



SCALABLE PARALLEL ASTROPHYSICAL CODES FOR EXASCALE

Progress report 1 on codes development

Deliverable number: D1.4

Version 1.0/1.0



Co-funded by
the European Union



EuroHPC
Joint Undertaking

Funded by the European Union. 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

Project Information

Project Acronym: SPACE
Project Full Title: Scalable Parallel Astrophysical Codes for Exascale
Call: Horizon-EuroHPC-JU-2021-COE-01
Grant Number: 101093441
Project URL: <https://space-coe.eu>

Document Information

Editor:	Geray Karademir - LMU
Deliverable nature:	Report (R)
Dissemination level:	Public (PU)
Contractual Delivery Date:	31.12.2024
Actual Delivery Date	31.12.2024
Number of pages:	38
Keywords:	modules and kernels, optimisation, scalability, GPU porting
Authors:	Fabio Bacchini – KU Leuven Benoît Commerçon – CNRS Tine Colman – CNRS Matthieu Kuhn – Eviden Pranab J Deka – KU Leuven Klaus Dolag – LMU Georgios Doulis – GUF Kristian Kadlubiak – IT4I Geray Karademir – LMU Andrea Mignone – UNITO Gino Perna – ES Khalil Pierre – GUF Marco Rossazza – UNITO Giuliano Taffoni – INAF Luca Tornatore – INAF Stefano Truzzi – UNITO Robert Wissing – UiO
Peer review:	Sijing Shen - UiO Stefano Truzzi - UNITO

History of Changes

Release	Date	Author, Organisation	Description of changes
0.1	22.10.2024	Eva Sciacca (INAF)	Started the document
0.5	09.12.2024	All authors	Version for review
0.6	12.12.2024	Geray Kardemir (LMU)	Introduction & conclusion
0.7	16.12.2024	Reviewers	Finished review
0.8	19.12.2024	All authors	Integration as requested by reviewer 1 & 2
0.9	20.12.2024	Geray Karademir (LMU)	Verification of changes
1.0	30.12.2024	Geray Karademir (LMU)	Final Version

Scalable Parallel Astrophysical Codes for Exascale

DISCLAIMER

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain. Neither the European Union nor the granting authority can be held responsible for them.



The space above and below the message intentionally is left blank.

Executive Summary

This deliverable reports on all optimization activities and their impact on code performance. Each code describes the actions that have been taken in optimising different kernels as well as their current status and obtained results. In short, the improvements for the codes vary from significant speed-up of the CPU implementation up to having their first fully functional GPU version available, but in all cases show large potential even for the kernels which haven't been finalised.

The second aspect of this deliverable is the report on the continuous integration that has been implemented by each code. Each code describes the measures which have been set in place to improve the code integrity. Since the type of algorithms as well as the scientific applications vary between the different codes, each code presents its own set of test cases suited for their individual situation.

This report also provides the status of porting each code to the EURO-HPC systems. In general, all codes have been ported on the majority of systems, and some have even been tested on all systems. Each code describes the current status of the porting in detail and in addition provides benchmarking results, where available.

In summary, this report provides a comprehensive overview of the current status of each code development and will show the potential of the improvements to be implemented soon.

Contents

1	Introduction	7
2	OpenGadget	8
2.1	Code status	8
2.2	CI/CD	8
2.3	Deployment on EuroHPC systems	10
2.4	Summary	10
3	PLUTO	12
3.1	Code Status	12
3.2	Kernel status	12
3.3	CI/CD	14
3.4	Deployment on EuroHPC systems	14
3.5	Summary and conclusion	14
4	BHAC	16
4.1	Code status	16
4.2	Kernel status	16
4.3	CI/CD	16
4.4	Deployment on EuroHPC systems	17
4.5	Summary and conclusion	19
5	ChaNGa	20
5.1	Code status	20
5.2	Kernel status	20
5.3	CI/CD	21
5.4	Deployment on EuroHPC systems	21
6	FIL/GRACE	24
6.1	Code status	24
6.2	Kernel status	24
6.3	CI/CD	25
6.4	Deployment on EuroHPC systems	25
6.5	Summary and conclusion	26
7	iPIC3D	27
7.1	Kernel status	29
7.2	CI/CD	29
7.3	Deployment on EuroHPC JU systems	30
8	RAMSES	32
8.1	Code status	32
8.2	Kernel status	32
8.3	CI/CD	32
8.4	Deployment on EuroHPC systems	33
9	Conclusions	36

List of Figures

1	Comparison of the required time per time step at different numbers of particles in each time bin for the old (black), the improved loop handling (blue) and the improved gradient calculation (red). It can be seen that the time required for the smallest time steps is reduced by a factor of ~ 5 in total.	9
2	Combined coverage report.	9
3	CI pipeline for the general case.	10
4	List of HPC systems with regular builds of the code. In addition to the systems by EuroHPC JU we also test the code on systems of the Leibniz-Rechenzentrum in Munich (LRZ) as their systems are used regularly by OpenGadget users.	11
5	Leonardo vs Marenostrum 5 weak scaling parallel efficiency for different square grids size on Leonardo Boost (left) and Marenostrum 5 ACC (right). Each line of the plots shown the weak parallel efficiency for one square grids.	14
6	Leonardo Boost vs Marenostrum 5 ACC scaling performances (cell/sec/nodes). Each line in the plots shows the time taken for one cell to advance one step of the algorithm as the number of nodes varies.	15
7	Pipeline duration of the last 30 commits in minutes.	17
8	CPU strong/weak scaling. Top to bottom: Lumi, Leonardo, and MareNostrum 5.	18
9	CPU scaling tests performed on several JU systems for a 25Mpc cosmological simulation using only gravity, with 2 billion particles. The speedup has been normalized so that at 32 nodes it is 1, with the relative speedup shown on the left and parallel efficiency on the right. We observe that parallel efficiency exceeds 80% at 65,536 CPUs/512 nodes.	22
10	Deployment and scaling of ChaNGa on the EuroHPC JU clusters. Green indicates complete, yellow in progress. "Ported" refers to the code being compiled and tested on a few sample cases. "Scaling" refers to the completion of scaling tests up to 64 nodes for CPU partitions and up to 128 GPUs for GPU partitions.	22
11	FIL strong and weak scaling plot on MareNostrum left and Leonardo right	24
12	Performance comparison between FIL and GRACE where the performance metric is cell updates per second. This metric ignores IO contributions to the runtime of both codes.	26
13	Top and middle rows: Strong and weak scaling of iPIC3D-CPU-SPACE-CoE on up to 16000 cores on two EuroHPC JU clusters. Ideal scaling is denoted with the red lines. Bottom panel: Strong and weak scaling of iPIC3D-GPU-SPACE-CoE on up to 64 GPUs on Karolina GPU.	28
14	Strong scaling on different EuroHPC JU clusters. Ideal scaling is denoted with the red lines.	31
15	Execution time of RAMSES for Use Case 1 on Marenostrum as a function of the internal <code>nvector</code> parameter. The best performance is obtained with <code>nvector</code> = 8. The test has been performed on a grid of 2048^3 using 8 nodes.	34
16	RAMSES strong (left) and weak scaling (right) on up to 65536 cores (on Lumi) on 7 out of the 8 EuroHPC JU clusters with Use Case 1 (Sedov blast). Ideal scaling is denoted with the black lines. The strong scaling has been performed using a uniform grid resolution of 2048^3 . For the weak scaling, we increased the number of cores keeping a uniform grid of 128^3 per core.	34

List of Tables

1	Access and deployment of OpenGadget on the CPU and GPU partitions of the EuroHPC systems.	10
2	Access, deployment and benchmarking of gPLUTO on the CPU and GPU partitions of the EuroHPC supercomputing clusters.	15
3	Access, deployment and benchmarking of BHAC on the CPU and GPU partitions of the 8 EuroHPC supercomputing clusters.	18
4	List of the modules used (explicitly loaded) on each of the clusters follows (CPU partition).	23
5	Access, deployment and benchmarking of FIL on the CPU partions of EuroHPC machines and GRACE on the GPU partitions of the EuroHPC machines.	26
6	Performance comparison of the old and new versions of iPIC3D. The new version shows over half an order of magnitude improvement in computational performance.	27
7	Performance comparison of the CPU and GPU versions of iPIC3D.	29

8	Configuration of CI/CD test problems for iPIC3D-CPU-SPACE-CoE.	29
9	Deployment and benchmarking of iPIC3D-CPU-SPACE-CoE on the EuroHPC JU clusters.	30
10	List of the modules used (explicitly loaded) on each of the clusters follows.	30
11	Status of RAMSES on the JU systems with CPU partitions.	33
12	List of the modules used (explicitly loaded) for RAMSES benchmarks on each of the JU cluster scalar partitions.	34
13	Strong scaling efficiency on the JU clusters with the maximum number of cores we achieved to use on the EuroHPC Development Access.	35
14	Deployment status of the codes on the different EuroHPC JU clusters.	36

List of Acronyms

AMR	Adaptive Mesh Refinement
BH	Black Hole
BHOSS	Black Hole Observations in Stationary Spacetimes
CD	Continuous Deployment
CI	Continuous Integration
CoE	Centre of Excellence
CRAL	Centre De Recherche Astrophysique de Lyon
DCGP	Data Centric General Partition
DD	Domain Decomposition
ET	Einstein Toolkit
EuroHPC JU	European High Performance Computing Joint Undertaking
GRMHD	General Relativistic Magnetohydrodynamics
GRHD	General Relativistic hydrodynamics
GRRT	General-Relativistic Ray-Tracing
HD	Hydrodynamics
HLL	Harten-Lax-van Leer
HLLC	Harten-Lax-van Leer contact
HLLD	Harten-Lax-van Leer discontinuity
HLRS	Höchstleistungsrechenzentrum Stuttgart
HPC	High Performance Computing
LB	Load Balancing
LRZ	Leibniz-Rechenzentrum
MHD	Magnetohydrodynamics
MHLLC	Maxwell-Harten-Lax-van Leer contact
MP	Monotonicity Preserving
MPI	Message Passing Interface
NCCL	NVIDIA Collective Communications Library
PIC	Particle-in-Cell
RHD	Relativistic Hydrodynamics
RMHD	Relativistic Magnetohydrodynamics
ResRMHD	Resistive Relativistic MHD
SPACE	Scalable Parallel Astrophysical Codes for Exascale
SPH	Smoothed Particle Hydrodynamics

SPMHD	Smoothed Particle Magnetohydrodynamics
SRHD	Special Relativistic Hydrodynamics
SRMHD	Special Relativistic MHD
TOV	Tolman–Oppenheimer–Volkoff
TVDLF	Total Variation Diminishing Lax-Friedrichs scheme
WENO	Weighted Essentially Non Oscillatory
MPI-AMRVAC	MPI-Adaptive Mesh Refinement-Versatile Advection Code
PDEs	partial differential equations
BHAC	Black Hole Accretion Code
FIL	Frankfurt Illinois GRMHD
BNS	binary neutron star
con2prim	conservative to primitive solver

1 Introduction

The objective of this report is to present an update on all the code optimisation activities for each code. While the implementation or improvement of the GPU capabilities for each code is essential for the hybrid architectures of modern supercomputers, the improvement of the CPU implementations is of high importance as well. The reasons are that the availability of GPUs is not always the case, especially for older systems, but also for newly procured systems. Secondly, as no code can run on GPUs only, CPUs still play an essential role in the total process. Consequently, to increase the exaflop capabilities of the codes it is also necessary to further develop the CPU implementations.

Continuous integration [CI](#) is an essential element in the code development process. It automates the workflow and streamlines the software development process. Its purpose is to ease the development cycle by introducing a method for early error detection, thus ensuring the code's integrity and improving its quality. In addition, a [CI](#) pipeline provides additional benefits such as continuous feedback loops as well as reduced downtime of the code as the quality of the code is checked regularly.

The main characteristic of a [CI](#) pipeline is the use of automation jobs to test the code quality. Thus these tests have to cover the building of the code in its various configurations, the successful execution of test cases and the validation of the test results. For most cases, this is achieved by running idealised test cases of which, ideally, analytic solutions exist so that the accuracy is guaranteed.

The deployment of all codes on all systems procured by the European High Performance Computing Joint Undertaking ([EuroHPC JU](#)) is an important milestone for each code as this guarantees the usability of the code for the scientific community. So far [EuroHPC JU](#) has procured eight supercomputers all over Europe, while the ninth is currently being constructed in Germany. This ninth machine, - JUPITER - is going to be the first exaflop machine in Europe and two modules have already been completed. Each of these supercomputers is built with a different hardware configuration. For the eight supercomputers which are in operation, there are in total 15 different partitions. This means that each code has to determine how it can be executed on each of these partitions. Especially the different GPU vendors increase the complexity of this task, as each vendor prefers their own programming framework due to which some implementations are not compatible with specific vendors. It is the aim to make all codes available on as many machines as possible as this increases the usability for users regardless of the computing facility they have access to.

The following chapters are organized as one per code, where the responsible code owners show their improvements on the different kernels, the status of their [CI](#) pipeline as well as the deployment status of the code to the diverse [EuroHPC JU](#) systems.

2 OpenGadget

OpenGadget is a code aimed at simulations of a cosmological environment. While the gravitational problem is solved numerically by coupling a Particle-Mesh algorithm for the average field and a tree-based Barnes & Hut algorithm for the interaction with the close neighbourhood, different methods for solving the hydrodynamical equations are used, such as Smoothed Particle Hydrodynamics (SPH) or Meshless finite mass (MFM).

2.1 Code status

This section describes general updates on code development. D2.3 provides more details about the individual sub-sections.

2.1.1 Data structures and vectorization

The code profiling activities in the first year showed that the vectorization ratio achieved by OpenGadget is, on average, small (see D2.1 and D1.2). For that purpose, we studied the achievable vectorization based on the direct (i.e. pairwise) Gravity force calculation using different approaches, such as Arrays of Structures (AoS) and Structures of Arrays (SoA) with and without explicit vectorization by using the vector intrinsics.

The results of this analysis showed that there is no clear best approach to be followed. The main takeaways from this study are that having neighbours stacked in a contiguous buffer as well as breaking the large structures into smaller pieces provides a general advantage.

2.1.2 GPU offloading

As described in D1.3, the Barnes & Hut oct-tree algorithm used in OpenGadget proved ineffective on modern hybrid architectures, as it leads to a large amount of thread divergence and locking. As described in more detail in D2.3, given the available resources by the device, it is underperforming. We consequently started a new implementation with a new strategy. This strategy is based on coalescing the walk for bunches of particles belonging to the same space region, synchronizing threads and avoiding memory divergence. Direct summation of all the particles within a given radius to avoid branching and conditionals, thus enhancing vectorization. This approach also allows for an updated tree-building since it doesn't require building the tree such that each leaf contains only one particle, instead the tree build can be stopped earlier saving additional computational time.

This new strategy has been implemented into an idealised stand-alone kernel that proved the potential of our approach. At the time of writing this report we already have implemented the new tree-building approach as well as an update on the tree-walking step of the algorithm and are currently at the stage of coupling to the gravitational force calculation and ensuring its correctness.

2.1.3 CPU development

The computation of the small time steps usually takes about half of the total simulation time. We have improved the code by refactoring loops over all particles to loops over gas or star particles only where possible. In addition, we changed the algorithm so that instead of allocating memory for the gradient matrices and then zeroing them for all particles at every step, we only create these matrices for currently active particles. These improvements reduce the time required for the small time steps by a factor of ~ 5 , which can be seen in Figure 1.

2.2 CI/CD

OpenGadget's CI pipeline consists of 26 tests to verify the code's integrity. In addition, several build tests are performed on all HPC systems procured by EuroHPC JU.

The total CI pipeline reaches an overall code coverage of $\approx 68\%$ (see Figure 2) and is always larger than 50% for each module. While most of the core parts of the code such as the hydrodynamics and time integration achieve high coverage, the modules related to extra physics (e.g. CosmicRays or CoolingSfr) and special modules (e.g. MFM) are not well tested. The low coverage on the gravity is due to the currently missing tests for the GPU implementation. We aim to use the information from the coverage report to extend the CI pipeline to achieve higher ratios for all modules.

The details of the different tests are described below.

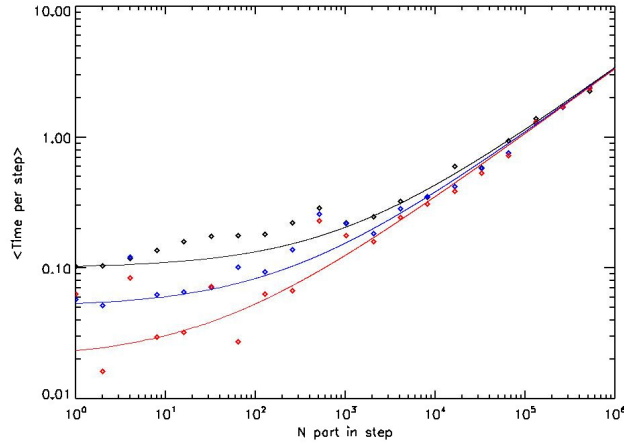


Figure 1: Comparison of the required time per time step at different numbers of particles in each time bin for the old (black), the improved loop handling (blue) and the improved gradient calculation (red). It can be seen that the time required for the smallest time steps is reduced by a factor of ~ 5 in total.

LCOV - code coverage report

Current view: top level

Test: combined_coverage_reduced.info

Test Date: 2024-11-26 10:06:59

Coverage

Lines: 67.4 %

Functions: 67.6 %

Total

29369

1538

Hit

19807

1039

Directory	Line Coverage ↕			Function Coverage ↕		
	Rate	Total	Hit	Rate	Total	Hit
OpenGadget3/	100.0 %	14	14	100.0 %	1	1
OpenGadget3/Blackholes/	59.0 %	974	575	90.3 %	31	28
OpenGadget3/CodeBase/	77.0 %	3973	3059	83.1 %	313	260
OpenGadget3/CoolingSfr/	71.4 %	899	642	80.0 %	60	48
OpenGadget3/CoolingSfr/Inflib/	57.5 %	388	223	60.9 %	23	14
OpenGadget3/CoolingSfr/Sfr_LT/	67.3 %	3580	2410	75.2 %	109	82
OpenGadget3/CosmicRays/	61.9 %	480	297	80.9 %	47	38
OpenGadget3/FofSubfind/	61.4 %	7021	4313	69.0 %	184	127
OpenGadget3/Gravity/	70.6 %	2996	2114	80.0 %	90	72
OpenGadget3/GreenTree_AR/	91.4 %	222	203	100.0 %	9	9
OpenGadget3/Hydro/	96.3 %	1968	1895	97.5 %	121	118
OpenGadget3/I0/	48.9 %	3222	1574	72.1 %	43	31
OpenGadget3/ImportFromL3_combined/	92.9 %	898	834	96.2 %	52	50
OpenGadget3/Integrator/	88.5 %	705	624	92.0 %	50	46
OpenGadget3/Mfm/	50.6 %	1387	702	20.5 %	342	70
OpenGadget3/System/	51.1 %	642	328	71.4 %	63	45

Generated by: LCOV version 2.1-60.gb7ebec0

Figure 2: Combined coverage report.

2.2.1 General tests

To verify the integrity of the code a pipeline with several tests is run daily and for every merge request. This consists of the following tests:

- Indent: checking for indentation and coding style of changed files. This is only done for merge requests.
- Config tests: in this test, several configurations are used to build the code.
- Gravity and Hydro tests:
Running a Sod shock tube problem, a Sedov blast wave problem and a sound wave test as well as a free-fall test. This is done for all three hydro solvers:
 - Smoothed Particle Hydrodynamics (SPH)
 - Particle-in-Cell and Embedded Source (PES)
 - Meshless Finite Mass (MFM)
- Conduction: testing energy transfer with different settings (e.g. with and without magnetic fields).

In total, this pipeline consists of 23 tests. An example where the organisation of the pipeline can be seen is displayed in Figure 3.

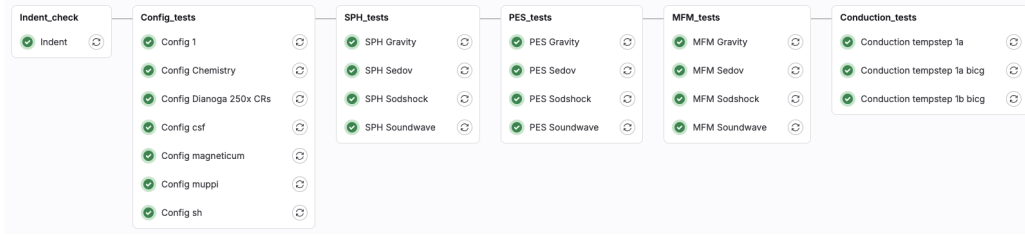


Figure 3: CI pipeline for the general case.

2.2.2 Extra tests

In addition to the general test described above, extra tests are run every week. Two of these tests consist of larger runs, namely the [MHD](#) test and the zoom-in simulation. The cosmic ray seeding test is a first test implemented to check the accuracy of the cosmic ray module of the code. In collaboration with the respective code owners, we aim to extend these physics tests in the future.

- Magnetohydrodynamics ([MHD](#)): testing the ability to accurately represent the shocks, rarefactions, contact discontinuities, and the compound structures of its [MHD](#) implementation the [MHD](#) test consists of a Brio-Wu shock tube.
- Zoom-in simulation: A cosmological zoom-in simulation of a galaxy cluster using full physics.
- Cosmic ray seeding: the accuracy of the cosmic ray seeding is tested by comparing the total CR energy from the snapshot files to the expected value from the star formation rate.

2.3 Deployment on EuroHPC systems

OpenGadget has been deployed on various [EuroHPC JU](#) systems and partitions, which are listed in Table 1. As a note we are currently verify the build of the original GPU implementation and not yet of the new one, since this one is not yet merged with the main. For more details on the new GPU implementation see D2.3.

Cluster	Deucalion	Discoverer	Karolina	Leonardo	Lumi	MareNostrum	MeluXina	Vega
Partitions	CPU,GPU	CPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU
Access	✓, ✓	✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓
Deploy	✓, ✓	✓	✓, ✓	✓, ✓	✓, x	✓, ✓	✓, ✓	✓, ✓
Benchmark	x, x	x	✓, x	✓, ✓	x, x	x, x	x, x	x, x

Table 1: Access and deployment of OpenGadget on the CPU and GPU partitions of the EuroHPC systems.

Since we use OpenACC as our model for GPU accelerators, we cannot use LUMI-G due to its AMD GPUs.

As part of the CI pipeline, the code is built on the following systems every week to guarantee the availability of the code for these systems. Figure 4 shows the list of partitions which are included in the automated system check. Since this list isn't complete we aim to extend this build test to all [EuroHPC JU](#) systems.

2.4 Summary

The code development is continuing as scheduled and significant improvements have been achieved. Regarding the improved GPU implementation, we are close to having a functional and tested mini-app version. The code has also been ported on all [EuroHPC JU](#) systems and corresponding systemtype variables have been added to the code make options. In addition, we have implemented a CI/CD pipeline which already provides descend total coverage which covers the main physical modules and checks the support of various [EuroHPC JU](#) systems.

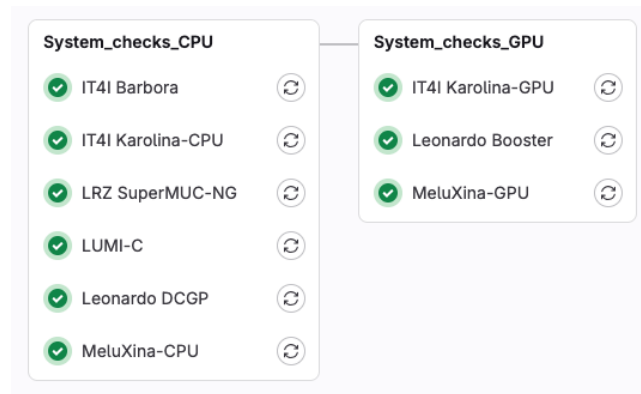


Figure 4: List of HPC systems with regular builds of the code. In addition to the systems by [EuroHPC JU](#) we also test the code on systems of the Leibniz-Rechenzentrum in Munich (LRZ) as their systems are used regularly by OpenGadget users.

3 PLUTO

PLUTO (<https://plutocode.ph.unito.it>) is designed to integrate a general system of conservation laws that we write as

$$\frac{\partial U}{\partial t} = -\nabla \cdot T(U) + S(U). \quad (1)$$

Here U represents a state vector of conserved quantities (typically density, momentum and energy), $T(U)$ is the flux tensor (rank 2) and $S(U)$ is a source term. The explicit definition of T , S and U depends on the particular physics module that the user selects. In its current implementation, the GPU-ported version (gPLUTO) comes with 5 different modules, Hydrodynamics ([HD](#)) (Euler equations of gasdynamics), Magnetohydrodynamics ([MHD](#)) (ideal and resistive) and their corresponding special relativistic extensions (see, e.g., [\[1\]](#), [\[2\]](#), [\[3\]](#)) plus the non-ideal Relativistic Magnetohydrodynamics ([RMHD](#)) equations. A comprehensive review of the numerical methods, algorithms and kernel identification has already been given elsewhere.

3.1 Code Status

Code activities on gPLUTO are continuing as originally planned. These actions are targeting mainly i) additions of more physics (such as geometry) and ii) performance and optimization improvements. At present, the code counts $\approx 65,000$ lines representing approximately 50% of the original code ported to GPU. A beta release of the code is now publicly available at <https://gitlab.com/PLUTO-code/gPLUTO> as well as from the PLUTO website (<https://plutocode.ph.unito.it/pluto-gpu.html>). In what follows, we described the various activities done on the kernels.

3.2 Kernel status

Previous achievements in the optimization process have included four physical modules (i.e., [HD](#), [MHD](#) and their relativistic extensions [RHD](#) and [RMHD](#)) and different kernels which include the conservative to primitive mapping functions (4 kernels), reconstruction of primitive or characteristic variables (3 kernels) and Riemann solvers (9 kernels). In the following subsections we briefly describe a second round of optimization. For a more detailed description of these optimizations, see DL2.3.

3.2.1 Geometry

The extension of the code to curvilinear geometries (i.e. cylindrical and spherical) is currently underway. At present, cylindrical coordinates have been implemented. This modification required introducing area and volume terms in the computation of the right hand side, as well as a number of source terms (e.g. centrifugal force or hoop stress). In addition, special care must be taken to ensure the conservation of angular momentum since curvilinear geometries are the preferred system of coordinates for disk / accretion modelling.

Differently from the legacy version of the code, here we will adopt a slightly different discretization approach that avoid splitting of the flux tensor into a transport and a pressure terms. The aim is, indeed, that of reducing code complexity in favour of a simpler (and equally robust) numerical scheme. Of course, extensive benchmarks are still necessary to assess the validity and accuracy of the scheme.

3.2.2 ResRMHD

The module for Resistive Relativistic MHD ([ResRMHD](#)) has been ported to GPUs. [ResRMHD](#) is a framework that takes into account the contribution of finite plasma conductivity. In [ResRMHD](#), the comoving electric field no longer vanishes as in ideal [MHD](#) and is directly proportional to the local current density, which is characterized by a scalar resistivity coefficient denoted η . The module involves an IMEX Runge-Kutta method. During the explicit stages the numerical fluxes are obtained by applying a 2D Maxwell solver at zone edges, while a standard 1D Riemann solver is used at zone interfaces to advance zone-centred hydrodynamic variables. When dealing with the stiff source term, a solution approach for the implicit update of staggered electric field at zone faces that retains the point-local character has been developed.

3.2.3 Asynchronous Boundary

The boundary routine permits data exchange between processors in multi-processors runs, to fill inter-process ghost zones, needed to reconstruct function values at zone interfaces. A major improvement has been undergoing during the last year, going from blocking (`MPI_Sendrecv`) to non-blocking (`MPI_Isend` and `MPI_Irecv`) MPI calls and from synchronous to asynchronous kernels. Aiming at having independent communication path for every direction of data exchange required the introduction of new paths to fill all the points needed, for a total of 26 direction of data exchange.

3.2.4 Functions Inlining

GPUs handle thousands of threads in parallel, and traditional function calls introduce overhead that can bottleneck performance. Inlined functions are expanded directly into the calling function's body, avoiding the cost of saving/restoring registers and jumping to/from another memory location. When a function is inlined, the compiler gains a complete view of the code, allowing it to perform aggressive optimizations.

3.2.5 Usage of C++ Templated Functions

C++ templated functions have been introduced in many kernels (such as Riemann solvers and reconstructions) and subroutines. Templated parameters helps for having indexes of multi-dimensional arrays known at compile time. In this way the compiler can better handle the memory path for those arrays, allocating them on registers when possible.

3.2.6 Debug Messages

In the GPU programming paradigm, print statements inside kernels are highly discouraged since these operations are typically not thread-safe. This results in multiple threads trying to print simultaneously inducing conflicts, eventually resulting in garbled or interleaved output and unpredictable behavior of the application. For these reasons, it is good practice to move all the outputs outside the kernel parallelization region to reduce the memory transfer from the host to the device. To this purpose, we store debug information in an array and analyze it after the parallel region.

3.2.7 OpenMP

gPLUTO is using OpenACC directives in the code to exploit the fast GPU parallelization speed. OpenACC is a very efficient solution to avoid the complexity of the low level CUDA software layer. Unfortunately only Fortran OpenACC software can be compiled and run on AMD GPUs architectures. Given that, gPLUTO is written in C++, so it cannot take advantage of such GPUs architectures.

Increasing portability by adding support for AMD GPUs is crucial for the Scalable Parallel Astrophysical Codes for Exascale ([SPACE](#)) project. For this reason, we decide to implement the OpenMP directives in the code. In collaboration with CINECA and Eviden we attended to the 6th Marenstrum Hackathon and started the implementation of the OpenMP directives. We managed to perform some preliminary simulation run and obtain the first kernels profiles using a 2D hydrodynamic test. The work is currently going on since the results differ by a factor of 6 with respect to the OpenACC implementation on LEONARDO cluster. Additional analysis is needed also to further investigate different compilers influence on this result by running the same simulation on LUMI cluster.

3.2.8 Riemann Solver

Riemann solver represents the most complex single kernel and computationally costly section of the code (D1.2), they can be of many types depending on the modules ([HD](#), [MHD](#), [RMHD](#) ... etc) and on the compromise between the algorithm complexity versus precision. In gPLUTO we implemented two newer Riemann solvers called two shock for HD ([\[4\]](#)) and RHD ([\[5\]](#)) modules. Also we introduced two new shock tubes test cases that use this solver in both [HD](#) and in [RHD](#) modules.

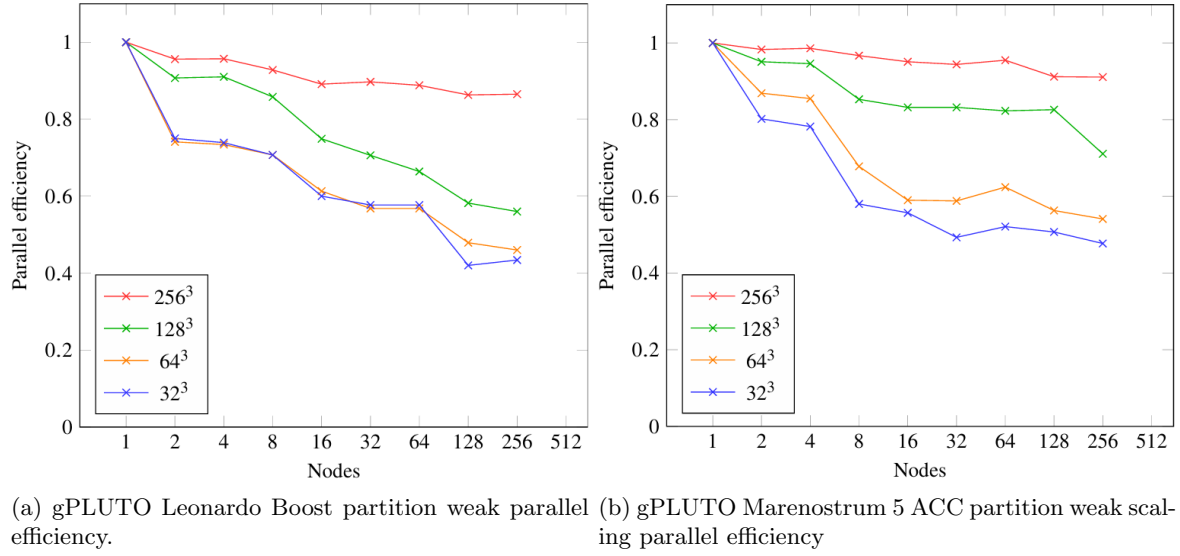


Figure 5: Leonardo vs Marenstrum 5 weak scaling parallel efficiency for different square grids size on Leonardo Boost (left) and Marenstrum 5 ACC (right). Each line of the plots shown the weak parallel efficiency for one square grids.

3.3 CI/CD

gPLUTO has two repositories: the public version <https://gitlab.com/PLUTO-code/gPLUTO> and the development version at <https://gitlab.com/PLUTO-code/gPLUTO-dev>. Actually, the public version is manually synchronized with the development version once or twice per year.

The development repository is where the team collaborates, utilizing Git's functionalities to synchronize their work effectively. Before each development action a developer launches an automatic Python script called `test_pluto.py` that runs more than 100 types of tests. Each automated test compiles all the test problems files and produces a log file that is compared with a reference log file. If the compilations does not produce errors and, if the comparison of the log files does not produce differences, the automatic script moves on to the next test and so on until the last one. To avoid machine differences, reference tests are generally performed on a dedicated machine equipped with: Intel(R) Xeon(R) Gold 5320 CPU @ 2.20GHz with two 20GB NVIDIA RTX A4500 and 250GB of RAM.

3.4 Deployment on EuroHPC systems

We achieved excellent results with EuroHPC: we compile and run gPLUTO on all of them CPU partitions and on all of the GPU partitions (tab. 2) with only one exception that regards LUMI GPU (AMD) partition. We also conduct extensive scaling tests, primarily focusing on the most powerful HPC systems equipped with NVIDIA GPUs. In figures: 5 and 6, we show the most recent GPU scaling results on Leonardo and Marenstrum, by varying the grid size and measuring the performance. We tested the Orszag-Tang problem using RK2, linear reconstruction, HLLD Riemann solver and CT divergence control.

Figs. 5 and 6 show the dependency on grid size of parallel efficiency and performance respectively. Especially from Fig. 5 it appears clear that the code does not behave well at the moment at low GPUs occupancy and in general it behaves better on Marenstrum than on Leonardo. While loss in performance at low resolution and low occupancy are expected due to the nature of GPUs these graphs show room for improvement. Other tests were present in previous deliverables.

3.5 Summary and conclusion

The activities on the code are continuing regularly as scheduled. Almost 50% of the original code has been ported to GPU by using the OpenACC programming model. The code has been deployed on all EuroHPC JU platforms and scaling performance measurements have been obtained on two systems (Leonardo and Marenstrum) showing good performance and scaling at high resolution but these decrease drastically at lower resolutions.

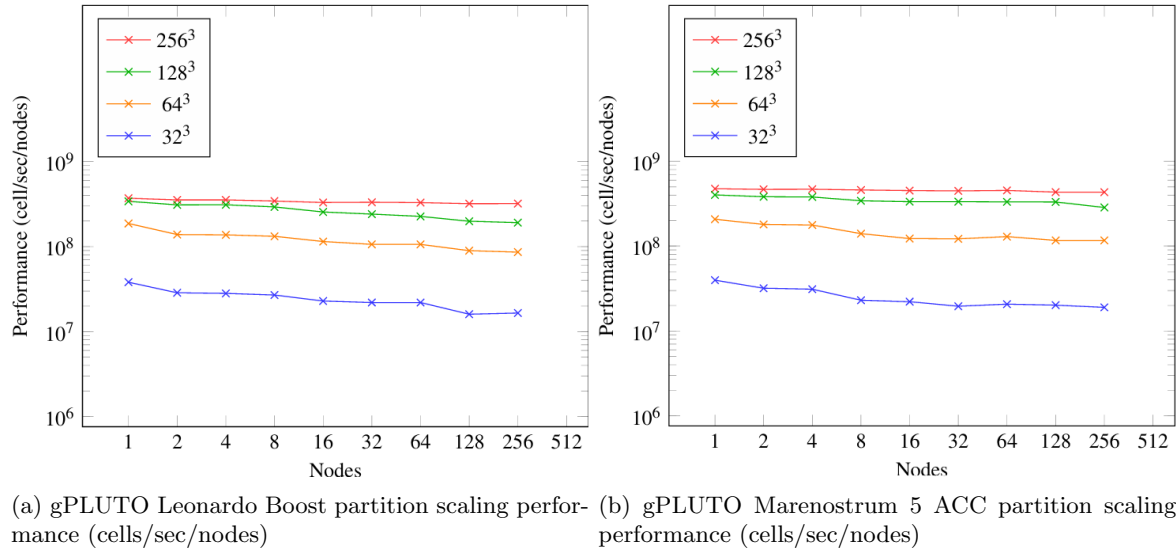


Figure 6: Leonardo Boost vs Marenostrum 5 ACC scaling performances (cell/sec/nodes). Each line in the plots shows the time taken for one cell to advance one step of the algorithm as the number of nodes varies.

Cluster	Deucalion	Discoverer	Karolina	Leonardo	Lumi	MareNostrum	MeluXina	Vega
Partitions	CPU,GPU	CPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU
Access	✓, x	✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓
Deploy	✓, x	✓	✓, ✓	✓, ✓	✓, x	✓, ✓	✓, ✓	✓, ✓
Benchmark	x, x	x	✓, ✓	✓, ✓	✓, x	✓, ✓	✓, ✓	✓, ✓

Table 2: Access, deployment and benchmarking of gPLUTO on the CPU and GPU partitions of the EuroHPC supercomputing clusters.

4 BHAC

The Black Hole Accretion Code (BHAC) [6, 7, 8, 9] is a multidimensional General Relativistic Magnetohydrodynamics General Relativistic Magnetohydrodynamics (GRMHD) code that solves the equations of ideal GRMHD in one, two or three dimensions in order to perform MHD simulations of accretion flows onto compact objects in arbitrary stationary space-times (Cowling approximation) using an efficient block based approach. BHAC is build upon the MPI-Adaptive Mesh Refinement-Versatile Advection Code (MPI-AMRVAC). MPI-AMRVAC [10, 11] is a parallel adaptive mesh refinement framework aimed at solving (primarily hyperbolic) partial differential equations (PDEs) by a number of different numerical schemes. The framework supports 1D to 3D simulations, in a number of different geometries (Cartesian, cylindrical, spherical). MPI-AMRVAC is written in Fortran 90 and uses MPI for parallelization.

4.1 Code status

On the CPU side BHAC underwent an extensive OpenMP porting and optimization campaign just before the start of the SPACE CoE project, see [12] for details. Thus, within the SPACE CoE project we have not devoted any efforts to further optimize BHAC as it is already highly optimized on a CPU level. The performance of the optimized version of BHAC improved by approximately 30% compared to its original version, making it viable for scaling on hundreds of thousands of supercomputer nodes. Concerning the GPU development, BHAC has been successfully ported to GPU using OpenACC and the Nvidia Nsight tools to track the performance of the porting. Among others we mostly focused at this stage on addressing issues of limited parallelism in kernels caused by compiler concerns about data races, optimizing these kernels through code transformation and OpenACC directive modification, reducing data transfer overhead by creating GPU-resident data. With these interventions we have achieved currently an approximate 90% performance improvement compared to the initially ported GPU version of BHAC.

4.2 Kernel status

Currently, two modules have been fully ported to a single and multi-GPU environment: the primitive reconstruction and Riemann solver. Of course they need further testing and optimisation in order to be able to function as efficiently as possible on a single as well as on multiple GPUs.

4.2.1 Conservative to primitive routine

The module responsible for the conservative to primitive inversion has already been ported to GPU using OpenACC. Only one of the three inversion strategies available in BHAC has currently been ported to GPU: the so-called *entropy inversion method*. This method is sufficient for most of BHAC's applications as it has been designed to deal with highly magnetised regions. Although it can also be used in regions of less extreme magnetisation, it is more efficient to use the other two less resource demanding methods 1D [13] and 2DW [14]. Thus, the porting of the entropy inversion method will be used as a blueprint for the GPU-porting of the other two inversion methods. The con2prim routine is based on a Newton-Raphson method, we have measured the average thread divergence due to the iterative nature of the Newton-Raphson method and have found that divergence is not hindering performance, so no further actions were needed here.

4.2.2 Riemann solver

The Rusanov flux, also known as the Total Variation Diminishing Lax-Friedrichs scheme (TVDLF), has been already ported to GPU using OpenACC. Currently, the ported TVDLF solver is being optimized in order to increase the parallelization efficiency and reduce—if possible eliminate—the host (CPU) to device (GPU) data transfer. With the TVDLF solver already ported we continue porting additional solvers available in BHAC's infrastructure like the Harten-Lax-van Leer (HLL), Harten-Lax-van Leer contact (HLLC), etc.

4.3 CI/CD

CI/CD, for BHAC, has been set up at the GitLab repository (https://code.it4i.cz/space_coe/bhac_mini_app) provided by IT4I and the corresponding runner is installed on IT4I's Karolina cluster. This is the repository where all development related to SPACE takes place. After each commit, a series of tests are performed, and

only when all these tests are successfully passed, is the commit accepted. Two applications that utilise all the basic routines of BHAC (e.g. computation of the fluxes, Riemann solvers, evolution, reconstruction, conservative to primitive recovery etc.) have been implemented in the CI/CD scheme. The first one comes from Special Relativistic MHD ([SRMHD](#)) and the second one from General Relativistic Magnetohydrodynamics ([GRMHD](#)):

1. The former is the 1D shocktube benchmark test, which is passed when
 - (a) BHAC is compiled and the executable specific to the shocktube application is produced.
 - (b) A stable and non-crashing simulation runs for 10 iterations.
2. The latter is the magnetised spherical accretion onto a Schwarzschild black hole, a static solution of [GRMHD](#), that is passed when
 - (a) BHAC is compiled and the executable specific to the magnetised spherical accretion application is produced.
 - (b) A stable and non-crashing simulation runs for 10 iterations.
 - (c) The L_2 -norm of the difference between the profile of the rest-mass density at the final time and at $t = 0$ (which is the exact solution) is less than a small number ϵ , where $\epsilon = 10^{-6}$.

Currently, a third test based on the main application of BHAC: the 3D magnetised accretion around a rotating black hole, is in the process of implementation in BHAC's CI/CD pipeline. This test will be passed when

1. BHAC is compiled and the executable specific to the 3D magnetised accretion around a rotating black hole is produced.
2. A stable and noncrashing simulation runs for several iterations.
3. The profile of the rest-mass density at $t = 1000$ replicates the corresponding results of BHAC's "golden file" produced with the original version of BHAC before developments within the SPACE CoE project begin.

The duration of the above CI set of tests is on average 2 minutes, see [Figure 7](#), which fits well with a quite dense development workflow.

Pipeline durations for the last 30 commits

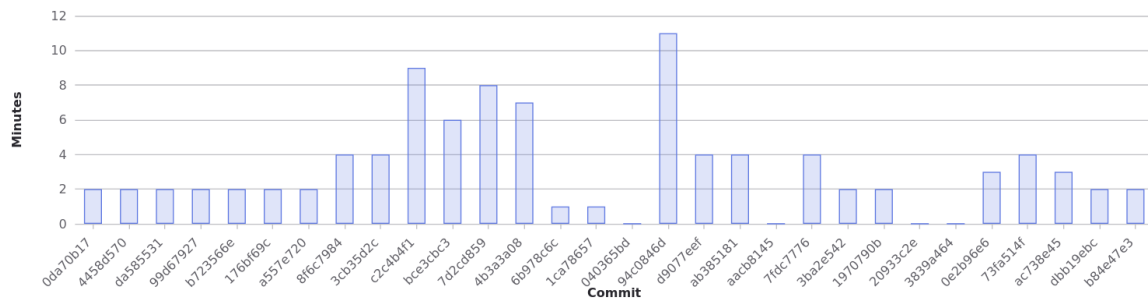


Figure 7: Pipeline duration of the last 30 commits in minutes.

4.4 Deployment on EuroHPC systems

We have applied and got access to all 8 EuroHPC supercomputing clusters through EuroHPC benchmark/development calls. Specifically, we have access to the CPU partitions of all 8 EuroHPC machines and to the GPU partitions of Karolina, Leonardo, Lumi. We are currently applying for access to the remaining clusters with GPU partitions, namely Deucalion, MareNostrum 5 and Vega. We have also access to the supplementary systems of IT4I and the CPU and GPU partitions of E4.

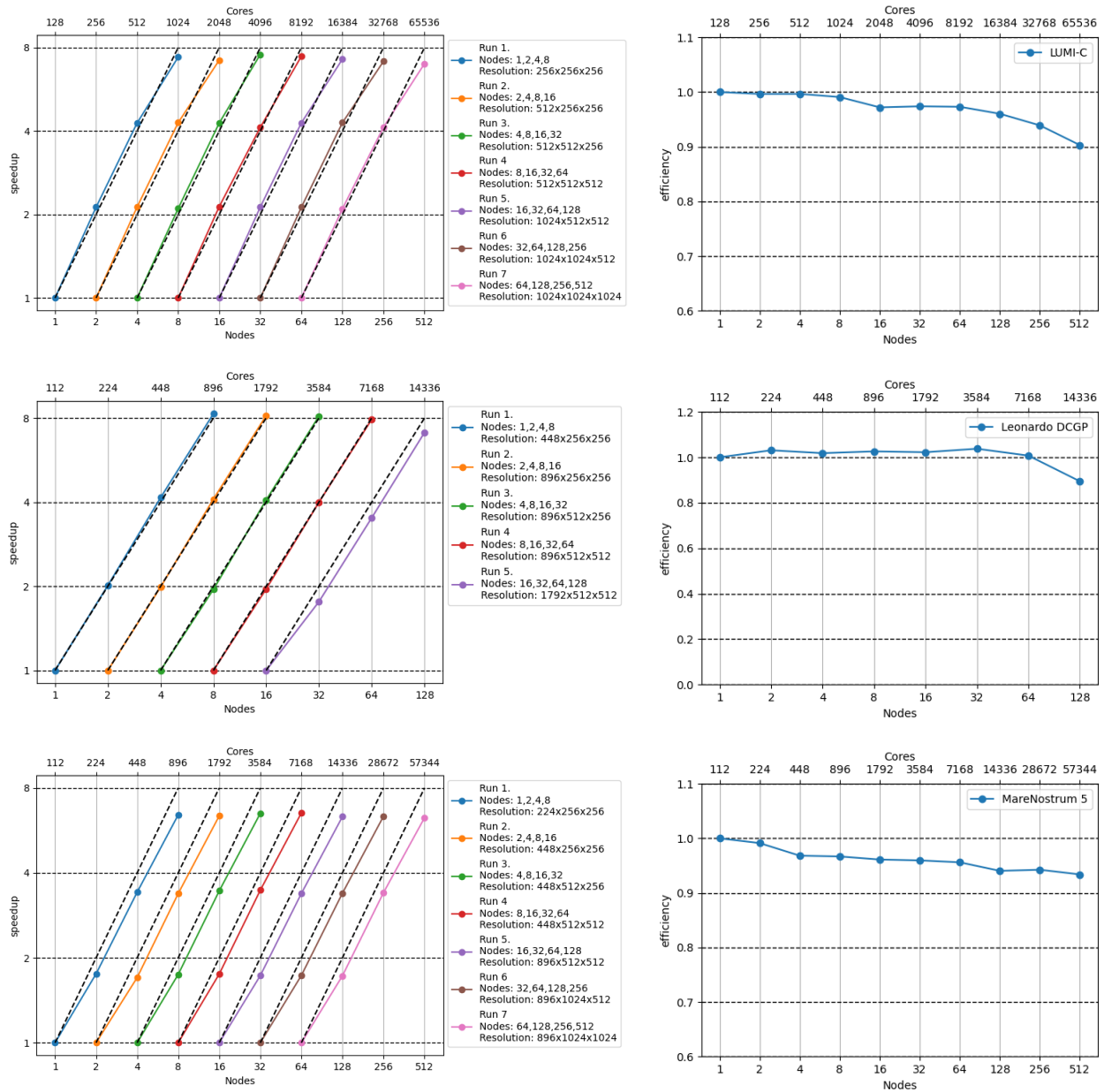


Figure 8: CPU strong/weak scaling. Top to bottom: Lumi, Leonardo, and MareNostrum 5.

Cluster	Deucalion	Discoverer	Karolina	Leonardo	Lumi	MareNostrum	MeluXina	Vega
Partitions	CPU,GPU	CPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU
Access	✓, x	✓	✓, ✓	✓, ✓	✓, ✓	✓, x	✓, x	✓, x
Deploy	✓, x	✓	✓, ✓	✓, ✓	✓, x	✓, x	✓, x	✓, x
Benchmark	x, x	x	✓, ✓	✓, x	✓, x	✓, x	✓, x	✓, x

Table 3: Access, deployment and benchmarking of BHAC on the CPU and GPU partitions of the 8 EuroHPC supercomputing clusters.

BHAC has been ported and compiled successfully on all EuroHPC CPU partitions. The main application of BHAC, i.e. the 3D magnetised accretion around a Kerr black hole, has been deployed also on all the machines and run for at least ~ 1000 iterations to test its performance and accuracy. The same has been done on the (Nvidia) GPU partitions of Karolina and Leonardo. The AMD GPU's of Lumi have delayed the deployment of BHAC on its GPU partition. We are currently investigating how can we combine the GPU version of BHAC

with the available software stacks on Lumi to make its deployment possible. BHAC has been also successfully deployed on the available CPU and GPU partitions of the supplementary systems of IT4I and E4.

BHAC has been benchmarked by producing strong/weak scaling plots up to ~ 65000 cores on the CPU partitions of a number of EuroHPC machines: Lumi, Leonardo, MareNostrum 5, and Karolina. [Figure 8](#) depicts a sample of the strong/weak scaling plots on a number of EuroHPC machines. The recent multi-GPU ported version of BHAC has been initially benchmarked on Karolina with quite encouraging results.

4.5 Summary and conclusion

BHAC has been ported to GPU using OpenACC. Recently, BHAC was enabled to run on multiple GPUs with very encouraging first scaling tests. CI has already been set up here https://code.it4i.cz/space_coe/bhac_mini_app with a variate of tests implemented, which have to be passed before a commit is accepted. BHAC has been deployed on the CPU partitions of all 8 EuroHPC supercomputers and on the GPU-partitions of Karolina and Leonardo—currently we are in the process of deploying BHAC on the rest EuroHPC machines.

5 ChaNGa

ChaNGa [15][16][17] is an N-body and Smoothed Particle Magnetohydrodynamics (SPMHD) code which is used to study a wide array of astrophysical systems. While the gravity and SPMHD algorithms are based on the gasoline [18] and pkdgrav [19] codes, the unique feature of ChaNGa is its implementation of the Charm++ framework, which enables highly efficient parallel scaling. Charm++ employs overdecomposition to achieve this. That is to divide the work into many more pieces (chares/tree pieces) than you have processors and let the Charm++ runtime system load balance by appropriately assigning pieces to real processors. During runtime, Charm++ applies dynamic load rebalancing strategies, to determine which tree pieces should be migrated to new processors for better load balance. Charm++ also provides support to execute CUDA kernels on the GPU asynchronously and to manage data transfers between the CPU and GPU. A high-level description of the code, its main algorithms and kernels targeted for optimization can be found in §8 of Deliverable 1.2.

5.1 Code status

The main focus of ChaNGa in terms of optimization and parallelization has in the past been the gravity module. This has enabled the current version of ChaNGa to scale to a very high number of compute nodes in highly clustered gravity-only simulations, where a 25 Mpc cosmological simulation with 2 billion particles was run up to 131072 cores on the Blue Waters supercomputer cluster <https://bluewaters.ncsa.illinois.edu/documentation>. This has been repeated on similar number of cores on JU clusters (see Section 5.4). Other similar scaling has been done for GPU, where a 25 Mpc cosmological simulation ¹ with 24 billion particles was run up to ≈ 3000 nodes (1 gpu per node) on Piz Daint supercomputer cluster [https://en.wikipedia.org/wiki/Piz_Daint_\(supercomputer\)](https://en.wikipedia.org/wiki/Piz_Daint_(supercomputer)).

In Deliverable 2.1 we have shown that ChaNGa suffers from some communication imbalances when running highly clustered full-physics simulations. That is why fixing the communication imbalance is one of the main optimization objectives that we aim to improve during the course of the SPACE project. We have solved this by implementing a tree piece replication technique (see more code details in D2.3), which distributes the tree pieces on several processors, avoiding one processor being overwhelmed with messages.

The second big objective within the SPACE project is to improve the GPU module of ChaNGa and offload computationally heavy modules to the GPU. We have already done a preliminary offload of the radiative cooling to the GPU and after that is done we will begin to offload the SPMHD routines.

5.2 Kernel status

In D1.2, three kernels were defined, Gravity remote communication, SPMHD remote communications and SPMHD GPU offloading. From that time we have decided to instead GPU offload the radiative cooling first to GPU instead of SPMHD. As such we have added the GPU offloading of Radiative cooling as a kernel. More details on the kernels can be found in D1.2. The following is the current implementation status of the kernels.

5.2.1 Kernel 1,2 - Gravity+SPMHD - Improving communication imbalance

We have tackled the communication imbalance seen in the performance analysis (D2.1) by introducing a tree piece replication method within the code. In this method, we replicate the information about the tree nodes on multiple processors, which spreads out the communication load and ensures that no single processor becomes overloaded with messages. A rough version of tree replication was implemented in an older version of ChaNGa, which we have now merged and updated to the newest version of ChaNGa. This has been shown to significantly reduce the communication imbalance seen in the initial performance analysis. We will continue to test the tree piece replication method and explore other ways to reduce the communication volume (cache improvements, node-wide cache, etc.). More technical details and scaling improvements of this optimization can be found in D2.3.

5.2.2 Kernel 3 - Radiative cooling - GPU offloading

Previously, we planned to prioritize porting the SPMHD module. However, we have decided to begin with the radiative cooling module instead. While the SPMHD module is often the second most computationally

¹A higher resolution version of the cosmological simulation presented in [?].

demanding, the radiative cooling module can surpass it in specific scenarios. This is because radiative cooling module involves a semi-implicit integration method, which requires significantly larger computational effort when the cooling timestep is much smaller than the dynamical timestep. The radiative cooling module has now been preliminarily ported to the GPU, demonstrating a speedup of around 4x-20x depending on the test case. More technical details and improvements of the GPU offloading can be found in D2.3.

5.2.3 Kernel 4 - SPMHD - GPU offloading

For the porting of the [SPMHD](#) module, we are currently discussing strategies of implementation. We will likely either perform the tree walk on CPU and produce an interaction list (as has been done for other SPH codes, such as SPH-EXA), which is sent to the GPU for calculation. Or we will attempt to perform both the local tree-walk (neighbour finding) and the calculation on the GPU (similar to what is done for our gravity module).

5.3 CI/CD

ChaNGa is publicly available on github (<https://github.com/N-BodyShop/changa>) and has been for some time. Commits to the public git are done through pull requests, which allow developers to review and discuss changes before being merged into the main project. This means that a large amount of work can be done on separate branches and then merged into the head/public branch of ChaNGa (while still being publicly available for review during the open pull request). [SPMHD](#) which has previously been only accessible through the private branch is currently under a pull request to be reviewed and pushed into the public repository.

The public repository of ChaNGa already includes a basic Continuous Integration (CI) pipeline, that needs to be passed for changes to be committed. This pipeline is configured to compile Charm++ and ChaNGa, followed by executing a simple test to verify the time integration and gravity functionality. This can be seen in <https://github.com/N-BodyShop/changa/actions>. However, this setup does not cover many of ChaNGa's other modules, such as hydrodynamics, star formation, feedback, magnetic fields, etc. To address this, we have prepared a comprehensive suite of test cases to ensure that all physics modules work properly within ChaNGa. This test suite is also beneficial for users of ChaNGa's predecessor, Gasoline, and for the development of numerical schemes within both codes. Consequently, it is available on its own public Git repository (<https://github.com/robertwissing/testsuite>). We will continue to create additional test cases for this test suite to further improve our CI and enable users to easily run a wide range of tests. The extended CI/CD pipeline will consist of several stages and test cases. These tests were chosen to ensure the integrity of all regular code modules:

- Compilation of Charm++ and ChaNGa, with different build configurations.
- Time integration and gravity solver
- Hydrodynamics tests
- Magneto-hydrodynamics tests
- Cosmological tests
- Star formation, cooling and feedback test

The initial conditions to each of these tests will be included with ChaNGa either in raw form or in setup scripts. The log files of each test case (total energy, angular momentum, magnetic field strength, etc.) is checked so that they reproduce the same result as previous versions of the code. Our CI/CD pipeline is configured to run automatically on merge requests and when new commits are pushed. We also aim to implement a similar EuroHPC system check as OpenGadget (see figure 4), to verify that the current build works on all EuroHPC systems.

5.4 Deployment on EuroHPC systems

ChaNGa has been deployed and tested across all JU clusters, for both CPU and GPU (see figure 10) partitions. Additionally, we have run cosmological box simulations (gravity only) with 2 billion particles to evaluate CPU scaling across multiple JU clusters (see figure 9), demonstrating excellent scalability with over 80% efficiency even at 512 nodes. As previously mentioned, ChaNGa has been compiled and successfully run on all GPU

partitions, with scaling tests on Karolina conducted up to 128 GPUs, achieving approximately 90% efficiency. However, due to a current bug causing a race condition when using multi-GPU configurations, we have decided to postpone further large-scale GPU tests and scaling on other GPU partitions until this issue is resolved.

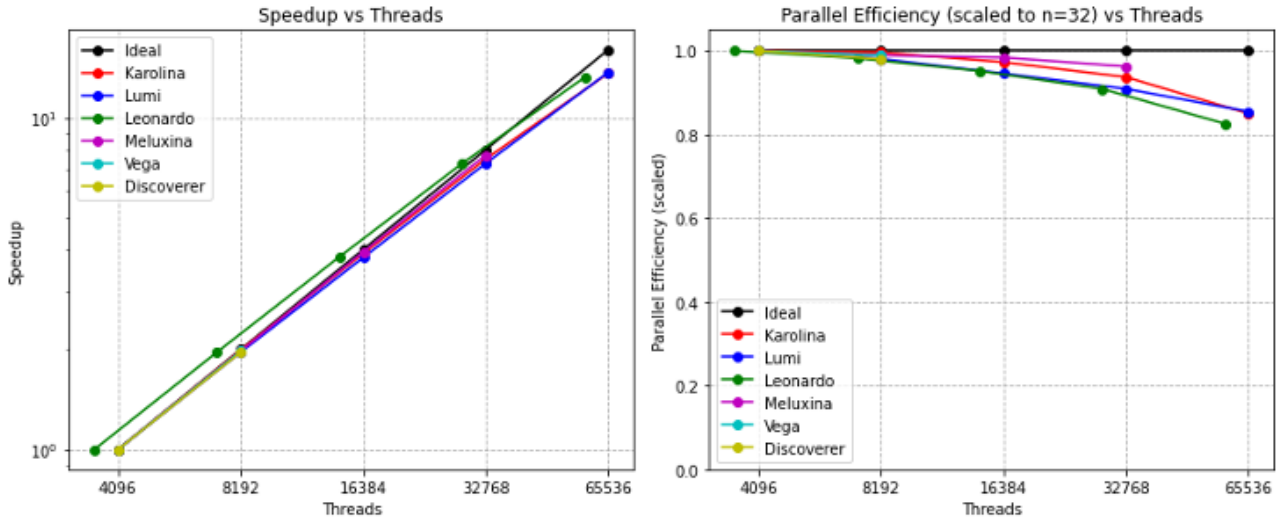


Figure 9: CPU scaling tests performed on several JU systems for a 25Mpc cosmological simulation using only gravity, with 2 billion particles. The speedup has been normalized so that at 32 nodes it is 1, with the relative speedup shown on the left and parallel efficiency on the right. We observe that parallel efficiency exceeds 80% at 65,536 CPUs/512 nodes.

	LUMI		LEONARDO		MNOSTR V		MELUXINA		KAROLINA		VEGA		DEUCALION		D.VERER
access	✓		✓		✓		✓		✓		✓		✓		✓
	cpu	gpu	cpu	gpu	cpu	gpu	cpu	gpu	cpu	gpu	cpu	gpu	cpu	gpu	cpu
ported	✓	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
scaling	✓	-	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Figure 10: Deployment and scaling of ChaNGa on the EuroHPC JU clusters. Green indicates complete, yellow in progress. "Ported" refers to the code being compiled and tested on a few sample cases. "Scaling" refers to the completion of scaling tests up to 64 nodes for CPU partitions and up to 128 GPUs for GPU partitions.

Cluster	Modules
Deucalion	OpenMPI/4.1.5-GCC-12.2.0
Discoverer	openmpi/4/gcc/4.1.6
Karolina	OpenMPI/4.1.4-GCC-12.2.0
Leonardo	intel-oneapi-compilers/2023.2.1, intel-oneapi-mpi/2021.10.0
Lumi	LUMI/23.09, PrgEnv-cray-amd, craype-hugepages2M, gcc-mixed/12.2.0
MareNostrum	intel/2023.1
MeluXina	OpenMPI/4.1.5-GCC-12.3.0
Vega	OpenMPI/4.1.4-GCC-12.2.0

Table 4: List of the modules used (explicitly loaded) on each of the clusters follows (CPU partition).

6 FIL/GRACE

The Frankfurt Illinois GRMHD code FIL has been the flagship code of the Relativistic astrophysics group at Frankfurt since its development in 2018 [20]. FIL is written in C++ and uses OpenMP and MPI. The code was designed to simulate binary neutron star (BNS) collisions and uniquely provides a fourth-order accurate conservative finite-difference scheme, making it a state-of-the-art within the numerical relativity community. FIL leverages the computational infrastructure provided by the Einstein Toolkit (ET)[21], an open source framework designed to facilitate numerical relativity simulations. The ET provides the numerical grid, the AMR and parallelization that FIL uses. CPU scaling tests as seen in figure 11 shows that FIL has excellent scaling up to 512 nodes.

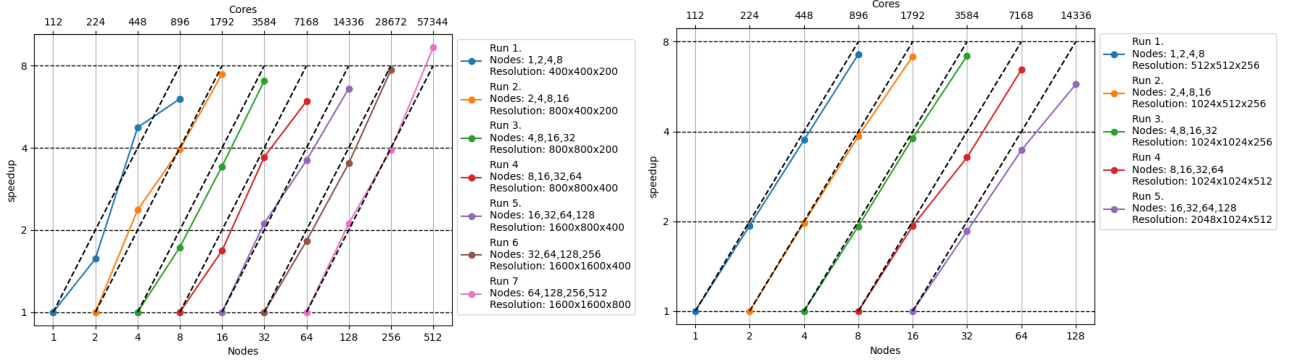


Figure 11: FIL strong and weak scaling plot on MareNostrum left and Leonardo right

The CPU performance of FIL has been highly optimized by the ET team over the previous two decades. For performance gains the focus has been on porting the physics of FIL to GPU. Currently the ET team in North America is developing a newer version of the Einstein toolkit that is GPU compatible, the current version is not, having been written in the early 2000s. Unfortunately, the development of the new GPU ET has experienced delays and the current timeline is incompatible with the timeline of the SPACE project. It was therefore decided to create GRACE, an internal C++ code that is designed from scratch to perform GRMHD simulations on GPUs. The development goal of GRACE is to perform BNS simulations, exactly the same as FIL, with all the FIL kernels targeted for optimization in D2.1 being ported to GRACE. GRACE leverages Kokkos for its parallelization strategy on heterogeneous hardware and uses the p4est library which provides an Octree based AMR scheme for computational efficiency. With Kokkos, we are able to run GRACE on CPUs and GPUs with support for AMD and CUDA.

GRACE has been a joint collaboration within the relativistic astrophysics group since the start of this year. At the interim review, in October, it was presented to the reviewers and given ascent to formally be worked on as part of the SPACE collaboration. FIL will remain the main CPU code and GRACE will be the GPU code that is worked on as part of the SPACE project. In this report, we will outline the current status of both codes.

6.1 Code status

Currently GRACE is capable of doing GRHD simulations on static backgrounds, meaning it can simulate an isolated stationary neutron star on multiple GPUs. Current work is focused on implementing the Baumgarte, Shapiro, Shibata, Nakamura formalism, which will allow for evolving space times required to do BNS simulations. As discussed in D2.3, there is a well-tested development path for numerical relativity codes and GRACE has progressed significantly along this path. Because of Kokkos, GRACE is able to run on AMD and NVIDIA GPUs, it can even run on CPUs if desired. GRACE is currently installed and running on LUMI's and on the IT4I's complementary systems AMD GPUs and work is underway to compile and run on Karolina's NVIDIA GPUs.

6.2 Kernel status

As shown in D2.3 both kernels targeted for optimisation (see D2.1) have been ported to GRACE with the add fluxes to rhs, kernel 1, seeing a 99% performance increase and the conservative to primitive solver, kernel 2,

seeing a 91% performance increase. Both Kernels can run on multiple GPUs.

6.2.1 Kernel 1 - Add fluxes to rhs

GRACE and [FIL](#) like most relativistic fluid codes use the Valencia formulation to re-write the set of fluid evolution equations in conservative form as shown below:

$$\partial_t \mathbf{U} + \partial_i \mathbf{F}^i = \mathbf{S}, \quad (2)$$

where $\mathbf{U}$ are the set of conservative variables, $\mathbf{F}$ are the fluxes and $\mathbf{S}$ are the source terms. The set of conservative variables are constructed from the set of primitive variables, which are physical meaningful. The primitive variables are given by $\mathbf{U} = \mathbf{U}(\rho_0, P, v^i, B^i)$ where ρ_0 is the rest mass density of a fluid element, P is the pressure, v^i is the fluid 3 velocity and B^i is the magnetic field vector. Kernel 1 performs the finite difference methods on $\mathbf{F}$ and adds the values to the rhs of equation (2).

Kernel 1 has been partially ported to GRACE, currently magnetic fields contributions are not included. So far GRACE has been tested for non-evolving stationary spacetimes. The contributions to the dynamics from an evolving space time appear as factors within the flux equation. Currently, even though the space time does not evolve, these factors are evaluated at every time step for the static space times. Kernel 1 will not have to be modified to implement evolving space times, the factors will be calculated in other parts of the code and then will be multiplied by terms in the flux equation. For the magnetic fields an additional set of variables will be added to the set of equations (2). Currently, work has started on implementing magnetic fields within GRACE.

6.2.2 Kernel 2 - Conservative to primitive solver

Kernel 2, the conservative to primitive solver ([con2prim](#)), the calculation of the conservative variables from the primitives is trivial, but to calculate the primitives from the conservatives, which is required at each sub-step of a Runge-Kutta method requires a Newton–Raphson-based root finder algorithm. This algorithm has been implemented in GRACE. Like for kernel 1 the addition of magnetic fields will require the [con2prim](#) kernel to be modified slightly. The bigger problem with the addition of magnetic fields is that within the presence of strong magnetic fields [con2prim](#) algorithms are known to be less robust. This is a problem we will have to solve in the future when magnetic fields will be introduced in GRACE. However, as of now both kernel 1 and kernel 2 show excellent performance on GPUs as seen in D2.3. As mentioned at the end of D2.3, we expect the performance gains at the stage to be the high water mark, but with such strong results we expect solid performance improvements of GRACE when compared to [FIL](#) for the same simulations.

6.3 CI/CD

Currently for [FIL](#) a runner is set up on the IT4I complementary systems that performs 3 tests every time there is a commit. The first test is a simple build test, the code is compiled. Next, the executable is tested by printing out a hello world statement to make sure it is functional. The final test is a shock tube test where the code performs a simulation of a shock tube and the results are compared to the analytical shock tube results to measure the code accuracy. For GRACE we want to set up a similar pipeline but will deploy the runner on multiple systems with different architecture.

6.4 Deployment on EuroHPC systems

We have applied and got access to all 8 EuroHPC supercomputing clusters through EuroHPC benchmark/development calls. Specifically, we have access to the CPU partitions of all 8 EuroHPC machines and to the GPU partitions of Karolina, Leonardo, Lumi. We are currently applying for access to the remaining clusters with GPU partitions, namely Deucalion, MareNostrum 5 and Vega. We have also access to the supplementary systems of IT4I and the CPU and GPU partitions of E4. Currently [FIL](#) has been compiled and run on all of the EuroHPC systems, whereas GRACE has been compiled and run on LUMI and the IT4I-complementary systems, see table 5 for summary. In figure 12 a comparison of cell updates per second for [FIL](#) compared to GRACE is presented.

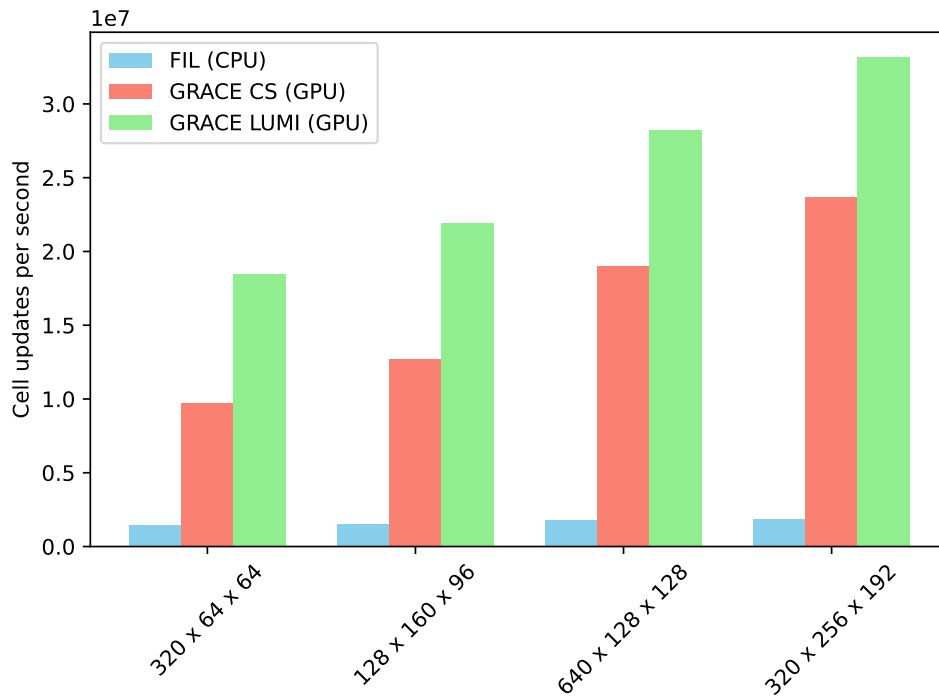


Figure 12: Performance comparison between [FIL](#) and GRACE where the performance metric is cell updates per second. This metric ignores IO contributions to the runtime of both codes.

Cluster	Deucalion	Discoverer	Karolina	Leonardo	Lumi	MareNostrum	MeluXina	Vega
Partitions	CPU,GPU	CPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU
Access	✓, x	✓	✓, ✓	✓, ✓	✓, ✓	✓, x	✓, x	✓, x
Deploy	✓, x	✓	✓, x	✓, x	✓, ✓	✓, x	✓, x	✓, x
Benchmark	x, x	x	✓, x	✓, x	✓, ✓	✓, x	✓, x	✓, x

Table 5: Access, deployment and benchmarking of [FIL](#) on the CPU partions of EuroHPC machines and GRACE on the GPU partitions of the EuroHPC machines.

6.5 Summary and conclusion

[FIL](#) as a module of the [ET](#) is a highly optimised CPU code that scales well up to 512 nodes as can be seen in [Figure 11](#). The [ET](#) was first created in 2004 and has had a dedicated team continuing the development and optimisation for the last twenty years. This created a difficult dilemma when it came to the [SPACE](#) project because [FIL](#) is already well CPU-optimised and currently the [ET](#) and therefore [FIL](#) is incompatible with GPUs. For this reason we decide to build GRACE, a [GRMHD](#) code that will be capable of performing binary neutron star mergers on GPUs. Current progress of GRACE is good, and thanks to access to the [SPACE](#) project resources we expect to be able to create a state-of-the-art of numerical relativity codes that outperforms [FIL](#).

7 iPIC3D

iPIC3D [22] is a Particle-in-Cell (PIC) code developed to study plasma dynamics at the kinetic scale, where the individual (macro)particles of a plasma are evolved in a Lagrangian framework, whereas the moments (electric current, density, etc.) and the electric and magnetic fields are tracked on a Eulerian grid. It solves the Maxwell-Vlasov equation self-consistently using the implicit moment method. The three main kernels of iPIC3D are (a) Particle Mover, (b) Moment Gatherer, and (c) Field Solver.

Owing to the “implicitness” of the method involved, unresolved scales do not result in any numerical instabilities, as opposed to explicit PIC methods, where one is restricted to the Courant-Friedrichs-Lewy constraint. This allows us to choose time-step sizes and grid sizes that can be 10 to 100 times larger than those used in explicit PIC codes.

7.0.1 Code Status

We have recently started collaborating with one of the original authors of iPIC3D, Prof. Stefano Markidis (KTH Sweden), and we have decided to switch to a new publicly available version of the code, iPIC3D-CPU-SPACE-CoE (<https://github.com/Pranab-JD/iPIC3D-CPU-SPACE-CoE>), that is primarily developed by . Markidis and his team. Significant improvements have been made in the newer version of the code, which includes non-blocking asynchronous communication across the different MPI subdomains, reducing data dependence for writing data to files, and in situ visualisation with ParaView Catalyst (<https://docs.paraview.org/en/latest/Catalyst/index.html>). We compare the performance of the new version of iPIC3D with that of the old one (https://code.it4i.cz/space_coe/iPic3D and <https://github.com/CmPA/iPic3D>) for two test cases: a two-dimensional isotropic Maxwellian plasma and a three-dimensional case of magnetic reconnection (Table 6). These test cases, performed on Leonardo DCGP, show a performance enhancement of a factor of 5 to 7. Due to its significantly superior performance, we will use this version of the code for developments within the scope of SPACE. We note that iPIC3D-CPU-SPACE-CoE has been highly optimised for computational performance and is ready for production. We demonstrate strong and weak scaling on two different EuroHPC JU clusters, Lumi and MeluXina. We obtain over 90% efficiency on up to 16000 cores for all cases considered here (Fig. 13). Future developments will include new mathematical algorithms such as the energy-conserving semi-implicit method (ECSIM, [23]) and relativistic semi-implicit method (RelSIM, [24]).

Problem	iPIC3D (new)	iPic3D (old)	Speedup
Maxwell 2D	104 s	749 s	7.2
GEM 3D	278 s	1559 s	5.6

Table 6: Performance comparison of the old and new versions of iPIC3D. The new version shows over half an order of magnitude improvement in computational performance.

GPU

For the first time, we have a fully functioning GPU version of the code — iPIC3D-GPU-SPACE-CoE (<https://github.com/Pranab-JD/iPIC3D-GPU-SPACE-CoE>) whose basic infrastructure has also primarily developed by Prof. Markidis and his team. In this version, the particle mover and moment gatherer modules run on the GPU whereas the field solver runs on the CPU. This necessitates data communication from the GPU to the CPU and back at every time step. Preliminary tests show that porting the field solver to the GPU does not offer any significant benefits. This GPU code is written using CUDA which is supported on NVIDIA GPUs. We compare its performance with the CPU version of the code for the test case of 3D magnetic reconnection on a single node on the Leonardo Booster (Table 7). As the field solver runs on the CPU, no difference in runtime is observed between the two versions of the code for the field solver part. However, for the particle mover and moment gatherer we obtain a speedup of ~ 40 and ~ 100 times respectively, thereby determining an overall speedup of ~ 40 with respect to the iPIC3D-CPU-SPACE-CoE (CPU version). We show initial scaling on Karolina GPU in Fig. 13.

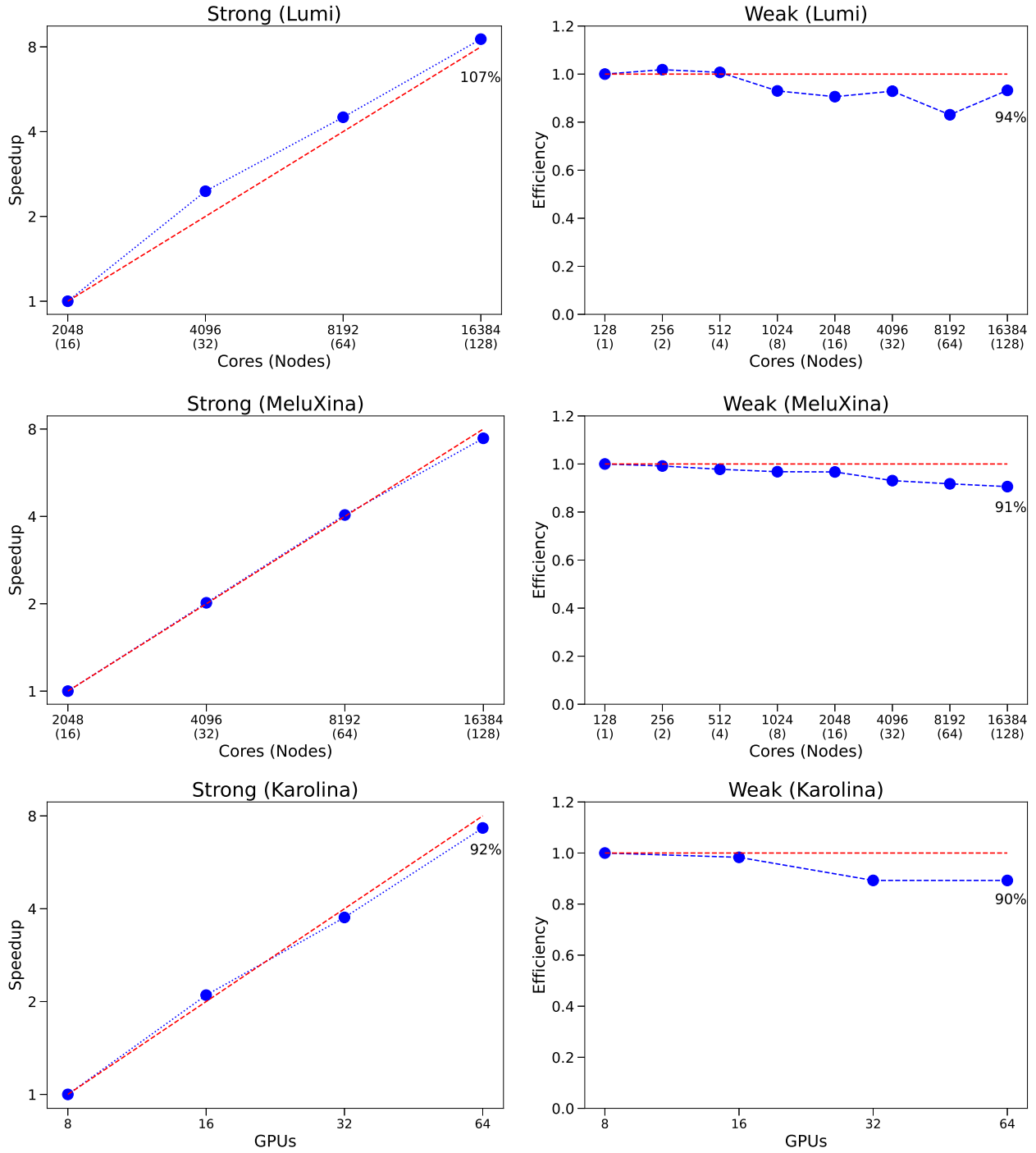


Figure 13: Top and middle rows: Strong and weak scaling of iPIC3D-CPU-SPACE-CoE on up to 16000 cores on two EuroHPC JU clusters. Ideal scaling is denoted with the red lines. Bottom panel: Strong and weak scaling of iPIC3D-GPU-SPACE-CoE on up to 64 GPUs on Karolina GPU.

Module	iPIC3D-GPU-SPACE-CoE (Leonardo Booster)	iPIC3D-CPU-SPACE-CoE (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 7: Performance comparison of the CPU and GPU versions of iPIC3D.

7.1 Kernel status

The new version of the code, iPIC3D-CPU-SPACE-CoE, is better optimised for high performance, with a focus on modularity, parallelism, and cache-friendly memory access patterns. One of the main improvements in the code involves the use of OpenMP for multithreaded parallelism in addition to MPI parallelisation. Additionally, new template classes have been defined for storing multi-dimensional arrays. Previously, multiple pointers (i.e., `double ****Ex`) were used to initialise and store moments and field components. The use of templates for storing these variables significantly minimises the risk of memory leaks and improve code readability and safety.

7.1.1 Particle Mover

The new version facilitates OpenMP parallelisation through the use of `#pragma omp parallel` and `#pragma omp for`. This enables multithreaded looping over the particles. Furthermore, `#pragma simd` has been employed for multithreaded vectorisation. There has been a focus on manually unrolling loops to assist compilers in vectorization. The components of electromagnetic fields are computed by the function `get_field_components_for_cell` for any given cell over computing all components individually without proper abstraction. This facilitates modularity and code-readability.

Currently, we are implementing the particle movers for the ECSIM and RelSIM algorithms.

7.1.2 Field Solver

The electric field calculation has been consolidated into one function `calculateE`, whereas the previous version splits the process into multiple functions (`startEcalc`, `calculateE`, `endEcalc`). The communication of boundary data and the smoothing of electric fields is handled directly within `calculateE`, avoiding separate calls in different stages. Reducing the number of functions can lead to a reduction in the overhead of function calls. A conditional check, `cycle % PoissonCorrectionCycle == 0`, is implemented to ensure that divergence cleaning is performed periodically, improving stability and reducing unnecessary calculations.

7.2 CI/CD

	2D Maxwellian	3D Magnetic Reconnection
Resolution	20×20	$60 \times 60 \times 10$
Time cycles	50	600
Particles per cell	$95 \times 95 \times 2$ (species)	$15 \times 15 \times 15 \times 4$ (species)
MPI topology	2×2	$2 \times 2 \times 1$
Runtime	~ 2 mins	~ 30 mins

Table 8: Configuration of CI/CD test problems for iPIC3D-CPU-SPACE-CoE.

CI/CD has been set up for iPIC3D-CPU-SPACE-CoE using GitHub workflows. We consider two test cases, one of a 2D isotropic Maxwellian plasma and the other of a 3D reconnection problem. The configurations for these test problems are presented in Table 8. These tests involve downloading `mpich`, building and compiling iPIC3D, and running the test problem (which involves running all three modules of iPIC3D since they are inherently coupled). The test case for the Maxwellian has a runtime of about two minutes on the default number of free CPU cores (4) provided by GitHub, whereas the one for magnetic reconnection runs for about 30 minutes to ensure that the code runs successfully for relatively long periods of time.

We are currently working towards employing CI/CD on Leonardo and Karolina. This would allow us to use more than the default free cores provided by GitHub and implement larger test cases to ensure proper functionality of the code for large-scale problems.

7.3 Deployment on EuroHPC JU systems

iPIC3D-CPU-SPACE-CoE has been deployed (Table 9) and benchmarked on up to 32 nodes (Fig. 14) on the CPU partitions of **all of the eight EuroHPC JU clusters**. We state the modules to be loaded to compile and run iPIC3D-CPU-SPACE-CoE on the different clusters (Table 10). iPIC3D-GPU-SPACE-CoE has been deployed on **two of the six JU clusters** that have NVIDIA GPUs. We have recently gotten access to the remaining four and we will present these benchmarking results in the next deliverable.

Cluster	Deucalion	Discoverer	Karolina (CPU + GPU)	Leonardo (CPU + GPU)
Access	✓	✓	✓	✓
Deploy	✓	✓	✓	✓
Benchmark	✓	✓	✓	✓

Cluster	Lumi	MareNostrum	MeluXina	Vega
Access	✓	✓	✓	✓
Deploy	✓	✓	✓	✓
Benchmark	✓	✓	✓	✓

Table 9: Deployment and benchmarking of iPIC3D-CPU-SPACE-CoE on the EuroHPC JU clusters.

Cluster	Modules
Deucalion	CMake/3.29.3-GCCcore-13.3.0 OpenMPI/5.0.3-GCC-13.3.0
Discoverer	cmake/3/3.29.6 gcc/14/14.2.0 openmpi/5/gcc/5.0.3
Karolina	CMake/3.27.6-GCCcore-13.2.0 OpenMPI/4.1.6-GCC-13.2.0
Leonardo	cmake/3.27.7 openmpi/4.1.6--gcc--12.2.0
Lumi	
MareNostrum	ucx/1.15.0-gcc mpich/4.2.2-gcc cmake/3.25.1
MeluXina	CMake/3.26.3-GCCcore-12.3.0 HDF5/1.10.7-gompi-2023a
Vega	CMake/3.26.3-GCCcore-12.3.0 OpenMPI/4.1.5-GCC-12.3.0

Table 10: List of the modules used (explicitly loaded) on each of the clusters follows.

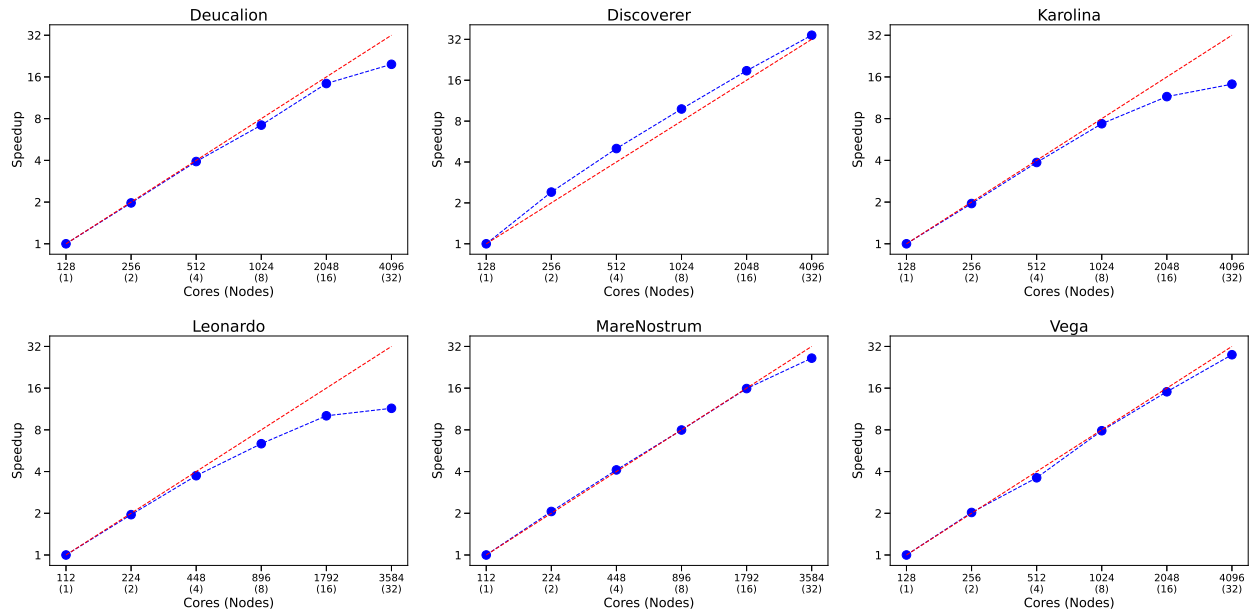


Figure 14: Strong scaling on different EuroHPC JU clusters. Ideal scaling is denoted with the red lines.

8 RAMSES

RAMSES [25, 26] is an Adaptive Mesh Refinement (AMR) code which is used to study astrophysical fluid dynamics and the formation of structures in the Universe. It is based on a Fully Threaded Tree oct-tree structure [27], where parent cells (oct) are refined into children cells on a cell-by-cell basis following some user-defined criteria. RAMSES can deal with 1D, 2D and 3D Cartesian grids. RAMSES is written in Fortran 90.

The public version of RAMSES² comes with a full MPI parallelization and runs on CPU only. It only needs a Fortran and MPI compiler to be deployed on the EuroHPC scalar partitions. RAMSES does not require any external library to be installed.

8.1 Code status

We have conducted a twofold strategy for the development of RAMSES, depending on the target architecture (CPU or GPU). For CPU, we develop from the public version of RAMSES. We have begun to port hybrid MPI and OpenMP parallelization which shows already promising results on the hydrodynamical solver only. RAMSES has also been compiled and benchmarked on Intel Sapphire Rapids and Grace-Grace architectures.

For GPU, we employ the SHAMROCK³ framework to handle the AMR structure fully on GPUs. SHAMROCK can handle particles and AMR grids on GPUs thanks to a radix tree. We have ported the hydrodynamical solver of RAMSES (Module 1) into the SHAMROCK framework (new branch `ramses2shamrock`) and benchmarked it against Use Case 1 (Sedov blast in 3D, 2048³ uniform grid) on LUMI-G. We obtained a weak scaling efficiency of 84% on 512 GPUs (using 128³ cells per GPU). The code is not yet public as the first paper presenting the code is under review. Once it will be accepted, we will make it public.

8.2 Kernel status

8.2.1 Kernel 1 - Hydro module

For the CPU partitions, thanks to the collaboration with Eviden's CEPP that highlights the relevancy of an hybrid approach to leverage many-core CPU, we have implemented an hybrid OpenMP-MPI parallelization, for the hydrodynamical solver. We have tested the implementation against Use Case 1 (Sedov blast in 3D, 1024³ uniform grid) and obtain an increase of strong scalability of 11% on 64 nodes on MeluXina. The performance gained is due to the reduction of the number of MPI domains for a fixed number of cores. This also leads to a reduction of the memory consumption by the ghost region of each MPI domain. We obtained a memory gain of a factor 4 on MeluXina using 8 OMP threads per MPI task. For the RAMSES community, this gain is very useful. This will enable to run production simulations with a larger number of variables as needed, for instance, when radiative transfer of chemistry are activated (e.g. the SPHINX reionisation project, <https://sphinx.univ-lyon1.fr>).

8.2.2 Kernel 2 - Communications

The communication routines have not yet been ported to hybrid parallelization. We shall stress that communications between MPI domains (`MPI_ALLREDUCE`, `MPI_SEND`, and `MPI_RECEIVE`) are found in many routines of the code, not only the ones we have identified. Nevertheless, we obtained "for free" significant speed up using the hybrid parallelization (see D2.3). Indeed, the number of MPI domains decreases when OpenMP is activated, by a factor 8 when 8 OpenMP threads are used. As a consequence, less MPI domains have to communicate. We plan to optimize further the communication routine for the next deliverable.

8.3 CI/CD

We use the gitlab of our institute CRAL for development. We have set up a CI to compile the code and run a test suite. For the moment, it runs after each commit. The test suite is done on physical setups (Sod tube, implosion, advection, collapse) which test the main kernel of RAMSES, and in particular the hydro module we have identified for optimization. Our future plan for CI/CD is as follows:

²<https://github.com/ramses-organisation/ramses>

³<https://github.com/Shamrock-code>

- 1/ optimize the CI pipeline to only compile the code after each commit and run the test suite only once a day.
- 2/ use the mirror repository at IT4I to run the test suite on a supercomputer architecture.
- 3/ integrate additional tests to run the CIC, adaptive-mesh-refinement and gravity modules.

8.4 Deployment on EuroHPC systems

The status of RAMSES deployment on the CPU EuroHPC systems is listed in table 11. There was no particular difficulty encountered to deploy the code since RAMSES uses standard Fortran and MPI. The combination of modules we have used is listed for each system in table 12. Note that we have used here the stable release of RAMSES which can be obtained from the public repository on [github](#).

For this deliverable, we have benchmarked RAMSES with Use Case 1, i.e. the Sedov blast. It consists of a 3D computational volume and uniform grid resolution is used. In this Use Case, the hydro solver and the communication routines are tested.

8.4.1 Optimal internal vectorization

In order to save computations where high precision is not required, the RAMSES code uses an [AMR](#) mesh. An important side effect of this approach is that the mesh might change during the simulation. This implies the complexification of the data description. In particular, it is complicated to retrieve the neighborhood of every cell in the mesh, and the underlying indirect memory access patterns might be harmful regarding code performance. To avoid this problem from occurring too many times, the hydro kernel of RAMSES begins by filling a kind of software cache, often referred to as vectors, into the code. This enables it to load a contiguous portion of the mesh and all its neighborhood once for all before calling the Godunov solver.

The size of these vectors, denoted by `nvector`, is defined at compile time. This parameter has a strong impact on code performance. Indeed, large vectors lead to a better vectorization rate, but small vectors are better for hardware cache performance. Hence, a compromise has to be made between hardware cache performances and vectorization rate for each architecture.

Prior to benchmarking, we first performed a study to obtain the best `nvector` internal vectorization parameter, which depends on the architecture of each JU system. An example of this pre-benchmark analysis is given in fig. 15. The optimal value is found to be 8 for the Use Case 1 and it is quite consistent across all JU systems. Note that in the case of a simulation with a larger number of variables (e.g., with MHD or radiative transfer), this optimal value might change and this test has to be done again.

8.4.2 Benchmarks

We have benchmarked RAMSES on 8 out of the 8 JU scalar systems. We summarize in table 12 the modules we use on each system. Figure 16 shows strong and weak scaling done for Use case 1. Table 13 summarises the efficiency obtained on the strong scaling test with a 2048^3 grid. We obtain a strong scaling $> 70\%$ up to 45000 cores (or MPI domains) on MareNostrum. We expect that the scaling will improve with the hybrid MPI and OpenMP parallelisation.

Cluster	Deucalion	Discoverer	Karolina	Leonardo	Lumi	MareNostrum	MeluXina	Vega
Access	✓	✓	✓	✓	✓	✓	✓	✓
Deploy	✓	✓	✓	✓	✓	✓	✓	✓
Benchmark	✓	✓	✓	✓	✓	✓	✓	✓

Table 11: Status of RAMSES on the JU systems with CPU partitions.

Cluster	Modules
Deucalion	GCC/13.3.0, OpenMPI/5.0.3-GCC-13.3.0
Discoverer	gcc/latest, openmpi/5/gcc/latest
Karolina	GCC/13.2.0, OpenMPI/4.1.6-GCC-13.2.0
Leonardo	intel-oneapi-compilers, intel-oneapi-mpi
Lumi	LUMI, cpeGNU
MareNostrum	intel/2023.2.0, impi/2021.10.0
MeluXina	GCC, OpenMPI/4.1.5-GCC-12.3.0
Vega	GCC, openmpi/gnu/4.1.2.1

Table 12: List of the modules used (explicitly loaded) for RAMSES benchmarks on each of the JU cluster scalar partitions.

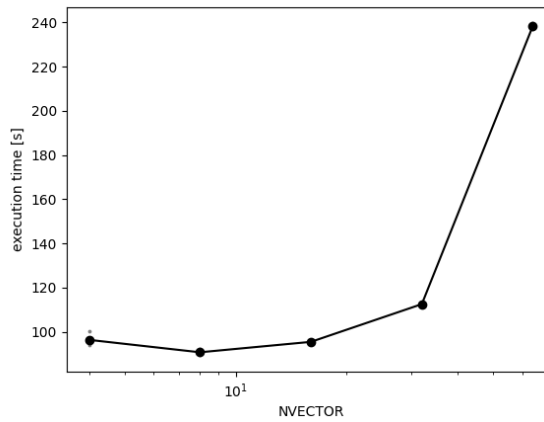


Figure 15: Execution time of RAMSES for Use Case 1 on Marenostrum as a function of the internal `nvector` parameter. The best performance is obtained with `nvector` = 8. The test has been performed on a grid of 2048^3 using 8 nodes.

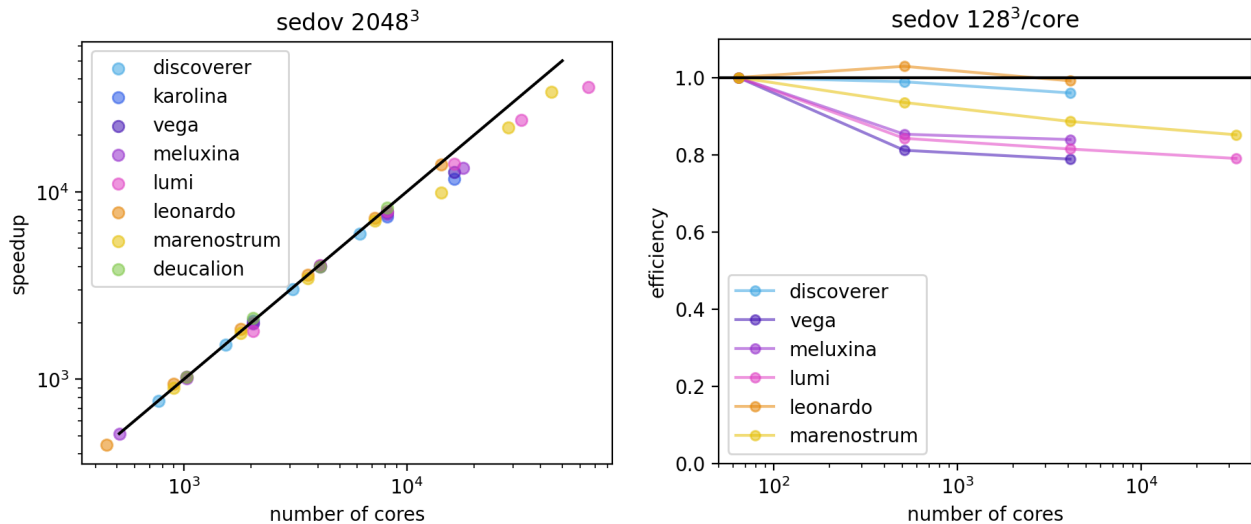


Figure 16: RAMSES strong (left) and weak scaling (right) on up to 65536 cores (on Lumi) on 7 out of the 8 EuroHPC JU clusters with Use Case 1 (Sedov blast). Ideal scaling is denoted with the black lines. The strong scaling has been performed using a uniform grid resolution of 2048^3 . For the weak scaling, we increased the number of cores keeping a uniform grid of 128^3 per core.

Cluster	Ncores	Efficiency
Deucalion	8192	100%
Discoverer	6144	97%
Karolina	16384	71%
Leonardo	14336	97%
Lumi	65536	55%
MareNostrum	44800	76%
MeluXina	17920	75%
Vega	16384	78%

Table 13: Strong scaling efficiency on the JU clusters with the maximum number of cores we achieved to use on the EuroHPC Development Access.

9 Conclusions

The deliverable provided a summary of code activities for each code. While for some cases large improvements have been achieved in deploying the code to GPU, while others have achieved strong results on their CPU implementations. Since no code is able to run on GPU only, every improvement on the CPU side also increases the exaflop capabilities of each code. In addition to the results achieved, the potential of improvements in development show large potential for increased performance. In conclusion, all the code development activities are within the planned schedule.

As it was presented, each code has implemented a CI pipeline to ensure the code's integrity. While the exact size, job number, schedule and other parameters of the CI implementation vary between the codes significant progress has been achieved.

In the following, the status of the different codes regarding the different [EuroHPC JU](#) systems is summarized. As it can be seen in Table 14 a very high fraction of codes has been deployed to all systems. In addition to the deployment, several codes have also been running benchmarks on a range of systems. This shows that the codes are capable of being used on all the [EuroHPC JU](#) systems.

Code \ Cluster	Deucalion	Discoverer	Karolina	Leonardo	Lumi	MareNostrum	MeluXina	Vega
Partitions	CPU,GPU	CPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU	CPU,GPU
OpenGadget								
Access	✓, ✓	✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓
Deploy	✓, ✓	✓	✓, ✓	✓, ✓	✓, x	✓, ✓	✓, ✓	✓, ✓
Benchmark	x, x	x	✓, x	✓, ✓	x, x	x, x	x, x	x, x
PLUTO								
Access	✓, x	✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓
Deploy	✓, x	✓	✓, ✓	✓, ✓	✓, x	✓, ✓	✓, ✓	✓, ✓
Benchmark	x, x	x	✓, ✓	✓, ✓	✓, x	✓, ✓	✓, ✓	✓, ✓
BHAC								
Access	✓, x	✓	✓, ✓	✓, ✓	✓, ✓	✓, x	✓, x	✓, x
Deploy	✓, x	✓	✓, ✓	✓, ✓	✓, x	✓, x	✓, x	✓, x
Benchmark	x, x	x	✓, ✓	✓, x	✓, x	✓, x	✓, x	✓, x
ChaNGa								
Access	✓, x	✓	✓, ✓	✓, ✓	✓, x	✓, ✓	✓, ✓	✓, ✓
Deploy	✓, x	✓	✓, ✓	✓, ✓	✓, x	✓, ✓	✓, ✓	✓, ✓
Benchmark	x, x	✓	✓, ✓	✓, x	✓, x	x, x	✓, ✓	✓, x
FIL/GRACE								
Access	✓, x	✓	✓, ✓	✓, ✓	✓, ✓	✓, x	✓, x	✓, x
Deploy	✓, x	✓	✓, x	✓, x	✓, ✓	✓, x	✓, x	✓, x
Benchmark	x, x	x	✓, x	✓, x	✓, ✓	✓, x	✓, x	✓, x
iPIC3D								
Access	✓, ✓	✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓
Deploy	✓, x	✓	✓, ✓	✓, ✓	✓, x	✓, x	✓, x	✓, x
Benchmark	✓, x	✓	✓, ✓	✓, ✓	✓, x	✓, x	✓, x	✓, x
RAMSES								
Access	✓, ✓	✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓	✓, ✓
Deploy	✓, x	✓	✓, x	✓, x	✓, ✓	✓, x	✓, x	✓, x
Benchmark	✓, x	✓	✓, x	✓, x	✓, ✓	✓, x	✓, x	✓, x

Table 14: Deployment status of the codes on the different [EuroHPC JU](#) clusters.

References

- [1] A. Mignone, G. Bodo, S. Massaglia, T. Matsakos, O. Tesileanu, C. Zanni, and A. Ferrari, “PLUTO: A Numerical Code for Computational Astrophysics,” *The Astrophysical Journal Supplement Series*, vol. 170, no. 1, pp. 228–242, May 2007.
- [2] A. Mignone, M. Ugliano, and G. Bodo, “A five-wave Harten-Lax-van Leer Riemann solver for relativistic magnetohydrodynamics,” *Monthly Notices of the Royal Astronomical Society*, vol. 393, pp. 1141–1156, Mar. 2009.
- [3] A. Mignone, C. Zanni, P. Tzeferacos, B. van Straalen, P. Colella, and G. Bodo, “The PLUTO Code for Adaptive Mesh Computations in Astrophysical Fluid Dynamics,” *The Astrophysical Journal Supplement Series*, vol. 198, no. 1, p. 7, Jan. 2012.
- [4] F. et al, “Flash: an adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes,” *ApJS* 131:273 (pages 292-294), 2000.
- [5] P. Mignone and Bodo, “The piecewise parabolic method for multidimensional relativistic fluid dynamics,” *ApJS* 160,199, 2005.
- [6] O. Porth, H. Olivares, Y. Mizuno, Z. Younsi, L. Rezzolla, M. Moscibrodzka, H. Falcke, and M. Kramer, “The Black Hole Accretion Code,” *Computational Astrophysics and Cosmology*, vol. 4, 2017.
- [7] H. Olivares, O. Porth, J. Davelaar, E. R. Most, C. M. Fromm, Y. Mizuno, Z. Younsi, and L. Rezzolla, “Constrained transport and adaptive mesh refinement in the black hole accretion code,” *Astronomy & Astrophysics*, vol. 629, p. A61, 2019.
- [8] “The Black Hole Accretion Code – Documentation,” <https://bhac.science>, accessed: 2023-11-01.
- [9] “Repository of BHAC on GitLab,” <https://gitlab.itp.uni-frankfurt.de/BHAC-release/bhac>, accessed: 2023-11-01.
- [10] “The MPI - Adaptive Mesh Refinement - Versatile Advection Code – Documentation,” <https://amrvac.org/>, accessed: 2023-11-01.
- [11] “Repository of MPI-AMRVAC on GitHub,” <https://github.com/amrvac/amrvac>, accessed: 2023-11-01.
- [12] S. Cielo, O. Porth, L. Iapichino, A. Karmakar, H. Olivares, and C. Xia, “Optimizing the hybrid parallelization of bhac,” *Astronomy and Computing*, vol. 38, p. 100509, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2213133721000639>
- [13] B. van der Holst, R. Keppens, and Z. Meliani, “A multidimensional grid-adaptive relativistic magnetofluid code,” *Comput. Phys. Commun.*, vol. 179, pp. 617–627, 2008.
- [14] S. C. Noble, C. F. Gammie, J. C. McKinney, and L. Del Zanna, “Primitive variable solvers for conservative general relativistic magnetohydrodynamics,” *Astrophys. J.*, vol. 641, pp. 626–637, 2006.
- [15] “Repository of the changa code on github,” <https://github.com/N-BodyShop/changa>, accessed: 28.10.2023.
- [16] P. Jetley, F. Gioachin, C. Mendes, L. V. Kale, and T. R. Quinn, “Massively parallel cosmological simulations with ChaNGa,” in *Proceedings of IEEE International Parallel and Distributed Processing Symposium 2008*, 2008.
- [17] H. Menon, L. Wesolowski, G. Zheng, P. Jetley, L. Kale, T. Quinn, and F. Governato, “Adaptive techniques for clustered N-body cosmological simulations,” *Computational Astrophysics and Cosmology*, vol. 2, p. 1, Mar. 2015.
- [18] J. W. Wadsley, B. W. Keller, and T. R. Quinn, “Gasoline2: a modern smoothed particle hydrodynamics code,” *Monthly Notices of the Royal Astronomical Society*, vol. 471, no. 2, pp. 2357–2369, Oct. 2017.
- [19] J. G. Stadel, “Cosmological N-body simulations and their analysis,” Ph.D. dissertation, University of Washington, Seattle, Jan. 2001.

- [20] Z. B. Etienne, V. Paschalidis, R. Haas, P. Mösta, and S. L. Shapiro, “IllinoisGRMHD: An Open-Source, User-Friendly GRMHD Code for Dynamical Spacetimes,” *Class. Quant. Grav.*, vol. 32, p. 175009, 2015.
- [21] M. Zilhão and F. Löffler, “An Introduction to the Einstein Toolkit,” *Int. J. Mod. Phys. A*, vol. 28, p. 1340014, 2013.
- [22] S. Markidis, G. Lapenta, and Rizwan-uddin, “Multi-scale simulations of plasma with iPIC3D,” *Math. Comput. Simul.*, vol. 80, no. 7, pp. 1509–1519, 2010.
- [23] G. Lapenta, “Exactly energy conserving semi-implicit particle in cell formulation,” *J. Comput. Phys.*, vol. 334, pp. 349–366, 2017.
- [24] F. Bacchini, “RelSIM: A Relativistic Semi-implicit Method for Particle-in-cell Simulations,” *ApJS*, vol. 268, no. 2, p. 60, 2023.
- [25] R. Teyssier, “Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES,” *Astronomy and Astrophysics*, vol. 385, pp. 337–364, Apr. 2002.
- [26] “Ramses repository on GitLab,” <https://bitbucket.org/rteyssie/ramses/src/master/>.
- [27] A. Khokhlov, “Fully Threaded Tree Algorithms for Adaptive Refinement Fluid Dynamics Simulations,” *Journal of Computational Physics*, vol. 143, no. 2, pp. 519–543, Jul. 1998.