GRaM-X: a new GPU-accelerated dynamical spacetime GRMHD code for Exascale computing with the Einstein Toolkit

OPEN ACCESS
IOP Publishing
Classical and Quantum Gravity
Class. Quantum Grav. **40** (2023) 205009 (33pp)
https://doi.org/10.1088/1361-6382/acf2d9

GRaM-X: a new GPU-accelerated dynamical spacetime GRMHD code for Exascale computing with the Einstein Toolkit

Swapnil Shankar[1,*], Philipp Mösta[2,*], Steven R Brandt[3], Roland Haas[4,5], Erik Schnetter[3,6,7] and Yannick de Graaf[8]

[1] Anton Pannekoek Institute for Astronomy and GRAPPA, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands
[2] GRAPPA, Anton Pannekoek Institute for Astronomy and Institute of High-Energy Physics, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands
[3] Center for Computation & Technology, Louisiana State University, Baton Rouge, LA, United States of America
[4] National Center for Supercomputing applications, University of Illinois, 1205 W Clark St, Urbana, IL, United States of America
[5] Department of Physics, University of Illinois, 1110 West Green St, Urbana, IL, United States of America
[6] Perimeter Institute for Theoretical Physics, Waterloo, Ontario, Canada
[7] Department of Physics and Astronomy, University of Waterloo, Waterloo, Ontario, Canada
[8] University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands

E-mail: swapnilshankar1729@gmail.com and p.moesta@uva.nl

Received 13 December 2022; revised 14 July 2023
Accepted for publication 22 August 2023
Published 19 September 2023

Abstract

We present GRaM-X (**G**eneral **R**elativistic **a**ccelerated **M**agnetohydrodynamics on AMRe**X**), a new GPU-accelerated dynamical-spacetime general relativistic magnetohydrodynamics (GRMHD) code which extends the GRMHD capability of Einstein Toolkit to GPU-based exascale systems. GRaM-X supports 3D adaptive mesh refinement (AMR) on GPUs via a new AMR driver for the Einstein Toolkit called CarpetX which in turn leverages AMReX, an AMR

* Authors to whom any correspondence should be addressed.

library developed for use by the United States DOE's Exascale Computing Project. We use the Z4c formalism to evolve the Einstein equations and the Valencia formulation to evolve the equations of GRMHD. `GRaM-X` supports both analytic as well as tabulated equations of state. We implement TVD and WENO reconstruction methods as well as the HLLE Riemann solver. We test the accuracy of the code using a range of tests on static spacetime, e.g. 1D magnetohydrodynamics shocktubes, the 2D magnetic rotor and a cylindrical explosion, as well as on dynamical spacetimes, i.e. the oscillations of a 3D Tolman-Oppenheimer-Volkhof star. We find excellent agreement with analytic results and results of other codes reported in literature. We also perform scaling tests and find that `GRaM-X` shows a weak scaling efficiency of ∼40%–50% on 2304 nodes (13824 NVIDIA V100 GPUs) with respect to single-node performance on OLCF's supercomputer Summit.

Keywords: magnetohydrodynamics, general relativity, exascale computing, GPUs

(Some figures may appear in colour only in the online journal)

1. Introduction

In the last decade dynamical-spacetime general-relativistic magnetohydrodynamics (GRMHD) codes have developed into robust tools to perform production simulations of astrophysical systems. They are routinely applied to predict gravitational waves from compact-object mergers, the amount and composition of ejected material, and to determine the remnant object left behind, e.g. [1–10]. The outputs from these simulations are often used to identify the multimessenger signatures of these events in multi-stage pipelines [11, 12]. In the supernova context, GRMHD simulations have matured significantly over the last decade. Multiple groups have led proof-of-concept studies demonstrating the importance of inclusion of magnetic fields in rapidly-rotating progenitors [13–17] and are beginning to study the impact of magnetic fields in neutrino-driven supernova [18]. Most of these codes employ the Valencia formulation of the ideal magnetohydrodynamics (MHD) equations [19] and solve them numerically via finite-volume methods while solving Einstein's equations via finite-differences or spectral methods. Much work in the last decade has gone into performing high-resolution simulations [3, 20, 21] and adding more realistic microphysics via tabulated equations of state and better neutrino transport approximation schemes [10, 16, 18]. Many of these results have been enabled by making these multi-physics simulations run effectively in massively-parallel environments as found on modern high-performance computing systems.

The current challenge is to run these multi-physics simulations on modern high-performance compute systems that often contain the majority of their compute power in graphical processing units (GPUs). Standard scientific coding practices targeting employment on central processing units (CPUs) do not work effectively on GPUs due to the drastic hardware differences. Common bottlenecks that have to be taken into account are the significantly simpler control logic of GPUs, register number and capacity, memory layout and bandwidth, as well as transfer of data from CPU to GPU. Different strategies exist for programming for GPUs, namely via direct `CUDA` implementation, `OpenMP`, or secondary libraries like `Kokkos`. A good number of GRMHD codes that use static spacetime backgrounds have been successfully ported/redesigned to run effectively on GPUs [22, 23]. Dynamical spacetime GRMHD

codes however present a bigger porting challenge due to the order-of-magnitude larger memory footprint.

Here we present GRaM-X, a dynamical-spacetime GRMHD code developed for the Einstein toolkit that runs efficiently on GPUs. GRaM-X follows the Valencia formulation of GRMHD and can utilize analytic and tabulated equations of state. It is built on the Cactus computational framework and the CarpetX mesh-refinement driver which utilizes AMReX. It currently utilizes a spacetime solver using the Z4c formulation of the Einstein equations. GRaM-X can be run on both CPUs and GPUs and uses AMReX GPU kernel launches via lambda functions. We have tested GRaM-X with a suite of standard GRMHD test problems and demonstrate its performance on GPU supercomputers like ORNL's Summit on up to $\sim$2300 nodes.

This paper is structured as follows. In section 2 we briefly present the equations solved in the code and the Valencia formulation of GRMHD. We describe the numerical techniques and implementation details in section 3 and present a set of code verification and sensitivity tests in section 4. We conclude by discussing the performance of the code in section 5 and by summarizing and discussing future directions in section 6.

2. GRMHD/Valencia formulation and numerical methods

GRaM-X is a dynamical spacetime GRMHD code which means we evolve the equations of General Relativity (GR) as well as the equations of MHD coupled together. We use the Z4c formalism [24] to solve the Einstein equations and the Valencia formulation to evolve the equations of relativistic ideal MHD. In the ideal MHD approximation, the fluid is assumed to have infinite conductivity and there is no charge separation.

2.1. Z4c formulation of the Einstein equations

The Einstein equations are a system of ten coupled second-order partial differential equations in the four-metric $g_{\mu\nu}$. We formulate the Einstein equations in the Z4c formulation [24, 25]. This formulation is similar to the well-known BSSN formulation [26–29]. It introduces extra dynamical fields that lead to a well-posed formulation and which dynamically dampens (makes decay) the constraints of the Einstein equations. The gauge conditions associated with this formulation are the $1+\log$ foliation and Γ-driver shift. We use standard gauge and constraint damping parameters for our calculations which include constraint damping parameters $\kappa_1 = 0.02$ and $\kappa_2 = 0.0$, as well as lapse parameter $\mu_L = 2/\alpha$ and shift parameters $\mu_S = 1$ and $\eta = 2$ [30].

As usual in the Einstein Toolkit, the Z4c state vector is not exposed to other thorns in Cactus. Instead, other thorns are written in terms of the standard ADM variables: the 3-metric γ_{ij}, the extrinsic curvature K_{ij}, lapse α, shift β^i, the time derivative of the lapse $A = \partial_t \alpha$, and the time derivative of the shift $B^i = \partial_t \beta^i$.

2.2. Valencia formulation

The equations of ideal GRMHD used in GRaM-X are obtained from the conservation of mass, energy-momentum and lepton number as well as from the Maxwell's equations:

$$\nabla_\mu(\rho u^\mu) = 0\,, \qquad \nabla_\mu(\rho Y_e u^\mu) = 0\,, \qquad \nabla_\mu T^{\mu\nu} = 0\,, \qquad \nabla_\nu \star F^{\mu\nu} = 0 \quad (1)$$

where ∇_μ denotes the covariant derivative with respect to the 4-metric, $\rho u^\mu = J^\mu$ is the mass current, Y_e is the electron fraction and $\star F^{\mu\nu}$ is the dual of the relativistic Faraday tensor $F^{\mu\nu}$.

In the ideal MHD approximation, electric fields vanish in the rest frame of the fluid which leads to the condition:

$$E_\nu = u_\mu F^{\mu\nu} = 0\,. \tag{2}$$

The equations of MHD are coupled to the Einstein equations via the stress-energy tensor $T^{\mu\nu}$. The stress-energy tensor has both the hydrodynamic contribution $T^{\mu\nu}_{\rm H}$ and the electromagnetic contribution $T^{\mu\nu}_{\rm EM}$ given by:

$$T^{\mu\nu}_{\rm H} = \rho h u^\mu u^\nu + P g^{\mu\nu} = (\rho + \rho\epsilon + P)\, u^\mu u^\nu + P g^{\mu\nu} \tag{3}$$

$$T^{\mu\nu}_{\rm EM} = F^{\mu\lambda} F^{\nu}{}_{\lambda} - \frac{1}{4} g^{\mu\nu} F^{\lambda\kappa} F_{\lambda\kappa} = b^2 u^\mu u^\nu - b^\mu b^\nu + \frac{b^2}{2} g^{\mu\nu} \tag{4}$$

where ρ, ϵ, P, u^μ, and $h \equiv 1 + \epsilon + P/\rho$ are the fluid rest mass density, specific internal energy, gas pressure, 4-velocity, and specific enthalpy, respectively, and $b^\mu = u_\nu \star F^{\mu\nu}$ is the magnetic 4-vector (the projected component of the Maxwell tensor parallel to the 4-velocity of the fluid). The combined stress-energy tensor is given by:

$$\begin{aligned} T^{\mu\nu} &= \left(\rho + \rho\epsilon + P + b^2\right) u^\mu u^\nu + \left(P + \frac{b^2}{2}\right) g^{\mu\nu} - b^\mu b^\nu \\ &\equiv \rho h^* u^\mu u^\nu + P^* g^{\mu\nu} - b^\mu b^\nu, \end{aligned} \tag{5}$$

where $P^* = P + b^2/2$ is the fluid pressure combined with magnetic pressure, and $h^* \equiv 1 + \epsilon + \left(P + b^2\right)/\rho$. In `GRaM-X`, fluid variables are cell-centered variables stored as cell averages while the spacetime variables and the stress-energy tensor are vertex-centered variables stored as samples at the cell vertices. $T^{\mu\nu}$ is vertex-centered because it is given as input to the spacetime solver whose discretization is vertex centered. Therefore, in order to calculate $T^{\mu\nu}$ in equation (5), we need all the variables at cell vertices. Hence, we perform a 4th-order symmetric 3D interpolation [31] from cell center values to calculate the fluid variables at cell vertices.

The evolution equations of ideal relativistic MHD that we use in `GRaM-X` are written in a first-order hyperbolic flux-conservative form for the conserved variables D, S^i, τ, and $\mathscr{B}^i$. The conserved variables are related to the primitive variables ρ, ϵ, v^i, and B^i as

$$D = \sqrt{\gamma}\rho W\,, \tag{6}$$

$$S_j = \sqrt{\gamma}\left(\rho h^* W^2 v_j - \alpha b^0 b_j\right)\,, \tag{7}$$

$$\tau = \sqrt{\gamma}\left(\rho h^* W^2 - P^* - (\alpha b^0)^2\right) - D\,, \tag{8}$$

$$\mathscr{B}^k = \sqrt{\gamma} B^k\,, \tag{9}$$

where $B^i = n_\nu \star F^{i\nu}$ is the spatial magnetic field in the spacelike slice with unit normal n^μ, γ is the determinant of γ_{ij}, $W \equiv (1 - v^i v_i)^{-1/2}$ is the Lorentz factor and v^i is the 3-velocity defined as

$$v^i = \frac{u^i}{W} + \frac{\beta^i}{\alpha}\,. \tag{10}$$

The MHD evolution equations, also known as Valencia formulation, are

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}^i}{\partial x^i} = \mathbf{S}\,, \tag{11}$$

with

$$\mathbf{U} = [D, S_j, \tau, \mathscr{B}^k, DY_e]^{\mathrm{T}},$$

$$\mathbf{F}^i = \alpha \times \begin{bmatrix} D\tilde{v}^i \\ S_j\tilde{v}^i + \sqrt{\gamma}P^*\delta^i_j - b_j\mathscr{B}^i/W \\ \tau\tilde{v}^i + \sqrt{\gamma}P^*v^i - \alpha b^0\mathscr{B}^i/W \\ \mathscr{B}^k\tilde{v}^i - \mathscr{B}^i\tilde{v}^k \\ DY_e\tilde{v}^i \end{bmatrix}, \tag{12}$$

$$\mathbf{S} = \alpha\sqrt{\gamma} \times \begin{bmatrix} 0, \\ T^{\mu\nu}\left(\frac{\partial g_{\nu j}}{\partial x^\mu} - \Gamma^\lambda_{\mu\nu}g_{\lambda j}\right) \\ \alpha\left(T^{\mu 0}\frac{\partial \ln\alpha}{\partial x^\mu} - T^{\mu\nu}\Gamma^0_{\mu\nu}\right) \\ \vec{0} \\ 0 \end{bmatrix}. \tag{13}$$

Here, $\tilde{v}^i = v^i - \beta^i/\alpha$ and $\Gamma^\lambda_{\mu\nu}$ are the 4-Christoffel symbols.

2.3. Numerical methods

As described above, the Valencia formulation for GRMHD consists of a set of 8 coupled hyperbolic partial differential equations for the 8-element state vector **U**. The state vector contains only the so-called *conserved* variables. The so-called *primitive* variables need to be calculated from the conserved variables before the fluxes **F** can be calculated. The primitive variable vector is $\mathbf{P} = [\rho, v^i, \epsilon, B^i, Y_e, T]$, where Y_e is the electron fraction as defined in equation (1) and T is the temperature.

Our initialization scheme proceeds as follows:

(1) Initial conditions are set up in terms of the primitive variables **P** (and the ADM variables for the spacetime metric).
(2) From these, the conserved variables **U** are calculated. This step is straightforward.

Our evolution scheme is based on the *Method of Lines*, allowing us to use a common time integration mechanism for all evolved variables, i.e. for both spacetime and hydrodynamics quantities. In the method of lines, one needs to provide a so-called right hand side (RHS) function that calculates $\partial_t\mathbf{U}$ from a given **U**. The time integrator evaluates the RHS function multiple times to step from a solution at time t to a solution at time $t + \Delta t$.

According to the Courant-Friedrichs-Lewy (CFL) condition [32], the timestep Δt has an upper limit for the stable evolution of the system, given by: $\Delta t \leqslant C\Delta x/u_c$, where C is the CFL factor and u_c is the maximum characteristic speed of the system. The characteristic speeds for the spacetime evolution are generally larger than MHD and can usually not be bounded ahead of time, because they depend on the gauge parameters (see for example [33] for BSSN characteristic speeds). Hence, we do not choose the timestep based on the characteristic speeds of the MHD equations and always assume the maximum characteristic speed to be equal to $\sqrt{2}$ times the speed of light, and express this in our coordinate frame, which leads to the most restrictive timestep for stable evolution.

Evaluating the RHS proceeds as follows, starting from the state vector **U** and ending with its time derivative $\partial_t\mathbf{U}$:

(1) Find the primitive variable vector **P** from the conserved variable vector **U**. This step is non-trivial because it requires finding the root of a multi-dimensional nonlinear equation

for each grid cell. The cells are not coupled, so that this expensive step can be easily parallelized. This step also requires evaluating the EoS (equation of state—see below), which might require interpolation in a nuclear EoS table. This is the most complex step of our scheme, and it can fail in several ways. The scheme can fail to converge or converge to unphysical values such as negative density, negative velocity, velocity greater than the speed of light, or negative pressure. Moreover, even if it converges to physical values, the obtained or intermediate values at any step can go out of bounds from tabulated values of ρ, T, Y_e or any other dependent quantity in the table.

(2) The primitive variables are now known at the cell centers. We reconstruct their values on the cell faces using either a TVD (total variation diminishing) [34] or a WENO5 (5th order weighted-ENO) [35] scheme. This step also requires the spacetime metric, which is known at the cell vertices and is interpolated linearly to the face centers. It is possible to reconstruct either primitive or conserved variables, but we choose to reconstruct primitive variables because they allow us to guarantee physically valid results such as positive pressure and velocities less than the speed of light very easily. Also, our previous experience with `GRHydro` suggests that the evolution is generally less stable when reconstructing conserved variables.

(3) For a given cell face, reconstruction from left and right side lead to potentially discontinuous hydrodynamic states on either side of the interface. We construct these Riemann problems at all the interfaces and solve them using Riemann solvers. We use an HLLE (Harten-Lax-van Leer-Einfeldt) [36–38] solver.

(4) The solutions of these Riemann problems provide us with the net flux across all the faces of a given cell. The divergence of this flux defines the flux term of the RHS (12) together with the source term (13). This completes the RHS evaluation.

In practice, Y_e is advected along with density, which means that we do not calculate the flux for DY_e by solving the Riemann problem. Instead, we use the flux of D and reconstructed value of Y_e to calculate the flux for DY_e as follows:

$$F_{DY_e} = Y_e^{\text{left}} F_D, \qquad F_D > 0 \tag{14}$$

$$= Y_e^{\text{right}} F_D, \qquad F_D < 0. \tag{15}$$

We have implemented TVD and WENO reconstruction methods in `GRaM-X`. TVD is 2nd-order accurate in regions of smooth, monotonic flows but reduces to first order in the presence of extrema and shocks. For WENO, we have implemented a version which is 5th order accurate for smooth, monotonic flows [35] and this is the reconstruction method we plan to use in our production simulations. The reconstruction method can be set using a runtime parameter.

We employ an HLLE Riemann solver in `GRaM-X`. This is an approximate solver which is less expensive numerically. More complex solvers such as Roe and Marquina have been numerically very resource intensive on traditional CPU-based codes, but we plan to implement them in the future in `GRaM-X` to extract the full compute capability of GPUs while attaining higher accuracy. This is because methods which are compute intensive but not memory intensive add little extra cost on GPUs.

Another important aspect of a GRMHD code is the EoS. We employ analytic equations of state such as Polytropic and Ideal Gas EoS, as well as realistic nuclear equations of state made available in the form of tables. For a Polytropic EoS, fluid pressure is given by $P = K\rho^{\gamma}$, where K is the Polytropic constant and γ is the Polytropic index. For an ideal gas (also known as Γ-law) EoS, the fluid pressure is given by $P = (\Gamma - 1)\rho\epsilon$. For nuclear tabulated equations of state, a total of 19 fluid variables such as pressure P, specific enthalpy h, the speed of sound

c_s etc are given as a function of (ρ, T, Y_e) in the form of a table, where T is the temperature and Y_e is the electron fraction [39].

To find the value of a given variable in a table, one needs to look up those values in the table and perform any necessary interpolations. Since we have a GPU-based code, we load the table in the `unified memory` which is accessible from both the CPU (*host*) and the GPU (*device*). We use tables in the format as found on [40].

The recovery of primitive variables ρ, v^i, ϵ, and B^i from the conserved variables D, S^i, τ and $\mathscr{B}^i$ is in principle a five-dimensional (5D) root-finding problem involving the inversion of equations (6)–(8), given that inverting the magnetic field components is trivial because they just differ by a factor of $\sqrt{\gamma}$. This, in turn, renders the unknown, first term of S^i in equation (7) collinear to v^i, thus reducing the dimensionality of the recovery problem to 3D.

In `GRaM-X`, we use either of two methods to perform this conserved to primitive transformation (referred to as `con2prim` hereafter): A 3D Newton-Raphson (3D-NR hereafter) [41, 42] root finder and the method of Newman & Hamlin (Newman's method hereafter) [41, 43]. In 3D-NR, the system of equations is converted to 3 equations in unknowns $(W, z = \rho h W^2, T)$ and solved for them, as described in [41]. Newman's method is an effective 1D method in which we iterate over the fluid pressure to find a solution. It solves a cubic polynomial of the form $f(\epsilon) = \epsilon^3 + a\epsilon^2 + d$ in variable $\epsilon \equiv B^2 + z$, where a and d are determined by fluid variables. This method requires the calculation of P from ρ and h using the EoS at every iteration step.

In practice, we use 3D NR as the primary method for `con2prim`. If 3D-NR does not converge, we fall back to Newman's method. The reason for choosing 3D-NR as the primary method over Newman's method is that the number of EoS calls is $\sim$20 times larger in case of Newman's method compared to 3D-NR [41], which makes Newman's method more expensive computationally. If Newman's method does not converge either, then we use bisection in temperature because it is guaranteed to converge to a solution, but is much more expensive computationally.

In order to preserve the divergence free constraint of the magnetic field, we use a simpler variant of the original constrained transport scheme [44–46] called 'flux-CT' [47–49]. While the original constrained transport scheme uses staggered magnetic field components, the flux-CT scheme instead uses cell centered values of the magnetic field. In this scheme, the evolution of magnetic field components is performed by calculating electric field components, which can in turn be expressed in terms of the magnetic field fluxes. We refer the reader to the `GRHydro` code paper [49] and the original literature for further details.

3. Implementation

`GRaM-X` is a complete redesign of `GRHydro` [16] which we have used successfully in production for many years [9, 16, 17, 50–52]. Like `GRHydro`, `GRaM-X` is based on the `Cactus` computational framework [53, 54], and uses the upcoming `CarpetX` driver to provide mesh refined grids, I/O and inter-node communication. The `Cactus` framework has been in use since 1998 [55] and has been chosen twice as a `SPEC` benchmark module [56] and has been used in hundreds of scientific publications.

`CarpetX` and thus `GRaM-X` use a traditional block structured mesh refinement approach that has been proven to be efficient for astrophysics problems targeted by `GRaM-X` in the past. At the same time, `GRaM-X` automatically benefits from improvements in the underlying `AMReX` library (see below), which provides efficient data management for GPU- and CPU-based simulations. Stencil based codes, such as `GRaM-X`, are usually limited by memory bandwidth and we employ a tiling and a hybrid MPI+OpenMP based approach to improve cache locality

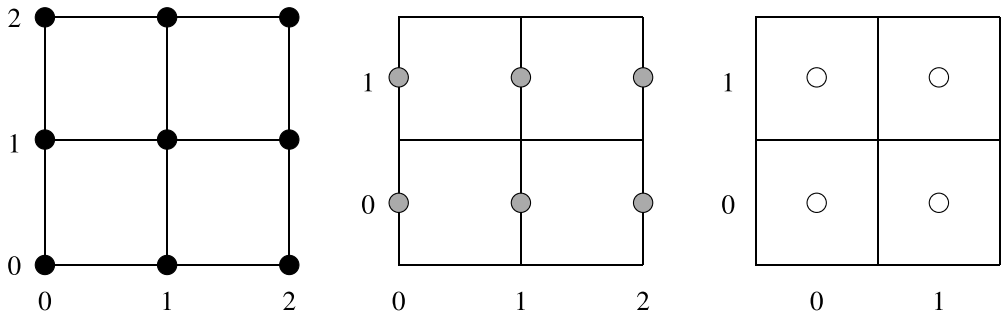

Figure 1. A sample of data arrangements supported by `CarpetX`, displaying different types of variables. From left to right: fully vertex centered, face centered in the *y* direction, and fully cell centered. Notice that there is an extra value in any vertex centered direction compared to cell centered directions. This figure was modeled after [61].

when using CPUs and kernel coalescing on GPUs to avoid scheduling a large number of small kernels. `AMReX` provides a portability layer similar to, e.g. `Kokkos`, that allows identical code to run on CPUs as well as GPUs by all current vendors (NVIDIA, AMD, Intel). `GRaM-X` uses this framework while it also contains optimized code for different architectures for performance critical code paths. Since `AMReX` and most current GPU frameworks target C++ code, `GRaM-X` reimplements the tested algorithms of our current `GRHydro` code, which contains legacy Fortran 90 routines, in C++, at the same time also removing experimental or no longer used code from `GRHydro`. `GRaM-X` uses modern C++17 features, in particular templates and type agnostic code, to allow for compile time optimization. On CPUs `GRaM-X` uses explicit vectorization using CPU architecture specific vector widths and optimized operations provided by `NSIMD` [57]. We have found that, in particular, the Einstein Field equations are too complex and lengthy for current compilers to optimize and vectorize well and instead require explicit vectorization by user code. Using C++ and the available SIMD libraries `GRaM-X`'s source code remains portable between different SIMD implementations on different CPU architectures.

`Cactus` provides a *flesh*, which works as a connection layer between end-user provided application code, *thorns*. Most of the functionalities necessary for complex multi-physics simulations are provided by the thorns. These thorns use `Cactus`' domain-specific language (DSL) to schedule subroutines, define interfaces for externally accessible subroutines, etc. All the desired thorns are given in a thornlist at compile time, but runtime parameters select which of these thorns are active in a given simulation.

Of preeminent importance for the performance of a simulation is the *driver* thorn. This special thorn handles all data movement and memory allocations, it implements the simulation workflow, and provides I/O services. Our driver is `CarpetX`. It implements Berger-Oliger block-structured AMR, calling user defined functions on blocks of data for refined sections of grid as needed. `CarpetX` also provides parallel I/O using the `Silo` [58], `ADIOS2` [59], and `openPMD` [60] libraries, high-order prolongation and restriction operators for variables defined at different grid locations (see figure 1 for an illustration), inter-process exchange of ghost-halo data using the `MPI` standard, intra-process parallelization using `OpenMP`, and collective data reduction operators for computing norms of grid functions.

We have developed `CarpetX` over the course of the past 3 years. `CarpetX` leverages the `AMReX` adaptive mesh refinement library that is developed for use by the DOE's Exascale Computing Project (ECP) [62]. With `AMReX`, our codes leverage the support of the `AMReX` community to ensure that the underlying AMR driver scales to ever larger systems and new

architectures. AMReX already implements algorithms to enable efficient use of current CPUs, and GPUs, as well as asynchronous iterators overlapping computation with communication, which benefit all our codes. The continued support for AMReX through the ECP co-design approach [63], where it is currently used by seven applications funded by ECP, provides a stable basis for CarpetX.

We have extended the inter-mesh transport operators in AMReX to support higher order accurate operations based on generic templated stencils to support the vertex centered high accuracy operations used for spacetime geometry quantities as well as cell centered conservative operations used for fluid quantities. All of CarpetX's basic operations are fully GPU enabled using AMReX's GPU facilities, thus avoiding costly data transfers from GPU to CPU memory. This includes facilities to extract gravitational waves.

Spacetime geometry quantities are stored on vertices, ensuring high accuracy in inter-mesh interpolation operations. In principle, both vertex-centered and cell-centered quantities can be interpolated to the same accuracy, but it is more expensive for cell-centered variables. During prolongation, if a fine grid point is at the same location as a coarse grid point, then interpolation is exact. Similarly, during restriction, vertex-centered quantities are always exact whereas cell-centered quantities might require higher-order interpolation. The fluid quantities are stored as cell averages in cells and updated in a flux-conservative scheme ensuring conservation of rest mass in a finite volume scheme. We pre-compute derivatives and fluxes using temporary storage locations and improve cache utilization (on CPUs).

When executing on CPUs, we use the NSIMD library [57] to achieve efficient SIMD vectorization of arithmetic operations. We also use a set of classes to handle vector and tensor objects and their interactions, such as dot products and small matrix inversions. These classes can also be used with NSIMD. We use OpenMP and grid tiling methods to distribute CPU work among multiple cores on a single compute node while ensuring efficient use of L2 and L3 caches. For GPUs AMReX supports CUDA, HIP/ROC or DPC++/SYCL as applicable for accelerator access.

Inter-node parallelism is handled using MPI as provided by AMReX. Cells at inter-processes and inter-mesh interfaces are filled via ghost zone (halo) exchange and prolongation (interpolation) from coarse to fine grid. At the same time, at the outer boundaries of the simulation domain, either periodic or a user-supplied boundary condition are applied.

CarpetX uses OpenPMD and its ADIOS2 backend to output checkpoint data, which we found to be more efficient than AMReX's built in BoxLib output. We employ Silo for 3D grid data output, which can be directly read by VisIt for visualization using visualization resources available at compute centers.

Our code contains extensive facilities to enforce correctness of the data access, tracking which parts of the grid are valid and detecting access attempts to invalid or out of date data. These facilities can be enabled via runtime parameters and are typically enabled for development and test simulations but disabled for production runs due to their impact on performance.

3.1. Adaptive mesh refinement

AMReX (and thus also CarpetX) provides so-called *block-structured mesh refinement*. This means that there is a rectangular *coarse grid*, and this coarse grid is overlaid by various *refined grids*, which are also rectangular, and which are organized in *refinement levels*. Each level has half the spacing of the next coarser level. The refined grid must be *properly nested*, which means that the grid of level $L+1$ must be wholly contained in the grids of level L.

During *regridding*, one needs to flag which cells of which grid need to be refined, and which currently refined cells are not needed any more. AMReX will then combine the region of flagged cells into new, rectangular grids, enlarging the refined region somewhat if necessary.

This ensures that the new refined region can easily be described by a set of rectangles. One assumes that the cost of the AMR algorithm scales not only with the total number of refined cells, but also with the number of rectangular grids, so that having fewer (but slightly larger) grids can be desirable.

Each refined grid is surrounded by a layer of *ghost zones*. These ghost zones are defined via interpolation (*prolongation*) from the next coarser level. Altogether, this means that the evolution system need only be implemented for rectangular arrays, and does not need to be immediately aware of the shape and relation of the refined regions. It also need only handle a single grid resolution, since sufficient ghost zones are provided by the AMR algorithm.

Finally, since the refined regions overlay coarser regions, this leads to a double covering of the domain. It is necessary to keep these consistent. When a refined grid's ghost zones are filled via prolongation from the next coarser grid, the coarse grid regions that are overlaid by the refined grid is reset to the respective fine grid values (*restriction*).

During time evolution, the CFL criterion states that coarser grids can take larger time steps than finer grids. This is also called *subcycling in time*, since the finer grids take more time steps than the coarser grids. For simplicity, we do not implement this yet; instead, we use the same time step size for all refinement levels. This increases the computational cost somewhat, but also increases parallel scalability (see section 5 below.), since all refinement levels can be evolved simultaneously. Subcycling in time necessarily serializes evolving different refinement levels.

`AMReX`'s AMR algorithm structures the refined grids as rectangular blocks of a given size that can be chosen at runtime. A typical block size would be $8 \times 8 \times 8$ cells. This means each refined grid is always a multiple of e.g. 8 cells large in each direction.

3.2. Staggered grids

It is often convenient to stagger evolved variables with respect to each other. For example, fluxes (section 2.2) are naturally defined on cell interfaces located halfway in between cell centers (*staggered*). `AMReX` allows quantities to be located either at cell centers, on cell faces, cell edges, or at the cell vertices. Its refinement scheme is based on cells. Quantities living on faces, edges, or vertices then live on grids that are one point larger in certain directions [64].

Being able to stagger fluxes between evolved conserved quantities, or placing the magnetic vector potential A^i at cell edges, has important advantages, in particular near mesh refinement interfaces where coarser and finer grids meet. For example, such schemes allow quantities to be exactly conserved during time evolution, or that the divergence of the magnetic field remains exactly zero. 'Exactly' here means up to error introduced by floating point precision. In `GRaM-X`, we store the spacetime quantities (e.g. $g_{\mu\nu}, T_{\mu\nu}$) on vertices, matter quantities (e.g. ρ, B^k) in cells, fluxes on faces and electric fields for constrained transport scheme on edges.

3.3. Parallelism

`CarpetX` provides three levels of parallelism, suitable for modern systems ranging from laptops to high-end supercomputers: shared memory parallelism (multi-threading), accelerators (aka GPUs), and distributed memory parallelism. These mechanisms are implemented in `AMReX`.

3.3.1. Shared memory parallelism (Multi-Threading). `CarpetX` uses OpenMP [65] for multi-threading. In this setup, a computational grid is allocated as a single entity, and is split into

several logical *tiles* for processing. Each tile is handled by one thread. The tile size can be chosen at runtime. We find that efficient tile sizes are very large in the x direction (because each cache line contains multiple grid points that are neighbors in the x direction) and rather small in the y and z directions (since our evolution system contains many variables that quickly fill the cache). We find that a typical efficient tile size would be e.g. $1000 \times 4 \times 4$ grid points, where the value 1000 means that the tile is in practice as large as the grid block in the x direction.

The routines acting on tiles are scheduled as `OpenMP` *tasks* and then execute independently. When necessary e.g. for *synchronization* (ghost zone exchange) or I/O, a barrier is introduced to ensure all scheduled tasks have finished.

3.3.2. Accelerators (GPUs). Most new high-end supercomputers require codes to make efficient use of GPUs or other accelerators. Accelerators on such systems provide the majority of the computing power. It is clearly important to be able to use GPUs efficiently on such systems.

One commonly used approach to do so is to ensure that all data are stored on the accelerator's built-in memory at all times, and are moved from and to the CPU memory only for I/O. It is thus, unfortunately, necessary that all routines of a simulation code run on the accelerator. Copying data to the CPU memory for even a simple task (e.g. to find the maximum of the density) is prohibitively slow; it is more efficient to run such a task on an accelerator.

The goal of `CarpetX` is thus to make it easy to write code that runs on accelerators, even if the code would not run efficiently there. Fortunately, code that has been written to run on accelerators will usually also run efficiently on CPUs.

The loop kernels running on an accelerator are scheduled and then execute as independent tasks. A barrier at the end ensures the accelerator has finished processing tasks before synchronization or I/O, and to ensure tasks execute in a correct order.

3.3.3. Distributed memory parallelism (Message-Passing). For distributed memory parallelism, i.e. to run across several compute nodes simultaneously, `AMReX` offers parallelization via MPI. The rectangular grids which make up the coarse grid and all refined grids are called *blocks* in `AMReX`. Each grid block is surrounded by a layer of ghost zones that are filled by copying from other MPI processes that may run on different nodes.

Overall, ghost zones are either filled via communication from another process or via prolongation from a coarser grid. This synchronization needs to be explicitly scheduled by the application code. Grids are automatically restricted (see above) at the same time they are synchronized.

In our implementation of a flux-conservative hydrodynamics scheme, the state vector consists of conserved quantities that are stored at cell centers. From these, we calculate the fluxes between the cells. These fluxes live on the faces between the cells. The fluxes are synchronized in a conservative manner, ensuring that the fluxes are consistent between all grid blocks, and also between coarse and fine grids. The fine grid fluxes are restricted to coarser grids. This restriction averages the fluxes, so that the integrated fluxes are the same on all refinement levels.

As we are using a global time step, so-called *refluxing* is not necessary. (Refluxing would keep fluxes between time steps with different step sizes consistent by integrating fluxes in time.)

From the difference of the fluxes, the new state vector is calculated, which is then is also synchronized.

4. Tests

To verify the correctness of the code we perform a number of tests to ensure that the numerical methods implemented are correct and robust. We perform a number of standard tests with a variety of initial conditions and compare the results with analytical solutions as well as results from other codes reported in literature. In this section we describe various tests in gradually increasing level of complexity, ranging from one-dimensional shocktube tests in static Minkowski spacetimes to a three-dimensional magnetized Tolman-Oppenheimer-Volkhof (TOV) star in dynamical spacetime.

4.1. 1D MHD shocktube

Planar MHD shocktube tests serve as the primary tests for magnetohydrodynamic codes because they are simple to implement yet provide an excellent testbed for the ability of the code to capture shocks and MHD-wave structures. We perform five different shocktube tests using the initial conditions proposed by Balsara [66], namely 'Balsara 1', 'Balsara 2', 'Balsara 3', 'Balsara 4' and 'Balsara 5'. Balsara 1 is the relativistic generalisation of the shocktube test problem originally proposed by Brio and Wu [67, 68]. Balsara 2 and 3 are blast wave test problems, with the difference being their initial pressure difference. Balasara 2 has a moderate initial pressure difference ($P_{\rm left}/P_{\rm right} = 30$) while Balsara 3 has a very strong initial pressure difference ($P_{\rm left}/P_{\rm right} = 10^4$). Balsara 4 constitutes a strongly relativistic test problem where two streams with high Lorentz factor ($W \approx 22.37$) collide with each other.

We perform the tests along x, y and z directions independently, however we show the results only for the x-direction. y- and z-directions give the same result. For the x-direction, we divide the domain in 'left' and 'right' states, with 'left' state being the region $x < 0$ and 'right' state being the region $x > 0$. This means $x = 0$ serves as the plane of discontinuity for the Riemann problem. For each test, we have used the Γ-law (ideal gas) EOS given by $P = (\Gamma - 1)\rho\epsilon$, with $\Gamma = 2$ for Balsara 1 and $\Gamma = 5/3$ for others. We divide the domain between -0.5 and 0.5 in 1600 points which gives a resolution of $\Delta x = 1/1600$. We perform all the tests without AMR and without constrained transport (see [49] for a detailed explanation of why this setup maintains the divergence-free constraint). We use fourth-order Runge-Kutta (RK4) for time integration with a high CFL factor of 0.8 (i.e. $\Delta t/\Delta x = 0.8$). We use Neumann boundary conditions i.e. we set the flux at the boundary points to be zero and 'copy' the data from the nearest point in the interior to the points at the boundary. We perform reconstruction of primitive variables using the TVD reconstruction with the minmod limiter and use the HLLE Riemann solver. We have used a static Minkowski spacetime. We set the left and right states using the values tabulated in table 1 and evolve the system until $t = t_{\rm ref} = 0.55$ for Balsara 5 and $t = t_{\rm ref} = 0.4$ for others. We then compare the results obtained at time $t_{\rm ref}$ with the exact solution. We have obtained the exact solution from [69]. This test took 29 s to evolve the $1600 \times 8 \times 8$ grid for 800 iterations on an NVIDIA RTX A6000 GPU.

We show the results for Balsara 1 and Balsara 4 in the left and the right panel of figure 2 respectively. We find that the results are in very good agreement with the analytical solution (limited only by numerical errors) and also agree well with other results reported in literature [19, 48, 66, 69–73]. In the left panel of figure 2 we plot the density, pressure, x-velocity, y-velocity, Lorentz factor and y-component of the magnetic field for Balsara 1. We find that

Table 1. Parameters used for the 1D MHD shocktube tests. We perform the tests in x-direction such that 'Left' state means the state at $x < 0$ and 'Right' state means the state at $x > 0$. Γ is the adiabatic index for the ideal gas equation of state, $t_{\rm ref}$ is the time at which we report the results, ρ is the density, ϵ is the specific internal energy, $\vec{v} = (v^x, v^y, v^z)$ is the velocity of the fluid and (B^x, B^y, B^z) are the x, y and z components of the magnetic field respectively.

Test name	B^x	Γ	$t_{\rm ref}$	State	ρ	ϵ	$\vec{v}$	(B^y, B^z)
Balsara 1	0.5	2	0.4	Left	1.0	1.0	$\vec{0}$	(1.0, 0)
				Right	0.125	0.8	$\vec{0}$	(−1.0, 0)
Balsara 2	5.0	5/3	0.4	Left	1.0	45.0	$\vec{0}$	(6.0, 6.0)
				Right	1.0	1.5	$\vec{0}$	(0.7, 0.7)
Balsara 3	10.0	5/3	0.4	Left	1.0	1500.0	$\vec{0}$	(7.0, 7.0)
				Right	1.0	0.15	$\vec{0}$	(0.7, 0.7)
Balsara 4	10.0	5/3	0.4	Left	1.0	0.15	(0.999, 0, 0)	(7.0, 7.0)
				Right	1.0	0.15	(−0.999, 0, 0)	(−7.0, −7.0)
Balsara 5	2.0	5/3	0.55	Left	1.08	1.3194	(0.4, 0.3, 0.2)	(0.3, 0.3)
				Right	1.0	1.5	(−0.45, −0.2, 0.2)	(−0.7, 0.5)

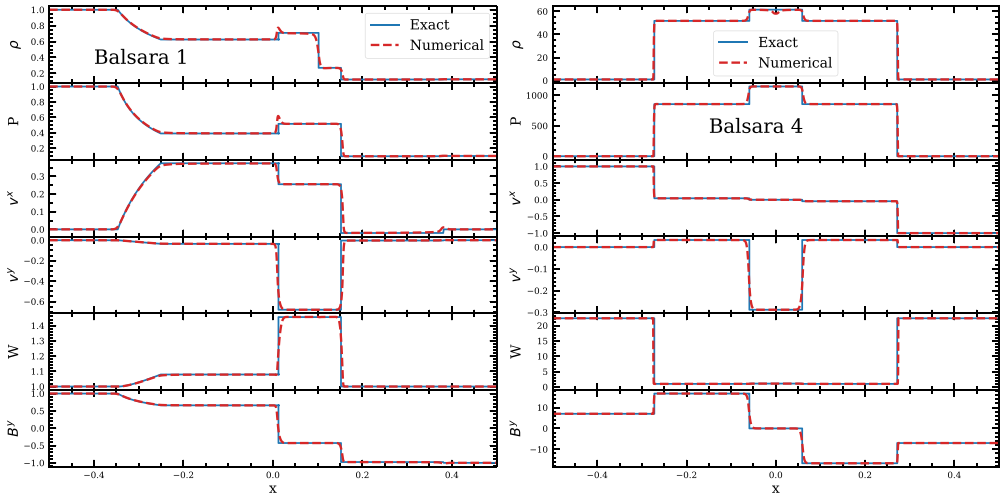

Figure 2. Results for the evolution of 1D MHD shocktube tests Balsara 1 and Balsara 4 in the left and the right panel respectively. We plot the density ρ, pressure P, normal velocity v^x, tangential velocity v^y, Lorentz factor W and tangential component of the magnetic field B^y at time $t_{\rm ref} = 0.4$ for both tests. We choose the initial conditions as listed in table 1 on a single-level grid between $[-0.5, 0.5]$ with a spatial resolution $\Delta x = 1/1600$ and evolve with a time resolution $\Delta t = 5 \times 10^{-4}$. We find that our results agree well with the analytical solution [69] (limited by numerical errors) and any minor deviations are consistent with those reported by other codes [48, 49, 66]. This test took 29 s to evolve on an NVIDIA RTX A6000 GPU.

a reasonably relativistic flow with a Lorentz factor of $\sim$1.458 has developed from the initial discontinuity and the code has captured all the elementary waves which include a left-going fast rarefaction wave, a left-going compound wave, a contact discontinuity, a right-going slow shock and a right-going fast rarefaction wave. This is also in good agreement with the exact

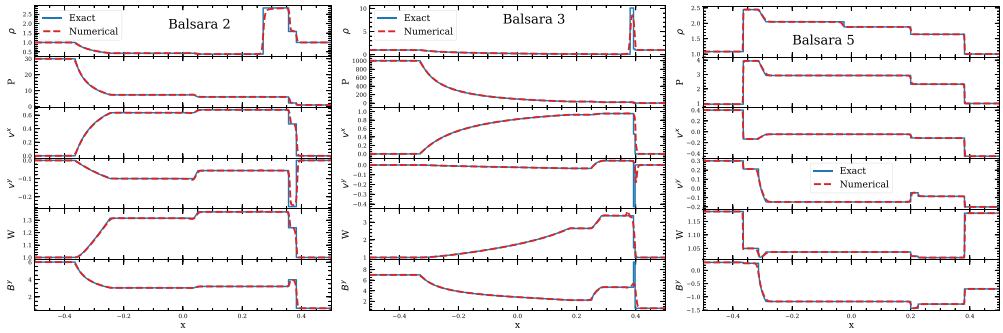

Figure 3. Same as figure 2 but for Balsara 2, Balsara 3 and Balsara 5. We plot the results at time $t_{\rm ref} = 0.4$ for Balsara 2 and 3, and at time $t_{\rm ref} = 0.55$ for Balsara 5. We find excellent agreement with the analytical results [69] for Balsara 2 and Balsara 5. Balsara 3 is the most demanding of all Balsara tests and we find that there is a considerable undershoot near the right moving fast shock for ρ, v^y and B^y. However, using WENO reconstruction instead of TVD reconstruction at the same resolution resolves this issue to a large extent and makes the deviations comparable to those observed by [49, 66].

solution. In the right panel of figure 2 we plot the same quantities for the MHD collision problem Balsara 4. The flow develops two very strong fast shocks and two slow shocks, each going to the right and left respectively. We find deviations from the exact result at the center but they are on the same level as [48, 49, 66].

We also show the results for Balsara 2, Balsara 3 and Balsara 5 in figure 3. For more details about the physics that these tests are testing please see [49]. For Balsara 2 and Balsara 5, we reproduce the analytic results very well. Balsara 3 is the most challenging of all the Balsara tests and we find that the result shows a considerable undershoot near the right moving shock at the current resolution of $\Delta x = 1/1600$. This however gets resolved to a large extent when we use WENO instead of TVD reconstruction, as shown in the left panel of figure 4. That makes the deviations comparable to those reported by [49, 66]. We also perform other Balsara tests (1, 2, 4, 5) with WENO instead of TVD reconstruction and find that the evolution is correct and stable in each case, though Balsara 2 and Balsara 4 develop minor oscillations near shocks with WENO at this resolution. However, we do note that we have not fine tuned the WENO parameters for each individual case which may altogether get rid of these minor oscillations.

We test the correctness of our AMR implementation using the Balsara 3 shocktube test. Balsara 3, with TVD reconstruction and resolution $\Delta x = 1/1600$, shows the highest disagreement with the analytic results for ρ, v^y and B^y at the right moving shock. Use of WENO instead of TVD rectifies this to a large extent (left panel of figure 4). Another solution is to increase the resolution near the right moving shock. This can be achieved by either increasing the resolution uniformly over the entire simulation domain or increasing the resolution only near the right moving shock using AMR. We therefore set up Balsara 3 shocktube test with TVD reconstruction for two different grid setups:

- `gridA`—a grid with one refinement level with extent $[-0.5,0.5]$ and resolution $\Delta x = 1/6400$.
- `gridB`—a grid with 3 refinement levels: extent $[-0.5,0.5]$ and resolution $\Delta x = 1/1600$ for level 0, and 2 more levels with lengths 0.18 ($\Delta x = 1/3200$) and 0.12 ($\Delta x = 1/6400$) that are centered at the right moving shock and move along with it.

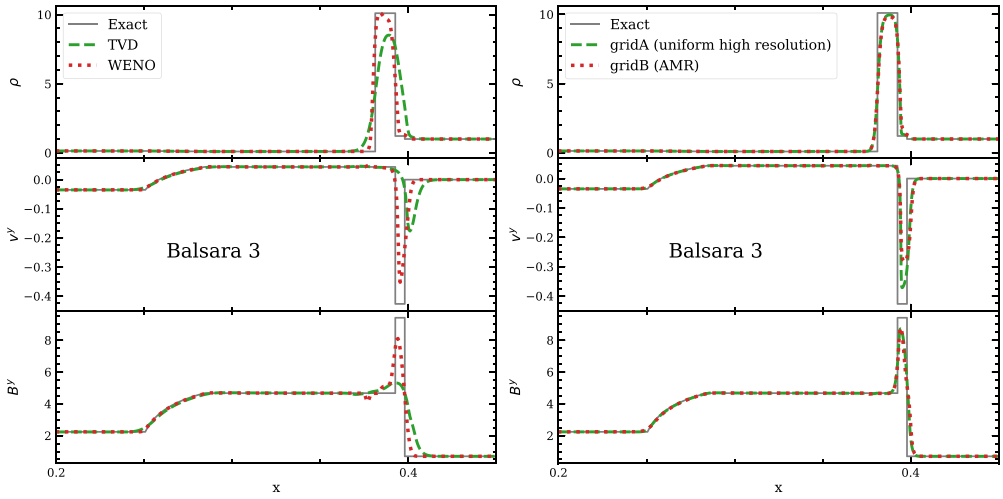

Figure 4. Comparison of different numerical methods using Balsara 3 shocktube test. We plot ρ, v^y and B^y between $0.2 \leqslant x \leqslant 0.45$ at time $t_{\rm ref} = 0.4$ to zoom-in on the right moving shock, because these quantities showed the highest discrepancy from the analytic solution near the right moving shock in figure 3. In the left panel, we show a comparison of TVD and WENO reconstruction. We find that WENO reconstruction considerably improves the accuracy at the same resolution ($\Delta x = 1/1600$). In the right panel, we test the AMR implementation of `GRaM-X`. We first perform the test using a uniform high resolution grid with $\Delta x = 1/6400$ (6400 cells total). We then repeat the test using AMR with 3 refinement levels (2944 cells total), with the finest level of length 0.12 and $\Delta x = 1/6400$ centered around the right moving shock. We find that AMR provides accuracy comparable to that of uniform high resolution grid.

Thus, `gridA` has effectively 6400 cells along x-direction, while `gridB` has effectively $1600+576+768 = 2944$ cells along x-direction. We clearly see in the right panel of figure 4 that the agreement with analytical results gets much better, and `gridB` leads to similar accuracy as `gridA` despite having a lower number of cells. This demonstrates the correctness of our AMR implementation as well as an efficiency gain of $\sim$50% for similar accuracy in this particular case.

4.2. Circularly polarized (CP) Alfvén wave

In order to study the convergence properties of `GRaM-X`, we test the propagation of a CP Alfvén wave along a uniform background magnetic field, first proposed by [70]. CP Alfvén waves are exact solutions of the equations of MHD and are smooth, making them suitable for convergence testing. Our setup is same as [49] where the initial velocities are given by

$$v^x = 0;\ v^y = -v_A A_0 \cos(kx);\ v^z = -v_A A_0 \sin(kx)\ , \tag{16}$$

and initial magnetic fields are given by

$$B^x = 1.0;\ B^y = A_0 B^x \cos(kx);\ B^z = A_0 B^x \sin(kx)\ . \tag{17}$$

The density is uniform with $\rho = 1.0$ and wave amplitude is chosen to be $A_0 = 1.0$. The Alfvén speed v_A is given by

$$v_A^2 = \frac{2(B^x)^2}{\rho h + (B^x)^2(1+A_0^2)} \left[1 + \sqrt{1 - \left(\frac{2A_0(B^x)^2}{\rho h + (B^x)^2(1+A_0^2)}\right)^2}\right]^{-1}, \tag{18}$$

where $k = 2\pi/L_x$ is the wavevector, L_x being the length of the domain along x direction. We perform the test in the domain $-0.5 \leqslant x \leqslant 0.5$ such that $L_x = 1$ and $k = 2\pi$. We choose a uniform pressure $P = 0.5$ which leads to the convenient value $v_A = 0.5$. We evolve the system with TVD reconstruction (mc2 limiter), HLLE Riemann solver, 2nd order (RK2) time integrator and periodic boundary condition, as well as constrained transport to maintain the divergence free character of the magnetic field. We use the Γ-law EOS with $\Gamma = 5/3$ and CFL factor of 0.2.

In the ideal case, the wave propagates across the grid and exactly reproduces the initial condition after every time period $T = L_x/v_A = 2$. We can test the accuracy of our code by measuring the differences between magnetic fields or velocities at $t = 0$ and $t = nT$, where n is an integer. We choose to test it for $n = 5$ i.e. when the wave has travelled 5 wavelengths across the grid. We perform the test with $N = 16$, 32, 64 and 128 cells along the x direction. For each N, we define the error $||\delta B||_2$ as the L_2-norm of the difference between the magnetic fields at $t = 0$ and $t = 10$. We plot the error as a function of N in figure 5 and find good agreement with the expected 2nd order error convergence rate. The results for error convergence rates also agree well with those reported in literature for similar setup of CP Alf vén wave test [49, 70, 74]. Our error magnitudes agree well with that of [49] who have also shown the result after 5 grid crossings.

4.3. 2D cylindrical explosion

We next perform a two dimensional test which involves a strong cylindrical explosion to test the ability of our code to capture an expanding blast wave in multiple dimensions. This test is known to push the limits of an MHD scheme and spot well-hidden bugs because of the simplicity of the setup. Unlike the Balsara tests we do not have an exact solution for this test, hence we compare our results with results of other codes available in literature. We use the test setup described in [49] which is originally based on test setup of [75] for the relatively weak magnetic field case ($B^x = 0.1$).

We perform the test in the x-y plane with the density profile given by

$$\rho(r) = \begin{cases} \rho_{\rm in} & ; \quad r \leqslant r_{\rm in}\,, \\ \exp\left[\frac{(r_{\rm out}-R)\ln\rho_{\rm out} + (r-r_{\rm in})\ln\rho_{\rm in}}{r_{\rm out}-r_{\rm in}}\right] & ; \quad r_{\rm in} < r < r_{\rm out}\,, \\ \rho_{\rm out} & ; \quad r \geqslant r_{\rm out}\,, \end{cases} \tag{19}$$

where $r = \sqrt{x^2+y^2}$ is the radial distance from the origin and $(r_{\rm in}, r_{\rm out})$ are radial parameters. The pressure gradient has an equivalent form where $\rho_{\rm in}$ and $\rho_{\rm out}$ are replaced by $P_{\rm in}$ and $P_{\rm out}$ respectively. We set the initial fluid velocity to zero in all directions and use a uniform magnetic field along x-direction. The values of the parameters are as follows:

$$r_{\rm in} = 0.8,\ r_{\rm out} = 1.0;\ \rho_{\rm in} = 10^{-2},\ \rho_{\rm out} = 10^{-4};\ P_{\rm in} = 1.0,\ P_{\rm out} = 3\times10^{-5};\ B^i = (0.1, 0, 0). \tag{20}$$

Thus there is a large pressure jump of $\sim 10^5$ going from $r_{\rm out}$ to $r_{\rm in}$. We use a $200 \times 200 \times 8$ grid with x- and y-directions spanning the coordinate range $[-6, 6]$. This results in a resolution of $\Delta x = \Delta y = 0.06$. We use the Γ-law EOS with $\Gamma = 4/3$ and use constrained transport for the

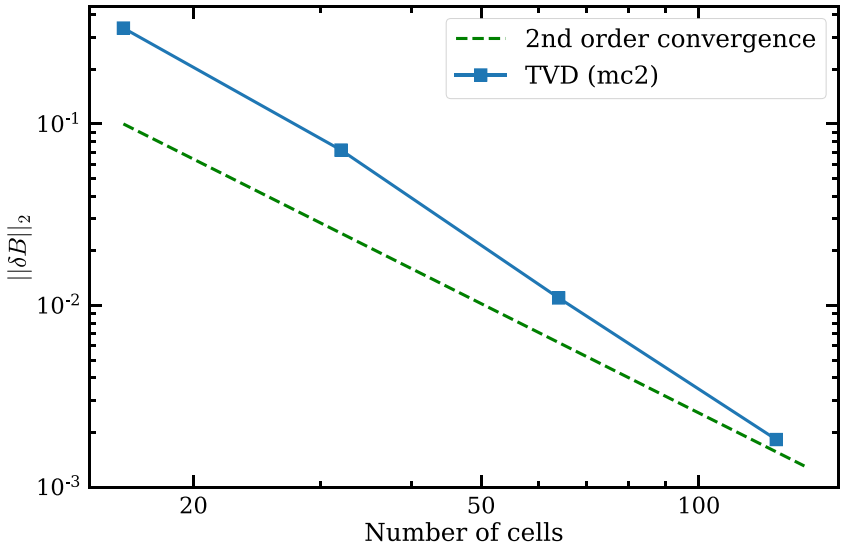

Figure 5. Figure shows the L_2-norm of the difference between the initial and final magnetic fields $||\delta B||_2$ as a function of the number of cells along the x-direction N, for the 1D circularly polarized Alf vén wave test. The green dashed line shows the expected error convergence rate for 2nd order convergence. We perform the test for $N = 16$, 32, 64 and 128 cells between $[-0.5, 0.5]$ along x-direction and calculate the error $||\delta B||_2$ after 5 wave cycles in each case. We find that the error convergence rate shows good agreement with the expected $\sim$2nd order convergence and similar results have been reported in the literature for the numerical methods used [49, 70, 74]. The error magnitudes agree well with that of [49] who also measure the errors after 5 wave cycles.

magnetic field evolution to conserve the divergence-free constraint of the magnetic field. We use Neumann boundary conditions as described in section 4.1 and do not use mesh refinement. We use TVD reconstruction with the minmod limiter, the HLLE Riemann solver and RK4 for time integration with the CFL factor of 0.25. With this numerical setup, we run the simulation until $t = 4$ and compare the two-dimensional and one-dimensional profiles of different physical quantities with previous results. This test took 15 s to evolve the $200 \times 200 \times 8$ grid for 267 iterations on an NVIDIA RTX A6000 GPU.

We show the two-dimensional profiles for gas pressure P, Lorentz factor W as well as the magnetic field in the $x - y$-plane at time $t = 4$ in the left panel of figure 6. We also show the numerically determined magnetic field lines as arrows on top of the Lorentz factor plot. The plot for P is in logarithmic scale. We find that our two-dimensional results show very good agreement with figure 5 of [49] and figure 10 of [75]. Next, we plot the one-dimensional profiles by taking slices along $y = 0$ and $x = 0$ for rest-mass density, gas pressure, magnetic pressure and Lorentz factor at time $t = 4$. Gas pressure and magnetic pressure are shown in logarithmic scale. We show these 1D profiles in the right panel of figure 6. We find good agreement with figure 6 of [49]. Slight deviations from the 1D profiles of [49] are present because our code uses cell-centering for hydrodynamic variables and vertex-centering for spacetime variables while [49] uses the same centering for both hydrodynamic and spacetime variables. This leads to slightly different numerical values (limited to numerical precision) for initial density and pressure profiles on the discrete grid due to their coordinate dependence. This small

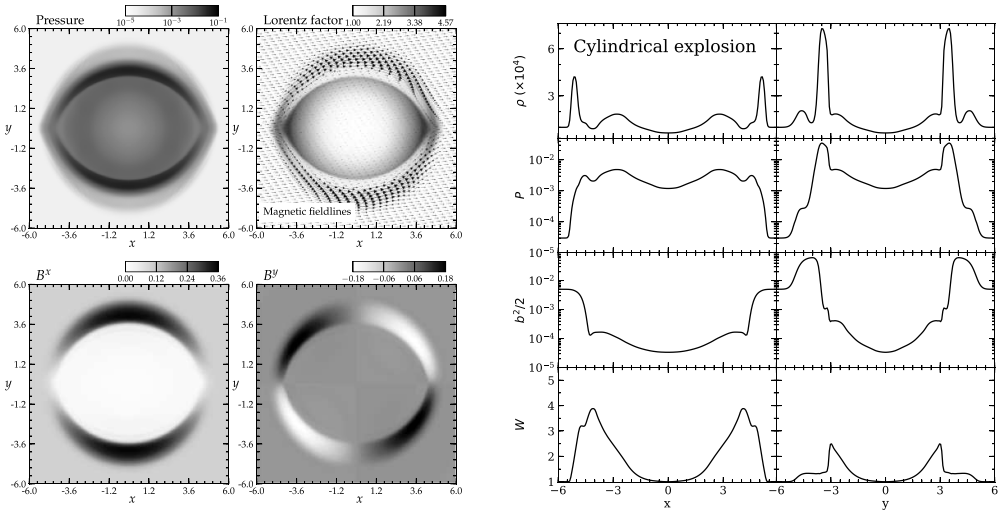

Figure 6. Left panel: The evolved state of the 2D cylindrical explosion test at time $t = 4$. We plot the logarithm of pressure, the Lorentz factor as well as the x and y components of the magnetic field. We also overlay arrows representing the magnetic field lines on top of the Lorentz factor plot. We perform the test on a single level grid between $-6 < x, y < 6$ with a spatial resolution of $\Delta x = \Delta y = 0.06$ and evolve with a temporal resolution of $\Delta t = 0.015$. We find that our results agree well with the results reported by [49, 75] for the same test setup. Right panel: 1D profile along $x = 0$ and $y = 0$, for density ρ, pressure P, magnetic pressure $b^2/2$ and Lorentz factor W. We find overall good agreement but minor deviations with figure 6 of [49] for density and magnetic pressure. These deviations occur because [49] uses vertex-centering while `GRaM-X` uses cell-centering for hydrodynamic variables. The deviations go away when we perform the test with same coordinate discretization as [49]. This test took 15 s to evolve on an NVIDIA RTX A6000 GPU.

discrepancy is resolved (not shown here) when we use the same coordinate discretization for hydrodynamic variables in [49] and our code.

4.4. 2D magnetic rotor

Next we perform the magnetic rotor test which has high initial Lorentz factors and a strong initial discontinuity. This test was generalized to the relativistic MHD case in [73]. Like the cylindrical explosion test, we do not have an analytical solution for this test and hence we compare our results with the results of other codes. The test consists of a cylindrical column of radius $r_{\rm in} = 0.1$ with its axis along z-direction rotating with an angular velocity of $\Omega = 9.95$. Thus the fluid 3-velocity at the outer edge of the cylinder reaches the value $v_{\rm max} = 0.995$ which corresponds to a high Lorentz factor of $\sim$10.012. The density inside the cylinder is uniform with the value $\rho_{\rm in} = 10$ while the medium surrounding the cylinder has a lower uniform density $\rho_{\rm out} = 1$. We do not apply any smoothing to the density profile, thus a strong initial discontinuity is present at the edge of the cylinder. The pressure is the same both inside and outside the cylinder with the value $P_{\rm in} = P_{\rm out} = 1$. A uniform magnetic field along x-direction is present both in the interior and exterior of the cylinder with the value $B^x = 1.0$.

We use a $400 \times 400 \times 8$ grid with x and y directions spanning the coordinate range $[-0.5, 0.5]$. This gives us a resolution of $\Delta x = \Delta y = 0.0025$. We perform the test on a flat

spacetime and without mesh refinement. We use Neumann boundary condition as described in section 4.1 and use constrained transport to evolve the magnetic field. We use a Γ-law EoS with an adiabatic index $\Gamma = 5/3$. We use TVD reconstruction with the minmod limiter and the HLLE Riemann solver. We perform time integration using RK4 with a CFL factor of 0.25. With this numerical setup we evolve the system until $t = 0.4$ and again compare the two-dimensional and one-dimensional profiles of different physical quantities with previous results. This test took 97 s to evolve the $400 \times 400 \times 8$ grid for 640 iterations on an NVIDIA RTX A6000 GPU.

In the left panel of figure 7 we show the 2D profiles of density, fluid pressure, magnetic pressure and Lorentz factor at $t = 0.4$. Arrows representing the magnetic field lines are shown on top of the Lorentz factor plot. As expected, we find that magnetic braking slows down the fluid and this causes the maximum Lorentz factor to drop to $\sim$1.85 from the initial value of $\sim$10. Magnetic field lines are also dragged by the rotation of the fluid which causes the field lines to rotate by almost 90 degrees in the central region while the field lines remain unchanged at large radii. Material has been swept away from the central region causing the central density to decrease from the initial maximum value of 10 to the minimum value of $\sim$0.4. Our results show very good agreement with figure 5 of [73], figure 7 of [49] and figure 8 of [76]. In the right panel of figure 7 we show the 1D profiles taken along $y = 0$ and $x = 0$ for density, fluid pressure, magnetic pressure and Lorentz factor at $t = 0.4$. Our results show excellent agreement with figure 8 of [49] and figure 9 of [76]. Again, we find minor deviations from the results of [49] but these again are due to the difference in centering of variables between [49] and our code. In fact, these deviations become irrelevant when we take into account the differences in results from the same code at different resolutions, for example figure 9 of [76].

4.5. 3D TOV star

As the final test we simulate a stationary and spherically symmetric neutron star with poloidal magnetic field in three dimensions. This test is more challenging than previous tests because (1) we perform it in three dimensions and (2) this test probes both GR and MHD. This is our first dynamical spacetime test and we use the Z4c formalism to evolve the spacetime variables. We obtain the neutron star model via the solution of TOV equation [77] and set up a poloidal magnetic field on top of this fluid configuration. We choose the test parameters, as described in the next paragraph, such that the maximum strength of magnetic field is $B_{\rm max} = 8.5 \times 10^{-6}$ and the maximum pressure is $P_{\rm max} \approx 1.6 \times 10^{-4}$ in code units. The ratio of fluid pressure to magnetic pressure is $\beta = \frac{P_{\rm max}}{B^2_{\rm max}/2} \approx 10^7$, hence the magnetic field is dynamically unimportant and only will insignificantly impact the stationary configuration on the timescales we perform this test. Hence this test helps us to investigate the ability of our code to maintain this stationary configuration during evolution in a dynamical spacetime setup.

The test involves setting up self-consistent initial data for fluid and spacetime variables, and then evolving the hydrodynamic variables with `GRaM-X` and the spacetime variables using the Z4c formalism with `Z4c`. In order to calculate the initial data, we solve the TOV equation in 1D using a polytropic EoS with polytropic constant $K = 100$, adiabatic index $\Gamma = 2$ and initial central density of $1.28 \times 10^{-3} M_\odot^{-2}$. These parameters form a TOV star with mass $M = 1.4 M_\odot$ and radius $R = 8.125 M_\odot$. We obtain the initial data for poloidal magnetic field using the vector potential $A_\varphi = A_b \varpi^2 (1 - \rho_0/\rho_0^{\rm max})^{n_p} \max(P - P_{\rm cut}, 0)$ with $\varpi \equiv \sqrt{(x - x_\star)^2 + y^2}$, where A_b determines the strength of the initial magnetic field, n_p shifts the location of maximum initial magnetic field, $P_{\rm cut}$ is the cut-off pressure below which we set magnetic field to zero and $(x_\star, 0)$ is the location of center of the star. For the test we choose $A_b = 1.0$, $n_p = 0$ and $P_{\rm cut} = 10^{-6}$. We use an artificial atmosphere outside the star where we set the density to $10^{-10} M_\odot$ and velocity to zero. We then calculate the pressure and specific energy density in the atmospheric

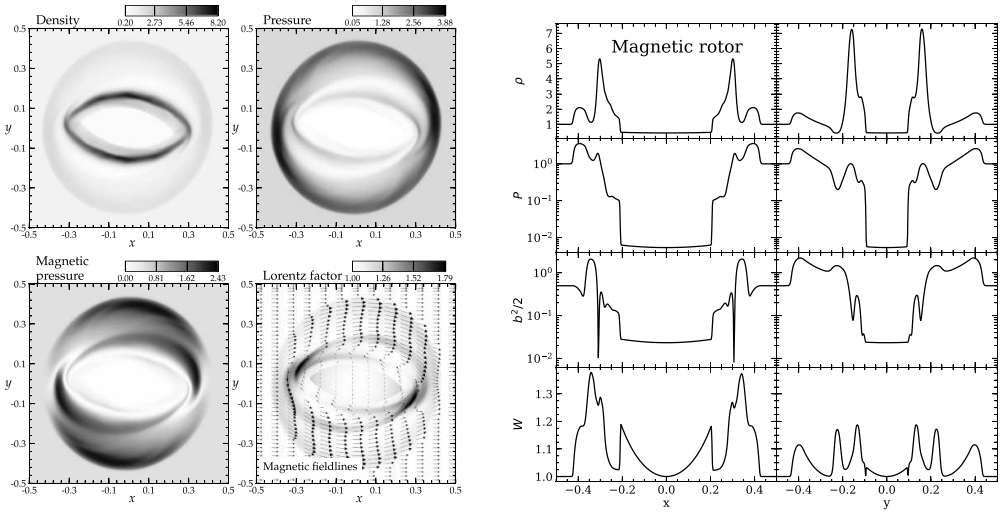

Figure 7. Left panel: The evolved state of the 2D magnetic rotor test at time $t = 0.4$. We plot the density, pressure, magnetic pressure and Lorentz factor, and also overlay arrows representing the magnetic field lines on top of the Lorentz factor plot. We perform the test on a single-level grid between $-0.5 < x, y < 0.5$ with a spatial resolution of $\Delta x = \Delta y = 1/400$ and evolve with a temporal resolution of $\Delta t = 6.25 \times 10^{-4}$. These 2D results show very good agreement with [49, 73, 76]. Right panel: 1D profiles along $x = 0$ and $y = 0$ for density ρ, pressure P, magnetic pressure $b^2/2$ and Lorentz factor W. Again, we find excellent agreement overall but some minor deviations with figure 6 of [49] and figure 9 of [76] due to different centering of variables. In fact, these deviations becomes irrelevant when we compare the results of other codes at different resolutions (see figure 9 of [76] for example). This test took 97 s to evolve on an NVIDIA RTX A6000 GPU.

region using a polytropic EoS with $K \simeq 100$ and $\gamma = 2$. We interpolate the one-dimensional TOV initial data to the three dimensional grid and use it as initial data for the test.

We set up four simulations on multiple AMR levels with fine grid resolutions of $1M_\odot(r0)$, $0.5M_\odot(r1)$, $0.25M_\odot(r2)$ and $0.125M_\odot(r3)$. The corresponding coarse grid resolutions are $8M_\odot(r0)$, $8M_\odot(r1)$, $8M_\odot(r2)$ and $4M_\odot(r3)$ respectively. The extent of the outermost cube is $640M_\odot$ for all simulations while the extent of the innermost cube is $96M_\odot(r0)$, $48M_\odot(r1)$, $24M_\odot(r2)$ and $22M_\odot(r3)$ respectively. We perform the simulations in the entire domain without the use of any symmetry. We evolve the system with a Γ-law EOS and $\Gamma = 2.0$, the HLLE Riemann solver, 5th order WENO reconstruction and an RK4 time integrator. We use the CFL factor $(\Delta t/\Delta x)$ of 0.25 where Δx is the resolution of the finest grid. We use constrained transport for the magnetic field. For the spacetime evolution using `Z4c`, we use the values for constraint damping parameters $\kappa_1 = 0.02$ and $\kappa_2 = 0.0$, and dissipation coefficient of 0.32 [78]. The finite numerical resolution acts as a perturbation to the equilibrium configuration of the star which leads to oscillations. The frequencies of these oscillations can be measured by taking a Fourier transform of the time variation of the central density of the star and then compared with the frequencies calculated using linear perturbation theory [79, 80]. These frequencies are different for static vs dynamical spacetime evolution and modes of gravity can only be obtained when the test is run with dynamical spacetime.

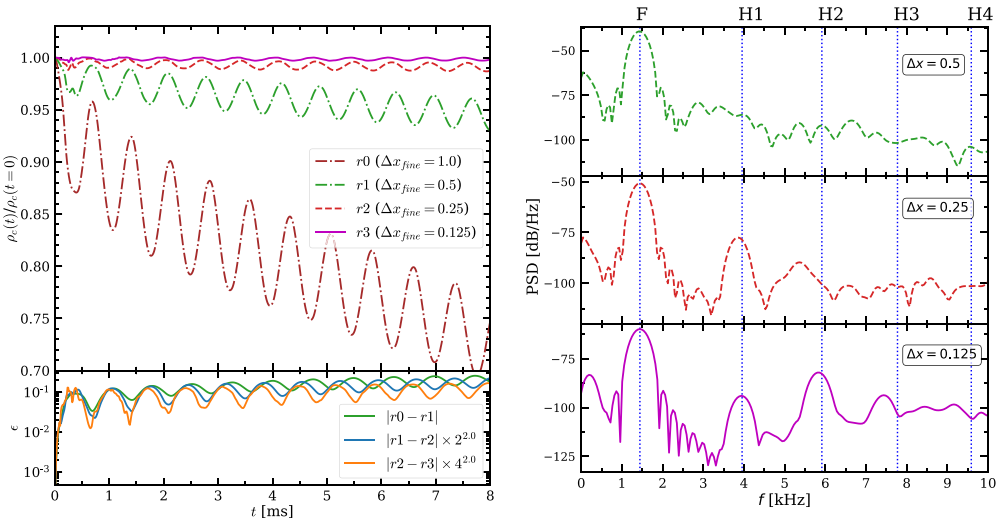

Figure 8. Left Panel: Normalized central density as a function of time for resolutions $r0(1M_\odot)$, $r1(0.5M_\odot)$, $r2(0.25M_\odot)$ and $r3(0.125M_\odot)$ until $t = 8ms$, where Δx_{fine} represents the resolution of the finest grid. Boundaries are present at $\pm 320M_\odot$. We find that the solution converges to the analytic solution (constant density) as we increase the resolution. To quantify this we show the time variation of absolute difference ε between $\{r0, r1\}$, $\{r1, r2\}$ and $\{r2, r3\}$ at the bottom. This gives us a convergence order of ~ 2. Right Panel: Power spectral density (PSD) of the oscillations of the central density of the star for different resolutions. We also show the frequencies of oscillations calculated using perturbation theory for comparison. For our highest resolution run with $\Delta x_{\text{fine}} = 0.125$, we find good agreement with the fundamental frequency (F) and the first two harmonics ($H1, H2$). The third harmonic ($H3$) appears slightly shifted from the correct value at this resolution. We expect that correct $H3$ and even higher modes can be obtained by increasing the resolution further.

We show the results for time variation of the normalized central density ρ_c in the left panel of figure 8. We find that the central density gets progressively closer to the ideal solution going from resolution $r0$ to $r3$. We also show this using the plot of absolute difference between $\{r0, r1\}$, $\{r1, r2\}$ and $\{r2, r3\}$ as a function of time. We get a convergence order of 2 going from low to high resolution. The amplitude of oscillations is larger in this case when compared to the results of `GRHydro` with the BSSN formalism. This happens because the centering is different for hydrodynamic and spacetime variables in `GRaM-X`. The stress-energy tensor $T_{\mu\nu}$ acts as the coupling term between hydrodynamic and spacetime variables. $T_{\mu\nu}$ itself is vertex-centered just as spacetime variables, but hydrodynamic variables are cell-centered. $T_{\mu\nu}$ depends on both hydrodynamic and spacetime variables and hence we have to interpolate the values of hydrodynamic variables from cell centers to vertex centers, which means that $T_{\mu\nu}$ is only an approximate value in `GRaM-X` and not an exact value as in [49]. This leads to higher numerical errors thus causing larger amplitude of oscillations. We have used 4th order 3D interpolation of fluid variables in the calculation of $T_{\mu\nu}$ because we found that 2nd order interpolation (i.e averaging) leads to an even larger, unacceptable amplitude of oscillations.

In the right panel of figure 8 we show the power spectral density (PSD) of the oscillations of the central density for runs at 3 different resolutions: $r1$, $r2$ and $r3$. We obtain the PSD in a manner similar to [81]. We also plot the frequencies obtained using perturbation theory against the PSD. We find that we only get the fundamental mode F at the resolution $\Delta x_{\text{fine}} = 0.5$.

We obtain the F mode as well as the first harmonic $H1$ when we increase the resolution to $\Delta x_{\rm fine} = 0.25$. For our highest resolution run with $\Delta x_{\rm fine} = 0.125$, we find clear agreement between the F, $H1$ and $H2$ modes. The third harmonic ($H3$) appears to be shifted slightly at this resolution. This trend suggests that at higher resolutions we should obtain the higher modes too, although at a substantially higher computational cost.

5. Performance and scalability

We test the performance of `GRaM-X` mainly on OLCF's supercomputer Summit. Each compute node on Summit consists of 6 NVIDIA Tesla V100 GPUs with 16 GB of High Bandwidth Memory per GPU. The amount of memory per GPU determines the maximum limit on the number of cells per GPU that can be used during a simulation. The computation/communication ratio should be kept high to use the GPUs efficiently and reduce the effect of memory transfers on the performance of the code.

In all the benchmarks that we report hereafter, we have used a periodic boundary condition, 5th order WENO reconstruction, and 3 ghost zones. We do not write any output files during runtime and do not include the time required to set up the initial condition in the benchmarks. We report the performance results for unigrid setups (i.e. without AMR) as well as for a 4-level AMR grid setup.

We perform the benchmark for 3 different physics setups:

(1) `SetupA`: static spacetime + ideal gas EoS + 2nd order Runge-Kutta (RK2) time integrator
(2) `SetupB`: static spacetime + *tabulated* EoS + *4th* order Runge-Kutta (RK4) time integrator
(3) `SetupC`: *dynamic* spacetime + tabulated EoS + 4th order Runge-Kutta time integrator

We use `setupA` mainly to compare our performance with other codes while `setupB` and `setupC` are our production simulation setups in a static and a dynamic spacetime, respectively. Physically, these setups represent the evolution of a $25M_{\odot}$ presupernova star. We map the 1D profile from a pre-collapse star obtained with MESA [82, 83] onto our 3D computational grid and evolve this system for a few iterations in static spacetime and dynamic spacetime for `setupB` and `setupC` respectively. The tabulated EoS that we use is the $K_0 = 220$ MeV variant of [84]. We report the results in terms of zone-cycles/s which we calculate using $(n_{\rm iter} \times n_{\rm cells})/t$, and zone-cycles/s/GPU which we calculate using $(n_{\rm iter} \times n_{\rm cells})/(t \times n_{\rm GPU})$, where t is the total time in seconds taken to finish $n_{\rm iter}$ iterations on a grid with $n_{\rm cells}$ cells running on $n_{\rm GPU}$ GPUs. Thus a single zone-cycle in the calculation above includes all intermediate steps for RK2/RK4 time integrators. Zone-cycles/s measures how efficient the simulation is performing overall (wall-time efficiency), whereas zone-cycles/s/GPU measures how efficient the simulation is performing on each GPU (cost efficiency).

5.1. Single GPU benchmarks

We first present the results for single GPU benchmarks (i.e. without communication) for the different physics setups. For `setupA` on a single Summit V100 GPU, we get 0.44×10^8 and 0.47×10^8 zone-cycles/s running with 152^3 cells/GPU and 240^3 cells/GPU respectively. This is comparable with the single summit V100 GPU performance of H-AMR [85] which obtains $\sim 0.85 \times 10^8$ zone-cycles/s for a similar physics setup. For `setupB`, the performance drops to 0.24×10^8 and 0.26×10^8 zone-cycles/s running with 152^3 cells/GPU and 240^3 cells/GPU respectively. This drop of ~ 1.8 times in speed occurs because we use tabulated EoS and RK4

integrator which has 4 (twice as many) intermediate steps per zone-cycle. Table lookups/interpolations are also more expensive computationally.

For `setupC`, we get 0.062×10^8 zone-cycles/s on a single GPU running with 168^3 cells/GPU. We cannot use more than 168^3 cells/GPU in this case because of the increased GPU memory consumption by the additional spacetime variables. The performance drops by a factor of $\sim$4 compared to the static spacetime case due to additional computations for spacetime evolution, but also partly due to the fact that our spacetime solver is not fully optimized for GPUs yet. (For example, there are a few instances of register spills for which work is being done to resize the kernels.) We expect the single GPU benchmark for this case to be $\sim 0.1 \times 10^8$ zone-cycles/s after optimization.

We also perform benchmarks on a single NVIDIA RTX A6000 GPU. For `setupA` we get 0.090×10^8 zone-cycles/s running on both 152^3 and 240^3 cells/GPU which indicates that the compute load gets saturated already at 152^3 cells/GPU. The performance is $\sim$5 times slower than that of a V100 GPU. For `setupB` we obtain 0.045×10^8 zone-cycles/s at 152^3 cells/GPU which is $\sim$5.5 times slower than the performance on V100 GPU. `setupC` gives us 0.020×10^8 zone-cycles/s at 168^3 cells/GPU which is $\sim$3 times slower than the V100 GPU performance for this test. Thus overall we find that the performance of `GRaM-X` on RTX A6000 GPU is 3-6 times slower than that of V100 GPU, depending on the test setup used. The reason for this difference is that the peak double precision performance (FP64) of NVIDIA V100 GPU is 7.8 TeraFlop/s while that of NVIDIA RTX A6000 GPU is only 1.25 TeraFlop/s which makes A6000 $\sim$6 times slower than V100 theoretically. The corresponding costs are today $\sim$\$10,500 for a V100 while $\sim$\$4500 for an RTX A6000, which makes a V100 GPU also more cost-efficient compared to an A6000 GPU.

We have also performed detailed profiling of our code for `setupC` where both the MHD solver (`GRaM-X`) and the spacetime solver (`Z4c`) run together. We find that the dominant kernel for MHD solver is the flux calculation routine (`GRaMX_Fluxes`) which accounts for $\sim 22\%$ of total runtime. We do however note that the flux calculation routine in `GRaM-X` also performs reconstruction in the same kernel before the flux calculation step, which makes this kernel the most expensive. The most expensive routines in the spacetime solver are the kernels `Z4c_ADM2`, `Z4c_RHS` and `Z4c_Constraints` which combined account for $\sim 67\%$ of total runtime. `Z4c_ADM2` calculates the first time derivative of extrinsic curvature and second time derivative of gauge variables, `Z4c_RHS` calculates the RHSs for evolution of `Z4c` variables, and `Z4c_Constraints` calculates the constraints of `Z4c` variables. Thus, these 4 kernels account for $\sim 90\%$ of the total runtime of the code which makes them the focus of our future optimization efforts, especially the spacetime kernels.

5.2. Weak scaling

Weak scaling is defined as how the performance of the code changes when the number of GPUs is increased, keeping the problem size per GPU fixed. This means that we increase the number of GPUs, and proportionately, increase the overall problem size to test how the time to solve the problem changes. In the ideal scenario, the time to solution should remain the same when we increase the number of GPUs, but in reality the time to solution generally increases on scaling up due to the communication overhead. We perform the weak scaling tests for `setupB` and `setupC` because they represent the setups that we will run for production simulations.

We first report the weak scaling performance for unigrid setups and discuss the performance with AMR at the end of this subsection since AMR requires different considerations. We perform the weak scaling test starting at 1 Summit node (6 V100 GPUs) going up to 2304 nodes (13 824 V100 GPUs) with 240^3 cells/GPU for `setupB` and 168^3 cells/GPU for `setupC`.

Table 2. Weak scaling grid setup and performance results for `setupB` and `setupC` for unigrid.

		`setupB` (240^3cells/GPU)			`setupC` (168^3cells/GPU)		
Nodes	GPUs	Grid	zone-cycles/ s/GPU	Efficiency	Grid	zone-cycles/ s/GPU	Efficiency
1	6	$720 \times 480 \times 240$	0.154×10^8	1.00	$504 \times 336 \times 168$	0.0412×10^8	1.00
36	216	$1440 \times 1440 \times 1440$	0.066×10^8	0.43	$1008 \times 1008 \times 1008$	0.0205×10^8	0.50
288	1728	$2880 \times 2880 \times 2880$	0.064×10^8	0.42	$2016 \times 2016 \times 2016$	0.0207×10^8	0.50
972	5832	$4320 \times 4320 \times 4320$	0.060×10^8	0.39	$3024 \times 3024 \times 3024$	0.0204×10^8	0.50
2304	13 824	$5760 \times 5760 \times 5760$	0.063×10^8	0.40	$4032 \times 4032 \times 4032$	0.0203×10^8	0.49

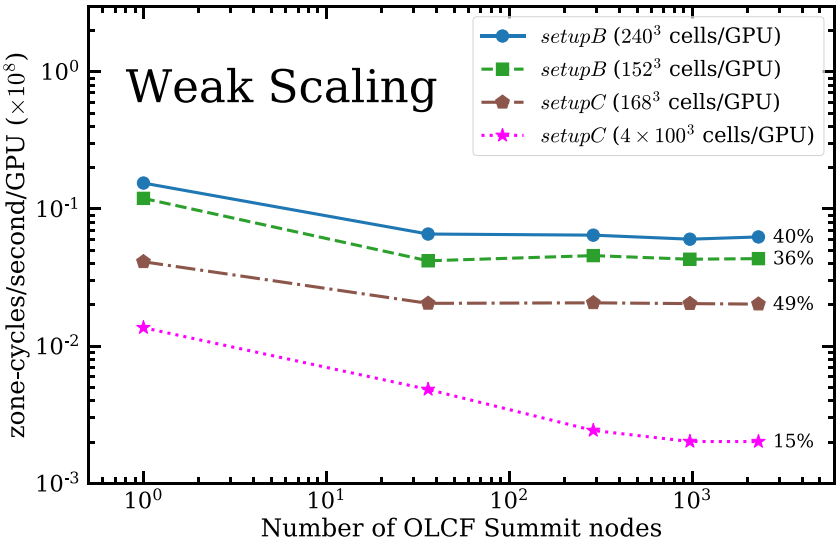

Figure 9. `GRaM-X` shows a weak scaling efficiency of ∼40%–50% on 2304 nodes (13824 NVIDIA V100 GPUs) with respect to single node performance on OLCF's supercomputer Summit. The weak scaling efficiency shows a steep drop going from 1 node to 36 nodes but remains almost constant going from 36 nodes up until 2304 nodes. We perform weak scaling test for `setupB` (static spacetime + Tabulated EoS + RK4) at 240^3 and 152^3 cells per GPU, and for `setupC` (*dynamic* spacetime + Tabulated EoS + RK4) at 168^3 cells per GPU. `setupB` shows better performance and scaling at 240^3 cells/GPU compared to 152^3 cells/GPU. `setupC` is the slowest due to extra computations for spacetime evolution but it shows the best scaling due to higher work load per GPU than other cases. We also perform weak scaling test for an AMR grid with 4 refinement levels, having 4×100^3 cells/GPU. The weak scaling shows a steep decline in this case even beyond 36 nodes, and the efficiency drops to ∼15% on 2304 nodes. The reason for ∼3 times less efficient performance compared to unigrid is due to the prolongation/restriction operations needed for AMR which require both more computations and more communication. Due to the current `CarpetX` infrastructure, there is also a serial component in evolving boxes on different levels that belong to the same GPU which further decreases the weak scaling efficiency.

Table 2 shows the grid setup used in each case for different number of nodes as well as the corresponding zone-cycles/s/GPU. We plot the results in figure 9. We find that the weak scaling efficiency shows a steep drop going from 1 node to 36 nodes but remains almost constant after

Table 3. Weak scaling grid setup and performance results for `setupC` with AMR.

		`setupC` (4×100^3cells/GPU)		
		Grid	zone-cycles/s/GPU	Efficiency
1	6	4 levels, each ($182 \times 182 \times 182$)	0.0136×10^8	1.00
36	216	4 levels, each ($600 \times 600 \times 600$)	0.0048×10^8	0.35
288	1728	4 levels, each ($1200 \times 1200 \times 1200$)	0.0024×10^8	0.18
972	5832	4 levels, each ($1800 \times 1800 \times 1800$)	0.0020×10^8	0.15
2304	13 824	4 levels, each ($2400 \times 2400 \times 2400$)	0.0020×10^8	0.15

that up until 2304 nodes. It should be noted that there is no inter-node communication when using 1 node and this might be the reason that this case is much faster. We get a weak scaling efficiency of $\sim$40% for `setupB` and $\sim$50% for `setupC` at 2304 nodes when compared with the respective single node benchmarks. We also find that the weak scaling efficiency depends on the way we choose the grid, with more asymmetric grid choice providing better scaling efficiency. In the extreme case where we choose the grid as $1\,399\,680 \times 240 \times 240$ on 972 nodes, we get a weak scaling efficiency of $\sim$98% with respect to single node performance. This happens because when the boxes are stacked in a cubical shape, the amount of ghost cell communication increases as $\sim n_{\text{boxes}}^3$ while when the boxed are stacked next to one another in an elongated shape, the amount of ghost cell communication increases only as $\sim n_{\text{boxes}}$, where n_{boxes} is the number of boxes in the domain. Therefore in the general case of an asymmetric grid, the number of ghost cells decreases thus leading to less communication overhead and better scaling.

Hence our weak scaling efficiency is limited to $\sim$40%–50% due to the communication required to fill ghost cells which, in turn, is limited by communication bandwidth (not latency). The solution to this problem is to either decrease the bandwidth requirements of our algorithms, increase available communication bandwidth, or possibly increase the overlap of computation and communication. The first can be achieved by using for example, Discontinuous Galerkin methods [86], which reduce the number of ghost zones needed thus reducing bandwidth requirements. We have not yet attempted to implement such methods. The second depends on the hardware on which we run the code and we cannot change it. The latter depends on our code infrastructure and there is room for improvement in this case. In the current implementation of `CarpetX`, the next GPU kernel is only launched once all the ghost zones from the previous kernel have been filled. However, if the next kernel does not need the values from the previous kernel, it can be launched already while ghost zones are getting filled. Another strategy would be to start the computations of those cells of the next kernel which do not need ghost cells while ghost cells from previous kernel are getting filled, and then compute the remaining cells thereafter. This would lead to more overlap of computation and communication, and could dramatically improve the scaling efficiency of the code. `CarpetX` is still in active development, and work is being carried out to increase the computation-communication overlap.

We also perform the weak scaling test with AMR on up to 2304 Summit nodes for `setupC`. We perform the test on a grid with 4 refinement levels, with each level containing $\sim 100^3$ cells/GPU. This makes the effective number of cells per GPU to be $4 \times 100^3 \approx 160^3$ which nearly saturates the GPU memory. Table 3 shows the grid setup and performance results for different number of nodes. We also plot the results in figure 9. We find that the performance drop is steeper for AMR grids compared to unigrid setups. We also find that unlike the unigrid case, the performance keeps dropping beyond 36 nodes and reaches an efficiency of

$\sim$15% on 2304 nodes, which is $\sim$3 times lower than unigrid setups. The absolute number of zone-cycles/s/GPU is also lower for AMR grids compared to unigrid setups, despite having similar cells/GPU. The reasons for these are two-fold (1) prolongation and restriction operations need to be performed for AMR grids to interpolate to fine grids and coarse grids respectively, which lead to extra computations compared to unigrid setups thus leading to lesser zone-cycles/s/GPU (2) communication from coarse grid to fine grid is required during prolongation to fill the ghost zones of boxes on *all* the finer levels and communication is also required to write restricted values to coarse grids, which means overall more communication is needed for a 4-level grid compared to a single-level grid. This leads to less efficient weak scaling even when the number of cells/GPU is same. This also means that increasing the number of levels further will make the weak scaling efficiency worse.

The suggestions for improving weak scaling efficiency for unigrid setups also apply to AMR grids, with a few additional points. Improving the performance of prolongation/restriction operations will lead to higher overall performance with AMR and may also lead to better scaling. Another solution is related to the way `CarpetX` currently handles the boxes in an AMR simulation. Currently, each GPU handles one or more boxes from *each* refinement level, which means that the calculations for boxes that a GPU owns happen in a serial manner since each level contributes at least one box to each GPU. A more efficient strategy would be that each GPU handle one (or more) box(es) from *any* one level, which will remove this serialization thus improving the weak scaling considerably.

5.3. Strong scaling

Strong scaling is defined as how the time to solution changes when number of GPUs is increased keeping the problem size fixed. In order to use GPUs efficiently, it is important to maximize the compute load per GPU which reduces the effect of communication overhead. Thus, strong scaling is not suited for GPUs because keeping the problem size fixed and increasing the number of GPUs decreases the work load per GPU which reduces the efficiency. However, it is also true that running the code at the most cost efficient setup is generally not the fastest. It is, therefore, important in case of an actual production simulation to identify the optimal compute load per GPU. Hence we perform a strong scaling test by varying the compute load per GPU from 48^3 cells/GPU to 240^3 cells/GPU.

We perform the strong scaling test for `setupB` with a grid size of $1440 \times 960 \times 480$ which is equal to a total of $\sim 872^3$ cells. We need to run this on a minimum of 8 Summit nodes (48 V100 GPUs) due to GPU memory constraints which corresponds to a compute load of 240^3 cells/GPU. We then increase the number of nodes keeping the grid size fixed, going up to 1000 nodes (6000 V100 GPUs) which corresponds to a compute load of 48^3 cells/GPU. Using the benchmarks, we calculate the total time (in hours) it will take to run this simulation till 10^5 iterations and how much it will cost in total (in GPU-hours) for each case. We tabulate the results in table 4 and show the plot in figure 10. We find that running this simulation on 8 nodes will take $\sim$46 hours but cost only $\sim$2 200 GPU-hours. On the other hand, running the same simulation on 1000 nodes will take only $\sim$2 hours but have a much larger cost of $\sim$12 000 GPU-hours. The value of zone-cycle/s/GPU drops below 50% at compute load of 120^3 cells/GPU compared to 240^3 cells/GPU which happens because we are increasing the communication overhead and decreasing the compute load per GPU both of which bring down the efficiency. So running on anything below a compute load of $\sim 120^3$ cells/GPU will be very inefficient. Running with $\sim 160^3$ cells/GPU is a good compute load for reasonable total runtime and cost efficiency on unigrid for `setupB`. This value will be different when we consider problems

Table 4. Strong scaling test for `setupB` with a fixed problem size of $1440 \times 960 \times 480$ varying number of nodes from 8 to 1000.

Nodes	GPUs	cells/GPU	zone-cycles/s	zone-cycles/s/GPU	Efficiency	Time for 10^5 iterations (hours)	Cost for 10^5 iterations (GPU-hours)
8	48	240^3/GPU	4.02×10^8	0.084×10^8	1.00	45.8	2200
27	162	160^3/GPU	7.72×10^8	0.048×10^8	0.57	23.9	3870
64	384	120^3/GPU	14.07×10^8	0.037×10^8	0.44	13.1	5031
125	750	96^3/GPU	22.03×10^8	0.029×10^8	0.35	8.4	6276
1000	6000	48^3/GPU	93.46×10^8	0.016×10^8	0.19	2.0	11 833

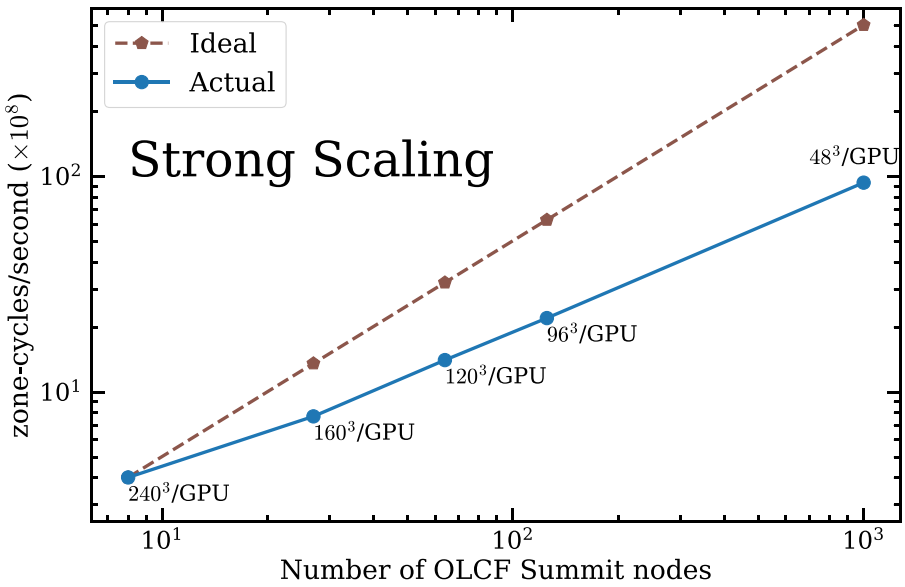

Figure 10. We perform the strong scaling test on `setupB` (static spacetime + Tabulated EoS + RK4) by varying the number of nodes from 8 to 1000 for a fixed problem size of $1\,440 \times 960 \times 480$. This increase in the number of nodes for the fixed problem size progressively reduces the compute load per GPU from 240^3 cells/GPU (8 nodes) to 48^3 cells/GPU (1000 nodes). Lowering the compute load until 120^3 cells/GPU (64 nodes) gives us a strong scaling efficiency of $\sim\%$ which is mainly because our weak scaling efficiency also drops up until $\sim$36 nodes. However, we find that strong scaling efficiency drops further to $\sim$19% going from 64 to 1000 nodes even though our weak scaling efficiency did not show any drop in this region. This is because going below $\sim120^3$–160^3 cells/GPU (>27–64 nodes), the compute load per GPU is not enough to keep all GPU threads occupied resulting in inefficient GPU utilization.

involving AMR or a different physics setup but similar reasoning can be applied there to arrive at a reasonable value of cells/GPU.

6. Summary

We present a new GPU-accelerated dynamical-spacetime GRMHD code `GRaM-X` (**G**eneral **R**elativistic **a**ccelerated **M**agnetohydrodynamics on AMRe**X**) which runs efficiently and

scales well across thousands of GPUs. GRaM-X is built upon the new AMR driver for Einstein Toolkit CarpetX and extends the capability of the Einstein Toolkit to simulate relativistic astrophysical systems such as core-collapse supernovae (CCSN) and binary neutron-star mergers (BNS) to GPU-based exascale supercomputers.

GRaM-X features 3D adaptive-mesh refinement (AMR) and support for both analytic as well as tabulated equations of state. The AMR and GPU functionality of GRaM-X is enabled by the new AMR driver CarpetX which, in turn, leverages AMReX, an AMR library developed as part of the Block-Structured AMR Co-Design Center in the US DOE's Exascale Computing Project (ECP). We evolve the equations of general relativity using the Z4c formalism and the equations of ideal MHD using the Valencia formulation. GRaM-X includes 2nd-order accurate TVD (Total Variation Diminishing) and 5th-order accurate WENO (Weighted Essentially Non-Oscillatory) reconstruction methods. The Riemann solver GRaM-X employs is the HLLE (Harten-Lax-van Leer-Einfeldt) solver which is an approximate solver that uses a two-wave approximation to compute the update terms across the discontinuity at the cell interface. We use 3D-NR as the primary method for conserved-to-primitive transformation and fall back to the method of Newman & Hamlin (an effective 1D method) in cases when the 3D-NR does not converge or converges to unphysical values.

We have written GRaM-X from scratch in C++ with the core routines such as reconstruction methods and Riemann solver adapted from the already well-established code GRHydro. In order to test the validity and accuracy of the code, we perform a series of tests on static spacetime which include 1D MHD shocktubes, 2D magnetic rotor, and a 2D cylindrical explosion. We also test the code in dynamical spacetime using the linear oscillations of a 3D TOV star and extract the modes of gravity including the fundamental mode (F) and the first two overtones ($H1$, $H2$). In all the tests, we find very good agreement with the analytic results (wherever available) as well as the results reported by other codes in literature for similar test setups. We also perform single GPU benchmarks and scaling tests on OLCF's Summit to test the performance of GRaM-X. We find that the weak scaling efficiency of GRaM-X is $\sim$40%–50% on up to 2304 Summit nodes (13824 V100 GPUs) with respect to single node performance. We also find that higher compute load per GPU leads to more efficient performance, with a compute load of 240^3 and 168^3 cells/GPU leading to the most efficient performance for our production simulations with static and dynamical spacetime respectively on summit V100 GPUs.

GRaM-X, for the first time, enables us to perform dynamical-spacetime general-relativistic astrophysics simulations such as core-collapse supernovae and neutron-star mergers at a speed 10 times faster and computational cost 30 times lesser compared to traditional CPU-based codes. Adjusting for the typical allocations awarded for CPU vs GPU supercomputers, one can perform $\sim$5 times more simulations with GPU-based codes such as GRaM-X compared to traditional CPU-based codes at a fraction of total runtime. This number will be higher if one maximizes the compute load per GPU which will lead to longer runtimes but even more cost efficiency. This makes GPU-computing both cost-effective as well as more environmentally friendly. We are currently testing and implementing a moment-based neutrino transport (M1) scheme [87] to be used with GRaM-X for future production simulations. We also plan to include more compute intensive routines such as more accurate Riemann solvers Roe [88] and Marquina [89, 90]. We expect the performance to be more efficient and the scaling to be the same or even better with the addition of these more compute intensive routines.

Data availability statement

The data cannot be made publicly available upon publication because they are not available in a format that is sufficiently accessible or reusable by other researchers. The data that support the findings of this study are available upon reasonable request from the authors.

Acknowledgments

This research was supported by National Science Foundation Grants Nos. NSF-2004879, NSF-2103680, NSF-1550514, NSF-2004157 and ACI-1238993. The authors thank Ann Almgren, Weiqun Zhang, Andy Nonoka, Don Willcox, and the entire `AMReX` developer team for helpful discussions designing the code. S S and P M thank Matthew Liska for helpful discussions regarding scalability. S S thanks L B L for hosting him during the summer internship 2022 where some of this work was carried out. This work has benefited from participation in the NERSC December 2021 GPU hackathon [91] and the authors like to especially thank the team mentors during the event Ronnie Chatterjee and Vassilios Mewes and the event leaders Iris Chen and Kevin Gott. The simulations were carried out on OLCF's Summit under allocation AST154 and LSU's DeepBayou. Research at Perimeter Institute is supported in part by the Government of Canada through the Department of Innovation, Science and Economic Development and by the Province of Ontario through the Ministry of Colleges and Universities.

ORCID iDs

Swapnil Shankar https://orcid.org/0000-0002-5109-0929
Philipp Mösta https://orcid.org/0000-0002-9371-1447
Steven R Brandt https://orcid.org/0000-0002-7979-2906
Roland Haas https://orcid.org/0000-0003-1424-6178
Erik Schnetter https://orcid.org/0000-0002-4518-9017

References

[1] Palenzuela C 2013 Modelling magnetized neutron stars using resistive magnetohydrodynamics *Mon. Not. R. Astron. Soc.* **431** 1853–65
[2] Etienne Z B, Paschalidis V, Haas R, Mösta P and Shapiro S L 2015 Illinois GRMHD: an open-source, user-friendly GRMHD code for dynamical spacetimes *Class. Quantum Grav.* **32** 175009
[3] Kiuchi K, Cerdá-Durán P, Kyutoku K, Sekiguchi Y and Shibata M 2015 Efficient magnetic-field amplification due to the Kelvin-Helmholtz instability in binary neutron star mergers *Phy. Rev.* D **92** 124034
[4] Paschalidis V 2017 General relativistic simulations of compact binary mergers as engines for short gamma-ray bursts *Class. Quantum Grav.* **34** 084002
[5] Ruiz M, Lang R N, Paschalidis V and Shapiro S L 2016 Binary neutron star mergers: a jet engine for short gamma-ray bursts *Astrophys. J. Lett.* **824** L6
[6] Ciolfi R, Kastaun W, Giacomazzo B, Endrizzi A, Siegel D M and Perna R 2017 General relativistic magnetohydrodynamic simulations of binary neutron star mergers forming a long-lived neutron star *Phy. Rev.* D **95** 063016
[7] Ciolfi R, Kastaun W, Vijay Kalinani J and Giacomazzo B 2019 First 100 ms of a long-lived magnetized neutron star formed in a binary neutron star merger *Phy. Rev.* D **100** 023005
[8] Ciolfi R 2020 Collimated outflows from long-lived binary neutron star merger remnants *Mon. Not. R. Astron. Soc.* **495** L66–L70
[9] Mösta P, Radice D, Haas R, Schnetter E and Bernuzzi S 2020 A magnetar engine for short GRBs and kilonovae *Astrophys. J. Lett.* **901** L37

[10] Kiuchi K, Held L E, Sekiguchi Y and Shibata M 2022 Implementation of advanced Riemann solvers in a neutrino-radiation magnetohydrodynamics code in numerical relativity and its application to a binary neutron star merger (arXiv:2205.04487)
[11] Coughlin M W, Dietrich T, Antier S, Bulla M, Foucart F, Hotokezaka K, Raaijmakers G, Hinderer T and Nissanke S 2020 Implications of the search for optical counterparts during the first six months of the Advanced LIGO's and Advanced Virgo's third observing run: possible limits on the ejecta mass and binary properties *Mon. Not. R. Astron. Soc.* **492** 863–76
[12] Raaijmakers G *et al* 2021 The challenges ahead for multimessenger analyses of gravitational waves and kilonova: a case study on GW190425 *Astrophys. J.* **922** 269
[13] Burrows A, Livne E, Dessart L, Ott C D and Murphy J 2006 A new mechanism for core-collapse supernova explosions *Astrophys. J.* **640** 878
[14] Obergaulinger M, Cerdá-Durán P, Müller E and Aloy M A 2009 Semi-global simulations of the magneto-rotational instability in core collapse supernovae *Astron. Astrophys.* **498** 241
[15] Winteler C, Käppeli R, Perego A Arcones A, Vasset N Nishimura N, Liebendörfer M and Thielemann F-K 2012 Magnetorotationally driven supernovae as the origin of early galaxy r-process elements? *Astrophys. J. Lett.* **750** L22
[16] Mösta P, Mundim B C, Faber J A, Haas R, Noble S C, Bode T, Löffler F, Ott C D, Reisswig C and Schnetter E 2014 GRHydro: a new open-source general-relativistic magnetohydrodynamics code for the Einstein toolkit *Class. Quantum Grav.* **31** 015005
[17] Mösta P, Roberts L F, Halevi G, Ott C D, Lippuner J, Haas R and Schnetter E 2018 R-process nucleosynthesis from three-dimensional magnetorotational core-collapse supernovae *Astrophys. J.* **864** 171
[18] Obergaulinger M and Aloy M A 2020 Magnetorotational core collapse of possible GRB progenitors—I. Explosion mechanisms *Mon. Not. R. Astron. Soc.* **492** 4613–34
[19] Anton L, Zanotti O, Miralles J A, Marti J M, Ibanez J M, Font J A and Pons J A 2006 Numerical 3+1 general relativistic magnetohydrodynamics: a local characteristic approach *Astrophys. J.* **637** 296
[20] Mösta P, Ott C D, Radice D, Roberts L F, Haas R and Schnetter E 2015 A large-scale dynamo and magnetoturbulence in rapidly rotating core-collapse supernovae *Nature* **528** 376
[21] Kiuchi K, Sekiguchi Y, Kyutoku K, Shibata M, Taniguchi K and Wada T 2015 High resolution magnetohydrodynamic simulation of black hole-neutron star merger: mass ejection and short gamma ray bursts *Phys. Rev.* D **92** 064034
[22] Liska M *et al* 2019 H-AMR: a new GPU-accelerated GRMHD code for exascale computing with 3D adaptive mesh refinement and local adaptive time-stepping (arXiv:1912.10192)
[23] Stone J M, Tomida K, White C J and Felker K G 2020 The Athena++ adaptive mesh refinement framework: design and magnetohydrodynamic solvers *Astrophys. J. Suppl.* **249** 4
[24] Hilditch D, Bernuzzi S, Thierfelder M, Cao Z, Tichy W and Brügmann B 2013 Compact binary evolutions with the Z4c formulation *Phys. Rev.* D **88** 084057
[25] Ruiz M, Hilditch D and Bernuzzi S 2011 Constraint preserving boundary conditions for the z4c formulation of general relativity *Phys. Rev.* D **83** 024025
[26] Shibata M and Nakamura T 1995 Evolution of three-dimensional gravitational waves: harmonic slicing case *Phys. Rev.* D **52** 5428–44
[27] Baumgarte T W and Shapiro S L 1999 On the numerical integration of Einstein's field equations *Phy. Rev.* D **59** 024007
[28] Alcubierre M, Brügmann B, Dramlitsch T, Font J A, Papadopoulos P, Seidel E, Stergioulas N and Takahashi R 2000 Towards a stable numerical evolution of strongly gravitating systems in general relativity: the conformal treatments *Phys. Rev.* D **62** 044034
[29] Alcubierre M, Brügmann B, Diener P, Koppitz M, Pollney D, Seidel E and Takahashi R 2003 Gauge conditions for long term numerical black hole evolutions without excision *Phys. Rev.* D **67** 084023
[30] Bernuzzi S and Hilditch D 2010 Constraint violation in free evolution schemes: Comparing the bssnok formulation with a conformal decomposition of the z4 formulation *Phys. Rev.* D **81** 084003
[31] Albin E, d'Angelo Y and Vervisch L 2010 Using staggered grids with characteristic boundary conditions when solving compressible reactive Navier-Stokes equations *Int. J. Numer. Methods Fluids* **68** 546–63
[32] Courant R, Friedrichs K and Lewy H 1928 Über die partiellen Differenzengleichungen der mathematischen Physik *Math. Ann.* **100** 32–74

[33] Brown D, Diener P, Sarbach O, Schnetter E and Tiglio M 2009 Turduckening black holes: An analytical and computational study *Phys. Rev.* D **79** 044023
[34] Toro E F 1999 *Riemann Solvers and Numerical Methods for Fluid Dynamics* (Springer)
[35] Shu C-W 1998 Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws *Advanced Numerical Approximation of Nonlinear Hyperbolic Equations* (*Lecture Notes in Mathematics* vol 1697) (Institute for Computer Applications in Science and Engineering, NASA Langley Research Center) pp 325–432
[36] Einfeldt B 1988 On Godunov-type methods for gas dynamics *SIAM J. Numer. Anal.* **25** 294–318
[37] Harten A 1983 High resolution schemes for hyperbolic conservation laws *J. Comp. Phys.* **49** 357–93
[38] Gammie C F, McKinney J C and Toth G 2003 HARM: a numerical scheme for general relativistic magnetohydrodynamics *Astrophys. J.* **589** 444–57
[39] Buyukcizmeci N, Botvina A S and Mishustin I N 2014 Tabulated equation of state for supernova matter including full nuclear ensemble *Astrophys. J.* **789** 33
[40] `stellarcollapse.org`: a community Portal for stellar collapse, core-collapse supernova and GRB simulations
[41] Siegel D M, Mösta P, Desai D and Wu S 2018 Recovery schemes for primitive variables in general-relativistic magnetohydrodynamics *Astrophys. J.* **859** 71
[42] Cerdá-Durán P, Font J A, Antón L and Müller E 2008 A new general relativistic magnetohydrodynamics code for dynamical spacetimes *Astron. Astrophys.* **492** 937–53
[43] Newman W I and Hamlin N D 2014 Primitive variable determination in conservative relativistic magnetohydrodynamic simulations *SIAM J. Sci. Comput.* **36** B661–83
[44] Evans C R and Hawley J F 1988 Simulation of magnetohydrodynamic flows: a constrained transport model *Astrophys. J.* **332** 659
[45] Yee K 1966 Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media *IEEE Trans. Antennas Propag.* **14** 302–7
[46] Balsara D S and Spicer D S 1999 A staggered mesh algorithm using high order godunov fluxes to ensure solenoidal magnetic fields in magnetohydrodynamic simulations *J. Comput. Phys.* **149** 270–92
[47] Tóth G 2000 The $\nabla.B = 0$ constraint in shock-capturing magnetohydrodynamics codes *J. Comp. Phys.* **161** 605–52
[48] Giacomazzo B and Rezzolla L 2007 WhiskyMHD: a new numerical code for general relativistic magnetohydrodynamics *Class. Quantum Grav.* **24** S235–58
[49] Mösta P, Mundim B C, Faber J A, Haas R, Noble S C, Bode T, Löffler F, Ott C D, Reisswig C and Schnetter E 2013 GRHydro: a new open-source general-relativistic magnetohydrodynamics code for the Einstein toolkit *Class. Quantum Grav.* **31** 015005
[50] Mösta P, Andersson L, Metzger J, Szilágyi B and Winicour J 2015 The Merger of Small and Large Black Holes *Class. Quantum Grav.* **32** 235003
[51] Halevi G and Mösta P 2018 *r*-Process nucleosynthesis from three-dimensional jet-driven core-collapse supernovae with magnetic misalignments *Mon. Not. R. Astron. Soc.* **477** 2366–75
[52] Curtis S, Mösta P, Wu Z, Radice D, Haas R, and Schnetter E 2021 Nucleosynthetic yields and kilonova lightcurves from hypermassive neutron-star merger remnants (arXiv:2112.00772)
[53] Goodale T, Allen G, Lanfermann G, Massó J, Radke T, Seidel E and Shalf J 2023 Cactus computational toolkit (available at: www.cactuscod e.org/)
[54] Allen G, Goodale T, Lanfermann G, Radke T, Rideout D and Thornburg J 2011 Cactus Users' Guide (available at: https://einsteintoolkit.org/usersguide/UsersGuide.html)
[55] Bona C, Masso J, Seidel E and Walker P 1998 Three-dimensional numerical relativity with a hyperbolic formulation vol 3
[56] Tao J, Allen G, Diener P and Schnetter E 2017 SPEC benchmark module (available at: www.spec.org/cpu2017/Docs/benchmarks/507.cactuBSSN_r.html)
[57] Agenium Scale 2019 NSIMD documentation (available at: https://agenium-scale.github.io/nsimd/)
[58] Silo: a library for reading and writing scientific data
[59] ADIOS 2: the adaptable input/output system
[60] Open standard for particle-mesh data
[61] AMReX collaboration 2022 AMReX documentation (available at: https://amrex-codes.github.io/amrex/docs_html/Basics.html)
[62] Exascale Computing Initiative 2016 Exascale computing project (available at: www.exascaleproject.org/)

[63] Zhang W, Myers A, Gott K, Almgren A and Bell J 2020 AMReX: block-structured adaptive mesh refinement (AMR) co-design center (available at: www.exascaleproject.org/amrex-co-design-center-helps-five-application-projects-reach-performance-goals/)
[64] Zhang W, Myers A, Gott K, Almgren A and Bell J 2020 AMReX documentation (available at: https://amrex-codes.github.io/amrex/docs_html/)
[65] Chandra R, Dagum L, Kohr D, Menon R, Maydan D and McDonald J 2001 *Parallel Programming in Openmp* (Morgan kaufmann)
[66] Balsara D 2001 Total variation diminishing scheme for relativistic magnetohydrodynamics *Astrophys. J. Suppl. Ser.* **132** 83–101
[67] Brio M and Wu C C 1988 An upwind differencing scheme for the equations of ideal magnetohydrodynamics *J. Comput. Phys.* **75** 400–22
[68] van Putten M H P M 1995 A two-dimensional numerical implementation of magnetohydrodynamics in divergence form *SIAM J. Numer. Anal.* **32** 1504–18
[69] Giacomazzo B and Rezzolla L 2006 The exact solution of the riemann problem in relativistic magnetohydrodynamics *J. Fluid Mech.* **562** 223–59
[70] Del Zanna L, Zanotti O, Bucciantini N and Londrillo P 2007 Echo: a eulerian conservative high-order scheme for general relativistic magnetohydrodynamics and magnetodynamics *Astron. Astrophys.* **473** 11–30
[71] Anderson M, Hirschmann E W, Liebling S L and Neilsen D 2006 Relativistic MHD with adaptive mesh refinement *Class. Quantum Grav.* **23** 6503–24
[72] Balsara D S 1998 Total variation diminishing scheme for adiabatic and isothermal magnetohydrodynamics *Astrophys. J. Suppl. Ser.* **116** 133–53
[73] Del Zanna L, Bucciantini N and Londrillo P 2003 An efficient shock-capturing central-type scheme for multidimensional relativistic flows - ii. magnetohydrodynamics *Astron. Astrophys.* **400** 397–413
[74] Beckwith K and Stone J M 2011 A second-order godunov method for multi-dimensional relativistic magnetohydrodynamics *Astrophys. J. Suppl. Ser.* **193** 6
[75] Komissarov S S 1999 A Godunov-type scheme for relativistic magnetohydrodynamics *Mon. Not. R. Astron. Soc.* **303** 343–66
[76] Etienne Z B, Tung Liu Y and Shapiro S L 2010 Relativistic magnetohydrodynamics in dynamical spacetimes: A new adaptive mesh refinement implementation *Phys. Rev.* D **82** 084031
[77] Tolman. R C 1939 Static solutions of Einstein's field equations for spheres of fluid *Phys. Rev.* **55** 364–73
[78] Brügmann B, González J A, Hannam M, Husa S, Sperhake U and Tichy W 2008 Calibration of moving puncture simulations *Phys. Rev.* D **77** 024027
[79] Yoshida S'ichirou and Eriguchi Y 2001 Quasi-radial modes of rotating stars in general relativity *Mon. Not. R. Astron. Soc.* **322** 389–96
[80] Font J A, Goodale T, Iyer S, Miller M, Rezzolla L, Seidel E, Stergioulas N, Suen W-M and Tobias M 2002 Three-dimensional numerical general relativistic hydrodynamics. ii. long-term dynamics of single relativistic stars *Phys. Rev.* D **65** 084024
[81] Löffler F *et al* 2012 The Einstein toolkit: a community computational infrastructure for relativistic astrophysics *Class. Quantum Grav.* **29** 115001
[82] Paxton B, Bildsten L, Dotter A, Herwig F, Lesaffre P and Timmes F 2011 Modules for experiments in stellar astrophysics (MESA) *Astrophys. J. Suppl.* **192** 3
[83] Paxton B *et al* 2013 Modules for experiments in stellar astrophysics (MESA): planets, oscillations, rotation and massive stars *Astrophys. J. Suppl.* **208** 4
[84] Lattimer J M and Douglas Swesty F 1991 A Generalized equation of state for hot, dense matter *Nucl. Phys.* A **535** 331–76
[85] Liska M *et al* 2019 H-amr: a new gpu-accelerated grmhd code for exascale computing with 3D adaptive mesh refinement and local adaptive time-stepping
[86] Kidder L E *et al* 2017 Spectre: a task-based discontinuous galerkin code for relativistic astrophysics *J. Comput. Phys.* **335** 84–114
[87] Radice D, Bernuzzi S, Perego A and Haas R 2022 A new moment-based general-relativistic neutrino-radiation transport code: methods and first applications to neutron star mergers *Mon. Not. R. Astron. Soc.* **512** 1499–521
[88] Roe P L 1981 Approximate Riemann solvers, parameter vectors and difference schemes *J. Comput. Phys.* **43** 357–72

[89] Donat R and Marquina A 1996 Capturing shock reflections: an improved flux formula *J. Comput. Phys.* **125** 42–58
[90] Aloy M A, Ibanez J M, Marti J M and Muller E 1999 Genesis: a high resolution code for 3-D relativistic hydrodynamics *Astrophys. J. Suppl.* **122** 151–66
[91] NERSC December GPU Hackathon 2021 (available at: www.gpuhackathons.org/event/nersc-december-gpu-hackathon-2021)