The Astrophysical Journal Supplement Series 249:4 40pp), 2020 July https: doi.org 10.3847 1538-4365 ab929b

The Athena Adaptive Mesh Re nement Framework: Design and Magnetohydrodynamic Solvers

James M. Stone[1,2,5], Kengo Tomida[3,6], Christopher J. White[4], and Kyle G. Felker[2,7]
[1] Department of Astrophysical Sciences, Princeton University, Princeton, NJ 08544, USA; jmstone@ias.edu
[2] Program in Applied and Computational Mathematics, Princeton University, Princeton, NJ 08544, USA
[3] Department of Earth and Space Science, Osaka University, Toyonaka, Osaka, 560-0043, Japan
[4] Kavli Institute for Theoretical Physics, University of California Santa Barbara, Santa Barbara, CA 93107, USA
Received 2019 June 26; revised 2020 April 21; accepted 2020 May 11; published 2020 June 25

Abstract

The design and implementation of a new framework for adaptive mesh re nement calculations are described. It is intended primarily for applications in astrophysical fluid dynamics, but its flexible and modular design enables its use for a wide variety of physics. The framework works with both uniform and nonuniform grids in Cartesian and curvilinear coordinate systems. It adopts a dynamic execution model based on a simple design called a "task list" that improves parallel performance by overlapping communication and computation, simpli es the inclusion of a diverse range of physics, and even enables multiphysics models involving different physics in different regions of the calculation. We describe physics modules implemented in this framework for both nonrelativistic and relativistic magnetohydrodynamics MHD). These modules adopt mature and robust algorithms originally developed for the Athena MHD code and incorporate new extensions: support for curvilinear coordinates, higher-order time integrators, more realistic physics such as a general equation of state, and diffusion terms that can be integrated with super-time-stepping algorithms. The modules show excellent performance and scaling, with well over 80 parallel ef ciency on over half a million threads. The source code has been made publicly available.

Uni ed Astronomy Thesaurus concepts: Astronomy software 1855); Magnetohydrodynamics 1964)

1 Introduction

Computational methods are now rmly established as essential tools for studying many problems in astrophysical fluid dynamics. A number of publicly available codes that implement a range of algorithms and features are widely used for such problems. Examples of widely used based on, e.g., citations) grid-based codes include ZEUS Stone & Norman 1992a, 1992b; Stone et al. 1992; Hayes et al. 2006), ART Kravtsov et al. 1997), FLASH Fryxell et al. 2000), RAMSES Teyssier 2002), HARM Gammie et al. 2003), PLUTO Mignone et al. 2007, 2012), Athena Stone et al. 2008, hereafter S08), and Enzo Bryan et al. 2014), among others.

There is a common trend among modern codes for astrophysical fluid dynamics toward increasingly complexity. This trend is driven by a number of factors. First, realistic models of many astrophysical systems require the inclusion of additional physics, such as radiation transfer, self-gravity, chemical or nuclear reaction networks, and for relativistic flows) dynamical spacetimes. Second, in order to resolve widely disparate length scales and timescales, it is now common for grid-based methods to adopt one of several different adaptive mesh re nement AMR) strategies. In addition, such codes often implement a variety of algorithmic options, such as different coordinate systems, Riemann solvers in the case of Godunov schemes), and spatial and temporal approximations of varying formal orders of accuracy. Supporting all possible combinations of physics and algorithmic options on an AMR mesh is challenging. Finally, modern high-performance computing systems are becoming increasingly heterogeneous. Developing portable code that performs well on the wide range of available architectures presents an additional challenge.

The Athena code S08), written in C, is a prototypical illustration of this evolution toward increasing complexity. The numerical algorithms, based on the extension of unsplit nite-volume methods to MHD using upwind constrained transport CT), were initially described in Gardiner & Stone 2005, 2008). Subsequently, the code was augmented with different time integrators Stone & Gardiner 2009), the shearing-box approximation Stone & Gardiner 2010), cylindrical coordinates Skinner & Ostriker 2010), special relativity SR; Beckwith & Stone 2011), particles Bai & Stone 2010), sink particles Gong & Ostriker 2013), a total energy-conserving formalism for self-gravity Jiang et al. 2013), and radiation transport Davis et al. 2012; Jiang et al. 2012, 2014a; Skinner & Ostriker 2013), among many other features. Maintaining and updating Athena as progressively more physics and algorithms are implemented has become increasingly untenable. Moreover, the AMR strategy adopted in the original code, based on overlapping patches Berger & Oliger 1984; Berger & Colella 1989), was found not to perform well on modern highly parallel architectures.

The need to address these issues has led to a complete redesign and rewrite of the code from scratch. The rst and most important aspect of this redesign has been the abstraction of the mesh from the physics modules solved on it. In the new design, the mesh exists as an independent, abstract framework on which various discretizations of the dependent variables such as cell-entered volume averages, face-centered area averages, or vertex- or cell-centered pointwise values) are constructed and stored. Methods for AMR, various boundary conditions, and distributed-memory parallelization using domain decomposition are then implemented

[5] Current address: School of Natural Sciences, Institute for Advanced Study, Princeton, NJ 08544, USA.
[6] Current address: Astronomical Institute, Tohoku University, Sendai, Miyagi 980-8578, Japan.
[7] Current address: Argonne National Laboratory, Lemont, IL 60439, USA.

for these discrete variables, without speci c references to any particular physics. This greatly simpli es the extension of the code to both new coordinates and new physics that are immediately compatible with AMR in any geometry. Moreover, isolating the mesh infrastructure from the physics allows each to be developed independently: for example, a performance-portable version of the AMR infrastructure based on the Kokkos library Edwards et al. 2014) that can be run on heterogeneous architectures including GPUs) is now under development.

A second important aspect of this redesign has been the adoption of a block-based AMR design e.g., Stout et al. 1997), as opposed to the patch-based AMR in the style of Berger & Oliger 1984) implemented in the original version of Athena and also used in codes like PLUTO and Enzo. There are a number of compelling reasons that motivate the adoption of block-based AMR. In patch-based AMR, re ned regions are covered by multiple levels of meshes. Quantities derived from the conserved variables such as temperature) can therefore possess different values on different levels. In turn, this can lead to different dynamics on separate levels if, for example, there are source terms such as cooling or chemical or nuclear reaction networks that depend on temperature. By carefully designing our block-based AMR so that each position in the domain is covered with one and only one mesh level, this complication is eliminated. Moreover, when patch-based AMR is parallelized using domain decomposition, the overlap between Message Passing Interface MPI) domains on different levels can become complex. With our implementation of block-based AMR, different levels communicate only through the boundaries. This simpli es the implementation and greatly improves the performance and scaling on parallel architectures. Perhaps the best-known code that uses a block-based AMR strategy is FLASH Fryxell et al. 2000), which is based on the PARAMESH AMR library MacNeice et al. 2000). However, rather than using preexisting libraries, we have instead written our own AMR framework in order to support face-centered variables as required by our implementation of MHD), reduce library dependencies, and improve performance by sacri cing generality.

Finally, a third important aspect of this redesign is the use of dynamic scheduling. Rather than hard-code the order of execution of steps in the numerical algorithms including MPI send and receives), these steps are instead assembled into lists of encapsulated tasks. Individual tasks can be executed in any order, provided that the tasks upon which they depend are complete. The ability to dynamically adapt the order of execution of tasks allows the overlap of parts of the computation with MPI communication which in turn can improve parallel scaling on very large number of processors). Moreover, the design enables a wide range of calculations containing different subsets of physics, as it is simple to change the composition of the task list. Even more powerfully, calculations in which different physics is simulated in disjoint regions are enabled simply by constructing separate task lists for each region. For example, this organization facilitates the straightforward inclusion of a particle-in-cell PIC) code for modeling the collisionless dynamics of the corona that is formed in the upper regions of an MHD simulation of an accretion disk e.g., Miller & Stone 2000). A variety of sophisticated libraries, such as Legion[8] or CHARM++,[9] which implement dynamic execution using task-based parallelism in which a master processes schedules data and tasks to available processors), among many other useful features, are available. Because we only require the ability to schedule tasks dynamically, and in order to reduce dependencies on external libraries, we have implemented our own design, requiring a few thousand lines of special-purpose code.

To take advantage of language extensions that improve modularity and organization, we have adopted the C++ language for this framework; therefore, we refer to the resulting new code as Athena++. This paper provides an introduction to the AMR framework in Athena++. We focus on the new features of this design, especially the implementation of block-based AMR with both cell- and face-centered variables as required for MHD), extension of the design to various coordinate systems, and dynamic execution using task lists. These features constitute the basic building blocks of the framework, upon which any physics solver can be implemented.

In the interest of providing concrete examples of physics modules within the framework, we also describe the implementation of algorithms for both nonrelativistic and relativistic MHD in this framework, based on the methods used in Athena. Because these algorithms have been described in detail in previous papers Gardiner & Stone 2005, 2008; S08; Beckwith & Stone 2011; White et al. 2016), we con ne the scope of our description to only novel features related to the new framework design. There are a variety of other physics modules in development within the Athena++ framework, and these will be described in future publications.

This paper is organized as follows. In the following section, we describe the design, implementation, and major features of the AMR framework. In Section 3, we describe the implementation of a solver for nonrelativistic MHD in this framework, including tests. In Section 4 we describe a relativistic MHD solver and tests. Throughout Section 5, we discuss other new physics modules under development, and in Section 6, we summarize and conclude.

2 Framework Design

As mentioned above, the most important design feature in Athena++ is the abstraction of the mesh from the physics. In this section, we describe the code framework that achieves this design.

2.1. The Mesh

The computational domain in an Athena++ calculation is a logically rectangular region whose overall properties are stored within a C++ class called the Mesh. The domain is further divided into a regular array of subvolumes whose properties are stored in another class called the MeshBlock. The latter stores discrete values for the dependent variables in cells as N-dimensional arrays, as well as one-dimensional arrays of coordinate positions along each direction. The number of cells stored in each MeshBlock, is arbitrary but it must be identical for all MeshBlocks. Similarly, the decomposition of the Mesh into MeshBlocks is arbitrary.

In both uniform mesh and AMR calculations, the logical relationship between MeshBlocks is encoded in a tree data structure, either a binary tree in one spatial dimension), a quadtree in two dimensions), or an octree in three dimensions). With AMR, the use of a tree is crucial for encoding the

[8] https: legion.stanford.edu overview index.html
[9] http: charm.cs.illinois.edu software

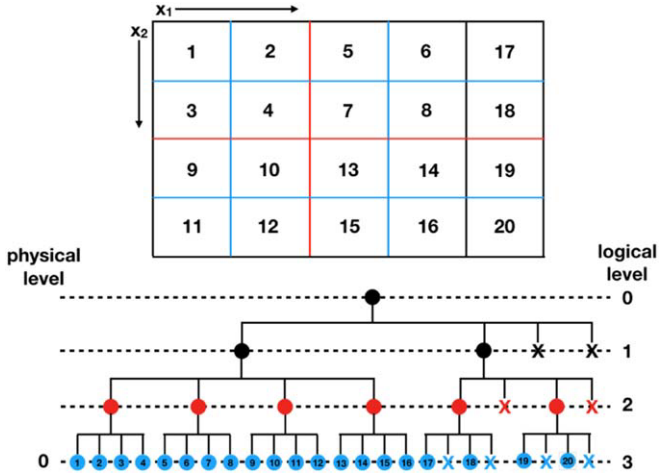

Figure 1 Labeling of `MeshBlocks` top) and their organization into a quadtree bottom) for an example uniform grid calculation in two dimensions.

relationship between parent and child `MeshBlocks`, and even with uniform grids it greatly simpli es nding neighboring blocks. Moreover, it results in the natural assignment of `MeshBlocks` to processors using Z-ordering, which helps improve locality and speeds up communications.

2.1.1. Uniform Grids

For uniform grid calculations, `MeshBlocks` are used to parallelize the calculation using domain decomposition. In this case, the total number of `MeshBlocks` used typically equals the total number of physical processors available although this is not required). For serial calculations on a uniform grid, only one `MeshBlock` is needed.

To construct the tree, the smallest value of n such that 2^n exceeds the largest number of `MeshBlocks` in any dimension is determined. Different levels n in the tree are referred to as logical levels. Thus, the tree is constructed beginning at logical level 0 and continuing to logical level n, and then each `MeshBlock` is assigned to the appropriate leaves at level n. Only in the case where the number of `MeshBlocks` in each dimension is equal and a power of 2 will every leaf in the tree be assigned a `MeshBlock`. In general, there will be both leaves and nodes that are empty.

To illustrate the process, Figure 1 diagrams the organization of a uniform grid into `MeshBlocks` and a quadtree for the speci c example of a two-dimensional calculation consisting of 5×4 `MeshBlocks`. In this case, the `MeshBlocks` are stored at logical level 3, and there are empty nodes and leaves at every logical level except the root, $n = 0$). Note that the physical level of the grid which corresponds to the re nement level in AMR) does not equal the logical level, and that the labels of the `MeshBlocks` are automatically organized into a Z-ordering across the domain this can be seen by connecting the labels shown in the top panel with a line). This ordering helps improve the locality of communications.

As in the `Athena` code, boundary conditions for the dependent variables stored on each `MeshBlock` are applied through the use of ghost zones. The ghost region consists of an extra N_G row of cells added to each array at each boundary. Any number of ghost cells are allowed; however, for second-order spatial integration algorithms on a uniform mesh for MHD, $N_G = 2$, whereas for spatial orders up to four $N_G = 3$ for hydrodynamics and $N_G = 4$ for MHD. With AMR, N_G must be an even integer because the restriction step see Section 2.1.5) reduces the number of cells by a factor of 2. When a calculation contains multiple `MeshBlocks`, data in the ghost zones may overlap with active cells in adjacent `MeshBlocks`. In this case, the data must be swapped between `MeshBlocks`, either via MPI calls if the `MeshBlocks` are on different processors or via calls to `memcpy`) otherwise. Because `Athena++` supports cell-, face-, and edge-centered variables, the communication of data between `MeshBlocks` can become complicated.

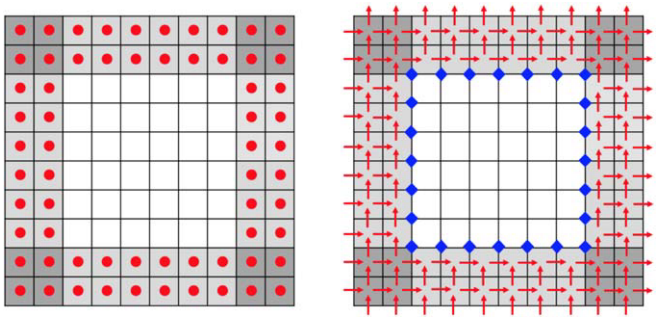

Figure 2 Left panel: example of cell-centered data red dots) that must be communicated to neighboring `MeshBlocks` in two dimensions. The shaded cells are ghost cells that overlap with active cells in the eight neighbors. Right panel: face-centered magnetic elds red arrows) and edge-centered EMFs blue dots) that are communicated in this example.

Figure 2 diagrams what data must be received from neighbor `MeshBlocks` for the speci c example of a two-dimensional calculation with `MeshBlocks` of size 6×6 with two ghost zones. White cells are "active cells," which are updated in each `MeshBlock`, while light gray cells are "face ghost cells" and dark gray cells are "edge ghost cells," which are lled by data from neighboring `MeshBlocks` that abut the faces and edges, respectively. In 3D, the algorithm also considers "corner ghost cells" corresponding to neighboring `MeshBlocks` that abut the corners. Note that the ghost cells overlap with as many as 8 neighbors in 2D 4 edges and 4 corners), and up to 26 neighbors in 3D 6 faces, 12 edges, and 8 corners). In `Athena++`, independent communication requests and message buffers are posted for each neighbor. This differs from the implementation in `Athena`, in which entire edges in each dimension were communicated sequentially. The sequential approach introduces dependencies in the third dimension, ghost cells cannot be communicated until those in the second are nished, which in turn requires those in the rst to be nished). We have found that such dependencies can reduce the parallel ef ciency on very large numbers of processors. On the other hand, the large number of communications required per `MeshBlock` with `Athena++` can tax some network architectures, thus an additional communication layer that pools messages between `MeshBlocks` may be a useful feature for future development, at least for some machines.

The left-hand panel in Figure 2 shows the cell-centered data that must be communicated for $N_G = 2$, while the right-hand panel shows the same for face-centered and edge-centered data associated with the CT algorithm for MHD used in `Athena++`. This algorithm requires storing area averages of the magnetic eld on cell faces, and computing line averages of the electric eld EMF) on cell edges see S08, Figure 1). Note that adjacent `MeshBlocks` share the same face-centered

vectors on their surfaces. In order to enforce the divergence-free constraint, both MeshBlocks must store and evolve the magnetic eld components on their surfaces. However, we have found that in some pathological cases, especially in curvilinear coordinates, round-off error can cause the values for the same magnetic eld component stored on different MeshBlocks to diverge in time. To prevent this, the EMFs computed at cell corners edges) in two dimensions three dimensions) are swapped between MeshBlocks, and the average of the values, computed independently on each MeshBlock, is used to update the magnetic elds on the surface. This adds an additional communication, but ensures consistency within the round-off error) between the eld on adjacent MeshBlocks.

While this discussion is motivated by the data associated with the MHD solvers in Athena++, in fact, the implementation of communication of ghost cells is highly modular and not specialized to any particular solver. Communication functions for arbitrary numbers of cell-centered, face-centered, and edge-centered data are provided in separate classes, derived from an abstract base class that implements generic MPI communication patterns. In turn, these functions can be enrolled using the task list when necessary.

2.1.2. Static and Adaptive Mesh Re nement

In the Athena++ implementation of AMR, in n dimensions, each MeshBlock is re ned into 2^n ner MeshBlocks, and the resulting MeshBlock structure is stored in a binary tree $n = 1$), quadtree $n = 2$), or octree $n = 3$). As one cell on a given level corresponds to 2^n cells on the next re ned level, the number of cells in a MeshBlock in each direction must be even. In addition, N_G must be even, and only re nement by a factor of 2 in each dimension simultaneously is allowed. A MeshBlock can contact neighboring MeshBlocks on the same level, one level coarser, or one level ner. Changes in resolution by more than one level at a boundary is not allowed, and this restriction affects which MeshBlocks are flagged for re nement or dere nement) in addition to the re nement criteria.

A driving feature for the tree design of the MeshBlock structure in Athena++ is AMR. Figure 3 shows how the 2D grid shown as an example in Figure 1 might be re ned with AMR. In the example, MeshBlocks 4, 7, 10, and 13 shown in Figure 1 have been re ned by up to two levels. This requires inserting additional logical levels corresponding to extra physical levels) at the appropriate leaves in the tree. Moreover, the labeling of all subsequent MeshBlocks beyond the rst re nement is modi ed. The 2D quadtree design is crucial for managing the logical structure of the MeshBlocks, as well as keeping the Z-ordering of labels. Note that in a parallel calculation, load balancing would be required see Section 2.1.6 below).

In this octree in 3D) block-based AMR design, the flexibility of the re nement depends on the size of MeshBlocks. If the root level is tiled with a large number of small MeshBlocks, then smaller volumes can be selected for re nement, reducing the computational work required. However, because each MeshBlock contains a xed number of ghost zones, the fraction of ghost cells compared to active cells is larger for smaller MeshBlocks. This surface-to-volume effect makes smaller MeshBlocks computationally less ef cient. Thus, the

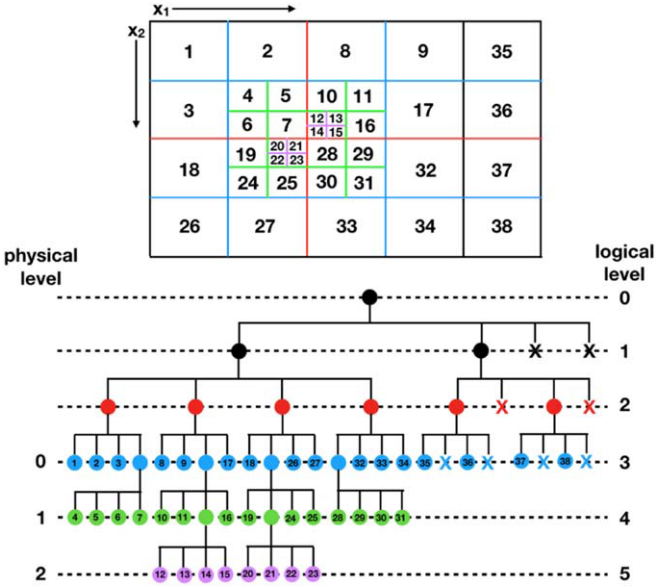

Figure 3 Same as Figure 1 with AMR.

best performance requires a careful choice of MeshBlock size in order to balance re nement flexibility requiring smaller MeshBlocks) and computational ef ciency requiring larger MeshBlocks); see Sections 3.6.4 and 3.6.5 for discussion. This is one possible disadvantage of tree block-based AMR. On the other hand, because each MeshBlock has the same logical shape in this design, it is easy to write optimized and flexible code that achieves high performance on modern parallel systems. This is one of the biggest advantages of the octree block-based AMR design.

2.1.3. Communication between Different Levels

The majority of the complexity with block-based AMR is associated with communications between MeshBlocks at different re nement levels. With AMR, each MeshBlock must communicate with up to 12 neighbors in 2D 4×2 faces and 4 edges), and up to 56 neighbors in 3D 6×4 faces, 12×2 edges, and 8 corners). When neighboring MeshBlocks are located on the coarser level, the data are rst restricted and then communicated at the lower resolution. This proceeds through "coarse buffers" that contain copies of the cell-centered and face-centered variables restricted to half the resolution, so that each cell in the coarse buffer including ghost zones) corresponds to 2^3 cells in 3D) in the MeshBlock.

To illustrate how boundary communications between different levels proceed, consider an example of a two-dimensional grid in which neighboring MeshBlocks at one face and one edge are at higher resolution. In the discussion that follows, MeshBlock A refers to the MeshBlock of interest, MeshBlocks B and C are neighbors along one face at higher resolution), and MeshBlock D is the neighbor at the lower-right edge at higher resolution; see Figure 4). For simplicity, suppose the MeshBlocks contain 6^2 cells and 2 ghost zones, i.e., $N_G = 2$. In the gure, red symbols indicate data points communicated between MeshBlocks A and B, while blue symbols indicate data communicated between MeshBlocks A and D.

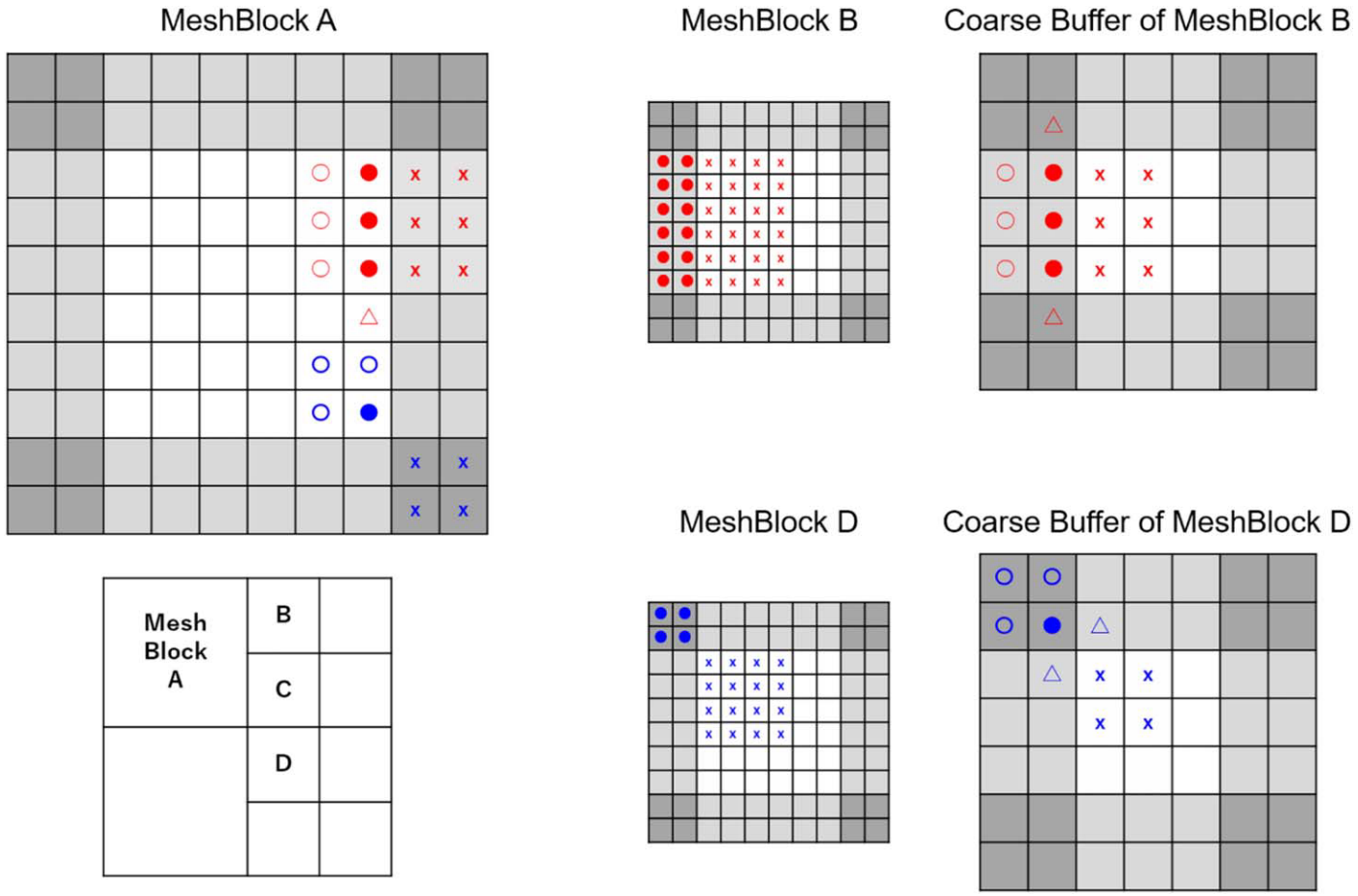

Figure 4 Example of communication of cell-centered data between neighboring `MeshBlocks` at different re nement levels. The lower-left panel shows the con guration of the `MeshBlocks` in a two-dimensional mesh. The upper panels show data communicated between `MeshBlocks` A and B using red symbols), while the upper-left and lower-right panels show data communicated between `MeshBlocks` A and D using blue symbols). See the text for a description of the symbols.

From the perspective of `MeshBlock` A, the communication procedure for cell-centered variables to and from ner `MeshBlocks` B and D proceeds as follows:

1. Send active cells overlapping the neighboring `MeshBlocks` marked by , •, and).
2. Receive ghost cells from neighboring `MeshBlocks` marked by ×).

On `MeshBlocks` B and D, the communication of cell-centered variables to and from the coarser `MeshBlock` A is more complicated:

1. Restrict the active cells overlapping `MeshBlock` A marked by ×) to the coarse buffer and send them.
2. Receive the coarse cells from `MeshBlock` A marked by , •, and) into the coarse buffer.
3. Wait until all boundary communications including both cell-centered and face-centered variables) are completed.
4. Fill in the cells adjacent to the cells to be prolongated marked by next to •). If these cells are on the same level as the `MeshBlock`, they must be restricted. If they are on the coarser level i.e., the same level as the coarse buffer), then they have already been received in the coarse buffer.
5. Apply physical boundary conditions on the coarse buffer if necessary).
6. Perform prolongation and store results into ghost zones overlapping `MeshBlock` A marked by •).

The restriction and prolongation algorithms are explained in Section 2.1.5. It is important to note that all the sends, receives, and restriction operations steps 1 and 2 in the above lists) are independent of each other, while the prolongation can only be performed after the arrival of all the boundary data. Because all of the communications are independent, the implementation of the algorithm using the task list is straightforward.

For face-centered variables, the communication procedure is slightly more complicated see Figure 5; note that the eld component perpendicular to the page in each cell is not shown —it can be transferred in the same way as cell-centered variables discussed above). On `MeshBlock` A, the communication procedure to and from ner `MeshBlocks` B and D proceeds as follows:

1. Send active faces overlapping the neighboring `MeshBlocks` marked by the , •, , , ▽, and ▼ symbols).
2. Receive ghost faces from neighboring `MeshBlocks` marked by × and).

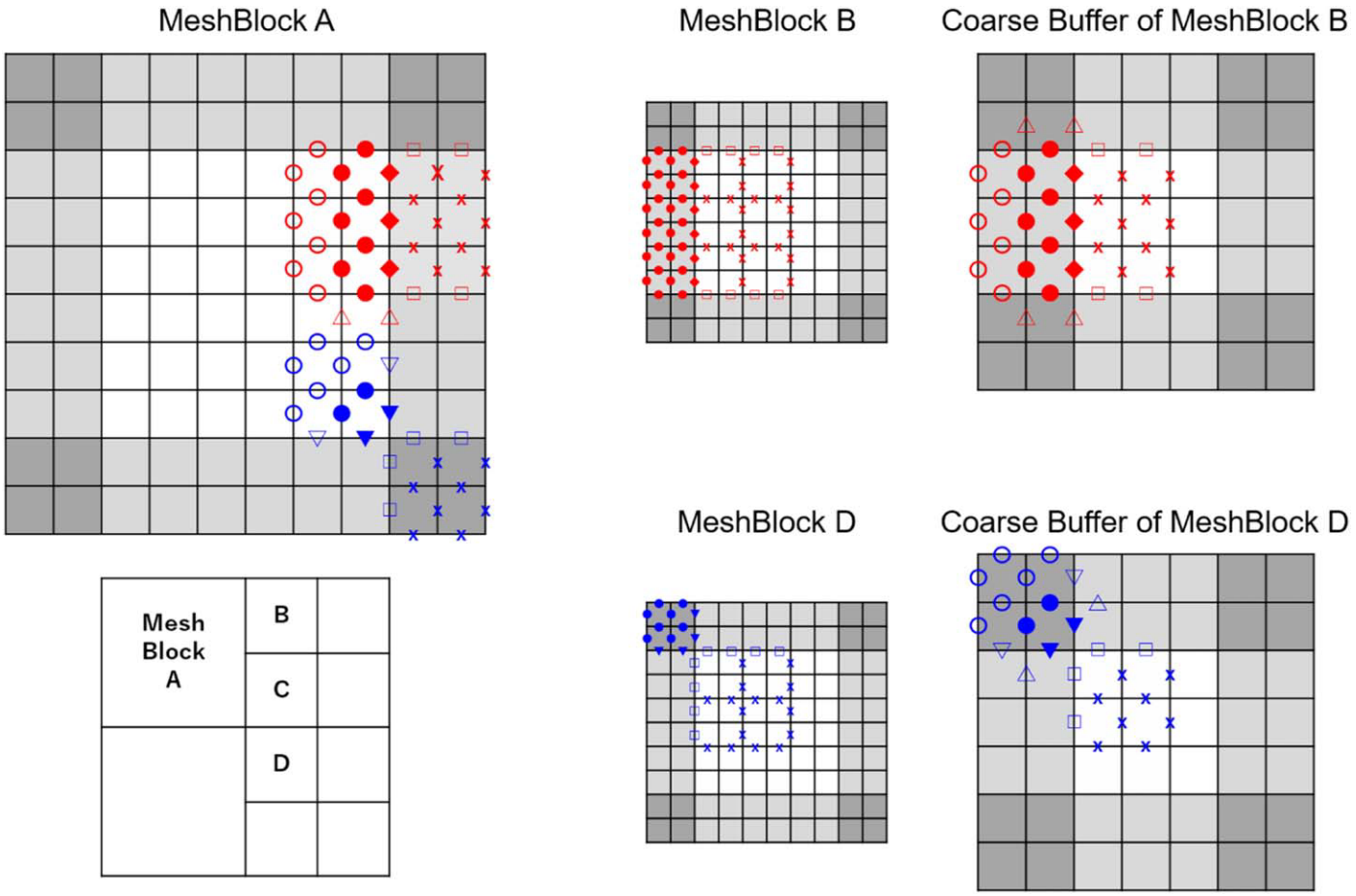

Figure 5 Same as Figure 4 but for face-centered variables.

Note that the faces marked with the symbols on MeshBlock A are active faces shared with the neighboring MeshBlocks, and they are not modi ed by boundary communications as this may cause a violation of the solenoidal constraint if the data on these faces represent the magnetic eld). Instead, these faces are sent to the ner MeshBlocks for prolongation. In addition, the faces marked with in the ghost zones are also shared by two MeshBlocks. Both MeshBlocks send these faces, and the values that arrive last are stored because as the restriction operation is conservative) the values should match even if one of the MeshBlocks is on the ner level. Any small differences between the values at the level of the round-off error) are prevented from growing via the flux and EMF correction steps see Section 2.1.4), and the error if any) will not lead to a violation of the solenoidal constraint because these values are only used during the reconstruction step and EMF calculation.

On MeshBlocks B and D, the exchange of face-centered variables to and from a coarser MeshBlock A proceeds as follows:

1. Restrict active faces overlapping MeshBlock A marked by × and) to the coarse buffer and send them.
2. Receive coarse faces from MeshBlock A marked by the , •, , , ∇, and ▼ symbols) into the coarse buffer.
3. Wait until all the boundary communications are completed.
4. Fill in the faces adjacent to the faces to be prolongated marked by next to ▽ and ▼). If these cells are on the same level as the MeshBlock, they have to be restricted. If they are on the coarser level i.e., the same level as the coarse buffer), then they have already been received in the coarse buffer.
5. Apply physical boundary conditions on the coarse buffer if necessary).
6. Perform prolongation and store the results into the ghost zones overlapping MeshBlock A marked by • and ▼).

Again, all the sends, receives, and restriction operations are independent of each other. Moreover, the communications for cell-centered and face-centered variables are mutually independent. As in the case of MeshBlock A, the faces marked with on MeshBlock B are active and are not modi ed, and only the faces marked with • are updated by the prolongation operation. On the other hand, the cells marked with ▼ on MeshBlock D are in the ghost zone and shared between two MeshBlocks. When both of the MeshBlocks sharing the same face are on the same level one level coarser than MeshBlock D), the prolongated values ▼ on the horizontal line in this example) are used. If one of them is on the ner level same as MeshBlock D), the values from the ner MeshBlock are used because the prolongated values are less accurate ▼ on the vertical line).

The communication between MeshBlocks on different levels at the corners in 3D is analogous to the above descriptions.

2.1.4. Flux and EMF Correction

In MHD calculations with static mesh re nement SMR) and or AMR, the area integral of the fluxes of the conserved

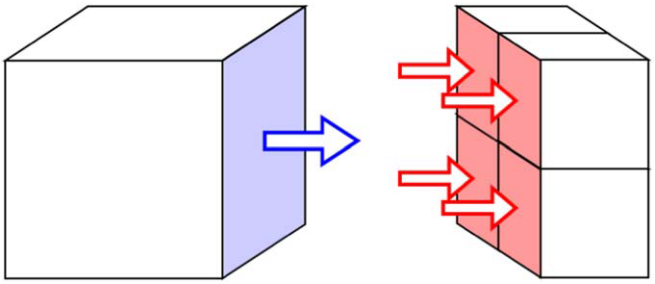

Figure 6 Flux correction on cell faces between neighboring MeshBlocks at different re nement levels in 3D. The area-integrated flux on the face of a coarse cell blue) is replaced by the area-integrated fluxes on the corresponding faces of the ne cells red).

variables on cell faces at the boundaries between MeshBlocks on different levels as well as the line integral of the EMF along cell edges) must be exactly equal. This requires a special step to correct the coarse cell fluxes with the generally more accurate) integral of the ne cell fluxes Berger & Colella 1989). The implementation of this correction procedure in Athena++ is described below.

For the face-centered fluxes of the cell-centered conserved variables, this flux-correction step is straightforward see Figure 6). In 3D calculations at the interface between different levels, one coarse cell abuts four ne cells two cells in 2D, and one in 1D). The flux used to update the coarse cell on the face that overlaps with the ne cells is simply replaced with the area-weighted sum of the fluxes from these four ne cells. The step makes use of the communication strategy outlined in the previous section for face-centered data.

For the edge-centered EMFs needed for the CT algorithm for MHD, this flux-correction step is considerably more complicated. Because the CT schemes preserve the divergence-free constraint to machine precision, it is crucial that the EMFs used to update the eld on overlapping cell edges at different levels be identical; otherwise, the magnetic flux at the faces of the cells will be inconsistent, and the resulting divergence error can grow and cause unphysical dynamics.

When MeshBlocks on different levels share the same face, the EMF on the coarse MeshBlock is replaced with the line-weighted sum over the corresponding ne edges see Figure 7):

$$\varepsilon_{\rm coarse}\,\Delta l_{\rm coarse} = \sum \varepsilon_{\rm fine}\,\Delta l_{\rm fine}. \tag{1}$$

Note that cell edges on the ne MeshBlock that have no corresponding edge on the coarse cell marked with in the gure) are not needed. With this correction, the line integral over the coarse cell edges will match those over the corresponding ne faces, which ensures consistent evolution of the magnetic eld on the shared face.

This procedure becomes more complicated when MeshBlocks on different levels share an edge rather than a face. Figure 8 shows some representative con gurations in this case. In order to satisfy the divergence-free constraint, the line integral of the EMF along the shared edges must match exactly. However, there is no guarantee this will be the case even for shared edges at the same re nement level due to different arrangements of the prolongation operations, nondeterministic ordering of the MPI communications, and round-off error that differs in the calculation of the same EMF on different MeshBlocks. Therefore, both ne and coarse EMFs must be corrected. First, the EMFs on the ne shared edges marked by

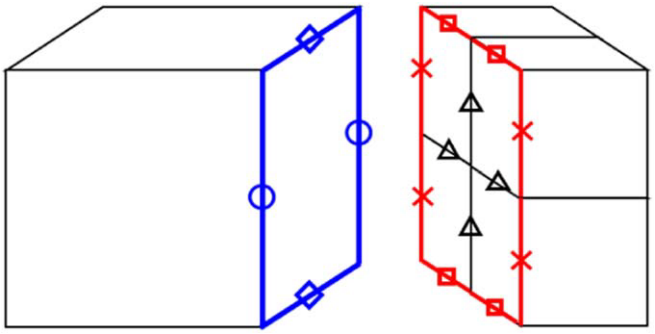

Figure 7 EMF correction on cell edges between neighboring MeshBlocks at different re nement levels in 3D. The line-integrated EMFs on the edges of the coarse cell blue) are replaced by the line-integrated EMFs on the corresponding edges of the ne cells red). Edges of ne cells that do not overlap any coarse cell edges marked by) are not used.

red and orange ×) are replaced with their average. Then, the EMFs on the coarse shared edges blue) are corrected using the EMFs on the ne edges so that the line integrals of the EMFs match as in Equation 1). The same procedure is applied to edges in the middle of a coarse MeshBlock that overlaps edges of ne MeshBlocks e.g., the edge shared by MeshBlocks B and C facing MeshBlock A in Figure 4).

Even without mesh re nement, numerical errors can cause a slight mismatch between the EMFs on shared edges between MeshBlocks. With the CT scheme, such errors never disappear once generated. This problem becomes more prominent when more complex grids with nonuniform mesh spacing and or curvilinear coordinates are in use. Moreover, aggressive compiler non-ANSI-conformant) optimizations can introduce and exacerbate differences associated with round-off errors. Therefore, the EMF correction step is applied even when mesh re nement is not used. In this case, the EMFs on two shared edges are replaced with the arithmetic average of their values.

2.1.5. Restriction and Prolongation Operators

For simulations with mesh re nement, data on ner MeshBlocks must be mapped onto overlapping cells on coarse MeshBlocks restriction) and vice versa prolongation). With our block-based AMR strategy, these interactions occur only at the boundaries between MeshBlocks on different levels, or when MeshBlocks are created or destroyed during re nement or dere nement.

When cell-centered variables are restricted, the volume-weighted average is used:

$$U_{\rm coarse} = \frac{\sum U_{\rm fine}\,\Delta V_{\rm fine}}{\Delta V_{\rm coarse}}, \tag{2}$$

where U denotes the variables being restricted for MHD the conserved variables are used) and V is the volume of the cells on the ne and coarse mesh. For face-centered variables, the area-weighted average is used for quantities de ned on the faces shared by MeshBlocks on the ne and coarse levels:

$$F_{\rm coarse} = \frac{\sum F_{\rm fine}\,\Delta S_{\rm fine}}{\Delta S_{\rm coarse}}. \tag{3}$$

Faces on the ner MeshBlock that do not coincide with faces on the coarser MeshBlock are not involved in the restriction step. With MHD, cell-centered magnetic elds and primitive

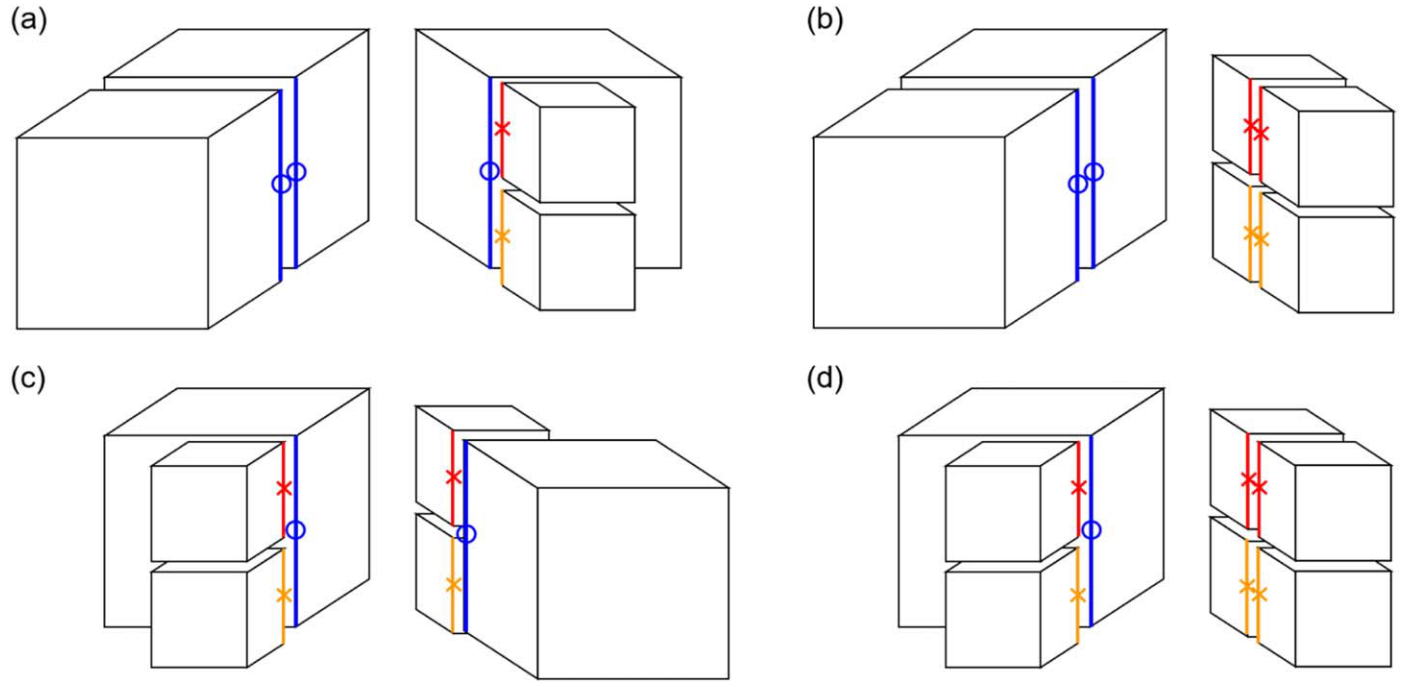

Figure 8 Examples of EMF corrections at the edges of cells between `MeshBlocks` on different re nement levels in various con gurations.

variables are calculated after both cell-centered conservative variables and face-centered elds have been restricted.

For prolongation of cell-centered variables, a multidimensional, slope-limited linear reconstruction is used. First, the gradients between neighboring cells in each direction are calculated and slope limiters are applied as in the reconstruction step of the hydrodynamic solver, which is discussed below in Section 3.2.1. Unlike the limiter used to compute the states at the faces for the Riemann solver, the less aggressive minmodslope limiter is used for prolongation. We have observed that using limiters that are sharper than minmodcan produce unphysical structures around the level interfaces, as reconstruction during prolongation involves a multidimensional pro le unlike the 1D reconstruction during hydrodynamic flux calculations). Then, the cell-centered variables are interpolated to the cell centers on the ner level. For example, to prolongate a cell at $k\ j\ i$),

$$\frac{\Delta_1 U_{i,j,k}}{\Delta x} = \mathrm{minmod}\left(\frac{U_{i,j,k} - U_{i-1,j,k}}{\Delta x_{i-1/2}}, \frac{U_{i+1,j,k} - U_{i,j,k}}{\Delta x_{i+1/2}}\right),$$
$$\frac{\Delta_2 U_{i,j,k}}{\Delta y} = \mathrm{minmod}\left(\frac{U_{i,j,k} - U_{i,j-1,k}}{\Delta y_{j-1/2}}, \frac{U_{i,j+1,k} - U_{i,j,k}}{\Delta y_{j+1/2}}\right),$$
$$\frac{\Delta_3 U_{i,j,k}}{\Delta z} = \mathrm{minmod}\left(\frac{U_{i,j,k} - U_{i,j,k-1}}{\Delta z_{k-1/2}}, \frac{U_{i,j,k+1} - U_{i,j,k}}{\Delta z_{k+1/2}}\right), \quad (4)$$

$$U_{i\pm1/2,j\pm1/2,k\pm1/2} = U_{i,j,k} \pm \frac{\Delta_1 U_{i,j,k}}{\Delta x}\Delta x_{f\pm} \pm \frac{\Delta_2 U_{i,j,k}}{\Delta y}\Delta y_{f\pm} \pm \frac{\Delta_3 U_{i,j,k}}{\Delta z}\Delta z_{f\pm}, \quad (5)$$

where U de ned at the points with integer indexes are on the coarser level while those with half-integer indices are on the ner level, and $x_{f\pm}$, $y_{f\pm}$, and $z_{f\pm}$ are the distances between the volume-weighted cell centers of the coarse cell and right left ne cells in each direction. For the prolongation at interfaces between `MeshBlocks` on different levels, this prolongation operation is performed using the primitive variables because use of the conservative variables can produce negative pressure. This does not violate the conservation law because the values in the ghost zones are used only through the flux calculation, and conservation in the active zones is ensured by the flux-correction procedure. As the communications between `MeshBlocks` use the conservative variables, they are converted into primitive variables, prolongated, and then converted back to the conservative variables after the prolongation. On the other hand, the conservative variables are used when new `MeshBlocks` are created by mesh re nement in order to satisfy the conservation law. A pressure floor is applied if negative pressures appear in the re ned cells. While the pressure floor violates conservation of the total energy, this method still satis es conservation of mass and momentum.

For prolongation of face-centered variables, the method of Tóth & Roe 2002) is adopted, which preserves the divergence of the face-centered elds. First, 2D interpolation on each coarse face is performed with the minmodslope limiter to the corresponding ne faces. When ne faces already have values at the ne level e.g., on `MeshBlock` B in Figure 5), they are not overwritten by the prolongated values; the ne face values are used instead. To determine the eld on internal faces on the ne mesh, the method adopted by Tóth & Roe 2002) is adopted, which assumes that the divergence of each ne cell matches the coarse cell which is zero), while the curl computed at internal edges matches that estimated using the coarse-level elds. As pointed out in the original paper, enforcing the curl of the eld currents) to match between levels is an assumption; nevertheless, it seems to work well. While this method was originally designed for uniformly spaced Cartesian grids, it is straightforward to extend it to nonuniform mesh spacing and curvilinear coordinates in the

" nite area" fashion. For further details, see Tóth & Roe 2002).

2.1.6. Load Balancing

In parallel simulations, it is important to keep the computational load balanced among the independent computing elements. By default, Athena++ distributes MeshBlocks among computing elements as evenly as possible, assuming each MeshBlock incurs the same computational expense. While this works quite satisfactorily for the hydrodynamic and MHD solvers, calculations involving additional physics can incur uneven computational cost. For example, chemical reactions updated using an iterative solver may require different numbers of iterations on different MeshBlocks. Moreover, when particles such as passive tracers or sink particles are used, they may concentrate in a speci c region and increase the load imbalance.

In order to provide more flexible load balancing, each MeshBlock is given its own "cost" parameter and Athena++ attempts to redistribute MeshBlocks so that the total cost per process is as even as possible. This cost can be manually set by users or automatically determined by the code using measurements of the compute time on each MeshBlock gathered from system-timing calls. This load balancing is performed periodically and whenever MeshBlocks are newly created or destroyed. The implementation does have several limitations. First, because a MeshBlock is a unit of both domain decomposition and load balancing, more than one MeshBlock per process is required to adjust the load balance. Second, because the ordering of MeshBlocks cannot be shuffled in the current implementation, certain pathological cases in which the load changes dramatically from one MeshBlock to another can be hard to distribute evenly. Although more complex load-balancing strategies are possible, they lack the simplicity and ease of use of the method implemented in Athena++.

2.1.7. Time Stepping with AMR

If the maximum signal speed in an MHD calculation $|v| + C_f$, where v is the fluid velocity and C_f the fast magnetosonic speed) on an AMR mesh is the same on all levels, then the maximum stable time step used to integrate each level will be proportional to the spatial resolution used at each level. Thus, standard adaptive time stepping can be used, in which each level l uses a time step that is 2^l smaller than that used at the root level $l = 0$). Such algorithms require interpolation in both time and space at ne coarse boundaries to enforce flux conservation e.g., see Mignone et al. 2012).

In Athena++, we do not use adaptive time stepping, but instead adopt the same xed time step to integrate all levels. There are several reasons for this choice. First, in many MHD applications the maximum signal speed is not constant across all levels. In fact, it is often the case that the highest speeds and therefore smallest stable time steps) occur on the root level, where densities may be small and the Alfvén speed large. In this case, by requiring smaller time steps than necessary at the highest re ned levels, adaptive time stepping makes the calculation more expensive. Second, the temporal interpolation required by adaptive time stepping introduces additional error to the solution, especially when self-gravity is included. Finally, the complexity of adaptive time stepping makes the overall calculation more dif cult to optimize and load balance; moreover, the cost savings in many cases is not substantial. For example, if an equal number of cells are being updated at each level which implies in 3D that roughly 10 of the volume of the domain is re ned at each level), then the reduction in the number of cell updates required is only about N 2, where N is the number of levels. Unless N is large, these savings may be offset by the reduced ef ciency of the method on highly parallel systems, making the overall reduction in the amount of CPU time required even smaller. Moreover, the reduction in work will not decrease the minimum possible wall clock time, which is bounded by the number of time steps needed to update the solution on the nest level.

Recently, several authors have explored the use of variable time stepping both across AMR levels and even within MeshBlocks at a given level Gnedin et al. 2018; Nordlund et al. 2018). Tests indicate speed-ups of about an order of magnitude are possible, as well as an increase in accuracy due to the ability to run at close to the maximum stable time step everywhere. Adaptive and or variable time stepping may be advantageous for very deep AMR hierarchies, or when a very small fraction of the volume is re ned, or when the time step varies dramatically within different regions at the same level. Extending Athena++ to enable such capabilities is a topic for future investigation.

2.2. Comparison to Other AMR Codes

The discussion in the previous sections has focused on the speci c implementation of AMR in the Athena++ framework. It is instructive to compare the algorithms we have adopted with those used in other codes.

There are three commonly used algorithms for AMR. The rst is cell-by-cell re nement, as adopted in codes such as RAMSES Teyssier 2002) and ART Kravtsov et al. 1997), in which each individual cell can be re ned independently. The second is patch-based AMR in which re ned regions of arbitrary size and shape can be created to cover areas of interest, following the original algorithm of Berger & Oliger 1984) and Berger & Colella 1989). This method is perhaps the most popular and is implemented in a variety of codes including Enzo Bryan et al. 2014), PLUTO Mignone et al. 2012), and AMRVAC Keppens et al. 2003). Moreover, sophisticated libraries that implement patch-based AMR for general systems of equations, including Chombo[10] and AMReX Zhang et al. 2019), are available. Finally, the third algorithm is block-based AMR in which re nement can occur only in xed locations using blocks of xed size. This is the algorithm adopted in Athena++ and described in detail above. Other codes that adopt this approach include FLASH Fryxell et al. 2000; which uses an AMR framework implemented in the PARAMESH library MacNeice et al. 2000)), the most recent version of NIRVANA Ziegler 2008), and DISPATCH Nordlund et al. 2018). Another important ingredient to the algorithm is the time-stepping strategy. Many implementations of AMR use adaptive time stepping, in which different levels are integrated at different time steps. As discussed in Section 2.1.7, in Athena++ we use a single global time step, which is the same for all levels. This makes our approach less ef cient when the grid contains a large number of levels more than 10) that cover a small fraction of the volume 1 or less).

[10] Astrophysics Source Code Library, record ascl:1202.008.

Several authors have explored the parallel ef ciency of the particular implementation of AMR algorithms in speci c codes e.g., Keppens et al. 2003; Ziegler 2008). In Sections 3 and 4, we present similar tests of the ef ciency of the AMR algorithms in Athena++.

In fact, the determination of which of the above three approaches for AMR is most ef cient is highly application dependent. The cell-by-cell and patch-based strategies can adapt the mesh to features in the flow more ef ciently than the block-based AMR adopted here, mostly because in the latter case re nement can only occur in the prede ned locations of MeshBlocks e.g., see Figure 3). On the other hand, block-based AMR is easier to implement, and therefore easier to optimize on modern highly parallel computing architectures. For example, Nordlund et al. 2018, Figure 4) show more than an order of magnitude improvement in ef ciency using the block-based approach in the DISPATCH code compared to cell by cell as in RAMSES on one test. A comprehensive investigation of the relative merits of each AMR strategy for various applications of interest, including performance and scaling on highly parallel systems, would be extremely instructive, but it is beyond the scope of this paper.

2.3. Coordinate Systems

Up to this point, the AMR framework in Athena++ has been described without reference to any particular geometry or coordinate system. Instead, all of the functionality is implemented for logically rectangular arrays of cells. In principle, this enables the code to be used in any coordinate system.

In practice, grid cells stored on the MeshBlocks may have nonuniform spacing, that is, the spatial size of the cells may be a smooth function of position in each dimension independently. Options to create both uniform and logarithmically spaced cells are provided as built-in features, and there is a simple mechanism to create custom cell spacing from a user-de ned input function. The physical size, areas, and volumes of cells are constructed and stored in the Coordinates class. These values are then used whenever needed to construct vector and tensor operators in the speci c coordinates. Currently, Athena++ has built-in support for Cartesian, cylindrical, and spherical-polar coordinates for nonrelativistic calculations, as well as those using SR. General relativity GR) capabilities support optimized Minkowksi, Schwarzschild, and spherical Kerr–Schild coordinates, as well as any stationary coordinates speci ed via metric coef cients by the user. It is straightforward to add new coordinate systems to the code.

Some coordinates systems for example, spherical-polar) introduce coordinate singularities that require special care. We have implemented "polar" boundary conditions on the pole in spherical-polar and spherical-like coordinates. For this boundary condition, the cell-centered and face-centered variables in the ghost cells are copied from the other side of the pole considering physical symmetry across the pole. The flux on a face contacting the pole does not have any influence on the active zone because the surface area of the face is zero. On the other hand, the EMFs on the radial edges contacting the pole are replaced with their average because they must have the same value. The robustness of this boundary condition is demonstrated in Section 3.5.2.

2.4. Hybrid Parallelization Strategy

Distributed-memory parallelism through domain decomposition is an integral part of the design of the AMR framework in Athena++ and has been discussed extensively in the preceding sections. On some architectures, it is also advantageous to employ shared-memory parallelism based on, e.g., the OpenMP standard. Because of the signi cant overhead of launching and terminating threads, we have found that a ne-grained approach to shared-memory parallelism in which parallel regions are forked and joined at the for-loop level is not very ef cient. In addition, this approach requires signi cant effort to identify and parallelize every region in the code. Instead, we have found that a coarse-grained approach, in which each MPI rank possesses multiple MeshBlocks that are updated by individual OpenMP threads, is more ef cient. This design does require a thread-safe implementation of the MPI library with support for MPI_THREAD_MULTIPLE, which is the fourth and highest level of thread safety de ned in MPI. MPI implementations are not required to support this functionality although most major distributions offer at least partial support), and occasionally, users have discovered that the compiled MPI library on their shared cluster was con gured with this thread safety disabled.

2.5. Dynamic Scheduling via the Task List

One of the most important capabilities of the Athena++ AMR framework is the dynamic execution of tasks. Similar ideas have been implemented in other codes such as DISPATCH Nordlund et al. 2018), and libraries such as CHARM++ and Legion enable task-based parallelism along with many other features). We have implemented our own design for task-based dynamic execution in Athena++, which we describe in detail in this section.

Dynamic execution is implemented in a class called the TaskList. Rather than hard-coding the order of execution of functions associated with a physics module, all of the steps in the algorithm are assembled into an array of Task structures. Each Task structure contains a unique task_id, a dependency encoding of other tasks that must be nished before the current task can be executed, and a pointer to a function that implements the actual work associated with the task. The task_id and dependency are implemented as bit elds of arbitrary length, and each task_id has a different) single bit set to 1. Each MeshBlock owns a task_state to store which tasks are completed, which is also implemented as a bit eld.

The key to this implementation is controlling dependencies between Tasks. There are two types of dependencies: the rst is an internal dependency between Tasks within a single MeshBlock, and the second is an external dependency between different MeshBlocks. The internal dependency controls the ordering of Tasks, and it is implemented using the dependency flag in the Task structure. The external dependency controls coherency between MeshBlocks associated with boundary communications, and the return value of a Task function implements this control flow.

A flow chart demonstrating how the TaskList is processed is shown in Figure 9. Execution begins with selection of the rst available Task from the TaskList and a check of its internal dependency implemented with bitwise operations for ef ciency). If the dependency is not

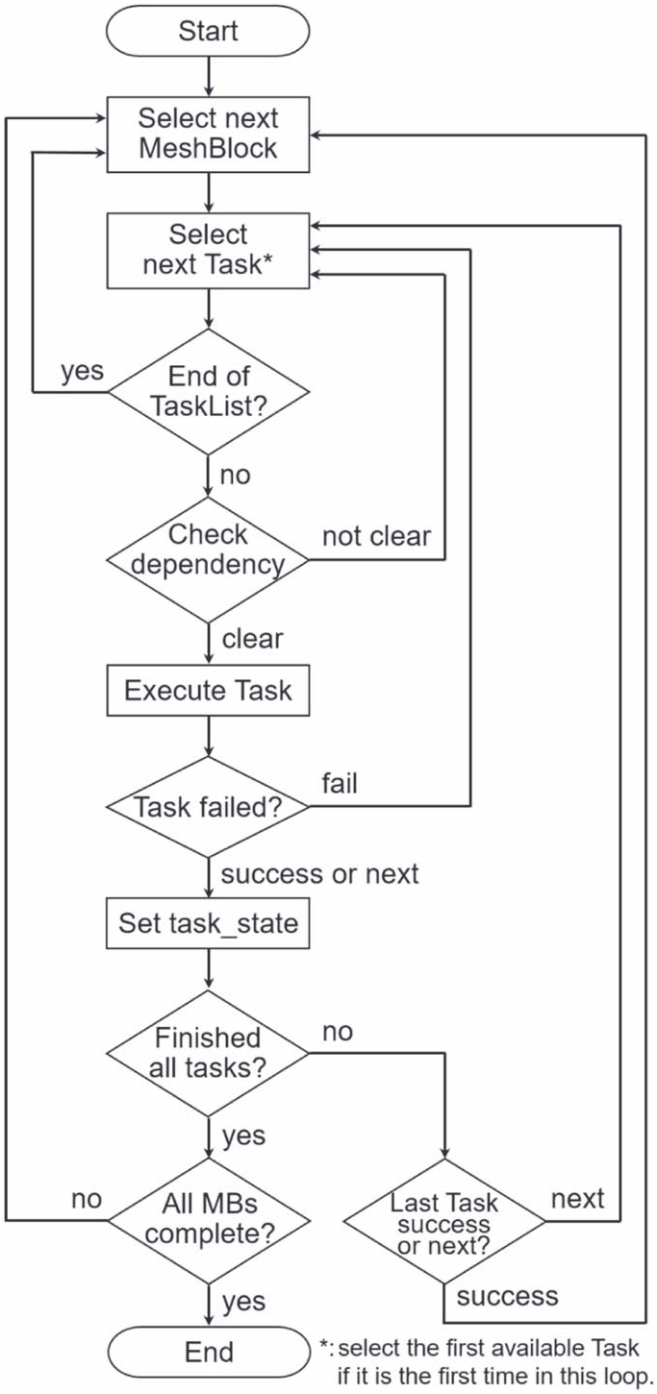

Figure 9 A flow chart of dynamic execution using the `TaskList`. For details, see the discussion in the text.

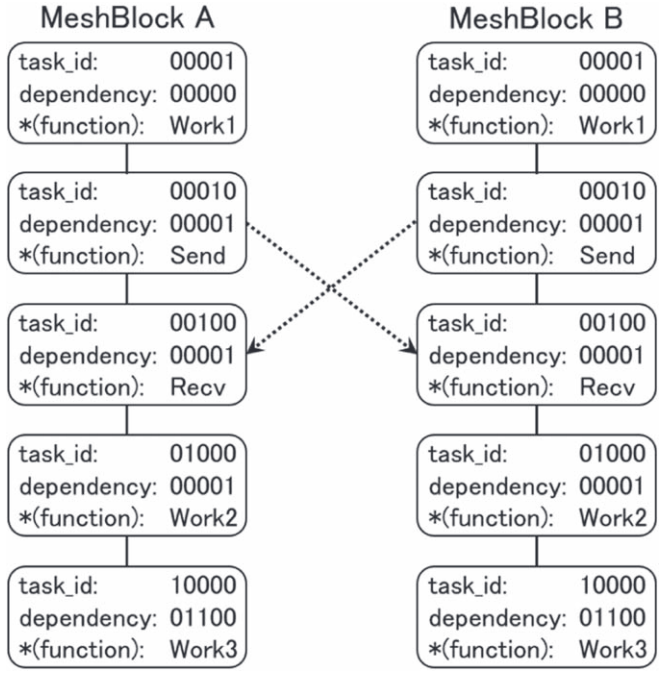

Figure 10 Example of a ve-step `TaskList` executed on two `MeshBlocks`. The dotted arrows indicate communications between `MeshBlocks`. For details, see the discussion in the text.

cleared, the `Task` is skipped. If there is no dependency, the `Task` function is executed. A `Task` function returns one of three possible results: `success`, `next`, or `fail`. When either `success` or `next` is returned, the `Task` is marked as completed, and its `task_id` is stored in the `task_state` by a bitwise `OR` operation. When the return value is `success`, the code begins processing another `MeshBlock` if any), whereas when `next` is returned, the subsequent `Task` on the same `MeshBlock` is processed. This is used when the ensuing `Task` should be executed immediately, for example if it involves sending boundary communications. When a `Task` function returns `fail`, which typically happens when the function is waiting for MPI communications but one or more messages have not arrived, the `task_state` is not updated and the next `Task` on the same `MeshBlock` is processed. This procedure is repeated until all the `Tasks` in all the `TaskLists` are completed.

To illustrate these concepts further, Figure 10 illustrates an example of two `MeshBlocks` with very simple ve-step `TaskLists`. These `MeshBlocks` can be either on the same process or on different processes. Before starting the `TaskList`, nonblocking MPI receive operations are initiated. When `TaskList` execution begins, the `Work1` function referenced in the rst `Task` structure would be called, and provided it completes successfully, it will be marked as complete and its `task_id` is stored in the `task_state` of the `MeshBlock`. Next, the second `Task` consisting of boundary communications would be executed, as its dependency on the rst `Task` is already cleared. These communications are performed by a standard library `memcpy`) function call if the neighbor `MeshBlock` is on the same process and by nonblocking MPI send operations if it is on a different process. Control will then pass to the third `Task` in the list. This `Task` does not depend on the second `Task` but only on the rst `Task`, which is already cleared. However, this `Task` also has external dependency on boundary communications from the other `MeshBlock`. This `Task` checks completion of the boundary communications using the `MPI_Test` functions if the neighbor is owned by another process, and returns `fail` if the messages have not been delivered yet. In this case, the `Task` is not flagged to be completed, and the next `Task` in the `TaskList` is processed. As the fourth `Task` depends only on the rst, this `Task` is executed even if the third `Task` is not completed. The fth `Task` is then processed, but because it depends on both the third and fourth `Tasks`, it cannot be executed until those dependencies are cleared. As the execution

has now reached the end of the TaskList, control returns to the top of the list and repeats this loop until all of the Tasks are completed.

There are three important reasons why we have found the TaskList to be such a useful design. The rst is that it enables communication to be hidden behind computation. In the example given in Figure 10, this is possible because the algorithm contains work the fourth Task) that does not depend on the completion of some prior communication. Even if this is not the case, by having multiple MeshBlocks on a processor, the communication required by the rst and subsequent MeshBlocks can be hidden by the work required at the start of other MeshBlocks. We have found that this feature improves the scaling ef ciency of Athena++ on very large numbers millions) of cores.

A second important advantage is that the TaskList provides tremendous flexibility and modularity in incorporating different combinations of physics modules. In the previous version of the code, different physics algorithms were hard-coded into the main loop and conditionally executed based on a set of nested preprocessor flags. Coding every possible combination of modules in this manner became burdensome. With the TaskList in Athena++, physics modules are included at runtime by adding the appropriate steps to the list. Calculations do not even have to include the MHD modules in order to run. It is possible to build task lists that simply execute chemistry or radiation transfer modules in a test or postprocessing mode. This makes the code extremely flexible. Even different numerical algorithms such as higher-order time integrators see Section 3.2.3 below) can be constructed simply by encoding them into the task list, rather than hard-coding special purpose functions.

The third advantage of the TaskList is that different MeshBlocks can operate with independent TaskLists and are therefore able to model different physics. This enables heterogeneous computation in which, for example, some processes solve MHD equations while others solve self-gravity. Heterogeneous parallelization can improve the overall scalability of the code by allocating fewer distributed computing processes for algorithms e.g., self-gravity) that scale less well. It is even possible to solve different physical models on different MeshBlocks. For example, chemistry or nuclear reaction networks might only be included in certain regions of the flow where they are important, or the general-relativistic MHD equations might be solved only on MeshBlocks near a compact object, while the much less complex) nonrelativistic MHD equations are solved everywhere else. A nal example is that MeshBlocks in regions of very low density may use hybrid PIC methods to properly capture kinetic physics, while MeshBlocks in denser regions solve the kinetic MHD equations Garcia et al. 1999). We will report the usage of the TaskList in such multiphysics applications in the future.

2.6. Software Design Principles

Athena++ is free-and-open-source software. Stable, public releases of the code are hosted on a public GitHub repository[11]; however, development primarily occurs on a private GitHub repository. Thus, the engineering of Athena++ is not based on a true open development model, although bug reporting, issue tracking, and contributions from the user community are welcomed. Documentation and tutorials are provided on the public GitHub Wiki. The software is licensed under the permissive three-clause Berkeley Software Distribution BSD-3) license, chosen because it has more relaxed rules for redistribution of derivative works than, e.g., a copyleft license such as GNU General Public License Version 3 GPLv3). This can be an important consideration when integrating Athena++ with closed-source software, for example, frameworks developed at national laboratories.

In order to reduce the barriers to entry for using the code, and to maximize the portability of the software from personal laptops to leadership-class supercomputers and cloud-based containers), Athena++ was designed with the smallest number of dependencies possible. Only a C++ compiler and a Python distribution versions 2.7+ and 3.4+ both supported) are required in the default con guration. Strict adherence to the C++11 standard is enforced in the source code to ensure compatibility with most modern compilers. More recent standards are not adopted until all major compilers support new features; to this end, migration to the C++14 standard is underway. To deploy Athena++ in parallel, an OpenMP-enabled compiler and or an MPI library is required. Additional optional functionalities may require linking the solver with compatible FFTw3 and or HDF5 libraries, although we are working hard to eliminate the latter dependency in the future. The code has been developed by a core team consisting of the coauthors, with substantial commits from more than a dozen other contributors. The rst Athena++ Developers Meeting and Users Workshop was held in 2019, with 63 attendees and speakers.[12]

The decision to not follow an open development model is driven by several factors. Managing an open development project including quality control) is more time consuming and burdensome; the primary focus of the core developers is science applications rather than supporting software development. Moreover, some algorithmic features take years of development and testing before they are generally useful, and granting open access too early seems counterproductive. Athena++ s current development model strikes a balance between centralizing control over the code s development while also encouraging the dozens of unique clones of the public version that occur per week. However, there are bene ts to the open development model Turk 2013), both for accelerating development of new features and for cultivating a more productive relationship with a self-sustaining community of user-developers who provide valuable contributions. For this reason, we are actively exploring the reorganization of the Athena++ AMR framework and physics modules into separate development repositories. Because almost all of the factors that drive a private development repository are related to the physics solvers, this would allow the AMR framework to become truly open development. Moreover, this would enable others to build their own physics solvers on top of the AMR capabilities developed for Athena++.

An important argument in favor of open development models is reproducibility; science applications that use a private development version cannot be easily rerun by the community. However, the ability to reproduce results simply by running the same calculations using the same code does not guarantee those results are correct. True reproducibility requires results to be

[11] https: github.com PrincetonUniversity athena-public-version

[12] http: www.physics.unlv.edu astro athena2019 index.html

checked by an independent implementation of the same algorithms or, even more importantly, by running different algorithms as implemented in different codes to solve the same mathematical model. Open-source software and open development are useful instruments for supporting reproducibility, but they are not suf cient to guarantee it on their own Stodden & Miguez 2014). Nevertheless, we support such efforts by bundling input les and validation test scripts with the source code distribution. The analysis and plotting scripts used to produce many of the published results from Athena++ are also included; this is an increasingly popular best practice that many other projects have adopted for example Oishi et al. 2018).

Perhaps the most important ingredient for reproducibility is validation and veri cation. In this paper, and in S08, we provide a comprehensive series of test problems based on known analytic solutions and comparison of results computed by Athena++ with those from other codes see especially Section 3.3.6). Another crucial component for promoting computational reproducibility and manageability in a codebase the size of Athena++ is automated testing. A regression test suite written in Python is distributed with the source code. It consists of more than 60 separate tests ranging from simple compilation checks to multiphysics benchmark problems. Whenever possible, such tests involve comparison to analytic solutions such as linear wave convergence, or planar shock-tube problems) to avoid issues related to numerical precision. In addition, style checks and code linting of C++ and Python source are provided by Google s open-source cpplint.py static code checker and the Flake8 tool, respectively. Every pull request and change to the repository s main branch are automatically tested using continuous integration CI). A local Jenkins[13] server and the cloud-based Travis CI[14] service independently execute every available test. We have found that it is valuable to repeat the tests with multiple combinations of compilers, target architectures, and dependency library versions in order to catch subtle bugs that may only emerge in certain programming environments. Code coverage analysis is provided by GCC s gcov utility combined with the Linux Testing Project s graphical front-end lcov.[15] The testing regime currently achieves approximately 65 of C++ line coverage. The important role that CI and regression testing have played in the development of Athena++ cannot be overemphasized.

3 A Nonrelativistic MHD Solver

As we have previously highlighted, the AMR framework described in the preceding section can be used with any grid-based physics solver. In order to provide a concrete example of the most popular) use of the Athena++ AMR framework, in this section we describe the implementation of a module to solve the equations of nonrelativistic hydrodynamics and MHD.

The underlying algorithms implemented in this module are nearly identical to those used in the original C version of Athena, and are described in detail in S08. Therefore, we only provide an overview of the method in this section with particular focus on any changes we have made in reimplementing the methods in Athena++.

[13] https: jenkins.io
[14] https: travis-ci.org
[15] http: ltp.sourceforge.net coverage lcov.php

3.1. Equations and Discretization

The module solves the equations of nonideal MHD:

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho \boldsymbol{v}) = 0, \tag{6a}$$

$$\frac{\partial \rho \boldsymbol{v}}{\partial t} + \nabla\cdot(\rho \boldsymbol{v}\boldsymbol{v} - \boldsymbol{B}\boldsymbol{B} + \mathsf{P}^* + \boldsymbol{\Pi}) = 0, \tag{6b}$$

$$\frac{\partial E}{\partial t} + \nabla\cdot\Bigg[(E + P^*)\boldsymbol{v} - \boldsymbol{B}(\boldsymbol{B}\cdot\boldsymbol{v}) + \boldsymbol{\Pi}\cdot\boldsymbol{v} + \eta \boldsymbol{J}\times\boldsymbol{B} + \frac{\eta_{\rm AD}}{|\boldsymbol{B}|^2}\{\boldsymbol{B}\times(\boldsymbol{J}\times\boldsymbol{B})\}\times\boldsymbol{B} + \boldsymbol{Q}\Bigg] = 0, \tag{6c}$$

$$\frac{\partial \boldsymbol{B}}{\partial t} - \boldsymbol{\nabla}\times\Bigg[(\boldsymbol{v}\times\boldsymbol{B}) - \eta\boldsymbol{J} - \frac{\eta_{\rm AD}}{|\boldsymbol{B}|^2}\boldsymbol{B}\times(\boldsymbol{J}\times\boldsymbol{B})\Bigg] = 0, \tag{6d}$$

where P^* is a diagonal tensor with components $P^* = P + B^2/2$ with P being the gas pressure), $\boldsymbol{\Pi}$ is the viscous stress tensor,

$$\Pi_{ij} = \rho\nu\left(\frac{\partial v_i}{\partial x_j} + \frac{\partial v_j}{\partial x_j} - \frac{2}{3}\delta_{ij}\nabla\cdot\boldsymbol{v}\right), \tag{7}$$

and is the coef cient of kinematic viscosity. E is the total energy density

$$E = e + \frac{1}{2}\rho v^2 + \frac{B^2}{2}, \tag{8}$$

with e as the internal energy density; $\boldsymbol{Q}$ is the heat flux,

$$\boldsymbol{Q} = \kappa\nabla T, \tag{9}$$

with thermal conductivity κ and temperature T; and $= \nabla\times B$ is the current density. These equations are written in units such that the magnetic permeability $\mu = 1$.

These equations include terms for isotropic viscosity and thermal conduction, as well as ohmic resistivity and ambipolar diffusion in the strong coupling limit. The coef cients of kinematic viscosity , thermal conductivity κ, and ohmic resistivity η are constants by default; however, it is straightforward to extend them to be functions of position and the dynamical variables. There is no single form for the conductivity $\eta_{\rm AD}$ needed with ambipolar diffusion as this depends on the ionization, recombination, and collision rates in the plasma. Therefore, no default form is provided. Instead, a function to compute $\eta_{\rm AD}$ must be implemented as part of the calculation, and a simple mechanism is provided to users in order to do this.

An equation of state EOS) is needed to compute the pressure P and temperature T from the total energy and other conserved quantities. In Athena++, any general EOS can be used. This includes both an ideal gas law in which case $P = \gamma 1)e$) or a barotropic EOS for example, isothermal, in which case $P = c_s^2\rho$, where c_s is the isothermal sound speed). Any general EOS that provides $P = P\ \rho, e)$ and $a^2 = a^2\ \rho, p)$ where a is the sound speed), either as an analytic function or through interpolation of tabular data, can be used. A complete description of the implementation of the general EOS

functionality in Athena++ is provided in Coleman 2020). This functionality is validated using tests from Chen et al. 2019).

Equations 6 a) through 6 c) are discretized using a nite-volume approach, with the cell-averaged conserved variables stored at the volume centers of cells. Note that in curvilinear coordinates, it is important to distinguish volume centers from geometric centers, especially for algorithms with formal spatial accuracy higher than second order Blondin & Lufkin 1993). The induction Equation 6 d) is discretized using the upwind CT algorithm developed in Gardiner & Stone 2005, 2008), and therefore, the components of the magnetic eld are area averages stored at cell faces. See S08 Section 3) for details.

3.2. Numerical Algorithm

To provide robust and accurate shock capturing, the MHD module in Athena++ is based on a Godunov-type method. The major components of such algorithms for ideal hydrodynamics are 1) a method for the nonoscillatory spatial reconstruction of the fluid variables to compute interface states, 2) a Riemann solver to compute upwind fluxes and electric elds at cell faces, and 3) a time-integration algorithm to advance the solution. Each of these steps is described in subsections below.

In order to preserve the divergence-free constraint on the magnetic eld at every substep, a dimensionally unsplit algorithm is required. The most accurate unsplit algorithm used in Athena, the corner transport upwind CTU) method Colella 1990, described in detail in S08), requires a characteristic projection of the interface states during the reconstruction phase. For relativistic MHD, such projections are very complex, and for that reason, in Athena++ the CTU integrator is not used but instead simpler unsplit integration algorithms are adopted see Section 3.2.3). Of course, it would still be possible to implement the CTU algorithm in Athena++ provided its use is restricted to nonrelativistic MHD.

3.2.1. Spatial Reconstruction Methods

As in Athena, three different spatial reconstruction methods are implemented in the Athena++ MHD module: 1) a rst-order donor cell DC) method, 2) a second-order piecewise linear method PLM), and 3) a fourth-order piecewise parabolic method PPM).

Variable reconstruction is performed on either the primitive variables $\boldsymbol{W} = (\rho, \boldsymbol{v}, P, \boldsymbol{B})$ or for nonrelativistic, ideal EOS problems) on the characteristic variables $\boldsymbol{C} = \mathrm{L} \cdot \boldsymbol{W}$, where L is the matrix of left-eigenvectors of the system of equations see S08, Appendix A). The latter approach can help reduce oscillations in the solutions, especially for MHD problems, as we demonstrate in Section 3.3 below. However, the projection procedure is different from the approach used in Athena and described in S08 Section 4.2.2). The reader is referred to Felker & Stone 2018, Section 2.2.2) for a detailed description of the characteristic reconstruction steps used in Athena++.

There are several other important changes to the reconstruction algorithms implemented in Athena++ compared to those in the original version of Athena and described in Section 4.2 in S08. First, the characteristic tracing performed in step 7 of Section 4.2.2 and step 10 in Section 4.2.3 of S08 is no longer required because the CTU integrator is not implemented. Second, the reconstruction stencils and slope limiters are modi ed to ensure the reconstruction remains total variation diminishing TVD) with both nonuniform and curvilinear meshes. For PLM reconstruction, Athena++ uses the original van Leer limiter van Leer 1974) when the grid is uniformly spaced and there is no geometric factor e.g., uniform Cartesian grids and uniformly spaced direction in cylindrical spherical coordinates), and the modi ed van Leer limiter described in Mignone 2014) for nonuniform and or curvilinear meshes. The weights for the smooth reconstruction stencil are automatically modi ed for nonuniform and or curvilinear grids if the backwards and forwards difference approximations to the derivative are divided by the distance to the centroid of volume.

The PPM reconstruction algorithm in Athena++ has also been signi cantly modi ed to improve accuracy on curvilinear and nonuniform meshes. We again refer the reader to Felker & Stone 2018, Section 2.2.2) for a complete description of the ve PPM limiter formulations that were considered during the development of Athena++ and a summary of the errata in the original references for each limiter.

The primary PPM limiter is a smooth extrema-preserving limiter described in McCorquodale et al. 2015), which extends the work of Colella & Sekora 2008); it is used for all problems on Cartesian meshes in Athena++. For curvilinear grids, the steps of the original PPM limiter of Colella & Woodward 1984) are modi ed in Section 3.3 of Mignone 2014) to account for the difference between the geometric and volumetric centers of the cells.

For nonuniform, Cartesian-like grid directions, Equation 1.6) of the original PPM publication Colella & Woodward 1984) provides the reconstruction stencil for the initialization of the variable face averages at fourth-order spatial accuracy. For uniform grids, it reduces to the well-known weights of Colella & Woodward 1984, Equation 1.9)):

$$Q_{i-1/2} = \frac{7}{12}(Q_{i-1} + Q_i) - \frac{1}{12}(Q_{i-2} + Q_{i+1}). \qquad (10)$$

The procedure outlined in Mignone 2014, Section 2.2) is followed for computing the curvilinear counterparts to the weights in Equation 10) along the radial direction in spherical-polar and cylindrical coordinates, and along the meridional direction in spherical-polar coordinates.

By default, Athena++ uses a second-order accurate CT solver for MHD problems. With this con guration, the overall accuracy of the MHD solver remains formally $\mathcal{O}(\Delta x^2)$ even when a higher-order reconstruction method is employed. However, the use of higher-order algorithmic components often still signi cantly improves the accuracy of solutions see Section 3.3.1 for a demonstration). Extension to a fully fourth-order accurate scheme has already been implemented in Athena++ and published in Felker & Stone 2018).

3.2.2. Riemann Solvers

As in Athena, the HLLE, HLLC, and HLLD approximate Riemann solvers are implemented in Athena++, as well as Roe s linearized solver. We nd exact solvers do not provide any signi cant increase in accuracy for most problems although they may make the algorithms more robust on

problems involving strong rarefactions), so currently none are implemented.

The HLLE, HLLC, and HLLD solvers have been extended to be compatible with a general EOS. This requires the sound speed a be provided either as an analytic function, or through interpolation of tabular data. A complete description of the changes to these solvers for a general EOS is provided in Coleman 2020).

3.2.3. Time Integrators

The nal major component of the main MHD algorithm concerns the temporal evolution of the fluid variables. A method of lines formulation is adopted, in which the spatial discretization steps in Sections 3.2.1 and 3.2.2 provide an estimate of the flux divergence of the system of conservation equations at a single time t. When combined with a suitable method for integrating the time-dependent system of ordinary differential equations ODEs), a complete scheme with formal $\mathcal{O}(\Delta x^n, \Delta t^m)$ accuracy is constructed. It is important that dimensionally unsplit integrators are used for MHD so that the divergence-free constraint applies at every substep. In `Athena`, both the $\mathcal{O}(\Delta t^2)$ accurate van Leer VL2) predictor–corrector integrator described in Stone & Gardiner 2009) and the CTU method of Colella 1990) are implemented. However, as discussed earlier, the characteristic projection method required by the CTU integrator makes it dif cult to use for relativistic flows. Thus, in `Athena++`, the VL2 integrator is implemented along with several strong-stability preserving SSP) and or low-storage Runge–Kutta RK) methods.

In `Athena++`, the 2S class of low-storage RK methods discussed in Ketcheson 2010) is adopted. Let $u^{0)}$, $u^{1)}$ refer to the two registers in memory for storing the conserved fluid variables de ned across the mesh at different time abscissae within a single time step. We now describe our implementation of Algorithm 3 of Ketcheson 2010). The notation is modi ed to use zero-based indexing for the variable registers and the integrator stages, and the relative index of $\delta_i \equiv \delta_{j-1}$ increased by 1 from the original δ_j.

At every cycle, $u^{0)} = u^n$, $u^{1)} = 0$ is assigned before the rst stage of the integrator. While $u^{0)} = u^n$ is already implicitly guaranteed from the output of the 2S algorithm in the previous time step, these two-register integrators typically require explicit assignment operations in order to clear the cached data in $u^{1)}$. Then, for $s = 0$ $N_{\rm stages}$ 1:

$$u^{(1)} \leftarrow u^{(1)} + \delta_s u^{(0)}$$
$$u^{(0)} \leftarrow \gamma_{s0} u^{(0)} + \gamma_{s1} u^{(1)} + \beta_{s,s-1} \Delta t F(u^{(0)}), \tag{11}$$

where $u^{n+1} \equiv u^{(0)}$ after the nal stage in the cycle. In all cases, $\delta_0 = 1$ and $\gamma_{01} = 1$ as the rst stage is always a forward Euler step using data from the previous cycle.

A wide range of integrators of varying orders of accuracy, number of stages, and stability properties can be represented within this framework. For completeness, the coef cients of the most commonly used and simplest) selections available in `Athena++` are documented below. All of the following integrators are de ned with $\delta_i = 0$ for $i > 0$; however, the generality of Equation 11) enables the trivial implementation of more advanced limiters such as the non-SSP RK4)4 2S] see Ketcheson 2010, Table 2). Furthermore, it is straightforward to extend the framework to three-register 3S methods which are useful for high-order schemes Felker & Stone 2018).

The integrators available in `Athena++` are:

RK1: forward Euler method:

$$\gamma_0 = \{0, 1\} \quad \beta_{0,-1} = 1. \tag{12}$$

VL2: default) predictor–corrector midpoint method. The predictor step must always compute and apply diffusive rst-order accurate fluxes that are produced by DC reconstruction:

$$\gamma_0 = \{0, 1\} \quad \beta_{0,-1} = 1/2$$
$$\gamma_1 = \{0, 1\} \quad \beta_{1,0} = 1. \tag{13}$$

RK2: Gottlieb et al. 2009, Equation 3.1)), also known as SSPRK 2,2) and Heun s second-order method. Optimal in error bounds) explicit two-stage, second-order SSPRK method:

$$\gamma_0 = \{0, 1\} \quad \beta_{0,-1} = 1$$
$$\gamma_1 = \{1/2, 1/2\} \quad \beta_{1,0} = 1/2. \tag{14}$$

RK3: Gottlieb et al. 2009, Equation 3.2)), also known as SSPRK 3,3). Optimal explicit three-stage, third-order SSPRK method:

$$\gamma_0 = \{0, 1\} \quad \beta_{0,-1} = 1$$
$$\gamma_1 = \{1/4, 3/4\} \quad \beta_{1,0} = 1/4$$
$$\gamma_2 = \{2/3, 1/3\} \quad \beta_{2,1} = 2/3. \tag{15}$$

Note that the RK2 and RK3 methods each have an SSP coef cient of $c = 1$, which implies that their CFL constraint $C_0 = 1$, the same as the stability limit for RK1. In practice, the RK1 integrator is only stable with rst-order DC) fluxes. The stability of RK2 and RK3 is hard to prove with high-order fluxes, but in practice, the limit $C_0 = 1$ seems to work for both PLM and PPM reconstruction for most problems. In 1D, VL2 is stable up to $C_0 = 1$, while in 2D and 3D, VL2 $C_0 = 1$ 2. Moreover, the method is positive-de nite for $C_0 \leqslant 1$ 3 when rst-order fluxes are used in both the predictor and corrector steps Stone & Gardiner 2009). In our experience, the most useful combinations of integrators and reconstruction algorithms are RK1+DC for testing), VL2 or RK2 with either PLM or PPM, and RK3+PPM.

3.2.4. Discretization of the Momentum Equation in Curvilinear Coordinates

Equations 6) are written in conservative form, enabling numerical algorithms that exactly preserve the integrals of the dependent variables over the domain. However, in general curvilinear coordinates, the tensor operators associated with the flux divergence lead to geometrical factors that usually are written as source terms. For example, in cylindrical coordinates R, , Z), the component of the momentum equation can be written as

$$\frac{\partial \rho v_\phi}{\partial t} + \frac{1}{R}\frac{\partial (R M_{R\phi})}{\partial R} + \frac{1}{R}\frac{\partial M_{\phi\phi}}{\partial \phi} + \frac{\partial M_{Z\phi}}{\partial Z} = -\frac{M_{R\phi}}{R}, \tag{16}$$

while in spherical-polar coordinates $r, \theta, \ )$, it can be written as

$$\frac{\partial \rho v_\phi}{\partial t} + \frac{1}{r^2}\frac{\partial (r^2 M_{r\phi})}{\partial r} + \frac{1}{r\sin\theta}\frac{\partial \sin\theta M_{\theta\phi}}{\partial \theta} + \frac{1}{r\sin\theta}\frac{\partial M_{\phi\phi}}{\partial \phi} = -\left(\frac{M_{\phi r} + \cot\theta M_{\phi\theta}}{r}\right), \tag{17}$$

where the M_{ii} are components of the total stress tensor. However, when these equations are written using the angular momentum for example, $R\rho V$ in cylindrical coordinates), they again can be expressed in conservation form, with the geometrical factors embedded in the divergence of the fluxes of angular momentum.

It is possible to express the source terms that appear in the component of the momentum equation in cylindrical and spherical-polar coordinates in a discrete form that also guarantees conservation of the angular momentum to machine precision. In particular, the term on the right-hand side of Equation 16) must be written as

$$\frac{M_{R\phi}}{R} \approx \frac{(R_{i+1/2} - R_{i-1/2})}{(R_{i-1/2} + R_{i+1/2})V_R} \times (R_{i+1/2}M_{R\phi,i+1/2} + R_{i-1/2}M_{R\phi,i-1/2}), \tag{18}$$

where the half-integer indices denote quantities at radial cell faces, $V_R = (R^2_{i+1/2} - R^2_{i-1/2})/2$, and the components of the stress tensor at radial cell faces are the fluxes of momentum given by the solution to the Riemann problem that are used to update the cell. When the source term in Equation 16) is written in this form, it can be shown that the discrete form of the full equation including the flux-divergence terms) is algebraically identical to the conservative difference formula for the angular momentum equation in cylindrical coordinates. Thus, by using this form for the "geometric source term," it is possible to conserve angular momentum to machine precision. This discretization of the momentum equation is adopted in `Athena++` in cylindrical coordinates.

Similarly, in spherical-polar coordinates, the angular momentum can be conserved to machine precision if the source terms on the right-hand side of Equation 17) are discretized appropriately. The rst term can be written in a form similar to Equation 18), but using $V_R = (r^3_{i+1/2} - r^3_{i-1/2})/3$. The second term must be approximated as

$$\frac{\cot\theta M_{\phi\theta}}{r} \approx \frac{(S_{j+1/2} - S_{j-1/2})}{r_i(S_{j-1/2} + S_{j+1/2})V_\theta} \times (S_{j+1/2}M_{\phi\theta,j+1/2} + S_{j-1/2}M_{\phi\theta,j-1/2}), \tag{19}$$

where $S = \sin\theta$, $V_\theta = (\cos\theta_{j+1/2} - \cos\theta_{j-1/2})/2$, and once again the components of the stress tensor at cell faces in the θ direction are the momentum fluxes returned by the Riemann solver and used to update the cell.

Of course, there are also similar terms that appear in the other components of the momentum equation. For these terms, the appropriate volume average can be used. In addition, a variety of coordinate source terms appear in the momentum equation in general-relativistic calculations, depending on the choice of variables. A discrete form that conserves the z-angular momentum is possible; refer to Section 4.1 for additional details.

3.2.5. Diffusion Terms

The MHD module includes terms for modeling many different diffusion processes, for example, isotropic viscosity, resistivity, thermal conduction, and ambipolar diffusion. These terms can be included as an explicit update in each step of the time integrator in a fully unsplit fashion. This is the most accurate formulation for the terms, as it ensures they are evolved at the same temporal order of accuracy as the main, nondiffusive integration algorithm.

To guarantee conservation of momentum, energy, and magnetic flux, the diffusion terms are added as the divergence of the respective fluxes see Equation 6)). Second-order nite differencing is used to compute the components of the viscous stress tensor, heat flux, or EMF as appropriate. For higher-order algorithms, higher-order difference approximations for these fluxes may be required.

Explicit integration of diffusive physics requires a very restrictive time-step stability limit that is inversely proportional to the square of the spatial resolution. When the diffusive terms are relatively large for example, at low Reynolds number) or at very high resolution, this time-step limit can severely restrict the calculation. Therefore, a Runge–Kutta–Legendre RKL) super-time-stepping STS) module Meyer et al. 2012, 2014) has been implemented P. Mullen 2020, private communication), which includes both the RKL1 temporally rst-order accurate) and RKL2 temporally second-order accurate) schemes. When STS is enabled, diffusive physics is advanced forward in time by a separate super time step in an operator-split update. Each super time step comprises s stages and is equivalent to $O\ s^2$) times the explicit diffusive time step. The super-time-step size is set to be equal to the full M)HD time step for the RKL1 algorithm, or half the M)HD time step for the RKL2 algorithm. Two operator-split super time steps are required in a single M)HD update for the second-order accurate RKL2 scheme. All schemes have been shown to 1) produce errors that converge at the appropriate rate for smooth flows and 2) yield the expected speed-up roughly $\propto s$). The algorithm has been parallelized and employs the same task-based execution strategy discussed in the previous sections.

3.2.6. Additional Physics

There are a number of extensions to the basic algorithms for nonrelativistic MHD that have been implemented in `Athena++`, in addition to the general EOS and diffusion terms for nonideal MHD described above. We describe three such extensions below.

Passive Scalars. An arbitrary number of passive scalars that are advected with the fluid flow can be added to the MHD solver. These quantities independently obey a simple conservative transport equation

$$\frac{\partial \rho C_i}{\partial t} + \nabla\cdot[\rho \boldsymbol{v} C_i] = 0, \tag{20}$$

where C_i primitive variable) is the speci c density of each scalar and ρC_i is the mass of each scalar species conserved variable). These quantities provide useful flow diagnostics for following transport and mixing, and they are also necessary for coupling chemical or nuclear reactions to the MHD. In the latter case, source terms representing the net reaction rates are added to the right-hand side of each transport equation, typically via an operator-split method. A complete description

of the implementation of chemical networks in Athena++ will be given in a future publication.

Shearing-box Approximation. For the purposes of studying the dynamics of an accretion disk in a locally rotating frame, the shearing-box approximation is a valuable tool in astrophysical fluid dynamics. A complete description of the implementation of the local shearing-box approximation in the Athena code was presented in Stone & Gardiner 2010). This feature has also been implemented in Athena++ using the same algorithm.

Orbital Advection. In order to speed up and improve the accuracy of calculations in the local shearing box, an orbital advection algorithm has been implemented in Athena++, following the methods described in Stone & Gardiner 2010). The method was developed for hydrodynamics by Masset 2000), implemented in the FARGO code, and later extended to MHD Johnson et al. 2008; Benítez-Llambay & Masset 2016). Orbital advection algorithms have also been implemented in the PLUTO code Mignone et al. 2012). The algorithm in Athena++ also can be employed in global calculations of accretion disk dynamics in cylindrical and spherical-polar coordinates.

3.3. Tests of Nonrelativistic MHD Algorithms

A comprehensive test suite of the MHD algorithms in Athena++ is presented in S08 and will not be repeated here. In this section, we present test results only to demonstrate the properties of new algorithmic features in the code, such as the new reconstruction algorithms and time integrators. We emphasize that whenever values for the magnetic eld are listed, they are given in code units with magnetic permeability $\mu = 1$.

3.3.1. Linear Wave Convergence Test

Measuring the convergence of linear waves provides a quantitative test of errors in the algorithm. For this test, parameters similar to those used in the original Athena paper Gardiner & Stone 2008; S08) are adopted. The box size is L_x, L_y, L_z) = 3.0, 1.5, 1.5), and a grid of $2N \times N \times N$ cells is used with periodic boundary conditions. A plane wave with a perturbation wavelength $\lambda = 1$ and amplitude $A = 10^{-6}$ is initialized propagating along the diagonal of the mesh. Uniform grid resolutions ranging from $N = 16$ to $N = 256$ are adopted, and the error at each resolution is measured by the rms of the volume-weighted L1 norms of each variable as

$$\langle E \rangle = \left[\sum_n \left(\frac{\sum |U_n - U_{n,\mathrm{exact}}| \Delta V}{\sum \Delta V} \right)^2 \right]^{1/2}, \tag{21}$$

where U_n and $U_{n,\mathrm{exact}}$ are the numerical and exact solutions of the nth variable and V is the volume of a cell.

Figure 11 displays the results for each different wave family slow and fast magnetosonic, Alfvén, and entropy waves) computed using different time integrators both VL2 and RK3) and different spatial reconstruction algorithms both PLM and PPM). In all cases, the HLLD approximate Riemann solver is used. As expected, strict second-order overall convergence is observed when either the VL2 time integrator or the PLM reconstruction method is used. The error amplitudes are somewhat lower for each wave when the more accurate PPM reconstruction is used with the VL2 time integrator, although the convergence rate is still exactly second order. The most accurate combination of algorithms is clearly RK3+PPM. Errors in the solution computed with this choice can be an order of magnitude or more lower than those produced by VL2 and PLM. Moreover, for some wave families, the convergence rate of the error is higher across a signi cant range of resolutions close to third order). Because the method does not possess formal third-order spatial accuracy in multidimensional problems, this likely indicates that temporal errors dominate in these cases.

3.3.2. Linear Waves in Nonideal MHD

The MHD module in Athena++ includes terms to model diffusive processes such as isotropic viscosity, resistivity, and thermal conduction. To test these terms, a 2D variant of the linear wave problem described in the previous subsection is considered. The domain size is $(2/\sqrt{5}) \times (1/\sqrt{5})$, and a linearized fast-mode wave is initialized with $\lambda = 1$, at an angle $\theta = \tan^{-1}(2) \approx 63.^{\circ}43$ inclined with respect to the x_1 axis and with a perturbation amplitude $A = 10^{-4}$. The CFL number is set to 0.4, and the fast wave with wave speed $c_f = 2$) is evolved to $t = 0.75$. The explicit, unsplit algorithm for the diffusion terms is used. Because an exact eigenmode of the nonideal MHD wave equation is not initialized, at very high resolutions, the error is limited by errors in the initial data. We discuss this further below.

Kinematic viscosities ranging from $= 10^{-2}$ to 10^{-4} are considered. The standard and magnetic Prandtl numbers are xed to be $\mathrm{Pr} = \mathrm{Pr_m} = 1$ 2 for all tests; that is, $\kappa = \eta =$ 2 . The decay rate of the wave in each simulation is measured by applying weighted least-squares WLS) tting to the time series of $\ln(\max(|v_2|))$ in the solution.

Following Ryu et al. 1995, see Equation 3.13)), the decay rate of the fast wave including thermal conduction) is

$$\Gamma_{f,\ \mathrm{analytic}} = \left(\frac{19\nu}{4} + 3\eta + \frac{3\kappa(\gamma - 1)^2}{4\gamma} \right) \frac{2k^2}{15}. \tag{22}$$

As the authors note, this expression is applicable only up to rst order in the diffusion coef cients, and in the limits

$$\nu k \text{ and } \eta k \ll c_f,\ c_A,\ c_s, \text{ or } a, \tag{23}$$

where $k = 2\pi$ here. For these parameters, the Reynolds number is de ned as Ryu et al. 1995, Equation 3.15))

$$R_f \equiv \frac{4\pi^2 c_f}{\lambda \Gamma_f} = \frac{8\pi^2}{\Gamma_f}. \tag{24}$$

Figure 12 shows the convergence of the numerically measured decay rates to these analytic values over a wide range of Reynolds numbers as the spatial resolution of the mesh is increased. The analytic rates given by the above equation are juxtaposed as dashed black lines in all four cases. Excellent agreement is observed.

Figure 13 demonstrates second-order convergence with mesh resolution of the decay rate at a single xed Reynolds number the largest value considered in the previous plot). As is evident in the previous gure, the RK3+PPM con guration

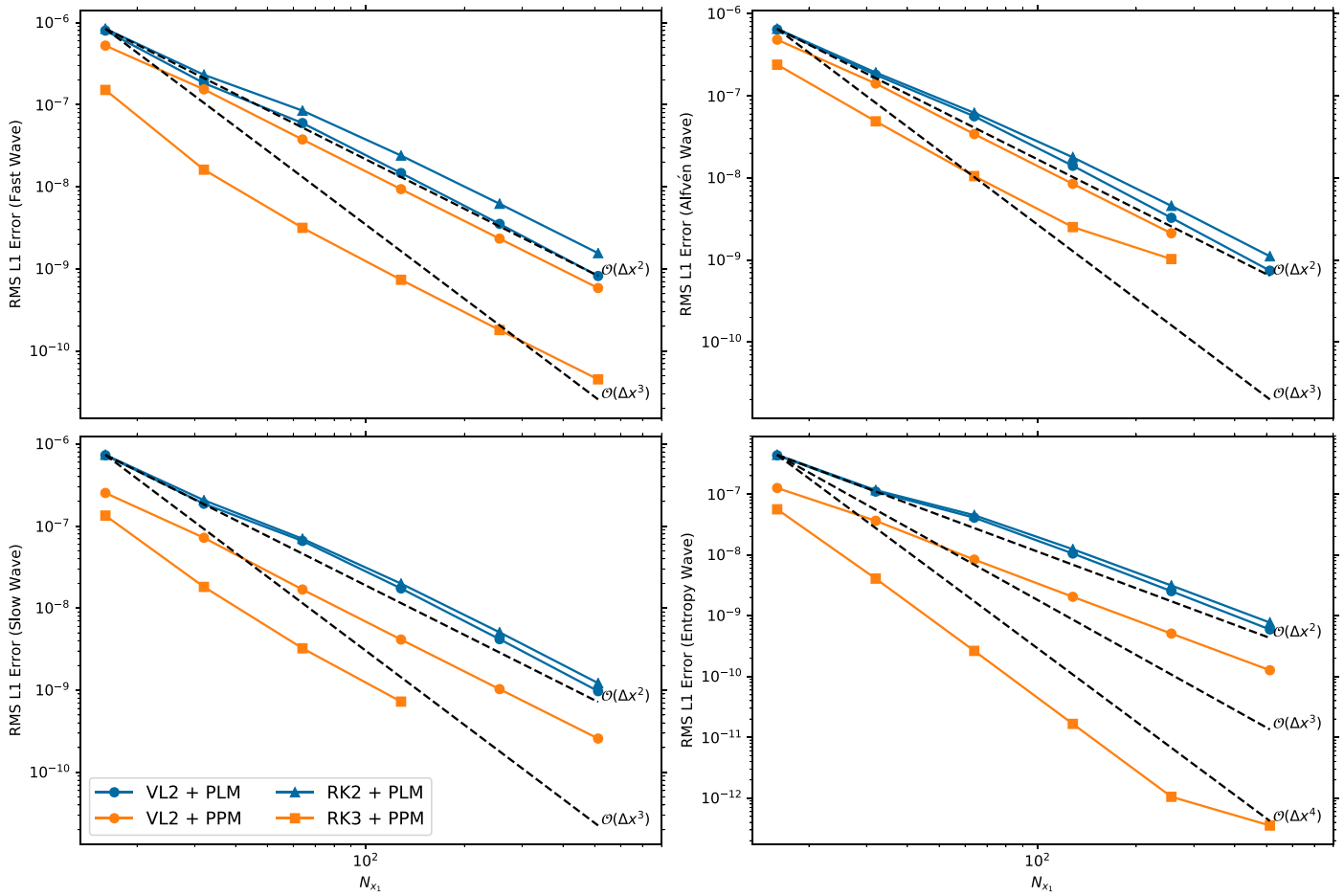

Figure 11 MHD linear wave convergence plots produced using a variety of temporal integrators and variable reconstruction methods without mesh re nement. Clockwise from top left: fast magnetosonic, Alfvén, entropy, and slow magnetosonic wave modes of the linearized system.

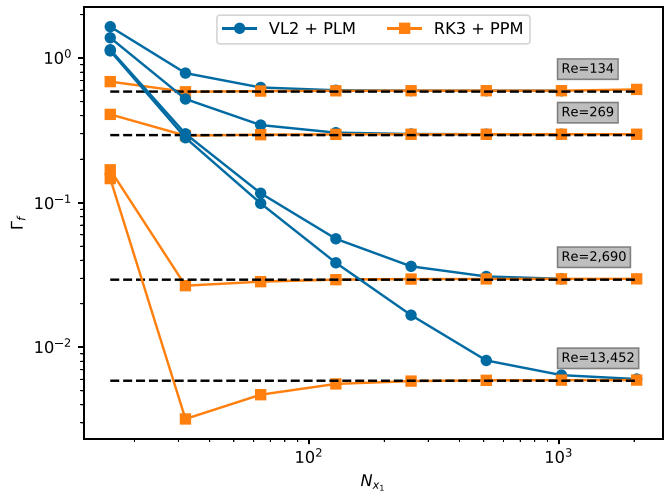

Figure 12 Convergence to the analytic fast-mode decay rates dashed black lines) for a range of Reynolds numbers spanning two orders of magnitude. The more accurate RK3+PPM solver produces linear wave decay rates signi cantly closer to the predicted values than VL2+PLM solutions at higher resolutions.

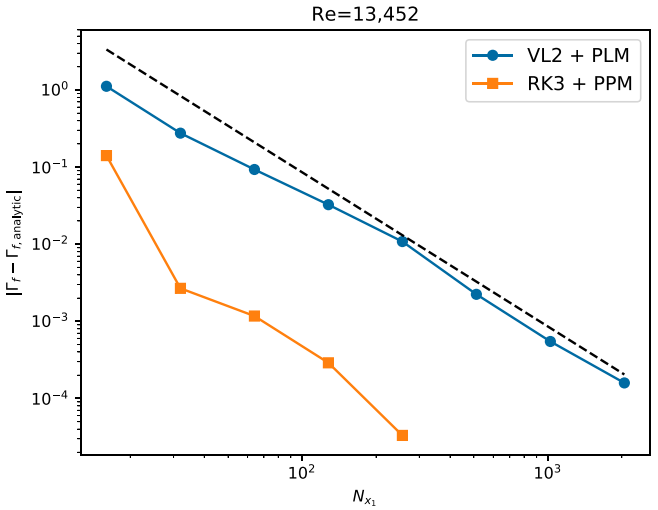

Figure 13 Convergence of the L_1 error of the fast-mode decay rate for the largest Reynolds number case shown in Figure 12.

initially converges to the analytic decay rate much more quickly than the formally second-order accurate VL2+PLM solver. Below values of about 10 ^{4}, the error is dominated by the initial conditions as a wave solution for the ideal rather than nonideal) MHD equations is used. Thus, the errors stop converging beyond the values shown in the plot.

3.3.3. Riemann Problems

In order to test the MHD algorithms with nonlinear solutions, we present the results from multiple shock-tube Riemann) problems. In all cases, the problems are calculated in one dimension along the x_1-axis we have tested that the code generates identical solutions when the tests are run along

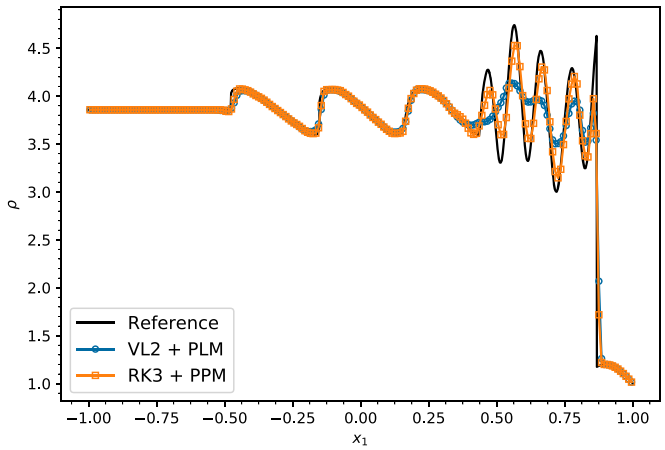

Figure 14 Density in the Shu–Osher hydrodynamic shock-tube test at $t = 0.47$ for $N = 200$ cells. The reference solution was computed using RK3+PPM with $N = 8192$ cells.

the x_2- or x_3-axis, and in S08, we have shown the results for shock tubes run along a grid diagonal in multidimensions). While a huge range of such Riemann problems are available for testing, we focus on two that demonstrate key features of the algorithms: the Shu–Osher problem in hydrodynamics and the Brio–Wu problem in MHD.

Figure 14 shows the density at $t = 0.47$ for the Shu–Osher shock-tube problem Shu & Osher 1989), which involves the interaction of a shock with a smoothly varying background medium. The ability to resolve the sharp features formed by shock compression is a measure of the numerical diffusion in the scheme. Results for both the VL2+PPM and RK3+PPM solver con gurations using the HLLC Riemann solver and $N = 200$ cells are shown along with a reference solution computed using $N = 8192$ cells and the RK3+PPM algorithm. It is clear that the RK3+PPM method is able to capture the short-wavelength oscillations present in the density pro le using only a few cells, and it signi cantly outperforms the VL2 +PLM method on this test.

Figure 15 compares the results of the Brio–Wu MHD shock-tube test Brio & Wu 1988) at time $t = 0.1$ for the same two algorithms but with the HLLD Riemann solver. In this test, reconstruction is performed using the characteristic rather than the primitive variables. If the latter approach is used, the solver produces signi cant oscillations behind the right-moving fast rarefaction. For this reason, this test is an important demonstration of the need for characteristic reconstruction for certain problems. The results show little difference between the two algorithms: RK3+PPM captures the head and foot of rarefactions slightly more accurately. However, both methods perform well for solutions involving MHD shocks and rarefactions in each wave family.

3.3.4. Oblique C Shock with Ambipolar Diffusion

In addition to ohmic resistivity, the core MHD module in `Athena++` includes terms to model ambipolar diffusion. To test this term, results for an oblique C-shock test are presented. The test is identical to the problem described in Masson et al. 2012; see also Wardle & Ng 1999). An adiabatic EOS is used in order to test the heating and energy flux associated with ambipolar diffusion as well. Like the shock-tube test, the left state is $(\rho, v_x, v_y, B_x, B_y, P) = (0.5, 5, 0, \sqrt{2}, \sqrt{2}, 0.125)$ and the right state is $(\rho, v_x, v_y, B_x, B_y, P) = (0.9880, 2.5303, 1.1415, \sqrt{2}, 3.4327, 1.4075)$. A density-dependent ambipolar diffusion coef cient $\eta_{AD} = 1$ 75ρ) is used. In order to reduce the symmetry in the problem, a two-dimensional domain of 0.5, 0.5] × 0.0078125, 0.0078125] with resolution of 1 128 is used, with the initial interface rotated by an angle $\theta = \tan^{-1}(3/4)$ using shifted-periodic boundary condition in the y direction see Tomida et al. 2015, Appendix A.6). The boundary conditions in the x direction are both outflow. Starting from the initial discontinuity, the problem is run until $t = 10$ so that the shock pro le reaches a steady state. The pro le along the shock propagation direction is shown in Figure 16. Even at this relatively low resolution, `Athena++` successfully reproduces the analytic solution.

3.3.5. Liska Wendroff Implosion

The implosion test discussed in Section 4.7 of Liska & Wendroff 2003, hereafter LW) and rst introduced in Hui et al. 1999) is an extraordinarily sensitive test of the directional symmetry-preserving abilities of a hydrodynamics code. The initial condition consists of two uniform states separated by a diagonal discontinuity near the bottom-left corner of the domain, with the jumps in the variables identical to those in the familiar Sod shock-tube test Sod 1978); see Table 1 in S08 for the precise values. Reflecting boundary conditions are used on all four sides. A shock wave launched by the high-pressure region is reflected by the bottom and left boundaries, generating narrow jets of gas characteristic of double Mach reflections Woodward & Colella 1984). Refer to Section 3.4.2 for the full double Mach reflection test. The resulting two jets collide at the lower-left corner and launch two vortices and a single, narrow jet of low-density gas along the grid diagonal. As the evolution progresses, reflected shocks interact with the contact discontinuity and seed the growth of ngers via the Richtmeyer–Meshkov instability. The key ingredient of the test is that the jet will not propagate exactly along the domain diagonal unless the solver maintains reflective symmetry to machine precision across this plane.

Figure 17 shows the density at $t = 2.5$ for a 512×512 mesh. PPM reconstruction of the characteristic variables was used in conjunction with the HLLC Riemann solver and the RK3 time stepper. The results are perfectly symmetric to double-precision machine epsilon for all output variables. Symmetry is maintained for all resolutions and solver permutations that we applied to this problem.

Achieving exact symmetry on this problem is in fact extremely challenging. The PPM reconstruction algorithm is particularly sensitive; the nonassociativity of floating-point arithmetic necessitates that the stencils are written in the C++ source code such that they are calculated without a directional bias. The use of MPI or compiler options that do not guarantee a value-safe floating-point arithmetic mode break `Athena++` s ability to preserve directional symmetry in this test. In order to produce the results shown in Figure 17 with the Intel C++ compiler, options to disable reassociation of operands and contractions of expressions into fused multiply add operations were both required.

Finally, we have also con rmed that when run with AMR, exact symmetry is preserved for this problem, although the details of the solution for example, the strength of the vortices that produce the jet) depend on the re nement condition adopted. This is similar to the behavior on a uniform grid; lower resolution produces weaker vortices and a shorter jet.

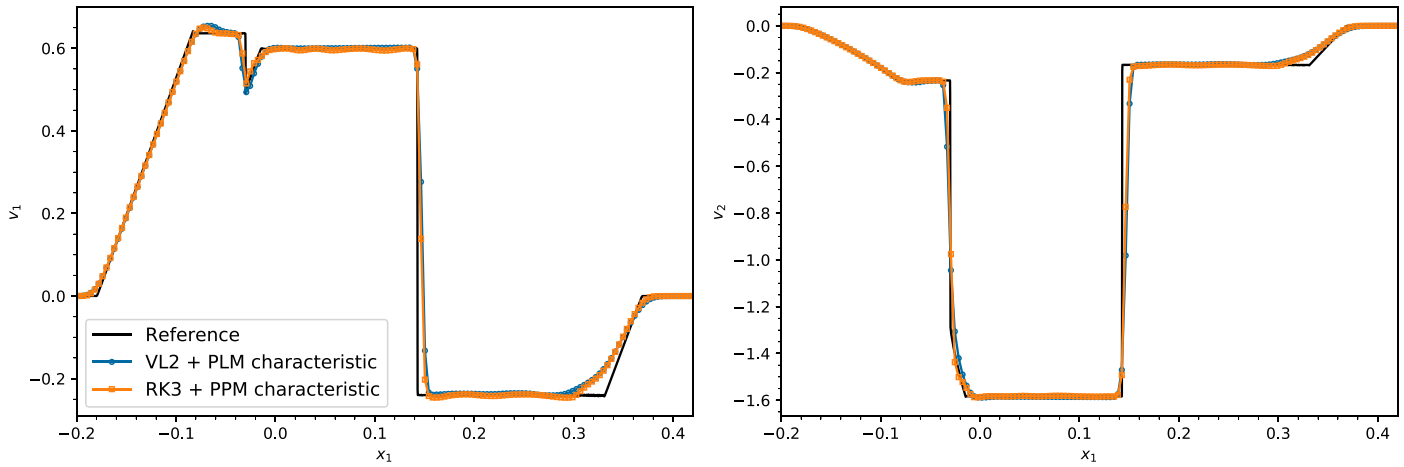

Figure 15 Longitudinal and transverse velocity solutions in the Brio–Wu MHD shock-tube test at $t = 0.1$ for $N = 256$ cells. The reference solution was computed using RK3+PPM reconstruction on characteristic variables with $N = 8192$ cells.

3.3.6. Kelvin Helmholtz Accuracy Benchmark

Finally, to benchmark these algorithms for hydrodynamics against known reference solutions, we consider the Kelvin–Helmholtz instability KHI) test described in Lecoanet et al. 2015). This paper described a well-posed benchmark problem, presented resolved reference solutions computed using the pseudo-spectral code Dedalus Burns et al. 2020), and compared these results to those produced by the original C version of Athena. In this section, we reproduce the analysis of Lecoanet et al. 2015) and compare the results from Athena++ to Dedalus and therefore to Athena, as well).

The strati ed variant of the problem considers an initial condition with a smooth transition of ρ $\rho_0 = 1$ between the two shearing layers and results in behavior that is challenging for a numerical method to resolve with respect to instabilities and small-scale structure. The authors found that Athena C) required a resolution of 16,384 × 32,768 cells in order to converge to the same solution that Dedalus achieved with 2048 × 4096 Fourier modes.

When repeating the tests and comparing the results from Athena C) with those from Athena++, it is worth keeping in mind several key algorithmic differences between the two codes. The results presented in Lecoanet et al. 2015) were generated by the Athena C) code using the CTU integrator combined with PPM reconstruction of the characteristic variables although the authors found that other algorithmic options produced similar results), and the diffusion terms were applied at rst-order accuracy in time using operator splitting. In contrast, Athena++ computes the diffusion processes in an unsplit fashion and does not implement the CTU integrator. All of the Athena++ results shown in this section were produced with reconstruction on primitive hydrodynamic variables and the HLLC Riemann solver.

Explicit diffusion is added via isotropic fluid viscosity , thermal conduction κ, and a separate passive dye diffusion process $_{\rm dye}$. For the test shown in this section, $= \kappa = _{\rm dye} = 2 \times 10^{-5}$, corresponding to a Reynolds number Re $= 10^5$. The CFL number used for the Athena++ tests is $C_0 = 0.4$.

Figure 18 plots the dye eld of the lower half of the domain at $t = 2$, 4, 6, and 8 for Athena++ VL2+PLM and RK3 +PPM at various resolutions. As in Lecoanet et al. 2015), the columns are labeled with "A" for Athena++ or "D" for Dedalus and the N degrees of freedom in the horizontal direction. The Dedalus results shown were produced from the same data as the original study, which was furnished by the authors of Lecoanet et al. 2015).

The results in Figure 18 compare very favorably to the original Athena C) results. Note that only 8192 × 16,384 cells are required to converge to the Dedalus reference solution when RK3+PPM is used with Athena++, which is half the resolution required in the original study. An important contribution to this improvement is the use of an unsplit algorithm for the diffusion terms. The 4096 × 8192 second-order VL2+PLM solution suffers from the onset of the inner vortex instability IVI) at $t = 4$, albeit at a much smaller amplitude than the A4096 CTU+PPM results from Lecoanet et al. s 2015) Figure 8. Both A4096 RK3+PPM and A8192 VL2+PLM avoid the onset of the IVI, although these solutions still exhibit visible differences from D4096 in the lament structure at $t = 6$. However, the A4096 RK3+PPM solution qualitatively appears very close to the converged solution. A detailed comparison of the results, along with quantitative study of the errors between solutions, is provided in Felker 2019). A further notable result is that due to the much higher computational performance of the nite volume compared to spectral methods, the A8192 solution took only one-half of the time required to compute the D4096 solution. Thus, Athena++ achieves spectral accuracy for this problem at less cost.

3.4. Tests of AMR with MHD

Next, we present the results for a series of test problems that demonstrate the accuracy of our AMR methods.

3.4.1. Linear Wave Convergence with AMR

Locally re ned grids should produce more accurate solutions than a uniform resolution root grid, and the global convergence rate on AMR grids should be second order. To test these expectations, the MHD linear convergence test can be used to provide quantitative measures of the errors and convergence rate of solutions on an AMR grid in Athena++.

The test is identical to that already presented in Section 3.3.2 for a uniform grid. Results with the same range of resolutions from $N = 16$ to $N = 256$ are presented; however, with AMR,

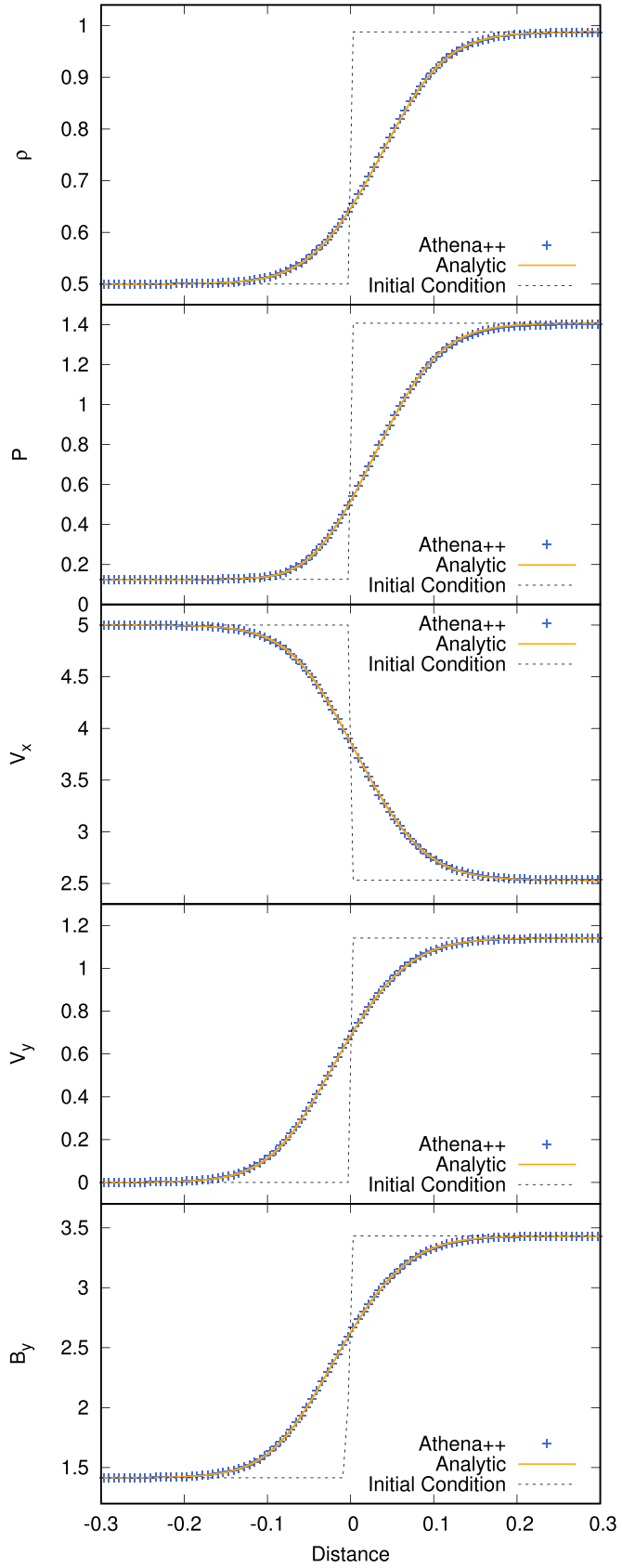

Figure 16 Steady-state solution in the adiabatic oblique C-shock test.

one additional ner level at twice higher resolution per dimension) is introduced in regions where the density is within 90 of the peak value. Note that this re nement condition was

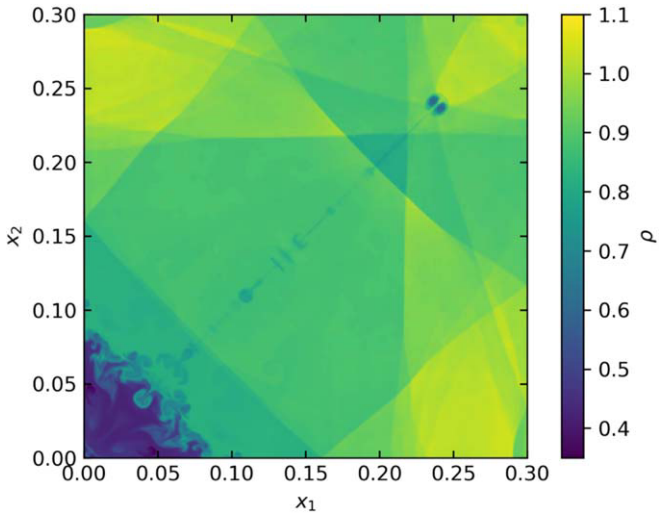

Figure 17 Density in the LW implosion test at $t = 2.5$ using RK3+PPM. Exact symmetry is maintained along the diagonal, and a low-density jet is produced there as a consequence.

chosen solely to demonstrate the behavior of the AMR, and it was not motivated by any physical requirements. Each `MeshBlock` consists of 4^3 cells for the lowest resolution run and 64^3 cells for the highest, so that the re ned regions occupy the same volume. As before, errors are measured using the rms of the volume-weighted L1 norms of each variable Equation 21)). The VL2 time integrator and both the PLM and PPM reconstruction algorithms are used for comparison.

The results for the fast wave are shown in Figure 19; the other waves behave similarly. As expected, both unre ned and AMR simulations achieve global second-order accuracy, and the AMR simulations exhibit slightly better error than the unre ned grid simulations with the same root grid resolution. As in Figure 11, using PPM with the van Leer integrator yields a lower spatial error for most of the resolutions. However, at the largest resolution, the second-order accurate truncation errors of the AMR prolongation and restriction operators dominate the higher-order terms associated with the PPM reconstruction. This plot is extremely informative and clearly demonstrates that second-order convergence of global errors is achieved with the AMR algorithm in `Athena++`.

Additional computational expense is incurred when enabling AMR due to the addition of re nement, dere nement, prolongation, and restriction operations. Although more cells are added when the grid is re ned, the overall ef ciency of calculating the solution for a single cell remains high. At the highest tested root grid resolution of $512 \times 256 \times 256$ cells, the second-order solver advances the unre ned mesh at 132.5 million zone-cycles s ^{1} when deployed with MPI on four dual-socket nodes of an Intel Skylake system. When AMR capabilities are enabled, the solver slows to 106.0 million zone-cycles s ^{1}, representing a performance overhead of about 20 . This relative performance cost is larger at lower resolutions, but it is quickly amortized by increasing the size of the blocks. Furthermore, the ef ciency does not decline as more levels are added or as the re ned region grows. However, we emphasize that the overhead of AMR depends on many factors such as the volume of the re ned region, frequency of re nement operations, and the size of `MeshBlocks`, and therefore, it is highly problem dependent see discussion of results from other tests below).

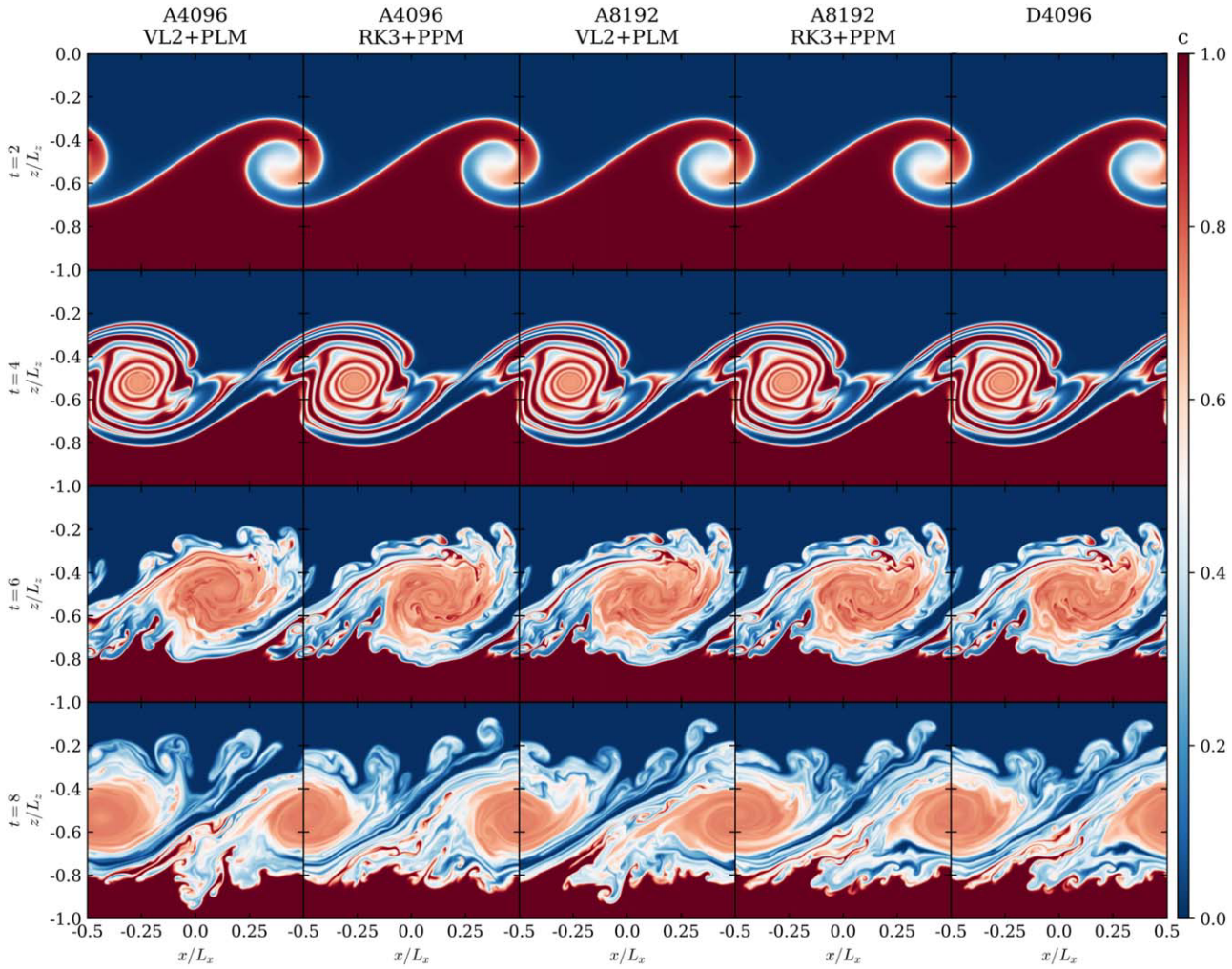

Figure 18 Snapshots of the solution to the KHI problem with `Athena++` and `Dedalus` at various resolutions and times. Compare to Figure 8 of Lecoanet et al. 2015).

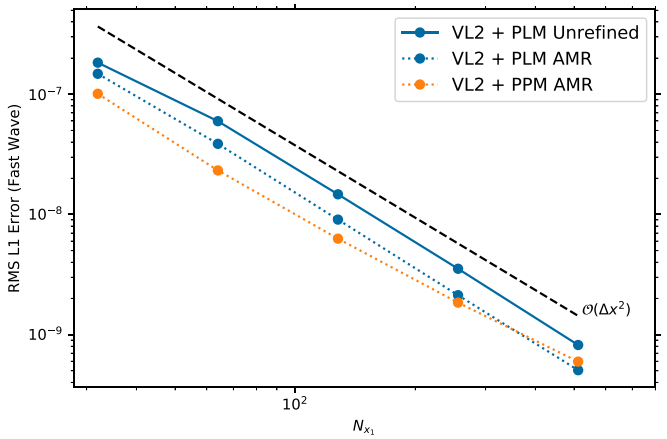

Figure 19 Errors in a linear wave convergence test with and without AMR. Results are shown for the fast wave, but other modes show similar trends. Second-order convergence is achieved in all cases.

Note that the linear wave convergence test is not only simple but also highly sensitive to most subtle defects in the code. For example, if boundary communication between levels is implemented incorrectly, the AMR calculation will have a larger error than a uniform grid at the resolution of the root level. Moreover, even if only one boundary cell is communicated incorrectly for example, at the edge or the corner of the `MeshBlock`; see Section 2.1.3), this will be evident through the lack of convergence of the L_∞ error.

3.4.2. Double Mach Reflection Test

The double Mach reflection problem Woodward & Colella 1984) is a standard test for hydrodynamics codes. It involves a Mach 10 shock that reflects from an inclined plane. This interaction produces complex structures such as discontinuities, a triple point, and a jet. Therefore, this is a good problem for evaluating the correctness of the AMR implementation and the robustness of the code with shocks.

For this test, characteristic reconstruction is used in order to suppress numerical oscillations produced at the strong shock, and the HLLE approximate Riemann solver is chosen in order to suppress the Carbuncle-like instability at the head of the jet Gittings et al. 2008). The H-correction scheme in the `Athena` code S08) suppresses these instabilities; however, it has not yet been implemented in `Athena++`. The initial and boundary conditions are given in Woodward & Colella 1984). For the uniform grid simulation, the resolution is $x = 1$ 120. For the AMR simulation, the root grid is set to be

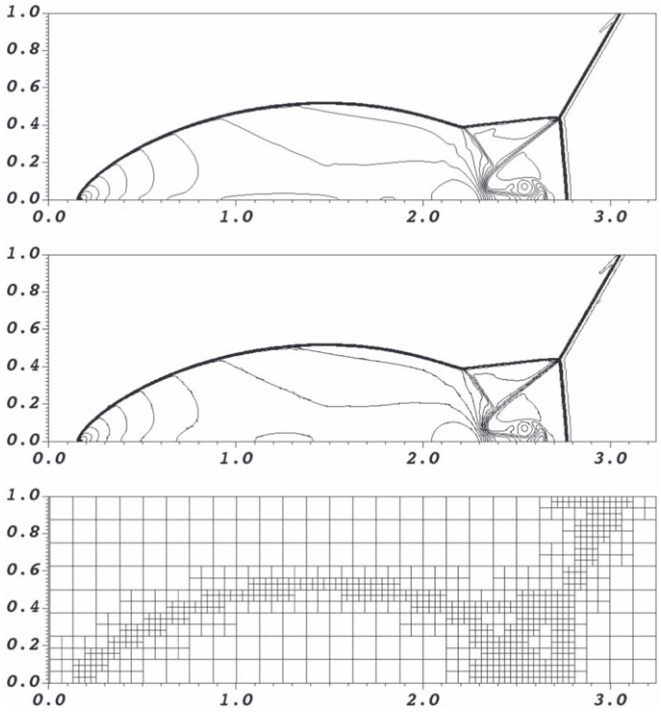

Figure 20 Double Mach reflection test with a uniform grid top) and AMR middle) using the same effective resolution. The density at $t = 0.2$ is shown with 30 levels of contours. The bottom panel shows the distribution of `MeshBlocks`.

four times coarser $x = 1$ 30) and up to two ner levels are used so that the nest structures are captured with the same resolution as the uniform grid. Each `MeshBlock` has 6×6 cells. A re nement condition based on the second spatial derivatives i.e., curvature) is used, as in Matsumoto 2007):

$$\epsilon = \max\left(\frac{|\partial_x^2 q_{i,j} + \partial_y^2 q_{i,j}|\Delta x^2}{q_{i,j}}\right), \tag{25}$$

where $q_{i,\ j}$ is a quantity such as density or pressure. A `MeshBlock` is flagged to be re ned when ϵ exceeds 0.01 and dere ned when ϵ falls below 0.005.

The results are shown in Figure 20. The AMR grid produces results that are essentially indistinguishable from those on a uniform mesh. The lower panel shows the distribution of `MeshBlocks` in the AMR calculation. A relatively small volume of the domain requires the nest resolution, so our block-based AMR algorithm remains fairly ef cient for this problem. In particular, a uniform grid tiled with 6^2 `MeshBlocks` requires 279 CPU seconds on a single core of a Skylake 6148 processor, whereas the AMR run using the same-size `MeshBlocks` required only 63.2 CPU seconds, for a speed-up of 4.4. It must be noted, however, that performance depends strongly on `MeshBlock` size see Sections 3.6.4 and 3.6.5). For example, doubling the size of the `MeshBlocks` to 12^2 decreases the run times to 127 and 31.8 CPU seconds for the uniform mesh and AMR runs respectively, and on a uniform grid with a single `MeshBlock`, the runtime is only 42.8 CPU seconds. Even though using larger `MeshBlocks` with AMR results in the nest level covering a larger fraction of the domain making the calculation less ef cient by this measure), nevertheless, the time to solution is decreased.

3.4.3. KHI Tests

To further compare the accuracy and costs of solutions computed with an AMR grid with those using a uniform mesh, results from a series of KHI tests in both hydrodynamics and MHD are presented. The computational domain is chosen to span $0.5 < x < 0.5$ and $0.5 < y < 0.5$ with periodic boundary conditions in both x and y directions. A shear flow with a density contrast of two and a velocity jump of one is initialized, using a smooth resolved) interface so that the pro les of density and velocity follow

$$\rho = 1.5 - 0.5\tanh\left(\frac{|y - 0.25|}{L}\right), \tag{26a}$$

$$v_x = 0.5\tanh\left(\frac{|y - 0.25|}{L}\right), \tag{26b}$$

$$v_y = A\cos(4\pi x)\exp\left[-\frac{(y - 0.25)^2}{\sigma^2}\right]. \tag{26c}$$

Here, $L = 0.01$ is the thickness of the shearing layer, $A = 0.01$ is the amplitude of the initial perturbation with a wavelength of 0.5, and $\sigma = 0.2$ is the thickness of the perturbed layer. The total pressure is constant everywhere and equal to $p = 2.5$, with adiabatic index $\gamma = 1.4$. This gives a sound speed $C^2 = 3.5$ in the lowest density region. The use of a smooth initial pro le for the interface rather than a discontinuity is crucial for obtaining a well-posed problem that converges with resolution e.g., McNally et al. 2012).

For the MHD test, a uniform horizontal eld of $B_x = 0.1$ is added. The HLLD flux for MHD, HLLC flux for hydrodynamics, PLM reconstruction, and VL2 integrator are all used. The problem is run rst with a uniform grid of 2048×2048, and then the calculation is repeated with AMR using four levels so that the same maximum resolution is achieved as the uniform mesh when the root grid resolution is 256×256. `MeshBlocks` of size 8^2 and 16^2 are used with a re nement condition based on the velocity shear:

$$g = h \times \max(\partial_x v_y, \partial_y v_x). \tag{27}$$

A `MeshBlock` is re ned if g is larger than 0.01 or dere ned if g is smaller than 0.005.

The results for the hydrodynamic test are shown in Figure 21. The density shown in the top panels) in the AMR and uniform grid runs is indistinguishable. The fractional difference in the density between the two calculations, shown in the lower-left panel, is more illustrative. It is dominated by short-wavelength sound waves that are damped in the low-resolution coarse mesh) regions of the AMR calculation. Very narrow features that follow the cat s eye rolls produced by the KHI are barely discernible. They are associated with slight less than one grid cell) differences in the positions of the interfaces in the two calculations. It is likely that such differences are unavoidable, as the interaction of the sound waves that cannot be represented in the AMR calculation but are present on the uniform grid) with the interfaces can produce differences of

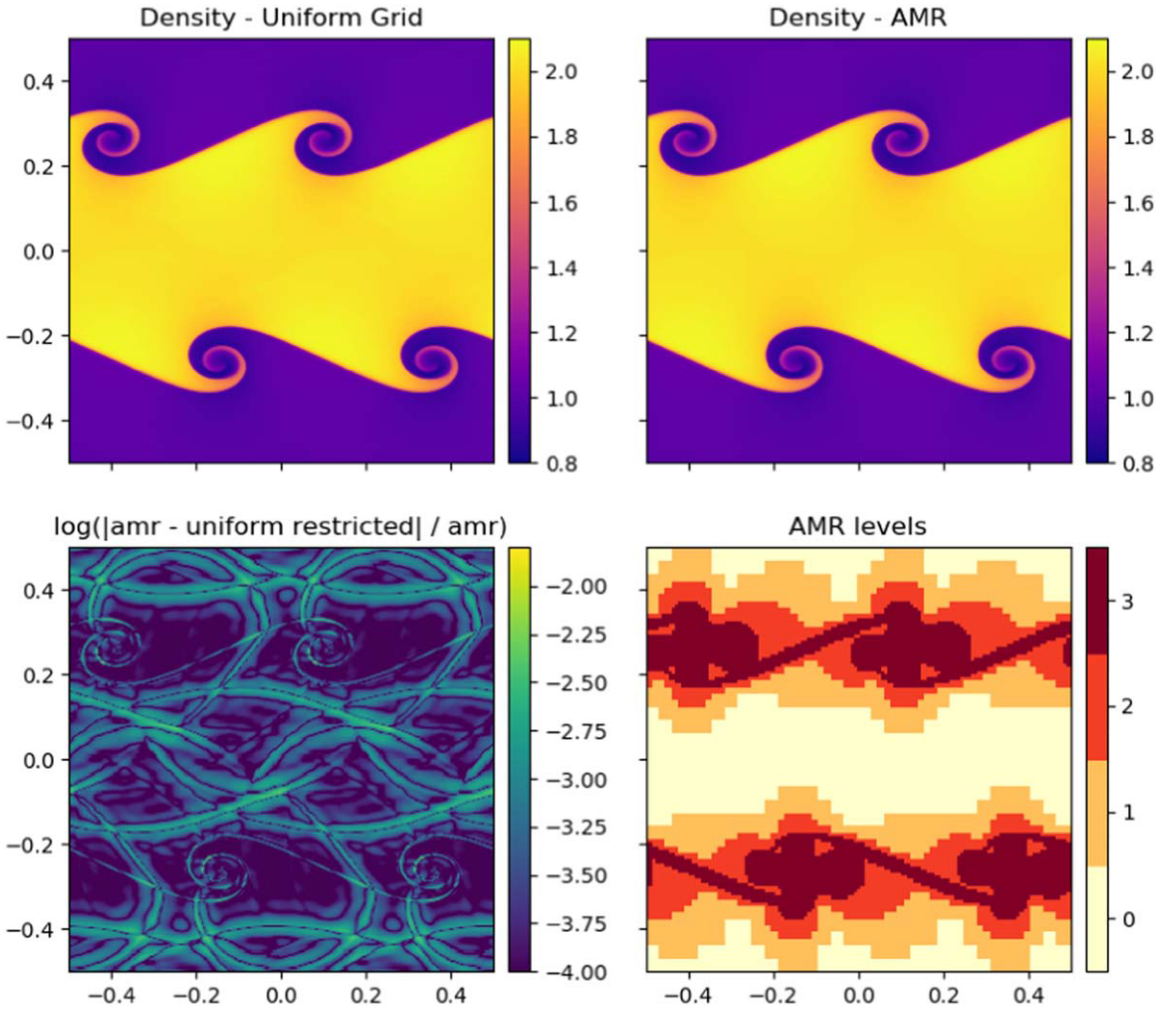

Figure 21 Hydrodynamic Kelvin–Helmholtz instability test with AMR. The top-left panel shows the density at $t = 1.2$ using a 2048 × 2048 uniform grid, while the top-right panel is the result with the same effective resolution using 4 levels of AMR with `MeshBlocks` of 8^2. The two are indistinguishable. The bottom-left panel shows the fractional difference in density between the uniform grid and AMR runs. The bottom-right panel shows the distribution of `MeshBlocks` in the AMR run.

the observed magnitude. This is an interesting lesson on the limitations of AMR. If the dynamics of these waves are important for example, body modes of the KHI in astrophysical jets; e.g., Hardee 1979), then AMR cannot be used for the problem in this way. The lower-left panel in Figure 21 shows that the volume- lling factor of `MeshBlocks` at the nest level in the AMR solution is relatively small.

The results for the MHD test are shown in Figure 22. The results and conclusions are nearly identical to those for the hydrodynamic version of this problem. The fractional density difference shown in the lower-left panel reveals a more intricate pattern in the MHD calculation because it consists of both fast and slow modes that are both damped. Taken together, Figures 21 and 22 show that AMR is able to capture the dynamics of the KHI on isolated interfaces very successfully.

The fractional difference in the density between uniform grid and AMR calculations, computational time, performance, and number of cells per calculation are summarized in Table 1. These performance measurements include le outputs every $t = 0.01$. The largest differences emerge mainly at the discontinuities, because even a tiny phase error can produce large pointwise differences. The computational time and performance are measured using 32 cores 16 cores per socket) of an Intel Skylake Xeon 6148 node. While the computational throughput considerably degrades when AMR is in use ∼40 with 16^2 and ∼18 with 8^2), it still reduces the overall computational cost and data size. For this speci c test, `MeshBlocks` of 16^2 are optimal, but in other problems, this size should be chosen carefully based on the required accuracy and ef ciency.

3.4.4. 3D Blast-wave Tests

The Sedov–Taylor solution Sedov 1946; Taylor 1950) provides the basis for useful quantitative tests involving the propagation of blast waves. In order to demonstrate the AMR capabilities of `Athena++` in 3D, we have performed 3D blast-wave tests with and without magnetic elds.

For both nonmagnetized and magnetized models, the same initial condition apart from the magnetic eld) is used. The computational domain spans a cubic region with edge length $L = 1$ and periodic boundary conditions on all faces. The initial density is set to 1, while the pressure is 0.001 everywhere. To initialize the blast wave, the total internal energy in a region of radius 0.01 at the center of the domain $E_{\rm tot} = \int dVP/(\gamma - 1) = 1$ with $\gamma = 5$ 3, giving a pressure of 1.6×10^5 in this region. For the MHD version of the problem,

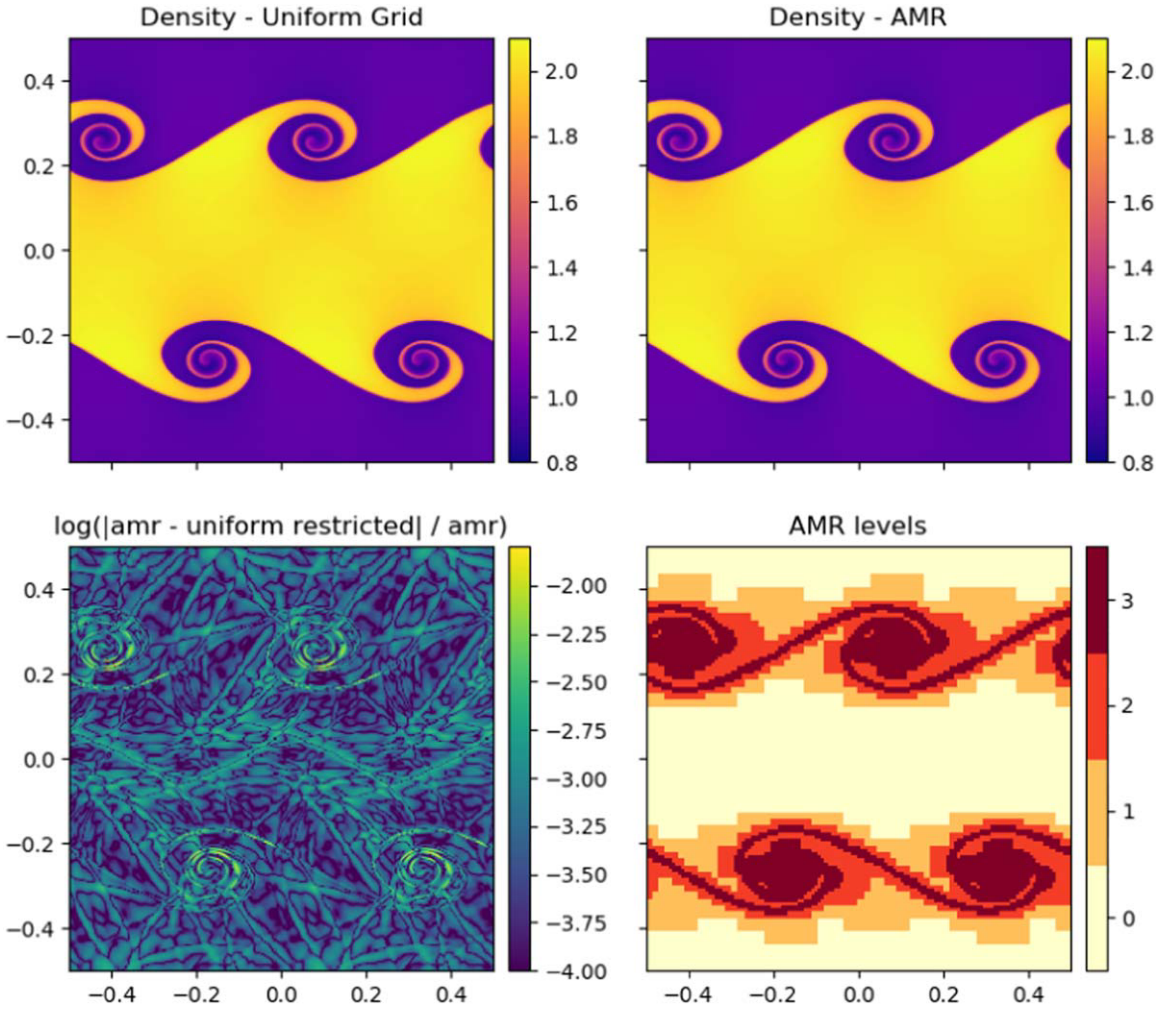

Figure 22 Same as Figure 21 but for the MHD Kelvin–Helmholtz instability at $t = 1.5$.

the magnetic field is uniform and inclined to the grid: $B_x = \sqrt{3}$ and $B_y = 1$.

The VL2+PLM algorithm is used for both models, along with the HLLE solver for hydrodynamics to suppress the Carbuncle instability) and the HLLD solver for the MHD simulation. A refinement condition based on the pressure jump is used:

$$g = h \times \max\left(\frac{|\nabla p|}{p}\right). \tag{28}$$

A `MeshBlock` is refined when g exceeds a threshold value 0.1 in hydrodynamics and 0.2 for MHD) and is flagged for derefinement if g is smaller than 1 4 of this value. The root grid consists of 128^3 cells with two additional levels of refinement, resulting in an effective resolution of 512^3. For comparison, the result from a uniform grid calculation using 512^3 cells is also shown.

Figure 23 shows the distributions of the pressure and `MeshBlocks` at the end of the hydrodynamic calculation. Comparison of the solutions on the uniform and AMR grids shows essentially no difference. Excellent spherical symmetry is maintained. `MeshBlocks` at the nest level ll only a small fraction of the domain. Figure 24 shows the same plots for the MHD calculation. Again, the solutions on the uniform and AMR mesh are visually identical. Although the magnetic eld breaks the spherical symmetry of the problem, reflection symmetry perpendicular to the eld direction is maintained.

A more quantitative comparison of the hydrodynamic solution is shown in Figure 25. The pressure in each grid point in the AMR solution is plotted as a function of radial distance from the center, and this is compared to the analytic Sedov–Taylor blast solution obtained using `sedov3.f` developed by F. X. Timmes.[16] Note the excellent agreement. The nite width of the points from the numerical solution is in part an unavoidable consequence of the representation of a sphere on a Cartesian mesh. These gures demonstrate that AMR can reproduce both the uniform grid and analytic solutions very well.

3.5. Tests of Curvilinear Coordinates and AMR

Finally, we show results for test problems in curvilinear coordinates a new capability in `Athena++`), both with and without AMR.

3.5.1. Advection Tests in Curvilinear Coordinates

Figure 26 plots the pro les of the `Athena++` solutions to the radial 1D advection problem of Mignone 2014, Section 5.1.1) in cylindrical and spherical-polar coordinates for both PLM and PPM. For these tests, a passive scalar is initialized with a Gaussian pro le and advected with a linear velocity

[16] http: cococubed.asu.edu research_pages sedov.shtml

Table 1
Summary of AMR KHI Test Accuracy and Ef ciency

	Grid	Maximum Density Difference)	Mean Density Difference)	Time s)	Performance MZone-cycles s ^{1})	Number of Cells 10^6)
Hydro	Uniform	···	···	319	270.0	4.19
	AMR 8^2	0.70	0.046	319	46.7	1.09
	AMR 16^2	0.80	0.044	152	108.5	1.25
MHD	Uniform	···	···	883	118.7	4.19
	AMR 8^2	1.59	0.060	688	21.4	0.90
	AMR 16^2	1.27	0.056	387	49.1	1.16

eld. The parameters a and b in the plot labels control the width of the Gaussian and location of the curve s center, respectively.

Because a variant of the original PPM limiter is used with curvilinear coordinates in Athena++, the smooth extrema in the right column solutions are clipped; this does not occur with the more advanced limiter used in Cartesian coordinates. Future work will consider extending the curvilinear corrections to the smooth extrema-preserving PPM limiter.

Figures 27 and 28 show the convergence of the L_1 error in the radial and a 2D counterpart of the meridional see Mignone 2014, Section 5.1.2) scalar advection problems, respectively. They demonstrate the formal second- and fourth-order convergence of PLM and PPM reconstruction in Athena++. There are slight differences between the PLM results shown here and those shown in the original reference Mignone 2014); their choice of a modi ed monotonized central MC) limiter for PLM results in lower errors than the van Leer limiter for the nonmonotonic tests and higher errors in the monotonic tests. Overall, these plots demonstrate the delity of the solvers in curvilinear grids.

3.5.2. Field Loop Advection through the Pole

Near coordinate singularities such as the poles in a 3D spherical-polar mesh), numerical discretizations generally have nonuniform truncation error which can imprint visible features in solutions. Moreover, flow through the pole requires special boundary conditions that load the ghost cells of MeshBlocks that overlap regions across the pole with data from the appropriate azimuthal angle. To test the implementation of curvilinear coordinates at the poles, the results for an advection of flow through the pole is presented.

The problem consists of a uniform parallel velocity eld $v_x = 1$ that is represented on a spherical-polar mesh, with the poles perpendicular to the flow velocity. A passive magnetic eld loop is then initialized and advected through the poles. Following Gardiner & Stone 2005), the magnetic elds are initialized with a vector potential of the form

$$A_z = B_0 \exp\left[-\frac{(z - z_0)^2}{\sigma^2}\right] \times \max\left(R - \sqrt{(x - x_0)^2 + (y - y_0)^2}, 0\right), \quad (29)$$

where $(x_0, y_0, z_0) = (-\sqrt{2}/2, 0, \sqrt{2}/2)$ is the initial center of the loop, B_0 the magnetic eld strength, $R = 0.5$ the radius of the loop, and $\sigma = 0.2$ the thickness of the loop. The eld strength B_0 is set so that $\beta = 2p/(B_0^2) = 10^5$ at the midplane of the loop. PLM reconstruction, the HLLD approximate Riemann solver, and an adiabatic EOS with $\gamma = 5\ 3$ are used. The computational domain is $0.1 < r < 2.0$, $0 < \theta < \pi\ 2$, and $0 < \ < 2\pi$, and the resolution is $160 \times 80 \times 160$ using logarithmic spacing in the r direction.

Figure 29 shows the magnetic eld strength on slices through the computational mesh at the center of the eld loop in the initial and nal states. The loop shows evidence for numerical diffusion, especially at the center where oppositely directed eld lines are closely spaced. However, the structure is well preserved even after advection through highly anisotropic coordinates and the coordinate singularity. While this test is perhaps arti cial spherical-polar grids are not a good representation of the initial flow geometry), it nevertheless demonstrates the robustness of our nite-volume scheme in curvilinear coordinates.

3.5.3. Blast-wave Test in Spherical-polar Coordinates

To demonstrate AMR in curvilinear coordinates, the same blast-wave tests detailed in Section 3.4.4 were run using spherical-polar coordinates. The problem domain is $0.5 < r < 1.5$, $\pi\ 6 < \theta < \pi\ 2$, and $\pi\ 5 < \ < \pi\ 5$. The grid is nonuniformly spaced along the r direction so that the aspect ratio of the cells remains close to unity everywhere. The other parameters are the same as the problem in Cartesian coordinates. For the MHD model, the magnetic eld is initially uniform along the pole.

The results for the hydrodynamic test are shown in Figure 30. Note that the blast remains spherically symmetric even on the curvilinear mesh. The plot shows excellent agreement with Figure 23. The results for the MHD test are shown in Figure 31. Again, there is excellent agreement with the previous results found for a Cartesian grid and shown in Figure 24.

Finally, Figure 32 plots the pressure as a function of radial position from the center of the blast for the hydrodynamic problem shown in Figure 30, along with the analytic solution for the Sedov–Taylor blast wave. The results can be compared to Figure 25, which used a Cartesian grid. The peak of the pressure curve at the location of the blast wave is slightly smeared in the spherical-polar grid solution. However, this phenomenon is explained by the use of a nonuniform radial grid that has larger cells at larger radii. Otherwise, excellent agreement is obtained.

3.6. Performance and Scaling of the MHD Solver

Most of the scienti c applications of Athena++ require multidimensional calculations at high resolution. Performance of the solver is often a rate-limiting step for progress, and therefore, we have spent considerable effort trying to maximize the performance and scaling of the MHD solver. For example,

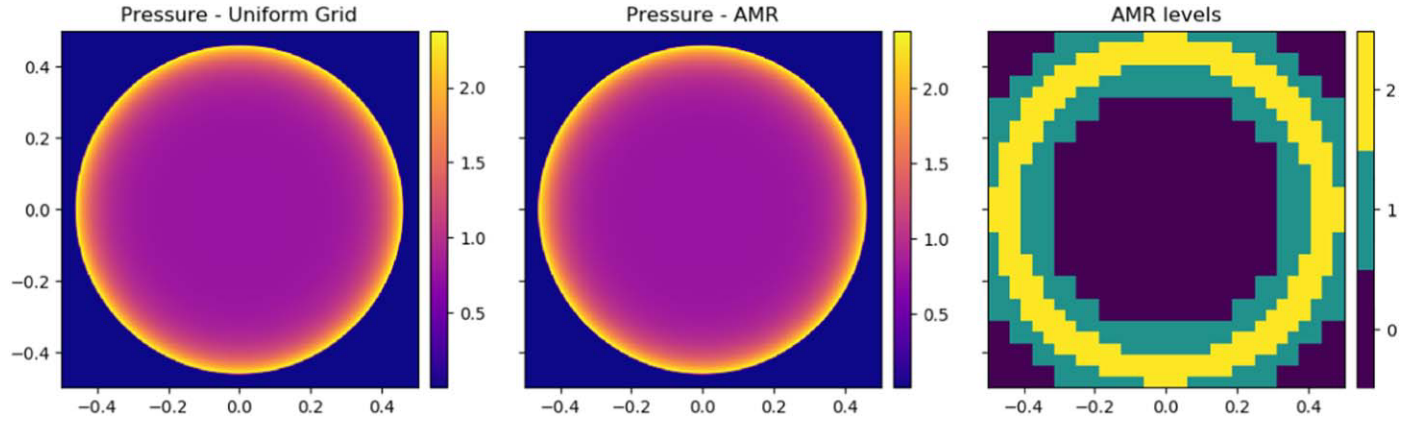

Figure 23 Two-dimensional slice of the pressure in a hydrodynamic blast-wave test at $t = 0.1$ with a uniform grid left) and AMR middle) using the same effective resolution. The right panel shows the distribution of `MeshBlocks`.

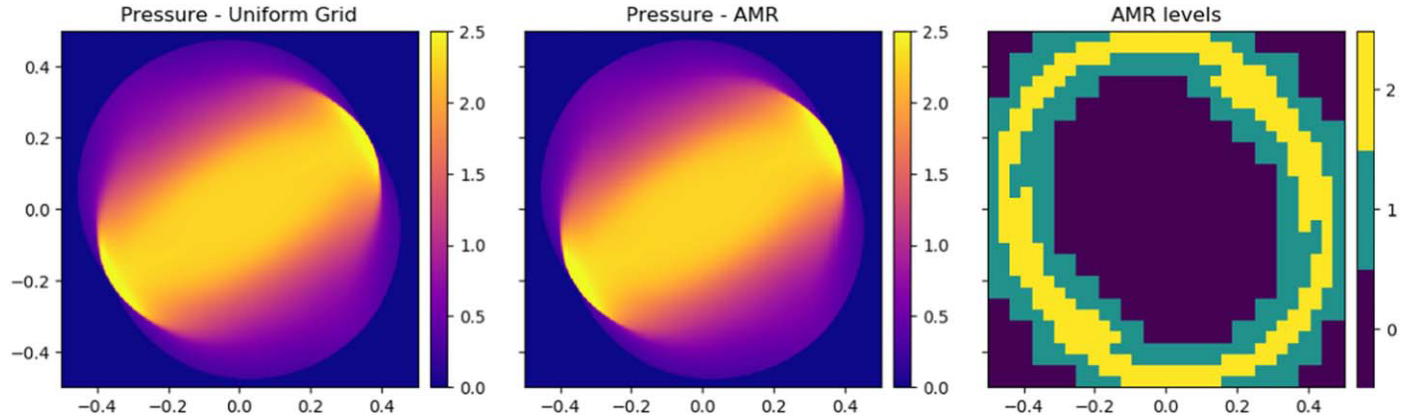

Figure 24 The same as Figure 23 but for the MHD blast-wave test at $t = 0.08$.

the initial design of the C++ classes used in `Athena++` resulted from extensive performance benchmarking of the core computational kernel of the algorithm. The design was continually compared against the highest performance achieved for raw C code that implemented the same steps. Only once the design of the C++ mocked classes met or exceeded the performance of the raw C code was this design used to implement the full code. In this way, we ensure that none of the abstractions of the object-oriented design inhibit performance optimization by the compiler. In this section, we report the performance and scaling we have achieved for the MHD solver in the `Athena++` AMR framework.

3.6.1. Single-core Performance

Table 2 summarizes the performance averaged across 20 independent trials) using only a single physical core on a single node of three target Intel architectures. The test is based on a three-dimensional benchmark problem the blast-wave test in Section 3.4.4) for adiabatic hydrodynamics and MHD, and it considers multiple Riemann solvers and primitive variable reconstruction techniques. The default second-order accurate VL2 time integrator is used in all cases, and the problem size is xed to a single 64^3 `MeshBlock`. Performance is measured in the number of cells updated per second the inverse of which is the CPU time required to update a single cell).

For 3D MHD with the HLLD Riemann solver and PLM reconstruction a typical combination of algorithmic options), we achieve nearly 3 million zone updates per second per core on the Intel Skylake processor. With PPM reconstruction, the performance drops by about a factor of 2. For comparison, we have run the same benchmark problem using the same algorithmic choices and the same compiler optimizations for the latest public versions of the `FLASH`, `PLUTO`, and `Enzo` codes, and we nd that the per-core performance of `Athena++` is the highest of all four, in some cases by as much as a factor of 10. Good performance on modern processors is achieved only through the use a high percentage of vectorized instructions. Using Intel diagnostic tools, we nd about 85 based on the CPU time) of the MHD code is vectorized using the AVX AVX2 AVX512 vector instruction sets.

3.6.2. Multicore Performance

Table 3 summarizes the code s performance when using all of the cores available on a single node. For this test, both Broadwell 14 cores per socket) and Skylake 20 cores per socket) CPUs con gured as dual-socket nodes were used, for a total of 28 and 40 cores, respectively. The KNL node possesses a total of 68 physical cores, but we use only 64 cores in order to minimize jitter from the operating system. A single MPI rank is pinned to each core in these tests. In addition, the KNL tests bene t from using 4 OpenMP threads per MPI rank in order to utilize the 4 way hyperthreading of the 64 physical 256 logical) cores on these nodes. In this case, each thread owns a `MeshBlock` of size $64 \times 32 \times 32$.

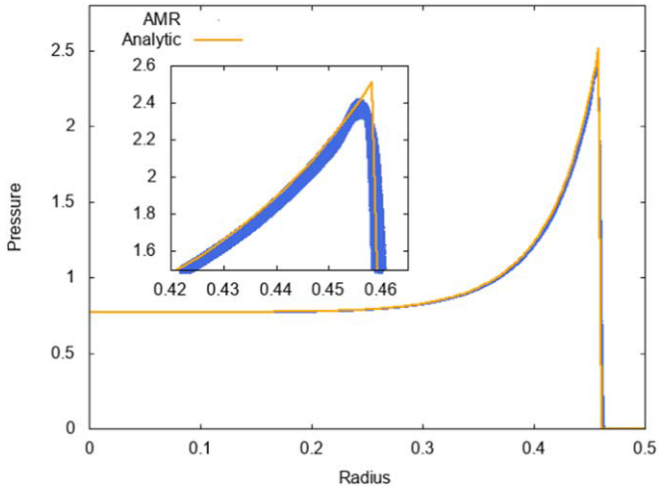

Figure 25 Radial pro le of the gas pressure in the hydrodynamic blast-wave test with AMR. The pressure in each cell as a function of distance from the center is shown with a blue dot, and the orange line indicates the analytic solution.

Note that in all cases, the performance per core is signi cantly less than that reported in Table 2, typically by a factor of 2 regardless of the choice of algorithm. Modern Intel processors decrease the overall clock speed when all cores are active and are executing AVX2 AVX512 instructions, which contributes in part to the decrease. However, most of the decrease is due to memory bandwidth limits and less-than-optimal use of cache. Generally, algorithms with higher arithmetic intensity ratio of flops to memory accesses) are less affected by memory bandwidth limits. However, we observe the same decrease in performance when all cores are used independent of which algorithm we adopt. For example, PPM reconstruction with the HLLD Riemann solver for MHD requires nearly three times the number of floating-point operations per cell than PLM reconstruction and the HLLE solver for hydrodynamics, yet both display the same factor of 2 decrease in performance when all cores are used. We have observed the same trend even for the complex fourth-order algorithm implemented in Felker & Stone 2018). This indicates the overall design and implementation of the MHD solver in `Athena++` is cache-limited.

It is important to note that different algorithmic choices can greatly improve cache performance. For example, Woodward et al. 2019) describe an approach for organizing data into small "mini briquettes" that t entirely into cache and which enables excellent performance for the dimensionally split PPM algorithm for hydrodynamics. However, this approach requires special-purpose coding, and it is not clear if it is extensible to the dimensionally unsplit integrators required for MHD that are implemented in `Athena++`. Nevertheless, exploring such approaches in the future could be important for achieving further performance increases.

Recently, Grete et al. 2019) reported the port of the public version of `Athena++` to GPUs based on the `Kokkos` library Edwards et al. 2014). Figure 3 in their paper explores the ef ciency of the implementation on various architectures. Generally, excellent results are obtained, with between 75 –90 architectural ef ciency on most processors including both CPUs and GPUs. Figure 4 in their paper compares the performance of the resulting code, called `K-ATHENA`, on both Intel CPUs and NVIDIA GPUs. The performance per CPU shown in the right panel of their gure is somewhat lower than the value reported in Table 3 for the same test MHD using PLM reconstruction and the Roe Riemann solver), due to recent optimizations that were not available in the public version they used. Using our values, the ratio of the performance of `K-ATHENA` on the latest NVIDIA Volta GPU to a single Intel Skylake 20 core) CPU performance is about a factor of 5, which is about the same as the ratio of the peak performance for these two architectures. This indicates that despite the limitations of cache performance in `Athena++` inherent in Table 3, overall the code performs extremely well.

3.6.3. Weak Scaling on Uniform Grids

On modern architectures, good parallel scaling is essential to make large calculations feasible. Figure 33 shows the results of weak scaling tests on a Cray XC50 machine containing dual Intel Skylake 6148 processors with 40 cores per node. The test uses a uniform grid with 64^3 cells per `MeshBlock` and one `MeshBlock` per process. The test uses up to 250 nodes 10^4 cores). For the hydrodynamic tests, the HLLC Riemann solver is used. For the MHD tests, the HLLD solver is used, and both use the VL2+PLM integration algorithm. Performance is again measured in zone updates per CPU second per core.

Note the rapid decrease in performance when scaling from 1 to 40 processes, as all cores on a node are used and memory bandwidth limits performance. This behavior reflects the trends already noted in Tables 2 and 3 and discussed above. While improving the cache utilization would likely reduce the memory bandwidth per node limitations evident in Figure 33 as has been achieved in a few other codes; e.g., Woodward et al. 2019), this will require substantial changes to the implementation. Once all cores on a node are utilized, the weak scaling of `Athena++` is essentially perfect. The parallel ef ciencies of the hydrodynamic and MHD simulations between 8 and 250 nodes are about 97 and 95 , respectively. Thus, only a small fraction of the time for the calculation is used for communication costs.

To test the weak scaling and parallel ef ciency on even larger core counts, we have performed another set of weak scaling tests on the Oakforest-PACS supercomputer equipped with Intel Xeon Phi 7250 Knights Landing) multicore processors. We use 64 cores per node 4 cores are left unused to accommodate the operating system and other tasks), and 4 OpenMP threads per process using the `COMPACT` af nity. Each thread owns one `MeshBlock` consisting of $64 \times 32 \times 32$ cells. The results are shown in Figure 34.

Note that even when using 2048 nodes equivalent to 524,288 threads), the parallel ef ciency compared to 8 nodes is 86 for hydrodynamics and 84 for MHD. This excellent scaling is due in part to the ability of the `TaskList` to interleave communications and calculations. The test demonstrates that nite-volume algorithms show excellent scaling up to millions of cores and are highly capable of exploiting emerging resources in the exascale era.

3.6.4. Strong Scaling with AMR

Quantifying the performance of the AMR framework when used with the MHD solver is dif cult, because the amount of work per calculation is highly variable and depends on the re nement criteria, the size of the `MeshBlocks`, and the ef ciency of the implementation. In Section 3.4.3, we discussed

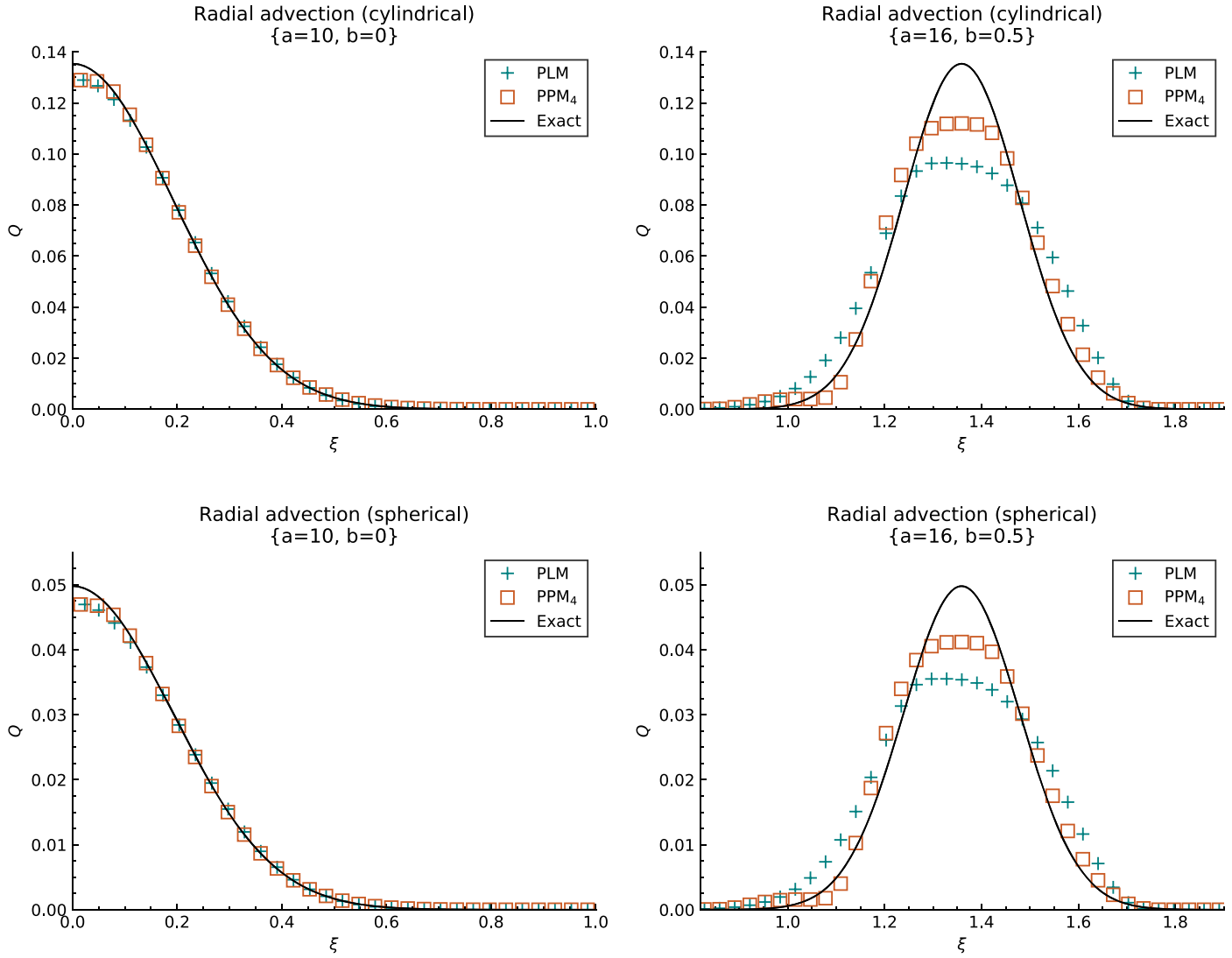

Figure 26 Pro les of the 1D radial advection problems in curvilinear coordinates at $t = 1$ and $N = 64$. The top row plots show cylindrical coordinate solutions, and the bottom row plots show the results in spherical-polar coordinates. The left column is the advection of monotonic initial data, while the right column is advection of a nonmonotonic pro le. Compare to Mignone 2014, Figure 2).

the performance of AMR in terms of reducing the time to solution of some given accuracy compared to a uniform grid for the particular problem of the KHI test. To further quantify the performance of our AMR framework, we measure strong scaling in this section using a different problem.

We use the blast-wave test discussed in Section 3.4.4. Our timing measurements include outputs at every $t = 0.01$. The result is presented in Figure 35. The computational throughput of AMR with 16^3 `MeshBlocks` in terms of cell updates per second is about half of the uniform grid s ef ciency, but its time to solution is about ve times shorter than that of uniform grid. AMR, when used with relatively small 8^3 `MeshBlocks`, has even higher overhead, but its time to solution is as fast as AMR with 16^3 `MeshBlocks`. The short computing time with AMR is not only due to the reduced number of cells in AMR, but also the larger time step by a factor of $\sim$2 because the hot region near the center of the explosion is dere ned.

The optimal choice for the re nement parameters depends on many factors such as the volume of the re ned regions, the size of the root grid, the number of re nement levels, the number of processes, etc., and thus is highly problem dependent. While smaller `MeshBlocks` give more flexibility to adapt to solutions, they are computationally less ef cient. It is reasonable to start with `MeshBlocks` of size 8^3 or 16^3, but ultimately the best choice for each problem must be found through experimentation.

3.6.5. Size of `Mesh locks` and Performance

In order to quantify how the size of `MeshBlocks` affects performance, we have run a series of tests using the 3D blast-wave problem described earlier but with le outputs disabled. In each case, the computational domain is resolved with 128^3 cells, and the CPU time required for solution is measured with `MeshBlocks` ranging in size from 4^3 32,768 `MeshBlocks`) to 128^3 1 `MeshBlock`). All tests were run on a single core of a Skylake 6148 processor. The results are shown in Figure 36, with each point normalized to the CPU time required for the run with a single 128^3 `MeshBlock`. Similar trends are observed in both hydrodynamic and MHD runs, with the CPU time increasing by nearly an order of magnitude as the `MeshBlock` size decreases from 128^3 to 4^3.

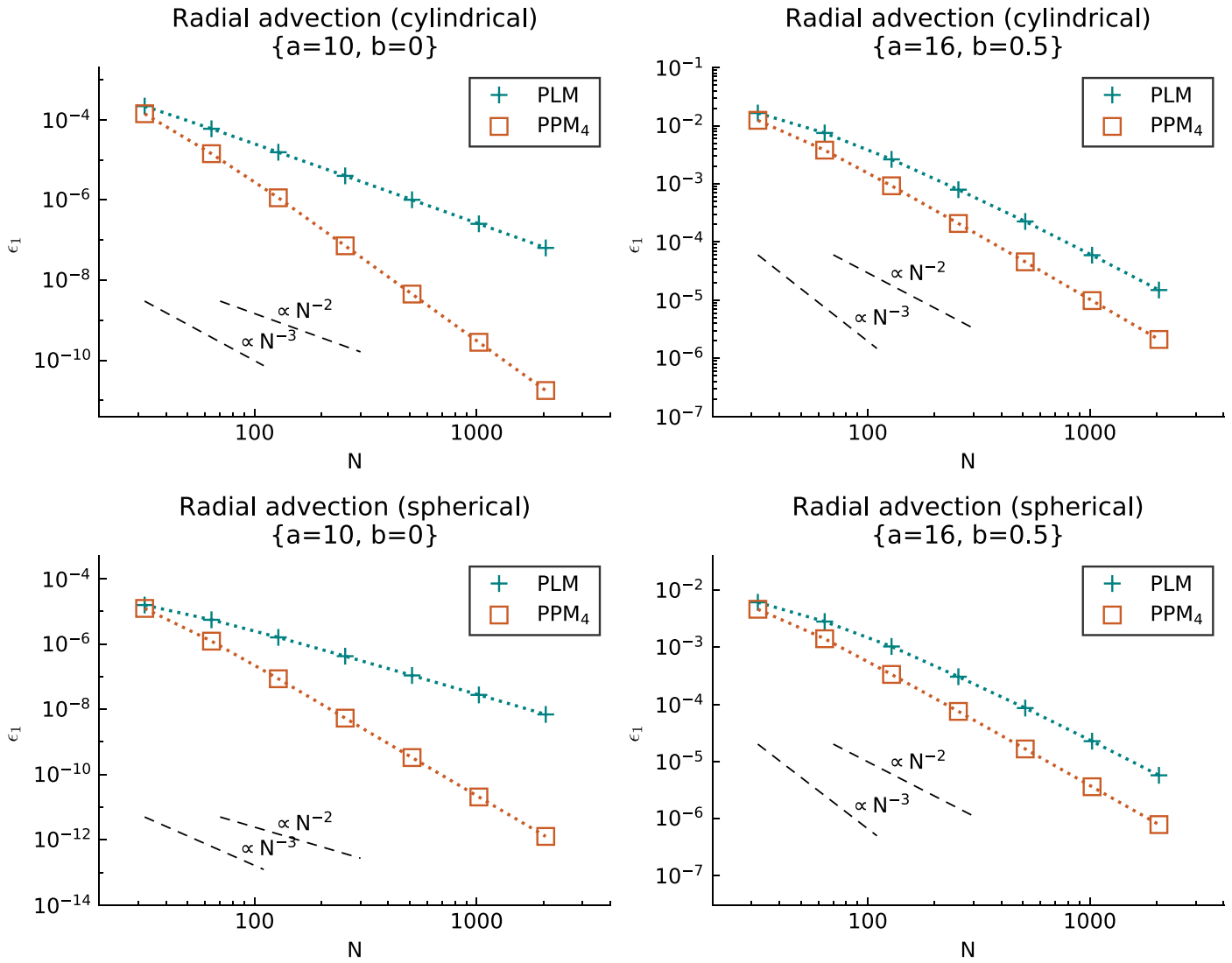

Figure 27 Convergence of the L_1 errors of the 1D radial advection problems: cylindrical and spherical-polar coordinates, monotonic and nonmonotonic data. Compare to Mignone 2014, Figure 3).

This behavior can be explained by a simple model that assumes there are three contributions to the cost of runs with different-size MeshBlocks. The rst part A represents the actual cost to update all of the active cells. Because the total number of cells is xed for all runs, this term does not depend on MeshBlock size. The second part B represents the cost of communications and therefore is proportional to the total surface area of MeshBlocks. Let x be the number of cells in each dimension per MeshBlock. The total number of MeshBlocks is 128 $x)^3$, while the surface area per MeshBlock scales as x^2; therefore, the cost of this second part B $x)$ $\propto x$ ^{1}. Finally, the last part of the model C accounts for the overhead incurred in generating and managing MeshBlocks and therefore is proportional to the total number of MeshBlocks C $x)\propto x$ ^{3}. The total cost W can be expressed as the sum of these three parts:

$$W(x) = A + B + C = A + b/x + c/x^3 \qquad (30)$$

where b and c are constant coef cients. We plot this model, along with each of the three contributing terms A through C, to the measured normalized CPU time shown in Figure 36. We use a least-squares method to t the coef cients in each term, because they cannot be predicted analytically. This simple model can explain the observed performance trends very well. As expected, with large MeshBlocks, the cost is dominated by the actual computation A). However, as the MeshBlock size decreases, the communication cost B) becomes more important, exceeding A around 16^3 in both hydrodynamics and MHD. For MeshBlocks as small as 4^3, the overhead term C) becomes dominant and makes the simulation inef cient.

Although the actual balance between these cost components depends on many factors including the size of the simulation, physics modules in use, CPU and memory performance, parallelization, use of AMR, etc.), this result clearly demonstrates that small MeshBlocks are computationally inef - cient. Therefore, users must choose the optimum MeshBlock size speci c to their problem. For uniform grid simulations, larger MeshBlocks are obviously better. For AMR, it is not trivial to balance performance and flexibility, but somewhere between 8^3 and 16^3 should be reasonable as discussed in the previous section.

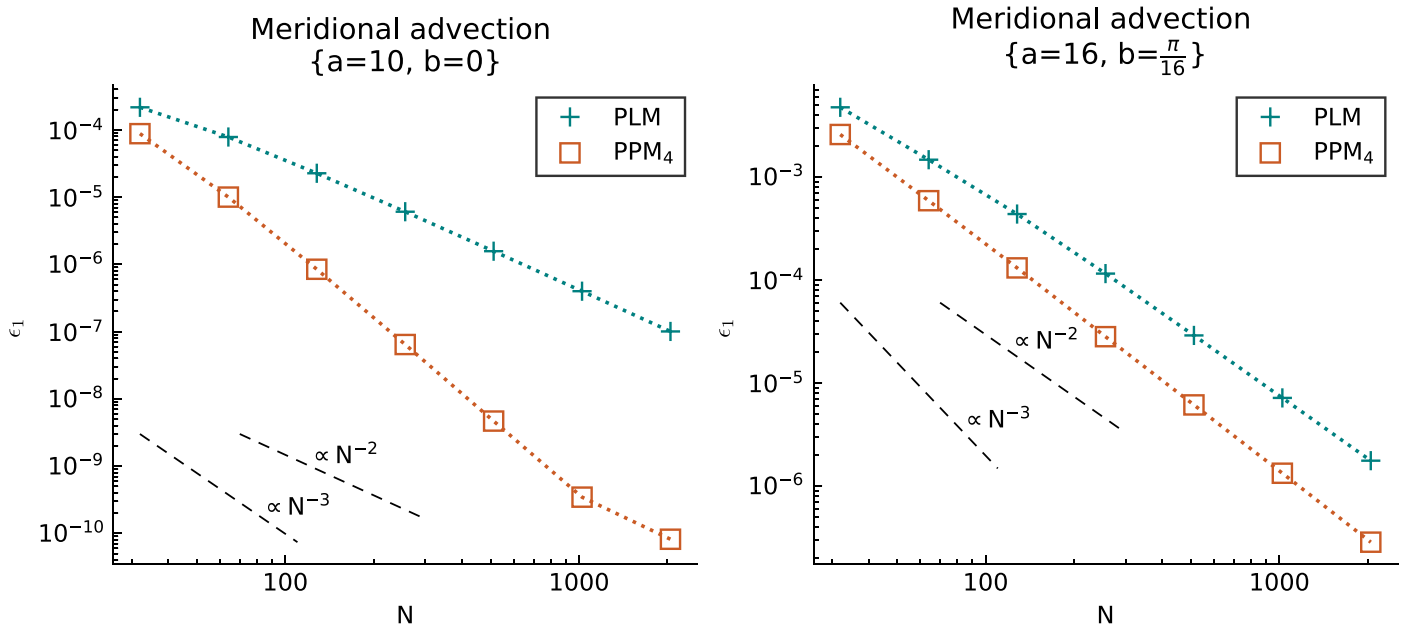

Figure 28 Convergence of the L_1 errors of the polar advection problems in spherical-polar coordinates. Compare to Mignone 2014, Figure 4).

4 A Relativistic MHD Solver

The details of the SR and GR methods have already been presented in White et al. 2016). Here we summarize the important equations and highlight salient differences from Newtonian MHD. In this section, we use units with $c = 1$.

4.1. Equations and Discretization

The differential equations of relativistic MHD can be written in a form similar to those of Newtonian MHD. In SR, the primitive variables are fluid-frame density ρ, fluid-frame gas pressure p_g, spatial part of lab-frame fluid four-velocity $\boldsymbol{u}$, and lab-frame magnetic eld $\boldsymbol{B}$. The Lorentz factor is $\gamma = 1 + \boldsymbol{u}^2)^{1\ 2}$, and the three-velocity is $\boldsymbol{v} = \boldsymbol{u}\ \gamma$. The magnetic pressure is

$$p_m = \frac{1}{2}\left(\frac{1}{\gamma^2}\boldsymbol{B}^2 + (\boldsymbol{v}\cdot\boldsymbol{B})^2\right) \tag{31}$$

and the total enthalpy is

$$w = \rho + \frac{\Gamma}{\Gamma - 1}p_g + 2p_m. \tag{32}$$

Here, Γ is the adiabatic index, taken to be constant. The analogs of Equation 6) are then

$$\frac{\partial D}{\partial t} + \nabla\cdot(D\boldsymbol{v}) = 0, \tag{33a}$$

$$\frac{\partial \boldsymbol{M}}{\partial t} + \nabla\cdot\boldsymbol{S} = 0, \tag{33b}$$

$$\frac{\partial E}{\partial t} + \nabla\cdot\boldsymbol{M} = 0, \tag{33c}$$

$$\frac{\partial \boldsymbol{B}}{\partial t} - \nabla\times(\boldsymbol{v}\times\boldsymbol{B}) = 0. \tag{33d}$$

Here the conserved variables include the lab-frame density, energy, and momentum given by

$$D = \gamma\rho, \tag{34a}$$

$$E = \gamma^2 w - \gamma^2(\boldsymbol{v}\cdot\boldsymbol{B})^2 - (p_g + p_m), \tag{34b}$$

$$\boldsymbol{M} = (E + p_g + p_m)\boldsymbol{v} - (\boldsymbol{v}\cdot\boldsymbol{B})\boldsymbol{B}, \tag{34c}$$

and the stress tensor is

$$\boldsymbol{S} = \gamma^2 w\boldsymbol{v}\boldsymbol{v} - \frac{1}{\gamma^2}\boldsymbol{B}\boldsymbol{B} - (\boldsymbol{v}\cdot\boldsymbol{B})(\boldsymbol{v}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{v}) - \gamma^2(\boldsymbol{v}\cdot\boldsymbol{B})^2\boldsymbol{v}\boldsymbol{v} + (p_g + p_m)\boldsymbol{I}. \tag{35}$$

Given that the forms of Equation 33) are the same as those for Newtonian MHD, the same discretization scheme applies, with cell-centered volume averages and face-centered fluxes of hydrodynamical quantities and with face-centered area averages and edge-centered fluxes of magnetic elds.

With GR, all of our equations acquire a dependence on the metric $\boldsymbol{g}$. The primitive variables in the GRMHD module of `Athena++` are fluid-frame density ρ, fluid-frame gas pressure p_g, normal-frame spatial velocity components $u^{i'}$, and coordinate-frame magnetic eld B^i. The primitive velocities are related to the coordinate-frame velocity components via $u^0 = \gamma\ \alpha$ and $u^i = u^{i'} - \beta^i\gamma/\alpha$, where $\alpha = (-g^{00})^{-1/2}$ is the lapse, $\beta^i = \alpha^2 g^{0i}$ is the shift, and

$$\gamma = (1 + g_{ij}u^{i'}u^{j'})^{1/2} \tag{36}$$

is the Lorentz factor in the normal frame the frame with time direction orthogonal to surfaces of constant time). The contravariant magnetic eld $b^\mu = u_\nu(^*F)^{\nu\mu}$ has components

$$b^0 = u_i B^i, \tag{37a}$$

$$b^i = \frac{1}{u^0}(B^i + b^0 u^i), \tag{37b}$$

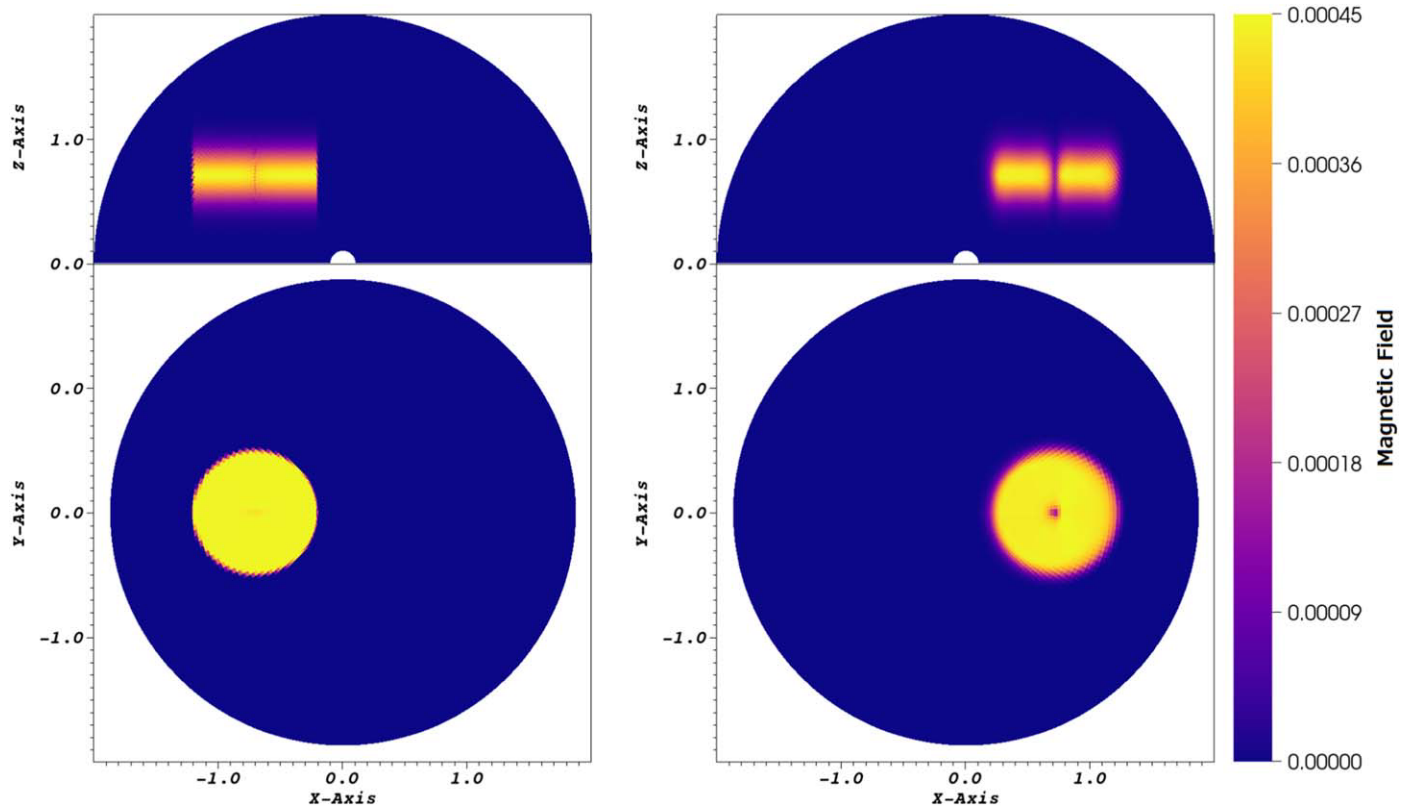

Figure 29 Slices of the magnetic eld strength for the eld loop advection test in spherical-polar coordinates. The left panels indicate the initial condition, while the right panel is the result at $t = \sqrt{2}$. The top panels are vertical cross sections through the $y = 0$ plane, while the bottom panels are horizontal cross sections through the $z = \sqrt{2}/2$ plane. Despite having passed directly through the coordinate singularity at the pole, the loop at the nal time is symmetric.

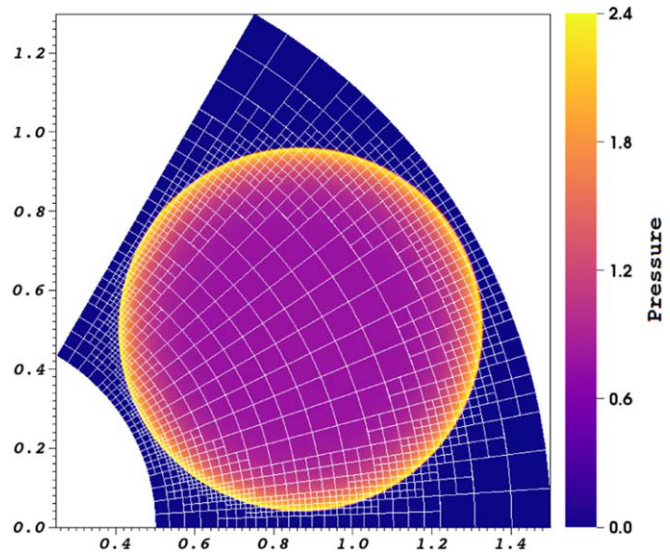

Figure 30 Pressure on a slice at = 0 in the hydrodynamic blast-wave test at $t = 0.1$ in spherical-polar coordinates with AMR. The `MeshBlock` distribution is superimposed as white boxes.

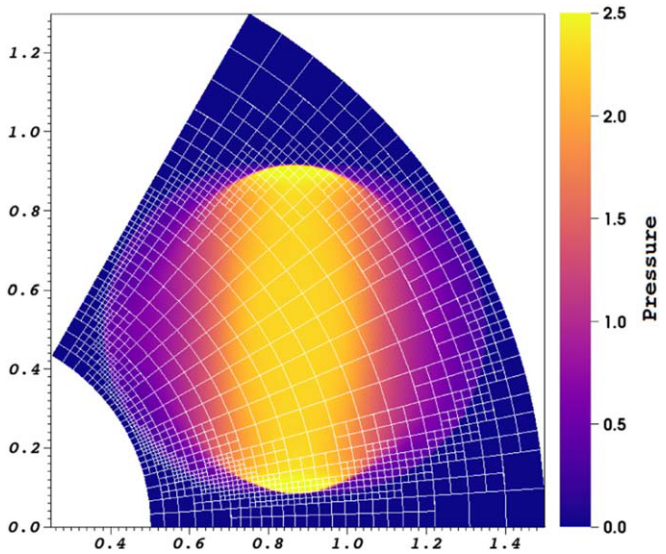

Figure 31 Same as Figure 30 but for the MHD model at $t = 0.08$.

and with this, a magnetic pressure

$$p_{\rm m} = \frac{1}{2} b_\mu b^\mu \tag{38}$$

and total enthalpy

$$w = \rho + \frac{\Gamma}{\Gamma - 1} p_{\rm g} + 2p_{\rm m} \tag{39}$$

can be de ned.

The equations of GRMHD are simply

$$\nabla_\mu (\rho u^\mu) = 0, \tag{40a}$$

$$\nabla_\mu T^\mu{}_\nu = 0, \tag{40b}$$

$$\nabla_\mu ({}^*F)^{\nu\mu} = 0, \tag{40c}$$

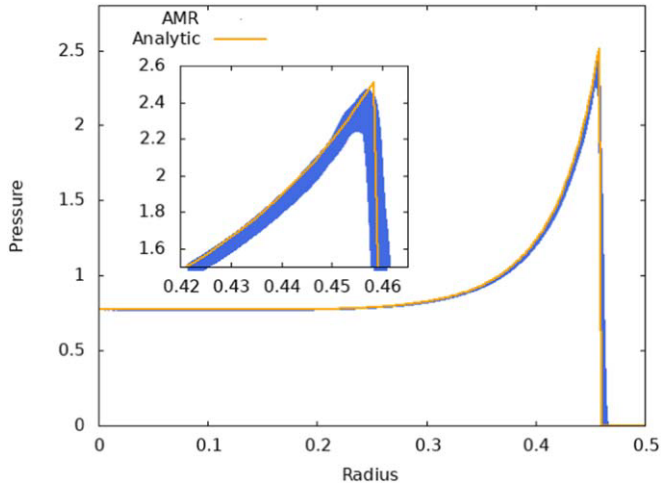

Figure 32 Same as Figure 25, but for the blast-wave test in spherical-polar coordinates.

Table 2
Athena++ Single-node Performance: Single Core

			MZone-cycles s^{-1}		
			Xeon Phi KNL 7250	Broadwell E5-2680 v4	Skylake-SP Gold 6148
Hydro	PLM	HLLC	1.472	3.136	5.227
		HLLE	1.617	3.346	5.814
		Roe	1.520	3.367	5.471
	PPM	HLLC	0.665	1.316	2.527
		HLLE	0.689	1.353	2.643
		Roe	0.674	1.352	2.593
MHD	PLM	HLLD	0.754	1.519	2.924
		HLLE	0.875	1.626	2.757
		Roe	0.689	1.294	2.191
	PPM	HLLD	0.381	0.775	1.559
		HLLE	0.437	0.799	1.512
		Roe	0.347	0.708	1.323

Table 3
Athena++ Single-node Performance: Multicore

			MZone-cycles s^{-1}		
			Xeon Phi KNL 7250	2×) Broadwell E5-2680 v4	2×) Skylake-SP Gold 6148
Hydro	PLM	HLLC	81.992	49.744	84.769
		HLLE	83.110	51.278	87.877
		Roe	79.129	51.425	87.754
	PPM	HLLC	42.554	24.834	49.759
		HLLE	42.804	25.183	50.012
		Roe	42.002	25.242	49.875
MHD	PLM	HLLD	37.953	24.624	44.361
		HLLE	43.139	25.480	43.345
		Roe	35.287	22.045	39.853
	PPM	HLLD	21.024	14.624	28.826
		HLLE	24.090	14.954	28.457
		Roe	19.683	13.657	26.612

where the stress-energy tensor has components

$$T^{\mu}{}_{\nu} = wu^{\mu}u_{\nu} - b^{\mu}b_{\nu} + (p_{\rm g} + p_{\rm m})\delta^{\mu}_{\nu} \tag{41}$$

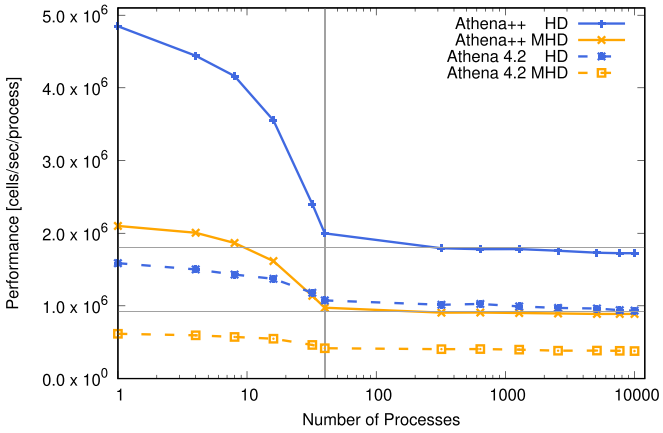

Figure 33 Weak scaling test on Cray XC50 2× Skylake 6148, 40 cores per node). The vertical line indicates one node. To the left of this delimiter, the performance is limited mainly by memory bandwidth, and beyond this, we observe the influence of the network overhead. The gray horizontal lines indicate the Athena++ performance with eight nodes.

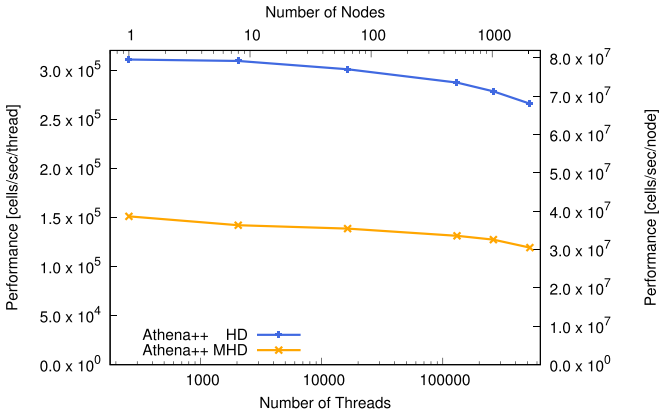

Figure 34 Weak scaling test on Intel Xeon Phi 7250 on the Oakforest-PACS supercomputer.

and the electromagnetic field tensor can be written $(^*F)^{\mu\nu} = b^{\mu}u^{\nu} - b^{\nu}u^{\mu}$. Put into a more useful form, the equations solved by Athena++ are

$$\partial_t(\sqrt{-g}\,\rho u^0) + \partial_j(\sqrt{-g}\,\rho u^j) = 0, \tag{42a}$$

$$\partial_t(\sqrt{-g}\,T^0{}_{\mu}) + \partial_j(\sqrt{-g}\,T^j{}_{\mu}) = \frac{1}{2}\sqrt{-g}\,(\partial_{\mu}g_{\alpha\beta})T^{\alpha\beta}, \tag{42b}$$

$$\partial_t(\sqrt{-g}\,B^i) + \partial_j(\sqrt{-g}\,(^*F)^{ij}) = 0, \tag{42c}$$

where $g = \det \boldsymbol{g}$. The conserved variables are ρu^0, $T^0{}_{\mu}$, and B^i. Again, the equations have the same form and can be discretized as before, as long as the volumes, areas, and lengths used account for the appropriate factors of $\sqrt{-g}$.

Note the source term on the right-hand side of Equation 42 b) also appears in Equation 6 b) when expressing the divergence operator in terms of partial derivatives in non-Cartesian coordinate systems. By choosing the free index in Equation 42 b) to be lowered, the source term vanishes for ignorable coordinates, as noted in Gammie et al. 2003). In practice, this often means the global energy and z-angular momentum are easily conserved to machine precision.

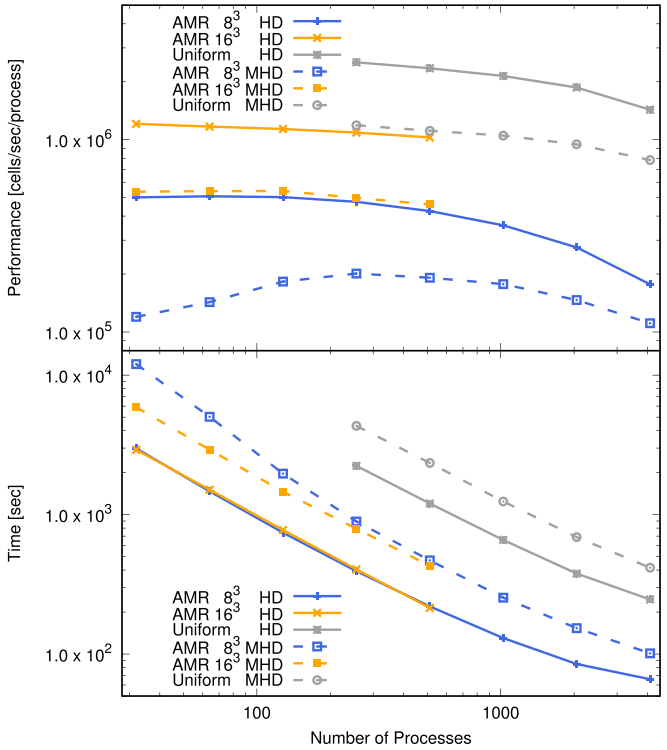

Figure 35 Strong scaling test on Cray XC50 2× Skylake 6148 per node, using only 32 out of 40 cores per node), including le outputs.

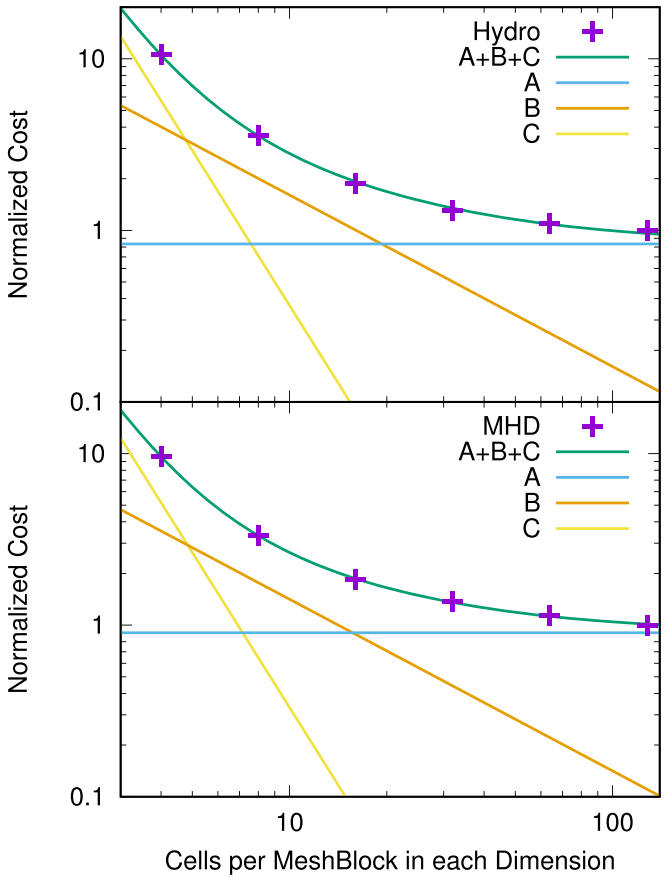

Figure 36 Computational cost for a 3D blast-wave test using `MeshBlocks` of different sizes, normalized to the cost using a single 128^3 `MeshBlock`. The purple crosses are measured results, while the three lines labeled *A* through *C* are different components of a simple model t to the data. See text for details.

4.2. Numerical Algorithms

4.2.1. Reconstruction in Relativistic MHD

In SR and GR, reconstruction is only allowed on the primitive variables. This avoids the numerical expense and potential variable inversion failures associated with characteristic reconstruction. Note also that the choice of primitive velocities ensures there is a unique, physically admissible i.e., subluminal) state of the fluid for any nite real numbers u^i SR) or $u^{i'}$ GR). This would not be true in general were v^i SR) or u^i GR) to be used. `Athena++` as described in White et al. 2016) originally used three-velocities for SR, but we have found the change to spatial four-velocity components makes the code more robust.

4.2.2. Relativistic Riemann Solvers

`Athena++` includes relativistic versions of the HLLE Riemann solver for both pure hydrodynamics and MHD. It also includes the relativistic HLLC solver for hydrodynamics Mignone & Bodo 2005) and HLLD solver for MHD Mignone et al. 2009). The latter two solvers are designed for SR only, but can be used in GR via the local frame transformations described in White et al. 2016).

4.2.3. Variable Inversion

The highly nonlinear, tightly coupled nature of the primitive-conserved variable relations in relativity make nding primitives both expensive requiring iterative solvers that are dif cult to vectorize) and prone to failure, such as when a conserved state has no corresponding subluminal primitives. Noble et al. 2006) catalog six root- nding procedures used for variable inversion—four one-dimensional, one two-dimensional, and one ve-dimensional, with the lower-dimensional versions solving for an enthalpy-like variable and or a velocity-like variable, and with some of them only working for select equations of state. In practice, different methods nd use in modern relativistic codes. For example, `GENESIS` Aloy et al. 1999) and `Enzo` Wang et al. 2008; Bryan et al. 2014) perform a 1D iteration on pressure, the initial implementation of `HARM` uses the 5D method Gammie et al. 2003), `ECHO` del Zanna et al. 2007) uses a velocity-based 1D method, `RAMSES` Teyssier 2002; Lamberts et al. 2012) uses a modi ed enthalpy-based method from Mignone & McKinney 2007), `PLUTO` Mignone et al. 2007; Mignone 2014) uses the enthalpy-based 1D method of Mignone & Bodo 2006), and `BHAC` uses both an enthalpy-based 1D method and a 2D method Porth et al. 2017).

Early versions of `Athena++` employed an enthalpy-based 1D method White et al. 2016). However, we have found inversion to be more robust by adapting the algorithm presented in Newman & Hamlin 2014), which involves a one-dimensional root- nd operation and guarantees that a solution will be found if it exists.

Additionally, robustness relies on the ability to impose appropriate floors and ceilings depending on the problem being solved. The relativistic modules not only put floors on ρ and p_g in a position-dependent way, if desired), but also employ ceilings on γ, $\beta^{-1} = p_m/p_g$, and $\sigma = 2p_m/\rho$. In the latter two cases, the eld components B^i are never altered by ceilings, but rather these constraints are interpreted as additional eld- and velocity-dependent floors on ρ and p_g. We are also exploring

the use of a rst-order flux-correction step, as implemented by Lemaster & Stone 2009, see the Appendix).

4.3. Tests of the Relativistic MHD Module

In the following subsections, we present several tests of the relativistic MHD module in Athena++, focusing especially on the use of mesh re nement with both SR and GR.

4.3.1. Relativistic Shock Tube

Relativistic Riemann problems can be challenging because very thin features can be formed Zhang & MacFadyen 2006, Section 6.1), which are hard to resolve with a uniform mesh. To demonstrate the use of AMR with relativistic MHD, we present results for a strong shock-tube problem using the initial conditions from Mignone et al. 2012, Section 6.1):

$$(\rho, p_g, B^y, B^z) = \begin{cases} (1, 1000, 7, 7), & x < 0.5, \\ (1, 0.1, 0.7, 0.7), & x > 0.5, \end{cases} \tag{43}$$

with $B^x = 10$ and $v^i = 0$ everywhere and with $\Gamma = 5$ 3. The root grid consists of 400 cells divided into MeshBlocks of 16 cells each. The VL2 integrator with PLM reconstruction and the HLLD Riemann solver are used. The CFL number is set to 0.6. The re nement criterion is the maximum value of the curvature on the MeshBlock,

$$g = \max\left(\frac{|q_{i-1} - 2q_i + q_{i+1}|}{q_i}\right), \tag{44}$$

where

$$q = \frac{(B^y)^2 + (B^z)^2}{\gamma\rho}. \tag{45}$$

The re nement and dere nement thresholds are set to 10 ^{3} and 10 ^{4}, and up to 6 levels of re nement beyond the root grid are allowed.

Figure 37 shows the results for this test at time $t = 0.4$. AMR naturally re nes the very thin shell propagating to the right, as well as the steep parts of the rarefaction fans. The results compare favorably to those presented in Mignone et al. 2012, Figure 23). When run on 4 cores of a Skylake Xeon 8160) node, this simulation takes 22.8 core-seconds. The same simulation done at a uniform resolution of 25,600 cells takes 520 core-seconds, so the use of AMR results in a speed-up of a factor of 23 for this test. While this speed-up from mesh re nement is slightly lower than that reported for the PLUTO code Mignone et al. 2012, Table 3), this is in part because the runtime on a uniform mesh is signi cantly lower using Athena++.

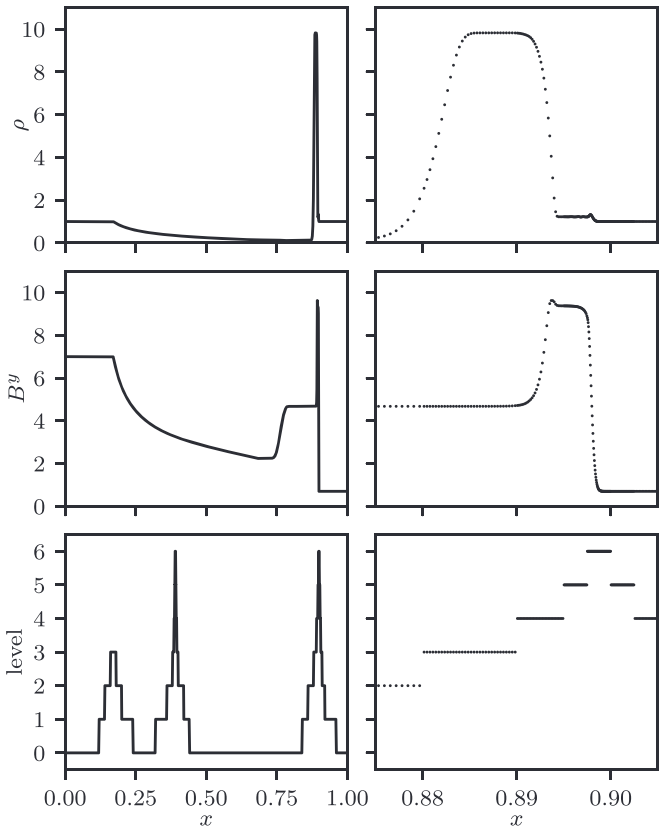

Figure 37 Density, single component of the transverse magnetic eld, and re nement level for the relativistic shock tube. Each cell is represented by a single point on the right. The thin shell consisting of multiple shocks is captured by high levels of re nement, as are the steepest parts of the rarefaction zones.

4.3.2. Relativistic KHI

The ability to simulate relativistic MHD problems with AMR is illustrated in two dimensions with a magnetized KHI problem. The same primitive state as in Mignone et al. 2012) is initialized: the domain has constant values $\rho = 1$, $p_g = 20$, $v^z = 0$, $B^x = \sqrt{2/5}$, $B^y = 0$, and $B^z = 10\sqrt{2/5}$, and the in-plane velocity has the perturbed shear pro le

$$v^x = \frac{1}{4}\tanh(100y), \tag{46a}$$

$$v^y = \frac{1}{400}\sin(2\pi x)\exp(-100y^2). \tag{46b}$$

Here, $\Gamma = 4$ 3 is used in the EOS. Note, Mignone et al. use the Taub–Mathews EOS instead, though the differences are small: the initial enthalpy is $w = 81$ in our case, and $w \approx 80.02$ in theirs.

The domain spans $0 \leqslant x \leqslant 1$ and $0.25 \leqslant y \leqslant 0.25$, with periodic boundary conditions in x and outflowing conditions in y. The root grid consists of 64×32 cells in MeshBlocks of size 16^2. Up to ve levels of re nement beyond the root grid are allowed. Re nement is based on the curvature of the conserved energy in each dimension:

$$g = \max(g_x + g_y), \tag{47a}$$

$$g_x = \frac{|E_{i-i,j} - 2E_{i,j} + E_{i+1,j}|}{E_{i,j}}, \tag{47b}$$

$$g_y = \frac{|E_{i,j-1} - 2E_{i,j} + E_{i,j+1}|}{E_{i,j}}. \tag{47c}$$

The re nement and dere nement thresholds are set to be 10 ^{2} and 10 ^{3}, respectively.

The VL2 integrator, PPM reconstruction, and, separately, the HLLE and HLLD Riemann solvers are used. The simulation is run to a time of $t = 5$, using a CFL number of 0.4. The density and ratio of in-plane to perpendicular magnetic eld strength at the end of the simulation are shown in Figure 38, where the

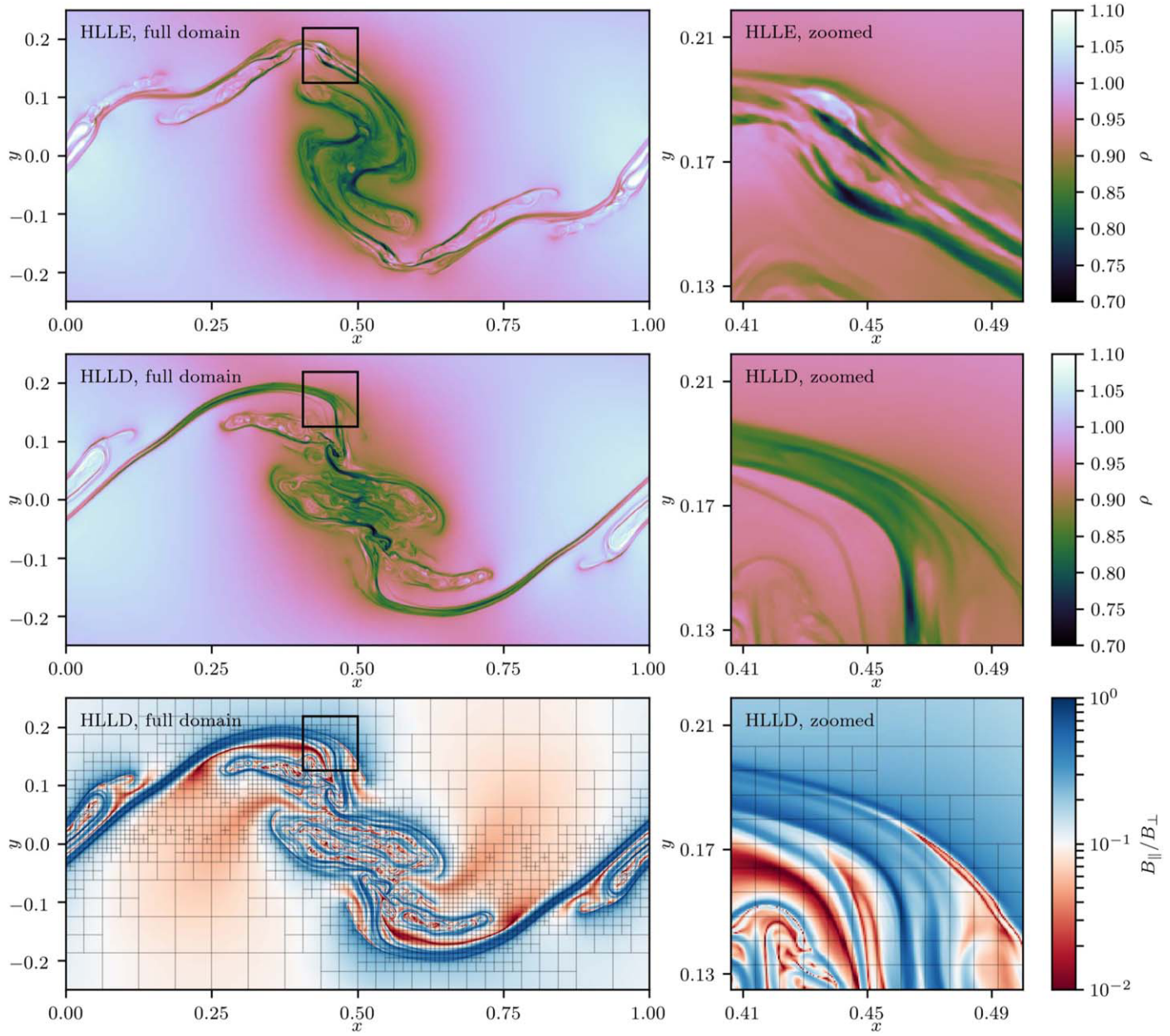

Figure 38 Density and ratio of $B_\parallel = ((B^x)^2 + (B^y)^2)^{1/2}$ to $B_\perp = B^z$ at $t = 5$ in the relativistic magnetized KHI test. The grid lines denote `MeshBlocks` consisting of 16^2 cells. The top row shows results with the HLLE Riemann solver, while the lower rows use HLLD. The left panels show the full domain with re nement levels 2 through 5 present, while the right panels zoom in to the region with a black border, with re nement levels 4 and 5.

re ned regions can be seen to track the locations of small-scale structures.

There are some differences between Figure 38 and Figure 31 of Mignone et al. 2012). While some of these may be attributable to the different EOSs, we also note that the result at the end of the simulation depends strongly on details of the numerical algorithms employed. For example, when the same test is performed with two different Riemann solvers but all else being equal, the locations and shapes of even the largest KHI rolls shift e.g., compare the top and middle panels in Figure 38). In fact, many of the short-wavelength features in the solution are introduced by changes in resolution at ne coarse boundaries. For example, Figure 39 shows the ratio of

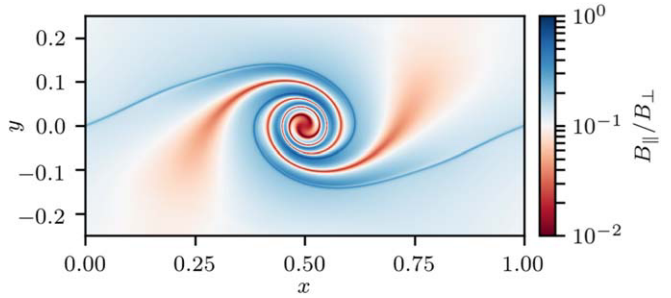

Figure 39 Ratio of $B_\parallel = ((B^x)^2 + (B^y)^2)^{1/2}$ to $B_\perp = B^z$ at $t = 5$ in the relativistic magnetized KHI test on a uniform grid. Compare to the bottom panel in Figure 38.

the parallel and perpendicular components of the magnetic eld in the same problem run on a uniform grid with a resolution of 4096×2048 twice the effective resolution at the highest re nement level in the AMR calculation). In this case, the vortex produced by the instability is smooth. Therefore, we conclude that most of the complex features visible in Figure 38 are due to the AMR boundaries, and this in part contributes to the difference between these solutions and those shown in Mignone et al. 2012).

In the AMR runs, the re nement criterion partially re nes the root grid before time evolution begins, and so the run begins with 176 `MeshBlocks`. Over the course of the HLLD simulation, 1174 `MeshBlocks` are re ned and 177 coarser blocks are created from ner ones. The AMR simulation takes 32.2 core-hours to run on 2 KNL nodes Xeon Phi 7250, 68 cores each), while the same problem run on a uniform 2048×1024 grid takes 251 core-hours. Thus, using AMR gives a speed-up of 7.8.

4.3.3. Relativistic Magnetized Blast Wave

A further test of the relativistic MHD module in the code is provided by the evolution of a magnetized blast wave. Variations on this test are commonly used to test the propagation of strong shocks in relativistic MHD codes, for example, in Komissarov 1999), Leismann et al. 2005), del Zanna et al. 2007), Beckwith & Stone 2011), and Mignone et al. 2012).

We rst run a strongly magnetized blast in two dimensions on a Cartesian grid, with the magnetic eld not aligned with the grid. On a domain 6, 6]^2, we have initial values $v^i = 0$, $B^x = 1/\sqrt{20}$, $B^y = 1/\sqrt{20}$, and $B^z = 0$. The density ρ is 10^{-2} within a distance $r = 0.8$ of the origin, 10^{-4} outside $r = 1$, and it varies linearly with radius between these circles. p_g varies from 1 to 5×10^{-3}. $\Gamma = 4/3$ for this test.

The simulation is evolved to a time of $t = 4$ using a CFL number of 0.25, PLM reconstruction, and the HLLD Riemann solver. We use both a uniform grid with 1536^2 cells and an AMR grid with 48^2 cells at root level. In both cases, `MeshBlocks` with 16^2 cells are used. The AMR grid can have up to ve additional levels of re nement. Re nement is triggered with the same curvature condition in Equation 47) as in the previous test, except using conserved density D instead of energy. The thresholds are set to be 0.025 and 0.005.

The upper panel of Figure 40 shows the gas pressure at the end of the AMR simulation. Re nement tracks the shock fronts that are directed by the magnetic eld. The lower panel shows the relative difference in conserved energy between the simulations. In most of the volume, the agreement is better than 1 .

The uniform simulation takes 24.7 core-hours on 2 KNL nodes Xeon Phi 7250, 68 cores each), while the AMR simulation takes 6.18 core-hours. Thus AMR gives us a factor of 4.0 speed-up.

We next run a similar but spherical test in three dimensions, using the same physical parameters as in Mignone et al. 2012). The domain is 6, 6]^3, with initial values $v^i = 0$, $B^x = 1/\sqrt{200}$, $B^y = 1/\sqrt{200}$, and $B^z = 0$. Density and pressure are the same as in the 2D case, and again we have $\Gamma = 4/3$.

The simulation is evolved to a time of $t = 4$ using a CFL number of 0.25, PLM reconstruction, and the HLLD Riemann solver, using both a uniform grid with 768^3 cells and an AMR grid with 48^3 cells at root level. In both cases, `MeshBlocks` with 16^3 cells are used. The AMR grid can have up to four additional levels of re nement. The curvature condition for re nement is extended naturally to 3D, with thresholds set to be 0.15 and 0.03.

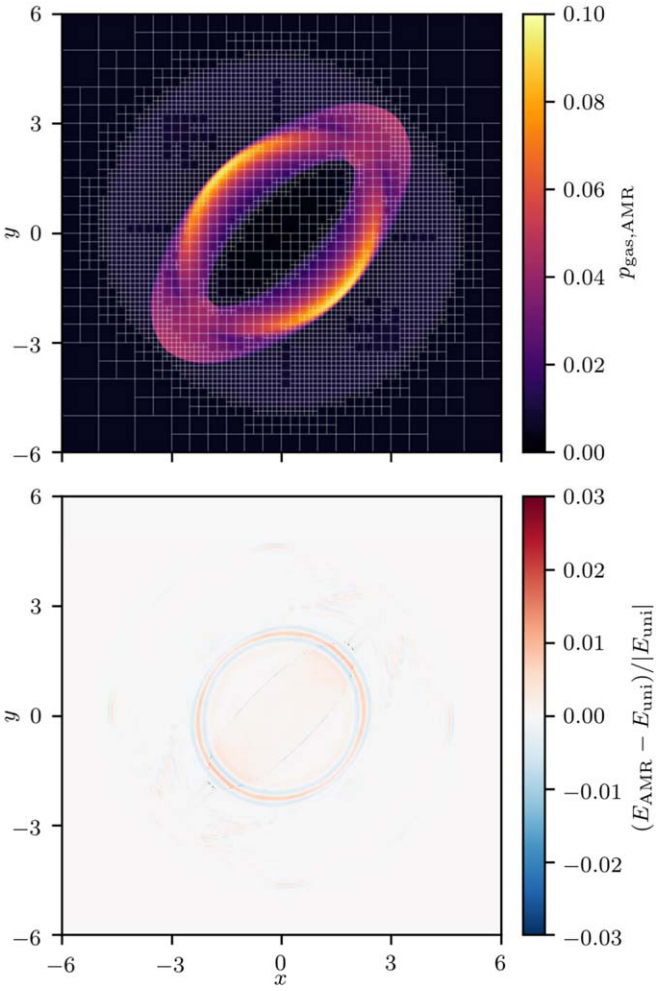

Figure 40 Top: gas pressure in the 2D strongly magnetized relativistic blast test using AMR. The grid lines denote `MeshBlocks` consisting of 16^2 cells, and re nement levels 2 through 5 are present. Bottom: relative difference in conserved energy between the AMR grid and a uniform grid.

Figure 41 shows the gas pressure in the $z = 0$ slice at the end of the simulation. As in the 2D case, we see re nement tracking the shock fronts. Again, the agreement is better than 1 over most of the volume, with most of the relative error in the interior, which has been evacuated to near the density and pressure floors.

The uniform simulation takes 2720 core-hours on 16 KNL nodes, while the AMR simulation takes 316 core-hours. Thus AMR gives us a factor of 8.6 speed-up.

4.3.4. Black Hole Accretion

As a demonstration of the general-relativistic capabilities of `Athena++`, we show the evolution of a weakly magnetized, hydrostatic equilibrium torus around a spinning black hole. The initial conditions are those of Fishbone & Moncrief 1976) with dimensionless spin $a = 0.9$, inner edge at $r = 15r_g$, and pressure maximum at $r = 25r_g$, where $r_g = GM/c^2$ is the characteristic length scale of a black hole of mass M. The

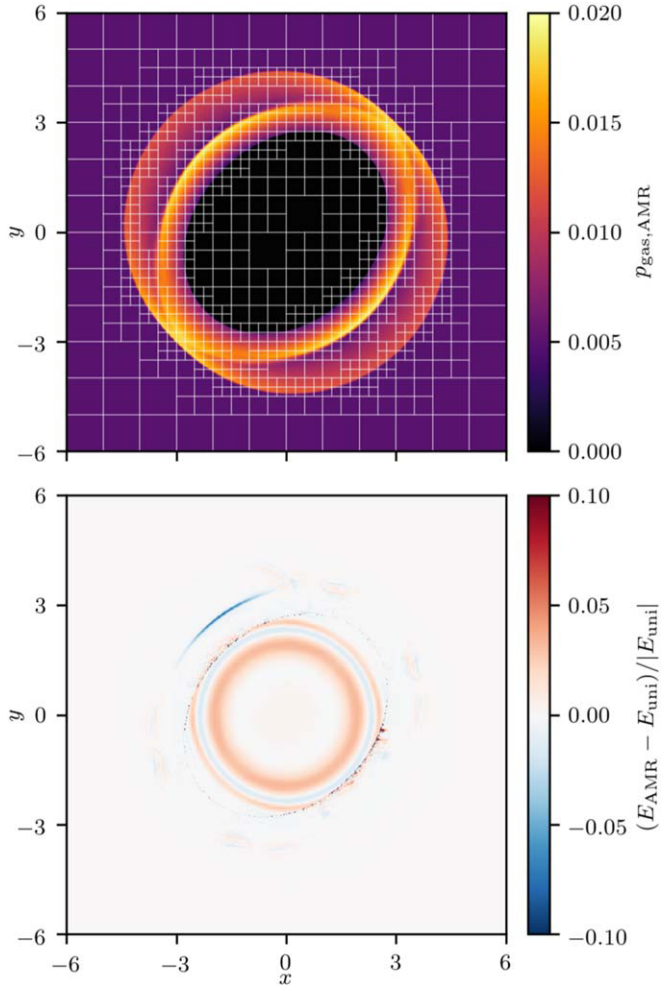

Figure 41 Top: midplane $z = 0$ slice of gas pressure in the 3D relativistic magnetized blast test using AMR. The grid lines denote `MeshBlocks` consisting of 16^3 cells. Re nement levels 2 through 4 are present. Bottom: relative difference in conserved energy between the simulations.

magnetorotational instability Balbus & Hawley 1991) is seeded with a single magnetic eld loop in the poloidal plane, normalized such that the mass-weighted average of β^{-1} is 0.01.

We evolve the torus in horizon-penetrating, spherical Kerr–Schild coordinates. Our root grid has $56 \times 32 \times 44$ cells in r, θ, and . Cells are spaced logarithmically in radius, from $r = 1.329r_{\rm g}$ to $r = 100r_{\rm g}$, and they are uniform in both angles. SMR adds three successive re nement levels away from the polar axis, achieving an effective resolution of $448 \times 256 \times 352$ everywhere within 50°.625 of the midplane while still keeping cells from being unnecessarily small near the axis.

The density after a time of $10{,}000r_{\rm g}/c$ is shown in Figure 42. By this point, the turbulence has saturated, and inflow equilibrium has been achieved in the inner parts of the thick disk that has formed. The evolution of accretion flows such as this, at similar resolutions, is ubiquitous in the black hole modeling community, and it is used as a test of codes GRMHD capabilities see Porth et al. 2019, including a comparison of `Athena++` with other codes).

5 Additional Physics

In this paper, we have described in detail modules for nonrelativistic and relativistic MHD that have already been implemented in the AMR framework. These modules include additional physics for MHD, including nonideal MHD, a general EOS, the shearing-box approximation, and orbital advection. Below we describe some of the additional physics that will be available in new modules in the future.

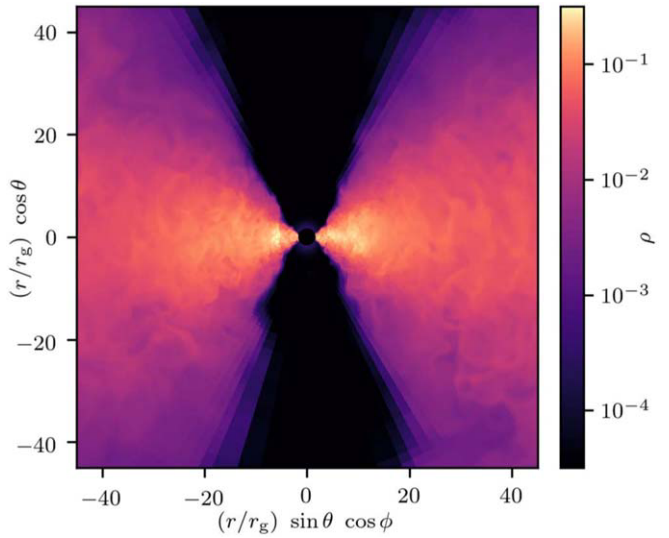

Figure 42 Poloidal slice of density in the GR torus demonstration. Turbulence is fully developed after 10,000 gravitational times $r_{\rm g}/c$.

Self-gravity. Two different methods are implemented for self-gravity. The rst solves the Poisson equation using fast Fourier transforms FFTs), following the method implemented in `Athena`. This module is included in the public version. There is also a new implementation of self-gravity based on the solution of the Poisson equation using the full multigrid FMG) method, which is more ef cient and scalable than FFTs FMG is $O\ N$) while FFTs are $O(N \log N)$). The new FMG solver is being extended to work with AMR K. Tomida et al. 2020, in preparation). In addition, the FMG solver is designed to be flexible so that it can be used for a variety of applications, for example, solving implicit discretizations of the radiation transfer moment equations as in Jiang et al. 2012).

Radiation transfer. A variety of modules for incorporating radiation transfer into MHD calculations are being developed. The time-dependent radiation transport algorithm described in Jiang et al. 2014a) has already been implemented and has been used to study a variety of problems in radiation-dominated accretion disks Jiang et al. 2014b) and massive stars Jiang et al. 2018). This module is being extended to full GR. Moment-based methods such as flux-limited diffusion and the variable Eddington tensor VET) method Davis et al. 2012; Jiang et al. 2012) will also be implemented. For postprocessing calculations that compute synthetic images and spectra, a Monte Carlo–based radiation transfer solver is also under development S. Davis et al. 2020, in preparation). A method for following radiation from point sources using adaptive ray tracing has been implemented in `Athena` Kim et al. 2017) and will be reimplemented in `Athena++`.

Reaction networks. A module to solve chemical reaction networks is implemented using a publicly available sparse matrix solver M. Gong et al. 2020, in preparation). Nuclear reaction networks are also being implemented using the same algorithmic infrastructure G. Halevi et al. 2020, in preparation). When coupled with the passive scalar capabilities described in Section 3.2.6, `Athena++` becomes capable of

solving chemohydrodynamics. When these solvers are coupled with the radiation transfer module and the general EOS capabilities, they enable new studies of a diverse set of problems from the dynamics of the multiphase interstellar medium ISM) to the merger of compact objects.

Dust Particle Dynamics: To simulate particle motions coupled with hydrodynamics, a general particle module is under development C.-C. Yang et al. 2020, in preparation), based on the methods implemented in Athena and described in Bai & Stone 2010). The module will enable calculations of the dynamics of dust particles in planet formation, the kinematics of tracer particles to diagnose flow, and the use of sink particles to represent stars and compact objects.

6 Summary and Conclusion

In this paper, we have described a new framework for AMR as implemented in the Athena++ code. This framework adopts a block-based AMR design, with blocks organized into a tree data structure, for improved performance, scalability, and ease of implementation. It can be used with any logically rectangular coordinates and with nonuniform mesh spacing. We also describe a dynamic execution model based on a simple design we call a task list. This model is capable of overlapping communication with computation on distributed-memory parallel systems, which helps improve the parallel ef ciency and scalability of the algorithms on very large numbers of processors. Moreover, different combinations of physics can be included in calculations by simply adding new steps to the task list. Finally, because different regions of the calculation can have different task lists, it is even straightforward to implement multiphysics calculations that include different physics in different locations such as kinetic MHD in dense regions of a plasma that are weakly collisional, and PIC methods in diffuse regions that are collisionless). The task list could also be used to solve different physics on different physical cores in a parallel calculation; for example, the Poisson equation for self-gravity could be solved on different cores from those dedicated to hydrodynamics or MHD.

We have also described two physics modules that have been implemented in this framework, for nonrelativistic and relativistic MHD. These modules are based on the numerical algorithms for MHD developed in the Athena code S08) using a nite-volume discretization combined with the CT algorithm to enforce the divergence-free constraint on the magnetic eld. They have been updated with new algorithmic extensions, such as higher-order reconstruction in curvilinear and or nonuniform meshes, new higher-order time integrators based on a method of lines approach, and diffusive terms that can be updated using new Runge–Kutta–Legendre super-time-stepping methods. Most importantly, these modules for MHD work effectively with AMR. A variety of test problems were presented to show the accuracy and delity of the MHD algorithms with AMR; see S08 and White et al. 2016) for a more comprehensive list of tests we have used to validate the algorithms.

A signi cant aspect of this new framework is excellent performance and parallel scaling. The MHD solvers have been highly optimized to exploit vector instructions on modern processors. Based on tests run with the public versions of several MHD codes built using the same compilers and optimizations, the performance of the Athena++ MHD module is among the highest of any publicly available astrophysical MHD code of which we are aware, with only the DISPATCH code being similar Nordlund et al. 2018). Using all cores on a single Intel Skylake CPU, the performance is only about 5× slower than the same algorithm implemented on the latest NVIDIA Volta GPU Grete et al. 2019; as expected given the ratio of peak performance for these two devices). On up to 500,000 threads, the MHD module shows excellent weak scaling, with 84 parallel ef ciency. Thus, the nite-volume algorithms implemented in Athena++ are clearly capable of exploiting the new hardware emerging in the exascale era.

A variety of new physics modules are under development, including self-gravity, radiation transfer, chemical and nuclear reaction networks, and particles coupled to the fluid. In addition, improvements to the algorithms in existing modules is planned. For example, a fully fourth-order accurate algorithm for MHD has been implemented in the Athena++ framework Felker & Stone 2018) and is currently being tested and compared with existing algorithms in the code on astrophysical applications to determine the relative advantages and disadvantages of each. Finally, a performance-portable version of the entire Athena++ AMR framework is being built using the Kokkos library J. Dolence et al. 2020, private communication) and will be released as open source in the near future.

Athena++ is publicly available through a GitHub repository and is distributed under the BSD open-source license. Once new modules are thoroughly tested and deemed reliable, they will also be made publicly available. While the code has been developed primarily to enable scienti c applications by the core members of the development team, it is hoped that others will nd the code useful.

We thank the many contributors to the Athena++ code project, especially Matt Coleman, Shane Davis, Munan Gong, Goni Halevi, Yan-Fei Jiang, Chang-Goo Kim, Alwin Mao, Patrick Mullen, Tomohiro Ono, Ji-Ming Shi, Bei Wang, Chao-Chin Yang, and Zhaohuan Zhu. We also thank the referee for comments which improved the manuscript.

J.M.S. was supported by National Science Foundation, grant number AST-1715277. K.T. was supported by Japan Society for the Promotion of Science JSPS) KAKENHI grant numbers 16H05998, 16K13786, 17KK0091, and 18H05440. K.T. also acknowledges support by Ministry of Education, Culture, Sports, Science and Technology MEXT) of Japan as "Exploratory Challenge on Post-K Computer" Elucidation of the Birth of Exoplanets Second Earth] and the Environmental Variations of Planets in the Solar System). C.J.W. was supported in part by the National Science Foundation, grant number PHY-1748958. K.G.F. was supported by the Department of Energy Computational Science Graduate Fellowship CSGF), grant number DE-FG02-97ER25308. The initial design and development of Athena++ was undertaken while J.M.S. was a Simons Distinguished Visiting Scholar at the KITP, University of California Santa Barbara, in 2014; the support of the KITP is gratefully acknowledged.

The simulations presented in this article were performed partly on computational resources managed and supported by Princeton Research Computing, a consortium of groups including the Princeton Institute for Computational Science and Engineering PICSciE) and the Of ce of Information Technology at Princeton University. We thank David Luet for his support of the Jenkins server at PICSciE. This work also

used the Oakforest-PACS supercomputer at the Joint Center for Advanced High Performance Computing through the HPCI System Research Project Project IDs: hp180128, hp190088); the Cray XC50 supercomputer at Center for Computational Astrophysics, National Astronomical Observatory of Japan; and the Extreme Science and Engineering Discovery Environment XSEDE) Stampede2 at the Texas Advanced Computing Center allocation: AST170012).

ORCID iDs

James M. Stone https: orcid.org 0000-0001-5603-1832
Kengo Tomida https: orcid.org 0000-0001-8105-8113
Kyle G. Felker https: orcid.org 0000-0002-3501-482X

References

Aloy, M., Ibáñez, J., Martí, J., & Müller, E. 1999, ApJS, 122, 151
Bai, X.-N., & Stone, J. M. 2010, ApJS, 190, 297
Balbus, S. A., & Hawley, J. F. 1991, ApJ, 376, 214
Beckwith, K., & Stone, J. M. 2011, ApJS, 193, 6
Benítez-Llambay, P., & Masset, F. S. 2016, ApJS, 223, 11
Berger, M. J., & Colella, P. 1989, JCoPh, 82, 64
Berger, M. J., & Oliger, J. 1984, JCoPh, 53, 484
Blondin, J. M., & Lufkin, E. A. 1993, ApJS, 88, 589
Brio, M., & Wu, C. C. 1988, JCoPh, 75, 400
Bryan, G. L., Norman, M. L., O Shea, B. W., et al. 2014, ApJS, 211, 19
Burns, K. J., Vasil, G. M., Oishi, J. S., Lecoanet, D., & Brown, B. P. 2020, PhRvD, 2, 023068
Chen, Z., Coleman, M. S., Blackman, E. G., & Frank, A. 2019, JCoPh, 388, 490
Colella, P. 1990, JCoPh, 87, 171
Colella, P., & Sekora, M. D. 2008, JCoPh, 227, 7069
Colella, P., & Woodward, P. R. 1984, JCoPh, 54, 174
Coleman, M. S. B. 2020, ApJS, 248, 7
Davis, S. W., Stone, J. M., & Jiang, Y.-F. 2012, ApJS, 199, 9
del Zanna, L., Zanotti, O., Bucciantini, N., & Londrillo, P. 2007, A&A, 473, 11
Edwards, H. C., Trott, C. R., & Sunderland, D. 2014, JPDC, 74, 3202
Felker, K. G. 2019, PhD thesis, Princeton Univ.
Felker, K. G., & Stone, J. M. 2018, JCoPh, 375, 1365
Fishbone, L. G., & Moncrief, V. 1976, ApJ, 207, 962
Fryxell, B., Olson, K., Ricker, P., et al. 2000, ApJS, 131, 273
Gammie, C. F., McKinney, J. C., & Tóth, G. 2003, ApJ, 589, 444
Garcia, A. L., Bell, J. B., Crutch eld, W. Y., & Alder, B. J. 1999, JCoPh, 154, 134
Gardiner, T. A., & Stone, J. M. 2005, JCoPh, 205, 509
Gardiner, T. A., & Stone, J. M. 2008, JCoPh, 227, 4123
Gittings, M., Weaver, R., Clover, M., et al. 2008, CS&D, 1, 015005
Gnedin, N. Y., Semenov, V. A., & Kravtsov, A. V. 2018, JCoPh, 359, 93
Gong, H., & Ostriker, E. C. 2013, ApJS, 204, 8
Gottlieb, S., Ketcheson, D. I., & Shu, C.-W. 2009, JSCom, 38, 251
Grete, P., Glines, F. W., & O Shea, B. W. 2009, arXiv:1905.04341
Hardee, P. E. 1979, ApJ, 234, 47
Hayes, J. C., Norman, M. L., Fiedler, R. A., et al. 2006, ApJS, 165, 188
Hui, W., Li, P., & Li, Z. 1999, JCoPh, 153, 596
Jiang, Y.-F., Belyaev, M., Goodman, J., & Stone, J. M. 2013, NewA, 19, 48
Jiang, Y.-F., Cantiello, M., Bildsten, L., et al. 2018, Natur, 561, 498
Jiang, Y.-F., Stone, J. M., & Davis, S. W. 2012, ApJS, 199, 14
Jiang, Y.-F., Stone, J. M., & Davis, S. W. 2014a, ApJS, 213, 7
Jiang, Y.-F., Stone, J. M., & Davis, S. W. 2014b, ApJ, 796, 106
Johnson, B. M., Guan, X., & Gammie, C. F. 2008, ApJS, 177, 373
Keppens, R., Nool, M., Tóth, G., & Goedbloed, J. P. 2003, CoPhC, 153, 317
Ketcheson, D. I. 2010, JCoPh, 229, 1763
Kim, J.-G., Kim, W.-T., Ostriker, E. C., & Skinner, M. A. 2017, ApJ, 851, 93
Komissarov, S. 1999, MNRAS, 303, 343
Kravtsov, A. V., Klypin, A. A., & Khokhlov, A. M. 1997, ApJS, 111, 73
Lamberts, A., Fromang, S., Dubus, G., & Teyssier, R. 2012, A&A, 560, A79
Lecoanet, D., McCourt, M., Quataert, E., et al. 2015, MNRAS, 455, 4274
Leismann, T., Antón, L., Aloy, M., et al. 2005, A&A, 436, 503
Lemaster, M. N., & Stone, J. M. 2009, ApJ, 691, 1092
Liska, R., & Wendroff, B. 2003, SIAM J. Sci. Comput., 25, 995
MacNeice, P., Olson, K. M., Mobarry, C., de Fainchtein, R., & Packer, C. 2000, CoPhC, 126, 330
Masset, F. 2000, A&AS, 141, 165
Masson, J., Teyssier, R., Mulet-Marquis, C., Hennebelle, P., & Chabrier, G. 2012, ApJS, 201, 24
Matsumoto, T. 2007, PASJ, 59, 905
McCorquodale, P., Dorr, M., Hittinger, J., & Colella, P. 2015, JCoPh, 288, 181
McNally, C. P., Lyra, W., & Passy, J.-C. 2012, ApJS, 201, 18
Meyer, C. D., Balsara, D. S., & Aslam, T. D. 2012, MNRAS, 422, 2102
Meyer, C. D., Balsara, D. S., & Aslam, T. D. 2014, JCoPh, 257, 594
Mignone, A. 2014, JCoPh, 270, 784
Mignone, A., & Bodo, G. 2005, MNRAS, 364, 126
Mignone, A., & Bodo, G. 2006, MNRAS, 368, 1040
Mignone, A., Bodo, G., Massaglia, S., et al. 2007, ApJS, 170, 228
Mignone, A., Flock, M., Stute, M., Kolb, S. M., & Muscianisi, G. 2012, A&A, 545, A152
Mignone, A., & McKinney, J. C. 2007, MNRAS, 378, 1118
Mignone, A., Ugliano, M., & Bodo, G. 2009, MNRAS, 393, 1141
Mignone, A., Zanni, C., Tzeferacos, P., et al. 2012, ApJS, 198, 7
Miller, K. A., & Stone, J. M. 2000, ApJ, 534, 398
Newman, W. I., & Hamlin, N. D. 2014, SIAM J. Sci. Comput., 36, B661
Noble, S. C., Gammie, C. F., McKinney, J. C., & del Zanna, L. 2006, ApJ, 641, 626
Nordlund, Å., Ramsey, J. P., Popovas, A., & Küffmeier, M. 2018, MNRAS, 477, 624
Oishi, J. S., Brown, B. P., Burns, K. J., Lecoanet, D., & Vasil, G. M. 2018, arXiv:1801.08200
Porth, O., Olivares, H., Mizuno, Y., et al. 2017, ComAC, 4, 1
Porth, O., Chatterjee, K., Narayan, R., et al. 2019, ApJS, 243, 26
Ryu, D., Jones, T. W., & Frank, A. 1995, ApJ, 452, 785
Sedov, L. I. 1946, JApMM, 10, 241
Shu, C.-W., & Osher, S. 1989, JCoPh, 83, 32
Skinner, M. A., & Ostriker, E. C. 2010, ApJS, 188, 290
Skinner, M. A., & Ostriker, E. C. 2013, ApJS, 206, 21
Sod, G. A. 1978, JCoPh, 27, 1
Stodden, V., & Miguez, S. 2014, JORS, 2, 21
Stone, J. M., & Gardiner, T. 2009, NewA, 14, 139
Stone, J. M., & Gardiner, T. A. 2010, ApJS, 189, 142
Stone, J. M., Gardiner, T. A., Teuben, P., Hawley, J. F., & Simon, J. B. 2008, ApJS, 178, 137
Stone, J. M., Mihalas, D., & Norman, M. L. 1992, ApJS, 80, 819
Stone, J. M., & Norman, M. L. 1992a, ApJS, 80, 753
Stone, J. M., & Norman, M. L. 1992b, ApJS, 80, 791
Stout, Q. F., de Zeeuw, D. L., Gombosi, T. I., et al. 1997, in Proc. EEE Conf. on Supercomputing, SC 97 New York: ACM), 1, doi:10.1145 509593. 509650
Taylor, G. I. 1950, RSPSA, 201, 159
Teyssier, R. 2002, A&A, 385, 337
Tomida, K., Okuzumi, S., & Machida, M. N. 2015, ApJ, 801, 117
Tóth, G., & Roe, P. 2002, JCoPh, 180, 736
Turk, M. J. 2013, in Proc. Conf. on Extreme Science and Engineering Discovery Environment Gateway to Discovery, XSEDE 13, ed. N. Wilkins-Diehr New York: ACM) doi:10.1145 2484762)
van Leer, B. 1974, JCoPh, 14, 361
Wang, P., Abel, T., & Zhang, W. 2008, ApJS, 176, 467
Wardle, M., & Ng, C. 1999, MNRAS, 303, 239
White, C. J., Stone, J. M., & Gammie, C. F. 2016, ApJS, 225, 22
Woodward, P., & Colella, P. 1984, JCoPh, 54, 115
Woodward, P. R., Lin, P.-H., Mao, H., Andrassy, R., & Herwig, F. 2019, J. Phys. Conf. Ser., 1225, 012020
Zhang, W., & MacFadyen, A. I. 2006, ApJS, 164, 255
Zhang, W., Almgren, A., Beckner, V., et al. 2019, JOSS, 4, 1370
Ziegler, U. 2008, CoPhC, 179, 227