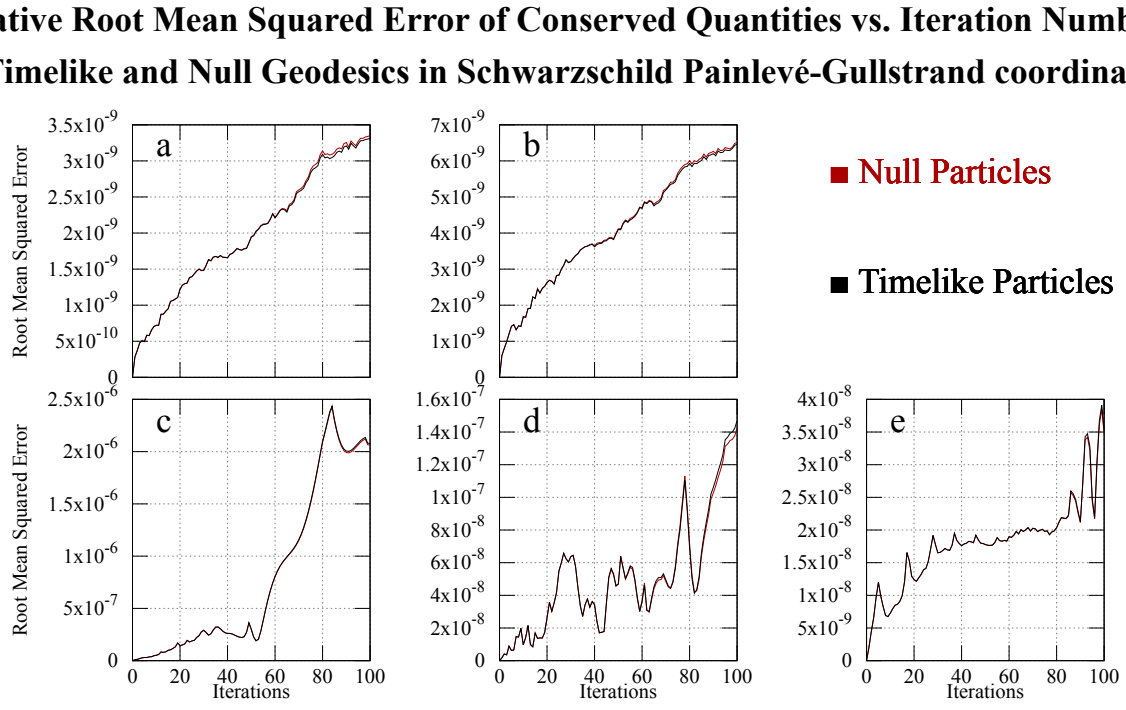

Figure 6.2: Relative Root Mean Squared Error (RMSE) of the conserved quantities as a function of iteration number for timelike and null geodesics integrated over 100 steps using the Schwarzschild Painlevé-Gullstrand coordinates.

error remains below 7×10^{-9} throughout the simulation for both timelike and null geodesics.

The conserved quantities and velocity normalization for the Schwarzschild spacetime in Painlevé-Gullstrand coordinates exhibit the expected behavior shown in theory, with maximum RMSE deviations below 10^{-5}. The results for the Painlevé-Gullstrand coordinates show a higher accumulated error than the results for the isotropic ones. This could be due to the non-zero nature of the extrinsic curvature and shift vector, which makes the spacetime more difficult to interpolate. Overall, the measured values are consistent with the anticipated interpolation and time integration error orders, given the grid spacings (Δx, Δy, and Δz) and timestep (Δt).

6.3 Kerr Spacetime

The Kerr spacetime is the unique vacuum solution that is stationary and asymptotically flat. It describes a single rotating black hole spacetime [26]. The Kerr metric is more complex than the Schwarzschild one and represents a valuable test for the code behavior in a more realistic scenario.

6.3.1 Kerr-Schild Coordinates

In the original paper [27], the Kerr-Schild coordinates were presented for the first time. These coordinates can be defined in terms of Cartesian coordinates (t, x, y, z) [28], which is convenient for our purposes. Defining

$$r^2 = \frac{x^2 + y^2 + z^2 - a^2}{2} + \sqrt{\left(\frac{x^2 + y^2 + z^2 - a^2}{2}\right)^2 + a^2 z^2}, \tag{6.18}$$

we define the Kerr metric in Kerr-Schild coordinates as:

$$g_{ab} = \eta_{ab} + \frac{2mr^3}{r^4 + a^2 z^2} l_a l_b, \tag{6.19}$$

where η_{ab} is the Minkowski metric and

$$l_a = \left(1, \frac{rx + ay}{r^2 + a^2}, \frac{ry - ax}{r^2 + a^2}, \frac{z}{r}\right). \tag{6.20}$$

Here $l_a l^a = 0$ with respect to both η_{ab} and g_{ab} and

$$g^{ab} = \eta^{ab} - \frac{2mr^3}{r^4 + a^2 z^2} l^a l^b, \tag{6.21}$$

with

$$l^a = \left(-1, \frac{rx + ay}{r^2 + a^2}, \frac{ry - ax}{r^2 + a^2}, \frac{z}{r}\right). \tag{6.22}$$

Using the definitions shown in equation 2.33, we can obtain the 3+1 variables needed for the particle evolution,

$$\alpha^2 = -\frac{1}{g^{00}} = \frac{1}{1 + \dfrac{2mr^3}{r^4 + a^2 z^2}}, \tag{6.23}$$

$$\beta^i = \alpha^2 g^{0i} = \frac{\dfrac{2mr^3}{r^4 + a^2 z^2}}{1 + \dfrac{2mr^3}{r^4 + a^2 z^2}} l^i \tag{6.24}$$

and finally

$$\gamma_{ij} = g_{ij} = \delta_{ij} + \frac{2mr^3}{r^4 + a^2 z^2} l_i l_j. \tag{6.25}$$

For the extrinsic curvature K_{ij} we can use the equation 2.71, and using the fact that γ_{ij} is independent of the time, we have

$$
\begin{aligned}
K_{ij} &= \frac{1}{2\alpha}\left(\partial_i\beta_j + \partial_j\beta_i - 2\,{}^3\Gamma^k_{ij}\beta_k\right) \\
&= \frac{1}{2\alpha}\left(\partial_i\beta_j + \partial_j\beta_i - \beta^k\left(\partial_i\gamma_{kj} + \partial_j\gamma_{ik} - \partial_k\gamma_{ij}\right)\right).
\end{aligned}
\tag{6.26}
$$

Here we have to compute analytically the derivatives of β and γ to obtain the extrinsic curvature for use in the simulation. The computation of the extrinsic curvature is present in the code but is too large to be presented here.

6.3.1.1 Conserved Quantities in Kerr-Schild Coordinates

As we did with the other spacetime tests, we tracked the conserved quantities of the Kerr spacetime. We ran a simulation with the same settings as the previous test cases. The grid spacing was set to 0.078125 for all three dimensions, and the timestep was also 0.078125. The black hole mass was set to 0.5, and we included a spin parameter $a = 0.1$. The particles' position and velocity are randomly set at the beginning, and the initial energy is set to $E = 1$ for both timelike and null geodesics. The timelike particles have a mass $m = 0.01$. The errors were measured only for particles that remain valid at the end of the simulation. In this case, we are not including errors introduced by the event horizon and singularities.

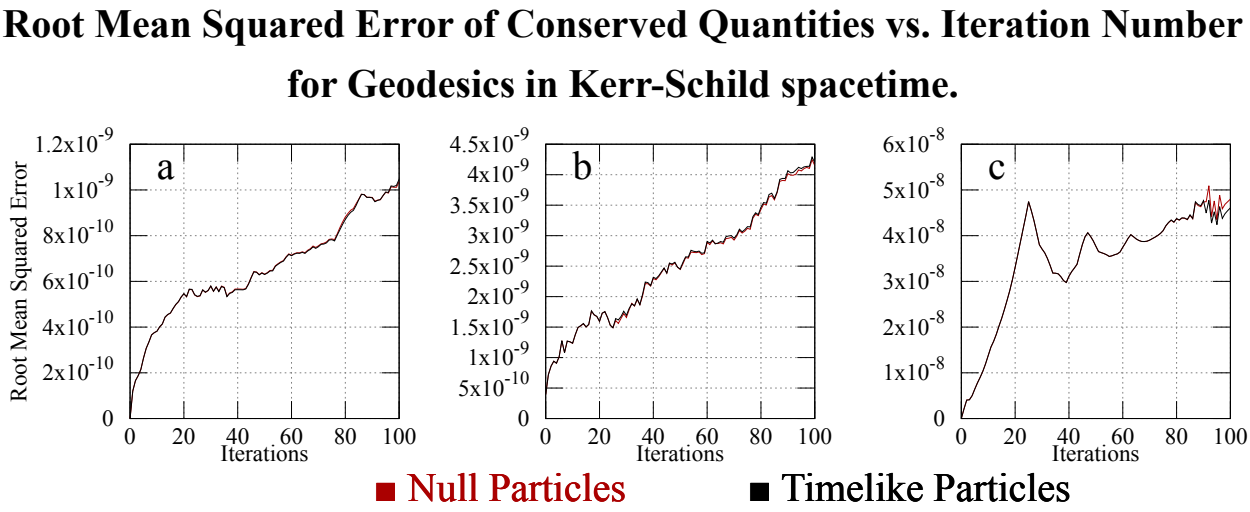

Figure 6.3: Relative Root Mean Squared Error (RMSE) of the conserved quantities as a function of iteration number for timelike and null geodesics integrated over 100 steps using the Kerr-Schild coordinates.

Figure **6.3** is similar to what we showed for the Schwarzschild spacetime, but now displaying the conserved quantities of the Kerr spacetime. It shows the relative deviation

$(C(t = 0) - C(t))/C(t = 0)$ for the conserved quantities. The panels (a)–(c) present p_t, the velocity normalization and L_z, respectively. It also shows the absolute RMSE for the velocity normalization over a 100-step simulation for both null and timelike geodesics.

The graphs of p_t and L_z show how the relative error increases as the system evolves for both particle types. The plots show maximum RMSE values of approximately 1.2×10^{-9} and 5×10^{-8} for the conserved quantities p_t and L_z, respectively. The results consistent within the tolerance range given by the grid resolution and remain at the same order of magnitude as those measured on the Schwarzschild spacetime with Painlevé-Gullstrand coordinates.

As illustrated in the center plot, the RMSE of the velocity normalization constraint is monitored. The measured relative error remains below 4.5×10^{-9} throughout the simulation for both timelike and null geodesics.

The conserved quantities and velocity normalization for the Kerr spacetime in Kerr-Schild coordinates exhibit the expected behavior shown in theory, with maximum RMSE deviations below 10^{-7}. Again, the results for the Kerr spacetime show a higher accumulated error than those for the isotropic coordinates, but remain of the same order of accuracy as the Painlevé-Gullstrand coordinates. Overall, the measured values are consistent with the anticipated interpolation and time integration error orders, given the grid spacings (Δx, Δy, and Δz) and timestep (Δt).

6.4 Error Convergence Tests

To test the error convergence of the code, we must examine both the space interpolation error and the numerical error introduced by the Runge-Kutta integrator. Taking into account that we have been using both numerical approximation, the total error in the application should be a mixture of spatial and temporal numerical errors of fourth order.

In order to isolate the spatial error, we can just execute a single iteration of the simulation and, using known function such as the 3+1 variables for the test cases, compute the interpolation error using a L_2 norm error, for both the functions and its derivatives, for different spatial resolutions. In this way, using a linear regression of the log-log data, we can extract the slope, which should coincide with the expected values. This error, according to the theory presented in Chapter 3, should scale as fifth order for the functions and as fourth order for the derivatives.

For temporal convergence, we track the error throughout the entire simulation using the conserved quantities shown in Section 6.1. We then compute the error for multiple geodesics at the end of the simulation. That is, given a conserved quantity $C(t)$ and P particles at the

end of an N-step simulation, the error is measured using the Root Mean Squared Deviation (RMSD):

$$\Delta E = \sqrt{\frac{\sum_{i=1}^{P} \left(C(0) - C(N)\right)^2}{P}}. \tag{6.27}$$

Measuring the conserved quantities error for the same initial conditions but with different $\Delta t = \delta \Delta x$, and thus different Δx, for fixed δ, allows us to obtain the error convergence order of the entire code by, again, using a linear regression on the log-log data.

6.4.1 Interpolation Convergence Test

For the interpolation convergence test, we ran the simulation for one iteration and then computed the error between the analytical expressions and the interpolated values for known functions, such as the 3+1 formalism variables described for some of the test spacetimes. We can compute the error obtained by halving the grid resolution and then perform a log-log regression to obtain the interpolation error order.

Figure **6.4** presents the interpolation error convergence measured for the functions $\gamma_{xx}, \gamma_{xy}, \gamma_{xz}$ and the derivatives $\partial_x \alpha, \partial_y \alpha$, and $\partial_z \alpha$ in the Kerr spacetime with Kerr-Schild coordinates. The tests were run with a discretized cubic mesh with corners at $(-10, -10, -10)$ and $(10, 10, 10)$, using approximately 20,000 data-points uniformly distributed outside of the event horizon, and varying the number of cells from 32 to 256 in each direction.

The plots in the upper row shows the measured root mean squared error in the interpolation, along with their correspondent log-log regressions. The log-log regression shows error convergence orders of 4.46173, 4.82312 and 4.94976 with a Pearson coefficients $R^2 = 0.98431, 0.99879$, and 0.99999 for the γ_{xx}, γ_{xy} and γ_{xz} functions, respectively. This indicates strong correlation with the data, particularly in the latter two cases. The convergence order shows a relative error with respect to the expected fifth-order convergence of $10.77\%, 3.53\%$ and 1.01%. Overall, the results are consistent with the theoretical fifth-order accuracy of the scheme.

The lower row shows the measured root mean square deviation in the interpolation of the derivatives, along with their correspondent log-log regressions. In this case, we obtain error convergence orders of $3.91328, 4.29911$ and 4.38525 with Pearson coefficients $R^2 = 0.99319, 0.99880$ and 0.99913 for the $\partial_x \alpha, \partial_y \alpha$, and $\partial_z \alpha$, respectively, suggesting a strong correlation with the data. For this case, the relative errors with respect to the

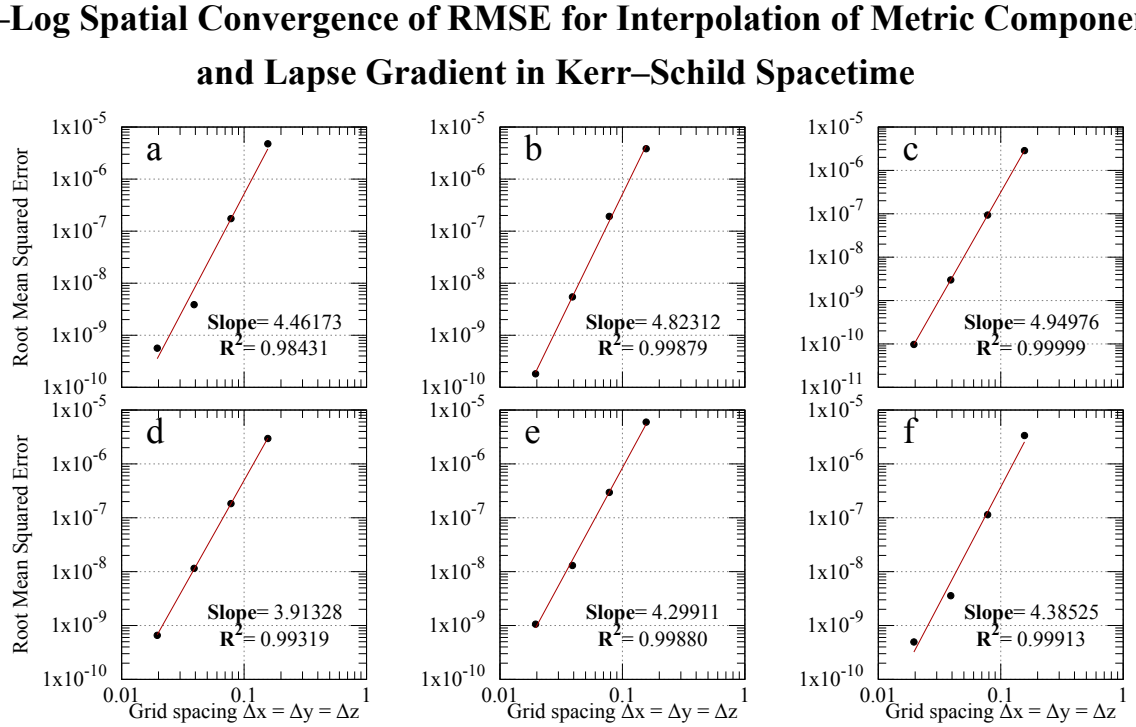

Figure 6.4: Log–log convergence of the root mean square interpolation error as a function of grid spacing $\Delta x = \Delta y = \Delta z$ for barycentric Lagrange interpolation applied to Kerr–Schild data. The top row shows the spatial metric components, the panels (a)-(c) present γ_{xx}, γ_{xy}, and γ_{xz}, while the bottom row shows the gradient of the lapse function, i.e., the panels (d)-(f) present $\partial_x\alpha, \partial_y\alpha, \partial_z\alpha$. Linear regression in log–log scale is used to estimate the observed order of convergence for each quantity. The fitted slopes indicate approximately fifth-order spatial convergence for the interpolation and approximately fourth-order convergence for the derivatives.

fourth-order behavior are $2.17\%, 7.48\%$ and 9.63%. The results are in agreement with the fourth-order accuracy of the interpolator.

Both cases are consistent with what theoretical expectations, with relative errors up to a 10.77%. The higher error shown in the interpolation of the γ_{xx} component exhibits saturation at the finest grid resolution. This could be caused by statistical errors or machine rounding errors.

6.4.2 Mixed Total Error Convergence Test

To achieve the convergence of the total error of the simulation, the case of $\Delta t = \Delta x = \Delta y = \Delta z$ can be tracked. In this particular instance, the interpolation

order and the Runge-Kutta truncation error are found to be interrelated, with the variation occurring as the number of cells in each direction is altered.

The simulation was executed for the three spacetimes using a cubic mesh with side length $20M$, where M denotes the black hole mass. The origin of the coordinate system is set at the point of mass concentration (i.e., the origin in Cartesian coordinates). The simulation was performed by varying the grid resolution (doubling the number of cells in each direction) while maintaining a fixed total simulation time. The initial number of cells was set to 64 ($\Delta x = 1.5625 \times 10^{-1}$), and the final number of cells was set to 1024 ($\Delta x = 9.765625 \times 10^{-3}$). The simulation time was maintained at $20M$ for all test cases.

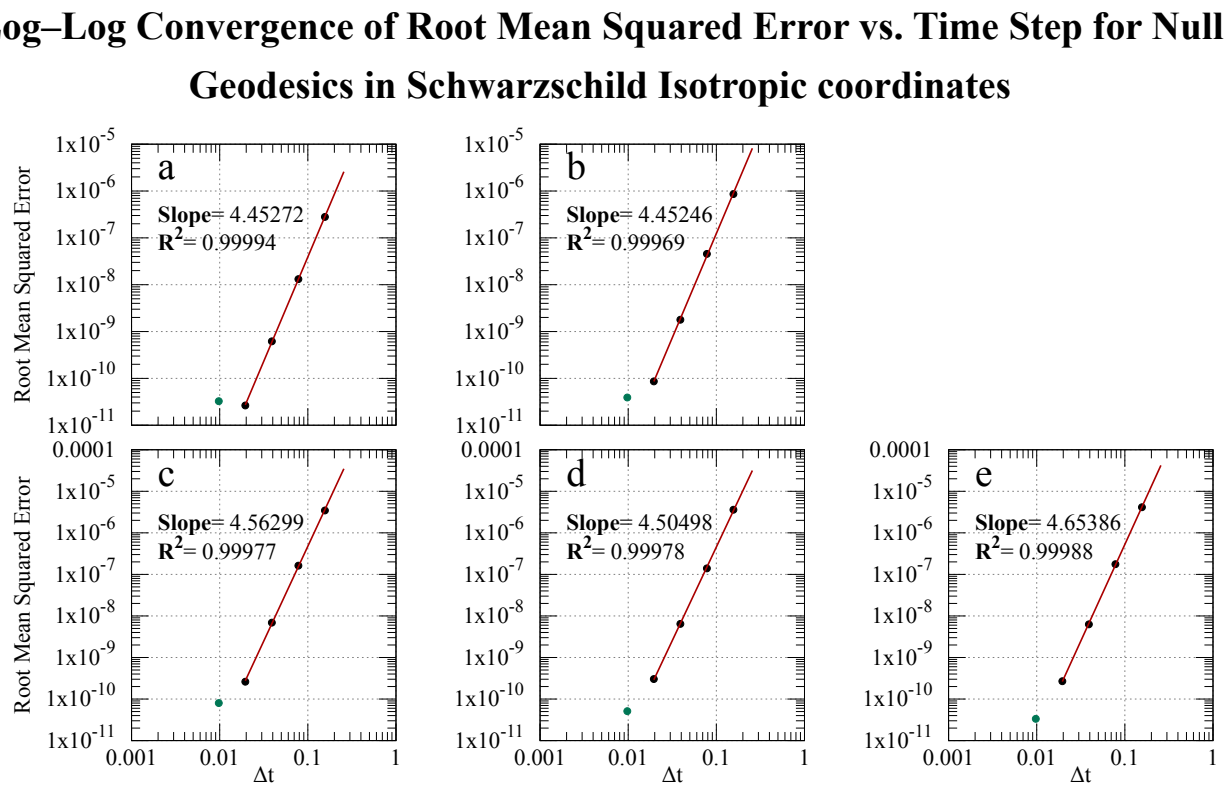

Figure 6.5: Log–log convergence of the root mean square error as a function of the time step size Δt. Linear regression yields an observed convergence order of approximately $p = 4.5$, consistent with the expected accuracy of the fourth-order Runge-Kutta scheme for Schwarzschild isotropic coordinates. The gray points corresponds to the case which we achieve a saturation, those data points were not included into the log-log regression. The panels (a)–(e) present p_t, the velocity normalization, L_x, L_y, and L_z, respectively.

As illustrated in Figures **6.5**, **6.6**, and **6.7**, the RMSE of the conserved quantities (comparing initial and final values) is shown as a function of grid resolution for Schwarzschild spacetime

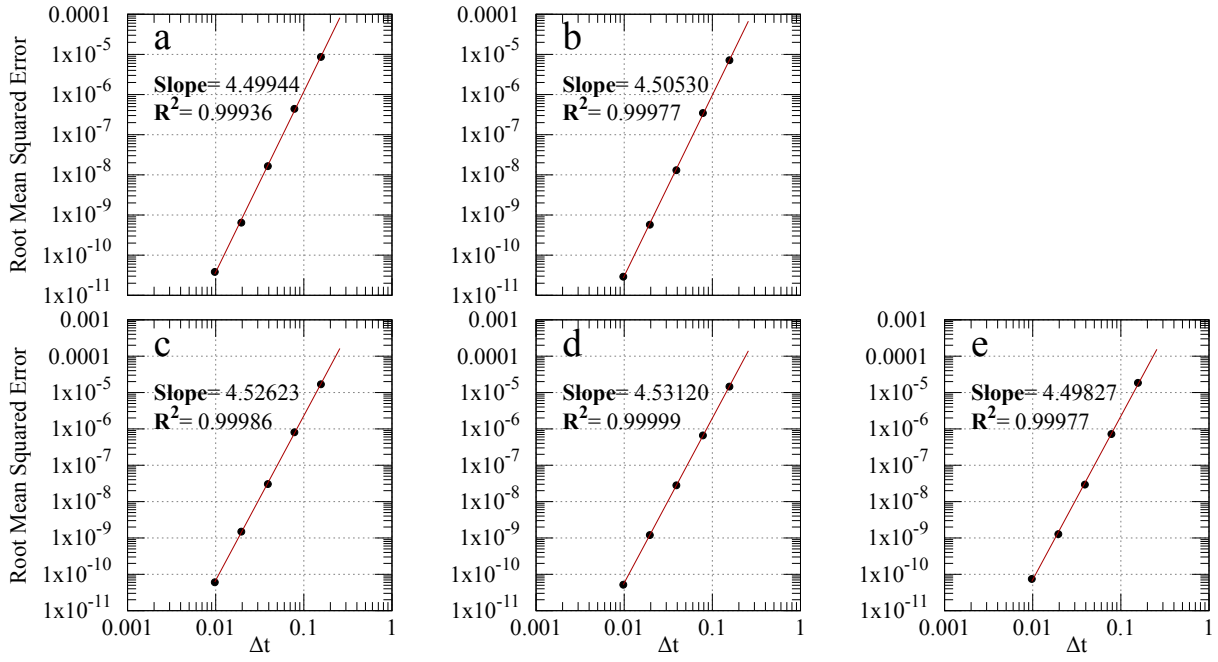

Figure 6.6: Log–log convergence of the root mean squared error as a function of the time step size Δt. The linear regression yields an observed convergence order of approximately $p = 4.5$, for the Schwarzschild Painlevé-Gullstrand coordinates. The panels (a)–(e) present p_t, the velocity normalization, L_x, L_y, and L_z, respectively.

in isotropic and Painlevé-Gullstrand coordinates, and for Kerr spacetime in Kerr-Schild coordinates, with varying spatial and temporal spacing. Each data point is associated with particles that remain valid at the conclusion of the simulation. This approach is employed to exclude potential errors arising from metric divergences in proximity to event horizons.

Figure **6.5** shows the RMSE log-log convergence as a function of the timestep Δt for Schwarzschild spacetime in isotropic coordinates. The log-log regressions show error convergence orders of 4.45272, 4.45246, 4.56299, 4.50498, and 4.65386 with Pearson coefficients $R^2 = 0.99994$, 0.99969, 0.99977, 0.99978, and 0.99988 for the conserved quantities p_t, $V_i V^i$, L_x, L_y, and L_z, respectively, showing strong linear correlation with the data. The minimum and maximum relative errors with respect to the expected fourth-order convergence are 11.31% (for the x-component of angular momentum) and 16.34% (for the p_t component of the 4-momentum), respectively, in agreement with theoretical expectations. For the p_t case, we see a saturation for the lowest value of Δt, it also happens to the other cases but is less important.

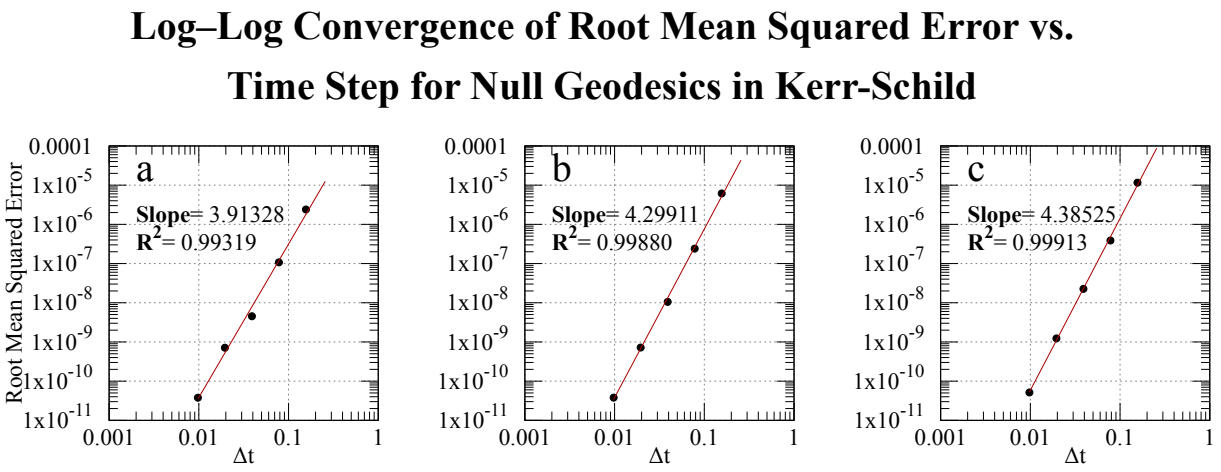

Figure 6.7: Log–log convergence of the root mean squared error as a function of the time step size Δt for null geodesics. The linear regression yields an observed convergence order of approximately $p = 4$, for the Kerr-Schild coordinates. The panels (a)–(c) present p_t, the velocity normalization, and L_z, respectively.

Figure **6.6** shows the RMSE log-log convergence as a function of the timestep Δt for Schwarzschild spacetime in Painlevé-Gullstrand coordinates. The log-log regressions shows error convergence orders of 4.49944, 4.50530, 4.52623, 4.53120, and 4.49827 with Pearson coefficients $R^2 = 0.99936$, 0.99977, 0.99986, 0.99999 and 0.99977 for the conserved quantities $p_t, V_i V^i, L_x, L_y$, and L_z, respectively, showing strong linear correlation with the data. The findings indicate a convergence order that exceeds initial expectations. While the relative discrepancy reaches up to 13.28%, it remains within acceptable limits given that the error is a composite of all approximations executed during the simulation. In this case, there is no data saturation, and the behavior better for the smallest time steps compared to the isotropic coordinates.

Finally, Figure **6.7** shows the RMSE log-log convergence as a function of the timestep Δt for the Kerr spacetime in Kerr-Schild coordinates. The log-log regressions shows error convergence orders of $3.91328, 4.29911,$ and 4.38525 with Pearson coefficients $R^2 = 0.99319, 0.99880$ and 0.99913 for the conserved quantities $p_t, V_i V^i$, and L_z. respectively, showing a strong linear correlation with the data. The minimum and maximum relative errors with respect to the expected fourth-order convergence are 2.17% (for the p_t component of the 4-momentum) and 9.63% (for the z-component of angular momentum), respectively, in agreement with theoretical expectations. This case shows that the expected results for measuring the error convergence order are correct. It also shows that the saturation we have experienced in the Schwarzschild spacetime in isotropic

coordinates is not present.

6.4.3 AMR Error Test

The main motivation for implementing Adaptive Mesh Refinement is to increase resolution in regions where non-smooth functions require greater precision. For example, this is necessary close to event horizons.

We tested the difference in error near the event horizon with and without mesh refinement. The simulation was performed using a cubic domain with sides of length $20M$ centered at the origin using the Kerr-Schild coordinate system. Two additional cubic boxes with sides of $4M$ and $2.5M$ were set for the refinement levels, also centered at the origin of the Cartesian coordinates. The coarsest level has a spatial spacing of 0.078125 in all directions, and this value was also used for the timestep. Each refinement level has a spatial spacing that is half that of the previous level. The non-refinement case uses the coarsest level throughout the simulation. The Kerr-Schild parameters are $M = 1$ and $a = 0.5$, and the timelike particles have a mass $m = 0.01$. The event horizon of the black hole under consideration is located at a radius of approximately 1.968. The zone in proximity to the event horizon is an area where particles require better resolution.

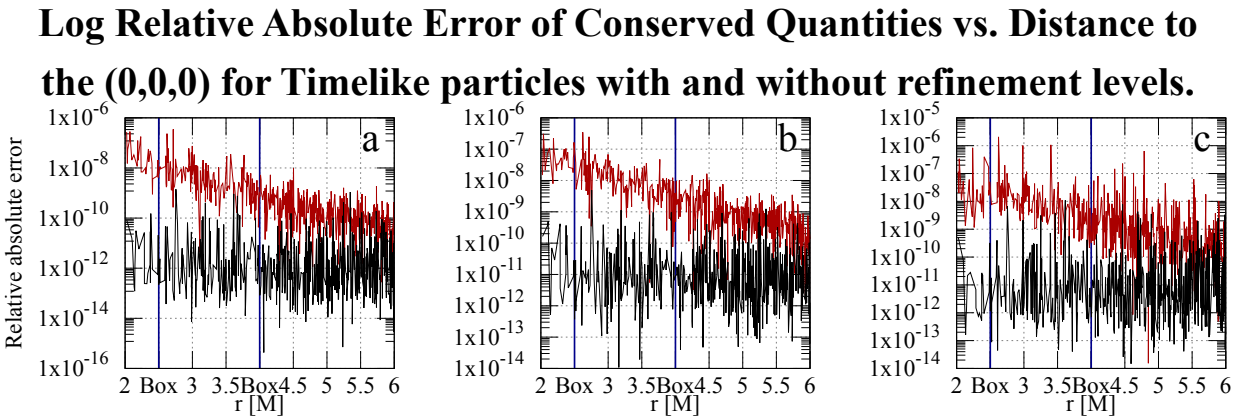

Figure 6.8: Logarithmic relative absolute error of Kerr–Schild conserved quantities for timelike geodesics as a function of radial distance. Black curves correspond to simulations with adaptive mesh refinement (AMR), and red curves to runs without refinement. Vertical red lines mark refinement level boundaries, with each level represented by a Cartesian box centered at the origin. The AMR configuration reduces conservation errors and maintains improved accuracy across refinement interfaces. The panels (a)–(c) present p_t, the velocity normalization, and L_z, respectively.

As illustrated in Figure **6.8**, the relative absolute error, defined as $\frac{|C(0)-C(10)|}{C(0)}$, is measured for the conserved quantities p_t and L_z. The figure also presents the total absolute error for the velocity normalization, denoted as $\left|1 - \frac{m^2}{E^2} - V^i V_i\right|$, as a function of distance from the black hole center. These measurements are obtained from simulations both with and without adaptive mesh refinement. To accentuate the differences, the data is plotted with a logarithmic y-axis.

The plots illustrate an increasing trend in the error as the black hole event horizon is approached. For timelike particles evolved in the simulation without AMR (blue line), error values ranging from 10^{-14} to 10^{-5} were recorded for the three quantities under observation. Conversely, the results from the AMR simulations (black line) demonstrate a more favorable behavior of the data, exhibiting a reduction in errors by up to six orders of magnitude, approaching the black hole's event horizon under certain conditions. The data indicates that the errors are sustained within the range of 10^{-8} to 10^{-14}, thereby attenuating the maximum error of the non-AMR case by three orders of magnitude.

Chapter 7

Performance Evaluation

To evaluate the performance of the code, a combination of strong and weak scaling techniques was employed using the CINECA's Leonardo cluster on the DCGP and Booster partitions. The parallel speedup for the strong scaling and the efficiency for the weak scaling tests were computed.

Furthermore, a robust scaling test for the implementation of adaptive mesh refinement was conducted. This test utilized the Grace Hopper partition of the Texas Advanced Computing Center (TACC) Vista cluster and the flex partition of the TACC Frontera cluster. Concurrently, a hydrodynamic simulation was executed alongside the particle evolution to assess the overhead introduced by particle evolution. The AMR test was conducted using Leonardo's Booster partition, but without the hydrodynamic simulation due to limitations inherent in the software.

7.1 Code Scaling

We created a simulation using a cubic box of size $200M$ with 512 cells in each direction. This box was used to check the performance of the code without adaptive mesh refinement. The simulation used timelike particles with mass $m = 0.01$ that were randomly initialized for the Kerr spacetime in Kerr-Schild coordinates. We used the parameters $M = 1$ and $a = 0.5$.

In order to determine the code's behavior in the context of smaller boxes, a simulation was conducted using a box with dimensions of 20M and 320 cells in each direction. This configuration was selected to investigate the potential impact of additional process domains being located inside the event horizon, which could lead to a deterioration of workload balance.

In order to obtain strong and weak scaling, the code was executed on CINECA's Leonardo cluster. The Leonardo cluster is divided into two distinct partitions:

- **DCGP partition** is equipped with 112 cores per node and a maximum of 16 nodes per job file.
- **Booster partition** is equipped with 32 cores per node, 4 GPUs per node, and a maximum of 32 nodes per job file. In order to ensure the effective utilization of the GPU, it is important to employ a single MPI process for each GPU in each simulation.

For both partitions, the relevant data concerning strong and weak scaling has been obtained.

7.1.1 Leonardo DCGP Partition

On the DCGP partition, the simulation was executed for 10 iterations, with each iteration involving 112 MPI processes allocated to a single computing node. In order to assess the efficacy of the simulation in terms of strong scaling, the simulation was executed with a constant number of geodesics (10^8). The number of nodes utilized was varied, and the resulting data were used to calculate the parallel speedup of the simulation. In contrast, in the weak scaling case, a constant number of geodesics is established per node, amounting to 6.25×10^6. This is accompanied by a variable number of nodes. Utilizing the collected data, the code efficiency can be calculated.

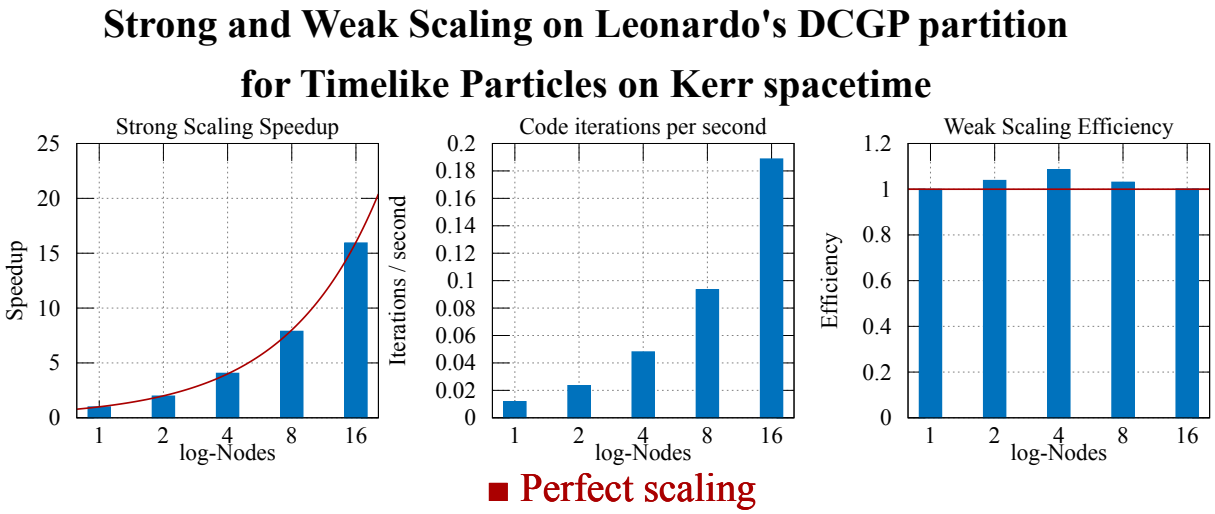

Figure 7.1: Strong and weak scaling results on the Leonardo DCGP partition for timelike particles evolved in Kerr–Schild spacetime. Strong scaling is performed with a fixed total of 10^8 particles, while weak scaling uses 6.25×10^6 particles per node. Panels show speedup (left), iteration throughput (center), and parallel efficiency (right).

Figure **7.1** shows the results for the strong scaling test (left panel), iterations per second (central panel), and weak scaling test (right panel) obtained from the simulation described previously on the DCGP partition. The iterations per second displayed in the central panel will facilitate a comparison of the GPU performance on the Booster partition.

The left panel of Figure **7.1** displays the parallel speedup achieved through the parallel execution, while keeping the number of particles and the grid resolution constant. The results indicate a parallel scaling that closely approximates ideal strong scaling, with a maximum relative error of 1.57% from the ideal speedup.

The right panel Figure **7.1** displays the parallel efficiency achieved while maintaining a constant number of particles per node and fixed grid resolution. The results indicate a parallel scaling that approximates the ideal weak scaling scenario as well, exhibiting a maximum error of around 8.56% relative to the ideal efficiency.

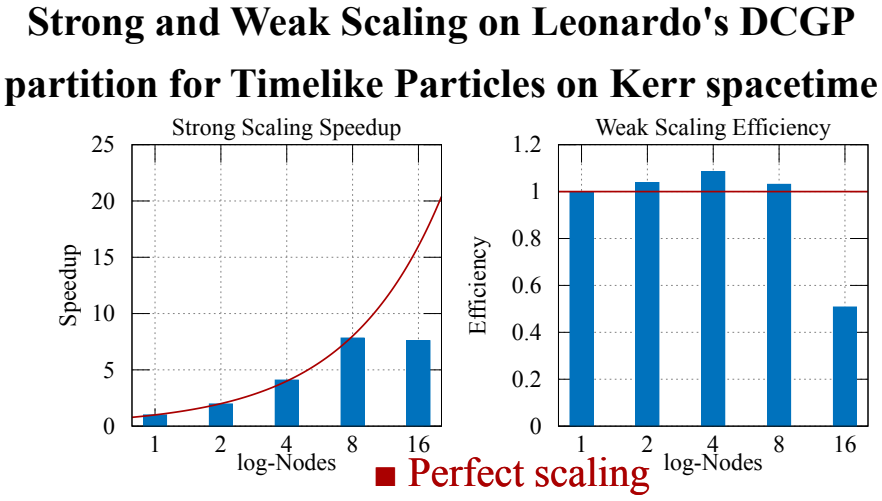

Figure 7.2: Strong and weak scaling results on the Leonardo DCGP partition for timelike particles evolved in Kerr–Schild spacetime. Strong scaling is performed with a fixed total of 10^8 particles, while weak scaling uses 6.25×10^6 particles per node. Panels show speedup (left) and parallel efficiency (right). The simulations were executed on a box of size $20M$, which affects workload balance.

Figure **7.2** demonstrates the execution of the test in a cubic box of size $20M$, which exhibits suboptimal performance in both strong and weak scaling scenarios. In the context of the speedup, optimal performance is attained with the 8-node execution. However, saturation is encountered when executing on 16 nodes, limiting the scalability of the code. A similar trend is observed in the weak scaling scenario, wherein around a 50% reduction in efficiency is observed for the 16-node configuration. The remaining cases demonstrate close proximity

to optimal efficiency.

The performance characteristics are attributable to the dimensions of the simulation box. In the case of smaller boxes, the majority of the process domains are contained within the event horizon. Consequently, many processes are unable to compute, negatively impacting the workload balance.

7.1.2 Leonardo Booster Partition

In the Booster partition, the simulation was executed 100 times. Each execution involved four MPI processes and four GPUs allocated to a single computing node. Both simulations used the same parameters as the DCGP partition execution.

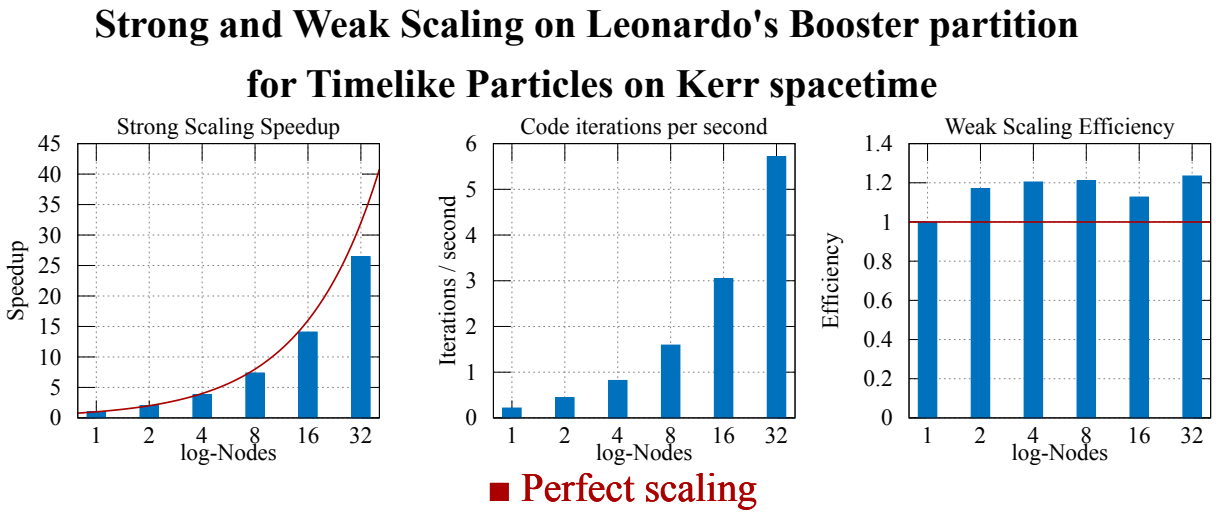

Figure 7.3: Strong and weak scaling results on the Leonardo Booster partition using the GPUs for timelike particles evolved in Kerr–Schild spacetime. Strong scaling is performed with a fixed total of 10^8 particles, while weak scaling uses 6.25×10^6 particles per node. Panels show speedup (left), iteration throughput (center), and parallel efficiency (right).

The left panel of Figure **7.3** displays the parallel speedup achieved through parallel execution, while keeping the number of particles and the grid resolution constant. Results from 1 to 4 nodes indicate parallel scaling that approximates ideal strong scaling, with a maximum deviation of 7.81% from the ideal speedup. The execution of the program with 16 and 32 nodes demonstrates a saturation of the parallel speedup, resulting in relative errors of 11.95% and 17.37% from the ideal speedup, respectively.

The right panel of Figure **7.3** displays the parallel efficiency achieved while maintaining a constant number of particles per node and constant grid resolution. The results indicate that the code performs similarly when increasing the number of nodes. An enhancement in performance by 23.47% was observed for 32 nodes in comparison with the 1-node scenario. The performance enhancement is attributable to the fact that mesh data processing dominates over geodesic evolution time. As the number of nodes increases, the mesh size per node decreases, improving cache efficiency and reducing memory access time. This results in a scaling factor that reduces the mesh size per node.

The central panels of Figures **7.1** and **7.3** show the iterations per second achieved on each partition. The results show peak iteration rates of $0.188767, \mathrm{it/s}$ and $5.71109, \mathrm{it/s}$ for the DCGP and Booster partitions, respectively, yielding an acceleration factor $\left(\frac{(it/s)_{\mathrm{Booster}}}{(it/s)_{\mathrm{DCGP}}}\right)$ of 30.25, consistent with the PFLOPS values reported in the Leonardo User Manual [29].

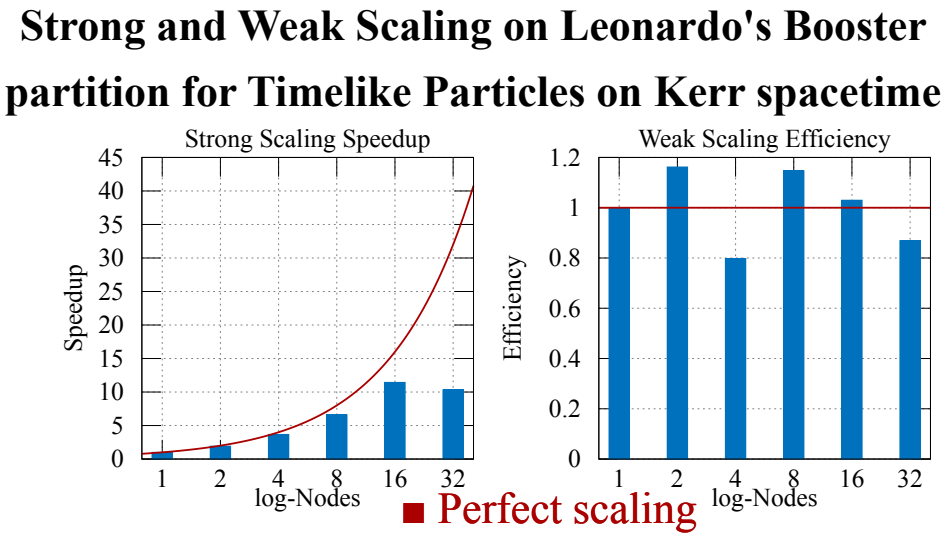

Figure 7.4: Strong and weak scaling results on the Leonardo Booster partition for timelike particles evolved in Kerr–Schild spacetime. Strong scaling is performed with a fixed total of 10^8 particles, while weak scaling uses 6.25×10^6 particles per node. Panels show speedup (left) and parallel weak scaling efficiency (right). The simulations were executed on a box of size $20M$, which affects parallel efficiency.

Figure **7.4** presents the performance of the test within a cubic domain of size $20M$, where both strong and weak scaling results indicate non-ideal behavior. Regarding the speedup, the best performance is achieved in the 16-node configuration, whereas a clear saturation appears at 32 nodes, signaling a scalability limit of the implementation. A comparable pattern emerges in the weak scaling case, with an efficiency drop of approximately 20% for the 32-node and 4-node configurations. The other configurations remain close to the ideal

efficiency.

7.2 AMR Overhead

We have conducted similar tests using the Frontera and Vista clusters. Frontera, hosted by the Texas Advanced Computing Center (TACC), provides 8,368 compute nodes based on Intel Xeon Platinum 8280 (Cascade Lake, CLX) processors, each node equipped with 56 CPU cores (2×28 cores at 2.7 GHz), 38.5 MB of L3 cache per socket, and 192 GB of DDR4 RAM. For our test cases, up to 32 nodes were employed.

Vista comprises two distinct partitions. The Grace–Grace partition is based on NVIDIA Grace CPUs, providing 144 Arm Neoverse V2 cores per node and 237 GB of LPDDR5X memory, designed for high memory bandwidth workloads. The Grace–Hopper partition integrates an NVIDIA Grace CPU with an NVIDIA H100 GPU, offering 72 CPU cores, 116 GB of system RAM, and a single H100 accelerator with 96 GB of HBM3 memory and high-bandwidth interconnect between CPU and GPU. The H100 GPU delivers substantial FP64 throughput and memory bandwidth, making this partition particularly suitable for massively parallel particle-based simulations. All tests were executed on the Grace-Hopper partition using 1 MPI process per GPU.

To test the overhead introduced by AMR, we ran a test using the AsterX GRMHD gas evolution code [11] concurrently with the geodesic integration on the TACC Vista and Frontera clusters. We ran the test on the Leonardo cluster without the gas evolution component, but with the same AMR grid and parameters as those used on Vista and Frontera, due to software incompatibilities.

The simulations were performed on a cubic computational domain $x, y, z \in [-8192, 8192]$ discretized with a base resolution of 256 cells per direction. Adaptive mesh refinement (AMR) was handled by CarpetX with a maximum of 12 refinement levels and a refinement ratio of two between consecutive levels. The refinement hierarchy was defined through three BoxInBox regions, allowing nested cubic structures around the regions of interest. Regridding was performed every 4096 iterations with an error threshold of 0.9. Time subcycling was disabled, so all refinement levels evolve with the same timestep. The simulations were executed using 4, 8, 16, and 32 compute nodes in order to analyze the scaling behavior and quantify the AMR-related overhead with 10^8 null geodesics.

The physical setup corresponds to a fixed Cartesian Minkowski spacetime (Cowling approximation), with GRMHD evolution performed using the HLLE flux scheme and PPM reconstruction, together with an ideal-gas equation of state. Conversely, on Leonardo, we

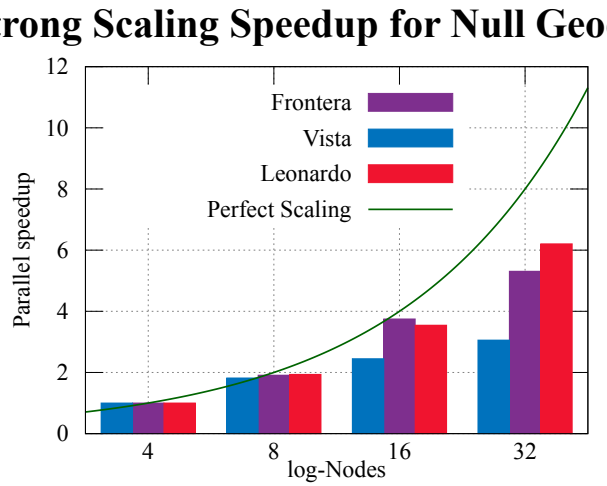

Figure 7.5: Strong scaling speedup as a function of number of nodes (log scale) on the TACC Frontera (purple), TACC Vista (blue), and CINECA Leonardo (red) clusters. The test case consists of 10^8 null geodesics. Both Frontera and Vista simulations were executed with the AsterX gas evolution module running concurrently; on Leonardo, only the particles were evolved. The green line represents ideal linear scaling. Deviation from ideal behavior reflects communication overhead and load imbalance at higher node counts.

integrated the geodesics using the Schwarzschild spacetime in isotropic coordinates without any other concurrent thorns.

Figure **7.5** presents the parallel speedup obtained for the simulation with 10^8 particles using the previously described AMR mesh. The maximum relative deviations from ideal speedup are 33.64%, 61.78%, and 22.43% for Frontera, Vista, and Leonardo, respectively. Despite these deviations, the code continued to scale with increasing node count. Overall, the scaling behavior is noticeably better on Frontera and Leonardo than on Vista.

7.2.1 Geodesic Integration Overhead

In order to assess the overhead introduced by the particle infrastructure, the number of particles was varied across a series of values: $0, 10^5, 10^6, 10^7$, and 10^8. These values were used to evolve the previous simulation on Frontera and Vista. Both the geodesic integration and the gas evolution were followed for a total of 20 iteration steps in Frontera and 256 steps in Vista.

Figures **7.6** and **7.7** present the results obtained for the Frontera and Vista clusters, respectively. These results are presented as functions of particle count and node count,

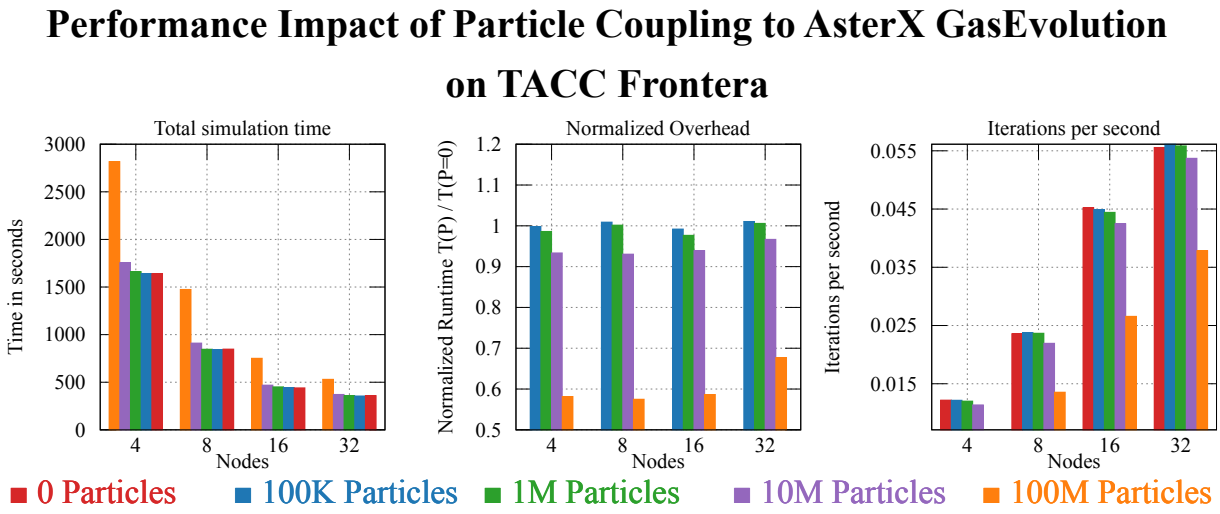

Figure 7.6: Performance impact of coupling null particle integration to the AsterX gas evolution module (3+1 formalism). Left: total simulation time as a function of node count for particle loads ranging from $P = 0$ to 10^8. Center: runtime normalized by the baseline hydrodynamic case (no particles), isolating the overhead introduced by particle evolution. Right: achieved iteration rate (iterations per second) as a function of node count and particle number. Results shown for the Frontera cluster.

showing total simulation time (left), normalized overhead (center), defined as the time without particles divided by the time with particles, and iterations per second achieved (right).

As shown in Figure **7.6**, the total simulation time is not significantly affected when evolving up to 10^7 geodesics, remaining at approximately 90% of the baseline performance (i.e., the case without geodesic integration). In contrast, when evolving 10^8 geodesics, the execution time is substantially impacted, decreasing to about 60% of the baseline performance for configurations using 4 to 16 nodes, and to roughly 70% when using 32 nodes.

Moreover, the best overall performance is obtained with 32 nodes, reaching approximately 0.055 iterations per second for simulations with up to 10^7 particles and about 0.04 iterations per second when evolving 10^8 particles.

As shown in Figure **7.7**, the total simulation time is not significantly affected when evolving up to 10^7 geodesics, remaining at approximately 90% of the baseline performance. In contrast, when evolving 10^8 geodesics, the execution time is substantially impacted, decreasing to about 70% of the baseline performance for configurations using 4 to 16 nodes, and to roughly 77% when using 32 nodes. Moreover, the best overall performance is obtained with 32 nodes, reaching approximately 0.4 iterations per second for simulations

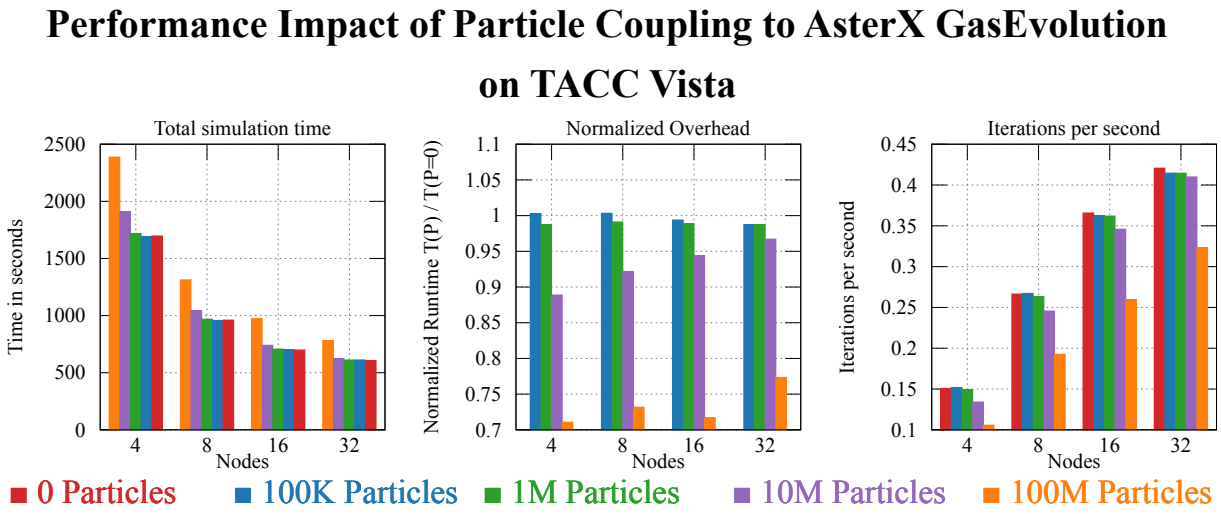

Figure 7.7: Performance impact of coupling null particle integration to the AsterX gas evolution module (3+1 formalism). Left: total simulation time as a function of node count for particle loads ranging from $P = 0$ to 10^8. Center: runtime normalized by the baseline hydrodynamic case (no particles), isolating the overhead introduced by particle evolution. Right: achieved iteration rate (iterations per second) as a function of node count and particle number. Results shown for the Vista cluster.

with up to 10^7 particles and about 0.3 iterations per second when evolving 10^8 particles. A comparison of the maximum iterations per second achieved on Vista with those on Frontera reveals an acceleration factor of $\left(\frac{(it/s)_{\text{Vista}}}{(it/s)_{\text{Frontera}}}\right) = 7.49$.

Chapter 8

Conclusions and Future Work

We have developed a new numerical integration tool for geodesics in the 3+1 formalism of general relativity, GInX. The code has been developed in a Cactus-like manner using C++17, making it compatible for use with other Einstein Toolkit thorns such as AsterX. The utilization of the C++ framework AMReX allowed us to integrate AMR functionality into the integrator, increasing the accuracy in numerically unstable regions. Furthermore, AMReX parallel management has made both CPU and GPU execution efficient and effective.

The code was implemented using the barycentric Lagrange interpolation and the fourth-order Runge-Kutta method as a time integrator. It includes additional capabilities such as velocity normalization, spacetime initialization, and output of particles data. GInX is able to integrate null and timelike geodesics in Schwarzschild and Kerr spacetimes.

GInX convergence tests have exhibited the expected results. The barycentric Lagrange interpolation showed averaged convergence orders of 4.74 and 4.19 for function and derivative interpolation, respectively, with errors ranging from 1.01% to 10.77% relative to the expected values. The fourth-order Runge-Kutta method exhibited an average convergence order of 4.44 over the test cases, with errors between 2.17% and 16.34% relative to the expected results. The results were obtained from log-log regressions with Pearson coefficients over $R^2 = 0.98$.

GInX was tested with three different cases: Schwarzschild spacetime in isotropic and Painlevé-Gullstrand coordinates, and Kerr spacetime in Kerr-Schild coordinates. The program satisfies the physical constraints related to particle's quantities conservation, with errors up to 10^{-8}, 10^{-5}, and 10^{-8} for the isotropic, Painlevé-Gullstrand, and Kerr-Schild coordinates, respectively, throughout the 100-step evolution of null and timelike particles. Figure **6.8** shows the benefits of the AMR usage on the reduction of the maximum error measured throughout the simulation. Integration with AMR close to the Kerr black hole event horizon exhibited a reduction of three orders of magnitude in error compared to the

simulation without AMR, decreasing from a maximum measured error of approximately 10^{-5} to 10^{-8}.

GInX performance was measured on three different clusters: CINECA's Leonardo, TACC Frontera, and TACC Vista. For the Leonardo cluster, we performed tests with different physical box sizes on the DCGP and Booster partitions. The larger box of size $200M$ exhibited behavior close to perfect scaling with the number of nodes, with maximum deviations of 1.57% (DCGP partition) and 17.37% (Booster partition) from ideal parallel speedup. The smaller box of size $20M$ showed reduced performance due to workload imbalance, with saturation around 16-nodes on DCGP and 32-nodes on Booster. The iterations per second achieved on the Booster partition are 30.35 higher than those obtained on the DCGP partition.

The AMR overhead tests presented in Figure **7.5** showed the scaling of the code with the number of nodes using a multi-level background. The maximum relative deviations from ideal speedup are 33.64%, 61.78%, and 22.43% for Frontera, Vista, and Leonardo, respectively. The scaling behavior is noticeably better on Frontera and Leonardo than on Vista. GInX was tested in conjunction with the gas evolution implemented by AsterX to check the code's overhead. Up to 10^7 particles, the overhead on total simulation time is less than 15% of the total time without particles, while for 10^8 particles we ovserved an overhead of approximately 30% to 40%.

Several limitations of the present work should be acknowledged. First, the code uses the fast light approximation. Consequently, the code only works for static spacetimes or snapshots of dynamic ones, limiting its potential use in most GRMHD simulations, such as radiative transport problems. The extension to dynamic spacetimes will be addressed in future work.

Second, the code uses the AMReX single-grid approach, which stores both field data and particle data in the same *BoXArray* and *DistributionMap*. AMReX allows users to employ a dual-grid approach, separating the data and enabling better workload balancing. This would address the performance issues exposed previously, as particles are often not uniformly distributed in space. This directly affects performance when building images with the software, because particles are not uniformly distributed in space.

Third, the code has not been tested using a more complex spacetimes, such as binary black holes systems. Testing on more complex spacetimes would improve confidence in the observed results while adding a new spacetime to the code's capabilities set. Testing with more complex spacetimes will be addressed in future work.

In order to study the EM emission from astrophysical sources, ray-tracing has been employed

in several studies. For instance, in the study of EM signals from SMBBHs, developing an algorithm that accurately performs ray-tracing on the photons they emit is crucial for understanding and identifying the source when a signal is detected. Effectively identifying EM observations of SMBBHs and their environments has the potential to provide critical new information about both galaxy evolution and strong-field gravity. This code serves as the foundation for an upcoming ray-tracing tool which, working together with simulations of SMBBHs and their environments, will enable further study of their EM signals.

GInX has proven to be an open-source alternative for null and timelike geodesic integration using the fast-light approximation in the 3+1 formalism of general relativity. It is designed for programs integrated with the Einstein Toolkit platform. It integrates the barycentric Lagrange interpolation and the fourth-order Runge-Kutta methods with the AMReX particle containers, including the parallel execution on both CPU and GPU architectures. It was tested for both numerical accuracy and parallel performance on top-tier clusters (Leonardo, Frontera, Vista). It provides a starting point for future implementations of post-processing analysis tools for astrophysical simulations.

Chapter 9

Bibliography

[1] F H Vincent, E Gourgoulhon, and J Novak. 3+1 geodesic equation and images in numerical spacetimes. *Classical and Quantum Gravity*, 29(24):245005, November 2012. ISSN 1361-6382. doi: 10.1088/0264-9381/29/24/245005. URL http://dx.doi.org/10.1088/0264-9381/29/24/245005.

[2] K. R. Perchenick, Ftaclas, and J. M. C. Cohen. Hot spots on neutron stars - The near-field gravitational lens. *The Astrophysical Journal*, 274:846–857, November 1983. ISSN 0004-637X. doi: 10.1086/161498. URL https://scixplorer.org/abs/1983ApJ...274..846P/abstract.

[3] S. Refsdal. The gravitational lens effect. *Monthly Notices of the Royal Astronomical Society*, 128:295, 1964. doi: 10.1093/mnras/128.4.295. URL https://scixplorer.org/abs/1964MNRAS.128..295R/abstract.

[4] R. D. Blandford and R. Narayan. Cosmological applications of gravitational lensing. 30:311–358, 1992. doi: 10.1146/annurev.astro.30.1.311. URL https://scixplorer.org/abs/1992ARA%26A..30..311B/abstract.

[5] Eduardo M. Gutiérrez, Luciano Combi, Scott C. Noble, Manuela Campanelli, Julian H. Krolik, Federico López Armengol, and Federico García. Electromagnetic Signatures from Supermassive Binary Black Holes Approaching Merger. *The Astrophysical Journal*, 928(2):137, apr 2022. doi: 10.3847/1538-4357/ac56de. URL https://doi.org/10.3847/1538-4357/ac56de.

[6] Stéphane d'Ascoli, Scott C. Noble, Dennis B. Bowen, Manuela Campanelli, Julian H. Krolik, and Vassilios Mewes. Electromagnetic Emission from Supermassive Binary Black Holes Approaching Merger. *The Astrophysical Journal*, 865(2):140, oct 2018. doi: 10.3847/1538-4357/aad8b4. URL https://doi.org/10.3847/1538-4357/aad8b4.

[7] Weiqun Zhang, Andrew Myers, Kevin Gott, Ann Almgren, and John Bell. AMReX: Block-structured adaptive mesh refinement for multiphysics applications.

The International Journal of High Performance Computing Applications, 35(6): 508–526, 2021. doi: 10.1177/10943420211022811. URL https://doi.org/10.1177/10943420211022811.

[8] Weiqun Zhang, Ann Almgren, Vince Beckner, John Bell, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves, Max P. Katz, Andrew Myers, Tan Nguyen, Andrew Nonaka, Michele Rosso, Samuel Williams, and Michael Zingale. AMReX: a framework for block-structured adaptive mesh refinement. *Journal of Open Source Software*, 4(37):1370, 2019. doi: 10.21105/joss.01370. URL https://doi.org/10.21105/joss.01370.

[9] Jhon S Moreno Trana. GInX GitHub repository. URL https://github.com/Jh0mpis/GInX/tree/main.

[10] Jhon S Moreno Triana. GInX doxygen documentation. URL https://jh0mpis.github.io/GInX/index.html.

[11] Jay V Kalinani, Liwei Ji, Lorenzo Ennoggi, Federico G Lopez Armengol, Lucas Timotheo Sanches, Bing-Jyun Tsao, Steven R Brandt, Manuela Campanelli, Riccardo Ciolfi, Bruno Giacomazzo, Roland Haas, Erik Schnetter, and Yosef Zlochower. AsterX: a new open-source GPU-accelerated GRMHD code for dynamical spacetimes. *Classical and Quantum Gravity*, 42(2):025016, December 2024. ISSN 1361-6382. doi: 10.1088/1361-6382/ad9c11. URL http://dx.doi.org/10.1088/1361-6382/ad9c11.

[12] E Gourgoulhon. *3+1 Formalism in General Relativity*. Springer Berlin, Heidelberg, 2012. ISBN 978-3-642-24524-4. doi: 10.1007/978-3-642-24525-1.

[13] Eric Poisson. *A relativist's Toolkit: The Mathematics of Black-Hole Mechanics*. Cambridge University Press, 04 2012. ISBN 978-0-521-83091-1. doi: 10.1017/CBO9780511606601. URL https://www.cambridge.org/core/books/relativists-toolkit/DA7ED68B971708A0F782257F948981E7.

[14] Andy Bohn, William Throwe, François Hébert, Katherine Henriksson, Darius Bunandar, Mark A Scheel, and Nicholas W Taylor. What does a binary black hole merger look like? *Classical and Quantum Gravity*, 32(6):065002, February 2015. ISSN 1361-6382. doi: 10.1088/0264-9381/32/6/065002. URL http://dx.doi.org/10.1088/0264-9381/32/6/065002.

[15] Alberto Valli Alfio Quarteroni. *Numerical Approximation of Partial Differential Equations*. Springer Berlin, Heidelberg, 1994. ISBN 978-3-540-85267-4. doi: 10.1007/978-3-540-85268-1. URL https://link.springer.com/book/10.1007/978-3-540-85268-1.

[16] Luciano Rezzolla and Olindo Zanotti. *Relativistic Hydrodynamics*. Oxford University Press, 09 2013. ISBN 9780198528906. doi: 10.1093/acprof:oso/9780198528906.001.0001. URL https://doi.org/10.1093/acprof:oso/9780198528906.001.0001.

[17] Gerhard Zumbusch. *Parallel Multilevel Methods*. Vieweg+Teubner Verlag Wiesbaden, 2003. ISBN 978-3-519-00451-6. doi: 10.1007/978-3-322-80063-3. URL https://link.springer.com/book/10.1007/978-3-322-80063-3.

[18] J.C. Butcher. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons Ltd., 2008. ISBN 9780470723357.

[19] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric Lagrange Interpolation. *Society for Industrial and Applied Mathematics*, 46(3):501–517, 2004. doi: 10.1137/S0036144502417715. URL https://doi.org/10.1137/S0036144502417715.

[20] Lei Wang, Robert Krasny, and Svetlana Tlupova. A Kernel-Independent Treecode Based on Barycentric Lagrange Interpolation. *Communications in Computational Physics*, 28 (4):1415–1436, June 2020. ISSN 1991-7120. doi: 10.4208/cicp.oa-2019-0177. URL http://dx.doi.org/10.4208/cicp.OA-2019-0177.

[21] Cactus. https://www.cactuscode.org/index.html.

[22] Cactus Users' Guide. https://einsteintoolkit.org/usersguide/UsersGuide.html#x1-38000C, .

[23] The Einstein Toolkit. https://www.einsteintoolkit.org/index.html.

[24] CarpetX User Manual. https://einsteintoolkit.org/thornguide/CarpetX/CarpetX/documentation.html, .

[25] The AMReX documentation. https://amrex-codes.github.io/amrex/docs_html/index.html.

[26] S. Chandrasekhar. *The Mathematical Theory of Black Holes*. International series of monographs on physics. Clarendon Press, 1998. ISBN 9780198503705. URL https://books.google.it/books?id=LBOVcrzFfhsC.

[27] Roy P. Kerr. Gravitational Field of a Spinning Mass as an Example of Algebraically Special Metrics. *Phys. Rev. Lett.*, 11:237–238, Sep 1963. doi: 10.1103/PhysRevLett.11.237. URL https://link.aps.org/doi/10.1103/PhysRevLett.11.237.

[28] Matt Visser. The Kerr spacetime: A brief introduction, 2008. URL https://arxiv.org/abs/0706.0622.

[29] CINECA User Support Team. Leonardo — CINECA HPC Documentation 1.0 documentation, 2025. URL https://docs.hpc.cineca.it/hpc/leonardo.html.