IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

gPLUTO: an exa-scale ready code for computational plasma astrophysics

This is a pre print version of the following article:

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/2125650 since 2026-02-18T07:20:20Z

Publisher:

Andrea Mignone

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

13 October 2026

The PLUTO Code on GPUs: A First Look at Eulerian MHD Methods

M. Rossazza[1],A. Mignone[1], M. Bugli[2,1,3], S. Truzzi[1], L. Riha[4], T. Panoc[4], O. Vysocky[4], N. Shukla[5], A. Romeo[5], V. Berta[1]
[1]Dipartimento di Fisica, Università degli Studi di Torino, Via Pietro Giuria 1, I-10125 Torino, Italy
[2]Institut d'Astrophysique de Paris, UMR 7095, CNRS & Sorbonne Université, 98 bis bd Arago, 75014 Paris, France
[3]INFN - sezione di Torino, Via Pietro Giuria 1, I-10125 Torino, Italy
[4]IT4Innovations, VSB - Technical University of Ostrava, Ostrava, Czech Republic
[5]CINECA, Via Magnanelli 6/3, 40033 Casalecchio di Reno, BO, Italy
Corresponding author: Marco Rossazza, marco.rossazza@unito.it

arXiv:2511.20337v1 [physics.plasm-ph] 25 Nov 2025

Abstract

We present preliminary performance results of gPLUTO, the new GPU-optimized implementation of the PLUTO code for computational plasma astrophysics. Like its predecessor, gPLUTO employs a finite-volume formulation to numerically solve the equations of magnetohydrodynamics (MHD) in multiple spatial dimensions. Still, this new implementation is a complete rewrite in C++ and leverages the OpenACC programming model to achieve acceleration on NVIDIA GPUs. While a more comprehensive description of the code and its several other modules will be presented in a future paper, here we focus on some preparatory results that demonstrate the code potential and performance on pre exa-scale parallel architectures.

Keywords: GPU computing, OpenACC, PLUTO code, Parallel computing , High-performance computing (HPC), Magnetohydrodynamics (MHD).

1. Introduction

Magnetohydrodynamics (MHD) simulations play a pivotal role in understanding a vast range of complex physical phenomena. In the context of astrophysical processes, these include the formation and evolution of planets [1], stars [2], and galaxies [3], the acceleration of high-energy particles [4, 5], and the multi-wavelength emission associated to the activity of compact objects such as black holes [6] and neutron stars [7, 8]. There are also numerous terrestrial applications that exploit the predictive power of the MHD framework, such as nuclear fusion [9, 10], space weather predictions [11], and plasma confinement [12].

The computational demands of MHD simulations are inherently tied to the need to describe at once spatial and temporal scales that often span many orders of magnitude. The ability to resolve phenomena occurring at scales much smaller than the typical size of the global system (e.g., the turbulent structures in the plasma constituting an accretion disk or a jet propagating through the interstellar medium) plays a crucial role in enabling a far deeper understanding of their properties and allowing more accurate predictions of their observational/experimental signatures. Prime examples in the past decade of the qualitative impact of higher computational efficiency in multi-dimensional astrophysical MHD simulations include models of stellar explosions [13, 14], dynamo processes [15, 16], and accretion/ejection of matter around central compact objects [17, 18]. In recent years, the field of high-performance computing has witnessed a transformative shift thanks to the advent of Graphics Processing Units (GPUs) that offer massive parallelism and computational capabilities. The use of GPUs in scientific computing has indeed opened up new frontiers in numerical simulations, enabling researchers to explore complex physical systems at unprecedented resolutions and speeds. Notable examples are the codes H-AMR [19], IDEFIX [20], AthenaK [21], ECHO [22], and AsterX [23].

In this paper, we present the latest noteworthy advancements in the development of the GPU-ready version of the PLUTO code [24, 25] for modeling astrophysical plasmas. PLUTO has been widely used by hundreds of researchers in the astrophysical community over the past two decades. Owning to the variety of physical modules provided (i.e., from classical to relativistic flows, ideal and non-ideal dynamics, different geometries, and equations of state), the code has been employed in very different astrophysical contexts, ranging from high-energy astrophysics [26, 27, 28, 29] to young stellar objects [30], supernova remnants [31, 32, 33, 34] plasma physics [35, 36], planet formation [37, 38], and solar physics [39, 40], to name just a few.

The new GPU-accelerated implementation of the code (gPLUTO) is a complete C++ rewrite of the legacy code, which employs the OpenACC programming model to harness the vast computing power and energetic efficiency of modern GPU architectures, offering significant advantages in terms of computational speed, power consumption, and cost-effectiveness. Switching to C++ is justified by the need to to take advantage of specific language extensions such as function templates, classes and polymorphism. gPLUTO keeps the strengths of the previous versions, such as stability, versatility, and user-friendliness, while also giving access to the most recent high-performance computing resources. This work outlines the code's funda-

mental design principles and the remarkable results achieved in terms of simulation performance, while a more comprehensive description of the new code and all of its physical modules will appear in a future manuscript.

In this work we present the technical details of gPLUTO by introducing its underlying algorithmic framework, data management and parallelization strategies, and implementation solutions. We also showcase a series of benchmark tests, highlighting the code's effectiveness in reproducing them as well as its notable performance. This paper has a companion, complementary paper that focuses on the Lagrangian Particles module integration and its GPU offloading [41]. The paper is organized as follows: in §2, we briefly recap the equations of MHD and the finite volume framework. In §3, we illustrate technical aspects concerning the OpenACC implementation, while in §4, we show the parallel performance on two standard use cases. Finally, we draw our conclusions in §5.

2. Numerical methods

2.1. MHD equations

Following its predecessor [24, 25], gPLUTO is designed to solve a variety of systems of conservation laws,

$$\frac{\partial \mathcal{U}}{\partial t} + \nabla \cdot \mathsf{T} = \mathcal{S}, \tag{1}$$

where $\mathcal{U}$ is an array of conservative quantities, T is a flux tensor and $\mathcal{S}$ defines the source terms. Although different sets of equations may be solved, for the present purpose, we only focus on the single fluid classical MHD equations:

$$\mathcal{U} = \begin{pmatrix} \rho \\ \rho\mathbf{v} \\ \mathbf{B} \\ \mathcal{E} \end{pmatrix}, \quad \mathsf{T} = \begin{pmatrix} \rho\mathbf{v} \\ \rho\mathbf{v}\mathbf{v} - \mathbf{B}\mathbf{B} + p_t\mathsf{I} \\ \mathbf{v}\mathbf{B} - \mathbf{B}\mathbf{v} \\ (\mathcal{E} + p_t)\mathbf{v} - (\mathbf{v}\cdot\mathbf{B})\mathbf{B} \end{pmatrix}^T, \tag{2}$$

where ρ is the fluid density, $\mathbf{v}$ is the fluid velocity, $\mathbf{B}$ is the magnetic field vector, $p_t = p + \mathbf{B}^2/2$ is the total (thermal + magnetic) pressure and

$$\mathcal{E} = \frac{p}{\Gamma - 1} + \frac{\rho\mathbf{v}^2}{2} + \frac{\mathbf{B}^2}{2}, \tag{3}$$

is the total energy density comprising thermal, kinetic and magnetic contributions. In Eq. 3 we have assumed an ideal equation of state with adiabatic index Γ.

The divergence-free constraint, $\nabla \cdot \mathbf{B} = 0$, is not automatically preserved by the underlying discretization method and must be specifically enforced. To this purpose, the code allows different algorithms to be selected, including include Powell's eight wave formulation [42], divergence cleaning [43, 44] and the constrained transport method [see 45, 46] in the more recent formulation by [47].

Relativistic and non-ideal extensions of the previous equations will be described in a forthcoming work.

2.2. Discretization

We employ a Cartesian coordinate system with axes defined by the unit vectors $\hat{\mathbf{e}}_x = (1,0,0)$, $\hat{\mathbf{e}}_y = (0,1,0)$ and $\hat{\mathbf{e}}_z = (0,0,1)$. A uniform mesh with coordinate spacing Δx, Δy and Δz is assumed to cover the computational domain.

gPLUTO employs a finite volume (FV) framework to evolve cell-averaged conservative variables in time, given specified initial and boundary conditions. The FV method is based on the integral form of the governing differential equations, obtained by integrating Eq. (1) over a control volume corresponding to a single computational zone. Integrating over space within a single cell yields the semi-discrete form of the conservation equations:

$$\begin{aligned} \frac{d\langle\mathcal{U}\rangle_{\boldsymbol{c}}}{dt} = &- \frac{\Delta t}{\Delta x}\left(\hat{F}_{i+\frac{1}{2},j,k} - \hat{F}_{i-\frac{1}{2},j,k}\right) \\ &- \frac{\Delta t}{\Delta y}\left(\hat{G}_{i,j+\frac{1}{2},k} - \hat{G}_{i,j-\frac{1}{2},k}\right) \\ &- \frac{\Delta t}{\Delta z}\left(\hat{H}_{i,j,k+\frac{1}{2}} - \hat{H}_{i,j,k-\frac{1}{2}}\right), \end{aligned} \tag{4}$$

where $\boldsymbol{c} = (i, j, k)$ is a short-hand notation for a single computational cell centered at (x_i, y_j, z_k), $\{i,\ j,\ k\}$ is a triplet of zone indices in the three coordinate directions while cell interfaces are located at $\boldsymbol{c} \pm \hat{\mathbf{e}}_x/2$, $\boldsymbol{c} \pm \hat{\mathbf{e}}_y/2$, and $\boldsymbol{c} \pm \hat{\mathbf{e}}_z/2$ in the three directions, respectively. In Eq. (4), $\langle\mathcal{U}\rangle_{\boldsymbol{c}}$ represents the volume-averaged conserved quantities within each cell, while $\hat{F}, \hat{G}$, and $\hat{H}$ are the surface-averaged numerical fluxes across the corresponding cell interfaces, computed via a Riemann solver.

At the numerical level, Eq. (4) is treated as a system of ordinary differential equations (ODEs) in time, and discretized using a strong-stability-preserving (SSP) Runge-Kutta (RK) method [48]. For example, a third-order SSP Runge-Kutta scheme proceeds as:

$$\begin{aligned} \langle\mathcal{U}\rangle^{*} &= \langle\mathcal{U}\rangle^{n} + \mathcal{L}^{n} \\ \langle\mathcal{U}\rangle^{**} &= \frac{3\langle\mathcal{U}\rangle^{n} + \langle\mathcal{U}\rangle^{*} + \mathcal{L}^{*}}{4} \\ \langle\mathcal{U}\rangle^{n+1} &= \frac{\langle\mathcal{U}\rangle^{n} + 2\langle\mathcal{U}\rangle^{**} + \mathcal{L}^{**}}{3}, \end{aligned} \tag{5}$$

where, for brevity, the cell index subscript $\boldsymbol{c}$ has been omitted. In the expression above, $\mathcal{L}^n \equiv \mathcal{L}(\langle\mathcal{U}\rangle^n)$ represents the right-hand side of Eq. (4), evaluated using the solution at time level n, and similarly for $\mathcal{L}^*$ and $\mathcal{L}^{**}$ at intermediate stages.

For each Runge-Kutta stage, the right-hand side $\mathcal{L}$ is typically computed by first performing a reconstruction step, in which zone-centered primitive variables are extrapolated to cell interfaces. For instance, in the x-direction:

$$\mathcal{V}^{L}_{i+\frac{1}{2}} = \mathcal{R}^{+}_{x}(\mathcal{V}_{\boldsymbol{c}}), \qquad \mathcal{V}^{R}_{i-\frac{1}{2}} = \mathcal{R}^{-}_{x}(\mathcal{V}_{\boldsymbol{c}}) \tag{6}$$

where $\mathcal{R}^{\pm}_x$ denotes a non-oscillatory, high-order polynomial reconstruction operator (see, e.g., [49, 50, 51]), applied in the positive (+) or negative (−) x-direction.

Here, $\mathcal{V}_{\boldsymbol{c}} = \{\rho, \mathbf{v}, \mathbf{B}, p\}$ represents the array of primitive variables (density, velocity, magnetic field, and pressure),

which are preferred over conservative variables during the reconstruction process. This choice is made because primitive variables typically exhibit smoother variations across the computational domain — except at shocks or contact discontinuities — thereby reducing numerical errors and mitigating spurious oscillations near sharp gradients. In MHD, primitive variables are recovered from the conservative ones via straightforward inversion:

$$\mathbf{v} = \frac{\rho \mathbf{v}}{\rho}, \qquad p = (\Gamma - 1)\left(\mathcal{E} - \frac{\rho \mathbf{v}^2}{2} - \frac{\mathbf{B}^2}{2}\right) \tag{7}$$

where Γ is the adiabatic index, and $\mathcal{E}$ is the total energy density. Employing primitive variables during reconstruction helps limit the occurrence of unphysical states, such as negative pressures, especially in regions near strong gradients.

The reconstructed values at cell interfaces serve as input states to the Riemann solver, which computes a stable, upwinded numerical flux. For example:

$$\hat{F}_{i+\frac{1}{2}} = \frac{F(\mathcal{V}^L_{i+\frac{1}{2}}) + F(\mathcal{V}^R_{i+\frac{1}{2}})}{2} - \mathcal{D}_{i+\frac{1}{2}} \cdot \left(\mathcal{U}^R_{i+\frac{1}{2}} - \mathcal{U}^L_{i+\frac{1}{2}}\right) \tag{8}$$

where $\mathcal{D}$ is a dissipation matrix whose form depends on the specific Riemann solver employed. The subscripts j and k have been omitted here for clarity, as the context is now unambiguous.

Magnetic fields can be evolved either in a zone-centered formalism, as described so far, or using the constrained transport (CT) method [45, 52]. The advantage of CT is its ability to maintain the divergence-free condition for the magnetic field to machine precision throughout the simulation. Further details on the numerical implementation can be found in the original PLUTO code paper [24], and in recent developments related to CT techniques in [47].

As its predecessor, gPLUTO offers a variety of algorithms with different levels of accuracy and computational cost to choose from. In the present work, we employ the third-order SSP Runge-Kutta time-stepping scheme (Eq. (5)), combined with either parabolic reconstruction [53] or the WENOZ reconstruction method [51].

3. PLUTO design for GPUs

Modern Graphics Processing Units (GPUs) are specialized, highly parallel processors that have become cornerstones of high-performance scientific computing. Their computational power originates from an architecture containing hundreds or thousands of simple cores designed to execute the same instruction simultaneously across large sets of data. This paradigm makes GPUs exceptionally efficient for algorithms characterized by high data parallelism, such as the mesh-based solvers used in our work. Computations are offloaded from the host CPU to the GPU (called in this context device or accelerator) by launching functions called kernels, which are managed and executed by the device's hardware scheduler. The execution is organized hierarchically. A thread is a single stream of instructions of that runs in parallel with other threads and is mapped to the GPU cores. Individual threads that execute the same instruction are grouped into warps and multiple warps are contained in blocks, that are scheduled for execution on the GPU's fundamental hardware units, the Streaming Multiprocessors (SMs). Each SM is an independent multiprocessor containing its own execution cores, scheduler, and a high-speed shared memory, allowing it to execute one or more thread blocks concurrently.

3.1. OpenACC

The programming model chosen for GPU porting of PLUTO is OpenACC, a simpler and higher-level alternative to CUDA. OpenACC is a directive-based programming model for GPU computing developed by Cray, CAPS, NVIDIA, and PGI in 2011. It relies on information provided by the programmer to the compiler through directives, library routines, and environment variables. These annotations instruct the compiler in porting the code to the device. The complex and low-level aspect of the parallelization is left to be handled by the compiler, facilitating the user's work. Another advantage of choosing a directive-based approach is interchangeability, an important feature allowing the code to be compiled and run, ignoring the OpenACC directives, making it suitable for both CPU and GPU computing. For every GPU-accelerated application, the computing work is divided among CPUs and GPUs. In this context, "host" refers to the primary processor in a system (the CPU) while the "device" refers to the secondary processor (the GPU) responsible for the acceleration.

In broad terms, in gPLUTO the entire computational workload is handled by the GPUs, while the CPUs oversee overall system operations, including running the operating system, managing system memory, and handling I/O operations. The reason for this strict division is performance. Executing all the computations on the device permits, in fact, the exploitation of its potential without introducing data movement between the memories of the two processors. The alternate use of host and device would introduce memory copies to assure coherence between data and, hence, correct results. OpenACC directives are indicated with the keyword `pragma` followed by the `acc` sentinel, the directive itself, and optional clauses:

```
#pragma acc < directive > < clause , clause >
```

The directive is applied to the instruction, loop, or structured code block that follows. The base approach when accelerating an application with OpenACC is using compute directives (we mainly use `parallel loop`) that indicate to the compiler regions of the code to execute on the device, the kernels. Data directives, on the other hand, such as `enter data copyin` and `exit data delete`, permit explicit control of the data movement. This is crucial since, in many cases, the performance of the kernels is limited by the memory bandwidth of the device, and thus, reducing the amount of data needed to be moved gives a huge benefit.

3.2. Compute Directives

A typical compute construct of gPLUTO is reported in Fig. 1, where NX, NY, NZ define domain size, data->Vc is the array of the primitive quantities (i.e., density, pressure and velocities), that contains NVAR physical quantities for each point of the 3-dimensional domain.

```
#pragma acc parallel loop collapse(3) present(data)
for (k = 0; k < NZ; k++){
for (j = 0; j < NY; j++){
for (i = 0; i < NX; i++){
  ...
  #pragma acc loop seq
  for (nv = 0; nv < NVAR; nv++){
    data->Vc(i,j,k,nv) = ...;
  }
  ...
}}}
```

Figure 1: Kernel accelerated with a compute directive.

The first directive is at the core of the acceleration of the application: #pragma acc parallel loop directive tells the compiler to parallelize the loop, distributing its iterations across the threads of the target accelerator. The loop body is then executed concurrently, significantly improving the performance of the code. The collapse(3) clause applies in this case to three tightly nested loops. Hence, the compiler treats the construct as a single loop of NX x NY x NZ iterations. This makes it easier for the compiler to map the construct onto the device's computing resources, potentially improving parallelization efficiency. The present clause implies that data is already present on the GPU and that a copy from the main memory is not necessary. If this is not true a fatal error at the execution would be generated.

The innermost loop, over NVAR elements, is marked with seq, meaning that the operation on different values of nv is executed sequentially by the same thread. Note that while parallelization of this inner loop could be possible, it usually tends to lower the performance, as the number of variables NVAR is typically much smaller than the number of grid points. Moreover, having the threads corresponding to loops with many iterations makes sense for optimal utilization and memory access (see Sec. 3.3).

3.3. Memory Management

To manage data transfer to the device in gPLUTO, we combine explicit OpenACC data directives with CUDA Unified Memory, enabled via the compile-time flag -gpu=managed. Unified Memory provides a shared address space accessible by both the CPU and GPU through a single pointer, making it particularly suitable for dynamically allocated data. In most cases, the NVIDIA driver automatically handles data movement. However, we employ explicit data directives to ensure correct deep copying of structured and nested data that are widely used within the code.

GPUs employ various techniques to hide memory latency that can represent an important performance bottleneck. Most of these techniques, such as warp scheduling[1] and prefetching[2] are handled by the device itself but the programmer can actively act on other important aspects such as coalesced memory access.

Coalesced memory access. Coalesced memory access is a memory access pattern in parallel computing, particularly on GPUs, where multiple threads access consecutive or contiguous memory locations in a single memory transaction. This pattern is essential for achieving efficient memory bandwidth utilization and improving the overall performance of parallel applications. In the pseudo-code of Fig. 1, coalesced memory access is achieved, granting optimal parallelization. Here, consecutive values of the index i correspond to consecutive threads but also to adjacent memory access (i is the fastest index of data->Vc). The advantage of this memory pattern can be traced back to the thread memory access. When a thread needs data for an operation, it receives access to many adjacent memory locations. Neighbor threads can be exploited immediately if they need data received in this memory transaction, hence the advantage of them accessing adjacent memory locations.

Ary classes. Optimal memory access pattern is one of the main reasons behind the profound change of many arrays of the code together with the migration to C++. We implemented classes called Ary to allocate and handle memory access for multi-dimensional arrays. In the details, we moved from a pointer to pointer approach to one-pointer, flat arrays using classes to hide index computations. Additionally, element ordering can be changed whether or not we're compiling with OpenACC or based on the array type, covering the complexity of this operation in the class itself. We now discuss a technique we adopted widely, using the Ary class with flat arrays to improve memory access.

In Fig. 2 rhs is a 2D array of size NX x NVAR created from the single, contiguous block of memory d->sweep.rhs of size NX x NY x NZ x NVAR, based on an offset defined by j and k. This assures that each thread writes on a unique memory address (privatization[3]), which is necessary to grant correct results. Moreover, class definition assures that neighboring threads corresponding to consecutive values of i access adjacent memory locations.

Then, when computing the y-direction contribution, the same technique is applied to the same memory pool d->sweep.rhs, changing the order of write and access to the elements to obtain again optimal access pattern again. Other arrays such as Vc and Uc have fixed ordering: i,j,k,nv in order of decreasing index speed.

[1] When some threads within a warp need to access memory, the scheduler can switch to other warps with active threads, allowing the GPU to continue executing instructions without waiting for the memory access to complete.

[2] GPUs can predict and preload data into caches before it is actually needed, reducing the time it takes to fetch the data when it is required by the computation.

[3] Thread privatization is the process of giving each thread within a kernel its own private copy of a variable, ensuring that parallel computations don't interfere with each other by creating race conditions.

```
#pragma acc parallel loop collapse(2)
for (k = 0; k < NZ; k++){
for (j = 0; j < NY; j++){
  long int offset = NVAR*NX*(j + NY*k);
  Ary2D flux(&d->sweep.flux[offset],NX,NVAR);
  Ary2D rhs(&d->sweep.rhs[offset],NX,NVAR);

  #pragma acc loop vector
  for (i = 0; i < NX; i++){
    #pragma acc loop seq
    NVAR_LOOP(nv) {
      rhs(i,nv) = -dtdx*(flux(i+1,nv) - flux(i,nv));
    }
}}}
```

Figure 2: Computation of the right-hand side contribution along the x-direction.

Template function. Another useful feature of C++ that has been adopted is function templates. Many kernels operate on arrays whose indexing depends on the direction of integration, which can vary at runtime. This runtime variability prevents the compiler from knowing in advance which registers will be needed, often resulting in these variables being placed in slower memory spaces such as local or shared memory instead of registers. By using templated functions, we can specialize the code based on template parameters—in our case, the integration direction `dir`. This enables the compiler to inline the specialized functions and propagate constant values, eliminating branches and improving performance. Fig. 3 shows an example of the Riemann solver with `dir` as a template parameter. In this case, the array `v` can be allocated in registers at compile time.

```
template<int dir>
void Riemann_Solver (Data *d, Grid *grid, RBox *box);

void Riemann_Solver_T (Data *d, Grid *grid, RBox *box){
  if(dir==IDIR) Riemann_Solver<IDIR>(d, grid, box);
  else if(dir==JDIR) Riemann_Solver<JDIR>(d, grid, box);
  else if(dir==KDIR) Riemann_Solver<KDIR>(d, grid, box);
}

template<int dir>
void Riemann_Solver (Data *d, Grid *grid, RBox *box)
{
  int VXn = VX1+dir;

   #pragma acc parallel loop collapse(2)
   for (k = kbeg; k <= kend; k++){
   for (j = jbeg; j <= jend; j++){
     #pragma acc loop
     for (i = ibeg; i <= iend; i++){
       int v[NVAR];
       v[VXn] = ...;
     }
    }}
}
```

Figure 3: Templated Riemann Solver. Three version of the routine are compiled with different values of the direction of integration `dir`: `IDIR`, `JDIR`, `KDIR`

3.4. Multi GPU communication

Multi-process communication at each time step is necessary in gPLUTO to assure correct results when the domain is split between processes. Communication is based on the message passing interface (MPI) protocols. While in PLUTO the data exchange was based on the library ArrayLib ([54]), it has here been rewritten using a more basic scheme based on standard buffer arrays and `MPI_Isend`/`MPI_Irecv` calls. To exploit the characteristics of GPUs, we employ an asynchronous approach for data exchange. This requires each processor to communicate potentially simultaneously with its 8 neighbors in 2D or 26 neighbors in 3D (this includes also nearby processors in the diagonal directions). The pseudo-code in Fig. 4 describes the dedicated routine, considering one direction of exchange (`IDIR`).

```
MPI_Irecv (recv_buf_2, ..., nrnks[IDIR][1], ...);
MPI_Irecv (recv_buf_1, ..., nrnks[IDIR][0], ...);

#pragma acc parallel loop collapse(4) async(1)
LOOP(...){
  send_buf_1[ind] = Vc((ibeg+i),j,k,nv);
}

#pragma acc parallel loop collapse(4) async(2)
LOOP(...){
  send_buf_2[ind] = Vc((iend-(n1g-1)+i),j,k,nv);
}

#pragma acc wait

MPI_Isend(send_buf_1, ..., nrnks[IDIR][0], ...);
MPI_Isend(send_buf_2, ..., nrnks[IDIR][1], ...);

MPI_Waitall(..., &recv[2], ...);
MPI_Waitall(..., &recv[1], ...);

#pragma acc parallel loop collapse(4) async(1)
LOOP(...){
  Vc(i,j,k,nv) = recv_buf_1[ind];
}

#pragma acc parallel loop collapse(4) async(2)
LOOP(...){
  Vc((iend+1+i),j,k,nv) = recv_buf_2[ind];
}

#pragma acc wait

MPI_Waitall(..., &send[1], ...);
MPI_Waitall(..., &send[2], ...);
```

Figure 4: Asynchronous implementation of the parallel data exchange routine.

Initially, the code initializes non-blocking receives (`MPI_Irecv`) to get data from neighboring processes. Then, data to be sent to neighboring processes are packed into buffers allocated with `acc_malloc`[4], and each kernel has an `async` clause with a different number: each kernel is independent, and they could ideally run at the same time on distinct streams (sequences of operations). After the code

[4] OpenACC function that allocates memory directly in the accelerator's global memory.

initializes non-blocking send calls (`MPI_Isend`) to send data to neighboring processes and then it waits for all receive operations to complete. When this happens, the received data is unpacked into the appropriate variables, again concurrently for each `recv_buf`. Finally, it waits for all non-blocking send operations to complete and ensures all asynchronous operations on the GPUs are terminated before proceeding.

4. Numerical tests and Discussion

Here, we present numerical benchmarks conducted on CPU and GPU based partitions of three different HPC platforms: MareNostrum 5 Accelerated Partition (GPU), Leonardo Booster (GPU), MeluXina GPU nodes (GPU), and Leonardo DCGP (CPU). Every run is performed employing the entire computational power of each node, meaning 4 GPUs per node on GPU partitions and 112 CPU-cores on Leonardo DCGP.

Leonardo is hosted by Cineca and has two main partitions: Booster module and Data-centric General Purpose (DCGP) module. The Booster partition is equipped with Atos BullSequana X2135 "Da Vinci" single-node GPU blade with four NVIDIA Ampere GPUs/node (64 GB HBM2e). The DCGP module has instead Atos BullSequana X2140 three-node CPU blade, with dual socket 56-core Intel Sapphire Rapids Intel Xeon Platinum 8480+. MeluXina belongs to LuxProvide and the GPU nodes provide Atos BullSequana XH2000 platform with four NVIDIA Ampere GPUs (40 GB HBM). Finally, MareNostrum 5 is at the Barcelona Supercomputing Center (BSC), and the Accelerated Partition (ACC) we used is a BullSequana XH3000 model with four NVIDIA Hopper GPUs (64 GB HBM2). See Tab. 1 for a summary of the systems specifics.

4.1. 3D Orszag Tang Vortex

The first benchmark is a 3D version of the well-known Orszag-Tang test problem, with a similar to that of [55]. We consider a periodic Cartesian domain of size $L_x = L_y = L_z = 1$ with constant initial pressure $p = 5/3$ and density $\rho = 25/9$. Velocity and magnetic field are initialized as follows

$$\begin{aligned} \mathbf{v} &= -\zeta(z)\sin(2\pi y)\hat{\mathbf{e}}_x + \zeta(z)\sin(2\pi x)\hat{\mathbf{e}}_y + 0.2\sin(2\pi z)\hat{\mathbf{e}}_z \,, \\ \mathbf{B} &= -\sin(2\pi y)\hat{\mathbf{e}}_x + \sin(4\pi x)\hat{\mathbf{e}}_y \,, \end{aligned} \tag{9}$$

where $\zeta(z) = 1 + \sin(2\pi z)/5$. Computations are stopped at $t = 0.5$ and the test employs the HLLD Riemann solver [56], WENOZ reconstruction [51], and the RK3 time-stepping scheme. All computations are performed in double-precision arithmetic.

System	Resolution per Node
Leonardo Booster	704 x 704 x 352
Leonardo DCGP	704 x 704 x 350
MareNostrum 5 ACC	704 x 704 x 352
MeluXina GPU nodes	512 x 512 x 512

Table 2: Weak scaling resolution (per node) for the 3D Orszag-Tang test.

Weak scaling. We first assess the parallel performance using weak scale metric, implying that grid resolution per node - reported in Tab. 2 - is kept constant. For these runs, we employ the constrained transport method (with UCT HLLD average, [47]) to enforce the divergence-free condition of magnetic field. On the GPU partitions, we chose a resolution that maximized device occupancy to achieve the best performance. A $704 \times 704 \times 352$ resolution was used to nearly fill the device memory on both Leonardo Booster and Marenostrum 5 ACC. A lower resolution was necessary for MeluXina to accommodate its smaller GPU memory. The runs on Leonardo DCGP used a slightly different resolution to allow for domain decomposition across 112 processes while maintaining a nearly identical computational load per process as: $\frac{N_{\text{t,CPUnode}}}{N_{\text{t,GPUnode}}} \sim 0.99$, with N_{t} the total number of grid points.

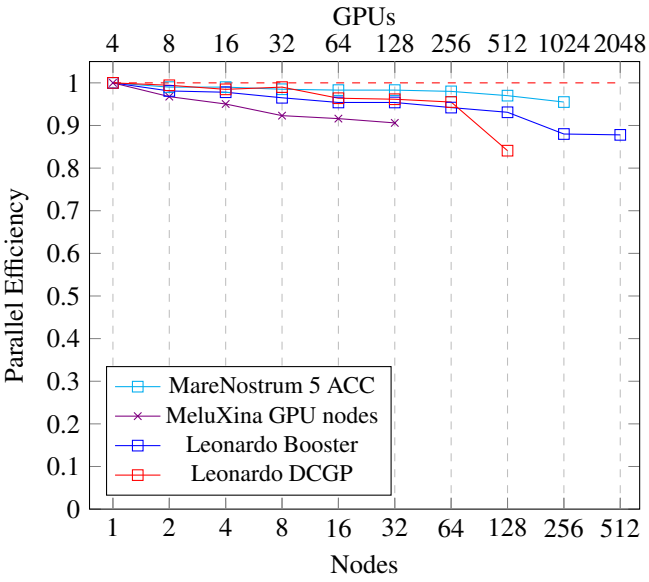

Figure 5: Weak scaling results for the 3D Orszag-Tang on MareNostrum 5 ACC, MeluXina GPU nodes and Leonardo (Booster and DCGP).

Fig. 5 presents the parallel efficiency expressed as the ratio of the wall clock times on 1 node and N nodes, T_1/T_N. We have repeated the test on all 4 HPC systems up to 512 nodes (when permitted). Among the tested systems, MareNostrum 5 ACC delivered the best results, maintaining a parallel efficiency above 95 % even at 256 nodes. On Leonardo Booster, we scaled up to 512 nodes, achieving 88 % efficiency at the largest scale. In contrast, MeluXina GPU nodes showed a steeper efficiency drop, reaching 91 % at just 32 nodes. The CPU runs on Leonardo DCGP maintain an efficiency above 95 % for up to 64 nodes, but this value drops to 84 % at 128 nodes. Further investigation of the reasons for this low efficiency at 128 nodes will be needed.

In Fig. 6, we show the corresponding performance of the weak scaling runs, measured in terms of the number of cells updated per second per node (computed as the number of grid points multiplied by the number of time steps divided by the execution time and the number of nodes). Note that CPU runs on Leonardo DCGP were approximately 9 times slower than

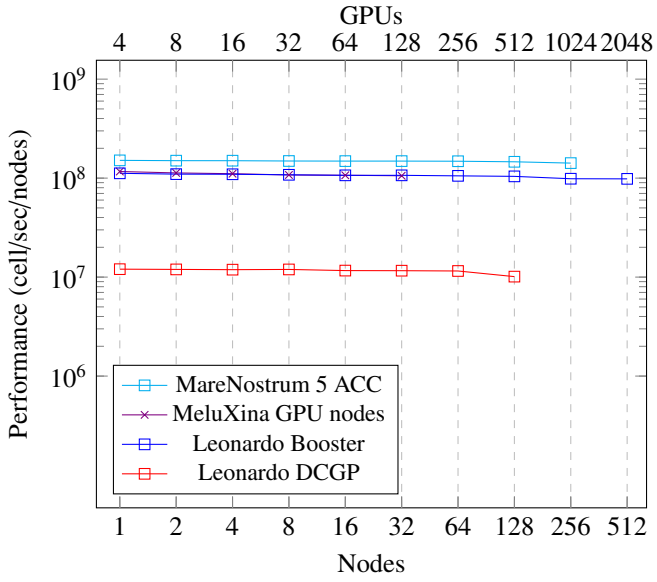

Figure 6: Performance of Orszag Tang weak scaling tests on MareNostrum 5 ACC, MeluXina GPU nodes and Leonardo (Booster and DCGP).

Nodes	T_{GPUs} (sec)	T_{CPUs} (sec)	Acceleration ($T_{\text{CPUs}}/T_{\text{GPUs}}$)
1	312	2885	9.24
2	318	2901.4	9.12
4	319	2918.6	9.14
8	323	2903.2	8.98
16	327	2981.6	9.11
32	327	2989	9.14
64	331	3009.6	9.02
128	335	3430.8	10.24

Table 3: CPU-GPU comparison of Orszag Tang weak scaling tests (Fig. 5) on Leonardo DCGP (4 A100 GPUs per node) and Booster partition (112 Intel Xeon 8480+ CPUs cores per node), respectively.

GPU runs on the Leonardo Booster partition. Tab. 3 provides the acceleration factors for the two types of runs.

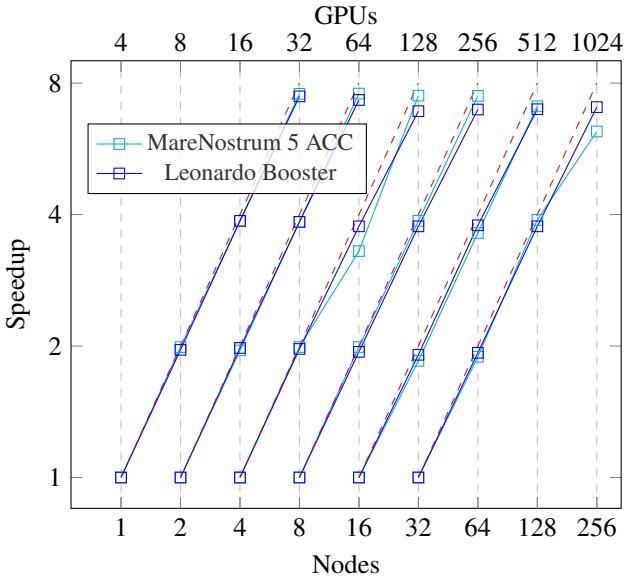

Figure 7: Strong scaling tests of Orszag Tang on MareNostrum 5 ACC and Leonardo Booster.

Strong scaling. We then present, in Fig. 7, strong scaling performance results of gPLUTO, obtained by repeating the test using the divergence cleaning method for the control of the magnetic field divergence. This allowed for choosing a slightly larger grid than that used in the weak scaling test because this algorithm requires less device memory than the constrained transport method (Tab. 4). In this test, the domain size remains fixed while the number of processes varies. As a result, GPU utilization is optimal in the initial runs but degrades, along with performance, as more nodes are employed. To address this effect, we performed six sets of four runs each, maintaining a fixed resolution as detailed in Tab. 4. The resolution was selected to ensure that the grid size nearly filled the GPU VRAM at the lowest node count within each group. We then successively doubled the number of nodes and processes four times per group, obtaining strong scaling efficiencies. The results ob-

	Leonardo		**MeluXina**	**MareNostrum 5**
Detail	**Booster**	**DCGP**	**GPU**	**ACC**
Proprietary Company	CINECA		LuxProvide	BSC
Active from year	2022		2021	2023
Node Model	BullSequana X2135	BullSequana X2140	BullSequana XH2000	BullSequana XH3000
CPUs (per node)	Intel Ice Lake 8358	2x 56-core Intel Sapphire Rapids 8480+	2x AMD EPYC Rome 7452	2x Intel Sapphire Rapids 8460Y+
GPUs (per node)	4x NVIDIA Ampere (64 GB HBM2e)	N/A (CPU-only)	4x NVIDIA Ampere (40 GB HBM)	4x NVIDIA Hopper (64 GB HBM2)

Table 1: Summary of computational hardware resources utilized.

tained on MareNostrum 5 ACC and Leonardo Booster largely overlap, with average efficiencies of the runs at higher numbers of nodes of each group at 0.90 and 0.89, respectively. Tab. 4 provides the exact scaling efficiencies, computed by comparing the measured values of speedup, plotted in Fig. 7, to the ideal ones that correspond to the red lines in figure (speedup=1,2,4,8).

Nodes	MareNostrum Efficiency	Leonardo Efficiency
Group 1, Base: 1 Node, Resolution: 832 x 832 x 416		
1	1.00	1.00
2	0.98	0.98
4	0.97	0.97
8	0.94	0.93
Group 2, Base: 2 Nodes, Resolution: 832 x 832 x 832		
2	1.00	1.00
4	0.99	0.99
8	0.96	0.96
16	0.94	0.92
Group 3, Base: 4 Nodes, Resolution: 1664 x 832 x 832		
4	1.00	1.00
8	0.99	0.99
16	0.83	0.94
32	0.94	0.86
Group 4, Base: 8 Nodes, Resolution: 1664 x 1664 x 832		
8	1.00	1.00
16	0.99	0.97
32	0.97	0.94
64	0.93	0.87
Group 5, Base: 16 Nodes, Resolution: 1664 x 1664 x 1664		
16	1.00	1.00
32	0.93	0.95
64	0.91	0.94
128	0.89	0.87
Group 6, Base: 32 Nodes, Resolution: 3328 x 1664 x 1664		
32	1.00	1.00
64	0.94	0.97
128	0.97	0.94
256	0.77	0.88

Table 4: Strong scaling Efficiencies at different resolutions for the tests presented in Fig. 7 on MareNostrum 5 ACC and Leonardo Booster.

4.2. 3D Circularly Polarized Alfvén

The propagation of circularly polarized Alfvén waves is a widely employed benchmark as it provides an exact nonlinear solution of the ideal MHD equations. Following [57], we rotate a one-dimensional frame to obtain a 3D configuration and employ the same grid size used in Sec. 4.1 for strong and weak scaling. The domain is a Cartesian cube with $L_x = L_y = L_z = 1$ with periodic boundaries. The initial speed $\mathbf{v}$ and magnetic field $\mathbf{B}$ are chosen as

$$\begin{aligned} \mathbf{v} &= \epsilon \sin(\phi)\hat{\mathbf{e}}_y + \epsilon \cos(\phi)\hat{\mathbf{e}}_y \,, \\ \mathbf{B} &= \hat{\mathbf{e}}_x + \epsilon \sin(\phi)\hat{\mathbf{e}}_y + \epsilon \cos(\phi)\hat{\mathbf{e}}_z \,, \end{aligned} \tag{10}$$

where $\epsilon = 0.1$ is the wave amplitude, while $\phi = \mathbf{k} \cdot \mathbf{x}$ with $\mathbf{k} = 2\pi(1/L_x, 1/L_y, 1/L_z)$. Density and gas pressure are initially constant and equal to $\rho_0 = 1.0$ and $p_0 = 0.1$, respectively. We employ an ideal equation of state with adiabatic index $\Gamma = 5/3$.

We solve the MHD equations using the 3$^{\text{rd}}$-order Runge-Kutta scheme with WENOZ reconstruction and the HLLD Riemann solver. The divergence-free condition is enforced using either divergence cleaning (for strong scaling) or constrained transport (for weak scaling) to differentiate the algorithms employed.

All computations are performed in double-precision arithmetic. The tests have been performed on Leonardo Booster and MareNostrum 5 ACC partitions, using the same grid resolution (Tab. 2) of 4.1,

Fig. 8 presents the parallel efficiency expressed as the ratio of the wall clock times on 1 node and N nodes, T_1/T_N. The efficiency decreases slightly as the number of nodes grows due to the increase in communications and physical distances between the devices. The best results are obtained, this time, with Leonardo with an efficiency of 0.97 for 256 nodes (1024 GPUs), while we obtain 0.95 on MareNostrum 5. Strong scaling test results are presented in Tab. 5 and in Fig. 9. As for the Orszag-Tang test, six strong scaling tests are presented for four groups of runs with fixed resolution. As expected, the strong scaling efficiency slightly decreases as the number of nodes rises. Leonardo Booster's results are marginally better than MareNostrum 5 ACC for this test, with an average efficiency of 0.97 and 0.93, respectively.

4.3. Energy Efficiency

Energy efficiency for an application is defined as performance in Floating point Operations Per Second (FLOPS) per unit of power consumption (Watt). The total energy consumed (in Joules) is approximated from power samples (in Watts) obtained at a specific sampling frequency (in Hertz) from a power monitoring system, as depicted in the following equation:

$$\text{Energy}(t) = \int_0^t \text{Power}(t')\mathrm{d}t' \approx \frac{\sum_{i=0}^{n} P_i}{f_s},$$

where Ps_i is i-th power sample, f_s is sampling frequency.

4.3.1. Methodology

For energy measurements, we used GPU-accelerated nodes of the EuroHPC Karolina cluster [58] equipped with two AMD EPYC 7763 CPUs and eight NVIDIA A100-SXM4 GPUs. The energy consumption of the GPU is measured using performance counters of the GPU via NVIDIA Management Library (NVML) [59]. For CPU, we use AMD RAPL [60, 61]. The

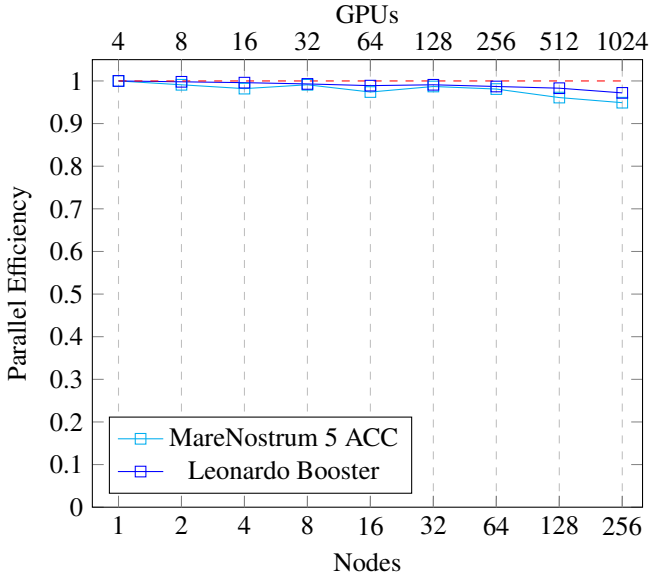

Figure 8: Weak scaling tests of circularly polarized Alfv́en waves on MareNostrum 5 ACC and Leonardo Booster.

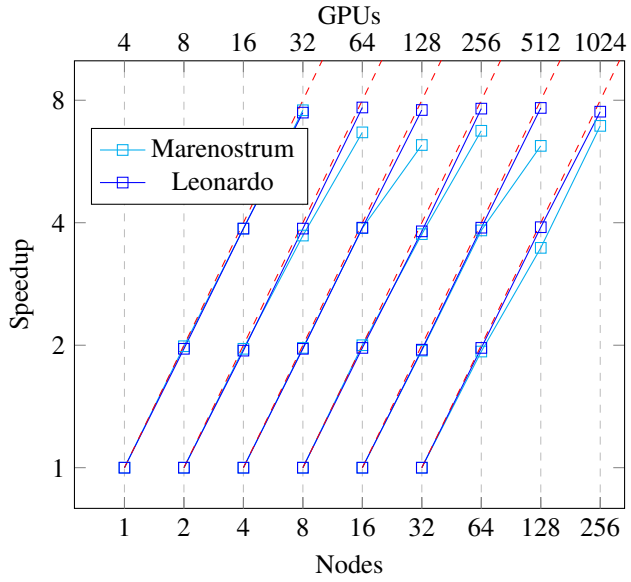

Figure 9: Strong Scaling tests of Circularly Polarized Alfv́en on MareNostrum 5 ACC and Leonardo Booster.

power consumption of the remaining components of the compute node is estimated using a simple mathematical model (see Tab. 6).

To investigate the energy efficiency of the application, we have tuned the frequency of the GPUs' Streaming Multiprocessors (SMs), which is analogous to the CPU core frequency. For this purpose, we use NVML function `nvmlDeviceSetApplicationsClocks()` that sets a specific clock speed of a GPU SMs. Since A100-SXM4 uses HBM2 memory, its frequency cannot be tuned; on GPUs with GDDR memory, on the other hand, the frequency can be scaled using the same function.

Nodes	MareNostrum 5 Efficiency	Leonardo Efficiency
Group 1, Base: 1 Node, Resolution: 832 x 832 x 416		
1	1.00	1.00
2	0.98	0.97
4	0.93	0.97
8	0.83	0.96
Group 2, Base: 2 Nodes, Resolution: 832 x 832 x 832		
2	1.00	1.00
4	0.98	0.98
8	0.97	0.98
16	0.78	0.95
Group 3, Base: 4 Nodes, Resolution: 1664 x 832 x 832		
4	1.00	1.00
8	1.00	0.99
16	0.94	0.95
32	0.84	0.95
Group 4, Base: 8 Nodes, Resolution: 1664 x 1664 x 832		
8	1.00	1.00
16	0.97	0.98
32	0.95	0.97
64	0.81	0.96
Group 5, Base: 16 Nodes, Resolution: 1664 x 1664 x 1664		
16	1.00	1.00
32	0.97	0.99
64	0.96	0.97
128	0.77	0.94
Group 6, Base: 32 Nodes, Resolution: 3328 x 1664 x 1664		
32	1.00	1.00
64	0.96	0.99
128	0.87	0.97
256	0.87	0.94

Table 5: Strong scaling Efficiencies at different resolution for the tests presented in Fig. 9 on MareNostrum 5 ACC and Leonardo Booster.

We use static frequency tuning, meaning that a single GPU SMs frequency is applied at the start of the application execution and remains constant until the end. The key advantage of static tuning is that it can be performed by a standard HPC job scheduler like SLURM [62] or PBS [63] at the job launch time. The disadvantage of this method is that the applied configuration may not be optimal for all executed kernels. Thus, dynamic tuning may bring more energy savings for the same, or even less, performance penalty [64].

4.3.2. Platform analysis

The key properties of the Karolina GPU-accelerated node from the energy consumption point of view are shown in Tab. 6. We break the hardware into 3 key groups: CPUs, GPUs, and the rest of the server (including motherboard, memory DIMMs, network cards, GPU board with NVlink switches, etc.). For

Karolina GPU node	**CPUs**	**GPUs**	**Other parts**	**Total**
Units per server	2× AMD EPYC 7763	8× NVIDIA A100	–	–
Range of power consumption	90–280 W per CPU	55–400 W per GPU	520–625 W per server	–
Static power consumption	180 W	420 W	520 W	1120 W
Dynamic power consumption	max. 380 W	max. 2780 W	max. 105 W	max. 3265 W
PLUTO power consumption range	115–120 W per CPU	99–248 W per GPU	600–610 W	1622–2834 W
PLUTO dynamic pow. consumption	5 W per CPU; 10 W total	149 W per GPU; 1192 W total	10 W	1212 W

Table 6: Power consumption of the key components of the GPU accelerated compute node of the Karolina supercomputer. Static power consumption is always present, even when a server is idle. Dynamic power consumption is additional power consumption when a server is under load (CPUs and GPUs perform some computations) and it is workload-dependent. The large gPLUTO power consumption range for GPUs is caused by both lowering the SM frequency to 990 GHz as well as lowering GPU utilization for the high number of compute nodes in the strong scaling test.

each group, we show the "Range of power consumption", where the lower number is the power consumption of the component when the server is idle (no computations are running) and the higher value presents maximum power consumption under full load. When all values for idle power consumption for all components are summed, we get the "Static power consumption" of the entire server. In this case, just to keep the server turned on, it consumes 1.12 kW. The "Dynamic power consumption" then denotes the additional power that can be consumed when the server is under full load. This is a maximum/theoretical value if a workload is capable of full utilization of the CPUs or GPUs. By summing the static and dynamic consumption, we obtain the server's peak power consumption. Our methodology can only control and reduce dynamic power consumption. From the server user perspective, the static power cannot be controlled, and it is the main reason why energy consumption increases when runtime is extended, as energy is the integral of power over time. Finally, the last two rows in the Tab. present the platform's behavior when gPLUTO runs the 3D Orszag-Tang vortex benchmark. We can see that CPUs and remaining server components consumption is mostly static (because we do not tune CPU frequency), while the key component with predominantly dynamic power consumption are the 8 GPUs. In summary, each compute node can consume between 1.6 and 2.8 kW depending on how many nodes are used for the experiment (we use strong scaling) and the selected frequency of the GPU SMs, resulting in 1.2 kW dynamically controllable power consumption.

	1 node 8 GPUs			2 nodes 16 GPUs			4 nodes 32 GPUs			8 nodes 64 GPUs			16 nodes 128 GPUs			32 nodes 256 GPUs		
SM freq. [MHz]	GPUs	CPUs	Server	GPUs	CPUs	Server	GPUs	CPUs	Server	GPUs	CPUs	Server	GPUs	CPUs	Server	GPUs	CPUs	Server
1410	2,0	0,2	0,6	1,9	0,2	0,6	1,7	0,2	0,6	1,6	0,2	0,6	1,4	0,2	0,6	1,2	0,2	0,6
1350	1,8	0,2	0,6	1,7	0,2	0,6	1,5	0,2	0,6	1,5	0,2	0,6	1,2	0,2	0,6	1,1	0,2	0,6
1290	1,6	0,2	0,6	1,5	0,2	0,6	1,4	0,2	0,6	1,3	0,2	0,6	1,2	0,2	0,6	1,0	0,2	0,6
1230	1,5	0,2	0,6	1,4	0,2	0,6	1,3	0,2	0,6	1,2	0,2	0,6	1,1	0,2	0,6	1,0	0,2	0,6
1170	1,4	0,2	0,6	1,3	0,2	0,6	1,2	0,2	0,6	1,2	0,2	0,6	1,0	0,2	0,6	0,9	0,2	0,6
1110	1,3	0,2	0,6	1,2	0,2	0,6	1,1	0,2	0,6	1,1	0,2	0,6	1,0	0,2	0,6	0,9	0,2	0,6
1050	1,2	0,2	0,6	1,2	0,2	0,6	1,1	0,2	0,6	1,0	0,2	0,6	0,9	0,2	0,6	0,8	0,2	0,6
990	1,2	0,2	0,6	1,1	0,2	0,6	1,0	0,2	0,6	1,0	0,2	0,6	0,9	0,2	0,6	0,8	0,2	0,6

	1 node 8 GPUs			2 nodes 16 GPUs			4 nodes 32 GPUs			8 nodes 64 GPUs			16 nodes 128 GPUs			32 nodes 256 GPUs		
SM freq. [MHz]	GPUs	CPUs	Server	GPUs	CPUs	Server	GPUs	CPUs	Server	GPUs	CPUs	Server	GPUs	CPUs	Server	GPUs	CPUs	Server
1410	70	8	21	69	9	22	67	9	24	66	10	24	62	11	27	58	12	30
1350	68	9	23	67	9	24	65	10	25	64	10	26	60	11	29	57	12	31
1290	66	10	24	65	10	25	63	10	27	62	11	28	58	12	30	55	12	33
1230	64	10	26	63	10	27	61	11	28	60	11	29	57	12	31	54	13	33
1170	62	11	27	61	11	28	59	11	29	58	12	30	55	13	33	52	13	35
1110	61	11	28	60	11	29	58	12	30	57	12	31	53	13	34	51	14	36
1050	59	12	29	58	12	30	56	12	31	55	13	32	52	13	35	49	14	37
990	58	12	30	57	12	31	56	13	32	54	13	33	51	14	35	49	14	37

Figure 10: Power consumption per node for the 3D Orszag-Tang Vortex benchmark. The top panel shows absolute values in [kW]. The bottom one shows the relative values with respect to the entire node power consumption in [%].

The compute node behavior is presented in Fig. 10 in more detail. We can see that: (a) by changing the SM frequency from the default 1.41 GHz to 0.99 GHz the power consumption of all 8 GPUs per node is reduced from 2.0 to 1.2 kW (when the benchmark runs on one node); (b) by performing the strong scaling from 1 to 32 compute nodes, we also reduce GPUs power consumption per node from 2.0 to 1.2 kW. This is caused by the strong scaling because the problem size per GPU is halved with every step of strong scaling. This results in lower utilization of the GPUs and, therefore, lower average power consumption. For 32 nodes, if we scale the SM frequency down to 0.99 GHz, we can further reduce GPU power consumption to 0.8 kW. It is important to realize that, for this configuration, the static power consumption of the non-GPU components amounts to half the total. When this effect is combined with the extended runtime due to SM frequency reduction (for 32 nodes, reducing the SM frequency from 1.4 to 0.99 GHz yields a runtime extension to 117.6 % - see Figure 11 top) it becomes dominant in terms of energy consumption of the computation. This explains why the highest energy savings (see bottom panels of Fig. 11 and 12) are not achieved for the lowest SM frequency we used.

4.3.3. Energy consumption optimization results

We have performed strong scaling from 1 to 32 GPU-accelerated compute nodes of Karolina for both 3D circularly polarized Alfvén wave and 3D Orszag-Tang vortex benchmarks. For each number of nodes, we scaled down the SM frequency from 1.4 to 0.99 GHz and measured the runtime and energy consumption of all compute nodes employed for the benchmark. The measurement results are in Fig. 11 and 12. We do not present results for frequencies lower than 0.99 GHz because, for all cases, the energy savings for 0.99 GHz are smaller than the savings for 1.11 and 1.05 GHz.

SM freq. [MHz]	1 node 8 GPUs	2 nodes 16 GPUs	4 nodes 32 GPUs	8 nodes 64 GPUs	16 nodes 128 GPUs	32 nodes 256 GPUs
1410/ **default**	129 s **100%**	68,2 s **100%**	38,1 s **100%**	21,6 s **100%**	15,5 s **100%**	11,2 s **100,0%**
1350	102,8%	102,3%	102,8%	102,6%	100,5%	100,8%
1290	105,8%	105,4%	105,3%	105,2%	102,1%	101,1%
1230	108,9%	108,4%	108,4%	108,0%	104,8%	103,7%
1170	112,6%	112,1%	111,4%	113,1%	108,8%	107,5%
1110	116,2%	115,3%	115,5%	115,9%	112,9%	110,0%
1050	120,7%	119,9%	119,7%	121,2%	116,3%	115,2%
990	125,7%	125,0%	124,1%	123,9%	120,5%	117,6%

SM freq. [MHz]	1 node 8 GPUs	2 nodes 16 GPUs	4 nodes 32 GPUs	8 nodes 64 GPUs	16 nodes 128 GPUs	32 nodes 256 GPUs
1410/ **default**	365 kJ **100%**	368 kJ **100%**	387 kJ **100%**	423 kJ **100%**	543 kJ **100%**	719 kJ **100%**
1350	95,7%	94,9%	96,2%	96,4%	95,7%	96,4%
1290	92,0%	92,2%	93,5%	93,7%	93,7%	93,5%
1230	90,1%	90,4%	91,7%	91,8%	91,9%	93,7%
1170	88,8%	89,4%	90,5%	91,8%	91,8%	93,0%
1110	87,6%	88,4%	90,2%	90,9%	92,1%	92,8%
1050	87,4%	88,5%	90,4%	91,7%	92,4%	94,5%
990	88,9%	90,5%	92,1%	92,5%	94,3%	95,5%

Figure 11: Impact of the static tuning of the GPU SM frequency on the runtime (top) and energy consumption (bottom) of the 3D Orszag-Tang vortex benchmark. The top panel shows runtime variations with respect to the default execution time shown in the first line. The bottom one shows relative energy consumption with respect to the energy consumption of the default execution, as shown in the first line of the table.

SM freq. [MHz]	1 node 8 GPUs	2 nodes 16 GPUs	4 nodes 32 GPUs	8 nodes 64 GPUs	16 nodes 128 GPUs	32 nodes 256 GPUs
1410/ **default**	189 s **100%**	96,3 s **100%**	51,1 s **100%**	29,1 s **100%**	18,1 s **100%**	13,2 s **100%**
1350	101,9%	101,7%	101,1%	102,0%	103,1%	100,6%
1290	104,1%	103,4%	103,2%	104,0%	103,5%	101,2%
1230	106,5%	106,1%	105,3%	105,9%	106,7%	103,5%
1170	109,3%	108,8%	108,5%	108,0%	110,0%	106,4%
1110	112,2%	111,5%	111,6%	112,0%	113,4%	108,5%
1050	116,0%	114,7%	114,8%	115,8%	116,6%	111,9%
990	120,5%	119,0%	119,0%	119,8%	120,2%	116,2%

SM freq. [MHz]	1 node 8 GPUs	2 nodes 16 GPUs	4 nodes 32 GPUs	8 nodes 64 GPUs	16 nodes 128 GPUs	32 nodes 256 GPUs
1410/ **default**	524 kJ **100%**	519 kJ **100%**	534 kJ **100%**	575 kJ **100%**	678 kJ **100%**	885 kJ **100%**
1350	95,3%	95,1%	95,1%	95,9%	96,7%	96,0%
1290	91,3%	91,9%	91,9%	92,8%	93,0%	93,2%
1230	88,8%	89,6%	90,2%	90,7%	91,7%	92,1%
1170	86,7%	88,2%	88,5%	89,1%	90,9%	91,6%
1110	85,6%	87,1%	87,6%	88,9%	90,6%	90,9%
1050	85,2%	86,5%	87,3%	89,0%	90,5%	91,3%
990	86,7%	88,0%	88,8%	90,6%	92,0%	93,4%

Figure 12: Same as Fig. 11 but for the 3D circularly polarized Alfvén test.

If we consider acceptable an increase in runtime up to 6% (values close to this threshold are highlighted in red in both figures), we see that we can save up to 8 % of energy for the 3D Orszag-Tang vortex and up to 11 % for the 3D circularly polarized Alfvén wave benchmark. We also see that by increasing the number of compute nodes we gain lower savings. As mentioned previously, this is due to a high ratio of static power consumption and extended runtime.

We want to highlight that these servers are air-cooled, and therefore, the energy consumption of the remaining components is relatively high. By using modern hot water-cooled servers and racks, the static power consumption can be reduced, with a corresponding increase in energy savings.

5. Summary

In this paper we have presented a first look into gPLUTO, a GPU-optimized implementation of the widely used PLUTO code for computational plasma astrophysics. The new code is written in C++ and adopts OpenACC as a programming model to efficiently exploit modern GPU architectures. The new implementation maintains the flexibility and robustness of the legacy code while significantly improving computational performance, power efficiency, and scalability on HPC clusters.

This paper outlines the fundamental design principles behind gPLUTO, including the numerical methods employed, the OpenACC-based parallelization strategy, memory management optimizations, and multi-GPU communication. The performance of gPLUTO has been evaluated through two numerical benchmarks: the 3D Orszag-Tang vortex and the 3D circularly polarized Alfvén wave, conducted on three European HPC platforms: Marenostrum 5, Leonardo, and Meluxina. Both weak and strong scaling tests have been conducted.

Results demonstrate excellent strong and weak scaling efficiencies, maintaining around 90% performance across multiple nodes, in a situation of high GPU occupancy. Direct comparison between the GPU- and CPU-specialized partitions of Leonardo reveals a factor $\approx$ 9 acceleration (per node usage). The results of this study confirm that gPLUTO successfully harnesses the power of GPUs to enhance the performance of MHD simulations.

The strong scaling behavior suggests that gPLUTO can efficiently utilize a large number of GPUs, making it well-suited for exascale computing applications.

Energy efficiency analyses highlight the impact of GPU frequency tuning in optimizing computational power consumption. Our experiments indicate that adjusting GPU clock frequencies can lead to significant energy savings with minimal runtime penalties, highlighting potential avenues for optimizing HPC resource utilization. In particular, lowering the SMs frequency can save around 8 % and 11 % of energy consumption for 3D Orszag-Tang tests and circularly polarized Alfvén waves, respectively, at a cost of 6 % longer execution time.

The development of gPLUTO represents a significant step toward enabling more efficient and scalable astrophysical simulations on next-generation, Exascale supercomputers. A more

comprehensive description of the code along with all its physical modules (including relativistic and non-ideal MHD extensions, particles and different systems of coordinates) and the implemented numerical methods - including high-order methods [65, 66] - will be presented in a forthcoming work. We also invite to refer to the companion paper [41] for the Lagrangian Particles module integration in gPLUTO. Future works on HPC porting will also explore additional acceleration frameworks such as OpenMP for execution on AMD-based HPC systems.

Acknowledgements

MR and AM wish to thank F. Spiga from NVIDIA for his active support during the very early stages of this development. MR and AM wish also to thank M. Bettencourt for his technical guidance and precious advices while porting the code to C++.

This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain under grant agreement No 101093441 (SPACE).

This project has received funding from the European Union's Horizon Europe research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 101064953 (GR-PLUTO).

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic through the e-INFRA CZ (ID:90254).

We acknowledge ISCRA for awarding this project access to the Leonardo supercomputer and the EuroHPC Joint Undertaking for granting us access to the Leonardo, MareNostrum and MeluXina supercomputer through an EuroHPC [Extreme/Regular/Benchmark/Development/...] Access call.

This paper is supported by the Fondazione ICSC, Spoke 3 Astrophysics and Cosmos Observations and National Recovery and Resilience Plan (Piano Nazionale di Ripresa e Resilienza, PNRR), Project ID CN_00000013 "Italian Research Center on High-Performance Computing, Big Data and Quantum Computing" funded by MUR Missione 4 Componente 2 Investimento 1.4: Potenziamento strutture di ricerca e creazione di "campioni nazionali di R&S (M4C2-19)" - Next Generation EU (NGEU).

References

[1] G. Lesur, Magnetohydrodynamics of protoplanetary discs, Journal of Plasma Physics 87 (2021) 205870101. doi:10.1017/S0022377820001002.

[2] R. Teyssier, B. Commerçon, Numerical Methods for Simulating Star Formation, Frontiers in Astronomy and Space Sciences 6 (2019) 51. doi:10.3389/fspas.2019.00051.

[3] D. Zhang, A Review of the Theory of Galactic Winds Driven by Stellar Feedback, Galaxies 6 (2018) 114. doi:10.3390/galaxies6040114.

[4] A. Lazarian, L. Vlahos, G. Kowal, H. Yan, A. Beresnyak, E. M. de Gouveia Dal Pino, Turbulence, Magnetic Reconnection in Turbulent Fluids and Energetic Particle Acceleration, Space Science Reviews 173 (1-4) (2012) 557–622. arXiv:1211.0008, doi:10.1007/s11214-012-9936-7.

[5] E. Amato, P. Blasi, Cosmic ray transport in the Galaxy: A review, Advances in Space Research 62 (10) (2018) 2731–2749. arXiv:1704.05696, doi:10.1016/j.asr.2017.04.019.

[6] R. Narayan, Black holes in astrophysics, New Journal of Physics 7 (1) (2005) 199. arXiv:gr-qc/0506078, doi:10.1088/1367-2630/7/1/199.

[7] J. A. Faber, F. A. Rasio, Binary Neutron Star Mergers, Living Reviews in Relativity 15 (1) (2012) 8. arXiv:1204.3858, doi:10.12942/lrr-2012-8.

[8] D. A. Uzdensky, S. Rightley, Plasma physics of extreme astrophysical environments, Reports on Progress in Physics 77 (3) (2014) 036902. arXiv:1401.5110, doi:10.1088/0034-4885/77/3/036902.

[9] B. Knaepen, R. Moreau, Magnetohydrodynamic Turbulence at Low Magnetic Reynolds Number, Annual Review of Fluid Mechanics 40 (1) (2008) 25–45. doi:10.1146/annurev.fluid.39.050905.110231.

[10] G. Inghirami, L. Del Zanna, A. Beraudo, M. H. Moghaddam, F. Becattini, M. Bleicher, Numerical magneto-hydrodynamics for relativistic nuclear collisions, European Physical Journal C 76 (12) (2016) 659. arXiv:1609.03042, doi:10.1140/epjc/s10052-016-4516-8.

[11] K. Shibata, T. Magara, Solar Flares: Magnetohydrodynamic Processes, Living Reviews in Solar Physics 8 (1) (2011) 6. doi:10.12942/lrsp-2011-6.

[12] E. J. Doyle, W. A. Houlberg, Y. Kamada, V. Mukhovatov, T. H. Osborne, A. Polevoi, G. Bateman, J. W. Connor, J. G. Cordey, T. Fujita, X. Garbet, T. S. Hahm, L. D. Horton, A. E. Hubbard, F. Imbeaux, F. Jenko, J. E. Kinsey, Y. Kishimoto, J. Li, T. C. Luce, Y. Martin, M. Ossipenko, V. Parail, A. Peeters, T. L. Rhodes, J. E. Rice, C. M. Roach, V. Rozhansky, F. Ryter, G. Saibene, R. Sartori, A. C. C. Sips, J. A. Snipes, M. Sugihara, E. J. Synakowski, H. Takenaga, T. Takizuka, K. Thomsen, M. R. Wade, H. R. Wilson, ITPA Transport Physics Topical Group, I. Confinement Database, Modelling Topical Group, I. Pedestal, Edge Topical Group, Chapter 2: Plasma confinement and transport, Nuclear Fusion 47 (6) (2007) S18–S127. doi:10.1088/0029-5515/47/6/S02.

[13] A. Mezzacappa, Toward Realistic Models of Core Collapse Supernovae: A Brief Review, in: D. Bisikalo, D. Wiebe, C. Boily (Eds.), The Predictive Power of Computational Astrophysics as a Discover Tool, Vol. 362 of IAU Symposium, 2023, pp. 215–227. arXiv:2205.13438, doi:10.1017/S1743921322001831.

[14] H. T. Janka, Long-Term Multidimensional Models of Core-Collapse Supernovae: Progress and Challenges, arXiv e-prints (2025) arXiv:2502.14836arXiv:2502.14836, doi:10.48550/arXiv.2502.14836.

[15] P. Charbonneau, Dynamo models of the solar cycle, Living Reviews in Solar Physics 17 (1) (2020) 4. doi:10.1007/s41116-020-00025-6.

[16] L. Del Zanna, N. Tomei, K. Franceschetti, M. Bugli, N. Bucciantini, General Relativistic Magnetohydrodynamics Mean-Field Dynamos, Fluidika 7 (2) (2022) 87. doi:10.3390/fluids7020087.

[17] L. Mayer, Super-Eddington accretion; flow regimes and conditions in high-z galaxies, in: M. Latif, D. Schleicher (Eds.), Formation of the First Black Holes, 2019, pp. 195–222. doi:10.1142/9789813227958_0011.

[18] F. Cattorini, B. Giacomazzo, GRMHD study of accreting massive black hole binaries in astrophysical environment: A review, Astroparticle Physics 154 (2024) 102892. arXiv:2401.02521, doi:10.1016/j.astropartphys.2023.102892.

[19] M. T. P. Liska, K. Chatterjee, D. Issa, D. Yoon, N. Kaaz, A. Tchekhovskoy, D. van Eijnatten, G. Musoke, C. Hesp, V. Rohoza, S. Markoff, A. Ingram, M. van der Klis, H-AMR: A New GPU-accelerated GRMHD Code for Exascale Computing with 3D Adaptive Mesh Refinement and Local Adaptive Time Stepping, The Astrophysical Journal Supplement Series 263 (2) (2022) 26. arXiv:1912.10192, doi:10.3847/1538-4365/ac9966.

[20] G. R. J. Lesur, S. Baghdadi, G. Wafflard-Fernandez, J. Mauxion, C. M. T. Robert, M. Van den Bossche, IDEFIX: A versatile performance-portable Godunov code for astrophysical flows, Journal of Computational Physics 677 (2023) A9. arXiv:2304.13746, doi:10.1051/0004-6361/202346005.

[21] J. M. Stone, P. D. Mullen, D. Fielding, P. Grete, M. Guo, P. Kempski, E. R. Most, C. J. White, G. N. Wong, AthenaK: A Performance-Portable Version of the Athena++ AMR Framework, arXiv e-prints (2024) arXiv:2409.16053arXiv:2409.16053, doi:10.48550/arXiv.

2409.16053.

[22] L. Del Zanna, S. Landi, L. Serafini, M. Bugli, E. Papini, A GPU-Accelerated Modern Fortran Version of the ECHO Code for Relativistic Magnetohydrodynamics, Fluids 9 (1) (2024) 16. doi:10.3390/fluids9010016.

[23] J. V. Kalinani, L. Ji, L. Ennoggi, F. G. Lopez Armengol, L. Timotheo Sanches, B.-J. Tsao, S. R. Brandt, M. Campanelli, R. Ciolfi, B. Giacomazzo, R. Haas, E. Schnetter, Y. Zlochower, AsterX: a new open-source GPU-accelerated GRMHD code for dynamical spacetimes, Classical and Quantum Gravity 42 (2) (2025) 025016. arXiv:2406.11669, doi:10.1088/1361-6382/ad9c11.

[24] A. Mignone, G. Bodo, S. Massaglia, T. Matsakos, O. Tesileanu, C. Zanni, A. Ferrari, PLUTO: A Numerical Code for Computational Astrophysics, The Astrophysical Journal Supplement Series 170 (2007) 228–242. arXiv:astro-ph/0701854, doi:10.1086/513316.

[25] A. Mignone, C. Zanni, P. Tzeferacos, B. van Straalen, P. Colella, G. Bodo, The PLUTO Code for Adaptive Mesh Computations in Astrophysical Fluid Dynamics, The Astrophysical Journal Supplement Series 198 (2012) 7. arXiv:1110.0740, doi:10.1088/0067-0049/198/1/7.

[26] A. Mignone, P. Rossi, G. Bodo, A. Ferrari, S. Massaglia, High-resolution 3D relativistic MHD simulations of jets, Monthly Notices of the Royal Astronomical Society 402 (1) (2010) 7–12. arXiv:0908.4523, doi:10.1111/j.1365-2966.2009.15642.x.

[27] A. Mignone, E. Striani, M. Tavani, A. Ferrari, Modelling the kinked jet of the Crab nebula, Monthly Notices of the Royal Astronomical Society 436 (2) (2013) 1102–1115. arXiv:1309.0375, doi:10.1093/mnras/stt1632.

[28] G. Mattia, L. Del Zanna, M. Bugli, A. Pavan, R. Ciolfi, G. Bodo, A. Mignone, Resistive relativistic MHD simulations of astrophysical jets, Journal of Computational Physics 679 (2023) A49. arXiv:2308.09477, doi:10.1051/0004-6361/202347126.

[29] A. Pavan, R. Ciolfi, J. V. Kalinani, A. Mignone, Jet-environment interplay in magnetized binary neutron star mergers, Monthly Notices of the Royal Astronomical Society 524 (1) (2023) 260–275. arXiv:2211.10135, doi:10.1093/mnras/stad1809.

[30] S. Orlando, F. Reale, G. Peres, A. Mignone, Mass accretion to young stars triggered by flaring activity in circumstellar discs, Monthly Notices of the Royal Astronomical Society 415 (4) (2011) 3380–3392. arXiv:1104.5107, doi:10.1111/j.1365-2966.2011.18954.x.

[31] B. Olmi, N. Bucciantini, G. Morlino, Numerical simulations of mass loading in the tails of bow-shock pulsar-wind nebulae, Monthly Notices of the Royal Astronomical Society 481 (3) (2018) 3394–3400. arXiv:1809.03807, doi:10.1093/mnras/sty2525.

[32] B. Olmi, N. Bucciantini, On the origin of jet-like features in bow shock pulsar wind nebulae, Monthly Notices of the Royal Astronomical Society 490 (3) (2019) 3608–3615. arXiv:1910.01926, doi:10.1093/mnras/stz2819.

[33] S. Orlando, M. Miceli, O. Petruk, M. Ono, S. Nagataki, M. A. Aloy, P. Mimica, S. H. Lee, F. Bocchino, G. Peres, M. Guarrasi, 3D MHD modeling of the expanding remnant of SN 1987A. Role of magnetic field and non-thermal radio emission, Journal of Computational Physics 622 (2019) A73. arXiv:1812.00021, doi:10.1051/0004-6361/201834487.

[34] S. Orlando, M. Ono, S. Nagataki, M. Miceli, H. Umeda, G. Ferrand, F. Bocchino, O. Petruk, G. Peres, K. Takahashi, T. Yoshida, Hydrodynamic simulations unravel the progenitor-supernova-remnant connection in SN 1987A, Journal of Computational Physics 636 (2020) A22. arXiv:1912.03070, doi:10.1051/0004-6361/201936718.

[35] G. Bodo, G. Mamatsashvili, P. Rossi, A. Mignone, Current-driven kink instabilities in relativistic jets: dissipation properties, Monthly Notices of the Royal Astronomical Society 510 (2) (2022) 2391–2406. arXiv:2111.14575, doi:10.1093/mnras/stab3492.

[36] M. Bugli, E. F. Lopresti, E. Figueiredo, A. Mignone, B. Cerutti, G. Mattia, L. Del Zanna, G. Bodo, V. Berta, Relativistic reconnection with effective resistivity: I. Dynamics and reconnection rate, Journal of Computational Physics 693 (2025) A233. arXiv:2410.20924, doi:10.1051/0004-6361/202452277.

[37] G. R. J. Lesur, Systematic description of wind-driven protoplanetary discs, Journal of Computational Physics 650 (2021) A35. arXiv:2101.10349, doi:10.1051/0004-6361/202040109.

[38] M. Flock, A. Mignone, Streaming instability in a global patch simulation of protoplanetary disks, Journal of Computational Physics 650 (2021) A119. arXiv:2103.15146, doi:10.1051/0004-6361/202040104.

[39] V. Réville, A. S. Brun, S. P. Matt, A. Strugarek, R. F. Pinto, The Effect of Magnetic Topology on Thermally Driven Wind: Toward a General Formulation of the Braking Law, The Astrophysical Journal 798 (2015) 116. doi:10.1088/0004-637X/798/2/116.

[40] B. Perri, A. S. Brun, V. Réville, A. Strugarek, Simulations of solar wind variations during an 11-year cycle and the influence of north-south asymmetry, Journal of Plasma Physics 84 (2018) 765840501. doi:10.1017/S0022377818000880.

[41] A. Suriano, S. Truzzi, A. Costa, M. Rossazza, N. Shukla, V. Berta, A. Mignone, C. Zanni, The PLUTO Code on GPUs: Offloading Lagrangian Particle Methods, Manuscript submitted for publication (companion paper) (2025).

[42] K. G. Powell, P. L. Roe, T. J. Linde, T. I. Gombosi, D. L. De Zeeuw, A Solution-Adaptive Upwind Scheme for Ideal Magnetohydrodynamics, Journal of Computational Physics 154 (2) (1999) 284–309. doi:10.1006/jcph.1999.6299.

[43] A. Dedner, F. Kemm, D. Kröner, C.-D. Munz, T. Schnitzer, M. Wesenberg, Hyperbolic Divergence Cleaning for the MHD Equations, Journal of Computational Physics 175 (2002) 645–673. doi:10.1006/jcph.2001.6961.

[44] A. Mignone, P. Rossi, G. Bodo, A. Ferrari, S. Massaglia, High-resolution 3D relativistic MHD simulations of jets, Monthly Notices of the Royal Astronomical Society 402 (2010) 7–12. arXiv:0908.4523, doi:10.1111/j.1365-2966.2009.15642.x.

[45] C. R. Evans, J. F. Hawley, Simulation of magnetohydrodynamic flows - A constrained transport method, The Astrophysical Journal 332 (1988) 659–677. doi:10.1086/166684.

[46] D. S. Balsara, D. S. Spicer, A Staggered Mesh Algorithm Using High Order Godunov Fluxes to Ensure Solenoidal Magnetic Fields in Magnetohydrodynamic Simulations, Journal of Computational Physics 149 (1999) 270–292. doi:10.1006/jcph.1998.6153.

[47] A. Mignone, L. Del Zanna, Systematic construction of upwind constrained transport schemes for MHD, Journal of Computational Physics 424 (2021) 109748. arXiv:2004.10542, doi:10.1016/j.jcp.2020.109748.

[48] C.-W. Shu, S. Osher, Efficient Implementation of Essentially Non-oscillatory Shock-Capturing Schemes, Journal of Computational Physics 77 (2) (1988) 439–471. doi:10.1016/0021-9991(88)90177-5.

[49] P. Colella, P. R. Woodward, The Piecewise Parabolic Method (PPM) for Gas-Dynamical Simulations, Journal of Computational Physics 54 (1984) 174–201. doi:10.1016/0021-9991(84)90143-8.

[50] A. Suresh, H. T. Huynh, Accurate Monotonicity-Preserving Schemes with Runge Kutta Time Stepping, Journal of Computational Physics 136 (1997) 83–99. doi:10.1006/jcph.1997.5745.

[51] R. Borges, M. Carmona, B. Costa, W. S. Don, An improved weighted essentially non-oscillatory scheme for hyperbolic conservation laws, Journal of Computational Physics 227 (6) (2008) 3191–3211. doi:10.1016/j.jcp.2007.11.038.

[52] P. Londrillo, L. del Zanna, On the divergence-free condition in Godunov-type schemes for ideal magnetohydrodynamics: the upwind constrained transport method, Journal of Computational Physics 195 (2004) 17–48. arXiv:astro-ph/0310183, doi:10.1016/j.jcp.2003.09.016.

[53] A. Mignone, T. Plewa, G. Bodo, The Piecewise Parabolic Method for Multidimensional Relativistic Fluid Dynamics, The Astrophysical Journal Supplement Series 160 (1) (2005) 199–219. arXiv:astro-ph/0505200, doi:10.1086/430905.

[54] A. Mignone, G. Muscianisib, M. Rivib, G. Bodoc, I/o optimization strategies in the pluto code.

[55] C. Helzel, J. A. Rossmanith, B. Taetz, An unstaggered constrained transport method for the 3D ideal magnetohydrodynamic equations, Journal of Computational Physics 230 (10) (2011) 3803–3829. arXiv:1007.2606, doi:10.1016/j.jcp.2011.02.009.

[56] T. Miyoshi, K. Kusano, A multi-state HLL approximate Riemann solver for ideal magnetohydrodynamics, Journal of Computational Physics 208 (1) (2005) 315–344. doi:10.1016/j.jcp.2005.02.017.

[57] A. Mignone, P. Tzeferacos, G. Bodo, High-order conservative finite difference glm–mhd schemes for cell-centered mhd, Journal of Computational Physics 229 (17) (2010) 5896–5920. doi:10.1016/j.jcp.2010.04.013.

URL http://dx.doi.org/10.1016/j.jcp.2010.04.013

[58] IT4Innovations, Karolina supercomputer User Guide, https://docs.it4i.cz/karolina/introduction/, "[Online; accessed 2025-02-28]" (February 2025).

[59] NVIDIA Corp., NVML API Reference, https://docs.nvidia.com/deploy/nvml-api/nvml-api-reference.html, "[Online; accessed 2025-02-28]" (January 2025).

[60] AMD, Kernel driver amd_energy, https://github.com/amd/amd_energy, "[Online; accessed 2025-02-28]".

[61] R. Schone, T. Ilsche, M. Bielert, M. Velten, M. Schmidl, D. Hackenberg, Energy efficiency aspects of the AMD zen 2 architecture, 2021 IEEE International Conference on Cluster Computing (CLUSTER) abs/2108.00808 (2021) 562–571.
URL https://arxiv.org/abs/2108.00808

[62] A. B. Yoo, M. A. Jette, M. Grondona, Slurm: Simple linux utility for resource management, in: D. Feitelson, L. Rudolph, U. Schwiegelshohn (Eds.), Job Scheduling Strategies for Parallel Processing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 44–60.

[63] B. Nitzberg, J. M. Schopf, J. P. Jones, PBS Pro: Grid computing and scheduling attributes, Kluwer Academic Publishers, USA, 2004, p. 183–190.

[64] F. Catthoor, T. Basten, N. Zompakis, M. Geilen, P. G. Kjeldsberg, System-Scenario-based Design Principles and Applications, 1, Springer International Publishing, 2020. doi:10.1007/978-3-030-20343-6.

[65] V. Berta, A. Mignone, M. Bugli, G. Mattia, A 4^{th}-order accurate finite volume method for ideal classical and special relativistic MHD based on pointwise reconstructions, Journal of Computational Physics 499 (2024) 112701. arXiv:2310.11831, doi:10.1016/j.jcp.2023.112701.

[66] A. Mignone, V. Berta, M. Rossazza, M. Bugli, G. Mattia, L. Del Zanna, L. Pareschi, A fourth-order accurate finite volume scheme for resistive relativistic MHD, Monthly Notices of the Royal Astronomical Society 533 (2) (2024) 1670–1686. arXiv:2407.08519, doi:10.1093/mnras/stae1729.