superB/NRPy: Scalable, Task-Based Numerical Relativity for 3G Gravitational Wave Science

Nishita Jadoo*

Department of Physics, University of Idaho, Moscow, ID 83843, USA

E-mail: njadoo@uidaho.edu

Terrence Pierre Jacques

Department of Physics and Astronomy, West Virginia University, Morgantown, WV 26506, USA
Center for Gravitational Waves and Cosmology, West Virginia University, Chestnut Ridge Research Building, Morgantown, WV 26505, USA
Department of Physics, University of Idaho, Moscow, ID 83843, USA

E-mail: tp0052@mix.wvu.edu

Zachariah B. Etienne

Department of Physics, University of Idaho, Moscow, ID 83843, USA
Department of Physics and Astronomy, West Virginia University, Morgantown, WV 26506, USA
Center for Gravitational Waves and Cosmology, West Virginia University, Chestnut Ridge Research Building, Morgantown, WV 26505, USA

E-mail: zetienne@uidaho.edu

Abstract. Modern gravitational-wave science demands increasingly accurate and computationally intensive numerical relativity (NR) simulations. The Python-based, open-source NRPy framework generates optimized C/C++ code for NR, including the complete NR code BlackHoles@Home (BH@H), which leverages curvilinear coordinates well-suited to many astrophysical scenarios. Historically, BH@H was limited to single-node OpenMP CPU parallelism. To address this, we introduce superB, an open-source extension to NRPy that enables automatic generation of scalable, task-based, distributed-memory Charm++ code from existing BH@H modules. The generated code partitions the structured grids used by NRPy/BH@H, managing communication between them. Its correctness is validated through bit-identical results with the standard OpenMP version on a single node and via a head-on binary black hole simulation in cylindrical-like coordinates, accurately reproducing quasi-normal modes (up to $\ell = 8$). The superB/NRPy-generated code demonstrates excellent strong scaling, achieving an $\approx$ 45x speedup on 64 nodes (7168 cores) compared to the original single-node OpenMP code for a large 3D vacuum test. This scalable infrastructure benefits demanding simulations and lays the groundwork for future multi-patch grid support, targeting long inspirals, extreme parameter studies, and rapid follow-ups. This infrastructure readily integrates with other NRPy/BH@H-based projects, enabling performant scaling for the general relativistic hydrodynamics code GRoovy, and facilitating future coupling with GPU acceleration via the NRPy-CUDA project.

1. Introduction

Numerical relativity (NR) serves as an essential tool for modeling strongly gravitating systems like binary black hole (BBH) and binary neutron star (BNS) coalescences. Its importance has grown significantly with the era of gravitational-wave (GW) astronomy, with many dozens of direct detections by the LIGO and Virgo detectors to date [1, 2, 3, 4]. These simulations, which solve the full Einstein equations numerically, yield vital predictions for GW signals, facilitating the extraction of astrophysical and fundamental physics insights from observations [5, 6, 7]. In addition, NR waveforms are crucial for calibrating faster approximate waveform models [8, 9, 10, 11, 12, 13] used extensively in GW data analysis pipelines.

Ongoing detector upgrades and the prospect of next-generation ground-based [14, 15, 16] and space-based observatories [17, 18, 19] promise a dramatic increase in GW detections and observable phenomena. This progress demands higher accuracy from NR waveforms [20, 21, 22] and requires more efficient and scalable NR codes in order to produce long-duration GW signals, such as those from high-mass-ratio binaries [23]. Developing next-generation NR codes optimized for high-performance computing (HPC) clusters is therefore paramount. This growing demand for accurate, scalable simulations motivates the development of new NR code generation frameworks capable of targeting distributed-memory HPC systems—the focus of this work.

A comprehensive list of scalable NR codes under active development can be found in Table 1 of Ref. [24]. These codes can be classified according to their discretization schemes and grid structures. A significant group employs finite-difference methods with patch-based adaptive mesh refinement (AMR), including those utilizing the `CarpetX` [25] framework in the `Einstein Toolkit` (`ET`) [26, 27] such as `Llama` [28], `McLachlan` [29], and `LEAN` [30], as well as `BAM` [31], `AMSS-NCKU` [32], and `GRChombo` [33]. Other finite-differencing codes, like `Dendro-GR` [34, 35] and `GR-Athena++` [36, 37, 38], employ a block-based oct-tree AMR. Alternative discretization schemes include the pseudo-spectral techniques used in `SpEC` [39] and `bamps` [40, 41], and the hybrid finite-difference and discontinuous Galerkin methods implemented in `SpECTRE` [42, 43, 44] and `Nmesh` [45]. Modern development trends include the adoption of task-based parallelism for dynamic load balancing and performance portability, as exemplified by codes like `SpECTRE` and `AthenaK` [46]—an approach also adopted in this work—and automatic code generation via symbolic algebra, an approach facilitated by frameworks such as `Kranc` [47], `Simflowny` [48] and `NRPy` [49].

Achieving stable, long-term simulations of inspiraling and merging BBHs represented major breakthroughs in NR [50, 51, 52]. These successes relied heavily on robust evolution systems, such as the generalized harmonic formulation [53, 54], as well as the BSSN formulation [55, 56] and its covariant extension [57], which are now widely adopted for their stability properties. Crucially, the development of the "moving puncture" technique [51, 52]—which employs the 1+log slicing and Gamma-

† *Corresponding author: njadoo@uidaho.edu

driver shift gauge condition to manage BH singularities and allow them to move across computational grids without requiring excision—proved essential for enabling routine BBH simulations.

Although NR codes have historically relied upon Cartesian grids with AMR, more recent efforts have explored curvilinear coordinates (e.g., spherical-like or cylindrical-like), which offer significant advantages for certain problems. Many astrophysical systems exhibit approximate symmetries, making curvilinear coordinates a natural choice. They can concentrate computational resources near regions of interest (such as BHs or neutron stars), simplify the implementation of outer boundary conditions for isolated systems, and potentially reduce artificial reflections found in AMR schemes [58, 59, 60]. However, adopting curvilinear coordinates introduces challenges, including the treatment of coordinate singularities and the complexity of tensor basis transformations [61, 49].

`NRPy` [49, 62] is an open-source, Python-based code generation framework designed to address such complexities. It builds on `sympy` [63] and its code generation capabilities to automatically convert high-level tensorial expressions in Einstein notation into highly optimized C/C++ code. This approach forms a foundation for a modular, maintainable, and high-performance NR infrastructure.

A key output of this framework is `BlackHoles@Home` (`BH@H`) [49, 64], which should be understood not as a static program but as a complete, stand-alone NR code entirely generated by `NRPy`. The generated `BH@H` code includes a full infrastructure for simulations—including modules for initial data, timestepping, single-patch numerical gridding, checkpointing, and diagnostics—all managed by a simple Makefile build system also constructed by `NRPy`. The framework's power extends beyond stand-alone codes; `NRPy` can also generate complete modules ("thorns") for the established `ET` community framework, supporting both its legacy `Carpet` and modern `CarpetX` AMR drivers. All open-source `NRPy` code, including the functionality to generate the `BH@H` examples discussed in this paper, is publicly available‡. `NRPy`-generated codes robustly handle coordinate singularities by treating the singular parts of tensor components analytically, so that only the regular (nonsingular) parts are ever interpolated or finite-differenced [61, 65, 49].

Despite these capabilities, the application of `NRPy`-generated codes to the most computationally demanding problems, which require HPC clusters, has been significantly limited. These codes, including `BH@H`, have largely relied upon shared-memory parallelism via `OpenMP`. This restriction confined simulations to single compute nodes, preventing exploration of larger or more complex physical scenarios. Addressing this critical need for scalability is the primary motivation for our current work.

To overcome this limitation, we introduce `superB`, an open-source extension to the `NRPy` framework specifically designed to enable distributed-memory parallelism. `superB` automatically generates scalable, task-based parallelized C++ code utilizing

‡ See the GitHub repository at https://github.com/nrpy/nrpy. The 'README.md' file provides instructions for generating numerous example codes.

the Charm++ parallel programming system [66, 67]. The resulting integrated framework, superB/NRPy, leverages the same underlying physics modules and infrastructure defined within NRPy for BH@H, but extends them for distributed execution. superB/NRPy partitions the structured grids inherent to BH@H's design into parallel tasks (chares) managed by Charm++. It automatically handles the necessary ghost zone communication between tasks, including the complex communication patterns required for both standard 'outer' boundaries and the 'inner' boundaries arising from coordinate singularities [49], using point-to-point messaging.

Our approach with superB/NRPy can be most closely compared to other HPC-oriented efforts in the NR community employing singular curvilinear coordinates. Notably, the closed-source SphericalNR framework [68, 69, 70], developed within the ET, performs dynamical-spacetime general relativistic magnetohydrodynamic (GRMHD) evolutions in spherical coordinates. Similar to NRPy/BH@H, SphericalNR uses a reference-metric formalism [57, 65] and tensor rescaling to manage coordinate singularities, evolving the BSSN or fCCZ4 [71] equations. It adapts the Cartesian GRMHD code GRHydro [72] and made use of an earlier version of NRPy [49, 73] to generate specific source terms and spacetime kernels within the ET's existing MPI-parallelized Cactus/Carpet [74, 75] infrastructure.

In contrast, our open-source superB/NRPy provides distributed-memory capabilities specifically for the NRPy code generation framework. superB/NRPy supports a significantly broader range of curvilinear coordinate systems (including spherical-like, cylindrical-like, prolate-spheroidal-like, and Cartesian-like) and integrates with the general relativistic hydrodynamics (GRHD) code GRoovy [76]. Crucially, superB/NRPy automates the generation of Charm++ parallelized BH@H evolution codes directly from high-level symbolic inputs, offering significant advantages in flexibility, extensibility, and accessibility compared to integrating pre-existing code into larger frameworks.

We validate superB/NRPy by demonstrating numerical equivalence (bit-identical results on a single node) with the standard OpenMP version of BH@H. To showcase its capabilities for demanding simulations, we simulate a head-on collision of two equal-mass, non-spinning BHs starting from a separation of $10\,M$ in cylindrical-like coordinates. We extract gravitational waveforms using the Weyl scalar ψ_4 and decompose the result into spin-weighted spherical harmonics, finding excellent agreement with quasi-normal mode (QNM) predictions up to $\ell = 8$ during the ringdown phase. We further demonstrate compatibility with recent algorithmic improvements like the Slow Start Lapse (SSL) technique [64], which reduces constraint violations and is implemented directly within the shared NRPy/BH@H framework.

Strong scaling tests confirm that superB/NRPy achieves excellent parallel performance. While the original OpenMP code generated by NRPy is more efficient on a single 16-core desktop node for a small 3D vacuum evolution test, the Charm++-based version demonstrates a speedup of approximately 45x on 64 nodes (7168 cores) of a high-core-count cluster compared to the original single-node OpenMP code for a large 3D vacuum evolution test. This scalability significantly expands the scope of NR

applications accessible to the NRPy ecosystem on modern HPC systems.

The HPC-capable parallel infrastructure developed in superB/NRPy establishes a foundation for numerous high-impact applications, including long-duration inspirals, extreme parameter studies, and rapid GW event follow-ups. It directly benefits other NRPy-generated projects, such as the GRoovy code, and paves the way for future integration with node-level GPU acceleration via the NRPy-CUDA project [77]. This scalable superB foundation will also be extended to support code generation for the multi-patch, multi-coordinate grid structures central to the BH@H ecosystem, enabling more flexible domain geometries and higher-fidelity simulations of vacuum, GRHD, and GRMHD systems. Together, the NRPy framework and its superB extension provide a unified, scalable, and extensible platform for next-generation NR modeling, suitable for deployment ranging from volunteer desktops involved in the BlackHoles@Home volunteer computing project to leadership-class supercomputers.

The remainder of this paper is structured as follows: Section 2 reviews the BSSN formulation, gauge conditions, and tensor rescaling as implemented within NRPy/BH@H. Section 3 describes the superB/Charm++ parallelization strategy applied to NRPy-generated code. Section 4 presents validation, the head-on collision simulation, and performance results using superB/NRPy. Section 5 summarizes and discusses future work for the NRPy framework and its superB extension.

2. Basic Equations

We briefly review the formalism used in NRPy for vacuum spacetime evolution, focusing on the evolution equations and tensor rescaling. For more details, see Ref. [49]. Throughout this paper we adopt geometrized units ($G = c = 1$), and Latin indices $(i, j, k, \ldots)$ denote spatial components with repeated indices implying summation.

2.1. Evolution equations

Our formulation employs the covariant BSSN equations with the Lagrangian choice $\partial_t \bar{\gamma} = 0$ [57], adapted for curvilinear coordinates [61, 49]. The spacetime metric undergoes the standard ADM decomposition:

$$ds^2 = -\alpha^2 dt^2 + \gamma_{ij}(dx^i + \beta^i dt)(dx^j + \beta^j dt) \ , \tag{1}$$

where α is the lapse function, β^i the shift vector, and γ_{ij} the spatial 3-metric.

The conformal metric $\bar{\gamma}_{ij}$ is defined via the decomposition

$$\gamma_{ij} = e^{4\phi}\bar{\gamma}_{ij} \ , \tag{2}$$

where ϕ is the conformal factor. Following common practice [78, 79], we evolve $W = e^{-2\phi}$ for improved numerical behavior near punctures.

To effectively handle curvilinear coordinates, we introduce a time-independent reference metric $\hat{\gamma}_{ij}$ $(\partial_t \hat{\gamma}_{ij} = 0)$, typically chosen as the flat metric in the adopted

coordinate system [49]. The conformal metric is then decomposed relative to this reference metric as

$$\bar{\gamma}_{ij} = \hat{\gamma}_{ij} + \varepsilon_{ij} \ , \tag{3}$$

where ε_{ij} represents the deviation from the reference metric. This decomposition is essential both for defining difference-based connection functions and for enabling the tensor rescaling described in Sec. 2.2.

Quantities computed using the conformal metric (e.g., the covariant derivative $\bar{D}_i$ and Christoffel symbols $\bar{\Gamma}^k_{ij}$) are denoted by a bar, while those associated with the reference metric (e.g., $\hat{D}_i$ and $\hat{\Gamma}^k_{ij}$) carry a hat; both are defined with respect to the same coordinate system. The difference between their connection coefficients defines the tensor

$$\Delta^k_{ij} = \bar{\Gamma}^k_{ij} - \hat{\Gamma}^k_{ij} \ , \tag{4}$$

$$\Delta^k \ = \bar{\gamma}^{ij}\Delta^k_{ij} \ . \tag{5}$$

A new BSSN variable, the conformal connection function $\bar{\Lambda}^i$, is introduced and is constrained to satisfy the algebraic constraint

$$\mathcal{C}^i = \bar{\Lambda}^i - \Delta^i = 0 \, .$$

The trace-free part of the conformal extrinsic curvature is defined as

$$\bar{A}_{ij} \equiv e^{4\phi}\Big(K_{ij} - \frac{1}{3}\gamma_{ij}K\Big) \ , \tag{6}$$

where K_{ij} is the extrinsic curvature and $K = \gamma^{ij}K_{ij}$ its trace. The normal derivative operator relative to the Eulerian observer is defined as

$$\partial_\perp = \partial_t - \mathcal{L}_\beta \, ,$$

where $\mathcal{L}_\beta$ denotes the Lie derivative along the shift vector β^i.

The BSSN formulation evolves the metric deviation ε_{ij}, the trace-free conformal extrinsic curvature $\bar{A}_{ij}$, the conformal factor variable W, the trace of the extrinsic curvature K, and the conformal connection functions $\bar{\Lambda}^i$. As implemented in `NRPy`, their evolution equations are:

$$\partial_\perp \varepsilon_{ij} \ = \frac{2}{3}\bar{\gamma}_{ij}\Big(\alpha\bar{A}^k_k - \bar{D}_k\beta^k\Big) + 2\hat{D}_{(i}\beta_{j)} - 2\alpha\bar{A}_{ij} \tag{7}$$

$$\begin{aligned}\partial_\perp \bar{A}_{ij} = & -\frac{2}{3}\bar{A}_{ij}\bar{D}_k\beta^k - 2\alpha\bar{A}_{ik}\bar{A}^k_j + \alpha\bar{A}_{ij}K \\ & + e^{-4\phi}\Big\{-2\alpha\bar{D}_i\bar{D}_j\phi + 4\alpha\bar{D}_i\phi\,\bar{D}_j\phi \\ & \quad + 4\bar{D}_{(i}\alpha\,\bar{D}_{j)}\phi - \bar{D}_i\bar{D}_j\alpha + \alpha\bar{R}_{ij}\Big\}^{\rm TF}\end{aligned} \tag{8}$$

$$\partial_\perp W \ = -\frac{1}{3}W\Big(\bar{D}_k\beta^k - \alpha K\Big) \tag{9}$$

$$\partial_\perp K \ = \frac{1}{3}\alpha K^2 + \alpha\bar{A}_{ij}\bar{A}^{ij} - e^{-4\phi}\Big(\bar{D}_i\bar{D}^i\alpha + 2\bar{D}^i\alpha\,\bar{D}_i\phi\Big) \tag{10}$$

$$\partial_\perp \bar{\Lambda}^i = \bar{\gamma}^{jk}\,\hat{D}_j\hat{D}_k\beta^i + \frac{2}{3}\Delta^i\,\bar{D}_j\beta^j + \frac{1}{3}\bar{D}^i\bar{D}_j\beta^j - 2\bar{A}^{ij}\Big(\partial_j\alpha - 6\,\partial_j\phi\Big) + 2\,\bar{A}^{jk}\Delta^i_{jk} - \frac{4}{3}\alpha\,\bar{\gamma}^{ij}\,\partial_j K \tag{11}$$

Here, TF indicates that the enclosed expression is taken to be trace-free with respect to $\bar{\gamma}^{ij}$, and the conformal Ricci tensor $\bar{R}_{ij}$ is computed as

$$\bar{R}_{ij} = -\frac{1}{2}\bar{\gamma}^{kl}\hat{D}_k\hat{D}_l\bar{\gamma}_{ij} + \bar{\gamma}_{k(i}\hat{D}_{j)}\bar{\Lambda}^k + \Delta^k\Delta_{(ij)k} + \bar{\gamma}^{kl}\left(2\Delta^m_{k(i}\Delta_{j)ml} + \Delta^m_{ik}\Delta_{mjl}\right)\,. \tag{12}$$

The trace-free condition $\bar{A}_{ij}\bar{\gamma}^{ij} = 0$ is enforced dynamically via the $\bar{A}^k_k$ term in Eq. (7).

We employ the standard 1+log slicing [80] and Gamma-driver shift [81] conditions, which evolve the lapse α, shift β^i, and an auxiliary variable B^i:

$$\partial_t\alpha = -2\alpha K + \beta^j\partial_j\alpha\,, \tag{13}$$

$$\partial_t\beta^i = B^i + \beta^j\partial_j\beta^i\,, \tag{14}$$

$$\partial_t B^i = \frac{3}{4}\partial_t\bar{\Lambda}^i - \eta B^i + \beta^j\partial_j B^i\,, \tag{15}$$

where η is a damping parameter [82]. Note that $\partial_t\bar{\Lambda}^i$ on the right-hand side of Eq. (15) is substituted using Eq. (11). These choices yield a total of 24 evolved variables.

While the BSSN equations above describe the time evolution of the geometric variables, the constraint equations complete the description of Einstein's equations: the Hamiltonian constraint $\mathcal{H}$ and the momentum constraints $\mathcal{M}^i$, which must be satisfied on each spatial hypersurface. While mathematically preserved by the evolution equations, numerically they provide a crucial check on the accuracy of the simulation. Throughout the simulation we monitor these constraints [49, 61]:

$$\mathcal{H} \equiv \frac{2}{3}K^2 - \bar{A}_{ij}\bar{A}^{ij} + e^{-4\phi}\left(\bar{R} - 8\bar{D}^i\phi\bar{D}_i\phi - 8\bar{D}^2\phi\right) = 0\,, \tag{16}$$

$$\mathcal{M}^i \equiv e^{-4\phi}\left(\hat{D}_j\bar{A}^{ij} + 2\bar{A}^{k(i}\Delta^{j)}_{jk} + 6\bar{A}^{ij}\partial_j\phi - \frac{2}{3}\bar{\gamma}^{ij}\partial_j K\right) = 0\,. \tag{17}$$

Here, $\bar{R} = \bar{\gamma}^{ij}\bar{R}_{ij}$.

2.2. Tensor rescaling

Coordinate singularities intrinsic to curvilinear systems (e.g., at $r = 0$ or $\theta \in \{0, \pi\}$ in spherical coordinates) can cause tensor components expressed in the coordinate basis $\hat{e}_{(i)}$ to diverge, even for physically regular fields. Numerical methods like finite differencing and interpolation require sufficiently smooth input functions; applying them directly to divergent components induces large errors and potential instabilities. Tensor rescaling addresses this challenge by expressing tensor components in a carefully chosen non-coordinate basis $\tilde{e}_{(i)}$ such that the components remain smooth and regular at these singularities [61, 49].

Consider, for example, flat 3D space described by Cartesian coordinates $x^k = (x, y, z)$ and standard spherical coordinates $\hat{x}^i = (r, \theta, \phi)$ related by

$$x = r \sin\theta \cos\phi \ , \tag{18}$$

$$y = r \sin\theta \sin\phi \ , \tag{19}$$

$$z = r \cos\theta \ . \tag{20}$$

A tensor with Cartesian components A^{kl} transforms to spherical coordinate basis components $\hat{A}^{ij}$ according to the standard transformation rules:

$$\hat{A}^{ij} = A^{kl} \frac{\partial \hat{x}^i}{\partial x^k} \frac{\partial \hat{x}^j}{\partial x^l} \ .$$

The flat Euclidean metric becomes the reference metric in spherical coordinates, $\hat{\gamma}_{ij} = \mathrm{diag}(1,\, r^2,\, r^2 \sin^2\theta)$. While $\hat{\gamma}_{ij}$ itself is regular, other tensors can acquire divergent components in the spherical *coordinate basis* $\hat{e}_{(i)}$ at $r = 0$ or where $\sin\theta = 0$. For instance, a constant Cartesian vector $V^k = (1, 0, 0)$ transforms to spherical coordinate components

$$\hat{V}^i = (\sin\theta\cos\phi, \frac{\cos\theta\cos\phi}{r}, -\frac{\sin\phi}{r\sin\theta}) \ . \tag{21}$$

These components diverge, rendering standard numerical finite differencing, quadrature, or interpolations problematic near the singularities.

The solution involves defining a rescaled, non-coordinate basis $\tilde{e}_{(i)}$ derived from the reference metric. For a diagonal reference metric $\hat{\gamma}_{ij} = \mathrm{diag}(\hat{\gamma}_{11},\, \hat{\gamma}_{22},\, \hat{\gamma}_{33})$, we define scale factors

$$S_{(i)} = \sqrt{\hat{\gamma}_{ii}} \quad \text{(no summation implied)} \, ,$$

and corresponding rescaled basis vectors

$$\tilde{e}_{(i)} = \frac{\hat{e}_{(i)}}{S_{(i)}} \, .$$

In this orthonormal rescaled basis, the reference metric becomes the identity matrix, $\tilde{\gamma}_{ij} = \delta_{ij}$. Tensor components are transformed to represent them in this regular basis: contravariant components become $\tilde{V}^i = \hat{V}^i \, S_{(i)}$, and covariant components become $\tilde{V}_i = \hat{V}_i / S_{(i)}$ (no sum). For the example vector $\hat{V}^i$ above, the rescaled components in the tilde basis are

$$\tilde{V}^i = (\sin\theta\cos\phi,\, \cos\theta\cos\phi,\, -\sin\phi) \, ,$$

which represents the same vector but using components that are regular everywhere. `NRPy` automatically applies this rescaling procedure for supported curvilinear coordinate systems by using the appropriate scale factors.

3. Charm++ parallelization of NRPy-generated code

The NRPy framework generates code that operates on structured, logically rectangular grids, even when representing curvilinear coordinate systems. Spatial coordinates x^i ($i = 0, 1, 2$) correspond to (x, y, z), (r, θ, ϕ), or (ρ, ϕ, z) for Cartesian-like, spherical-like, or cylindrical-like systems, respectively. The grid spans a logical domain from $x^i_{\rm min}$ to $x^i_{\rm max}$ with N_{x^i} points in each direction. Grid points are cell-centered and uniformly distributed between $x^i_{\rm min} + dx^i/2$ and $x^i_{\rm max} - dx^i/2$, where $dx^i = (x^i_{\rm max} - x^i_{\rm min})/N_{x^i}$. This cell-centering strategy inherently avoids placing grid points directly on coordinate singularities, as detailed in Ref. [49].

Spatial derivatives are computed using finite differencing of selectable order $N_{\rm FD}$ (typically 2, 4, 6, 8, 10). Most derivatives employ centered stencils, requiring $N_G = N_{\rm FD}/2$ layers of ghost points surrounding the computational domain to compute derivatives near the boundaries. By contrast, shift–advection derivatives (terms involving $\beta^i \partial_i$) are computed using upwind finite differences, which require $N_G = N_{\rm FD}/2 + 1$ ghost layers on the appropriate side. The nature of these ghost points depends on the coordinate system and the location on the numerical grid. Figure 1 visualizes a spherical grid, illustrating two distinct types:

- **Outer boundary points** (blue circles in Fig. 1) lie beyond the physical boundary of the domain (e.g., near $r = r_{\rm max}$). These are filled using physical boundary conditions, such as Sommerfeld radiation conditions [83] or extrapolation.
- **Inner boundary points** (red circles in Fig. 1) arise from coordinate singularities (e.g., at $r = 0$) or periodicity (e.g., at $\phi = \pm\pi$). These points logically map to interior grid points or outer boundary points elsewhere on the grid (indicated by the points they encircle in Fig. 1). They are filled using values from the corresponding points, applying appropriate parity transformations based on the tensor type and coordinate symmetries, a process handled automatically by NRPy [49].

In the original single-node NRPy implementation, filling both types of ghost points involves only local data access or calculation.

To enable distributed-memory parallelism using Charm++, the superB extension partitions the global logically rectangular computational grid into smaller rectangular blocks, known as 'chares'. Figure 2 illustrates this partitioning concept for the spherical grid depicted logically via its (r, ϕ) coordinates. The grid is divided along each logical coordinate direction x^i into $N_{{\rm chare}^i}$ segments, resulting in a total of $N_{{\rm chare}^0} \times N_{{\rm chare}^1} \times N_{{\rm chare}^2}$ chares (Fig. 2 shows an example with $N_{{\rm chare}^r} = 2, N_{{\rm chare}^\phi} = 4$). Each chare is responsible for managing a subgrid containing $\prod_{i=0}^{2}(N_{x^i}/N_{{\rm chare}^i})$ interior points, ensuring this size remains larger than the required ghost zone width N_G.

This domain decomposition necessitates communication between chares. Since finite differencing stencils require data from neighboring grid points, these points may now reside on different chares (and thus different processing units or nodes). Filling the ghost points for each chare now involves managing three distinct scenarios, building upon the classifications illustrated in Figure 2:

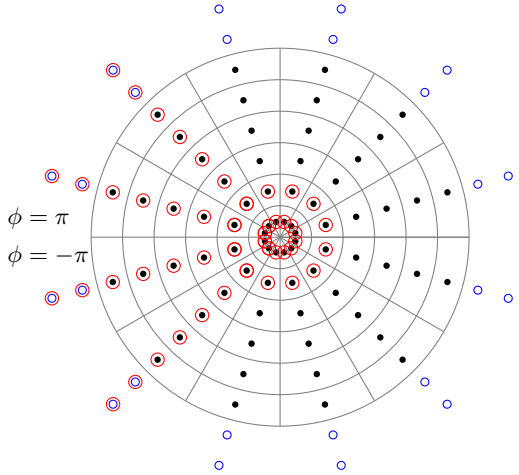

Figure 1. Physical grid layout in the equatorial plane for a spherical coordinate system used by `NRPy/BH@H`, illustrating different ghost point types. Black dots represent interior grid points. Blue circles are outer boundary ghost points, located beyond the maximal radius ($r > r_{\max}$). Red circles are inner boundary ghost points, arising from coordinate system properties: the origin ($r = 0$) and periodicity ($\phi = \pm\pi$). These inner boundary points map logically to the points they encircle, requiring specific parity conditions upon copying. Two layers of ghost points ($N_G = 2$) are shown.

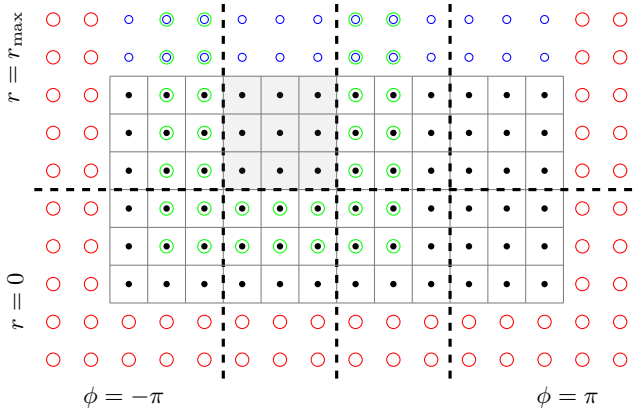

Figure 2. Schematic of the parallel decomposition strategy. The spherical grid from Fig. 1 is shown mapped to its logically rectangular domain (r, ϕ) and partitioned into chares (indicated by dashed lines). Black dots are interior points; blue circles are outer boundary ghost points ($r > r_{\max}$); red circles are inner boundary ghost points ($r = 0$ mapping and $\phi = \pm\pi$ periodicity). The shaded region highlights one example chare. This partitioning introduces **neighbor ghost points** (green circles), which map to grid points lying on neighboring chares. Communication via message passing is required to fill some inner boundary points and all neighbor points correctly.

(i) **Outer boundary points:** Chares located at the global physical boundary (e.g., the top edge at $r = r_{\max}$ in Fig. 2) fill their respective outer ghost points (blue circles) locally using the prescribed physical boundary conditions (e.g., Sommerfeld), identical to the single-node case. No inter-chare communication is needed for these points.

(ii) **Inner boundary points:** These points (red circles at $r = 0$ and $\phi = \pm\pi$ edges in Fig. 2) still logically map to interior or outer boundary points elsewhere on the global grid. If the source point for the mapping resides within the *same* chare, the data is copied locally with appropriate parity transformations, as before. However, if the source point lies on a *different* chare (which might be non-adjacent in the logical grid partitioning, especially for mappings near $r = 0$), `superB/NRPy` generates code implementing direct point-to-point communication using `Charm++` messages to retrieve the required data. The underlying mapping logic from `NRPy` is used to identify the coordinates of the target source point and determine its owning chare. This is optimized by having each chare, at initial setup, compute a list of points required by other chares and sending only the required data directly to each chare.

(iii) **Neighbor ghost points:** These points (green circles for the shaded chare in Fig. 2) are interior, inner boundary or outer boundary points of one chare that are needed to fill the ghost zones of an adjacent chare due to the finite difference stencil overlapping the boundary between chares. These points constitute the "halo" region and must be populated via inter-chare communication. This is optimized by sending face data, including ghost points, sequentially in the east-west, then north-south, then top-bottom directions. Correct data for the ghost points in chares sharing an edge or a corner are thus automatically updated without direct communication with these adjacent chares.

To ensure correct data is available for derivatives, the boundary conditions and communication must be performed in a specific order within each Runge-Kutta time step substage:

(i) Apply outer boundary conditions locally on relevant chares.

(ii) Apply inner boundary conditions, involving local copies (intra-chare mappings) and point-to-point inter-chare communication (remote mappings).

(iii) Exchange neighbor ghost point data (halo exchange) between adjacent chares. This step typically uses the standard technique of communicating face data sequentially (e.g., east-west, then north-south, then top-bottom) to correctly update face, edge, and corner neighbor data.

This sequence guarantees that all required ghost point data is valid before finite differences are computed for the interior points of each chare.

This communication pattern is executed after each substage of the chosen time integrator (e.g., SSPRK3 [84]). We use the `Ckio` library within `Charm++` for efficient parallel I/O of diagnostic data and integrate `Charm++`'s native checkpointing capabilities for enhanced fault tolerance.

4. Results

The NRPy framework [49], upon which superB/NRPy is built, has undergone rigorous validation. Specifically, NRPy-generated codes, including BH@H, have achieved round-off level agreement with established NR codes in spherical coordinates [61], produced results consistent with other ET modules for BBH evolutions in Cartesian coordinates [64], and exhibited expected convergence rates for constraint violations across various coordinate systems. Further, the NRPy-based BH@H code accurately extracts ringdown waveforms via the Newman–Penrose scalar $\psi_4 = \ddot{h}_+ - i\,\ddot{h}_\times$, matching analytical quasinormal-mode frequencies and exponential amplitude damping rates [49] for a dominant mode. Since superB/NRPy reuses the core physics and infrastructure modules from NRPy also employed by BH@H, it directly inherits this robustly validated foundation.

The primary contribution of this work is the Charm++-based distributed-memory parallelization enabled by superB. We first verified its correctness by comparing the computed values of evolved variables, such as the conformal factor variable $W \equiv e^{-2\phi}$, against those from the standard OpenMP-based BH@H code generated by NRPy for a small grid simulation of a head-on BH collision in axisymmetry using cylindrical-like coordinates on a 16-core desktop. When compiled with the same compiler and run with identical parameters on a single node, the Charm++ code generated by superB/NRPy produces bit-identical results to the OpenMP code, as shown in Fig. 3. As part of this work, we also extended the decomposition of the Weyl scalar ψ_4 using spin-weighted spherical harmonics (${}_{-2}Y_{\ell m}$) to properly handle cylindrical-like coordinates, enabling waveform extraction in these systems within the superB/NRPy framework.

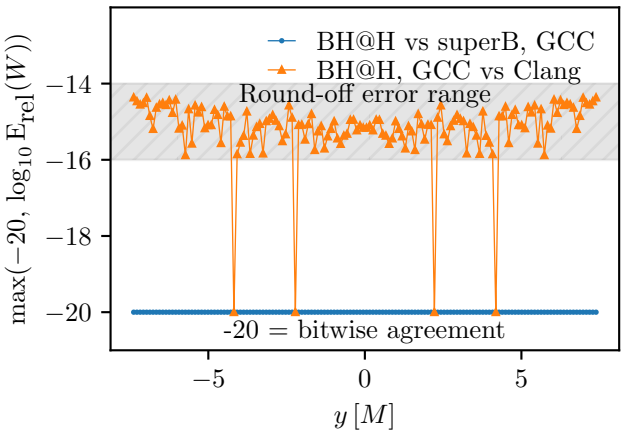

Figure 3. Demonstration of bitwise agreement for the evolved variable W along the y-axis at the end of an axisymmetric head-on BH collision simulation in cylindrical-like coordinates on a 16-core desktop comparing BH@H (OpenMP) and superB (Charm++, partitioned with $N_{\text{chare}^i} = \{18, 2, 1\}$). When using the same compiler, the results are identical (solid circles, difference is zero). For comparison, using BH@H compiled with different compilers results in expected round-off-level differences (solid triangles).

To demonstrate the capabilities and performance of superB/NRPy, particularly for

simulations benefiting from curvilinear coordinates and parallelism, we present two main sets of results. First, we analyze a head-on BH collision, validating waveform extraction against QNM predictions and examining constraint violations, including the impact of the SSL technique [64] (which is implemented within the shared `NRPy/BH@H` infrastructure and thus available in `superB/NRPy`). Second, we present performance benchmarks, including single-node comparisons to the `OpenMP` baseline and multi-node strong scaling tests on an HPC cluster.

4.1. Black hole head-on collision

We simulate the head-on collision of two equal-mass, non-spinning puncture BHs with bare masses $m_1 = m_2 = 0.5\,M$, initially at rest on the z-axis with a coordinate separation of $10\,M$.§ This axisymmetric scenario benefits from the use of cylindrical-like coordinates generated via `NRPy/BH@H`, which naturally concentrate resolution near the collision axis ($\rho = 0$) and the origin. These coordinates are defined by applying a sinh rescaling to the standard cylindrical coordinates (ρ, z):

$$x'(x) = x_{\rm max}\frac{\sinh(x/w)}{\sinh(1/w)}\ , \tag{22}$$

where x represents the original coordinate (ρ or z), $x_{\rm max}$ is the outer boundary of the domain extent in that coordinate, and $w = 0.2$ is a parameter controlling the concentration of points near the origin ($x = 0$). The computational grid domain extends from $\rho_{\rm min} = 0$ to $\rho_{\rm max} = 300\,M$ and from $z_{\rm min} = -300\,M$ to $z_{\rm max} = 300\,M$. The simulation runs until a final time of $t = 450\,M$.

Initial data are generated using `TwoPunctures` [85]. The initial spatial slice is specified to be conformally flat and maximal ($K = 0$). The evolution employs the standard moving puncture gauge conditions (1+log slicing for α and the Gamma-driver shift condition for β^i). This mismatch between initial data ($K = 0$) and gauge condition ($1 + \log$) generates a known initial pulse of spurious ("junk") radiation [86, 87, 59]. We use eighth-order spatial finite differencing ($N_{\rm FD} = 8$), eighth-order Sommerfeld radiation boundary conditions, SSPRK3 time integration with a Courant-Friedrichs-Lewy (CFL) factor of 0.5, and Gamma-driver damping parameter $\eta = 2.0$. Kreiss-Oliger dissipation is applied with strength $k = 0.99$ for gauge variables (α, β^i, B^i) and $k = 0.3$ for metric variables, using ninth-order finite-differencing for the dissipation operators. We compare runs performed with and without the SSL technique enabled.

The baseline grid resolution is $N_{(\rho,\phi,z)} = \{800, 2, 2400\}$. Due to axisymmetry, only two points are needed in the ϕ direction. We perform runs at low (1x), medium (1.25x), and high (1.5x) resolutions, achieved by scaling N_ρ and N_z proportionally.

§ This example can be easily reproduced. First, install `NRPy` by running `pip install nrpy`. (Further details can be found in the project's 'README.md' file at https://github.com/nrpy/nrpy.) Next, install `Charm++` by following the official manual. Finally, run `python3 -m nrpy.examples.superB_blackhole_spectroscopy --paper` to generate the example code and follow the instructions displayed on the terminal for compiling and running.

Gravitational waveforms are extracted by decomposing the Weyl scalar ψ_4 into spin-weight -2 spherical harmonics up to $\ell = 8$ at various radii. To validate the physical accuracy of the evolution and extraction, we compare the ringdown portion of the dominant modes with analytical predictions from BH perturbation theory. Figure 4 shows the real part of the $m = 0$ modes ($\ell = 2, 4, 6, 8$) extracted at $R_{\rm ext} = 80\,M$ for the high-resolution run with SSL enabled, plotted against fits using the known fundamental quasi-normal mode (QNM) frequencies ($\omega_{\ell 00}$) and damping rates ($\alpha_{\ell 00}$) for a Schwarzschild BH [88]. The amplitude A_f and phase ϕ_f are fitted between retarded times $t - R_{\rm ext} = 100\,M$ and $130\,M$. As seen in the figure, the numerical waveforms show excellent agreement with the analytical QNM predictions for both frequency and damping during the ringdown phase. Minor deviations near the waveform peak are expected due to the presence of decaying overtones [89], which are not included in our simple fundamental mode fit. The late-time deviations occur as the physical signal decays below the simulation's numerical noise floor.

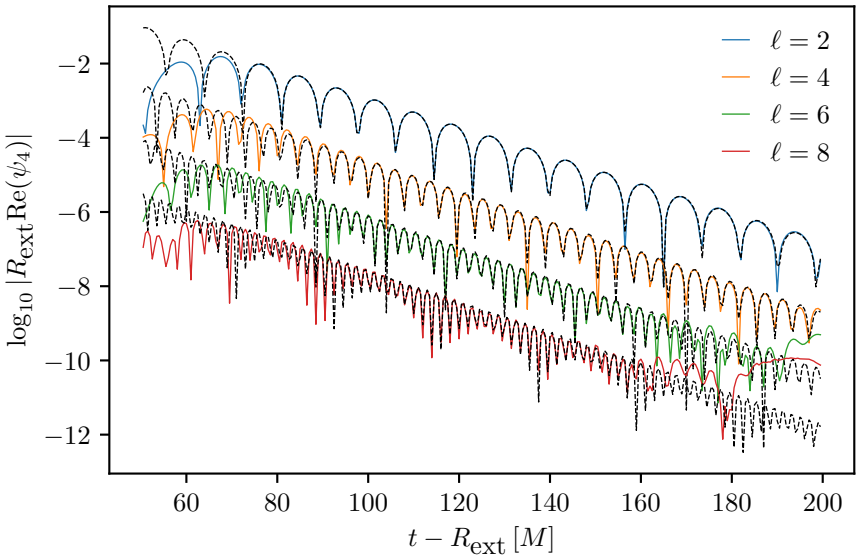

Figure 4. Head-on collision of two equal-mass, non-spinning BHs from an initial separation of $10\,M$. Real part of the extracted Weyl scalar ψ_4, decomposed into spin-weighted spherical harmonics ${}_{-2}Y_{\ell m}$, for the $m = 0$ modes ($\ell = 2, 4, 6, 8$). Waveforms are extracted at radius $R_{\rm ext} = 80\,M$ from the high-resolution simulation with SSL enabled (solid colored lines). These are compared against analytical fits (dashed black lines) using the fundamental quasi-normal mode (QNM) frequencies and damping rates for the final Schwarzschild BH [88]. Deviations near the waveform peak are primarily caused by overtones [89], which are present in the simulation but omitted from our fundamental-mode-only fit. The late-time deviations occur as the physical signal decays below the simulation's numerical noise floor.

We also examine the evolution of constraint violations, which serve as a measure of numerical accuracy, and test the impact of the SSL technique. Figure 5 presents the logarithm of the Hamiltonian and momentum constraint violations along the y-axis (ρ-axis at $z = 0$) and z-axis ($\rho = 0$) at $t = 50\,M$ for the three resolutions, comparing runs

with and without SSL. The SSL technique significantly reduces constraint violations—particularly the transient spikes associated with the initial gauge dynamics—by about an order of magnitude, consistent with the findings in Ref. [64]. Further, the constraints exhibit the expected convergence towards zero with increasing resolution for both sets of runs.

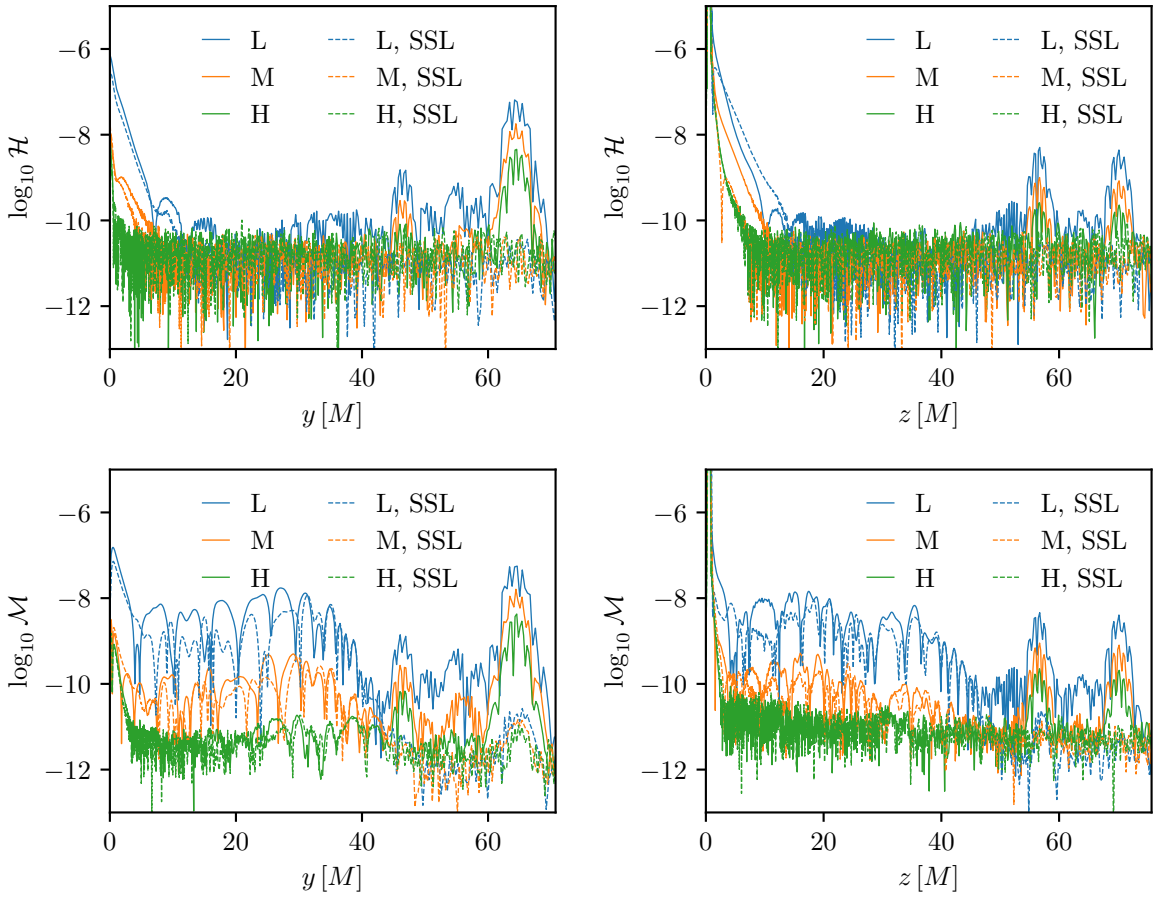

Figure 5. Head-on collision of two BHs from a separation of $10\,M$. Logarithm of Hamiltonian, $\mathcal{H}$, and momentum, $\mathcal{M} \equiv \sqrt{\mathcal{M}_i \mathcal{M}^i}$, constraint violations (Eqs. (16) and (17)) along the y-axis (ρ-axis at $z = 0$) (left panels) and z-axis ($\rho = 0$) (right panels) at $t = 50\,M$. Results are shown for low (L), medium (M), and high (H) resolutions, comparing runs with SSL (dashed lines) and without SSL (solid lines).

4.2. Performance tests

4.2.1. Single node We modify the test case from that in Sec. 4.1, by setting the BHs initially at a coordinate separation of $1\,M$ and sharing a common horizon similar to Sec. V B 3 of Ref. [49], while the outer boundary is at $7.5\,M$. This configuration of the BHs at close separation and merged are well-suited for spherical-like coordinates that present a more stringent test than cylindrical-like and Cartesian-like coordinates for parallel scaling due to more complex communication patterns. We increase the

resolution in the ϕ direction to create a genuinely 3D problem. The test setup uses a grid size of $N_{x^i} = \{N_r, N_\theta, N_\phi\} = \{72, 12, 24\}$. We employ fourth-order finite differencing ($N_{\rm FD} = 4$), requiring $N_G = N_{\rm FD}/2 + 1 = 3$ ghost zones to handle the upwinded shift advection terms.

Recall that the original `NRPy` framework generates code parallelized using `OpenMP` for intra-node shared-memory execution, which typically incurs relatively low overhead. In contrast, our `superB/NRPy` extension generates `Charm++` code that partitions the grid into chares. This approach introduces overhead, primarily from the explicit management of tasks and the inter-chare communication required to fill ghost layers ($N_G = 3$ in this test), even when running on a single node.

We first compare the performance of the `Charm++` parallelization against the original `OpenMP` version on a single node. This comparison quantifies the baseline overhead associated with the distributed-memory approach, which is relevant for users running on desktops or single HPC nodes (e.g., for volunteer computing via `BlackHoles@Home`). We perform these tests on a 16-core desktop machine equipped with an AMD Ryzen 9 3950X processor.

We run the 3D test case described above for 7.5 M (8428 iterations) with diagnostics output disabled. Table 1 summarizes the wall-clock times for code generated by `superB/NRPy`, using either the shared-memory SMP `Charm++` build or the `Charm++` MPI build, both with partitioning $N_{\rm chare^i} = \{4, 2, 2\}$, and code generated by `NRPy` targeting its standard `OpenMP` backend.

Table 1. Wall-clock time (seconds) for single desktop node test case.

Version	Wall-clock time [s]
`NRPy` (`OpenMP` backend, 32 threads)	37.2
`superB/NRPy` (`Charm++` SMP, 16 cores)	62.0
`superB/NRPy` (`Charm++` MPI, 16 cores)	82.1

As expected, the original `NRPy`-generated `OpenMP` version executes fastest on a single node due to its lower overhead compared to the explicit tasking and message passing inherent in the `Charm++` versions. The `Charm++` SMP build, which uses memory copies for communication between cores on the same node, is slower due to the overhead of managing explicit ghost zone data transfers and task scheduling. The `Charm++` MPI build incurs the highest overhead on a single node due to the involvement of the network protocol stack, even when communication occurs via the loopback interface.

To gain further insight into the performance characteristics of the `Charm++` version, we made use of the `Projections` analysis tool [90]. Figure 6 shows the time profile visualization for the MPI build run on the single 16-core desktop node, indicating average core utilization. The colored blocks, repeated four times, correspond to the computations for the four RK4 substages.‖ The blue regions correspond to evaluation

‖ While the head-on collision used SSPRK3, these performance benchmarks employed RK4.

of the right-hand-side of the RK substages and applying boundary conditions, the red regions correspond to the RK substages update, the yellow regions correspond to sending and processing inner boundary points residing on other chares' grids and the cyan regions correspond to exchanging neighbor ghosts points. The gray region corresponds to idle time, which is approximately 20%, and the black region to the `Charm++` runtime overhead. The idle time arises partly from load imbalance between chares and synchronization waits during the communication phases (halo exchange, point-to-point messages for inner boundaries). Future work involving multi-patch grids may offer better load balancing and potentially improve performance by overlapping computation and communication more effectively, leveraging `Charm++`'s task-based asynchronous execution model.

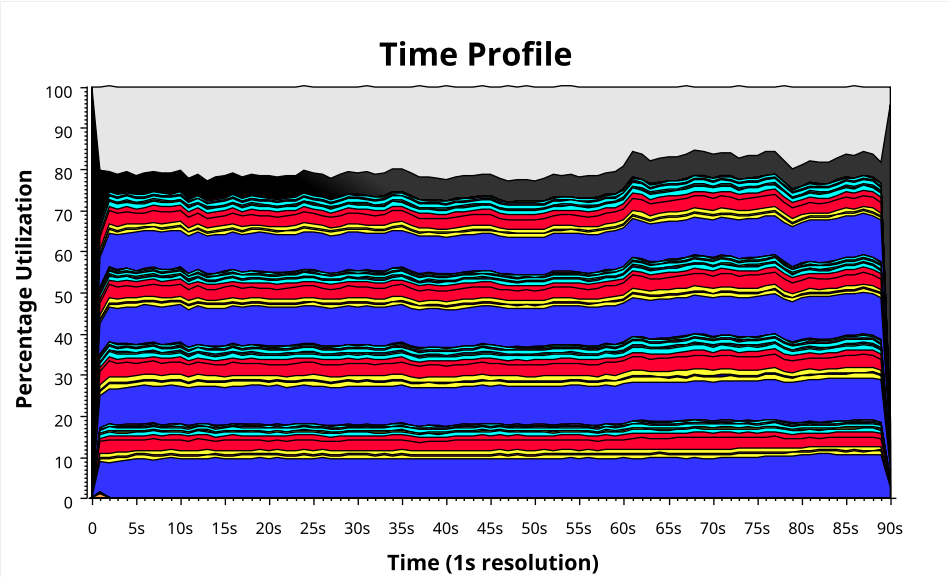

Figure 6. (Color online.) Time profile view generated by the `Projections` analysis tool [90] for the 3D BH vacuum test case running on a single 16-core desktop node using the `superB/NRPy`-generated code (`Charm++` MPI build, $N_{\text{chare}} = 16$). The plot shows average core utilization over the total execution time. The repeated sequence of colored blocks corresponds to the substages of the RK4 time integrator: blue segments represent right-hand-side evaluation and boundary condition application; red segments correspond to the RK substage update step; yellow segments indicate communication for inner boundary ghost points; cyan segments show communication for neighbor ghost points (halo exchange); gray regions represent idle time (approximately 20% in this run); and black indicates overhead associated with the `Charm++` runtime system.

4.2.2. Strong scaling on multiple nodes To assess the distributed-memory scalability—the primary objective of developing `superB/NRPy`—we performed strong scaling tests. While both strong and weak scaling are valuable performance metrics, their relevance is application-dependent. We focus on strong scaling as it is the most critical benchmark for the scientific problems `superB` is designed to solve.

Weak scaling, where the problem size per core is held constant, is indispensable for applications where the required resolution scales with the domain volume, such as simulations in which e.g., GR(M)HD turbulence plays an important role. In contrast, `superB` is engineered to tackle challenges in *vacuum* binary black hole evolution, such as extreme mass-ratio inspirals, high spins, or relativistic scattering. In these scenarios, the primary difficulty is not an expanding domain but the need to resolve disparate physical scales in highly localized regions (e.g., near the smaller black hole's horizon).

The most computationally efficient strategy for such problems involves concentrating resources where they are most needed. This is typically achieved with multi-patch grid structures, where additional, finer grids are placed around features of interest. This approach keeps the total problem size (i.e., the total number of grid points across all patches) relatively fixed, even as the physical parameters become more extreme. Thus, the critical performance question is how efficiently the code can solve a large, fixed-size problem as more computational resources are allocated. This is precisely what a strong scaling analysis measures.

To this end, we conducted our strong scaling tests on the WindRiver HPC cluster at Idaho National Laboratory. This cluster consists of nodes equipped with dual-socket Intel Xeon Platinum 8480+ "Sapphire Rapids" CPUs, providing 56 cores per socket, for a total of 112 physical cores per node. To ensure sufficient computational work per core and that our test was representative of future production runs, we used a large 3D grid with $N_{x^i} = \{N_r, N_\theta, N_\phi\} = \{1008, 168, 336\}$, totaling approximately 5.7×10^7 grid points. The simulation was run for a very short duration ($0.0001\,M$, corresponding to 124 iterations) with diagnostic output disabled to focus purely on the computational scaling.

We executed the runs on 1, 8, 27, and 64 nodes, corresponding to 112, 896, 3024, and 7168 total cores, respectively. In each run, the total number of `Charm++` chares was set equal to the total number of cores. The partitioning across the logical grid dimensions ($N_{\text{chare}^r}, N_{\text{chare}^\theta}, N_{\text{chare}^\phi}$) was adjusted for each run to keep the aspect ratio of the chares constant while matching the total number of chares to the core count.

Figure 7 presents the resulting wall-clock time versus the total number of cores used. The plot demonstrates performance scaling that is close to ideal (indicated by the dashed line with slope -1) up to approximately 1000 cores. Good scaling continues through the 64-node (7168-core) run. The speedup achieved by the 64-node `superB/NRPy` run relative to its 1-node counterpart (both using the `Charm++` code) is approximately 29x. More significantly, comparing the 64-node `superB/NRPy` run (wall time $\approx$ 33 s) to the original single-node `NRPy`-generated `OpenMP` code run on one node of WindRiver (using all 112 cores, wall time $\approx$ 1462 s for this large problem), the `superB/NRPy` code achieves a speedup of approximately 45x. This result demonstrates the substantial performance advantage enabled by the distributed-memory parallelization for executing large-scale NR simulations on HPC clusters. Because the `superB` (`Charm++` with MPI build) code outperforms the original `OpenMP` version on a single WindRiver node, we concluded that pursuing a hybrid `OpenMP`-MPI approach (`OpenMP` within a node and MPI across nodes)

is not worthwhile.

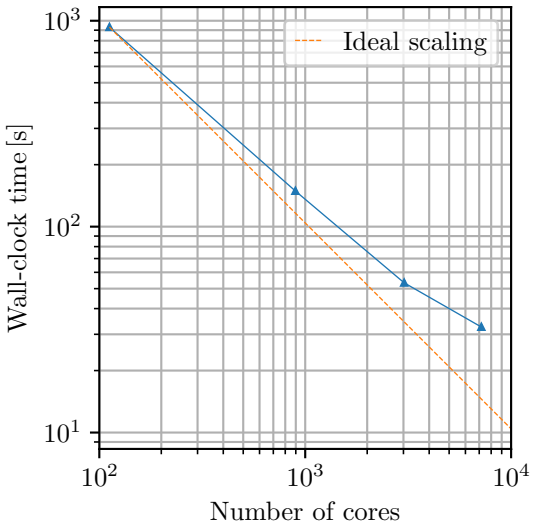

Figure 7. Strong scaling performance of `superB/NRPy` for the large 3D vacuum BH test case on the WindRiver HPC cluster (wall-clock time vs. number of cores). The number of chares is set equal to the number of cores. The dashed orange line indicates ideal scaling (slope -1). The speedup of the 64-node (7168 core) run relative to the 1-node (112 core) `superB/NRPy` run is approximately 29x. Compared to the original single-node `NRPy/OpenMP` code, the speedup at 64 nodes is approximately 45x.

5. Conclusions

The rapidly evolving field of gravitational-wave astronomy demands increasingly sophisticated and computationally intensive numerical relativity simulations. While the `NRPy` framework offers powerful tools for generating optimized C/C++ code for NR—particularly `BH@H` for curvilinear-coordinate simulations—its prior restriction to single-node `OpenMP` parallelism limited its applicability to large-scale astrophysical systems.

In this paper, we introduced `superB`, a major extension to the `NRPy` ecosystem designed to overcome this limitation. `superB` leverages the task-based parallelism features of the `Charm++` programming system to automatically generate scalable, distributed-memory C++ code directly from existing `NRPy/BH@H` physics modules. The resulting integrated framework, `superB/NRPy`, partitions the logically rectangular grids employed by `NRPy` into computational tasks (chares) distributed across multiple nodes and cores. Crucially, `superB` automatically generates the code to handle the requisite communication for ghost zone updates, including those associated with physical boundaries, coordinate singularities, and inter-task adjacencies, using local transfers, point-to-point messaging, and halo exchanges.

We validated the correctness and capabilities of the `superB/NRPy`-generated code through several tests. We demonstrated bit-identical numerical results compared to the

original OpenMP code when run on a single node with identical parameters and compiler, and performed a physically relevant head-on binary black hole collision using cylindrical-like coordinates. The extracted gravitational waveforms showed excellent agreement with analytical quasi-normal mode predictions (up to $\ell = 8$) during the ringdown phase. We also demonstrated seamless compatibility with recent algorithmic enhancements implemented within the shared NRPy framework, such as the SSL technique [64], which effectively suppressed constraint violations.

Performance benchmarks confirmed the excellent scalability of the superB/NRPy approach. While the original OpenMP code remains efficient for smaller problems on a single 16-core node, the Charm++-based code generated by superB/NRPy exhibits strong scaling across distributed-memory HPC systems. On the high-core-count WindRiver cluster, we measured a speedup of approximately 45x for a large 3D vacuum evolution running on 64 nodes (7168 cores) compared to the optimized single-node OpenMP code baseline. This demonstrates the effectiveness of superB/NRPy in enabling large-scale simulations.

The superB extension provides a robust and scalable foundation for the entire NRPy ecosystem. Its immediate impact includes enabling NRPy-generated codes for problems requiring significantly larger computational resources, such as exploring challenging binary parameter spaces (e.g., high spins or large mass ratios) or performing rapid follow-up simulation campaigns for significant gravitational-wave detections. The utility of this infrastructure was highlighted by the rapid integration (approx. two weeks) of Charm++ parallelism into the NRPy-generated GRoovy GRHD code [76]. This minimal effort was the direct result of superB being built specifically to interface with the BH@H framework. GRoovy was an existing code built within BH@H, and superB was engineered to provide a Charm++-based distributed-memory layer for such codes. Therefore, the core evolution equations and infrastructure were already compatible. The short integration period primarily consisted of implementing communication routines for non-evolved quantities, as is typically required for GRHD codes. We therefore anticipate that any future native NRPy-based codes will have even faster integration times. Key future technical developments will focus on integrating this distributed-memory capability with node-level GPU acceleration via the ongoing NRPy-CUDA project [77] and extending code generation to support the multi-patch, multi-coordinate grids vital for tackling long inspirals and complex accretion physics.

Ultimately, by providing pathways for both efficient single-node execution (suitable for efforts like the BlackHoles@Home volunteer computing project) and massively parallel scalability on HPC clusters, the NRPy framework equipped with the superB extension offers a unified, open-source, and extensible platform. It empowers next-generation gravitational-wave source modeling by effectively leveraging computational resources ranging from desktops to leadership-class supercomputers, tailored to the demands of the scientific application.

Acknowledgments

This work was primarily supported by the NASA ATP award 80NSSC22K1898. ZBE also gratefully acknowledges support from NSF awards OAC-2004311, OAC-2411068, PHY-2108072, and PHY-2409654. TPJ gratefully acknowledges support from NASA FINESST-80NSSC23K1437. This research made use of Idaho National Laboratory's High Performance Computing systems located at the Collaborative Computing Center and supported by the Office of Nuclear Energy of the U.S. Department of Energy and the Nuclear Science User Facilities under Contract No. DE-AC07-05ID14517. We also thank the `Charm++` developers for their support and advice. `Charm++`/Converse was developed by the Parallel Programming Laboratory in the Department of Computer Science at the University of Illinois at Urbana-Champaign.

References

[1] Abbott B P *et al.* (LIGO Scientific, Virgo) 2019 *Phys. Rev. X* **9** 031040 (*Preprint* 1811.12907)
[2] Abbott R *et al.* (LIGO Scientific, Virgo) 2021 *Phys. Rev. X* **11** 021053 (*Preprint* 2010.14527)
[3] Abbott R *et al.* (LIGO Scientific, VIRGO) 2024 *Phys. Rev. D* **109** 022001 (*Preprint* 2108.01045)
[4] Abbott R *et al.* (KAGRA, VIRGO, LIGO Scientific) 2023 *Phys. Rev. X* **13** 041039 (*Preprint* 2111.03606)
[5] Nitz A H, Capano C D, Kumar S, Wang Y F, Kastha S, Schäfer M, Dhurkunde R and Cabero M 2021 *Astrophys. J.* **922** 76 (*Preprint* 2105.09151)
[6] Baiotti L and Rezzolla L 2017 *Rept. Prog. Phys.* **80** 096901 (*Preprint* 1607.03540)
[7] Shibata M and Hotokezaka K 2019 *Ann. Rev. Nucl. Part. Sci.* **69** 41–64 (*Preprint* 1908.02350)
[8] Pan Y, Buonanno A, Buchman L T, Chu T, Kidder L E, Pfeiffer H P and Scheel M A 2010 *Phys. Rev. D* **81** 084041 (*Preprint* 0912.3466)
[9] Ramos-Buades A, Buonanno A, Estellés H, Khalil M, Mihaylov D P, Ossokine S, Pompili L and Shiferaw M 2023 *Phys. Rev. D* **108** 124037 (*Preprint* 2303.18046)
[10] Khan S, Husa S, Hannam M, Ohme F, Pürrer M, Jiménez Forteza X and Bohé A 2016 *Phys. Rev. D* **93** 044007 (*Preprint* 1508.07253)
[11] Blackman J, Field S E, Galley C R, Szilágyi B, Scheel M A, Tiglio M and Hemberger D A 2015 *Phys. Rev. Lett.* **115** 121102 (*Preprint* 1502.07758)
[12] Varma V, Field S E, Scheel M A, Blackman J, Gerosa D, Stein L C, Kidder L E and Pfeiffer H P 2019 *Phys. Rev. Research.* **1** 033015 (*Preprint* 1905.09300)
[13] Albanesi S, Gamba R, Bernuzzi S, Fontbuté J, Gonzalez A and Nagar A 2025 *arXiv preprint* ArXiv:2503.14580v1 (*Preprint* 2503.14580)
[14] Punturo et al 2010 *Classical and Quantum Gravity* **27** 194002
[15] Reitze D *et al.* 2019 *Bull. Am. Astron. Soc.* **51** 141 (*Preprint* 1903.04615)
[16] Evans M *et al.* 2021 A horizon study for cosmic explorer: Science, observatories, and community Technical Report CE-P2100003-v6 Cosmic Explorer Project (*Preprint* 2109.09882) URL https://dcc.cosmicexplorer.org/CE-P2100003/public
[17] Luo J *et al.* (TianQin) 2016 *Class. Quant. Grav.* **33** 035010 (*Preprint* 1512.02076)
[18] P Amaro-Seoane et al 2017 *arXiv e-prints* arXiv:1702.00786 (*Preprint* 1702.00786)
[19] Babak S, Petiteau A and Hewitson M 2021 Lisa sensitivity and snr calculations Technical Note LISA-LCST-SGS-TN-001 LISA Consortium (*Preprint* 2108.01167)
[20] Pürrer M and Haster C J 2020 *Phys. Rev. Research* **2** 023151 (*Preprint* 1912.10055)
[21] Ferguson D, Jani K, Laguna P and Shoemaker D 2021 *Phys. Rev. D* **104** 044037 (*Preprint* 2006.04272)

[22] Jan A, Ferguson D, Lange J, Shoemaker D and Zimmerman A 2024 *Phys. Rev. D* **110**(2) 024023 URL https://link.aps.org/doi/10.1103/PhysRevD.110.024023

[23] Amaro-Seoane P 2018 *Phys. Rev. D* **98** 063018 (*Preprint* 1807.03824)

[24] Foucart F, Laguna P, Lovelace G, Radice D and Witek H 2022 *arXiv e-prints* arXiv:2203.08139 (*Preprint* 2203.08139)

[25] Schnetter E, Hawley S H and Hawke I 2004 *Classical and Quantum Gravity* **21** 1465–1488 (*Preprint* gr-qc/0310042)

[26] Löffler F, Faber J, Bentivegna E, Bode T, Diener P, Haas R, Hinder I, Mundim B C, Ott C D, Schnetter E, Allen G, Campanelli M and Laguna P 2012 *Classical and Quantum Gravity* **29** 115001 ISSN 1361-6382 URL http://dx.doi.org/10.1088/0264-9381/29/11/115001

[27] Haas R *et al.* 2024 The Einstein Toolkit: The "Annie Jump Cannon" release (ET_2024_11) released on December 1, 2024 URL https://doi.org/10.5281/zenodo.14193969

[28] Pollney D, Reisswig C, Schnetter E, Dorband N and Diener P 2011 *Phys. Rev. D* **83** 044045 (*Preprint* 0910.3803)

[29] Brown D, Diener P, Sarbach O, Schnetter E and Tiglio M 2009 *Phys. Rev. D* **79** 044023 (*Preprint* 0809.3533)

[30] Sperhake U 2007 *Phys. Rev. D* **76** 104015 (*Preprint* gr-qc/0606079)

[31] Brügmann B, González J A, Hannam M, Husa S, Sperhake U and Tichy W 2008 *Phys. Rev. D* **77** 024027 (*Preprint* gr-qc/0610128)

[32] Cao Z, Yo H J and Yu J P 2008 *Phys. Rev. D* **78**(12) 124011 URL https://link.aps.org/doi/10.1103/PhysRevD.78.124011

[33] Clough K, Figueras P, Finkel H, Kunesch M, Lim E A and Tunyasuvunakool S 2015 *Classical and Quantum Gravity* **32** 245011 (*Preprint* 1503.03436)

[34] Fernando M, Neilsen D, Lim H, Hirschmann E and Sundar H 2019 *SIAM Journal on Scientific Computing* **41** C97–C138 (*Preprint* 1807.06128)

[35] Fernando M, Neilsen D, Zlochower Y, Hirschmann E W and Sundar H 2022 *arXiv e-prints* arXiv:2211.11575 (*Preprint* 2211.11575)

[36] Daszuta B, Zappa F, Cook W, Radice D, Bernuzzi S and Morozova V 2021 *Astrophys. J. Suppl.* **257** 25 (*Preprint* 2101.08289)

[37] Daszuta B and Cook W 2024 *arXiv e-prints* arXiv:2406.05126 (*Preprint* 2406.05126)

[38] Cook W, Daszuta B, Fields J, Hammond P, Albanesi S, Zappa F, Bernuzzi S and Radice D 2025 *Astrophys. J. Suppl.* **277** 3 (*Preprint* 2311.04989)

[39] Szilágyi B, Lindblom L and Scheel M A 2009 *Phys. Rev. D* **80** 124010 (*Preprint* 0909.3557)

[40] Hilditch D, Weyhausen A and Brügmann B 2016 *Phys. Rev. D* **93** 063006 (*Preprint* 1504.04732)

[41] Bugner M, Dietrich T, Bernuzzi S, Weyhausen A and Bruegmann B 2015 *arXiv e-prints* arXiv:1508.07147 (*Preprint* 1508.07147)

[42] Kidder L E, Field S E, Foucart F, Schnetter E, Teukolsky S A, Bohn A, Deppe N, Diener P, Hébert F, Lippuner J, Miller J, Ott C D, Scheel M A and Vincent T 2016 *arXiv e-prints* arXiv:1609.00098 (*Preprint* 1609.00098)

[43] Deppe N, Throwe W, Kidder L E, Vu N L, Nelli K C, Armaza C, Bonilla M S, Hébert F, Kim Y, Kumar P, Lovelace G, Macedo A, Moxon J, O'Shea E, Pfeiffer H P, Scheel M A, Teukolsky S A, Wittek N A *et al.* 2024 SpECTRE v2024.08.03 10.5281/zenodo.13207712 URL https://spectre-code.org

[44] Lovelace G, Nelli K C, Deppe N, Vu N L, Throwe W, Bonilla M S, Carpenter A, Kidder L E, Macedo A, Scheel M A, Afram A, Boyle M, Ceja A, Giesler M, Habib S, Jones K Z, Kumar P, Lara G, Melchor D, Mendes I B, Mitman K, Morales M, Moxon J, O'Shea E, Pannone K, Pfeiffer H P, Ramirez-Aguilar T, Sanchez J, Tellez D, Teukolsky S A and Wittek N A 2025 *Classical and Quantum Gravity* **42** 035001 (*Preprint* 2410.00265)

[45] Tichy W, Ji L, Adhikari A, Rashti A and Pirog M 2023 *Classical and Quantum Gravity* **40** 025004 (*Preprint* 2212.06340)

[46] Zhu H, Fields J, Zappa F, Radice D, Stone J, Rashti A, Cook W, Bernuzzi S and Daszuta B 2024

arXiv e-prints arXiv:2409.10383 (*Preprint* 2409.10383)
[47] Husa S, Hinder I and Lechner C 2006 *Computer Physics Communications* **174** 983–1004 (*Preprint* gr-qc/0404023)
[48] Palenzuela C, Miñano B, Arbona A, Bona-Casas C, Bona C and Massó J 2021 *Computer Physics Communications* **259** 107675 (*Preprint* 2010.00902)
[49] Ruchlin I, Etienne Z B and Baumgarte T W 2018 *Phys. Rev. D* **97** 064036 (*Preprint* 1712.07658)
[50] Pretorius F 2005 *Phys. Rev. Lett.* **95** 121101 (*Preprint* gr-qc/0507014)
[51] Campanelli M, Lousto C O, Marronetti P and Zlochower Y 2006 *Phys. Rev. Lett.* **96** 111101 (*Preprint* gr-qc/0511048)
[52] Baker J G, Centrella J, Choi D I, Koppitz M and van Meter J 2006 *Phys. Rev. Lett.* **96** 111102 (*Preprint* gr-qc/0511103)
[53] Garfinkle D 2002 *Phys. Rev. D* **65** 044029 (*Preprint* gr-qc/0110013)
[54] Pretorius F 2005 *Class. Quant. Grav.* **22** 425–452 (*Preprint* gr-qc/0407110)
[55] Shibata M and Nakamura T 1995 *Phys. Rev. D* **52**(10) 5428–5444 URL https://link.aps.org/doi/10.1103/PhysRevD.52.5428
[56] Baumgarte T W and Shapiro S L 1998 *Phys. Rev. D* **59** 024007 (*Preprint* gr-qc/9810065)
[57] Brown J D 2009 *Phys. Rev. D* **79** 104029 (*Preprint* 0902.3652)
[58] Zlochower Y, Ponce M and Lousto C O 2012 *Phys. Rev. D* **86** 104056 (*Preprint* 1208.5494)
[59] Etienne Z B, Baker J G, Paschalidis V, Kelly B J and Shapiro S L 2014 *Phys. Rev. D* **90** 064032 (*Preprint* 1404.6523)
[60] Black W K, Neilsen D, Hirschmann E W, Van Komen D F and Fernando M 2025 *arXiv e-prints* arXiv:2502.20282 (*Preprint* 2502.20282)
[61] Baumgarte T W, Montero P J, Cordero-Carrión I and Müller E 2013 *Phys. Rev. D* **87** 044026 (*Preprint* 1211.6632)
[62] 2025 NRPy GitHub repository https://github.com/nrpy/nrpy
[63] Meurer A *et al.* 2017 *PeerJ Computer Science* **3** e103:1–27 ISSN 2376-5992 URL https://doi.org/10.7717/peerj-cs.103
[64] Etienne Z B 2024 *Phys. Rev. D* **110** 064045 (*Preprint* 2404.01137)
[65] Montero P J and Cordero-Carrión I 2012 *Phys. Rev. D* **85** 124037 (*Preprint* 1204.5377)
[66] Kale L V and Krishnan S 1993 Charm++: A portable concurrent object oriented system based on C++ *Proceedings of the 8th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)* (Washington, D.C., USA: Association for Computing Machinery) pp 91–108 URL https://doi.org/10.1145/165854.165873
[67] The Charm++ Development Team 2025 Charm++: The Parallel Programming System https://github.com/UIUC-PPL/charm accessed: 2025-04-13
[68] Mewes V, Zlochower Y, Campanelli M, Ruchlin I, Etienne Z B and Baumgarte T W 2018 *Phys. Rev. D* **97** 084059 (*Preprint* 1802.09625)
[69] Mewes V, Zlochower Y, Campanelli M, Baumgarte T W, Etienne Z B, Lopez Armengol F G and Cipolletta F 2020 *Phys. Rev. D* **101** 104007 (*Preprint* 2002.06225)
[70] Ji L, Mewes V, Zlochower Y, Ennoggi L, Armengol F G L, Campanelli M, Cipolletta F and Etienne Z B 2023 *Phys. Rev. D* **108** 104005 (*Preprint* 2305.01537)
[71] Alic D, Bona-Casas C, Bona C, Rezzolla L and Palenzuela C 2012 *Phys. Rev. D* **85** 064040 (*Preprint* 1106.2254)
[72] Mösta P, Mundim B C, Faber J A, Haas R, Noble S C, Bode T, Löffler F, Ott C D, Reisswig C and Schnetter E 2014 *Class. Quant. Grav.* **31** 015005 (*Preprint* 1304.5544)
[73] 2019 Deprecated NRPy+ repository, hosted on bitbucket https://bitbucket.org/zach_etienne/nrpy
[74] Schnetter E, Hawley S H and Hawke I 2004 *Class. Quant. Grav.* **21** 1465–1488 (*Preprint* gr-qc/0310042)
[75] 2024 Carpet: Adaptive mesh refinement for the Cactus Framework http://www.carpetcode.org/
[76] Jacques T P, Cupp S, Werneck L R, Tootle S D, Hamilton M C B and Etienne Z B 2024 *arXiv*

preprint (*Preprint* 2412.03659)
[77] Tootle S D, Werneck L R, Assumpcao T, Jacques T P and Etienne Z B 2025 *arXiv preprint* (*Preprint* 2501.14030)
[78] Tichy W and Marronetti P 2007 *Phys. Rev. D* **76** 061502 (*Preprint* gr-qc/0703075)
[79] Marronetti P, Tichy W, Brügmann B, González J and Sperhake U 2008 *Phys. Rev. D* **77** 064010 (*Preprint* 0709.2160)
[80] Bona C, Massó J, Seidel E and Stela J 1995 *Phys. Rev. Lett.* **75** 600–603 (*Preprint* gr-qc/9412071)
[81] Alcubierre M, Brügmann B, Diener P, Koppitz M, Pollney D, Seidel E and Takahashi R 2003 *Phys. Rev. D* **67** 084023 (*Preprint* gr-qc/0206072)
[82] Schnetter E 2010 *Classical and Quantum Gravity* **27** 167001 (*Preprint* 1003.0859)
[83] Assumpção T, Werneck L R, Pierre Jacques T and Etienne Z B 2022 *Phys. Rev. D* **105** 104037 (*Preprint* 2111.02424)
[84] Gottlieb S, Shu C W and Tadmor E 2001 *SIAM Review* **43** 89–112 URL https://www.jstor.org/stable/3649684
[85] Ansorg M, Brügmann B and Tichy W 2004 *Phys. Rev. D* **70** 064011 (*Preprint* gr-qc/0404056)
[86] Alcubierre M 2003 *Classical and Quantum Gravity* **20** 607–623 (*Preprint* gr-qc/0210050)
[87] Alcubierre M 2005 *Classical and Quantum Gravity* **22** 4071–4081 (*Preprint* gr-qc/0503030)
[88] Stein L C 2019 *J. Open Source Softw.* **4** 1683 (*Preprint* 1908.10377)
[89] Anninos P, Hobill D, Seidel E, Smarr L and Suen W M 1993 *Phys. Rev. Lett.* **71**(18) 2851–2854 URL https://link.aps.org/doi/10.1103/PhysRevLett.71.2851
[90] Kalé L V, Zheng G, Lee C W and Kumar S 2006 *Future Generation Computer Systems* **22** 347–358 ISSN 0167-739X URL https://www.sciencedirect.com/science/article/pii/S0167739X04002262