A GPU-Accelerated AMR Solver for Gravitational Wave Propagation

Milinda Fernando
The University of Texas at Austin
Austin, TX, USA.
milinda@oden.utexas.edu

David Neilsen
Brigham Young University
Provo, UT, USA.
david.neilsen@byu.edu

Eric Hirschmann
Brigham Young University
Provo, UT, USA.
ehirsch@physics.byu.edu

Yosef Zlochower
Rochester Institute of Technology
Rochester, NY, USA.
yosef@astro.rit.edu

Hari Sundar
The University of Utah
Salt Lake City, UT, USA.
hari@cs.utah.edu

Omar Ghattas
The University of Texas at Austin
Austin, TX, USA.
omar@oden.utexas.edu

George Biros
The University of Texas at Austin
Austin, TX, USA.
biros@oden.utexas.edu

SC22: International Conference for High Performance Computing, Networking, Storage and Analysis | 978-1-6654-5444-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/SC41404.2022.00080

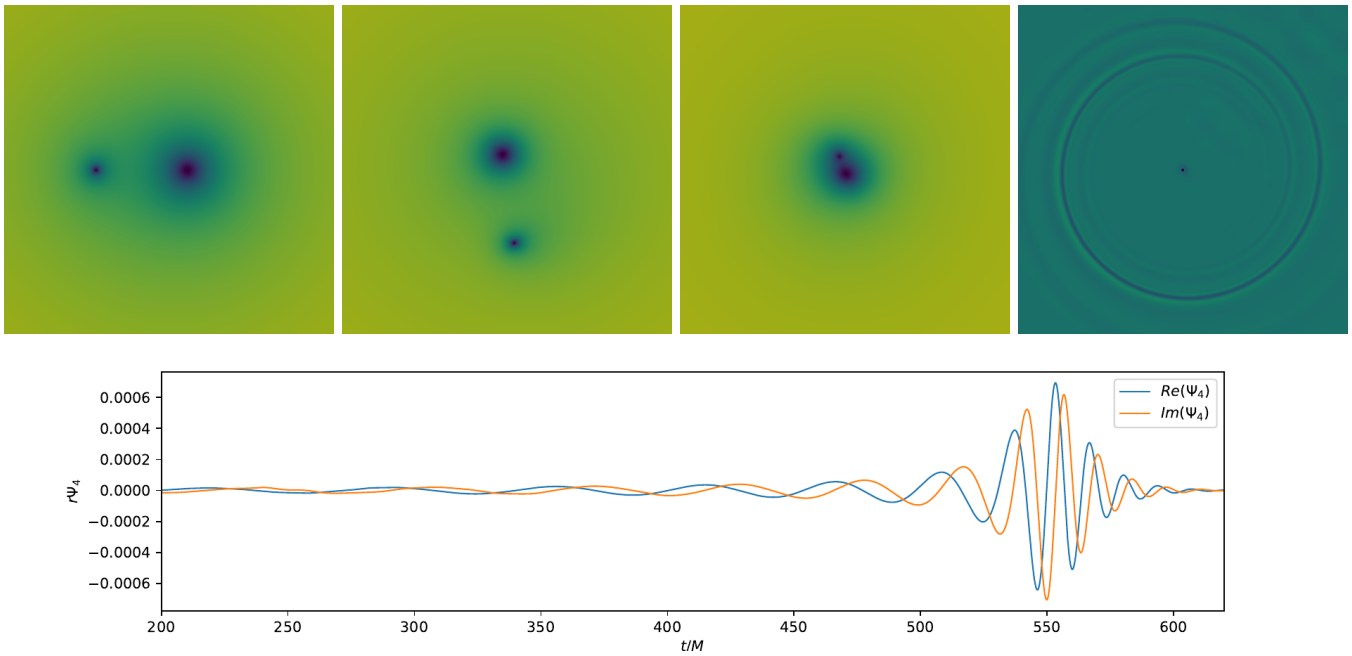

Fig. 1: The simulation snapshots of a binary black hole merger of mass ratio 1:4 ($q = 4$) and extracted gravitational waves using 4 NVIDIA A100 GPUs with a wall clock time of 5.3 days with 13 levels of refinement (i.e., coarsest level=3, finest level=15, with finest resolution of 4.06e-3), and without assuming any spacetime symmetries.

***Abstract*—Simulations to calculate a single gravitational waveform (GW) can take several weeks. Yet, thousands of such simulations are needed for the detection and interpretation of gravitational waves. Future detectors will require even more accurate waveforms than those currently used. We present here the first large scale, adaptive mesh, multi-GPU numerical relativity (NR) code together with performance analysis and benchmarking. While comparisons are difficult to make, our GPU extension of the DENDRO-GR NR code achieves a 6x speedup over existing state-of-the-art codes. We achieve 800 GFlops/s on a single NVIDIA A100 GPU with an overall 2.5x speedup over a two-socket, 128-core AMD EPYC 7763 CPU node with an equivalent CPU implementation. We present detailed performance analyses, parallel scalability results, and accuracy assessments for GWs computed for mass ratios q=1,2,4. We also present strong scalability up to 8 A100s and weak scaling up to 229,376 x86 cores on the Texas Advanced Computing Center's Frontera system.**

***Index Terms*—High performance computing, Gravitational waves, Astrophysics, Numerical simulation, Scientific computing**

I. INTRODUCTION

Gravitational waves are generated in the merger of astrophysical compact objects, such as black holes and neutron stars. These waves carry information about the merging system in the complicated pattern of the changing amplitude and frequency of the wave. The first detection of gravitational waves was made by the Laser Interferometer Gravitational-Wave Observatory (LIGO) [1] in 2015 [2]. Since that time,

SC22, November 13-18, 2022, Dallas, Texas, USA
978-1-6654-5444-5/22/$31.00 ©2022 IEEE

over 80 additional detections have been made by the LIGO-Virgo collaboration [3], [4].

The interaction of gravitational waves and matter is very weak. For example, gravitational waves from compact object mergers deflect the mirrors of LIGO and Virgo by distances on the order of 10^{-19} m. These detectors produce an enormous amount of data (approximately 800 TB/year) and require advanced data analysis techniques to extract information from GW events. Information about the progenitor binary systems is extracted using templated waveforms that are parameterized with 15 parameters. One key parameter is the ratio of the masses of the two objects in the binary, $q = m_1/m_2$, where m_1 is the mass of the primary body, $m_1 \geq m_2$. Solving the full set of Einstein's equations to generate model waveforms using the techniques of numerical relativity (NR) requires expensive, large-scale supercomputer simulations. Thus, the library of candidate waveforms for the large parameter space is primarily created using a combination of semi-analytical and phenomenological approximate models [5]–[9], which can be rapidly computed. However, these models must be tuned and verified through comparison with full numerical relativity solutions over different regions of the parameter space.

To enable such comparisons, among other types of gravitational waveform analysis, catalogs of NR waveforms for binary black hole (BBH) mergers are being constructed, such as the SXS [10], RIT [11], GaTech [12], and CoRe [13] catalogs. Waveforms for BBHs with large mass ratios, $q \gg 10$, cannot be reliably computed from semi-analytical methods. The post-Newtonian approximation, for example, diverges in this limit. These systems are particularly difficult to calculate using NR simulations. This is due to the extreme resolution requirements needed to resolve the smaller black hole, the consequent reduction in the size of the maximum allowable timestep, and the resulting greater number of timesteps required to reach the merger event. Figure 3 shows the computational grids for a $q = 8$ binary, and Table I shows the estimated resolution requirements and the number of timesteps for binary black hole systems with different mass ratios. The largest mass ratios in the SXS and GaTech catalogs are, for example, $q = 10$ and $q = 15$, respectively; the RIT catalog has a few waveforms up to $q = 128$. However, it is essential that waveforms for large q binaries are available for a large region of the binary parameter space, so that detected waveforms from these systems can be properly analyzed and understood.

A further challenge for numerical relativity will come as new detectors (such as LISA [15], the Einstein Telescope [16], and Cosmic Explorer [14]) come online and existing detectors are improved. As the signal-to-noise ratio (SNR) increases, higher fidelity waveforms will be required to extract the full scientific impact of the detected waveforms. For example, Figure 2 shows simulated gravitational waveforms that could be detected by the future A+ configuration for LIGO [17], [18] and the proposed Cosmic Explorer detector. Gravitational wave signals will be detected with both higher SNR and for longer times. Generating comparable quality gravitational waveforms using NR is currently beyond the capabilities of most NR codes [19]–[21].

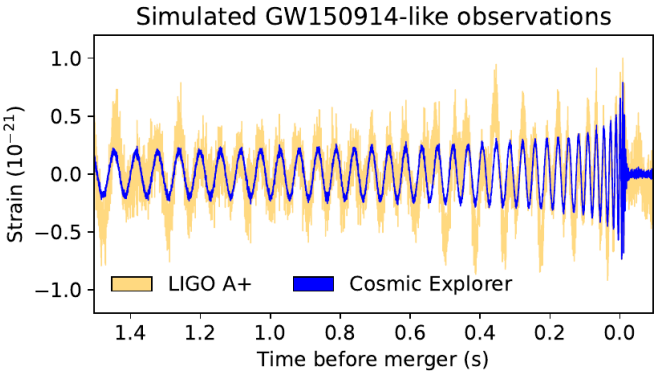

Fig. 2: Simulated gravitational-wave detector strain measurements for two merging black holes with simulated noise for LIGO A+ noise (yellow) and Cosmic Explorer (blue). The source is a GW150914-like black hole binary. Compare Ref [2] for the measured signal by Advanced LIGO. The figure is from [14].

mass-ratio $q = m_1/m_2$	Δx_{min} (BH1)	Δx_{min} (BH2)	time (M)	timesteps
1	8.33e-03	8.33e-03	650	7.8e4
4	3.33e-03	1.33e-02	700	2.1e5
16	9.80e-04	1.57e-02	1 400	1.4e6
64	2.56e-04	1.64e-02	6 000	2.3e7
256	6.46e-05	1.65e-02	24 000	3.7e8
512	3.23e-05	1.65e-02	48 000	1.5e9

TABLE I: Approximate resolution requirements needed to resolve the black holes in binaries with increasing mass ratio, q, with fixed total mass $M = 1$. (Note that all quantities are given in geometric units.) We assume ∼120 grid points across event horizons and an initial separation of $d = 8$. Merger times for $q \leq 16$ are from simulations of the full Einstein equations. Other times are approximated using evolutions of the post-Newtonian 2.5 equations.

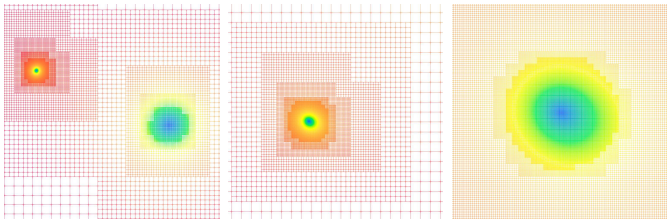

Fig. 3: Two-dimensional slices of the grid generated for a binary merger of mass ratio $q = 8$. From left to right, the grid is zoomed in on in order to show the refinement levels surrounding the smaller black hole.

To overcome these challenges, we present an octree-based adaptive multi-resolution scheme that enables efficient use of GPUs for numerical relativity. Using the so-called BSSN 3D+t formulation of the Einstein equations (section III-A) and an octree-based sixth-order in space and explicit fourth-order in time discretization (section III-B), we study the efficient

evaluation of the right hand side (RHS) of these PDEs. As we will see, this involves 24 state variables and 210 derivatives, thus generating tremendous memory pressure. This, in addition to the need to use adaptive mesh refinement (AMR)—which has long been considered a challenge on parallel architectures due to the complex dynamic data structures [22]—makes even small performance improvements extremely formidable. Our implementation extends the open-source DENDRO-GR library [23]–[25], one of the state-of-the-art AMR NR codes (section II). Our contributions are summarized below.

- To the best of our knowledge, this work is the first highly-scalable, adaptive, multi-GPU numerical relativity code performing binary black hole mergers;
- Algorithms for efficient use of GPUs for numerical simulations on adaptive octree grids (section IV);
- A detailed roofline analysis for complicated Einstein equation evaluations;
- Novel code generation approach to reduce the register pressure with additional GPU and CPU optimizations (sections IV-A and IV-B);
- A 2.5× overall speedup using a single NVIDIA A100 GPU compared to a single CPU node consisting of two AMD EPYC 7763 64-core processors (section V-A);
- Demonstration of strong and weak scalability using multi-GPU and full Frontera system runs (section V-B); and
- Demonstration of accuracy and convergence of our algorithms (section V-C).

In addition to the above, we also introduce several CPU performance improvements in the DENDRO-GR implementation.

II. RELATED WORK

Adaptivity in numerical relativity codes: AMR or adaptive coordinates are essential in numerical relativity for the simulation of merging binaries and the extraction of gravitational waves. We highlight a few examples here, while a more complete listing can be found in Ref. [26]. Many relativity codes use nested box-based adaptivity in which a sequence of box-in-box regions with varying resolution is used for spatial discretization. For example, the Einstein Toolkit [27] is an open-source AMR code that is based on the Carpet [28] and Cactus [29] frameworks. LAZEV [30], used in some of our comparisons below, is also based on Carpet and Cactus. Octree-based adaptive mesh resolution is widely used in many computational applications. DENDRO-GR [25] and Athena++-GR [31] use octree-based refinement in NR. As an alternative approach, the spectral Einstein code (SpEC) [32] and the NRPy+/SENR [33], [34] code use adaptive coordinates that conform to the spacetime properties in place of, or in addition to, AMR.

NR on GPUs: GPUs have had very limited use in NR. A GPU extension of the SpEC code [35] has been implemented for single black hole spacetimes. The authors presented an overall runtime breakdown for two SpEC benchmarks but did not include a detailed performance analysis for the GPU code. An attempted GPU extension of DENDRO-GR [25] is presented in Ref. [36] and relies on the asynchronous movement of evolution vectors between host and device for each timestep. This work also lacks detailed performance evaluations.

In summary, despite the significance and computational costs associated with numerical relativity, there has been little work on GPU accelerated codes. To provide some context relative to the scale of the problem, the LAZEV code (used in the RIT catalog) requires approximately 30 and 50 days for $q = 2$ and $q = 4$ BBH simulations, respectively. These are for medium resolution runs without assumed symmetries using 256 cores. Note, however, that comparisons between different codes are difficult due to the use of different PDE formulations, discretization schemes, and target accuracies. Due to the complexity of the problem, finding studies that report performance as a function of degrees of freedom and accuracy is difficult. Nevertheless, using our estimates, we believe our DENDRO-GR GPU extension can achieve a 6× speed up over the state of the art.

III. BACKGROUND

A. BSSN formalism of Einstein's equations

We use the Baumgarte-Shapiro-Shibata-Nakamura (BSSN) formulation of Einstein's equations, consisting of 24 coupled, nonlinear, partial differential equations (PDEs). The BSSN [37]–[39] system is strongly hyperbolic, with first-order derivatives in time and second-order spatial derivatives. This formulation is widely used in NR, and we use conventional methods for numerically solving the equations [40], [41]. Spatial derivatives are approximated using finite difference stencils that are $\mathrm{O}(h^6)$ in the grid spacing, h, and the equations are integrated in time using an explicit Runge-Kutta (RK) scheme with global timestepping. We use RK4 with a Courant factor of $\lambda = 0.25$ for the tests below. Kreiss-Oliger (KO) dissipation [42] is added to the solution to eliminate high-frequency noise that can be generated near the black hole singularities.

The BSSN equations are written in tensor form, using the Einstein summation convention [43], i.e., repeated raised and lowered indices are implicitly summed over the values $1, 2, 3$. The BSSN evolution equations are

$$\partial_t \alpha = \mathcal{L}_\beta \alpha - 2\alpha K, \tag{1}$$

$$\partial_t \beta^i = \beta^j \, \partial_j \beta^i + \frac{3}{4} f(\alpha) B^i, \tag{2}$$

$$\partial_t B^i = \partial_t \tilde{\Gamma}^i - \eta B^i + \beta^j \, \partial_j B^i - \beta^j \, \partial_j \tilde{\Gamma}^i, \tag{3}$$

$$\partial_t \tilde{\gamma}_{ij} = \mathcal{L}_\beta \tilde{\gamma}_{ij} - 2\alpha \tilde{A}_{ij}, \tag{4}$$

$$\partial_t \chi = \mathcal{L}_\beta \chi + \frac{2}{3} \chi \left(\alpha K - \partial_a \beta^a \right), \tag{5}$$

$$\partial_t \tilde{A}_{ij} = \mathcal{L}_\beta \tilde{A}_{ij} + \chi \left(-D_i D_j \alpha + \alpha R_{ij} \right)^{TF} + \alpha \left(K \tilde{A}_{ij} - 2 \tilde{A}_{ik} \tilde{A}^k_j \right), \tag{6}$$

$$\partial_t K = \beta^k \partial_k K - D^i D_i \alpha + \alpha \left(\tilde{A}_{ij} \tilde{A}^{ij} + \frac{1}{3} K^2 \right), \tag{7}$$

$$\partial_t \tilde{\Gamma}^i = \tilde{\gamma}^{jk} \partial_j \partial_k \beta^i + \frac{1}{3} \tilde{\gamma}^{ij} \partial_j \partial_k \beta^k + \beta^j \partial_j \tilde{\Gamma}^i -$$

$$\tilde{\Gamma}^j \partial_j \beta^i + \frac{2}{3}\tilde{\Gamma}^i \partial_j \beta^j - 2\tilde{A}^{ij}\partial_j \alpha + 2\alpha\left(\tilde{\Gamma^i}_{jk}\tilde{A}^{jk} - \frac{3}{2\chi}\tilde{A}^{ij}\partial_j \chi - \frac{2}{3}\tilde{\gamma}^{ij}\partial_j K\right). \quad (8)$$

We now define some of the mathematical operations needed to evaluate the equations. Lie derivatives of scalars and tensors with respect to the shift vector, $\vec{\beta}$, can be written as

$$\mathcal{L}_\beta \alpha = \beta^i \partial_i \alpha + w\alpha \partial_i \beta^i \quad (9)$$

$$\mathcal{L}_\beta u_{ij} = \beta^k \partial_k u_{ij} + u_{ik}\partial_j \beta^k + u_{kj}\partial_i \beta^k + w u_{ij}\partial_k \beta^k, \quad (10)$$

where w denotes the tensor weight. The trace-free part of a tensor T_{ij} is defined as

$$(T_{ij})^{TF} = T_{ij} - \frac{1}{3}\tilde{\gamma}_{ij}(\tilde{\gamma}^{lm}T_{lm}). \quad (11)$$

The Christoffel symbols are evaluated as

$$\tilde{\Gamma}^k_{ij} = \frac{1}{2}\tilde{\gamma}^{kl}\left(\partial_j \tilde{\gamma}_{li} + \partial_i \tilde{\gamma}_{lj} - \partial_l \tilde{\gamma}_{ij}\right) \quad (12)$$

$$\Gamma^k_{ij} = \tilde{\Gamma}^k_{ij} - \frac{1}{2\chi}\left(\delta^k_i \partial_j \chi + \delta^k_j \partial_i \chi - \tilde{\gamma}_{ij}\tilde{\gamma}^{kl}\partial_l \chi\right), \quad (13)$$

and D_i denotes the covariant derivative with respect to the spatial metric γ_{ij},

$$D^i D_i \alpha = \gamma^{ij}(\partial_{ij}\alpha - \Gamma^k_{ij}\partial_k \alpha) \quad (14)$$

$$D_i D_j \alpha = \partial_{ij}\alpha - \Gamma^k_{ij}\partial_k \alpha. \quad (15)$$

Evaluating the RHS of equation (6) requires the evaluation of the Ricci tensor. The R_{ij} computation can be split as

$$R_{ij} = \tilde{R}_{ij} + R^\chi_{ij} \quad (16)$$

where

$$\tilde{R}_{ij} = \frac{1}{2}\tilde{\gamma}^{lm}\partial_{lm}\tilde{\gamma}_{ij} + \frac{1}{2}\left(\tilde{\gamma}_{ki}\partial_j \tilde{\Gamma}^k + \tilde{\gamma}_{kj}\partial_i \tilde{\Gamma}^k\right) + \frac{\tilde{\Gamma}^k}{2}(\tilde{\Gamma}_{ijk} + \tilde{\Gamma}_{jik}) + \tilde{\gamma}^{lm}\left((\tilde{\Gamma}^k_{li}\tilde{\Gamma}_{jkm} + \tilde{\Gamma}^k_{lj}\tilde{\Gamma}_{ikm}) + \tilde{\Gamma}^k_{im}\tilde{\Gamma}_{klj}\right) \quad (17)$$

$$M_{ij} = \frac{1}{2\chi}\left(\partial_{ij}\chi - \tilde{\Gamma}^k_{ij}\partial_k \chi\right) - \frac{1}{4\chi^2}\partial_i \chi \partial_j \chi \quad (18)$$

$$R^\chi_{ij} = M_{ij} + \frac{1}{2\chi}\tilde{\gamma}_{ij}\left(\tilde{\gamma}^{kl}\left(\partial_{kl}\chi - \frac{3}{2\chi}\partial_k \chi \partial_l \chi\right)\right) - \tilde{\Gamma}^m \partial_m \chi. \quad (19)$$

We use the Penrose scalar, Ψ_4 [37], [44] for gravitational wave extraction. This time-dependent quantity captures the oscillations in the spacetime geometry. Ψ_4 is extracted by expanding it in a basis of spin-weighted spherical harmonics (ℓ, m modes) on spherical shells; integrations being performed using Lebedev quadrature [45]. Multiple extraction spheres are located between 50–$100M$ (see Figure 4), where, again, M denotes the total mass of the binary.

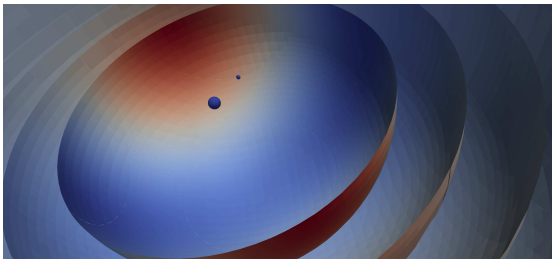

Fig. 4: A black hole binary inspiral simulation snapshot with the corresponding extraction spheres used to extract gravitational waves.

B. Octrees

We use octrees as our primary data structure for spatial grid generation. Octrees are widely used in many computational applications due to their hierarchical structure (see Figure 5) and their ability to achieve point local refinement. In terms of storage, we store only the leaf nodes of the tree since non-leaf nodes can be computed by performing a top-down or bottom-up traversal of the tree. We enforce a 2:1 balancing constraint to smoothly vary refinement over the spatial domain. We rely on the open-source numerical relativity framework DENDRO-GR [23], [25], which supports octree construction, 2:1 balancing [46], [47], and space filling curve-based octree partitioning [48] to ensure scalability of the proposed methods.

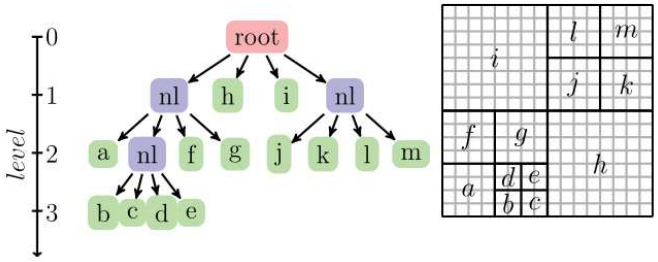

Fig. 5: A simplified illustration of a 2D quadtree (in 3D, it would be an octree) as a data structure to represent a 2D adaptive grid. Note that we start from the root level and perform a hierarchical division of each dimension to generate spatially varying resolution on the computational domain.

C. Notation

Here we summarize octree nomenclature used throughout the paper. A node in the octree is referred to as an octant. Each leaf octant consists of r^3 uniformly placed grid points. The points that violate the geometric conformality are referred to as "hanging" points. Duplicate and hanging points are removed during the grid construction phase. Each leaf octant is padded with k points per direction, and a padded octant is referred to as a "patch" (see Figure 6). Each patch consists of $(r+2k)^3$ grid points. To enable 6th order finite difference computations, we set $r = 7$ and $k = 3$. The DENDRO-GR framework provides two maps for each octree partition: (1) O2O, an

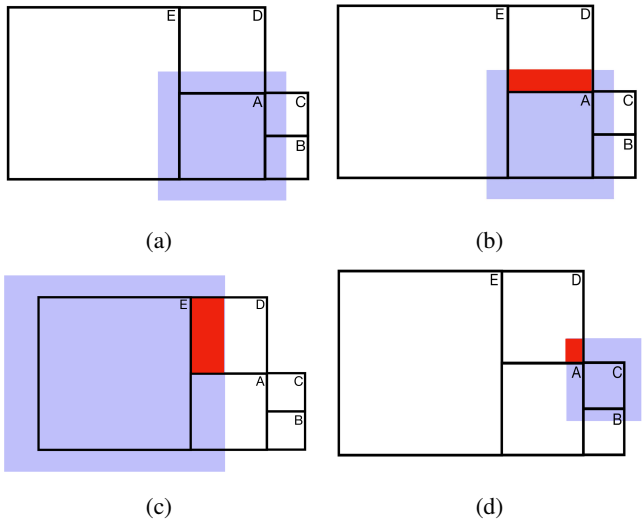

Fig. 6: In this figure, the black squares define octants A to E, with each octant labeled in its top right corner. The shaded blue area in 6a shows the padding zone of octant A. The red zones in 6b, 6c, 6d denote the padding zones that overlap with octant D. The above regions are computed using information from octant D.

octant to face neighbors map, and (2) O2N, an octant to its corresponding grid point map. These maps enable numerical computations on the octree through looping over octants.

D. Performance models

We use slow-fast memory models for analyzing single-GPU kernels, the slow memory representing the GPU main memory and the fast memory a combination of L2 and registers. This assumes the execution model defined by the abstract random access machine (RAM) [49]. This helps us characterize the upper bounds for the ideal attainable performance. We consider two models. The first assumes infinite fast memory; the second uses a finite-sized fast memory. Let τ_f be the machine time per double precision FLOP; τ_m the RAM access/byte; C_R the entire register file; C_L the L2 size and assume the time to load and store from it is $\ell\tau_m$, $\ell < 1$; and f the double precision flops and loads/store m bytes needed in the computation. Let $Q = f/m$ be the arithmetic intensity (AI) of the kernel.

In the infinite-cache model, the kernel time is $T^{\infty}(f,m) = f\tau_f + m\tau_m = m\tau_m\left(1 + \frac{\tau_f}{\tau_m}Q\right)$. For the finite-cache the memory costs become $\tau_m m\left(\frac{m}{C_L} + \ell\frac{m}{C_R}\right)$, because we need at least $\frac{m}{C}$ loads from RAM to the L2 cache and $\frac{m}{C_R}$ loads from the L2 to the registers. Then, using the machine-specific parameter $\xi = \left(\frac{1}{C_L} + \ell\frac{1}{C_R}\right)$, we obtain $T(f,m) = m\tau_m(m\xi) + f\tau_f$ or $T(f,m) = m\tau_m\left(\max(1, m\xi) + \frac{\tau_f}{\tau_m}Q\right)$. If we ignore the FLOPS term, $T = m\tau_m \max(1, m\xi)$. For the A100, $C_L =$ 40MB, $C_R =$ 27MB, $\ell \approx 1/4$, and thus, $\xi \approx$4e-8; $\tau_f =$1.0e-13s, $\tau_m =$6.4e-13s, and τ_f/τ_m is 0.16. If $Q < 1/0.16 = 6.25$, the FLOPS are negligible and the kernel is bandwidth limited.

Our kernels are memory bound. For example, m can be up to 2MB for just a single octant; for 108 octants, one for each A100 SM, $m\xi \approx 10$. This analysis does not take into account the structure of the calculations per octant, but shows the difficulty of obtaining good performance. Next, we discuss several optimizations that exploit these dependencies.

IV. Methodology

In this section, we present novel algorithmic contributions that enable efficient numerical relativity simulations on GPU architectures. The key computation in numerical relativity is the time integration of the governing BSSN equations, which describe the evolution of spacetime. An overview of our GPU time evolution is presented in Algorithm 1. Minimizing synchronous data movement between the host and the device is crucial to achieving high-compute throughput. In the proposed approach, the re-grid operation (i.e., re-discretization to capture the evolving fields) is the only operation that requires synchronous data movement between the host and the device. The host generates the grid (or re-grids) and passes the data to the device. The time integration is entirely performed on the device until the host issues the next re-grid operation. The host uses asynchronous streams to extract the gravitational waves (e.g., every 16 timesteps) from the evolved quantities.

Algorithm 1 Overview: Time evolution

Require: u state at $t = t_0$, T: time horizon, Δt timestep size, f_r: re-grid frequency
Ensure: u state at $t = T$

```
N ← (T − t_o)/ΔT
for each i ∈ [0:N:f_r] do                ▷ 0 to N with f_r increments
    M ← construct_grid(u)                ▷ use DENDRO-GR
    v ← host_to_device(u)
    for each f_r timesteps do
        v ← halo_exchange(v)             ▷ synchronize partitions
        v̂ ← octant-to-patch(v)           ▷ compute octant patches
        ŵ ← RHS(v̂, t)                    ▷ evaluate RHS
        w ← patch-to-octant(ŵ)           ▷ revert back to octants
        v ← AXPY(w, v, Δt)               ▷ evolve state v = v + Δtw
    u ← device_to_host(v)
return u
```

A. Computing padding zones

The computation of padding zones is referred to as the *octant-to-patch* operation. This computation requires appropriate interpolations or injections at coarser and finer octant boundaries while performing direct data copy between octants at the same resolution.

loop-over-patches: The current DENDRO-GR supports padding zone computation via loop-over-patches while *gathering* the padding information from neighboring octants with proper interpolations. This creates redundant interpolations between coarser and finer patch boundaries. Most importantly, gathering octant information with random memory accesses will not work well on GPU architectures.

loop-over-octants: To minimize the number of interpolations and increase data locality, we propose the computation of

padding zones by looping over octants. In the octant-loop each octant *scatters* its information to neighboring patches with appropriate interpolations, injections, or a direct copy (see Figure 6). We pre-compute and store the octant to neighboring patches map (O2P) at the grid generation phase to be used during padding zone computation. The 2:1 balance constraint ensures that for any octant, all its neighbors can differ by at most a single level. This simplifies the different cases to be handled during the *octant-to-patch* operation. For a given octant o, its neighbor octant can be at the same refinement level or one level coarser or finer than o (see Algorithm 2). Single core CPU performance comparison of the two approaches is presented in Figure 7. The above comparison shows that looping-over-octants is significantly more efficient (3x faster) than looping-over-patches, due to increased data locality and reduced redundant interpolation.

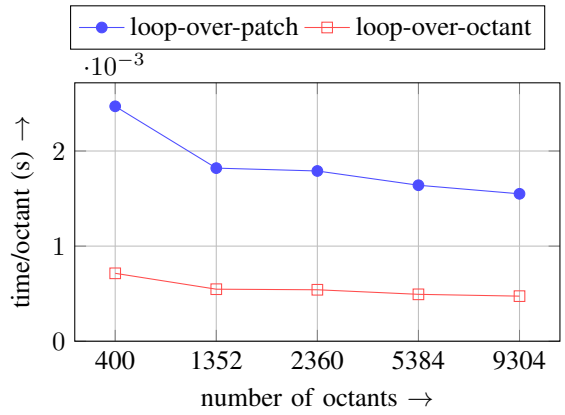

Fig. 7: A single core CPU comparison of padding zone computation with loop-over-patches vs. loop-over-octants. The proposed looping-over-octants with scattering approach has higher data locality (i.e., during the read operation) with no redundant interpolations.

Algorithm 2 *octant-to-patch* computation

Require: E-octant list, O2B, u-field variable
Ensure: $\hat{u}$ - u with filled padding zones

```
e ← gpu_block_id_x
u_e ← load(u[O2N[e]])                    ▷ global to shared load
I[0,1] ← load(Ig)                        ▷ const to shared load
if is_hanging(e) then
    u_e ← interp_hanging(u_e)            ▷ shared to shared load/store
for b ∈ O2B[e] do
    if same resolution then
        û[b] ← copy(u_e)                 ▷ shared to global store
    else if b is coarser then
        û[b] ← inject(u_e)               ▷ shared to global store
    else if b is finer then
        û[b] ← interp(u_e, I)            ▷ shared to global store
return
```

The *octant-to-patch* kernel is launched with kernel grid dimensions of $(|E|, dof, 1)$ with $(r, r, 1)$ thread block, where $|E|$ denotes the number of octants and dof denotes the number of degrees of freedom per grid point. Each GPU block reads the octant nodal values from global to shared memory. All of the required interpolations or injections are performed using the shared memory. Once the octant data is ready to be scattered, data is moved from the shared to global memory according to the O2P map (see Figure 8).

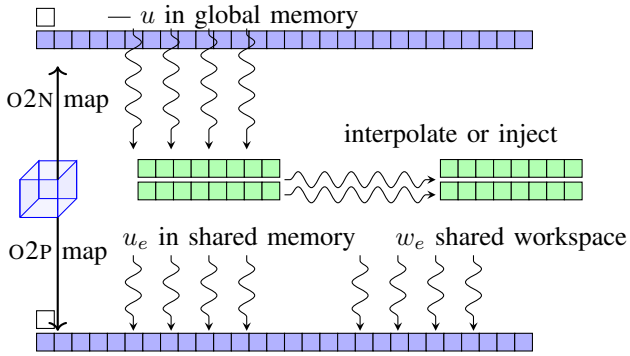

Fig. 8: An overview of the data movement during the *octant-to-patch* operation for GPUs. Each GPU block operates on a single octant in the local partition. A global to shared load is performed to move the octant nodal values using the O2N map. All the required interpolations and injections are performed in the block shared memory. Following this, a shared to global store operation is performed to scatter the octant nodal values to their corresponding neighboring blocks. This uses the O2P map to resolve the neighboring blocks of an octant.

Interpolations: The required interpolations are performed as tensor products of 1D interpolation operators. First, the thread block operates on xy slices performing interpolation in the x direction, followed by interpolations in the y directions. Finally, the thread block operates in xz slices, performing interpolations in the z direction. During this process, appropriate synchronizations are deployed. A single coarser to finer interpolation requires $\mathcal{O}(3(2r-1)r^3)$ operations.

In our computations, the padding zones are used only during the stencil evaluations. Once the required stencils are applied, padding zones are discarded and the fields are reverted back to the unpatched representation. The above is referred to as the *patch-to-octant* operation. During the *patch-to-octant* operation, each octant patch copies its internal grid points (i.e., grid points not in padding zones) to the unpatched representation.

Performance bounds: During *octant-to-patch* operations we read 1D interpolation operators ($2r^2$) and octant grid points values (r^3), and we write octant grid point values (r^3), padding zones along with faces ($6r^2k$), edges ($12rk^2$), and vertex corners ($8k^3$). The number of flops performed during the *octant-to-patch* depends on the adaptivity structure of the grid. An interpolation operation has $3(2r-1)r^3$ operations. The maximum number of interpolations (i.e., octant to all its 8 children) is performed when all the neighboring blocks of

an octant are finer than the octant resolution. Therefore, the upper bound for the arithmetic intensity of the *octant-to-patch* operation (Q_U) is given by

$$Q_U \leq \frac{8 \times 3(2r-1)r^3}{8\left(2r^2+2r^3+12rk^2+6r^2k+8k^3\right)} \approx 5.07 \quad (20)$$

Therefore, this kernel is memory bound as discussed in section III-D. The memory requirements per octant per state variable are small, so the infinite cache model should be predictive as $m\xi \ll 1$.

B. Computing BSSN equations

In this section, we discuss the BSSN RHS evaluation and related challenges. We define two main components of the RHS: computing the derivatives (denoted by $\mathcal{D}$) and the algebraic combination of the derivatives (denoted by $\mathcal{A}$). Since we are using an explicit RK scheme, the RHS evaluation is the key computational kernel for evolving these equations. All 24 field variables require all first partial derivatives (i.e., $3 \times 24 = 72$); variables $\alpha, \beta^i, \chi, \tilde{\gamma}_{ij}$ require all second derivatives (i.e., $6 \times 11 = 66$). Additionally for all evolution variables, we need KO dissipation derivatives (i.e., $3 \times 24 = 72$). **Hence, the RHS computation requires 210 derivative evaluations.** Furthermore, these derivatives are combined in $\mathcal{A}$ in a highly connected manner. The easy way to implement it is to precompute these derivatives with a separate kernel and then combine them in $\mathcal{A}$. This turns out to be slow, but more importantly imposes significant memory constraints.

The RHS is evaluated on octants and the patch is used only in the derivative computations. Therefore, each octant patch is mapped to a GPU block consisting of (r, r, r) thread dimensions. This enables the storage of the derivatives at each grid point in the thread-local memory (see Figure 9).

The $\mathcal{A}$ component of the RHS is a mapping between 234 (i.e., 24 + 210) inputs to 24 outputs. Due to the complexity of these equations, manually writing code is nearly impossible. For example, Kranc [50] is a commonly used framework to generate code for the Einstein Toolkit [27]. Other projects, such as LAZEV [51] and SpEC [32] use custom Mathematica [52] scripts to generate executable C code. Other efforts use SymPy [53] for NR code generation, such as SymPyGR [53] and NRPy+ [54]. Both of these projects use SymPy's common sub-expression elimination (CSE) to reduce the number of operations in the evaluation of $\mathcal{A}$. We use SymPyGR for code generation with CSE as the baseline code for RHS performance evaluations. The high data dependencies in $\mathcal{A}$ will cause register spilling during evaluations. The minimum number of registers required to evaluate $\mathcal{A}$ is an unsolved problem.

The CSE approach creates ∼900 temporary variables, which reduces the number of operations significantly but causes heavy register spilling. We argue that minimizing compute operations is not ideal, in the presence of register spilling, since memory operations have significantly higher overhead compared to compute operations. In the CSE code generation approach, the final expressions are evaluated once all of the intermediate sub-expressions are evaluated. The above can

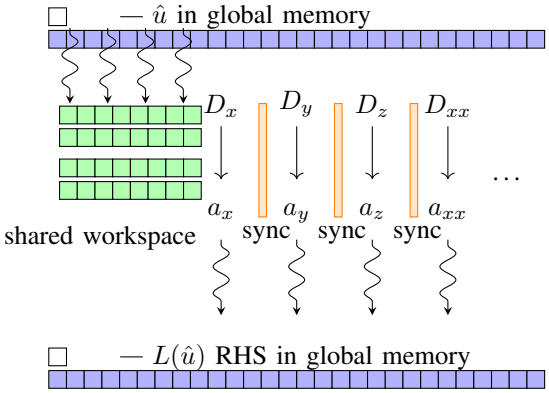

Fig. 9: An overview of the data movement during fused RHS evaluation. Each GPU block operates on a single octant patch. Each evolution variable is moved from the global to block shared memory one at a time. Once a variable is in the block shared memory, all its derivatives (i.e., $D_x, D_y, D_z, \ldots$) are computed using a shared memory workspace. Appropriate thread synchronizations are enforced to resolve race conditions in subsequent stencil applications. Once the stencil is computed, each thread grabs its corresponding point and stores the computed derivative values (i.e., $a_x, a_y, a_z, \ldots$) in the thread-local memory. Once all the required derivatives are computed, the corresponding RHS is updated.

increase the live range of the allocated temporary variables, and cannot rely on the compiler to reorder the expressions to reduce the live range of variables. To reduce the register pressure, we propose a binary reduction-based code generation for $\mathcal{A}$ evaluation. The key motivation for the above is to reduce the live ranges of allocated thread-local temporary variables. For each (i.e., 24 equations) we build its computational graph (see Figure 10) using the SymPy expression tree and NetworkX package [55]. Let G be the composed graph (i.e.,

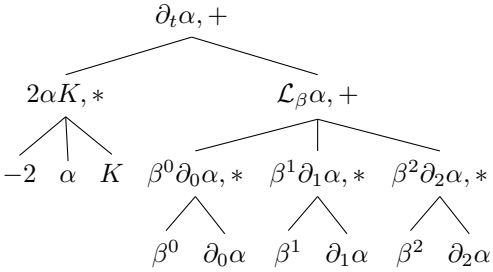

Fig. 10: A simple illustration of the associated computational graph for the algebraic component of the $\partial_t\alpha$ RHS evaluation.

directed acyclic graph with 2516 nodes and 6708 edges) of the constructed 24 subgraphs. A valid traversal of the graph ensures that node v is visited only when its descendants u have been computed and can be used to compute node v. The above traversals are not unique (e.g., any topological ordering of G). As a heuristic, we employ the traversal order generated by the topological sort of line graph of G. The binary

Algorithm 3 *visit_node*(v)

Require: $G = (V, E), v \in V$, $B-$ local memory

```
v.DONE ← true
for u ∈ V.descendants do
    store(v, u, B)                ▷ Store in local memory
    reduce(u, v)
    remove edge (u, v) from G
    if degree(u) is 0 then
        evict (u, B)
if v is a final expr then
    store_to_global(v)
    if degree(v) is 0 then
        evict(v,B)
return
```

reduction-based code generator is summarized in Algorithm 3. For the generated topological traversal, we visit each node v of the graph and store v and its descendants in the thread-local memory followed by reduction based on the node operation (i.e., +, *). Once the reduction is performed, edge (u, v) is removed from G, and u is evicted from the thread-local memory once it becomes disconnected from G. The $\mathcal{A}$ kernel generated from the above is referred to as "binary-reduce." This approach reported a maximum of 675 live allocated temporary variables during the traversal of the graph.

We consider another variation of the RHS evaluation to minimize register spills due to thread-local stored derivative variables. The motivation is to compute the RHS of an equation, as soon as its derivatives are ready. This can help to reduce the live range of computed derivatives. This approach is referred to as "staged + CSE."

Next we present a detailed performance comparison of the existing SymPyGR approach (baseline) and the proposed "binary-reduce" and "staged + CSE" approaches. We use the NVIDIA A100 GPU with CUDA compiler version 11.4 with `--ptxas-options=-O3` compiler option. Table II shows the compiler reported ptx-spill loads and stores for `__launch_bounds__(343,3)` (i.e., maximum 56 registers per thread). Figure 11 shows a performance comparison of the above three approaches with varying number of octants.

RHS variation	ptx-spill stores (bytes)	ptx-spill loads (bytes)	average speedup w.r.t. SymPyGR
SymPyGR	15892	33288	1.00x
binary-reduce	10176	22012	1.55x
staging + CSE	8876	22028	1.76x

TABLE II: A summary of the compiler reported spill loads and stores for the generated RHS variations. The last column represents the average speedup reported compared to the SymPyGR baseline.

Performance bounds: The arithmetic intensities of the complete RHS (Q_L) and $\mathcal{A}$ (Q_A) computations assuming a random access machine model is given by eq. (21a) and eq. (21b), where O_A denotes the approximate number of operations in the $\mathcal{A}$ component.

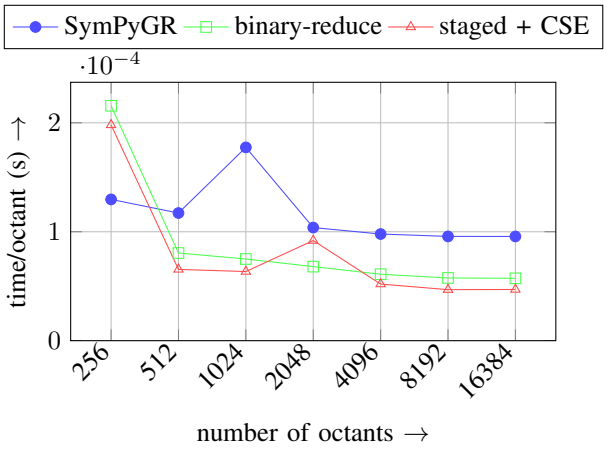

Fig. 11: The reported time per octant for 10 RHS evaluations, using SymPyGR code generator, binary reduction based evaluations, and staging + CSE approaches on a single NVIDIA A100 GPU.

$$Q_L = \frac{r^3(33(2d^2-1)+177(2d-1)+O_A)}{8(24(r+2k)^3+24r^3)} \approx 6.68 \quad (21a)$$

$$Q_A = \frac{r^3(O_A)}{8(24\times 2+210)r^3} \approx 1.94 \quad (21b)$$

Again we see that these are not large enough to overcome the machine imbalance, especially given the memory requirements of the RHS evaluations, which result in a large $m\xi$ (section III-D).

V. Results

This section describes numerical and performance evaluation of the proposed computational methodology. A detailed single node performance analysis is presented in section V-A. We performed our experiments on Frontera and Lonestar 6 at the Texas Advanced Computing Center (TACC). Frontera has 8K Intel Cascade Lake nodes [56] and Lonestar 6 has 16 dual-NVIDIA A100 nodes. All GPU-CPU comparisons were done on a single NVIDIA A100 GPU. Frontera is used for performing a large weak scaling study, and the GPU strong and weak scalability tests are performed on Lonestar 6 (section V-B). Accuracy and convergence of binary black hole simulations are presented in section V-C.

A. Single node performance

Padding zones: Unlike the RHS, this computation is sensitive to the grid refinement level and its overall structure. For example in a uniform grid, no interpolations take place. In contrast, in real simulations, the grid changes significantly, especially during the inspiral stage (see Figure 12) and following merger (see Figure 13). For performance evaluation of the *octant-to-patch* operation we construct five different grids m_i for $i = \{1, 2, 3, 4, 5\}$ where moving from m_1 to m_5 decreases the adaptivity (i.e. the grid becomes more uniform). Table III shows the empirically observed arithmetic intensity values reported by the NVIDIA nv-compute tool. The corresponding

empirical roofline plot for the above is shown in Figure 14. For *octant-to-patch* operations on average, we observe roughly 900 GFlops/s for all grids. The *patch-to-octant* operation is purely a data movement kernel with zero arithmetic intensity.

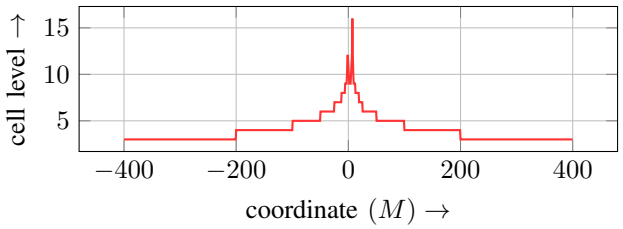

Fig. 12: The octants level variation along the x coordinate for 1:8 mass ratio binary system during the inspiral stage.

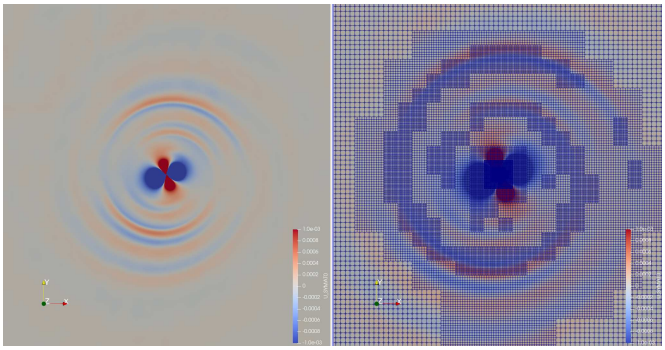

Fig. 13: A snapshot of the binary black hole system after the merger. After the merger the grid adaptivity changes to capture the radially outgoing gravitational waves.

grid	octants × dof	reported AI *octant-to-patch*	*octant-to-patch* (ms)	*patch-to-octant* (ms)
m_1	400 × 24	4.07	1.31	0.064
m_2	1352 × 24	2.52	3.38	0.2
m_3	2360 × 24	2.20	5.60	0.3
m_4	5384 × 24	1.90	11.92	0.8
m_5	9304 × 24	1.74	19.94	1.56

TABLE III: A summary of the *octant-to-patch* and *patch-to-octant* operations observed operational intensity and execution times on a single NVIDIA A100 GPU where each grid point consists of 24 field variables. The arithmetic intensity of the *octant-to-patch* operation in RAM execution model is bounded by $Q_u \leq 5.07$.

RHS evaluation: We compare one A100 GPU to two EPYC sockets on Lonestar 6 for a varying number of octants (see Figure 15). On the EPYC, patch-level parallelism is achieved using OpenMP. A roofline performance analysis for the RHS evaluation is presented in Figure 14. The observed arithmetic intensity for the overall RHS is $\approx 0.62 \ll 6.68$, which is expected due to L2 misses and register spilling (section III-D).

BSSN solver: We present a detailed performance comparison between the CPU DENDRO-GR and the proposed GPU extension. To make the comparison algorithmically fair, we use loop-over-octant based *octant-to-patch* operations for both

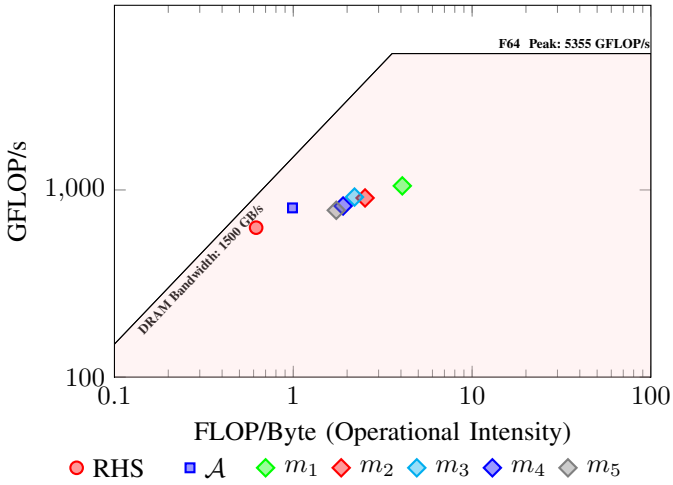

Fig. 14: The empirical roofline performance evaluation for the key computational kernels on a single NVIDIA A100 GPU. Overall RHS evaluation is denoted by RHS and the algebraic combination of derivatives is denoted by $\mathcal{A}$. The *octant-to-patch* operation is performed for varying grids $m_i (i = 1, 2, 3, 4, 5)$ with decreasing adaptivity. Highly adaptive grids have a higher number of interpolations, leading to higher arithmetic intensities and vice versa. The overall RHS evaluation and *octant-to-patch* operations achieve compute throughout of 700 GFlops/s and 900 GFlops/s respectively.

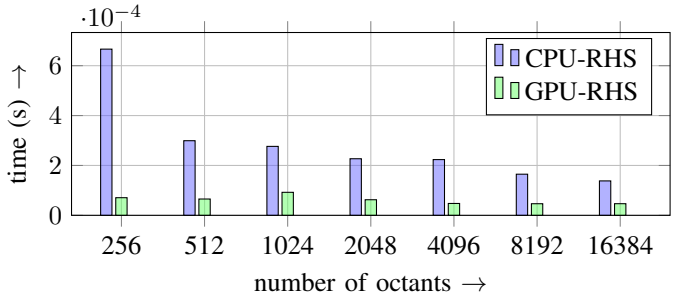

Fig. 15: Wall clock time to compute patches (padding zones) for 10 RHS evaluations using one A100 vs. two EPYC sockets. The CPU RHS evaluation is parallelized using OpenMP, for which the octants are equally partitioned across 128 threads. The GPU results show the execution time for RHS evaluation.

codes. We report wall clock time to perform five RK4 steps in Figure 16. Once the octant patches have been constructed, the RHS evaluation does not depend on the grid refinement.

B. Parallel scalability

We conduct GPU/CPU strong and CPU weak scaling studies for the BSSN evolution on binary black hole grids. In strong scaling, we fix problem size at 125M unknowns and perform 5 RK4 timesteps (see Figure 17). We observe parallel efficiencies of 97%, 89%, and 64% for 4, 8, and 16 GPUs respectively. For the CPU strong scaling results, we observe parallel efficiencies of 93%, 79%, and 66%. Figure 18 shows the conducted weak scalability study with 35M unknowns per

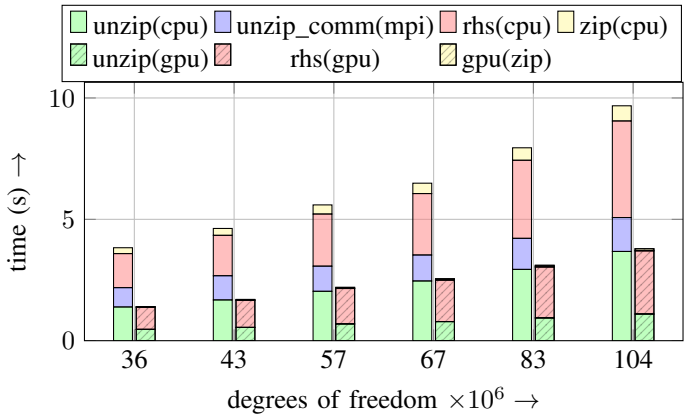

Fig. 16: Overall wall clock time for 5 RK4 timesteps on one A100 on a two-socket EPYC node for binary black hole grids with problem size varying from 36M to 104M unknowns.

GPU across 16 A100 GPUs with average parallel efficiency of 83%.

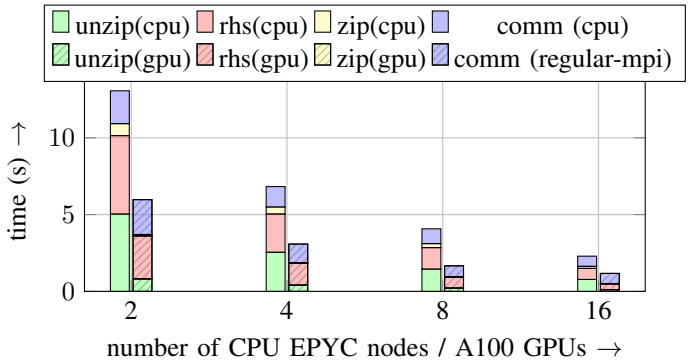

Fig. 17: Strong scaling: Overall wall clock time for 5 RK4 timesteps for binary black hole grids with a fixed problem size of 257M unknowns with increasing number of GPUs.

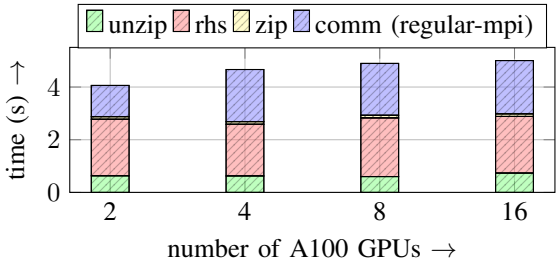

Fig. 18: Weak scaling: Overall wall clock time for 5 RK4 timesteps for binary black hole grids with increasing problem size. The above weak scalability study performed with approximately 35M unknowns per GPU where the largest problem consists of 560M unknowns.

We perform weak scalability for the overall framework across 4,096 nodes on the Frontera supercomputer. Keeping the number of unknowns per core constant is a challenging task in an adaptive mesh refinement setting. For this experiment, we start with an octree grid generated for the binary black hole problem and increase the refinement radius black hole locations until the desired number of unknowns per core is reached. For the weak scaling experiment (see Figure 20), we use roughly 500K unknowns per core. The largest problem size consists of 118B unknowns, which used 4096 nodes on Frontera.

C. Accuracy and convergence

To establish the accuracy of our new framework, we first compare the gravitational waveforms for an equal-mass binary computed with the CPU code to those computed using the well-known LazEv [30] code. In Figure 19, we plot the difference between the CPU waveforms and a high-resolution LazEv waveform as a function of the error tolerance used in the refinement algorithm. As the refinement error tolerance is decreased, the waveforms converge to the high-resolution LazEv waveform. This gives confidence that the waveforms computed with sparse refinement using an octree are correct.

Gravitational waveforms computed with the GPU code for $q = 1$ and $q = 2$ binaries are plotted against the corresponding CPU waveforms in Figure 21. These waveforms have also been verified in comparisons with LazEv. The overall wall clock times used for these production runs, including file I/O and re-grid, are presented in Table IV. Again, these waveforms match very closely, indicating that the GPU version of the code accurately computes the gravitational waveforms for merging black holes.

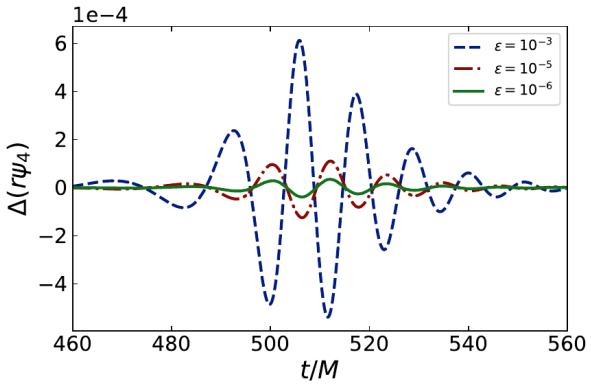

Fig. 19: The convergence of the numerically computed waveforms with increased refinement (i.e., decreasing ϵ) surrounding the black hole locations. The plot shows the difference between the extracted GWs using LazEv and the proposed approach for the real part of the Ψ_4 scalar. We can see that with increasing refinement the computed waveforms converge to the LazEv waveform.

VI. Conclusions

This paper presents the first known gravitational waveforms computed using solutions of the full Einstein equations on GPUs. The waveforms were verified through comparisons to other known solutions. We optimized the core computational kernels of the algorithms and presented a detailed analysis of

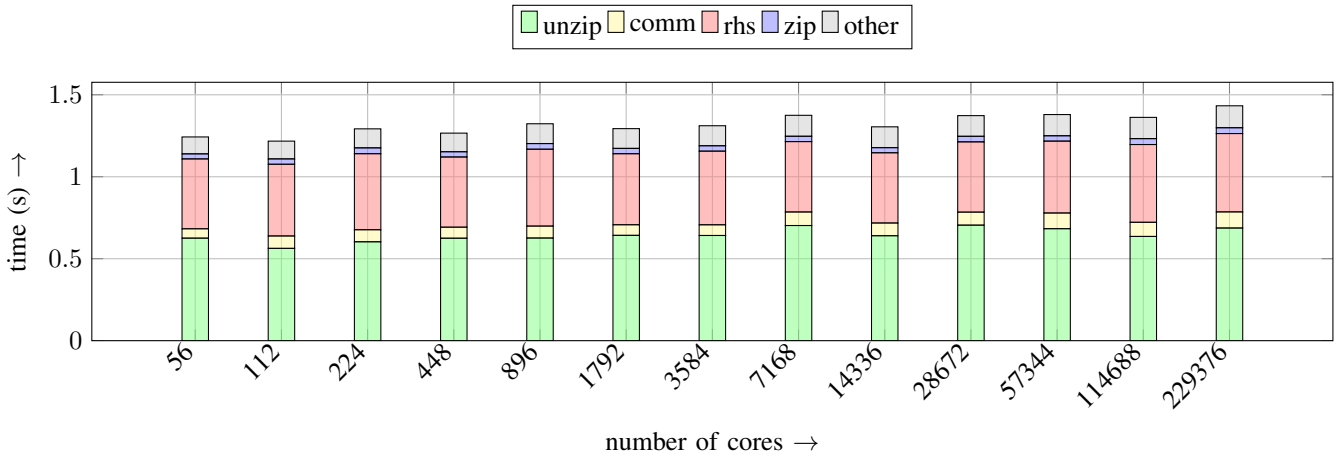

Fig. 20: Weak scalability study on TACC's Frontera, for evolving the BSSN formulation of the Einstein equations. Shown is the overall cost breakdown to perform a single RK4 step, using 6th order finite difference stencils. For the above study, we use approximately 500K unknowns per core, where the largest problem contains a total of 118B unknowns on the grid.

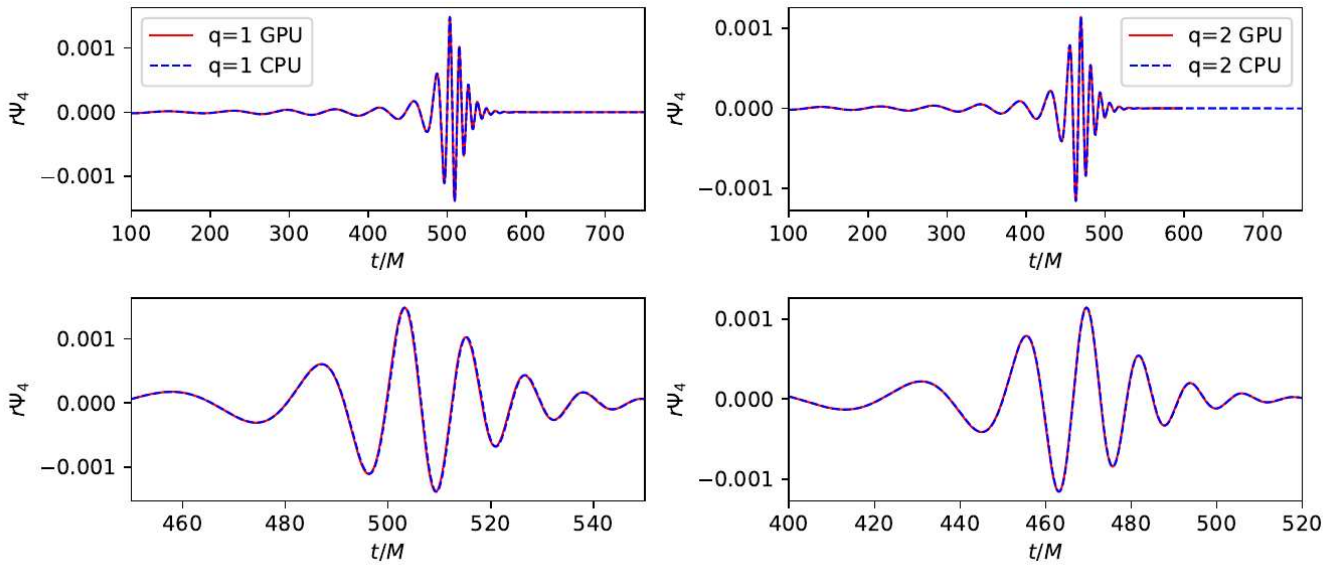

Fig. 21: Numerically computed gravitational waves for mass ratios $q = 1$ and $q = 2$ binary black hole mergers. The plotted signal corresponds to the $l = 2$, $m = 2$ mode of the Ψ_4 projection. The plot shows the extracted waveforms using the proposed GPU approach compared to the previously verified result computed using the CPU code.

TABLE IV: The table summarizes wall clock time to evolve binary black holes with mass ratios $q = 1, 2, 4, 8$, where T denotes the time horizon the binary system was evolved to (where M denotes the total mass of the binary, and $M = 1$). The $q = 8$ BBH is not evolved to completion, but included in the table to provide an overall wall clock time estimate.

Mass ratio $q = m_1/m_2$	$\Delta x_{\min}$ (BH1)	$\Delta x_{\min}$ (BH2)	GPUs NVIDIA A100	T	timesteps	Wall time (hrs)
1	1.62e-2	1.62e-2	4	748M	183K	87
2	8.13e-3	3.25e-2	4	600M	252K	96
4	4.06e-3	3.25e-2	4	602M	506K	129
8	2.03e-3	3.25e-2	8	1400M	4M	388

their performance. We demonstrated that the time-to-solution for computing gravitational waveforms in NR can be significantly decreased by using GPUs.

The increased computational performance achievable with GPUs will allow researchers to construct larger and more accurate NR gravitational wave catalogs. A denser sampling of the parameter space is urgently needed, since a wider range of sources will soon be detected. Finally, it is imperative that the accuracy of NR waveforms keeps pace with the rapid advances in detector technology. New algorithms and computational approaches, such as those investigated here, will be required to meet this computational challenge.

Acknowledgment: This work is supported by the U.S. National Science Foundation (NSF) awards OAC-1808652, PHY-1912930, CCF 1922862; by the U.S. National Aeronautics and Space Administration (NASA) grant 80NSSC20K0528 by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Number DE-SC0019393; and by

the U.S. Department of Energy, National Nuclear Security Administration Award Number DE-NA0003969. Any opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the DOE, NASA and NSF. This work used computing resources from the Texas Advanced Computing Center allocations TG-PHY180054 and ASC21034.

REFERENCES

[1] LIGO, "Laser interferometer gravitational-wave observatory," 2022. [Online]. Available: https://www.ligo.caltech.edu/

[2] B. P. Abbott, R. Abbott, T. Abbott, M. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. Adhikari *et al.*, "Observation of gravitational waves from a binary black hole merger," *Physical review letters*, vol. 116, no. 6, p. 061102, 2016.

[3] B. Abbott, R. Abbott, T. Abbott, S. Abraham, F. Acernese, K. Ackley, C. Adams, R. Adhikari, V. Adya, C. Affeldt *et al.*, "Gwtc-1: a gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs," *Physical Review X*, vol. 9, no. 3, p. 031040, 2019.

[4] R. Abbott, T. Abbott, S. Abraham, F. Acernese, K. Ackley, A. Adams, C. Adams, R. Adhikari, V. Adya, C. Affeldt *et al.*, "Gwtc-2: compact binary coalescences observed by ligo and virgo during the first half of the third observing run," *Physical Review X*, vol. 11, no. 2, p. 021053, 2021.

[5] A. Buonanno and T. Damour, "Effective one-body approach to general relativistic two-body dynamics," *Phys.Rev.*, vol. D59, p. 084006, 1999.

[6] ——, "Transition from inspiral to plunge in binary black hole coalescences," *Phys. Rev.*, vol. D62, p. 064015, 2000.

[7] S. Husa, S. Khan, M. Hannam, M. Pürrer, F. Ohme, X. Jiménez Forteza, and A. Bohé, "Frequency-domain gravitational waves from nonprecessing black-hole binaries. I. New numerical waveforms and anatomy of the signal," *Phys. Rev. D*, vol. 93, no. 4, p. 044006, 2016.

[8] S. Khan, S. Husa, M. Hannam, F. Ohme, M. Pürrer, X. Jiménez Forteza, and A. Bohé, "Frequency-domain gravitational waves from nonprecessing black-hole binaries. II. A phenomenological model for the advanced detector era," *Phys. Rev. D*, vol. 93, no. 4, p. 044007, 2016.

[9] J. Blackman, S. E. Field, M. A. Scheel, C. R. Galley, C. D. Ott, M. Boyle, L. E. Kidder, H. P. Pfeiffer, and B. Szilágyi, "Numerical relativity waveform surrogate model for generically precessing binary black hole mergers," *Phys. Rev. D*, vol. 96, p. 024058, Jul 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.96.024058

[10] M. Boyle, D. Hemberger, D. A. Iozzo, G. Lovelace, S. Ossokine, H. P. Pfeiffer, M. A. Scheel, L. C. Stein, C. J. Woodford, A. B. Zimmerman *et al.*, "The sxs collaboration catalog of binary black hole simulations," *Classical and Quantum Gravity*, vol. 36, no. 19, p. 195006, 2019.

[11] J. Healy and C. O. Lousto, "Third rit binary black hole simulations catalog," *Physical Review D*, vol. 102, no. 10, p. 104018, 2020.

[12] K. Jani, J. Healy, J. A. Clark, L. London, P. Laguna, and D. Shoemaker, "Georgia tech catalog of gravitational waveforms," *Classical and Quantum Gravity*, vol. 33, no. 20, p. 204001, 2016.

[13] T. Dietrich, D. Radice, S. Bernuzzi, F. Zappa, A. Perego, B. Brügmann, S. V. Chaurasia, R. Dudi, W. Tichy, and M. Ujevic, "CoRe database of binary neutron star merger waveforms," *Classical and Quantum Gravity*, vol. 35, no. 24, p. 24LT01, nov 2018. [Online]. Available: https://doi.org/10.1088/1361-6382/aaebc0

[14] M. Evans, R. X. Adhikari, C. Afle, S. W. Ballmer, S. Biscoveanu, S. Borhanian, D. A. Brown, Y. Chen, R. Eisenstein, A. Gruson *et al.*, "A horizon study for cosmic explorer: science, observatories, and community," *arXiv preprint arXiv:2109.09882*, 2021.

[15] P. Amaro-Seoane, H. Audley, S. Babak, J. Baker, E. Barausse, P. Bender, E. Berti, P. Binetruy, M. Born, D. Bortoluzzi, J. Camp, C. Caprini, V. Cardoso, M. Colpi, J. Conklin, N. Cornish, C. Cutler, K. Danzmann, R. Dolesi, L. Ferraioli, V. Ferroni, E. Fitzsimons, J. Gair, L. G. Bote, D. Giardini, F. Gibert, C. Grimani, H. Halloin, G. Heinzel, T. Hertog, M. Hewitson, K. Holley-Bockelmann, D. Hollington, M. Hueller, H. Inchauspe, P. Jetzer, N. Karnesis, C. Killow, A. Klein, B. Klipstein, N. Korsakova, S. L. Larson, J. Livas, I. Lloro, N. Man, D. Mance, J. Martino, I. Mateos, K. McKenzie, S. T. McWilliams, C. Miller, G. Mueller, G. Nardini, G. Nelemans, M. Nofrarias, A. Petiteau, P. Pivato, E. Plagnol, E. Porter, J. Reiche, D. Robertson, N. Robertson, E. Rossi, G. Russano, B. Schutz, A. Sesana, D. Shoemaker, J. Slutsky, C. F. Sopuerta, T. Sumner, N. Tamanini, I. Thorpe, M. Troebs, M. Vallisneri, A. Vecchio, D. Vetrugno, S. Vitale, M. Volonteri, G. Wanner, H. Ward, P. Wass, W. Weber, J. Ziemer, and P. Zweifel, "Laser interferometer space antenna," 2017. [Online]. Available: https://arxiv.org/abs/1702.00786

[16] M. Punturo *et al.*, "The Einstein Telescope: A third-generation gravitational wave observatory," *Class. Quant. Grav.*, vol. 27, p. 194002, 2010.

[17] J. Miller, L. Barsotti, S. Vitale, P. Fritschel, M. Evans, and D. Sigg, "Prospects for doubling the range of advanced ligo," *Phys. Rev. D*, vol. 91, p. 062005, Mar 2015. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.91.062005

[18] L. Barsotti, L. McCuller, M. Evans, and P. Fritschel, "The a+ design curve," *Tech. Report No.*, pp. LIGO–T1 800 042, 2018. [Online]. Available: https://dcc.ligo.org/LIGO-T1800042/public

[19] C. Cutler and M. Vallisneri, "Lisa detections of massive black hole inspirals: Parameter extraction errors due to inaccurate template waveforms," *Phys. Rev. D*, vol. 76, p. 104018, Nov 2007. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.76.104018

[20] M. Pürrer and C.-J. Haster, "Gravitational waveform accuracy requirements for future ground-based detectors," *Phys. Rev. Res.*, vol. 2, no. 2, p. 023151, 2020.

[21] D. Ferguson, K. Jani, P. Laguna, and D. Shoemaker, "Assessing the readiness of numerical relativity for LISA and 3G detectors," *Phys. Rev. D*, vol. 104, no. 4, p. 044037, 2021.

[22] L. F. Diachin, R. Hornung, P. Plassmann, and A. Wissink, "Parallel adaptive mesh refinement," in *Parallel Processing for Scientific Computing*, ser. Software, Environments, and Tools, M. A. Heroux, P. Raghavan, and H. D. Simon, Eds. SIAM, 2006, no. 20, ch. 8, pp. 143–162.

[23] github, "Dendro-gr: Numerical relativity with octree based wavelet adaptive mesh refinement," 2021. [Online]. Available: https://github.com/paralab/Dendro-GR

[24] M. S. Fernando and H. Sundar, "paralab/Dendro-5.01: Massively parallel adaptive octree based PDE solver," Jun. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3879315

[25] M. Fernando, D. Neilsen, H. Lim, E. Hirschmann, and H. Sundar, "Massively parallel simulations of binary black hole intermediate-mass-ratio inspirals," *SIAM Journal on Scientific Computing*, vol. 41, no. 2, pp. C97–C138, 2019.

[26] F. Foucart, P. Laguna, G. Lovelace, D. Radice, and H. Witek, "Snowmas2021 cosmic frontier white paper: Numerical relativity for next-generation gravitational-wave probes of fundamental physics," *arXiv preprint arXiv:2203.08139*, 2022.

[27] F. Löffler, J. Faber, E. Bentivegna, T. Bode, P. Diener, R. Haas, I. Hinder, B. C. Mundim, C. D. Ott, E. Schnetter *et al.*, "The einstein toolkit: a community computational infrastructure for relativistic astrophysics," *Classical and Quantum Gravity*, vol. 29, no. 11, p. 115001, 2012.

[28] E. Schnetter, S. H. Hawley, and I. Hawke, "Evolutions in 3d numerical relativity using fixed mesh refinement," *Classical and quantum gravity*, vol. 21, no. 6, p. 1465, 2004.

[29] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf, "The Cactus framework and toolkit: Design and applications," in *Vector and Parallel Processing – VECPAR'2002, 5th International Conference, Lecture Notes in Computer Science*. Berlin: Springer, 2003. [Online]. Available: http://edoc.mpg.de/3341

[30] Y. Zlochower, J. G. Baker, M. Campanelli, and C. O. Lousto, "Accurate black hole evolutions by fourth-order numerical relativity," *Physical Review D*, vol. 72, no. 2, p. 024021, 2005.

[31] B. Daszuta, F. Zappa, W. Cook, D. Radice, S. Bernuzzi, and V. Morozova, "Grathena++: puncture evolutions on vertex-centered oct-tree amr," *arXiv preprint arXiv:2101.08289*, 2021.

[32] L. E. Kidder, M. A. Scheel, S. A. Teukolsky, E. D. Carlson, and G. B. Cook, "Black hole evolution by spectral methods," *Physical Review D*, vol. 62, no. 8, p. 084032, 2000.

[33] I. Ruchlin, Z. B. Etienne, and T. W. Baumgarte, "SENR/NRPy+: Numerical Relativity in Singular Curvilinear Coordinate Systems," *Phys. Rev. D*, vol. 97, no. 6, p. 064036, 2018.

[34] P. Chang and Z. Etienne, "General relativistic hydrodynamics on a moving-mesh I: static space–times," *Mon. Not. Roy. Astron. Soc.*, vol. 496, no. 1, pp. 206–214, 2020.

[35] A. G. Lewis and H. P. Pfeiffer, "Gpu-accelerated simulations of isolated black holes," *Classical and Quantum Gravity*, vol. 35, no. 9, p. 095017, 2018.

[36] M. Fernando, D. Neilsen, E. W. Hirschmann, and H. Sundar, "A scalable framework for adaptive computational general relativity on heterogeneous clusters," in *Proceedings of the ACM International Conference on Supercomputing*, 2019, pp. 1–12.

[37] M. Alcubierre, *Introduction to 3+1 numerical relativity*, ser. International series of monographs on physics. Oxford: Oxford Univ. Press, 2008.

[38] T. W. Baumgarte and S. L. Shapiro, *Numerical relativity: starting from scratch*. Cambridge University Press, 2021.

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

[40] M. Alcubierre, *Introduction to 3+1 Numerical Relativity*. Oxford University Press, 2012.

[41] T. W. Baumgarte and S. L. Shapiro, *Numerical Relativity: Solving Einstein's Equations on the Computer*. Cambridge University Press, 2010.

[42] H.-O. Kreiss and J. Oliger, "Comparison of accurate methods for the integration of hyperbolic equations," *Tellus*, vol. 24, no. 3, pp. 199–215, 1972.

[43] R. Wald, *General Relativity*. University of Chicago Press, 1984.

[44] N. T. Bishop and L. Rezzolla, "Extraction of gravitational waves in numerical relativity," *Living Reviews in Relativity*, vol. 19, no. 1, Oct 2016. [Online]. Available: http://dx.doi.org/10.1007/s41114-016-0001-9

[45] V. I. Lebedev, "Spherical quadrature formulas exact to orders 25–29," *Siberian Mathematical Journal*, vol. 18, no. 1, pp. 99–107, 1977.

[46] T. Isaac, C. Burstedde, and O. Ghattas, "Low-cost parallel algorithms for 2: 1 octree balance," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium*. IEEE, 2012, pp. 426–437.

[47] H. Sundar, R. Sampath, and G. Biros, "Bottom-up construction and 2:1 balance refinement of linear octrees in parallel," *SIAM Journal on Scientific Computing*, vol. 30, no. 5, pp. 2675–2708, 2008.

[48] M. Fernando, D. Duplyakin, and H. Sundar, "Machine and application aware partitioning for adaptive mesh refinement applications," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, 2017, pp. 231–242.

[49] G. S. Boolos, J. P. Burgess, and R. C. Jeffrey, *Computability and logic*. Cambridge university press, 2002.

[50] S. Husa, I. Hinder, and C. Lechner, "Kranc: A Mathematica application to generate numerical codes for tensorial evolution equations," *Comput. Phys. Commun.*, vol. 174, pp. 983–1004, 2006.

[51] Y. Zlochower, J. G. Baker, M. Campanelli, and C. O. Lousto, "Accurate black hole evolutions by fourth-order numerical relativity," *Phys. Rev.*, vol. D72, p. 024021, 2005.

[52] W. R. Inc., "Mathematica, Version 13.0.0," champaign, IL, 2021. [Online]. Available: https://www.wolfram.com/mathematica

[53] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, Jan. 2017. [Online]. Available: https://doi.org/10.7717/peerj-cs.103

[54] I. Ruchlin, Z. B. Etienne, and T. W. Baumgarte, "Senr/nrpy+: Numerical relativity in singular curvilinear coordinate systems," *Physical Review D*, vol. 97, no. 6, p. 064036, 2018.

[55] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[56] D. Stanzione, J. West, R. T. Evans, T. Minyard, O. Ghattas, and D. K. Panda, "Frontera: The evolution of leadership computing at the National Science Foundation," in *Practice and Experience in Advanced Research Computing*. ACM, 2020.

Appendix: Artifact Description/Artifact Evaluation

SUMMARY OF THE EXPERIMENTS REPORTED

1 COMPILING DENDRO-GR GPU EXTENSION

The developed Dendro-GR GPU extension is available here. The following dependencies are required for the compilation. The core kernel generator is based on SymPyGR.

- C/C++ compilers with C++11 standards and OpenMP support
- CUDA compilation tools version 10.2
- Python3 SymPy, networkx for code generation.
- MPI implementation (e.g. openmpi, mvapich2)
- ZLib compression library
- BLAS , LAPACK, and GSL libraries.
- CMake 2.8 or higher version

To build the code use the following commands.

```
$cd <path to root source dir >
$ mkdir build
$ cd build
$ cmake ../
$ make all -j4
```

1.1 Using Singularity

The singularity container definition file is provided in the repository under the folder `container`. The following command can be used to build the Dendro-GR container which installs all the required dependencies and compile the Dendro-GR code.

```
sudo singularity build --sandbox dgr-cuda dgr.def
singularity run dgr-cuda dgr.def
```

The main Dendro-GR solver can be initiated by executing the following command.

```
singularity exec dgr-cuda \
sc22-dgr/build_gpu/BSSN_GR/./bssnSolverCtx\
        sc22-dgr/build_gpu/q1.par.json 1
```

More details on the running of the solvers are described in the §3

2 EXPERIMENTAL SETUP

The CPU/GPU performance comparisons were performed in TACC's Lonestar6 Cluster. We performed our experiments on Frontera and Lonestar 6 at the Texas Advanced Computing Center (TACC). Frontera has 8K Intel Cascade Lake nodes and Lonestar 6 has 16 dual-NVIDIA A100 nodes. All GPU-CPU comparisons were done on a single NVIDIA A100 GPU with full CPU node (i.e., 128 cores, 64 cores per socket) with AMD EPYC 7763 CPU. Frontera is used for performing a large scale weak scaling study.

• Lonestar6 Cluster module environment used is given below.

```
1) intel/19.1.1   2) impi/19.0.9   3) python3/3.9.7
4) cmake/3.21.3  5) pmix/3.2.3   6) xalt/2.10.32
7) TACC   8) cuda/11.4 (g)   9) gsl/2.7

Where:
g:  built for GPU
```

• Frontera Cluster module environment used is given below.

```
Currently Loaded Modules:
1) intel/19.1.1   2) impi/19.0.9   3) git/2.24.1
4) autotools/1.2   5) python3/3.7.0
6) pmix/3.1.4   7) hwloc/1.11.12   8) xalt/2.10.34
9) TACC  10) gsl/2.6  11) cmake/3.20.3
```

3 RUNNING EXPERIMENTS

The following executables are used in the paper.

- BSSN_GR/bssnSolverCUDA - GPU BSSN solver
- BSSN_GR/bssnSolverCtx - CPU BSSN solver
- BSSN_GR/tpid - two puncture initial condition solver.

The parameter files used to perform the runs can be found in `BSSN_GR/pars` folder. For each parameter file, first run `tpid` to solver the initial conditions followed by the `bssnSolverCUDA` or `bssnSolverCtx` for GPU and CPU versions respectively.

```
$ ./BSSN_GR/tpid q1.par.json <number of threads to use>
$ ibrun -np <number of GPUs> \
     ./BSSN_GR/bssnSolverCUDA q1.par.json 1
```

- `BSSN_GR/pars/q1.par.json` : q=1 binary black hole merger
- `BSSN_GR/pars/q2.par.json` : q=2 binary black hole merger
- `BSSN_GR/pars/q4.par.json` : q=4 binary black hole merger

AUTHOR-CREATED OR MODIFIED ARTIFACTS:

Artifact 1

Persistent ID: `https://github.com/paralab/Dendro-GR`
Artifact name: Dendro-GR
Citation of artifact: Open source Dendro-GR CPU code

Artifact 2

Persistent ID: `https://zenodo.org/record/6618080`
Artifact name: Numerical relativity on GPUs, Dendro-GR GPU extension
Citation of artifact: Milinda Shayamal Fernando. (2022). paralab/sc22-dgr: sc22 paper artifact description code (v1.0). Zenodo. https://doi.org/10.5281/zenodo.6618080

Artifact 3

Persistent ID: `https://github.com/paralab/sc22-dgr/releases/tag/v1.0`
Artifact name: Numerical relativity on GPUs, Dendro-GR GPU extension

Artifact 4

Persistent ID: `https://github.com/paralab/sc22-dgr/blob/main/container/dgr.def`
Artifact name: Singularity definition file to build Dendro-GR container

Reproduction of the artifact with container: The produced GPU results we performed in TACC's Lonestar6 cluster

(https://portal.tacc.utexas.edu/user-guides/lonestar6) with the following module environment with NVIDIA A100 GPUs. The presented gravitational waves were computed using the Lonestar6 GPU cluster with the same module environment.

Currently Loaded Modules: 1) intel/19.1.1 2) impi/19.0.9 3) python3/3.9.7 4) cmake/3.21.3 5) pmix/3.2.3 6) xalt/2.10.32 7) TACC 8) cuda/11.4 (g) 9) gsl/2.7

Where: g: built for GPU

The presented weak scalability study was performed, with TACC's Frontera supercomputer (https://www.tacc.utexas.edu/systems/frontera) with the following module environment.

Currently Loaded Modules: 1) intel/19.1.1 2) impi/19.0.9 3) git/2.24.1 4) autotools/1.2 5) python3/3.7.0 6) pmix/3.1.4 7) hwloc/1.11.12 8) xalt/2.10.34 9) TACC 10) gsl/2.6 11) cmake/3.20.3