DENDRO-GR: Massively Parallel Simulations of Binary Black Hole Intermediate-Mass-Ratio Inspirals

Milinda Fernando *(student)*, Hari Sundar *(advisor)*
University of Utah

David Neilsen, Hyun Lim, Eric Hirschmann
Brigham Young University

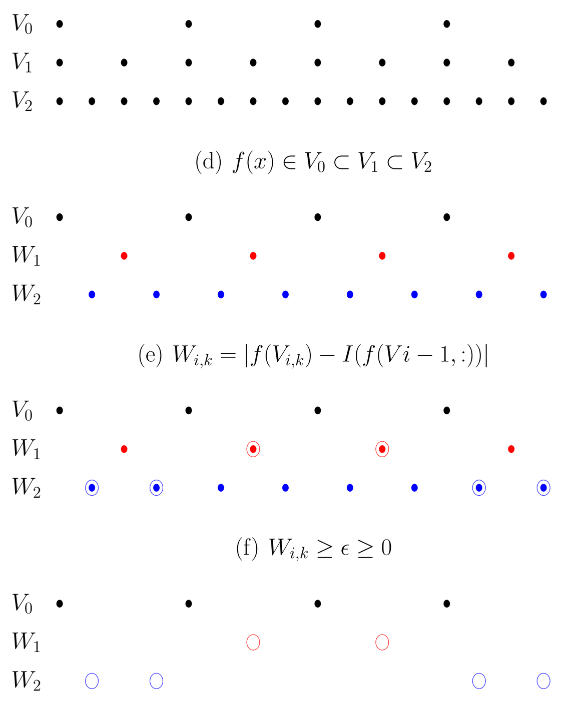
Motivation

The recent detection of Gravitational Waves (GW) has revolutionized multi-messenger astronomy and produced several exciting discoveries. Current GW observations have been limited to observations of binary mergers of roughly equal mass. This has to a large part been due to the inability of existing computational relativity codes to simulate Intermediate Mass Ratio Inspirals (IMRIs) with a mass-ratio (q) greater than 10. In this work we present a new highly adaptive & scalable computational relativity framework, DENDRO-GR, that is able to simulate IMRIs with mass ratio as high as $q = 100$.

Contributions

- **Wavelet Adaptive GR**. This is the first highly adaptive computational fully relativistic—i.e., including the full Einstein equations—code with an arbitrary localized unstructured wavelet based adaptive mesh.
- **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 which helps to improve code portability.
- **Performance**. We developed a new parallel search algorithm, TREESEARCH to improve the efficiency of octree meshing. We also developed efficient `unzip` and `zip` operations to allow the application of the computational kernels on small process-local regular blocks which leads to greater performance and code portability (i.e. GPU architectures).
- **Simulations**. We demonstrate the ability to scale to large mass ratios, enabling simulations and extraction of gravitational waves for mass ratios as high as 100.
- **Implementation** DENDRO is implemented in C++ using MPI except for the automatic code generation framework which is implemented using SymPy. Our code is freely available at `https://github.com/paralab/Dendro-GR` under the MIT license.

Wavelet based adaptivity


(d) $f(x) \in V_0 \subset V_1 \subset V_2$

(e) $W_{i,k} = |f(V_{i,k}) - I(f(Vi-1,:))|$

(f) $W_{i,k} \geq \epsilon \geq 0$

(g) wavelet/sparse representation of $f(x)$

Figure 1: For a given function $f : V \rightarrow \mathcal{R}$ let $V_i \subset V$ be the finite dimensional approximation of f (see figure 1d). As number of nodes increases (i.e. going from V_i to V_{i+1}) for each additional node introduced, we compute wavelet coefficients based on the absolute difference between $f(V_{i,k})$ and interpolated value from previous level $f(V_{i-1,:})$ (see figure 1e). In figure 1f shows the chosen nodes that violate specified wavelet tolerance *epsilon* and these nodal wavelets are stored as the sparse/wavelet representation of the function f (see figure 1g).

Example octrees generated based on Wavelet Adaptive Multi-Resolution (WAMR) for binary compact mergers.

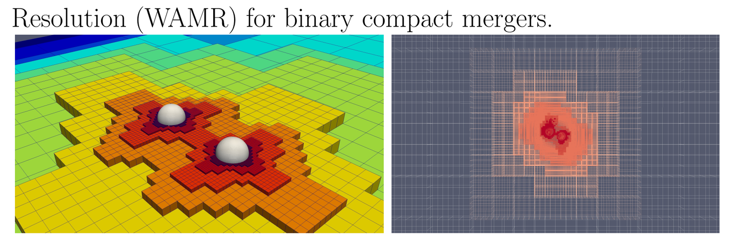


Figure 2: **(left)** A example of the adaptive mesh created by DENDRO for the binary black-hole system. (**right**) the hierarchical wavelet grids generated for the binary black hole system.

Methodology

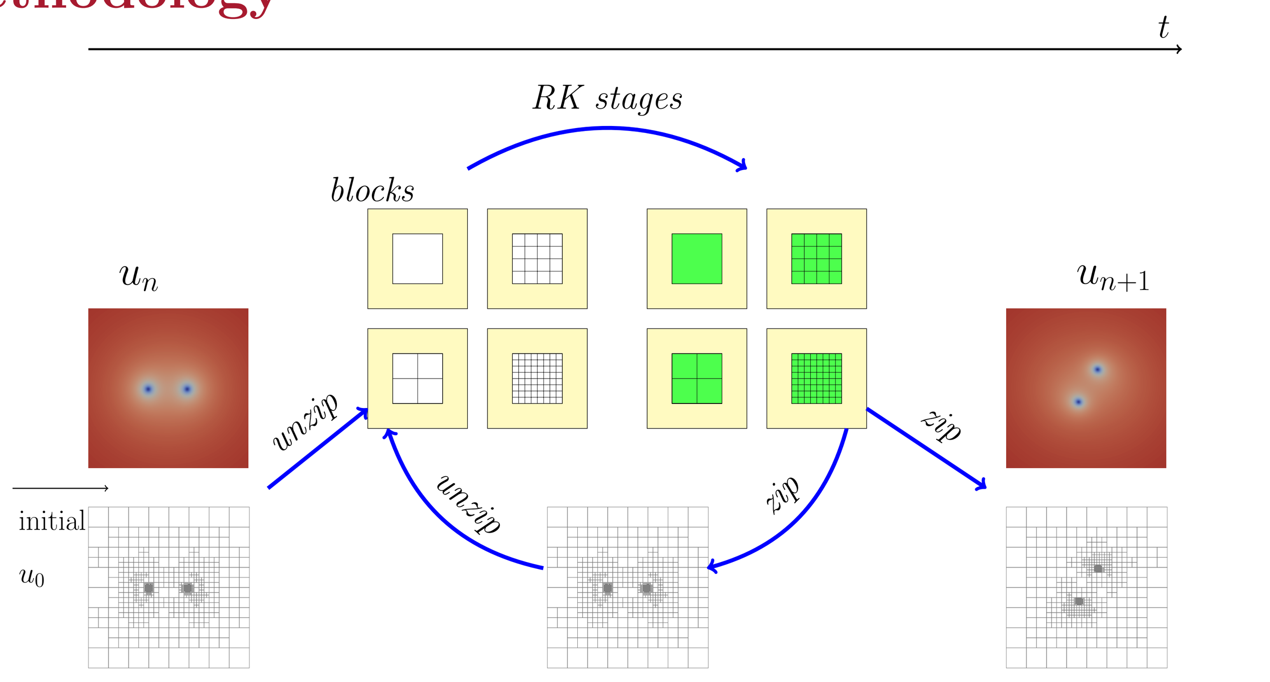


Figure 3: This figure illustrates the calculation of a single Runge-Kutta (RK) time step, computing the solution at the advanced time, u_{n+1}, from data at the previous time step, u_n. For computational efficiency, spatial and time derivatives are evaluated on equispaced blocks *(unzipped)*; a sparse 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. Note that the re-meshing is performed as needed based on the wavelet expansion of the current solution.

TreeSearch

It is required to perform search operations in an adaptive octrees, in order to build octant to octant ($O2O$) and octant to nodal ($O2N$) neighborhood data structures.

Traditional	TREESEARCH
> + binary search operations $\mathcal{O}(n \log n)$ random memory accesses	comparison free + single pass $\mathcal{O}(nD)$, $D \leq \log(n)$ structured memory accesses

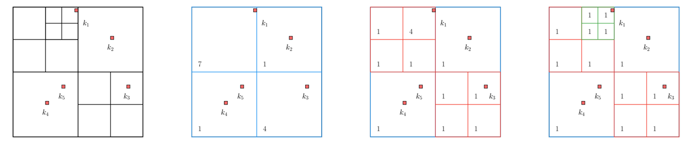


Figure 4: TREESEARCH performs the traversal in a top-down order over the set of keys, while flagging k_2, k_4, k_5 at the level 1 split, k_3 at level 2 split, and k_1 at level 3 split.

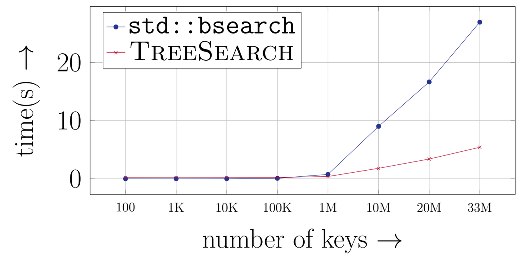


Figure 5: Comparison of `std::bsearch` with operator < and comparison free TREESEARCH approach for performing, varying number of keys on $33M$ sorted octree in Stampede2 `SKX` node.

Zip & Unzip: FD computations on adaptive octrees

Unzip : process of generating regular grid blocks from adaptive octree with required padding region for stencil computations.

Zip : inverse process of unzip to *shared octant nodes* representation.

Note: All the communication happens in the *zip* states since *unzip* contains duplicate nodes.

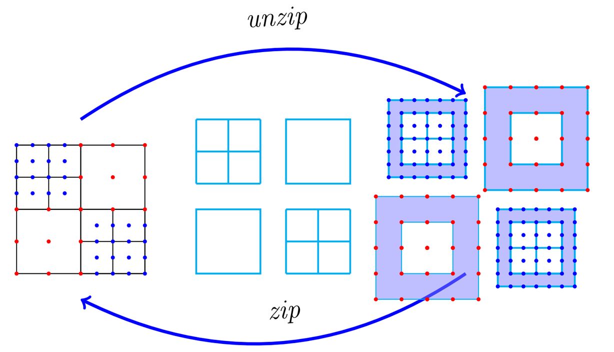


Figure 6: A simplistic example of octree to block decomposition and *unzip* operation. The leftmost figure shows the considering adaptive octree with *shared octant nodes* and its block decomposition is shown in the middle. Note that the given octree is decomposed into four regular blocks of different sizes. The rightmost figure shows the decomposed blocks padded with values coming from neighboring octants with interpolation if needed. In order to perform *unzip* operation both ψ and ψ mappings are used.

Symbolic code generation for BSSNKO equations

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

```
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]
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))
```

The left panel shows the BSSNKO 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.

Simple example: Non-linear sigma model

$$\partial_t \phi = \Delta\chi - \frac{\sin(2\chi)}{\|x\|_2{}^2}$$
$$\partial_t \chi = \phi$$

```
from DENDRO_sym import *
phi_rhs = sum(d2(i,i,chi)for i in dendro.e_i)
                - sin(2*chi)/r**2
chi_rhs = phi
```

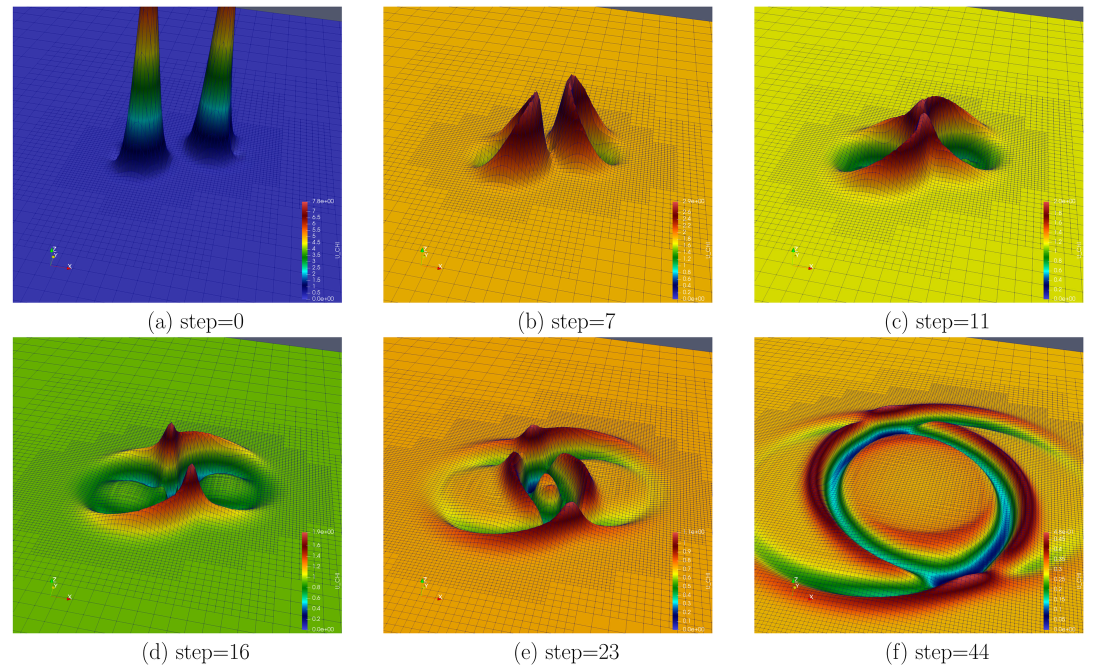
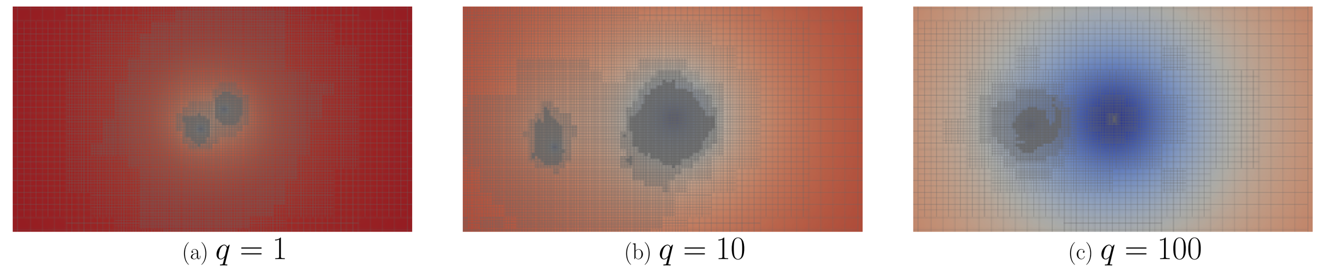

(a) step=0 (b) step=7 (c) step=11 (d) step=16 (e) step=23 (f) step=44

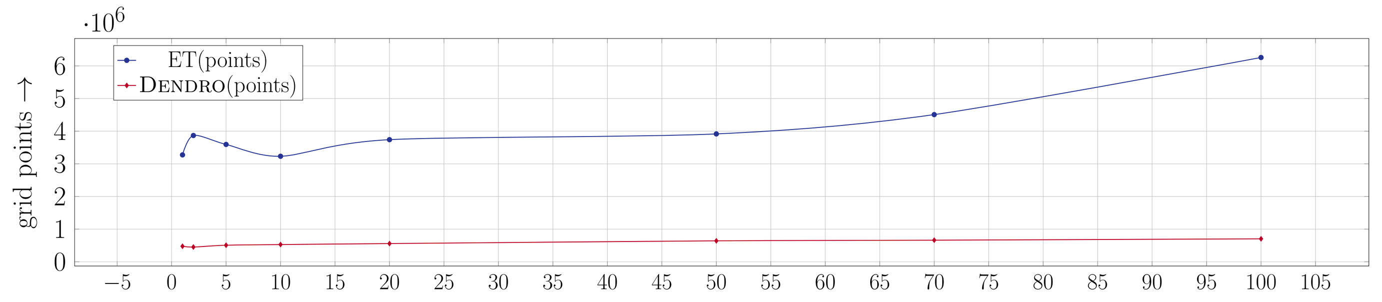
Figure 7: Time step snapshots of a z plane slice of $3D$ NLSM with two initial Gaussian distributions with Runge-Kutta time stepping scheme. Note that how the mesh refines (based on the WAMR) as the waves propagate away from the origin.

Scalability & Adaptivity

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. In $3d$ the number of points required to resolve the smallest black hole grows as q^3.


(a) $q = 1$ (b) $q = 10$ (c) $q = 100$

WAMR vs block/structured adaptivity



Comparison between ET and DENDRO-GR for number of spatial points with increasing mass ratios. Note that these are not from complete simulations, but it illustrates the rate of increase for both approaches.

Binary compact mergers of different mass ratios

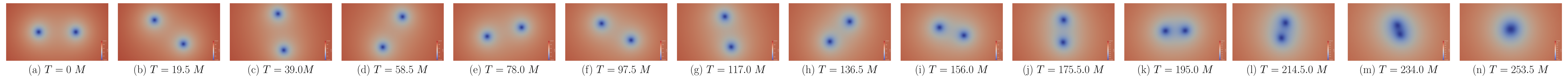

(a) $T = 0\ M$ (b) $T = 19.5\ M$ (c) $T = 39.0M$ (d) $T = 58.5\ M$ (e) $T = 78.0\ M$ (f) $T = 97.5\ M$ (g) $T = 117.0\ M$ (h) $T = 136.5\ M$ (i) $T = 156.0\ M$ (j) $T = 175.5.0\ M$ (k) $T = 195.0\ M$ (l) $T = 214.5.0\ M$ (m) $T = 234.0\ M$ (n) $T = 253.5\ M$

Figure 8: Slice of the BSSNKO variable χ along the z-plane for equal mass ratio binary black holes. Above simulation performed in TACC's Stampede2 using $1K$ cores with total simulation run time of $48hrs$ from the beginning to the merger event.

More detailed text available at →

More simulation videos & images available at →

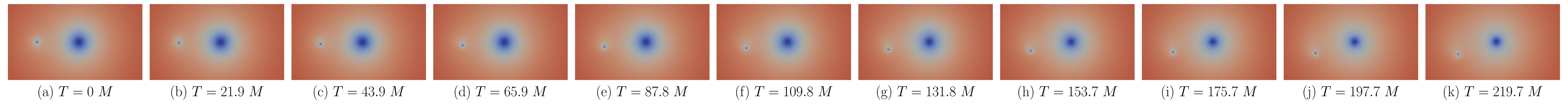

(a) $T = 0\ M$ (b) $T = 21.9\ M$ (c) $T = 43.9\ M$ (d) $T = 65.9\ M$ (e) $T = 87.8\ M$ (f) $T = 109.8\ M$ (g) $T = 131.8\ M$ (h) $T = 153.7\ M$ (i) $T = 175.7\ M$ (j) $T = 197.7\ M$ (k) $T = 219.7\ M$

Figure 9: Slice of the BSSNKO variable χ along the z-plane for mass ratio $q = 10$ binary black holes.

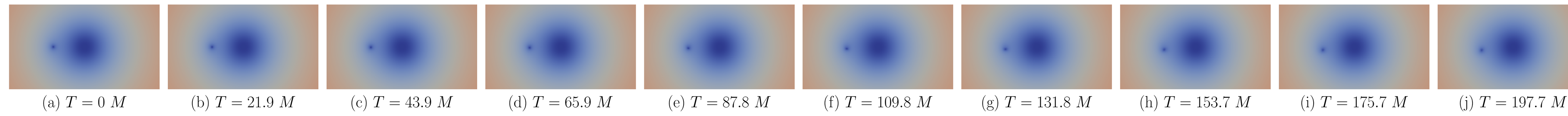
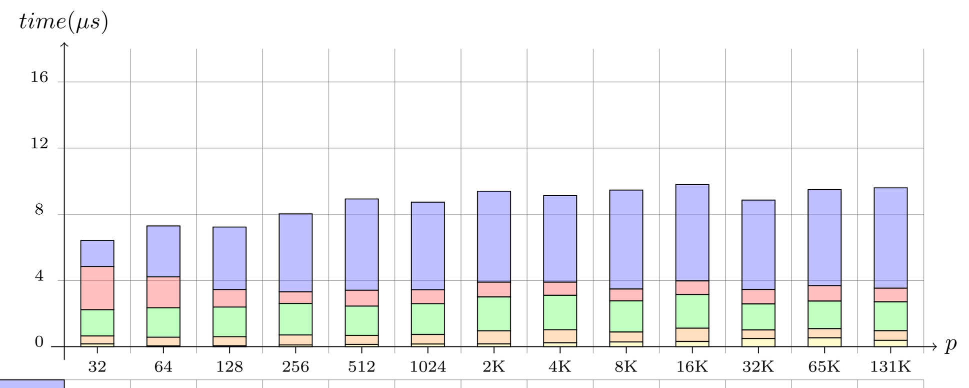

(a) $T = 0\ M$ (b) $T = 21.9\ M$ (c) $T = 43.9\ M$ (d) $T = 65.9\ M$ (e) $T = 87.8\ M$ (f) $T = 109.8\ M$ (g) $T = 131.8\ M$ (h) $T = 153.7\ M$ (i) $T = 175.7\ M$ (j) $T = 197.7\ M$

Figure 10: Slice of the BSSNKO variable χ along the z-plane for mass ratio $q = 100$ binary black holes.

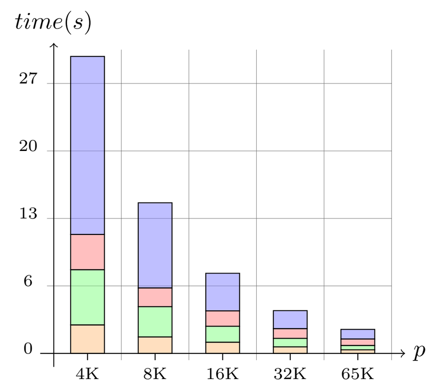
Results



p	32	64	128	256	512	1024	2K	4K	8K	16K	32K	65K	131K
communication	1.65	3.21	3.94	4.91	5.75	5.52	5.73	5.46	6.23	6.08	5.63	6.05	6.32
unzip	2.72	1.95	1.10	0.73	1.00	0.88	0.93	0.83	0.75	0.86	0.91	0.97	0.86
rhs	1.65	1.85	1.87	1.99	1.85	1.94	2.14	2.17	1.96	2.12	1.64	1.74	1.82
derivatives	0.50	0.55	0.57	0.63	0.57	0.60	0.82	0.82	0.63	0.84	0.54	0.58	0.61
wavelets	0.18	0.05	0.06	0.11	0.14	0.17	0.18	0.25	0.30	0.33	0.52	0.56	0.40
Total time	6.7	7.6	7.5	8.4	9.3	9.1	9.8	9.5	9.9	10.2	9.2	9.9	10.0
Total dofs (zipped)	48.6M	92.9M	187.7M	342.1M	694.0M	1.4B	2.8B	5.6B	11.0B	21.7B	46.4B	93.9B	205.8B

Weak scaling results

Weak scaling results in ORNL's Titan for $RK/(dof/p)$ (averaged over 10 steps) where RK, dof, p denotes the time for single RK step, degrees of freedom, and number of cores respectively, with derivative computation (`deriv`), right hand side(`rhs`) computation, `unzip` cost, wavelet computation(`wavelets`) and communication cost (`comm`) with the approximate grain size of 1.536M unknowns where the number of cores ranging from 32 to 131,072 cores on 8192 nodes where the largest problem having 206 Billion unknowns.



p	4K	8K	16K	32K	65K
communication	18.31	8.76	3.86	1.85	0.99
unzip	3.62	1.92	1.59	1.00	0.66
rhs	5.68	3.12	1.64	0.88	0.45
derivatives	2.93	1.69	1.15	0.67	0.37
Total time (s)	30.5	15.5	8.2	4.4	2.5

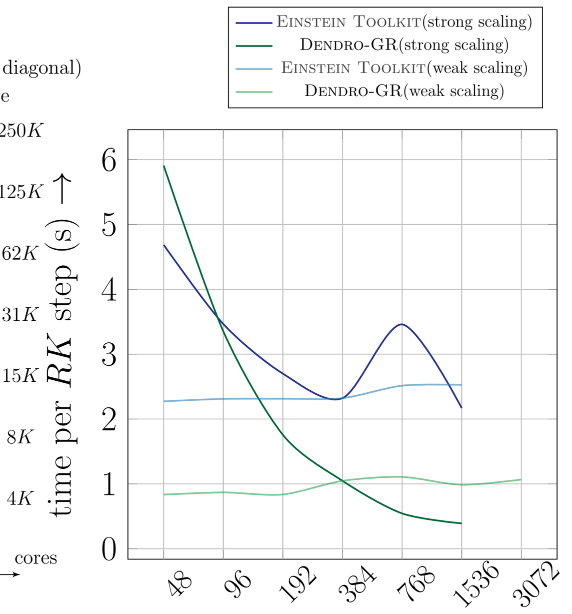
Strong scaling results

Strong scaling results in ORNL's Titan for a single RK step (averaged over 10 steps) with derivative computation (`deriv`), right hand side(`rhs`) computation, `unzip` cost and 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.

DENDRO-GR

dofs (along the diagonal) per core

total dofs ↑ \ cores →	48	96	192	384	768	1536	3072	dofs per core
$768M$	35.46	–	12.34	6.88	4.01	2.06	1.06	$250K$
$384M$	19.22	11.08	6.22	3.20	1.64	0.98	0.56	$125K$
$192M$	11.23	6.59	3.67	1.92	1.10	0.69	0.38	$62K$
$96M$	5.90	3.35	1.75	1.04	0.54	0.39	–	$31K$
$48M$	2.98	1.57	0.83	0.61	0.38	–	–	$15K$
$24M$	1.58	0.86	0.51	0.35	–	–	–	$8K$
$12M$	0.83	0.55	0.37	–	–	–	–	$4K$



EINSTEIN TOOLKIT

dofs (along the diagonal) per core

total dofs ↑ \ cores →	48	96	192	384	768	1536	3072	dofs per core
$768M$	23.38	12.65	7.64	4.93	3.71	2.95	–	$250K$
$384M$	12.44	7.51	4.87	3.48	2.92	2.52	–	$125K$
$192M$	7.13	4.76	3.45	2.74	2.51	2.31	–	$62K$
$96M$	4.68	3.46	2.69	2.32	3.45	2.16	–	$31K$
$48M$	3.36	2.68	2.31	2.08	2.12	2.08	–	$15K$
$24M$	2.67	2.31	2.08	1.94	2.07	2.04	–	$8K$
$12M$	2.27	2.06	1.93	1.88	2.03	2.07	–	$4K$

Comparison with Einstein Toolkit

Comparison between EINSTEIN TOOLKIT and DENDRO-GR without factoring in adaptivity (i.e. both EINSTEIN TOOLKIT & DENDRO-GR support uniform grids). For a fixed tolerance, we expand the domain for a 1 : 1 mass-ratio simulation such that both EINSTEIN TOOLKIT and DENDRO-GR have roughly the same number of dofs. We present both weak and strong scaling results using both codes. On the left table are results from DENDRO-GR and from EINSTEIN TOOLKIT on the right. In the middle, we plot a representative strong and weak scaling curve for each code. The DENDRO-GR scaling is plotted in green (lighter shade for weak) and blue for EINSTEIN TOOLKIT. The corresponding data entries are also marked in the tables. Note that the rows represent strong scaling and the diagonal entries represent weak scaling results and runtime is reported in seconds(s).