A scalable framework for Adaptive Computational General Relativity on Heterogeneous Clusters

Milinda Fernando
School of Computing,
University of Utah.
milinda@cs.utah.edu

David Neilsen
Department of Physics and Astronomy,
Brigham Young University.
david.neilsen@byu.edu

Eric W. Hirschmann
Department of Physics and Astronomy,
Brigham Young University.
ehirsch@physics.byu.edu

Hari Sundar
School of Computing,
University of Utah.
hari@cs.utah.edu

ABSTRACT

We present a portable and highly-scalable framework that targets problems in the astrophysics and numerical relativity communities. This framework combines together the parallel Dendro octree with wavelet adaptive multiresolution and an automatic code-generation physics module to solve the Einstein equations of general relativity in the BSSNOK formulation. The goal of this work is to perform advanced, massively parallel numerical simulations of binary black hole and neutron star mergers, including Intermediate Mass Ratio Inspirals (IMRIs) of binary black holes with mass ratios on the order of 100:1. These studies will be used to study waveforms for use in LIGO data analysis and to calibrate approximate methods for generating gravitational waveforms. The key contribution of this work is the development of automatic code generators for computational relativity supporting SIMD vectorization, OpenMP, and CUDA combined with efficient distributed memory adaptive data-structures. These have enabled the development of efficient codes that demonstrate excellent weak scalability up to 131K cores on ORNL's Titan for binary mergers for mass ratios up to 100.

CCS CONCEPTS

• **Applied computing** → *Astronomy*.

KEYWORDS

computational relativity,automatic code generation, symbolic code generation, BSSNOK equations, HPC

ACM Reference Format:
Milinda Fernando, David Neilsen, Eric W. Hirschmann, and Hari Sundar. 2019. A scalable framework for Adaptive Computational General Relativity on Heterogeneous Clusters. In *2019 International Conference on Supercomputing (ICS '19), June 26–28, 2019, Phoenix, AZ, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3330345.3330346

ICS '19, June 26–28, 2019, Phoenix, AZ, USA

ACM ISBN 978-1-4503-6079-1/19/06…$15.00
https://doi.org/10.1145/3330345.3330346

1 INTRODUCTION

We present a portable and highly-scalable algorithm and framework that targets problems in the astrophysics and numerical relativity communities. The key challenges in developing such codes are related to the complexity of the underlying equations with a need to solve partial differential equations for dynamic, curved spacetimes, the need for extreme scales across the domain and extreme refinement at the location of black holes, and the need to perform extremely long simulations (in terms of the number of timesteps). These translate to the need for a simple interface to encode the mathematical formulations, large-scale parallelism to handle the extreme scale, highly localized refinement or adaptivity and extreme performance and scalability on modern heterogeneous architectures. This work addresses and demonstrates all these qualities in our state-of-the-art computational relativity framework Dendro-GR for the simulation of gravitational waves resulting from binary black hole mergers.

In 2015, shortly after beginning its first observing run, the Laser Interferometer Gravitational-Wave Observatory (LIGO) [1, 70] made the first direct detection of gravitational waves from the merger of two black holes [5]. Since that time, gravitational waves from ten other binary black hole mergers [4, 6, 9, 10] have been detected by LIGO and the European Virgo detectors [2, 12]. In August 2017, LIGO/Virgo detected gravitational waves from the merger of a neutron star binary [3]. This latter detection was particularly exciting because of electromagnetic radiation from the resulting gamma-ray burst was detected by the Fermi Gamma-Ray Burst Monitor [34], INTEGRAL [67], and several other observatories [8]. The combination of gravitational and electromagnetic observations of binary mergers are giving new insight into the physics of black holes (BH) and neutron stars (NS) [7, 11, 13].

Gravitational waves carry the imprint of their origins in the complicated pattern of their waveform. The information therein can be extracted through a careful comparison of the gravitational wave signal with a library of possible waveforms constructed using approximate methods and results from numerical simulations. Waveform information from numerical relativity is particularly important for certain black hole binary configurations, including black hole binaries with arbitrary spin configurations [19], orbital eccentricity, and dissimilar masses [36, 44]. With regard to the latter case–referring to the the mass ratio of a binary as $q \equiv m_1/m_2$,

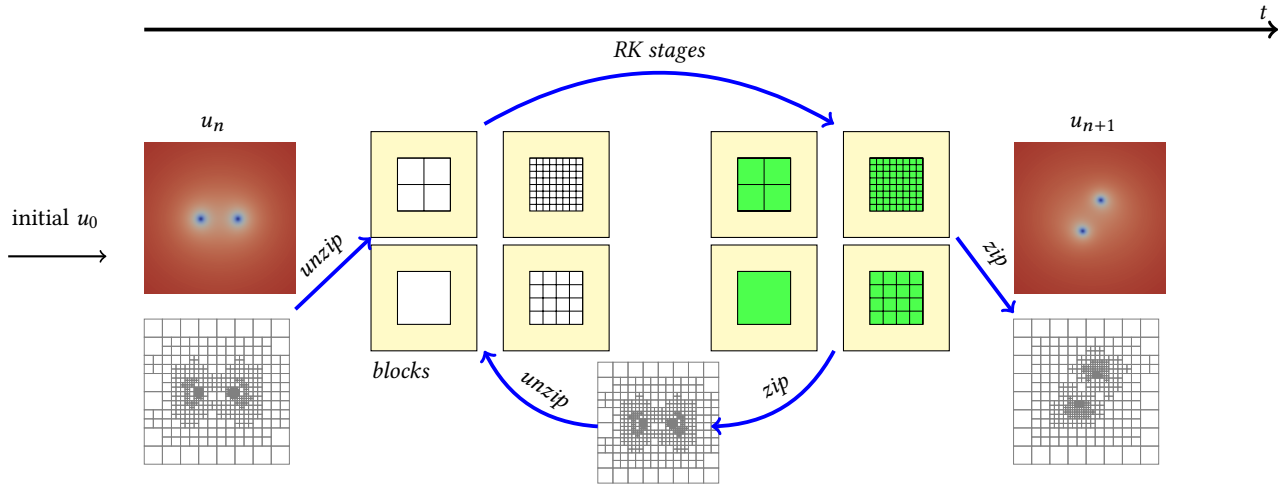

Figure 1: This figure illustrates the calculation of a single Runge-Kutta (RK) time step, computing the solution at the advanced time, u_{n+1}, from data at the previous time step, u_n. For computational efficiency, spatial and time derivatives are evaluated on equispaced blocks *(unzipped)*; a sparse grid constructed from wavelet coefficients is used for communication and to store the final solution *(zipped)*. For each RK stage s we perform the *unzip* operation which results in a sequence of blocks which are used to compute the solution on the internal block (■), using the padding values at the block boundary (■) followed by a *zip* operation in between RK stages while the final update (i.e. next time step) performed using the *zip* version of the variables. All application code at the block level is autogenerated and currently supports SIMD vectorization, OpenMP and CUDA.

where m_1 is the mass of the larger BH–nearly equal mass ($q \approx 1$) BH binaries have been extensively studied in recent years [49, 72]. Codes developed for these binaries are accurate and well tuned, so the problem is well-understood and numerical results are confidently used in the LIGO data-analysis pipeline. However, configurations with larger q remain mostly beyond the capabilities of current techniques in numerical relativity.

The size of the smaller black hole adds an extra length-scale to the problem, compared to the $q \approx 1$ case. The need to resolve this scale, together with the large range of other important length scales for the binary system, make this a very challenging computational problem. It requires a highly adaptive and efficient computational algorithm tuned to binaries in this region of parameter space. The small spatial scale also requires an equally small time step, requiring an increasing number of simulation steps as q becomes larger. This necessitates the need for highly efficient codes on modern supercomputers that are increasingly heterogeneous in nature. *The central goal of our effort is to create a general purpose code to study the evolution of spacetimes with black holes or neutron stars, including black hole binaries with $q \simeq 100$.* Here we present our portable, highly-scalable, extensible, and easy-to-use framework for general relativity (GR) simulations that will be forward-compatible with next-generation heterogeneous clusters.

We build on our octree-based adaptive mesh refinement (AMR) framework Dendro[27]to support the Wavelet Adaptive Multiresolution Representation (WAMR) [29, 60, 61]. The fast wavelet transform can be used to create a sparse representation of functions that retains sharp features and an *a priori* error bound. We use an efficient block-decomposition of the distributed octree to produce a collection of overlapping regular grids (at different levels of refinement) §3.3.1. The block decomposition decouples the distributed memory parallelism using `MPI` from the heterogeneity that can exist within each node. We assume that at the finest level each node will have at least 1 complete block, although it is possible to specify the size of the largest block. The Einstein equations of general relativity describe the spacetime geometry and, expressed in terms of the BSSNOK formulation [54, 69], there are 24 degrees of freedom at each spatial location. Given their complexity and a desire for portable code, we auto-generate the core computational kernels automatically from the equations written in symbolic Python (`SymPy` [43]) (§3.4). The auto-generated code is applied at the block-level and is therefore very efficient and addresses the issue of portability and heterogeneity within the node. Our code generation currently supports vectorization (`avx, avx2`), OpenMP and CUDA. The equations are integrated in time using the method of lines with a third-order Runge-Kutta (RK) integrator. The high-level overview of our approach is illustrated in Figure 1.

The key contributions of this work include:

Wavelet Adaptive GR. This project builds on the Dendro AMR library and GR software packages (for uniform block grids) that have been developed in our group, with the addition of Wavelet Adaptive Multiresolution (WAMR). This is the first highly adaptive computational fully relativistic—i.e., including the full Einstein equations—code with an arbitrary localized adaptive mesh. Such fine-grained adaptivity makes it harder to achieve high-performance on heterogeneous architectures and reduces code portability. This motivates the main contribution of this work.

Automatic code-generation. Given the complexity of the Einstein equations, we have developed an automatic code generation framework for GR using `SymPy` that automatically generates architecture-optimized codes. This greatly improves code portability, use by domain scientists and the ability to add additional constraints and checks to validate the code. The auto-generated

code only needs to support shared memory parallelism, since the distributed memory parallelism is handled by Dendro. We demonstrate generators for vectorization, OpenMP and CUDA as well as an asynchronous operation with work divided across CPU and GPU cores, along with overlapped communication with computation.

Implementation Dendro is implemented in C++ using MPI except for the automatic code generation framework which is implemented using `SymPy`. We provide code generators for `C++` including shared memory parallelization using OpenMP and vectorization `avx, avx2` and CUDA. We expect to release our code on Github under the MIT license upon acceptance.

The rest of the paper is organized as follows. In §2 we summarize the motivating application and review related work. In §3, we provide details about our computational relativity framework focusing on the specific contributions of this work. In §4 we provide experimental results demonstrating the efficacy of our methods and the scalability and portability of our code. Finally, we conclude with directions for future research in §5.

2 BACKGROUND

In this section, we discuss the motivating applications and summarize the most relevant work of other groups in this area. As gravitational waves pass through the Earth, their effect on matter is extremely small. LIGO searches for gravitational waves by using laser interferometry to detect changes in the relative position of the mirrors, to a precision of four orders of magnitude smaller than an atomic nucleus. The gravitational wave signals in the detector are often smaller in magnitude than noise from other sources, but the signals can be extracted using matched filtering [66], which uses a library of hundreds of candidate waveforms that are convolved with the data to search for matches. Including complete numerical waveforms in the waveform library is important to maximize the detection rate of LIGO-class detectors [71].

The Einstein equations of general relativity describe how the geometry of spacetime curves in response to the presence and motion of matter and energy. The Einstein equations contain both hyperbolic evolution equations and elliptic constraint equations. Commonly, the hyperbolic equations are solved while the elliptic equations are used to monitor the quality of the solution [14, 68]. While the equations can be formulated in many different ways, a few formulations are well adapted for numerical work. One such formulation is the BSSNOK formulation [20]. The BSSNOK evolution equations are a set of strongly hyperbolic [65] coupled PDEs that are first-order in time and second-order in space.

Several computer codes have been developed to solve the Einstein equations for binary BH and neutron star systems. One of the oldest open source projects in this community is the Cactus Computational Toolkit [22, 35], that provides a modular infrastructure for solving time-dependent PDEs in parallel using structured grids. Modules for specific tasks, known as *thorns* in Cactus parlance, can be shared and combined to produce a sophisticated evolution code. The Einstein Toolkit (ET) is a suite of community-developed thorns for relativistic physics [32]. It includes thorns for constructing binary BH initial data, for evolving the Einstein equations and/or the relativistic fluid equations, and for data analysis. Similar codes include [33, 46, 47, 56, 74]. Further, the SXS collaboration has developed SpEC [73], a spectral code for solving the Einstein equations that has produced the longest and most-accurate binary waveforms to date.

The challenge of running on modern massively parallel computers is pushing new developments in numerical relativity. The use of structured grids with block-based AMR, such as used by Cactus/ET and similar codes, is not ideal for new massively parallel architectures. These approaches can lead to inefficient refinement, especially for $q \gg 1$. One new approach for the ET is the SENR project [51, 64], that uses coordinate systems adapted to the binary to eliminate the need for AMR. Another approach is to use discontinuous Galerkin (DG) methods, that requires less communication between processes. The first three-dimensional ADER-DG simulations of the Einstein equations were performed by Dumbser et al. [31]. The SXS collaboration is developing the SpECTRE code [45, 63] that uses task-based parallelism and DG. Thus far only results for the relativistic MHD equations have been published. To the best of our knowledge none of these codes support GPUs due to the complexity of the equations and the need to autogenerate the code.

We have chosen to focus on one type of BH binary that is particularly difficult to study both numerically and with semi-analytical approximations. These are Intermediate Mass-Ratio Inspiral (IMRIs), BH binaries with mass ratios between $50 \lesssim q \lesssim 1000$. The successful numerical simulation of IMRIs and their predicted gravitational wave signal is difficult because of the larger difference in the two mass-scales in the problem. Gravitational waves must be resolvable far from the binary system while the region around both black holes must also be accurately simulated. Standard approaches to black hole simulations often include mesh adaptivity by which necessary resolution is concentrated in dynamic regions.

Dendro-GR uses unstructured grids based on the wavelet expansion [41, 60, 61, 75, 76], that produces refinement regions adapted to features in the solution with a minimum number of points. This is important for problems with fine-scale features that are not spatially localized, or problems with widely disparate scales, such as IMRIs. Moreover, the numerical methods are based on the conventional finite difference and finite volume methods that have been widely used and tested (see, Sec. 3.1). This allows previously written code to be more easily adapted to the Dendro framework. Given the scale of our problem, even with adaptivity, massively parallel computing resources are required.

A key reason to develop scalable codes is that as the relative difference in masses becomes larger ($\sim 100\times$), the computational requirements will grow significantly, potentially requiring exascale resources. A simple calculation illustrates how spatial resolution requirements increase with q. A convenient measure for the size of a black hole is the radius of its event horizon, which is proportional to its mass. In a black hole binary, the mass of the smallest black hole effectively sets the minimum length scale for the simulation. The total mass of the binary $M = m_1 + m_2$ is a global scaling parameter and is typically fixed to a constant value. Then the mass of the smaller black hole can be written $m_2 = M/(q+1)$, showing that the minimum resolution scale for the binary is inversely proportional to the mass ratio. Thus, in three spatial dimensions the number of points required to resolve the smallest black hole grows as q^3. This

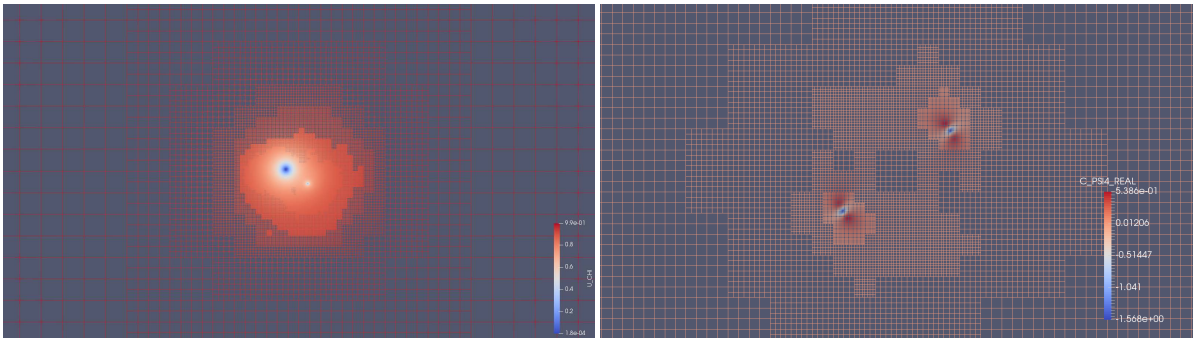

Figure 2: The above figure depicts underlying wavelet adaptive octree grid structure at time t for binary black hole simulations of mass ratios of 10 (left) & 1 (right) performed by Dendro-GR .

presents both a challenging problem in computational relativity as well as a challenge for high-performance computing. Additionally, since most modern HPC machines are heterogeneous, and rely on GPUs for a significant portion of their computational capability, support for such platforms is important. The successful scaling of our heterogeneous, adaptive, GR code is the first step in this direction.

3 METHODOLOGY

Research in relativistic astrophysics requires specialized computational models for gravitational, plasma, and nuclear physics. The massively parallel infrastructure that we propose is compatible with the standard finite difference or finite volume discretizations that are currently used in these communities. We solve the BSSNOK equations using conventional finite-difference discretizations, standard gauges, and puncture initial data (see § 3.1). We also generate C++ code automatically for CPUs–OpenMP parallel with vectorization–and for Nvidia GPUs using CUDA. While this paper focuses on the vacuum Einstein equations, we are currently working to add modules for the relativistic MHD equations, nuclear equations of state, and neutrino leakage. In §3.1-3.3, we briefly give an overview of the overall computational framework and then focus on the methods that are specific to this work.

3.1 Numerical Methods

There is extensive literature on solving the BSSNOK equations in general relativity, and some general reviews include [14, 62, 68]. In this section, we briefly outline our particular choices for solving these equations. We write the BSSNOK equations in terms of the conformal factor χ [23]. We use the parameterization of the "1+log" slicing condition and the Γ-driver shift used in [56]. Spatial derivatives are calculated using finite difference operators that are $O(h^4)$ in the grid spacing, h, with upwind derivatives for Lie derivative terms [78]. We calculate derivatives for the Ricci tensor and enforce the algebraic constraints as described in [21]. Outgoing radiative boundary conditions are applied to each BSSNOK function. The BSSNOK equations are integrated in time using an explicit Runge-Kutta (RK) scheme. The solution at each point is integrated with a single global time step that is set by the smallest grid spacing and the Courant condition [26]. The tests in this paper were done using third order RK with Courant factor $\lambda = 0.1$. Kreiss-Oliger dissipation is added [14, 48] to the solution to eliminate high-frequency noise that can be generated near the black hole singularities.

3.2 Wavelet Adaptive Multiresolution

WAMR uses a basis of interpolating wavelets to create a sparse, unstructured grid that naturally adapts to the features of the solution [29, 60, 61]. This grid adaptivity is realized by expanding functions using the fast wavelet transform [41], and thresholding the solution to create a sparse representation that retains small-scale features [30]. Wavelet basis functions are localized both spatially and with respect to scale. In comparison, spectral bases are infinitely differentiable, but have global support; basis functions used in finite difference or finite element methods have small compact support, but poor continuity properties. As an example, in Figure 2 we show a binary black hole spacetime generated with WAMR using Dendro-GR.

3.3 Adaptivity & Distributed Memory Parallelism

We build on our octree-based adaptive meshing framework Dendro[27] Dendro is a scalable distributed memory library written in C++ using MPI. Dendro supports generating 2:1 balanced octrees, i.e., octrees where neighbouring octants differ in size by at most a factor of two. We have extended this work to support high-order finite differencing and finite volume computations as well as higher order finite element computations. Finite difference computations on adaptive grids are achieved via an *unzip* operation that decomposes the distributed octree into a set of small, node-local structured data blocks. These blocks are then processed and restored to the distributed data-structure using a *zip* operation. This decouples the distributed memory parallelism that deals with the octree data-structures and performs data-exchange in the compressed format. The node-local computations are performed on the structured data blocks, making it easier to ensure portability of the code.

3.3.1 Zip - Unzip. The main objective of the *unzip* and *zip* operations is to enable stencil computations on a given adaptive grid. The distributed octree can be decomposed into a set of regular grid blocks of different sizes–basically a set of octants that are all at the same level of refinement. Due to the memory allocation and performance, we enforce block sizes to be powers of two. In

order to perform stencil operations on these blocks, we need information from neighboring blocks, similar to how we need ghost layers in the distributed case. In the context of blocks, we refer to this layer as the padding. During meshing, we compute and save the octree-to-block decomposition, i.e., in which octants are grouped together as a block. The computation of octree-to-block decomposition primarily involves a top-down traversal over the local octants, and stopping when all elements in the block are at the same level. The block membership of elements is stored and used during the *zip* and *unzip* operations. All simulation variables are stored in their most compact or *zipped* representation, i.e., without any duplication. During the *unzip* operation, we convert the *zipped* representation to the block representation with padding, the *unzipped* representation. This involves copying the data within the block, and copying–potentially with interpolation–from neighboring octants. All block internal nodes are non-hanging by construction and can be directly copied. Nodes on the block boundary and padding might be hanging hence might need to be interpolated during the copy. The 2:1 balance condition guarantees that at most a single interpolation is performed for any given octant. In our implementation, we overlap the communication of ghost nodes with *unzip*, which improves the scalability of our approach. The stencil and other update operations are only performed on the block as the padding is read-only. At the end of the update, the simulation variables are *zipped* back, i.e., injected back to the *zipped* representation. This step does not involve any interpolations or communication and is very fast. Note that several key operations such as RK update & inter-process communications operate using the *zip* representation, and are extremely efficient.

3.4 Symbolic interface

The Einstein equations are a set of non-linear, coupled, partial differential equations. On discretization, one can end up with 24 or more equations with thousands of terms. Writing, optimizing and maintaining code for this is very challenging. Sustainability and keeping it relevant for new architectural changes are additional difficulties. To address these issues, we have developed a symbolic interface to Dendro. We leverage symbolic python (`SymPy`) as the backend for this along with the python package `cog` to embed python code within our application-level `C++` code. The `Dendro_sym` package allows us to write the discretized versions of the equations similar to how they are written mathematically and enable improved usability for non-computational scientists. `Dendro_sym` provides several functions supporting curved geometries, including computing Lie derivatives, Christoffel symbols, the Ricci tensor etc. This greatly simplifies programming computational relativity formulations in curved spacetime. An example for the BSSNOK equations are shown in Figure 3, with the equations on the left and the corresponding python code on the right.

3.5 Code Generation

There are several advantages to using a symbolic interface like `Dendro_sym` for the application-specific equations. First, it greatly improves the sustainability of the code by separating the high-level description of the equations from the low-level optimizations, which can be handled by architecture-specific code generators. Note that, there are several significant attempts such as Kranc[42] and NRPy [64] on symbolic code generation for computational relativity due to the complexity of the BSSNOK equations. Our model of using structured blocks at the node-level and separation from distributed memory parallelism makes it easy to support additional architectures. We currently support `avx,avx2`, OpenMP and `CUDA` code-generators. Since these are applied at a block level, it is straightforward to schedule these blocks across cores or GPUs. Note that the auto-generated code consists of several derivative terms that are spatially dependent as well as other point-wise update operations. We perform common subexpression elimination (CSE) [25] to minimize the number of operations (see Figure 4). Additionally, we auto-vectorize the pointwise operations and have specialized implementations based on the stencil-structure for the derivative terms. We now describe the strategy for each of the currently supported generators.

3.5.1 Custom Functions: Derivative Computations. Compared to Euclidean space computations, GR computations can be complicated due to the number of variables and curvature tensors involved. Evaluating the BSSNOK equations on a single grid point requires 282 derivative computations on different variables. The derivative computations are incorporated into the symbolic framework through the use of custom functions. Currently, for the GR equations in vacuum, we have `grad`, `grad2`, `adv` and `kograd` where these functions represent first derivative (in direction i, 5 point stencil), second derivative (in direction ij, 25 point stencil), upwind/downwind (in direction i, which is used to evaluate $\mathcal{L}_\beta$, 7 point stencil) and Kriess-Oliger dissipation operators (on direction i, 7 point stencil). The approach taken towards the computing and storage of derivative stencils must be optimized to exploit the memory hierarchies on different architectures.

3.5.2 SIMD vectorization. The RHS computation consists of two distinct data-access patterns, derivative computations that are standard stencil computations and depend on spatial locality, and the RHS expressions that use the derivatives and the evolved variables in a pointwise fashion. The focus of this work is on the second part, and we use hand-tuned vector code for all stencil operations. Several efficient packages exist [24, 38, 39] for vectorizing stencil codes, and can be easily incorporated within our framework. While our code-generation framework is capable of generating stencil codes, our focus in this work is to automatically vectorize the RHS expressions. Since these are applied in a pointwise fashion, it is straightforward to generate the vectorized `load` and `stream/store` commands given the length of the SIMD register and the dimensions of the datablock. The complexity for the generation is related to the depth of several of the expression tree and the number of dependencies that a given expression might have. The expression tree is analyzed and pattern matching is used to generate corresponding intrinsic code. Currently the following transformations are performed,

- Replace conditional statements with algebraic expressions. This is needed for advective derivatives with upwinding where the stencils are dependent on the sign of another variable. We use the vectorized compare functions (*e.g.* `_mm256_`

$$
\begin{aligned}
\partial_t \alpha &= \mathcal{L}_\beta \alpha - 2\alpha K, \\
\partial_t \beta^i &= \lambda_2 \beta^j \partial_j \beta^i + \frac{3}{4} f(\alpha) B^i \\
\partial_t B^i &= \partial_t \tilde{\Gamma}^i - \eta B^i + \lambda_3 \beta^j \partial_j B^i - \lambda_4 \beta^j \partial_j \tilde{\Gamma}^i \\
\partial_t \tilde{\gamma}_{ij} &= \mathcal{L}_\beta \tilde{\gamma}_{ij} - 2\alpha \tilde{A}_{ij}, \\
\partial_t \chi &= \mathcal{L}_\beta \chi + \frac{2}{3} \chi \left(\alpha K - \partial_a \beta^a\right) \\
\partial_t \tilde{A}_{ij} &= \mathcal{L}_\beta \tilde{A}_{ij} + \chi \left(-D_i D_j \alpha + \alpha R_{ij}\right)^{TF} + \\
&\quad \alpha \left(K \tilde{A}_{ij} - 2 \tilde{A}_{ik} \tilde{A}^k_j\right), \\
\partial_t K &= \beta^k \partial_k K - D^i D_i \alpha + \\
&\quad \alpha \left(\tilde{A}_{ij} \tilde{A}^{ij} + \frac{1}{3} K^2\right), \\
\partial_t \tilde{\Gamma}^i &= \tilde{\gamma}^{jk} \partial_j \partial_k \beta^i + \frac{1}{3} \tilde{\gamma}^{ij} \partial_j \partial_k \beta^k + \beta^j \partial_j \tilde{\Gamma}^i - \\
&\quad \tilde{\Gamma}^j \partial_j \beta^i + \frac{2}{3} \tilde{\Gamma}^i \partial_j \beta^j - 2 \tilde{A}^{ij} \partial_j \alpha + \\
&\quad 2\alpha \left(\tilde{\Gamma}^i_{jk} \tilde{A}^{jk} + 6 \tilde{A}^{ij} \partial_j \phi - \frac{2}{3} \tilde{\gamma}^{ij} \partial_j K\right)
\end{aligned}
$$

```
from dendro_sym import *

a_rhs = dendro.Lie(b, a) - 2*a*K

b_rhs = [3/4 * f(a) * B[i] +
l2*vec_j_del_j(b, b[i]) for i in e_i]

B_rhs = [Gt_rhs[i] - eta * B[i] +
l3 * vec_j_del_j(b, B[i]) -
l4 * vec_j_del_j(b, Gt[i])
for i in e_i]

gt_rhs =  dendro.Lie(b, gt) - 2*a*At

chi_rhs = dendro.Lie(b, chi) +
2/3*chi*(a*K - del_j(b))

At_rhs = dendro.Lie(b, At) + chi *
dendro.TF(-DiDj(a) + a*dendro.Ricci) +
a*(K*At -2*At_ikAtKj)

K_rhs = vec_k_del_k(K) - DIDi(a) +
a*(1/3*K*K + A_ij_A_IJ(At))
```

Figure 3: The left panel shows the BSSNOK formulation of the Einstein equations. These are tensor equations, with indices $i, j, \ldots$ taking the values $1, 2, 3$. On the right we show the `Dendro_sym` code for these equations. `Dendro_sym` uses `SymPy` and other tools to generate optimized `C++` code to evaluate the equations. Note that $\mathcal{L}_\beta$, D, ∂ denote Lie derivative, covariant derivative and partial derivative respectively, and we have excluded $\partial_t \Gamma^i$ from `Dendro_sym` to save space. (See [14, 20] for more information about the equations and the differential operators.)

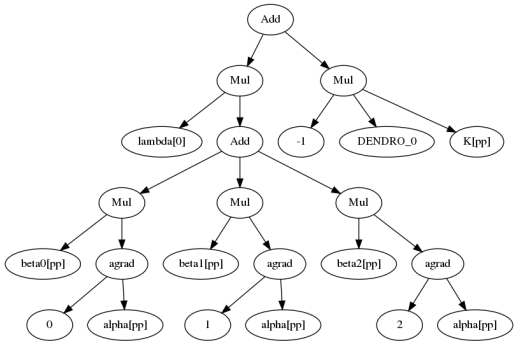

Figure 4: Computational graph for $\partial_t \alpha$ generated after Common Sub-expression Elimination (CSE) for the entire BSSNOK system of equations. Note that the prefix, `Dendro_`, denotes sub-expressions frequently used for evaluation of the BSSNOK equations. Simpler equations, such as $\partial_t \alpha$, result in smaller graphs while complex equations which involve the extrinsic curvature tensor (K_{ij}) can result in larger, more complicated graphs with as many as $250 - 350$ vertices. During the code generation phase, CSE is mainly utilized for reducing the number of FLOP needed for BSSNOK evaluation.

`cmp_pd`) to convert the comparison into a vectorized variables with 1s and 0s, and solve an equation in 2 variables to get the corresponding vectorized expression.

- Replace functions with specialized implementations if available. These are cases where hand-written codes are available, such as the derivative functions.
- Replace integer powers by repeated multiplications. Our expressions contain a large number of integer powers, that within SymPy get generated using the `pow` function. We preprocess to replace these using multiplications and divisions (for negative powers).
- SymPy `MUL` and `ADD` operators support multiple arguments, whereas the intrinsics only support two at a time, so we split these expressions in a binary tree fashion.
- We perform a number of simplifications involving SymPy `MUL/DIV` and `MUL` by -1 followed by `ADD` in lieu of subtraction. Since subtraction intrinsics are available, these patterns are identified from the SymPy expression and converted to subtraction intrinsics.
- Convert $ab + c$ into Fused-Multiply-Add (`FMA(a,b,c)`). Similarly for subtraction.

3.5.3 OpenMP. We use a simple strategy for OpenMP parallelization. Since in the typical use case, the number of data-blocks per node is much larger than the number of available threads, these blocks are scheduled across the threads using `omp parallel for` with dynamic scheduling with a chunk size of 1. It is easy to change this parallelization strategy for cases where we have a larger number of cores. In such cases, the OpenMP parallelization is pushed one level deeper into the outmost loop of the block computation.

3.5.4 CUDA. Due to the limited shared memory available in GPUs, generating GPU specific code is more involved compared to CPU

code generation. The complete input and output variables (24×2) do not fit in the GPU shared memory even for the smallest block size of 11^3. GPU shared memory management is an active research area in the field, and there are significant attempts [40, 52] at stencil code generation for GPU architectures. Performing stencil computations while keeping the data on the GPU global memory is highly inefficient, therefore we use sub blocks of size m^3, where m depends on the GPU architecture. GPU code is generated using patterns such that *load* (global to shared memory), *compute* and *store* (shared to global memory) aims to minimize the number of global memory accesses. For example once a variable is loaded from global to shared memory (say α), all the required derivatives are computed ($\partial_x\alpha, \partial_y\alpha, \partial_z\alpha$), avoiding the need to load α again for stencil computations.

Stencil operations : For a given symbolic variable that needs to be evolved, `Dendro_sym` evaluates the SymPy expression tree and computes its dependence on derivative terms (i.e. stencils applied to input variables). In order to maximize the data reuse, we perform `load` operations from global memory to shared memory for a specific tile (T_m), compute all the derivatives required by applying high-order stencils in shared memory and subsequently `store` the computed stencil from shared memory to global memory. For example, consider the BSSNOK variable α. After the load operation all the derivatives `grad_i`, `agrad_i`, `kograd_i` and `grad_ij` for $i, j \in \{1, 2, 3\}$ are computed (see Figure 5). Note that in order to evaluate the BSSNOK formulation at a single grid point, we need to perform 282 stencil operations. In order to reduce the global memory footprint by derivative variables, we use a reusable derivative workspace where the size of the workspace is bounded by the maximum data block size and the number of `SM` units in the GPU.

BSSNOK evaluation: Once all the derivatives are computed, we can proceed to evaluate the BSSNOK equations at each grid point. Note that due to the limited shared memory it is not feasible to compute all the BSSNOK equations at once. Therefore we compute each BSSNOK equation separately in a staged manner. The dependencies and therefore the memory footprint for each variable is different and the code-generator automatically generates appropriately tiled code based on the architectural parameters. An outline of the CUDA code for the evaluation of BSSNOK equation is provided in the algorithm 1 to illustrate the overall structure of the generated CUDA code.

Algorithm 1 GPU kernel for BSSNOK evaluation

```
D ← malloc()                                   ▷ allocate memory for derivatives
for do w ∈ SymVarDerivs
    __shared V1 ← loadG2S(w)
    __shared V2 ← apply_stencil(V1)
    storeS2G(V2, D[w])
for w ∈ SymEqus do
    for v, Dv ∈ Dependencies(w) do             ▷ variable (v) & derivative(Dv)
        __shared V_i ← loadG2S(v)
        __shared V_j ← loadG2S(Dv)
    w_rhs ← compute(w)                         ▷ shared mem. computation
    storeS2G(w_rhs)
```

3.6 Parallelizing across CPU and GPUs

The automatic code generation is also setup to perform computations in a true heterogeneous manner. During meshing, while the octree-to-block decomposition is computed (§3.3.1) we also tag the blocks as being on the inter-process boundary or being internal. The CPU initiates asynchronous transfer of data in the ghost region (via MPI). Simultaneously, it initiates a data transfer stream to transfer the interior blocks to the GPU followed by an additional stream to compute the RHS. The GPU then transfers the computed RHS update variables back to the CPU. By the use of streams, we get very good overlap of data-transfer with computation. Once the ghost layer is obtained, we have two options. Firstly, the CPU can process while the GPU processes the interior blocks. Alternatively, if the compute capabilities are heavily in favor of the GPU, then it is possible to transfer the boundary blocks to the GPU as well at this stage. The second strategy is illustrated in Figure 6.

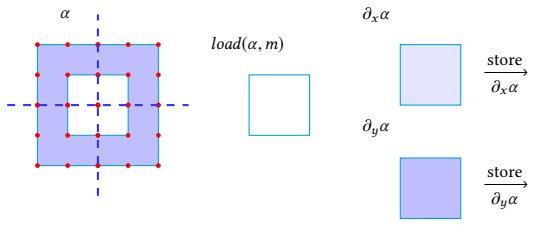

Figure 5: An example of *load, compute* and *store* for derivative computation for BSSNOK variable α. Depending on the GPU architecture, specific sub-block size is determined during code generation. The number of iterations required to process the entire data block will depend on the computed tile size. Note that for simplicity, in the figure only ∂_x and ∂_y computation is shown, but during actual code generation the derivatives required for each variable are determined and computed after a single *load* operation and executing a *store* operation for each derivative that has computed.

3.7 Putting it all together

Algorithm 2 Overview of our approach

```
M ← initialize mesh
u ← initialize variables (M)
while t < T do
    for r = 1 : 3 do                           ▷ Runge-Kutta stages
        B, û ← Unzip(M, u)                     ▷ §3.3.1
        for b ∈ B do
            Compute derivatives                ▷ Machine generated code §3.4
            Compute û_rhs(b)                   ▷ Machine generated code §3.4
        u_rhs ← Zip(M, B, û_rhs)               ▷ §3.3.1
        RK update
    t ← t + dt
    if need remesh M then
        M' ← remesh(M)
        u' ← Intergid_Transfer(M, M', u)
```

We use a 3^{rd} order Runge-Kutta time stepper, to perform time evolution. A given RK stage is computed by performing *unzip* operations with overlapped exchange of the ghost layer for the evolution variables, computation of the derivatives and right-hand-side (rhs) (using the code generated by the symbolic framework) for all local blocks and finally performing a *zip* operation to get the computed *zipped* rhs variables. The RK update is then performed on the *zipped*

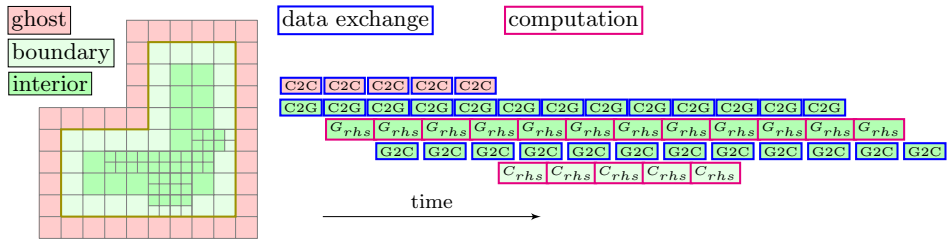

Figure 6: Illustration of the overlapped data-exchanges with computation. On the left a sub-domain assigned to a single node is illustrated, along with the ghost region in red. The subdomain is separated into *boundary* and *interior* data blocks. The CPU initiates ghost-layer data exchange with other nodes via `MPI` and simultaneously initiates a stream for asynchronous data transfer of the interior blocks to the GPU. The GPU starts processing the data blocks as they arrive in an asynchronous manner and transfers the results back to the CPU. On completion of the ghost exchange, the CPU processes the boundary blocks. In this figure, the data-exchange blocks are outlined in blue and the computation blocks in magenta.

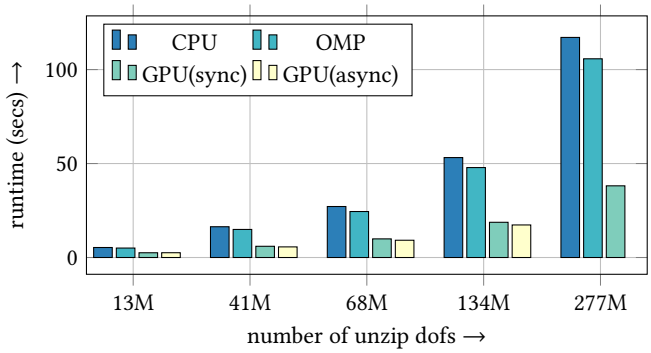

Figure 7: Performance on different autogenerated codes on a single node of Titan. We evaluated a normal distribution of blocks levels with increasing grain sizes. The GPU on Titan is the Nvidia K20.

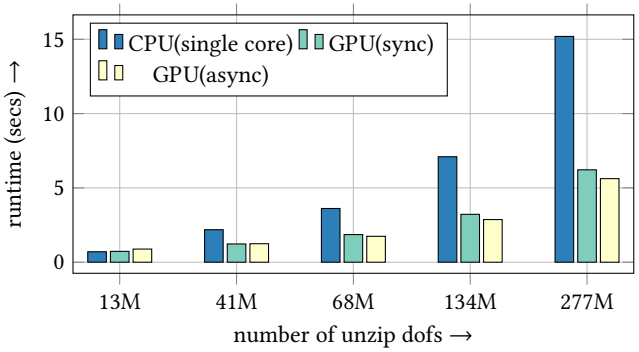

Figure 8: Performance on different autogenerated codes on a single node of A1. We evaluated a normal distribution of blocks levels with increasing grain sizes. The GPU on A1 is the Nvidia P100.

variables. A complete outline of our approach for simulating binary BH mergers demonstrating how the various components come together is listed in Algorithm 2 and illustrated in Figure 1.

4 RESULTS

Experimental Setup: The large scalability experiments reported in this paper were performed on Titan. Titan is a Cray XK7 supercomputer at Oak Ridge National Laboratory (ORNL) with a total of 18,688 nodes, each consisting of a single 16-core AMD Opteron 6200 series processor Nvidia Kepler graphics processing units (GPUs), with a total of CPU 299,008 cores. Each node has 32GB of memory. It has a Gemini interconnect and 600TB of memory across all nodes.

To evaluate single node performance we use the A1 & B1 clusters at our local university supercomputing facility. A1 has 385 total nodes (8292 cores), with the nodes having 16, 20, 24, 28, or 32 cores each, and memory between 32GB and 1TB each. A1 also includes 4 Nvidia Pascal 100 GPUs. B1 consists of 15 dual socket nodes with 32 cores each and has 3 Nvidia Volta 100 GPUs.

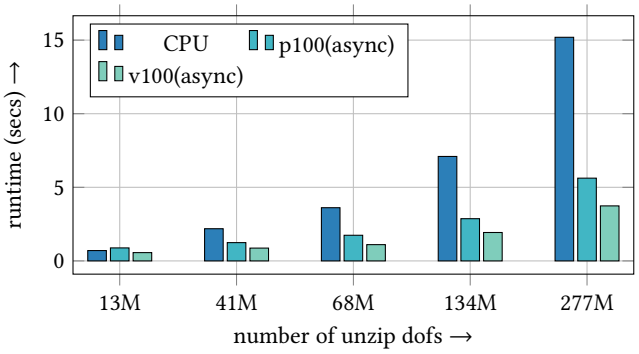

Figure 9: Performance on different autogenerated codes on a single node of B1. We evaluated a normal distribution of blocks levels with increasing grain sizes. The GPU on B1 is the Nvidia V100.

4.1 Correctness of code

HAD vs. Dendro-GR: HAD [37] is a computational framework for distributed block-adaptive mesh refinement that has been used

in numerical relativity to study binary mergers of black holes [55, 57–59], and neutron stars [15–18, 50, 56]. To ensure the correctness of the Sympy based code generation, we evaluate the BSSNOK equations using HAD and the Dendro-GR CPU and GPU codes using smooth, non-zero initial data for all functions ("fake initial data"). Differences in the evaluation of the equations on a uniform grid after a single computation of the BSSNOK equations between the codes are shown in Table 1. The evolution equations for $\partial_t A_{ij}$ are the most complicated equations, resulting in higher finite-precision errors for these variables. Extensive validation of the generated CPU code can be found in [28]

variable	$\|\|f_i\|\|_2$	$\|\|H - D_{\text{cpu}}\|\|_\infty$	$\|\|D_{\text{cpu}} - D_{\text{gpu}}\|\|_\infty$
$\partial_t \alpha$	1.0	0	3.6×10^{-15}
$\partial_t \chi$	12	1.7×10^{-13}	1.4×10^{-14}
$\partial_t K$	1.0	2.5×10^{-11}	1.0×10^{-7}
$\partial_t \tilde{\Gamma}^1$	0.16	4.0×10^{-13}	2.0×10^{-11}
$\partial_t \tilde{\Gamma}^2$	0.16	3.1×10^{-13}	1.8×10^{-10}
$\partial_t \tilde{\Gamma}^3$	0.16	8.0×10^{-13}	9.2×10^{-14}
$\partial_t \beta^1$	0.12	0	3.5×10^{-18}
$\partial_t \beta^2$	0.32	0	4.4×10^{-16}
$\partial_t \beta^3$	0.12	0	1.1×10^{-16}
$\partial_t B^1$	130	4.5×10^{-13}	2.0×10^{-11}
$\partial_t B^2$	30	3.4×10^{-13}	1.8×10^{-10}
$\partial_t B^3$	21	8.0×10^{-13}	9.2×10^{-14}
$\partial_t \tilde{\gamma}_{11}$	1.4	1.1×10^{-13}	2.9×10^{-15}
$\partial_t \tilde{\gamma}_{12}$	0.21	1.8×10^{-14}	2.2×10^{-15}
$\partial_t \tilde{\gamma}_{13}$	0.44	8.5×10^{-14}	4.4×10^{-16}
$\partial_t \tilde{\gamma}_{22}$	1.4	1.1×10^{-13}	3.0×10^{-15}
$\partial_t \tilde{\gamma}_{23}$	0.39	2.5×10^{-14}	4.4×10^{-16}
$\partial_t \tilde{\gamma}_{33}$	0.73	4.3×10^{-14}	1.7×10^{-15}
$\partial_t \tilde{A}_{11}$	14	1.8×10^{-11}	1.4×10^{-7}
$\partial_t \tilde{A}_{12}$	1.8	1.9×10^{-12}	1.7×10^{-7}
$\partial_t \tilde{A}_{13}$	19	7.7×10^{-12}	5.9×10^{-11}
$\partial_t \tilde{A}_{22}$	12	9.1×10^{-12}	1.7×10^{-7}
$\partial_t \tilde{A}_{23}$	2.8	3.0×10^{-12}	8.9×10^{-14}
$\partial_t \tilde{A}_{33}$	6.7	9.1×10^{-12}	7.0×10^{-8}

Table 1: Differences in evaluating the BSSNOK equations for test initial data using HAD (H), and the Dendro-GR CPU (D_{cpu}) and GPU (D_{gpu}) codes. The second column shows the $||.||_2$ norm of the test functions f_i, and the remaining columns show the $||.||_\infty$ norms of differences. Note, all numbers are given to two significant figures, and numbers smaller than machine epsilon are given as 0.

Equal mass binary black holes: The BSSNOK equations consist of 24 coupled PDEs evolved in time, while 4 constraint equations (i.e. momentum and Hamiltonian constraints) are used to monitor the quality of the solution. In Figure 10, we present how constraint norms change until the merger event for a simulation of an equal mass ratio binary done using Dendro-GR. The constraint norms serve as an additional test for the accuracy of the generated code as well as for the overall computational framework.

4.2 Single Node Performance

Our code separates parallelism into MPI-based distributed memory parallelism that is handled by Dendro and autogenerated node-local parallelism. In this section we evaluate and demonstrate the performance of our autogeneration framework on different node configurations including three generations of Nvidia GPUs (Kepler, Pascal and Volta). The first experiment was performed on Titan (Figure 7) where we tested the performance of sequential CPU, OpenMP and CUDA codes for increasing grain sizes. We are able to achieve significant speedup from the use of GPUs. Similar speedups are seen for the P100 (Figure 8) and V100 (Figure 9). In all cases, the codes were autogenerated simply by specifying the configuration for each GPU. This corresponds to information about the size of shared memory, number of SMs, etc. We also note the significant speedup on B1 using V100 compared to the performance on Titan. Overall, our code should continue to get good performance on future generations of GPUs.

4.3 Distributed Memory Scalability

While the autogeneration of architecture specific codes is important for portability, for large-scale runs we still rely on distributed memory parallelism. The distributed memory scalability of Dendro is excellent and in this section we present results on Titan demonstrating both strong and weak scalability. In Figure 11, we present weak scalability from 2 to 8,192 nodes on Titan. We operated with a grain size of 22.5M per process for a largest problem size of 206 billion unknowns. This experiment corresponds to a binary black hole system with a mass ratio $q = 10$, a maxdepth 18 and a wavelet tolerance of 10^{-6}. Note that the unknowns per core show a slight variation because with WAMR we do not have explicit control over the grid size and WAMR decides the refinement region on the mesh based on the how wavelets behave during the time evolution. This is why we have used a normalized RK with dof/p metrics to report accurate weak scaling results. As can be seen, the weak scalability is very stable over a wide range of nodes.

Similarly, in Figure 12 we present strong scalability, again for a binary black hole system with a mass ratio $q = 10$, a maxdepth 18 and a wavelet tolerance of 10^{-6} for a problem size of 10.5 billion unknowns from 256 to 4096 nodes on Titan. Again, we are able

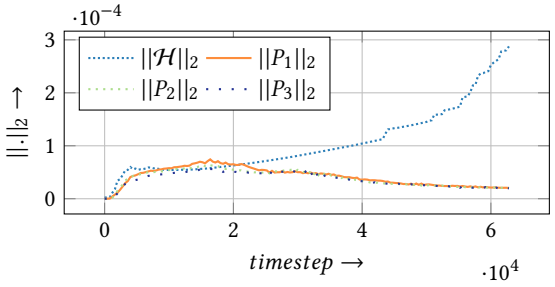

Figure 10: Hamiltonian and Momentum constraint equation norm variation throughout the equal mass ratio binary compact merger simulation performed using Dendro-GR. Note that the highest constraint violations occurs at the event of the merger.

to maintain excellent strong scalability over a large range of distributed nodes.

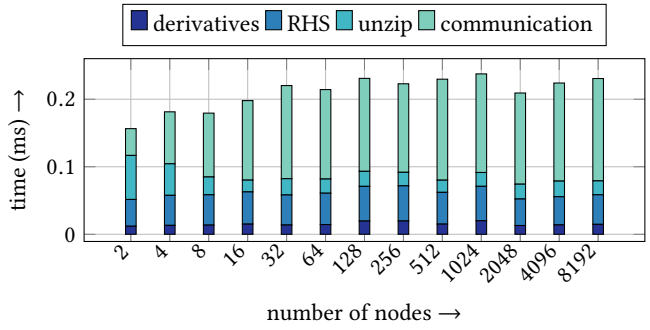

Figure 11: Weak scaling results in ORNL's Titan for $RK/(dof/p)$ (averaged over 10 steps) where RK, dof, and p denote the time for a single RK step, the number of degrees of freedom, and the number of cores respectively. We show the derivative computation (deriv), the right hand side computation (rhs), the unzip cost, the wavelet computation (wavelets) and the communication cost (comm) as an average over 1.41M unknowns per core in which the number of cores ranges from 32 to 131,072 cores on 8,192 nodes and for which the largest problem has 206 billion unknowns. The above results are generated with a mass ratio $q = 10$, a MAXDEPTH 18 and a wavelet tolerance of 10^{-6}. Note that the unknowns per core show a slight variation because with WAMR we do not have explicit control over the grid size and WAMR decides the refinement region on the mesh based on the how wavelets behave during the time evolution. This is why we have used a normalized RK with dof/p metrics to report accurate weak scaling results.

4.4 Einstein Toolkit vs. Dendro-GR

The Einstein Toolkit (ET) [32] is a well-known computational framework to advance and support research in relativistic astrophysics and gravitational physics. In this section, we present performance comparison results for ET and Dendro-GR to perform

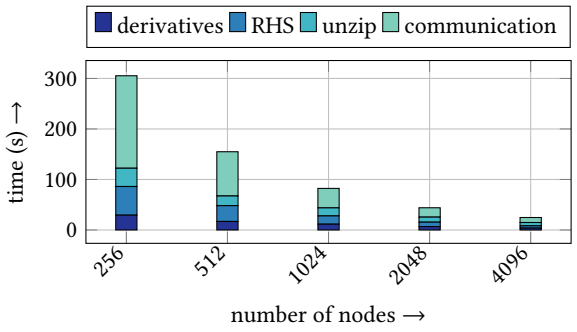

Figure 12: Strong scaling results in ORNL's Titan for a single RK step (averaged over 10 steps) with the derivative computation (deriv), the right hand side (rhs) computation, the unzip cost and the communication cost (comm) for a fixed problem size of $10.5B$ unknowns where the number of cores ranges from 4,096 to 65,536 cores on 4096 nodes. Note that for strong scaling results re-meshing is disabled in order to keep the problem size fixed.

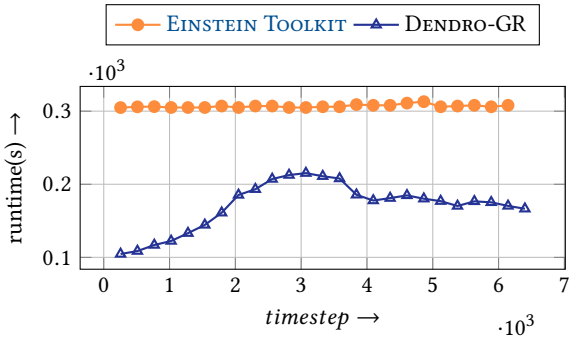

Figure 13: Comparison between Dendro-GR and ET to perform 256 time steps using $RK4$ time integration as the simulation progress in A1 cluster using 112 cores. Note that the number of grid points is different due to the underlying adaptivity framework utilized between the two frameworks. ET uses block adaptivity with z-axis symmetry, while Dendro-GR uses WAMR where adaptivity is determined based on the wavelets, at the runtime and no symmetry is assumed.

RK time evolutions of the BSSNOK equations. We have used the roughly equal mass ratio binary black hole configuration provided in [77] and used the auto-generated CPU code in Dendro-GR for the performance comparison. This particular system has mirror symmetry about the orbital plane, and ET exploits this symmetry by evolving only the upper half of the domain, reducing the computational cost by a factor of two. Systems with more generic initial data, such as black holes with arbitrary spins or neutron stars with magnetic fields, do not possess mirror symmetry, and this simplification cannot be used. Dendro-GR is designed for these more general spacetimes, and the current version does not support imposing mirror symmetry on the solutions.

The comparison results presented in Figure 13 are performed using 4 nodes (112 cores) on the A1 cluster, for $7K$ time steps using $RK4$ time integration. Note that, due to the minimal grid changes in ET grid, ET shows a stable runtime, while the runtime in Dendro-GR fluctuates with the grid point variation due to the use of WAMR. For large mass ratio configurations, block adaptive approaches leads to inefficient refinement patterns compared to WAMR, hence the performance gap between Dendro-GR and ET can be expected to increase for large mass ratio runs.

5 CONCLUSIONS

In the short time that LIGO and Virgo have been searching for gravitational waves, we have already learned exciting things about neutron stars [53], the production of heavy elements (such as gold) and the population of black holes in the universe. When gravitational wave observations are combined with observations of electromagnetic radiation—from radio waves to gamma rays—there is a multiplicative effect that magnifies the scientific impact. This is the promise of multi-messenger astronomy.

The full scientific impact of multi-messenger astronomy is only realized when the observations are informed by sophisticated computer models of the underlying astrophysical phenomena. Dendro provides the ability to run these models in a scalable way, with

local adaptivity criteria using WAMR. While AMR codes with block-adaptivity typically lose performance as the number of adaptive levels increases, Dendro achieves impressive scalability on a real application even with many levels of refinement. The combination of scalability and adaptivity will allow us to study the gravitational radiation from IMRIs without simplifying approximations in direct numerical simulations.

The Dendro code reported on here, with a module for vacuum black hole spacetimes, is just the first step in creating a highly adaptive computational platform for studying relativistic astrophysics on the next-generation of supercomputers. This work will be followed with additional modules for solving the relativistic magneto-hydrodynamics equations, nuclear equations of state, and radiation hydrodynamics. For application developers, a key advantage of Dendro is the ability to use conventional numerical methods for these modules.

As LIGO and Virgo are joined by other gravitational wave detectors and observatories around the world, we expect many exciting discoveries to come.

ACKNOWLEDGMENTS

We thank the reviewers whose feedback greatly improved this paper. This work was supported in part by the National Science Foundation grants NSF-PHY1607356, CCF-1704715 and OAC-1808652. This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725 and the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant TG-PHY180002 and TG-PHY180054.

REFERENCES

[1] 2018. LIGO home page. (2018). http://flash.uchicago.edu/website/home/.
[2] 2018. Virgo home page. (2018). http://http://www.virgo-gw.eu.
[3] B.âĂŁP. Abbott et al. 2017. GW170817: Observation of Gravitational Waves from a Binary Neutron Star Inspiral. *Phys. Rev. Lett.* 119, 16 (2017), 161101. https://doi.org/10.1103/PhysRevLett.119.161101 arXiv:gr-qc/1710.05832
[4] B. P. Abbott et al. 2016. GW151226: Observation of Gravitational Waves from a 22-Solar-Mass Binary Black Hole Coalescence. *Phys. Rev. Lett.* 116 (Jun 2016), 241103. Issue 24. https://doi.org/10.1103/PhysRevLett.116.241103
[5] B. P. Abbott et al. 2016. Observation of Gravitational Waves from a Binary Black Hole Merger. *Phys. Rev. Lett.* 116, 6 (2016), 061102. https://doi.org/10.1103/PhysRevLett.116.061102 arXiv:gr-qc/1602.03837
[6] B. P. Abbott et al. 2016. Observation of Gravitational Waves from a Binary Black Hole Merger. *Phys. Rev. Lett.* 116 (Feb 2016), 061102. Issue 6. https://doi.org/10.1103/PhysRevLett.116.061102
[7] B. P. Abbott et al. 2017. Estimating the Contribution of Dynamical Ejecta in the Kilonova Associated with GW170817. *Astrophys. J.* 850, 2 (2017), L39. https://doi.org/10.3847/2041-8213/aa9478 arXiv:astro-ph.HE/1710.05836
[8] B. P. Abbott et al. 2017. Gravitational Waves and Gamma-rays from a Binary Neutron Star Merger: GW170817 and GRB 170817A. *Astrophys. J.* 848, 2 (2017), L13. https://doi.org/10.3847/2041-8213/aa920c arXiv:astro-ph.HE/1710.05834
[9] Benjamin P. Abbott et al. 2017. GW170104: Observation of a 50-Solar-Mass Binary Black Hole Coalescence at Redshift 0.2. *Phys. Rev. Lett.* 118, 22 (2017), 221101. https://doi.org/10.1103/PhysRevLett.118.221101 arXiv:gr-qc/1706.01812
[10] B. P. Abbott et al. 2017. GW170814: A Three-Detector Observation of Gravitational Waves from a Binary Black Hole Coalescence. *Phys. Rev. Lett.* 119, 14 (2017), 141101. https://doi.org/10.1103/PhysRevLett.119.141101 arXiv:gr-qc/1709.09660
[11] B. P. Abbott et al. 2017. Multi-messenger Observations of a Binary Neutron Star Merger. *Astrophys. J.* 848, 2 (2017), L12. https://doi.org/10.3847/2041-8213/aa91c9 arXiv:astro-ph.HE/1710.05833
[12] F. Acernese et al. 2015. Advanced Virgo: a second-generation interferometric gravitational wave detector. *Class. Quant. Grav.* 32, 2 (2015), 024001. https://doi.org/10.1088/0264-9381/32/2/024001 arXiv:gr-qc/1408.3978
[13] A. Albert et al. 2017. Search for High-energy Neutrinos from Binary Neutron Star Merger GW170817 with ANTARES, IceCube, and the Pierre Auger Observatory. *Astrophys. J.* 850, 2 (2017), L35. https://doi.org/10.3847/2041-8213/aa9aed arXiv:astro-ph.HE/1710.05839
[14] Miguel Alcubierre. 2008. *Introduction to 3+1 numerical relativity*. Oxford Univ. Press, Oxford.
[15] Matthew Anderson et al. 2008. Magnetized Neutron Star Mergers and Gravitational Wave Signals. *Phys. Rev. Lett.* 100 (2008), 191101. https://doi.org/10.1103/PhysRevLett.100.191101 arXiv:gr-qc/0801.4387
[16] Matthew Anderson et al. 2008. Simulating binary neutron stars: dynamics and gravitational waves. *Phys. Rev.* D77 (2008), 024006. https://doi.org/10.1103/PhysRevD.77.024006 arXiv:gr-qc/0708.2720
[17] Matthew Anderson, Eric Hirschmann, Steven L. Liebling, and David Neilsen. 2006. Relativistic MHD with Adaptive Mesh Refinement. *Class. Quant. Grav.* 23 (2006), 6503–6524. arXiv:gr-qc/0605102
[18] Matthew Anderson, Luis Lehner, Miguel Megevand, and David Neilsen. 2010. Post-merger electromagnetic emissions from disks perturbed by binary black holes. *Phys.Rev.* D81 (2010), 044004. https://doi.org/10.1103/PhysRevD.81.044004 arXiv:astro-ph.HE/0910.4969
[19] Stanislav Babak, Andrea Taracchini, and Alessandra Buonanno. 2016. Validating the effective-one-body model of spinning, precessing binary black holes against numerical relativity. (2016). arXiv:gr-qc/1607.05661
[20] Thomas W. Baumgarte and Stuart L. Shapiro. 1999. On the numerical integration of Einstein's field equations. *Phys.Rev.* D59 (1999), 024007. https://doi.org/10.1103/PhysRevD.59.024007 arXiv:gr-qc/gr-qc/9810065
[21] Bernd Bruegmann, Jose A. Gonzalez, Mark Hannam, Sascha Husa, Ulrich Sperhake, and Wolfgang Tichy. 2008. Calibration of Moving Puncture Simulations. *Phys. Rev.* D77 (2008), 024027. https://doi.org/10.1103/PhysRevD.77.024027 arXiv:gr-qc/gr-qc/0610128
[22] Cactus Computational Toolkit. [n. d.]. ([n. d.]). http://www.cactuscode.org.
[23] Manuela Campanelli, C. O. Lousto, P. Marronetti, and Y. Zlochower. 2006. Accurate evolutions of orbiting black-hole binaries without excision. *Phys. Rev. Lett.* 96 (2006), 111101. arXiv:gr-qc/0511048
[24] Matthias Christen, Olaf Schenk, and Helmar Burkhart. 2011. Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 676–687.
[25] John Cocke. 1970. Global Common Subexpression Elimination. In *Proceedings of a Symposium on Compiler Optimization*. ACM, New York, NY, USA, 20–24. https://doi.org/10.1145/800028.808480
[26] R. Courant, K. Friedrichs, and H. Lewy. 1967. On the Partial Difference Equations of Mathematical Physics. *IBM Journal of Research and Development* 11, 2 (March 1967), 215–234. https://doi.org/10.1147/rd.112.0215
[27] Nom de Plume. 2018. Manuscript Title. *Ars Anon* (2018).
[28] Nom de Plume. 2018. Manuscript Title. *Ars Anon* (2018).
[29] Jackson DeBuhr, Bo Zhang, Matthew Anderson, David Neilsen, and Eric W. Hirschmann. 2015. Relativistic Hydrodynamics with Wavelets. (2015). arXiv:astro-ph.IM/1512.00386
[30] D. L Donoho. 1992. Interpolating wavelet transforms. *Preprint, Department of Statistics, Stanford University* 2 (1992).
[31] Michael Dumbser, Federico Guercilena, Sven Koeppel, Luciano Rezzolla, and Olindo Zanotti. 2017. A strongly hyperbolic first-order CCZ4 formulation of the Einstein equations and its solution with discontinuous Galerkin schemes. (2017). arXiv:gr-qc/1707.09910
[32] Einstein Toolkit. [n. d.]. ([n. d.]). http://einsteintoolkit.org.
[33] Zachariah B. Etienne, Vasileios Paschalidis, Roland Haas, Philipp MÃűsta, and Stuart L. Shapiro. 2015. IllinoisGRMHD: An Open-Source, User-Friendly GRMHD Code for Dynamical Spacetimes. *Class. Quant. Grav.* 32 (2015), 175009. https://doi.org/10.1088/0264-9381/32/17/175009 arXiv:astro-ph.HE/1501.07276
[34] A. Goldstein et al. 2017. An Ordinary Short Gamma-Ray Burst with Extraordinary Implications: Fermi-GBM Detection of GRB 170817A. *Astrophys. J.* 848, 2 (2017), L14. https://doi.org/10.3847/2041-8213/aa8f41 arXiv:astro-ph.HE/1710.05446
[35] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf. 2003. The Cactus Framework and Toolkit: Design and Applications. In *Vector and Parallel Processing - VECPAR '2002, 5th International Conference*. Springer. http://www.springerlink.com/content/2fapcbeyyc1xg0mm/
[36] Philip B. Graff, Alessandra Buonanno, and B. S. Sathyaprakash. 2015. Missing Link: Bayesian detection and measurement of intermediate-mass black-hole binaries. *Phys. Rev.* D92, 2 (2015), 022002. https://doi.org/10.1103/PhysRevD.92.022002 arXiv:gr-qc/1504.04766
[37] had home page. 2009. (2009). http://had.liu.edu.
[38] Tom Henretty, Kevin Stock, Louis-Noël Pouchet, Franz Franchetti, J Ramanujam, and P Sadayappan. 2011. Data layout transformation for stencil computations on short-vector simd architectures. In *International Conference on Compiler Construction*. Springer, 225–245.

[39] Tom Henretty, Richard Veras, Franz Franchetti, Louis-Noël Pouchet, Jagannathan Ramanujam, and Ponnuswamy Sadayappan. 2013. A stencil compiler for short-vector SIMD architectures. In *Proceedings of the 27th international ACM conference on International conference on supercomputing*. ACM, 13–24.

[40] Justin Holewinski, Louis-Noel Pouchet, and Ponnuswamy Sadayappan. 2012. High-performance code generation for stencil computations on GPU architectures. In *Proceedings of the 26th ACM international conference on Supercomputing*. ACM, 311–320.

[41] M. Holmström. 1999. Solving hyperbolic PDEs using interpolating wavelets. *SIAM J. Sci. Comput.* 21, 2 (1999), 405–420.

[42] Sascha Husa, Ian Hinder, and Christiane Lechner. 2006. Kranc: A Mathematica application to generate numerical codes for tensorial evolution equations. *Comput. Phys. Commun.* 174 (2006), 983–1004. https://doi.org/10.1016/j.cpc.2006.02.002 arXiv:gr-qc/gr-qc/0404023

[43] David Joyner, Ondřej Čertík, Aaron Meurer, and Brian E Granger. 2012. Open source computer algebra systems: SymPy. *ACM Communications in Computer Algebra* 45, 3/4 (2012), 225–234.

[44] David Keitel et al. 2017. The most powerful astrophysical events: Gravitational-wave peak luminosity of binary black holes as predicted by numerical relativity. *Phys. Rev.* D96, 2 (2017), 024006. https://doi.org/10.1103/PhysRevD.96.024006 arXiv:gr-qc/1612.09566

[45] Lawrence E. Kidder et al. 2017. SpECTRE: A Task-based Discontinuous Galerkin Code for Relativistic Astrophysics. *J. Comput. Phys.* 335 (2017), 84–114. https://doi.org/10.1016/j.jcp.2016.12.059 arXiv:astro-ph.HE/1609.00098

[46] Kenta Kiuchi, Koutarou Kyutoku, Yuichiro Sekiguchi, Masaru Shibata, and Tomohide Wada. 2014. High resolution numerical-relativity simulations for the merger of binary magnetized neutron stars. *Phys.Rev.* D90, 4 (2014), 041502. https://doi.org/10.1103/PhysRevD.90.041502 arXiv:astro-ph.HE/1407.2660

[47] Kenta Kiuchi, Koutarou Kyutoku, and Masaru Shibata. 2012. Three dimensional evolution of differentially rotating magnetized neutron stars. *Phys.Rev.* D86 (2012), 064008. https://doi.org/10.1103/PhysRevD.86.064008 arXiv:astro-ph.HE/1207.6444

[48] H-O Kreiss and J Oliger. 1973. *Methods for Approximate Solution of Time Dependent Problems.* GARP Publication Series, Geneva.

[49] Luis Lehner and Frans Pretorius. 2014. Numerical Relativity and Astrophysics. *Ann.Rev.Astron.Astrophys.* 52 (2014), 661–694. https://doi.org/10.1146/annurev-astro-081913-040031 arXiv:astro-ph.HE/1405.4840

[50] Steven L. Liebling, Luis Lehner, David Neilsen, and Carlos Palenzuela. 2010. Evolutions of Magnetized and Rotating Neutron Stars. *Phys.Rev.* D81 (2010), 124023. https://doi.org/10.1103/PhysRevD.81.124023 arXiv:gr-qc/1001.0575

[51] Vassilios Mewes, Yosef Zlochower, Manuela Campanelli, Ian Ruchlin, Zachariah B. Etienne, and Thomas W. Baumgarte. 2018. Numerical Relativity in Spherical Coordinates with the Einstein Toolkit. (2018). arXiv:gr-qc/1802.09625

[52] Paulius Micikevicius. 2009. 3D finite difference computation on GPUs using CUDA. In *Proceedings of 2nd workshop on general purpose processing on graphics processing units*. ACM, 79–84.

[53] Elias R. Most, Lukas R. Weih, Luciano Rezzolla, and JÃijrgen Schaffner-Bielich. 2018. New constraints on radii and tidal deformabilities of neutron stars from GW170817. (2018). arXiv:gr-qc/1803.00549

[54] T. Nakamura, K. Oohara, and Y. Kojima. 1987. General Relativistic Collapse to Black Holes and Gravitational Waves from Black Holes. *Progress of Theoretical Physics Supplement* 90 (1987), 1–218. https://doi.org/10.1143/PTPS.90.1

[55] David Neilsen, Luis Lehner, Carlos Palenzuela, Eric W. Hirschmann, Steven L. Liebling, et al. 2011. Boosting jet power in black hole spacetimes. *Proc.Nat.Acad.Sci.* 108 (2011), 12641–12646. https://doi.org/10.1073/pnas.1019618108 arXiv:astro-ph.HE/1012.5661

[56] David Neilsen, Steven L. Liebling, Matthew Anderson, Luis Lehner, Evan O'Connor, et al. 2014. Magnetized Neutron Stars With Realistic Equations of State and Neutrino Cooling. *Phys.Rev.* D89, 10 (2014), 104029. https://doi.org/10.1103/PhysRevD.89.104029 arXiv:gr-qc/1403.3680

[57] Carlos Palenzuela, Matthew Anderson, Luis Lehner, Steven L. Liebling, and David Neilsen. 2009. Stirring, not shaking: binary black holes' effects on electromagnetic fields. *Phys. Rev. Lett.* 103 (2009), 081101. https://doi.org/10.1103/PhysRevLett.103.081101 arXiv:astro-ph.HE/0905.1121

[58] Carlos Palenzuela, Travis Garrett, Luis Lehner, and Steven L. Liebling. 2010. Magnetospheres of Black Hole Systems in Force-Free Plasma. *Phys.Rev.* D82 (2010), 044045. https://doi.org/10.1103/PhysRevD.82.044045 arXiv:gr-qc/1007.1198

[59] Carlos Palenzuela, Luis Lehner, and Steven L. Liebling. 2010. Dual Jets from Binary Black Holes. *Science* 329 (2010), 927. https://doi.org/10.1126/science.1191766 arXiv:astro-ph.HE/1005.1067

[60] S. Paolucci, Z. J. Zikoski, and T. Grenga. 2014. WAMR: An adaptive wavelet method for the simulation of compressible reacting flow. Part II. The parallel algorithm. *J. Comput. Phys.* 272 (2014), 842 – 864.

[61] S. Paolucci, Z. J. Zikoski, and D. Wirasaet. 2014. WAMR: An adaptive wavelet method for the simulation of compressible reacting flow. Part I. Accuracy and efficiency of algorithm. *J. Comput. Phys.* 272 (2014), 814 – 841.

[62] L. Rezzolla and O. Zanotti. 2013. *Relativistic Hydrodynamics.* Oxford Univ. Press, Oxford.

[63] Luke F. Roberts, Christian D. Ott, Roland Haas, Evan P. O'Connor, Peter Diener, and Erik Schnetter. 2016. General Relativistic Three-Dimensional Multi-Group Neutrino Radiation-Hydrodynamics Simulations of Core-Collapse Supernovae. (2016). https://doi.org/10.3847/0004-637X/831/1/98 arXiv:astro-ph.HE/1604.07848

[64] Ian Ruchlin, Zachariah B. Etienne, and Thomas W. Baumgarte. 2017. SENR/NRPy+: Numerical Relativity in Singular Curvilinear Coordinate Systems. (2017). arXiv:gr-qc/1712.07658

[65] Olivier Sarbach, Gioel Calabrese, Jorge Pullin, and Manuel Tiglio. 2002. Hyperbolicity of the Baumgarte-Shapiro-Shibata-Nakamura system of Einstein evolution equations. *Phys. Rev. D* 66 (Sep 2002), 064002. Issue 6. https://doi.org/10.1103/PhysRevD.66.064002

[66] B. S. Sathyaprakash and S. V. Dhurandhar. 1991. Choice of filters for the detection of gravitational waves from coalescing binaries. *Phys. Rev.* D44 (1991), 3819–3834. https://doi.org/10.1103/PhysRevD.44.3819

[67] V. Savchenko et al. 2017. INTEGRAL Detection of the First Prompt Gamma-Ray Signal Coincident with the Gravitational-wave Event GW170817. *Astrophys. J.* 848, 2 (2017), L15. https://doi.org/10.3847/2041-8213/aa8f94 arXiv:astro-ph.HE/1710.05449

[68] Masaru Shibata. 2015. *Numerical Relativity.* World Scientific Publishing Co., Inc., River Edge, NJ, USA.

[69] M. Shibata and T. Nakamura. 1995. Evolution of three-dimensional gravitational waves: Harmonic slicing case. *Phys. Rev. D* 52 (Nov. 1995), 5428–5444. https://doi.org/10.1103/PhysRevD.52.5428

[70] David Shoemaker et al. 2011. Advanced LIGO reference design. LIGO-M060056-v2. (2011). https://dcc.ligo.org/LIGO-M060056-v2/public.

[71] R. J. E. Smith, I. Mandel, and A. Vecchio. 2013. Studies of waveform requirements for intermediate mass-ratio coalescence searches with advanced gravitational-wave detectors. *Phys. Rev. D* 88 (Aug 2013), 044010. Issue 4. https://doi.org/10.1103/PhysRevD.88.044010

[72] Ulrich Sperhake. 2015. The numerical relativity breakthrough for binary black holes. *Class. Quant. Grav.* 32, 12 (2015), 124011. https://doi.org/10.1088/0264-9381/32/12/124011 arXiv:gr-qc/1411.3997

[73] Bela Szilagyi, Lee Lindblom, and Mark A. Scheel. 2009. Simulations of Binary Black Hole Mergers Using Spectral Methods. *Phys. Rev.* D80 (2009), 124010. https://doi.org/10.1103/PhysRevD.80.124010 arXiv:gr-qc/0909.3557

[74] Marcus Thierfelder, Sebastiano Bernuzzi, and Bernd Bruegmann. 2011. Numerical relativity simulations of binary neutron stars. *Phys.Rev.* D84 (2011), 044012. https://doi.org/10.1103/PhysRevD.84.044012 arXiv:gr-qc/1104.4751

[75] O. V. Vasilyev and S. Paolucci. 1996. A dynamically adaptive multilevel wavelet collocation method for solving partial differential equations in a finite domain. *J. Comput. Phys.* 125 (1996), 498–512.

[76] O. V. Vasilyev, S. Paolucci, and M. Sen. 1995. A multilevel wavelet collocation method for solving partial differential equations in a finite domain. *J. Comput. Phys.* 120 (1995), 33 – 47.

[77] Barry Wardell, Ian Hinder, and Eloisa Bentivegna. 2016. Simulation of GW150914 binary black hole merger using the Einstein Toolkit. https://doi.org/10.5281/zenodo.155394

[78] Y. Zlochower, J. G. Baker, Manuela Campanelli, and C. O. Lousto. 2005. Accurate black hole evolutions by fourth-order numerical relativity. *Phys. Rev.* D72 (2005), 024021. https://doi.org/10.1103/PhysRevD.72.024021 arXiv:gr-qc/gr-qc/0505055