Computational Physics

ExaGRyPE: Numerical general relativity solvers based upon the hyperbolic PDEs solver engine ExaHyPE ☆

Han Zhang [a,b,*], Baojiu Li [b], Tobias Weinzierl [a], Cristian Barrera-Hinojosa [c]

[a] *Department of Computer Science, Durham University, Upper Mountjoy, Stockton Rd, Durham, DH13LE, United Kingdom*
[b] *Institute for Computational Cosmology, Department of Physics, Durham University, Lower Mountjoy, South Rd, Durham, DH13LE, United Kingdom*
[c] *Instituto de Física y Astronomía, Universidad de Valparaíso, Gran Bretaña 1111, Valparaíso, Chile*

ARTICLE INFO

ABSTRACT

ExaGRyPE describes a suite of solvers and solver ingredients for numerical relativity that are based upon ExaHyPE 2, the second generation of our Exascale Hyperbolic PDE Engine. Numerical relativity simulations are crucial in resolving astrophysical phenomena in strong gravitational fields and are fundamental in analyzing and understanding gravitational wave emissions. The presented generation of ExaGRyPE solves the Einstein field equations in the standard CCZ4 formulation under a 3+1 foliation and focuses on black hole space-times. It employs a block-structured Cartesian grid carrying a higher-order Finite Difference scheme with full support of adaptive mesh refinement (AMR), it facilitates massive parallelism combining message passing, domain decomposition and task parallelism, and it supports the injection of particles into the grid as static data probes or as moving tracers. We introduce the ExaGRyPE-specific building blocks within ExaHyPE 2, and discuss its software architecture and compute-n-feel.

For this, we formalize the creation of any specific astrophysical simulation with ExaGRyPE as a sequence of lowering operations, where a few abstract logical tasks are successively broken down into finer and finer tasks until we obtain an abstraction level which can directly be mapped onto a C++ executable. The overall program logic is fully specified via a domain-specific Python interface, we automatically map this logic onto a more detailed set of numerical tasks, subsequently lower this representation onto technical tasks that the underlying ExaHyPE engine uses to parallelize the application, before eventually the technical tasks in turn are mapped onto small task graphs including the actual astrophysical PDE term evaluations, initial conditions, boundary conditions, and so forth. These can be injected manually by the user, or users might instruct the solver on the most abstract user interface level to use out-of-the-box ExaGRyPE implementations. We end up with a rigorous separation of concerns which shields ExaGRyPE users from technical details and hence simplifies the development of novel physical models. We present the simulations and data for the gauge wave, static single black holes and rotating binary black hole systems, demonstrating that the code base is mature and usable. However, we also uncover domain-specific numerical challenges that need further study by the community in future work.

1. Introduction

More than one hundred years ago, Einstein wrote his famous equations of General Relativity [1]. Revealing the intrinsic coupling of space and time, those equations today lay the foundation of our understanding of the space-time geometry of the Universe. The Einstein equations have a rather compact mathematical formulation manifesting in ten highly non-linear tensor equations. Due to their complexity, analytical solutions are notoriously difficult to find once the initial condition of the system lacks strong symmetries, i.e. once we tackle setups of practical relevance. We thus need numerical techniques.

The field of numerical relativity has been under active development for many decades, and numerical simulations are now omnipresent when studying astrophysical phenomena involving strong gravitational fields. By recasting the Einstein field equations, numerical relativity reduces the underlying physics to a Cauchy initial value problem, which
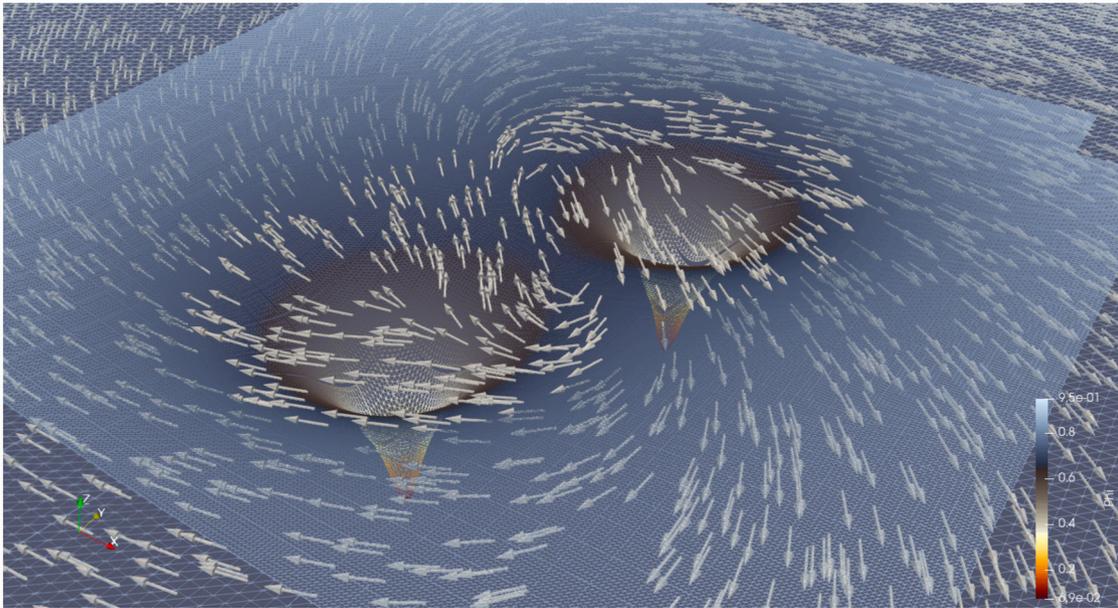
Fig. 1. Snapshot of two black holes orbiting around each other. The colored frame represents the conformal factor field ϕ, while the arrows show the distribution of the shift vector field β^i.

we can solve via standard numerical schemes: The space-time is cut into a sequence of three-dimensional hypersurfaces, labeled by a global temporal parameter. Every hypersurface represents one "time step", which is a snapshot or slice through the actual space-time [2]. Despite being a well-established approach, the number of mature and actively developed open-source codes available to the community is limited (e.g., BAM [3], GH3D2M [4], EINSTEINTOOLKIT [5], GRCHOMBO [6], SPEC [7], DENDRO-GR [8], GR-ATHENA++ [9], SENR/NRPY+ [10] and SACRA [11]. A more comprehensive list can be found in Table 1 of [12]).

The software packages share some similarities and have to tackle similar technological and numerical challenges. First, numerical relativity codes have to be able to simulate a vast span of spatial scales (spatial is to be read in the sense of numerical discretization of a space-time snapshot). They have to simulate over a huge computational domain, while some features—around black holes—require a very detailed spatial resolution (Fig. 1). At the same time, explicit time-stepping remains the state-of-the-art in the field, even though many astrophysical phenomena of interest such as the merger of black holes require us to simulate them over a long time span while explicit time-stepping yields tiny time-step sizes. Second, numerical relativity codes usually cast Einstein's equations into hyperbolic formulations under proper gauge choices, and its combination with many nonlinear terms implies that we have to pick our numerical techniques carefully to avoid numerical instabilities and numerical inaccuracies introducing long-term pollution of the results. Higher-order methods thus have established themselves as state-of-the-art, while several further ingredients, such as appropriate boundary conditions, have to be chosen carefully to fit the experimental challenge. Finally, the complex equations lead to complicated compute kernels involving potentially thousands of lines of code. The translation of PDE terms into code is not simple. As a result of these three characteristics, competitive numerical relativity codes have to combine sophisticated software technology, e.g., good (strong) scalability plus efficiency, with advanced numeric and a clear strategy how to master the arising code complexity.

In this paper, we present ExaGRyPE (The General Relativity Solver on ExaHyPE). It currently contains a suite of numerical relativity solvers and related code blocks focusing on black hole spacetimes. ExaGRyPE is based upon a from-scratch rewrite, i.e. a second generation of ExaHyPE (An Exascale Hyperbolic PDE Engine) [13], which in turn uses a complete rewrite of its underlying meshing framework Peano [14]. Besides

the core solvers, which are really just instances of ExaHyPE solvers, ExaGRyPE adds many ingredients to ExaHyPE, and it compasses domain-specific utilities. Astrophysical simulations have been conducted before with the first-generation ExaHyPE [15]. However, ExaGRyPE is a complete rewrite, brings all ingredients together to simulate systems of black holes, and it comprises previously unavailable numerical ingredients. It also implements a new, rigorous separation of concerns.

The evolving system adopted in our code is a recast version of CCZ4 (Conformal and covariant formulation of the Z4) [16], which transforms the equations into a pure first-order formulation via a set of auxiliary variables. This idea is first proposed in [15], serving as the foundation for our code. Besides this original formalism where auxiliary variables are introduced into the evolving system alongside the primary variables, ExaGRyPE comes along with a second flavor: here, auxiliary variables are derived from the primary variables in a post-processing stage after every time step. This feature enables users to implement a second-order formulation similar to many other numerical relativity codes. With lower-order and higher-order implementations of both formulations at hand, in ExaGRyPE we implement both lower-order and higher-order schemes (solvers) for both formulations, they provide the infrastructure to investigate the numerical details and modifications of the underlying physics. We even provide an ecosystem to couple various solvers, allowing us to utilize different schemes according to the qualitative features of the solution.

ExaGRyPE's solvers combine several state-of-the-art numerical building blocks realized over patch-based AMR [17], but it rigorously strips users from the opportunity to "program" their workflow. Instead, we introduce a high-level, Python-based API (Application Programming Interface) in which users specify "what" they want to compute. Our code then maps this specification onto a C++ code base, i.e. determines "how" this is realized. This strict separation-of-concern follows the philosophy introduced with Peano [14] and adopted by the first-generation ExaHyPE code, too [13]. Eventually, the abstract specification yields one relatively simple C++ base class, into which users can inject their actual PDE terms. ExaGRyPE provides pre-defined realizations for them. We refer to this mapping of a high-level specification onto a running code as *lowering* [18]. As a whole, solvers built with ExaGRyPE differ from other mainstream codes in the field: at no point can the user control the simulation flow or orchestrate calculations; at no point, do users have the opportunity to loop over resolution levels or mesh enti-

ties, and at no point can they decide how to distribute tasks between ranks or accelerators, and so forth. These decisions are made by the actual engine under the hood. Our present contribution formalizes this transition from the specification into a ready-to-run simulation as a sequence of lowering steps yielding a hierarchy of abstraction levels. Each level serves a particular purpose or represents a particular knowledge domain—numerical schemes or parallelization for example—and each level can in itself be represented as a task graph.

Our driving vision behind this software design is that physicists can puzzle their application together on a very abstract level, i.e. pick solvers, domain size, simulation timespan, minimal resolution, and so forth. We favor Python on this level. Scientific computing experts enlarge the set of available components where appropriate and necessary. Our own ExaGRyPE development work for example contributes higher-order Finite Difference (FD) schemes, novel GPU-focused task parallelism [19,20] or bespoke boundary conditions. Performance experts finally tune and optimize various code building blocks on the appropriate abstraction level (cmp. [21], e.g., for an example in the context of ExaGRyPE). We consider this strict separation of responsibilities or roles [22] and the hiding of the program complexity a blueprint for future simulation software stacks.

The remainder of this paper is organized as follows: Section 2 introduces the physics scenario we aim to simulate, served as an overview of the domain challenge driving the development of ExaGRyPE. In Section 3, we describe the technical ingredients that are newly developed for ExaGRyPE and are essential for achieving stable and accurate simulations in numerical relativity. We continue with an in-depth discussion of the lowering of abstraction levels and highlight how they enable us to deliver fast and efficient code, before we summarize the overall code usage, i.e. the user's look-n-feel (Section 4). Numerical results (Section 5) demonstrate the validity of the approach and allow us to close the paper with a summary of insights and identified future work and challenges (Section 6). An extensive appendix provides reproducibility information, details regarding the underlying physics and implementation remarks.

2. Problem statement

The space-time evolution equations used in ExaGRyPE stem from the Z4 formulation of the Einstein field equations, where an additional dynamic field Z_a and a corresponding damping term are added to the system to enhance its stability (see Appendix B for the convention) [23, 24]:

$$R_{ab} - \frac{1}{2} g_{ab} R + \nabla_a Z_b + \nabla_b Z_a - g_{ab} \nabla^c Z_c$$
$$- \kappa_1 [n_a Z_b + n_b Z_a + \kappa_2 g_{ab} n_c Z^c] = 8\pi T_{ab}. \tag{1}$$

To solve these equations, our code uses the standard ADM (Arnowitt–Deser–Misner) 3+1 space-time foliation [25]. It cuts the space-time into a sequence of three-dimensional space-like hypersurfaces labeled by a global scalar field $t(x^a) = $ const. The value of this scalar field is later treated as the time parameter t in the 3+1 simulation. The induced spatial metric on the hypersurfaces is denoted by γ_{ab}, and it is linked to the space-time line element via

$$ds^2 = -\alpha^2 dt^2 + \gamma_{ij} \left(dx^i + \beta^i dt \right) \left(dx^j + \beta^j dt \right), \tag{2}$$

where the γ_{ij} are the spatial parts of γ_{ab}. α and β^i are respectively the lapse function and shift vector, representing the four degrees of freedom of coordinates during the evolution (Fig. 2).

One can construct simulations with a formulation only including γ_{ij}, the spatial component K_{ij} of the extrinsic curvature $K_{ab} := -\frac{1}{2} \mathcal{L}_{\mathbf{n}} \gamma_{ab}$, α, β^i and the Z^a field, as they already form a complete evolving system. We further follow [16] and introduce a conformal factor ϕ and a conformal metric $\tilde{\gamma}_{ij}$, such that

$$\tilde{\gamma}_{ij} := \phi^2 \gamma_{ij}, \quad \phi = [\det(\gamma_{ij})]^{-1/6}, \tag{3}$$
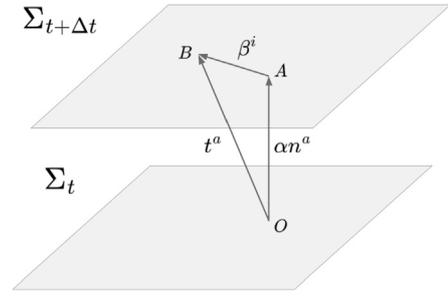
Fig. 2. Illustration of the lapse function and shift vector. Given two neighboring hypersurfaces labeled as Σ_t and $\Sigma_t + \Delta t$, the four-dimensional time vector OB can be decomposed into OA and AB, which yield the proper time α passed for a normal observer at O in the coordinate time Δt, and the coordinate shift between the two hypersurfaces. B and O have the same spatial coordinates while the spatial coordinates of A are different by $-\beta^i$.

and further define $\tilde{A}_{ij}$, K, Θ, $\hat{\Gamma}^i$ following a similar approach in the original formulation. Further details can be found in Appendix B. We note that this formulation is just one of several candidates for a strongly hyperbolic evolution system. Other options include the generalized-harmonic formalism [26,27], the BSSN formalism [28,29], and other variants within the "Z4 family", such as the original Z4 [30] and Z4c [31]. For ExaGRyPE, we have chosen to utilize CCZ4 to achieve better control over singularities and gauge conditions in black hole spacetimes.

2.1. Gauge conditions

Our four degrees of freedom are determined by the evolution equations of the lapse field α and shift vector β^i, which are called gauge conditions. ExaGRyPE allows users to define their bespoke gauge conditions, but common choices for the evolving system above are available off-the-shelf.

The default evolution equation of α, which is also known as the *time slicing* condition [32], is

$$(\partial_t - \beta^i \partial_i)\alpha = -\alpha^2 g(\alpha)K, \tag{4}$$

where $g(\alpha)$ is a positive scalar function depending only on the lapse. In simulations of black hole spacetimes, a popular choice of this function is $g(\alpha) = 2/\alpha$, as it has a good property of avoiding singularities. The resulting gauge equation is the *1+log* condition given as

$$\partial_t \alpha = \beta^k A_k - \alpha^2 g(\alpha) \left(K - K_0 - 2c\Theta \right) \tag{5}$$

in our notations. The source term gets modified accordingly as we implement the CCZ4 formulation [16].

The default gauge equation we adopt for the shift vector β^i in ExaGRyPE is the *gamma-driver* condition [33]:

$$\partial_t \beta^i = \beta^k B_k^i + f b^i, \tag{6}$$

$$\partial_t b^i - \beta^k \partial_k b^i = \partial_t \hat{\Gamma}^i - \beta^k \partial_k \hat{\Gamma}^i - \eta b^i. \tag{7}$$

Here, b^i is a helper vector field that enhances the hyperbolicity of the system and also appears in the evolving system, cf. Eqs. (8) and (10). This gauge is also called *shifting shift*. One can remove all advection terms above to switch to the *no-shifting shift* variant, which is also available in ExaGRyPE.

2.2. First- and second-order formulations

Collecting all derived quantities and the helper variables b^i from the gauge condition yields an evolving system over 24 variables

$$\vec{Q}(t) = \left(\tilde{\gamma}_{ij}, \alpha, \beta^i, \phi, \tilde{A}_{ij}, K, \Theta, \hat{\Gamma}^i, b^i \right)(t), \tag{8}$$

with a first-order time-derivative and second-order spatial derivatives defined in the hypersurface. Let

$$A_i := \partial_i \alpha, \ B_k^i := \partial_k \beta^i, \ D_{kij} := \frac{1}{2} \partial_k \tilde{\gamma}_{ij}, \ P_i := \partial_i \phi \qquad (9)$$

be auxiliary variables. Adding them to the system and treating them as further evolving variables yields a system of 58 independent variables with

$$\vec{Q}(t) = \left(\tilde{\gamma}_{ij}, \alpha, \beta^i, \phi, \tilde{A}_{ij}, K, \Theta, \hat{\Gamma}^i, b^i, A_k, B_k^i, D_{kij}, P_k \right). \qquad (10)$$

This system is first-order in both space and time. Both formulations can be phrased over 58 quantities. The first-order formulation (FO) from Eq. (10) evolves all quantities. The second-order formulation (SO) Eq. (8) evolves only 24 out of 58 entries, then computes (or "reconstructs") Eq. (9) from the new solution in a post-processing step, and continues with the next time step which is fed by both the primary and the reconstructed auxiliary quantities.

As a result, both the second-order and first-order formulations can be schematically written in a first-order hyperbolic formulation of partial differential equations (PDEs):

$$\partial_t \vec{Q} + \nabla_i F_i(\vec{Q}) + B_i(\vec{Q}) \nabla_i \vec{Q} = S(\vec{Q}), \qquad (11)$$

with $\vec{Q} : \mathbb{R}^{3+1} \mapsto \mathbb{R}^n$, $n = 24$ in SO or 58 in FO. We can use the same numerical building blocks subject to an additional reconstruction step for Eq. (9) for both formulations. In Eq. (11), $\vec{Q}$ represents the array of 58 evolving variables. $F_i(\vec{Q})$, $B_i(\vec{Q})$ and $S(\vec{Q})$ are the flux, non-conservative product (NCP) and source term respectively. In the implementation, we let the NCP absorb the flux term and hence have $\nabla_i F_i(\vec{Q}) = 0$ (cmp. Appendix B for details on the numerics and physics).

2.3. Initial conditions

For simple test scenarios where an analytical solution exists, ExaGRyPE can be initialized by directly setting every evolving variable to a hard-coded initial value. We use this approach for the gauge wave setup, for example.

For systems involving black holes, we adopt the *Bowen-York solution* [34] to initialize the scenario. It assumes an initial space-time hypersurface with maximal slicing $K = 0$ and conformal flatness $\bar{\gamma}_{ij} = \eta_{ij}$. The curvature on this hypersurface can be calculated analytically by the momentum P^i and spins S_i of the black holes. The initial metric of the black hole system is then derived from the *moving puncture approach* [35], where a second-order elliptic equation is solved to obtain the conformal factor numerically. The current version of ExaGRyPE links against the TwoPunctures module from the numerical library EINSTEINTOOLKIT [5], using the numerical approach introduced in [36].

2.4. Gravitational wave extraction

ExaGRyPE extracts the gravitational wave signal from the evolving system using the *Newman-Penrose approach* [37]: we calculate the complex scalar field ψ_4 as

$$\psi_4 = {}^{(4)} C_{abcd} k^a \bar{m}^b k^c \bar{m}^d. \qquad (12)$$

In this expression, ${}^{(4)} C_{abcd}$ is the four-dimensional Weyl tensor, and k^a and $\bar{m}^a$ are two members of a null tetrad (Appendix C). ψ_4 can be further decomposed into a superposition of modes with the base $s = -2$ spin-weighted spherical harmonics ${}_{-2} Y_{lm}$:

$$\psi_4(t, r, \theta, \phi) = \sum_{l=2}^{\infty} \sum_{m=-l}^{l} \psi_4^{lm}(t, r)_{-2} Y_{lm}(\theta, \phi), \qquad (13)$$

where the coefficients (the mode strength) are found by the inner product via spherical integrals

$$\psi_4^{lm} = \int_0^\pi d\theta \int_0^{2\pi} {}_{-2} Y_{lm}^* \psi_4 \sin \theta \, d\phi. \qquad (14)$$

In ExaGRyPE, the spherical integration is done by the spherical t-design scheme [38], where we interpolate the values of the integrated function from the closest grid points onto a set of N sample points on the sphere of interest. It is shown that the average of those sample values is also the average of the integrated function itself, i.e.

$$\int_0^\pi d\theta \int_0^{2\pi} {}_{-2} Y_{lm}^* \psi_4 \sin \theta \, d\phi = \frac{4\pi}{N} \sum_0^{N-1} {}_{-2} Y_{lm}^* \psi_4. \qquad (15)$$

The method is most effective when the function of interest has a degree of freedom that is close to or lower than t (i.e., it can be roughly described by a polynomial up to the degree of t). N needs to be increased if we want higher accuracy by raising t. In the current version of ExaGRyPE, the default set of sample points is $t = 43$ with $N = 948$ from [39] for a balance of accuracy and performance. In practice, sets of different orders can be selected in the code, based on the accuracy requirements of individual simulations.

3. Simulation building blocks

ExaGRyPE is based upon a rewrite of ExaHyPE. To the user, an ExaGRyPE solver is a Python script which creates an ExaHyPE solver and uses bespoke ingredients from ExaHyPE. All of these building blocks designed for ExaGRyPE have been integrated into the ExaHyPE software base. They are hence available to other solvers from other application domains, too. However, they uniquely map the requirements arising from astrophysical challenges onto code building blocks (Fig. 3).

3.1. Spatial discretization

As ExaHyPE solvers, ExaGRyPE relies on a block-structured Cartesian mesh [17] constructed through a spacetree formalism [40]: the whole (three-dimensional) domain is embedded into a single cube and split into three equal parts along each coordinate axis. This yields $3^3 = 27$ smaller cubes. We continue recursively, i.e., decide for each cube whether to refine it into 27 subcubes again. Our code base provides a plug-in point for users to guide the refinement. The process yields an adaptively refined Cartesian grid of cubes.

Every cube in this grid hosts a small Cartesian mesh (*patch*). Every patch consists of $p \times p \times p$ mesh elements (again small cubes), and each element holds either a piecewise constant solution of the evolving system, i.e. defines one Finite Volume, or we read its center as a sample point of the solution in a Finite Difference sense. K additional layers of elements are logically attached to the faces of the patches (Fig. 4). Those extra layers are called *halos*. They provide data from neighboring patches which are needed to update the solution in the current patch, thus allowing the latter to advance one step in time without any further data input. The patch carrying the actual evolving solution is hence the minimal atomic unit of computation and represents the finest discretization level.

Technically, faces hold copies of the closest volumes from both adjacent patches. They hold data from $2K \times p \times p$ mesh elements. Prior to the update of a cell, we glue the respective half of the face data and patch data together to obtain our logic patch-plus-halo input data. After the update, we write the new relevant data back into the $2d$ (d is the number of space dimensions) adjacent faces of a patch.

Each solver picks its K such that it can update its cell elements independently of all neighbor patches. For the traditional 3-point stencil, $K = 1$ is sufficient.

3.2. Numerical solvers

The update of a patch is performed by a *compute kernel*. ExaHyPE's baseline offers a generic Finite Volume (FV) kernel employing a Rusanov Riemann solver with explicit time stepping, implementations of Runge-Kutta Discontinuous Galerkin (RKDG) solvers, and ADER-DG
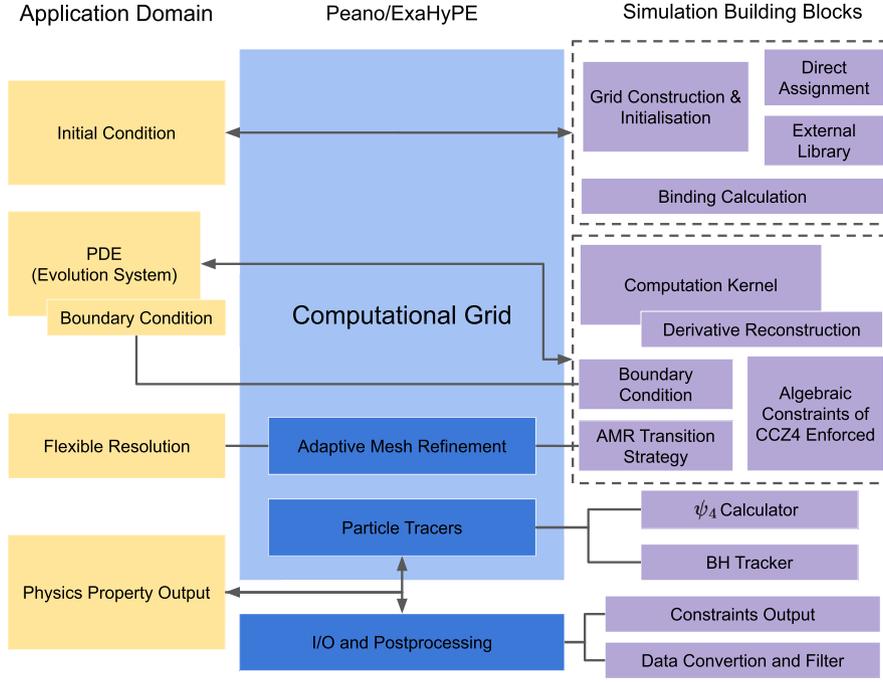
Fig. 3. The overview of the code ingredients of ExaGRyPE, showing how the physics problem (application domain) is translated into the simulation code (building blocks), bridged by the code base Peano and ExaHyPE: the targeted physics system can be seen as a Cauchy initial value problem with a set of PDEs, an initial condition and a boundary condition. They are deployed on a computational grid following certain spatial discretization schemes and are handled by different code modules. Our code supports adaptive mesh refinement to achieve flexible resolution. It also provides output of standard snapshot I/O with filtering and particle tracer features, the latter is utilized as data probes.
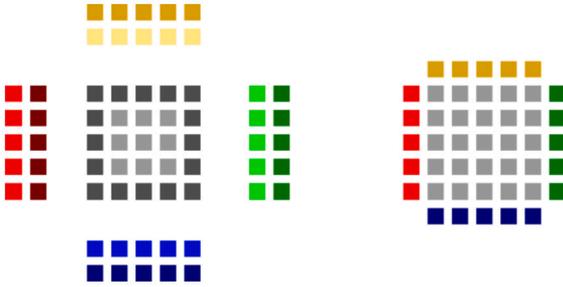
Fig. 4. Two-dimensional, schematic illustration of patch and face data arrangements in ExaHyPE 2: A two-dimensional patch with 5×5 mesh elements (volumes) and a halo of width $K = 1$ are used. Left: The faces host the halo data of both adjacent patches. Right: Prior to the kernel invocation, we merge the cell patch data and the data from the $2d$ adjacent faces into one $(5 + 2) \times (5 + 2)$ patch augmented with its halo.

(arbitrary-high-order-method-using-derivatives DG). For ExaGRyPE, we add a fourth-order Finite Differences (FD4) scheme that can be combined with Runge-Kutta (RK) time stepping to fourth-order, too. It mirrors state-of-the-art solvers from the field (cmp. GRCHOMBO). We use exclusively FD4 in ExaGRyPE, while FV is also used in some benchmark and test implementations, as well as other applications based on ExaHyPE (e.g. [41]).

Finite Volumes (FV) Our FV solver is a straightforward implementation of a generic Rusanov scheme in an explicit Euler time integrator: each volume is updated according to the flux through its six adjacent volume interfaces. Let each volume host a constant function of $Q \in \mathbb{R}^n$. The time stepping scheme of FV from Eq. (11) can be written as

$$\vec{Q}(t + \delta_t) = \vec{Q}(t) + \delta t S(\vec{Q}) + \sum_{i \in d} \text{Flux}_i^{\pm}(\vec{Q})\Big|_{\partial v}, \quad (16)$$

with the flux from the corresponding two faces in i dimension:

$$\text{Flux}_i^- = \frac{\delta t}{\delta x}\left[-\frac{1}{2}\left(\vec{Q} - \vec{Q}^-\right) B_i\left(\frac{1}{2}(\vec{Q}^+ + \vec{Q}^-)\right)\right. \quad (17)$$

$$\left. - \eta \max\left(\lambda_{\max}(\vec{Q}), \lambda_{\max}(\vec{Q}^-)\right)\left(\vec{Q} - \vec{Q}^-\right)\right],$$

$$\text{Flux}_i^+ = \frac{\delta t}{\delta x}\left[-\frac{1}{2}\left(\vec{Q}^+ - \vec{Q}^-\right) B_i\left(\frac{1}{2}(\vec{Q}^+ + \vec{Q}^-)\right)\right. \quad (18)$$

$$\left. + \eta \max\left(\lambda_{\max}(\vec{Q}^+), \lambda_{\max}(\vec{Q})\right)\left(\vec{Q}^+ - \vec{Q}^-\right)\right],$$

where B_i is the NCP term along the ith dimension, $\vec{Q}^+$ and $\vec{Q}^-$ are the volume solutions left and right of the considered volume, δx is the volume size, and η is a problem-specific constant with a default value of 0.5. We have used $\partial_t \vec{Q} = \left(\vec{Q}(t + \delta t) - \vec{Q}(t)\right)/\delta t$ as the Euler time integration scheme. This yields a 7-point stencil coupling each volume to its face-connected neighbors: $K = 1$ is sufficient to realize this scheme, and we need the ExaGRyPE specification to provide a function that evaluates the non-conservative product B_i for a given input, as well as one that returns the maximum eigenvalue $\lambda_{\max}(\vec{Q})$ along a coordinate axis for the flux calculations. For the current CCZ4 equations, ExaGRyPE follows the conservative upper estimate for this eigenvalue from the work [15].

Finite Differences (FD4) To implement the fourth-order finite difference solver in ExaHyPE, ExaGRyPE samples the solution in the center of the mesh elements. Therefore, the FD4 scheme can use the same storage scheme as the FV scheme besides a larger ($K = 2$) halo.

On this grid over sample points, we use central finite differences with a "tensor product" approach. For $K = 1$, we could construct a seven-point stencil by cutting the Taylor expansion's second-order term. In ExaGRyPE, we adopt the fourth-order accuracy scheme which requires $K \geq 2$ (we use $K = 3$ in the practice of the code for KO dissipation, see below). Its time-stepping scheme is

$$\vec{Q}(t + \delta t) = \vec{Q}(t) + \delta t S(\vec{Q}) - \tag{19}$$
$$- \sum_{i \in d} B_i \frac{\delta t}{12 \delta x_i} \left[8 \left(\vec{Q}^{i+} - \vec{Q}^{i-} \right) - \left(\vec{Q}^{i++} - \vec{Q}^{i--} \right) \right],$$

where the $i+$ and $i-$ upper indices of quantities indicate the relative position of the indexed quantities with the current grid point along i direction, and the number of signs specifies the distance. Again we have used $\partial_t \vec{Q} = \left(\vec{Q}(t + \delta t) - \vec{Q}(t) \right) / \delta t$. As all the approximation happens within the truncation of the Taylor expansion, it is sufficient for the user code to provide an implementation of B_i to the compute kernel. As our default kernel of ExaGRyPE, readers are referred to Appendix D for more details on the FD4 solver.

The fourth-order differences can be combined with an arbitrary-order time step integrator. ExaGRyPE indeed provides Runge-Kutta schemes up to order four, where we would naturally hit the Butcher barrier. However, by default, we use a first-order Runge-Kutta scheme, i.e. an explicit Euler as higher-order time integration yield to be non-economical given its extra computational cost, see Section 5.1.

Time step size calculation The admissible time step size of all explicit schemes is subject to the Courant–Friedrichs–Lewy (CFL) condition/constraint with

$$\Delta t < C \frac{|\delta x|}{(2R - 1) \lambda_{\max}}, \tag{20}$$

where $C < 1$ is a user-specified safety parameter that may vary in different simulations and R is the order of the Runge-Kutta scheme ($R = 1$ for Euler). Our scheme employs a global time-stepping scheme and thus uses the smallest global volume length $|\delta x|$ to guide Eq. (20). For all present experiments, it remains invariant over time as we fix the finest resolution (static refinement) in our simulations which naturally clusters around the black hole.

The time step size calculation again requires the solver to evaluate $\lambda_{\max}$ over each patch after it has been updated and we continue to use the same evaluation as we introduced in the Finite Volume solver above. This quantity encodes the fastest propagation speed of the quantities in our system. As it changes over time we have to restrict it globally and adopt the time step size accordingly.

Kreiss-Oliger damping Astrophysical solutions with explicit time stepping are notoriously vulnerable to numerical instabilities. It therefore is helpful to involve a penalty term in the evolution equation, as it smooths out solutions affected by numerical inaccuracies in certain regions of the domain, such as those near physical or AMR boundaries.

This smoothing is often done using the so-called Kreiss-Oliger (KO) dissipation [42], which traditionally evaluates a higher-order term (a "fourth-order curvature" for example) for second-order PDEs. Its order needs to exceed that of the kernel updating scheme to avoid introducing larger numerical errors in smoothing. As we work with the fourth-order accuracy in FD4, ExaGRyPE employs the KO term with $N = 3$ (i.e., the term use value of cells up to three points away) in the code and makes this feed into our evaluation equation (19) with a user-defined parameter to suppress oscillations (Appendix D). The KO term can be calibrated or switched off upon demand.

Explicit gradient reconstruction In the first-order formulation, all 58 variables of Eq. (10) are updated by the numerical scheme on a patch (Algorithm 1). In the second-order formulation, however, only the primary variables Eq. (8) are evolved. After that, the remaining auxiliary variables Eq. (9) are calculated from those primary variables. We update the solution, compute the auxiliary variables, i.e., their gradients, in a postprocessing step, and feed them into the subsequent time step again. With this scheme, we can realize a second-order formulation even though all PDEs are written down as first-order (Algorithm 2). For the auxiliary variables, we use a central fourth-order finite difference scheme again (D.1).

3.3. Domain decomposition

ExaHyPE splits the computational domain spanned by the spacetree along the Peano space-filling curve (SFC) [14]. The embedded patches are atomic, i.e. never split. Segments along the SFC then can be deployed among ranks and threads, i.e., we facilitate domain decomposition between ranks and on ranks. It is a non-overlapping domain decomposition.

Non-overlapping here is to be read logically: Each volume or degree of freedom within any patch has a unique owner, i.e., a unique thread on a unique rank that is responsible for updating it. The patch data replicated within the faces establish an effective overlap of $2K$.

The copy from patch data into faces happens after every time step. We use this copied data in the neighboring patch in the subsequent mesh sweep. Therefore, no direct, synchronous data exchange of face data is required. Instead, we can send out data in one mesh traversal and receive it prior to the subsequent one. With this non-blocking data transfer, our code allows for some overlap of communication and computations.

3.4. Task parallelism and GPU offloading

Patch updates in ExaHyPE are atomic units. Patches are not subdivided further. Therefore, ExaHyPE models them as tasks, and Peano provides one abstract interface which allows us to spawn them into either Intel's TBB, OpenMP, or plain C++ tasks. We also offer an experimental SYCL back-end.

The task creation patterns realize the paradigm of enclave tasking [43]: tasks that do not feed into MPI or are adjacent to adaptive mesh resolution transitions are spawned into our tasking backend of choice. ExaHyPE refers to them as enclave tasks. Other tasks are executed directly with high priority. Once they terminate, we trigger all MPI data exchange and AMR mesh inter-grid transfer operators, while the tasking back-end now handles the enclave tasks.

This principle can be used to offload batches of enclave tasks in one rush to an accelerator: we collect the enclave tasks without any side effects, i.e., those that do not alter the global solver state, in a separate queue. Once the number of buffered tasks in this queue exceeds a prescribed threshold, we bundle them into one meta task and ship them off to the GPU [20,21]. The remaining enclave tasks are handed over to our task back-end of choice. For the GPU offloading, we support OpenMP, SYCL and C++ offloading.

3.5. Boundary treatment

Boundary conditions in ExaHyPE 2 are implemented through the halo layers of faces that are adjacent to the global domain boundary. Such faces hold elements $\vec{Q}_{\text{out}}$ and $\vec{Q}_{\text{in}}$. After each mesh sweep or the initialization step, the array $\vec{Q}_{\text{in}}$ holds a valid copy of the data within the adjacent patch. To facilitate an update of this patch, we have to manually set the $\vec{Q}_{\text{out}}$, as there is no adjacent patch here, and we can use this "reconstruction" step to impose appropriate boundary conditions.

To implement a homogeneous Neumann boundary condition, we simply set $\vec{Q}_{\text{out}} = \vec{Q}_{\text{in}}$ for example. For solvers that host multiple halo layers for their time-stepping scheme, i.e. $K \geq 2$, we apply the initialization scheme to specify the values in the ghost volumes layer by layer: we move from the element layer closest to the domain boundary outwards (Fig. 5).

While ExaHyPE 2 ships a variety of pre-defined boundary treatments, one particular type of boundary condition required by ExaGRyPE is the Sommerfeld (radiative) boundary condition [33] to mitigate the effect of open-domain boundaries. It eliminates spurious modes arising from homogeneous Newmann boundary conditions, by effectively suppressing wave-like reflection and enforces that the wave at the boundary only travels out of the domain. The mathematical formulation of the Sommerfeld boundary condition reads as

---

Algorithm 1 Time stepping scheme (explicit Euler) for the fourth-order finite difference solver of ExaGRyPE in the first-order formulation.

---

for each patch in Domain **do**
    for each volume in patch **do**
        compute source term $S(\vec{Q})$
        $RHS_{\vec{Q}} \leftarrow \Delta t \cdot S(\vec{Q})$
        for $i = x, y, z$ **do**   ▷ Run over all elements in the patch
            compute fourth-order Finite Difference $(\Delta\vec{Q})_i$ ▷ Eq. (D.1)
            compute non-conservative product $B_i(\vec{Q})(\Delta\vec{Q})_i$
            compute KO term $(KO_{\vec{Q}})_i$ ▷ Eq. (D.6)
        end for
        $RHS_{\vec{Q}} \leftarrow RHS_{\vec{Q}} - \sum_i \Delta t \cdot B_i(\vec{Q})(\Delta\vec{Q})_i$
        $RHS_{\vec{Q}} \leftarrow RHS_{\vec{Q}} + \sum_i \Delta t \cdot (KO_{\vec{Q}})_i$
        $\vec{Q} \leftarrow \vec{Q} + RHS_{\vec{Q}}$
        $t \leftarrow t + \Delta t$
        compute λ_{max} to inform next time step
    end for
end for

Algorithm 2 Time-stepping scheme with an explicit Euler for the fourth-order finite difference solver of ExaGRyPE in the second-order formulation.

---

for each patch in Domain **do**
    for each volume in patch **do**
        compute Source $S(\vec{Q}_{pri})$
        $RHS_{\vec{Q}_{pri}} \leftarrow \Delta t \cdot S(\vec{Q}_{pri})$
        for $i = x, y, z$ **do**
            compute fourth-order FDs $(\Delta\vec{Q}_{pri})_i$ ▷ Eq. (D.1)
            compute NCP $B_i(\vec{Q}_{pri})(\Delta\vec{Q}_{pri})_i$
            compute KO term $(KO_{\vec{Q}_{pri}})_i$ ▷ Eq. (D.6)
        end for
        $RHS_{\vec{Q}_{pri}} \leftarrow RHS_{\vec{Q}_{pri}} - \sum_i \Delta t \cdot B_i(\vec{Q}_{pri})(\Delta\vec{Q}_{pri})_i$
        $RHS_{\vec{Q}_{pri}} \leftarrow RHS_{\vec{Q}_{pri}} + \sum_i \Delta t \cdot (KO_{\vec{Q}_{pri}})_i$
        $\vec{Q}_{pri} \leftarrow \vec{Q}_{pri} + RHS_{\vec{Q}_{pri}}$
        for $i = x, y, z$ **do**
            compute fourth-order FDs of the primary $(\Delta\vec{Q}_{pri})_i$
            Assign the auxiliary variables $\vec{Q}_{aux} \leftarrow (\Delta\vec{Q}_{pri})_i$
        end for
        $t \leftarrow t + \Delta t$
        compute λ_{max} to inform next time step
    end for
end for

$$\partial_t Q + c \frac{r}{x_i} \partial_i Q = -\frac{c}{r}(Q - Q_{\text{ini}})\Big|_{\partial\Omega}, \quad i \in \{1, 2, 3\} \tag{21}$$

where $r = \sqrt{x^2 + y^2 + z^2}$ is the coordinate distance from the system's origin to the considered boundary position and $x_i \in \{x, y, z\}$ is the co-ordinate perpendicular to the considered boundary. Q_{ini} is the value of evolving quantities at the boundary in the initial condition, as it serves as a "background" and thus does not enter the wave-like equation. c is the wave speed, i.e., the eigenvalue of the characteristic matrix of the evolution system.

In the code, the derivatives are replaced by corresponding finite differences. To initialize layer 3 of a FD4 setup with $K = 3$ along a vertical boundary face (cmp. Fig. 5) we compute

$$\frac{1}{\delta t}\left(Q_3^{n+1} - Q_3^n\right) + c \frac{r}{x_i \delta x_i}\left(Q_3^n - Q_2^n\right) = -\frac{c}{r}\left(Q_3^n - Q_{\text{ini}}\right), \tag{22}$$

with an element size of δx_i. The indices for the y and z directions are omitted as they are all the same in this case. The upper indices n and $n + 1$ represent the previous and current time step. As we work our way outside of the domain to fill all layers of the boundary data iteratively, r, c, and x_i in Eq. (22) need to be adjusted accordingly.

3.6. Adaptive mesh refinement

Adjacent patches are coupled with each other through their shared faces. On a regular grid, keeping face data consistent with its adjacent cells requires trivial one-to-one copies. Special care is needed along mesh refinement transitions, i.e., along the boundary where the resolution changes between patches. However, the principle remains the same: adjacent patches communicate through their shared faces.

Logically, we can read the adaptive meshing over a spacetree structure as a mechanism to span ragged, non-overlapping Cartesian meshes: the patches of one resolution form one regular mesh which has holes where the code decides to use a finer or coarser resolution, respectively. This means we encounter "boundary" faces on this level, where the $\vec{Q}_{\text{out}}$ are not set by an adjacent patch.

The $\vec{Q}_{\text{out}}$ data on such hanging faces are initialized from the coarser volumes prior to the mesh traversal. We *interpolate*. The counterpart is the *restriction*. Both schemes are implemented exclusively over face data. Alternative implementations [44] realize the interpolation over the patches themselves, i.e. use a volumetric restriction. While ExaHyPE facilitates such volumetric coupling in principle (cmp. discussion below), we do not use expensive volumetric coupling in ExaGRyPE's use cases presently. Furthermore, we write the interpolation rules down with a tensor-product approach which ignores diagonal coupling, i.e., interpolation of elements that are not aligned along a coordinate axis, and we use exclusively a trilinear scheme.

Denoting the quantities of the volumes in the coarse and fine faces are $Q_{i,j,k}^c$ and $Q_{i,j,k}^f$, the interpolation map can be written down as

$$Q_{i,j,k}^f = P_{il}^x P_{jm}^y P_{kn}^z Q_{l,m,n}^c, \tag{23}$$

where P^x, P^y, and P^z are the matrices responsible for the corresponding one-dimensional map. Analogously, we obtain the matrix R^i
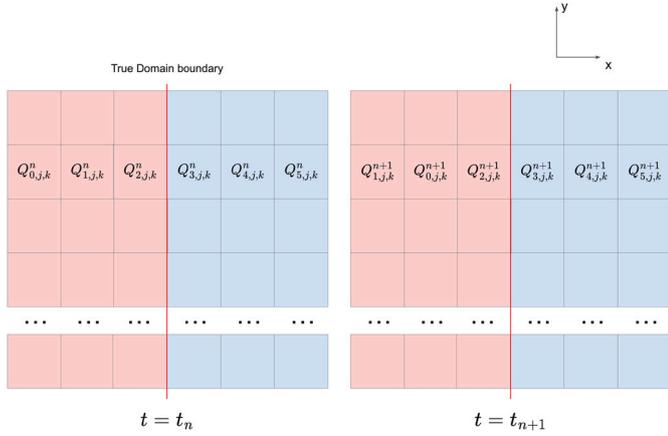
Fig. 5. Schematic illustration of boundary initialization for $K = 3$: Half of the face's solution data is outside the domain (right, light blue). We need to initialize the outside layers of the boundary ($Q^{n+1}_{3,j,k}$, $Q^{n+1}_{4,j,k}$, $Q^{n+1}_{5,j,k}$) from values inside the domain to impose proper boundary conditions and allow the adjacent patch to continue its time stepping with valid halo data. We first assign the values $Q_{3,j,k}$ use $Q_{2,j,k}$ by treating them as $\vec{Q}_{out}$ and $\vec{Q}_{in}$ in the boundary condition function, then assign the values $Q_{4,j,k}$ using $Q_{3,j,k}$, and finally assign the values $Q_{5,j,k}$ use $Q_{4,j,k}$ to fill all three layers. We work our way outwards. More sophisticated schemes setting all values in one go are feasible within ExaHyPE yet not used within ExaGRyPE. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

$$Q^c_{i,j,k} = R^x_{il} R^y_{jm} R^z_{kn} Q^f_{l,m,n} \qquad (24)$$

for the restriction (see Appendix E for the exact formulation of those matrices). ExaGRyPE currently defaults to averaging for the restriction.

ExaHyPE's face-centric scheme is convenient to implement, as it allows users to solely focus on face-to-face interactions. We also can use it to construct interpolations and restrictions over multiple levels of refinement automatically: we recursively apply the intergrid transfer operator. ExaGRyPE does not enforce any 2:1 balancing [45]. One might want to call 3:1 balancing for Peano as we use tri-partitioning, which enforces that the cells adjacent to any face differ in size at most by one level of refinement, i.e., a factor of three in Peano. No such constraint exists, although users might want to impose it manually.

The logic where to refine or coarsen can be realized by the user. ExaGRyPE ships with examples of how to use feature-based refinement for example, where the maximal difference between any two adjacent elements within a patch triggers either a refinement or coarsening in the subsequent time step. This allows for solution-driven, dynamic adaptive mesh refinement. However, the present experiments with ExaGRyPE all use a static refinement: we know where the black holes reside or the orbits on which they rotate initially, and prescribe concentric spheres of refinement levels around these areas. Dynamical adaptive mesh refinement is not free and induces load-balancing challenges. The static refinement circumnavigates these.

There are three open research questions to study: First, codes without 2:1 balancing are prone to strong reflections along the resolution transitions, as high-frequency waves cannot escape regions of high resolution [46]. Second, higher-order interpolation might be required to match the higher-order spatial discretion of the solvers, and we have already observed improvement in solution by using the second-order scheme in certain benchmarks below. Finally, our resolution transition scheme does not guarantee the conservation of the interpolated and restricted physical quantities, which may unexpectedly affect the long-term evolution of the spacetime.

3.7. Volumetric coupling of multiple solvers

In ExaHyPE, the mesh can carry multiple solvers at the same time. In such a case, they run parallel while the code traverses the mesh. ExaGRyPE's solvers can be composed, i.e., they can be told to run in-sync. This way, solvers with different time step sizes (e.g., different λ_{max}), solvers with different time integrators, or solvers with different mesh resolutions are kept synchronous: we can, for example, disable them for some mesh traversals or manually prescribe their time step size.

This is a synchronization or coupling of the global solvers' states. ExaGRyPE can also couple different solvers per patch: if a spacetree cube holds two patches—of an FD4 and an FV solver, e.g.—we can hook into a patch update postprocessing step and manually overwrite either of them after each time step. This way, it is possible for ExaGRyPE, for example, to run an FV solver around a black hole and to use FD4 further away, we conduct a simulation test based on this and report in Section 5.2. The feature is typically combined with a localization of the solvers, where each solver holds data only for some cubes of the underlying spacetree.

3.8. Particle tracers as data probes

ExaHyPE features particles which are embedded into the computational mesh. This feature originally has been introduced to support Particle-in-Cell schemes [47]. In ExaGRyPE, we use it to realize tracers. Particles are inserted into the mesh for data probing and object tracking. They are not evolved independently, but instead either follow evolving variable fields in PDEs—the quantities in $\vec{Q}$ which represent a velocity field for example—or are spatially invariant and simply record the values in $\vec{Q}$ at this point.

Particles allow users to track the solution's evolution at certain points in the domain without high I/O overhead, as they sample the solution in one point and do not require us to dump large output files. If they move, they yield trajectory information of important objects in the domain. Adding tracers to an ExaGRyPE simulation involves three stages:

- *Initialization*. The users specify the number and initial coordinates of the tracers. ExaHyPE offers two approaches for initializing the tracer coordinates: one can set the coordinates explicitly by writing them down in the ExaGRyPE specification script from where they are passed into ExaHyPE, or we can provide a coordinate file. Besides the coordinates, users also need to specify what evolving variables the tracers need to plot.
- *Data Projection*. Users then decide which and how the solver's values on the patches are projected onto tracer attributes. ExaGRyPE currently uses trilinear interpolation, i.e., it uses the values within the closest 2^d elements (Finite Volumes or Finite Difference sample points) to determine the value at the tracer particles. Along with the projection instructions, users can declare three quantities which serve as velocity field v_i for the tracers. If specified, the tracers' positions are updated using an explicit Euler time integrator

$$x^{t+\delta t}_i = x^t_i + \delta t v^t_i, \qquad (25)$$

where x^t_i is the position of the considered tracer at time t, and v^t_i is the velocity reconstructed from the solver solution. It allows tracers to follow the flows within the simulation. δt can be set to the solver's time step size.
- *Data dumping*. Tracers record "their" values in each and every time step. ExaGRyPE can instruct them to dump this chronological data into output files in CSV format. For large-scale simulations, ExaHyPE 2 offers various thresholds to control the dumping: users can ask tracers to dump in fixed code time intervals, or to dump when recorded variables or tracking positions have changed substantially. To reduce the memory footprint, the recording can be lossy, i.e., only record new values whenever their relative difference to previously recorded values exceeds a certain threshold.

The tracer module is used in ExaGRyPE extensively as the "extractor" of gravitational wave signals and for black hole puncture trackers.
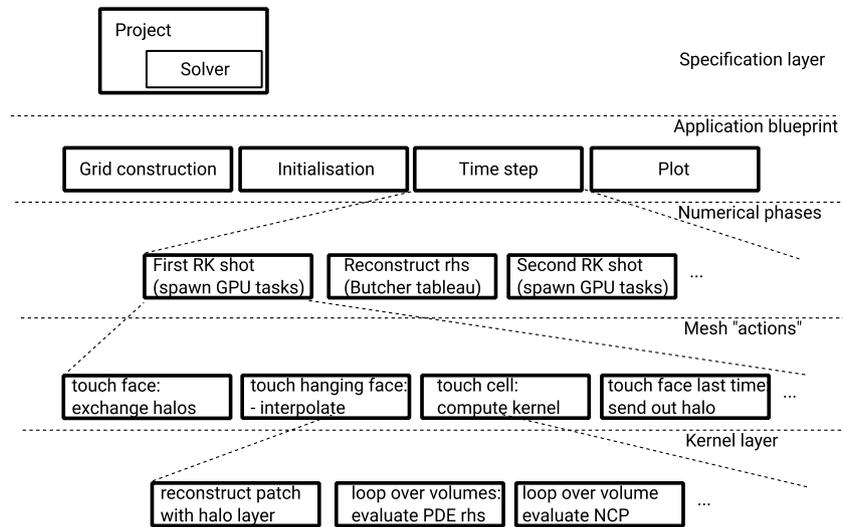
Fig. 6. Starting from an abstract specification (which solvers to use, initial mesh resolution, total runtime, time in-between plots, . . .), we generate a tailored series of tasks in multiple steps. We lower the representation from a high-level description into something that reflects the C++ implementation one-to-one.

4. Functional decomposition and software architecture

An *ExaHyPE project* forms the starting point of any ExaGRyPE endeavor. It is eventually translated into a set of C++ classes defining the actual executable. It is lowered in multiple steps into the actual execution code. In ExaGRyPE, we rely on Python bindings for an ExaHyPE project, while the actual code is represented by a Peano project that can be translated one-to-one into C++ code. The starting point is hence an instance of `exahype2.Project` in Python.

Once an ExaHyPE project, aka instance of the Python class, is set up, we lower it into a sequence of four tasks: initial mesh generation, solution initialization, time stepping and I/O steps. Every ExaGRyPE project consists of these fundamental steps, where plotting and time stepping are iteratively executed. The lowering is a mere task sequence instantiation. From hereon, we focus on the time step.

An ExaHyPE project hosts an arbitrary number of solvers. The solvers' time-stepping scheme determines how many mesh traversals we need to implement in one time step, as well as the semantics per traversal. Let a solver be a Runge-Kutta scheme of order $k \leq 4$. ExaHyPE lowers a time step into four mesh traversals. For $k > 4$, we hit the Butcher barrier [48] and hence require additional mesh sweeps. This lowering follows the solver's time stepping specification. Besides the translation into a sequence of mesh traversals, a solver carries meta information such as maximum mesh size, patch dimensions p, the halo size K or the number of unknowns. This yields a specification of the required data structures: an ExaGRyPE FD4 for example makes the lowering fit a $p \times p \times p$ patch into each cell, equip the faces with an overlap of three, i.e. a $2 \cdot 3 \times p \times p$ patch, and store helper data for the selected Runge-Kutta scheme such as temporary storage for the right-hand sides. The solver also informs the lowering process what global parameters the solver has to host—such as $\lambda_{\max}$—and how they are exchanged and kept consistent. For multiple solvers, our translation into a lower abstraction level ensures automatically that solvers are kept in sync.

Once the nature of a mesh sweep is known, we can lower it once more into fundamental tasks over mesh entities: Per cell, we know how to equip the current time step or Runge-Kutta guess data with halo data information, i.e., how to reconstruct the $(p + 2K) \times (p + 2K) \times (p + 2K)$ patch, which compute kernel (update of a patch) to invoke, which halo data to exchange, which AMR criteria to evaluate, which patch postprocessing to call, and so forth. We take the numerical recipe and lower it onto a task graph over the actual mesh in use.

The compute kernels themselves can be lowered one final time: a time step in a Finite Volume scheme for example maps onto a sequence

of d-dimensional loops over all volumes within a patch [19]. Per volume, we evaluate the actual PDE terms. This is the lowest level within ExaGRyPE's cascade of abstraction levels: within this final lowering step, we create a C++ class per solver. It allows the user to hook into mandatory steps of the simulation such as the refinement control or the patch initialization, but it notably also provides a method stub per PDE term in use. This empty stub has to be filled by the user. Alternatively, users can define in the top-level specification which functions have to be called for the PDE terms, the initialization, the refinement criterion, and so forth. If specified, the final lowering step inserts these function calls into the stub. ExaGRyPE provides pre-defined functions for all features used in the present manuscript.

Our description of ExaGRyPE's abstraction levels relies on multiple levels of abstraction layers. Users provide an abstract description in Python. From this *specification layer*, we can break down the workflow into finer and finer logical representations. This lowering is completely hidden from the user—it is realized in the ExaHyPE engine layer—and it relies on a task language (Fig. 6). On the higher abstraction levels, the tasks describe sequences of activities that are performed to produce the simulation output (*application blueprint* or workflow). Once we break down these activities into activities per mesh entity (mesh *actions*) [14], we obtain a real task graph over the mesh: what has to be done for a cell exclusively relies on the fact that all adjacent hanging faces have been interpolated completely and the halo layers of the adjacent faces of the $2d$ faces of this cell have been made consistent. Data along a face that is to be exchanged via MPI to keep the halos for the subsequent time step consistent can be sent out immediately after the corresponding adjacent cell has finished its kernel. This list is not comprehensive.

All lowering down onto mesh traversals and activities per mesh entity is automated. We implement it within the ExaHyPE engine once our ExaGRyPE description has built up the program specification. It is only the very last step that links back to actual user code and might require manual intervention unless the used implementations are inserted into the specification script as well.

4.1. Static software architecture

As ExaGRyPE is built on top of ExaHyPE, it inherits its multi-layer static architecture (Fig. 7). A technical architecture wraps away various external libraries such as OpenMP, TBB and MPI, but also I/O facilities such as HDF5. In ExaGRyPE, we can hence swap different multithreading backends in and out. It is written in plain C++ although some subcomponents use extensions such as SYCL or OpenMP pragmas.
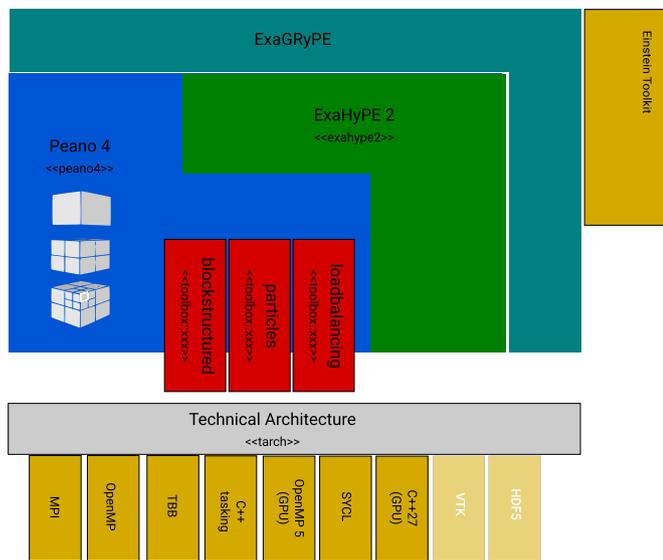
Fig. 7. The schematic architecture of the C++ libraries used within ExaGRyPE. The Python layer on top mirrors the core architecture, but can omit representations of the technical architecture and third-party tools in Python.

The technical architecture is a utility collection used by all of our code. Peano is the actual AMR and mesh traversal component. It is written in C++. Its design principles follow the "*don't call us, we call you*" paradigm introduced with its third generation [14], i.e. this component owns the mesh and its traversal. While it runs through the mesh, it issues events such as "*I read this vertex for the first time*" [40]. Codes can hook into these events to realize the actual solver semantics.

ExaHyPE is such a code. Per solver and solver flavor (different optimization variants of each solver exist for example), it provides classes and functions that plug into these events. It is all written in C++ and also uses routines from Peano as well as the technical architecture. As many features within ExaHyPE are used by multiple solvers or even other applications built on top of Peano, they are modeled as separate toolboxes: inside the C++ core of Peano plus ExaHyPE we functionally decompose the code again.

ExaGRyPE adds additional domain-specific features on top of ExaHyPE, but also makes contributions to ExaHyPE's core functionality. The latter includes a solver for fourth-order Finite Differences that utilizes the block-structured routines or bespoke boundary conditions for example. Other features such as the implementation of the CCZ4 PDE terms or the KO damping are domain-specific and kept separate from the ExaHyPE core. Here, ExaGRyPE is a strict extension of ExaHyPE. These extensions comprise also routines to initialize the simulation setup. While Peano and ExaHyPE are independent of third-party libraries or, through the technical architecture, shielded from them, the initialization routines within ExaGRyPE link to the EINSTEINTOOLKIT [5].

All of ExaGRyPE, ExaHyPE and Peano are written in plain C++ with a technical architecture which separates them from extensions such as SYCL. While we do use metaprogramming in many places, a lot of these C++ ingredients are rather independent and not tied together. Instead, Peano offers a Python wrapper that can represent a whole Peano application including its Makefile, global variables/settings and so forth. This Python wrapper comprises all the glue code required to link certain components to each other. For example, it exactly defines which memory is to be allocated per spacetree cube, which functions are to be called whenever we run through a face throughout the mesh traversal, and so forth. The Python component represents the lowest level of our automatic code lowering and can be translated one to one onto C++ glue code, i.e., a working executable linking to all static Peano, ExaHyPE and ExaGRyPE libraries comprising all building blocks.

The C++ architecture is mirrored by the Python API. We do not need a representation for the technical architecture or any third-party library underneath. The most basic representation is the Python Peano layer which represents a Peano C++ application exactly plus all required build information (makefile). All toolboxes have their Python equivalent such that we can add them to a Peano project and configure them on the Python level throughout the lowering process. ExaHyPE adds an additional Python abstraction on top, where individual objects represent solvers, data structure definitions, and so forth. On this level, we also explicitly host the main routine.

On top of the ExaHyPE Python layer, we install the ExaGRyPE layer, which reflects the program specification and offers many domain-specific helper routines. In this code design, all lowering can be done exclusively in Python. Objects within the lower Python layers however often correspond one-to-one to C++ objects.

4.2. Programming workflow

We run through the structure and programming of a ExaGRyPE solver by means of a simple gauge wave benchmark. The example highlights how the individual simulation steps manifest in Python instructions and illustrates how they inform the lowering. Every ExaHyPE project follows the same structure (Algorithm 3): we create a project and hence implicitly define the application blueprint (Fig. 6) with its four distinct phases.

The project is manually lowered down into a Peano project, which is the one-to-one equivalent of C++ source code. Consequently, we dump this project then onto disk. We assume that all ExaGRyPE, Peano and ExaHyPE libraries have been built before using either the autotools or CMake. The Python script automatically picks up the used configuration arguments and creates an appropriate Makefile alongside the generated C++ code. Load balancing, additional third-party libraries, or global I/O settings all can be added to the Peano project before we write to disk. While ExaHyPE allows users to define their own load-balancing strategy, the current version of ExaGRyPE utilizes the default built-in option. We plan to conduct a full investigation of load balancing strategies in future work along with the performance analysis.

4.2.1. Solver configuration

The actual ExaGRyPE solver has to be added to the ExaHyPE project before triggering the lowering. While the creation is close to trivial (Algorithm 4), different variants are on the table about how to connect the vanilla PDE solver to the actual physics (Fig. 3). Without any further information, the lowering yields an empty C++ class that users have to fill (Algorithm 5). After that, the lowering process will create almost all simulation code without any further user interference. Only on the very lowest level, when we actually evaluate the PDE terms or the refinement criterion or require initial values, it will call routines from the PDE solver.

As an alternative to the manual filling of the routines in C++, all solvers in ExaHyPE offer a set_implementation routine. With this one, users can inject C++ code directly into the generated code from the Python API. ExaGRyPE provides wrappers around the generic ExaHyPE solvers. There is a set of bespoke Finite Volume solvers, bespoke FD4 solvers, and so forth. They already set some default implementations, configure halo layer sizes, or set the correct number of unknowns.

Grid construction As the first step of the simulation, Peano constructs the computational grid and builds the corresponding spacetree. In this stage, it also already splits up the computational domain following the advice of the load balancing metrics. Peano offers plug-in points to guide the mesh refinement, and ExaHyPE automatically runs through all the ExaGRyPE solvers searching for the minimum of their maximum mesh sizes. It ensures that the initialization starts from a sufficiently fine grid.

Once we have a reasonably fine initial mesh, ExaHyPE starts to query the solver objects if a certain area should be refined. In this stage, the

---

Algorithm 3 Baseline structure of any ExaGRyPE solver within ExaHyPE.

1: project = exahype2.Project(namespace = ["benchmarks", "exahype2", "ccz4"], name = "CCZ4", executable = "test ")
2: project.set_global_simulation_parameters(dimensions = 3, offset = [-0.5, -0.5, -0.5], domain_size = [1, 1, 1], periodic_boundary_conditions = [True, True, True], end_time = 1.0,...)
3: project.set_Peano4_installation(directory =..., build_mode = peano4.output.CompileMode.Release)
4:         ▷ Parse Peano's configuration
5: ...   ▷ Actual solver construction
6: peano4_project = project.generate_Peano4_project()
7:         ▷ Lowering
8: peano4_project.generate()
9:         ▷ Spill out C++ code plus Makefile

Algorithm 4 Creating an actual ExaGRyPE solver is a two-liner.

1: my_solver =
2: exagrype.api.CCZ4Solver_FD_GlobalAdaptiveTimeStep(name = "CCZ4FD", patch_size = 9, min_volume_h =...)
3: project.add_solver(my_solver)

Algorithm 5 By default, ExaHyPE creates one (empty) class per solver. Users can fill this one manually. The class signature resembles the example given as a pseudo-code below. The (generated) superclass holds generic information passed into the Python representation such as the used patch size p.

```
1: class ...::EXAHYPE2::CCZ4::CCZ4FD:  PUBLIC  ...::EXAHYPE2::CCZ4::AB-
    STRACTCCZ4FD {
2:     virtual REFINEMENTCOMMAND REFINEMENTCRITERION (
3:         double Q[58],
4:         const Vector<Dimensions,double>& x,
5:         ...
6:     ) override;
7:     virtual VOID INITIALCONDITION (
8:         double Q[58], // out
9:         const Vector<Dimensions,double>& x,
10:        ...
11:    ) override;
12:    virtual VOID NCP (
13:        double Q[58],
14:        const Vector<Dimensions,double>& x,
15:        ...
16:        int normal,
17:        double BgradQ[58] // out
18:    ) override;
19:    ...
20: };
```

---

solvers do not yet have a solution at hand. Either we hard-code the initial refinement pattern, or we make the algorithm run through a decision logic which analyses "*if we had a solution here, would we refine*". This implements a feature-based initial refinement. The present case studies all use hard-coded patterns, adopting static grids where the finest resolution covering the whole interested region. The refinement criterion can be either written into the C++ template or injected the implementation of the C++ code directly from Python via `my_solver.set_implementation(refinement_criterion = "my fancy C++ code snippet")`. The dynamic grid refinement during evolution is also available in ExaGRyPE and can tag volumes for refinement either based on coordinates, similar to the "Tracker-based AMR" in [49], or based on variables, similar to the "tagging on gradient" in [50]. This dynamic feature is not used for the current benchmarks, and a full investigation is left for future work.

Grid initialization As soon as the initial grid is initialized, i.e., as soon as ExaHyPE's main logic realizes that the grid does not change anymore, and as soon as the initial load balancing has terminated, the code switches into the initialization phase. ExaHyPE now embeds the actual compute data structure, i.e., the patches, into the spacetree cubes and then runs through each element (Finite Volume or FD sample point) and

sets the initial value. This happens by calling the solvers' `initialCondition` routine.

Algorithm 6 The initial conditions for the Gauge wave setup can be set directly through the Python interface. It injects C++ code snippets into the generated solver that call a ExaGRyPE C++ factory function setting the Gauge wave data.

```
1: my_solver.set_implementation( initial_conditions = """
2:     for (
3:         int i = 0;
4:         i<NumberOfUnknowns+NumberOfAuxiliaryVariables;
5:         i++
6:     ) Q[i] = 0.0;
7: ::applications::exahype2::ccz4::gaugeWave(Q, x, 0);
8: """)
```

Our present ExaGRyPE solvers realize two different approaches for this: for simple setups where we know the analytical solution, the function body of `initialCondition` is hard-coded and assigns the initial values of evolving variables directly. This happens for the gauge wave benchmark, e.g., for which ExaGRyPE provides a helper function (Algorithm 6).

For more complex setups such as the black hole system where a numerical solution is required that obeys the Hamiltonian constraints, we link against the TWOPUNCTURE module from EINSTEINTOOLKIT. It calculates the spacetime quantities γ_{ij}, K_{ij} and α from the specified physical parameters like masses and momenta of the black holes. ExaGRyPE provides a wrapper which converts the basic spacetime quantities (γ_{ij}, K_{ij}) from the external library into the ones that ExaGRyPE utilizes ($\tilde{\gamma}_{ij}$, $\tilde{A}_{ij}$, ϕ, K).

If combined with FD4, ExaGRyPE offers a fourth-order finite difference helper function to compute the first derivative feeding into the auxiliary variables (9). This is a post-processing step which can be combined with any initial condition.

Evolution equations (timestepping) The actual PDE evolution is pre-configured in all ExaGRyPE solvers, i.e. they call `set_implementation` for the non-conservative product and source code throughout the construction, and they also provide an implementation of the λ_{max} calculation. Users can redefine and alter it, but our present experiments all rely on the default settings. The three core routines are re-implementations in C++ of the Fortran implementations of [15], although we modified some Fortran code entries to align them with the CCZ4 formulation as outlined in the Appendix.

An explicit time stepping over the CCZ4 formulation yields solutions which violate the CCZ4 constraints [51]. The default ExaGRyPE solvers therefore add a post-processing of $\vec{Q}$ to each time step's computational kernel. It enforces the two *algebraic constraints* $\det(\tilde{\gamma}_{ij}) = 1$ and $\text{tr}(\tilde{A}_{ij}) = 0$ point-wisely: the corresponding evolving variables in $\vec{Q}$ are

over-written to satisfy the constraints, i.e. the traceless property is re-achieved by removing the trace from every component of the tensor, and the unity of determinant is re-achieved by rescaling every component.

If an FD4 solver with a second-order formulation is instantiated, ExaGRyPE's solvers add an additional mesh sweep to each solver which serves as an epilogue to each time step. It reconstructs the derivatives (Section 3.2). The additional mesh sweep is necessary, as the reconstruction requires some inter-patch/MPI data exchange to get all halo data consistent.

Further features Boundary conditions can be injected via `set_imple-mentation` but are set to Sommerfeld by default. The routine to control dynamic adaptive mesh refinement defaults to ExaHyPE's KEEP command, i.e., the mesh is neither refined nor coarsened. More advanced setups might want to alter this behavior.

4.2.2. I/O and post-processing

ExaHyPE dumps data into a bespoke patch format, which can dump patches with cell- or vertex-associated degrees of freedom, higher-order polynomials or mesh metadata with low overhead and without any constraints on dynamic adaptivity. The mesh can change from each and every data dump. The dump is realized through a writer in the technical architecture. Peano ships a Python tool to merge and convert them into the standard VTU files, which is compatible with various visualization software. The realization relies on the VTK/Paraview software.

Given the scale of the simulations, dumping all 58 equations over the whole computational domain results in large output files. Therefore, we incorporated a *data filter* into our code. This filter enables users to selectively extract slices and clips from the domain. One can also specify what variables are to be included in the output files.

This in-situ filtering reduces the I/O pressure significantly. Auxiliary variables used as helper quantities in the evolution for example are often omitted in the snapshots. If phenomena are known to show intrinsic symmetry or invariance along certain coordinate axes—cmp. the static single black hole scenario which has rotational symmetry or the gauge wave setup which effectively yields a one-dimensional evolution of interest—we can boil down the output file sizes significantly.

Besides mere filtering, we can also compute accuracy metrics in situ: if enabled, *Hamiltonian and Momentum constraints* (B.37)–(B.38) are calculated after each time step and attached to the evolving quantities $\vec{Q}$ as "additional variables". As the calculation of the extra quantities is separated from the computational kernel, ExaGRyPE offers a big freedom to pick the computational rule or cardinality of the output. We can calculate intermediate quantities, e.g., the Ricci tensor R_{ij} or the conformal connection function $\bar{\Gamma}^i$, for testing purposes.

4.2.3. Tracers

Algorithm 7 A one-liner adds tracers to an ExaGRyPE solver.
```
1: my_solver.add_tracer(
2:     name = "Tracer",
3:     coordinates = [[-0.4251, 0, 0], [0, 0, 0], [0.4251, 0, 0]],
4:     project = project,
5:     number_of_entries_between_two_db_flushes = 100,
6:     data_delta_between_two_snapsots = 1e-8,
7:     ...
8: )
```

Each ExaGRyPE solver is pre-prepared to host tracers (Algorithm 7). If they are added, the solver automatically picks an appropriate interpolation scheme and informs Peano that appropriate particle handling is to be added. Multiple tracer types can be added to each solver.

Tracers write their outcome into a plain CSV file for all unknowns from $\vec{Q}$ specified plus the tracers' position. ExaGRyPE also automatically assigns each tracer a unique ID, such that post-processing scripts can track their evolution over many time spans. The I/O capabilities of

a tracer can be configured such that updates to the CSV file are written if and only if enough particles have altered their state, i.e., we can constrain the output frequency. We can furthermore configure after which relative change of a particle quantity we consider the particle worth writing a trajectory update into the database.

Tracers unfold significant potential once we couple it with the in-situ post-processing: for the ψ_4 *calculation* for example, we calculate the ψ_4 quantities following Section 2.4 and attach them to the solution vector $\vec{Q}$. Rather than plotting, we interpolate it onto the tracers and let them record the ψ_4 evolution over time. Once the tracers are arranged over a sphere, they enter the integral (15) to extract the gravitational waves.

Puncture trackers are different tracers. They move. We drop the virtual particles into the initial locations of the black holes and specify the opposite shift vector $-\beta^i$ to serve as their velocity. They feed into the explicit Euler time integrator (Section 3.8). The trajectories of those particles represent the movement of the black holes.

4.2.4. Coupling

Coupling of two ExaGRyPE solvers starts from the creation of two solvers within the Python script which are both added to the ExaHyPE project. Without further steps, these solvers run completely independent of each other. Different to all steps previously, coupling requires users to run through a series of steps:

First, one solver has to determine the other solver's time step sizes. Each solver's generated C++ class allows users to overwrite the routine `startTimeStep` which is the canonical place to let one discard its own time step size determined by the $\lambda_{\max}$, and instead use the other solver's step size. From here on, the two solvers run in-sync.

Second, it might be reasonable to restrict the computational domain of one solver, unless we want both solvers to cover the whole domain and to couple volumetrically everywhere. For this, ExaHyPE solvers can redefine an internal function `_store_cell_data_default_guard` and similar routines for faces such that the solver's area, i.e. where it exists, is narrowed down. The routines return a C++ statement which has to be evaluated into a boolean expression.

Finally, the solvers have to be coupled. For doing this, all solvers host an attribute `_action_set_preprocess _solution` which is by default `None`. We reset it to an instance of `exahype2.solvers.rkfd-.actionsets .PreprocessReconstructedSolutionWithHalo`, e.g., and inject the interpolation/restriction from one solver to the other here. Again, a simple if statement can localize the operations.

5. Numerical results

In this section, we present ExaGRyPE simulation results for various physical scenarios. We begin with a standard gauge wave benchmark and then examine a black hole space-time setup, discussing both the single Schwarzschild black hole and the binary black hole merger. The tests are used to validate the correctness of the code implementation. While they affirm that the code base is working properly and reasonably robust for numerical astrophysics applications, they also reveal required optimizations and challenges that will be addressed in the future.

5.1. Gauge wave

The gauge wave scenario is one of the standard test cases for numerical relativity codes [52]. A flat Minkowski spacetime is considered and no actual "physical" phenomenon occurs in the system, i.e. solution characteristics should stay invariant under a consistent and robust numerical scheme. For this, we slice the static space-time by performing a time-dependent coordinate transformation:

$$\hat{t} = t - \frac{A}{2k\pi} \cos[k\pi(x-t)], \tag{26}$$

$$\hat{x} = x + \frac{A}{2k\pi} \cos[k\pi(x-t)], \qquad \text{and} \tag{27}$$

$$\hat{y} = y, \ \hat{z} = z, \tag{28}$$
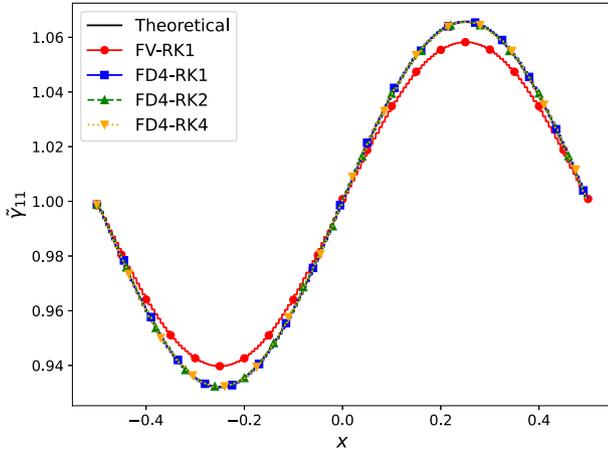
Fig. 8. The profile of $\tilde{\gamma}_{11}$ along x-axis at code time 0.5 for different solvers: FV-RK1 (red solid), FD4-RK1 (blue solid), FD4-RK2 (green dashed) and FD4-RK4 (orange dotted).

which transcribes the original Minkowski metric $ds^2 = -d\hat{t}^2 + d\hat{x}^2 + d\hat{y}^2 + d\hat{z}^2$ into

$$ds^2 = -H(x,t)dt^2 + H(x,t)dx^2 + dy^2 + dz^2, \tag{29}$$

$$H(x,t) = 1 - A\sin[k\pi(x-t)]. \tag{30}$$

It yields a gauge wave propagating along the x-axis (positive-wards) with an amplitude of A. As we know the complete four-dimensional metric here, and the related quantities can be read straightforwardly:

$$\alpha = \sqrt{H}, \quad \beta^i = 0, \quad \phi = H^{-1/6}, \quad \text{and} \tag{31}$$

$$K_{xx} = -\frac{\partial_t H}{2\alpha}$$

$$= -\frac{k\pi A}{2} \frac{\cos[k\pi(x-t)]}{\{1 - A\sin[k\pi(x-t)]\}^{1/2}}, \tag{32}$$

$$K_{ij,\text{others}} = 0, \tag{33}$$

$$K = \gamma^{ij} K_{ij} = \frac{K_{xx}}{H}$$

$$= -\frac{k\pi A}{2} \frac{\cos[k\pi(x-t)]}{\{1 - A\sin[k\pi(x-t)]\}^{3/2}}. \tag{34}$$

The static zero shift $\beta^i = \partial_t \beta^i = 0$ and the harmonic slicing $f(\alpha) = 1$ are adopted as the gauge conditions for this scenario. Despite its simplicity, this test poses a highly nontrivial challenge to the formulation and implementations of numerical relativity codes, as the periodicity, wave shape and amplitude have to be preserved over a long time span [53].

The first simulation test we report in this subsection utilizes a computational domain $\Omega = [-0.5, 0.5]^3$ on a regular grid subject to periodic boundary conditions. Periodic boundary conditions are also a newly added feature released with ExaHyPE 2 as opposed to the first-generation code. We divide the domain into 162 volumes per dimension, thus leading to a volume size of 0.006173. AMR is not enabled for this test.

We pick $A = 0.1$, $k = 2$ for the physical parameters. For the running parameters, we follow previous literature, setting $\kappa_1 = 1.0, \kappa_2 = 0, \kappa_3 = 0$ and $e = c = \tau = 1.0$. $\mu = 0.2$ keeps the impact of the constraints small (cmp. reports in [15]). As for the solver-specific parameters, the coefficient of the KO dissipation ϵ is set to be 8.0 and a CFL ratio of $C = 0.1$ is adopted. The parameters f and η are irrelevant as the shift vector does not evolve in the gauge wave scenario.

We illustrate the simulation result at code time $T = 0.5$ in Fig. 8. It shows the profiles of $\tilde{\gamma}_{11}$ along the x-axis for different kernels (FV, FD4-RK1, FD4-RK2 and FD4-RK4), employed with the first-order formulation. All the solvers yield the correct traveling speed and phase of the wave. However, the Finite Volume solver suffers from severe am-
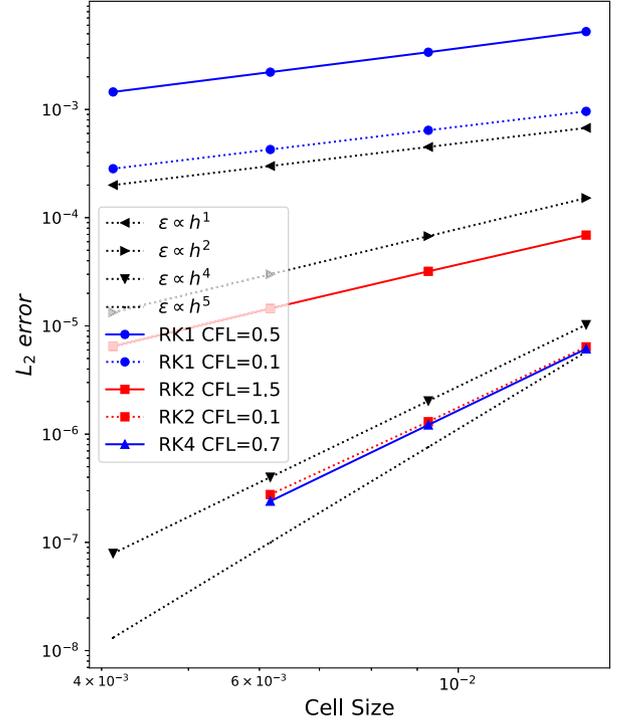
Fig. 9. Convergence behavior of FD4 with three different time integrators for different CFL constants. The convergence of the final error depends on the relative magnitude of the contributing terms, i.e., spatial and temporal discretization errors. See the text for detailed explanations.

plitude damping, and a similar dissipation is also observed in the black hole benchmarks. Consequently, we have chosen to use the FD4 scheme as our default solver.

On the other hand, different orders of the RK schemes give rather consistent results against the analytical solution. To further investigate different effects from the temporal discretization scheme, we calculate the L_2 error of the Hamilton constraint as

$$L_{2,H} := \sqrt{\frac{\sum_i (H_{\text{simulated}}^i - H_{\text{theoretical}})^2}{n}}, \tag{35}$$

where n is the total number of grid points, and we always have $H_{\text{theoretical}} = 0$. We conduct a convergence study for different orders of the RK scheme based on this accuracy metric at code time $t = 0.4$ and report the result in Fig. 9. A similar study can be conducted for any other reasonably small time stamp, and long-term effects will be the subject of study later. All physical and running parameters of tests here are identical to the first benchmark above except for the KO coefficient, which is set to 0.1.

In our explicit time-stepping scheme, three terms contribute to the L_2 error:

1. The spatial discretization error of the physical quantities, which is the same for all setups and scales with the mesh size in a convergence speed $\sim h^4$.
2. The time discretization error due to the Runge-Kutta scheme of choice. Through Eq. (20), the chosen time step size correlates linearly to the mesh size, thus following the convergence line $\sim h^k$ where k is the order of the RK scheme.
3. The influence of the KO term, which is an artificial construct and hence constitutes an error in itself. It has a convergence speed $\sim h^5$ (see Appendix D).

How the L_2 error in our tests converges depends on which term takes the lead in magnitude. As shown in Fig. 9, for the RK1 scheme, the temporal
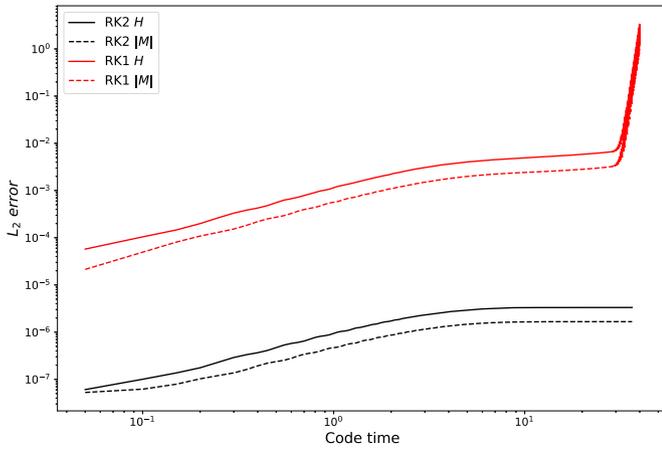
Fig. 10. Long-term evolution of the L_2 error with first-order (RK1) and second-order (RK2) time-integrator for the same gauge wave setup. The CFL constraint and KO term weight are chosen to dampen the constraint violation effectively. The low-order temporal scheme exhibits significant error accumulation, eventually leading to a crash, while the second-order scheme maintains a more stable error evolution throughout the simulation.

discretization error is quite large and results in an expected convergence speed $\sim h^1$ in the Hamiltonian constraint violation (blue lines with circle markers). RK2 and RK4 have relatively small temporal errors and if we have a too-large KO coefficient, then term 3 would overtake the leading term in error and yield an incorrect fifth-order convergence speed (not shown). When a proper (small) KO coefficient is set, the RK4 prediction also shows an expected fourth-order convergence (blue line with triangle markers). However, if the orders of spatial and temporal schemes do not match, the convergence of the final error will depend on the comparison of the magnitudes of each term. To demonstrate this, we plot two lines of the RK2 scheme with different CFL ratios, i.e., different sizes of the time steps. When the timestep is small (CFL = 0.1, red dotted line with square markers), the RK2 scheme shows nearly identical error magnitude and convergence speed with the RK4 line, indicating we are in a regime dominated by spatial discretization errors. On the other hand, in the big timestep case (CFL = 1.5, red solid line with square markers), the temporal discretization error is big enough and takes the lead, thus the line shows $\sim h^2$ convergence speed as we expected. We also identify more complex convergence behavior for the RK2 scheme when an intermediate time step size is used (0.1 < CFL < 1.5). We hope to investigate the convergence properties of ExaGRyPE's numerical schemes in more detail in future work.

For a code with known convergence behavior and a reasonably chosen damping through the KO term and the CFL factor, we expect to be able to run long-term simulations. Simulations without a damping term are in turn expected to become unstable after a while [16]. This hypothesis is subject to our final gauge wave experiment. Unfortunately, our data suggest that a mere application of the KO correction plus a reasonably small CFL factor is not sufficient to construct a long-term stable solution (Fig. 10): Even though the error seems to plateau after a while, it "suddenly" explodes within the plateauing region. Plots of the wave profile just before the error explosion reveal that the wave the pattern becomes skewed (not shown): The original sinusoidal pattern approaches a smoothed-out sawtooth profile, i.e. we observe some stiffening of the numerical solution. This is still a smooth effect, i.e. difficult for an artificial KO term to flag.

Convergence studies at a certain sample point consequently are insufficient to make statements on the long-term applicability of a code base. They might suggest that low-order time integrators are capable of long-term evolution of the system as long as we are willing to accept the resulting temporal discretization error. However, in practice, errors in the time evolution accumulate and eventually render the simulation unstable, i.e. make the scheme consistent but not converging. We need to

employ a high-order temporal discretization scheme (RK2 or RK4) that matches our high spatial discretization order to construct stable schemes for the gauge wave. Our results demonstrate that a second-order temporal scheme has significantly reduced the L_2 error and stabilizes the simulation, as shown by the black lines in Fig. 10.

The exact interplay of the KO term, CFL condition, Hamiltonian constraint violation, and long-term stability has to be the subject of future studies. We notably cannot make statements yet if this behavior results from an error accumulation that only arises for particular stationary or quasi-stationary solutions. Empirical data suggest that this is the case and that notably outflow conditions allow the error to escape from the domain (see the single black hole benchmark below). From hereon we pick reasonably small CFL factors and assume that the solution remains stable and consistent, while the arising error is dominated by the temporal discretization order.

5.2. Single Schwarzschild black hole

We next examine a single Schwarzschild black hole. This is yet another stationary test, where all numerical ingredients have to be carefully balanced against each other such that the initial setup effectively yields a long-term stable, time-invariant solution. The black hole is placed at the origin of the three-dimensional domain, with an ADM mass $M = 1$ and zero spin $S = 0$. All initial conditions for this test stem from the `TwoPuncture` module of the EINSTEINTOOLKIT library and the initial lapse is set to

$$\alpha = \frac{1}{2}\left(\frac{1 - \frac{1}{2}\,(M/r)}{1 + \frac{1}{2}\,(M/r)} + 1\right). \tag{36}$$

Other initial profiles and lapses (e.g., a power of conformal factor) are also used in other numerical codes. However, convergence from different setups has been observed: different initial lapses evolve into nearly the same static solution. The initial shift vector is set to zero. Also, the extrinsic curvature K_{ij} vanishes in the initial condition: its trace is zero as we assume the maximal slicing $K = 0$, and its residual is zero as no linear or angular momentum is presented in this system. Finally, we also assume conformal flatness in the initial condition, i.e. $\tilde{\gamma}_{ij} = 1$.

Besides the lapse field α, the only non-trivial quantity in the initial condition of this scenario is the conformal factor ϕ, which can be solved analytically as its source term now vanishes. The solution of its equation is

$$\phi \equiv \psi^{-2} = \left(1 + \frac{M}{2r}\right)^{-2}. \tag{37}$$

The domain in this test has a size of $[-9M, 9M]^3$. We use three levels of SMR (Static Mesh Refinement) based on the radial distance from the origin, which refines the domain at radius $r = 7M$ and once again at $r = 3M$. As we adopt a three-partition [14], the volume size of each level are $[0.333M, 0.111M, 0.037M]$. Sommerfeld conditions (Section 3.5) suppress reflections along the domain boundary. No volume center coincides with the origin to avoid resolving numerical infinity at the puncture location.

For setup and running parameters, we adopt $\kappa_1 = 0.1, \kappa_2 = 0, \kappa_3 = 0.5, e = c = \tau = 1.0$ and $\mu = 0.2$ for a similar evolving system as the original CCZ4 system [16]. The fully functional gamma driver condition is employed for the evolution of the shift vector β^i in the black hole system, and we set the parameters as $f = 0.75$ and $\eta = 1$ in Eqs. (B.14) and (B.20). The KO coefficient and the CFL ratio remain at 8.0 and 0.1, respectively, and the simulation runs until a code time of $T = 240$ with the RK1 temporal scheme. All figures below utilize $x := r/M$ as the unit of distance for simplicity.

Semi-singularity around black hole Over an initial ramp-up phase, our code yields a stable evolution of the space-time for ExaGRyPE's second-order formulation (Fig. 11). The solution approaches the static solution as expected [54]. However, a first-order formulation for the same
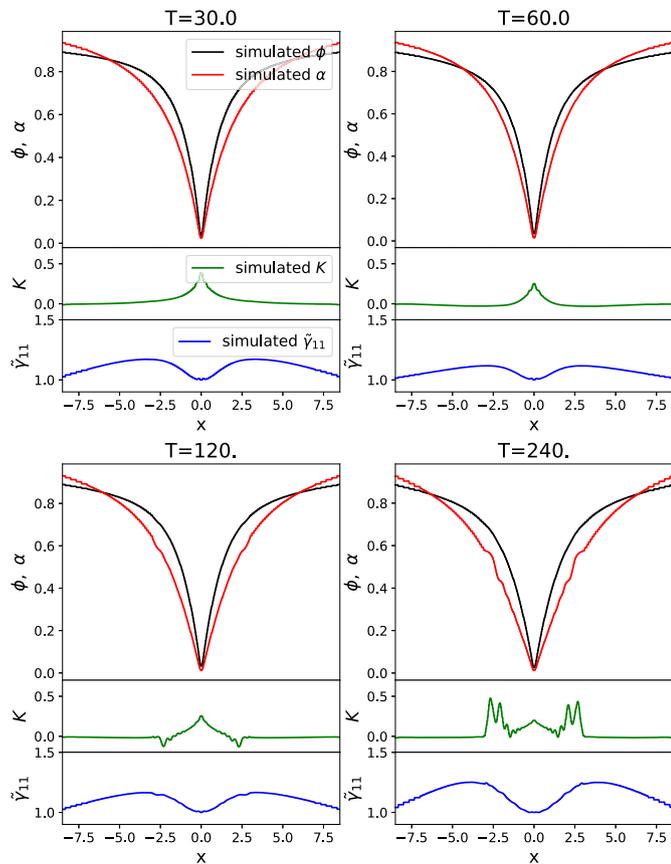
Fig. 11. Profiles of ϕ, α, K and $\tilde{\gamma}_{11}$ along the x-axis at four-time stamps $T \in \{30, 60, 120, 240\}$ for a single Schwarzschild black hole subject to a second-order formalism.

setup is not stable, the simulation crashes due to an unphysical solution (Fig. 12). While the second-order formulation yields a rather smooth and long-term stable solution, the first-order formulation develops instabilities around the center, which eventually lead to the crash. The same experiment with a higher-order time integrator brings the crash forward in time for the first-order formulation.

One possible explanation for the instability of the first-order formulation is that the derivatives of the auxiliary variables become large compared to those of the primary variables. Their errors are hence relatively large, too. For example, a primary quantity with a magnitude of 0.05 (a typical value for the conformal factor around puncture) would have a derivative around unity given a volume size of 0.04. Even a 1% numerical error within the derivative evolution therefore has a high impact.

The second-order formulation avoids this issue as the first-order derivatives are re-calculated every time step. They do not decouple from the primary variables (i.e., evolving upon their own evolution equations) and amplify themselves. It is not clear if a recalibration of additional damping terms could tackle the first-order instabilities which are inherent due to the semi-singularity of the solution close to the black hole. The derivatives are physically large. Adaptive mesh refinement can possibly mitigate this effect, but it may only serve to postpone the numerical difficulties in time.

Challenges arising around refinement boundaries Aggressive SMR introduces its specific challenges: with SMR, the evolution of a second-order formulation with KO damping remains stable until a distortion appears around the resolution boundary ($r = 3M$), where the fine grid transitions into a coarser resolution. There, K and α show a discontinuity of the gradient which gets amplified over time. $\tilde{\gamma}_{11}$ also exhibits fluctuations around the resolution boundary. The pattern is more clearly shown in Fig. 13, where the profiles for violation of the Hamiltonian constraint H are plotted along the positive x-axis, at six timestamps. Any violation around the black hole located at the center of the domain gets damped. However, the fluctuations along the resolution boundary ($r = 3M$) continue to grow and start to propagate inwards. The simulation is likely to become unstable when the fluctuations reach the puncture. This is confirmed by Fig. 14, where we plot the evolution of the L_2 error of the Hamiltonian constraint up to code time 120. The L_2 error in the first-order formulation increases quickly, departs from the stable second-order counterpart around 20 code time, and leads to a crash later. In contrast, the second-order formulation can provide a stable solution for the puncture all the time and shows continuous damping of the constraint violation until the errors at the refinement boundary start to take the lead and amplify the violations.

Any adaptive mesh resolution transition on non-conformal meshes is known to trigger reflections of outgoing waves: We have hanging vertices and hanging faces, i.e. fine grid vertices with less than 2^d adjacent cells on the finest mesh level and faces where the left and the
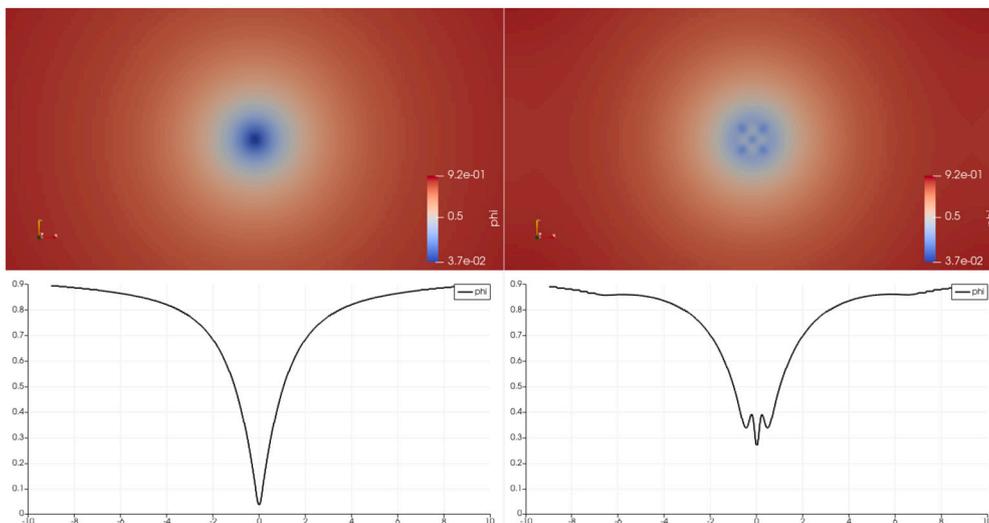
Fig. 12. Comparison of the ϕ field of the single Schwarzschild black hole at timestamp $T = 36$ in the second-order (left) and first-order formulation (right). Color maps illustrate a 2d cutting through the black hole center (upper panels), while the profiles track ϕ along the x-axis (lower panels).
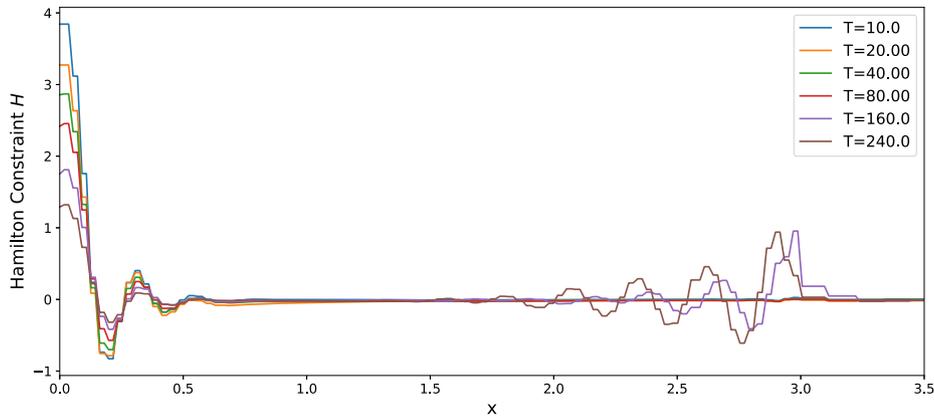
Fig. 13. Violation of the Hamiltonian constraints H for the Schwarzschild black hole along the positive x-axis at $T \in \{10, 20, 40, 80, 160, 240\}$.
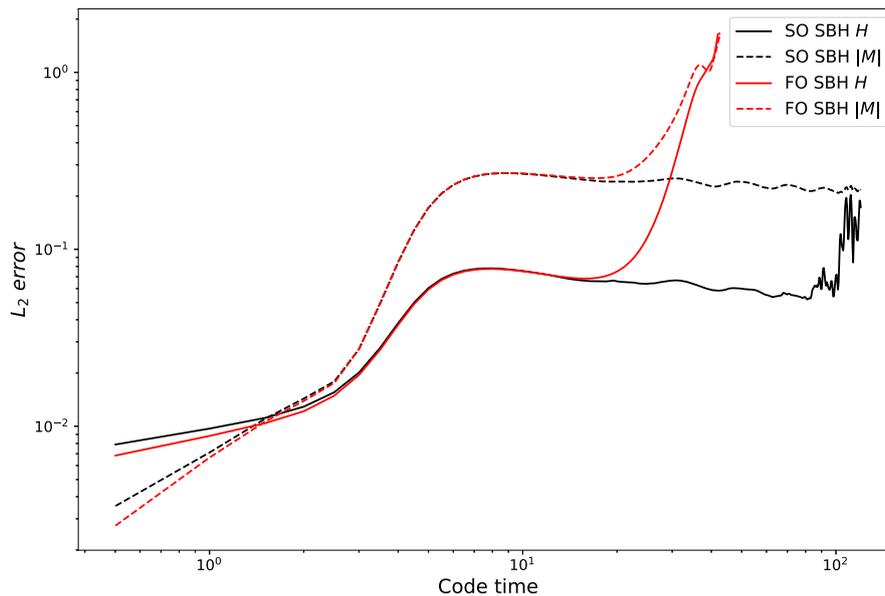
Fig. 14. Evolution of the L_2 error of the Hamiltonian constraint H and the magnitude of Momentum constraint $|M|$ for the single black hole scenario in two formulations. First-order and second-order formulations give nearly identical evolution initially, but eventually the former becomes unstable and crashes. The second-order formulation can run stably until the errors raised on the refinement boundary become non-negligible.

right adjacent cell are of different size. Let a wave travel from a fine mesh patch through a face into an adjacent coarser one. High-frequency modes stemming from the fine-resolution domain cannot be represented on an adjacent coarser grid and hence are reflected back into the direction they are coming from. For dynamic simulations as encountered for moving black holes, these spurious reflections typically do not cause major stability issues. For the stationary black hole, however, we observe that the reflection waves are caught within the fine resolution domain and eventually amplify each other over time. They accumulate. A stationary black hole instead can create standing error waves that grow larger over time.

ExaGRyPE's out-of-the-box linear interpolation and restriction schemes are not powerful enough to handle our highly non-linear PDE system over a long simulation span if we encounter a stationary black hole resolved with mesh refinement. However, as long as black holes move, the KO dissipation is sufficient to compensate for any spurious reflections. A more general solution is to implement higher-order interpolation and restriction schemes, which can provide higher accuracy around the refinement boundary. We have already observed a significant improvement in reflection suppression when we apply a second-order scheme in our recent tests of single black hole spacetimes (not shown). The refinement transition strategies play an important role in

achieving high-quality numerical relativity simulations (e.g., see discussion in [49] and [50]), and we plan to report more comprehensively on how different strategies affect the simulation behavior of ExaGRyPE in future work.

Coupled numerical schemes Finite Volumes rule itself out as a global solver due to their high dissipation and high computing cost and memory footprint vs. accuracy. We need a very fine mesh to compete with a higher-order scheme. However, we recognize that the KO term used to stabilize FD4 injects numerical diffusion, i.e., mimics what we suffer from with FV, is in itself a "magic" stabilization term lacking a physical motivation.

It is hence a natural idea to combine different schemes. We use FD4 throughout the domain. However, we also use the FV scheme around the black hole (Fig. 15, upper panel). It is a localized solver typically covering seven patches around the puncture. Its Dirichlet boundary values are projected from the FD4 solution. Within the overlap region, the FV scheme is considered to be valid and overwrites the corresponding FD4 solution after each time step, served as a limiter. We assume the CFL condition is dominated by

$$\delta t \leq C \frac{h}{p^2}, \qquad (38)$$
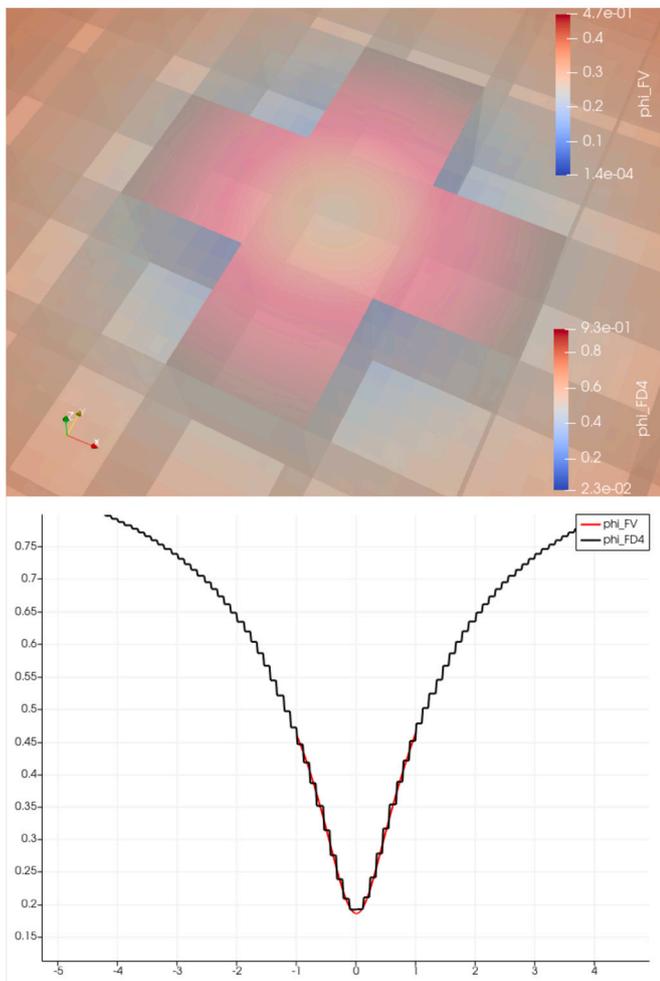
Fig. 15. An FV ExaGRyPE solver is embedded within an FD4 scheme such that it covers the stationary black hold (top). The FV solution on a significantly finer mesh yields a smooth representation of the solution around the black hole and overwrites the behavior of FD4 there. $T = 3$ (bottom).

where we have a polynomial degree of $p = 4$ for the FD4 solver and $p = 1$ for FV. To match the time step sizes, the FV patch size is 4^2 times larger (per dimension) than its FD4 cousin, i.e., contains 16^3 times more volumes per patch.

The profile of the ϕ field of the two solvers shows a stronger smoothness of the FV scheme around the black hole (Fig. 15, lower panel). As the FV solution overwrites FD4 in the center, while the FD4 solution determines the halo data of the FV solution, the solutions appear to overlap. However, only the FV scheme evolves without interference.

The solver coupling (Sections 3.7 and 4.2.4) complements spatial adaptivity with numerical adaptivity, where the numerical scheme changes throughout the domain. It is a realization of the classic a posteriori limiter concept [15]. The Finite Volume patches serve as a tool to resolve the semi-singularities of the solution as well as a sponge sucking up any high-frequency error modes. Its overly dissipative character is now used as a feature rather than a challenge.

While the coupling is an appealing concept to create smooth, stable solutions without the need to use an artificial KO damping, i.e. solely due to first-order numerical principles, several challenges remain: tests with ExaGRyPE suggest that a direct overwrite of the FD4 solution can sometimes yield spurious waves similar to the reflections from the refinement boundary. ExaGRyPE hence offers averaging and lets the "validity" of the FV scheme fade in: Directly around the black hole, the FV scheme dominates. Further away, FD4 yields a valid solution to and the FV solution is not even computed. In-between, we have a buffer zone. Both

solvers run and the eventual result is a weighted average between the two solutions. The averaging shifts from "all FD4" over "predominantly FD4" over to "predominantly FV". While this seems to yield higher-quality data, the precise effects have yet to be studied. It is also an open question of how big the FV region around the black hole has to be chosen overall. If it is too small, the damping effect of the FV region does not cover the unstable domain and we suffer from instabilities again. If it is too large, the numerical dissipation of FV pollutes the result in the region we are interested in. The biggest challenge arising from the coupled approach, however, is the likely difficult load balancing, as almost all computing effort is spent on a few FV patches of a very high resolution. We aim to investigate and explore more the potential of this coupling scheme in future work.

5.3. Rotating binary black hole merger

We finally switch to an equal-mass rotating black hole merger simulation. To construct the correct initial condition that allows for quasi-circular orbits, we utilize one set of the computed parameters from [55] through ExaGRyPE's ported TwoPuncture component. The binary black holes both have a bare (puncture) mass of $m_+ = m_- = 0.46477$ and zero spin, $S_+ = S_- = 0$. Their initial distance is $d = 4M$. The black holes are located at the coordinates $[2M, 0, 0]$ and $[-2M, 0, 0]$ respectively, and rotate on the x-y plane, with the initial linear momentum $P_\pm^i = [0, \pm 0.19243M, 0]$. This setup corresponds to an ADM mass of $M = 0.5$ for each black hole and a total ADM mass of the system $M_{\text{tot}} = 0.98074$.

The average initial lapse

$$\alpha = \frac{1}{2}\left(\frac{1 - \frac{1}{2}\left(m_-/r_-\right) - \frac{1}{2}\left(m_+/r_+\right)}{1 + \frac{1}{2}\left(m_-/r_-\right) + \frac{1}{2}\left(m_+/r_+\right)} + 1\right), \tag{39}$$

with vanished initial shift is used. Otherwise, we preserve the black hole settings, i.e. $\kappa_1 = 0.1, \kappa_2 = 0, \kappa_3 = 0.5, e = c = \tau = 1.0$ and $\mu = 0.2$. The gamma driver gauge condition uses the parameters $f = 0.75$ and $\eta = 1$. Compared to the FD4 setup, no modifications are made to the KO coefficient (8) and the CFL ratio (0.1). We also use the RK1 temporal scheme for this scenario, and convergence tests have confirmed that we achieve the expected order ($\sim h^1$) of convergence across different resolutions with it (test not shown).

We employ a domain of $[-12M, 12M]^3$ with three levels of SMR. The domain gets refined at radius $r = 9M$ and again at $r = 5M$ around the center of the coordinate system, where we assume the black holes to merge. We obtain FD spacings at each level of $[0.333M, 0.111M, 0.037M]$. Sommerfeld boundary conditions are used. We run both the first-order and second-order formulation for this test, taking into account previously described stability challenges for the former once the black holes have merged.

The binary black holes complete one-and-a-half circular orbits before they merge at a code time of $T \approx 70$. The system continues to evolve as a single black hole until $T \approx 120$, where we terminate the simulation. The rotation and merging processes in the simulation are relatively smooth and subject to only limited violations of the constraints (Figs. 16 and 17). As the punctures are rotating, the shift vector exhibits a spiral pattern. Its reversed direction at the puncture location indicates the velocity direction of the puncture, i.e., we can use it to inform ExaGRyPE's tracers from Section 3.8 (Fig. 18). The spiral pattern in the post-merge stage shows that the remnant of the merged black holes carries a spin. Besides the actual physical data, some fluctuation starts to creep into the simulation. It stems from the resolution transitions and gets magnified in the post-merge phase. In line with previous experiments, the first-order formulation encounters instability after merging and soon runs into a crash. However, it can properly resolve the rotation of black holes before that. The dynamic behavior of the first-order and second-order scheme are presented in Fig. 19, both schemes have the same trajectories until the approaching punctures accumulate errors and start to affect the
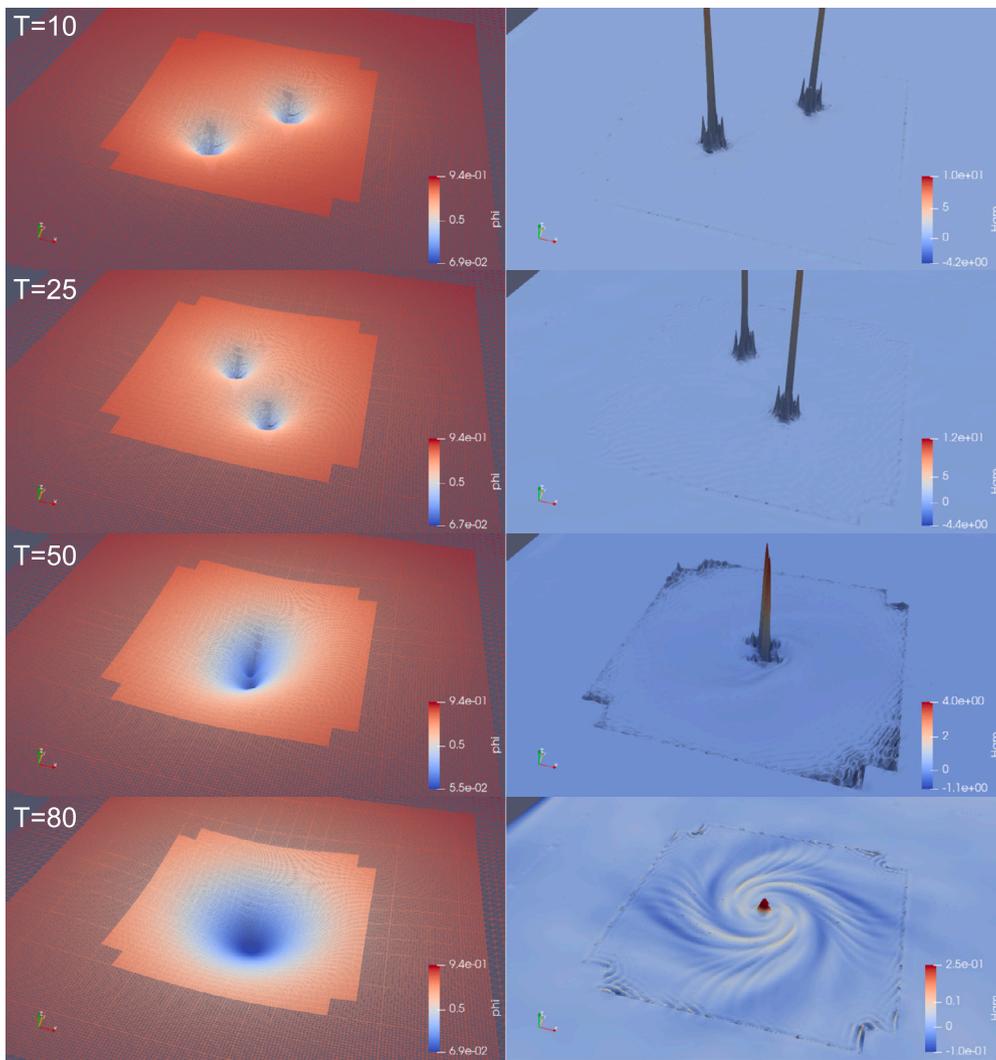
Fig. 16. The snapshots of the conformal factor ϕ and the violation of the Hamiltonian constraints H at time stamps $T \in \{10, 25, 50, 80\}$ over three-dimensional warped wireframes and surfaces, respectively.

evolution. Furthermore, Fig. 20 shows the evolution of the L_2 error of the Hamiltonian constraint H and the magnitude of Momentum constraint $|M|$ for both formulations. The first-order formulation can not stabilize the post-merge remnant as it degenerates back to a static single black hole. In contrast, the second-order formulation remains stable and shows a continuous damping of the constraints violation.

To further validate the accuracy of our code, we conducted a benchmark comparison with GRChombo using the same initial physical conditions. Our simulation replicates the same dynamics with the published code, where the black holes complete approximately one and a half orbits before entering the ring-down phase. Minor differences in the puncture positions were observed over time, which may be from the difference in initial lapse conditions—GRChombo uses a conformal-factor-power initial lapse, while we use an averaged initial lapse. Notably, these differences decrease when higher resolutions or higher-order temporal schemes are employed in our simulation, suggesting a qualitative convergence (not shown). A more robust comparison should be made in terms of observables, such as gravitational waveforms. We plan to conduct a more comprehensive study with large-scale benchmarks in future work.

Overall, the rotation and pre-merger phase are accurately captured in our benchmark, and the locations of the punctures are correctly tracked by the (inverse) shift vector guiding the tracers. The simulation of the dynamic behavior is stable, which supports the argument that any constraint violation is "traveling" through the domain, might be reflected at AMR boundaries, but eventually leaves the domain. We avoid the accumulation of numerical errors at specific locations as long as the black holes move.

5.4. Context of experimental studies

Our application successfully completes a rotating binary black hole merger. It is able to handle both the approach phase as well as the actual ringdown. Our studies focus on the introduction of numerical ingredients within ExaGRyPE. They validate that the software is fully operational and prepared for more challenging benchmarks on larger scales, which will be crucial for further verifying its accuracy and performance. Here, we have to combine all ingredients and notably study a larger computational domain to avoid boundary artifacts (cmp. squeeze-in effect in Fig. 19) and obtain realistic measurements. Further stationery tracers (cmp. Section 5.1) then can be used to sample the solution on a huge sphere around the area of interest. With such samples, we can finally calculate ψ_4 and its mode quantities.

Our data illustrates how delicate CCZ4 simulations react to numerical inaccuracies. We have seen two schools of thought in literature to tackle the intrinsic instabilities which are due to the nonlinear nature of the equations, their complexity, and notably pollution to the long-term quality of the numerical approximation: The first school arti-
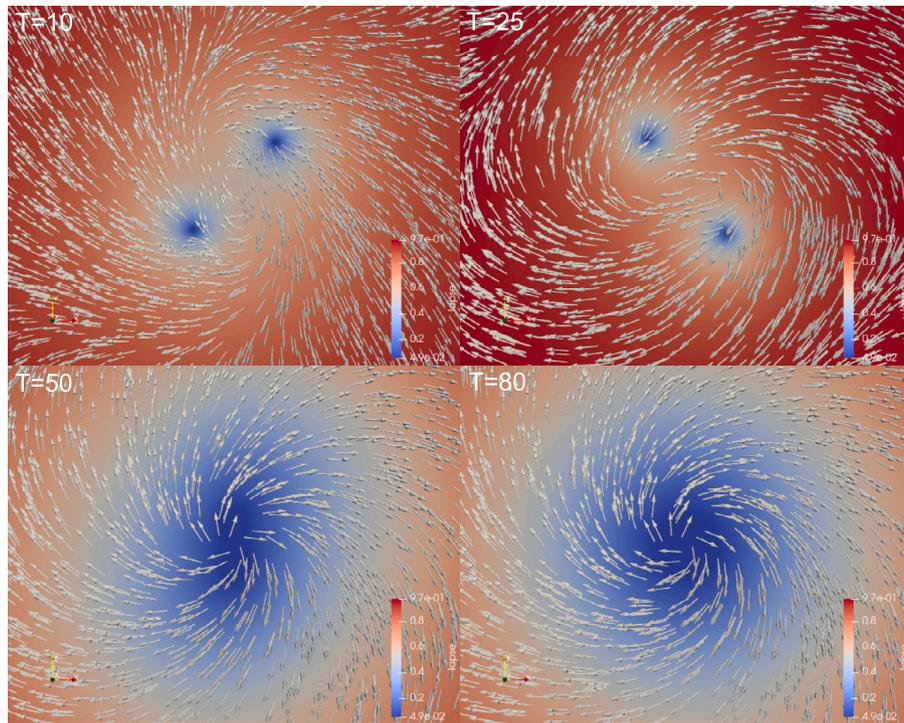
Fig. 17. Snapshots of the gauge quantities, lapse α and shift vector β^i, for Fig. 16 as color map or vector field respectively.
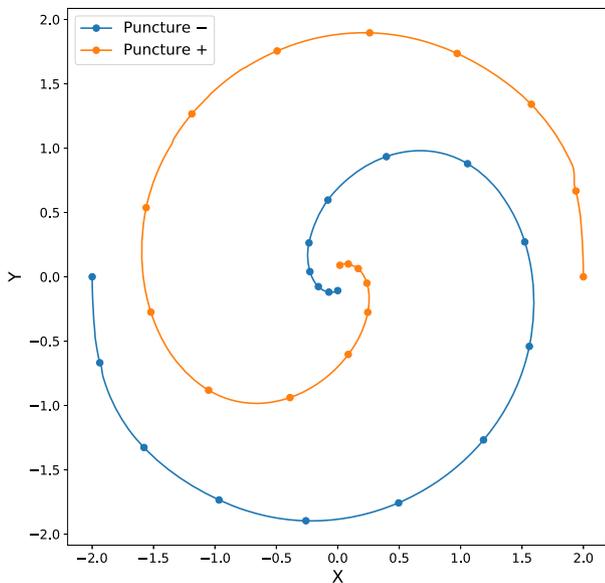
Fig. 18. The trajectories of the two punctures in the binary black hole merger test. The two black holes $\pm$ are initially located at $[x, y, z] = [2M, 0, 0], [-2M, 0, 0]$. The trajectories for the final ring-down stage were removed for clarity. The kink observed in the initial stage of the + puncture arises from a minor inconsistency in data output as the tracer transitions between different subpartitions managed by separate MP threads. The symmetric trajectories that follow confirm that there is no underlying computational error.

ficially damps equation components such that instabilities materializing in high-frequency solution components diffuse away. Kreiss-Oliger dissipation stands in this tradition, and we implemented it for both first-order and second-order formulation. We notice that these additional terms lack, to the best of our knowledge, physical motivation. Groups seem to pick the damping calibration depending on their simulation setup, but systematic studies on its impact are rare. Typically, we find papers lacking systematic studies if this artificial term can pollute the long-term

solution as well for some setups, e.g., with strong shock fronts. It remains subject of future work to assess to which degree the magnitude of the KO dissipation and other damping components in the evolution system play a pivotal role in the quality of the simulation outcome, i.e., if larger or certain choices introduce artificial long-term damping similar to the effects we documented for FV.

The other school of thought uses traditional numerical schemes (a limiter) to damp out instabilities. A posteriori limiting as used in ADER-DG methods [15] stands in this tradition. However, we notice that the subgrid Finite Volume method that we use as a posteriori limiter is very costly, and we are not aware of systematic studies on how to couple the limiter spatially and temporally into the domain to avoid global over-dissipation.

Our data cannot conclusively answer if first-order or second-order formulations of the underlying Einstein equations are superior. The second-order formulations fit more naturally to the PDEs of the numerical relativity and it is widely used by codes of other groups with Finite Difference kernels. In our benchmarks, the second-order formulation prevents error accumulations in the derivative calculation and thus provides better stability in static black hole systems. Meanwhile, first-order formulations allow us to provide more generic implementations to deploy new solvers–indeed we can drop them into FV and DG formulations quite naturally—and we can realize them with smaller data overlaps and hence less synchronization effort. The halos for example can be chosen smaller, and we do not need additional auxiliary variable reconstruction steps. This is an important property for future large-scale simulations, as data movements dominate the compute cost. Indeed, our second-order ExaGRyPE variant is about a factor of two slower than its first-order cousin.

ExaGRyPE is not a finished software product but a tool suite offering some important astrophysical simulation tools. Several sophisticated ingredients are not yet available with this release paper yet might have a substantial impact on simulation outcomes. Notably, the effects of higher-order interpolation along the AMR resolution boundaries have to be studied. We may assume that coefficient optimization as described in [56] yields a smoother and more stable resolution transition in our code which might suppress the error accumulation for quasi-stationary
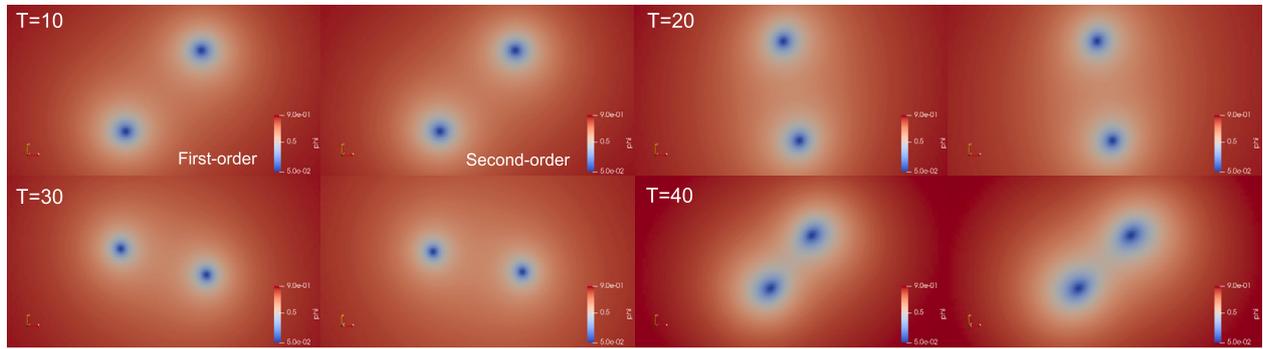
Fig. 19. Evolution of the ϕ field in the central region at $t \in \{10, 20, 30, 40\}$. We compare the data stemming from a first-order formulation (left) to the second-order formulation (right).
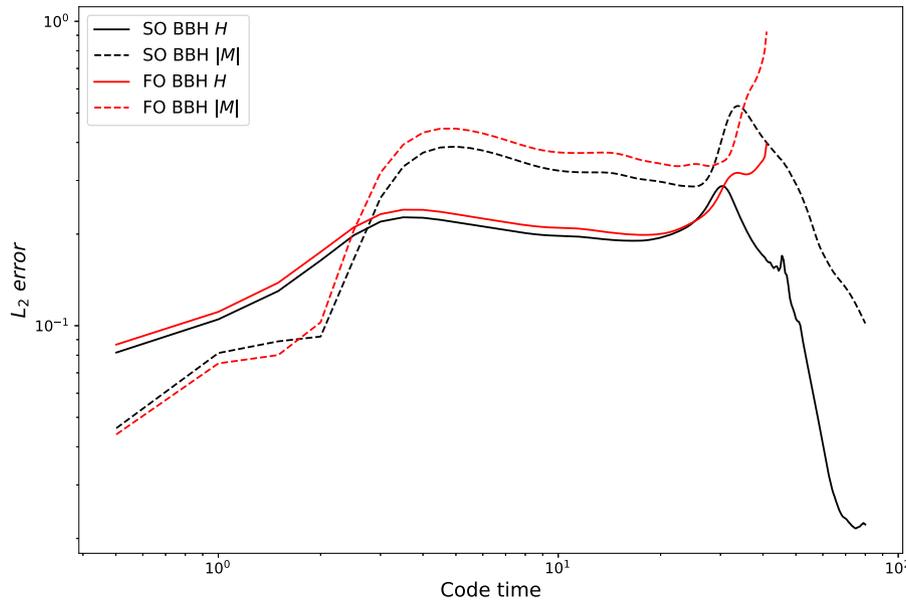
Fig. 20. Evolution of the L_2 error of the Hamiltonian constraint H and the magnitude of Momentum constraint $|M|$ for the binary black hole scenario for both formulations. The second-order formulation is stable throughout the evolution and shows a continuous damping of the violations after the merge. The first-order formulation can capture the dynamics of black holes correctly and yields only a slight increase in error during the rotation phase. However, it runs into a crash soon after the merge, when the system degenerates back to a single black hole scenario.

setups. For the dynamic phases, we have yet to study the impact of dynamic adaptive mesh refinement on the solution quality, while techniques such as manual curl cleaning [51] might help us to make the simulation overall more stable and accurate.

ExaGRyPE provides the infrastructure to study various solver variants and arrangements of ingredients. Its clear separation of concerns allows us to inject new components easily. Notably, we can even compose and couple them freely. It is subject of future work to study the impact of many numerical techniques, to understand them, and to construct simulation codes that are long-term stable and yield high-quality outputs. Our initial data suggests that this might be best achieved through a dynamic combination of ingredients and by giving up on the concept of a "one solver for numerical astrophysics". It might be better to switch between different solvers and numerical techniques, balancing between computational efficiency (first-order vs. second-order) and stability (higher order vs. low order discretizations).

6. Discussion and conclusion

Our work introduces ExaGRyPE, a suite of solvers and related code modules designed for conducting numerical relativity simulations on ExaHyPE 2. The latter is a rewrite of the ExaHyPE PDE solver engine from [13]. The rewrite has been motivated by the urge that we wanted

to design a simulation software base which realizes a clear separation of concerns on all levels and yields a composable software stack such that we can relatively quickly study new physical and numerical approaches in computational astrophysics. Our presentation focuses on the software design in the context of this application domain.

The software design of ExaGRyPE is guided by the concept of lowering, which is inspired by the compiler construction community: we aim for a very high-level API exposing the software to the user in Python. It allows the user to specify *what* is to be computed. This high-level specification is then successively lowered into a numerical representation of the compute steps and from hereon into technical views which eventually are mapped onto plain C++ code. The mantra here is that the lowering and the underlying code base determine *where* (on a parallel computer), *when* (in which order) and *how* (generic provision of some numerical ingredients) the specification is realized in the source code.

To implement this vision and to showcase the applicability and maturity of the design, the ExaGRyPE endeavor had to add further building blocks to the underlying ExaHyPE engine, and we had to run an extensive set of numerical experiments. The experiments demonstrate that our code is fully operational, but also raise further questions, notably around the numerical stability of any composition of ingredients and the philosophical question if first- or second-order formulations are superior. Many aspects of these questions are intrinsically tied to this

area of research and have been documented, either implicitly or explicitly, by other codes and groups, too. We consider it a strength of ExaGRyPE that we can quite flexibly study and swap different methodological ingredients to address these fundamental research questions, although the present work does not yet provide answers to many of them. The unique solver coupling features of ExaGRyPE make it an ideal tool for more advanced numerical relativity simulations. For instance, it allows the combination of a finite volume scheme for the matter field with a finite difference scheme for the spacetime background, leading to higher-quality simulations of relativistic hydrodynamics in scenarios such as black hole accretion and neutron star spacetimes. In the same way, undergoing extensions to the particle toolbox—currently used for the tracers—could enable general-relativistic smoothed particle hydrodynamics (SPH) simulations [57–59].

The success of any astrophysical code hinges upon the question of to which degree the code is able to scale up and to exploit modern machinery efficiently. They allow users to conduct astrophysics simulations with larger scale and higher accuracy. Our presentation largely neglects these aspects and focuses on software and domain challenges. However, we note that our strict separation of concerns and lowering approach yields a software architecture where supercomputing considerations can generically enter the software design in lower abstraction levels, i.e. performance engineering can be realized without domain expertise. Further to that, our lowering formalism uses the notion of tasks, i.e. expresses concurrency explicitly. We have started to publish work on how to exploit these properties—for example by offloading sets of tasks to GPUs [20]—and will make supercomputing aspects subject of future publications using the ExaGRyPE software architecture.

CRediT authorship contribution statement

Han Zhang: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Baojiu Li:** Writing – review & editing, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Tobias Weinzierl:** Writing – review & editing, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Conceptualization. **Cristian Barrera-Hinojosa:** Writing – review & editing, Validation, Investigation, Formal analysis.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Experiments

The following section describes how to reproduce our results. We assume that we rerun the experiments against the latest version of Peano which contains the latest version of ExaHyPE. However, all ExaGRyPE experiments here are made with the tag version `2024ExaGRyPE`.

The code base is available directly from our gitlab [60]. Peano provides support for cmake and the autotools. We present the autotools configuration here (Algorithm 8).

As Peano is a generic framework used for multiple projects, we have to enable ExaGRyPE-specific extensions plus their dependencies manually. In the code snippet, we enable the block-structured data management, ExaHyPE and the particle toolbox which provides us with tracer facilities. Setups might want to add `-with-multithreading=x`, where `x` can be either `omp` (OpenMP), `tbb` Intel's Threading Building Blocks, or `cpp` (native C++ threading). Further to that, multi-node experiments require a build with `-with-mpi=mpiicpc`, where the argument is the MPI C++ compiler wrapper to be used. GPU extensions are enabled via `-with-gpus=omp` if OpenMP 5 offloading is to be used [20]. The switch `-with-gpus=sycl` enables offloading via SYCL [19].

Specific `CXXFLAGS`, `LDFLAGS` and `LIBS` complement the configuration, and vary depending on your working platform. For initialization of current black hole experiments, the GSL library [61] math library is needed to support the TwoPuncture module from EINSTEINTOOLKIT. The `make` instruction in Algorithm 8 builds a set of static libraries which will be used by every ExaGRyPE application.

Every ExaGRyPE application is realized through a Python script, which uses ExaHyPE's Python API lowering into Peano's Python modules. They then generate exclusively C++ code plus a makefile. The C++ code is linked against the static Peano and ExaHyPE modules as well as the astrophysics-specific ExaGRyPE source files. We eventually end up with one big executable reflecting the ExaGRyPE script settings: Peano realizes a strict "we create one executable comprising all required input settings" paradigm, i.e. from hereon no further input data are required. The generated makefile parses the `configure` settings and adopts all the configurations settings from there, i.e. we do not have to re-specify compilers, features or compiler and linker settings.

A.1. Gauge wave

The gauge wave benchmark simulates a simple standing wave in a domain with periodic boundary conditions (Algorithm 9). This setup utilizes the first order formulation, and by adding `-so` flag one can switch to the second order formulation. The setup as specified uses tracer markers at script-specified points which track the solution in these points. The outcome is a single text table file (`.csv`). We ship matplotlib scripts with the repository, which will produce "seismograms" at the tracer points.

A.2. Single black hole

The Algorithm 10 builds a small benchmark of the single black hole scenario, which one can use for tests and profiling. The output is very likely to be unphysical as the resolution is quite low.

The Algorithm 11 constructs the standard static single black hole simulation used to produce snapshots and results in the main content. This setup requires a significant amount of computational resources.

Algorithm 8 Clone repository, create autotools environment and build Peano's and ExaHyPE's core libraries.

```
git clone https://gitlab.lrz.de/hpcsoftware/Peano
cd Peano
libtoolize; aclocal; autoconf; autoheader; cp src/config.h.in
automake --add-missing
./configure –enable-blockstructured –enable-particles –enable-exahype
make
```

Algorithm 9 The Gauge Wave benchmark.

```
cd application/exahype2/ccz4
python3 ccz4.py -impl fd4-rk1-adaptive -s gauge -maxh 0.05 -minh 0.05 -ps 3 -plt 0.1 -et 0.5 -exn test –domain_r 0.5 –ReSwi 0 -cfl 0.1 –KOSigma 8.0
-tracer 1                                                                                                                  ▷ Invokes make internally
./peano4_test
pvpython ../../../python/peano4/visualisation/render.py solution-CCZ4.peano-patch-file
```

Algorithm 10 The Single Black Hole benchmark.

```
cd application/exahype2/ccz4
python3 ccz4.py -impl fd4-rk1-adaptive -s single-puncture -maxh 0.05 -minh 0.05 -ps 3 -plt 0 -et 0.01 -exn test –domain_r 0.5 –ReSwi 1 -cfl 0.1 –KOSigma 8.0 -so
./peano4_test
pvpython ../../../python/peano4/visualisation/render.py solution-CCZ4.peano-patch-file
```

Algorithm 11 The Single Black Hole Productive Run.

```
cd application/exahype2/ccz4
python3 ccz4.py -ext adm -impl fd4-rk1-adaptive -s single-puncture -maxh 0.4 -minh 0.04 -ps 6 -plt 0.5 -et 240 -exn sbh –domain_r 9.0 –ReSwi 7 -cfl 0.1 -outdir
/path/to/your/storage/ –KOSigma 8.0 -sommerfeld -so
./peano4_sbh
cd /path/to/your/storage/
pvpython ../../../python/peano4/visualisation/render.py solution-CCZ4.peano-patch-file
```

Algorithm 12 The binary Black Hole Productive Run.

```
cd application/exahype2/ccz4
python3 ccz4.py -impl fd4-rk1-adaptive -s two-punctures -maxh 0.4 -minh 0.04 -ps 8 -plt 0.5 -et 120 -exn bbh –domain_r 12.0 –ReSwi 6 -cfl 0.1 -outdir
/path/to/your/storage/ –KOSigma 8.0 –BBHType 2 -sommerfeld -so -ext Psi4
./peano4_bbh
cd /path/to/your/storage/
pvpython ../../../python/peano4/visualisation/render.py solution-CCZ4.peano-patch-file
```

A.3. Binary black hole

Algorithm 12 builds an application for production runs on the rotating binary black hole scenario, used to produce data for the main content. Similarly, this setup is computational-resource demanding.

Appendix B. Spacetime foliation and evolution equations

This appendix gives an overview of the problem formulation of ExaGRyPE and also lists all the evolution equations used in the code.

Throughout the paper, we have chosen the $(-,+,+,+)$ metric signature and followed the standard Einstein convention of summing over repeated indices. We use the Latin letters $a, b, \ldots,$ for spacetime indices which run from 0 to 3, following the *abstract index notation*. Tensor components are labeled by Greek letters $\mu, \nu, \ldots,$ and indices running from 1 to 3 are labeled using the middle part of the Latin alphabet $(i, j, k, \ldots)$ as usual. Finally, we will use the geometrized unit system where both the gravitational constant and the speed of light are set to unity, $c = G = 1$.

The evolution equations for the spacetime utilized by ExaGRyPE are from the Z4 formulation of Einstein field equations, where an extra algebra dynamic field Z_a and corresponding damping term are added into the system to enhance stability [23,24]:

$$R_{ab} - \frac{1}{2}g_{ab}R + \nabla_a Z_b + \nabla_b Z_a - g_{ab}\nabla^c Z_c -$$
$$\kappa_1[n_a Z_b + n_b Z_a + \kappa_2 g_{ab} n_c Z^c] = 8\pi T_{ab}. \tag{B.1}$$

To solve the evolution equations, our code uses the standard ADM 3+1 spacetime foliation which cuts the spacetime into a set of three-dimensional spacelike hypersurfaces labeled by a global time scalar field

$t(x^a)$. The spatial metric on those hypersurfaces is named γ_{ab}, and linked to the spacetime metric

$$ds^2 = -\alpha^2 dt^2 + \gamma_{ij}\left(dx^i + \beta^i dt\right)\left(dx^j + \beta^j dt\right), \tag{B.2}$$

where γ_{ij} are the spatial part of γ_{ab}. α and β^i are called lapse functions and shift vector fields, representing four degrees of freedom of coordinates during the evolution. Their relations with the time axis t^a are illustrated in Fig. 2.

The unit normal vector on the spacelike hypersurface n^a is given by

$$n^a := -\alpha g^{ab}\nabla_b t, \tag{B.3}$$

and the extrinsic curvature K_{ab}, a dynamic variable determining the evolution of the spatial structure, can be defined as the Lie derivatives of the spatial metric over n^a:

$$K_{ab} := -\frac{1}{2}\mathcal{L}_{\mathbf{n}}\gamma_{ab}. \tag{B.4}$$

Similar to the spatial metric γ_{ab}, K_{ab} is a spatial tensor. We therefore can only track the evolution of its spatial part, i.e., K_{ij}, in our simulations.

One can already conduct simulations in this stage as $\gamma_{ij}, K_{ij}, \alpha, \beta^i$ with Z^a field form a complete evolution system. We further follow [16] and introduce a conformal factor ϕ and a conformal metric $\tilde{\gamma}_{ij}$, such that

$$\tilde{\gamma}_{ij} := \phi^2 \gamma_{ij}, \ \phi = [\det(\gamma_{ij})]^{-1/6}. \tag{B.5}$$

The second equation guarantees that $\tilde{\gamma}_{ij}$ has a unit determinant. We also decompose the extrinsic curvature into its trace part $K = K_{ij}\gamma^{ij}$ and traceless part A_{ij}, using the same scaling as for the metric:

$$\tilde{A}_{ij} := \phi^2 A_{ij} = \phi^2 \left(K_{ij} - \frac{1}{3} K \gamma_{ij} \right). \tag{B.6}$$

The Z^a represents four additional evolving variables, which are divided into its temporal component $\Theta := -n_a Z^a = \alpha Z^0$ and spatial component $Z^i := \gamma^a_b Z^b$. The latter is included in the evolution system by adding to the contracted Christoffel symbol:

$$\hat{\Gamma}^i := \tilde{\Gamma}^i + 2\tilde{\gamma}^{ij} Z_j, \quad \tilde{\Gamma}^i := \tilde{\gamma}^{jk} \tilde{\Gamma}^i_{jk}, \tag{B.7}$$

where $\tilde{\Gamma}^i_{jk}$ is the Christoffel symbol for the conformal metric $\tilde{\gamma}_{ij}$

$$\tilde{\Gamma}^i_{jk} = \frac{1}{2} \tilde{\gamma}^{il} \left(\partial_j \tilde{\gamma}_{kl} + \partial_k \tilde{\gamma}_{jl} - \partial_l \tilde{\gamma}_{jk} \right). \tag{B.8}$$

For the first-order formulation, ExaGRyPE further introduces the following auxiliary variables:

$$A_i := \partial_i \alpha, \quad B_k^i := \partial_k \beta^i,$$
$$D_{kij} := \frac{1}{2} \partial_k \tilde{\gamma}_{ij}, \quad P_i := \partial_i \phi. \tag{B.9}$$

These variables satisfy the natural second-order constraints as

$$\partial_k A_i - \partial_i A_k = 0, \quad \partial_k B_l^i - \partial_l B_k^i = 0,$$
$$\partial_k D_{lij} - \partial_l D_{kij} = 0, \quad \partial_k P_i - \partial_i P_k = 0. \tag{B.10}$$

Because $\tilde{A}_{ij}$ is traceless $\gamma^{ij} \tilde{A}_{ij} = 0$ and $\tilde{\gamma} = \det(\tilde{\gamma}_{ij}) = 1$, we also have

$$\partial_k \left(\tilde{\gamma}^{ij} \tilde{A}_{ij} \right) = \partial_k \tilde{\gamma}^{ij} \tilde{A}_{ij} + \tilde{\gamma}^{ij} \partial_k \tilde{A}_{ij} = 0, \tilde{\gamma}^{ij} D_{kij} = 0. \tag{B.11}$$

The formulation of the evolution system in ExaGRyPE is given as the following nine equations for primary variables:

$$\partial_t \tilde{\gamma}_{ij} = \beta^k 2 D_{kij} + \tilde{\gamma}_{ki} B_j^k + \tilde{\gamma}_{kj} B_i^k - \frac{2}{3} \tilde{\gamma}_{ij} B_k^k - $$
$$2\alpha \left(\tilde{A}_{ij} - \frac{1}{3} \tilde{\gamma}_{ij} \operatorname{tr} \tilde{A} \right) - \tau^{-1} (\tilde{\gamma} - 1) \tilde{\gamma}_{ij}, \tag{B.12}$$

$$\partial_t \alpha = \beta^k A_k - \alpha^2 g(\alpha) \left(K - K_0 - 2c\Theta \right), \tag{B.13}$$

$$\partial_t \beta^i = \beta^k B_k^i + f b^i, \tag{B.14}$$

$$\partial_t \phi = \beta^k P_k + \frac{1}{3} \phi \left(\alpha K - B_l^l \right), \tag{B.15}$$

$$\partial_t \tilde{A}_{ij} - \beta^k \partial_k \tilde{A}_{ij} - \phi^2 \left[-\nabla_i \nabla_j \alpha + \alpha \left(R_{ij} + \nabla_i Z_j + \nabla_j Z_i \right) \right]^{\text{TF}}$$
$$= \tilde{A}_{ki} B_j^k + \tilde{A}_{kj} B_i^k - \frac{2}{3} \tilde{A}_{ij} B_k^k + \alpha \tilde{A}_{ij} (K - 2\Theta c) - $$
$$2\alpha \tilde{A}_{il} \tilde{\gamma}^{lm} \tilde{A}_{mj}, \tag{B.16}$$

$$\partial_t K - \beta^k \partial_k K + \nabla^i \nabla_i \alpha - \alpha \left(R + 2\nabla_i Z^i \right)$$
$$= \alpha K (K - 2\Theta c) - 3\alpha \kappa_1 \left(1 + \kappa_2 \right) \Theta, \tag{B.17}$$

$$\partial_t \Theta - \beta^k \partial_k \Theta - \frac{1}{2} \alpha e^2 \left(R + 2\nabla_i Z^i \right)$$
$$= \frac{1}{2} \alpha e^2 \left(\frac{2}{3} K^2 - \tilde{A}_{ij} \tilde{A}^{ij} \right) - \alpha \Theta K c - Z^i A_i$$
$$- \alpha \kappa_1 \left(2 + \kappa_2 \right) \Theta, \tag{B.18}$$

$$\partial_t \hat{\Gamma}^i - \beta^k \partial_k \hat{\Gamma}^i - 2\alpha \tilde{\gamma}^{ki} \partial_k \Theta - \tilde{\gamma}^{kl} \partial_{(k} B_{l)}^i - s \frac{1}{3} \tilde{\gamma}^{ik} \partial_{(k} B_{l)}^l$$
$$- 2\alpha \tilde{\gamma}^{ik} \tilde{\gamma}^{nm} \partial_k \tilde{A}_{nm} + \frac{4}{3} \alpha \tilde{\gamma}^{ij} \partial_j K = \frac{2}{3} \tilde{\Gamma}^i B_k^k - \tilde{\Gamma}^k B_k^i$$
$$+ 2\alpha \left(\tilde{\Gamma}^i_{jk} \tilde{A}^{jk} - 3\tilde{A}^{ij} \frac{P_j}{\phi} \right) - 2\alpha \tilde{\gamma}^{ki} \left(\Theta \frac{A_k}{\alpha} + \frac{2}{3} K Z_k \right)$$
$$- 2\tilde{A}^{ij} A_j - 4\alpha \tilde{\gamma}^{ik} D_k^{nm} \tilde{A}_{nm} - 2\alpha \kappa_1 \tilde{\gamma}^{ij} Z_j$$
$$+ 2\kappa_3 \left(\frac{2}{3} \tilde{\gamma}^{ij} Z_j B_k^k - \tilde{\gamma}^{jk} Z_j B_k^i \right), \tag{B.19}$$

$$\partial_t b^i - \beta^k \partial_k b^i = \partial_t \hat{\Gamma}^i - \beta^k \partial_k \hat{\Gamma}^i - \eta b^i, \tag{B.20}$$

and four equations for the auxiliary variables, if the first-order scheme is used:

$$\partial_t A_k - \beta^l \partial_l A_k + \alpha^2 g(\alpha) \left(\partial_k K - \partial_k K_0 - 2c\partial_k \Theta \right)$$
$$+ \alpha g(\alpha) \tilde{\gamma}^{nm} \partial_k \tilde{A}_{nm} = B_k^l A_l + 2\alpha g(\alpha) D_k^{nm} \tilde{A}_{nm} \tag{B.21}$$
$$- \left[2\alpha g(\alpha) + \alpha^2 g'(\alpha) \right] \left(K - K_0 - 2c\Theta \right) A_k,$$

$$\partial_t B_k^i - \beta^l \partial_l B_k^i - f \partial_k b^i + \alpha^2 \mu \frac{\tilde{\gamma}^{ij}}{\phi} \left(\partial_k P_j - \partial_j P_k \right)$$
$$- \alpha^2 \mu \tilde{\gamma}^{ij} \tilde{\gamma}^{nl} \left(\partial_k D_{ljn} - \partial_l D_{kjn} \right) = B_k^l B_l^i, \tag{B.22}$$

$$\partial_t D_{kij} - \beta^l \partial_l D_{kij} + s \left(-\frac{1}{2} \tilde{\gamma}_{mi} \partial_{(k} B_{j)}^m - \frac{1}{2} \tilde{\gamma}_{mj} \partial_{(k} B_{i)}^m \right.$$
$$\left. + \frac{1}{3} \tilde{\gamma}_{ij} \partial_{(k} B_{m)}^m \right) + \alpha \partial_k \tilde{A}_{ij} - \alpha \frac{1}{3} \tilde{\gamma}_{ij} \tilde{\gamma}^{nm} \partial_k \tilde{A}_{nm}$$
$$= B_k^l D_{lij} + B_j^l D_{kli} + B_i^l D_{klj} - \frac{2}{3} B_l^l D_{kij} \tag{B.23}$$
$$- \alpha \frac{2}{3} \tilde{\gamma}_{ij} D_k^{nm} \tilde{A}_{nm} - A_k \left(\tilde{A}_{ij} - \frac{1}{3} \tilde{\gamma}_{ij} \operatorname{tr} \tilde{A} \right),$$

$$\partial_t P_k - \beta^l \partial_l P_k - \frac{1}{3} \phi \left(\alpha \partial_k K - \partial_{(k} B_{l)}^l \right) - \frac{1}{3} \alpha \phi \tilde{\gamma}^{nm} \partial_k \tilde{A}_{nm}$$
$$= B_k^l P_l + \frac{1}{3} \left(\alpha K - B_l^l \right) P_k + \frac{1}{3} \phi K A_k - \frac{2}{3} \alpha \phi D_k^{nm} \tilde{A}_{nm}. \tag{B.24}$$

Most of the symbols already appear in the context above and are self-explaining. The TF index means the Trace-Free part of the quantities, i.e., $R_{ij}^{\text{TF}} = R_{ij} - \frac{1}{3} \gamma_{ij} \gamma^{kl} R_{kl}$. $\operatorname{tr} \tilde{A} = \gamma^{ij} \tilde{A}_{ij}$ is the trace of the conformal traceless extrinsic curvature. There are several red terms added to the evolution systems, using the constraints (B.11) above for a more symmetric characteristic matrix [15]. Those terms are only employed in the first formulation in ExaGRyPE and are removed in the second-order formulation. The $g(\alpha)$ function is the one defined in the slicing condition in Section 2.1, and the "1+log" slicing corresponds to $g(\alpha) = 2/\alpha$. Several new parameters are also introduced for numerical optimization:

- τ represents the relaxation time to enforce the algebraic constraints ($\det \tilde{\gamma}_{ij} = 1$);
- e is the cleaning speed for Hamiltonian constraint, following the idea of [62];
- $\mu > 0$ in equation (B.22) determines the effect of the terms from the constraints;
- c controls the contribution from some algebraic source terms in Z4 systems. Its default value is 1 when the original CCZ4 is used [16].

The standard hyperbolic Gamma driver shift gauge is given above but for some physical scenarios, we also provide the option of the static (zero) shift condition.

The evolution equations above are not completed as we abbreviate some key quantities for readability. The whole system is closed with the evolving quantities using the following list of relations:

$$\partial_k \tilde{\gamma}^{ij} = -2\tilde{\gamma}^{in} \tilde{\gamma}^{mj} D_{knm} := -2D_k^{ij}, \tag{B.25}$$

$$\tilde{\Gamma}^k_{ij} = \tilde{\gamma}^{kl} \left(D_{ijl} + D_{jil} - D_{lij} \right), \tag{B.26}$$

$$\partial_k \tilde{\Gamma}^m_{ij} = -2D_k^{ml} \left(D_{ijl} + D_{jil} - D_{lij} \right) + \tilde{\gamma}^{ml} (\partial_{(k} D_{i)jl}$$
$$+ \partial_{(k} D_{j)il} - \partial_{(k} D_{l)ij}), \tag{B.27}$$

$$\Gamma^k_{ij} = \tilde{\Gamma}^k_{ij} - \frac{1}{\phi} \tilde{\gamma}^{kl} \left(\tilde{\gamma}_{jl} P_i + \tilde{\gamma}_{il} P_j - \tilde{\gamma}_{ij} P_l \right), \tag{B.28}$$

$$\partial_k \Gamma^m_{ij} = -2D_k^{ml} \left(D_{ijl} + D_{jil} - D_{lij} \right)$$
$$+ \tilde{\gamma}^{ml} \left[\partial_{(k} D_{i)jl} + \partial_{(k} D_{j)il} - \partial_{(k} D_{l)ij} \right]$$
$$+ \frac{2}{\phi} D_k^{ml} \left(\tilde{\gamma}_{jl} P_i + \tilde{\gamma}_{il} P_j - \tilde{\gamma}_{ij} P_l \right)$$
$$- \frac{2}{\phi} \tilde{\gamma}^{ml} \left(D_{kjl} P_i + D_{kil} P_j - D_{kij} P_l \right) \tag{B.29}$$
$$- \frac{1}{\phi} \tilde{\gamma}^{ml} \left[\tilde{\gamma}_{jl} \partial_{(k} P_{i)} + \tilde{\gamma}_{il} \partial_{(k} P_{j)} - \tilde{\gamma}_{ij} \partial_{(k} P_{l)} \right]$$
$$+ \frac{1}{\phi^2} \tilde{\gamma}^{ml} \left(\tilde{\gamma}_{jl} P_i P_k + \tilde{\gamma}_{il} P_j P_k - \tilde{\gamma}_{ij} P_k P_l \right),$$

$$R^m_{ikj} = \partial_k \Gamma^m_{ij} - \partial_j \Gamma^m_{ik} + \Gamma^l_{ij} \Gamma^m_{lk} - \Gamma^l_{ik} \Gamma^m_{lj}, \quad R_{ij} = R^m_{imj}, \tag{B.30}$$

$$\nabla_i \nabla_j \alpha = \partial_{(i} A_{j)} - \Gamma^k_{ij} A_k, \tag{B.31}$$

$$\partial_k \tilde{\Gamma}^i = -2 D^{jl}_k \tilde{\Gamma}^i_{jl} + \tilde{\gamma}^{jl} \partial_k \tilde{\Gamma}^i_{jl}, \tag{B.32}$$

$$Z_i = \frac{1}{2} \tilde{\gamma}_{ij} \left(\hat{\Gamma}^j - \tilde{\Gamma}^j \right), \quad Z^i = \frac{1}{2} \phi^2 \left(\hat{\Gamma}^i - \tilde{\Gamma}^i \right), \tag{B.33}$$

$$\nabla_i Z_j = D_{ijl} \left(\hat{\Gamma}^l - \tilde{\Gamma}^l \right) + \frac{1}{2} \tilde{\gamma}_{jl} \left(\partial_i \hat{\Gamma}^l - \partial_i \tilde{\Gamma}^l \right) - \Gamma^l_{ij} Z_l, \tag{B.34}$$

$$R + 2 \nabla_k Z^k = \phi^2 \tilde{\gamma}^{ij} \left(R_{ij} + \nabla_i Z_j + \nabla_j Z_i \right). \tag{B.35}$$

Notice the second-order constraints (B.10) are used to "symmetrize" the spatial derivatives of the auxiliary variables in some equations and relations. In code practice, we also use a simplified expression for the modified Ricci tensor $R_{ij} + 2\nabla_{(i} Z_{j)}$ from [50]:

$$
\begin{aligned}
R_{ij} + 2\nabla_{(i} Z_{j)} = \\
&- \frac{1}{2} \tilde{\gamma}^{kl} \partial_k \partial_l \tilde{\gamma}_{ij} + \tilde{\gamma}_{k(i} \partial_{j)} \hat{\Gamma}^k + \frac{1}{2} \hat{\Gamma}^k \partial_k \tilde{\gamma}_{ij} \\
&+ \tilde{\gamma}^{lm} \left(\tilde{\Gamma}^k_{li} \tilde{\Gamma}_{jkm} + \tilde{\Gamma}^k_{lj} \tilde{\Gamma}_{ikm} + \tilde{\Gamma}^k_{im} \tilde{\Gamma}_{klj} \right) \\
&+ \frac{1}{\phi} \left(\bar{\nabla}_i \bar{\nabla}_j \phi + \tilde{\gamma}_{ij} \tilde{\gamma}^{kl} \bar{\nabla}_k \bar{\nabla}_l \phi \right) - \frac{2}{\phi^2} \tilde{\gamma}_{ij} \tilde{\gamma}^{kl} \partial_k \phi \partial_l \phi \\
&+ \frac{2}{\phi^3} Z^k \left(\tilde{\gamma}_{ik} \partial_j \phi + \tilde{\gamma}_{jk} \partial_i \phi - \tilde{\gamma}_{ij} \partial_k \phi \right),
\end{aligned}
\tag{B.36}
$$

with $\bar{\nabla}_i \bar{\nabla}_k \phi = \partial_i \partial_k \phi - \tilde{\Gamma}^k_{ij} \partial_k \phi$ the second *conformal* covariant derivative of ϕ and $\tilde{\Gamma}_{kij} := \tilde{\gamma}_{kl} \tilde{\Gamma}^l_{ij}$ the newly-defined lowered Christoffel symbol.

There are two extra physical (ADM) constraints can be derived from the 3+1 foliation of (B.1), which are the *Hamiltonian Constraint*

$$H := R + \frac{2}{3} K^2 - \tilde{A}_{ij} \tilde{A}^{ij}, \tag{B.37}$$

and *Momentum Constraints*

$$M^i := \tilde{\gamma}^{kl} \left(\partial_k \tilde{A}_{li} - 2 \tilde{\Gamma}^m_{l(i} \tilde{A}_{k)m} - 3 \tilde{A}_{ik} \frac{P_l}{\phi} \right) - \frac{2}{3} \partial_i K. \tag{B.38}$$

The matter source terms are dropped as we currently focus on the vacuum solution. According to the Bianchi identities, if the constraints are zero at the initial hypersurface (which is guaranteed by the Bowen-York solution), they vanish through the whole evolution [63]. Any deviation from zero can be seen as a result from the numerical errors. Therefore, they are widely used in numerical relativity code as accuracy metrics to assess the quality of the simulations. We examine the violation of the constraints in our simulation tests in Section 5.

Appendix C. ψ_4 in ExaGRyPE

The complex scalar field ψ_4 is written as

$$\psi_4 = {}^{(4)} C_{abcd} k^a \bar{m}^b k^c \bar{m}^d, \tag{C.1}$$

where ${}^{(4)} C_{abcd}$ is the four-dimensional Weyl tensor, and k^a and $\bar{m}^a$ are two members of a null tetrad. A tetrad is a set of four null vectors that form a complete basis, and we name them l^a, k^a, m^a and $\bar{m}^a$. The tetrad is constructed such that the l^a and k^a are radial outgoing and ingoing vectors respectively, and only non-vanishing inner products between them are:

$$-l^a k_a = m^a \bar{m}_a = 1. \tag{C.2}$$

In our code, the unit vectors in three spatial directions are calculated from a Gram-Schmidt orthonormalization of the following vectors [35]:

$$v^i = [-y, x, 0] \rightarrow v^i / \sqrt{L_{vv}}, \tag{C.3}$$

$$u^i = [x, y, z] \rightarrow (u^i - v^i L_{vu}) / \sqrt{L_{uu}}, \tag{C.4}$$

$$
\begin{aligned}
w^i = \gamma^{ij} \epsilon_{jkl} v^k u^l \rightarrow \\
(w^i - v^i L_{vw} - u^i L_{uw}) / \sqrt{L_{ww}},
\end{aligned}
\tag{C.5}
$$

where L represents the inner product of corresponding vectors, i.e., $L_{vu} := \gamma_{ij} v^i u^j$. Combining with the temporal unit vector, $e^a_{\hat{t}} = \left[\alpha^{-1}, -\alpha^{-1} \beta^i \right]$, the tetrad is given as

$$l^a = \frac{1}{\sqrt{2}} [\alpha^{-1}, -\alpha^{-1} \beta^i + u^i], \tag{C.6}$$

$$k^a = \frac{1}{\sqrt{2}} [\alpha^{-1}, -\alpha^{-1} \beta^i - u^i], \tag{C.7}$$

$$m^a = \frac{1}{\sqrt{2}} [0, w^i + i v^i], \tag{C.8}$$

$$\bar{m}^a = \frac{1}{\sqrt{2}} [0, w^i - i v^i]. \tag{C.9}$$

On the other hand, the Weyl tensor can be simplified to [64]

$$\psi_4 = (E_{ij} - i B_{ij}) \bar{m}^i \bar{m}^j, \tag{C.10}$$

where E_{ij} and B_{ij} are the spatial parts of the so-called electric and magnetic parts of the Weyl tensor. Their expression in the Z4 system is

$$E_{ij} = \left(R_{ij} - K^m_i K_{jm} + K_{ij}(K - \Theta) + D_{(i} Z_{j)} - 4\pi S_{ij} \right)^{\mathrm{TF}}, \tag{C.11}$$

$$B_{ij} = \frac{1}{2} \left(\epsilon_{mni} D^m K_j{}^n + \epsilon_{mnj} D^m K_i{}^n \right)^{\mathrm{TF}}. \tag{C.12}$$

TF is the trace-free index with the same definition as in Appendix B. Considering the systems only involve black holes, the source term in equation (C.11) is zero. The final expression of ψ_4 is given by combining Eqs. (C.11), (C.12) and (C.9), putting them back in Eq. (C.10).

Appendix D. Formulation for the kernel FD4 solver

The code utilizes the five-point stencil in one dimension to approximate the derivatives in fourth-order accuracy:

$$
\begin{aligned}
\left. \frac{\partial Q(x)}{\partial x} \right|_{x_0} &= \frac{1}{12\delta x} \left[8 \left(Q^+ - Q^- \right) - \left(Q^{++} - Q^{--} \right) \right] \\
&\quad + \mathcal{O} \left(\delta x^4 \right),
\end{aligned}
\tag{D.1}
$$

where x_0 is the grid point of interest, the $+$ and $-$ upper indices of quantities indicate the relative position of the indexed quantities with the current grid point, and the number of signs specifies the distance, e.g., $Q^{++} := Q(x_0 + 2\delta x)$, $Q^- := Q(x_0 - \delta x)$. We have omitted the vector symbol of the evolving variables Q for simplicity.

Notice that we can apply this formulation to any spatial-dependent function, therefore the divergence of the flux is given as:

$$
\begin{aligned}
\left. \frac{\partial F}{\partial x} \right|_{x_0} &= \frac{1}{12\delta x} \left[8 \left(F^+ - F^- \right) - \left(F^{++} - F^{--} \right) \right] \\
&\quad + \mathcal{O} \left(\delta x^4 \right).
\end{aligned}
\tag{D.2}
$$

Equipped with the relations (D.1) and (D.2), we can derive the time stepping scheme of the fourth-order finite difference (FD4) solver from (11):

$$
\begin{aligned}
\vec{Q}(t + \delta t) &= \vec{Q}(t) + \delta t S(\vec{Q}) \\
&- \sum_i \frac{\delta t}{12 \delta x_i} \left[8 \left(F_i^{i+} - F_i^{i-} \right) - \left(F_i^{i++} - F_i^{i--} \right) \right] \\
&- \sum_i B_i \frac{\delta t}{12 \delta x_i} \left[8 \left(Q^{i+} - Q^{i-} \right) - \left(Q^{i++} - Q^{i--} \right) \right],
\end{aligned}
\tag{D.3}
$$

where we have used the forward finite differences for the time derivatives, i.e. $\partial_t \vec{Q} = \left(\vec{Q}(t + \delta t) - \vec{Q}(t) \right) / \delta t$ for a timestep of size δt. The index i goes from 1 to 3 and indicates the direction of the neighboring points we are looking into. The five-point stencil of (D.1) is the central difference and it is our preferred choice in most cases. However, some terms may need a lopsided stencil to capture specific features or behaviors, especially those that may lead to asymmetric patterns in the system, e.g. the advection terms of the CCZ4 system [65] in numerical
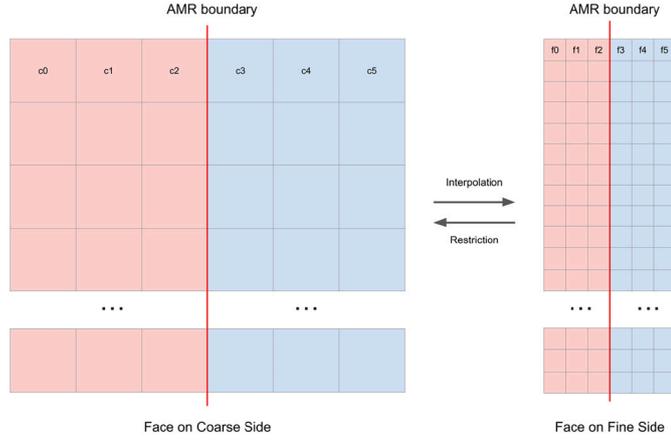
Fig. E.21. The illustration of the face structures on the coarse and fine side of a refinement transition boundary. The mapping between them determines the strategy of the refinement transition in the application. For the interpolation, we need to calculate the outer half of the fine face (the light red squares on the right) using the information of the coarse face. On the other hand, we need to compute the outer half of the coarse face (the light blue squares on the left) utilizing the fine face for a restriction scheme. The layers of the faces are labeled from c0 to c5 and f0 to f5 respectively, it is not the actual enumeration we employed in the actual code.

relativity, which appear in our application as well. The left and right lopsided stencils with also the fourth-order accuracy are

$$\left.\frac{\partial Q(x)}{\partial x}\right|_{x_0} = \frac{1}{12\delta x}\left[-Q^{---} + 6Q^{--} - 18Q^{-} + 10Q + 3Q^{+}\right],$$

$$\text{if } \beta^x > 0 \tag{D.4}$$

$$\left.\frac{\partial Q(x)}{\partial x}\right|_{x_0} = \frac{1}{12\delta x}\left[-3Q^{-} - 10Q + 18Q^{+} - 6Q^{++} + Q^{+++}\right],$$

$$\text{if } \beta^x \leq 0. \tag{D.5}$$

To avoid numerical errors of high-frequency modes, we also introduce the standard Kreiss-Oliger (KO) dissipation at order $N = 3$ in the code as a low-pass filter. Its form in one dimension is [42]:

$$\text{KO}^{(3)} = \frac{\epsilon}{64\delta x}(Q^{---} - 6Q^{--} + 15Q^{-} - 20Q$$
$$+ 15Q^{+} - 6Q^{++} + Q^{+++}), \tag{D.6}$$

where ϵ is a user-defined parameter to control the strength of the numerical dissipation. This term has a convergence speed $\sim \delta x^5$ which can be easily verified by counting the Taylor expansion coefficients. The final numerical dissipation term is the sum over all directions of Eq. (D.6).

Appendix E. Matrices for tensor product interpolation & restriction

The adaptive mesh in ExaHyPE 2 is constructed by refining patches recursively in three partitions and creating new patches accordingly, and refinement transition of multiple levels is not allowed in the code. Therefore, the difference in patch length can only be a factor of three. As all patches in the domain have the same number of volumes, we, therefore, shall have $\delta x_{\text{coarse}} = 3\delta x_{\text{fine}}$.

For the interpolation scheme compatible with our default FD4 kernel, the outer half of a face contains three layers, i.e., layers f0, f1 and f2 in Fig. E.21. The face of the coarse patch, on the other hand, inputs as a whole. Assuming our patches contain p volumes per dimension, this scheme yields a map from $6p^2$ variables to $3 \times (3p)^2$ variables. The current implementation of ExaGRyPE adopts the tensor product approach to perform this mapping, which decomposes the high-dimensional map into the product of three one-dimensional maps. Assuming the volumes in the coarse and fine faces are $Q_{i,j,k}^c$ and $Q_{i,j,k}^f$, the interpolation map can be written as

$$Q_{i,j,k}^f = P_{il}^x P_{jm}^y P_{kn}^z Q_{l,m,n}^c, \tag{E.1}$$

where P^x, P^y, and P^z are the matrices responsible for the corresponding one-dimensional map. As we shall have a symmetry for the two dimensions parallel to the face normal, the two matrices for mappings along them are identical. The matrix for the mappings parallel to the face normal and along the face normal are labeled as $P^{\parallel}$ and $P^{\perp}$, respectively.

Matrix $P^{\parallel}$ should have a size of $3p \times p$, as it maps p coarse volumes to $3p$ fine volumes; on the other hand, matrix $P^{\perp}$ has a size of 3×6, as it maps 6 coarse layers to 3 fine layers (only the outer half of the fine face needs the interpolation). The $P^{\parallel}$ and $P^{\perp}$ matrices of the trilinear interpolation scheme we currently adopt read as

$$P^{\parallel} = \begin{pmatrix} 4/3 & -1/3 & 0 & 0 & 0 & \dots \\ 1 & 0 & 0 & 0 & 0 & \dots \\ 2/3 & 1/3 & 0 & 0 & 0 & \dots \\ 1/3 & 2/3 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & 0 & \dots \\ 0 & 2/3 & 1/3 & 0 & 0 & \dots \\ & & \dots & & & \end{pmatrix}, \tag{E.2}$$

$$P^{\perp} = \begin{pmatrix} 0 & 1/3 & 2/3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2/3 & 1/3 & 0 & 0 \end{pmatrix}. \tag{E.3}$$

In this scheme, we apply a simple linear interpolation to compute the value in the fine volumes according to the relative positions of the closest two coarse volumes. The extrapolation is used for those volumes at the edge of the considered fine face. There is no such requirement in the case of $P^{\perp}$ as all three layers of the fine face are inside the coarse face. We also implement the matrices for the zero-order piecewise constant interpolations as

$$P_{\text{const}}^{\parallel} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \dots \\ 1 & 0 & 0 & 0 & 0 & \dots \\ 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & 0 & \dots \\ 0 & 1 & 0 & 0 & 0 & \dots \\ & & \dots & & & \end{pmatrix}, \tag{E.4}$$

$$P_{\text{const}}^{\perp} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}. \tag{E.5}$$

This means that we use the value in the coarse volume (layer) if the targeted fine volume (layer) is within it. It is the most straightforward interpolation one may have.

The restriction, which is the reverse procedure of the interpolation, has a similar expression to interpolation:

$$Q_{i,j,k}^c = R_{il}^x R_{jm}^y R_{kn}^z Q_{l,m,n}^f, \tag{E.6}$$

where we label the restriction matrix as R. Similarly, we define the matrix $R^{\parallel}$, which maps $3p$ fine volumes to p coarse volumes, and the matrix $R^{\perp}$, mapping 6 fine layers to 3 coarse layers.

In the code practices, we incorporate two versions of matrix $R^{\parallel}$: one for the injection (constant) scheme and the other for the average scheme. However, only one version of $R^{\perp}$ is implemented in the code as the corresponding scheme of zero-order accuracy yields significant numerical errors.

The matrix $R^{\parallel}$ for the injection scheme is

$$R_{\text{injection}}^{\parallel} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ & & & \dots & & & & & \end{pmatrix}. \tag{E.7}$$

In this scheme, we utilize the value in the central volume per three fine volumes for the coarse volume, as this central volume has the same central coordinate (in this parallel direction) as the restricted coarse volume. The matrix $R^{\parallel}$ for the average scheme is

$$R_{\text{average}}^{\parallel} = \begin{pmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 0 & 1/3 & \dots \\ & & & \dots & & & & \end{pmatrix}. \tag{E.8}$$

The average of the three corresponding fine volumes is used to assign the coarse volume and provide improved accuracy.

The version of matrix $R^\perp$ implemented in the code is

$$R^\perp = \begin{pmatrix} 0 & 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 0 & 0 & 0 & -2 & 3 \\ 0 & 0 & 0 & 0 & -5 & 6 \end{pmatrix}. \tag{E.9}$$

This matrix uses a combination of average and extrapolation for the map along the face normal. Notice the enumeration depends on the face orientation rather than relative position. So the right half of the matrix instead of the left half represents the inner part of the fine face (layers f3, f4 and f5).

Data availability

Appendix A provides detailed instructions on reproducing the data used in the manuscript.

References

[1] A. Einstein, Die grundlage der allgemeinen relativitätstheorie, 354 (7) (1916) 769–822, https://doi.org/10.1002/andp.19163540702.

[2] L. Rezzolla, O. Zanotti, Relativistic Hydrodynamics, 2013.

[3] B. Brügmann, W. Tichy, N. Jansen, Numerical simulation of orbiting black holes, 92 (21) (2004) 211101, https://doi.org/10.1103/PhysRevLett.92.211101, publisher: American Physical Society.

[4] F. Pretorius, Numerical relativity using a harmonic decomposition, in: APS April Meeting Abstracts, in: APS Meeting Abstracts, vol. 2004, 2004, R13.006.

[5] F. Löffler, J. Faber, E. Bentivegna, T. Bode, P. Diener, R. Haas, I. Hinder, B.C. Mundim, C.D. Ott, E. Schnetter, G. Allen, M. Campanelli, P. Laguna, The Einstein toolkit: a community computational infrastructure for relativistic astrophysics, Class. Quantum Gravity 29 (11) (2012) 115001, https://doi.org/10.1088/0264-9381/29/11/115001.

[6] K. Clough, P. Figueras, H. Finkel, M. Kunesch, E.A. Lim, S. Tunyasuvunakool, Grchombo: numerical relativity with adaptive mesh refinement, Class. Quantum Gravity 32 (24) (2015) 245011, https://doi.org/10.1088/0264-9381/32/24/245011.

[7] The spectral Einstein code (spec), Website, http://www.black-holes.org/SpEC.html.

[8] M. Fernando, D. Neilsen, H. Lim, E. Hirschmann, H. Sundar, Massively parallel simulations of binary black hole intermediate-mass-ratio inspirals, 41 (2) (2019) C97–C138, https://doi.org/10.1137/18M1196972.

[9] B. Daszuta, F. Zappa, W. Cook, D. Radice, S. Bernuzzi, V. Morozova, GR-athena++: puncture evolutions on vertex-centered oct-tree adaptive mesh refinement, 257 (2) (2021) 25, publisher: IOP Publishing.

[10] I. Ruchlin, Z.B. Etienne, T.W. Baumgarte, SENR/NRPy+: numerical relativity in singular curvilinear coordinate systems, 97 (6) (2018) 064036, publisher: APS.

[11] T. Yamamoto, M. Shibata, K. Taniguchi, Simulating coalescing compact binaries by a new code (SACRA), 78 (6) (2008) 064054, publisher: APS.

[12] D. Hilditch, Solving the Einstein equations numerically, arXiv:2405.06035 [gr-qc].

[13] A. Reinarz, D.E. Charrier, M. Bader, L. Bovard, M. Dumbser, K. Duru, F. Fambri, A.-A. Gabriel, J.-M. Gallard, S. Köppel, L. Krenz, L. Rannabauer, L. Rezzolla, P. Samfass, M. Tavelli, T. Weinzierl, Exahype: an engine for parallel dynamically adaptive simulations of wave problems, Comput. Phys. Commun. 254 (2020) 107251, https://doi.org/10.1016/j.cpc.2020.107251.

[14] T. Weinzierl, The Peano software-parallel, automaton-based, dynamically adaptive grid traversals, ACM Trans. Math. Softw. 45 (2) (Apr. 2019), https://doi.org/10.1145/3319797.

[15] M. Dumbser, F. Guercilena, S. Köppel, L. Rezzolla, O. Zanotti, Conformal and covariant z4 formulation of the Einstein equations: strongly hyperbolic first-order reduction and solution with discontinuous Galerkin schemes, Phys. Rev. D 97 (2018) 084053, https://doi.org/10.1103/PhysRevD.97.084053.

[16] D. Alic, C. Bona-Casas, C. Bona, L. Rezzolla, C. Palenzuela, Conformal and covariant formulation of the z4 system with constraint-violation damping, Phys. Rev. D 85 (2012) 064040, https://doi.org/10.1103/PhysRevD.85.064040.

[17] A. Dubey, A.S. Almgren, J.B. Bell, M. Berzins, S.R. Brandt, G. Bryan, P. Colella, D.T. Graves, M. Lijewski, F. Löffler, B. O'Shea, E. Schnetter, B. van Straalen, K. Weide, A survey of high level frameworks in block-structured adaptive mesh refinement packages, CoRR 74 (12) (2016) 3217–3227.

[18] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, O. Zinenko, Mlir: a compiler infrastructure for the end of Moore's law, arXiv preprint, arXiv:2002.11054, 2020.

[19] C.M. Loi, H. Bockhorst, T. Weinzierl, SYCL compute kernels for ExaHyPE, in: Proceedings of the 2024 SIAM Conference on Parallel Processing for Scientific Computing (PP), 2024, pp. 90–103.

[20] M. Wille, T. Weinzierl, G. Brito Gadeschi, M. Bader, Efficient GPU offloading with OpenMP for a hyperbolic finite volume solver on dynamically adaptive meshes, in: A. Bhatele, J. Hammond, M. Baboulin, C. Kruse (Eds.), High Performance Computing, Springer Nature Switzerland, 2023, pp. 65–85.

[21] B. Li, H. Schulz, T. Weinzierl, H. Zhang, Dynamic task fusion for a block-structured finite volume solver over a dynamically adaptive mesh with local time stepping, in: A.-L. Varbanescu, A. Bhatele, P. Luszczek, B. Marc (Eds.), High Performance Computing, Springer International Publishing, Cham, 2022, pp. 153–173.

[22] J.-M. Gallard, L. Krenz, L. Rannabauer, A. Reinarz, M. Bader, Role-oriented code generation in an engine for solving hyperbolic pde systems, in: G. Juckeland, S. Chandrasekaran (Eds.), Tools and Techniques for High Performance Computing, Springer International Publishing, Cham, 2020, pp. 111–128.

[23] C. Bona, T. Ledvinka, C. Palenzuela, M. Žáček, General-covariant evolution formalism for numerical relativity, Phys. Rev. D 67 (2003) 104005, https://doi.org/10.1103/PhysRevD.67.104005.

[24] C. Gundlach, G. Calabrese, I. Hinder, J.M. Martín-García, Constraint damping in the z4 formulation and harmonic gauge, Class. Quantum Gravity 22 (17) (2005) 3767, https://doi.org/10.1088/0264-9381/22/17/025.

[25] R. Arnowitt, S. Deser, C.W. Misner, Republication of: the dynamics of general relativity, 40 (9) (2008) 1997–2027, https://doi.org/10.1007/s10714-008-0661-1, arXiv:gr-qc/0405109.

[26] L. Lindblom, M.A. Scheel, L.E. Kidder, R. Owen, O. Rinne, A new generalized harmonic evolution system, 23 (16) (2006) S447–S462, https://doi.org/10.1088/0264-9381/23/16/s09, publisher: IOP Publishing.

[27] B. Szilágyi, D. Pollney, L. Rezzolla, J. Thornburg, J. Winicour, An explicit harmonic code for black-hole evolution using excision, 24 (12) (2007) S275–S293, https://doi.org/10.1088/0264-9381/24/12/s18, publisher: IOP Publishing.

[28] T.W. Baumgarte, S.L. Shapiro, Numerical integration of Einstein's field equations, 59 (2) (1998) 024007, https://doi.org/10.1103/PhysRevD.59.024007, publisher: American Physical Society.

[29] M. Shibata, T. Nakamura, Evolution of three-dimensional gravitational waves: harmonic slicing case, 52 (10) (1995) 5428–5444, https://doi.org/10.1103/PhysRevD.52.5428, publisher: American Physical Society.

[30] C. Bona, T. Ledvinka, C. Palenzuela, M. Žáček, Symmetry-breaking mechanism for the z4 general-covariant evolution system, Phys. Rev. D 69 (2004) 064036, https://doi.org/10.1103/PhysRevD.69.064036.

[31] D. Hilditch, S. Bernuzzi, M. Thierfelder, Z. Cao, W. Tichy, B. Brügmann, Compact binary evolutions with the z4c formulation, 88 (8) (2013) 084057, https://doi.org/10.1103/PhysRevD.88.084057, publisher: American Physical Society.

[32] C. Bona, J. Massó, E. Seidel, J. Stela, New formalism for numerical relativity, 75 (4) (1995) 600–603, https://doi.org/10.1103/PhysRevLett.75.600, publisher: American Physical Society.

[33] M. Alcubierre, B. Brügmann, P. Diener, M. Koppitz, D. Pollney, E. Seidel, R. Takahashi, Gauge conditions for long-term numerical black hole evolutions without excision, 67 (8) (2003) 084023, https://doi.org/10.1103/PhysRevD.67.084023, publisher: American Physical Society.

[34] J.M. Bowen, J.W. York, Time-asymmetric initial data for black holes and black-hole collisions, 21 (8) (1980) 2047–2056, https://doi.org/10.1103/PhysRevD.21.2047, publisher: American Physical Society.

[35] B. Brügmann, J.A. González, M. Hannam, S. Husa, U. Sperhake, W. Tichy, Calibration of moving puncture simulations, 77 (2) (2008) 024027, https://doi.org/10.1103/PhysRevD.77.024027, publisher: American Physical Society.

[36] M. Ansorg, B. Brügmann, W. Tichy, Single-domain spectral method for black hole puncture data, 70 (6) (2004) 064011, https://doi.org/10.1103/PhysRevD.70.064011, publisher: American Physical Society.

[37] E. Newman, R. Penrose, An approach to gravitational radiation by a method of spin coefficients, 3 (3) (2004) 566–578, https://doi.org/10.1063/1.1724257.

[38] J.S. Brauchart, P.J. Grabner, Distributing many points on spheres: minimal energy and designs, 31 (3) (2015) 293–326, https://doi.org/10.1016/j.jco.2015.02.003.

[39] R. Womersley, Efficient spherical designs with good geometric properties, Website, http://web.maths.unsw.edu.au/~rsw/Sphere/EffSphDes/index.html.

[40] T. Weinzierl, M. Mehl, Peano—a traversal and storage scheme for octree-like adaptive Cartesian multiscale grids, SIAM J. Sci. Comput. 33 (5) (2011) 2732–2760, https://doi.org/10.1137/100799071.

[41] H. Zhang, T. Weinzierl, H. Schulz, B. Li, Spherical accretion of collisional gas in modified gravity i: self-similar solutions and a new cosmological hydrodynamical code, 515 (2) (2022) 2464–2482, https://doi.org/10.1093/mnras/stac1991.

[42] H. Kreiss, J. Oliger, Methods for the Approximate Solution of Time Dependent Problems, GARP Publications Series, SELBSTVERL. FEBR., 1973, https://books.google.co.uk/books?id=wj9szwEACAAJ.

[43] D. Charrier, B. Hazelwood, T. Weinzierl, Enclave tasking for dg methods on dynamically adaptive meshes, SIAM J. Sci. Comput. 42 (3) (2020) C69–C96.

[44] M.J. Berger, P. Colella, Local adaptive mesh refinement for shock hydrodynamics, 82 (1) (1989) 64–84, https://doi.org/10.1016/0021-9991(89)90035-1.

[45] H. Sundar, R.S. Sampath, G. Biros, Bottom-up construction and 2:1 balance refinement of linear octrees in parallel, 30 (5) (2008) 2675–2708, https://doi.org/10.1137/070681727.

[46] J.G. Baker, J.R. van Meter, Reducing reflections from mesh refinement interfaces in numerical relativity, 72 (10) (2005) 104010, https://doi.org/10.1103/PhysRevD.72.104010, publisher: American Physical Society.

[47] T. Weinzierl, B. Verleye, P. Henri, D. Roose, Two particle-in-grid realisations on spacetrees, 52 (2016) 42–64, https://doi.org/10.1016/j.parco.2015.12.007.

[48] J.C. Butcher, Numerical Methods for Ordinary Differential Equations, John Wiley & Sons, 2016.

[49] A. Rashti, M. Bhattacharyya, D. Radice, B. Daszuta, W. Cook, S. Bernuzzi, Adaptive mesh refinement in binary black holes simulations, 41 (9) (2024) 095001, https://doi.org/10.1088/1361-6382/ad36a5, arXiv:2312.05438 [gr-qc].

[50] M. Radia, U. Sperhake, A. Drew, K. Clough, P. Figueras, E.A. Lim, J.L. Ripley, J.C. Aurrekoetxea, T. França, T. Helfer, Lessons for adaptive mesh refinement in numerical relativity, 39 (13) (2022) 135006, https://doi.org/10.1088/1361-6382/ac6fa9, publisher: IOP Publishing Ltd.

[51] M. Dumbser, F. Fambri, E. Gaburro, A. Reinarz, On glm curl cleaning for a first order reduction of the ccz4 formulation of the Einstein field equations, J. Comput. Phys. 404 (2020) 109088, https://doi.org/10.1016/j.jcp.2019.109088.

[52] M. Alcubierre, G. Allen, C. Bona, D. Fiske, T. Goodale, F.S. Guzmán, I. Hawke, S.H. Hawley, S. Husa, M. Koppitz, C. Lechner, D. Pollney, D. Rideout, M. Salgado, E. Schnetter, E. Seidel, H.-a. Shinkai, D. Shoemaker, B. Szilágyi, R. Takahashi, J. Winicour, Towards standard testbeds for numerical relativity, 21 (2) (2003) 589, https://doi.org/10.1088/0264-9381/21/2/019.

[53] M.C. Babiuc, S. Husa, D. Alic, I. Hinder, C. Lechner, E. Schnetter, B. Szilágyi, Y. Zlochower, N. Dorband, D. Pollney, J. Winicour, Implementation of standard testbeds for numerical relativity, 25 (12) (2008) 125012, https://doi.org/10.1088/0264-9381/25/12/125012.

[54] M. Hannam, S. Husa, D. Pollney, B. Brügmann, N.A. Murchadha, Geometry and regularity of moving punctures, 99 (24) (2007) 241102, https://doi.org/10.1103/PhysRevLett.99.241102, publisher: American Physical Society.

[55] W. Tichy, B. Brügmann, Quasiequilibrium binary black hole sequences for puncture data derived from helical killing vector conditions, 69 (2) (2004) 024006, https://doi.org/10.1103/PhysRevD.69.024006, publisher: American Physical Society.

[56] P. McCorquodale, P. Colella, A high-order finite-volume method for conservation laws on locally refined grids, 6 (1) (2011) 1–25, https://doi.org/10.2140/camcos.2011.6.1, publisher: MSP.

[57] S. Rosswog, Relativistic smooth particle hydrodynamics on a given background spacetime, Class. Quantum Gravity 27 (11) (2010) 114108, https://doi.org/10.1088/0264-9381/27/11/114108.

[58] D. Liptai, D.J. Price, General relativistic smoothed particle hydrodynamics, arXiv:1901.08064, 1 2019, https://doi.org/10.1093/mnras/stz111.

[59] S.J. Magnall, D.J. Price, P.D. Lasky, H.J. Macpherson, Inhomogeneous cosmology using general relativistic smoothed particle hydrodynamics coupled to numerical relativity, Phys. Rev. D 108 (10) (2023) 103534, https://doi.org/10.1103/PhysRevD.108.103534, arXiv:2307.15194.

[60] hpcsoftware, Peano: A framework for high performance numerical solvers on adaptive Cartesian grids, https://gitlab.lrz.de/hpcsoftware/Peano, 2024.

[61] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, D.R. Johnson, GNU Scientific Library Reference Manual, 3rd edition, Network Theory Ltd., Bristol, UK, 2009.

[62] A. Dedner, F. Kemm, D. Kröner, C.-D. Munz, T. Schnitzer, M. Wesenberg, Hyperbolic divergence cleaning for the MHD equations, 175 (2) (2002) 645–673, https://doi.org/10.1006/jcph.2001.6961.

[63] S. Frittelli, Note on the propagation of the constraints in standard 3+1 general relativity, 55 (10) (1997) 5992–5996, https://doi.org/10.1103/PhysRevD.55.5992, publisher: American Physical Society.

[64] M. Alcubierre, Introduction to 3+1 Numerical Relativity, Oxford University Press, 2008.

[65] Y. Zlochower, J.G. Baker, M. Campanelli, C.O. Lousto, Accurate black hole evolutions by fourth-order numerical relativity, 72 (2) (2005) 024021, https://doi.org/10.1103/PhysRevD.72.024021, publisher: American Physical Society.