Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 267 (2025) 112–123

Procedia Computer Science

www.elsevier.com/locate/procedia

Proceedings of the Third EuroHPC user day

Towards Exascale Computing for Astrophysical Simulation Leveraging the Leonardo EuroHPC System

Nitin Shukla[a,*], Alessandro Romeo[a,*], Caterina Caravita[a], Michael Redenti[a], Radim Vavrik[b], Lubomir Riha[b], Andrea Mignone[c], Marco Rossazza[c], Stefano Truzzi[c], Luca Tornatore[d], Antonio Ragagnin[d], Tiago Castro[d], Geray S. Karademir[e], Klaus Dolag[e], Pranab J. Deka[f], Fabio Bacchini[f], Rostislav-Paul Wilhelm[f], Daniele Gregori[g], Elisabetta Boella[g]

[a]*CINECA, Casalecchio di Reno, Italy*
[b]*IT4Innovations, VSB-TU Ostrava, Ostrava, Czech Republic*
[c]*University of Turin, Turin, Italy*
[d]*INAF - Osservatorio Astronomico di Trieste, Trieste, Italy*
[e]*Universitäts-Sternwarte, Fakultät für Physik, Ludwig-Maximilians-Universität München, Munich, Germany*
[f]*KU Leuven, Leuven, Belgium*
[g]*E4 Computer Engineering SpA, Scandiano, Italy*

Abstract

Developing and redesigning astrophysical, cosmological, and space plasma numerical codes for existing and next-generation accelerators is critical for enabling large-scale simulations. To address these challenges, the SPACE Center of Excellence (SPACE-CoE) fosters collaboration between scientists, code developers, and high-performance computing experts to optimize applications for the exascale era. This paper presents our strategy and initial results on the Leonardo system at CINECA for three flagship codes, namely `gPLUTO`, `OpenGadget3` and `iPIC3D`, using profiling tools to analyze performance on single and multiple nodes. Preliminary tests show all three codes scale efficiently, reaching 80% scalability up to 1,024 GPUs.

Keywords: Exascale; EuroHPC; HPC; astrophysics; space plasma; simulation; performance analysis; CI/CD; GPU scaling; SPACE CoE

1. Introduction

High-performance computing (HPC) has transformed large-scale astrophysical simulations, enabling researchers to study complex phenomena such as dark matter interactions [1, 2], gravitational lensing [3, 4], magnetic reconnection [5, 6, 7], X-ray emissions from galaxy clusters [8], and γ-ray bursts driven by fireball beam–plasma interactions

* Corresponding author.
E-mail addresses: n.shukla@cineca.it (Nitin Shukla), a.romeo@cineca.it (Alessandro Romeo)

10.1016/j.procs.2025.08.238

[9, 10]. Advances in HPC hardware, particularly in GPU architectures and high-speed interconnect technologies, are driving progress towards exascale computing (10^{18} calculations per second). These improvements enable significantly larger, more detailed, and faster simulations. To fully exploit this capability, scientific codes need to adopt new programming models. The SPACE-CoE project brings together scientists, developers, and tech experts to redesign astrophysical software for exascale systems. It focuses on developing efficient, portable, and scalable applications, encouraging collaboration in the European astrophysics community through shared tools, data standards, and development practices.

The SPACE-CoE [11] focuses on adapting seven widely used open-source astrophysics codes, covering magnetohydrodynamics, cosmology, general relativity, and space plasma physics, including `gPLUTO` [12], `OpenGadget3` [13], and `iPIC3D` [14], for which CINECA co-develops GPU-centric algorithms. While these codes are already optimized for CPU parallelism, leveraging the full power of modern EuroHPC GPU-based super computers requires further optimization to enhance performance and scalability. CINECA plays a key role in code redesign, optimization, profiling, and performance analysis, as well as supporting the project on the Leonardo system. This paper is organized as follows: Section 2 gives an overview of the codes and their POP3 [15] analysis. Section 3 covers GPU-focused development and code validation after each change. Section 4 outlines our benchmarking process and software engineering practices adapted for HPC. We present profiling and performance results for each application.

2. Code development workflows

In this section, we describe our collaborative work with the three European research teams involved in the development activity. We also provide a brief overview of the main features of the codes and how they leverage GPU computational capabilities. The three aforementioned astrophysical codes tackle a diverse range of scientific problems, involving different algorithmic methodologies and approaches.

2.1. *Implementation of `gPLUTO`*

`gPLUTO` is the GPU-enabled version of `PLUTO`, a multi-algorithm framework for solving the equations governing plasma dynamics at high Mach numbers. It supports a range of physical models, including the compressible Navier-Stokes equations, ideal Magneto Hydro Dynamics (MHD), relativistic MHD (RMHD) and resistive relativistic MHD (ResRMHD). The new code version is implemented in `C++` and employs Godunov-type finite volume solvers for hyperbolic and parabolic MHD conservation laws in up to three spatial dimensions, on both static and mapped grids. Currently, `gPLUTO` implements approximately 65% of the modules and features available in the original `PLUTO` code. Many of the missing features of `PLUTO`, such as adaptive mesh refinement (AMR) grids, heating and cooling mechanisms, dissipation process, etc., are being ported to the GPU. The code uses `MPI` to handle multiple tasks, while GPU acceleration is handled through `OpenACC` (Rossazza et al., submitted). GPU offloading using `OpenMP` is currently in progress and will be included in a future release of the code.

2.2. *Implementation of `OpenGadget3`*

`OpenGadget3` (Dolag et al., in prep.) is an open-source version of a `Gadget2` [16] code successor. It is a N-Body `C/C++` code and deals with several astrophysical scales. Gravitational force uses Barnes & Hut algorithm [17] for short range interactions and particle mesh algorithm to treat large scales, performed with the use of `FFTW3` libraries [18]. The code employs an advanced smoothed particle hydrodynamics (SPH) solver [19]. As an alternative hydro solver, the code is also equipped with a meshless finite mass method [13]. Moreover, this code also includes other baryonic physics processes, such as radiative cooling, star formation, energy feedback, radiative transfer, magnetic fields, and black holes. The code uses a hybrid `MPI + OpenMP` parallelization scheme, exploiting Hilbert space filling curve locality [20], and both inter-node and intra-node parallelism. GPU offloading is currently implemented via `OpenACC` [21], while support for `OpenMP` offloading is under development.

2.3. Implementation of `iPIC3D`

`iPIC3D` [14] is a semi-implicit Particle-in-Cell (PIC) `C/C++` code developed primarily to study collisionless plasma dynamics at kinetic scales. Macro-particles, which are used to represent an ensemble of plasma particles, are evolved in a Lagrangian framework, whereas the moments (such as plasma density, current, pressure, etc...) and the self-consistent electric and magnetic fields are tracked on an Eulerian grid. The three main kernels of `iPIC3D` are Particle Mover, Moment Gatherer, and Field Solver. Due the implicit nature of the underlying algorithms, unlike explicit PIC methods, insufficiently resolved scales do not result in numerical instabilities. This allows us to choose time step sizes and spatial grid sizes 10–100 times greater than those used in traditional explicit PIC codes. `iPIC3D` is an `OpenMP` + `MPI` hybrid code, and two of its modules, the particle mover and moment gatherer, are offloaded to GPU using `HIP` (for AMD GPUs) and `CUDA` (for NVIDIA GPUs).

2.4. Leonardo HPC system and allocation of resources

The initial activities of the SPACE-CoE project was conducted on the EuroHPC Tier-0 supercomputer Leonardo, hosted and managed by CINECA. Leonardo consists of two main partitions [22]: the DCGP partition with 1,536 dual-socket nodes, each equipped with 112-core Intel Xeon 8480+ CPUs, 512 GB RAM, and 3.84 TB NVMe SSDs; and the Booster partition with 3,456 accelerated nodes, each with a 32-core Intel Xeon 8358 CPU and four customized NVIDIA A100 GPUs, totaling 13,824 GPUs. Each Booster node is equipped with 512 GB of memory and 64 GB of HBM2e memory per GPU. The GPUs are interconnected via NVLink 3.0 and connected to the host system through PCIe Gen4. Inter-node communication uses InfiniBand HDR with four 100 Gb/s ports per node. Leonardo Dragonfly+ network topology ensures high-bandwidth, low-latency connectivity through a two-level fat tree structure with spine and leaf switches, supporting scalable performance for large-scale simulations. Leonardo offers a pre-exascale computing capability of up to 240 Pflop/s.

Each SPACE-CoE code needs specific libraries, managed by the SPACK package manager [23]. These are organized into modules for different compilers and `MPI` versions. In addition to GNU compilers, DCGP uses Intel OneAPI compilers, while Booster uses NVIDIA HPC and `CUDA` compilers for GPU tasks. Computational resources on Leonardo are provided through allocations from ISCRA and EuroHPC, enabling jobs of up to 128 nodes on DCGP and up to 256 nodes on Booster. Additionally, CINECA has allowed larger runs during special *pre-exascale days*.

2.5. CPU performance assessment with `Extrae`

This section focuses on evaluating CPU code performance prior to any GPU adaptation. As the first step, we assessed the current status of the codes using the HPC performance analysis suite based on `Extrae` [24]. The profiling results, obtained in collaboration with IT4I, provide insights into code behavior and performance bottlenecks, forming the basis for optimization and porting to the Leonardo HPC system.

The performance analysis process is similar for most project codes. It begins with carefully selecting and setting up test cases for measurement. To get the most useful results of the analysis for the following optimization work, it is important to use a production-level test case that performs all the required computational routines ideally at the target scale in terms of computational resources, i.e. number of nodes, CPU cores, GPU accelerators, etc. At that scale, the typical simulation runs with the duration of many hours would generate an enormous amount of performance data. To decrease the size of the collected data to a manageable level, it is necessary to limit the simulations only to several units or tens of iterations, e.g. time steps, keeping the target scale.

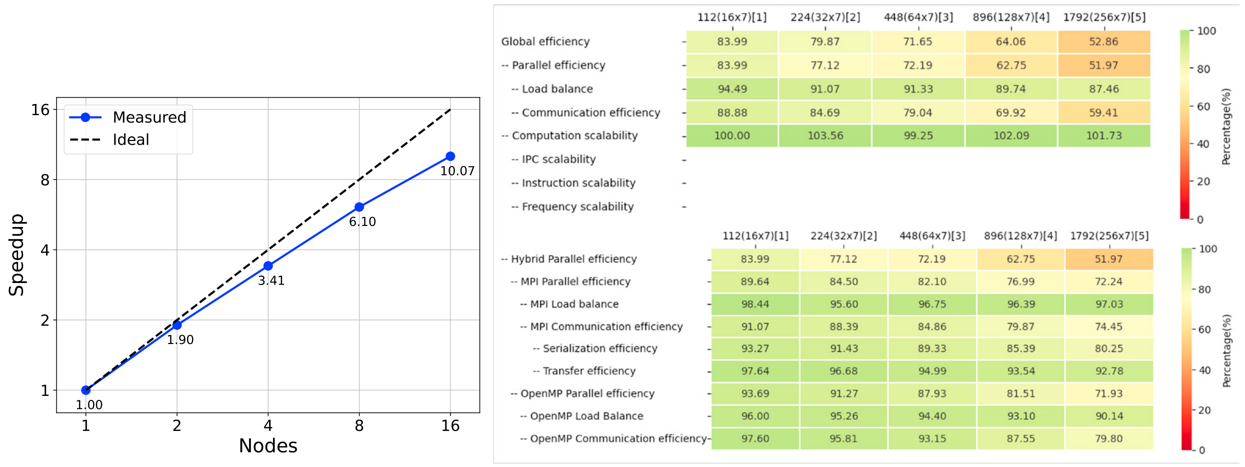
Among the three CoE codes only `OpenGadget3` and `iPIC3D` exhibited scalability issues with potential for improvement. In contrast, the CPU version of `gPLUTO` (`PLUTO`) had already undergone some optimization before the start of the SPACE project. As a result, no further scalability analysis or optimization was required for `PLUTO` at the beginning of the SPACE project.

Manual navigation and identification of relevant sections of complex simulation codes through the collected performance data tend to be very time-consuming work and prone to inaccuracies. Both issues might be addressed by instrumenting the selected regions of interest (RoI), meaning inserting simple non-intrusive probes of the particular tracing tool into the source code, typically at the beginning and end of the region.

Such an instrumented code and configured test case are used to perform the actual tracing. For `OpenGadget3` and `iPIC3D` analysis the following tool-chains were used on Leonardo DCGP: `GCC 12.2.0`, `OpenMPI 4.1.6`, `Extrae 4.0.6`, `HDF5 1.14`, `FFTW 3.3.10` (OG), `OpenBLAS 0.3.24` (OG), `GSL 2.7.1` (OG), `PETSc 3.21.1` (IP).

2.5.1. *OpenGadget3*

In the case of `OpenGadget3`, we simulated a box with a cosmological size of 30 Mpc/h and 256^3 particles were used. For this analysis, simulations were performed using from 1 to 16 nodes, where both gravity and hydrodynamics were enabled. The last 15 time steps were selected as RoI. Figure 1 left panel shows that the speedup of RoI gradually diverges from the optimum, with the efficiency below 80% on 8 nodes (896 `MPI` processes and 7 `OpenMP` threads per process). The hierarchical multiplicative model of efficiency metrics on Figure 1 right panel identifies Serialization efficiencies (SerE) and `OpenMP` Communication efficiencies (OCE)[1] as the main limiting factor of the RoI scaling. The detailed explanation of the model and the metrics can be found at POP3 CoE learning materials [15].



	112(16x7)[1]	224(32x7)[2]	448(64x7)[3]	896(128x7)[4]	1792(256x7)[5]
Global efficiency	83.99	79.87	71.65	64.06	52.86
-- Parallel efficiency	83.99	77.12	72.19	62.75	51.97
-- Load balance	94.49	91.07	91.33	89.74	87.46
-- Communication efficiency	88.88	84.69	79.04	69.92	59.41
-- Computation scalability	100.00	103.56	99.25	102.09	101.73
-- IPC scalability					
-- Instruction scalability					
-- Frequency scalability					

	112(16x7)[1]	224(32x7)[2]	448(64x7)[3]	896(128x7)[4]	1792(256x7)[5]
-- Hybrid Parallel efficiency	83.99	77.12	72.19	62.75	51.97
-- MPI Parallel efficiency	89.64	84.50	82.10	76.99	72.24
-- MPI Load balance	98.44	95.60	96.75	96.39	97.03
-- MPI Communication efficiency	91.07	88.39	84.86	79.87	74.45
-- Serialization efficiency	93.27	91.43	89.33	85.39	80.25
-- Transfer efficiency	97.64	96.68	94.99	93.54	92.78
-- OpenMP Parallel efficiency	93.69	91.27	87.93	81.51	71.93
-- OpenMP Load Balance	96.00	95.26	94.40	93.10	90.14
-- OpenMP Communication efficiency	97.60	95.81	93.15	87.55	79.80

Fig. 1: Extrae anlysis of `OpenGadget3` on Leonardo DCGP for up to 16 nodes: strong scaling (left panel) and efficiency metrics [%] (right panel). Note that the computation scalability sub-metrics were unavailable at this time due to incompatible tool suite components.

Each RoI time step consists of the following set of high-level routines that were instrumented and further analyzed: Domain decomposition intensity decision (DD) subdivided in the decision criteria computation (DD1), and the actual execution (DD2), which can either perform a full domain decomposition or only transfer particles that moved out of their domain; Gravitational accelerations (GRAV); Densities (DENS); Hydro-accelerations (HYDRO); Non-standard physics (PHYS).

The SerE describes any loss of efficiency due to dependencies between processes causing alternating processes to wait. The lowest SerE (80%) was observed in the DD2 routine; however, due to its relatively short execution time, this routine is not the primary contributor to performance degradation in this test case. The GRAV routine, although the most time-consuming, exhibits a high SerE of 92%, making it a low priority for SerE optimization. In contrast, the DENS routine, being the second longest and having a SerE of 86%, emerges as the most promising candidate for optimization. A simulation assuming an ideal network with infinite bandwidth and zero latency, where messages are transferred instantaneously, showed that each process spends approximately 12% of its runtime waiting in the `MPI_Alltoall` function for dependent processes.

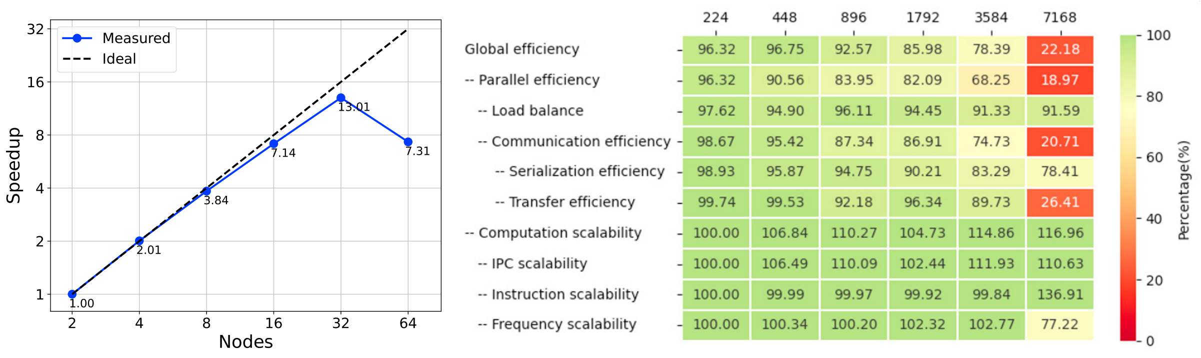
Regarding the OCE, which captures the synchronization and scheduling overhead induced by the `OpenMP constructs`, the lowest values 36% and 24% were observed in DD1 and DD2 respectively, though, being the shortest routines, they are probably not worth optimizing with the given test case. On the other hand, rather small room for improvement remains in the longest GRAV routine showing the best OCE 93%. The best candidate for optimization is again the DENS routine with OCE 80%, followed by HYDRO with 88% and much shorter PHYS with 67%. In

[1] See `https://co-design.pop-coe.eu/metrics/hm/omp_communication_efficiency.html`

DENS, each process spends a significant part of the total execution time in `OpenMP` runtime due to very small granularity of the parallel functions, mostly in `find_hsml` and `compute_unified_gradients` functions with 24% and 10% in average, respectively.

2.5.2. *iPIC3D*

We consider the test case of a Maxwellian distribution with 95^2 particles per cell with 2 species on a 2D grid with a resolution of 2240×1120 cells. The simulations were limited to four cycles and were conducted using between 2 and 64 nodes, with the second cycle selected as RoI. The strong scaling test in Figure 2 left panel reveals a clear speedup degradation on 64 nodes (7168 processes). The efficiency metrics in the right panel of Figure 2 point to drop in Transfer efficiency (TE) caused by an extreme growth of time in `MPI` communication and its latency.



	224	448	896	1792	3584	7168
Global efficiency	96.32	96.75	92.57	85.98	78.39	22.18
-- Parallel efficiency	96.32	90.56	83.95	82.09	68.25	18.97
-- Load balance	97.62	94.90	96.11	94.45	91.33	91.59
-- Communication efficiency	98.67	95.42	87.34	86.91	74.73	20.71
-- Serialization efficiency	98.93	95.87	94.75	90.21	83.29	78.41
-- Transfer efficiency	99.74	99.53	92.18	96.34	89.73	26.41
-- Computation scalability	100.00	106.84	110.27	104.73	114.86	116.96
-- IPC scalability	100.00	106.49	110.09	102.44	111.93	110.63
-- Instruction scalability	100.00	99.99	99.97	99.92	99.84	136.91
-- Frequency scalability	100.00	100.34	100.20	102.32	102.77	77.22

Fig. 2: Extrae anlysis of `iPIC3D` on Leonardo DCGP for up to 64 nodes: strong scaling (left panel) and efficiency metrics [%] (right panel).

Each cycle (RoI) is composed by the three high-level routines: CalculateField, ParticlesMover, and GatherMoments. The strongest TE deterioration at 64 nodes comes from CalculateField with 2%, followed by GatherMoments with 41%. That 2% TE together with 20% of Instruction scalability and other inefficiencies translates, in practice, into 0.2× speedup, i.e. 5× slowdown of the routine compared to the base run. This is caused by an excessive use of `MPI_Barriers` bounding sequences of very small `MPI_Sendrecv_replace` messages with no computation in between, resulting in significant accumulated latency of the `MPI` calls. Conversely, the ParticlesMover shows very good efficiencies and almost perfect scaling. In all routines, very high sensitivity for the system preemption is also detected.

3. Specific GPU offloading strategies

To leverage the compute capability of GPUs, all three codes adopt different numerical techniques, as each code is unique and uses distinct algorithms, each with its own set of challenges, performance bottlenecks, and numerical requirements. Additionally, these codes consist of numerous interlinked source files and are designed to simulate a wide range of physical mechanisms. As a result, optimizing or offloading to the GPU is not straightforward without focusing on a simplified, yet representative, core of the application. Therefore, we decided to extract mini-apps from the main applications, identify the relevant kernels, and analyze them to improve the overall performance of the full simulation.

Despite in the diversity numerical techniques, porting code to heterogeneous systems mainly relies on two principles: exposing parallelism and minimizing CPU-GPU data transfer. Data locality is key to GPU performance, as it minimizes memory transfers by running compute-heavy code on the GPU. Best practices include using private variables for thread independence and coalesced memory access for efficient bandwidth use. However, performance is still limited by numerical algorithms. For instance, `OpenGadget3` is memory-bound and needs new algorithms to fully utilize GPUs, while `gPLUTO` and `iPIC3D` have already offloaded their most compute-intensive tasks.

3.1. Profiling activity

We used NVIDIA Nsight tools and POP3 CoE resources to profile and analyze code performance, starting with CPU-only optimizations informed by POP3 analysis [24, 15]. Our strategy focused on minimizing CPU-GPU data transfers, optimizing kernel occupancy, and improving work distribution to reduce register spilling. We used profiling tools to identify performance-critical sections, analyzed memory access for coalesced patterns, and reduced data transfer overhead by overlapping communication with computation using asynchronous streams. Further, we optimized Streaming Multiprocessor (SM) occupancy and workload balance by investigating warp efficiency and control flow divergence, and reduced synchronization overhead by minimizing global atomics and barriers. This comprehensive approach guided iterative optimizations, enabling us to address bottlenecks and achieve high performance on the Leonardo Booster system [25].

3.2. Porting of gPLUTO

The core of the gPLUTO algorithm involves five main steps, which are typically repeated as many times as the number of stages employed by the Runge-Kutta solver. These steps include: boundary exchange/calculation, mapping of the conservative vectors to primitive vectors, reconstruction (i.e. interpolation) of the cells interfaces values, solving Riemann problem to calculate the fluxes at the interfaces, computing the right hand side of the conservation law. The divergence-free condition is controlled through the choice of one among different (mutually exclusive) algorithms (e.g. constrained transport, divergence cleaning, etc...). The only function that involves CPUs (or GPUs) intercommunication is the boundary exchange; this one, as well as all the other offloaded functions, was profiled and optimized for GPU parallelization: functions have been fine-tuned for coalesced memory access using a custom Array class (as shown in [12]), ensuring that independent threads access consecutive and unique memory addresses in both read and write operations. Managed memory is used to simplify memory management between the CPU (host) and GPU (device). gPLUTO also handles boundary conditions and ghost cells with non-blocking MPI communications across domains, using asynchronous data movement across multiple GPUs. This setup enables fast communication with nearby tasks (8 neighbors in 2D, 26 in 3D) and fills ghost zones between processes. Memory is optimized by inlining frequently used functions to reduce jumps and load/store costs, while also helping the compiler access memory efficiently in GPU shared memory. Memory paths for multi-dimensional arrays are also determined at compile time via C++ templated functions.

3.3. Porting of OpenGadget3

In OpenGadget3, the N-body gravitational problem is solved using the Barnes & Hut algorithm, which employs a hierarchical oct-tree structure to compute gravitational forces. While the benefit is a relatively low complexity of $O(n \log n)$ [17], the method is memory-bound and poorly suited for GPU offloading due to irregular tree traversal, uncoalesced memory accesses, and frequent data transfers triggered by tree reconstruction at each simulation step. The recursive implementation of the tree build also introduces thread divergence and performance bottlenecks on GPUs, despite the Peano-Hilbert decomposition improving spatial and memory locality. Further inefficiencies arise because only a subset of particles is active at each step, requiring reordering and additional condition checks that degrade memory coalescence and warp efficiency. Previous GPU-friendly adaptations [26] have not fully addressed these limitations or captured the complexity of OpenGadget3 cosmological simulations. To address this, a new approach is under development that groups particles by spatial and memory locality (e.g., using a common center of mass), limits computations to nearby particles within small Hilbert curve segments, and uses direct summation within a fixed radius (Tornatore et al., in prep.). This reduces conditional branching, minimizes data movement, and improves warp occupancy. Additionally, non-essential data structure members were removed to reduce register spilling and enhance GPU thread performance. A more detailed explanation of this method will be provided in a forthcoming publication.

3.4. Porting of iPIC3D

The scalability of iPIC3D depends significantly on the number of particles employed, which determines the runtime of the Particle Mover (updating position and velocity of the particles at each cycle). The Moment Gatherer module interpolates particle attributes to the grid (and vice versa) to compute charge and current densities and the pressure tensor. Both the Particle Mover and Moment Gatherer tend to be compute-bound for most input configurations. This makes them highly suitable for GPUs. The Field Solver, however, uses a generalized minimal residual algorithm (GMRes) to update the electromagnetic fields. GMRes is based on matrix-free Krylov subspace projections, which can easily saturate the memory on GPUs, thereby reducing the efficiency of the overall algorithm. GMRes also requires the computation of inner products of vectors, thereby necessitating multiple internode `MPI` communications. As this module is not as compute-bound as the other two, it runs exclusively on CPUs.

Following extensive optimization of the CPU-only version and the implementation of asynchronous non-blocking communication across the `MPI` processes, the Particle Mover and Moment Gatherer modules were offloaded to GPUs, while the field solver continues to run on CPU. This introduces an overhead of data communication between the GPU and CPU at every time step. However, this is an acceptable trade-off considering the speedups obtained in the computation of the GPU-offloaded modules.

4. Analysis and discussion of performance and scalability

Here, we describe our benchmarking efforts, tools, and methodology for evaluating the scalability and efficiency of these three parallel codes. A key principle in our approach is selecting problem sizes that fully utilize a single computational node, ensuring meaningful measurements of parallel performance. This strategy allows us to systematically assess how the full applications scale across multiple nodes while maintaining optimal resource utilization.

4.1. Automation of continuous integration and benchmarking tools

Continuous Integration and Continuous Deployment (CI/CD) are modern practices that automate the building, testing, and deployment of code. They streamline team collaboration, reduce human error, accelerate bug fixes, and enhance overall software quality. We teamed up with IT4I to set up CI/CD workflows for our three code-bases using their GitLab Runner system [27]. Regular benchmarking helps us spot performance issues and make sure our programs fully use modern hardware. It also catches slowdowns when we update code or add features, keeping performance steady over time [28]. To make benchmarking easier and work across different supercomputing systems, we use ReFrame [29], a Python tool for creating tests and benchmarks for HPC. ReFrame keeps system details separate from test designs, letting us write flexible tests that run smoothly on any system, speeding up both code checks and performance improvements.

4.2. Results for `gPLUTO`

Numerical benchmarks for `gPLUTO` have been carried out on both of Leonardo main computing partitions: DCGP and Booster. Two test cases were selected for this evaluation, the 3D Orszag-Tang vortex and the propagation of a 3D circularly polarized Alfvén wave. The scientific background and setup details for these tests are thoroughly discussed in [12]. The Orszag–Tang vortex involves complex shock interactions and the development of small-scale structures, making it ideal for assessing both numerical stability and performance under high-resolution conditions. On the other hand the circularly polarized Alfvén wave test provides an exact nonlinear solution to the ideal MHD equations and requires high accuracy and minimal numerical dissipation to preserve wave characteristics over long integration times, serving as a rigorous test of computational precision.

Weak scaling tests: The Orszag–Tang test simulations were conducted using a periodic 3D-Cartesian domain in double-precision arithmetic. The algorithm involved are the WENOZ reconstruction method, the HLLD Riemann solver, a third-order Runge-Kutta (RK3) time integration. To preserve the divergence-free condition of the magnetic field ($\nabla \cdot B = 0$) a constrained transport algorithm ([30], [31]) was used. A per-node resolution of $704 \times 704 \times 352$ was selected to optimize GPU memory usage, achieving a balance between high occupancy and minimal register spilling. Figure 3 left panel presents the parallel efficiency achieved on Leonardo Booster (up to 512 nodes) and Leonardo DCGP (up to 128 nodes) for this test, reaching approximately 88% and 90% efficiency, respectively. Table 1 compares absolute walltime values when running on both partitions the same weak scaling setup. Results indicate that, on average, executions on the DCGP partition are approximately an order of magnitude slower than those on the Booster partition, highlighting a significant performance gap between the two architectures.

Similar to the 3D Orszag-Tang test case, the circularly polarized Alfvén wave is a periodic 3D-Cartesian domain. The employed resolution is consistent with the previous Orszag-Tang test, and also in this case the computations are performed using double-precision arithmetic. Used algorithms are the same of the previous 3D Orszag-Tang test. As illustrated in the left panel of Figure 3, Leonardo Booster achieves an efficiency of 97% for 256 nodes in this case.

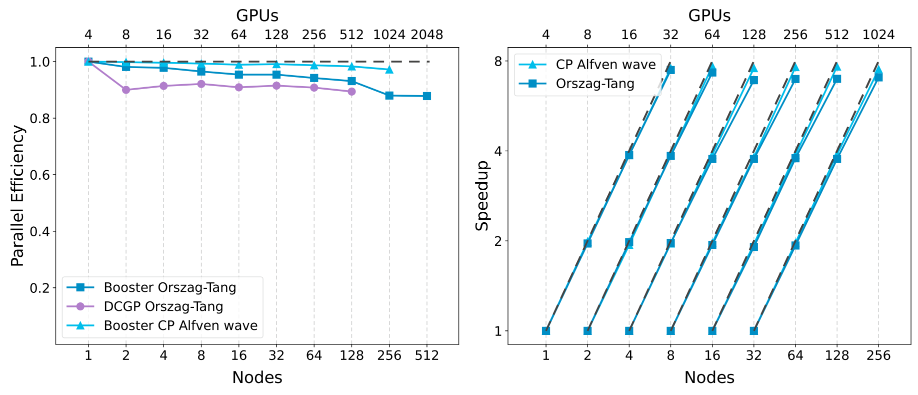


Fig. 3: Weak scaling tests for `gPLUTO` on both Leonardo Booster and DCGP partitions (left panel). Strong scaling tests for `gPLUTO` on Leonardo Booster (right panel). The black dashed lines indicate ideal scaling.

Nodes	T_{GPUs} (sec)	T_{CPUs} (sec)	Speedup
1	312	2982	9.55
2	318	3300	10.38
4	319	3263	10.23
8	323	3236	10.02
16	327	3281	10.03
32	327	3257	9.96
64	331	3283	9.92
128	335	3336	9.96

Table 1: CPU-GPU walltime comparison of `gPLUTO` 3D Orszag-Tang weak scaling test on Leonardo DCGP and Booster partitions.

Group	Nodes	Base Res	OT	CPA
1	1 to 8	$832^2 \times 416$	0.93	0.96
2	2 to 16	832^3	0.92	0.95
3	4 to 32	1664×832^2	0.86	0.95
4	8 to 64	$1664^2 \times 832$	0.87	0.96
5	16 to 128	1664^3	0.87	0.94
6	32 to 256	3328×1664^2	0.88	0.94
Mean	1 to 256	All res	0.90	0.95

Table 2: Strong scaling efficiencies for 3D Orszag-Tang (OT) and circularly polarized Alfvén (CPA) `gPLUTO` simulations performed on Leonardo Booster. Each group lists minimum obtained performance relative to its base resolution and node count.

Strong scaling tests: Strong scaling tests were performed with the same algorithm configurations of the weak scaling tests except for the technique to preserve the solenoidal condition of the magnetic field. This time the algorithm used was the divergence cleaning method ([32], [33]). This allowed for choosing a grid of $(832 \times 832 \times 416)$, slightly larger than that used in the weak scaling tests. This is because the divergence cleaning algorithm consumes less device memory than the constrained transport method. It is important to note that achieving ideal GPU strong scalability becomes increasingly challenging as the number of nodes grows. This difficulty arises due to overhead associated with inter-node communication, reduced computational workload per GPU, and increased synchronization costs, all of which can limit parallel efficiency at higher node counts. To address this, we conducted six distinct strong scaling campaigns for both Orszag-Tang and circularly polarized Alfén problems, as detailed in Table 2. Within each group, the computational resolution was held constant while the number of nodes and `MPI` processes was doubled three times. This approach ensures that the problem size is sufficiently large to fully utilize GPU memory at the higher number of nodes within each group, thereby maximizing hardware occupancy and minimizing inefficiencies. Strong scaling speedup values are presented in the right panel of Figure 3, demonstrating nearly 89% efficiency for the maximum number of nodes in each group for the Orszag-Tang problem, while an average efficiency of 95% is achieved for the circularly polarized Alfvén problem.

4.3. *Results for* `OpenGadget3`

Performance tests for `OpenGadget3` were run on the Leonardo Booster and DCGP systems. Although DCGP was mainly used to test near-full-physics setups and for comparisons with the GPU versions, the main results come from the Leonardo Booster system. Since GPU support is still in progress, current results reflect only limited physics setups. Simulations model the evolution of the Universe using particles representing dark matter and gas within a 3D cosmological box. These particles interact through gravity, and gas particles also interact with hydrodynamic forces using SPH. Gravity and hydrodynamics each account for $\approx$ 40% of the runtime in a standard production run. Consequently, this setup is a good representation of the GPU performance of the entire code. High-resolution simulations require many particles and large volumes, which increase the computational demands of the short-range Barnes & Hut part. Simulation performance generally depends on redshift (the cosmic time). As matter is more homogeneously distributed at high redshifts, the calculation of gravitational forces via the gravity tree are faster ($\sim 10\times$ *speedup*), compared to lower redshifts ($\sim 2\times$ *speedup* at $z = 0$), when dense structures form, and the calculation becomes more complex and consequently more costly. So, performance varies with redshift and must be evaluated accordingly, resulting in an effective speedup of $\sim 3\times$.

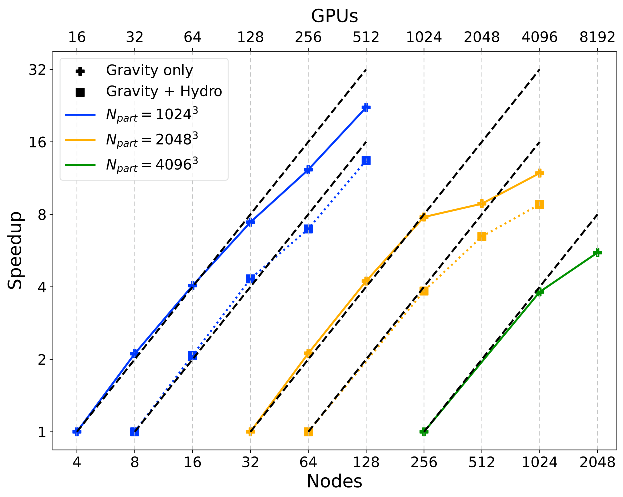


Fig. 4: Strong scaling tests for `OpenGadget3` gravity-only and gravity + hydrodynamics on Leonardo Booster. The black dashed lines indicate ideal scaling.

We ran strong scaling tests at high redshift ($z \sim 50$), using three cosmological boxes with 1024^3, 2048^3, 4096^3 particles and box sizes of 120, 240, and 480 Mpc/h, respectively. This ensured the problem size was large enough per process to avoid communication issues at high node counts, which could affect scaling results. While using the same code configuration, we run these tests using initial conditions with and without gas particles. We used 4 `MPI` tasks per node (1 per GPU) and 8 `OpenMP` threads per task.

As shown in Figure 4, the 1024^3 case exhibits a smooth and consistent scaling behaviour, maintaining over 80% efficiency up to a 32× resource increase. The 2048^3 case shows an efficiency drop beyond a 16× scaling factor in both gravity and hydro setups, with domain decomposition issues beyond 512 nodes. While only executed in the gravity-only setup due to limited computational resources, the 4096^3 case demonstrates speed-up of up to a 4× scaling. In addition, a full simulation to $z = 0$ with 2048^3 particles in gravity-only mode was performed to compare the run-times between DCGP and Booster. The right panel of Figure 4 shows speedup versus cosmic expansion factor a ($=1/(1+z)$). Most computational time occurs at $a \gtrsim 0.2$, where efficiency drops due to the Barnes & Hut algorithm struggling with deeper tree structures from increased clustering at lower redshifts. Smaller time steps are needed early on due to stronger gravitational accelerations, but performance gains are measured at larger a, yielding a 2-3 × speed-up. Despite being relatively low, this speedup is significant for simulations requiring 10^7 or more core-hours. Since short-range gravitational force calculations cause most performance issues, this supports our approach of optimizing the tree traversals, as discussed in Section 3.

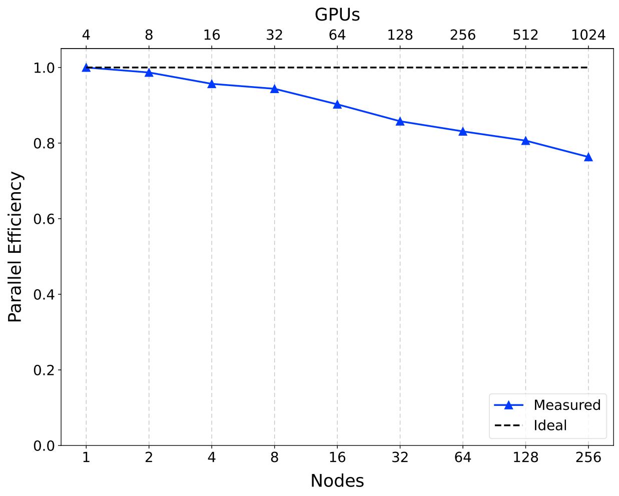


Fig. 5: Weak scaling test of `iPIC3D-GPU` on Leonardo Booster. The black dashed line indicates ideal scaling.

4.4. *Results for `iPIC3D`*

We show a performance comparison of the CPU and GPU versions of `iPIC3D` for the case of a Maxwellian distribution in 2D with $20 \times 20 \times 20$ particles per cell and 4 particle species on Leonardo DCGP and Booster, respectively (Table 3). We observe that the Moment Gatherer obtains a speedup of 100 whereas the Particle Mover achieves a speedup of a factor of 40, which determines the overall speedup of the code. As the Field Solver runs exclusively on CPUs, no speedup is achieved for this module. Furthermore, we illustrate the weak scaling of `iPIC3D-GPU` on

Module	`iPIC3D-GPU` (Leonardo Booster)	`iPIC3D-CPU` (Leonardo DCGP)	**Speedup**
Particle Mover	0.542 s	21.891 s	40.4
Moment Gatherer	0.123 s	12.271 s	99.8
Field Solver	0.185 s	0.183 s	0.98
Total	**0.870 s**	**35.007 s**	**40.2**

Table 3: Comparison of the CPU and GPU version of `iPIC3D` for a 2D Maxwellian distribution test case with $20 \times 20 \times 20$ particles per cell and 4 total species.

Leonardo Booster, the results of which are reported in Fig. 5. We consider the same Maxwellian distribution in 2D, where the number of grid cells was progressively increased from 128^2 to 2048^2. Each simulation modeled 4 particle species with 180×180 particles per cell per species. The code demonstrated an efficiency of 78% up to 1024 GPUs, corresponding to 256 nodes on the Leonardo Booster.

5. Summary

This paper presents the strategy and early achievements of the EuroHPC SPACE Center of Excellence (SPACE-CoE) in adapting three flagship astrophysical simulation codes i.e. `gPLUTO`, `OpenGadget3`, and `iPIC3D` , for exascale computing on the Leonardo supercomputer at CINECA. Through collaborative efforts, key modules were successfully offloaded to GPUs, enabling a transition from CPU to GPU architectures using profiling and optimization techniques. Preliminary results on the EuroHPC Leonardo system show notable scalability up to 1,024 GPUs with efficiencies around 80-97%. The study highlights `gPLUTO`'s near-ideal weak and strong scaling, `OpenGadget3`'s bottlenecks with the Barnes & Hut algorithm mitigated through restructuring, and `iPIC3D` 's substantial GPU acceleration, achieving up to 100× speedups in certain modules and 78% weak scaling efficiency by leveraging selective GPU offloading and asynchronous CPU-GPU communication. By combining performance profiling, code modularization, continuous integration, and GPU-specific kernel optimization, the project demonstrates how interdisciplinary collaboration can modernize complex simulation tools to fully exploit emerging exascale infrastructure.

Acknowledgements

This work was supported by the SPACE CoE, funded by the EU and several partner countries under grant No. 101093441. We also thank ISCRA and the EuroHPC Joint Undertaking for access to the computational resources on Leonardo supercomputer at CINECA (Italy). This work was partially funded by the POP3 project (grant No. 101143931), supported by the EuroHPC Joint Undertaking, its member countries, and additional funding from the Czech Ministry of Education (ID: MC2401) and e-INFRA CZ (ID: 90254). We also would like to acknowledge the NVIDIA experts for their invaluable assistance in porting the code to GPUs during and after dedicated hackathon events, especially Matt Bettencourt and Filippo Spiga.

References

[1] S. Sarkar, Is dark matter self-interacting?, Nat Astron 2 (2018) 856–857. doi:10.1038/s41550-018-0598-6.
[2] K. Schoeffler, et al, Can plasma physics establish a significant bound on long-range dark matter interactions?, Phys. Rev. D 111 (2025) L071701. doi:10.1103/PhysRevD.111.L071701.
[3] A. Romeo, et al, Simulations for 21 cm radiation lensing at eor redshifts, Mon. Not. R. Astron. Soc. 474 (2) (2017) 1787–1809. doi:10.1093/mnras/stx2733.
[4] R. Croft, et al, Weak lensing of the lyman α forest, Mon. Not. R. Astron. Soc. 477 (2) (2018) 1814–1821.
[5] S. Markidis, et al, Collisionless magnetic reconnection in a plasmoid chain, Nonlin. Processes Geophys. 19 (2012) 145–153. doi:10.5194/npg-19-145-2012.
[6] G. Mattia, et al, Resistive relativistic mhd simulations of astrophysical jets, Astronomy & Astrophysics 679 (2023) A49. doi:10.1051/0004-6361/202347126.
[7] S. Ferro, et al, Comparative simulations of kelvin–helmholtz induced magnetic reconnection at the earth's magnetospheric flanks, Phys. Plasmas 31 (5) (2024) 052902. doi:10.1063/5.0191674.
[8] S. Borgani, et al, X-ray properties of galaxy clusters and groups from a cosmological hydrodynamical simulation, Mon. Not. R. Astron. Soc. 348 (3) (2004) 1078–1096. doi:10.1111/j.1365-2966.2004.07431.x.
[9] N. Shukla, et al, Conditions for the onset of the current filamentation instability in the laboratory, J. Plasma Phys. 84 (3) (2018) 905840302. doi:10.1017/S0022377818000314.
[10] E. Boella, et al, Interaction between electrostatic collisionless shocks generates strong magnetic fields, New J. Phys. 24 (2022) 063016. doi:10.1088/1367-2630/ac6ef1.
[11] N. Shukla, et al, Eurohpc space coe: Redesigning scalable parallel astrophysical codes for exascale, in: 22nd ACM Int. Conf. Comput. Front, ACM, New York, NY, USA, Cagliari, Italy, 2025, p. 7. doi:10.1145/3706594.3728892.
[12] M. Rossaza, Gpu porting of the pluto code for computational plasma physics using openacc, Ph.D. thesis, University of Turin (2025). URL https://iris.unito.it/handle/2318/2065332
[13] F. Groth, et al, The cosmological simulation code OPENGADGET3 - implementation of meshless finite mass, Mon. Not. R. Astron. Soc. 526 (1) (2023) 616–644. arXiv:2301.03612, doi:10.1093/mnras/stad2717.
[14] S. Markidis, et al, Multi-scale simulations of plasma with iPIC3D, Math. Comput. Simul. 80 (7) (2010) 1509–1519. doi:10.1016/j.matcom.2009.08.038.
[15] POP3: Performance Optimization and Productivity Centre of Excellence, https://www.pop-coe.eu/, european Centre of Excellence on HPC performance optimization, project duration 2024–2027, coordinated by Barcelona Supercomputing Center (2024).
[16] V. Springel, The cosmological simulation code GADGET-2, Monthly Notices of the Royal Astronomical Society 364 (4) (2005) 1105–1134. arXiv:astro-ph/0505010, doi:10.1111/j.1365-2966.2005.09655.x.
[17] J. Barnes, P. Hut, A hierarchical O(N log N) force-calculation algorithm, Nature 324 (6096) (1986) 446–449. doi:10.1038/324446a0.
[18] M. Frigo, S. G. Johnson, The design and implementation of FFTW3, Proceedings of the IEEE 93 (2) (2005) 216–231, special issue on "Program Generation, Optimization, and Platform Adaptation".
[19] A. M. Beck, et al, An improved SPH scheme for cosmological simulations, Mon. Not. R. Astron. Soc. 455 (2) (2016) 2110–2130. arXiv:1502.07358, doi:10.1093/mnras/stv2443.
[20] A. Ragagnin, N. Tchipev, M. Bader, K. Dolag, N. J. Hammer, Exploiting the Space Filling Curve Ordering of Particles in the Neighbour Search of Gadget3, in: Advances in Parallel Computing, 2016, pp. 411–420. doi:10.3233/978-1-61499-621-7-411.
[21] A. Ragagnin, K. Dolag, M. Wagner, C. Gheller, C. Roffler, D. Goz, D. Hubber, A. Arth, Gadget3 on GPUs with OpenACC, Parallel Computing: Technology Trends 27 (2020) 209–218. doi:10.3233/APC200043.
[22] M. Turisini, et al, Leonardo: A pan-european pre-exascale supercomputer for hpc and ai applications, J. Large-Scale Res. Facilities 9 (01 2024). doi:10.17815/jlsrf-8-186.
[23] T. Gamblin, et al, The spack package manager: Bringing order to hpc software chaos, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC15), ACM, 2015, pp. 1–12. doi:10.1145/2807591.2807623.
[24] Barcelona Supercomputing Center, Extrae: The Extrae Tracing Tool, Barcelona Supercomputing Center, version 4.0, Available at https://www.ebrains.eu/tools/extrae (2025).
[25] L. Tornatore, et al., Code Modules and Kernels, SPACE-COE project document (2024).

[26] M. Burtscher, K. Pingali, Chapter 6 - an efficient cuda implementation of the tree-based barnes hut n-body algorithm, in: GPU Computing Gems Emerald Edition, Applications of GPU Computing Series, Morgan Kaufmann, Boston, 2011, pp. 75–92. doi:https://doi.org/10.1016/B978-0-12-384988-5.00006-1.

[27] B. Commercon, et al., Code release (alpha), SPACE-COE project document (2024).

[28] S. Williams, et al, Roofline: An insightful visual performance model for multicore architectures, Commun. ACM 52 (4) (2009) 65–76. doi:10.1145/1498765.1498785.

[29] V. Karakasis, et al, Enabling continuous testing of hpc systems using reframe, in: Tools and Techniques for High Performance Computing, Springer, Cham, 2020, pp. 49–68.

[30] C. Evans, et al, Simulation of Magnetohydrodynamic Flows: A Constrained Transport Model, Astrophys. J. 332 (1988) 659. doi:10.1086/166684.

[31] P. Londrillo, L. D. Zanna, On the divergence-free condition in godunov-type schemes for ideal magnetohydrodynamics: the upwind constrained transport method, J. Comput. Phys. 195 (1) (2004) 17–48. doi:10.1016/j.jcp.2003.09.016.

[32] A. Dedner, et al, Hyperbolic Divergence Cleaning for the MHD Equations, J. Comput. Phys. 175 (2002) 645–673. doi:10.1006/jcph.2001.6961.

[33] A. Mignone, P. Tzeferacos, A second-order unsplit Godunov scheme for cell-centered MHD: The CTU-GLM scheme, J. Comput. Phys. 229 (2010) 2117–2138. arXiv:0911.3410, doi:10.1016/j.jcp.2009.11.026.